EVALUATING BENCHMARK PROBLEMS BY RANDOM GUESSING

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de

Yoshua Bengio
Dept. Informatique et Recherche Opérationnelle
Université de Montréal, CP 6128, Succ. Centre-Ville,
Montréal, Québec, Canada, H3C 3J7
bengioy@iro.umontreal.ca

November 4, 1999

1 Introduction

Numerous recent papers focus on standard recurrent networks' problems with tasks involving long-term dependencies. In this chapter we will solve such tasks by random weight guessing (RG). Although RG cannot be viewed as a reasonable learning algorithm we find that it often outperforms previous,

more complex methods on widely used benchmark problems. One reason for RG's success is that the solutions to many of these benchmarks are dense in weight space. An analysis of cases in which RG works well versus those in which it does not can serve to improve the quality of benchmarks for novel recurrent network algorithms.

The main problem of conventional, gradient-based recurrent net learning algorithms (see overviews by Williams, 1989; Pearlmutter, 1995; see also Chapter 13) is this: when the net's attractor dynamics allow for storing long-term context (Bengio et al., 1994), error signals "flowing backwards in time" tend to decay exponentially, as was shown first by Hochreiter (1991) — see also Chapter 7.

In recent years numerous methods have been proposed to address this problem. For instance, Schmidhuber (1992) suggested to compress regular sequences with the chunking algorithm. Hochreiter's ideas (1991) led to the recent "Long Short-Term Memory" algorithm (LSTM, Hochreiter and Schmidhuber, 1997a). Bengio et al. (1994) investigated simulated annealing, multi-grid random search, time-weighted pseudo-Newton optimization, and discrete error propagation, to evaluate their performance in learning long-term dependencies. Bengio and Frasconi (1994) also proposed an EM approach for propagating targets.

Several researchers use the *2-sequence problem* (and *latch problem*) to compare various algorithms, e.g., Bengio et al. (1994), Bengio and Frasconi (1994), El Hihi and Bengio (1995), Lin et al. (1995). For the same purpose, some also use the *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 networks (see, e.g., Bengio and Frasconi, 1995; Watrous and Kuhn, 1992; Pollack, 1991; Miller and Giles, 1993; Manolios and Fanelli, 1994).

Here we will show that some instances of such problems can be solved much more quickly by random weight guessing (RG). In other cases, however, RG does not perform well. This paper's purpose is *not* to suggest that RG is a good algorithm for training recurrent networks. Instead it intends to contribute to understanding RG's significance and shortcomings in long time lag algorithms. It also suggests that RG should be used as a first test to evaluate some benchmark problem's difficulty.

2 Random Guessing (RG)

Given a particular network architecture, RG works as follows:

> **REPEAT** *randomly initialize the weights* **UNTIL** *the resulting network happens to classify all training sequences correctly. Then test on a separate test set.*

Note that unlike simulated annealing, RG does not involve a time consuming "cooling phase."

We use two architectures A1 and A2 suitable for many widely used "benchmark" problems: A1 is a recurrent, fully connected network with 1 input, 1 output, and n hidden units. Each hidden unit has a bias weight on a connection from a "true" unit with constant activation 1.0. 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 or experimental setups.

In all our experiments, we randomly initialize weights in [-100.0,100.0]. Activations of all units (except for the "true" unit) are set to 0 at the beginning of each sequence.

All the "benchmark" problems below require classifying two types of pattern sequences that are fed sequentially (pattern by pattern) into the network via its input units (in the standard way). To achieve a uniform setting, all sequence lengths are randomly chosen between 500 and 600 (in most previously reported experiments, shorter training/test sequences have been used). Training sets consist of 100 sequences, 50 from class 1 (target 0) and 50 from class 2 (target 1). Unless mentioned otherwise, binary inputs are -1.0 and 1.0. 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 separate test set (100 sequences). In this sense all our results measure generalization performance.

Our main motivation for using the same architectures and experimental conditions for many different problems is to prevent critique of problem-specific fine-tuning. For instance, in our experiments we do not adapt architectures until we find the best.

All results in the remainder of this note are averages of 10 or more simulations. In all our simulations, RG classifies correctly at least 99% of all test

set sequences; average absolute test errors are always below 0.02, in most cases below 0.005.

3 Experiments

3.1 Latch and 2-Sequence Problems

The first task is to observe and classify input sequences. There are two classes. There is only one input unit or input line. See, e.g., Bengio et al. (1994); Bengio and Frasconi (1994); Lin et al. (1995). The *latch problem* was designed to show how gradient descent fails. We tested two variants, both with three free parameters.

Latch variant I. The recurrent network itself has a single free parameter: the recurrent weight of a single *tanh* unit. The remaining two free parameters u_1 and u_2 are the initial input values, one for each sequence class: For sequences of the first class, the first input value of the sequence is the free parameter u_1, while for sequences of the second class, the first input value of the sequence is the free parameter u_2. These two free parameters allow to simulate and simplify the situation in which another sub-network computes the bit which should be stored in the latch. If it is not even possible to learn u_1 and u_2 using the gradient, then it would not have been possible to train this sub-network to compute the right output. The output targets at sequence end are +0.8 and -0.8. In an online fashion, Gaussian noise with mean zero and variance 0.2 is added to each sequence element except the first, at each presentation. Hence a large positive recurrent weight is necessary to accomplish long-term storage of the bit of information identifying the class determined by the initial input. The latter's absolute value must be large to allow for latching the recurrent unit.

RG solves the task within only 6 trials on average (mean of 40 simulations). This is better than the 1600 trials reported in Bengio et al. (1994) with several methods. RG's success is due to the few parameters and the fact that in search space $[-100, 100]^3$ it almost suffices to get the parameter signs right (there are only 8 sign combinations).

Latch variant II. Sequences from class 1 start with 1.0; others with -1.0. The targets at sequence end are 1.0 and -1.0. The recurrent network has a single unit with *tanh* activation function. There are 3 incoming connections:

one from itself, one from the input, and one bias connection (the inputs are not free parameters). Gaussian noise is added as in variant I.

RG solves variant II within 22 trials on average. This is better than the 1600 trials reported by Bengio et al. (1994) with several methods on the simpler variant I.

2-sequence problem. Only the first N real-valued sequence elements convey relevant information about the class (the inputs are not free parameters). Again we will consider two cases: $N = 1$, and $N = 3$ (used in Bengio et al. 1994). In the first case ($N = 1$) we set the first sequence element to 1.0 for class 1, and -1.0 for class 2. The output neuron has a sigmoid activation function; the target at sequence end is 1.0 for class 1 and 0.0 for class 2. In case $N = 3$ two 3-vectors in $[-1, 1]^3$ are randomly chosen in advance — they represent the initial subsequences determining the sequence class. $N > 1$ was chosen because the probability of obtaining random initial patterns with significant differences increases with N. Sequence elements at positions $t > N$ are generated by a Gaussian with mean zero and variance 0.2.

The best method among the six tested by Bengio et al. (1994) solved the 2-sequence problem (N=3) after 6400 sequence presentations, with a final classification error rate of 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 2900 trials.

RG with architecture A2 and $N = 1$ solves the problem within 718 trials on average. RG with A1 ($n = 1$ in the architecture) requires 1247 trials, and reaches zero classification error on all trials.

With $N = 3$, however, RG requires around 68,000 trials on average to find a solution (mean of 40 simulations). If we ignore that RG trials need much less computation time than EM trials then EM seems faster. RG's difficulties stem from the continuous nature of the search space. With $N > 1$, two tasks must be learned simultaneously: recognizing a multi-dimensional pattern (class 1 or 2), and latching it (storing one bit for the long term). The pattern recognition task is harder for RG.

3.2 Parity Problem

The second task requires to classify sequences consisting of 1's and -1's according to whether the number of 1's is even or odd (Bengio et al. 1994; Bengio and Frasconi 1994). The output neuron has a sigmoid activation

function; the target at sequence end is 1.0 for odd and 0.0 for even. Bengio et al. (1994) also add [-0.2, 0.2] uniform noise to the sequence elements.

For sequences with lengths between only 25 and 50 steps, among the six methods tested by Bengio et al. (1994), only simulated annealing was reported to achieve zero final classification (within about 810,000 trials). A method called *discrete error BP* took about 54,000 trials to achieve final classification error of 0.05. In Bengio and Frasconi's more recent work (1994), for sequences with 250-500 steps, an Input/Output Hidden Markov Model (IOHMM) took about 3400 trials to achieve final classification error of 0.12. Comparable results were obtained with and without noise.

Two RG cases are considered. In the first case, the inputs are binary +1's and -1's. In the second case, uniform noise in [-0.2, 0.2] is added to the input. With no noise, RG with A1 ($n = 1$, identical to Bengio et al.'s 1994 architecture) solves the problem within 2906 trials on average. This is comparable to or better than the best already reported results. RG with A2 (no noise) solves it within 2797 trials. With architecture A2, but without self-connections for the hidden units, RG solves no-noise parity within 250 trials on average. This is much better than any previously reported results. Such excellent results can be obtained even with sequence lengths exceeding 500.

In case of noisy inputs, however, RG (with architecture A1) requires 41,400 trials on average. Although RG trials take only half the time of IOHMM trials, IOHMMs were much faster. This suggests that adding input noise may be a way to break down candidate learning algorithms relying a lot on random search.

3.3 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 focus here on a few "parity-free" Tomita grammars (the grammars #1, #2, and #4). Most previous work facilitated the learning problem by restricting sequence length. E.g., Miller and Giles' maximal test sequence length is 15, and maximal training sequence length is 10. Miller and Giles (1993) report the number of sequences required for convergence (for various first and second order networks with

3 to 9 units): Tomita #1: 23,000 – 46,000; Tomita #2: 77,000 – 200,000; Tomita #4: 46,000 – 210,000. RG, however, performs better in these cases (as always, we use the experimental conditions described in section 2). The average results are: Tomita #1: 182 (with A1, $n = 1$) and 288 (with A2), Tomita #2: 1511 (with A1, $n = 3$) and 17953 (with A2), Tomita #4: 13833 (with A1, $n = 2$) and 35610 (with A2).

It should be mentioned, however, that by using our architectures and very short training sequences (in the style of Miller & Giles) one can achieve reasonable results with gradient descent, too.

4 Final Remarks

RG will not work well in case of complex architectures with many free parameters except when solutions are rather dense in weight space. In fact, RG-tests are useful for discovering whether this is the case or not. If so, then the problem will not be a convincing benchmark. Our experiments with A2 (which has comparatively many hidden units and free parameters) indeed provide examples of such cases.

A problem solvable by RG on a certain architecture, however, might still be a useful benchmark for a different architecture.

Successful RG typically hits "flat minima" of the error function (Hochreiter and Schmidhuber, 1997b), that is, regions in weight space where the training error is (a) low and (b) hardly affected by perturbations of the weight vector. The reason is: flat minima correspond to fat maxima of the posterior. Actually they correspond to large regions in weight space where solutions are dense in the mathematical sense. In other words: RG works well on problems where the precise weight values do not matter — and flat minima precisely correspond to low-precision weights.

RG's success depends on sufficient parameter initialization intervals (Kolen & Pollack, 1991). For instance, given a particular architecture, the intervals [-0.1,0.1] and [-5,5] may lead to quite different search results. Of course, the success of algorithms other than RG also heavily depends on their parameter initialization intervals, and on the length of the time intervals between re-initializations.

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. Compare, for instance, Gallant (1990).

Of course, we do not intend to say that RG is a good algorithm — it is just a reasonable first step towards benchmark evaluation. We would never use RG in realistic applications. Realistic tasks require either many free parameters (e.g., input weights) or high weight precision (e.g., for continuous-valued parameters), such that RG becomes completely infeasible. For example, Schmidhuber's task (1992) requires to memorize one particular event among numerous other locally represented events. This in turn requires numerous different input units and too many input weights to be guessed within reasonable time.

Some of the tasks mentioned above may be easily solvable by non-neural, symbolic methods such as Fu's and Booth's (1975) or graph search heuristics such as Lang's (1992), perhaps even much faster than by RG. The comparisons in this paper, however, are limited to various recent long time lag algorithms for neural networks. Note also that symbolic methods tend to fail in presence of noise — RNNs do not.

We are aware of two neural methods that have been successfully applied to long time lag tasks that RG cannot solve in reasonable time due to too many free parameters: the *sequence chunker* (Schmidhuber, 1992) and *Long Short-Term Memory* (LSTM — Hochreiter and Schmidhuber 1997a, 1997c). The chunker's applicability is limited to compressible sequences, LSTM's is not. LSTM eliminates some of gradient-based approaches' problems and can solve complex long time lag tasks involving distributed, high-precision, continuous-valued representations. Such tasks cannot be quickly solved by RG nor by any other method we are aware of. Other interesting long time lag approaches have been presented in papers by El Hihi and Bengio (1995) and Lin et al. (1995). They suggest mechanisms such as long time delays in recurrent connections to allow for handling proportionally longer temporal dependencies. Some ideas in these papers are also related to the ideas of handling long-term dependencies by introducing multiple time scales or a hierarchy of state variables (Mozer, 1992; Schmidhuber, 1992; Saul and Jordan, 1996; Jordan et al., 1997).

5 Conclusion

Previously proposed long time lag algorithms can do significantly worse than simple RG for certain already proposed benchmark tasks. In other cases (e.g., involving subtasks such as pattern recognition in continuous spaces, or additional input noise), RG does not do as well. This suggests that simple-minded strategies such as RG should be routinely included among algorithms against which newly introduced long time lag algorithms are compared. Furthermore, new benchmark tasks should be designed to make simple random search algorithms fail on them. In particular, we recommend tasks involving distributed and/or real-valued input and output representations.

For more sophisticated weight guessing biased towards networks with low Levin complexity, see Schmidhuber (1997).

An obvious extension of RG will apply gradient-based optimization phases to certain parameter initializations found by RG. In general, however, it is not obvious *a priori* how to optimally allocate overall computation time to more expensive gradient iterations as opposed to faster but less precise RG trials.

6 Acknowledgments

S. H. is supported by *DFG grant SCHM 942/3-1* from *Deutsche Forschungsgemeinschaft*. Y.B. would like to thank the National Sciences and Engineering Research Council of Canada for its support.

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*, pages 493–499. San Mateo, CA: Morgan Kaufmann.

Fu, K. S. and Booth, T. L. (1975). Grammatical inference: Introduction and survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 5:95.

Gallant, S. I. (1990). A connectionist learning algorithm with provable generalization and scaling bounds. *Neural Networks*, 3:191–201.

Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München. See www7.informatik.tu-muenchen.de/~hochreit.

Hochreiter, S. and Schmidhuber, J. (1997a). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hochreiter, S. and Schmidhuber, J. (1997b). Flat minima. *Neural Computation*, 9(1): 1–42.

Hochreiter, S. and Schmidhuber, J. (1997c). LSTM can solve hard long time lag problems. In M. C. Mozer, M. I. Jordan, T. Petsche, eds., *Advances in Neural Information Processing Systems 9,* pages 473–479, MIT Press, Cambridge MA, 1997.

Jordan, M., Ghahramani, Z., and Saul, L. (1997). Hidden markov decision trees. In *Advances in Neural Information Processing Systems 9*, volume 9, Cambridge, MA. MIT Press.

Kolen, J.F., and Pollack, J. B. (1991). Back Propagation is Sensitive to Initial Conditions. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., eds., *Advances in Neural Information Processing Systems 3*, pages 860-867, San Mateo, CA: Morgan Kaufmann.

Lang, K. J. (1996). Random dfa's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory.*

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.

Mozer, M. C. (1992). Induction of multiscale temporal structure. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 275–282. San Mateo, CA: Morgan Kaufmann.

Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228.

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

Saul, L. and Jordan, M. (1996). Exploiting tractable substructures in intractable networks. In Mozer, M., Touretzky, D., and Perrone, M., editors, *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge, MA.

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

Schmidhuber, J. (1997). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

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 NU-CCS-89-27, Boston: Northeastern University, College of Computer Science.