Forgetting Beneficial Knowledge in Decomposition-Based Reinforcement Learning Using Evolutionary Computation

Sean Mondesire[1] **and R. Paul Wiegand**[2]
[1]College of Engineering and Computer Science, University of Central Florida, Orlando, Florida, USA
[2]Institute for Simulation and Training, University of Central Florida, Orlando, Florida, USA

Abstract— *This work demonstrates that critical information can easily and prematurely be removed from a decomposition-based reinforcement learning system. One possible effect to the forgotten knowledge is the complete loss in ability to solve a previously learned problem when the system is given a new problem to optimize. In artificial neural networks, this is called catastrophic forgetting and has been shown to cripple performance. We study this phenomenon to understand its effects on problem performance and to investigate suspected consequences experienced by other decomposition-based approaches. Furthermore, using an abstract decomposition-based reinforcement learning paradigm with a simple evolutionary algorithm, we analyze the role stability-plasticity imbalance has in the premature loss of critical knowledge.*

Keywords: Stability-Plasticity Dilemma, Decomposition-Based Reinforcement Learning, Evolutionary Computation

1. Introduction

Using evolutionary computation, we demonstrate the unintentional loss of beneficial knowledge in decomposition-based reinforcement learning. This premature removal of critical information has had disastrous effects on performance and learning rates in other learning genres. By providing this demonstration, we highlight the challenge to those who use decomposition-based reinforcement learning, identify potential causes, and provide detailed analysis of the effects of forgetting. Additionally, the reinforcement learning and evolutionary computation communities gain a deeper understanding of why this type of forgetting can be so devastating to learning effectiveness. Finally, this demonstration is needed to uncover potential performance-hindering pitfalls future learning systems can prepare for.

Reinforcement learning (*RL*) is a machine learning area that solves problems based on maximizing the expected cumulative reward of a system's actions [1]. *Decomposition-based reinforcement learning* (*DBRL*) is a RL method that optimizes a learning system's action-selection by solving a series of simpler subproblems to solve a complex problem. Hierarchical abstract machines [2], MAXQ [3], and layered learning [4] are examples of DBRL techniques used to train computer-based agents to solve complex problems.

One challenge of DBRL is retaining the ability to solve previously learned problems (stability) while learning to solve new ones (plasticity). This balancing act is called the *stability-plasticity dilemma* [5]. Systems that favor stability may have difficulty in adapting to learn how to solve new problems quickly; on the other hand, a preference for plasticity can result in the rapid loss of solutions to previously learned problems. It is this imbalance that favors plasticity and the loss of *beneficial knowledge*, critical information that improves performance of a solution, that we are focused on in this work.

Existing literature reveals a preference for plasticity in artificial neural networks can severely hinder performance of a learning system [6], [7]. The problem arises when a network is optimized on a set of inputs to solve one problem, then optimizes its activation weights on a new problem's input. The second set of inputs causes the network to override weights necessary to solve first problem. This imbalance for plasticity can be extremely detrimental to earlier learned problems to a point where performance of the first problem is completely lost. This occurrence is called *catastrophic forgetting* [8].

Previous work in HBRL using evolutionary computation methods have suggested and demonstrated that catastrophic forgetting can occur in the learning approach. For example, in a direct policy search used to optimize autonomous agent policies in a predator-prey scenario, experiments have shown that previously learned proficiency of critical skills can drop significantly when an agent learned a new skill [9]. Because each skill was vital in solving the problem the agents were learning, overall performance and the learning rate suffered. Though the work did not explicitly focus on forgetting phenomenon, the work's experiment results suggest that a stability-plasticity imbalance was the cause of the knowledge loss.

In a second work, catastrophic forgetting in HBRL using an evolutionary algorithm was analyzed while verifying forgetting diagnostic and measurement metrics [10]. The experiments demonstrated that the optimization of new problems drove out critical, older knowledge. This caused previously learned solutions of older problems to be lost and older problem performance to drastically decrease. The work used contrived Boolean-logic problem that purposely contained conflicting subproblems, ensuring older beneficial

knowledge would be replaced by new, conflicting information.

In this work, we demonstrate the implicit premature loss of beneficial knowledge in DBRL using simple, well-crafted methods and problems. This work is different from others because it explicitly studies the loss of beneficial knowledge while learning. Additionally, it uses standard optimization problems that are not contrived to demonstrate forgetting in this area of RL.

To achieve the demonstration goal, we use the abstract DBRL paradigm of *layered learning* with a *(1+1) Evolutionary Algorithm* (*[1+1] EA*) to solve two Boolean-logic problems. Layered learning was selected because of its qualities shared with many other RL techniques, including robot shaping and transfer and incremental learning. Because of these shared characteristics, demonstrating catastrophic forgetting in layered learning may lead to important stability-plasticity findings in related approaches. A simple (1+1) EA is used because it reduces complexity to the employed learning algorithm and makes experiment results easier to interpret. By accomplishing our goal, we will provide a better understanding of what causes the imbalance and its the effects.

2. Related Work

Catastrophic forgetting is caused by the premature removal of critical information relied on for problem solving from a system. The challenge is trying to understand what causes the premature knowledge loss.

Several methods have had success in trying to delay the loss of important knowledge or regain forgotten information. These methods include the use of deletion strategies, complementary learning systems, and rehearsal techniques, to name a few.

Deletion strategies are mechanisms used to determine when a pair is removed from a policy. Some strategies include random, frequency, temporal, spatial, and utility-based removal.

Rehearsal techniques repeat the optimization of a problem to reintroduce forgotten knowledge, reinforce solutions previously learned, or further improve the current problem solving ability [11], [12]. By repeating the optimization of a problem, a system can revert policy changes to mappings that increased performance of a solution.

Complementary learning systems have shown measurable success in preventing complete proficiency loss in neural networks. The system applies dual-memory storage regions to a network originally with one, in a manner that is similar to short- and long-term memory in natural systems. Examples of these two-phased learning systems include Hattori's use of a hippocampus-neocortical configuration with a Hopfield network [13], *pseudorehearsal* [11], *pseudo-recurrent networks* [14].

These discussed works demonstrate that there is a forgetting problem in other learning systems and provide mitigation techniques; unfortunately, the same amount of attention has not been given to DBRL. Two works that explore forgetting in this type of learning are in [9], [10]. Both works combine layered learning, an abstract DBRL paradigm, with an evolutionary algorithm to facility learning of complex problems.

The first work studies the effectiveness of a DBRL when applied to a direct policy search to train autonomous *non-playable characters* (*NPCs*) in a predator-prey based video game [9]. The study decomposes a predator behavior into three subproblems and sequentially trains the NPC policies on each. The result shows that as the performance criterion changes during the optimization of each subproblem, performance of an older subproblem decreases. The work concludes that the learner's sub-optimal performance, which is compared to a monolithic learner, is caused by the decreases in subproblem performance.

The second work introduces two performance-based forgetting metrics [10]. The metrics, *direct* (*DFM*) and *maximized* (*MFM*) forgetting metrics, are proposed to classify and measure a type of forgetting that is occurs in an RL system. To validate and verify the metrics, experiments use layered learning with a (1+1) EA to optimize a bit-string over the *leading ones trailing zeros* (*LOTZ*) problem. LOTZ and the employed decomposition of subproblems are contrived to purposely drive out older information from the system by having conflicting subproblems optimize on between consecutive layers. Experiment results show performance of the LOTZ problem suffers because critical subproblem performance decreases. The work concludes that under these conditions, the metrics are validated and that a demonstration of forgetting in the abstract DBRL system took place.

The research in this paper is an extension of these two works ([9], [10]) and explicitly demonstrates the loss of beneficial knowledge in an DBRL system. To do so, we use two completely different, less contrived Boolean-logic problems to show the forgetting pitfalls and challenges of an RL system. Furthermore, we purposely avoid all mitigation techniques to study what happens when nothing is explicitly in place to safeguard the learning system.

3. Methodology

To demonstrate forgetting, we use layered learning in combination with an evolutionary algorithm to solve two Boolean-logic problems. The objective is to show that as the learner optimizes a solution for a new problem, performance of an older, previously learned problem significantly decreases. It is suspected that several factors will cause beneficial knowledge to be driven out of the system, including subproblem conflict and the greedy nature of the employed algorithm. This investigation will determine

the causes of catastrophic forgetting under these simple experiment conditions.

Layered learning is an abstract machine learning paradigm that solves a complex problem by incrementally optimizing performance of simpler sub-problems [15], [4]. The paradigm has been used to train real and simulated autonomous robots and computer-based systems to solve a variety of complex problems, such as performing motor-skills and playing robotic soccer [16], [17], [18]. The paradigm learns by establishing and then sequentially iterating through a set of *layers*. Each layer organizes a learning plan by indicating the subproblem to optimize, the training scenario and conditions for that optimization, and the halting condition to signal when to proceed to the next layer, each predefined by the developer. The paradigm is abstract because it provides a general process of how to solve a complex problem using decomposition. Therefore, layered learning is not a machine learning algorithm but instead uses existing learning algorithms to facility learning.

We combine layered learning with a *(1+1) evolutionary algorithm* (*[1+1] EA*) to provide our demonstration. The (1+1) EA is a simple EA that uses one genetic operator to modify the sole individual in the population [19], [20]. Because the employed learning algorithm only has one operator and individual, the learning process is less complicated, allows for easier interpretation of what occurs during problem optimization, and focuses on the decomposition instead of the underlying algorithm.

This learning system optimizes a 128-bit long bit-string to solve two Boolean-logic problems: OneMax and Spin Glass. These problems were selected because for their simplicity and difference in linearity. OneMax is a linear function that has become a standard optimization problem in machine learning. Spin Glass is a natural non-linear problem that is not designed to demonstrate forgetting. By using these completely different problems, we demonstrate that the unintentional loss of beneficial knowledge can occur under conditions on opposite sides of the linearity spectrum.

Finally, we recognize that these two Boolean-logic problems are optimization problems. We use these simple problems and bit-string representation to symbolize more complex and common RL problems and representations. For instance, the bit-string may abstractly represent a policy in a RL's direct policy search. Here, each bit represents a state-action mapping (a *pair*) an agent can perform at any decision point, while the bit-string is the *policy* (the collection of pairs). Optimization of the bit-string on a Boolean-logic problem could then correspond to the optimization of an autonomous agent's policy on a RL multi-agent system problem.

3.1 OneMax

OneMax is a standard linear test problem that rewards the bit string for the number of 1 bits it contains, where the all-one string is optimal. The OneMax performance function is defined in Equation 1.

$$x \in \{0,1\}^n, OneMax(x) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

3.2 Spin Glass

In the Spin Glass problem the bit string is rewarded for every occurrence where consecutive bits and the first and last bits differ [21]. The Spin Glass performance function is defined in Equation 1, where $\oplus$ represents exclusive-or (xor). This problem represents natural problems with interacting, non-linear relationships between individual decisions.

$$x \in \{0,1\}^n, SpinGlass(x) = \frac{1}{n}(\sum_{i=1}^{n-1}(x_i \oplus x_{i+1}) + (x_n \oplus x_1)) \tag{2}$$

Table 1: Task and Subproblem Descriptions

Subproblem	**Maximization Objective Description**
OneMax	One bits in the entire string.
FirstHalfOneMax	Ones in the first half of the string.
SecondHalfOneMax	Ones in the second half of the string.
SpinGlass	Different consecutive bits and first and last bits.
MiddleOneThirdSpinGlass	Spin Glass over the middle 1/3rd bits.
MiddleTwoThirdSpinGlass	Spin Glass over the middle 2/3rd bits.
FirstThirdSpinGlass	Spin Glass over the left-most 1/3rd bits.
FirstTwoThirdSpinGlass	Spin Glass over the left-most 2/3rd bits.

Each Boolean-logic problem is decomposed into subproblems, displayed in Table 1. Each problem has two *test cases* that determine the halting condition of a layer. The *find optimal* trigger to move to the next layer is not activated until the subproblem's performance has reached optimality. The second is *fixed duration*, which runs a fixed number of mutations and evaluations (*time-steps*) in a layer before proceeding to the next. Within a test case are several test plans. A *test plan* is the decomposition used to solve the overall complex problem the system is optimizing. In a layer, the (1+1) EA continuously optimizes the bit-string on a test plan's subproblem until the test case's halting condition is satisfied.

The learning process is as follows: in the initial layer, a randomly generated bit-string is passed into a layer. A clone of the bit-string is then made and is mutated. The mutation operator modifies the clone by traversing over each bit, toggling a bit's value at a 1/*n* probability. Using the subproblem's performance evaluation function, the clone's performance is compared to that of the parent. If the clone's subproblem score is greater than the parent's, the clone replaces the parent in the population. This process of cloning, mutating, and evaluating repeats until the test case's halting

condition is satisfied. Once a layer is halted, the bit-string at the end of one layer is the initial bit-string for the next.

Test cases with one subproblem/layer act as the control for these experiments as it monolithically learns the complex Boolean-logic problem without decomposition. This control is important to our demonstration because monolithic learning using this (1+1) EA does not experience forgetting of a complex problem because the sole individual is only replaced if a clone outperforms the parent. In other words, performance is guaranteed to never decrease.

Finally, the fixed duration limits the number of time-steps in a test plan to be 90% of the averaged required time-steps to produce the optimal bit-string in the corresponding monolithic *find optimal* test plan.

4. Experiment Results

All test plans for both Boolean-logic problems were evaluated for 100 trials to have a large enough sample to draw significant conclusions. First we discuss our catastrophic forgetting demonstration using OneMax.

4.1 OneMax

Both of the OneMax Experiment test cases contained five test plans, defined in Table 2.

Table 2: OneMax Test Plans with Subproblem Sequence

Test Plan #	Layer 1	Layer 2	Layer 3
Plan0	OneMax	-	-
Plan1	FirstHalfOneMax	OneMax	-
Plan2	SecondHalfOneMax	OneMax	-
Plan3	FirstHalfOneMax	SecondHalfOneMax	OneMax
Plan4	SecondHalfOneMax	FirstHalfOneMax	OneMax

Table 3 contains the averages and standard deviations over the 100 trials of the total number of time-steps required to maximize each layer's subproblem performance for each of the test plans. The monolithic plan (plan 0) required the least amount of time-steps to transform the randomly generated bit string into the all ones-string, requiring an average of 1,451.11 time-steps. Plans 1 and 2 required the next least amount of time-steps, 2,066.4 and 2,166.45, respectively. The plans with three layers of decomposition, plans 3 and 4, used the most time to generate the optimal bit strings for all of their subproblems, requiring 2,928.55 and 2,937.43 average time-steps to iterate through all of their layers.

A multi-sample comparison using a left-tailed Z-tests with the Bonferroni adjustment for multi-way comparisons confirms that the monolithic test plan outperformed all of the decomposition plans. From the Z-scores in Table 3 and the adjusted α value of 0.01 (pre-adjustment α = .05), each of the null hypotheses stating monolithic time-step averages are greater than or equal to those of plans 1 through 4 were rejected. The monolithic test plan's average number of time-steps to optimize the bit string over the OneMax problem is significantly less than the decomposition-based test plans.

Because each decomposition was dissimilar and found subproblem optimality at different rates, each test plan transitioned to new layers at different time-steps: plan 1 at 754, plan 2 at 832, plan 3 at 821 and 1,584, and plan 4 at 769 and 1,631.

Because a (1+1) EA has a greedy selection mechanism, the monolithic plan has no negative performance changes throughout the learning process. More saliently, it reaches optimality in all subproblem performances the fastest. Plans 1 and 2 reached OneMax optimality the second fastest, but both test plans experienced a slight decrease in their first layer's subproblem performance after a transition to the OneMax layer. The plans with three layers, plans 3 and 4, experienced the largest first layer's subproblem performance decrease of all of the plans. In plan 3's case, the average FirstHalfOneMax performance was optimized, yielding a perfect value of 1 at the end of layer 1. At the completion of layer 2, the same subproblem's average performance plummeted to 0.56, before optimizing again to a subproblem value of 1. Plan 4 had the same outcome, resulting in an average SecondHalfOneMax performance decrease from 1 at the end of the first layer to .59 at the end of the second layer.

Table 3: Average Time-Steps to Find Optimal OneMax Bit String

Test Plan	Time-Steps	STDV	Z-Score
Plan0	1,451.11	456.81	-
Plan1	2,066.4	460.35	-9.49
Plan2	2,166.45	536.29	-10.15
Plan3	2,928.55	574.57	-20.13
Plan4	2,937.43	582.16	-20.09

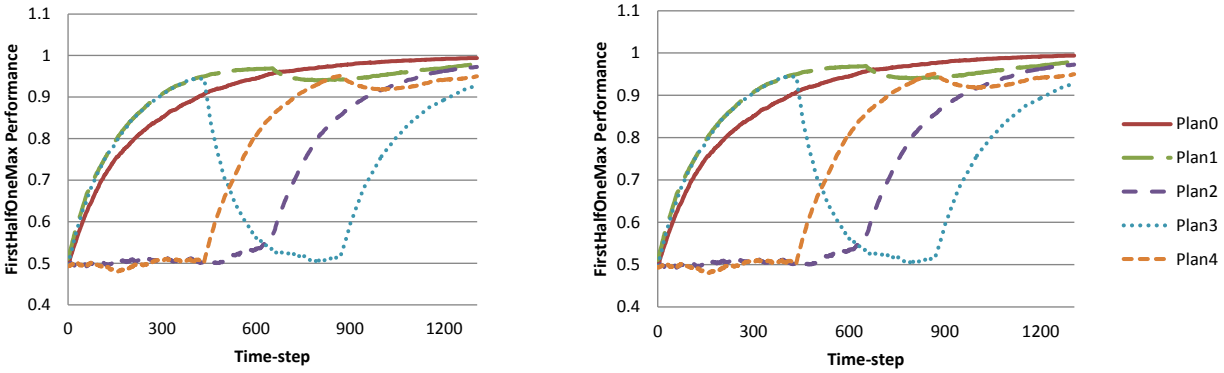
OneMax test case 2 continuously optimized the bit string on a subproblem for a fixed 1,306 time-steps to iterate through their layers. Each test plan's layer was then given the ceiling of the 1,306 divided by the number of layers in that particular test plan.

Table 4 displays the average final OneMax performances, their standard deviations, and Z-scores with the fixed duration halting condition. Contrasting from the first test case, the primary measure is subproblem performance instead of time-steps to find the optimal. Therefore, the closer the performance value is to 1, the more preferable the outcome.

Table 4: Average Final OneMax Performances with Fixed Layer Duration

Test Plan	OneMax Perf.	STDV	Z-Score
Plan0	0.9937	0.0065	-
Plan1	0.9723	0.013	14.73
Plan2	0.9732	0.01354	13.57
Plan3	0.9397	0.01901	26.87
Plan4	0.9382	0.01634	31.51

From the OneMax performance averages, again the monolithic test plan outperforms the others by producing a higher



(a) FirstHalfOneMax Performance

(b) SecondHalfOneMax Performance

Fig. 1: OneMax Subproblem Performance with Fixed Layer Duration

performing bit string at the end of the 1,306 time-steps. Bit strings developed by plans 1 and 2 were the next best performing and test plans 3 and 4 were the worst average performers. The Z-statistics tests confirm that the monolithic test plan significantly outperformed the decomposition approaches. In the Z-test, the claim was that the monolithic test plan's average OneMax performance over the 100 trials was greater than each of the decomposition-based test plans.

Following the first test case, performance of the first layer's subproblem decreases during the second layer in test case 2. In test plans 3 and 4, the layer transition from layer 2 to 3 also resulted in a small second layer subproblem performance decrease even though the first layer's subproblem increased. This observation is interesting because the third layer's subproblem depends on the first two layers' subproblem for optimality and a decrease was not expected. The explanation for the second layer's subproblem decrease is while the bit string is optimizing for the OneMax problem, over 90% of the bits for the second layer's subproblem are already set to 1. At the same time, close to 50% of the bits in the first layer's subproblem are still 0. These two differences in set bits means there was a higher probability that the mutation operator would correctly toggle zero bits affected in the first layer's subproblem simply because there were more zero bits. Combine these probabilities with the guarantee that the (1+1) EA only accepts mutations that progress the current layer's subproblem performance, it is understandable that the second subproblem's performance could decrease while the first subproblem's performance rises during the last layer.

Figures 1 (a) and (b) graph subproblem performances over the 1,306 time-steps. These figures show that when layer 2 is activated in plans 3 and 4, their first learned subproblem's performance suddenly drops then increases when layer 3 is activated.

These experiments demonstrate that in even very simple RL problems where decisions are linearly separable from a performance point of view, layered learning can experience a stability-plasticity imbalance.

4.2 Spin Glass

The test plans in the Spin Glass experiments used the subproblem sequences listed in Table 5.

In the first Spin Glass test case, the monolithic plan 0 required the least number of time-steps to generate the optimal bit string when compared to the two decomposition plans. Plan 0 required an average of 317,719.60 time-steps to satisfy the halting condition of its only layer. Plan 1 required 34,320.81 more time-steps and plan 2 required 15,804.66 more time-steps than plan 0. The monolithic plan's average number of time-steps to generate the optimal bit string was significantly lower than those required by the two decomposition plans. Table 6 displays the average time-steps, their standard deviations, and Z-scores comparing plans 1 and 2's average time-steps to the monolithic plan.

Table 5: Spin Glass Test Plans with Subproblem Sequence

Test Plan #	Layer 1	Layer 2	Layer 3
Plan0	SpinGlass	-	-
Plan1	Middle OneThird SpinGlass	Middle TwoThird SpinGlass	SpinGlass
Plan2	FirstThird SpinGlass	First TwoThird SpinGlass	SpinGlass

The second test case limited the number of time-steps to 285,947 derived from 90% of the first test case's monolithic average time-steps to generate the optimal string. From these trials, the monolithic test plan produced a Spin Glass average performance of 0.995 at the end of the limited layer duration. From the decompositions that segmented the subproblems in thirds, plan 1 averaged a Spin Glass performance of 0.821 and plan 2 averaged 0.989. The Z-tests comparing the monolithic and decomposition-based test plans confirm

that the monolithic approach produced a significantly higher average performance than plans 1 and 2. With a critical value of 1.96, Table 7 displays the average Spin Glass performance after all of the layer iterations have completed, their standard deviations, and Z-scores.

Table 8 displays the average subproblem performances for each test plan at the conclusion of each layer. One noteworthy observation is that the first subproblem of each decomposition-based test plan was optimized although the number of bit string manipulations was limited to 90% of the monolithic test plan's result in the first test case; the plan 0's final performance averaged very close to optimal, 0.995.

From Table 8, subproblem performance always decreased in the decomposition-based test plans after a transition from a subproblem's layer to another layer. For instance, plan 1's transition from layer 1 to layer 2 resulted in the performance of the MiddleOneThirdSpinGlass subproblem to decrease. The subproblem decreases were not as large as those in the OneMax experiment. The small decreases in performance is attributed to the problem decompositions being lower-ordered versions of the Spin Glass problem, where each layer's subproblem directly benefits from positive changes gained from earlier layers.

Table 6: Average Time-steps to Find Optimal Spin Glass Bit String

Test Plan	Time-steps	STDV	Z-Score
Plan0	317,719.69	295,666.7228	-
Plan1	352,040.5	318,712.1596	-0.79
Plan2	333,524.35	323,185.469	-0.36

Table 7: Average Spin Glass Performance with Fixed Layer Duration

Test Plan	SpinGlass Perf.	STDV	Z-Score
Plan0	0.9952	0.0072	-
Plan1	0.8209	0.0078	163.71
Plan2	0.9891	0.0081	5.61

Table 8: Average Subproblem Performances per Layer's Completion

Plan #	Subproblem	Layer 1	Layer 2	Layer 3
Plan0	SpinGlass	0.9952	-	-
Plan1	Middle OneThird SpinGlass	1	0.9843	0.9829
Plan1	Middle TwoThird SpinGlass	0.751	1	0.9933
Plan1	SpinGlass	0.6614	0.8209	0.9891
Plan2	FirstThird SpinGlass	1	0.981	0.9824
Plan2	FirstTwoThird SpinGlass	0.7498	0.9995	0.9936
Plan2	SpinGlass	0.6633	0.8202	0.9891

These experiments demonstrate that the stability-plasticity imbalance created by layered learning application highly non-linear problems may be nuanced and complex.

5. Conclusion

From these experiments, we have demonstrated that the loss of beneficial knowledge can take place in a DBRL system using a simple (1+1) EA on both linear and challenging non-linear problems. Because each test case for both of the tested Boolean-logic experiments always resulted in a decrease in the first optimized subproblem, we conclude that this system favors plasticity, causing an imbalance. Because of this imbalance, two main negative effects were experienced. First, it took longer for the decomposition-based learner to reach optimality compared to our control, the monolithic learner. The reason for the required extended learning duration is because the learner had to spend effort in the final layer to regain beneficial knowledge that was previously lost. The second negative effect was that overall performance suffered. The decomposition-based approach underperformed, averaging problem performances significantly below the monolithic learner.

Both of the negative effects are attributed to the premature removal of beneficial knowledge (in this case, bits), forgotten to optimize on newer subproblems. The attribution was uncovered by the realization that the first subproblem's performance decreased in the second layer for each test cases with three layers. Therefore, we believe it is the DBRL's transitions to new subproblems that is the main culprit for performance loss. In addition, the greedy behavior of the (1+1) EA and the problem decomposition both have played roles in driving out critical performance knowledge prematurely.

The implications of DBRL systems not heeding to this analysis' warnings can lead to the severe performance degradation experienced with neural networks from catastrophic forgetting. We have demonstrated using two simple, non-contrived problems that performance is affected with the premature removal of beneficial knowledge; a more complex system may suffer even more performance sub-optimality if stability and plasticity are not balanced or mitigation methods are not developed and deployed.

6. Acknowledgment

We thank the Florida Education Fund, the McKnight Fellowship, and the Advanced Research Computing Center (ARCC) in the Institute for Simulation and Training at the University of Central Florida. Access to STOKES HPCC has been instrumental to the research performed in this work.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems 10*. MIT Press, 1998, pp. 1043–1049.

[3] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning," in *In Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann, 1998, pp. 118–126.

[4] P. Stone and M. Veloso, "Layered learning," in *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, R. L. de Mántaras and E. Plaza, Eds. Barcelona,Catalonia,Spain: Springer Verlag, May/June 2000, pp. 369–381.

[5] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," 1994.

[6] Y. Jin and B. Sendhoff, "Alleviating catastrophic forgetting via multi-objective learning," in *International Joint Conference on Neural Networks*. IEEE Press, 2006, pp. 6367–6374.

[7] F. M. Richardson and M. S. Thomas, "Critical periods and catastrophic interference effects in the development of self-organizing feature maps," *Developmental Science*, vol. 11, no. 3, pp. 371–389, 2008. [Online]. Available: http://dx.doi.org/10.1111/j.1467-7687.2008.00682.x

[8] M. McCloskey and N. Cohen, "Catastrophic interference in connectionist networks: The sequential learning probleml," *The psychology of learning and motivation*, vol. 24, pp. 109–165, 1989.

[9] S. Mondesire and R. P. Wiegand, "Evolving a non-playable character team with layered learning," in *IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MDCM)*, 2011, pp. 52–59.

[10] ——, "Forgetting classification and measurement for decomposition-based reinforcement learning," in *Proceedings of The 15th International Conference on Artificial Intelligence (ICAI'13)*, 2013.

[11] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, pp. 123–146, 1995.

[12] A. Robins and S. Mccallum, "Catastrophic forgetting and the pseudorehearsal solution in hopfield type networks," *Connection Science*, vol. 10, pp. 121–135, 1998.

[13] M. Hattori, "Avoiding catastrophic forgetting by a dual-network memory model using a chaotic neural network," 2009.

[14] R. M. French, "Pseudo-recurrent connectionist networks: An approach to the "sensitivity-stability" dilemma," *Connection Science*, vol. 9, pp. 353–379, 1997.

[15] P. Stone and M. Veloso, "A layered approach to learning client behaviors in the robocup soccer server," *Applied Artificial Intelligence*, vol. 12, pp. 165–188, 1998. [Online]. Available: http://nn.cs.utexas.edu/pub-view.php?PubID=126580

[16] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone, "Evolving keepaway soccer players through task decomposition," *Machine Learning*, vol. 59, no. 1, pp. 5–30, May 2005.

[17] A. Cherubini, F. Giannone, and L. Iocchi, "Robocup 2007: Robot soccer world cup xi," U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Layered Learning for a Soccer Legged Robot Helped with a 3D Simulator, pp. 385–392.

[18] P. Fidelman and P. Stone, "Layered learning on a physical robot," 2005.

[19] P. A. Borisovsky and A. V. Eremeev, "A study on performance of the (1+1)-evolutionary algorithm," in *FOUNDATIONS OF GENETIC ALGORITHMS, 7*. Morgan Kaufmann, 2003, pp. 271–287.

[20] I. Wegener and C. Witt, "On the behavior of the (1+1) evolutionary algorithm on quadratic pseudo-boolean functions," 2000.

[21] M. Pelikan, M. Pelikan, D. E. Goldberg, and D. E. Goldberg, "Hierarchical boa solves ising spin glasses and maxsat," in *In Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2003), number 2724 in LNCS*. Springer, 2003, pp. 1271–1282.