Learning and Generalization in Random Automata Networks

Alireza Goudarzi[1], Christof Teuscher[2], and Natali Gulbahce[3]

[1] Computer Science and Systems Science Department
Portland State University (PSU), Portland, OR, USA
alirezag@cecs.pdx.edu
[2] Department of Electrical and Computer Engineering
Portland State University (PSU), Portland, OR, USA
teuscher@pdx.edu
[3] Department of Cellular and Molecular Pharmacology
University of California, San Francisco (UCSF), CA, USA
natali.gulbahce@ucsf.edu

Abstract. It has been shown [7,16] that feedforward Boolean networks can learn to perform specific simple tasks and generalize well if only a subset of the learning examples is provided for learning. Here, we extend this body of work and show experimentally that random Boolean networks (RBNs), where both the interconnections and the Boolean transfer functions are chosen at random initially, can be evolved by using a state-topology evolution to solve simple tasks. We measure the learning and generalization performance, investigate the influence of the average node connectivity K, the system size N, and introduce a new measure that allows to better describe the network's learning and generalization behavior. Our results show that networks with higher average connectivity K (supercritical) achieve higher memorization and partial generalization. However, near critical connectivity, the networks show a higher perfect generalization on the even-odd task.

1 Introduction

Pattern recognition is a task primates are generally very good at while machines are not so much. Examples are the recognition of human faces or the recognition of handwritten characters. The scientific disciplines of machine learning and computational learning theory have taken on the challenge of pattern recognition since the early days of modern computer science. A wide variety of very sophisticated and powerful algorithms and tools currently exist [5]. In this paper we are going back to some of the roots and address the challenge of learning with networks of simple Boolean logic gates. To the best of our knowledge, Alan Turing was the first person to explore the possibility of learning with simple NAND gates in his long forgotten 1948 paper, which was published much later [21,25]. One of the earliest attempts to classify patterns by machine came from Olivier Selfridge [19,20] in 1958. Later, many have explored random logical nets made up from Boolean or threshold (McCulloch-Pitts) neurons: [1–4, 18]. Martland [15] showed that it is possible to predict the activity of a boolean network with randomly connected inputs, if the characteristics of the boolean neurons can be described probabilistically. In a second paper [14], Martland illustrated how the boolean networks are used to store and retrieve patterns and even pattern sequences auto-associatively. Seminal contributions on random Boolean networks came from Stuart Kauffman [9–11] and Weisbuch [26, 27].

In 1987, Patarnello and Carnevali [6, 16] used *simulated annealing* and in 1989 also *genetic algorithms* [17] as a global stochastic optimization technique to train feedforward Boolean networks to solve computational tasks. They showed that such networks can indeed be trained to recognize and generalize patterns. Broeck and Kawai [7] also investigated the learning process in feedforward Boolean networks and discovered their amazing ability to generalize.

In 2007, Teuscher *et al.* [22] presented preliminary results that true RBNs, i.e., Boolean networks with recurrent connections, can also be trained to learn and generalize computational tasks. They further hypothesized that the performance is best around the critical connectivity $K = 2$.

In the current paper, we extend and generalize Patarnello and Carnevali's results to random Boolean networks (RBNs) and use genetic algorithms to evolve both the network topology and the node transfer functions to solve a simple task. Our work is mainly motivated by the application of RBNs in the context of emerging nanoscale electronics [23]. Such networks are particularly appealing for that application because of their simplicity. However, what is lacking is a solid approach that allows to train such systems for performing specific operations. Similar ideas have been explored with none-RBN building blocks by Tour *et al.* [24] and by Lawson and Wolpert [12]. One of the broader goals we have is to systematically explore the relationship between generalization and learning (or memorization) as a function of the system size, the connectivity K, the size of the input space, the size of the training sample, and the type of the problem to be solved. In the current paper, we restrict ourselves to look at the influence of the system size N and of connectivity K on the learning and generalization capabilities. In the case of emerging electronics, such as for example self-assembled nanowire networks use to compute simple functions, we are interested to find the smallest network with the lowest connectivity that can learn how to solve the task with the least number of patterns presented.

2 Random Boolean Networks

A *random Boolean network* (RBN) [9–11] is a discrete dynamical system composed of N nodes, also called *automata*, *elements* or *cells*. Each automaton is a Boolean variable with two possible states: $\{0, 1\}$, and the dynamics is such that

$$\mathbf{F} : \{0,1\}^N \mapsto \{0,1\}^N, \tag{1}$$

where $\mathbf{F} = (f_1, ..., f_i, ..., f_N)$, and each f_i is represented by a look-up table of K_i inputs randomly chosen from the set of N nodes. Initially, K_i neighbors and a look-up table are assigned to each node at random. Note that K_i (i.e., the fan-in) can refer to the *exact* or to the *average* number of incoming connections per node. In this paper we use K to refer to the average connectivity.

A node state $\sigma_i^t \in \{0,1\}$ is updated using its corresponding Boolean function:

$$\sigma_i^{t+1} = f_i(\sigma_{i_1}^t, \sigma_{i_2}^t, ..., \sigma_{i_{K_i}}^t). \tag{2}$$

These Boolean functions are commonly represented by *look-up tables* (LUTs), which associate a 1-bit output (the node's future state) to each possible K-bit input configuration. The table's out-column is called the *rule* of the node. Note that even though the LUTs of a RBN map well on an FPGA or other memory-based architectures, the random interconnect in general does not.

We randomly initialize the states of the nodes (initial condition of the RBN). The nodes are updated synchronously using their corresponding Boolean functions. Other updating schemes exist, see for example [8] for an overview. Synchronous random Boolean networks as introduced by Kauffman are commonly called *NK* networks or models.

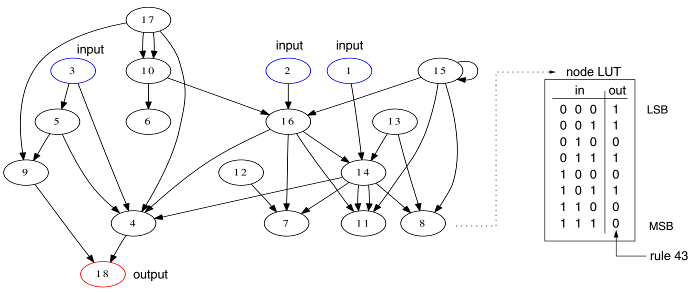

Fig. 1. Illustration of an 18-node RBN with 3 input nodes (node IDs 1, 2, and 3, colored in blue) and 1 output node (node ID 18, colored in red). The average connectivity is $K = 2.5$. The node rules are commonly represented by *lookup-tables* (LUTs), which associate a 1-bit output (the node's future state) to each possible K-bit input configuration. The table's out-column is commonly called the *rule* of the node.

The "classical" RBN is a closed system without explicit inputs and outputs. In order to solve tasks that involve inputs and outputs, we modify the classical model and add I input nodes and designate O nodes as output nodes. The input nodes have no logical function and simply serve to distribute the input signals to any number of nodes in the network. On the other hand, the output nodes are just like any other network node, i.e., with a Boolean transfer function, except that their state can be read from outside the network. Figure 1 shows an 18-node RBN with 3 input nodes and 1 output node.

3 Experimental Setup

We use *genetic algorithms* (GAs) to train the RBNs to solve the even-odd, the mapping task, and the bitwise AND task. The *even-odd task* consists of determining if an $l-$bit input has an even or an odd number of 1s in the input. If the number of 1s is an odd number, the output of the network must be 1, 0 otherwise. This task is admittedly rather trivial if one allows for counting the number of 1s. Also, if enough links are assigned to a single RBN node, the task can be solved with a single node since all the combinations can be enumerated in the look-up table. However, we are not interested to find such trivial solutions, instead, we look for networks that are able to generalize well if only a subset of the input patterns is presented during the training phase. In Section 5 we also use the *bitwise AND task*, which does exactly what its name suggests, i.e., form the logical AND operation bit by bit with two $l-$bit inputs and one $l-$bit output. The *mapping task* is used in Section 6 and consists of a $l-$bit input and an $l-$bit output. The output must have the same number of $l-$bits as the input, but not necessarily in the same order. Throughout the rest of the paper, we use I to refer to the total number of input bits to the network. For example, the bitwise AND for two 3-bit inputs is a problem with $I = 6$ inputs.

To apply GAs, we encode the network into a bit-stream that consists of both the network's adjacency matrix and the Boolean transfer functions for each node. The genetic operators consist of the standard mutation and one-point crossover operators that are applied to the genotypes in the network population. We further define a fitness function f and a generalization function g. For an input space M' of size m' and an input sample M of size m we write: $E_M = \frac{1}{m}\sum_{j \in M} d(j)$ with $f = 1 - E_M$, where $d(j)$ is the Hamming distance between the network output for the j^{th} input in the random sample from the input space and the expected network output for that input. Similarly, we write: $E_{M'} = \frac{1}{n}\sum_{j \in M'} d(j)$ with $g = 1 - E_{M'}$, where $d(i)$ is the Hamming distance between the network output for the i^{th} input from the entire input space and the expected network output for that input.

The simple genetic algorithm we use is as following:

1. Create a random initial population of S networks.
2. Evaluate the performance of the networks on a random sample of the input space.
3. Apply the genetic operators to obtain a new population.
4. Continue with steps 2 and 3 until at least one of the networks achieves a perfect fitness or after G_{max} generations are reached.

To optimize feedforward networks (see Section 5), we have to make sure that the mutation and crossover operators do not violate the feedforward topology of the network. We add an order attribute to each node on the network and the nodes accept connections only from lower order nodes.

Since RBNs have recurrent connections, their rich dynamics need to be taken into account when solving tasks, and in particular interpreting output signals.

Their finite and deterministic behavior guarantees that a network will fall into a (periodic or fixed point) attractor after a finite number of steps. The transient length depends on the network's average connectivity K and the network size N [11]. For our simulations, we run the networks long enough until they reach an attractor. Based on [11], we run our networks (with $k < 5$) for $2N$ time steps to reach an attractor. However, due to potentially ambiguous outputs on periodic attractors, we further calculate the average activation of the output nodes over a number of time steps equal to the size N of the network and consider the activity level as 1 if at least half of the time the output is 1, otherwise the activity will be 0. A similar technique was used successfully in [21].

4 Training and Network Performance Definitions

Patarnello and Carnevali [16] introduced the notion of *learning probability* as a way of describing the learning and generalization capability of their feedforward networks. They defined the learning probability as the probability of the training process yielding a network with perfect generalization, given that the training achieves perfect fitness on a sample of the input space.

The learning probability is expressed as a function of the fraction of the input space, $s = \frac{m}{m'}$, used during the training. To calculate this measure in a robust way, we run the training process r times and store both the fitness f and the generalization g values. We define the learning probability as a function of s, $\delta(s) = Pr(g = 1|f = 1) = \frac{\alpha'(s)}{\alpha(s)}$, where $\alpha(s) = Pr(f = 1)$ is the probability of achieving a perfect fitness after training, i.e., $f = 1$, and where $\alpha'(s) = Pr(g = 1)$ is the probability of obtaining a perfect fitness in generalization, $g = 1$. In the following sections, we will define new measures to evaluate the network performance more effectively.

One can say that the probabilistic measures, such as the learning probability described above, only focus on the perfect cases and hence describe the performance of the training process rather than the effect of the training on the network performance. Thus, we define the *mean training score* as $\beta(s) = \frac{1}{r}\sum_r f_{final}$ and the *mean generalization score* as $\beta'(s) = \frac{1}{r}\sum_r g_{final}$, where f_{final} and g_{final} are the training fitness and the generalization fitness of the best networks respectively at the end of training.

To compare the overall network performance for different training sample sizes, we introduce a *cumulative measure* for all four measures as defined above. The cumulative measure is obtained by a simple trapezoidal integration [28] to calculate the area under the curve for the learning probability, the perfect training likelihood, the mean generalization score, and mean training score.

5 Learning in Feedforward Boolean Networks

The goal of this first experiment was to simply replicate Patarnello and Carnevali's [17] results with feedforward Boolean networks. Figure 2 shows the learning probability of such networks on the even-odd (RIGHT) and the bitwise AND task (LEFT) for $K = 2$ networks. We observe that as the size I of the input space increases, the training process requires a smaller number of training examples to achieve a perfect learning probability. For $I = 3$, some of the networks can solve a significant number of patterns without training because the task is too easy. We have initially determined the GA parameters (see figure legends), such as the mutation rate and the maximum number of generations experimentally, depending on how quickly we achieved perfect fitness on average. We have found the GA to be very robust against parameter variations for our tasks. These result shown in Figure 2 directly confirm Patarnello and Carnevali's [17] experiments.

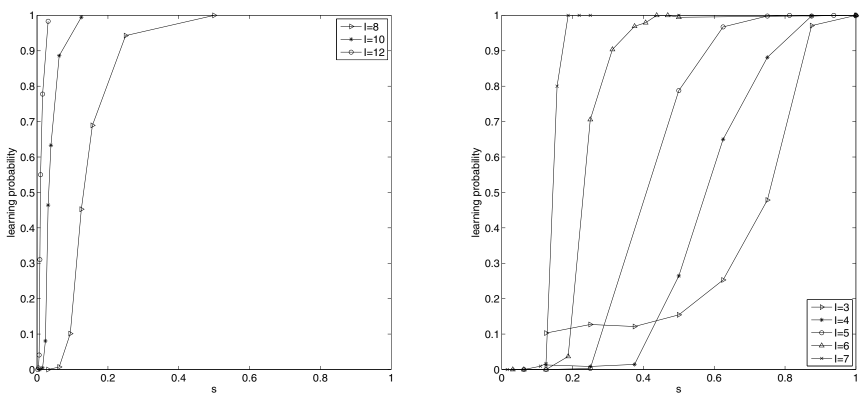

Fig. 2. LEFT: The learning probability of feedforward networks on the bitwise AND task for different input sizes I. $s = \frac{m}{m'}$ is the fraction of the input space used in training. As I increases, the learning process requires a smaller fraction of the input space during the training to achieve a perfect learning probability. RIGHT: The learning probability of feedforward networks on the even-odd task for various input sizes I. As I increases, the learning process requires a smaller fraction of the input space during the training to achieve a perfect learning probability. For $I = 3$, some of the networks can correctly classify a significant number of patterns without training because the task is too easy. For both plots: $N = 50$, $K = 2$, $G_{max} = 3000$, initial population size $= 50$, crossover rate $= 0.6$, mutation rate $= 0.3$. The GA was repeated over 700 runs.

6 Learning in RBNs

Next, we trained recurrent RBNs for the even-odd and the mapping tasks. Figure 3 (LEFT) shows the learning probability of the networks on the even-odd task with different input sizes I. While the problem size increases exponentially with I, we observe that despite this state-space explosion, a higher number of inputs I requires a smaller fraction of the input space for training the networks to achieve a high learning probability. Figure 3 (RIGHT) shows the same behavior for the mapping task, however, since the task is more difficult, we observe a worse generalization behavior. Also, compared to Figure 2, we observe in both cases that the generalization for recurrent networks is not as good as for feedforward Boolean networks. In fact, for the studied input sizes, none of the networks reaches a learning probability of 1 without training it on all the patterns.

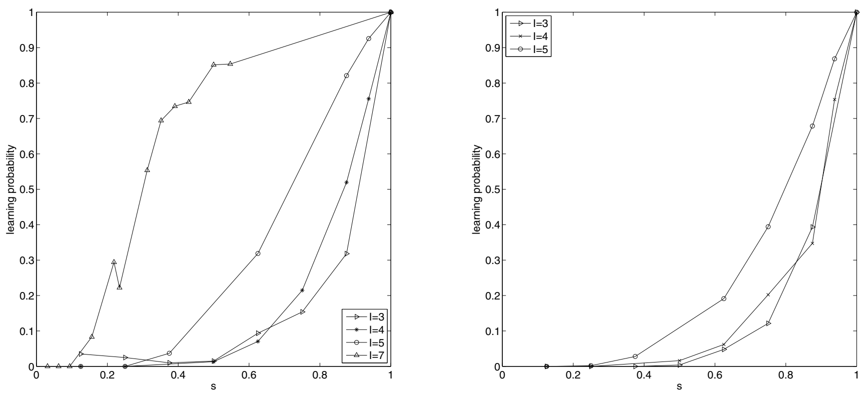

Fig. 3. LEFT: The learning probability of RBNs on the even-odd task for different problem sizes: $I = 3, 4, 5, 7$. With increasing I, the training process requires a smaller fraction of input space in order to reach a higher learning probability. $N = 20$, $G_{max} = 500$, init. population = 50, crossover rate = 0.7, mutation rate = 0.0. We calculate the data over 400 runs for all Is. RIGHT: The learning probability of RBNs on the mapping task for $I = 3, 4, 5$. We observe the same behavior, but the networks generalize even worse because the task is more difficult. $N = 40$, same GA parameters.

To investigate the effect of the average connectivity K on the learning probability, we repeat the even-odd task for networks with $K \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$. The network size was held constant at $N = 20$. In order to describe the training performance, we defined the *perfect training likelihood* measure $\alpha(s)$ as the probability for the algorithm to be able to train the network with the given fraction (s) of the input space (see section 4 for definition).

Considering the perfect training likelihood, the results in Figure 4 (RIGHT) show that for networks with subcritical connectivity $K < 2$, the patterns are harder to learn than with supercritical connectivity $K > 2$. Close to the "edge of chaos", i.e., for $K = 2$ and $K = 2.5$, we see an interesting behavior: for sample sizes above 40% of the patterns, the perfect training likelihood increases again. This transition may be related to the changes in information capacity of the network at $K = 2$ and needs further investigation with different tasks.

The significant difference between the learning probability and the perfect training likelihood for $s < 0.5$ in Figure 4 is due to the small sample size. It is thus very easy for the network to solve the task correctly, but over all r runs of the experiment, there is no network that can generalize successfully despite achieving a perfect training score. Also, according to the definitions in Section 4, it is not surprising that for a fraction $s = 1$ of the input space, i.e., all patterns are presented, the learning probability and the perfect training likelihood are different. Out of r runs, the GA did not find perfect networks for the task for all example, but if the networks solve the training inputs perfectly, they will also generalize perfectly because in this case, the training sample input includes all possible patterns.

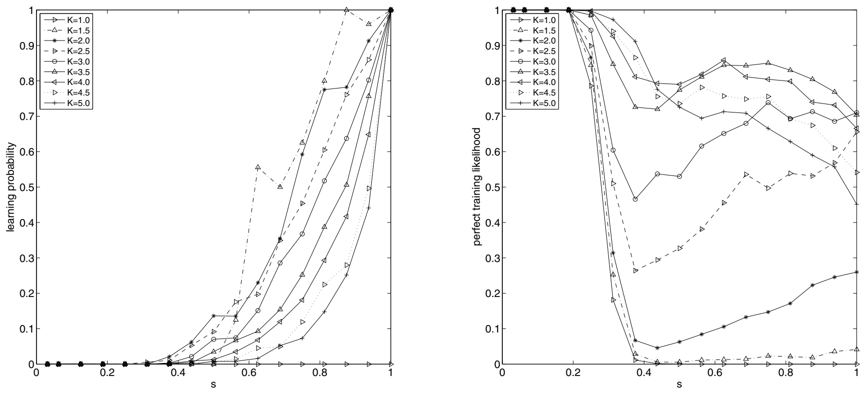

Fig. 4. LEFT: The learning probability of networks with size $N = 15$ and $K \in \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.5, 5.0\}$ for the even-odd task of size $I = 5$. Networks with connectivity $K = 1.5, 2, 2.5$ have a higher learning probability. RIGHT: The perfect training likelihood for the same networks and the same task. As the training sample size increases, subcritical connectivity networks are not able to correctly classify all training patterns, while for $K \geq 2$, correctly classifying them is easier.

7 Mean Generalization and Training Score

Figure 5 shows the learning probability (LEFT) and the perfect training likelihood (RIGHT) measured as Patarnello and Carnevali did, i.e., they only counted the number of networks with perfect generalization scores (see Section 4). Thus, if a network generalizes only 90% of the patterns, it is not counted in their score. That means that the probabilistic measures of performance that we used so far have the drawback of describing the fitness landscape of the space of possible networks rather than the performance of a particular network, which we are more interested in. To address this issue, we introduce a new way of measuring both the learning and the generalization capability. We define both of these measures as the average of the generalization and learning fitness over r runs (see section 4).

Figure 6 shows the generalization (LEFT) and the training score (RIGHT) with this new measure. As opposed to Carnevali and Patarnello's work, where higher K led to a lower learning probability, our results with the new measures for higher K lead to a higher performance with a better generalization and training score. Our measures therefore better represent the performance of the networks with regards to a given task because they also include networks that can partially solve the task.

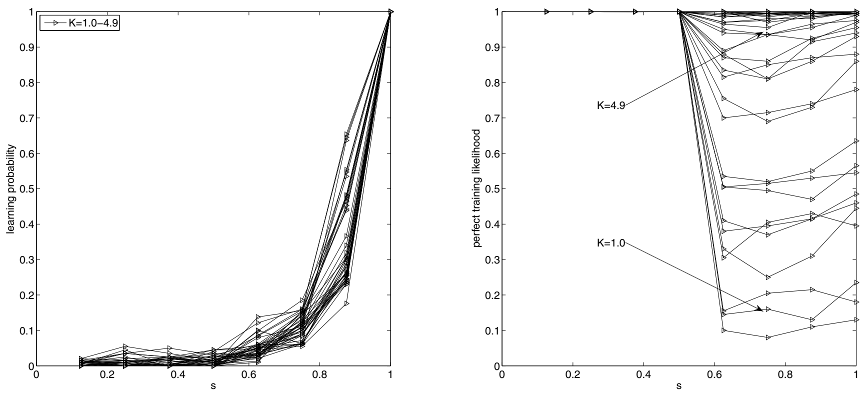

Fig. 5. Learning probability (LEFT) and perfect training likelihood (RIGHT). $I = 3, N = 15$, even-odd task. Compared to the learning probability for $I = 5$, there is not much difference between the learning probability of networks with various K for $I = 3$ because of the small input space. However, the perfect training likelihood still increases with K. K ranges from 1.0 to 4.9 with 0.1 increments.

8 Cumulative Measures

In all the previous generalization figures, the question arises which networks are "better" than others, in particular if they do not reach a maximal generalization score when less than 100% of the patterns are presented. This behavior can be observed in Figure 4 (LEFT) for the even-odd task.

Figure 7 shows the *cumulative learning probability* (LEFT) and the *cumulative training likelihood* (RIGHT) determined by integrating numerically (see Section 4 for definitions) the area under the curves of Figure 5. Figure 7 (LEFT) shows that K has no effect on the generalization and that the generalization capability is very low. Figure 7 (RIGHT) shows that higher K increases the chance of perfect training, i.e., the network can be trained to memorize all training patterns. Each cluster of connectivities in Figure 5 (RIGHT) corresponds to a "step" in the curves of Figure 7 (RIGHT).

Figure 8 shows the *cumulative generalization score* (LEFT) and the *cumulative training score* (RIGHT) based on the new measure as introduced in Section 7. We have used the even-odd task for two input sizes, $I = 3$ and $I = 5$. We observe that K has now a significant effect on the generalization score. The higher K, the better the generalization. Moreover, different intervals of K result

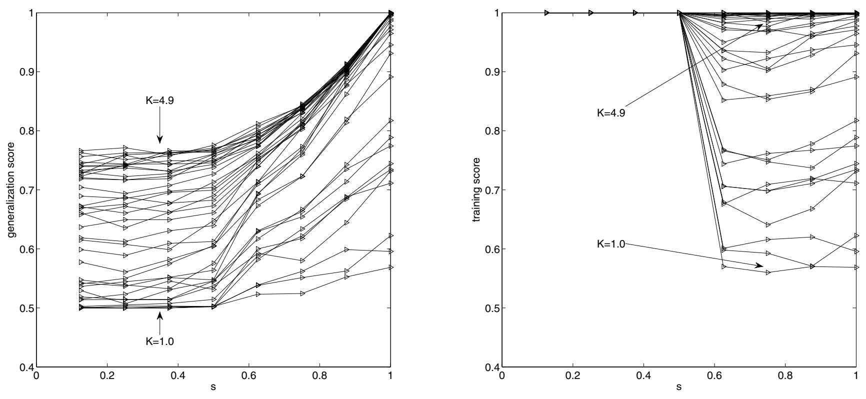

Fig. 6. The new generalization (LEFT) and training score (RIGHT), which better reflects the performance of the networks with regards to a given task. $I = 3, N = 15$, even-odd task. K ranges from 1.0 to 4.9 with 0.1 increments.

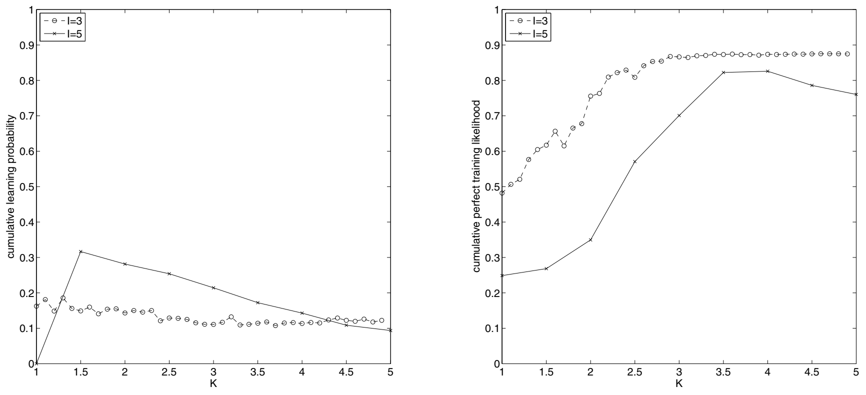

Fig. 7. LEFT: Cumulative learning probability. RIGHT: Cumulative training likelihood. Each data point in this figure corresponds to the area under the curves shown in Figure 5. $N = 15$ in both figures, even-odd task. As one can see, perfect memorization is more likely with higher K, but perfect generalization is more likely for near-critical connectivity $1.5 \leq K \leq 3$. The cumulative learning probability and the perfect training likelihood represent the area under the learning probability and perfect training likelihood curves respectively (see Figure 4 and 5, and Section 4).

in a step-wise generalization score increase. Figure 8 (RIGHT) shows that the cumulative training score for higher K increases the chance of perfect training, i.e., the network can be trained to memorize all training patterns. Also, the higher the input size I, the better the generalization, which was already observed by Patarnello and Carnevali (see also Section 5).

In summary, we have seen so far that according to our new measures, higher K networks both generalize and memorize better, but they achieve perfect generalization less often. The picture is a bit more complicated, however. Our data also shows that for networks around $K = 1.5$, there are more networks in the space of all possible networks that can generalize perfectly. For $K > 1.5$, the networks have a higher generalization score on average, but there is a lower number of networks with perfect generalization. That is because the fraction of networks with perfect generalization is too small with respect to the space of all the networks. For $K < 1.5$, the networks are hard to train, but if we manage to do so, they also generalize well.

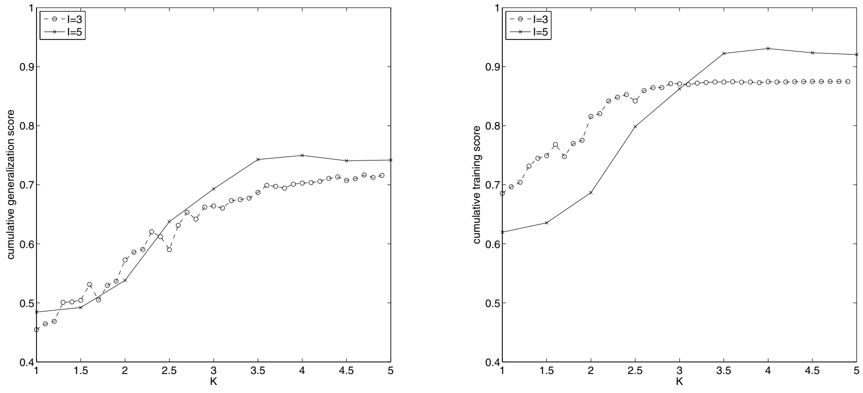

Fig. 8. LEFT: Cumulative generalization score. RIGHT: Cumulative training score. $N = 15$ in both figures, even-odd task, $I = 3$ and $I = 5$. As one can see, both the network's generalization and the memorization capacity increase with K. The cumulative generalization and training score represent the area under the mean generalization and training score curves respectively (see Figure 6 and Section 4).

Figure 9 shows the complete cumulative learning probability (LEFT) and cumulative training likelihood (RIGHT) landscapes as a function of K and N. We observe that according to these measures, neither the system size nor the connectivity affects the learning probability. Also, the networks have a very low learning probability, as seen in Figure 7. That means that the performance of the training method does not depend on the system size and the connectivity and confirms our hypothesis that Carnevali and Patarnello's measure is more about the method than the network's performance.

Finally, Figure 10 shows the same data as presented in Figure 9 but with our own score measures. For both the cumulative generalization score and the cumulative training score, the network size N has no effect on the generalization and the training, at least for this task. However, we see that for the cumulative generalization score, the higher K, the higher the generalization score. The same applies to the cumulative training score. This contrasts what we have seen in Figure 9.

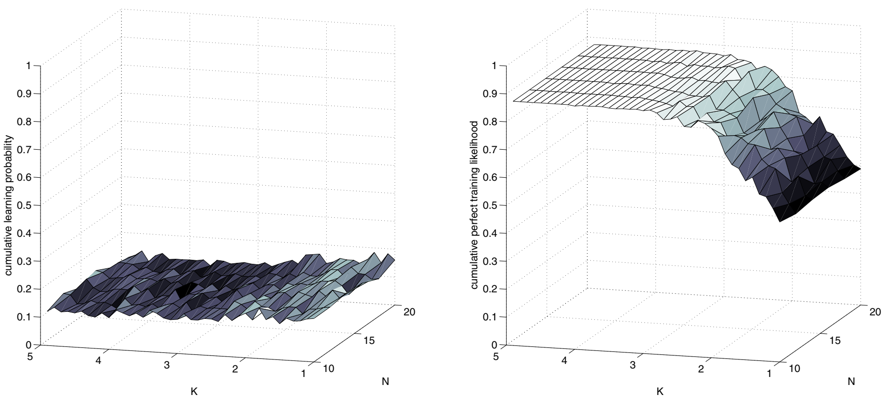

Fig. 9. LEFT: Cumulative learning probability. RIGHT: Cumulative training likelihood. Even-odd task.

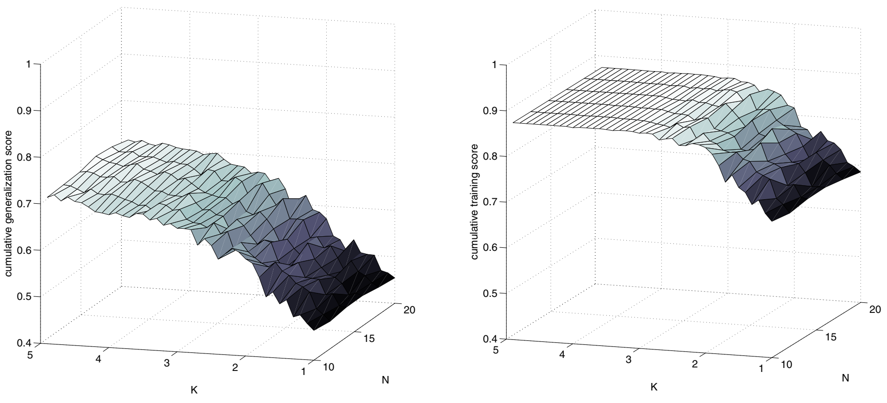

Fig. 10. LEFT: Cumulative generalization score. RIGHT: Cumulative training score. Even-odd task.

9 Discussion

We have seen that Patarnello and Carnevali's measure quantifies the fitness landscape of the networks rather than the network's performance. Our newly defined measures applied to RBNs have shown that higher K networks both generalize and memorize better. However, our results suggest that for large input spaces and for for $K < 1.5$ and $K > 3$ networks, the space of the possible networks changes in a way that makes it difficult to find perfect networks (see Figures 4 and 5). On the other hand, for $1.5 \leq K < 3$, finding the perfect networks is significantly easier. This is a direct result of the change in the number of possible networks and the number of networks that realize a particular task as a function of K.

In [13], Lizier *et al.* investigated information theoretical aspects of phase transitions in RBNs and concluded that subcritical networks ($K < 2$) are more suitable for computational tasks that require more of an information storage, while supercritical networks ($K > 2$) are more suitable for computations that require more of an information transfer. The networks at critical connectivity ($K = 2$) showed a balance between information transfer and information storage. This finding is purely information theoretic and does neither consider input and outputs nor actual computational tasks. In our case, solving the tasks depends on the stable network states and their interpretations. The results in Lizier *et al.* do not apply directly to the performance of our networks, but we believe there is a way to link the findings in future work. Compared to Lizier *et al.*, our experiments show that supercritical networks do a better job at both memorizing and generalizing. However, from the point of view of the learning probability, we also observe that for networks with $1.5 \leq K < 3$, we are more likely to find perfect networks for our specific computational tasks.

10 Conclusion

In this paper we empirically showed that random Boolean networks can be evolved to solve simple computational tasks. We have investigated the learning and generalization capabilities of such networks as a function of the system size N, the average connectivity K, problem size I, and the task. As feedforward Boolean networks, RBNs can learn simple tasks, however, their generalization capabilities are not as extraordinary as for feedforward Boolean networks. We have seen that the learning probability measure used by Patarnello and Carnevali [16] was of limited use and have thus introduced new measures, which better describe what the networks are doing during the training and generalization phase. The results presented in this paper are invariant of the training parameters and are intrinsic to both the learning capability of dynamical automata networks and the complexity of the computational task. Future work will focus on the understanding of the Boolean function space, in particular on the function bias.

References

1. Aleksander, I.: Random logic nets: Stability and adaptation. International Journal of Man-Machine Studies 5, 115–131 (1973)
2. Aleksander, I.: From Wisard to Magnus: A family of weightless virtual neural machines. In: Austin, J. (ed.) RAM-Based Neural Networks. Progress in Neural Processing, vol. 9. World Scientific (1998)
3. Aleksander, I., Thomas, W.V., Bowden, P.A.: WISARD: A radical step foward in image recognition. Sensor Review 4, 120–124 (1984)
4. Amari, S.I.: Characteristics of randomly connected threshold-element networks and network systems. Proceedings of the IEEE 59(1), 35–47 (1971)

5. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)
6. Carnevali, P., Patarnello, S.: Exhaustive thermodynamical analysis of Boolean learning networks. Europhysics Letters 4(10), 1199–1204 (1987)
7. Van den Broeck, C., Kawai, R.: Learning in feedforward Boolean networks. Physical Review A 42(10), 6210–6218 (1990)
8. Gershenson, C.: Classification of random Boolean networks. In: Standish, R.K., Bedau, M.A., Abbass, H.A. (eds.) Artificial Life VIII. Proceedings of the Eight International Conference on Artificial Life, pp. 1–8. MIT Press, Cambridge, MA (2003)
9. Kauffman, S.A.: Metabolic stability and epigenesis in randomly connected genetic nets. Journal of Theoretical Biology 22, 437–467 (1968)
10. Kauffman, S.A.: Emergent properties in random complex automata. Physica D 10(1-2), 145–156 (1984)
11. Kauffman, S.A.: The Origins of Order: Self–Organization and Selection in Evolution. Oxford University Press, New York (1993)
12. Lawson, J., Wolpert, D.H.: Adaptive programming of unconventional nano-architectures. Journal of Computational and Theoretical Nanoscience 3, 272–279 (2006)
13. Lizier, J., Prokopenko, M., Zomaya, A.: The information dynamics of phase transitions in random boolean networks. In: Bullock, S., Noble, J., Watson, R., Bedau, M.A. (eds.) Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, pp. 374–381. MIT Press, Cambridge (2008)
14. Martland, D.: Auto-associative pattern storage using synchronous boolean networks. In: Proceedings of the First IEEE International Conference on Neural Networks, San Diego, CA, vol. III, pp. 355–366 (1987)
15. Martland, D.: Behaviour of autonomous (synchronous) boolean networks. In: Proceedings of the First IEEE International Conference on Neural Networks, San Diego, CA, vol. II, pp. 243–250 (1987)
16. Patarnello, A., Carnevali, P.: Learning networks of neurons with Boolean logic. Europhysics Letters 4(4), 503–508 (1987)
17. Patarnello, S., Carnevali, P.: Learning capabilities of boolean networks. In: Aleksander, I. (ed.) Neural Computing Architectures: The Design of Brain-Like Machines, ch. 7, pp. 117–129. North Oxford Academic, London (1989)
18. Rozonoér, L.I.: Random logical nets I. Automation and Remote Control 5, 773–781 (1969); translation of Avtomatika i Telemekhanika
19. Selfridge, O.G.: "Pandemonium": A paradigm for learning. In: Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, pp. 513–526 (1958)
20. Selfridge, O.G., Neisser, U.: Pattern recognition by machine. Scientific American 203(2), 60–68 (1960)
21. Teuscher, C.: Turing's Connectionism. An Investigation of Neural Network Architectures. Springer, London (2002)
22. Teuscher, C., Gulbahce, N., Rohlf, T.: Learning and generalization in random Boolean networks. In: Dynamics Days 2007: International Conference on Chaos and Nonlinear Dynamics, Boston, MA, January 3-6 (2007)
23. Teuscher, C., Gulbahce, N., Rohlf, T.: An assessment of random dynamical network automata for nanoelectronics. International Journal of Nanotechnology and Molecular Computation 1(4), 39–57 (2009)

24. Tour, J., Van Zandt, W.L., Husband, C.P., Husband, S.M., Wilson, L.S., Franzon, P.D., Nackashi, D.P.: Nanocell logic gates for molecular computing. IEEE Transactions on Nanotechnology 1(2), 100–109 (2002)
25. Turing, A.M.: Intelligent machinery. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 5, pp. 3–23. Edinburgh University Press, Edinburgh (1969)
26. Weisbuch, G.: Dynamique des systèmes complexes: Une introduction aux réseaux d'automates. InterEditions, France (1989)
27. Weisbuch, G.: Complex Systems Dynamics: An Introduction to Automata Networks. Lecture Notes, Santa Fe Institute, Studies in the Sciences of Complexity, vol. 2. Addison-Wesley, Redwood City (1991)
28. Wittaker, E.T., Robinson, G.: The trapezoidal and parabolic rules. In: The Calculus of Observations: A Treatise on Numerical Mathematics, pp. 156–158. Dover, New York (1969)