Scalable Neural Networks for Board Games

Tom Schaul and Jürgen Schmidhuber

IDSIA, Galleria 2, Manno-Lugano, Switzerland
{tom,juergen}@idsia.ch

Abstract. Learning to solve small instances of a problem should help in solving large instances. Unfortunately, most neural network architectures do not exhibit this form of scalability. Our Multi-Dimensional Recurrent LSTM Networks, however, show a high degree of scalability, as we empirically show in the domain of flexible-size board games. This allows them to be trained from scratch up to the level of human beginners, without using domain knowledge.

1 Introduction

In a wide range of domains it is possible to learn from a simple version of a problem and then use this knowledge on a larger one. This particular form of incremental learning is commonly employed by humans, and for machine learning it is especially useful when training on the large version is much more expensive.

Board games are a particularly suitable domain for investigating this form of *scalability*, because for many of them either the board size can be varied, or the rules can be trivially adjusted to make it variable. In addition, despite being described by a small set of formal rules, they often involve highly complex strategies. One of the most interesting board games is the ancient game of *Go* (among other reasons, because computer programs are still much weaker than human players), which can be solved for small boards [1] but is very challenging for larger ones [2,3]. Its extremely large search space defies traditional search-based methods. Human experts rely heavily on patterns, and thus it is not surprising that a substantial amount of research effort has been devoted to applying neural networks – which are good at pattern recognition – to Go [2,4]. Unfortunately most of these methods do not scale well w.r.t. board size, i.e. networks trained successfully on small boards (where training is efficient) do not play well when the board is enlarged [5,3]. The present paper builds on the promising preliminary results [6,7] of a scalable approach based on Multi-dimensional Recurrent Neural Networks (MDRNNs; [8,9]) and enhances the ability of that architecture to capture long-distance dependencies.

We conduct experiments on three different Go-inspired games, which is possible without modifying our network architecture as it is free of *domain knowledge*. We train it against opponents of varying difficulty and measure how the playing performance scales to larger board sizes. Furthermore, we put our architecture into context by comparing it to a number of competing ones.

C. Alippi et al. (Eds.): ICANN 2009, Part I, LNCS 5768, pp. 1005–1014, 2009.

2 Scalable Neural Architectures

We consider a neural network architecture to be scalable if it is not tied to a fixed input dimension. This section provides an overview of such architectures that have been proposed for solving board games, it then describes MDRNNs in general and finally gives the details of the specific instantiation we propose.

2.1 Background

One approach to designing scalable network architectures is to scan across the board, processing the inputs from a limited *receptive field*, independently of their positions. The outputs of that stage are then fed into another architecture that combines them (e.g. [10]). An extension of this idea is the *convolutional network* [11], which repeats this step on multiple levels, thereby capturing higher-level features instead of just local patterns. These architectures introduce a trade-off: a small receptive field severely limits the kind of patterns that can be recognized, whereas a large one makes learning very difficult (because of the exploding number of parameters).

'Roving-eye'-based architectures [5] contain one component with a fixed receptive field that can be aimed at any part of the board. This is then combined with an active component that decides where to rove over the board, and when to choose an output action.

Other architectures have been proposed [12,13] which make use of weight-sharing to capture domain-specific symmetries, but these are limited to a particular game, and also restricted w.r.t. what kind of strategies they can learn.

For the related but different problem of scaling the problem *resolution*, a number of approaches for generative encodings of neural networks have been found to be successful (e.g. Compositional Pattern Producing Networks [14]).

2.2 MDRNNs

Multi-dimensional Recurrent Neural Networks [8,9], are an extension of bi-directional RNN proposed by Schuster [15], and a special case of the DAG-RNNs proposed by Baldi [16]. Their unbounded receptive fields (explained below) make them scalable by design. Successful applications include vision [8], handwriting recognition [17], and supervised learning of expert Go moves [4].

Unlike standard recurrent neural networks (RNNs) which are only effective for handling sequences with a single (time-)dimension, MDRNNs are are applicable to multi-dimensional sequences [9]. In the case of Go, the single time dimension is replaced by the two space dimensions of the game board.

Consider a hidden layer $h_{\nearrow}$ that *swipes* diagonally over the board from bottom-left to top-right. At each board position (i, j) its activation $h_{\nearrow(i,j)}$ is a function of the current input $in_{i,j}$ and its own earlier activations $h_{\nearrow(i-1,j)}$ and $h_{\nearrow(i,j-1)}$:

$$h_{\nearrow(i,j)} = f\left(w_i * in_{i,j} + w_h * h_{\nearrow(i-1,j)} + w_h * h_{\nearrow(i,j-1)}\right) \tag{1}$$

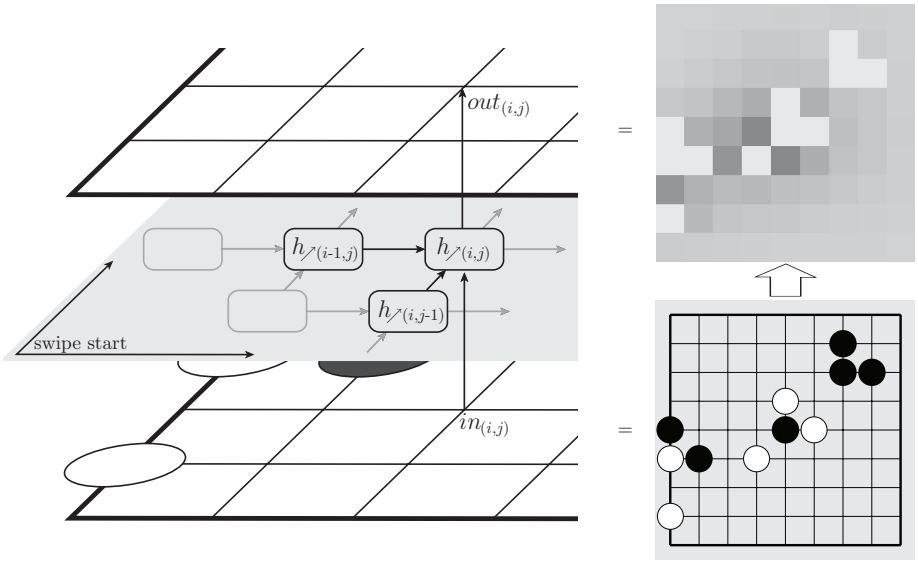


Fig. 1. MDRNN for Go The structure diagram on the left shows the connections of a hidden layer in one of the swiping directions: it receives its two earlier activations, as well as the local board input. The network as a whole takes the game board as input (bottom right) and outputs move preferences (top right). The darker the square, the higher the preference for the corresponding move (illegal ones are ignored).

where w_* are the connection weights. On the boundaries we use a fixed default value $h_{\nearrow(0,i)} = h_{\nearrow(i,0)} = w_b$, for $0 < i \leq n$. See also Figure 1 for an illustration.

Because of the recurrency, the layer has indirect access to board information from the whole rectangle between $(0, 0)$ and (i, j). In order to have access to the whole board, we use 4 swiping layers, one for each of the diagonal swiping directions in $D = \{\searrow, \nearrow, \swarrow, \nwarrow\}$. The output layer then, for every position, combines the outputs of these 4 layers into a single value $out_{i,j}$ (which is indirectly derived from the information of the entire input). More formally:

$$out_{i,j} = g\left(\sum_{\diamondsuit \in D} w_o * h_{\diamondsuit(i,j)}\right) \tag{2}$$

where the function g is typically the sigmoid function.

2.3 Proposed Architecture

We instantiate MDRNNs such that they are appropriately generating a playing policy, given a symmetric game board. At each position, the network takes two *inputs* which indicate the presence of a stone at this position. The first one is 1 if a stone of the network's own color is present and 0 otherwise, the second input encodes the presence of an opponent's stone in the same way. A black/white

symmetric encoding, as used in other approaches (e.g. [12]) is not applicable here, because the output is not symmetrical: the best move for both players might be the same.

The *output* value at each position expresses the network's *preference* for playing there (see also Figure 1). Assuming that a deterministic playing behavior is desired, moves are selected greedily, randomly breaking ties. This is the case in our experiments because the opponents act stochastically. In practice, we ignore the network's preferences for illegal moves. For stochastic play one can interpret the preferences probabilistically, e.g. by drawing a position from their Gibbs distribution.

MDRNNs are invariant w.r.t. stationary shifts of the input. In order to also enforce rotation and reflection symmetry, we use the same connection weights for all swiping directions and the same w_b on all boundaries.

Typically a swiping layer u is composed of k sigmoidal neurons (e.g. $f = \tanh$). Although in theory such an MDRNN can learn to make use of the whole board context, it is very difficult to achieve in practice, because the information is propagated recurrently through non-linear units and thus tends to decay quickly with distance [18]. One solution to this problem is to use *Long short-term memory* cells (LSTM), which are based on protecting the recurrent state with *gating* subunits [18]. As in [8,9] (and in contrast to [6]), we therefore use a swiping layer composed of k LSTM cells and call it *MDLSTM*.

In our implementation we unfold the MDRNN along both spacial dimensions, leading to a large but simple feed-forward network. On a normal desktop computer (Intel Xeon 2.8 GHz), a network needs about 3ms to choose a move on a 7x7 board, and 25ms on a 19x19 board. The total number of weights is $4k + k^2$ for sigmoidal swiping layers and $5k^2 + 16k$ for LSTM layers. All the code used for this paper is available as part of the PyBrain library at `www.pybrain.org`.

3 Methodology

We conduct our experiments on a number of different flexible-size board games, all of which are played on the Go board:

Atari-Go: Also known as Ponnuki-Go or 'Capture Game', is a simplified version of Go that is widely used for teaching the game of Go to new players. The rules are the same as for Go, except that passing is not allowed, and the first player to capture a predetermined number (here: one) of his opponent's stones wins.

Go-Moku: Is also known as 'Five-in-a-row'. Players alternate putting stones onto any of the intersections on the board. The first player to have 5 connected stones in a row, column or diagonal, wins.

Pente: Has similar rules to Go-Moku, except that it is now possible to capture stones, in pairs, by putting stones at both ends of a pair of the opponent. The game is won by the first player who either has 5 connected stones, or has captured 5 pairs.

Each game has a number of predefined opponents associated with it: *a*) a *random* player, which randomly chooses any of the legal moves, *b*) a *naive* player, which does a one-ply search. If possible, it always picks a move that makes it win the game immediately, and never picks a move that would make it lose the game immediately. In all other cases (the large majority), it randomly picks a legal move, *c*) a publicly available *heuristic* player (only for Atari-Go), based on a set of hand-coded heuristic tactics (exploited greedily [7,19]). Its difficulty can be adjusted by imposing that a proportion ϵ of its moves are chosen randomly. According to the author, the level of play (with $\epsilon = 0$) is 'challenging for beginners'.

As *fitness* we use the average outcome of 100 games against a fixed opponent, counting a win as 1, a draw as 0 and a loss as -1. Each player plays 50 times as black and 50 times as white.

In addition to MDRNNs with sigmoidal neurons or LSTM cells (as described in section 2.3), we use – as a performance baseline – standard multi-layer perceptrons (MLP), containing a single fully connected hidden layer of size k, with tanh units. We compare the performance of our architecture to simple convolutional networks (CONV), with one layer of k feature maps (of identical receptive field size ρxρ), no subsampling, and a sigmoid output layer that combines the features. Here, the input board is padded with additional positions around the borders. They have $k(\rho^2 + 1) + 1$ parameters.

One the one hand, we analyze the performance of networks produced by the simplest possible algorithm, namely random weight guessing (using the normal distribution $N(0, 1)$). On the other hand we train the networks using the the well-established Covariance Matrix Adaptation Evolution Strategy (CMA-ES [20]) to optimize all the weights.

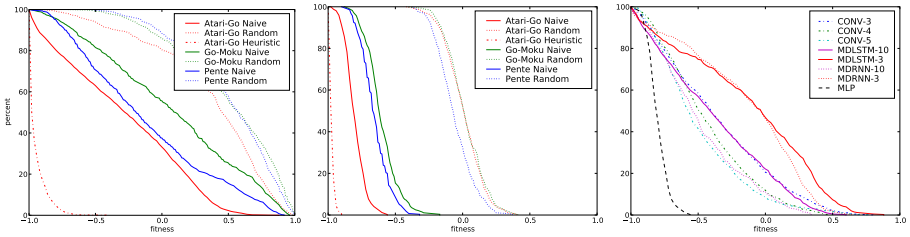
Our two quantitative measures of scalability are: *a*) the linear correlation (Pearson coefficient) between the fitness on different board sizes *b*) the proportion p of networks for which the fitness is higher on the larger board than on the smaller one.

4 Results

As training networks is expensive, we start by empirically investigating the performance of the different networks (and their parameters) with random weights. Moreover, we determine under what circumstances the performance scales to larger boards. We then train the networks on small boards and analyze whether their performance improvement is transferred to larger boards.

4.1 Random Networks

We measure the performance of different kinds of networks with randomly guessed weights, on different games and against various opponents. Figure 2(a) shows the percentiles of fitnesses of random MDRNNs, giving an indication of the difficulty of the different opponents on each game. Training is easier if initial weights with



(a) MDRNNs on different tasks. (b) MLPs on different tasks. (c) Different networks.

Fig. 2. Fitnesses of random networks evaluated on 7x7 (400 networks per scenario). The percentiles show what proportion of random networks reach at least a given fitness (e.g. at least 25% or random MDRNNs win at least $\frac{3}{4}$ of Go-Moku games against the naive opponent, i.e. have a fitness of 0.5). Figure 2(c) shows the differences between a number of different networks (on Atari-Go, vs. Naive).

reasonably good fitness (> 0) can be found relatively easily. This is indeed the case for the naive and the random opponent but not for the heuristic one. For MLPs, however, reasonable initial weights are very rare (Figure 2(b)).

Figure 2(c) more explicitly shows the differences between the network architectures, comparing MDRNNs, MDLSTMs (for varying k), CONVs (for varying ρ) and MLPs. Despite only corresponding to random networks, the results indicate that small values of k are appropriate for MDRNNs (we will fix $k = 3$ hereafter), and do not bode well for MLPs.

We determine the scalability of random networks by evaluating the fitness on multiple board sizes and then computing their correlation (see Table 1). As the linear correlation by itself can be a misleading measure, we provide a visual intuition about the high correlation in Figure 4(a). The results indicate that one can train networks on boards as small as 7x7 and use them to play on 19x19, for all three games.

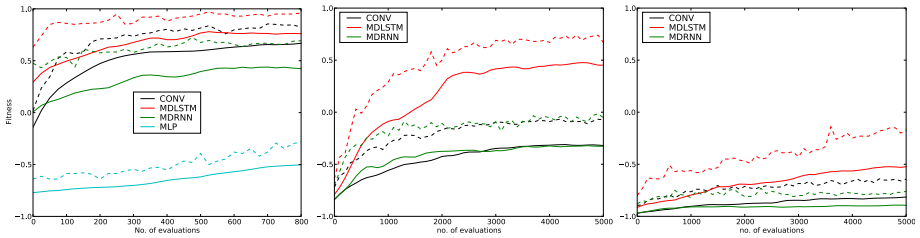
4.2 Trained Networks

Figure 3(a) shows the learning curves for different networks trained on Atari-Go against the naive player on 7x7, using CMA-ES to optimize the weights. MLPs are in this comparison as a baseline, but clearly fail to learn how to play. The other architectures learn to beat the naive opponent, with MDLSTMs clearly outperforming the others. Convolutional networks are learning slightly slower, but still faster than MDRNNs. The same conclusions also hold for the results on Go-Moku and Pente (results not shown).

Learning to play against the significantly stronger heuristic opponent is a bigger challenge. Figures 3(b) and 3(c) show the learning curves against the heuristic player with settings $\epsilon = 0.2$ and $\epsilon = 0.05$ respectively (averaged over 5 runs). Here, MDLSTMs clearly outperform the convolutional networks, for which only the best results are shown (produced with $\rho = 5$). We suspect that

Table 1. Correlations between the fitnesses of random MDLSTMs on different board sizes (based on 100 networks per scenario, evaluated against the naive opponent). They are high in all cases except Go-Moku between 5x5 and larger boards (which is due to the fact that it is disproportionately easier for a game to end in a draw on a 5x5 board).

Sizes	Atari-Go	Go-Moku	Pente
5x5 vs. 7x7	0.86	0.20	0.47
5x5 vs. 9x9	0.72	0.09	0.31
5x5 vs. 11x11	0.67	0.37	0.49
5x5 vs. 19x19	0.37	0.38	0.46
7x7 vs. 9x9	0.88	0.83	0.83
7x7 vs. 11x11	0.82	0.85	0.87
7x7 vs. 19x19	0.62	0.59	0.64
9x9 vs. 11x11	0.92	0.92	0.90
9x9 vs. 19x19	0.71	0.76	0.64



(a) Naive opponent. (b) Heuristic opponent ($\epsilon = 0.2$). (c) Heuristic opponent ($\epsilon = 0.05$).

Fig. 3. Learning curves for training against different opponents on Atari-Go (board size 7x7, averaged over 10 independent runs). The solid line corresponds to the average fitness per generation, the broken one corresponds to the best.

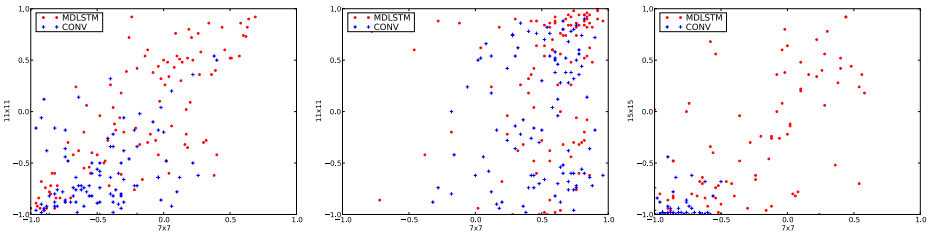
this is due to the limited receptive field: at a certain level of play it simply becomes necessary to use non-local information. MDLSTMs can *learn* how much context is necessary, and automatically increase their effective receptive field during training.

The scalability results for trained networks are summarized in Table 2. Generally, there is a low correlation for the convolutional networks, but a relatively high one for MDLSTMs. Figure 4(b) illustrates this difference for networks trained against the naive opponent in Atari-Go. Note the large number of convolutional networks on the bottom right, for which good performance on 7x7 corresponds to poor performance on 11x11.

Comparing the correlations and the proportions p to the values for random MDLSTMs, we find the correlations to be lower but p to be higher (especially when scaling to 19x19). This means that the fitness on a small board is not perfectly predictive of the fitness on the large board, *but* it is almost always significantly higher on the large board.

Table 2. Scalability of networks trained on 7x7 against the naive opponent (based on 100 networks per scenario). The correlations are higher for MDLSTMs than for convolutional networks. Also, note the really high proportion p of MDLSTMs that are stronger on 19x19 than on 7x7, for all games.

Game	Sizes	CONV		MDLSTM	
		Correlation	p	Correlation	p
Atari-Go	7x7 vs. 9x9	0.13	0.20	0.38	0.48
Atari-Go	7x7 vs. 11x11	0.17	0.18	0.27	0.45
Atari-Go	7x7 vs. 19x19	0.17	0.21	0.38	0.76
Go-Moku	7x7 vs. 9x9	0.06	0.42	0.47	0.67
Go-Moku	7x7 vs. 11x11	0.15	0.61	0.38	0.87
Go-Moku	7x7 vs. 19x19	0.04	0.68	0.66	0.84
Pente	7x7 vs. 9x9	0.05	0.45	0.08	0.61
Pente	7x7 vs. 11x11	0.24	0.58	0.39	0.79
Pente	7x7 vs. 19x19	0.23	0.58	-0.05	0.95



(a) Random, naive opponent. (b) Trained, naive opponent. (c) Trained, heuristic opp. ($\epsilon = 0.2$).

Fig. 4. Illustrations of fitness scalability on Atari-Go. The points correspond to MDLSTMs, the crosses to convolutional networks.

Of particular interest is the scalability of the networks trained against the strongest opponent – as figure 4(c) illustrates (for 7x7 versus 15x15), MDLSTMs achieve a high correlation (0.64), unlike convolutional networks (0.08).

5 Discussion

MDRNNs are scalable because they are approximately translation-invariant, in addition to capturing the board symmetries. They handle the same situation on a bigger board (with empty positions around it) in the exact same way as on a smaller board. This allows them to use a comparatively low number of weights (MDRNNs and MDLSTMs with $k = 3$ have 21 and 93 weights respectively, independently of board size), thereby reducing the dimensionality of the search space and making training efficient. Incorporating LSTM cells in the swiping layer further enabled the architecture to better handle long-distance context, which is necessary for learning complex strategies, as our experiments versus the

heuristic opponent show. Finally, the results show that MDLSTMs transfer the strategies learned on small boards to large ones, leading to a level of play on 15x15 that is on par with human beginners.

Directly training against a stronger opponent (e.g. $\epsilon = 0.0$) is like finding a needle in a haystack, as almost all initial networks will lose all games. In future work we will attempt to address this issue by training against incrementally harder opponents [21]. We expect to eventually reach a limit on the complexity of strategies that MDLSTMs can represent. In that case we propose increasing their representative power by stacking two (or more) MDLSTMs on top of each other, the lower one producing a map of high-level features that the top one scans over. MDLSTMs performed equally well on a number of different games, which we attribute to them being free from domain knowledge. However, in case performance is preferred over generality, our approach can easily be extended to make use of such knowledge, e.g. by feeding the network a number of domain-specific features [4] instead of the raw board. Due to the generality of our approach and the similarity of our three games to Go, we expect our positive results to carry over the game of Go itself, which we intend to investigate as our next step.

6 Conclusion

We have developed and investigated the properties of MDLSTMs, a scalable neural network architecture based on MDRNNs and LSTM cells. By validating our results on three different games, we showed that it is suitable for the domain of board games. Further, we found that training lead to an impressive level of play, given that they are devoid of domain knowledge and learn the games from scratch. Finally, the networks trained on small boards scale very well to large ones.

Acknowledgments

This research was funded by the SNF grant 200021-113364/1. We would like to thank Mandy Grüttner for building the framework to interact with the heuristic player, Justin Bayer for speeding up PyBrain to such a level that our extensive experiments became feasible, as well as Faustino Gomez for the constructive advice.

References

1. van der Werf, E., van den Herik, H.J., Uiterwijk, J.: Solving Go on small boards. International Computer Games Association Journal 26 (2003)
2. Richards, N., Moriarty, D.E., Miikkulainen, R.: Evolving neural networks to play Go. Applied Intelligence 8, 85–96 (1997)
3. Runarsson, T.P., Lucas, S.M.: Co-evolution versus Self-play Temporal Difference Learning for Acquiring Position Evaluation in Small-board Go. IEEE Transactions on Evolutionary Computation, 628–640 (2005)

4. Wu, L., Baldi, P.: A Scalable Machine Learning Approach to Go. In: Schölkopf, B., Platt, J., Hoffman, T. (eds.) Advances in Neural Information Processing Systems 19, pp. 1521–1528. MIT Press, Cambridge (2007)
5. Stanley, K.O., Miikkulainen, R.: Evolving a Roving Eye for Go. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 1226–1238. Springer, Heidelberg (2004)
6. Schaul, T., Schmidhuber, J.: A Scalable Neural Network Architecture for Board Games. In: Proceedings of the IEEE Symposium on Computational Intelligence in Games. IEEE Press, Los Alamitos (2008)
7. Grüttner, M.: Evolving Multidimensional Recurrent Neural Networks for the Capture Game in Go. Bachelor Thesis, Techniche Universität München (2008)
8. Graves, A., Fernández, S., Schmidhuber, J.: Multidimensional Recurrent Neural Networks. In: Proceedings of the 2007 International Conference on Artificial Neural Networks (September 2007)
9. Graves, A.: Supervised Sequence Labelling with Recurrent Neural Networks, Ph.D. in Informatics, Fakultat für Informatik – Technische Universität München (2008)
10. Silver, D., Sutton, R.S., Müller, M.: Reinforcement Learning of Local Shape in the Game of Go. In: IJCAI, pp. 1053–1058 (2007)
11. Lecun, Y., Bengio, Y.: Convolutional Networks for Images, Speech and Time Series, pp. 255–258. The MIT Press, Cambridge (1995)
12. Schraudolph, N.N., Dayan, P., Sejnowski, T.J.: Temporal Difference Learning of Position Evaluation in the Game of Go. In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing Systems, vol. 6, pp. 817–824. Morgan Kaufmann, San Francisco (1994)
13. Freisleben, B., Luttermann, H.: Learning to Play the Game of Go-Moku: A Neural Network Approach. Australian Journal of Intelligent Information Processing Systems 3(2), 52–60 (1996)
14. Gauci, J., Stanley, K.: Generating large-scale neural networks through discovering geometric regularities. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 997–1004 (2007)
15. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing 45, 2673–2681 (1997)
16. Baldi, P., Pollastri, G.: The principled design of large-scale recursive neural network architectures DAG-RNNs and the protein structure prediction problem. Journal of Machine Learning Research 4, 575–602 (2003)
17. Graves, A., Schmidhuber, J.: Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In: NIPS (2008)
18. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(9), 1735–1780 (1997)
19. Gherman, S.: Atari-Go Applet (2000), `http://www.361points.com/capturego/`
20. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
21. Gomez, F., Miikkulainen, R.: Incremental Evolution of Complex General Behavior. Adaptive Behavior 5, 317–342 (1997)