Self-Adaptation of Neuroevolution Algorithms using Reinforcement Learning

Anonymized For Double Blind Review

Anonymous

Abstract. Selecting an appropriate neural architecture for a given dataset is an open problem in machine learning. Neuroevolution algorithms, such as NEAT, have shown great promise in automating this process. An extension of NEAT called EXAMM has demonstrated the capability to generate recurrent neural architectures for various time series datasets. At each iteration the evolutionary process is furthered using randomly selected mutation or crossover operations, which are chosen in accordance with pre-assigned probabilities. In this paper we present a self-adapting version of EXAMM that incorporates finite action-set learning automata (FALA), a reinforcement learning technique. FALA is used to dynamically adjust the aforementioned probabilities, thereby guiding the evolutionary process, while also significantly reducing the number of required hyperparameters. It is also demonstrated that this approach improves the performance of the generated networks with statistical significance. Furthermore, the evolution of the adapted probabilities is analyzed to gain further insight into the inner workings of EXAMM.

Keywords: Neuroevolution · Reinforcement Learning · FALA

1 Introduction

Inspired by the architecture of the cerebral cortex, neural networks are arguably the most powerful machine learning model in use today [16]. The success of neural networks can be largely attributed to their flexible architecture, which can be readily adapted to suit the data at hand. This has enabled neural networks to succeed in problem spaces that other machine learning approaches find intractable, such as image recognition [11] and natural language processing [4]. The process of designing an appropriate neural architecture for most problem sets is rather complex, and so it is commonplace for machine learning practitioners adapt existing architectures instead of developing them from scratch. Often times networks pre-trained on generic data are tweaked on the target data in a process known as transfer learning [20].

Determining an appropriate neural architecture for a given dataset is an open problem, but one technique that has had considerable success is known as neuroevolution. This approach takes inspiration from natural evolutionary processes to automate the design of artificial neural networks [3]. Neuroevolution algorithms can generate architectures that are significantly different from handcrafted neural networks and have demonstrated the potential to outperform even the best human-designed architectures [3]. One of the most famous neuroevolution algorithms is known as NeuroEvolution of Augmenting Topologies (NEAT) [17]. NEAT is a genetic approach where complex neural architectures are progressively evolved from a simple baseline construct using pre–defined genetic operations. While NEAT was initially designed for regular feed forward neural network, it has recently been extended to evolve both convolutional [2] and recurrent [13] architectures. The latter of these is known as Evolutionary eXploration of Augmented Memory Models (EXAMM), and the improvement of this algorithm is the topic of this paper.

Work by Alba and Tomassini has shown that evolutionary algorithms can be greatly sped up through the incorporation of an island speciation strategy [1], and this has been implemented in EXAMM [13]. Lyu *et al.* have extended this concept further and incorporated island extinction and repopulation [10], which has been shown to outperform both the baseline island speciation strategy as well as NEAT's original speciation strategy. Recently, work by Radaideh and Shirvan has demonstrated that rule-based reinforcement learning can be used to guide evolutionary algorithms in constrained optimization [14]. In this paper, we propose an extension of EXAMM that incorporates a reinforcement learning technique known as Finite Action-set Learning Automate (FALA) to guide the evolutionary process. To our knowledge reinforcement learning has not yet been applied to this class of algorithms.

2 EXAMM

At the core of EXAMM is a master process that maintains a population of Recurrent Neural Network (RNN) genomes. This population is spread out across several islands, with each island containing an identical number of genomes. At each iteration, EXAMM selects an island in a round–robin manner, and then randomly selects a genome from that island to evolve. It chooses a genetic operation in accordance with pre-defined operational probabilities, and evolves the selected genome using the chosen operation. The genetic operations used in EXAMM and associated probabilities are listed in Table 1 (for further details please consult [13]). The evolved genome is trained for a given number of back propagation iterations using one of the worker nodes and its fitness is evaluated based on its test set performance. EXAMM then attempts to re–insert the trained genome into the island from which it was initially picked. If the fitness of the newly trained genome is superior to that of the worst genome in the selected island, the worst genome is dropped and the newly generated genome is added to the population of that island. This process continues until the maximum number of allowed genomes is generated. Certain operations will result in the insertion of new recurrent edges into the architecture of a given genome. The depths of these recurrent edges are drawn from a uniform probability distribution, which returns values between the minimum and maximum recurrent depths as set by the programmer prior to running the algorithm.

Table 1. Genetic Operations in EXAMM and their Default Probabilities

Operation	Prob.	Operation	Prob.	Operation	Prob.
Clone	0.07	Add Edge	0.07	Split Node	0.07
Add Recurrent Edge	0.07	Enable Edge	0.07	Merge Node	0.07
Disable Edge	0.07	Add Node	0.07	Crossover	0.20
Enable Node	0.07	Disable Node	0.07	Island Crossover	0.10

2.1 Opportunities for Improvement

It is not unreasonable to assume that certain genetic operations are more likely to result in well–performing genomes, and that the preferred operations will change over time as the population evolves. As such, guiding the evolutionary process by adjusting the probabilities associated with these genetic operations has the potential to improve the quality of generated networks. Operations that generate well-performing genomes should have their probabilities increased, whereas operations that produce undesirable genomes should have their probabilities decreased.

Additionally, treating all islands equally by cycling through them in a round-robin manner may not be the optimal approach. The populations of different islands tend to be heterogeneous with some islands containing better-performing genomes than others. Selecting the better islands more frequently could speed up the learning process by targeting desirable populations. Furthermore, it is possible that not all recurrent edge depths are pertinent to a given dataset, and so dynamically adjusting the probabilities associated with the selection of different depths could also speed up the learning process.

3 Finite Action-set Learning Automata

Finite Action-set Learning Automata is a reinforcement learning algorithm which evolves a vector of discrete probabilities associated with a set of actions. This approach has shown promise in the field of robotics engineering, where it has been used for behavioral design [15] and controller parameter tuning [6] [8]. A FALA process can be described by the following quadruple:

$$\mathit{\Gamma} = (\alpha, p(t), r, \tau) \tag{1}$$

In the formulation above, the four components represent the following [9]:

1. A finite set of possible actions referred to as the action set. For the purpose of this paper it will be defined as $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. The action selected at time point t is defined as $\alpha(t)$.

2. The probabilities associated with each action at time point t, which are defined as a probability vector $p(t) = (p_t(\alpha_1), p_t(\alpha_2), \dots, p_t(\alpha_n))$. This vector is initialized at time step 0 to the starting probability values. It can be thought of as the policy by which the agent selects an action.
3. A reinforcement function r which will observe the results of the action taken at time t and generate a reinforcement signal r_t.
4. A learning function τ which will update the action probability vector based on the selected action, the current action probabilities, and the generated reinforcement signal, such that $p(t+1) = \tau(\alpha(t), p(t), r_t)$.

FALA is an iterative algorithm with a relatively simple flow. At each time step t, the agent selects an action $\alpha(t)$ in accordance with the probability vector $p(t)$ and carries out that action against the environment. The result of this action is interpreted by the reinforcement function r and an appropriate reinforcement signal r_t is generated. This signal is passed to the agent, which updates it's probability vector in accordance with the learning function τ. The process repeats until the maximum number of iterations is reached or the action probabilities converge to a single value. In some cases the latter outcome is undesirable, and there exist a number of techniques that can be used to prevent this from happening [7]. The most obvious approach is to enforce a minimum value for each action probability, which is guaranteed to prevent convergence. Other techniques include negative reinforcement, proportional reinforcement, and staged learning.

3.1 Action Probability Minimums

Placing a floor on the values of the action probabilities ensures that no action will be completely removed as a possibility. The slack available to the learning algorithm is proportional to the sum of the differences between the starting and minimum probabilities of all actions. The decision of where to set the probability minimums comes down to trial and error as these values are effectively a set of hyperparameters tuned by the programmer.

3.2 Negative Reinforcement

Negative reinforcement is the punishment applied to actions that have undesirable outcomes and results in their probabilities being adjusted downwards. While it may be used at any point during the learning process, it is often applied only once an action probability exceeds a certain degradation threshold, which is generally set to be equal to or greater than the initial probability for a given action. The degradation thresholds for each action are determined by the programmer, and are another set of hyperparameters that need to be tuned. Negative reinforcement, when applied selectively to probabilities exceeding their degradation thresholds, makes it very difficult for any given action probability to dominate the others.

Algorithm 1 FALA Psuedocode

```
$iterations\_in\_stage$ = 0
for $t$ in $0..MAX\_ITERATIONS$ do
    for each $\Gamma^i$ do
        $\alpha^i(t) = select\_action(\alpha^i, p^i(t))$
        $execute(\alpha^i(t))$
    end for
    for each $\Gamma^i$ do
        $r_t^i = r^i(\alpha^1(t), \ldots, \alpha^n(t))$ # potential for negative reinforcement
        $r_t^i = scale(r_t^i, p^i(t))$ # proportional reinforcement, [OPTIONAL]
        $p^i(t+1) = \tau(\alpha^i(t), p^i(t), r_t^i)$ # must enforce probability floors
    end for
    if $iterations\_in\_stage$ == $ITERATIONS\_PER\_STAGE$ then
        $iterations\_in\_stage$ = 0
        for each $\Gamma^i$ do
            $p^i(t+1) = p^i(0)$ # may wrap with logic block, [OPTIONAL]
            $adjust(prob\_floors^i, degradation\_thresholds^i)$ # [OPTIONAL]
        end for
    end if
end for
```

3.3 Proportional Reinforcement

Proportional reinforcement means scaling the reinforcement signal based on the current probability of the action being reinforced. The positive reinforcement for high action probabilities is scaled downwards and the positive reinforcement for low probabilities is scaled upwards. The opposite is true for negative reinforcement. This helps actions with lower probabilities learn rapidly and slows the learning of actions with higher probabilities. The reinforcement may be scaled either to the starting probability of an action or to the largest value in the probability vector. This makes it progressively more difficult to learn as the probabilities for certain actions increase, and if used alongside negative reinforcement can significantly impede the learning of high probability actions.

3.4 Staged Learning

In staged learning, the FALA process is broken down into a number of stages. Each stage consists of a certain number of iterations, and a stage transition occurs at the end of each stage. During a stage transition the action probabilities may be reset to their initial values, thereby forgetting all the learning that took place in the previous stage. This allows the algorithm to re-learn from scratch, enabling it to continuously adapt to a changing environment. Optionally, the minimum values and the degradation thresholds for each action probability may be altered during a stage transition. This increases the slack afforded to the learning algorithm as it advances through the stages, and makes learning easier by applying negative reinforcement at increasingly higher thresholds.

3.5 A Game of FALA

A game of FALA refers to situations where at every timestep, an agent takes not one but several interdependent actions, meaning it is impossible to attribute the outcome to any single action [12]. The actions are chosen independently but due to the inherent interdependence, the choice of action from one process may affect the desirability of the others. Reinforcement is applied based on the cumulative result, where each FALA process has it's own interpreter but the outcome being interpreted is identical across all learning processes. The algorithm reinforces desirable combinations of actions from all processes and not the most desirable actions in each process. Mathematically, a game of FALA is nothing more than a list of quadruples: $\mathbf{\Gamma} = (\mathit{\Gamma}^1, \mathit{\Gamma}^1, \ldots, \mathit{\Gamma}^n)$, where n is the total number of concurrent FALA processes [18]. A high level pseudocode implementation of the complete FALA algorithm as described in this section is presented below in Algorithm 1.

4 Applying FALA to EXAMM

A single iteration of the combined FALA-EXAMM algorithm consists of a single EXAMM iteration augmented with a FALA component to adjust the action probabilities. It is composed of the following steps:

1. Choosing an island according to the island probabilities
2. Selecting a random genome from that island
3. Evolving it using an operation selected according to the genetic operation probabilities and training it for 10 back propagation iterations
4. Evaluating it to determine whether it should be inserted
5. Updating the probabilities of all actions accordingly

As EXAMM progresses through the evolutionary process it maintains a population of candidate genomes spread across a certain number of islands, with each island maintaining a given number of genomes. These two parameters are used to calculate an appropriate number of iterations per stage, which is a multiple of the product of the number of islands and the number of genomes per island. The number of iterations per stage is a constant calculated once at the beginning of the evolutionary process. It is used to determine when a stage transition occurs, and by extension the total number of stages in a given run. The performance of a newly generated genome is evaluated when EXAMM attempts to insert it back into the island it was picked from, and a reinforcement signal is calculated based on two factors: the fitness of the genome and its ranking within the island. Positive reinforcement is applied if the ranking is non–negative, which indicates that the genome was successfully inserted. Negative reinforcement is applied if the genome was not inserted and if the probability associated with the operation that generated that genome exceeds its degradation threshold. The probabilities of all actions are then updated using their respective learning functions, which ensure that no probability drops below its minimum threshold.

Stage transitions occur as necessary and the associated operations take place. The evolutionary process needs to warm up for approximately 100 iterations before probability adjustment is viable, so no learning takes place during the first stage. The starting action probabilities, minimum probabilities, degradation thresholds, and reinforcement functions are determined independently for each FALA process. For all learning processes, the positive reinforcement signal is calculated as a weighted average of a genome's ranking and its test set performance, whereas the negative reinforcement signal is a fixed constant. EXAMM's varying probabilities were adapted by FALA as follows:

Genetic Operation Probabilities: The action set and initial probability vector $p(0)$ are detailed in Table 1. There are no additional considerations as this process maps naturally to the FALA framework.

Island Selection Probabilities: At each iteration, EXAMM initially selects an island in a round–robin manner, choosing a genome from the island at random and evolving it. This is modified so that after the first stage is complete, EXAMM selects an island according to an evolving distribution. The action set is the collection of all islands, and the probability vector stores the associated probabilities. The probability vector is initialized uniformly with each island receiving a starting probability equal to $1/N_{islands}$. The reinforcement signal has an additional component, granting a small reward to the island with the best genome after each iteration.

Recurrent Depth Probabilities: The depth of a newly inserted recurrent edge is initially drawn from a uniform distribution, with values ranging between the minimum and maximum recurrent depths set by the programmer. Newly created recurrent edges are marked so that the learning process can identify which recurrent edges belong exclusively to the new genome, ensuring that only those depths that contributed to the improvement are reinforced. The action set is the list of all possible recurrent depths, and the initial probabilities are initialized uniformly to $1/N_{depths}$ for each depth. Some operations may result in the addition of multiple recurrent edges in a single iteration. The reinforcement function accounts for this by dividing the total reinforcement by the number of new recurrent edges and reinforcing each depth accordingly.

4.1 Exploration vs Exploitation

The exploration vs exploitation dilemma is common in machine learning, and is certainly present in this context. When it comes to FALA, exploration generally implies slower learning to allow the algorithm to explore various combinations without getting too entrenched in initial positive results. On the other hand, exploitation suggests faster learning where actions that initially lead to good results are strongly reinforced. It is possible to play these concepts off of one another: one option is to have fast learning with generous slack but to frequently reset the probabilities by having short stages. This allows the algorithm to rapidly increase the probabilities of the actions that are doing well while the frequent resets during stage transitions force the algorithm to re–discover desirable actions. Another option is to have slower learning and longer stages. This approach takes longer to learn but it is more robust to outlier situations where an action which is not generally desirable is reinforced due to some low–probability event. This tradeoff is controlled by adjusting the various FALA hyperparameters. It is interesting to note that a slower learning approach with no stage transitions may end up being more exploitative in the long run, as the probabilities get fixed over time. An approach that includes staging to ensure that probabilities are reset is going to be more exploratory due to the ability to reconfigure parameters to the current state of the evolutionary process more quickly.

5 Experimental Approach

In order to evaluate the efficacy of FALA in improving the performance of EXAMM, three different configurations were considered: Base EXAMM to provide a point of comparison, a slower learning approach, and a faster learning approach. Each configuration was executed 10 times, and each run was configured to use five islands each with a population of five genomes. Initial probabilities for genetic operations were set in accordance with the values shown in Table 1. The minimum recurrent depth was set to 1 and the maximum recurrent depth was set to 10. The runs were programmed to terminate when 2000 genomes were generated. The FALA hyperparameters for each configuration are summarized in Table 2. It is noteworthy that the number of hyperparameters for a given learning process is lower than that of base EXAMM, and may be reduced even further through additional refinement of the algorithm. This fact highlights an additional benefit of the proposed approach: there are less hyperparameters for the programmer to tune. Furthermore, the number of FALA hyerparameters is relatively constant and does not increase as new actions are introduced.

The best genome from each run was recorded and the results were compared using the Mann-Whitney U test to determine whether the difference between the run configurations was significant. All runs were carried out using the coal–fired powerplant dataset, which is a collection of minute– by–minute readings taken from 12 sensors in a coal powerplant burner over 120 days. Readings from sensors 1 to 11 are used as predictor variables and the reading from sensor 12 (main flame intensity) is the response.

6 Results

The experimental results are presented in Table 3 and visualized in Figure 1. The comparisons of different populations using the Mann-Whitney U test are presented in Table 4. These results demonstrate that the extension of EXAMM with FALA was successful in improving the overall performance. It is clear that the means and medians of the slower and faster learning configurations are better than those of the base algorithm. The result of the Mann-Whitney U test between

Table 2. Run Configurations

Approach	Learning Process	Stages	Learning Rates	Min Probs	Degr Thres	Neg Reinf
Slower Learning	Islands	4	0.005	0.2, mult by 0.965 every stage	0.2, add 0.02 every stage	0.3, add 0.05 every stage, scaled to degr thres
	Genetic	Unstaged	0.0005	Mutation: 0.055 CO: 0.17 Island CO: 0.09	Mutation: 0.075 CO: 0.21 Island CO: 0.105	0.5, scaled to degradation threshold
	Depths	Unstaged	0.0005	0.08	0.12	0.5
Faster Learning	Islands	10	0.007	0.2, sub 0.00625 every stage	0.2, add 0.00625 every stage	0.5, scaled to degr thres
	Genetic	10	0.002	Mutation: 0.04 CO: 0.14 Island CO: 0.06	Mutation: 0.10 CO: 0.26 Island CO: 0.14	1.0, scaled to degr thres
	Depths	10	0.002	0.06	0.12	1.0

the base configuration and slower learning configuration is significant at p = 0.05. The p–value of the Mann-Whitney U test between the base configuration and the faster learning configuration is relatively low but not considered significant at any commonly–accepted threshold.

Table 3. Experimental Results - Lowest Test Set MSE From Each Run

	Base EXAMM	Slower Learning	Faster Learning
Run 1	0.001418	0.0013	0.001778
Run 2	0.001584	0.001291	0.00119
Run 3	0.00178	0.001629	0.001228
Run 4	0.001472	0.001221	0.00127
Run 5	0.001492	0.001168	0.001436
Run 6	0.001342	0.00138	0.001271
Run 7	0.001254	0.000952	0.001356
Run 8	0.001456	0.001374	0.001526
Run 9	0.001966	0.001399	0.001218
Run 10	0.0012	0.001134	0.001536
Mean	0.0014964	**0.0012848**	0.0013809
Median	0.001464	**0.0012955**	0.0013135
Min	0.0012	**0.000952**	0.00119
Max	0.001966	**0.001629**	0.001778
Variance	5.4163*10-8	**3.3335*10-8**	3.5243*10-8

Assuming that the slower learning approach is indeed superior to the faster learning approach, it would appear that in the context of EXAMM, exploration is preferred to exploitation. This could be because exploitation is difficult in such a dynamic environment, where the population of genomes is constantly being renewed. Another explanation could be that favoring any one selection too heavily is detrimental for the evolution of new genomes, implying that a successful genome is evolved when all selections are used at least somewhat evenly. To better understand which selections were more successful, we can analyze the plots of the different probabilities over time from different runs. Figures 2 and 3 show the evolution of genetic operation, island selection, and recurrent depth

Table 4. Comparison of Results from Table 3 Using the Mann-Whitney U Test

Population 1	Population 2	P-Value
Base EXAMM	Slower Learning	**0.03546**
Base EXAMM	Faster Learning	0.2799
Slower Learning	Faster Learning	0.4359

probabilities for the best runs in both the slower and faster learning approaches respectively. Figures 4 and 5 show the same graphs but for the worst run from each configuration.

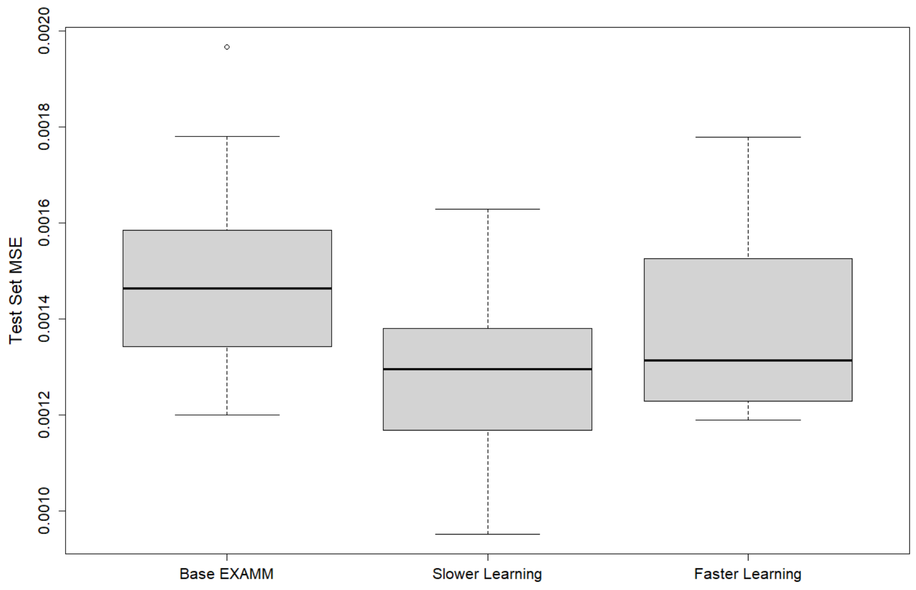

Fig. 1. Distribution of Results from Table 3 by Configuration

The difference between the slower and faster approaches is evident through the evolution of genetic operation and recurrent depth probabilities. This provides a good visualization of the exploration vs exploitation tradeoff discussed in section 4.4. The evolution of the island selection probabilities is actually somewhat comparable, apart from the fact that one has four stages and the other has ten. This makes sense because their hyperparameter configuration as outlined in Table 2 is almost identical. It is interesting that for both of the worst runs, the island selection learning process has trouble settling on a best island, whereas for the best runs the learning process settles on an island (or two) early on. A potential explanation for this is that the performance of a run is largely dependent on how well it starts. If the algorithm has a poor start, we tend to observe multiple islands with comparable genomes that will compete with each other throughout the learning process. However, if the algorithm has a good start, there will likely be one or two islands with genomes that are superior to the populations of the other islands, and this is reflected in the probabilities of those islands being heavily reinforced.

Probability dynamics for genetic operations are better visualized in the faster configuration, and it is evident that no single operation dominates the others, despite the high learning rates and frequent resets. That being said, there are operations that almost never see their probabilities increased, implying that they are less likely than the others to generate good genomes. The crossover operation seems to always perform well, as do the clone, add recurrent edge, and add node operations. The add edge, disable edge, and island crossover operations also perform well, whereas the remainder are almost never reinforced. Of the ones that are reinforced, the disable edge operation is most surprising, since intuition would suggest that disabling edges would lower the complexity of the network and therefore result in worse performance. One potential explanation is that disabling edges acts as a form of regularization, which is critical in preventing overfitting.

The evolution of the recurrent depth probabilities shows that some depths are initially reinforced more than others. As the search progresses the importance of those depths decreases and others are reinforced instead. This suggests a certain degree of saturation, in that there is no more information to be gained after a certain number of recurrent edges have been added at a given depth. It is noteworthy that larger depths appear to have been consistently reinforced in both learning configurations, corroborating findings by Desell *et al.*regarding the utility of deep recurrent connections [3].

7 Conclusion

This paper demonstrates that reinforcement learning can be successfully applied to guide EXAMM's neuroevolution process. FALA is used to dynamically adjust the probabilities associated with various selections made by the base algorithm at each iteration. Techniques for manipulating the learning process have also been presented, and should allow others to adapt FALA to different problem sets. The performance of this augmented algorithm was evaluated against real-world data and the results demonstrate a significant improvement in performance when the learning is configured correctly. This paper also opens up several avenues for future work. Testing different FALA configurations over a larger number of runs and iterations per run is likely to result in the discovery of an even better learning configuration. Applying other reinforcement learning approaches such as continuous action reinforcement learning automata [5] or multi-armed bandit [19] may result in even better performance. Furthermore, this approach can be applied to NEAT or its other derivatives, such as the Evolutionary Exploration of Augmenting Convolutional Topologies (EXACT) algorithm [2].

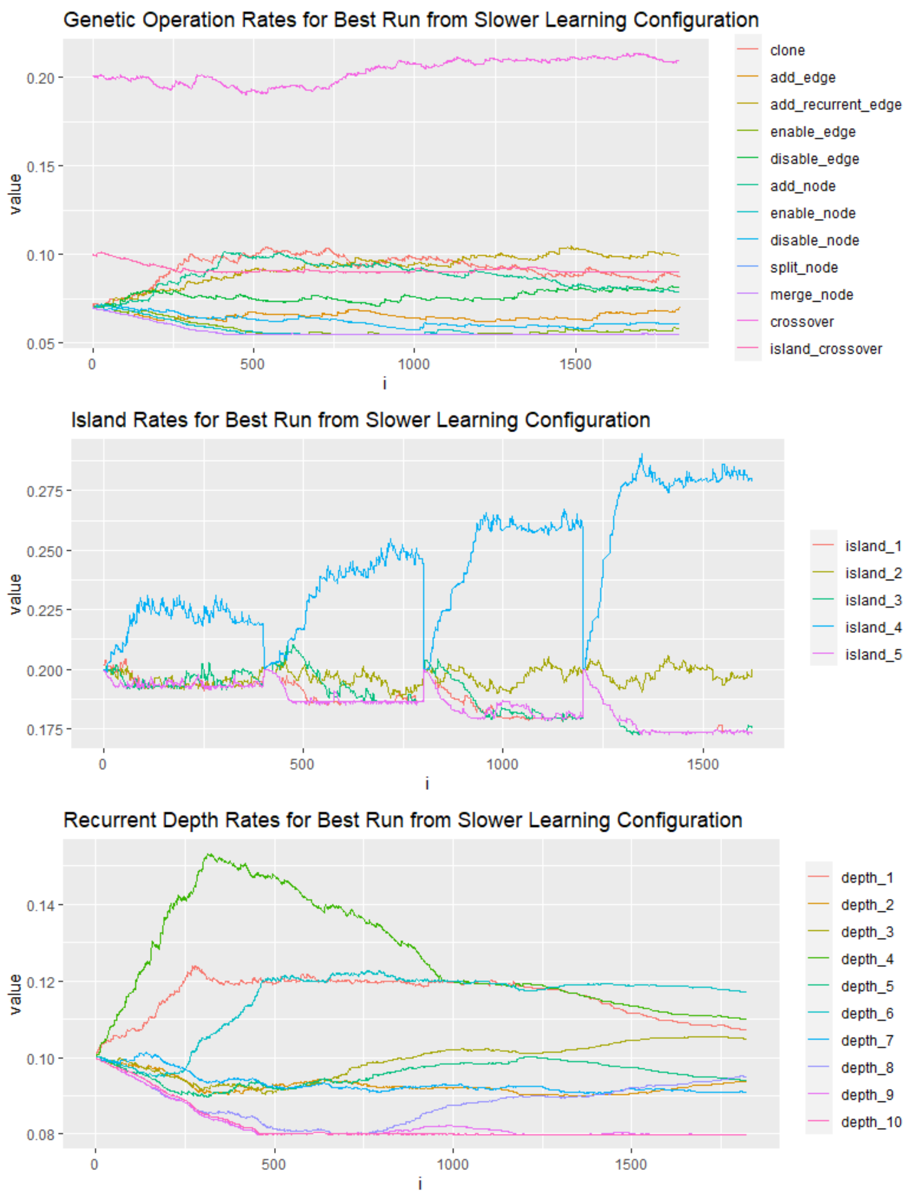

Fig. 2. Evolution of Probabilities from Best Run of Slower Learning Configuration

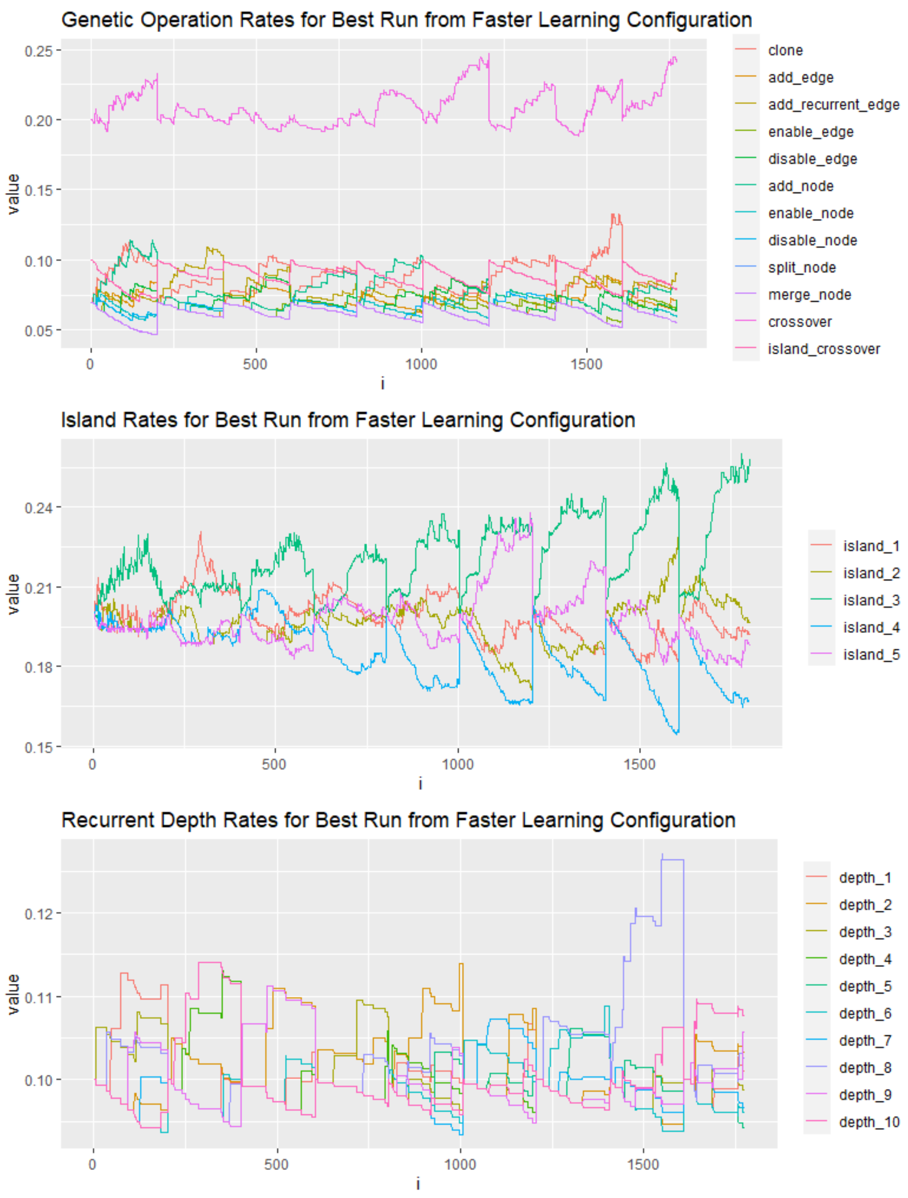

Fig. 3. Evolution of Probabilities from Best Run of Faster Learning Configuration

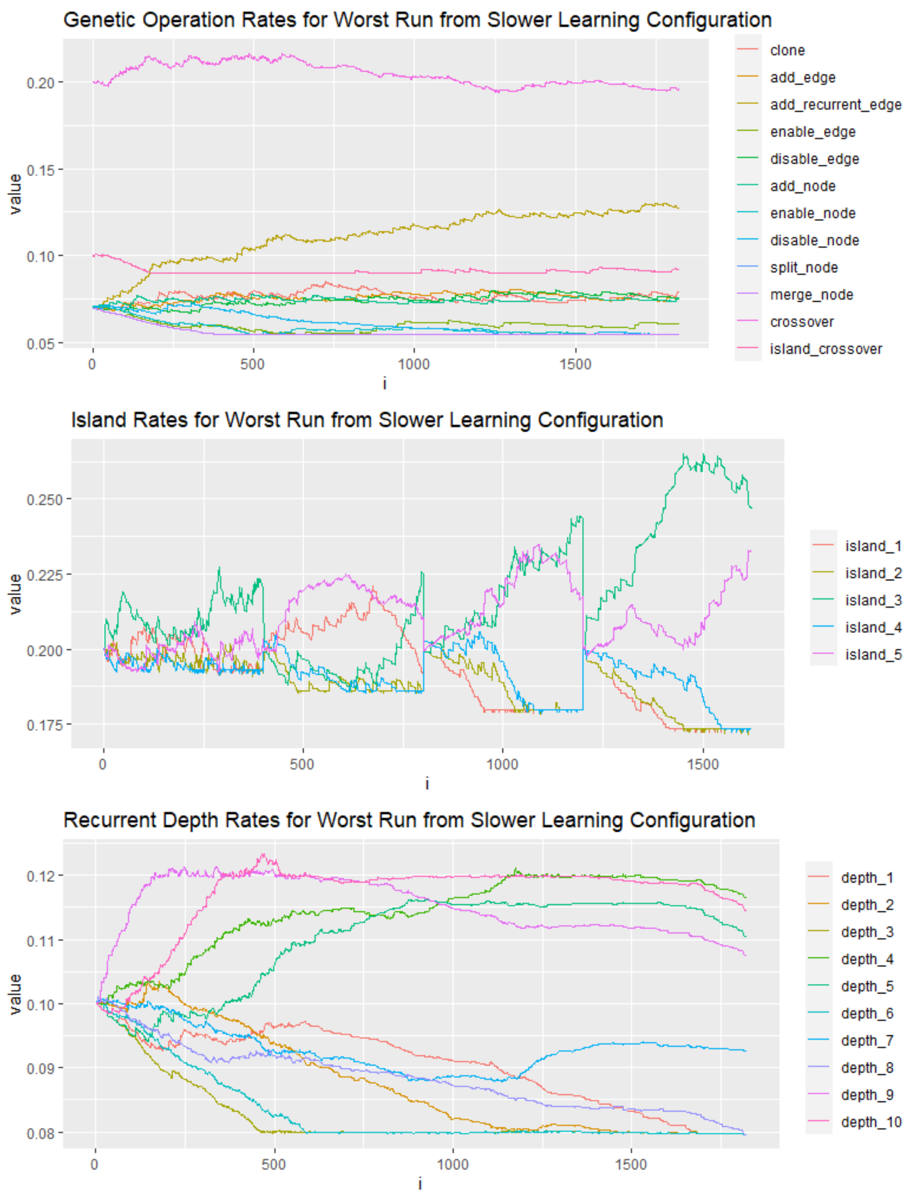

Fig. 4. Evolution of Probabilities from Worst Run of Slower Learning Configuration

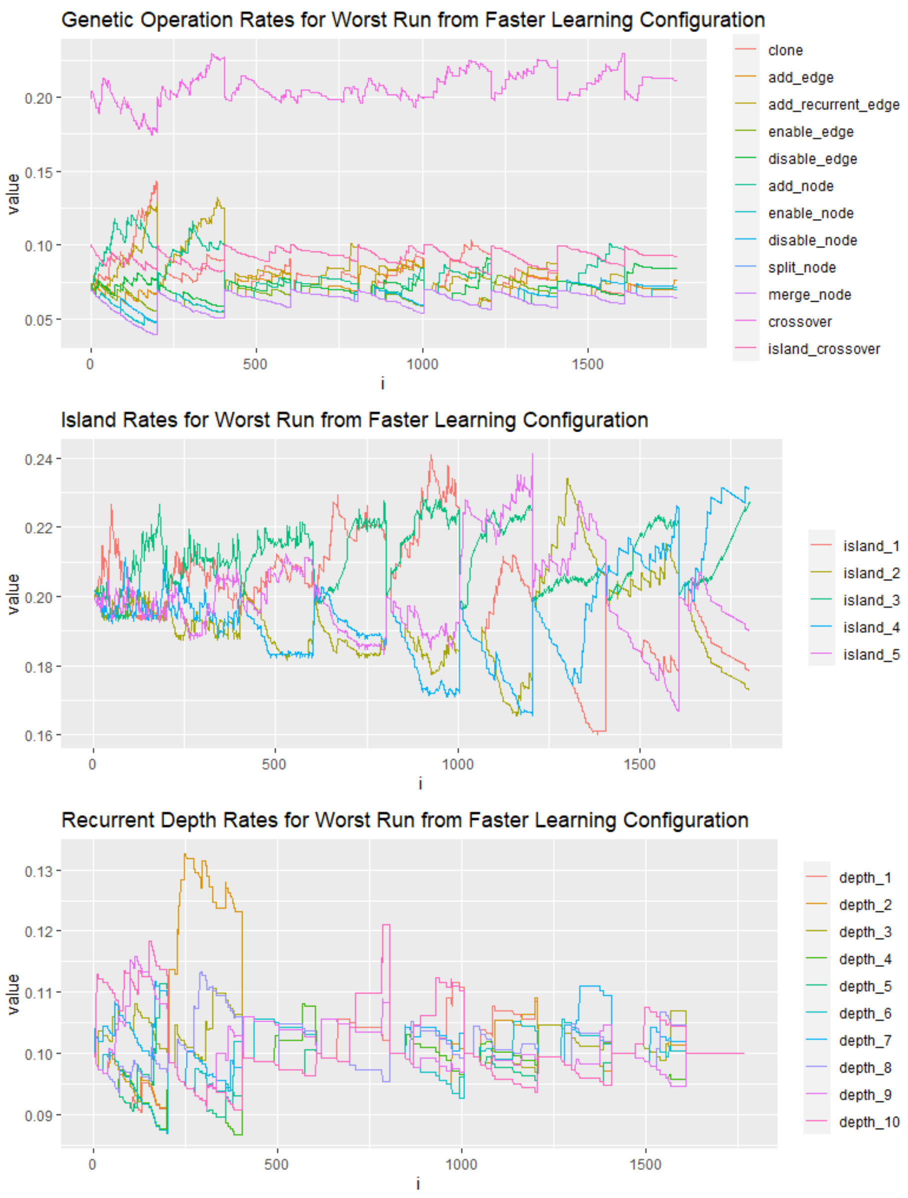

Fig. 5. Evolution of Probabilities from Worst Run of Faster Learning Configuration

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. Evolutionary Computation, IEEE Transactions on **6**, 443 – 462 (11 2002)
2. Desell, T.: Large scale evolution of convolutional neural networks using volunteer computing. pp. 127–128 (07 2017)
3. Desell, T., ElSaid, A., Ororbia, A.: An Empirical Exploration of Deep Recurrent Connections Using Neuro-Evolution, pp. 546–561 (04 2020)
4. Floridi, L., Chiriatti, M.: Gpt-3: Its nature, scope, limits, and consequences. Minds and Machines **30**, 1–14 (12 2020)
5. Howell, M., Best, M.: On-line pid tuning for engine idle-speed control using continuous action reinforcement learning automata. Control Engineering Practice **8**, 147–154 (02 2000)
6. Jardine, P.T., Kogan, M., Givigi, S.N., Yousefi, S.: Adaptive predictive control of a differential drive robot tuned with reinforcement learning **33**(2), 410–423 (Apr 2018)
7. Jardine, P.: A Reinforcement Learning Approach to Predictive Control Design: Autonomous Vehicle Applications. Ph.D. thesis (05 2018)
8. Jardine, P.T., Givigi, S.N., Yousefi, S.: Experimental results for autonomous model-predictive trajectory planning tuned with machine learning. In: 2017 Annual IEEE International Systems Conference (SysCon). pp. 1–7 (2017)
9. Kogan, M., Jardine, P.T., Givigi, S.N.: Architecture for testing learning-based autonomous vehicle control design. In: 2018 Annual IEEE International Systems Conference (SysCon). pp. 1–7 (2018)
10. Lyu, Z., Karns, J., ElSaid, A., Desell, T.: Improving neuroevolution using island extinction and repopulation (05 2020)
11. Matuszewski, J., Rajkowski, A.: The use of machine learning algorithms for image recognition. In: Radioelectronic Systems Conference 2019. vol. 11442, pp. 412 – 422 (2020)
12. Narendra, K.S., Thathachar, M.A.L.: Learning automata - a survey. IEEE Transactions on Systems, Man, and Cybernetics **SMC-4**(4), 323–334 (1974)
13. Ororbia, A., ElSaid, A., Desell, T.: Investigating recurrent neural network memory structures using neuro-evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 446–455 (2019)
14. Radaideh, M.I., Shirvan, K.: Rule-based reinforcement learning methodology to inform evolutionary algorithms for constrained optimization of engineering applications. Knowledge-Based Systems **217**, 106836 (2021)
15. Barros dos Santos, S.R., Givigi, S.N., Nascimento, C.L.: Autonomous construction of multiple structures using learning automata: Description and experimental validation. IEEE Systems Journal **9**(4), 1376–1387 (2015)
16. Sejnowski, T.J.: The unreasonable effectiveness of deep learning in artificial intelligence. Proceedings of the National Academy of Sciences **117**(48), 30033–30038
17. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation **10**(2), 99–127 (06 2002)
18. Thathachar, M.A.L., Sastry, P.S.: Networks of Learning Automata: Techniques for Online Stochastic Optimization. Springer-Verlag, Berlin, Heidelberg (2003)
19. Vermorel, J., Mohri, M.: Multi-armed bandit algorithms and empirical evaluation. In: Machine Learning: ECML 2005. pp. 437–448 (2005)
20. Weiss, K., Khoshgoftaar, T., Wang, D.: A survey of transfer learning. Journal of Big Data **3** (05 2016)