

Electronic Theses and Dissertations, 2004-2019

2014

Complementary Layered Learning

Sean Mondesire
University of Central Florida

 Part of the [Computer Sciences Commons](#), and the [Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Mondesire, Sean, "Complementary Layered Learning" (2014). *Electronic Theses and Dissertations, 2004-2019*. 3039.
<https://stars.library.ucf.edu/etd/3039>

COMPLEMENTARY LAYERED LEARNING

by

SEAN C. MONDESIRE

B.S. Computer Science, University of Central Florida, 2004

M.S. Computer Science, University of Central Florida, 2006

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2014

Major Professor: R. Paul Wiegand

© 2014 Sean C. Mondesire

ABSTRACT

Layered learning is a machine learning paradigm used to develop autonomous robotic-based agents by decomposing a complex task into simpler subtasks and learns each sequentially. Although the paradigm continues to have success in multiple domains, performance can be unexpectedly unsatisfactory. Using Boolean-logic problems and autonomous agent navigation, we show poor performance is due to the learner forgetting how to perform earlier learned subtasks too quickly (favoring plasticity) or having difficulty learning new things (favoring stability). We demonstrate that this imbalance can hinder learning so that task performance is no better than that of a sub-optimal learning technique, monolithic learning, which does not use decomposition. Through the resulting analyses, we have identified factors that can lead to imbalance and their negative effects, providing a deeper understanding of stability and plasticity in decomposition-based approaches, such as layered learning.

To combat the negative effects of the imbalance, a complementary learning system is applied to layered learning. The new technique augments the original learning approach with dual storage region policies to preserve useful information from being removed from an agent’s policy prematurely. Through multi-agent experiments, a 28% task performance increase is obtained with the proposed augmentations over the original technique.

To my wife, Helen. Thank you for your patience, love, and support.

ACKNOWLEDGMENTS

I would like to thank the Florida Education Fund and the McKnight Fellowship for their support and encouragement. Additionally, I thank and appreciate the high performance computing capabilities I received from the Advanced Research Computing Center (ARCC) in the Institute for Simulation and Training at the University of Central Florida. Access to STOKES has been instrumental to the research performed in this dissertation.

TABLE OF CONTENTS

LIST OF FIGURES	xii
LIST OF TABLES	xiv
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: BACKGROUND	6
Terminology	6
Layered Learning	8
Difficulties with Monolithic Learning	12
The State of Layered Learning	14
Feasibility Experiments of Layered Learning	14
Comparing Layered Learning with Monolithic Learning	16
Layered Learning vs. Automatically Defined Functions	18
Layered Learning vs. Coevolution	20
Physical Robot Development with Layered Learning	23
Discussion on Layered Learning	26

Advantages of Layered Learning	26
Challenges of Layered Learning	28
Layer Isolation	29
Layer Overlapping	32
Other Decomposition-Based Methods	35
Transfer Learning and Robot Shaping	36
Hierarchical Decomposition Learning	37
Hierarchical Reinforcement Learning	38
Compositional Q-Learning	40
Q-Nets	40
Q-Cut	41
Feudal Reinforcement Learning	42
Hierarchical Abstract Machines (HAMS)	43
MAXQ	43
Comparisons to Layered Learning	44
Benefits and Challenges of Decomposition Learning	47
Forgetting and Stability-Plasticity Imbalance	49

Types of Forgetting	49
Instances of Negative Forgetting	51
Stability-Plasticity Dilemma	52
Catastrophic Forgetting	53
Methods of Balancing Stability and Plasticity	54
Incremental Learning	54
Rehearsal	55
Complementary Learning Systems	57
Deletion Strategies	60
Random Deletion	63
Temporal-Spatial Deletion	64
Utility-Based Deletion	65
Preliminary Work	68
Approach	69
Results	70
CHAPTER 3: DEMONSTRATION OF IMBALANCE IN LAYERED LEARNING	72
Metrics for the Identification and Classification of Forgetting	72

Performance-Based Metrics	72
Forgetting Metrics	76
Policy-Based Metrics	78
Imbalance Demonstration	80
Boolean-Logic Problems	80
Leading Ones Trailing Zeros	81
Experiment Description	83
LO+TZ Results	84
LO*TZ Results	89
Discussion	94
UAV Surveying Problem	98
Layered Learning Configuration	100
Policy Representation	102
The Learning Algorithms	104
(1+1) EA	105
Q-Learning	106
Experiment Description and Results	107

The Effects of Stability-Plasticity Imbalance in Layered Learning	108
Negative Forgetting	108
Local Optima	114
Layered Learning vs. Monolithic Learning	116
Conclusion	118
CHAPTER 4: STABILITY-PLASTICITY BALANCING TECHNIQUES	120
Existing Balancing Techniques	121
Subtask Rehearsal	121
Direct Policy Search Performance Evaluation	122
Value Search Performance Evaluation	124
Concurrent Layered Learning	126
Direct Policy Search Performance Evaluation	127
Value Search Performance Evaluation	129
Direct Policy Search vs. Value Search	130
Complementary Layered Learning	132
Performance Comparison to Standard Layered Learning	135
Direct Policy Search Performance Evaluation	136

Value Search Performance Evaluation	139
Complementary Layered Learning Applied to Balancing Techniques	141
Direct Policy Performance	141
Value Search Performance	145
Discussion	147
CHAPTER 5: CONCLUSION	149
Contributions	150
First Study of Imbalance in Layered Learning	151
Improvement in Layered Learning Effectiveness	151
Literature Review on the State of Layered Learning	151
Introduction of New Forgetting and Policy Change Metrics	152
Overlapping Feasibility Demonstrated	152
Imbalance Awareness to Other Decomposition-Based Systems	153
Future Work	153
LIST OF REFERENCES	156

LIST OF FIGURES

Figure 2.1: A deep task decomposition with a hierarchy of abstraction.	46
Figure 2.2: A flat task decomposition with subtasks that are progressively more complex versions of the overall task.	46
Figure 2.3: Average Aggregate Fitness for Monolithic and Layered Learning Approaches [39]	71
Figure 3.1: LO subtask performance while learning LO*TZ with a fixed layer duration. . .	90
Figure 3.2: TZ subtask performance while learning LO*TZ with a fixed layer duration. . .	91
Figure 3.3: Subtask performance while learning Middle Third LO*TZ with a fixed layer duration.	91
Figure 3.4: LO*TZ performance with each decomposition test plan under a fixed layer duration.	92
Figure 3.5: Visualization of the UAV Surveying Problem at 10th the scale of the environ- ment	99
Figure 3.6: (1+1) EA Average Subtask Performance of all Runs	113
Figure 3.7: Average UAV Surveying Task Performance Using (1+1) EA. Vertical bars at epochs 10,000 and 20,000 represent the layer transitions.	117
Figure 4.1: (1+1) EA Average Subtask Performance with CLL of All Runs	138

Figure 4.2: (1+1) EA Refueling Subtask Performance	145
--	-----

LIST OF TABLES

Table 3.1: LOTZ Task and Subtask Descriptions	84
Table 3.2: LO+TZ Test Plans with Subtask Sequences	85
Table 3.3: Average Time-Steps to Find Optimal LO+TZ Bit String	85
Table 3.4: Average LO+TZ Performance with Fixed Layer Duration	87
Table 3.5: LO+TZ: Average Subtask Performances per Layer's Completion	88
Table 3.6: LO*TZ Test Plans with Subtask Sequences	89
Table 3.7: Average Time-Steps to Find Optimal LO*TZ Bit String	90
Table 3.8: Average LO*TZ Performance with Fixed Layer Duration	92
Table 3.9: Transfer learning metrics comparing each test plan to the monolithic control .	95
Table 3.10: Forgetting Metrics for Fixed Time Duration Test Case	97
Table 3.11: List of UAV Sub-States	103
Table 3.12: List of Actions Each UAV Can Perform.	104
Table 3.13: Q-learning Refueling Immediate Reward Criteria	106
Table 3.14: Q-learning Take Off & Survey Immediate Reward Criteria	107
Table 3.15: (1+1) EA: Average Policy Set Composition Changes Per Layer Completion .	111
Table 3.16: Q-learning: Average Policy Set Composition Changes Per Layer Completion	111

Table 3.17:(1+1) EA: Average Subtask Performance after each Layer Completion	112
Table 3.18:Q-learning: Average Subtask Performance after each Layer Completion . . .	112
Table 3.19:Average UAV Surveying Task Performance at the Completion of Each Layer .	117
Table 3.20:Standard deviations, Z-scores, and P-values ($\alpha = 0.05$) of the final average UAV surveying task performance scores.	118
Table 4.1: (1+1) EA: UAV Survey Task Performance and Multi-Way Z-Test ($\alpha = 0.025$)	122
Table 4.2: (1+1) EA: Subtask Performance Per Layer for Subtask Rehearsal	123
Table 4.3: (1+1) EA: Average Forgetting Scores	124
Table 4.4: (1+1) EA: Standard Deviations of the Forgetting Scores	124
Table 4.5: Q-learning: UAV Survey Task Performance and Multi-Way Z-Test ($\alpha = 0.025$)	125
Table 4.6: Q-learning: Subtask Performance Per Layer for Subtask Rehearsal	125
Table 4.7: (1+1) EA: Subtask Performance Per Layer for ConcLL	128
Table 4.8: Q-learning: Subtask Performance Per Layer for ConcLL	130
Table 4.9: (1+1) EA: Average UAV Survey Task Performance and Z-Test Results ($\alpha =$ 0.017)	135
Table 4.10:Q-Learning: Average UAV Survey Task Performance and Z-Test Results (α = 0.017)	136
Table 4.11:(1+1) EA: Average CLL Subtask Performance Per Layer Completion	137

Table 4.12:(1+1) EA: Standard Deviation of CLL Subtask Performance Per Layer Completion	137
Table 4.13:Q-learning: Average CLL Subtask Performance Per Layer Completion	140
Table 4.14:(Q-learning: Standard Deviation for CLL Subtask Performance Per Layer Completion	140
Table 4.15:(1+1) EA: Average CLL-Reh Subtask Performance Per Layer Completion . .	142
Table 4.16:(1+1) EA: Standard Deviation of CLL-Reh Subtask Performance Per Layer Completion	143
Table 4.17:(1+1) EA: Average CLL-Conc Subtask Performance Per Layer Completion .	143
Table 4.18:(1+1) EA: Standard Deviation of CLL-Conc Subtask Performance Per Layer Completion	143
Table 4.19:(1+1) EA: Forgetting Metrics Values for the Complementary Systems and Standard Layered Learning	144
Table 4.20:Q-learning: Average CLL-Reh Subtask Performance Per Layer Completion .	146
Table 4.21:Q-learning: Standard Deviation of CLL-Reh Subtask Performance Per Layer Completion	146
Table 4.22:Q-learning: Average CLL-Conc Subtask Performance Per Layer Completion .	147
Table 4.23:Q-learning: Standard Deviation of CLL-Conc Subtask Performance Per Layer Completion	147

CHAPTER 1: INTRODUCTION

The goal of this work is to improve the decomposition-based machine learning paradigm of *layered learning* by addressing its susceptibility to prematurely forgetting important knowledge. This challenge is known to cause learning agents to quickly and catastrophically lose the ability to perform vital subtasks and result in sub-optimal task performance, challenging the effectiveness of the learning system. The goal of this dissertation is achieved by satisfying three objectives: 1) demonstrate that layered learning’s performance can be significantly hindered by a stability-plasticity imbalance and identify the causes; 2) propose a new technique that augments layered learning with a complementary learning system to mitigate the negative effects caused by the imbalance; and 3) compare the proposed learning system’s performance to that of existing mitigation techniques. This new research contributes a warning to layered learning practitioners of an unexplored pitfall that can drastically hurt task performance and provides a tested solution to the anticipated problem. Additionally, the presented work’s analysis and solution opens the possibility of using layered learning with layer overlapping, a less restrictive form of layered learning that is unpopular because of its susceptibility to forgetting. Finally, the study of a stability-plasticity imbalance in the abstract learning paradigm can provide insight of how similar learning techniques may suffer from the imbalance and propose solutions to their problems.

Layered Learning is a machine learning technique that uses task-decomposition and reinforcement learning to teach agents how to perform a complex task. The technique decomposes a task into simpler subtasks and trains the agent to reach a desired level of performance on each component. Each subtask is learned in a layer that provides a training environment and scenario where the agent can optimize it’s policy on specific aspects of the task. With each passing layer, the agent’s proficiency of the task has the chance to increase because each layer optimizes performance on a new component of the task and builds on top of what was learned previously. Ideally, upon the

completion of the final layer, the agent has a policy capable of performing the complex task. The aim behind using this task-decomposition machine learning technique is to allow the learning agent to progressively acquire the ability to perform all of the necessary components to solve a complex task.

Layered learning is not a machine learning algorithm like evolutionary computation or Q-learning; instead, it is an abstract architecture and process that guides the agent to optimize performance of a task through task decomposition. It acts as a high-level paradigm that resides on top of other machine learning methods and can employ a variety of reinforcement learning techniques for actual learning.

Layered Learning is being used in an array of different domains, in particular, to develop the decision making policies of agents as they learn how to perform complex tasks autonomously. To date, layered learning has taught simulated and physical robots and autonomous computer-based agents how to perform motor skills, such as walking, running, and grabbing, play soccer, behave as challenging non-playable characters in videos games, solve Boolean-logic problems, and train a team of cooperating unmanned aerial vehicles (UAVs) to navigate through an environment. In each case, the process of training agents on decomposed tasks has led to a degree of success in agent policy development.

The learning paradigm is attractive from an empirical point of view: many implementations of layered learning have proven successful at improving task performance compared to standard, monolithic learning; for this reason, recently, the paradigm has become increasingly used by a subset of the robotics community to develop the behavioral policies of autonomous agents needed to solve complex tasks.

Indeed, layered learning robots are currently being deployed today, and developers and researchers are relying on the success of these agents; however, little attention has been spent trying to gain

a deeper understanding of the strengths and weaknesses of the approach. Moreover, as we will discuss, layered learning shares much in common with other decomposition-based approaches. The aim of this dissertation is to improve our understanding of the paradigm, as well as to address some obvious potential pitfalls so that those who are already using the approach can produce more effective agents or, in the worst case, when they should consider other approaches altogether. We do this by first looking at similar challenges and solutions from related machine learning methods.

From the literature review of similar decomposition-based approaches and the preliminary work, it is clear that the best way to improve layered learning is to study and address a clear and predictable challenge faced by the paradigm: its susceptibility to a stability-plasticity imbalance. A potential stability-plasticity imbalance is a problem all decomposition-based reinforcement learning approaches face. The *stability-plasticity dilemma* requires the learner to determine when to have a preference for older, acquired knowledge (*stability*) and when to favor new, incoming knowledge (*plasticity*). A major aim of this work is to provide the machine learning and robotics community with an understanding of the imbalance in layered learning and attempt to mitigate the effects of its occurrence.

The stability-plasticity imbalance is being targeted because it can clearly hinder the performance of layered learning agents, forcing the developers to design agent learning plans in the unnatural manner of layer isolation, accept inaccurate agent decision making, or waste additional computational effort in attempting to regain vital information that has been forgotten in the learning process. In autonomous robot development, these trade-offs can be extremely costly in terms of agent success and performance evaluation due to limitations in development time, robot battery life, and policy storage capacity.

Reducing or removing the occurrence of an imbalance and mitigating its negative effects will make layered learning more effective at learning how to solve complex tasks and ultimately improve the

paradigm’s standing among other learning techniques. Considering that layered learning has done well even with the imbalance flaw suggests that even greater levels of success can be achieved if its challenges are addressed. Enhancements to the paradigm will result in further success with agents who employ layered learning and possibly outperform techniques considered to be the current state of the art.

Additional motivation for the proposed work is that beyond changing a specific paradigm, solutions to the imbalance in layered learning also have the potential to mitigate the ill effects in other machine learning techniques. For example, techniques that use transfer learning, robot shaping, task decomposition, and hierarchical structures may be able to support the output from the proposed investigation to counter negative forgetting. The potential for a discovered layered learning solution to fit these other machine learning areas is because of layered learning’s foundation is based on these other techniques and they naturally share many fundamental similarities.

Finally, there are no existing published studies that investigate the effect of a stability-plasticity imbalance on layered learning. To date, the layered learning literature focuses on comparing the paradigm to other machine learning techniques, notably monolithic genetic algorithms and coevolution, and evaluating the effectiveness of employing the paradigm on real and simulated robots in multi-agent environments. The proposed work will serve as the first study of the deeper challenges of layered learning, the imbalance in particular. Specifically, we focus our analysis on challenges in direct policy search using layered learning but also provide insight on stability-plasticity imbalance in value search reinforcement learning. Lastly, this work also contributes the first suggested changes to the paradigm to address such problems. Listed below are the major contributions of this work:

- Provides the first empirical study of stability-plasticity imbalance in layered learning.
- Introduces a set of metrics for diagnosing and measuring the act of forgetting caused by

imbalance.

- Demonstrates that an imbalance exists and can significantly hinder learning's effectiveness.
- Produces a detailed analysis of new and existing simple solutions that mitigate the imbalance.

To accomplish the goal of improving layered learning by addressing the stability-plasticity dilemma, this dissertation provides a literature review of the paradigm and stability-plasticity imbalance in similar decomposition-based learning methods, demonstrates that an imbalance can exist in a layered learning system and identifies the causes and effects, examines the application of layered learning to different reinforcement learning techniques, including the a direct policy search with an evolutionary algorithm and value search with Q-learning, defines the proposed solution that augments layered learning with a complementary learning system, and analyzes the new system's performance to that of existing layered learning techniques, using experiments that train multiple learning agents to perform an unmanned aerial vehicle (UAV) surveying problem.

The paper is outlined as follows: chapter 2 provides the literature review of layered learning, stability-plasticity imbalance in other systems, and preliminary work; chapter 3 demonstrates the imbalance and identifies causes and effects; chapter 4 analyzes existing and proposed solutions; and chapter 5 provides the conclusion and description of future work.

CHAPTER 2: BACKGROUND

With the ability to make decisions, agents can display specific behaviors, achieve goals, and perform tasks when confronted with implementation choices. The work discussed here focuses on enhancing the decision making ability of autonomous agents by understanding a specific set of challenges facing an existing, well-used learning method and improving it based on this understanding. More specifically, the work addresses the retention of information used in making decisions to accomplish a complex task. This chapter provides the background that gives context to this work and problem. Here, we define common terminology that will be used throughout this work, define layered learning, discuss the state-of-the-art uses of the paradigm, compare layered learning to other reinforcement learning approaches, describe the stability-plasticity dilemma faced by other learning systems, and finally review preliminary work that has created the suspicion that layered learning may suffer from a performance hindering phenomenon caused by stability-plasticity imbalance.

Terminology

A *task* is a behavior or objective to be satisfied by an agent. Every task has a set of sequences of actions that must be executed to perform the task. The set of sequences is a subset of all possible actions available to an agent and may vary with the environment's state. *Subtasks* are sub-components of a task that can be used to solve a portion of the task. These subtasks can be easier to learn to perform than a task by requiring fewer actions to perform for subtask optimality or have less difficulty in generating the sequence of actions to perform for subtask optimality

Decisions are made by the agent to select which actions to perform and when to execute them to

carry out a task [57]. *Reinforcement learning (RL)* is an area of machine learning that searches for decision-making solutions that maximize a cumulative reward [57]. From the RL point of view, decisions are based on policies. A *policy* is a set of mappings that link states to actions; they represent the conditions when certain actions should be executed. When an agent is faced with a decision, it uses its policy to determine which action to perform based on the current state. The purpose of the RL technique is to provide the learning agent with a policy capable of performing the desired task well, according to the performance measure. In RL with *direct policy search*, the learner searches through a solution space of possible policies to find one that meets the target criteria, typically to maximize some cumulative reward over actions taken to accomplish the task [41]. *Value search* is another type of RL where utility values are used to make estimates on the rewards a policy would receive. Monte Carlo [27] and Q-learning [58] approaches are examples of value search methods. In this work, we are primarily focusing on direct policy search but also look at value search with layered learning.

$S = \{s_0, s_1, \dots, s_n\}$, where S is the set of environmental states and s_i is an individual state.

$A = \{a_0, a_1, \dots, a_n\}$, where A is the set of all potential actions.

$P: S \rightarrow A$, where P is a policy mapping states to actions. As such, a policy can also be seen as a collection of state-action pairs, one for each distinct state. In this work, a policy is a list of state-action pairs (later referred to simply as *pairs*) that together are used to solve a task.

Task decomposition breaks down a task into subtasks when the task is too complex to learn at once. The process of decomposing a task is a very general idea and there are many decomposition-based approaches in machine learning. In this work, task decomposition uses subtasks to represent fragments of the task in a whole-part manner. In this process, the subtasks are the stepping stones required for the completion of the overall task. With the use of task decomposition, each subtask is solved with a subset of the state-action pairs in the policy. For the rest of this dissertation, we

will refer to tasks as the overall problem the agent is trying to solve.

It is reasonable to consider task decomposition when random search in a large solution space takes an unacceptable amount of time. Two advantages stand out for using task decomposition in machine learning. The first advantage is that the agent can focus on one subtask at a time and hopefully navigate through the reduced solution space with less effort. The second advantage is that agent developers can order the subtasks in a way to guide how a task is learned. Without the task being decomposed, the agent may be overwhelmed to learn every aspect of the task at once and learn sub-optimally. With task decomposition, the agent is able to focus on learning parts of the task separately and can be easily guided with a learning plan on how to learn aspects of the task.

An example of a complex task being decomposed into sub-components is teaching autonomous robots how to play soccer. To decompose the complex game of soccer and all of its rules, the task may be broken up into subtasks to learn how to approach, intercept, pass, and shoot the ball. In this example, each subtask is simpler than the complex task, has a reduced solution space than the overall task, and is necessary to learn how to play the game of soccer.

Methods in RL that employ decomposition include feudal reinforcement learning [6], hierarchical abstract machines [44], MAXQ [7], and layered learning [56].

Layered Learning

In machine learning, a number of different approaches have been referred to as “layered learning,” including weight tuning in multi-layered perception neural networks [25, 48, 15] and in incremental learning systems [51, 11, 35]. In this dissertation, the focus is placed on the specific learning paradigm layered learning that was introduced by Stone and Veloso [56]. Stone and Veloso introduced layered learning as a machine learning paradigm that relies on task decomposition to

simplify a complex task into manageable components. At a high level, layered learning is given a complex task decomposition and optimizes the agent’s policy to perform each subtask sequentially. Once all of the subtasks are learned, the agent is expected to be able to perform the complex task.

Each subtask represents a component of the complex task the agent must learn to perform in order to complete the complex task effectively. Typical uses of layered learning has the agent learn in a bottom-up fashion by first learning the simplest subtasks and progressing to more complex ones until the agent can perform the overall complex task. Other uses train the agent to preform independent subtasks sequentially or by progressively training subtasks that build on the knowledge gained by previously learned subtasks. Layered learning is defined in Equation 2.1.

$$L_i = (F_i, O_i, T_i, M_i, h_i) \quad (2.1)$$

Here, each subtask learning phase is a *layer* (L_i), where the agent is given a training environment and set of training examples designed for learning the subtask (T_i), set of all state variables for the agent to analyze its environment (F_i), a set of all possible actions the agent can make (O_i), and a machine learning technique to learn the subtask (M_i). The output of the layer is a policy (h_i) the agent has developed for solving the subtask. The outputted policy of a layer is used as the initial policy its subsequent layer modifies to solve its own subtask, creating a dependency of learning among the layers. Stone and Veloso [56] define Layered learning as:

Layered learning is deliberately abstract in its task decomposition, training set, machine learning algorithms, and performance evaluation criteria inputs to allow the paradigm to be suited to solve a variety of specific complex tasks; this abstraction is the main distinction between layered learning and other decomposition-based reinforcement learners. The paradigm does not decompose the task into simpler sub-components and relies on developers to decompose the complex task into a meaningful way. To be effective, this task decomposition input must guarantee that the learning

agent can grasp the complex task by learning to correctly perform each subtask and that each subtask layer reduces the solution space enough to make learning the complex task tractable. In addition, the paradigm must be given training sets for each layer where the agent can correctly learn how to perform the subtask. Furthermore, all of the given training environments and training sets for each layer must adequately represent the subtask.

Because layered learning is not a machine learning algorithm, the paradigm depends on other learning techniques to modify agent decision-making policies. The machine learning algorithm is responsible for modifying the agent’s policy to perform the current layer’s subtask and uses the provided task decomposition and training set to guide the learning process. Although common uses of layered learning employ the same learning algorithm for all of the layers, such as using genetic programming [20, 21, 18, 23] and neuro-evolution [55, 60] throughout the learning process, using different algorithms for individual layers has also proven to be effective. An example of a mixture of algorithms is in Stone and Veloso’s [56] work where three different machine learning algorithms, neuro-evolution, C4.5 decision tree training algorithm, and TPOT-RL, are used in training simulated soccer playing agents. The flexibility of employing different machine learning algorithms in each layer allows for an effective algorithm to be used for solving a specific class of problem represented in the subtask.

Similar to other task decomposition machine learning techniques, layered learning addresses the monolithic direct-policy issues described below by biasing the solution space. The externally provided bias is in the decomposition, where each subtask has its own solution space that is typically smaller than the complex task’s set of possible solutions. By sequentially focusing on a smaller subset of solutions with each subtask, the paradigm reduces the difficulty of locating acceptable solutions in a large solution space by searching for solutions in a sequence of smaller spaces. By significantly reducing the solution space for each subtask, the learner will hopefully be able to learn all of the components needed to solve the complex task in a more manageable way.

Layered learning has four principles for when and how the paradigm should be used [56]. First, layered learning works best when learning a direct mapping from inputs to outputs is not tractable. Because layered learning decomposes the complex task into sub-components, the paradigm is most effective when the task is large enough to be decomposed into simpler subtasks, such as mapping sensors to actuators. With proper task decomposition inputted into a layered learning system, the complex task is learned by incrementally learning sub-components of the task.

Second, the paradigm must be given a task decomposition that is bottom-up and hierarchical. Because the paradigm learns progressively more complex subtasks, it must be given an effective decomposition where the learner can incrementally learn the task with each experienced layer. Not all problems are amendable to such decomposition, as we shall see.

Third, an appropriate machine learning algorithm is used at each layer to learn the desired subtask. To learn a complex task, the layered learning agent must be given a learning algorithm suitable to solving the subtask for each layer. Indeed, one of the key advantages of such a general framework is the flexibility to work with many different underlying learning heuristics. With the proper learning algorithm, task decomposition, and inputs, the learner will be able to learn subtasks vital to learning the overall complex task.

Finally, one layer's output policy seeds the next layer. Layered learning progressively improves the agent's ability to learn the task. To realize the progression, each layer must accept what was learned in its preceding layer to enhance the policy to solve the current subtask and retain information learned previously. The seeding can be used to adapt the policy to perform a new, higher-order subtask or provide the new layer with learned components needed to perform the new subtask.

These four principles define when the paradigm should be used and which conditions need to be met to implement the paradigm correctly.

Difficulties with Monolithic Learning

Learning effective mappings of sensors to actuators with monolithic direct-policy search for complex tasks is difficult for reinforcement learning agents [56, 55, 20, 21, 10]. One cause of the difficulty is that the solution space of a complex task can be too large to navigate through efficiently and exhibit insufficient gradient information. The second cause is that the task can be too complex to find an effective solution in an acceptable amount of time. These two reasons can cause direct-policy search algorithms to be both inefficient and ineffective at solving complex tasks across multiple domains.

Monolithic RL inefficiency is a side-effect of amalgamating multiple states into one performance criterion to evaluate solutions and the breadth of possible solutions through which the learner must navigate through. In cases when the monolithic property combines multiple state attributes into one performance criterion, the potential number of combinations of states to actions can create an extremely large solution space to navigate through. If the agent learns by trial and error, such as with a reinforcement learning algorithm, large amounts of evaluations, time, and effort can be required to locate an acceptable solution. The entire process becomes inefficient due to the time needed to search the large solution space for acceptable solutions.

Luke *et al.*'s [32] work is an example of the solution space being too large to efficiently navigate through. In the work, simulated autonomous agents were tasked with learning how to play soccer by learning various soccer-related behaviors with the learning technique of *genetic programming* (GP) [29]. The approach of using GP to develop policies for complex tasks of ball interception and movement between two way-points proved to be unsuccessful and forced the research to use hand-coded policies to learn the behaviors. In this example, the complex tasks each possess a solution space that is too large to identify acceptable mappings in the allotted amount of time.

The second reason a monolithic learning system can have difficulty discovering an effective solution for a complex task is the potential scarcity of effective solutions. This rarity can be due to the possibility of the added conditions the monolithic evaluation criterion places on the solution space to narrow what are acceptable solutions. A second cause of rarity can be due to the size of the solution space and the difficulty in finding acceptable solutions when there are so many possible candidates. In other words, a monolithic approach imposes additional conditions that must be met in order to deem a solution acceptable.

To illustrate this second reason, let's consider two instances of this problem. First, consider when the solution space is large but there are only a handful of acceptable solutions. Probabilistically, the learner may never reach one of the acceptable solutions because only a few solutions in the solution space meet the acceptable criteria for the complex task. Second, consider a small solution space with one acceptable solution. The learner may fixate on similar solutions in the search space and never reach the one dissimilar acceptable solution. Both of these examples focus on the possibility of the learner lacking novelty exploration, fixating on a local optima, or getting stuck on a performance plateau during its navigation of the solution space. These two obstacles are caused by the learner having to navigate for solutions with more conditions to satisfy in one performance measure and in a solution space with fewer solutions.

Two techniques for making monolithic RL for complex tasks tractable are to: 1) constrain or bias the search space, and 2) bias the search method itself. By constraining the solution space, only a subset of the complex task's true solution space is considered in the search. The reduction in size allows the learning algorithm to only focus on solutions considered important for the task. Biasing the search is navigating through the solution space with guidance, typically with the use of external domain knowledge. Two approaches include the use of a specialized representation that constrains the problem, such as GP with structured operators and functions, and to use the decomposition of the problem space into sub-components to learn separately, such as co-evolution and hierarchical

reinforcement learning.

Layered learning is a machine learning paradigm that possesses the ability to constrain both the solution and problem space. The paradigm is designed to ease the difficulty monolithic direct-policy search has with solving complex tasks. The paradigm relies on both task decomposition and bottom-up problem solving to reduce the solution space and incrementally learn complex tasks. Studies have shown that layered learning can improve autonomous agent learning when compared to standard monolithic approaches across several different domains. Success with the paradigm has been witnessed in such complex tasks as training real and simulated robots how to walk and play soccer, solve Boolean logic problems, and develop non-playable characters in video games to name a few.

Next, we survey the current state of the art in layered learning, followed by a discussion of the advantages and disadvantages of the paradigm and a look at similar learning approaches.

The State of Layered Learning

Our survey of layered learning research is organized based on the way in which empirical comparisons are examined in literature: feasibility studies, comparisons to monolithic approaches, comparisons to other modular/composition-oriented approaches, and application to challenging, realistic problem domains.

Feasibility Experiments of Layered Learning

Layered learning is introduced by Stone and Veloso [55] to train simulated agents how to play the complex game of soccer. The approach decomposes the complex task into two layers: to first learn the low-level individual skill of intercepting a moving ball and then learn a higher-level social skill

of passing to a teammate. The levels of subtask complexity were defined by the authors. This task decomposition is noteworthy because the low-level subtask is a vital component of the higher-level subtask and the higher-level subtask is complex because of its multi-agent nature, requiring coordination among teammates.

All agent decision making is controlled by a neural network that determines how the agent intercepts and passes the ball among teammates. As the agents reside in a layer, these neural networks are modified to adapt to the training scenario designed for each subtask.

To determine the effectiveness of layered learning, two experiments are made for the high-level passing task. The first experiment demonstrates that a layered learning team is more successful than a team that randomly selects to which teammate to pass the ball. This random passing foil is intended to baseline the experiments.

The second experiment demonstrates that a team learned via a layered learning approach is also more successful than a static approach where an agent with the ball always passes to the closest teammate, though this performance difference is small. This second comparison is meant to compare the results of layered learning to a known good, existing heuristic.

Stone and Veloso [56] extend their introduction of layered learning by formally defining layered learning and jointly using multiple machine learning techniques to solve the same complex task. To accomplish the use of multiple techniques, the work adds a third subtask and layer of scoring goals to the complex task's decomposition of soccer and uses three different machine learning algorithms, one for each layer, to effectively learn each subtask. Experiments compare this new take on layered learning to both agents who make random decisions and against real RoboCup competitors.

In the first layer, the individual agent learns how to intercept the ball by modifying a neural net-

work. The second layer is the complex task learned in the introductory work [55] of learning to evaluate passing options and making the pass. This second subtask is multi-agent based, requiring coordination among teammates, and the layer uses the C4.5 decision tree training algorithm. The final layer uses the subtasks learned in the first 2 layers to master scoring goals. In this final layer, the team tries to retain ball possession until it has a clear goal scoring opportunity, when at which a team member will shoot at goal. This pass-or-shoot team layer uses the TPOT-RL learning algorithm to learn the highest-level subtask.

These results provide some evidence of layered learning's potential utility for complex tasks. To further evaluate layered learning as a valid learning paradigm, the approach was used in the RoboCup simulation competition for autonomous soccer playing agent teams. The layered learning paradigm has been successful in these competitions, making it to semifinals of RoboCup-97 out of 29 teams, being RoboCup-98 champions in a field of 34 teams, and RoboCup-99 champions in a field of 37. These RoboCup results show layered learning can thrive in competitive environments making accurate, real-time decisions.

Comparing Layered Learning with Monolithic Learning

Making a direct comparison between layered learning and monolithic learning is a natural and common practice during the lifespan of the learning paradigm. The comparison allows the learning paradigm to have a direct measure against the very technique that serves as motivation for layered learning.

The works performed by Hsu and Gustafson [20, 21] and Gustafson and Hsu [18] serve as some of the earliest analysis of how well layered learning performs against a monolithic learning technique in producing complex task solving agents. Each of these works compare layered learning with genetic programming (LLGP) to monolithic standard genetic programming (SGP) in producing

keepers in the RoboCup sub-domain challenge of Keep-away. In short, Keep-away is a variant of the game of soccer, where instead of trying to score more goals than your opponent, the goal is to maintain ball possession if you are a *keeper* and take possession if you are a *taker*. Keep-away provides an excellent multi-agent behavior test-bed because, similarly to the RoboCup domain, agents must interpret noisy environment variables, make real-time decisions, and coordinate among teammates, to name a few features.

Hsu and Gustafson [20, 21] decomposed Keep-away into two subtasks: 1) first learn an intermediate skill of passing accurately to a teammate without a defender present and then 2) learn how to move and pass with a defender present. While making the comparison among layered learning and SGP, their work discovered that LLGP performs best when entire population of the previous layer (LLGP-all) is copied to seed the next layer instead of duplicating only the best individual (LLGP-best). The work also showed that early policy saturation in the first layer should lead to faster transition to the next layer because more time spent training in the higher-level subtask layer produced a more effective agent team. Their results show that the task decomposition method helps boost agent-learning’s effectiveness and even more so when the subtask more closely related to the complex task has more training time.

Gustafson and Hsu [18] extended their work by further examining whether the best individual (policy) should be duplicated to seed the next layer’s population or the entire population should be copied. They analyzed the effects of altering the duration the agent trained in each layer. Results again showed that copying the entire population and decreasing the time spent in the first layer further enhanced the agent team’s ability to perform the complex task of maintaining ball possession in Keep-away.

Jackson and Gibbons’s [23] work studies the effectiveness of layered learning on solving Boolean-logic problems and also compares LLGP’s ability against a monolithic GP learner. The Boolean

logic problems in these experiments are Even-4 Parity and Majority-On 5. Two main experiments are conducted to analyze layered learning by decomposing the task in 2 ways: 1) splitting training-sets-to-learn on among layers, and 2) using earlier layers to solve lower-ordered problems. In the first set of experiments, the training sets are split. The first layer is trained on half of the training set to solve the Boolean logic problems, and the entire training set is learned in the second layer. To measure LLGP's effectiveness, standard GP (SGP) is used for comparison. Results show that in instances where LLGP produces a higher success rate, the computation effort required to identify the solution is less than that of SGP. Results are similar for the Majority-On 5 problem.

In the second experiment, half of the population learns on half of the training set and the second half of the population learns the second half of the training set in layer 1. In layer 2, both population halves are combined and the entire training set is trained on the entire population. Test cases that divide the two sub-sets never produce a solution that bettered SGP for Even Parity 4, and in cases where LLGP matches performance with SGP, it requires more computation effort to do so.

In the final set of Boolean logic experiments, lower-order versions of the Boolean logic problems (Even-2 Parity, Even-3 Parity, and Majority-On 3) are learned in the first layer (each in separate runs) and then the complete problem is learned in the second layer. The results of these experiments show how successful LLGP is at accomplishing the two different Boolean logic programs in a different type of task decomposition where lower-ordered problems first seed the learning process. All together, the experiments highlight a key issue with decomposition-based approaches: the *way* in which problems are decomposed is a critical design choice.

Layered Learning vs. Automatically Defined Functions

Besides comparing layered learning to standard, monolithic genetic programming, the learning paradigm has been compared to other learning techniques to judge its overall effectiveness in

solving complex tasks. The use of the data type ADFs with genetic programming (ADFGP) is one of those techniques that have been used to evaluate layered learning's effectiveness. ADFs introduce a concept of modularity to GP, which makes it a natural comparison to approaches that rely on composition/decomposition, which can be seen as a kind of forced modularity.

In addition to comparing layered learning with standard, monolithic genetic programming in their work to produce Keep-away agents, Hsu and Gustafson [20, 21] also compare layered learning to ADFGP. In their earlier layered learning work, Hsu and Gustafson [20] created the LLGP-all approach which seeds a layer with the entire population from previous layer. The turnover averages for behaviors outputted from LLGP outperform the ADFGP outputs. On the other hand, ADFGP produces a better average team of keepers. It should be noted that when comparing the best fit individual in the final population at the end of all of the generations, the selective ADFGP produces a better keep-away team compared to LLGP. Although selecting the best 10 runs for the ADF approach produces a smaller turnover rate, the averages are comparable between this selective version and LLGP. The biggest factor to note is that LLGP does not require any additional analysis to determine which runs to consider in order to produce a more fit individual, as required by the "best ADFGP" run.

In the follow-up work, Hsu and Gustafson [21] conclude that LLGP has benefits over standard GP and hand-coded hierarchical techniques due to the ease latter layers have at solving their sub-tasks. With preceding layers solving subtasks needed by subsequent layers, these latter layers have easier solution spaces to navigate through, being an advantage compared to attempting to solve the complex task at once. Also, the performance of LLGP was very comparable, if not better, to the monolithic approaches. The conclusion is backed up by the experiment results of LLGP outperforming the other approaches in best of run and best individual after all of the generations in fitness.

In Jackson and Gibbons’s [23] work that compared layered learning with monolithic standard GP, as discussed earlier, the work shows that learning lower-ordered problems of complex tasks in earlier layers can increase effectiveness and decrease computation effort to reach the successful solutions. The work also compared layered learning with ADFGP. ADFGP outperformed both standard GP and LLGP in solving Even-4 Parity. However, when training sets were divided either among layers or populations, LLGP produced a best success rate. When LLGP first learns lower-ordered versions of the complex task, LLGP significantly outperforms ADFs.

Layered Learning vs. Coevolution

Up to this point, layered learning has been compared to learning techniques that use monolithic reinforcement learning to learn complex tasks. Next we analyze how layered learning performs against approaches that learn task components in parallel, using coevolution. Cooperative coevolution [45] is another kind of composition/decomposition-based learner, which makes comparisons to such particularly interesting.

Whiteson *et al.*’s [60] work expands layered learning comparison work by evaluating the paradigm against both a monolithic and a coevolution approach. In their work, the three approaches are tasked with producing keep-away *keeper* agents. In the experiments, each agent team shares among its teammates the same hand-coded decision tree that determine which neural network to activate, where each network controls a component responsible for a subtask of being a *keeper*. The subtasks are to intercept the ball, make a pass, evaluate passing options, and control agents getting open for a pass.

To implement layered learning, a layer is designed for each subtask and the subtasks are learned sequentially. In the coevolution implementation, all of the subtasks are learned simultaneously in the same, single training environment. The difference between the coevolution technique and the

standard monolithic approach is that in the monolithic approach, the agent attempts to learn the entire task in one learning sequence without any task decomposition. The coevolution approach in this study uses task decomposition by optimizing each network in parallel as the agent teammates learn how to play keep-away in the same, single training environment.

The first experiment evaluates the three approaches based on how well they learn the *keeper* task using incremental evolution. In this instance of incremental evolution, the *takers* start with movement that is 10% as fast as the *keepers*. The *taker* movement speed percentage is increased by 5% when the population's average fitness (# of completed passes) exceeds a threshold. The incremental increase continues until 100% speed is reached or when fitness plateaus. The increase in speed percentage represents an increase in difficulty in task that the *keeper* team must overcome to be successful. In other words, the keeper teams first learn how to play keep-away against an easy opponent. As the *keeper* team adapts to handle an opponent difficulty level, the task becomes more difficult to master.

The average task difficulty is analyzed in conjunction with learner performance on the keep-away task. The experiments show that coevolution outperforms the layered learning and standard monolithic approaches, reaching the highest (100% speed) taker difficulty. Layered learning and monolithic learning never reach the highest task difficulty.

The second experiment trains the three approaches on the hardest version of the task, *takers* at 100% speed. The average fitness score over the 150 generations are used to measure performance. Again, coevolution produced the best team of *takers* on average, having fitness of completed passes between 100-120, where layered learning plateaus around 30 and monolithic learning around 20.

Again, coevolution outperforms monolithic and layered learning in these keep-away experiments. Whiteson *et al.* [60] attribute coevolution's outperformance of layered learning because coevolution constrains the search space less than the layered approach, placing fewer restrictions on the

solutions to consider and increasing the range of viable possible solutions. Whiteson *et al.* also believe that coevolved subtasks can modify a policy’s behaviors in accordance with other subtasks; this holds true as the networks in this work are modified together, in parallel, where the networks can be adjusted to suit each other at the same time. Layered learning does not see this type of parallel behavior adjustment because of its sequential learning, which sees a network learned in a later layer adjust itself to another network learned earlier. In this case, the later network can adjust itself to the network learned earlier but the earlier learned network cannot adjust itself to suit the later network.

This work also observes that layered learning requires training environments and scenarios to be established to maximize the potential success for each layer. This requirement places additional responsibility on developers to not just get the task decomposition correct but also to design complete layers with training environments and scenarios that will give the agent a high probability of learning the subtasks.

Finally, Whiteson *et al.*’s work further highlights the difficulty monolithic learning has with directly mapping sensors to actuators for complex tasks. In all of the performed experiments, monolithic learning performed the worst of the three approaches in solving the complex task of keep-away. In each of the experiments, the approach of using decision trees and neural networks in the manner that they were used proved to be fruitless, and the complex task was too difficult to learn effectively.

Whiteson and Stone [61] address layered learning’s shortcomings against coevolution by introducing *concurrent layered learning (CLL)*. In CLL, the aim is to continue modifying already learned subtasks during subsequent layers to continuously improve task component effectiveness. To be more specific, CLL takes the best individual from L_i ’s population P_i to create a new population P_i' . P_i is changed to P_{i+1} and learns L_{i+1} . P_i' also trains on L_{i+1} . The two populations evolve in

parallel.

For performance evaluation, one individual is taken from each population, inserted into the same agent, and evaluated on its ability to perform L_{i+1} . Both individuals receive the same evaluation score. Layer isolation occurs here as the individuals represent components of the agent that drive different functions, such as subtasks.

Whiteson and Stone [61] use the same keep-away complex task and decomposition as in Whiteson et al's [60] and retrain the third layer of pass evaluation during the "get open" layer, in layer 4. Experimental results show that CLL outperforms traditional layered learning when keepers actively seek to intercept passed balls and when receivers are not permitted to move until the ball is kicked. CLL also outperforms a coevolution approach that coevolves the last 2 layers.

CLL is proven to be an improvement over standard layered learning and outperforms a coevolution technique that vastly excelled in the same task as the standard paradigm. By adding the ability to simultaneously modify aspects of a policy that controls two separate subtasks, CLL now has the coevolution advantage that the behavior learned in one layer can adapt to another behavior even when a new layer is being trained on. Also, CLL outperforms the coevolution approach because it effectively restricts the search space to where the best solutions are still present but the solution space is small enough to contain less ineffective solutions. In short, the layers reduce the search space to a more manageable size.

Physical Robot Development with Layered Learning

The last area that will be discussed that evaluates the effectiveness of layered learning involves using the learning paradigm to develop the decision making capabilities of real, physical robots. Cherubini, Giannone, and Iocchi [5] and Fidelman and Stone [10] produce work that studies the

possibility of using layered learning on robots in two separate approaches. These studies are important barometers because they represent applications to more realistic and challenging problems.

Cherubini, Giannone, and Iocchi [5] ask if a physical robot can learn to correctly perform a complex task by first using layered learning to grasp the task on a simulator. In their work, physical, legged robots are tasked to play soccer and are responsible for scoring goals on an opposition. In a RoboCup soccer 3D simulator, layered learning is used to learn all of the behaviors and their parameters to play the game. Once all of the learning on the simulator is complete, the learned behaviors and parameters are ported to the real legged robots. The work also compares three different machine learning techniques, a genetic algorithm, Nelder-Mead, and policy gradient, to determine which is the best for solving this complex task of robot soccer with layered learning. Policy gradient slightly outperforms the other approaches when the physical robot's performance is assessed. In a comparison between LLGA and when the initial population is randomly seeded, LLGA outperforms the random approach.

Fidelman and Stone [10] use layered learning to train physical robots to perform the complex task of learning gait movement and ball acquisition, the basics of robotic soccer. What is different between their work and Cherubini, Giannone, and Iocchi's is all learning is done on the physical robot without the need for a simulator. In addition, the work aims at having as little human interaction with the robot's learning process as possible, minus task decomposition, layer design, and robot battery changes. Fidelman and Stone's work also uses policy gradient as the machine learning algorithm for its two layers, the outstanding performer in Cherubini, Giannone, and Iocchi's work. Here, the task is decomposed in to two layers, the first layer is to master gait movement for the legged robot and the second subtask is to grasp a ball.

From the work's experiments, layered learning shows a noticeable improvement in success rates of robot speed compared to the initial hand-tuned robot and the reliability in successful acquisition

more than doubles when layered learning is used. Layered learning on the physical robot has been concluded to saving robot behavioral development time and generating better policies than the manually tuned approach. To further highlight layered learning effectiveness on physical robots, Fidelman and Stone use the same training approach described in this paper to train a RoboCup 2004 team which finished 3rd at the US Open 4-legged competition and reached quarters at the international event.

Both works that use layered learning on physical robots demonstrate that the learning paradigm can be successful at solving real, physical world complex problems. Cherubini, Giannone, and Iocchi's work proves that layered learning can be used to seed physical robots with information to solve complex tasks. Fidelman and Stone's work shows that even with all learning done on-board the physical robot, the paradigm can be successful. Combined, both works demonstrate the tangibility of using layered learning to solve complex tasks with physical entities.

The difficulties of training physical robots to learn new behaviors completely on-board makes the achievements in Cherubini, Giannone, and Iocchi's [5] and Fidelman and Stone's [10] works noteworthy. Barrett, Taylor, and Stone [2] discuss the challenges of on-board robotics reinforcement learning. Their work cites that learning with physical robots is difficult because of the physical challenges of robots, including mechanical issues (batteries swaps, part break-downs, etc), time requirements for establishing physical training environment, scenario, and robot setup, and the number of trials needed to grasp the behavior are challenges that must be considered in design. These difficulties are shared among the two instances of layered learning in robotics. Fidelman and Stone's work demonstrates that with a task decomposition and incremental learning approach, these challenges can be overcome to produce physical robots that can learn completely on-board.

Discussion on Layered Learning

From the survey of layered learning in the literature, several observations were made about the practicality of deploying the learning paradigm. These observations include the advantages of using layered learning, the challenges practitioners can face with the paradigm, and the distinction between layer overlapping and isolation.

Advantages of Layered Learning

A number of positive features can be taken from the collection of existing layered learning work. The most noteworthy features include the layered paradigm's simplistic method of solving complex tasks, similarly to how humans naturally learn, consistent outperformance of standard, monolithic learning, accelerated learning rate, learning reuse, and use of machine learning abstraction.

The simplistic nature of layered learning derives from the small number of fundamental steps in the paradigm's procedure:

Step 1: Accept a progressively complex task decomposition.

Step 2: Learn each subtask in sequential order of the decomposition.

These two steps guide layered learning to master complex tasks in an uncomplicated process. The simplicity stems from the fact that all of the complexities of learning reside outside of the responsibility of learning paradigm. For instance, the task decomposition is provided to the paradigm as an input prepared by an external source. The task decomposition is a vital component to the overall success of the learning process because it determines the order in subtasks the learner will learn and the solution space size that is navigated through at any point of the learning process. This great responsibility of providing an accurate task decomposition is provided by an external process of

the paradigm.

A second yielded responsibility is the process of learning each subtask. Layered learning requires layers to be established with a machine learning algorithm to be selected to learn each subtask; it is the responsibility of the employed algorithm to learn to perform the subtask. In other words, layered learning is responsible for providing each algorithm with the inputs it needs to master the subtask. The process of turning those inputs into a policy that can execute the subtask is solely the algorithm's responsibility. Both the generation of task decompositions and the details of subtask learning are provided by processes outside the direct responsibility of layered learning, making the layered paradigm simple to understand and uncomplicated to design.

In addition to the simplicity of the paradigm, layered learning is also similar to how animals naturally learn. Humans and other animals tend to rely on decomposition when learning new skills. The decomposition allows for previously learned skills to be incorporated in the learning of a new ones. This form of reuse allows learners to not have to relearn previously learned skills, saving time and effort to learn the new skill. A second advantage to having a learning paradigm be simple and natural is that it is easier to be understood by practitioners than other complicated approaches. With an approach that is easier to understand, a practitioner will be able to recognize the pros and cons of the approach and know when it is appropriate to use the approach.

Another feature of layered learning can be derived from the work that compares the paradigm to monolithic approaches. In the review of the state of layered learning, the decomposition approach is highlighted by consistently outperforming standard, monolithic learning techniques when solving complex tasks. The favorable performance is attributed to the learning paradigm's use of smaller solution space series that the paradigm sequentially navigates through as it solves subtasks.

Challenges of Layered Learning

Layered learning has a few challenges that need to be considered. First, the paradigm is always reliant on an external source to provide the task decomposition. The dependence of an external decomposition-provider is a challenge because the paradigm does not have any bearing on how the decomposition is made and how effective it will be in collaboration with the learning process. Clearly, this provided decomposition is vitally important for learning the complex task. This claim is held up in the work that analyzed layered learning’s effectiveness for solving Boolean logic problems [23], which showed how different decomposition of the same task can have a great effect of the outcome of learning. If the task is not decomposed in a way where the subtasks’ solution spaces are small enough to effectively learn the subtasks and the subtasks do not accurately represent essential components of the task, then the learning agent will not learn the task optimally.

A second challenge is the effort required to design each layer. In the paradigm, each layer requires careful decisions to be made on machine learning technique selection, training environment and scenario set up, layer halting conditions, etc. Each of these decisions requires considerable planning to ensure every layer can result in the subtask being solved at an acceptable level. Consider the advantage layered learning has in its ability to allow layers to be composed with the best machine learning technique to solve the subtask. The trade-off is that although this freedom allows for the selection of the best tool for the job, it requires additional planning and design of each layer compared to if one machine learning technique is forced on all layers.

Additional challenges lie in how the layers are designed; more specifically, if the layers follow a *layer isolation* or *layer overlapping* method of learning. Although these two design choices are dictated by the employed task decomposition and aggregation methods, the decision to use isolation or overlapping can have significant impact on how the learner learns the task and how effective the instance of the learning process will be.

Layer Isolation

In layer isolation, independent sub-policies are manipulated to learn subtasks and the learning in one layer does not alter the sub-policy of another. The majority of the layered learning studies use layer isolation because of its support for a clear, well defined hierarchical task decomposition. In addition, layer isolation conveniently sets up a straightforward comparison between layered learning and standard monolithic learning because both methods contrast each other. In the monolithic case, everything required to perform a task is learned at once, in one solution space. Examples of layer isolation are in Hsu and Gustafson’s [20, 21] and Gustafson and Hsu’s [18] works with keep-away. In their decomposition, the learning keepers learned the independent skills of first intercepting the ball then to pass to a teammate. After the learning phase, a hard-coded controller is used to determine when it is appropriate to perform either subtask. Although layered learning has been introduced to overcome the challenges of the monolithic standard, the use of isolation in the layered paradigm has a number of issues that can seriously hinder layered learning’s effectiveness.

First, under layer isolation, subtasks can be out of context of their intended use by other subtasks. For example, *subtask B* depends on and invokes *subtask A* to be able to perform *subtask B*. In *layer A*, the *sub-policy A* is modified to perform *subtask A*. Because *layer A* can only train *sub-policy A* with information it has already acquired (through learning or domain knowledge), information to mold *sub-policy A* to satisfy *subtask B* must be designed into the layer; if not, *sub-policy A* may perform *subtask A* effectively but not *subtask B* because *layer A* learned *subtask A* out of *subtask B*’s context. In other words, *sub-policy A* can learn to perform *subtask A* optimally from *layer A* but *subtask B* may perform sub-optimally because *subtask A* was not learned to satisfy *subtask B*.

Two solutions to overcome this context problem is to: 1) insert domain knowledge into *layer A* to guide *sub-policy A* to also support *subtask B*’s requirements of *subtask A*, and 2) use additional layers to refine learned subtasks. Inserting domain knowledge is difficult because such knowledge

may not be available *a priori* to layer design. If such knowledge is available, additional challenges to correctly incorporate the domain knowledge into training scenarios and environment may arise.

Refining subtasks to match the context of other subtasks is redundant and can further complicate contextual issues. *Sub-policy A* can be refined to support *subtask B* after *layer A* is complete by modifying *sub-policy A* in a layer that comes after *layers A* and *B*, let's call it *layer A'*. This refinement layer includes new training scenarios that will guide *sub-policy A* to performing *subtask A* and meeting objectives of *subtask B*. Refining *subtask A* in this manner can be redundant because additional learning time is given to *subtask A* after *layer A* is complete. Two arguments can be made whether this refinement is no longer layer isolation: 1) a layer is modifying the sub-policy of another (*sub-policy A* is modified in *layer A* and *layer A'*), and 2) *layer A* and *layer A'* can be considered the same layer just split and executed at different times.

Layer isolation experiences further problems when more than one subtask depends on another, such as *subtask A* being used by *subtasks B* and *C*. Even if *sub-policy A* satisfies *subtask B* through domain knowledge injection or the use of a refinement layer, there is no guarantee that *sub-policy A* will meet the context of *subtask C* as well. If domain knowledge is used to set the context correct, additional consideration must be used in the design of the layers to support the interdependence of all related subtasks, not a feat to be taken lightly; similar design considerations must be made with refinement layers. If refinement is the tool for correcting the context, then several refinement layers may be required to guarantee all subtasks that are used by other subtasks match their contextual requirements, such as the avoidance of circular subtask dependencies. Both cases require added design time and the potential for wasted redundant computational effort to make sure the components of the hierarchy satisfies the overall task.

The next issue is that simply aggregating sub-policies into one to solve the overall tasks brings on additional challenges to layer isolation. A common practice among layered learning is to use a

static hierarchical structure to decompose and organize the complex task. The structure, such as decision tree, acts as a container for all of the sub-policies and invokes each sub-policy when it is appropriate to perform a subtask. The hierarchy, decomposition, and how it makes decisions on when to invoke subtasks is given to the learner, but the learner learns each subtask through layered learning.

It is overly optimistic to assume in simple aggregation that a task can be decomposed into inter-dependent components and believe training sub-policies independently will often produce a reasonable solution. This optimism overlooks difficult questions about how one rewards the separate sub-policies in an effective manner that ensures the overall task can be learned optimally and how one guarantees that the learned ability to perform a subtask will seamlessly fit into another subtasks that depends on it. Both questions are related to the contextual issues discussed above and require deep consideration when attempting to aggregate a collection of sub-policies to perform a complex task.

Finally, layered learning is designed to incorporate progressive learning. Although layer isolation supports learning in low-to-high levels of task abstraction, isolation makes learning other forms of progressive learning more difficult. Consider the case of incremental learning, where an agent progressively learns higher orders of a task, from the most simplest to the most difficult tasks. In keep-away soccer, progressive learning could see a *keeper* team learn to maintain ball possession with one *taker* opponent, then two, three, etc., until the agents can perform the intended task against the target number of opponent *takers*. The reason why this type of progressive learning is difficult in layer isolation is because of the policy sharing that is required with each layer. Because learning in the manner of the keep-away example requires everything that has been learned to be passed to and modified in the next layer, layer isolation's sub-component building does not support this form of progressive learning. The lack of incremental learning support in layer isolation is unfortunate because it limits the range of problems layered learning can solve, especially those that rely on

iteratively building up to a complex task from simple, low-order subtasks.

Layer Overlapping

Layer isolation preserves learned information as the learner iterates through the layers at the cost of prohibiting subtask refinement, overreliance on the decomposition and aggregation being correct, and suboptimal task performance due to fragile relationships between subtasks. The alternative to layer isolation is *layer overlapping*: when two or more layers can modify shared policy information. Layer overlapping is the middle-ground between the two extremes of monolithic learning and layered learning with layer isolation. In overlapping, tasks are still decomposed but layers can be designed to modify the same policy in succession. Overlapping also supports the partitioning of a policy into separate segments and solving individual subtasks just like layer isolation; the difference is that overlapping layers modify shared sub-policies. Both types of overlapping layer design result in layers directly altering the performance of learned subtasks. The work that studies layered learning effectiveness in solving Boolean logic problems [23] provides an example of layer overlapping. In the latter experiments in the work, decomposing the task in low-to-high order complexity variations of the overall problem, showed how a layer can use and modify information learned prior to the current layer to learn the task. In the experiments, this overlap saw the Boolean logic problems solved more effectively and efficiently than the isolation approach.

Mondesire and Wiegand [39] provide another example of layer overlap and compares layered learning to standard monolithic learning. In the work’s experiments, policies to control individual agents of a predator team are evolved to capture a hand-coded prey. The complex task is for the predator team to capture the prey as quickly and reliably as possible. The task was decomposed into three subtasks: 1) capture the prey with a great probability of success, 2) minimize hunting time, and 3) capture the prey reliably and quickly. For subtask 1, the predator team is given 10 tri-

als to catch the prey; catching the prey 9 out of 10 trials equated a positive reward high capture rate of 90%. Because the roles of the predators and prey could temporarily switch through the course of a trial, subtask 2 sees the learning agents try to minimize the hunting time by either catching the prey quickly or by sacrificing themselves to be captured by their single opponent. subtask 3 aggregates the first 2 subtasks and represents the complex task the agents are learning. Layer overlapping occurs because all three layers for an individual agent modify the same decision tree select movement and communication actions the agent execute based on current state conditions. With this decomposition, the agent team is able to learn to achieve an effectiveness subtask, efficiency subtask, and finally, an aggregation subtask where each layer builds on previously learned information.

In addition to overlapping's favorable results in the Boolean logic work, layer overlapping has a number of positives over layer isolation. First, added design time, as seen in isolation cases, is not required to guarantee that the sub-policies are completely independent of each other. In layer overlapping, shared sub-policies can benefit the learning experience by improving earlier learned subtasks by means of providing additional contexts to related subtasks. Also, additional subtask training time can be incorporated into later layers to enhance a previously learned subtask with a new context and newly acquired information without adding to the overall time spent learning. Concurrent layered learning can be an example of this last point because a subtask learned early in the learning process can be refined in parallel with another subtask. Finally, unlike layer isolation, a task decomposition with layers progressively increasing the order of subtask difficulty is fully supported under layer overlapping. Because a policy can be modified in its entirety as it is passed from one layer to the next in layer overlapping, layers can be designed to refine every learned state-action pair to suit the current layer's subtask.

Layer overlapping introduces layered learning to a difficulty that is common in multiple objective learning, including decomposition learning, which is the requirement for the learner to have rules

and safeguards to preserve important learned information when learning to perform new objectives. In decomposition learning, learning tasks by solving subtasks, multi-objective learning is the process of learning the subtasks that make up a task. Two challenges arise when learning a new subtasks modifies a policy: 1) information needed to perform older subtasks is lost in favor of information used to perform the new task, and 2) information to perform the new subtask is not learned sufficiently so that old information can be retained.

The trade-off between these two challenges make up the *stability-plasticity dilemma*, a problem shared among all multiple objective learning approaches. The dilemma has been studied in machine learning, producing many different solutions to balance stability and plasticity. Common solutions include task rehearsal, policy deletion strategies, and the implementation of complementary learning systems.

Although the stability-plasticity dilemma in layered learning has not been analyzed in any published work, the common, possibly unintended, solution among layered learning approaches is to use layer isolation. Recall, in layer isolation, layers only modify portions of a policy that are not shared with other layers. This use of isolation guarantees that information learned in one layer does not destroy information used in another. The use of layer isolation also places additional design pressure to guarantee that the decomposition and means of aggregation are correct. Because isolation occurs, information used by a subtask is not further refined after the layer is complete, meaning the task must be decomposed in a way that the tasks build off of each other by only using previously learned information and not modifying it.

Stability and plasticity become a challenge with layer overlapping because information that is important to one layer can be lost in favor of another. The issue arises if a vital subtask is modified while learning another subtask because the performance of the overall task can be negatively affected due to the sub-component's sub-optimality.

The predator-prey work conducted by Mondesire and Wiegand [39] indicates the stability-plasticity imbalance with layer overlapping. The study shows that as the agent learns a new subtask, the performance of previously learned subtasks can be altered because the new layer modifies the shared policy. In the predator-prey experiments, favoritism of the newer subtask is evident in how the performance to capture the prey decreases considerably while the agent learns to minimize the hunt time. It takes the third layer, the aggregation subtask, to relearn the information lost in the second layer to regain the ability to capture the prey with a high success rate. This form of forgetting suggests that layers can be detrimental to each other and the time and effort required to regain lost important information is wasted opportunity to search for more optimal solutions instead of repairing what was lost.

The stability-plasticity dilemma caused by layer overlapping is a major challenge in layered learning because of the disastrous affect it can have on the learning process. The trade-offs between isolation and overlapping make the latter a viable option in layer design but because of the forgetting challenge, overlapping requires additional investigation into why the imbalance between stability and plasticity occurs and how it can be mitigated.

Other Decomposition-Based Methods

Layered learning is not unique in its use of task decomposition and incremental learning to autonomously train agents to learn new tasks. In fact, the paradigm has roots from transfer learning, robot shaping, and hierarchical decomposition learning, among others. Next, a discussion of similar machine learning approaches to layered learning will be provided, followed by a comparison between these and layered learning.

Transfer Learning and Robot Shaping

At layered learning's core lies the machine learning approaches of transfer learning and robot shaping. Transfer learning refers to the general reuse concept of knowledge gained in previous experiences applied to new problems [2]. This type of learning is founded on the principle that although it is conceivable that tasks be relearned every time a change occurs in the environment, task, or robot, it is often not practical to make the additional effort [2]. This form of reuse is seen in layered learning with how layers build on information gained in earlier layers. Layered learning is designed to exploit task decompositions where successively learned subtasks rely on the information learned previously. The repetitive use of earlier obtained information reduces unnecessary redundant learning.

Robot shaping is a development technique where the agent progressively learns successive approximations of a behavior to master the target task [9]. In relation to layered learning, these approximations are the subtasks decomposed from the overall task the agent is learning. Layered learning's process of progressively learning more complex subtasks guides the learner to the solution of the overall task. Another similarity between shaping and layered learning is when it is appropriate to employ either approach. Shaping is best used when learning to perform a task is too difficult to be learned directly [9]. Similarly, layered learning is best deployed when learning the task via a monolithic is too arduous and inefficient. Both approaches possess the ability to reduce the solution space to a series of smaller spaces and navigate through each one more efficiently.

Both transfer learning and robot shaping clearly influence layered learning. The emphasis on learning reuse in transfer learning and shaping's focus on progressive learning both are evident in layered learning's core principles. These properties of reuse and divide-and-conquer are repetitive themes in decomposition-base learning.

Hierarchical Decomposition Learning

Hierarchical decomposition learning consists of techniques that decompose a task into a meaningful hierarchy. These decompositions organize a task into a sub-component structure, typically in a bottom-up fashion where it is common to have the target task at the highest level of abstraction and low-level, basic subtasks underlying it. Learning with hierarchical decomposition takes place in a number of forms, ranging from learning the appropriate way to arrange the decomposition to modifying individual behaviors and their parameters.

Three hierarchical paradigms that share significant structural properties with layered learning are *subsumption*, *case base reasoning (CBR)*, and *context-based reasoning (CxBR)*. In the subsumption architecture, an agent is decomposed into modules, also called layers, to solve tasks [4]. The layers are arranged in a bottom-up hierarchy, where high-level, abstract behaviors are at the top and low-level, basic behaviors are at the bottom. In this hierarchy, each layer relies on underlying layers to perform the behavior it is responsible for.

Both CBR and CxBR base their decomposition hierarchy on state abstractions instead of implementable sub-behaviors of the overall task. In CBR, the agent's decision making is guided by a hierarchy of cases that represent abstractions of the state the agent will reside in at any given time. CBR case-trees are typically composed of the most abstract case the agent can be in, followed by cases that contain more detail the further it is from the root case.

CxBR is similar to CBR by basing decisions on a hierarchy of state abstractions, call contexts. Contexts represent abstractions of the agent state and guide the decision making. Within contexts reside *sub-contexts*, *actions*, and *transitions* to other contexts. At each decision step, every context evaluates the agent's state and returns a priority value that is used for the system to select which context to activate. The context with the highest value is selected and the actions contained in the

context are executed.

The main difference between these three paradigms and layered learning is that although subsumption, CBR, and CxBR guide the decision making ability for agents, they are not explicitly designed to evolve their hierarchies; layered learning is a process which guides a policy towards optimality by taking a decomposition and mastering each component.

Hierarchical Reinforcement Learning

The area of *hierarchical reinforcement learning (HRL)* contains a collection of machine learning approaches that share a number of similarities with layered learning. In this sub-domain of hierarchical decomposition learning, the structure and task decomposition is applied to RL to improve the range of effectiveness, especially when learning to perform complex task.

A majority of the HRL works discussed here seek to improve existing RL technique of Q-Learning due to Q-Learning's popularity in the RL domain. For a concise review, *computational RL* is a technique that focuses on maximizing future rewards [9]. RL relies on the *Markov Decision Process (MDP)* to base decisions on. The MDP is a 4-tuple decision making model defined by $(S, A, P_a(s, s'), R_a(s, s'))$. S is the set of states that contains states s and s' , A is the set of actions that contains action a , $P_a(s, s')$ is the probability of moving from state s to s' when action a is performed, $R_a(s, s')$ is the immediate reward received when transitioning to state s' from s after performing action a . The aim of any reinforcement learner is to maximize the reward for selecting an action.

Q-Learning is an RL technique that focuses on constructing a function that allows decisions to be made based on estimated reward of experiences, using a discounted cumulative reinforcement utility [31]. Q-Learning has become the standard RL algorithm having proven its effectiveness

in estimating future rewards. Standard Q-Learning is flat and is optimized to handle a single task that can be learned monolithically. Problems occur when the task is complex and contains a solution space that is too unwieldy to navigate through monolithically [6]. Many approaches turn to hierarchical decomposition to relieve this large solution space constraint.

Lin [31] outlines three steps that are essential to any HRL approach:

1. Task decomposition: establishing subtasks and reward function for each subtask.
2. Learning elementary skills: deploy learning algorithm to maximize effectiveness of each subtask.
3. Learning a high-level skill: requiring the coordination of elementary tasks to perform the composite task.

Layered learning follows these three steps. Step 1 is reflected in layered learning's requirement to accept a task decomposition and the establishment of layers (with the selection of a machine learning algorithm to solve each layer's subtask). Step 2 is the iteration of each layer where the layer's machine learning algorithm navigates through the subtask's solution space for the optimal policy. Step 3 is evident in layered learning's roll up of layers, where the dependency of previously learned subtasks is used by later layers.

Next, a list of noteworthy HRL approaches similar to layered learning is provided. In the discussion, descriptions of how each approach follows the three steps above will be made and underline the properties each approach shares with layered learning.

Compositional Q-Learning

Singh [52, 53] points out that Q-learning was designed to support tasks with only single goals and not tasks with multiple goals; this claim implies that there needs to be a method that permits elemental tasks to be shared among composite tasks to avoid duplicated and wasted effort in relearning elemental tasks. To address this lack of multiple goal support, Singh proposes *Compositional Q-Learning*, a learning algorithm that makes Q-Learning support *composite sequential decision tasks* (CSDTs) by applying costs to actions and rewards to states. CSDTs are tasks where a sequence of actions must be performed by an agent to accomplish its goal. The key is that the payoff for a subtask is assigned at the state level and not at the state-action pair. This assignment allows new tasks to be learned simply by updating the rewards at states the agent must arrive at in order to achieve the learning task.

Compositional Q-Learning proposes a new way of evaluated Q-values when the agent is learning to perform a composite task with elemental tasks already learned. The new composite Q-value considers the best elemental tasks that returns the highest estimated reward to get the agent to the highest paying reward state. The method also uses a scheduling architecture to learn the decomposition and Q-value function of the composite task.

Q-Nets

A Q-net is a multi-level neural network that uses back-propagation and temporal difference to optimize policy [31]. Q-nets apply hierarchical reinforcement learning to a complex problem using multilevel NNs for control policy development. In Lin's study, the hierarchical learning system is compared to a monolithic approach that uses a single layer NN that tries to locate the maximum $Q(\text{state}, \text{action})$ utility. The multilevel hierarchical approach first learns the elementary tasks with

$Q(\text{state}, \text{action})$ then finds the optimum policy with $Q(\text{state}, \text{skill})$ where skill is the elementary task to execute. Switching from action to skill reduces the solution space size in the high-level task. In Q-nets, both behaviors and gating function (a function that determines when to switch to other behaviors) are learned. First behaviors are learned, fixed, and the gating function is optimized [56]. The hierarchical approach outperforms the monolithic one because of the reduction in solution space from the better state representation and the ability to learn tasks easier with smaller networks.

Q-Cut

Q-Cuts [38] proposes a modified version of Q-Learning to address the issues of RL solving MDP problems in unreasonable amounts of time. Additionally, it addresses that task decomposition is difficult because a correct decomposition may not be obvious and may not be available *a-priori* to learning. The approach exploits a common way of identifying subtasks in the decomposition phase - by understanding the state-space context to identify *landmark* states.

Q-Cut uses graph theory to automatically identify sub-goals in complex tasks. The approach uses frequency of states that lead to the positive path of satisfying the target task. By identifying frequent states, the approach assumes that these states ("*bottlenecks*") are important to the agent at performing the given tasks. The algorithm then takes the notion that by creating sub-goals that lead the agent from bottleneck to bottleneck, the agent will master how to solve the task. The task decomposition results in the agent learning to reach each bottleneck (perform each subtask) separately, later to be combined to solve the overall task. This approach does require a lot of trials to identify which states are bottlenecks but results show that it is an improvement over standard Q-learning by speeding up the ability to learn these tasks [38]. The testing problem saw agents learn to navigate through an environment to reach a goal in another room.

Feudal Reinforcement Learning

Feudal reinforcement learning [6], a Q-Learning modification that decomposes a task into a feudal hierarchy controlled by *super-managers*, *managers*, and *sub-managers*. Each level of manager is responsible for a form of the original task. Super-managers are given the ultimate goal the agent is trying to solve. Super-managers assign managers a subtask, cost, and reward system. Managers further decompose their assigned subtask and provide the costs and rewards for solving their sub-subtask. Each type of manager is responsible for maximizing the reward for their task decomposition and is only provided information to achieve their goal. Returned values percolate upwards, allowing managers to determine which sub-manager is best to carry out their subtasks.

Managers and sub-managers are solely responsible for solving their assigned tasks. Although the overall agent may solve the task without a sub-manager solving its own, the sub-manager does not receive any positive reward. Also, if a sub-manager solves the assigned task and it does not lead to a favorable outcome for the super-manager or manager, the sub-manager is not punished.

Feudal reinforcement learning uses *reward-* and *information-hiding*. Reward-hiding is rewarding sub-managers for learning tasks whether or not it satisfies the commands of upper management. In information-hiding, managers are only given access to state information that affects their assigned tasks.

Dayan and Hinton's experiments to solve a maze task show the proposed feudal reinforcement learning system does speed up learning (locating the goal) compared to standard, flat Q-Learning over time, as it takes the technique a while to remove impossible subtask assignments.

Hierarchical Abstract Machines (HAMS)

Parr and Russell [44] propose *Hierarchical Abstract Machines (HAMS)* as a machine learning technique for learning decision making in “uncertain” environments. HAMS are hierarchies of nondeterministic finite state machines that move an agent from state to state. The goal of the work is to constrain the policy’s actions to those with the highest potential reward. The approach is based on MDPs where the aim is to decompose the decision making processes into tiers of task abstraction.

The HAM machine has four types of machine states: *action*, *call*, *choice*, and *stop*. When in a choice state, the agent decides what other machines (to handle another subtask) should be executed next. The call state calls the decided on machine. Action state invokes a primitive actions and the stop state halts a machine’s execution and returns control to the machine that invoked it. The top tier of a HAM can represent the overall task and will call another machine to perform a subtask. Once all of the recursive subtasks are complete, control returns to the top tier machine, like a main function in a program, to either continue invoking a sub-tier machine or halt execution. The approach has been applied to Q-Learning, calling the modification *HAM Q-Learning*.

MAXQ

Many RL algorithms that focus on task decomposition cater to specific tasks, creating a need for an approach that is a general, more abstract method of using RL with hierarchies to search for optimal policies.

Dietterich [7, 8] proposes *MAXQ*, a RL method for decomposing tasks into sub-components and solving the task by recursively solving all of its subtasks. The method requires decompositions to be hand-coded, but once provided, the method accepts a MDP assigns subtasks to nodes (*Max*

Nodes and *Q-nodes*) to perform the tasks. Max Nodes come in two varieties: 1) Max Nodes without children are primitive actions and 2) Max Nodes with children are subtasks. Max Nodes are context independent and focus solely on maximizing the expected cumulative reward for performing its own assigned subtask. MaxNodes (with children) can be shared among Q Nodes for reuse. Q Nodes are actions that are performed to get its parent node's subtask satisfied and are context dependent on its parent node. In other words, Q Nodes return rewards awarded for its performance and its parent's performance. It should also be noted that in MAXQ, there is a hierarchical-policy where each Max node has its own policy for satisfying its own task.

The main difference between the MAXQ form of hierarchical decomposition and others is that MAXQ has the advantages of possessing subtasks that can be reused without relearning and reward approximations are available at all levels of the decomposition; for instance, the root MAX Node contains the estimated reward for executing a policy at that high level. In addition, each node is responsible for maximizing its reward when performing its subtasks, both context dependent and independent.

Comparisons to Layered Learning

The characteristics of task decomposition, learning reuse, and divide-and-conquer are the core of any hierarchical decomposition approach, including layered learning, hierarchical RL, and many other approaches. There are, however, a number of differences in the details separating layered learning from the others.

The level of abstraction created by the hierarchical decomposition is similar to all of the discussed approaches. In most of the decomposition approaches, the task is decomposed in a bottom-up manner where the top tier is the most abstract definition of the task to be solved, followed by low-level behaviors that support higher tiered subtasks. The abstraction differences take place in

two key areas of how lower-level subtasks are learned. The first is that the layered paradigm uses a sequential plan to progressively learn complex tasks, where a layer relies on the information obtained earlier. Many of the HRL approaches learn these subtasks in a depth manner, where layered learning tackles the issue in a breadth, serial sequence. The second abstraction difference is how each subtask is learned. In layered learning, layers are constructed by having a machine learning algorithm selected to learn how to perform a subtask. This freedom explicitly allows for different algorithms to be used and is contrary to the standard RL approaches of selecting one algorithm for all subtasks. The machine learning algorithm abstraction at the layer level allows for the best algorithm to be selected to solve any given subtask. From the layered learning definition, any of the mentioned HRL algorithms can be employed within a layer as long as the HRL algorithm accepts the layer's subtask as its task to solve and can reuse learned information from previous layers.

A second feature is that layered learning has the flexibility of being wide or narrow in terms of hierarchical structure; the distinction is based on the decomposition the instance of layered learning is accepting. Consider a task decomposition of soccer where the highest tier is “play soccer”, next tier is to “play offense” and “play defense”, where “play offense” is decomposed into two subtasks of “score goals” and “move to the ball” and “play defense” has two subtasks of “steal the ball” and “move to the ball”, as shown in Figure 2.

In the layered learning paradigm, the learner may learn through the layer sequence of movement then steal, score goals, play offense, play defense, and finally, the aggregate layer of playing soccer. With this decomposition, the task is decomposed into a bottom-up hierarchical structure with the top tiers having width in their subtask options. On the other hand, a layered learning agent may decompose the task into progressively complex subtasks that have a width size of one, as depicted in Figure 2.

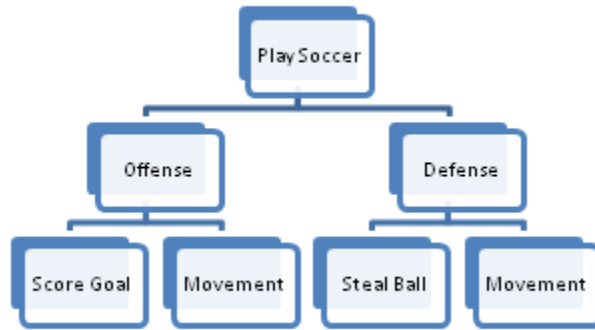


Figure 2.1: A deep task decomposition with a hierarchy of abstraction.



Figure 2.2: A flat task decomposition with subtasks that are progressively more complex versions of the overall task.

For instance, soccer can be decomposed to play the game with 6, then 7, then 8, ..., and eventually 11 players per team. In this decomposition, layers progressively aim at having the learner pick up all of the skills of soccer against an increasingly amount of players on the field. The first form of decomposition is widely supported through classical HRL but the former and latter are supported by layered learning.

Another feature that makes layered learning stand out is its use of learning reuse. The common way to reuse knowledge in hierarchical decomposition approaches is to have higher-tier subtasks access the same lower-tier components. By doing so, the learner only has to learn the lower-tier subtask once instead of redundantly relearning how to perform the lower level subtask for each higher-tier

one. For instance, let's again use the above example of soccer where two higher subtasks are to play defense and to play offense. Both subtasks require the low level subtask of movement. In this form of reuse and wide decomposition, the movement subtask will only need to be learned once instead of learning two separate moving subtasks, one for offense and one for defense. The exact type of reuse can be used in layered learning and the RL algorithms discussed earlier. The difference is that layered learning can also reuse knowledge in the narrow decomposition (6v6, 7v7, etc...) by making sure layers use the information gained in previous subtasks. In the example that progressively increase the number of field players, the reuse comes in from a layer using the information learned in a lower order layer. For example, the 8 vs. 8 layer will use the information learned to handle a field with 12 and 14 players.

Although the core properties are similar throughout, the level of abstraction and freedom layered learning has separates it from the other decomposition approaches.

Benefits and Challenges of Decomposition Learning

Improved exploration, task decomposition, and policy reuse are the benefits of using decomposition learning, such as HRL, over standard, monolithic approaches, like flat RL [7]. The improvement in exploration comes from the higher levels of the hierarchy's abstraction tree. Any changes at such a high level of abstraction can cause big changes in policy exploration as the entire decomposition's make up is affected. Second, task decomposition allows for fewer learning trials because each subtask has a smaller solution space to navigate through. Instead of searching through the entire solution space at once, as in a monolithic approach, hierarchical decompositions serialize the space into smaller, more manageable partitions. Finally, because of the hierarchy's organization of task components, subtasks can be reused by higher abstractions, saving time and effort that would be used on learning redundant subtasks used by higher level components.

Several challenges exist with decomposition learning, including issues with how the task is decomposed, how decomposition elements are aggregated to solve the original task, and how the learner balances stability and plasticity while learning subtasks.

A good task decomposition requires domain knowledge or a large amount of experience [31]. If a learning system does not have means to automate the task decomposition, domain knowledge and experience may be difficult to come across. Domain knowledge and experience are vital because the information they provide can seed a learner with an effective learning plan based on the decomposition that will reduce the solution space of each subtask. Incorrect decompositions can lead to the task not being correctly performed or wasted time navigating through a solution space dense with useless solutions. Jackson and Gibbons' [23] Boolean logic problem solving work demonstrates the importance of a good decomposition, showing how poor decompositions can result in a solution to never be found or sub-optimal performance. The complexities of issues surrounding decomposition and aggregation, in terms of cooperative coevolutionary approaches to function optimization, are well known [62].

Just as it is important to decompose the task into manageable components, recombining the learned subtasks is another challenge. Whether it is a static hierarchy that invokes subtask behaviors or if the hierarchy contains a dedicated subtasks that combines all of the information obtained in the learning process, a mechanism must be in place that reconstitutes the decomposition to solve the overall task. Any form of aggregation requires consideration while designing a decomposition-based approach that insures all of the learned components can be combined into one decision making system without any unintended loss of performance.

Of particular concern to us is the aforementioned challenge of the stability-plasticity dilemma in the learning paradigm, forgetting in particular. The following section discusses this challenge in detail.

Forgetting and Stability-Plasticity Imbalance

The act of forgetting can be a beneficial occurrence or hindrance to any learning algorithm. Generally, *forgetting* is the loss of information. In the context of direct policy search applications of layered learning, forgetting is the loss of one or more state-action pairs used to perform a task or subtask at a previously measured level. Here, we are concerned with either a subtask's, or the overall task's performance, decreasing as a new subtask is being learned.

The physical limitations an agent has for storing a policy is a reason why forgetting occurs. The storage limitation becomes an issue when the collection of state-action pairs grows larger than the size of storage for the policy. When this occurs, the learning algorithm must consider the fact that all acquired state-action pairs cannot be stored at once and must decide what items of the policy should be stored and which should be discarded.

The challenge is to determine which state-action pairs positively contribute to the task's performance. If the learning algorithm can determine which pairs are not useful, then the unnecessary pairs can make way for new mappings that may increase performance. Unfortunately, the identification of unneeded pairs is difficult because decision making tends to be dependent on the sum of its policy and pinpointing the exact cause of under-performance is a challenge.

Types of Forgetting

The common misconception is that all forgetting is bad. In fact, though forgetting might be categorized into a number of different types, in this work, forgetting is categorized simply as positive, neutral, or negative.

Negative forgetting is the loss of state-action pairs that degrades a task's performance. In essence,

this type of forgetting replaces a policy that has been beneficial for performing the task with a policy that is less effective. In layered learning, negative forgetting results in the progress made in one layer to be lost in another. *Positive forgetting* is the process of replacing an underperforming state-action pair with one that is more effective. This type of forgetting creates opportunities of arriving at novel solutions because it replaces a less effective policy with a more effective collection of state-action pairs. Additionally, positive forgetting is commonly referred to as *learning* because the agent is improving its solution to the targeted problem. *Neutral forgetting* takes place when the modification of a policy is neither positive nor negative for the performance of the task or subtask; it has no effect on the performance of what is being learned.

Following the same notation as the definition of layered learning used in the layered learning section, below are definitions of negative, positive, and neutral forgetting as these apply to layers. Again, function f is the performance function used to evaluate the proficiency of a subtask's (t_i) execution when given a policy (P_i).

Let t_i and t_j be subtasks for the i^{th} and j^{th} layer of learning, respectively, where $j > i$. The learner has exhibited forgetting if after learning policy P_i for layer i and P_j for layer j , $P_i \neq P_j$. The following definitions classify forgetting in types with the same notation:

- Negative forgetting: $f(t_i, P_i) < f(t_j, P_j)$
- Positive forgetting: $f(t_i, P_i) > f(t_j, P_j)$
- Neutral forgetting: $f(t_i, P_i) = f(t_j, P_j)$

It is understood that performance changes may be caused by many factors. We generalize those factors into the performance consequence of policy change. With that said, ideally, a learning algorithm is equipped with a design that attempts to maximize positive forgetting while minimizing

its negative counterpart. This objective means that the learning algorithm tries to retain beneficial state-action pairs and remove ones that are hindrances. Unfortunately, removing negative forgetting entirely is a challenge because it is not always possible to predict the effects of policy modification. The effects of forgetting are typically known at hindsight, after the policy has been changed. Only after the policy has been modified can the performance state be measured to compare the before and after effects of forgetting. Until this thesis, there were no serious studies to understand when and how negative forgetting occurs in layered learning or what techniques are best used to mitigate such problems.

Instances of Negative Forgetting

Markotvitch and Scott studied the role of forgetting in order to produce more effective learners [34]. In their work, *knowledge* is the output of what is obtained through the learning process and is akin to the state-action pairs of a policy. The term *negative knowledge* is used to describe the product of negative forgetting. It is synonymous to the less effective state-action pair that replaces a better performing pair when negative forgetting occurs. Nakayama and Yoshii also define an instance of negative forgetting as *obstacle data* [43]. They define obstacle data to be knowledge that leads to bad influences, or poor decision making, in a system.

In Markotvitch and Scott's discussion of forgetting, negative knowledge is described as the product of learning that may solve the task but is less effective than other knowledge items [34]. Their work claims that through the process of forgetting, negative knowledge can be removed from a policy and replaced with more effective instances; in other words, negative knowledge can be corrected with positive forgetting. To prove this claim, Markotvitch and Scott have shown in their experiments with autonomous agent graph traversal that random forgetting improves the performance of learners by removing under-performing knowledge items and that learned knowledge can be

ineffective even when it leads to a correct completion of a task [34].

Stability-Plasticity Dilemma

The *stability-plasticity dilemma* is a machine learning problem that is directly related to negative forgetting. Identified by McCloskey and Cohen, the stability-plasticity dilemma is a learner's balance between being able to consistently perform learned tasks at a specific level while possessing the ability to learn new things such as the ability to adapt to new tasks, states, and environments [37]. Also known as the *sensitivity-stability dilemma* [14], coping with stability-plasticity dilemma seeks to keep a stable performance of old tasks while the learner is learning from new ones.

Yaochu and Sendhoff make the case that the dilemma is a problem that all learners must face because a learner should be able to learn new information without completely forgetting information that has been learned previously [24]. The dilemma becomes an issue when a learner is given a new task to learn after previously learning another one. Without adequate balance, two generalized outcomes can occur: 1) on the one hand, when stability is favored, the learner will be slow to adapt to a new task because knowledge used to perform an older task is preferred, 2) while on the other hand, when plasticity is favored, the learner's policy optimizes on a new task with greater ease by replacing what it has learned to solve an older task with state-action pairs used to solve the new task. Both forms of imbalance affect the learning rate because pairs (either obtained while learning new or existing tasks) are discarded from the policy, resulting in a type of forgetting. For instance, when plasticity is favored, pairs used to solve an older task are modified to optimize the policy on a new task, and performance of that older task decreases, negative forgetting occurs. This type of forgetting takes place because the learner loses the ability to solve the original task at a previously obtained level.

The stability-plasticity dilemma is a trade-off that demands to have a balance between being able

to perform previous learned tasks while having the ability to learn new ones. Coping with the dilemma aims at maintaining or improving an old task's performance while maximizing the performance of a new task.

Catastrophic Forgetting

Catastrophic forgetting (also known as *catastrophic interference* and the *serial learning problem*) is the stability-plasticity dilemma applied to Neural Networks [24]. This type of forgetting occurs when a neural network has been trained on one set of inputs to locate the ideal activation weights and is then given a new set of inputs to train on. This change in inputs can cause the loss of information gained in the weight optimization phase for the first set of inputs when the new input is trained on. If the inputs represent a task the neural network is learning, a change in task can cause the ability to perform the old task to be completely forgotten while a new task (the new inputs) is trained on, hence the term catastrophic forgetting [50, 14].

The occurrence of catastrophic forgetting is due to how the learning process manipulates the neural network to optimize its activation weights. Back-propagation is a standard mechanism for adjusting connection weights in neural networks by applying propagating gradient-descent based output corrections backwards in the neural network, starting at its output nodes. When a neural network receives a new input pattern and a correlating target output pattern to learn, back-propagation has the ability to wipe out all weights optimized on a previously learned set of inputs. The catastrophic aspect comes from the weight adjustment having the ability to completely remove all information used by the original input pattern [14].

Catastrophic forgetting is a reason why many learning systems that use neural networks combine training inputs as a single entity [47]. By treating all of the training inputs as one complete training set to learn, the shift in input data does not occur and the negative effects of catastrophic forgetting

is reduced if not removed completely. The case of catastrophic forgetting when the inputs cannot be combined into a single has spawned many different works proposing ways to mitigate its negative effects. Works with major contributions to this type of forgetting will be discussed in the coming sections.

Methods of Balancing Stability and Plasticity

The impact of the stability-plasticity dilemma on learning systems is great. If the dilemma is ignored during design, a learning system can be susceptible to two extremes: 1) possessing instability of old tasks while learning new ones, or 2) only being able to effectively learn to perform one task and never adapt to new ones. Because the stability-plasticity dilemma is a major issue that many learning approaches must consider, many approaches have been proposed to produce a learner that is capable of balancing stability and plasticity. This section analyzes the commonly accepted approaches of incremental learning, rehearsal methods, complementary learning systems, and deletion strategies. Each approach uniquely preserves relevant learned data while learning new information.

Incremental Learning

Incremental learning is the process of gradually learning new information. The process allows the learner to adapt to new data (tasks, environment, etc.) at a pace that seeks to preserve what has been learned previously. The process makes small changes to the policy as it adapts to each new training set. Learning with decomposed tasks and gradually training the agent on a more difficult or challenging training set are examples of incremental learning.

The process has been used to cope with the stability-plasticity dilemma by reducing the drastic

shift in what is being learned. The assumption is that if an agent is given small changes to cope with in phases, then it will be able to adapt its policy accordingly as it learns new things. In the case of tasks, by gradually learning new subtasks or more difficult aspects of the task, the agent will not make wholesale changes to its policy as the policy adapts to new training sets.

Incremental evolution is a type of incremental learning as it combines the gradual learning process with neuro-evolution [16]. In approaches with incremental evolution, agents are trained to learn a complex task gradually through the use of evolving neural networks. Gomez and Miikkulainen outline their incremental evolution method by decomposing a task into smaller subtasks and training the agent on each subtask until a performance threshold has been reached. With each passing phase of learning, the thresholds gradually become more difficult to achieve.

Incremental reuse is another form of incremental learning. In this approach, agents learn to solve complex tasks by first learning to solve subtask decompositions. The solutions of the subtasks are then reused by applying the solutions to tackle a more difficult task. For example, a learning system with incremental reuse can produce functions to solve a series of small tasks. The outputted functions can then be selected to be used to try to solve a more difficult task with a learning technique. Automatically-defined functions (ADFs) are an example of incremental reuse in play [22]. The work in [22] compares incremental reuse of ADFs with monolithic genetic programming approaches for generating keep-away soccer playing agents. The work shows that agents can learn in phases of first learning to master subtasks and reapplying the learned solutions to the a more a difficult task.

Rehearsal

Rehearsing is a technique that retrains an agent on previously learned tasks during or after a new task is being learned. The technique is used to preserve and strengthen old information when new

information is learned. Because it is impractical for all learning agents to retain all information to handle new and old data, rehearsing is a technique used to keep old information active in memory and balance the ability to learn and retain new and old information [50]. By revisiting old information, the agent is given a chance to repair any lost or damaged information caused by the process of learning new information.

Interleaved learning [50], simple rehearsal [47], and pseudorehearsal [46] are rehearsing techniques used incorporate old training datasets with new ones to balance stability and plasticity.

Interleaved learning and simple rehearsal are basic rehearsal techniques used to refresh an agent's policy through simple repetition of old tasks. *Interleaved learning* is a form of rehearsing where old datasets are relearned while new datasets are trained on. Interleaved learning weaves new and old data together to produce learners capable of supporting new and old information. Interleaved learning differs from *focused learning* because in focused learning, the agent is given only new information to learn. Although, in focused learning, the agent can be given the same new information to learn repeatedly in a rehearsal fashion, old information is not reintroduced to the learner [14].

Simple rehearsal is the repetition of pushing old inputs through neural networks to refresh the network with what was previously learned. Although both rehearsal techniques of interleaved learning and simple rehearsal are basic, they both can be effective of keeping old information active within an agent.

The idea of *pseudorehearsal* is spawned from the problem that simple rehearsal methods are not always practical because old patterns are not guaranteed to be available to refresh a neural network. Old inputs may not always be available because they become outdated, forgotten, or deleted due to storage limitations [47]. Pseudorehearsal creates randomly generated input approximations to fill in the role of old inputs to refresh the network with. When a neural network goes through a

pseudorehearsal, it trains the network by interleaving new input data with these random approximations when old inputs are not available. The process of pseudorehearsal is described more in detail in the discussion of complementary learning systems.

Complementary Learning Systems

In order to produce more effective learning systems, several works have turned to modeling computer learning systems after how the human brain is perceived to learn. One such approach is the idea of *complementary learning systems*, which is modeled after the human short and long-term memory. More specifically, complementary learning systems are modeled after the hippocampus and neocortical portions of the human nervous system. In the human nervous system, the hippocampus handles short-term memory and is in charge of storing new information. The neocortical system is the long-term memory device. It is assumed that after information has resided in the hippocampus for duration of time, it is condensed and transferred to the neocortex for long-term storage [36]. *Consolidation*, the transfer of information from short- to long-term memory, has an unknown trigger and information may be pushed to the neocortex for various reasons.

The separation of memory into hippocampus and neocortex regions in agent learning systems applies directly to the stability-plasticity dilemma. The two-tiered memory structure allows new information to be placed in a separate region from information to solve old inputs; the separate regions create a buffer where new information can be stored without directly affecting old information. Through computer simulation, McClelland, McNaughton, and O'Reilly have shown how a complementary learning system with consolidation causes the learner to gradually adapt to new inputs [36]. Their work provides a complete basis of the complementary learning system.

Because it is impractical to store all information learned in the hippocampus and the neocortex, consolidation schemes must be in place to decide when and what information will be transferred

between the regions. The impracticality comes from memory storage limitations and increased search times for information retrieval existent in real systems [14].

Complementary learning systems have been applied to solving catastrophic forgetting in neural networks. The idea of a complementary learning system is a natural choice for neural networks as both are inspired by the neurological functions of the human brain. Second, introducing a two-tiered memory system to a neural network allows information to be placed in separate short/long-term regions where old information is safeguarded from the deleterious nature of back-propagation. Works that use a two-phased learning system to mitigate catastrophic forgetting include pseudorehearsal [46], pseudo-recurrent Networks [14], using a hippocampus-neocortical configuration with a Hopfield network and Hebbian Learning [19], and Seipone and Bullinaria's work of evolving a two-phased neural network learning system [50].

Robins introduced pseudorehearsal as a technique similar to interleaved learning; instead of mixing new inputs and stored old inputs, pseudorehearsal creates approximates of old inputs that are learned with the new inputs [46]. These new approximates are termed *pseudoitems* and are randomly generated binary vectors that are inputs to the neural network to be optimized. Pseudorehearsal was introduced because old information is not always available to be interleaved with new inputs during rehearsals; information to solve old inputs may have been removed from storage implicitly or explicitly. The process of pseudorehearsal randomly generates pseudoitems when old items are not available. The pseudoitems are then fed into the neural network, using whatever output is generated as the target output for weight refinement. The next step combines the generated pseudoitems and the new inputs to create a new training set for the neural network to adjust weights for. With the use of pseudoitems, pseudorehearsal produces robust networks capable of maintaining information used to solve old inputs while coping with new ones.

French also proposes a learning system that uses approximations to keep important, old informa-

tion valid in a neural network by introducing *pseudo-recurrent networks* [14]. In pseudo-recurrent networks, the neural network contains two storage areas: one for early processing and the other for final storage. The early processing area is based on the hippocampus and stores information gathered from learning new input patterns. The final storage area stores information that has resided in the early processing area of the network and is based on long-term memory.

Pseudopatterns use approximations, termed *pseudopatterns*, to represent old patterns that were previously trained on in the network. The purpose and construction of pseudopatterns are very similar to those of Robins's pseudoitems [46]. Just like pseudoitems, pseudopatterns are randomly generated binary vectors that are fed into the feed-forward neural network and uses back-propagation to optimize its weights. The target output is also generated by capturing the output of the network when a pseudopattern is inputted. Pseudo-recurrent networks create a new training set by interleaving a new input pattern with pseudopatterns then feeding the patterns into the network sequentially. When the network learns the new pattern and the pseudopatterns, it replaces the weights in the final storage area with the ones in the corresponding the early processing area [14].

Although the work in [46], [14], and [13] propose approximation methods and examine the effectiveness of the use of pseudorehearsal and pseudopatterns, a negative of the approach is the amount of error it introduces to the network. In a system tasked with alpha-numeric symbol recognition, a neural network with pseudorehearsal has shown to underperform compared to a system without the approximation technique. The use of interleaving new patterns with pseudoitems introduces too much error in the base population. The error is generated from the random vectors being introduced into the network while learning new patterns. The use of the random approximations train the network on inputs that do not match previously learned patterns and causes too many shifts in weight adjustment. Robins and McCallum conclude that pseudorehearsal performs best when learning is done as a single entity instead of serially inputted into the system.

To address the potential of adding error to the learning process, Hattori's work extends the use of pseudopatterns by interleaving inputs that are new patterns, pseudopatterns, and old patterns [19]. Also using a complementary system with hippocampus and neocortical storage regions, this work's dual-network model extends the conventional use of pseudopatterns by adding a chaotic neural network to the neocortical network. The chaotic neural network is added to retrieve original patterns and not just pseudopatterns from the neocortex. By retrieving original patterns for pattern interleaving, more training inputs closer to previous patterns are available than if just pseudopatterns were used. The rationale is that with more information available from the neocortex, fewer old patterns will be forgotten as new patterns are learned. Experiments show that with the use of chaotic neural networks in the hippocampus, the proposed model outperformed conventional dual-network implementations because original patterns were accessible in the neocortical network.

Complementary learning systems and the use of approximating training inputs are viable solutions to mitigating negative forgetting. Complementary learning systems provide a buffer for the information learning stream. The buffer permits the importance of information to be determined while it resides in the short-term storage before the information instance is discarded or preserved in long-term memory. In addition, the buffer isolates new and old learned information, providing one storage region that guards stability and one designated for plasticity. The process of interleaving approximations with new inputs, such as pseudo-recurrent systems and pseudorehearsal, provides a way to keep old information active in the learning agent.

Deletion Strategies

The use of deletion strategies is another mechanism used in learning systems to cope with the stability-plasticity dilemma. Deletion strategies seek to remove items of knowledge that do not provide positive benefits to the system [54]. These strategies explicitly select which learned infor-

mation, such as state-action pairs, is removed from the agent's memory and policy.

The act of deleting state-action pairs from a policy can be taken reactively or proactively. *Reactive deletion* waits until it is necessary for a pair to be removed, like when an agent's memory storage is full before selecting candidates for deletion. Here, candidate selection can be based on a utility function or chosen at random, but the main purpose is to free storage space for incoming information that otherwise cannot be accepted due to storage constraints. *Proactive deletion* constantly compares learned policies before selecting which candidate will be deleted. In this case, the library does not have to be full before a pair is chosen for removal.

Nakayama and Yoshii classify deletion schemes as passive and active forgetting [43]. *Passive forgetting* removes learned information based on elapsed time. This type of forgetting works at a constant rate, removing information as it expires. *Active forgetting* constantly searches for negative knowledge to target for forgetting at an increased rate than passive forgetting and is similar to proactive deletion.

Several reasons should be noted as why a system would want to incorporate a deletion strategy in its learning process. First, the learning algorithm must be able to cope with the possibility of its storage to become full when new information needs to be captured [26]. If the learner does not have a method of selecting which old information to replace with new information, then the agent cannot learn the new thing. In short, there will be no place for new information to be retained by the learning agent.

Second, even if the storage is considered large enough for newly learned information to constantly be added, the utility problem arises for the learner. The *utility problem* occurs when the effort to search for information in storage outweighs the reward of using the retrieved data [54]. It is well known that the presence of negative knowledge can degrade a system's performance but another hindrance on performance can actually be the time needed to make a decision. If the time needed

to make a decision surpasses the time the decision needs to be made in, the agent can find itself in a different state when the decision is made, requiring a completely different action to be performed than what was selected. The utility problem also shows that the more information an agent has is not always best. Again, if an agent's policy is loaded with state-action pairs, the agent's decision making must spend time traversing the storage for pairs that match the state the agent resides in; typically, the number of items in storage is proportionate to the search time needed to make the decision.

Case-based reasoning systems (CBR) have the overhead that is being balanced in the utility problem. CBR is a learning method where the learner applies data from experienced situations to current ones. Cases are generalized abstractions of experienced states and are applied to similar situations the agent will experience in the future [26]. In case-based systems, the agent must locate stored cases that match the state the agent resides in. The returned cases link the state to the actions the agent will then perform. Deletion strategies make the traversal of the case-base manageable as the storage size grows [54]. Without a deletion strategy for the case-base, retrieval times for cases can become unwieldy.

A third reason for the use of deletion strategies is that the environment and task the agent operates in and on can change. If the agent's domain is in a dynamic environment or is expected to perform multiple tasks, the policy the agent has may become outdated at any point. With a deletion strategy, the policy can remove outdated and under-performing state-action pairs and replace them with new, more effective ones [49].

Environments with drift, such as recommender systems, constantly require the agent to adapt to changing inputs. Drift is the occurrence of something changing gradually overtime. Recommender systems experience a form of drift. In recommender systems, the agent is tasked with making a recommendation based on an inputted profile of a target. For example, a movie rental company

may want to make movie recommendations to its users. In this example, the aim of the learner is to make a recommendation that will be taken by the target user and is generated by a profile the agent creates of the user. Because at any time the users can change preference in movie genres, time periods, casted actors, etc, the learning agent has to cope with the changing demands.

Dynamic environments and changing tasks are two aspects that can affect the agent's policy because the agent must decide what new inputs to accept, which old state-action pairs to remove, and which old pairs to preserve. Solutions to these design questions directly affect how the learning agent copes with the stability-plasticity dilemma.

Although deletion strategies vary in approach drastically, common strategies can be categorized as random, temporal-spatial, utility-based, and hybrids. Each strategy has a criteria for selecting state-action pairs to be removed from the policy so new pairs can be added.

Random Deletion

The random deletion strategy is the most basic method of selecting deletion candidates. The strategy randomly selects state-action pairs and removes them from the policy. Markovitch's work shows that random removal of stored decision making information can improve the effectiveness of the agent and can outperform learners without a forgetting/deletion mechanism in place [34]. The rationale is that random deletion promotes exploration of new solutions and negative knowledge may be removed through the process. McClelland, McNaughton and O'reilly have studied the effects of removing information from memory, called amnesia, and reaffirm the claim that random deletion can help remove non-positive knowledge from a learner [36].

The advantage of using a random deletion strategy is that it is simple to design a learning algorithm around and does not discriminate between new and old state-action pairs being chosen for deletion.

The disadvantage is that because there is no discrimination in a purely random strategy, important, effective pairs can be deleted, requiring the agent to relearn the effective pair.

Temporal-Spatial Deletion

Temporal and spatial deletion strategies are more complex than random deletion. *Temporal deletion* uses time as a utility function when selecting a state-action pair for deletion. The most basic temporal deletion strategy is the use of a queue in order to select which pair to remove. For example, first-in-first-out (FIFO), being the simplest of the queues, removes the oldest pair in the policy when the deletion strategy is invoked.

Time-based decay is another form of temporal deletion that makes the last time a candidate pair has been accessed the criterion for deletion. In the queue-based system, at least one candidate is removed per invocation of the deletion mechanism if the queue is full. In time-based decay, every pair in the policy is given an expiration date as soon as it is added to the policy. Each pair's prominence in the policy declines as time elapse. When the pair is accessed, a new expiration date is given to extend the pair's lifeline.

STAGGER is an incremental learning system that uses time-based decay. It is a learner used to recognize and adapt to state changes in recommender problems [28]. STAGGER applies gradual forgetting, a form of time-based decay, to allow the recommender system to quickly adapt to the changing environment by replacing old information used for making recommendations with new profile information. Gradual forgetting degrades the importance of the learned information by adjusting weights to distinguish prominence over time.

Kernel-based Online Anomaly Detection (KOAD) is another learning system that uses temporal deletion [1]. KOADS is a system used to solve network anomalies of traffic and IP network be-

havior. KOAD's system employs forgetting to remove out-dated information from its storage for anomaly recognition points. The algorithm uses gradual forgetting to slowly decrease the importance of old information as new information constantly flows in.

An advantage to using a temporal deletion strategy, such as time-base decay, is that it removes outdated and underutilized information quickly. The disadvantage is that it removes information without providing cover for what is being lost [49]. Cover is the ability to have a backup for a state whose pair is being removed. If a temporal deletion strategy removes the only pair that matches a state, then the learner will not have a mapping to the actions to perform when that state is reached again.

Spatial deletion is based on locality and can be seen as a direct solution to the cover disadvantage of temporal deletion. Spatial deletion searches for a state-action pair to delete based on redundancy in coverage. If multiple pairs share the same or very similar states and provide cover for each other, then the pairs become candidates for deletion. Salganicoff's DARLING algorithm is based on spatial deletion and sees new information only decay the prominence of their neighbors (pairs whom provide cover each other) [49]. A spatial deletion strategy and locality allows the learning algorithm to preserve pairs that are unique in the states that they support and remove redundant pairs.

Utility-Based Deletion

Utility-based deletion strategies use a performance metric to analyze which state-action pairs should be removed from the policy. Typically, those pairs with the lowest utility value are chosen for deletion. Q-learning is an example of utility-based deletion. In *Q-learning*, the *Q-value* ($Q(s,a)$), of a pair is the utility which reflects the reward expected to gain if the agent performs action a when in state s . Those pairs with the lowest Q-values are then compared to new pairs and

their Q-values. By basing the deletion strategy on Q-values, the learner makes educated estimates on which pairs are more beneficial to retain and which can be replaced.

Evolution algorithms are another example of utility-based deletions. The fitness function provides values that make up individual utility values. As the population evolves, an individual's fitness determines the probability of the individual remaining in the population.

Lieber's work with CBR is another example of a utility-based deletion strategy being used to preserve stability in a learner while exploring new solutions [30]. The work compares two case-bases on a similarity criterion; the aim is to minimize the loss of performance while maximizing the number of cases that can be removed from the case-base. By comparing the two cases-bases on a utility, the case-base with the weakest utility is discarded, while storing the case-base expected to yield the higher performance. The comparison of the two case-bases becomes useful in comparing new and old cases to determine which ones should be accepted and which should be deleted.

In their work that studies how different deletion strategies fare in teaching autonomous robot navigation in a CBR, Kira and Arkin have identified types of interferences used in deletion to be random, performance, recency, frequency, and activation levels [26]. The *performance* metric is the use of a utility function that analyzes how the overall policy or an individual pair performs. Earlier, random deletion was defined as simply the removal of random state-action pairs from a policy. *Recency* and *frequency* are two temporal-based approaches. *Recency* removes pairs based on their last accessed time; the pair that has resided in the policy the longest without being accessed is deleted. *Frequency* bases its deletion on how often a pair is accessed. Those pairs with the lowest access frequency are removed. With this interference, a rule has to be established to prevent new pairs from dominating old pairs. Activation levels reward pairs each time they are accessed. Because there can be different ways to access a pair, different access types can lead to different amounts of rewards being given.

An example of activation levels is in Maes and Brooks' work, where they propose an algorithm to teach behavior-based robots when to perform behaviors [33]. The algorithm uses a binary, immediate positive/negative feedback system for rewards to guide the robot to learn when to activate behaviors. The algorithm allows each behavior to learn in a distributed manner which conditions are met when to activate itself. The system is based on a feedback system that determines relevance and reliability of conditions for activation. Those conditions that are both relevant to the task and consistent in feedback have the greatest chance of activation.

Their work concludes that learning which conditions to activate in a distributed fashion accomplishes the task autonomously. There is no centralized learning component in this distributed behavior-based robot learning algorithm. The activation level deletion strategy is in use by behaviors constantly reevaluating the conditions for activation. Because relevance and reliability are a factor, the behaviors have the ability to adjust to new situations. The algorithm accepts that old pairs can be wiped away by new pairs but relies on quick adjustments to recover.

Smyth and Keane developed a system that uses a utility function to judge the importance of cases in a CBR system [54]. These authors believe that deletion strategies should be designed to consider both maximizing competence of the task and minimizing the case-base size. They categorize cases to be pivotal, auxiliary, supportive, and spanning. *Auxiliary* cases do not contribute to the competence of the agent against the task and are the least important. *Pivotal* cases are the most important to task competence, followed by *spanning*, then *support*. With these four classifications, Smyth and Keane propose the Footwork Deletion (FD) and Footwork Utility Deletion (FUD) deletion strategies. In FD, the cases that contribute the least to the learning agent's performance are removed, first searching for auxiliary through pivotal cases. FUD extends FD by using an additional utility function to break ties if multiple candidates within the lowest competency category are located. FD and FUD performed better than random and a simple utility-based deletion strategy.

Hybrid deletion strategies combine multiple techniques when choosing what to delete. They are most commonly used to select which state-action pair to remove when multiple deletion candidates are chosen by a deletion criterion. Symth and Keane’s FD and FUD strategies can be hybrids if the additional tie-breaking strategy is another type of technique from the first [54]. For example, if two auxiliary cases were located and one has to be deleted, a spatial or temporal deletion strategy can be applied to select the case to be removed.

ActSimple is one such hybrid example: it combines temporal deletion with a utility function for deletion selection [12]. ActSimple combines the learning systems of ACT-R and SIMPLE for a learner that uses both time-based decay and inference forgetting, respectively. ACT-R is a trace-based cognitive rule system that deteriorates state-action pairs over time and reactivates their activation levels when accessed. SIMPLE is a learning system that uses inference forgetting in selecting deletion candidates. The combination of the two approaches has produced a learning agent that is capable of outperforming standard SIMPLE, ACT-R, Queue-based forgetting, random forgetting, and learning with no forgetting at all.

Surprisingly, given how wide-spread layered learning has become, until recently, there was no effort investigating stability-plasticity imbalance in the paradigm.

Preliminary Work

Preliminary work that explored the possibility of using layered learning with layer overlapping uncovered a potential stability-plasticity imbalance that negatively affected the agents’ learning ability [39]. The goal of the work was to provide a performance comparison between agents developed by layered learning with overlapping against a standard monolithic technique. The objective was to use a predator-prey scenario to gage performance on agent policies developed

with layered learning to locate a solution where the predator team could capture its prey as quickly as possible. The produced policies were compared to others developed with monolithic techniques that trained the predator team based on a single subtask's performance. The effort provided the first rough evidence of an imbalance and motivated the deeper study provided by this thesis.

For the study, Pac-Clone, a PacMan-type simulator, was used to provide the training environment and scenario the agents learned in. Layered learning was tasked with training autonomous, *non-playable characters* (NPCs) to capture a hand-coded prey as it moves through a 2D environment.

Approach

The approach used a genetic algorithm to search for the optimum set of three policies that would guide a team of three predators to capture their prey. The genetic algorithm's population consists of individuals that represent a set of three scripts that make up a predator-team's policy for solving the task. The layered learning approach decomposed the capture PacMan task into three layers where each layer used the genetic algorithm to manipulate a population of team policies with the genetic operators of cross-over and mutation; individuals were evaluated based on the performance function of the current layer. Performance evaluation was determined by running each individual in the population in the simulator where each team of predators was given 10 attempts to capture its prey for each generation. The first layer trained the ghosts to capture PacMan effectively and based its performance function on the total number of captures. The second layer sought to perform the task efficiently by rewarding the agent on minimizing the time required in each training game. The final layer attempted to perform the task effectively and efficiently by combining the performance functions of the first two layers into one aggregate function. Layer overlapping was present because each layer could modify any pair in the policy, potentially changing a pair shared between multiple layers.

Results

Experiments compared the policies generated by layered learning and monolithic genetic algorithms by analyzing the techniques' performance with each subtask. Each of the monolithic approaches used one of the performance functions used in the proposed layered learning approach and their genetic algorithms used the same parameters as their layered learning counter-part.

Although layered learning's performance was comparable to the monolithic techniques, it does not clearly outperform any of the monolithic policies as might be expected. One cause of layered learning's inability to outperform the monolithic techniques was the adjustment each individual had to make while transition from layer to layer. As evident in the performance decreases as a new layer is trained on, the agent's policy required computation-time to adjust to the new subtask it was learning. These experiments show that new layers drive out state-action pairs used to solve old subtasks to accommodate pairs that solve the new subtask: an imbalance that favors plasticity. Performance of the older subtasks noticeably decreased as new layers were transitioned to and continued to decrease over-time while new subtasks were optimized on, as evident in Figure 2.

The use of the aggregate layer repaired forgotten information from the first layer but used critical learning time it could have spent focusing on optimizing the policies on its subtask to do so. The significance of this work was that it led to the suspicion that layered learning with overlapping struggles with balancing stability and plasticity and displayed that learning performance can be hindered because of the possible imbalance. Because this preliminary work does not explicitly investigate the role of stability and plasticity and nor does it propose any solutions to ill effects of an imbalance, further study of the dilemma in layered learning is necessary. This dissertation investigates the suspicion and fills the void of uncertainty by demonstrating that an imbalance can easily occur in the paradigm, identify causes and effects, and proposes solutions.

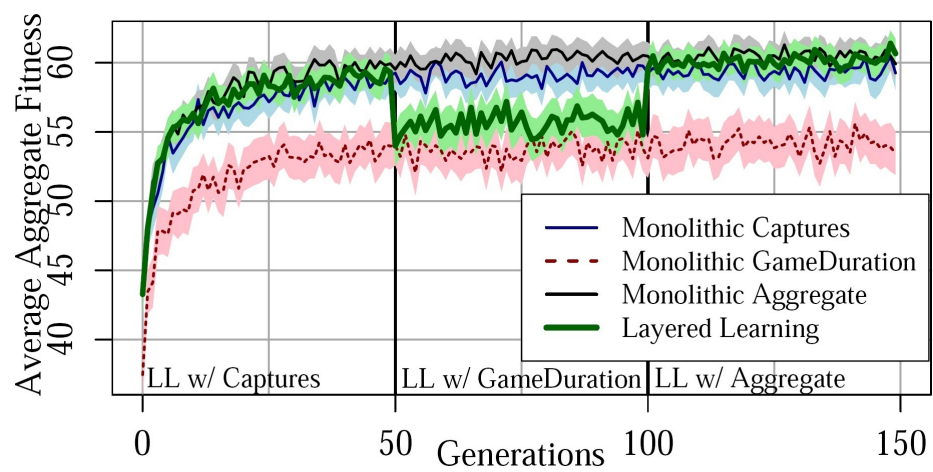


Figure 2.3: Average Aggregate Fitness for Monolithic and Layered Learning Approaches [39]

CHAPTER 3: DEMONSTRATION OF IMBALANCE IN LAYERED LEARNING

In order to show first-hand that a performance affecting problem exists in layered learning, we demonstrate that a stability-plasticity imbalance can exist in the technique. Using two distinct problem domains, an abstract set of Boolean-logic problems and a collaborative multi-agent, direct-policy search problem, we show how layered learning can easily favor plasticity, analyze performance loss and hindrance, and finally identify major causes for the imbalance and their negative effects.

Metrics for the Identification and Classification of Forgetting

Before providing the demonstration, we discuss an existing set of metrics and introduce two new ones that together identify, classify, and quantify the type of forgetting that takes place during direct-policy search in reinforcement learning. Determining which metrics are most effective is integral to our study of the effects of an imbalance. Effective metrics provide analytical tools for understanding performance changes that are correlated with policy modification.

Performance-Based Metrics

Because the classification of negative, neutral, and positive forgetting are performance-based, let us first discuss commonly used metrics that quantify performance changes. These metrics include performance difference, performance ratio, knowledge payoff, transfer ratio, transfer regret, and calibrated transfer ratio.

The simplest policy comparison measures are performance difference and ratio, which directly compare a policy's (P) performance with the policy after some knowledge has been forgotten (P'). *Performance difference (PD)*, $f(x)$, is the difference between the post-forgetting policy's performance $g(P')$ and the pre-forgetting policy performance $g(P)$. If $f(x)$ is positive, positive forgetting has taken place; if $f(x)$ is negative, negative forgetting has occurred; if $f(x)$ is zero, neutral forgetting has happened because there was no change in performance, although the policy has changed. PD is defined in Equation 3.1.

$$f(x) = g(P') - g(P) \quad (3.1)$$

Performance ratio (PR) is the quotient of the performance of P and P' s. If $f(x)$ is greater than one, then positive forgetting has happened; if $f(x)$ is less than one, negative forgetting has occurred; if $f(x)$ is 1, then both performance values are the same and neutral forgetting has taken place. PR is defined in Equation 3.2.

$$f(x) = \frac{g(P')}{g(P)} \quad (3.2)$$

Markovitchs and Scott's [34] economics of learning also measures the value of knowledge. In their approach, the payoff of learning is measured, where payoff can be positive, negative, or neutral and indicates the effect missing knowledge has on a policy. Payoff, $f(x)$ defined in Equation 3.4, is calculated by taking the difference between two benefits, which are two separate policies, solving the same task.

$$B_i = Q_i - P_i \quad (3.3)$$

A benefit (B_i), defined in Equation 3.3, for one policy is the difference between the quality of the solution (Q_i) and the cost of solving the problem (P_i).

$$f(x) = B_j - B_i; 0 < i < j \quad (3.4)$$

Here, we let B_i represent the benefit of control policy P and B_j represents the benefit of the modified policy P' where some knowledge x is removed from P . If $f(x)$ is positive, disadvantageous knowledge has been forgotten from the policy P ; if $f(x)$ is negative, beneficial knowledge has been acquired; if $f(x)$ is zero, neutral forgetting has happened and neutral knowledge was forgotten because there was no change in performance although the policy was modified.

In Gorski and Laird's [17] work on transfer learning metrics, *transfer ratio*, *transfer regret*, *calibrated transfer ratio (CTR)*, and *average relative reduction (ARR)* are examined for validity in comparing learning performances. These metrics determine if an experimental policy, one that has learned a new behavior from an old behavior, outperforms a controlled policy. These metrics are based on the overall task performance, using the difference of the area below the performance curve as the foundation of each metric.

The simplest of the four performance-change metrics in Gorski and Laird's work and the most similar to performance ratio is transfer ratio. *Transfer ratio*, defined in Equation 3.5 and used by Morrison et al. [42], is the ratio of the area under the experimental policy's performance from time 0 to time t over the area of the control's learning curve in the same time range.

$$\frac{\int_0^t f_E(t) dt}{\int_0^t f_C(t) dt} \quad (3.5)$$

Unlike simple performance ratio, transfer ratio considers the entire learning curve of the two compared policies. Below is the transfer ratio equation, where f_E is the experimental policy's learning curve and f_C is the control policy's learning curve.

Each of the metrics Gorski and Laird tested has their strengths and weakness. For instance, CTR and ARR have meaningful interpretations, while transfer regret and CTR provided the most consistent results. Each metric is "unitless" (not measured in domain specific units) and statistically

testable. This work described transfer ratio because of its simplicity in calculating performance comparison and because it directly inspired metrics proposed in this dissertation, which are described in upcoming sections. With that said, details on each of the transfer learning metrics are provided in Gorski and Laird’s [17] work.

Although the metrics above do not form a comprehensive list of all performance measures in transfer learning, they, in their entirety or in a modified form, represent common methods of comparing policy performance change. An overlooked issue with such performance measurements is that they do not capture an entire policy change that occurs in decomposition-based approaches; instead they capture the difference of only a single performance criterion: the overall task the policy tries to solve.

In decomposition-based approaches, a task is decomposed into subtasks that the policy must learn to perform to solve the overall task. Although these single criterion measurements are adequate for monolithic learning, the learning of a task without decomposition, they lack the ability to analyze the changes in subtask performance caused by a policy modification in decomposition-based approaches. Tracking forgotten subtask-specific knowledge is significant because it can identify instances of when beneficial knowledge is lost and when learning at the subtask level stagnates or declines. The significance is heightened when the subtask performance is a necessary component to the overall task but the evaluation of the task does not explicitly measure proficiency of subtasks. In this case, a single criterion-based forgetting metric only considers overall task performance changes and can overlook the loss of beneficial knowledge or the acquisition of negative knowledge used by important subtasks. Additionally, measuring performance changes only at the task level makes it difficult to determine the magnitude the lost knowledge has on task performance. Again, if subtask proficiency is an integral part of task performance, then measuring the impact of lost proficiency can be inaccurate if the task evaluation does not explicitly consider the importance of each subtask.

Forgetting Metrics

To capture these missing subtask performance changes, we introduced two new metrics: the *direct* and *maximized forgetting* metrics [40]. Both metrics classify which type of forgetting occurs at any point of the learning process and quantifies the magnitude of that diagnosed classification at different levels of fidelity. Furthermore, the introduced forgetting measurements can be used by a range of existing monolithic- and decomposition-based approaches to identify performance changes due to forgetting. The main distinction of the metrics is their increased level of fidelity in diagnosing and measuring forgetting because of their consideration of subtask performance changes.

Both metrics are purely performance-based. They compare performance of each subtask with its corresponding best observed performance to determine what effect the policy change has on each component of the task being learned. Because these metrics do not detect policy change, they are intended to be used when state-action pair mappings change is suspected.

The difference between these proposed metrics and the others mentioned earlier is that the proposed metrics explicitly factor in changes to all subtask performances instead of only the performance of the one, overall task. By considering subtask performance changes, each metric serves as an indicator of when a policy loses or gains performance for individual subtasks. With that said, the proposed metrics are heavily influenced by Markovitch and Scott’s [34] measure of the value of knowledge and the metric transfer ratio, examined by Gorski and Laird [17].

Before describing the proposed metrics, a few definitions must be made that are common for each method: a *policy* maybe modified at time-step t . At time-step t , *the policy’s* performance of each subtask is retrieved from the function $p(s_i, t)$, where s_i is a subtask in the set of all subtasks S used to perform task T . In addition, function p returns the performance measure of task T at time-

step t with the parameters of $p(T, t)$. Function p returns the real value ratio of performance to the optimum and is bounded to inclusively range from 0 to 1, where 1 represents optimal task or subtask performance and 0 represents the converse.

For each subtask s_i , there is a corresponding weight w_i . Also, there exists a weight for the task, w_t . Each weight is a real value inclusively ranging from 0 to 1, and the sum of all of the weights equals to 1. The weights determine the importance each subtask has on task performance and is defined by the developer. The weights also determine how forgiving the system should be for losing particular knowledge.

Direct Forgetting Metric (DFM): This metric calculates a direct difference between two policies using the weighted sum of subtask performances and is defined in Equation 3.6. For simplicity, we will use policies at time-steps t and $t-1$ as the immediate policies that will be directly compared. By calculating subtask PD from these two time-steps, $f(t)$ makes a direct comparison between a policy and its immediate change in the next time-steps to determine the significance of the knowledge that was lost or modified.

$$f(t) = \sum_{i=1}^{|S|} w_i (p(s_i, t) - p(s_i, t-1)) \quad (3.6)$$

Maximized Forgetting Metric (MFM): This metric is represented as $g(t)$ and is the weighted sum of the difference of $p(s_i, t)$ and the best performance of s_i from time 0 to $t-1$. MFM is defined in Equation 3.7. This measure utilizes the *max* function that retrieves the best performance of policy P on subtask s_i up to the time t . By calculating this difference, the entire performance history of each subtask is factored into the forgetting measure.

$$g(t) = \sum_{i=1}^{|S|} w_i \left(p(s_i, t) - \max_{t' \in \{0, \dots, t-1\}} \{p(s_i, t')\} \right) \quad (3.7)$$

If $f(t)$ or $g(t)$ return a positive value, then positive forgetting has occurred at time t . If the returned value is 0, then neutral forgetting has occurred. If $f(t)$ or $g(t)$ is less than 0, then negative forgetting has taken place. The magnitude of the occurred forgetting is represented by the returned value. For instance, if the returned value is negative, the smaller the value, the higher the negative forgetting magnitude is and denotes how much performance has suffered because of the lost or modified knowledge. Consequently, if the returned value is positive, the larger the number, the stronger performance has improved with the forgotten knowledge.

Through experimentation using layered learning to solve Boolean-logic, discussed later in the imbalance demonstration section, DFM and MFM were validated and verified to accurately identify the three classifications of forgetting and quantify their magnitudes [40].

Policy-Based Metrics

To further understand what happens to a policy during the layered learning process, a five quantitative policy-based metrics were used. These metrics coupled with the performance-based metrics provide deeper insight in how the learning process is affected by measuring the two factors of the forgetting classification: policy and performance change.

Pair Changes: Pair changes is the total number of state-action pairs that are different between two versions of a policy, a primary (P) and a secondary policy (P'). This metric only tallies states (s) that are in the primary that do not map to the same action (a) in the secondary policy.

Unchanged Pairs: The unchanged pairs metric is the converse of the number of pair changes. It is the number of states in the primary policy that map to the same actions as the secondary policy. This metric tracks how many pairs remain the same as a new subtask is optimized on.

New Pairs: This metric is the sum of new pairs introduced to the policy between different policy

versions. Assuming a policy never deletes a state and only modifies action mappings, which is followed in all of the experiments in this dissertation, the number of introduced pairs is the difference in total pairs between two policies. By tracking this statistic, we can see how many new states an agent reaches and how much a policy has grown while learning a new subtask.

Permanently Damaged Pairs: A permanently damaged pair is one that resides in a policy once a targeted layer is complete, has been changed in a subsequent layer, and never reverted back to the original mapping in the targeted layer. This statistic is gathered for each non-terminal layer and indicates how destructive a successive layer can be on a policy previously optimized on another subtask.

Repaired Pairs: A repaired pair is the converse of a permanently damaged pair and is a pair that was reverted back to the targeted layer’s state-action mapping. This statistic indicates if a layer relearned a forgotten pair and if the system retained that remapping until the end of the learning process. A high number of repaired pairs means although pair destruction took place, a large amount relearning and time spent on reverting forgotten pairs occurred.

In addition to the forgetting metrics and these policy-based gathered statistics, we measure performance of each subtask throughout learning. This simple statistic is the most powerful to our demonstration because it shows how proficiency of each subtask changes during a new subtask’s optimization and on layer transitions. Because an agent is actively learning in all of our experiments, we expect the policy to constantly experience changes. By tracking each subtask’s performance at each policy change, we are able to see the impact on previously achieved proficiency and diagnose if the learner has a clear preference for stability, plasticity, or balance.

Imbalance Demonstration

To demonstrate the susceptibility of a stability-plasticity imbalance in layered learning and that critical knowledge used to perform a task can be lost in a decomposition-based learning system, a series of Boolean-logic and collaborative multi-agent, direct-policy search problem experiments were performed. The experiments were design to demonstrate the ease at which an imbalance can occur in layered learning and identify the imbalance’s causes and effects. In these two different problem domains, we show that as the performance evaluation functions transitions from one sub-task to another, the policy experiences drastic changes to its state-action pairs, subtask and task performance degrades, and computational effort is wasted when a policy attempts to regain lost proficiency.

Boolean-Logic Problems

Two variants of Leading Ones Trailing Zeros (*LOTZ*) serve as the Boolean-logic problems used in the first imbalance demonstration. These problems were chosen because they are clearly defined and their problem properties are easily understood and constructed for analysis purposes. While they are not strictly speaking RL problems, they are analogous to real problems faced by such algorithms. First, each Boolean-logic problem represents a task the agent is given to learn to perform. Mastering a task is represented by the bit string’s transformation into an optimal solution as indicated by the received maximum performance score of the given problem. Each bit in the string symbolizes state-actions pairs (knowledge) used by an RL agent to make decisions, thus making the bit string a representative of an agent’s policy in RL. These representations and transformations are akin to a policy being modified to contain state-action pairs that guide the agent into making decisions to perform a task in a RL system. Also, the construction and destruction of behavioral knowledge is simulated by the toggling of bits. When a bit that positively affected subtask perfor-

mance proficiency is toggled and no longer has such an effect, *negative forgetting* has occurred. *Positive forgetting* is the converse reaction if performance increased after a toggle and is essentially learning. *Neutral forgetting* occurs when no performance change happens after a bit string change.

Leading Ones Trailing Zeros

To evaluate a bit string's optimality, a different performance metric is used for the two Boolean-logic problems and their subtask decompositions. For commonality among the problems, each performance metric is scaled to a value ranging from 0 to 1, where 1 is optimal. This problem is a representative of simple tasks where individual decisions contribute linearly to overall performance.

Leading Ones Trailing Zeros (LOTZ) is a task where performance is rewarded based on the lengths of an all-one string prefix and an all-zero suffix in the bit string. In these LOTZ experiments, two variations are examined, *Leading Ones Times Trailing Zeros (LO*TZ)* and *Leading Ones Plus Trailing Zeros (LO+TZ)*. There are two components to these problems: *leading ones (LO)* and *trailing zeros (TZ)*. The performance functions for LO and TZ are defined in Equation 3.8 and 3.9, respectively. The LO*TZ is defined in Equation 3.10 and LO+TZ in Equation 3.11. The LOTZ problems provide means of exploring extreme situations where some subtask solutions involve the opposite decisions as another and where the final task is either a linear or non-linear balance between these.

$$x \in \{0, 1\}^n, LO(x) = \sum_{i=1}^n \prod_{j=1}^i x_j \quad (3.8)$$

$$x \in \{0, 1\}^n, TZ(x) = \sum_{i=1}^n \prod_{j=1}^i (1 - x_{n-j+1}) \quad (3.9)$$

$$x \in \{0, 1\}^n, LO * TZ(x) = \frac{4}{n^2} (LO(x) \cdot TZ(x)) \quad (3.10)$$

$$x \in \{0, 1\}^n, LO + TZ(x) = \frac{1}{n} (LO(x) + TZ(x)) \quad (3.11)$$

Because the LO**TZ* performance function calls for the product of leading ones and trailing zeros, there is only one optimal bit string if the size of the string is even that solves the task completely and returns an LO**TZ* value of 1. When the number of bits is odd, there are two optimal solutions: each solution has a split between all-ones and all-zeros as even as possible where the total equals the number of bits. LO+*TZ*, on the other hand, has n optimal solutions, at 2-hamming distance intervals.

These two LOTZ variants were selected because of the conflicting bit string requirements the LO and TZ subtasks have between each other; the LO subtask requires the all-ones string, while TZ needs the all-zeros string. The decompositions with LO and TZ subtasks following one another forces one subtask to drive out information acquired to optimally perform the preceding subtask. This occurrence of *drive* is used to forcibly simulate forgetting while the agent learns to perform subtasks vital for complex task execution.

To be clear: these problems are constructed to create a situation in which forgetting will almost certainly occur. This provides an opportunity to demonstrate the stability-plasticity imbalance. For more realistic problems, we will not have such foreknowledge and will need to use the metrics previously discussed for diagnosis.

Experiment Description

To transform a randomly generated bit string into an optimal solution, layered learning was paired with a $(1+1)$ *Evolutionary Algorithm* [(1+1) EA]. A (1+1) EA is a simplified type of EA where one genetic operator is used to manipulate the only individual in the population. Borisovsky and Ereemeev [3] and Wegener and Witt [59] studied (1+1) EA performance and provide additional details on the approach.

In this work's EA, a single child is cloned from its parent and modified via the mutation operator. The child is then evaluated on the current layer's subtask and replaces its parent if it performs at least as well as the parent. The *mutation* operator is the only genetic operator used and functions by toggling each bit at a $1/n$ probability per manipulation phase of the EA. Also, a bit string size of 128 is used because it is large enough to make each Boolean-problem challenging.

There is a pair of test cases for each of the performed experiments, both containing a different layer halting condition. Within each test case is a set of test plans where each plan decomposes a Boolean-logic problem into a sequences of layers. Each test plan is repeated for 100 trials to have a large enough sample to draw significant conclusions.

The process of optimizing a string is as follows: a LOTZ problem is selected as the task to solve. If the task is to be learned monolithically, then layered learning is structured to have one layer where the subtask is the selected Boolean-logic problem. If the task is to be learned with decomposition, then multiple layers are constructed and each layer contains one subtask from Table 3.1. Each layer also has a halting condition, which indicates when the layer is complete and the learner can progress to the next layer. The halting condition is to either run until optimal performance for the layer's subtask is achieved or when a predefined number of time-steps have elapsed. A *time-step* is one complete EA cycle iteration, where the bit string finishes a manipulation and evaluation phase

by going through the cloning, evaluation, and replacement process. The fixed duration halting condition test case was set at 90% of the average number of time-steps required to generate the optimal bit string in the monolithic test plan of maximize subtask performance halting condition.

Table 3.1: LOTZ Task and Subtask Descriptions

Subtask	Maximization Objective Description
LO (Leading Ones)	Length of the all-ones prefix.
TZ (Trailing Zeros)	Length of the all-zeros suffix.
LO+TZ	The sum of LO and TZ (LO+TZ) across the entire string.
LO*TZ	The product of LO and TZ (LO*TZ) across the entire string.
MiddleThirdLO+TZ	LO+TZ over the middle 1/3rd bits.
MiddleTwoThirdLO+TZ	LO+TZ over the middle 2/3rd bits.
MiddleOneThirdLO*TZ	LO*TZ over the middle 1/3rd bits.
MiddleTwoThirdLO*TZ	LO*TZ over the middle 2/3rd bits.

In the monolithic case, once the halting condition has been met for the sole layer, the learning process is complete. For the decomposition case, when one layer has halted, the bit string is then passed to the next layer and is optimized on a new subtask. Once all of the layers have been iterated through, the learning process is complete. In all of the decomposition cases, the complex LOTZ variant, either LO*TZ or LO+TZ, is the final layer's subtask.

LO+TZ Results

Table 3.2 displays the test plans and their subtask sequences for the LO+TZ experiment. Based on the decomposition and the bit string composition requirement of the LO+TZ subtask, the maximize subtask performance halting condition test case resulted in plans 0-2 being monolithic while plans 3-5 were decomposition-based. Plans 0-2 are monolithic because strings optimized on the LO or

TZ subtask are also optimized for the LO+TZ subtask, not needing a single bit toggle in layer 2 to satisfy the final layer's LO+TZ subtask in plans 1 and 2. Table 3.3 displays the average time-steps needed to iterate through each plan's layers using the maximize performance halting condition, standard deviation, and Z-score comparing plans 1-5 to the control test plan, plan 0.

Table 3.2: LO+TZ Test Plans with Subtask Sequences

Test Plan #	Layer 1	Layer 2	Layer 3
Plan0	LO+TZ	-	-
Plan1	LO	LO+TZ	-
Plan2	TZ	LO+TZ	-
Plan3	LO	TZ	LO+TZ
Plan4	TZ	LO	LO+TZ
Plan5	MiddleThirdLO+TZ	MiddleTwoThirdLO+TZ	LO+TZ

Table 3.3: Average Time-Steps to Find Optimal LO+TZ Bit String

Test Plan	Time-steps	STDV	Z-Score
Plan0	6,948.13	1,028.799773	-
Plan1	13,809.1	2,050.667762	-29.90
Plan2	14,081.81	2,147.559721	-29.96
Plan3	28,324.04	3,130.869352	-64.86
Plan4	28,458.81	3,354.459956	-61.31
Plan5	12,074.22	1,355.825259	-30.12

From the average time-steps needed to generate optimal subtask performing strings, plan 0 required the least number of time-steps with 6,948.13. Test plan 5 that used a middle-third, lower-order decomposition (12,074.22 time-steps) then followed. Plans 1 and 2, which only used their first layers required 13,809.1 and 14,081.81 time-steps, respectively. Finally, plan 3 averaged 28,324.04 and plan 4 averaged 28,458.81 time-steps. The Z-scores were compared to a critical value of -2.326 based on a Bonferroni adjusted alpha of .01. Plan 0's 6,948.13 average time-steps is significantly less than those of the decomposition-based plans.

The difference between the monolithic approaches was that plan 0 has more permutations of optimal bit string solutions while plan 1 required the all-ones string and plan 2 the all-zero string. Plans 3 and 4 were the worst performers because their first two layers contained subtasks that contradicted each other. For example, in plan 3, after the all-ones string was generated to maximize performance of the LO subtask, the string then needed to be optimized on the all-zero string. Because optimization of the TZ subtask also is an optimized string for LO+TZ, layer three did not need to manipulate any bits for its subtask optimization, halting the learning phase for this plan. The contradiction between the first two layers' subtasks forced the bit string to be optimized twice, hence the averages between plans 2 and 3 being approximately double the time-steps of plans 1 and 2.

Plan 5's results demonstrated that learning lower-ordered versions of the overall task and progressively increase the problem's magnitude can produce competitive results to the type of decompositions exhibited in plan 1 and 2 and plans 3 and 4. By learning LO+TZ in progressively larger chunks from the middle of the bit string outward, the decomposition was able to reduce the solution space into manageable partitions and solve the overall task faster than decompositions with less layers.

The second test case split 6,253 time-steps between all of the layers of each test plan, for the fixed layer duration halting condition. 6,253 time-steps was chosen because it is 90% of the duration needed to reach optimality in the monolithic test-case. In this test case, plans 1 and 2 are no longer considered monolithic and plans 3 and 4 manipulate the bit string in all three of their layers. The change to plans 1-4 is because each layer does not meet optimal subtask performance, resulting in the LO+TZ layer always requiring evaluations and bit modification.

A second major change is in plan performance. Plan 5, which solved LO+TZ using the middle-based decomposition, became the worst average LO+TZ performer, producing a LO+TZ perfor-

mance score of 0.4316. The monolithic control of plan 0 produced an average LO+TZ performance score of 0.9275 at the end of the 6,253 time-steps. All average LO+TZ performance scores, standard deviations, and Z-scores comparing plans 1-5 to plan 0 are found in Table 3.4. Z-tests confirm plan 0 outperform the decomposition-based plans, using a Bonferroni adjusted $\alpha = 0.01$ and critical value of 2.326. The right-tailed test rejected the null hypotheses that plan 0's average LO+TZ performance was less than or equal to the averages of plans 1-4. Thus, the monolithic test plan's LO+TZ performance average was significantly greater than those of the decomposition-based approaches.

Table 3.5 displays the fixed duration halting condition's individual subtask performances at the end of each layer. From this table, it is noted that the first subtask performance decreases during a transition from layer 1 to 2 in test plans 3 and 4. The decrease in LO and TZ performance is attributed to the contradictory subtask of their respected second layer. As an example, consider plan 3's LO subtask. At the end of layer 1, the LO average performance is 0.239, resulting in approximately the first 30 bits being set to 1. After the second layer, which is the TZ subtask, LO performance decreases to .007; this means, that on average, the all-ones prefix is of length 1. Although the TZ subtask is a part of plan 3's decomposition of LO+TZ, the TZ layer was disastrous to LO subtask performance, affecting the first 30 bits negatively.

Table 3.4: Average LO+TZ Performance with Fixed Layer Duration

Test Plan	LO+TZ Perf.	STDV	Z-Score
Plan0	0.9275	0.072087661	-
Plan1	0.772421875	0.088247747	13.61
Plan2	0.787890625	0.092954798	11.87
Plan3	0.58375	0.078879791	32.17
Plan4	0.604453125	0.094654846	27.15
Plan5	0.431640625	0.081312889	45.63

Table 3.5: LO+TZ: Average Subtask Performances per Layer’s Completion

Test Plan #	Subtask	Layer 1	Layer 2	Layer 3
Plan0	LO+TZ	0.9275	-	-
Plan1	LO	0.336328	0.541641	-
Plan1	LO+TZ	0.345234	0.772422	-
Plan2	TZ	0.338438	0.558438	-
Plan2	LO+TZ	0.346172	0.787891	-
Plan3	LO	0.239297	0.006641	0.17875
Plan3	TZ	0.006719	0.234063	0.405
Plan3	LO+TZ	0.246016	0.240703	0.58375
Plan4	TZ	0.233359	0.008984	0.185859
Plan4	LO	0.006641	0.235391	0.418594
Plan4	LO+TZ	0.24	0.244375	0.604453
Plan5	MiddleThirdLO+TZ	0.986429	0.28381	0.050714
Plan5	MiddleTwoThirdLO+TZ	0.021412	0.608824	0.178941
Plan5	LO+TZ	0.016641	0.018281	0.431641

Because of the fixed duration halting condition, plan 5 suffers similarly to plans 3 and 4. The new halting condition causes the segments of plan 5’s bit string to not be optimized before moving to the next layer. The suboptimal performance of layer 1’s substring combined with the additional bits of layer 2’s subtask means not only does layer 2 have to optimize layer 1’s substring but also has to connect the newly added bits padded to the middle thirds’ substring to maximize the number of leading ones and trailing zeros. If the substring from layer 1 was optimized, layer 2 has to alter the padded bits to the left to maximize the all-ones prefix and generate all zeros for the added bits to the right of layer 1’s substring. If layer 1’s substring was optimized to contain the all ones string, then layer 2 would have to generate the all-ones string for its substring prefix. If layer 1’s substring was optimized on the all-zeros string, then layer 2 would attempt to generate an all-zero suffix. In short, layer transitions and task decomposition caused the decrease in subtask performance experienced by plans 3-5, as displayed in Table 3.5.

These experiments show that when subtasks require radically different behaviors, layered learning

can produce catastrophically poor performance due to a heavy preference for plasticity, even when the overall task requires a balance.

*LO*TZ Results*

Both test cases in the LO*TZ experiment follow similar test plans as the LO+TZ experiments. The only difference is the LO+TZ subtask is replaced with LO*TZ. Table 3.6 displays the LO*TZ test plans with their layer and subtask sequences.

Table 3.6: LO*TZ Test Plans with Subtask Sequences

Test Plan #	Layer 1	Layer 2	Layer 3
Plan0	LO*TZ	-	-
Plan1	LO	LO*TZ	-
Plan2	TZ	LO*TZ	-
Plan3	LO	TZ	LO*TZ
Plan4	TZ	LO	LO*TZ
Plan5	MiddleThirdLO*TZ	MiddleTwoThirdLO*TZ	LO*TZ

Like the LO+TZ experiment, the maximize fitness halting condition test case resulted in the monolithic test plan of plan 0 to outperform all of the other decomposition-based test plans. Here, plan 0 required 9,654.57 time-steps on average to generate the optimal LO*TZ bit string. Plan 5, with its middle-thirds decomposition, produced the next fastest optimal string in 16,316.47 time-steps. The two test plans with two layers, plans 1 and 2, generated the next fastest optimal strings with 23,892.62 and 23,649.88 time-steps, respectively. Plans 3 and 4 required the most time-steps and bit-manipulations, needing 38,010.62 and 37,457.38 time-steps, respectively. Z-tests with the Bonferroni adjusted alpha of 0.01 and a critical value of -2.326 confirm that plan 0's average required time-steps were significantly lower than those of the other plans. All averages, standard deviations, and Z-scores for the first test case are provided in Table 3.7.

Table 3.7: Average Time-Steps to Find Optimal LO*TZ Bit String

Test Plan	Time-steps	STDV	Z-Score
Plan0	9,654.57	2,766.02302	-
Plan1	23,892.62	3,543.243536	-31.67
Plan2	23,649.88	3,366.461891	-32.12
Plan3	38,010.62	3,750.248397	-60.85
Plan4	37,457.38	4,260.572191	-54.73
Plan5	16,316.47	3,464.359091	-15.03

The LO*TZ fixed duration halting condition test case produced a slightly different outcome to the LO+TZ experiment. Limited to 8,689 time-steps, plan 0 generated bit strings that performed LO*TZ with an average score of 0.981. Plans 1 and 2 produced the next best LO*TZ performers with average scores of 0.616 and 0.619, respectively. The slight difference between fixed duration LO+TZ and LO*TZ is in the similar, lowest scores of the final 3 plans: plans 3, 4, and 5 averaged 0.381, 0.367, and 0.369 LO*TZ values, respectively.

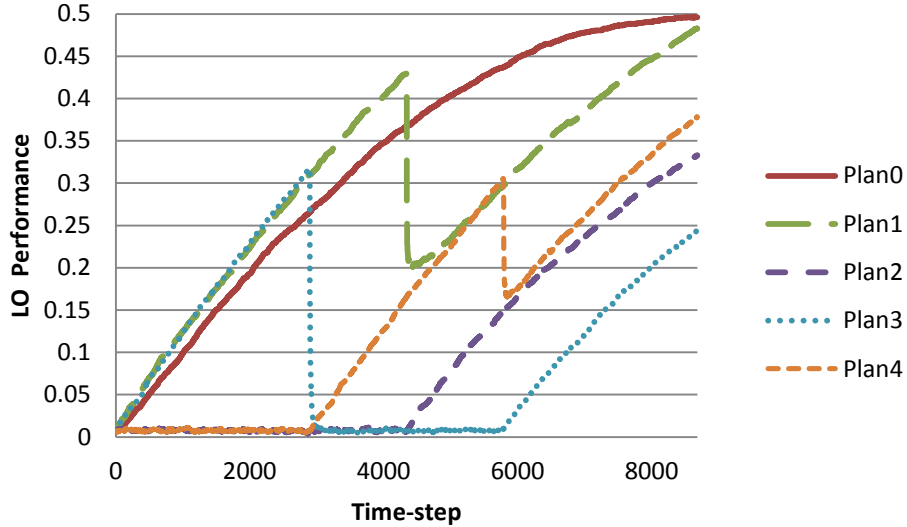


Figure 3.1: LO subtask performance while learning LO*TZ with a fixed layer duration.

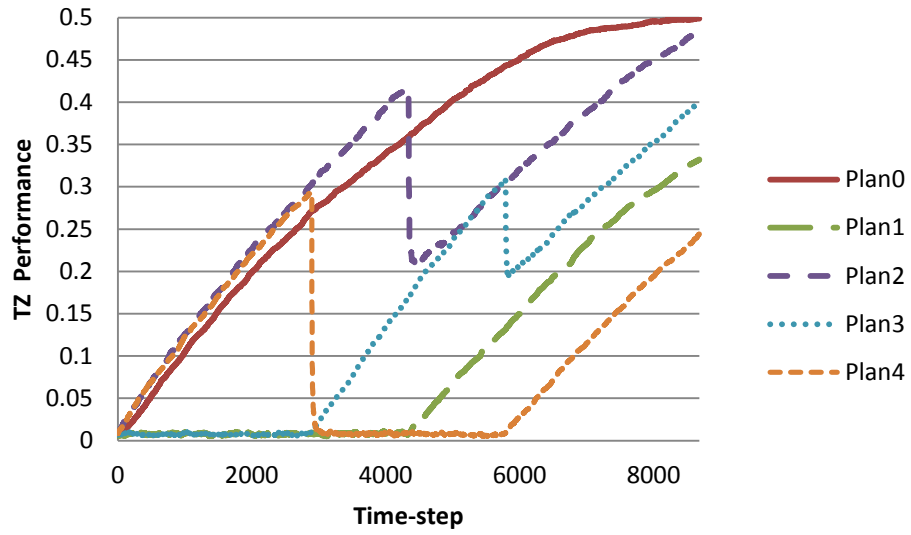


Figure 3.2: TZ subtask performance while learning LO*TZ with a fixed layer duration.

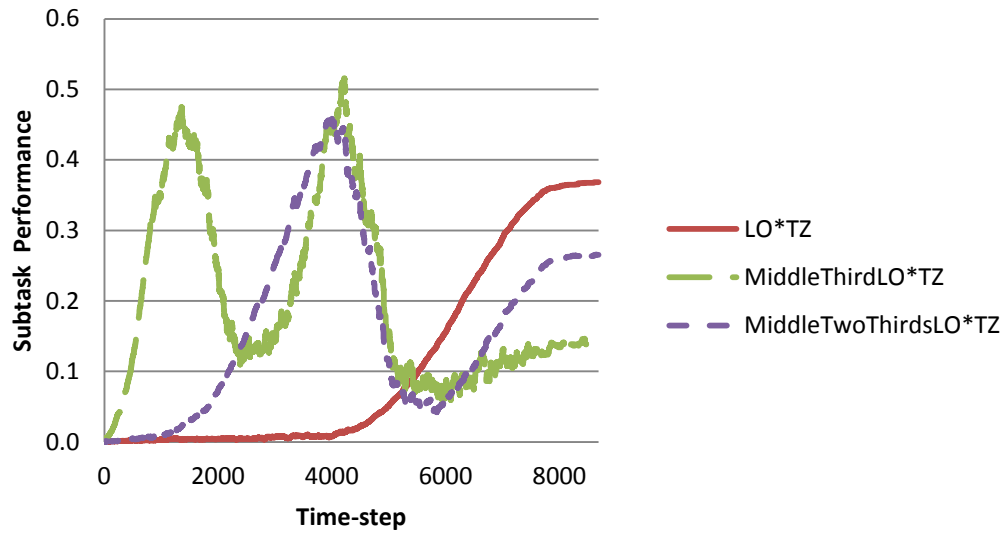


Figure 3.3: Subtask performance while learning Middle Third LO*TZ with a fixed layer duration.

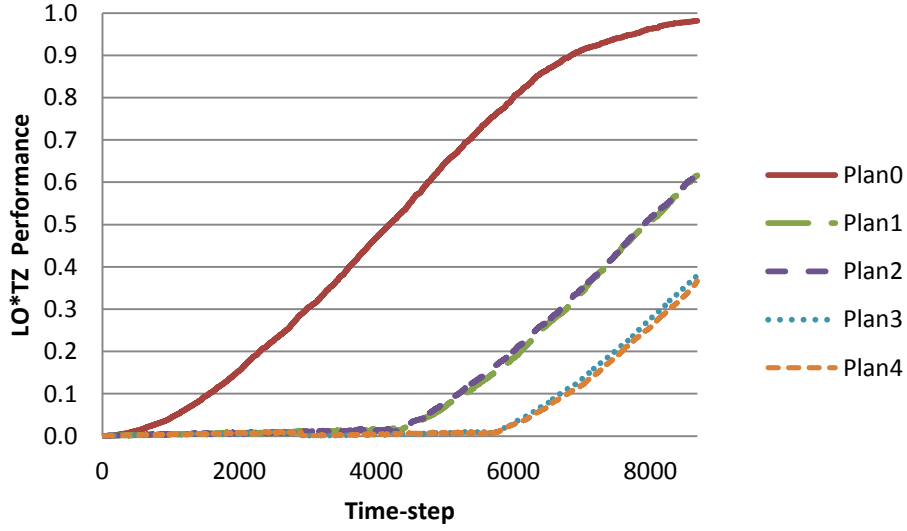


Figure 3.4: LO*TZ performance with each decomposition test plan under a fixed layer duration.

Table 3.8: Average LO*TZ Performance with Fixed Layer Duration

Test Plan	LO*TZ Perf.	STDV	Z-Score
Plan0	0.981740723	0.038980479	-
Plan1	0.61628418	0.16698787	21.31
Plan2	0.61901123	0.174793245	20.25
Plan3	0.380949707	0.140954749	41.08
Plan4	0.366726074	0.154949282	38.49
Plan5	0.368535156	0.152464694	38.97

The reason why plans 3 and 4 produced the lowest LO*TZ scores is because they contained the most layers, layer transitions, and subtask changes in the test case. In the case of plans 3 and 4, the first two layers contained subtasks that contradicted each other, undoing bit changes that affected positive performance. Figure 3.1 shows how plan 3 begins to optimize LO performance until that layer reaches its time-step limit of 2,897, then its subtask performance drops drastically while the bit string is optimized on layer 2's subtask of TZ. Figure 3.2 displays the same extreme

performance decrease in plan 4 at time-step 2,897. In both test plans, the first layer's subtask performance does not increase again until time-step 5,794, when the third layer's subtask of LO*TZ is used to optimize the bit string; recall that LO*TZ relies on a balanced LO prefix and TZ suffix for optimality, unlike LO+TZ. Figure 3.4 displays the average LO*TZ performance over time for plans 0-4. From this graph, it is evident how much faster plan 0 reaches suboptimal performance compared to the other plans and how much plans 3 and 4 suffer from the forced first layer's subtask performance decrease caused in layer 2.

Finally, plan 5 also suffers from having three layers of decomposition. Because the LO*TZ component of MiddleThirdLO*TZ and MiddleTwoThirdsLO*TZ required LO and TZ lengths to be greater than 0 for optimality, the decomposition forces each layers newly expanded substring to toggle all new left bits to be ones and all new right bits to be zeros to connect to and take advantage of the previous layer's LO*TZ performance. Any break between the newly appended bits and those of the previous layer's substring is not rewarded in the performance of LO*TZ. This hurdle makes it difficult for this type of decomposition because newly added bits must be toggled to maximize the LO prefix, maximize the TZ suffix, and make sure the previous layer's substring's LO*TZ performance is maximized and not negatively changed.

Figure 3.3 graphs the difficulty plan 5 had at maintaining the MiddleThirdLO*TZ performance. After the first layer's fixed duration at time-step 2,897, the second layer began to optimize the entire string on the MiddleTwoThirdsLO*TZ subtask and modified bits in the middle one-third of the string, causing MiddleThirdLO*TZ performance to decrease. A similar performance decrease happens to MiddleTwoThirdsLO*TZ in layer 3 as well, when the subtask encompasses the entire bit string for LO*TZ evaluation.

These experiments show that not only can a preference for plasticity create problems for layered learning on complex, non-linear problems with conflicting subtasks, but that performance can be

very sensitive to the order in which subtasks are presented.

Discussion

A number of observations and conclusions were made from these Boolean-logic experiments that demonstrate that layered learning is susceptible to a stability-plasticity imbalance. First, the monolithic approaches outperformed each of their decomposition-based counterparts. When allowed to optimize the bit string until task optimality was reached, the monolithic test plans always generated the optimal solution with the least number of evaluations. When the number of bit string evaluations and modifications were limited by fixing the number of time-steps spent in each layer, the monolithic approach produced a higher overall task performance than all of the tested decompositions.

In addition to executing statistical tests on the performances of each test plan, it has been further verified by Gorski and Laird’s [17] transfer learning metrics that each tested monolithic approach outperforms those based on decomposition. The previously discussed transfer metrics of *transfer ratio* (*T. Ratio*), *transfer regret* (*T. Regret*), and *calibrated transfer ratio* (*Calibrated T. Ratio*) were used in this work to compare a controlled transfer learner’s performance to an experimental, decomposition approach.

Table 3.9 displays the values of each decomposition test plan (the experimental approach) compared to the monolithic control (plan 0) of the fixed duration test case of each Boolean-logic problem. Because each of the transfer ratio values are positive and each of the transfer regret and calibrated transfer ratio values are negative, the transfer learning metrics reveal that the monolithic control produced a higher performing policy than the experimental decomposition-based approaches. This under-performance experienced by the decomposition-based approaches is a reaction to the learning paradigm driving out overall task-critical bits before reaching the final layer,

causing time-steps to be spent repairing ineffective bits.

Table 3.9: Transfer learning metrics comparing each test plan to the monolithic control

	Test Plan	T. Ratio	T. Regret	Calibrated T. Ratio
LO+TZ	1	0.698	-0.181	-0.362
	2	0.719	-0.168	-0.337
	3	0.423	-0.344	-0.690
	4	0.433	-0.338	-0.678
	5	0.241	-0.451	-0.908
LO*TZ	1	0.282	-0.376	-0.761
	2	0.289	-0.372	-0.752
	3	0.121	-0.460	-0.930
	4	0.113	-0.465	-0.939
	5	0.214	-0.412	-0.832

A second conclusion is that the number of overlapping layers in a decomposition affected performance and the rate of learning to perform tasks at an optimal level. It is observed that the more layers in a decomposition, the more policy evaluations and modifications were needed to discover the optimal solution. Additionally, the more layers in a decomposition, the lower the overall task performance. These two claims are derived from the average number of time-steps required to find optimality in the first test case of each experiment being the highest for test plans with 3 layers, second highest for test plans with 2 layers, and the lowest for the monolithic approaches. For test case 2, overall task performance decreased with an increase in the number of layers. Table 3.9 supports this claim with how test plans with 3 subtasks produced the lowest transfer learning values, followed by the test plans with 2 layers.

A third conclusion is that a learned subtask’s performance was affected by changes to the performance evaluation function of a new subtask. This conclusion is made from the observation that subtask performance of a previous layer decreases when the new subtask of the subsequent layer is learned; the most significant performance decreases occurred when the decomposition contained

3 layers. The LO*TZ fixed time duration test case provided the clearest example of this observation. Recall that in test plan 3 the decomposition optimized on LO, TZ, then LO*TZ. Test plan 3's LO performance increased during layer 1's LO optimization, decreased sharply in layer 2's TZ optimization, and increased again in layer 3, shown in Figure 3.1. Conversely, the performance of test plan 3 on the TZ subtask decreased while it learned LO in layer 1, increased during layer 2's optimization of TZ, and slightly decreased in layer 3. From this and the other decomposition test plans from the other experiments, it is concluded that explicitly learned subtask proficiency was negatively impacted when performance evaluation changed to optimize over a new subtask.

A fourth conclusion is that the poor performances noted in the first three conclusions are all caused by a stability-plasticity imbalance that favors knowledge that is being currently rewarded over what was learned previously. More specifically, all of the decomposition-based approaches performing worse than the monolithic controls and the number of layers negatively affecting performance were all caused by the learning system overriding information depended on to perform older subtasks for knowledge used to optimize the policy on a new subtask. In the case of these Boolean-logic problems, the switch to a new performance evaluation function when the layer was changed pressures the bit string to toggle bits to maximize new subtask performance without any regard to any previously learned subtasks. Because of the use of layer overlapping, the priority shift toward the new subtask caused bits that were used for older subtask performance to be negatively affected as bits relied on in one layer were modified in another.

Because these experiments provided a stable and repeatable environment for bit string optimization and subtask evaluation, the coupling of changes to the bit string and the observed changes to subtask performance makes it clear that negative forgetting can occur in layered learning. Additionally, this negative forgetting is the product of the learning paradigm's struggle to balance stability and plasticity. It should be reminded that in these experiments, although bit toggles were explicitly invoked by the use of the mutation operator, performance change was not guaranteed

with each mutation; changes in performance were the effect of bit mutation. Moreover, changes to the policy is not necessarily forgetting but the loss of the ability to perform the subtask at a previously obtained level is. It is these performance proficiency and policy changes that is referred to as *forgetting* and is what we have sought out to demonstrate occurs with this learning paradigm. These problems represent (at a high level) a range of different properties present in many real RL problems and stability-plasticity imbalance was found in all of them. It is therefor evident that this is a general challenge for layered learning.

The conclusion that negative forgetting dominated the learning process is made because the learning system lost learned subtask proficiency (stability) when the bit string was altered during the optimization for a new subtask (plasticity). To provide further evidence that negative forgetting occurred and to measure its magnitude, we have used our proposed forgetting metrics to analyze the performance changes. Table 3.10 displays the average *performance difference (PD)*, *direct forgetting measure (DFM)*, and *maximized forgetting measure (MFM)* values across all time-steps of the LOTZ experiments with the fixed duration halting condition. Again, for each of the metrics, if the value is positive, positive forgetting had the greatest impact on learning. If the forgetting value is 0, a balance between positive and negative forgetting took place, resulting in neutral forgetting occurred over the learning process. If the value is negative, negative forgetting dominated the learning process.

Table 3.10: Forgetting Metrics for Fixed Time Duration Test Case

	Plan 0: 1 Layer	Plan 1: 2 Layers			Plan 3: 3 Layers		
	PD/DFM/MFM	PD	DFM	MFM	PD	DFM	MFM
LO*TZ	0.00011	0.00007	0.00007	-0.01166	0.00004	0.00004	-0.02393
LO+TZ	0.00015	0.00012	0.00011	0.00005	0.00009	0.00008	-0.04363

Table 3.10 reveals that the monolithic approach of test plan 0 produced the highest positive for-

getting while plan 3, with a 3-layer decomposition, generated the least positive forgetting and the most negative forgetting. Again, poor decomposition performances and the increase in negative forgetting compared to test plans with less decomposed subtasks is attributed to the performance evaluation function transitions.

Finally, from the experiment results and observations, two common properties were identified to make layered learning and these problem decompositions susceptible to negative forgetting and stability-plasticity imbalance. First, in layered learning, the solution landscape changes throughout the course of learning a task, forcing the policy to adapt to each new subtask. These layer transitions are native to the learning paradigm and essential to its sequential learning process. Unfortunately, these transitions have caused vital policy information to be lost and negative forgetting to occur in these experiments. Second, although policy overlapping allows for an entire policy to be modified and optimized for a subtask, the layer transitions can have a disastrous affect on how the policy retains important subtask knowledge. In these experiments, every tested decomposition-based test plan used layer overlapping, meaning each layer had read/write access to the same bits. Because of this, subtasks in different layers forced policy change, resulting in knowledge obtained in older layers to be forgotten in order to make way for knowledge to solve newer subtasks.

UAV Surveying Problem

In the previous section, we contrived a set of simple optimization problems designed to create an imbalance. Here, we demonstrate an imbalance in layered learning that favors plasticity by training agents to solve a more realistic collaborative multi-agent problem. Specifically, in this studied problem, each agent represents a quadcopter *unmanned aerial vehicle* (UAV) that must learn when and how to take off and land, move to a coordinate in the environment, refuel, and transmit reached coordinates to its UAV teammates. *Quadcopters* (also known as *quadrocopters*)

are similar to remote controlled helicopters with four rotors instead only one. They are known for their mobility and ability to change direction and heading quickly.

In this surveying task, four UAVs are initially grounded at four different corners of the 50x50 2D discrete grid environment. Figure 3.5 represents the simulation scenario at 10th the scale of the 50x50 2D environment. In each run of the simulation with this task, performance is evaluated on the percentage of unique grid coordinates the UAV team has navigated to out of the 2,500 total number of cells in 2,000 time-steps. At every time-step, each UAV determines its state and selects and performs an action with that corresponding state from its policy. UAVs have the movement abilities to take off, land, and move towards any coordinate in the environment, one cell at a time.

UAVs remain active in the simulation as long as they have fuel. With each time-step a UAV is in flight, its fuel level depletes by one unit. If a UAV's fuel level reaches zero, the UAV becomes inactive and can no longer execute actions, such as movement to survey the environment or refuel. To replenish depleting fuel levels, a UAV must navigate back to a refueling depot (located on the ground of each of the four corners of the grid), land, and explicitly execute the “*refuel*” action.

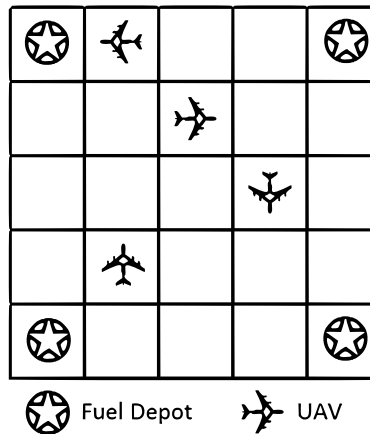


Figure 3.5: Visualization of the UAV Surveying Problem at 10th the scale of the environment

The UAV’s fuel level is replenished by 10 units for each time-step the refueling action is executed. Initially, a UAV has 100 units of fuel and refueling never exceeds a level of 100 units.

The surveying problem requires collaboration among the UAV teammates to maximize the task’s performance: no single UAV can navigate to every cell in the environment in the limited amount of time and duplication with UAVs covering the same cell wastes time. To facilitate collaboration, each UAV maintains a list of coordinates the team has visited and may execute the action “*transmit coordinates*” to broadcast a complete history of all of the coordinates that UAV has reached during the simulation up until the current time-step. This broadcast is instant and is guaranteed to reach each of the other UAVs.

Because each UAV must decide when to transmit and when to perform other actions, a UAV also contains a buffer of coordinates it has surveyed since its last transmit action was executed. Sub-states, such as “is buffer size at least 5”, exist so each UAV can determine their own thresholds of when to transmit unshared, reached coordinates with their teammates. We discuss policy representation more specifically in section 3.

The problem is similar to those studied in other reinforcement learning and direct-policy search works, including UAV navigation and search-and-rescue problems. This UAV exploration problem was selected because of its task complexity, straight-forward decomposition into subtasks, requirement of UAV collaboration among teammates to solve the problem optimally, and direct mapping to problems used in the real world.

Layered Learning Configuration

To learn to solve the problem, a decomposition of the UAV surveying problem and a layer configuration, including a subtask sequence, training scenarios, and halting conditions were used with

layered learning to optimize the UAV policies. The employed decomposition was as follows: the agent team first learned to refuel in layer 1, take off and survey the environment in layer 2, and lastly, the overall task of surveying and refueling when necessary in the final, aggregate layer of layer 3.

The process of layered learning takes one set of policies, one policy for each UAV, and passes it into layer 1 for optimization on the refueling subtask. In that layer, a learning algorithm is used to optimize performance on the subtask by repeatedly running the refueling simulation and modifying the policies to map states to appropriate actions. For underlying learning, we examine two simple variants of very different types of RL: a simple (1+1) EA for direct policy search and a Q-learner for value search. The evaluation and modification process, which is reliant on these learning algorithms, repeats for a set number of iterations, called epochs. These are discussed in more detail below. We use the term *epoch* to denote a *generation* in an evolutionary algorithm and a simulation iteration for Q-learning. The refueling optimized policy set is then fed into the second layer, which does the same process as layer 1 but optimizes the policy set on the take off and survey subtask. Finally, the policy set outputted in layer 3 is optimized on the overall task. Through this decomposition, the aim is to gradually learn each important aspect of the overall task in incremental phases of learning and have a policy set that solves the overall task at the end of the last epoch of layer 3.

UAV Refueling Layer: The refueling subtask trained the agents to recognize when their individual fuel levels were low and how to navigate to the refueling depot, land, and refuel. This layer's simulation initialized the UAVs to be 20 cells away from a fuel depot with 20 units of fuel, to force the UAVs to be low on fuel. Each UAV received .25 points for accomplishing each successive milestone of first navigating to a depot, then land, start to refuel, and continue to refuel until fuel level was above 90% full. Each simulation run lasted for 35 time-steps. For example, if a UAV navigated and landed at a depot, it received a score of .5 where as if the UAV performed all four

actions, it received a maximum score of 1.0. To evaluate the entire policy set, performance for one simulation was the average score of each UAV, where 1.0 meant the entire UAV team has mastered the subtask of refueling.

Take off and Survey Layer: The second layer’s subtask is the same as the overall UAV Survey task, except the simulation lasts 100 time-steps instead of 2000 and the maximum unique cells to reach is 396. Although the environment dimensions remains the same 50x50 grid, the reduction in the number of time-steps means the number of possible cells the team can reach is 396 (each UAV can reach a maximum of 99 cells after taking off in the time limit). Similarly, because the team is evaluated on the percentage of unique cells navigated to and because a refueling UAV would waste time refueling, the UAVs do not have to refuel to maximize performance of this subtask.

The limitations of this subtask make this layer critical and create layer overlapping. Because the simulation runs for 100 time-steps, the first and second layer shares states of when a UAV is low on fuel. The conflict occurs because the first layer’s subtask relies on a UAV to refuel while the second layer’s subtask relies on a UAV to survey the environment. Although it is acknowledged that the subtasks could be decomposed in a completely non-overlapping manner, the layer configuration was explicitly designed to uncover how layered learning responds to the forced conflict between desired actions with the shared states. Also, both of these subtasks are integral aspects of the overall aggregate task, optimized in the third layer. The decomposition allows the agents to master these critical and difficult subtasks individually instead of learning all of the required steps of each subtask all at once.

Policy Representation

Each UAV can perform exactly one action for each state that it is in at any given time-step. A UAV’s state is the collective value of the UAV’s internal conditions, called sub-states, similar to a

Boolean array or bit string, where each element corresponds to a specific sub-state value. A *sub-state* is a true or false value of a condition the UAV has at a time-step. For example, the sub-state “is grounded” is true if the UAV is on the ground, not in flight. At every time-step, each UAV evaluates all of its sub-states and represents that collection as a single state (similar to converting a boolean array into a decimal value). Table 3.11 lists all of the UAV sub-states and table 3.12 lists all of the actions.

Table 3.11: List of UAV Sub-States

State	Description
Is Fuel Level Low	True if the UAV can navigate to a fuel depot with 5 or less fuel units remaining.
Am I Grounded	True if the UAV is on the ground.
Am I at Fuel Depot	True if the UAV is at one of the fuel depots.
Is Fuel Level Above 90 Percent	True if the UAV’s fuel level is above 90%.
Is Untransmitted Buffer Size At Least 1	True if the UAV has at least 1 untransmitted coordinate in its buffer.
Is Untransmitted Buffer Size At Least 5	True if the UAV has at least 5 untransmitted coordinate in its buffer.
Is Untransmitted Buffer Size At Least 10	True if the UAV has at least 10 untransmitted coordinate in its buffer.
Is Untransmitted Buffer Size At Least 20	True if the UAV has at least 20 untransmitted coordinate in its buffer.

Table 3.12: List of Actions Each UAV Can Perform.

Action	Description
Move to Closest Unvisited Coordinate	Move one coordinate towards an unvisited coordinate.
Take Off	UAV takes off, instantly is in the air but the 2D coordinate remains the same.
Land	UAV becomes grounded at the last coordinate.
NOP	No operation is performed.
Move to Closest Fuel Depot	UAV moves one coordinate towards the closest fuel depot.
Refuel	If the UAV is grounded at a fuel depot, its fuel level is increase by 10 units, unless full.
Transmit Coordinates	Sends all of the coordinates the UAV has visited to the other UAVs.

The policy is a direct state-action mapping that links each state to one of seven actions that UAV can perform. In this problem, there are 7 unique actions with 8 sub-states. The simulation ignores impossible or redundant actions, such as landing when already grounded or fling to a new coordinate when grounded.

The Learning Algorithms

Two machine learning algorithms were examined separately to facilitate policy optimization over the problem: *(1+1) Evolutionary Algorithm* [(1+1) EA] and *Q-learning*. Q-learning is a RL technique that makes decisions from an estimated action-selection utility. The action utility is based on past decision outcomes and is used to predict the actions that will maximize reward in the future

[58].

The EA and Q-learning were selected to determine if an imbalance can be demonstrated in two different learning techniques and if they both share a cause for such an imbalance. Additionally, both techniques are popular with autonomous agents that come with backgrounds of work that address the stability-plasticity dilemma. Further, because we wish to focus on layered learning in general and not the underlying machine learning methods, we chose as simple examples from both direct-policy search and value-function search techniques as possible. These experiments demonstrate that although the employed learning algorithms can introduce their own biases toward stability or plasticity, it is layered learning's layer transitions that causes performance to negatively be affected by an imbalance with the decomposition-based approach.

(1+1) EA

The (1+1) EA uses a single “parent” candidate solution, generates a “child” candidate solution via a “mutation” genetic operator, and replaces the parent with the child if it performs at least as well as the parent. The sole parent is the set of policies of the team, where each policy represents the decision making capability of a UAV teammate.

For the (1+1) EA, a policy is represented as collection of state-action pairs, where no two pairs in a policy share a state. During an epoch, every reached pair corresponding to an activated state in the simulation has a 25% chance of being *mutated* by changing the pair's action to a random one. Through trial and error, this mutation rate produced a high enough exploration rate to find optimal subtask performing policies quickly. When an agent reaches a state that is not mapped in its policy, a new pair is created, linking that new state to a random action.

Q-Learning

For the Q-learner, a Q-table is used to represent the policy. The *Q-table* is a 1-to-many mapping that connects every state to each possible action’s estimated utility. Using the standard Q-learning function as defined in [58], a Q-value in the Q-table is updated after an action has been executed by a UAV and a reward determined. Immediate rewards were given by individual UAVs performing predetermined appropriate actions when in certain states; all other action rewards were set to zero. For example, when training to learn to refuel in layer 1, an immediate reward of 1.0 was given when a flying UAV located away from the fuel depot was low on fuel executed the action to move towards the closest depot. Tables 3.13 and 3.14 contain all sub-states and performed actions that would receive a maximum immediate reward for refueling and take off and survey, respectively. A learning rate (α) and discount factor (γ) of .2 and .1 were used, respectively. Through trial and error, these values produced the best task performers.

Table 3.13: Q-learning Refueling Immediate Reward Criteria

Sub-State	Move to Closest Unvisited Coordinate	Land	Refuel
Am I at Fuel Depot	TRUE	TRUE	TRUE
Am I Grounded	TRUE	TRUE	TRUE
Is Fuel Level Above 90 Percent	TRUE		TRUE
Is Fuel Level Low	TRUE	TRUE	

The policy set’s evaluation on subtasks followed the same scoring criteria as the (1+1) EA. Unlike the EA, a clone of the policy set is never made, therefore, one set of policies was continuously modified by updating Q-values in the Q-table.

Table 3.14: Q-learning Take Off & Survey Immediate Reward Criteria

Sub-State	Transmit Coordinates	Move to Closest Unvisited Coordinate	Take Off
Am I Grounded		FALSE	TRUE
Is Fuel Level Low	FALSE	FALSE	FALSE
Is Untransmitted Buffer Size At Least 5	TRUE		

In order for the learner to explore new solutions early in its optimization of a new subtask and finish the layer with higher reward-yielding action selection, an *epsilon-decreasing strategy* was use. In this strategy, the UAV selected the action with the highest Q-value at a proportion of $1-\epsilon$ of the time. When the highest Q-valued action was not selected, a random action was selected. At the start of a layer, ϵ is set to one and gradually decreases to zero as the number of epochs in that layer elapse.

Experiment Description and Results

The defined layer sequence of layers 1 through 3 was used to train the UAV team to perform the UAV Surveying task by using the (1+1) EA and then Q-learning, in two separate instances of layered learning. In each instance, every layer was given 10,000 epochs to learn a layer’s subtask, limiting the entire learning process to 30,000 epochs and simulations to learn the overall task. Each instance was evaluated over 100 runs to create a large enough sample size to reduce error.

To demonstrate imbalance in this UAV surveying implementation, we first analyzed the compositional changes made to the policies during learning. Tables 3.15 and 3.16 display the average state-action pair changes between a layer and one of its subsequent layers. From the (1+1) EA

experiment, 14.58 averaged pairs were changed between the last epoch of layer 1 and the last of layer 2, as displayed in Table 3.15. This means, that of the approximately 56 shared pairs of layers 1 and 2, about 26% of them were modified in the optimization of layer 2's subtask. Similarly, 32% of the overlapped pairs of layers 2 and 3 were modified in the final layer. In Q-learning, an even higher percentage of changes took place, averaging 158.29 shared pair changes between the first two layers, resulting in over 63% of the states in layer 1 had a new highest Q-valued action after the next 10,000 epochs. In the final policy set, layer 3 changed 41.55 existing pairs, shared between it and layer 2.

The Effects of Stability-Plasticity Imbalance in Layered Learning

The stability-plasticity imbalance caused a number of adverse effects to the learning system, including beneficial knowledge to be negative forgotten, policies to fall in local optima traps, and the learning system to perform as poorly as a monolithic system, which was expected to struggle learning the complex task.

Negative Forgetting

Negative forgetting clearly occurred in these layered learning experiments when vital refueling pairs were modified in the second layer and pairs for the take off and survey subtask were changed in the third, aggregate layer, leading to significant proficiency decreases in the first then second subtask, respectively. For instance, at the end of layer two, 25% of the pairs obtained to refuel in layer one were modified to keep surveying even when fuel levels were low, resulting in significant refueling performance drop from .87 to .37. Here, policy change directly led to subtask performance to drastically decrease. The Q-learner experienced the same type of forgetting with the refueling subtask, going from mastery of the subtask (refueling score of 1.0) to completely losing

the skill (score of 0.0), where the UAVs could not perform the first step in the refueling sequence at the end of layer two. The take off and survey subtask also experienced negative forgetting in the third layer as the UAVs repaired refueling pairs damaged in the second layer in an attempt to optimize on the overall task. These repairs led to a decrease from .74 to .44 in layer two's subtask performance and from .82 to .73 with the (1+1) EA and Q-learner, respectively.

To further confirm the occurrence of negative forgetting throughout these two layered learning experiments, the maximized forgetting metric was used for diagnosis. Evaluated through the completion of the third layer, MFM classified that layered learning suffered from negative forgetting with both the (1+1) EA and Q-learning. When the weights were equal ($w_i = .3333$, where $1 \leq i \leq |W|$), the (1+1) EA's average MFM was -0.1514 (STD 0.1164) and -0.3415 (STD 0.1705) for Q-learning. With unequal weights that placed more overall importance on performance of the aggregate task ($w_{Refuel} = .25$, $w_{TakeOffAndSurvey} = .25$, and $w_{UAVSurveyTask} = .5$), the EA's MFM was -0.1139 (STD 0.087) and -0.3772 (STD 0.1834) in Q-learning. By returning negative values, all four MFM values indicate that over the course of learning, negative forgetting dominated the learning process experienced by layered learning. The main contributing factor for these negative forgetting values was the large drops in refueling subtask performance experienced in layer two and the fact that refueling proficiency obtained in the first layer was never regained in layer three.

One consequence to the occurrence of negative forgetting was the amount of wasted learning time used to regain lost proficiency in the final layer. Because important subtask optimized pairs were forgotten by the start of the third layer, the learner had to perform double duty by balancing the relearning of forgotten aspects of the refueling subtask with improving its taking off and surveying functionality. This distraction significantly hurt overall task performance and led to sub-optimal performance.

To demonstrate how detrimental the second layer was to the ability to refuel, a new independent

run of the (1+1) EA was conducted. In this run, we wanted to determine how long it takes for refueling proficiency that was negatively forgotten in layer two to be relearned to the same level obtained at the end of layer one. To do so, we replaced layer three's task of the UAV surveying problem with a repeat of layer one; thus the decomposition was to learn to refuel, take off and survey, and refuel again. Epoch 27,608 was the first to equal or improve layer one's average high refueling performance of 0.8925 (STD .1359). This meant that it took, on average, 7,608 epochs from the start of the third layer and while focusing on solely relearning to refuel to regain lost proficiency experienced in layer two; over 76% of layer three was spent on reverting pair changes crucial to refueling or finding new ways to refuel. Additionally, of the 14.31 averaged pair changes of the policy set introduced in layer two (STD 4.1248), 2.38 pairs (STD 1.5019) were restored by the end of layer three. This small amount of repaired state-action mappings coupled with how long it took to restore these forgotten pairs meant that it was a difficult problem for the learner to revert the small subset of the modified pairs that affected refueling performance. It was difficult because of all of the pairs in the policy set (14.23 pairs total per policy), 4% (.595 averaged pairs per policy) of them needed to be identified as beneficial knowledge and reverted to their end of layer one mappings. For a clearer understanding of how difficult reversion is, consider that the affected state must be activated in an epoch, its pair has to then be reverted back to the pre-layer two action (the process includes a uniformly random selection of one of the seven possible actions) if mutation is activated via a random roll for that state, and no other pair could be modified to a value that negatively affects the third layer's subtask. That entire sequence is highly improbable to happen early in the third layer. Because of how difficult it was to recover lost beneficial knowledge in a decomposition where the final layer focused on repairing the forgotten refueling capability, it is understandable that the original decomposition of refueling, then taking off and surveying, and finally the UAV surveying problem saw its policy set never fully recover the lost subtask proficiency, ultimately negatively affecting overall task performance.

Table 3.15: (1+1) EA: Average Policy Set Composition Changes Per Layer Completion

		Pair Changes		Unchanged Pairs		New Pairs	
		Avg	STD	Avg	STD	Avg	STD
Layer 1	Layer 2	14.58	4.65	42.06	7.25	65.27	9.75
Layer 1	Layer 3	21.77	5.49	34.87	6.84	83.20	10.52
Layer 2	Layer 3	39.55	10.52	82.36	12.33	17.93	7.81

Table 3.16: Q-learning: Average Policy Set Composition Changes Per Layer Completion

		Pair Changes		Unchanged Pairs		New Pairs	
		Avg	STD	Avg	STD	Avg	STD
Layer 1	Layer 2	158.29	19.44	89.43	8.11	20.67	2.86
Layer 1	Layer 3	129.48	19.40	113.44	6.92	21.55	2.91
Layer 2	Layer 3	41.55	3.51	190.83	4.51	0.88	0.89

These composition changes caused subtask performance of the first two layers to drastically decrease during the next, immediate layer. When learning with the (1+1) EA, refueling proficiency dropped from 87% of optimality earned at the end of layer 1 to 37% at the conclusion of layer 2. The second layer’s subtask earned a proficiency of 74% optimality but dropped to 44% in the overall task’s layer. The same trend occurred with Q-learning. After mastering the ability to refuel in layer 1 with a 100% optimality score, the agent team completely lost proficiency in layer 2 with a 0% score, resulting in UAVs failing to perform any portion of the refueling subtask. The second subtask also suffered a loss in proficiency: in the transition from layer 2 to layer 3, subtask proficiency decreased from 82% to 73%. All of the average subtask proficiency changes for the (1+1) EA trained policy set and Q-learning can be respectively found in Tables 3.17 and 3.18.

Table 3.17: (1+1) EA: Average Subtask Performance after each Layer Completion

	Refuel	Take off & Survey	UAV Surveying Task
Layer 1	0.87063	0.02907	0.00408
Layer 2	0.36500	0.73896	0.12144
Layer 3	0.28875	0.43652	0.55824

Table 3.18: Q-learning: Average Subtask Performance after each Layer Completion

	Refuel	Take off & Survey	UAV Surveying Task
Layer 1	1	0.04846	0.04586
Layer 2	0	0.82444	0.12961
Layer 3	0.75313	0.72790	0.36917

Additionally, subtask proficiency of the previous layer was observed to sharply drop at the beginning of each new layer. Figure 3.6 plots subtask performance throughout the entire learning process, where each 10,000 epochs denotes a layer transition. This plot illustrates that after the first transition, refueling proficiency of 0.86 decreases rapidly before stabilizing around .38 for the remainder of layer 2. At the same time and consequently, performance of the take off and survey subtask increase quickly until halting at .74. A similar performance drop as the refueling case took place between layers 2 and 3, where the second layer's subtask decreases suddenly after the 20,000th epoch, when the performance evaluation criteria switches to the overall task.

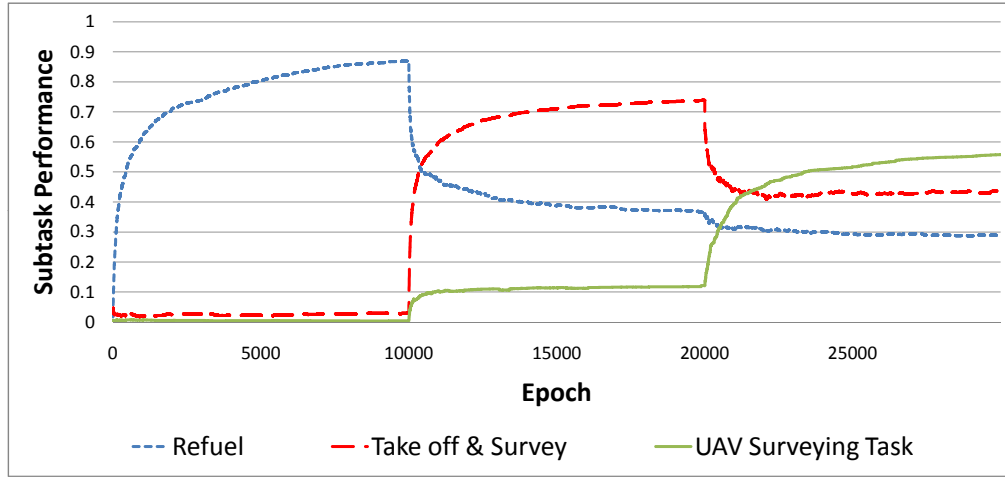


Figure 3.6: (1+1) EA Average Subtask Performance of all Runs

Lastly, after observing that the oldest learned subtask from layer 1 was the most volatile in terms of subtask performance and composition changes, the number of pairs that were reverted to layer 1's state at the end of layer 3 was used as an indicator of the damage caused by its subsequent layers. From the (1+1) EA's 14.58 pairs changed in layer 2, 13.33 were permanently damaged at the conclusion of learning, and remained different from the first layer. Comparably, of the 158.29 pairs changed at the end of layer 2 from layer 1 with Q-learning, 119.43 were permanently damaged. These high rates of permanent damage mean that once a refueling pair has been changed, the learning algorithm found it extremely difficult to revert the change to restore lost proficiency.

These experiments show that not only do new layers modify existing pairs that are shared with other layers, but proficiency is modified as well. Because subtasks were not retrained in this decomposition, once a layer has completed, subsequent layers modified pairs relied on to perform earlier learned subtasks. This occurrence is unfortunate for subtasks where their vital pairs are modified in newer layers because, as the permanently damaged pair statistics have highlighted, the probability for repair is modest (9% with the EA learner).

Furthermore, the examined instances behaved greedily by focusing all of their attention to the current layer's subtask for optimization. Simply put, there was no incentive to keep pairs used in older layers intact. This resulted in the learner purposely driving older, obtained knowledge out of a policy if it conflicted with a pair that improved the latest layer's subtask. This strong affinity for pair changes that lead the policy set closer to the latest layer's subtask optimality means the learner favors plasticity over stability; consequently, the affinity means imbalance can exist in the layered paradigm.

Local Optima

The occurrence of negative forgetting and the learner's inability to fully regain the lost subtask proficiency indicate that layer transitions have a great, negative affect in how the agents learn to perform the overall task. In fact, it is the performance criteria changes and the overlapping layers' ability to drive out conflicting, critical pairs. This can cause the policies to fall into a local optima trap as the layers transition away from the refueling subtask. The result is that the layered learning agents never gain the ability to perform the overall task optimally.

Again, an optimal solution to the UAV surveying task requires the UAVs to refuel and survey when appropriate. Simply, without high proficiency of refueling the UAVs would run out of fuel before reaching each coordinate. In the second layer specifically, pairs vital to refueling are replaced with ones that optimize the second layer's subtask. These policy changes force the policy set into a local optima and requires the agents to relearn the difficult action sequence for refueling while also learning to survey in the third layer if task optimization is to be reached.

This situation is a local optima trap because at the start of third layer, the policy is already optimized for the second layer's subtask of surveying until the UAVs crash due to lack of fuel. A *local optima* is a sub-optimal solution to a problem that the learner becomes fixated on, often due to difficulty

in finding a better solution. In this case, because the refueling subtask requires a specific sequence of actions to be executed, it is difficult for the policy set to revert those key pairs and maintain pairs important to surveying, all while improving the current policy set's overall task performance. Furthermore, each policy has no incentive to immediately repair lost refueling capability early in the third layer because the overall task performance increases quickly with minimal pair changes to the policy set at the end of the second layer. At this time, refueling pairs continue to be forgotten as the learner finds it easier to survey without refueling.

Observations of the policy set after layer three reveals that two UAVs retained refueling ability while two maintained layer two's abandonment of refueling and surveyed until fuel was depleted. Because it was more difficult to relearn the exact sequence of refueling for all four UAVs, the policy set fell into the optima trap influenced by layer two and spent the remaining time in layer three waiting for the improbable event of pair changes that re-enabled a UAV's refueling ability and survey when fuel was not low.

In comparison to the decomposition that solely relearned to refuel in the third layer, the original decomposition to solve the UAV Surveying problem was only able to recover half of the pairs necessary to regain forgotten refueling proficiency. At the end of the 30,000 epochs, the (1+1) EA with the original decomposition was only able to recover an average of 1.25 (STD 1.252) pairs changed in layer two compared to the relearning decomposition's 2.38 (STD 1.5019). In other words, the relearned decomposition repaired 4% of the negatively forgotten refueling pairs while the original decomposition only repaired 2%, half the amount of pairs. Because the original decomposition could not repair all of the forgotten critical refueling knowledge, the learner never regained the lost subtask proficiency required to perform the overall task optimally.

Layered Learning vs. Monolithic Learning

To highlight the overall negative impact the stability-plasticity imbalance can have on layered learning and show how difficult it was for the learner to overcome negative forgetting and a local optima trap, we compare the decomposition-based learning paradigm's performance against that of a standard, monolithic approach that learned to perform the UAV surveying task without decomposition. The monolithic approach is constructed in the same manner as the layered learning approach with the two exceptions that 1) there is only one layer (layered learning's third layer) that optimizes the policies on the overall task, and 2) the sole layer has the total number of epochs (30,000) used in all three layers of the decomposition approach to learn to perform the task.

Although a widely accepted and standard form of learning, monolithic learning was expected to struggle with the UAV surveying problem. The challenge was that each UAV policy had to learn to perform the task by navigating through one large solution space, essentially mapping every reached state to an appropriate action without the guidance of a decomposition. Lacking a decomposition and the advantages of transfer learning and robot shaping, each of the monolithic learner's policies had to be optimized to recognize when to refuel and learn the correct refueling sequence, when it is appropriate to survey, and how to take off, navigate to unvisited coordinates, and when to transmit its visited coordinate buffer to other UAVs all at the same time to contribute to the team's performance.

With the decomposition used in layered learning, it was assumed that the layers would reduce the solution space, making it more probable for the policies to be optimized on the task via the sequential learning of three simpler subtasks. With the introduction of monolithic learning, a total of four experimental runs were evaluated: layered and monolithic learning each using the (1+1) EA and Q-learning with the limitation to 30,000 epochs.

Table 3.19 displays the average performances of the overall task at the last epoch of each layer. Figure 3.7 plots the average performance over the 30,000 epochs with the (1+1) EA. Table 3.20 provides the standard deviations of the final, average UAV surveying task performances and two-sample Z-test results comparing the layered and monolithic learning results for each learning algorithm used.

Table 3.19: Average UAV Surveying Task Performance at the Completion of Each Layer

Learning Algorithm	Decomposition	Layer 1	Layer 2	Layer 3
(1+1) EA	Monolithic	0.4091	0.5021	0.5316
	Layered Learning	0.0041	0.1214	0.5582
Q-learning	Monolithic	0.0685	0.0690	0.3642
	Layered Learning	0.0459	0.1296	0.3692

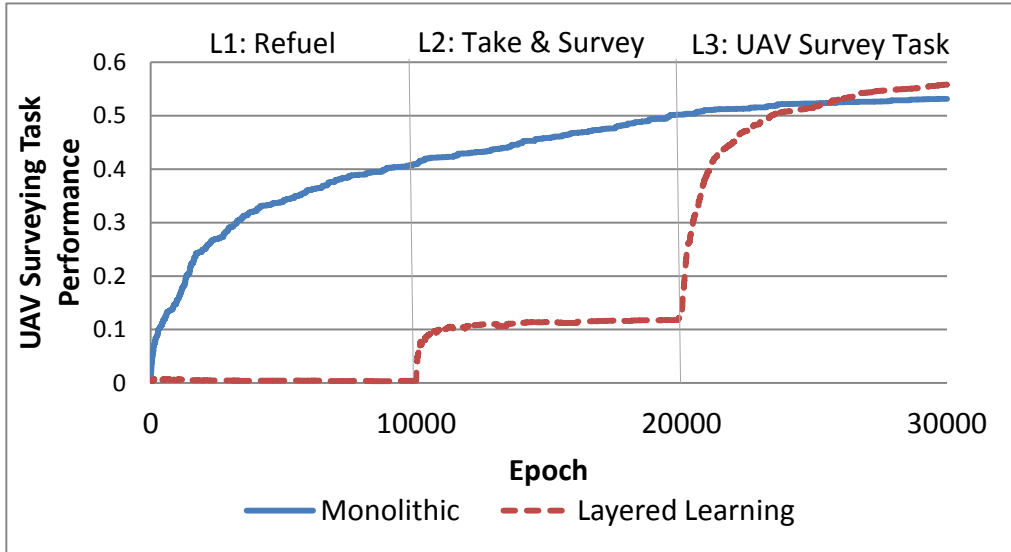


Figure 3.7: Average UAV Surveying Task Performance Using (1+1) EA. Vertical bars at epochs 10,000 and 20,000 represent the layer transitions.

These results revealed that layered and monolithic learning performed statistically similarly to each other. In the (1+1) EA case, the monolithic learner averaged a task score of 0.5316 (STD 0.1189) and layered learning averaged 0.5582 (STD 0.1506). The Q-learning resulted in monolithic learning averaging 0.3642 (STD 0.0277) and layered learning 0.3692 (STD 0.0461). Although both approaches returned similar task averages, the difference in how they generated their policies in the EA case are revealed in 3.7. Because the EA only kept policy changes that improved the current subtask’s performance, layered learning experienced plateaus as the subtasks changed through out the 30,000 epochs. The monolithic learner had only one subtask (the overall task) to maximize. Therefore, learning monolithically produced a smoother performance curve as it constantly accepted policy changes that improved task performance over the learning duration.

Table 3.20: Standard deviations, Z-scores, and P-values ($\alpha = 0.05$) of the final average UAV surveying task performance scores.

Learning Algorithm	Approach	STD Layer 3	Two-Sample Z-Test	
			Z-score	P-value
(1+1) EA	Monolithic	0.1189	-1.3863	0.1657
	Layered Learning	0.1506		
Q-learning	Monolithic	0.0277	-0.9297	0.3525
	Layered Learning	0.0461		

Conclusion

The negative effects of the imbalance, such as the waste of relearning cycles used to regain negatively forgotten knowledge and policies falling in local optima traps, cause the decomposition-based approach to perform as poorly as the monolithic case. The transitions to new layers with new subtasks have a traumatic effect on the policies trying to converge toward optimal performance. The identification of these layer transitions being the main culprit of the imbalance during

both the Boolean-logic experiments and these UAV surveying problem demonstrations is noteworthy because these transitions are integral to layered learning; without them, layered learning could not guide policies toward solving complex tasks via the mastering of simpler problems and transfer the learned knowledge from one layer to another. Moreover, because similar transitions are used in other learning approaches, this work's demonstrations imbalance and identification of its negative effects and problem-source provides warning to other decomposition-based learners about the pitfalls of the stability-plasticity dilemma.

CHAPTER 4: STABILITY-PLASTICITY BALANCING TECHNIQUES

It has been demonstrated that layered learning can suffer from a stability-plasticity imbalance from the preliminary work’s predator-prey, LOTZ Boolean-logic, and UAV Surveying problems. Layer transitions have been identified as the main cause of the imbalance due to the lack of learned beneficial knowledge retention, and the discovery that negative forgetting is a significant imbalance side-effect. We now turn our attention to analyzing solutions to balance stability and plasticity to improve the learning paradigm’s effectiveness. We first explore the possibility of using two existing techniques, subtask rehearsal and concurrent layered learning, discussed in this dissertation’s background to mitigate, the imbalance’s negative effects. Subtask rehearsal has demonstrated the ability to regain lost beneficial knowledge in other decomposition-based learners. Concurrent layered learning has outperformed standard layered learning because of its ability to learn a new subtask while retaining an old subtask’s proficiency. We demonstrate that although these techniques showed promise from the literature, under the conditions of our multi-agent problem, the two approaches do not sufficiently address the imbalance problem in general. We then introduce a solution that fills the gap of the other mitigation techniques by applying a complementary learning system to layered learning, called complementary layered learning. We see that the best result occurs when complementary layered learning is combined with other, more specialized approaches. Finally, we compare and contrast the proposed system to standard layered learning, layered learning with subtask rehearsal, and concurrent layered learning to demonstrate the solution’s effectiveness.

Existing Balancing Techniques

Subtask rehearsal and concurrent layered learning (ConcLL) were examined because of their promising results for improvement derived from the literature. Furthermore, because none of the layered learning approaches discussed in the literature review remove pairs from policy and only modify their state-action mappings, techniques such as deletion strategies are not relevant to existing uses of the learning paradigm.

Subtask Rehearsal

Subtask rehearsal is a technique where a subtask is explicitly relearned. We apply subtask rehearsal to layered learning by repeating a layer. The goal of this layer repetition is to regain lost knowledge by reintroducing pairs used by one layer but forgotten in another.

To implement subtask rehearsal in layered learning and train the agent team to solve the UAV Surveying Problem, we added an additional refueling layer after layer 2’s take off and survey and before the final, aggregate layer. Now, the layer sequence is refuel, take off and survey, refuel, and then the overall task. All layer compositions remain the same as described in the UAV Surveying Problem section, with one exception: the time spent on the refueling subtask is split in two, creating two separate layers both for refueling. In other words, the number of epochs in the first refueling layer is reduced from 10,000 to 5,000 and a new, second refueling layer with 5,000 epochs is created. Splitting the number of refueling epochs allows for the take off and survey and the overall task layers to keep their original number of learning iterations and maintain a fair comparison with standard layered learning’s performance.

Direct Policy Search Performance Evaluation

After evaluating subtask rehearsal on the UAV Surveying Problem for 100 runs, following the same experiment settings as in section 3, we can conclude that the technique does not improve layered learning’s direct-policy search performance. Subtask rehearsal averaged a task performance of .5547 (Std. Dev: .1016), showing no significant difference from layered learning’s .5582 (.0964). Table 4.1 displays the rehearsal and concurrent layered learning performances on the UAV Surveying Task with a two-tailed, multi-way Z-test using a Bonferroni adjusted α of 0.025. The Z-test compares both techniques to layered learning’s task performance.

Table 4.1: (1+1) EA: UAV Survey Task Performance and Multi-Way Z-Test ($\alpha = 0.025$)

	AVG	STD	% Delta	Z-Score	P-Value
Layered Learning	0.5582	0.0964			
Subtask Rehearsal	0.5547	0.1016	-0.0064	0.2548	0.7989
ConcLL	0.5382	0.0895	-0.0360	1.5259	0.1270

Subtask rehearsal performed as badly as standard layered learning because it suffered from the same problem as its predecessor: vital pairs of a subtask were forgotten during the optimization of a new subtask, and the final layer could not recover them.

Although the addition of a second refueling layer resulted in a recovery of the subtask’s lost proficiency experienced by layer 2, it was at the expense of the ability to take off and survey. In standard layered learning, a 58% refueling performance decrease took place between the first two layers. A similar 54% decrease occurred from layer 1 to layer 2 with subtask rehearsal. Layer 3 not only recovered the lost proficiency from layer 1, which averaged a refueling performance of 0.7919, but exceeded its performance by averaging a refueling score of 0.8569. Conversely, the take off and survey subtask saw a loss in proficiency during the second refueling layer, resulting in

the subtask score to drop from .7384 to .5389, a 27% decrease. These subtask fluctuations meant that the policy set entered the final layer with a high refueling performance and a moderate take off and survey capability, the same problem experienced with layered learning but with the subtasks swapped. Table 4.2 displays subtask performances at the end of each layer.

Table 4.2: (1+1) EA: Subtask Performance Per Layer for Subtask Rehearsal

	Refuel	Take off & Survey	UAV Surveying Task
Layer 1	0.7919	0.0291	0.0042
Layer 2	0.3631	0.7384	0.1139
Layer 3	0.8569	0.5389	0.1021
Layer 4	0.4488	0.4294	0.5547

Under subtask rehearsal, although the third layer recovered lost refueling proficiency, the chance to relearn lost knowledge for the take off and survey subtask relied on the final layer. This responsibility placed on the final layer meant that optimization was split to focus on both of the first two subtasks with a random, unguided search, as experienced by standard layered learning. This solution search is difficult, resulting in sub-optimal performance in standard layered learning and now with rehearsal.

The high number of permanently damaged take off and survey pairs supports the claim that the rehearsal method did not overcome the difficulty of recovering forgotten pairs. At the completion of all of the layers, each policy only averaged 1.41 (1.06) recovered pairs that were changed between the take off and survey layer and the second refueling layer, leaving 13.75 (5.75) pairs per policy permanently damaged. This low amount of reverted pairs is similar to layered learning's small average of 1.25 refueling pairs that were recovered in the final layer.

Lastly, in terms of forgetting, subtask rehearsal performs very closely to standard layered learning. The MFM Even metric, which weights each of the three subtasks the same, evaluated standard

layered learning’s negative forgetting at -0.1514 and rehearsal at -0.1495. Similarly, MFM 50, which weights the overall task at .5 and the remaining subtasks at .25 each, evaluated layered learning at -0.1139 and rehearsal at -0.1003. Both maximized forgetting metric scores show how similar both approaches perform and highlight that negative forgetting was not mitigated with the rehearsal technique. Table 4.3 displays all of the average forgetting scores for all of the tested (1+1) EA techniques. Table 4.4 displays their standard deviations.

Table 4.3: (1+1) EA: Average Forgetting Scores

	LL	Rehearsal	ConcLL	CLL	CLL-Reh	CLL-Conc
PD	0.000018	0.000018	0.000021	0.000021	0.000024	0.000022
DFM Even	0.000014	0.000015	0.000017	0.000018	0.000021	0.000022
MFM Even	-0.151428	-0.149456	-0.070811	-0.114623	-0.117318	-0.029358
DFM 50	0.000015	0.000016	0.000019	0.000019	0.000022	0.000022
MFM 50	-0.113855	-0.100327	-0.041490	-0.086355	-0.079330	-0.017407

Table 4.4: (1+1) EA: Standard Deviations of the Forgetting Scores

	LL	Rehearsal	ConcLL	CLL	CLL-Reh	CLL-Conc
PD	0.000195	0.000212	0.000218	0.000231	0.000246	0.000202
DFM Even	0.000232	0.000277	0.000238	0.000242	0.000275	0.000232
MFM Even	0.116387	0.091762	0.084097	0.079153	0.058851	0.032020
DFM 50	0.000188	0.000198	0.000172	0.000197	0.000210	0.000164
MFM 50	0.087000	0.060697	0.048898	0.059042	0.038924	0.018443

Value Search Performance Evaluation

Subtask rehearsal with Q-learning improved layered learning’s value search performance by 11%, averaging a task performance of 0.4113 (0.032). Table 4.5 shows the Z-test results comparing

subtask rehearsal and concurrent layered learning to standard layered learning. The rehearsal technique's performance was significantly different than that of layered learning even though the first two layers followed the trend set by layered learning of obtaining optimal refueling proficiency during the first layer and completely losing all of that subtask's performance by the end of the second layer. The third layer, which repeated optimization on the refueling subtask, performed as hoped. This third layer regained all of the lost proficiency by earning a refueling score of 1.0. Additionally, performance for the take off and survey subtask only decreased by 17% from layer 2 to layer 3, resulting in the final layer to be seeded with a refueling proficiency of 1.0 and 0.7047 for refueling and take off and survey, respectively. Table 4.6 shows the average subtask performance changes per layer completion.

Table 4.5: Q-learning: UAV Survey Task Performance and Multi-Way Z-Test ($\alpha = 0.025$)

	AVG	STD	% Delta	Z-Score	P-Value
Layered Learning	0.3692	0.0461			
Rehearsal	0.4113	0.0320	0.1142	7.8312	< 0.0001
ConcLL	0.4868	0.1260	0.3186	-8.7673	< 0.0001

Table 4.6: Q-learning: Subtask Performance Per Layer for Subtask Rehearsal

	Refuel	Take off & Survey	UAV Surveying Task
Layer 1	1.0000	0.0508	0.0567
Layer 2	0.0000	0.8212	0.1297
Layer 3	1.0000	0.7047	0.9574
Layer 4	0.7531	0.7345	0.4113

Concurrent Layered Learning

The second technique examined to see if it can improve layered learning was *Concurrent layered learning (ConcLL)*. ConcLL is an existing enhancement to learning paradigm that simultaneously learns two subtasks. ConcLL aims to retain older subtask knowledge by factoring in proficiency of this older subtask into the current epoch's performance evaluation, although a newer subtask is also being optimized on at the same time. We apply ConcLL to the UAV Surveying problem by evaluating the policy set's performance on the refueling and take off and survey subtasks during each epoch of belonging to layers in between the first and last layer; we call these layers *concurrent layers*. Individual, raw subtask performance evaluation remains the same as previously defined.

For each concurrent layer, we get two bounded $[0, 1]$ performance scores, one for refueling and one for taking off and surveying. We then weigh each score based on weights defined in the layer (the weights together sum to 1) and add the weighted values together. This value is the policy set's layer performance for that epoch. For instance, layer 2 weighs refueling at .90 and take off and survey at .1. At any epoch during this second layer, the refueling performance score is multiplied to the weight and added to the multiplication of the take off and survey performance score and its weight. In the tested layer sequence, there were 7 layers, 5 of which were consecutive concurrent layers that follow this scheme, each with different weights. The first and final layers were exactly the same as standard layer learning's layer configuration for refueling and the UAV Surveying Problem layer, respectively. The first layer remained the same to allow refueling optimization to occur without having influence of the take off subtask.

The concurrent layers and their weight scheme gradually adds the take off and survey ability to the policy, which facilitates a kind of gradual lending from one task to another, rather than abrupt task change. The weights for refueling and taking off and surveying were as follows: layer 2 (refueling .9, take off and survey .1), layer 3 (.8, .2), layer 4 (.7, .3), layer 5 (.6, .4), and layer 6 (.5, .5). This

weight sequence was determined by informal parameter sweep intended to give the best possible results.

Layers 1 and 7 each had 10,000 epochs while the concurrent layers had 1,000 each, for a total of 25,000 epochs. The reason for a difference of 5,000 epochs compared to standard layered learning and subtask rehearsal is because two simulation runs, one for refueling and one for taking off and surveying, has to be made for layers 2-6; this accounts for a total of 10,000 simulations for these layers and a total of 30,000 for all of the layers combined. With an equal number of simulation runs, we can make a fair performance comparison between ConcLL and the other tested approaches because each was given the same amount of time to optimize overall task performance.

Direct Policy Search Performance Evaluation

These ConcLL experiments also follow the same settings as described in the demonstration chapter. From their results, we conclude that just like subtask rehearsal, ConcLL does not improve layered learning's direct-policy search performance. ConcLL averaged a task performance of .5382 (.0895), not significantly different from layered learning's .5582 (.0964). Table 4.1 displays the statistical test results that compare the ConcLL's task performance to layered learning's.

The shared sub-optimal performance ConcLL has with standard layered learning is attributed to the technique's attempt to retain refueling capability while gradually learning how to take off and survey. By progressively increasing the influence of the take off subtask to a weight of 0.5 before the final layer, the technique allows the refueling subtask to dominate the policy. This domination stifles learning how to take off and survey by slowing down the learning rate of this subtask compared to how fast the subtask's proficiency increased in the other approaches. Table 4.7 contains all of the subtask performances for ConcLL at the end of each of the seven layers.

Table 4.7: (1+1) EA: Subtask Performance Per Layer for ConcLL

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.8888	0.0302	0.0085
Layer 2	0.8994	0.0920	0.0185
Layer 3	0.9069	0.1241	0.0239
Layer 4	0.9150	0.1656	0.0401
Layer 5	0.9125	0.2025	0.0404
Layer 6	0.8794	0.2804	0.0584
Layer 7	0.4713	0.3895	0.5382

Ultimately, ConcLL gives the final layer a challenge experienced by subtask rehearsal of having to simultaneously optimize the policy set on one lower performing subtask while maintaining, if not improve, proficiency on the other. In fact, both ConcLL and standard layered learning had subtask proficiencies that mirrored each other going into the final layer. At the end of layer 2 in standard layered learning, refueling performance was at 0.365 (0.2132) and the take off and survey was at 0.739 (0.0391). ConcLL went into its final layer with the opposite scenario of standard layered learning by having a high refueling proficiency and a moderate take off and survey performance capability, 0.8794 (0.1219) and 0.2804 (0.1355), respectively. The similar levels of proficiencies meant that ConcLL went into the final layer with similar difficulties as standard layered learning and the subtask rehearsal technique, explaining why ConcLL’s final task performance is not significantly different from the other two techniques.

Finally, ConcLL experienced the least amount of negative forgetting compared to the other two tested techniques. Table 4.3 displays the average forgetting metric scores. From this table, we see that ConcLL received a MFM Even score of -0.0708, more than half of that of standard layered learning and subtask rehearsal, and an MFM 50 of -0.415. These low forgetting values are due to the technique’s gradual convergence in subtask weights, which resulted in refueling proficiency to noticeably decrease starting in layer 5.

In short, subtask rehearsal and ConcLL did not enhance task performance of layered learning for a direct policy search. All three techniques suffered from the same challenge that ultimately limited task performance. The common difficulty was that the terminal layers all began with an imbalance in proficiency between the refueling and take off and survey subtask. In standard layered learning and subtask rehearsal, the difference in subtask proficiency was caused by the policies negatively forgetting pairs as the learner transitioned from one layer to another. In ConcLL, the changes in refueling and the take off and survey subtask performance was caused by the gradual convergence of subtask weights, which resulted in the learner favoring refueling. These imbalances caused the final layer to have the responsibility of maintaining a high capability in one subtask while searching for solutions that will increase proficiency in the other, under-performing subtask. Another contributing factor to these techniques' sub-optimal performance was that the (1+1) EA's search for optimality lost gradient across the solution space. This unguided search proved costly, specifically in the final layer, because of the learning time wasted evaluating pairs that do not improve overall performance. Neither of the two techniques provide any mechanisms to improve the final layer's solution space navigation. Lastly, although subtask rehearsal and ConcLL provide some incentive to retain learned knowledge or provide the capability to regain lost knowledge, vital pairs were still forgotten and a balance between old and new information was not achieved.

Value Search Performance Evaluation

ConcLL with Q-learning produced a 32% task performance increase over standard layered learning's value search. ConcLL averaged 0.4868 (0.1260) compared to layered learning's 0.3692 (0.0461). Table 4.5 contains results from the multi-sample Z-test comparing ConcLL, subtask rehearsal, and layered learning using Q-learning.

As witnessed in the rehearsal technique, after the first layer was complete, refueling proficiency

was optimal under ConcLL. The two techniques began to differ in performance afterwards. As the weights for refueling decreased while take off and survey's weight increased towards .50, refueling proficiency immediately dropped to 0.1669 (0.185) at the end of layer 2 and gradually decreased to 0.1294 (0.1379) by the end of layer 6. Consequently, layer 2 resulted in the take off and survey subtask performance to increase to 0.774 (0.0321) and stabilize until the end of the final layer, resulting in a final take off and survey average of 0.7317. The introduction of take off and survey subtask performance to the learner's policy evaluation immediately modified overlapping pairs shared between the two subtasks to favor the taking off and surveying capability. It was only at the final layer, where both subtasks implicitly contributed towards the policy evaluation, did refueling regain lost performance, averaging 0.7556 at the end of the learning process. Table 4.8 displays each subtask performance value per layer completion.

Table 4.8: Q-learning: Subtask Performance Per Layer for ConcLL

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	1.0000	0.0390	0.0433
Layer 2	0.1669	0.7740	0.1889
Layer 3	0.1544	0.7793	0.1634
Layer 4	0.1338	0.7789	0.1549
Layer 5	0.1369	0.7803	0.1351
Layer 6	0.1294	0.7798	0.1375
Layer 7	0.7556	0.7317	0.4868

Direct Policy Search vs. Value Search

The discrepancies between each of the tested balancing techniques using the (1+1) EA and Q-learning is attributed to both learning algorithms belonging to different types of reinforcement learning and having different search dynamics. We point to two different algorithm characteristics

that make the EA and Q-learning react differently to subtask rehearsal and ConcLL.

In these experiments, Q-learning was seen to be more sensitive than the EA. The sensitivity of the value search caused the learner to experience extremes in performance, resulting in subtask optimality to be achieved quickly and to “catastrophically” lose proficiency when a new subtask was learned. This same sensitivity allowed the policy set to easily recover lost proficiency. The sensitivity was witnessed in how each of the Q-learning approaches always achieved optimal refueling performance high proficiency in the take off and survey task but also lost refueling ability to zero or very low rates as the learning process progressed. Direct policy search did not experience such drastic proficiency changes; although high subtask performance was experienced and then decreased, neither subtask received optimal performance at any point nor did proficiency ever get completely lost like those in Q-learning. Because the Q-learner experienced more sensitivity and had to overcome variability in subtask performance, more time was spent on relearning lost subtask proficiency than the EA and resulted in lower task performance.

Next, we believe the difference in how the policy was represented attributed to why both rehearsal and ConcLL significantly improved UAV Surveying task performance with Q-learning’s value search but not in the (1+1) EA. Because the EA is represented by as a direct policy search, each state maps directly to a single action. When the learning algorithm modified a pair, the old mapping was removed from the learning system until the learner stumbled randomly across the pair again. Q-learning’s use of a Q-table meant that the selection of a state-action mapping was always persistent in the policy but was dependent on the utility function and ϵ value. For example, although a beneficial pair’s utility value could be lowered to make that action selection less likely in the future, to re-select the pair is more likely than that experienced by the EA. Because the Q-table kept state-action mappings persistent, the balancing techniques were able to reevaluate the lower utility actions more frequently. If these actions were beneficial to the current layer’s subtask performance criterion, then they would be rewarded, their utility value modified, and the probability

of future selection increased. Subtask rehearsal and ConcLL encouraged these pair reevaluations, catering to the characteristics of Q-learning.

It is clear that ConcLL can sometimes be beneficial in specialized cases, such as the Q-learning example for this problem because of the reasons just discussed; however, it is also clear that neither rehearsal nor ConcLL represents a reliable solution to the stability-plasticity dilemma in general. Even though subtask rehearsal and ConcLL did improve task performance in the value search, we continue our exploration for solutions that mitigate the negative effects of stability-plasticity imbalance and increase task performance for direct policy search.

Complementary Layered Learning

Complementary layered learning (CLL) is the balancing augmentation to layered learning that is introduced in this dissertation. It is inspired by complementary learning systems employed by neural networks to mitigate catastrophic forgetting when imbalance is expected in their systems. Just as in traditional complementary systems, CLL has dual memory storage regions used to retain previously learned beneficial knowledge that could be prematurely removed from the policy. CLL is designed to work by itself or in conjunction with subtask rehearsal or CLL.

The trend in complementary learning systems is to split the memory storage region into short- and long-term distinctions and make action selection by constantly accessing the two. The proposed enhancement to layered learning differs from these systems by augmenting the existing policy with an indexed collection of obtained pairs that are only accessed during the final layer. In the proposed system, the policy keeps its original functionality of being the action selection mechanism that is optimized through the layers. We add to the paradigm a simple indexed collection of pairs that the policy accesses when making pair updates. Once learning is complete, the indexed collection can

be discarded.

The process of CLL is as follows: at the end of a non-terminal layer, all pairs in the policy are copied into the indexed collection, permitting duplicates as new layers add their pairs. The terminal layer uses the indexed collection to modify the policy for its subtask optimization by reintroducing pairs obtained in earlier layers to the policy. Depending on the employed learning algorithm, the policy can either update its pairs after the penultimate layer has copied its pairs to the indexed collection (before the final layer begins) or during the terminal layer. For the case of updating the policy before the terminal layer, all pairs in the policy are iterated through and their mappings or utility function values are updated based on the corresponding pairs in the indexed collection. In the alternative case, at points during a layer that update policy pairs, individual pair updates are made based on corresponding pairs found in the indexed collection.

For instances of direct-policy search, all mapping updates are determined by a randomly generated number and a fixed probability of selecting from the indexed collection. If the random value is in favor of using the indexed collection, then the policy pair is updated to be a random pair from the indexed collection that matches the policy pair's state. A random pair from the collection is chosen because multiple pairs matching the policy pair's state is a possibility. For random draws against the indexed collection, the policy pair is updated to map to a randomly selected action from the set of all possible actions. The use of a random draw allows for both exploration and exploitation to take place in the final layer. High indexed collection probability favors exploitation and low favors exploration.

By updating the policy in the aggregate, final layer based on tested pairs from the indexed collection, the learning paradigm has an increased chance of regaining forgotten beneficial knowledge than the standard approach. This increased probability is attributed to the indexed collection's service as a knowledge bank, or a snap shot, of the policy after each subtask has been optimized. The

collection of tested pairs allows policy updates to not only rely on randomness for optimization but can also make policy changes based on pairs that at least one layer has considered beneficial.

To provide more explanation of CLL, we define how the new augmentation is implemented in the UAV Surveying Problem with the (1+1) EA and Q-learning algorithms.

In the (1+1) EA, a hash table is used as the indexed collection, where the state is the key and the action is the value. When a layer is complete, all pairs in the policy are inserted into the indexed collection's hash map, allowing duplicates during collisions. In the final layer, the mutation operator randomly generates a number between 0 and 1, both inclusive. If that number is less than .75, which represents a 75% probability of selecting from the indexed collection, then the pair being mutated will randomly select a pair in the indexed collection's hash table that shares the state. If the randomly generated number is greater than or equal to .75, then the pair selected for mutation is given a random action from the set of all possible actions the UAV can perform.

In the Q-learning implementation, a hash table is created for the indexed collection, setting the state-action mapping as the key and a Q-value as the value. Recall that we refer to the Q-table as the policy and changes to a state-action's Q-value is how action-selection is modified. When a layer is complete, pairs for each state with the highest Q-value are copied to the hash table, allowing duplicates upon collision and copying all pairs that share the same high Q-value for a given state. For example, multiple pairs may share a state and map to different actions but have the same Q-value. When the penultimate layer has finished copying its pairs to the indexed collection, the Q-value for every pair in the policy is set to the Q-value of the corresponding pair in the indexed collection. If there are multiple pairs in the index collection that share a state-action mapping, then their average is calculated and the corresponding pair in the policy is updated to it. Unlike the (1+1) EA implementation, when all of the pairs in the policy have completed their Q-value updates, the final layer begins and updates its Q-table (policy) in the same manner as its preceding layers.

Performance Comparison to Standard Layered Learning

To evaluate the effectiveness of CLL, we trained the autonomous UAV team using the proposed approach and compared results to those of standard layered learning. All of the experiment parameters, learning algorithms, and layer configurations were exactly the same as the UAV Surveying Problem implementation with layered learning, only now using the complementary learning system.

From 100 experiment trials of each algorithm, CLL produced a higher task performance than standard layered learning in both the direct policy search using the (1+1) EA and value search with Q-learning. In the direct policy search, CLL averaged a UAV Surveying task score of 0.631 (0.142) compared to layered learning's 0.5582 (0.0964), a 13% significant increase in task proficiency. When using Q-learning, CLL increased layered learning's task performance from 0.3692 (0.0461) to 0.4011 (0.831), a 9% increase. The increase in performance is caused by CLL's ability to reinsert forgotten beneficial knowledge back into the policy for direct policy search and boost utility values of diminished vital pairs in the value search. Table 4.9 displays UAV Survey task performance averages, standard deviations, and Z-tests results for the (1+1) EA experiments. Table 4.10 displays the Z-test results of the Q-learning experiments.

Table 4.9: (1+1) EA: Average UAV Survey Task Performance and Z-Test Results ($\alpha = 0.017$)

	AVG	STD	% Delta	Z-Score	P-Value
Layered Learning	0.5582	0.0964			
CLL	0.6310	0.1420	0.1304	-4.2395	< 0.0001
CLL-Reh	0.7139	0.1447	0.2789	-8.9543	< 0.0001
CLL-Conc	0.5670	0.1264	0.0158	-0.5536	0.5798

Table 4.10: Q-Learning: Average UAV Survey Task Performance and Z-Test Results ($\alpha = 0.017$)

	AVG	STD	% Delta	Z-Score	P-Value
Layered Learning	0.3692	0.0461			
CLL	0.4011	0.0831	0.0866	-3.3626	0.0008
CLL-Reh	0.4023	0.0347	0.0898	-5.7506	< 0.0001
CLL-Conc	0.4892	0.1337	0.3252	-8.4881	< 0.0001

Direct Policy Search Performance Evaluation

In the (1+1) EA case, lost beneficial knowledge is regained by the mutation operator having a 75% chance of updating a pair to be one that assisted in the optimization of a subtask from the first two layers. For instance, if the indexed collection contained two pairs for a state, one pair that assisted refueling to achieve a high performance score of 0.8644 at the end of the first layer and one pair that led to a take off and surveying performance score of 0.7423 at the end of the second layer, and if the random roll selected from the indexed collection, then there was a 50% probability that the updated action would benefit either subtask. On the other hand, if a random action was selected, such as what takes place in layered learning, subtask rehearsal, and ConcLL, then the probability of mutating a pair to something that is beneficial is less likely.

The benefit of having a high probability of selecting a tested pair that aided in a subtask's high performance takes place during the third layer. Here, both of the first two layers' subtask performances converge and stabilize quickly. Figure 4.1 displays performance for each of the subtasks during CLL's learning. Figure 3.6 displays performance for each of the subtasks in standard layered learning. When we compare the sharp decrease in performance of refueling and taking off and surveying during the first 2,000 epochs of the final layer, we see that CLL's subtask performance drops are not as drastic as in layered learning. Additionally, refueling subtask performance quickly stabilizes, while take off and survey performance gradually increases. Table 4.11 displays

the average performance of every subtask at the end of each layer; Table 4.12 displays their standard deviations. Layered learning ended its learning with refueling proficiency of 0.2888 and take off and survey with 0.4365. With CLL, refueling averaged 0.49 and take off and survey averaged 0.5736 at the end of the final layer; that is a 70% and 31% increase per subtask performance, respectively. From these results, we conclude that CLL outperformed standard layered learning.

Table 4.11: (1+1) EA: Average CLL Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.8644	0.0291	0.0046
Layer 2	0.3813	0.7423	0.1184
Layer 3	0.4900	0.5736	0.6310

Table 4.12: (1+1) EA: Standard Deviation of CLL Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.1332	0.0604	0.0076
Layer 2	0.2225	0.0394	0.0643
Layer 3	0.2227	0.0961	0.1420

Corresponding to CLL’s improved performance over layered learning’s direct policy search, the forgetting metrics confirm that significantly less negative forgetting took place between the paradigm and its augmentation. Layered learning received an MFM Even negative forgetting score of -0.1514 (.1164) while CLL scored -0.1146 (.07915). A two-tailed Z-test with an α of 0.05 showed that CLL’s negative forgetting decrease was significant with a z-value of -2.6145 and a p-value of 0.0089. MFM 50 provided the same conclusion that CLL’s negative forgetting was lessened: layered learning’s MFM 50 score was -0.1139 (0.087) and CLL produced -0.0864 (0.5904), with z-value of -2.6155 and p-value 0.0089. Tables 4.3 and 4.4 displays all of the forgetting averages and standard deviations.

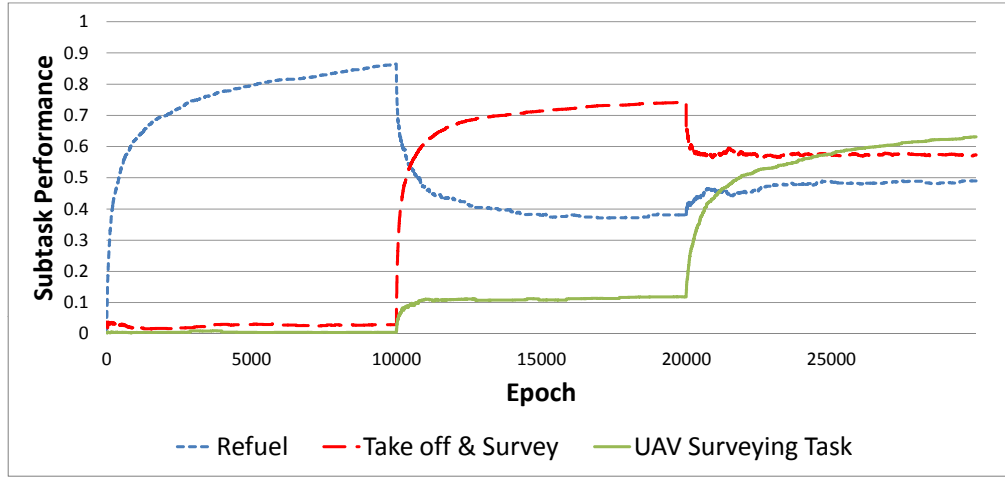


Figure 4.1: (1+1) EA Average Subtask Performance with CLL of All Runs

The decrease in the amount of negative forgetting experienced by CLL is directly related to the number of pairs that were repaired in the final layer. Of the 13.86 pairs per policy obtained during layer 1's optimization of the refueling subtask that were modified in layer 2, CLL reverted 3.67 pairs to their layer 1 mapping in layer 2, resulting in a 194% increase over layered learning's 1.25 repairs per policy. It should be noted that layered learning averaged 14.58 (4.6478) pair changes from layer 1 to layer 2. Because of the increase in repaired pairs that were reverted to their mappings originally found in layer 1, there was an increase in probability that some of those relearned pairs were beneficial knowledge to the refueling subtask and aided in its performance at the end of learning.

To further compare layered learning to CLL under direct policy search, we turn to the transfer learning metrics transfer regret, transfer ratio, and calibrated transfer ratio. Used to compare our control (layered learning) to our experimental transfer learning approach (CLL), all three metrics confirm that in our experiments, CLL outperformed layered learning. In order to make the metrics applicable to both our EA and Q-learning algorithms, we used an abbreviated form of these metrics,

where instead of tracking across all of the epochs, we sampled overall task performance at three critical points of learning: the end of each layer. This reduced sample was used because Q-learning employs an ϵ -increasing technique that purposely does not use the highest Q-value action in non-terminal epochs of each layer. All factors and variables for these metrics were samples from the same three points for both layered learning and CLL. Transfer ratio was 1.1026, favoring the CLL experimental approach because its performance integral ratio was greater than 1. Both transfer ratio (0.1127) and calibrated transfer ratio (0.0323) also indicated that better performance occurred in CLL because their values were greater than zero. In every respect, CLL resulted in an improvement upon standard layered learning and clearly made measurable progress mitigating the stability-plasticity imbalance.

Value Search Performance Evaluation

The complementary learning system's augmentation to layered learning provided a boost in task performance going into the final layer. By updating the Q-table based on averages from the indexed collection, the policy increased the chance to reevaluate beneficial pairs that have seen their Q-values decrease as a result of the take off and survey layer. Recall that the policy experiences all of the subtask proficiency changes as standard layered learning up to the third layer, only diverging from the original process by updating Q-values based on the indexed collection before the final layer begins. By having every other layered learning process untouched, this made it clear that the only difference between layered learning and CLL was this utility seeding inserted between layer 2 and 3, and that it is actually the cause for the performance increase.

Table 4.13 displays the average subtask performance at the end of each layer. Table 4.14 displays their standard deviations. From these layer transitions, we see that again, optimal refueling performance was gained in layer 1, followed by an extreme loss of capability and an average refueling

score of 0.0, and refueling proficiency was regained to 0.7506 (0.0062) by the end of layer 3. The take off and survey subtask performance is similar to that of standard layered learning as well. In this subtask, performance went from 0.0476 in layer 1 to 0.8187 in layer 2, finalizing at 0.7321. Again, the similarity between the standard layered learning and CLL lies in the final layer’s updated Q-table and how it boosted Q-values of more beneficial pairs than the standard approach.

Table 4.13: Q-learning: Average CLL Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	1.0000	0.0476	0.0648
Layer 2	0.0000	0.8187	0.1290
Layer 3	0.7506	0.7321	0.4011

Table 4.14: (Q-learning: Standard Deviation for CLL Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.0000	0.0612	0.1300
Layer 2	0.0000	0.0270	0.0042
Layer 3	0.0062	0.0245	0.0831

Once again, we evaluated the experimental approach of CLL to our control, standard layered learning, using the transfer learning metrics. The measures produced a transfer ratio of 1.0923, transfer regret of 0.1189, and a calibrated transfer ratio of 0.0205. Each of these values indicates there was in improvement in task performance using CLL over the control approach. These results further conclude that under these experiment conditions and overall task, CLL improved standard layered learning in both direct policy and value search.

Complementary Layered Learning Applied to Balancing Techniques

Due to the success in improving task performance of standard layered learning with the proposed complementary augmentation in both direct policy search and value search, complementary layered learning was applied to subtask rehearsal and concurrent layered learning. The aim of the application was to determine if the complementary learning system can be further improved by combining it with a more specialized balancing technique.

For these evaluations, minimal changes were necessary to apply CLL to subtask rehearsal and ConcLL. In fact, both rehearsal and ConcLL's layer configuration and implementation remained the same as described earlier in this chapter. CLL was applied by adding the same indexed collection as in the proposed CLL for both the (1+1) EA and Q-learning cases and following the pair integration and policy update procedures that were used to augment the complementary system to standard layered learning. Again, the same UAV Surveying task, 100 trials, layer configuration, and experiment settings were used to test the effectiveness of CLL with subtask rehearsal (*CLL-Reh*) and *complementary concurrent layered learning (CLL-Conc)*.

Direct Policy Performance

Of all of the approaches studied in this work, CLL-Reh with the EA produced the highest task performance, averaging 0.7139 (0.1447); that is a 28% improvement over standard layered learning, a 28% improvement to standard layered learning with rehearsal, and a 13% increase over CLL.

The difference in refueling subtask performance indicates how well each of the approaches retains and relearns older information. Figure 4.2 graphs the refueling performance of layered learning, CLL, and the two rehearsal techniques. From the graph, we see that layered learning's refueling ability decreases rapidly once layer 2 begins (after the first 10000 epochs) and does not regain

lost proficiency. Standard rehearsal regains lost proficiency during its second refueling layer but also sees the subtask’s performance decrease in the subsequent layer. CLL experiences the same refueling decrease as standard layered learning but regains some of its forgotten proficiency in layer 3. Complementary layered learning with rehearsal saw a combination of the positives of CLL and rehearsal: after the second refueling layer, refueling performance was high, and after the initial performance drop in layer 3 to .6481, subtask proficiency stabilized and began to increase to .7831.

Adding the complementary system to ConcLL did not enhance task performance over standard layered learning under direct policy search. With the (1+1) EA, CLL-Conc averaged 0.567 (0.1264), not improving performance over standard layered learning and significantly underperformed compared to CLL’s average of 0.631. A positive was that CLL-Conc did improve performance over concurrent layered learning by 5%. Table 4.10 contains task averages and standard deviations of the complementary direct policy search approaches. Tables 4.15 through 4.18 displays average and standard deviation of the task performance per layer completion for each of the direct policy search approaches.

Table 4.15: (1+1) EA: Average CLL-Reh Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.8206	0.0242	0.0062
Layer 2	0.3344	0.7423	0.1075
Layer 3	0.8488	0.5461	0.0959
Layer 4	0.6769	0.5617	0.7139

Table 4.16: (1+1) EA: Standard Deviation of CLL-Reh Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.1217	0.0398	0.0109
Layer 2	0.2287	0.0368	0.0119
Layer 3	0.1244	0.1145	0.0460
Layer 4	0.1865	0.0945	0.1447

Table 4.17: (1+1) EA: Average CLL-Conc Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.8869	0.0278	0.0056
Layer 2	0.8975	0.0936	0.0168
Layer 3	0.9075	0.1374	0.0230
Layer 4	0.9000	0.1906	0.0372
Layer 5	0.8988	0.2385	0.0448
Layer 6	0.8825	0.2849	0.0493
Layer 7	0.7388	0.3963	0.5670

Table 4.18: (1+1) EA: Standard Deviation of CLL-Conc Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.1078	0.0515	0.0089
Layer 2	0.1113	0.0887	0.0291
Layer 3	0.1040	0.1130	0.0194
Layer 4	0.1035	0.1201	0.0431
Layer 5	0.0975	0.1207	0.0559
Layer 6	0.1112	0.1319	0.0435
Layer 7	0.1836	0.1123	0.1264

To determine what happened to the policies during the (1+1) EA using CLL-Reh and CLL-Conc and why there were improvements in task performance, we analyzed the amount of pair repairing that took place in the final layer per policy. CLL-Reh repaired 3.81 (2.4071) refueling pairs lost during the second layer and 3.72 (2.4903) take off and surveying pairs lost during the second refueling layer. Standard layered learning with subtask rehearsal recovered 1.26 (1.2052) refueling pairs and 1.41 (1.0592) take off and survey pairs. Similarly, CLL-Conc repaired 2.46 (2.0514) refueling pairs from layer 1 while ConcLL only repaired 1.15 (1.1948). Because of the increased amount of refueling pairs repaired in the final layer, especially compared to standard layered learning’s average of repaired refueling pair repairs of 1.25 (1.252), it is concluded that the complementary system did improve task performance by reintroducing lost beneficial in the final layer.

Finally, from the forgetting metrics of MFM Even and MFM 50, CLL-Reh experienced significantly less negative forgetting than those of standard layered learning, standard layered learning with subtask rehearsal. The improvement in these metrics coupled with what was observed during layer transitions means CLL-Reh was able to recover lost proficiency more effectively than standard layered learning and subtask rehearsal. Table 4.19 displays the forgetting metrics for the complementary systems and standard layered learning using direct policy search.

Table 4.19: (1+1) EA: Forgetting Metrics Values for the Complementary Systems and Standard Layered Learning

	LL	Rehearsal	ConcLL	CLL	CLL-Reh	CLL-Conc
PD	.00002	.00002	.00002	.00002	.00002	.00002
DFM Even	.00001	.00002	.00002	.00002	.00002	.00002
MFM Even	-.15143	-.14946	-.07081	-.11462	-.11732	-.02936
DFM 50	.00001	.00002	.00002	.00002	.00002	.00002
MFM 50	-.11385	-.10033	-.04149	-.08635	-.07933	-.01741

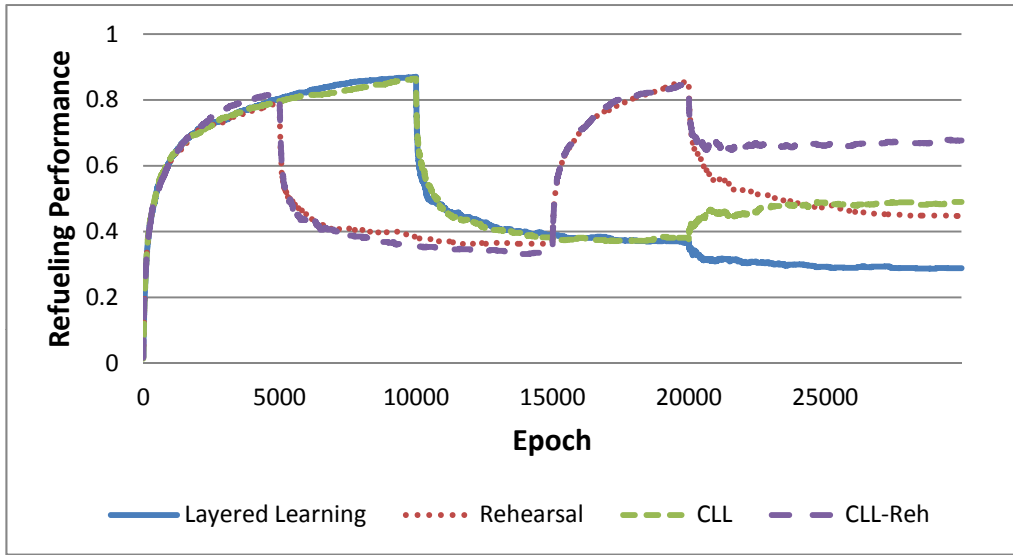


Figure 4.2: (1+1) EA Refueling Subtask Performance

Value Search Performance

When applied to value search, CLL-Reh and CLL-Conc did not improve performance of their corresponding standard versions, but they did improve standard layered learning. CLL-Reh produced an 8% task performance increase over standard layered learning while no significant difference was gained between layered learning with rehearsal and CLL. CLL-Conc generated the highest task performance of any value search approach, although not statistically different than that of ConcLL. CLL-Conc’s task performance of 0.4892 (0.1337) was higher than standard layered learning’s 0.3692 and CLL’s performance score of 0.4011 but did not significantly improve on ConcLL’s task performance of 0.4868. So, the more general CLL method is able to improve upon standard layered learning, while at the same time, it will not impede the success of more specialized approaches. Table 4.10 displays all task averages, standard deviations, and Z-test results for the value search experiments.

Comparing corresponding subtask rehearsal to CLL-Reh and ConcLL to CLL-Conc, subtask per-

formance at each layer completion was extremely similar. This similarity created an understanding of why overall task performance did not improve with the addition of a complementary system: the utility seeding before the final layer did not boost the Q-values of the beneficial pairs enough to make a positive impact on the policy. Tables 4.20 through 4.23 displays the average and standard deviation of the task performance for each approach per layer completion.

Table 4.20: Q-learning: Average CLL-Reh Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	1.0000	0.0395	0.0517
Layer 2	0.0000	0.8241	0.1290
Layer 3	1.0000	0.7053	0.9489
Layer 4	0.7506	0.7311	0.4023

Table 4.21: Q-learning: Standard Deviation of CLL-Reh Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.0000	0.0510	0.1054
Layer 2	0.0000	0.0275	0.0041
Layer 3	0.0000	0.0270	0.0892
Layer 4	0.0062	0.0265	0.0347

Table 4.22: Q-learning: Average CLL-Conc Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	1.0000	0.0464	0.0489
Layer 2	0.2144	0.7746	0.2276
Layer 3	0.1469	0.7762	0.1614
Layer 4	0.1306	0.7751	0.1533
Layer 5	0.1413	0.7774	0.1463
Layer 6	0.1444	0.7807	0.1285
Layer 7	0.7606	0.7343	0.4892

Table 4.23: Q-learning: Standard Deviation of CLL-Conc Subtask Performance Per Layer Completion

	Refuel	Take Off & Survey	UAV Surveying Task
Layer 1	0.0000	0.0562	0.1158
Layer 2	0.1766	0.0354	0.1693
Layer 3	0.1333	0.0328	0.0890
Layer 4	0.1302	0.0300	0.0792
Layer 5	0.1230	0.0328	0.0767
Layer 6	0.1246	0.0315	0.0141
Layer 7	0.0235	0.0206	0.1337

Discussion

From the results of all of the direct policy search and value search experiments, we conclude a few points about the evaluated existing and proposed solutions. First, due to the special search dynamics of value search on this problem, it benefited the most from the existing techniques of subtask rehearsal and ConcLL. Layered learning’s task performance was increased by 11% with subtask rehearsal and by 32% with ConcLL. The Q-learning implementation’s sensitivity to plasticity allowed these approaches to learn new subtasks quickly, and the use of a utility table kept beneficial knowledge accessible towards the end of the learning process.

Second, subtask rehearsal and ConcLL’s success did not transfer well to direct policy search, where the search dynamics were very different. Neither technique enhanced standard layered learning task performance in this case because the stability-plasticity imbalance was not mitigated. With the employed (1+1) EA, these techniques continued to suffer from the same negative effects as layered learning and could not retain or relearn lost beneficial knowledge any better.

Third, the proposed complementary learning system’s augmentation to layered learning addressed the disadvantages of rehearsal and ConcLL in direct policy search. The enhanced version of layered learning (CLL) increased task performance in the learning paradigm by 13% by reintroducing pairs that were present when subtasks were at their best observed proficiencies.

Fourth, the combination of the complementary system and subtask rehearsal or ConcLL did not impede Q-learning and resulted in improvements in the EA. In fact, CLL-Reh increased standard layered learning’s task performance by 28%, producing the best results in all of the tested approaches. The addition of the complementary system to ConcLL improved task performance of concurrent layered learning, which suggests that when specialized methods suitable to particular search dynamics are effective, CLL did not interfere with its efficacy.

Finally, all of these results suggest that concurrent layered learning can demonstrate significant improvement over standard layered learning in value search while CLL-Reh can provide the greatest enhancement with direct policy search. This suggests that CLL may be a good general technique to help mitigate negative forgetting, which can then be combined effectively with another technique more tailored for a specific search type.

CHAPTER 5: CONCLUSION

This dissertation contributes a deeper understanding and an enhancement to layered learning, a machine learning paradigm that is being used to develop autonomous robotic and computer-based systems. These contributions were achieved by demonstrating that a previously unexplored problem exists and hinders the paradigm’s effectiveness, identifying causes and effects of the problem, analyzing existing solutions and proposes new ones, and recommending effective uses for each solution.

In more detail, this work demonstrated that a previously unaddressed problem, a stability-plasticity imbalance, can exist in a decomposition-based reinforcement learning paradigm. We provided the demonstration in two, very different types of problems: the first analyzed the effects of imbalance when trying to solve Boolean-logic optimization problems; the second demonstration optimized the policies in a more realistic, cooperative multi-agent problem. In the latter, two different types of reinforcement learning were used to drive the policy optimization: a direct policy search using a simple (1+1) EA and a value search with Q-learning. From these demonstrations, we identified the main cause of the imbalance to be the layer transitions that change the performance evaluation criterion and causes older subtask performance to be lost when newer subtasks are being optimized on. Additional causes include the employed learning algorithm and conflicting subtasks used in the task decomposition. The most significant effect of the imbalance was the premature loss of beneficial knowledge that resulted in overall task performance lost (negative forgetting), limiting the effectiveness of the learning paradigm.

Next, we analyzed the effectiveness of applying an existing balancing solution (subtask rehearsal) to layered learning and an enhancement to layered learning (concurrent layered learning); both are known from the literature review to have improved learning performance in other systems and

problems. Under the cooperative multi-agent problem, these had mixed success that depended largely on particularities of the search dynamic. A complementary system was then proposed to improve the direct policy search's performance by seeding the policy change in the final layer with pairs that existed in the policy when each subtask was at their best observed performance. This augmentation significantly improved task performance compared to standard layered learning. The complementary system was then applied in conjunction with subtask rehearsal and concurrent layered learning. In the direct policy search learning, subtask rehearsal with complementary layered learning improved task performance substantially when compared to standard layered learning. When a complementary system was combined with subtask rehearsal and concurrent layered learning with value search, the performance of both techniques with standard layered learning was not impeded.

From these experiment results, we concluded that the proposed, simple complementary system improved performance under both types of reinforcement learning because of its ability to reintroduce forgotten beneficial knowledge. Specifically, complementary subtask rehearsal produced the best performance boost for direct policy search. Concurrent layered learning with either standard or complementary layered learning generated the highest task performance results for the value search.

Contributions

A number of contributions and major findings are provided from this work:

First Study of Imbalance in Layered Learning

Although the stability-plasticity dilemma is analyzed in other learning systems cited in the literature review, this work provides the first explicit study of imbalance in layered learning. We show that there is a problem and why, analyze potential solutions, and recommend solutions for the two types of reinforcement learning studied in our experiments.

Improvement in Layered Learning Effectiveness

With the examination of subtask rehearsal, concurrent layered learning, and complementary layered learning as mitigation techniques of negative forgetting, we provide methods to improve task performance in layered learning, ultimately enhancing the paradigm’s learning effectiveness. This improvement is a major contribution because a wide variety of autonomous systems rely on layered learning. By not only demonstrating that there is a problem but also providing an improvement to the learning paradigm, those who will depend on the learning system in the future may have an even more effective learning than previously experienced.

Literature Review on the State of Layered Learning

The literature review reveals and confirms a number of critical aspects of layered learning. First, layered learning is a viable alternative to monolithic learning, especially when the solution space’s gradient is steep. Second, the paradigm is being applied to and sees success in a variety of different problem domains. Next, layered learning is similar to other decomposition-based reinforcement learning techniques in how it breaks down a complex task into simpler problems, uses incremental learning to solve those simpler subproblems, and shares characteristics with shaping and transfer learning; it is unique in its abstractness and reliance on other approaches to facilitate that actual

learning. Fourth, the literature revealed that there is a problem, the stability-plasticity dilemma, that is a challenge in other decomposition-based reinforcement learning. Finally, other systems have addressed the dilemma with mitigation techniques, but imbalance in layered learning has been unexplored until this work.

Introduction of New Forgetting and Policy Change Metrics

As part of the imbalance demonstration, we have developed two forgetting metrics: the direct forgetting metric (DFM) and maximized forgetting metric (MFM). Based on our used performance-based definitions of negative, positive, and neutral forgetting, these metrics allow researchers to simply classify and measure the amount of suspected forgetting that has taken place during reinforcement learning. Additionally, we have introduced a set of measures that allow for a quantifiable way to interpret the impact of policy change across layer transitions. These metrics were discussed in the section on Policy-Based Metrics in Chapter 3. Included in this set are the total number of changed, unchanged, new, permanently damaged, and repaired pairs per layer. All of these metrics are effectively used in our analysis comparing standard layered learning to our proposed solutions. Moreover, we believe many may be useful for diagnosing similar problems in a wide array of decomposition-based RL, not just layered learning.

Overlapping Feasibility Demonstrated

To demonstrate an imbalance and facilitate forgetting, all of our experiments used layer overlapping; this was seen in conflicting optimality requirements in the *leading ones* and *trailing zero* subtasks and then in the *refueling* and *take off and survey* subtasks. Even though more design consideration has to take place when using layer overlapping, we believe it is a more natural method of learning than layer isolation. The experiments in this work demonstrate that layer overlapping

can be effective at learning complex tasks. Additionally, this dissertation provides methods to mitigate the negative effects of stability-plasticity imbalance that are present in overlapping but not in layer isolation, now making the layer design approach more attractive and feasible than previously viewed.

Imbalance Awareness to Other Decomposition-Based Systems

Because layered learning shares many similarities with other decomposition-based reinforcement learning systems, a demonstration of stability-plasticity imbalance in layered learning and proposed solutions may be applicable to these other learning approaches. For example, an imbalance favoring plasticity was shown to be prevalent in artificial neural networks. Complementary learning systems were applied and shown to be successful solutions at balancing stability and plasticity in neuro-evolution. In this work, we show that layered learning suffers from a similar challenge and our complementary layered learning was directly inspired by these preceding works. Other, similar learning approaches may benefit from our study by being able to apply our proposed solutions to bring balance to their systems.

Future Work

Because this dissertation serves as the first explicit investigation of stability-plasticity imbalance in layered learning, there are still unexplored areas of imbalance in the paradigm. For example, we primarily focused on understanding the imbalance in direct policy search, although we also analyzed problems and solutions using value search. Because so much focus was placed on direct policy search, our proposed complementary system may not have been optimal in mitigating negative forgetting when using Q-learning. This is suggested by how subtask rehearsal and concurrent

layered learning did not see a statistical difference in performance when the complementary system was applied. It is suspected that the integration of the indexed collection's average Q-values back into the Q-table before the final layer begins is too aggressive. For example, a pair A may earn a very high Q-value in the first layer and is placed in to the indexed collection. Meanwhile, pair B , that maps to a different action but at the same state as A , may receive smaller utility values but is in the policy at the end of multiple layers. During the Q-table updates in between the end of the penultimate and ultimate layers, A will always be seeded with a higher Q-value, although B has persisted in the policy as the highest utility valued pair for that state over multiple layers. This situation can isolate B from being reevaluated in the final layer or require a number of simulations to elapse before B 's Q-value is restored as the highest for that state or for B to be chosen again for action selection.

A second extension will be to analyze the balance of stability and plasticity in layered learning using more sophisticated algorithms. In our study, we purposely choose the (1+1) EA and the type of Q-learner because they are simple learning algorithms that make it less difficult to analyze what happens during the learning process. Using more complex algorithms may provide deeper understanding of the dilemma and layered learning. Additionally, the same goals maybe achieved if the dilemma was studied in layered learning with more types of problems; this would allow the paradigm to be evaluated on a diverse set of tasks and conditions not yet explored.

Another future work includes examining the proposed complementary learning system on direct policy searches where standard layered learning struggled in performance because of suspected imbalance. For example, in the preliminary work, layered learning's effectiveness was compared to that of monolithic learning. In the experiments, monolithic and the decomposition approach both learned at a similar rate, having no significant difference between the two performances. This result made monolithic learning seem more attractive of the two because of the reduction in design time required in order to produce the same task performance and simplicity of the approach. Applying

complementary layered learning to this predator-prey, direct policy search problem would provide further consensus of the efficacy of the proposed approach in a system that is suspected of suffering from stability-plasticity imbalance.

Next, the proposed forgetting metrics have shown their usefulness when comparing two direct policy searches. Unfortunately, the value searches performed in this work, like many others, use a mechanism that changes the ϵ as the number of simulations changes within a layer. Our use of ϵ -decreasing makes comparing two epoch results difficult because the policy is not guaranteed to return the highest Q-valued action for a state, a random action may be chosen. Therefore, the metrics must be revisited in order to factor in the action-selection variability experienced with such value-based strategies.

Finally, we would like to see the solutions and analysis made in this dissertation be applied to other decomposition-based reinforcement learning approaches. The complementary learning systems used to mitigate imbalance in neural networks have inspired the proposed CLL solution here. Applying our tested solutions and analysis to other systems may reveal that they translate well to other decomposition-based approaches, improving their understanding of stability-plasticity dilemma and efficacy.

Although these future works are all beyond of the scope of this dissertation, they are expected to provide more understanding of stability-plasticity imbalance in layered learning once investigated.

LIST OF REFERENCES

- [1] Ahmed, T., Oreshkin, B., Coates, M.: Machine learning approaches to network anomaly detection. In: Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques, pp. 7:1–7:6 (2007)
- [2] Barrett, S., Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning on a physical robot. In: Ninth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS - ALA) (2010). URL <http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127026>
- [3] Borisovsky, P.A., Ereemeev, A.V.: A study on performance of the (1+1)-evolutionary algorithm. In: FOUNDATIONS OF GENETIC ALGORITHMS, 7, pp. 271–287. Morgan Kaufmann (2003)
- [4] Brooks, R.: Intelligence without representation. *Artificial Intelligence* **47**, 139–159 (1991)
- [5] Cherubini, A., Giannone, F., Iocchi, L.: Robocup 2007: Robot soccer world cup xi. chap. Layered Learning for a Soccer Legged Robot Helped with a 3D Simulator, pp. 385–392. Springer-Verlag, Berlin, Heidelberg (2008)
- [6] Dayan, P., Hinton, G.E.: Feudal reinforcement learning. In: Advances in Neural Information Processing Systems 5, pp. 271–278. Morgan Kaufmann (1993)
- [7] Dietterich, T.G.: The maxq method for hierarchical reinforcement learning. In: In Proceedings of the Fifteenth International Conference on Machine Learning, pp. 118–126. Morgan Kaufmann (1998)
- [8] Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* **13**, 227–303 (2000)

- [9] Erez, T., Smart, W.D.: What does shaping mean for computational reinforcement learning? In: Proceedings of the 7th IEEE International Conference on Development and Learning (ICDL 2008), pp. 215–219 (2008)
- [10] Fidelman, P., Stone, P.: Layered learning on a physical robot (2005)
- [11] Fontenla-Romero, O., Guijarro-Berdiñas, B., Pérez-Sánchez, B., Alonso-Betanzos, A.: A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recogn.* **43**(5), 1984–1992 (2010). DOI 10.1016/j.patcog.2009.11.024. URL <http://dx.doi.org/10.1016/j.patcog.2009.11.024>
- [12] Freedman, S.T., Adams, J.A.: Human-inspired robotic forgetting: Filtering to improve estimation accuracy. In: Proceedings of the 14th IASTED International Conference on Robotics and Applications, pp. 434–441 (2009)
- [13] French, R.M.: Catastrophic forgetting in connectionist networks: Causes, consequences and solutions. In: Trends in Cognitive Sciences, pp. 128–135 (1994)
- [14] French, R.M.: Pseudo-recurrent connectionist networks: An approach to the "sensitivity-stability" dilemma. *Connection Science* **9**, 353–379 (1997)
- [15] Fukushima, K.: A hierarchical neural network model for associative memory. *Biological Cybernetics* **50**, 105–113 (1984). URL <http://dx.doi.org/10.1007/BF00337157>
- [16] Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. *Adaptive Behavior* **5**, 5–317 (1997)
- [17] Gorski, N.A., Laird, J.E.: Evaluating evaluations: A comparative study of metrics for comparing learning performances. Tech. rep., Computer Science Department, Rutgers University,

Center for Cognitive Architecture, University of Michigan, 2260 Hayward Ave, Ann Arbor, Michigan 48109-2121 (2009)

- [18] Gustafson, S.M., Hsu, W.H.: Layered learning in genetic programming for a cooperative robot soccer problem. In: J.F. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A. Tettamanzi, W.B. Langdon (eds.) *EuroGP, Lecture Notes in Computer Science*, vol. 2038, pp. 291–301. Springer (2001)
- [19] Hattori, M.: Avoiding catastrophic forgetting by a dual-network memory model using a chaotic neural network (2009)
- [20] Hsu, W.H., Gustafson, S.M.: Genetic programming for layered learning of multi-agent tasks. In: E.D. Goodman (ed.) 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, pp. 176–182. San Francisco, California, USA (2001). URL <http://www.cs.nott.ac.uk/~smg/research/publications/gecco-2001.pdf>
- [21] Hsu, W.H., Gustafson, S.M.: Genetic programming and multi-agent layered learning by reinforcements. In: In Genetic and Evolutionary Computation Conference, pp. 764–771. Morgan Kaufmann (2002)
- [22] Hsu, W.H., Harmon, S.J., Rodriguez, E., Zhong, C.: Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In: Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference (2004)
- [23] Jackson, D., Gibbons, A.: Layered learning in boolean gp problems (2007)
- [24] Jin, Y., Sendhoff, B.: Alleviating catastrophic forgetting via multi-objective learning. In: International Joint Conference on Neural Networks, pp. 6367–6374. IEEE Press (2006)
- [25] Kehoe, E.J.: A layered network model of associative learning: Learning to learn and configuration. *Psychological Review* **95**(4), 411 – 433 (1988). URL

<http://ezproxy.lib.ucf.edu/login?URL=http://search.ebscohost.com.ezproxy.lib.ucf.edu/login.aspx?direct=true&db=pdh&AN=rev-95-4-411&site=eds-live&scope=site>

- [26] Kira, Z., Arkin, R.C.: Forgetting bad behavior: Memory management for case-based navigation. In: Proc. IROS-2004, pp. 3145–3152 (2004)
- [27] Kocsis, L., Szepesvari, C.: Bandit based monte-carlo planning. In: In: ECML-06. Number 4212 in LNCS, pp. 282–293. Springer (2006)
- [28] Koychev, I.: Gradual forgetting for adaptation to concept drift. In: In Proceedings of ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning, pp. 101–106 (2000)
- [29] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
- [30] Lieber, J.: A criterion of comparison between two case bases. In: Proceedings of the nd European Workshop on Case-Based Reasoning, pp. 87–100. Springer-Verlag (1994)
- [31] Lin, L.J.: Hierarchical learning of robot skills by reinforcement. IEEE International Conference on Neural Networks pp. 181–186 (1993). URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=298553>
- [32] Luke, S., Hohn, C., Farris, J., Jackson, G., Hendler, J.: Co-evolving soccer softbot team coordination with genetic programming. In: In Hiroaki Kitano, editor, RoboCup-97: Robot Soccer World Cup I, pp. 398–411. Springer-Verlag (1997)
- [33] Maes, P., Brooks, R.A.: Learning to coordinate behaviors. In: AAAI, pp. 796–802 (1990)
- [34] Markovitch, S., Scott, P.D.: The role of forgetting in learning. In: In Proceedings of the Fifth International Conference on Machine Learning, pp. 459–465. Morgan Kaufmann (1988)

- [35] Matwin, S., Oppacher, F., Constant, P.: Knowledge acquisition by incremental learning from problem-solution pairs. *Computational Intelligence* **5**(2), 58 (1989).
URL <http://ezproxy.lib.ucf.edu/login?URL=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=64275862&site=eds-live&scope=site>
- [36] McClelland, J.L., McNaughton, B.L., O'Reilly, R.C.: 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)
- [37] McCloskey, M., Cohen, N.: Catastrophic interference in connectionist networks: The sequential learning problem. *The psychology of learning and motivation* **24**, 109–165 (1989)
- [38] Menache, I., Mannor, S., Shimkin, N.: Q-cut - dynamic discovery of sub-goals in reinforcement learning. In: *Machine Learning: ECML 2002, 13th European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Computer Science*, pp. 295–306. Springer (2002)
- [39] Mondesire, S., Wiegand, R.P.: Evolving a non-playable character team with layered learning. In: *IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MDCM)*, pp. 52–59 (2011)
- [40] Mondesire, S., Wiegand, R.P.: Forgetting classification and measurement for decomposition-based reinforcement learning. In: *Proceedings of The 15th International Conference on Artificial Intelligence (ICAI'13)* (2013)
- [41] Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research* **11**, 241–276 (1999)

- [42] Morrison, C.T., han Chang, Y., Cohen, P.R., Moody, J.: Experimental state splitting for transfer learning. In: Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning (2006)
- [43] Nakayama, H., Yoshii, K.: Effectiveness of active forgetting in machine learning applied to financial problems (2002)
- [44] Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems 10, pp. 1043–1049. MIT Press (1998)
- [45] Potter, M.A.: Design and analysis of a computational model of cooperative coevolution (1997)
- [46] Robins, A.: Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science* **7**, 123–146 (1995)
- [47] Robins, A., Mccallum, S.: Catastrophic forgetting and the pseudorehearsal solution in hop-field type networks. *Connection Science* **10**, 121–135 (1998)
- [48] R.Paul, G., Terrence J., S.: Original contribution: Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks* **1**, 75 – 89 (1987). URL <http://ezproxy.lib.ucf.edu/login?URL=http://search.ebscohost.com.ezproxy.lib.ucf.edu/login.aspx?direct=true&db=edselp&AN=0893608088900238&site=eds-live&scope=site>
- [49] Salganicoff, M., Salganico, M.: Density-adaptive learning and forgetting. In: In Proceedings of the Tenth International Conference on Machine Learning, pp. 276–283. Morgan Kaufmann (1993)

- [50] Seipone, T., Bullinaria, J.: The evolution of minimal catastrophic forgetting in neural systems. In: the Twenty-Seventh Annual Conference of the Cognitive Science Society, pp. 1991–1996 (2005)
- [51] Shen, F., Tomotaka, O., Osamu, H.: An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks* **20**, 893 – 903 (2007). URL <http://ezproxy.lib.ucf.edu/login?URL=http://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0893608007001190&site=eds-live&scope=site>
- [52] Singh, S.: Transfer of learning across compositions of sequential tasks. In: *Machine Learning: Proceedings of the Eighth International Workshop*, pp. 348–352. Morgan Kaufmann (1991)
- [53] Singh, S.P.: Transfer of learning by composing solutions of elemental sequential tasks. In: *Machine Learning*, pp. 323–339 (1992)
- [54] Smyth, B., Keane, M.T.: Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. pp. 377–382. Morgan Kaufmann (1995)
- [55] Stone, P., Veloso, M.: A layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence* **12**, 165–188 (1998). URL <http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=126580>
- [56] Stone, P., Veloso, M.: Layered learning. In: *Proceedings of the Eleventh European Conference on Machine Learning*, pp. 369–381. Springer Verlag (1999)
- [57] Sutton, R.S., Barto, A.G.: *Introduction to Reinforcement Learning*, 1st edn. MIT Press, Cambridge, MA, USA (1998)

- [58] Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge, UK (1989)
- [59] Wegener, I., Witt, C.: On the behavior of the (1+1) evolutionary algorithm on quadratic pseudo-boolean functions (2000)
- [60] Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P.: Evolving keepaway soccer players through task decomposition. *Machine Learning* **59**(1), 5–30 (2005)
- [61] Whiteson, S., Stone, P.: Concurrent layered learning. In: *Proceedings of the Eleventh European Conference on Machine Learning*, pp. 369–381. Springer Verlag (2003)
- [62] Wiegand, R.P., De Jong, K.A., Liles, W.C.: The effects of representational bias on collaboration methods in cooperative coevolution. In: J.J. Merelo Guervós, *et al* (eds.) *Parallel Problem Solving from Nature, PPSN-VII*, pp. 257–268. Springer (2002)