Self-building Neural Networks

Andrea Ferigo
Giovanni Iacca
University of Trento
Trento, Italy

ABSTRACT

During the first part of life, the brain develops while it learns through a process called synaptogenesis. The neurons, growing and interacting with each other, create synapses. However, eventually the brain prunes those synapses. While previous work focused on learning and pruning independently, in this work we propose a biologically plausible model that, thanks to a combination of Hebbian Learning and pruning, aims to simulate the synaptogenesis process. In this way, while learning how to solve the task, the agent translates its experience into a particular network structure. Namely, the network structure *builds itself* during the execution of the task. We call this approach *Self-building Neural Network* (SBNN). We compare our proposed SBNN with traditional feed forward neural networks (FFNNs) over three OpenAI classic control tasks. The results show that our model performs generally better than FFNNs. Moreover, we observe that the performance decay while increasing the pruning rate is smaller in our model than with FFNNs.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Genetic algorithms*; *Artificial life*; • **Theory of computation** → **Models of learning**.

KEYWORDS

Neural networks, plasticity, pruning, neuroevolution

ACM Reference Format:
Andrea Ferigo and Giovanni Iacca. 2023. Self-building Neural Networks. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3583133.3590531

1 INTRODUCTION

The natural brain is one of the most complex systems we know of. It can perform complex tasks and adapt to new situations with an efficiency that is currently unreachable by any modern Artificial Intelligence (AI) system. These performances derive from a long-lasting evolutionary process that has harmonized a vast amount of different elements that work at different scales, and arrange their structure through a combination of synaptogenesis [6] and pruning of the less relevant synapses [5].

GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0120-7/23/07.
https://doi.org/10.1145/3583133.3590531

In this work, we are interested in linking the growth and organization of the neurons in the brain to the experience of the agent during the task, as it happens in the natural brain. To simulate this process, we combine two well-known mechanisms from the literature, namely Hebbian Learning (HL) [2], a task-agnostic plasticity model that takes inspiration from the natural neurons, and a pruning mechanism based on the global magnitude algorithm [3]. The latter is modified in such a way to decide not only *how much* (i.e., how many synapses) to prune, but also *when* prune. We call the resulting model *Self-building Neural Network* (SBNN), due to its capability to compose its own structure based on the experience perceived by the agent during its life.

We test our proposed SBNN on three OpenAI classic control tasks, to show the capability of this model in terms of performance and how the structure of the networks actually depend on the task.

The rest of this paper is organized as follows. Section 2 describes the methods, and in particular the proposed SBNN. Then, Section 3 shows the results, followed by the conclusions in Section 4.

2 METHODS

Hebbian learning. HL is a plasticity model that allows an NN to change its weights during the execution of a task. Importantly, this change is agnostic w.r.t. the reward for the task, because it is based only on the local knowledge of each synapse, in particular the activation of the pre-synaptic and post-synaptic neurons. The ABCD model used in this work updates the weights after each forward pass of the network using the following rule:

$$w_{i,j} = w_{i,j} + \eta(Aa_i + Ba_j + Ca_i a_j + D)$$

where a_i is the pre-synaptic activation value, a_j is the post-synaptic value, and $w_{i,j}$ is the weight on the connection between the two neurons. The A, B, C, and D are parameters to optimize.

Pruning mechanism. The pruning mechanism aims to find a subset of an NN that performs as well as (or better than) the original network. This process is performed by removing connections based on a given strategy. In this work, we use the global magnitude pruning algorithm [3], that simply consists in removing all the connections whose weights are smaller, in absolute value, than a threshold that is defined as the pr-th percentile, where pr is the desired pruning rate (i.e., the percentage of connections to remove).

Self-building Neural Network. We start from an NN composed of I inputs, H hidden nodes, and O outputs. At the first episode of the task, the I inputs are connected to all the H hidden nodes and the O outputs. In turn, the H hidden nodes are fully connected with each other (excluding self-loops), and with all the O outputs.

We initialize the weights of all these connections to zero. In this way, we intend to simulate the initial condition where no connections between neurons exist. Then, *within* each episode, the

Hebbian procedure will update the weights based on the ABCD rule. Note that, in our model, each connection in the network has its own ABCD rule with its corresponding parameters. In this way, the network can arrange itself based on the experience that the agent accumulates during the task. We use a Hebbian rule for each connection because, starting from a condition where all the weights are 0, using a single rule could lead all the weights to change in the same direction, which in turn would make learning ineffective.

The second step of synaptogenesis is the process that prunes the synapses, as described before. As we will describe later, differently from HL, pruning occurs *across* episodes. Note that, as soon as pruning starts, HL is stopped. Figure 1 summarizes the pruning procedure, showing the state of the initial network and its development. Formally, we can analyze the network before and after pruning. In particular, before pruning, the hidden nodes are fully connected with each other and all the inputs are connected with all the hidden nodes. For this reason, it is not possible to define a fixed activation order. Hence, we maintain the overall activation order: firstly, the inputs, then the hidden nodes, and then the outputs. For the hidden nodes, we randomly select the activation order.

After pruning, the remaining connections define the network. Differently from before, in this phase we can define an activation order more easily because the pruning mechanism naturally resolves most cycles, especially if the ratio of connections removed is high enough. Hence, to find the activation order, we can perform a topological sort of the underlined graph $G(V, E)$, where V is the set of nodes (i.e., the neurons) and E is the set of connections. If, during the topological sort, we find a cycle, we apply the following procedure. Indicating with N_c the subset of V that contains all the nodes in the cycle, first we remove all the nodes in N_c from V and replace them with a *fake* node, f. Then, indicating with $E(N_c)_{incoming} = \{(v, n)\ \forall v \in V \setminus N_c\ \wedge\ \forall n \in N_c\}$ the subset of E composed of the connections that terminate in N_c and that do not start from N_c, we add to E the set of connections $\{(v, f)\ \forall (v, v') \in E(N_c)_{incoming} \wedge v' \in N_c\}$. We perform the same operation for the connections outgoing from N_c. This procedure, that we apply iteratively and independently for every cycle found in the graph, results in a new graph G' where the cycle N_c is replaced by the fake node f. All the nodes connected to N_c are now connected to f, and f is connected to all the nodes reached from N_c. We store the information that the node f replaces the N_c nodes in a *cycles_history* variable and then retry to find a topological order. We repeat this procedure until all the cycles have been replaced, and a topological order can be defined. Note that the nodes in N_c can also be fake nodes from a previous iteration of the procedure.

After calculating the topological order of the network, we can follow that for the activation of the hidden nodes. If we find a fake node during the activation, we retrieve from *cycles_history* the set of N_c nodes that compose the cycle, and proceed with a random activation order. If a node in N_c is a fake node f' covering the $N_{c'}$ cycle, we repeat the procedure solving the inner cycle $N_{c'}$, before continuing with the nodes in N_c.

Thanks to this process, the network after pruning can have one of three base structures. (1): the inputs are connected to all the hidden nodes that in turn are connected to the outputs. This creates an NN with a single hidden layer. (2): the pruning process cuts all the connections between the input and hidden nodes. Hence, the inputs are directly connected to the output nodes, creating a zero-layer NN. (3): the pruning process removes all the connections between the inputs and a subset of the hidden nodes, but the hidden nodes remain connected, creating an NN with more than one layer.

It is worth noticing that, starting from these three base structures, we can derive more complex structures. For example, each node in the third case can be a *fake* node, hence "hiding" a set of nodes.

While in principle promising, this model has drawbacks. Firstly, the number of weights in the SBNN increases quadratically with the number of hidden nodes, as each hidden node is fully connected with all the other hidden nodes. Moreover, each connection is associated with an ABCD rule with its corresponding 4 parameters. Secondly, before pruning the hidden nodes compose a single, fully connected subnetwork on which the activation order can influence the network's output, but cannot be uniquely determined.

OpenAI tasks. To measure the performance of the proposed SBNN, we use three OpenAI classic control tasks [1], namely *Cart Pole* (CP), *Mountain Car* (MC), and *Lunar Lander* (LL). We refer the interested reader to [1] for a complete description of these tasks.

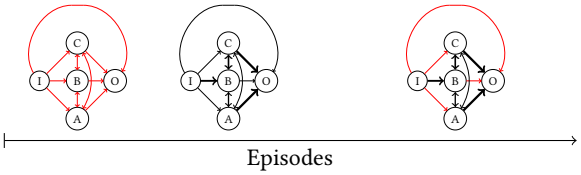

Figure 1: Scheme of the SBNN over the episodes. Initially, we set all the connections to 0 (red); then during the task Hebbian plasticity changes the weights, leading to the second NN, where different thickness indicates different weights. At a certain time, the pruning mechanism cuts the weakest connections, resulting in the final structure of the NN.

3 RESULTS

In this section, we analyze the performance and behavior of the SBNN in comparison with a feed forward neural network (FFNN) for which we apply the same pruning mechanism used in the SBNN, but before fitness evaluation. Note that, in our implementation, we use as activation function the *tanh* function, both on the hidden nodes and on the input/output nodes, for both the SBNN and the FFNN. Concerning the fitness evaluation, we measure the performance of the agent as the average reward over 100 episodes seen during training. In the case of the FFNN, as the pruning process happens before the first episode, the fitness by construction is measured after pruning. On the contrary, as for the SBNN the pruning process happens during the life of the agent (i.e., across episodes), in this case the fitness contains two components, one before and one after pruning (which are then averaged). We divide our experiments into two parts, to answer two different research questions:

RQ1 What is the performance of the SBNN? What are the main hyperparameters of this model that affect the performance?

RQ2 Is there any structural difference between the networks produced by the SBNN and an FFNN?

To answer these questions, we perform a campaign of simulations varying the three main hyperparameters of the SBNN: the number of hidden nodes *hn*, the pruning rate *pr*, and the pruning time *pt*, the latter indicating when pruning is applied. To calculate *pt*, we consider the number of episodes in the task, i.e., $pt = 10$ means that pruning happens after the 10-th episode.

For each combination of these parameters, we perform 30 independent runs. To optimize the parameters of the network (i.e., the weights for the FFNN, or the parameters of the ABCD rules for the SBNN), we use the Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [4]. We stop the evolution after the generation of a fixed number of 2000 individuals for LL and CP, and 4000 for MC. In all cases, we set $\lambda = 4 + \lfloor 3 * ln(|\mathbf{p}|) \rfloor$ and $\mu = \frac{\lambda}{2}$, where $\mathbf{p}$ is the vector of parameters to optimize. Table 1 summarizes the tested configurations. Note that we omit the results obtained on the CP task as there were no significant differences between the FFNN and the SBNN: in fact, all the individuals using both models solved the task, that is comparatively simpler than the other two. We make our code available at https://github.com/ndr09/SBM.

Parameter	CP	MC	LL
No. fitness evaluations	2000	4000	2000
No. hidden nodes (*hn*)	3, 4	3, 4	5, 6, 7, 8, 9
Pruning time (*pt*)	5, 10	1, 5, 10	1, 5, 10, 15, 20
Pruning rate (*pr*)	40, 60	40, 60	20, 40, 60, 80

Table 1: Parameter settings used for the RQ1 experiments. For RQ2, we use a subset of these configurations.

3.1 RQ1: Performance

Figure 2 shows the results for the MC environment. The upper and lower row relate, respectively, to an FFNN with one layer with $hn = 3$ or $hn = 4$. The left and right column present, respectively, the results with a *pr* of 40% and 60%. In each subfigure, we plot the average results of the best individual over 30 independent runs. The first three boxplots indicate the results of the SBNN with different *pt*, namely 10, 5, and 1, respectively from left to right. The last boxplot shows the baseline results of the FFNN. In MC, the results indicate that the SBNN reaches similar or better performance w.r.t. an FFNN with the same *pr*.

Interestingly, we observe a clear trend on *pt*, i.e., the performance increases when decreasing *pt*, regardless of *pr* and *hn*. Hence, we can conclude that, after the first episode, the agent has already received enough experience (information) to build the network.

To understand the effect of pruning on the performance, we make an additional analysis, by comparing the average reward before and after pruning. For instance, considering $pt = 10$, we measure separately the average reward until the 10-th episode, i.e., before pruning, and the average reward after the 10-th episode, i.e., the post-pruning one. Based on this procedure, we observe that, after pruning, the performance receives a $7 - 13\%$ boost on MC.

Figure 3 presents the results for the LL environment. For this environment, we consider *hn* between 5 and 9, because of the greater complexity of the task. Here, the results are shown differently from Figure 2: in particular, we plot the median rewards for the best individuals of each evolutionary run, varying *pr* while keeping *pt*

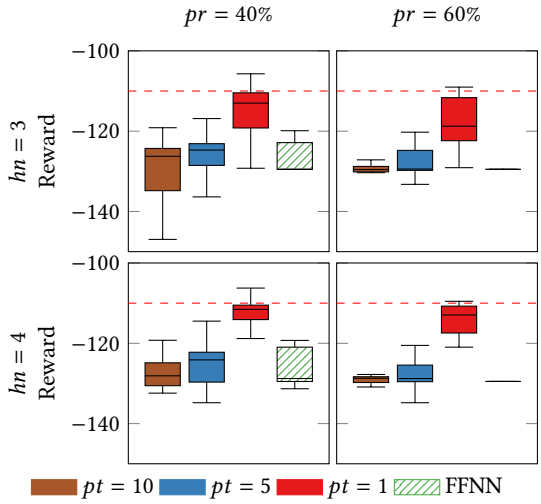

Figure 2: Results on the MC environment. The red dashed line indicates the solving threshold for the environment.

and *hn* fixed. In this way, we highlight two points. The first one is that the performances of the SBNNs are in most cases equal to or better than the ones obtained by the FFNN for the same *pr*. The second point is that, while increasing *pr*, the drop in performance for the SBNN is lower than what observed with the FFNN baseline. We can also observe that this trend is maintained when comparing solutions with a comparable number of connections. For example, the SBNN with $hn = 5$ and the FFNN with $hn = 9$ have a similar number of connections (respectively, 117 and 108). Moreover, we can see the same trend observed in MC, i.e., $pt = 1$ yields the best results. Hence, also in this task a single episode contains enough information to learn about the prunable connections. This suggests that, in the tested tasks, the SBNN is capable to exploit the network structure (and the information therein) better than the FFNN.

We perform the same analysis done before, dividing the results before and after pruning and observing the average results. Also for LL there is an increase in performance after pruning, in this case between 6% and 100%. This improvement indicates again the importance of pruning and its complementary effect w.r.t. HL.

3.2 RQ2: Difference between SBNN and FFNN

We now analyze the structural difference between the network found with the SBNN and the FFNN. For this reason, we characterize the networks based on the number of *working connections*, i.e., the connections that link inputs to outputs after pruning. To calculate these connections, we remove the synapses that lead to sink or come from source nodes. While a sink is a node with only incoming connections that is not an output node, a source is a node with only outgoing connections that is not an input node.

Figure 4 shows, for the LL task, the distribution of working connections for a representative subset of the configurations presented in the previous section, considering the best solution (one per each evolutionary run) obtained for each considered configuration. We omit, due to space limitations, the same figure for MC. On the x-axis, we indicate the percentage of remaining working connections

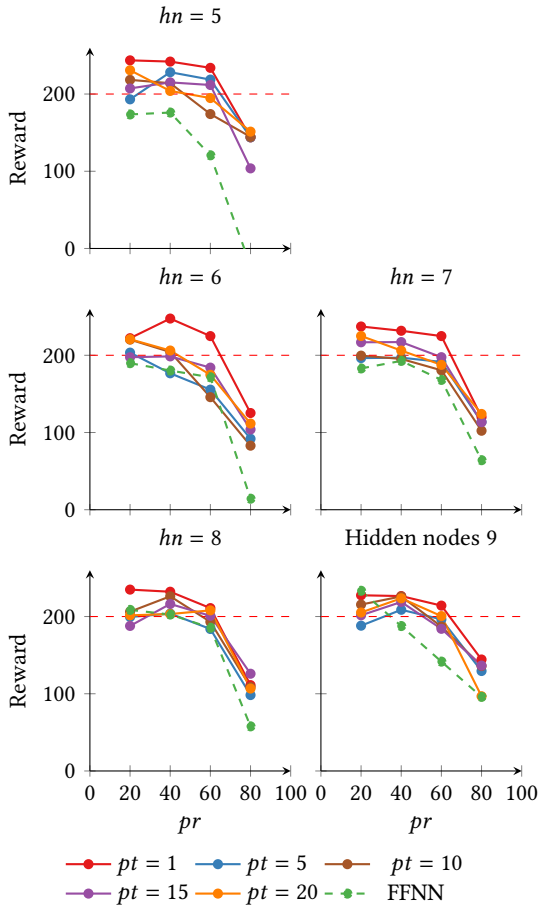

Figure 3: Median reward on the LL task. The red dashed line indicates the solving threshold.

(after pruning) w.r.t. the total number of synapses, grouped every 10%, while on the y-axis we indicate how many networks (out of 30, one per run) have that number of connections. For example, if we consider a point at $x = 40\%, y = 0.5$, we mean that in 50% of runs (i.e., 15 out of 30) networks after pruning have a number of working connections in the range $(30\%, 40\%]$.

We can observe a quite clear pattern: all the FFNN configurations use the majority of the connections available. On the other hand, the distributions of working connections for the SBNN have two peaks: the first one occurs between 10% and 20% for the MC task (not reported here) and between 20% and 40% for the LL one; the second peak is in common between the two tasks at around 90%. We visually analyzed all the networks and discovered that the SBNNs that compose the first peak have a structure where all the hidden nodes are disconnected. Interestingly, the percentage of this kind of structures increases when pt increases, as the first peak is higher for higher values of pt. This suggests that the later pruning occurs, the more probable it is that connections to the hidden nodes are pruned, thus leaving only input-output connections. Our intuition is that this form of simplification somehow correlates with the complexity needed to solve the task.

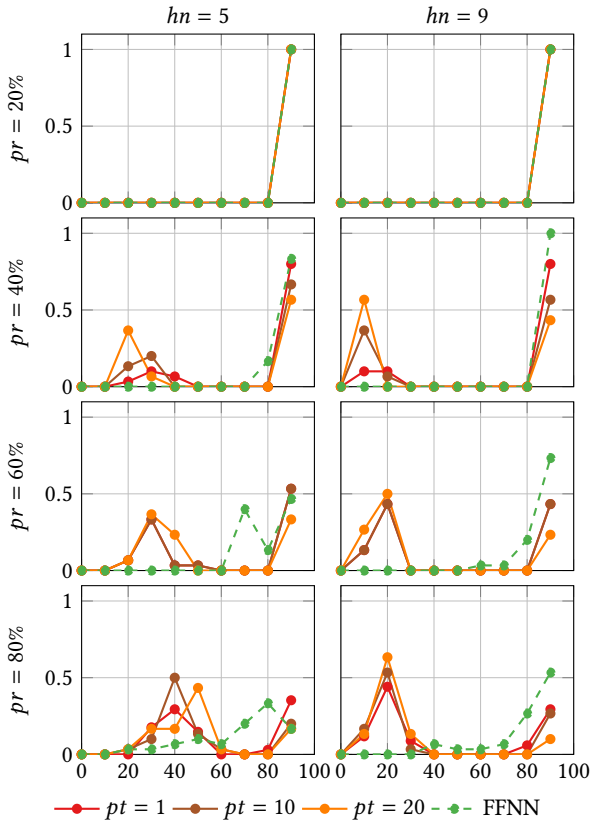

Figure 4: No. of connections after pruning on the LL task.

4 CONCLUSIONS

In this work, we took inspiration from the synaptogenesis process occurring in natural brains to propose a new learning model that combines both plasticity and pruning. We called this model Self-building Neural Network (SBNN), as it changes its structure based on the experience of the agent during the episodes of the task.

We tested our model on three classic control tasks from OpenAI, varying the three main parameters of the model, affecting respectively *when* to prune, *how much* to prune, and the number of hidden nodes. We showed that, in general, the SBNN reaches better performance than the FFNN. Furthermore, we assessed the effect of the model's parameters, in particular regarding when and how much to prune, and showed the advantages of the SBNN.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:1606.01540.
[2] Thomas H Brown, Edward W Kairiss, and Claude L Keenan. 1990. Hebbian synapses: biophysical mechanisms and algorithms. *Annual review of neuroscience* 13, 1 (1990), 475–511.
[3] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. arXiv:1506.02626.
[4] Nikolaus Hansen. 2016. The CMA Evolution Strategy: A Tutorial. arXiv:1604.00772.
[5] Gregory Z. Tau and Bradley S. Peterson. 2010. Normal Development of Brain Circuits. *Neuropsychopharmacology* 35 (2010), 147–168.
[6] Wendy Zukerman and Andrew Purcell. 2011. Brain's synaptic pruning continues into your 20s. *New Scientist* 211, 2826 (2011), 9.