


2013

Life Long Learning In Sparse Learning Environments

John Reeder
University of Central Florida

 Part of the [Computer 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

Reeder, John, "Life Long Learning In Sparse Learning Environments" (2013). *Electronic Theses and Dissertations, 2004-2019*. 2681.
<https://stars.library.ucf.edu/etd/2681>

LIFE LONG LEARNING IN SPARSE LEARNING ENVIRONMENTS

by

JOHN REEDER

B.S. Computer Engineering, University of Central Florida, 2005

M.S. Computer Engineering, University of Central Florida, 2008

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, FL

Summer Term
2013

Major Professor: Michael Georgiopoulos

© 2013 John Reeder

ABSTRACT

Life long learning is a machine learning technique that deals with learning sequential tasks over time. It seeks to transfer knowledge from previous learning tasks to new learning tasks in order to increase generalization performance and learning speed. Real-time learning environments in which many agents are participating may provide learning opportunities but they are spread out in time and space outside of the geographical scope of a single learning agent. This research seeks to provide an algorithm and framework for life long learning among a network of agents in a sparse real-time learning environment. This work will utilize the robust knowledge representation of neural networks, and make use of both functional and representational knowledge transfer to accomplish this task. A new generative life long learning algorithm utilizing cascade correlation and reverberating pseudo-rehearsal and incorporating a method for merging divergent life long learning paths will be implemented.

This thesis is dedicated to all my friends and family who have been so supportive while I've been eternally in school.

ACKNOWLEDGMENTS

I would like to thank my friends and family for being patient and supportive throughout this endeavor, and for the constant and mostly gentle pushing towards this goal. I would also like to thank my soon to be wife Mysti, whose love and support have been a blessing and a comfort during these final few years of my research. I would like to thank the Department of Defense, and the SMART fellowship for providing the funding for my school these last 5 years, and for giving me the chance to focus on my studies. Finally I'd like to thank Dr. Georgiopoulos for providing guidance and advice for nearly a decade starting with my undergraduate research experience all those years ago. I can literally say I wouldn't be in the position I am now without his influence.

TABLE OF CONTENTS

LIST OF FIGURES	xii
LIST OF TABLES	xxii
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: LITERATURE REVIEW	6
2.1 Life Long Learning	6
2.1.1 Functional Transfer	6
Multitask Learning	7
Learning from Hints	7
2.1.2 Representational Transfer	8
Meta-Neural Network	8
Discriminability Based Transfer	8
Learning one more thing	9
2.2 Generative Neural Networks	10
2.2.1 Neuroevolution	10

Neuroevolution of Augmenting Topologies	10
Symbiotic Adaptive Neuroevolution	13
2.2.2 Cascade Correlation	14
2.2.3 Knowledge Based Cascade Correlation	17
2.3 Multitask Learning Literature	21
2.3.1 Multi-task Learning	21
2.3.2 η MTL	25
2.3.3 Task Rehearsal	28
2.3.4 Context Sensitive MTL	32
2.4 Pseudo-Rehearsal	34
2.5 Learning from Observation and Experience	41
2.5.1 FALCONET	41
2.5.2 Multi-Agent Learning	44
CHAPTER 3: RESEARCH DESIGN AND METHODOLOGY	47
3.1 Applications of Sparse Life long learning	49
3.2 Methods & Implementation	57
3.2.1 MTL Impoverished Task Implementation	57

Unrelated Tasks	58
Unrelated tasks and CC	59
3.2.2 New Input Methods	60
3.2.3 Reverberated Dual CC Networks	63
Reverberated Cascade Correlation	64
3.2.4 Extending Dual CC Life Long Learning with KBCC	70
Merging Networks with Dual Reverberated KBCC networks	70
3.3 Simulation Implementation.	72
3.3.1 Game Scenarios	74
Chase Scenario	74
Obstacles Scenario	75
Chase-Obstacles Scenario	75
Chase-Shoot Scenario	75
Chase-Obstacles-Shoot	75
3.3.2 Scoring	76
3.3.3 Sensors and Actions	77
CHAPTER 4: RESULTS	84

4.1	Experimental Problems	86
4.1.1	Band Domain	87
4.1.2	Circle In Square	87
4.1.3	Linear Tasks	87
4.1.4	Glass Database	88
4.1.5	Dermatology Database	88
4.2	Experiments with Multi-task Learning and Cascade Correlation	88
4.2.1	Preliminary Experiments	89
	Preliminary Grid Search	89
	Method	89
	Results	89
	Preliminary Hidden Layer Size Experiment	90
	Method	91
	Results	91
4.2.2	MTL Impoverished Primary Task Experiments	91
	Method	91
	MTL Impoverished Primary Task Results	94

	MTL Impoverished Primary Task Results with Unrelated Tasks	95
4.2.3	CSMTL Impoverished Primary Task Experiment	96
	Method	97
	Results	97
4.2.4	CC MTL Conclusions	98
4.3	Dual CC Life Long Learning Experiments	99
4.3.1	New Input Experiments	100
	Method	100
	New Input Results	101
4.3.2	Dual CC Recall Experiments	105
	Method	106
	Dual CC Recall Results	106
4.3.3	Dual CC Consolidated Training Experiments	117
	Method	117
	Dual CC Consolidated Training Results	118
4.4	Generating Simulated Agents	122
4.4.1	Method	123

4.4.2	Simulated Results	126
4.4.3	Subjective Results For Generated Agents	128
	Method	128
	Results	128
4.5	Merging with Reverberated KBCC	130
4.5.1	Method	130
4.5.2	Merging Results	131
CHAPTER 5: CONCLUSION		136
5.1	Contributions	137
5.2	Future Development	139
5.2.1	Short-Term Tasks	139
5.2.2	Long-Term Tasks	140
LIST OF REFERENCES		142

LIST OF FIGURES

Figure 2.1: Cascade Correlation Network: The network starts with the output nodes fully connected. Candidate nodes are added as needed when the output training stagnates. Candidate nodes are cascaded meaning each new node is connected to all of the nodes previously added.	18
Figure 2.2: The structure of a KBCC Network. The sub-network H1 has weights connecting its inputs and outputs to the parent network, but its internal weights are not connected. The hidden layer node H2 is connected upstream to the inputs of the network and the outputs of the sub-network H1.	20
Figure 2.3: MTL vs STL networks: MTL and STL networks for a collection of tasks. The MTL network has a shared hidden layer connected to independent outputs for each task.	23
Figure 2.4: Task Rehearsal: Domain networks are used to provide secondary task targets for the primary networks MTL outputs through pseudo-examples. The η MTL algorithm is used to train the primary network. The domain networks allow the task-rehearsal system utilize the MTL technique without maintaining the previous tasks data.	30
Figure 2.5: CSMTL vs MTL networks: CSMTL networks have additional context inputs rather than additional task outputs. Using context inputs allows for the tasks to share representation at the output layer in addition to the hidden layer as in MTL.	34

Figure 2.6: Stage I and Stage II of the dual network reverberating pseudo rehearsal system of Bernard Ans. In stage I the first network, which has already been trained on the first task, is activated with random data and reverberated to create pseudo-items. These items are used to train Net 2. In stage II Net 2 is used to create reverberated pseudo-items that are interspersed with the new task's training data. New tasks are only learned during stage II, while stage I is used to back up the knowledge from Net I into Net 2. 39

Figure 2.7: CSMTL network set up for self-refreshing memory. The only modification needed to the structure of the network is the addition of auto-associative weights. The most straight forward implementation of this is to add a special set of auto-associative outputs. They are then trained in the same manner as the normal outputs, using the input values as the target values. 40

Figure 3.1: CC Life Long Learning scenarios for single users: Scenario 1: The single user could create an agent to chase a target while avoiding obstacles by first training it to approach a stationary target, then training it to chase a moving target, then finally training it how to avoid obstacles. Scenario 2: The user could create an agent for the same task by selecting existing agents with the chaser, and obstacle avoidance skills, then merging them together. 52

Figure 3.2: CC Life Long Learning large environment multi-user scenario. The large environment would be split up between several users. Each user would train an agent in their area of responsibility. The resulting agents would then be merged to share their knowledge of the different areas. 54

Figure 3.3: CC Life long Learning to consolidate practice and to combine user traits. Consolidating practice would allow several users to train an agent on an identical task. The goal would be to achieve a higher level of coverage of the agents state space, while minimizing user fatigue, and to overcome individual quirks of the human trainers. Combining user traits would allow a user architect to use multiple human trainers with different traits (aggressive or conservative) to create an agent with both traits.	55
Figure 3.4: Adding a new input to an existing CC network. This depicts the new structure only method. This method only attaches the new input to newly developed structure as the training continues. This is the most straight forward method, and would require no additional training steps.	62
Figure 3.5: Adding a new input while also adding weights to the existing structure. This would allow the new input to adapt the existing structure. In the case of mutually exclusive context inputs we do not have to worry that the new weights would affect previous tasks. This would require an additional training phase to adjust the new weights.	63
Figure 3.6: The process of the CC life long learning system as developed for this research. After the initial task is trained, the consolidate and new task training phases can be repeated as needed without having to modify the structure of the networks manually or creating any new networks.	65
Figure 3.7: CSMTL network set up for self-refreshing memory	66

Figure 3.8: The process of Silver’s CS MTL life long learning system. Between each new task and consolidation phase a new network must be created requiring the user to determine the appropriate topology. Additionally the system is only able to take advantage of the representational transfer mechanism during new task training and not during consolidation.	69
Figure 3.9: This figure shows a depiction of the auto-differentiation used in the KBCC algorithm. The partial derivatives are first calculated at the inputs and then propagated forward through each layer until the derivatives of the outputs have been calculated.	71
Figure 3.10: This figure depicts the merging process when only the original networks are available. The existing networks are reverberated to generate a new training set and then they are used as sub-networks to train a new dual KBCC network.	73
Figure 3.11: Depiction of the diverging and merging learning paths of the sparse life long learning system. Red tracks denote fast forward situations. These situations would be incremental learning situations handled by the standard life long learning system. The blue tracks represent instances where a merge would be necessary. These occur because the base network has been modified before the blue tracks have finished.	74
Figure 3.12: Screen shot of the OpenNERO Simulation. This scene depicts the chase scenario.	81
Figure 3.13: Screen shot of the OpenNERO Simulation. This scene depicts the obstacles scenario.	82

Figure 3.14: Screen shot of the OpenNERO Simulation. This scene depicts a replay of the chase-obstacles-shoot scenario	83
Figure 4.1: The effects of choosing the wrong hidden layer on MTL. In each data set the performance on the primary task is significantly degraded when a non optimal hidden layer size is chosen. The results are most pronounced in the band and linear data sets, where the networks perform no better than random selection.	92
Figure 4.2: The effects of the wrong hidden layer size for static networks compared to CC network for CSMTL problems.	93
Figure 4.3: MTL Impoverished Primary Task: STL vs MTL with Static and CC networks. This result shows that the CC network is still able to achieve similar results to the static network using the MTL technique. This shows that we can generate the structure of the network while using MTL as a source of inductive bias.	94
Figure 4.4: Results for η MTL in the impoverished primary task with unrelated secondary tasks. In the static case η MTL provides a benefit to each data set, with the benefit in the CirInSq and glass data sets being the most significant. In the results for the CC network using η MTL in the calculation of the candidate scores provided the most benefit in four of the five tasks. This would suggest that the weighting the candidate score based on how well it correlates with the primary and more related tasks is more beneficial than adjusting the gradients based on relatedness of the outputs.	96

Figure 4.5: Results for CSMTL Impoverished Primary Task. These results show that the CSMTL technique performs slightly better than MTL on the band task in both the static and CC cases. For the CirInSq and Linear data sets the results are split between the static and cc cases. CSMTL performs better on the linear data set on with the static network, while it performs better on the CirInSq data set on the CC network. One possible reason for this result could lie in the sensitivity of the data sets to the cascaded structure of CC. The results suggest that in the CirInSq case CSMTL takes advantage of the cascaded architecture of CC while it is hindered by the flat hidden layer in the fixed network. The opposite effect is apparent in the linear data set CSMTL performs poorly for the Glass and Dermatology data sets for both the Static and CC networks. 99

Figure 4.6: New Input Experiment: Classification error of the Iris data set using limited inputs and full inputs under two methods of adding new units. Both methods seem to provide the same level of improvement between the two feature sets. 102

Figure 4.7: New Input Experiment: Number of epochs required to learn the added features. Both techniques are again very similar in the number of epochs required.103

Figure 4.8: New Input Experiment: Classification error of the Linear data set using limited inputs and full inputs under two methods of adding new units. Both methods seem to provide the same level of improvement between the two feature sets. The performance from the limited set in the case is actually quite good. This is because the data set is almost separable using only the first input. This result shows the case of adding a new feature to an already decent classifier. 104

Figure 4.9: New Input Experiment: Number of epochs required to learn the added features. This shows the major difference between the two methods. In this set the first input was very good for the classification task. Adding the new input provided a modest boost to performance. In this case method 2 allowed the network to only adjust the existing structure to solve the problem requiring no additional hidden nodes. 105

Figure 4.10: First task recall during second task training for the Band data set. The case with no reverberation is obviously losing performance, while pseudo-rehearsal, and reverberated pseudo-rehearsal keep the recall error low. . . . 107

Figure 4.11: First task recall during second task training for the CirInSq data set. The case with no reverberation is obviously losing performance, while pseudo-rehearsal, and reverberated pseudo-rehearsal keep the recall error low. In this data set the 8 and 12 reverberations loss of performance is more pronounced. 108

Figure 4.12: First task recall during second task training for the Linear data set. The case with no reverberation is obviously losing performance, while pseudo-rehearsal, and reverberated pseudo-rehearsal keep the recall error low. In this data set the 4, 8, and 12 reverberations loss of performance is more pronounced. 109

Figure 4.13: First task recall during second task training for the second user's data sets. The first task learned in this case is the 'Chase' scenario, the second is the 'Chase-Obstacle' scenario. The user behaviors in both of these scenarios is nearly identical with the only major difference being the presence of obstacles. 113

Figure 4.14: First task recall during second task training for the third user's data sets.

The first task learned in this case is the 'Chase' scenario, the second is the 'Chase-Obstacle' scenario. The user behaviors in both of these scenarios is nearly identical with the only major difference being the presence of obstacles. 114

Figure 4.15: First task recall during second task training for the second user's data sets.

The first task learned in this case is the 'Chase' scenario, the second is the 'Obstacle' scenario. The user's behavior in these two scenarios is quite different. 115

Figure 4.16: First task recall during second task training for the third user's data sets. The

first task learned in this case is the 'Chase' scenario, the second is the 'Obstacle' scenario. The user's behavior in these two scenarios is quite different. 116

Figure 4.17: Results of consolidated training for the Linear data set task 1. The network

using 2 reverberations performs the best with 4 reverberations achieving similar results. The single network, and standard pseudo-rehearsal network don't seem to benefit from the consolidated training. 119

Figure 4.18: Results for consolidated training for the Circle in the Square data set task

1. The single network example denoted by the -1 column is not able to improve its performance with successive training tasks, while the standard pseudo-rehearsal and 2 reverberation networks improve after each training. . 121

Figure 4.19: The number of epochs used during each training phase for the Circle in the

Square data set. The increase in epochs during the pseudo-rehearsal and reverberated pseudo-rehearsal is readily apparent. This shows that while the performance is improving it comes at a cost of longer training times. . . . 122

Figure 4.20: Consolidation training results for user 1. These results look similar to what we saw with the classification data sets. The pseudo-rehearsal and 2 reverberation networks both see improvement after each training phase while the single network and higher levels of reverberation get initial improvements then lose some performance after the final training set. 123

Figure 4.21: Consolidation training results for user 2. These results show a different picture than the results for user 1. Like user 1 the standard pseudo-rehearsal shows an improvement in performance after each training phase, but unlike user 1, each of the reverberated networks loses performance after training phase 2 then gains some of the performance back after training phase 3. . . . 124

Figure 4.22: Consolidation training results for user 3. These results show a similar result to user 2. The standard pseudo-rehearsal performs the best, though it has a slight loss after the third phase, while the reverberated networks seem stagnate after each phase with a large spike in their variability. The single network gets a performance increase after the second training phase but loses it all and more after the third. 125

Figure 4.23: The classification error rate and the number of epochs to learn a single task using a previously trained network as a sub-network. The CC algorithm is used as comparison for learning the task without the sub-network. 132

Figure 4.24: The classification error rate and the epochs to learn a combined problem utilizing two previously trained networks. The CC network is used to show the comparison to learning the new task without using sub-networks 133

Figure 4.25: The classification error rate and the epochs to learn a merged task using only previously trained networks. The training data is generated by reverberating the sub-networks. These figures depict the out Box data set 134

Figure 4.26: The classification error rate and the epochs to learn a merged task using only previously trained networks. The training data is generated by reverberating the sub-networks. These figures depict the Band data set 134

Figure 4.27: The classification error rate and the epochs to learn a merged task using only previously trained networks. The training data is generated by reverberating the sub-networks. These figures depict the CirInSq data set 135

LIST OF TABLES

Table 2.1: Back-prop Definitions	26
Table 3.1: Differences between Silvers life long learning system and the Dual CC life long learning system	68
Table 3.2: User 1 scores. Scores for Chase are out of 30. The score for shooting is the number of his landed during the 30 second window.	77
Table 3.3: User 2 scores. Scores for Chase are out of 30. The score for shooting is the number of his landed during the 30 second window.	78
Table 3.4: User 3 scores. Scores for Chase are out of 30. The score for shooting is the number of his landed during the 30 second window.	78
Table 3.5: Sensors and Actions of the user agent. These are used to create an observa- tional data set from users running the scenarios.	80
Table 4.1: RProp & CC Training Parameters	86
Table 4.2: Experimental values derived from grid search.	90
Table 4.3: User 1 Agent simulation scores. The average scores for the consolidated agent, individual episode agents, and the user are shown for each scenario . .	126
Table 4.4: User 2 Agent simulation scores. The average scores for the consolidated agent, individual episode agents, and the user are shown for each scenario . .	126

Table 4.5: User 3 Agent simulation scores. The average scores for the consolidated agent, individual episode agents, and the user are shown for each scenario . . 126

Table 4.6: User 3 Subjective results. For all of the scenarios except Obstacles. The number of agents that passed for human and the total number of agents is reported. 128

Table 4.7: Subjective Results for User 1. For all of the scenarios except Obstacles. The number of agents that passed for human and the total number of agents is reported. 129

CHAPTER 1: INTRODUCTION

Life long learning is meant to mimic the way we learn tasks, in that we use our skills and knowledge previously acquired throughout our lifetimes to facilitate all new learning. In this way we are able to learn new things from very few, or even a single example. In life long learning one attempts to use previous experience to benefit future learning. This benefit could take the form of faster learning, better performance, or both. In either case the life long learner has an advantage over learners who start *tabula rasa*. This type of ability is very important for problems where new tasks have very little data, or in real time systems where examples filter in over time.

We are concerned in particular with sparse environments. A sparse environment can be defined as any learning environment where the opportunities for learning are spread out in time and/or space. This could mean that data is only available sporadically meaning the data generating process that is to be modeled happens infrequently or irregularly, or it could mean that the data is collected from a network of sensors spread out geographically with unreliable communications. A real-time sparse environment would be an extension of this concept where data needs to be acted on at the time of collection. Examples of this could be simulated agents in a large simulated world, or actual robotic agents in the real world. In this type of environment life long learning is even more important since events providing learning opportunities could be few and far between. We are interested in creating a life long learning system that allows a collection of agents in a sparse environment to learn throughout their lifetimes and to share their experience with their peers to increase the utility of the sporadic learning opportunities.

The life long learning aspect of this system will allow for the creation of intelligent agents that solve increasingly difficult problems by relying on previous experience. This type of learning could allow us to learn problems that are too complicated or time consuming to learn from scratch.

This type of complexification has been shown to lead to more efficient and elegant solutions in neural network domains in systems like NEAT [63] and CC [24]. They solve complex problems by starting with minimally connected networks and growing the solution to fit the problem. This is effective in neural networks for several reasons. First, it keeps the initial computation of the network to a minimum as smaller networks have fewer free parameters, leading to fewer weights to train and fewer computations to evaluate the network. In NEAT and CC this means that the early generations or epochs will require less computational time than other methods that begin with larger fixed structures. These smaller starting topologies also provide a smaller hypothesis space for the algorithms to search in the beginning, only expanding when necessary to improve the solution, leading to a quicker search for the solution.

Second, it is widely known that in neural networks smaller networks are less susceptible to over fitting and usually provide smoother solution surfaces [9]. Smaller networks are however more susceptible to local minima but in each of these approaches they add topology as necessary to overcome this [9]. The idea is that learning simple tasks first and adding complexity over time biases the learner in the next phase closer to the solution in the more complex domain, providing an advantage over a learner starting from scratch in the more complex domain. This idea is foundational in the areas of lifelong learning, and transfer learning.

In any learning system there is a method for knowledge representation. In our case we will be using neural networks. Neural networks have a long history in machine learning and have been shown to have many advantages. With the necessary structure they are able to approximate any continuous function [40, 9, 34]; they are also able to represent symbolic or categorical data when they are trained correctly [16, 17, 18]. Neural networks can also be used to represent large amounts of data, if they learn the underlying function well enough [52, 58, 47, 35, 11], and have been used to remember sequences [5, 23]. These qualities make them attractive candidates for a knowledge representation mechanism. There are disadvantages however; they can be very slow to learn,

especially when using back-propagation, and it can be difficult to understand how they represent the data. The issue of speed is less of a problem with recent developments, including faster back-propagation variants [22, 50], and generative techniques [24], while the black box nature can be somewhat overcome through rule extraction [18, 36] or using a network like ARTMAP [11], which has a geometric interpretation of its representation.

Life long learning in neural networks will rely heavily on transfer learning. Transfer learning is a family of techniques specializing in transferring information between learning scenarios. These can be problems from the same problem domain or they could be sequential tasks. Transfer learning in neural networks has been extensively investigated; it essentially deals with the correct application of bias. The techniques developed so far can be broken down into two separate categories, functional transfer and representational transfer. Representational transfer is when the structure of the network is used in the transfer process. Functional transfer is when the data, or knowledge in the network is used in the transfer. In our case we will use a mixture of both transfer techniques to facilitate the life long learning. The functional part of the transfer will come from multi-task learning techniques, while the representational transfer will come from the use of a cascaded architecture similar to cascade correlation. Neural networks have been used in previous life long learning methods developed by Thrun [70] and Silver [45, 60, 59, 58]. Thrun uses a combination of explanation based neural networks and an invariance network to learn how to classify concepts with fewer training samples, and Silver uses fixed neural networks and inductive bias from the multi-task learning technique of Caruana [12] to train new tasks using previously trained networks. Our system will be most comparable to Silvers CSMTL life long learning system [45].

The first contribution of this research will be a life long learning system using aspects of cascade correlation, context sensitive multi-task learning, and reverberating pseudo-rehearsal. The goal of the system is to build a single neural network representation to handle an arbitrary number of sequential learning tasks as they arrive without significant loss of performance on the previous

tasks. It will generate the structure of the network as it is needed, and will be able to incorporate new learning tasks. This system will overcome some of the problems of previous attempts at a life long learning system developed by Silver [58, 59]. Primarily by overcoming the need to define the long term storage network topology a priori, but also by providing a more elegant path for incorporating new tasks, and an enhanced consolidation mechanism through reverberated pseudo-rehearsal.

The second contribution will be a method of merging two divergent branches of life long learning using the generated representations of the previously mentioned system; this is a special case of transfer learning, and as such our method will fall under that category. This will be accomplished by merging the previously learned networks using the KBCC algorithm of Shultz [56], combined with reverberating pseudo-rehearsal. This will be a more direct method of consolidating separate learning paths than similar methods used in neuroevolution, like milestoneing [62], where separate learning populations are intermixed. Our method of consolidation will be able to take advantage of the nearly perfect recall of reverberating pseudo-rehearsal to transfer the experiences of both paths directly rather than having to settle for evolution selecting traits from each path. The combination of these two contributions will allow for a method that allows for the parallel learning and consolidation of experience by separate homogeneous agents in a sparse learning environment. The third contribution will be an open source software package developed to implement and test this system. This will facilitate the use of the proposed system in applications outside the scope of our work. It is available at <http://github.com/jreeder/jrnn>.

The remainder of this document will start with detail on the history of life long learning, multi-task learning, and transfer learning in neural networks. Following the literature review will be descriptions of our research design and methodology including a section on the broader problem area we intend to tackle with our system, and a section detailing our system development. After this we report on the results of our experiments, and finally we'll close with our conclusions and a

discussion on future development paths.

CHAPTER 2: LITERATURE REVIEW

2.1 Life Long Learning

Life long learning is a loose categorization of learning techniques that involve the transfer of knowledge between learning scenarios. Any system that contains transfer components can be said to do “life long learning” [71]. The unifying trait of these systems is the idea that learning should become easier the more you learn. In machine learning terms this means that learning the tenth task should be easier than learning the first task. Easier in the sense that learning new tasks should take fewer examples or less time. A theoretical frame work supporting this idea is provided by Baxter [7]. Our work will focus on the area of knowledge transfer for neural networks.

Knowledge transfer is an important area of research. Learning each new task from scratch is wasteful and does not take into account possible previous experience. For example an algorithm developed to recognize objects in a vision system might have to relearn its task under different conditions of visibility. If the training begins again from the beginning it is ignoring what could be useful information learned under the previous conditions. Transfer in this case could increase the speed at which the new conditions are learned, and could also lead to a more generalized object recognition algorithm able to deal with more varied lighting conditions. Transfer in neural networks can be broken down into two major categories; functional and representational [57].

2.1.1 *Functional Transfer*

In connectionist systems transfer is functional if the secondary and primary data are learned simultaneously. Where the primary data is the current problem to be solved and the secondary data are sets that can be useful in solving the primary. This is usually in the form of multiple data sources.

In this form of transfer the network may share representation between the primary and secondary tasks or it may not. Some forms of functional transfer are MTL and Learning from Hints.

Multitask Learning

Multitask learning [12] is a form of functional transfer because it uses data from multiple related tasks to facilitate knowledge transfer. The primary assumption in MTL is that each data set will carry information about the higher level domain as well as the specific task it represents. The domain information carried by all the related data sets is amplified in training because there are more examples. MTL utilizes this information by forcing the hidden layer of the network to find a representation that is mutually beneficial, while allowing the outputs to become task specific adaptations. MTL will be discussed in greater detail in section 2.3.1.

Learning from Hints

Learning from hints [1] is a technique that utilizes extra examples created from a set of rules. Examples of hints include symmetry and monotonicity. Hints are trained at the same time as normal examples similarly to MTL but they have separate error functions, one for each hint. The generation of the error function for the hint is very important, as this is the mechanism by which the information from the hint is transferred to the network. This is functional transfer in the same way MTL is functional transfer, because the hint examples and the regular training examples are learned simultaneously. However, it differs in that there are no additional inputs and the extra hint examples are usually artificially created. The generation of hints is an art, it is done using domain knowledge, heuristics, and best guesses. While this is normally used as a method of improving generalization through explicit inclusion of human knowledge into the network, it can be used as a mechanism of life long learning if the hints are used to consolidate previous experiences.

2.1.2 Representational Transfer

In representational transfer learning of the source and target tasks happen at different times. The source is learned first and then some form of knowledge representation is transferred explicitly to the target training task. This can be as simple as starting the target training from the network resulting from the source training, or could involve some transformation of the source network through a secondary method before being used to bias the target network.

Meta-Neural Network

Meta-Neural networks developed by Naik [42], is a training technique where a source network is trained several times with random starting positions. The starting and ending points for the weights of the network are collected and then used as a training set for a “meta-neural network”. The meta-neural network is then used in conjunction with the source network to train new tasks. The function of the meta-neural network in this system is to provide the training algorithm with a direction vector and step-size for the weights in the underlying network based on the previous training examples. The intended result is to improve the learning-rate of related problem sets. This is a form of representational transfer because the meta-neural network constrains the training of the network weights.

Discriminability Based Transfer

Discriminability based transfer by Pratt [46] utilizes non-literal transfer to selectively transfer source weights to the target network. The input to hidden weights of the source network are considered representative of hyper-plane decision surfaces for the hidden layer nodes. These decision surfaces are then evaluated on how well they discriminate in the target network. The weights that

are not very helpful to the target task are copied but are reduced so that they are more susceptible to change in the target training. This method was shown to perform at least as well as randomly generated starting weights in all cases, while improving performance in others. By contrast literal transfer of the weights sometimes led to worse performance than the randomly generated starting weights.

Learning one more thing

Thrun and Mitchell [70] developed a system meant to deal with a family of related tasks. Their system builds a network that learns to tell if two examples are the same object. This network is trained on many different objects, but the task is simply to tell if the two provided examples are the same or not. Once this network has been trained it is then possible to classify a new object with a single positive example, since each subsequent example can be compared to the first. This technique developed what is called an *invariance network*. The invariance network was then used to provide initial conditions for a target network using EBNN as the transfer mechanism. Explanation-based Neural Networks (EBNN) [71] is neural network training technique that uses neural networks trained on some useful function (in this case the invariance network) to approximate the slopes of the learned function in response to the input targets. These slopes are then used on conjunction with the training patterns to guide the training of the desired network using the tangent-prop algorithm. This technique has been applied to life long learning in robot controls [71], and robot perception [43].

2.2 Generative Neural Networks

The process of choosing the structure of a neural network can be a tedious and difficult task. It can significantly impact the performance and capabilities of the network, and as such the choice is very important. Knowing how to choose the proper structure usually requires an expert, or a significant amount of experimentation in addition to the experimentation required to solve the original problem. Generative algorithms that learn the correct structure as part of the learning problem alleviate this problem. Often they will find a structure that is sufficient but not quite optimal to learn the problem, but this is usually good enough. The rest of this section will discuss some of the methods used for generating network structure.

2.2.1 Neuroevolution

Neuroevolution is the artificial evolution of neural networks using a genetic algorithm. Genetic algorithms are powerful stochastic search algorithms that are well suited to large tasks especially those that have large state spaces or non-differentiable objective functions. For this reason they are used on conjunction with neural networks most often in reinforcement learning problems where the normal supervised techniques of neural networks are harder to apply. Early NE techniques used fixed structures and evolved the weights, however several techniques for evolving the structure and weights have been developed. These methods are often called TWEANNs or *Topology and Weight Evolving Artificial Neural Networks*. We discuss a few of these below.

Neuroevolution of Augmenting Topologies

Traditional NE approaches use a fixed topology of an input layer, an output layer, and a fully connected hidden layer. The evolution searches the connection weights and optimizes them to

find the solution to the given problem. Connections however are not the only thing affecting the behavior of neural networks. The number and composition of the layers and nodes of the network affect what sort of functions the network is able to replicate. Also the presence or absence of recurrent links affect the networks memory. The *Neuroevolution of Augmenting Topologies* [63] algorithm is one of the most popular and effective NE techniques.

Prior to NEAT it was not conclusive that evolving both the weights and topology was worthwhile, Stanley's work with NEAT shows that evolving both provides many benefits. It's widely known that size and topology of the network can have problematic effects on the training of neural network using techniques like back-propagation; over-fitting, an explosion in computation as the size of the network grows, or the complication of training recurrent connections to name a few. NEAT starts with minimal complexity and grows the structure as needed. Smaller networks that get stuck in local minima can be dislodged as the topology grows, and beginning from minimum complexity reduces the overall computational load needed during training. NEAT can also easily handle recurrent connections, as the differences in recurrent and normal connections are not important to the evolutionary search algorithm. These benefits lead to improvements in training speed, while also removing the burden of topology selection from the human user.

Evolving neural network structure gives rise to several technical challenges that must be overcome. The first challenge is how to cross over disparate topologies without losing important information, through competing conventions. The second problem is how to protect topological changes that might hinder performance when they are initially added so that they are given a few generations to adapt before they disappear from the population. Finally the last challenge is how to minimize the complexity of the network structure through the evolutionary process without using a fitness measure for complexity.

Cross over of topologies is difficult because of the competing conventions problem. It is possible

that two networks could share topological features but have different genomes. This means that during cross over it is possible that two parents are identical structurally, but different genetically. When cross-over is performed in this situation the child network will have lost important information that both parents possessed. NEAT solves this by using historical markers to keep track of when structure was added to the population. In this way when two networks are compared it is possible to see what parts of their topology is shared. Using these historical markers, dubbed *innovation numbers* it is possible to line up the genomes for cross-over in a meaningful way that will preserve the integrity of the offspring.

NEAT solves the problem of protecting novel topological structures through the use of speciation. Speciation is the process of dividing the population up into smaller niches that compete within themselves for selection. This is accomplished by developing a similarity measure based on the innovation numbers and using that to divide the population. When a new topological feature is added if it is different enough it is put into its own species. Since the networks compete within their own species for selection this protects new topological innovation from being removed from the population without having a few generations to improve its performance.

The issue of maintaining minimal network complexity in NEAT is solved in the simple way of starting from minimal complexity. All networks in NEAT's initial population are minimal structures, as opposed to other TWEANN algorithms who start with networks with random topology sizes. Since all of NEAT's networks start from a minimal topology and only those topologies that improve the performance of the population are kept, the complexity of NEAT's networks increase slowly as the algorithm searches for the best performing network.

NEAT is a very popular algorithm and has been used in a variety of problems, including neural controls, reinforcement learning, and more interesting places like music and image art creation. It is a very powerful and robust algorithm that is able to create networks for just about any problem.

An obvious question would then be why not use it for our application. While it is possible that we could use NEAT to create a life long learning system it does not fit perfectly with our desired situation. Primarily we will be dealing with supervised learning situations. While NEAT could be used to solve a supervised learning problem, it is unlikely that it would be able to do so before the CC algorithm found a solution. Evolution is a power stochastic search algorithm, but when you know which direction to travel in, gradient methods offer a more direct solution. Secondly the way NEAT builds the networks is poorly suited for the types of transfer learning we intend to use, we intend to use a single network that is grown to fit the problem, where NEAT would generate a population from which to select the network. Again this is more complicated than we would need it to be. Finally the method of overcoming catastrophic forgetting that we intend to use is much more direct than the mile-stoning method used in NEAT [62]

Symbiotic Adaptive Neuroevolution

Symbiotic evolution is a variant of normal evolutionary algorithms where each member of the population is only a partial solution, that must be combined with other members of the population to create a full solution to the problem. It generates diverse, un-converged populations, since the individuals rely on each other for fitness. SANE (Symbiotic, Adaptive Neuroevolution) [41] is a form of TWEANN that generates neural networks through symbiotic evolution.

SANE is a method of creating neural controls for reinforcement learning problems. Because it searches for a network that defines a behavior, instead of the correct action it doesn't have to deal with the credit assignment problem, and belongs to the general class of reinforcement learning algorithms.

SANE works by evolving individual neurons and then combining them into full solutions. The pressure of selection for the neurons is based on how well they cooperate with other neurons so

a single neuron can't overcome the entire population. The process begins by creating a network out of a random selection of the neurons in the population and evaluating the resulting network on the problem to be solved. This is repeated until all of the members of the population have been a part of a sufficient number of networks. At this point the fitness of each neuron is calculated by averaging the fitness of all the networks it was a part of. Once all of the neurons have a fitness value the normal operations of the genetic algorithm are carried out. Neurons who cooperated well with others will have the highest fitness functions while those who did not will have lower scores and will be selected less frequently.

SANE was shown at the time to improve significantly on the performance of reinforcement learning algorithms in the poll-balancing problem. However, it was ultimately surpassed by NEAT.

2.2.2 Cascade Correlation

Generative neural network techniques are designed to take the guess work out of developing neural network representations. The structure of a neural network is just as important as the algorithm used to train it, as the size and shape of the network will have an impact on what types of problems it can learn and how quickly. The Cascade Correlation architecture was chosen as a platform because of its fast learning and layered technique for adding structure that fits nicely with the sequential learning scenario.

The Cascade Correlation [24] architecture is a supervised learning technique for ANN's. In CC the network starts as a minimal fully connected multi-layer perceptron (MLP) with no hidden nodes. CC uses quick-prop [22] to train the weights, and adds hidden units one at a time when they are needed to increase accuracy and generalization. The new units are trained on the side until its output is maximally correlated with the error and then they are added to the network. As each new unit is added to the network its input weights are frozen so that it becomes a permanent feature

detector. This makes CC a good candidate for iterative learning as all previously learned hidden structure is maintained. Once a new set of training has been completed it is only necessary to update the output weights to adjust for new hidden layer structure.

CC was developed to overcome shortcomings in the standard back-propagation learning technique, primarily its slow learning rate. The two primary reasons for back-propagation's slow learning rate are the *step-size problem* and the *moving target problem*.

The step-size problem comes from the gradient descent search through the weight space to reduce error. If the step size is too small the learning takes a very long time. If the step-size is too large the algorithm might not reliably converge to a good solution. In order to pick a good step size it is necessary to know the higher order derivatives of the error function. There have been numerous techniques developed to deal with this problem, and one of the most successful techniques is Fahlman's quick-prop [22]. It computes the first order partial derivative the same as back-prop, but instead of simple gradient decent, it uses a second-order method to update the weights. This second order method is based on the heuristic that weight space is parabolic around the correct weight. Using this parabolic assumption the approximately correct step size is chosen. This allows quick prop to very quickly approach the correct weight orders of magnitude faster than standard back-prop.

The moving target problem is a phenomenon that arises from the fact that all of the weights of the networks are updated at the same time. Each of the weights are being updated to reduce the error of the network, but as each weight is changed the error moves. One way to combat this is to hold all but one of the weights fixed. This might seem slower but experiments by Fahlman show this actually speeds up training since the moving target problem is eliminated. CC deals with the moving target problem by freezing the input weights of the hidden layer nodes once they have maximized their correlation to the network output. This means that each new hidden layer node is

trained to alleviate some part of the error signal and then frozen. The output weights of the network are always allowed to adjust as the new hidden units are added.

The CC architecture combines two major features. The first is the cascade architecture, in which each new node is added to a new hidden layer that is fully connected to all previous layers leading to the creation of high order feature detectors. This structure can be seen in figure 2.1. As each new unit is added its input weights are frozen and only its output weights are trained from there on. The second feature is the learning algorithm, which creates and installs the new hidden units. Each new unit is trained so that its output is *correlated* to the error signal. This is done by presenting the data to one or more candidate units and using quick-prop to adjust the input weights in order to maximize the correlation between the unit's output and the global error signal. The candidate's score to be maximized is calculated by summing the correlation of the candidates output (V) to the output error (E_o) across all outputs according to:

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right| \quad (2.1)$$

Where o is the network output at which the error is calculated and p is the training pattern. The values \bar{V} and \bar{E}_o are the averages of V and E_o across all training patterns.

In order for quick-prop to maximize each candidates score S , the partial derivative of S to the candidates input weights w_i must be calculated. $\delta S / \delta w_i$ is defined as:

$$\delta S / \delta w_i = \sum_{p,o} \sigma_o (E_{p,o} - \bar{E}_o) f'_p I_{i,p} \quad (2.2)$$

Where σ_o is the sign of the correlation between the candidate's value and output o , f'_p is the derivative for pattern p of the candidate's activation function, and $I_{i,p}$ is the input for the candidate from input unit i .

Once the correlation is maximized the new unit with the highest correlation is chosen and added to the network with its input weights frozen. These two features combine to make a very fast and robust neural network training algorithm that is able to learn much faster than standard back-propagation and is able to develop the structure at the same time as its training the weights.

CC provides several benefits in the life long system we intend to build. First, the way it builds networks works very well with the dual nature of the long-term and short-term networks that are used in the life long learning system of Silver [59]. Where Silvers system requires two separate networks with temporary connections between the two, using CC we'll be able to have the long term and short term networks integrated into the same structure without worrying about the new task destroying our long term network. Second, this same structure will allow us to more quickly integrate the new task since we only have to use our reverberating pseudo-rehearsal method to keep the output layer of the network from forgetting its previously learned knowledge. Where Silver has to incorporate the new and old task knowledge into a new long term storage network using task-rehearsal.

2.2.3 Knowledge Based Cascade Correlation

Knowledge Based Cascade Correlation [56] is an extension of CC that allows for previously trained networks to be used as candidates in training new networks. The previously trained networks are made available alongside normal candidate nodes, and are treated as a black box node. This means that sub-networks only interact with the parent network through their inputs and outputs, while their internal connections are left untouched. Any type of network or function could be included as a sub-network as long as it has a derivative.

KBCC follows the same two-phase training as the standard CC algorithm; a candidate phase where new candidates are created and trained to correlate with the residual error of the network, and an

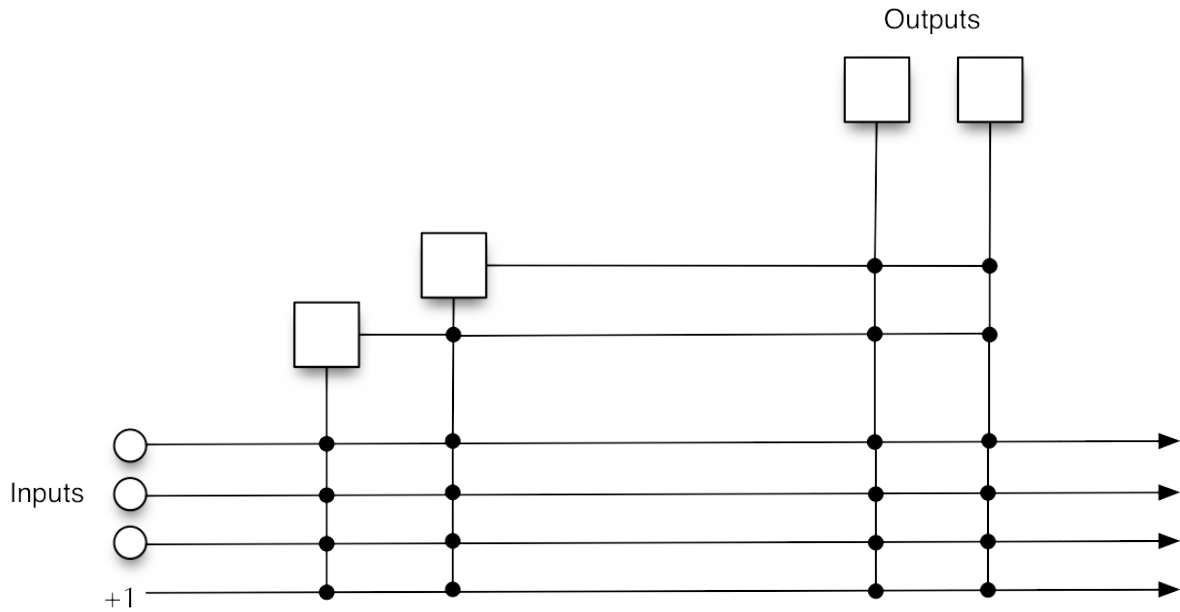


Figure 2.1: Cascade Correlation Network: The network starts with the output nodes fully connected. Candidate nodes are added as needed when the output training stagnates. Candidate nodes are cascaded meaning each new node is connected to all of the nodes previously added.

output phase that the updates the connections to the output layer. During the output training phase the weights connected to the outputs are trained to minimize the error of the network on a training set. This phase is exactly the same in both CC and KBCC, with KBCC using Fahlman's *Quick-prop*.

The major difference between KBCC and CC is in the input training phase. KBCC treats the sub-networks as special candidate nodes, and they are added and trained in parallel with standard candidate nodes. KBCC's candidate phase follows the same pattern as CC. Candidate units are created and connected to all previous hidden layer and input nodes and then trained so that their outputs correlate to the residual error of the networks outputs. The sub-networks however will have multiple inputs and outputs, and thus must be treated differently when computing the correlation

score and weight updates. KBCC uses a generalized correlation score function as the target value for each candidate to maximize. The equation is given by

$$G_c = \frac{\sum_{o_c} \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right|}{\#O_c \cdot \#O \cdot \sum_o \sum_p E_{o,p}^2} \quad (2.3)$$

Where \bar{E}_o is the mean error at the output unit o and \bar{V}_{o_c} is the mean activation of the output o_c of the candidate c . The output error at pattern p is:

$$E_{o,p} = (V_{o,p} - T_{o,p})f'_{o,p} \quad (2.4)$$

G_c is normalized by sum squared error of the outputs and averaged over the number of network outputs ($\#O$) and the number of outputs of the candidate ($\#O_c$). For standard candidate nodes with a single output this is simply the normalized version of equation 2.1, while sub-networks will average their score across all of their outputs.

In order to use Quick-prop to maximize the value G_c we must calculate its derivative. The derivative of G_c with respect to the weight w_{o_u, i_c} between the output o_u of the upstream node u and the input i_c of candidate c is given by

$$\frac{\delta G_c}{\delta w_{o_u, i_c}} = \frac{\sum_{o_c} \sum_o \sum_p \sigma_{o_c, o} (E_{o,p} - \bar{E}_o) \nabla_{i_c} f_{o_c, p} V_{o_c, p}}{\#O_c \cdot \#O \cdot \sum_o \sum_p E_{o,p}^2} \quad (2.5)$$

where $\sigma_{o_c, o}$ is the sign of the covariance between the output o_c of candidate c and the activation of output unit o , and $\nabla_{i_c} f_{o_c, p}$ is the partial derivative of the candidates output o_c with respect to its input i_c . For normal candidates with a single input and output this is simply the derivative of the activation function. For sub-networks it is calculated by propagating the derivatives through the

network with a forward pass, similar to how error is propagated in back-prop.

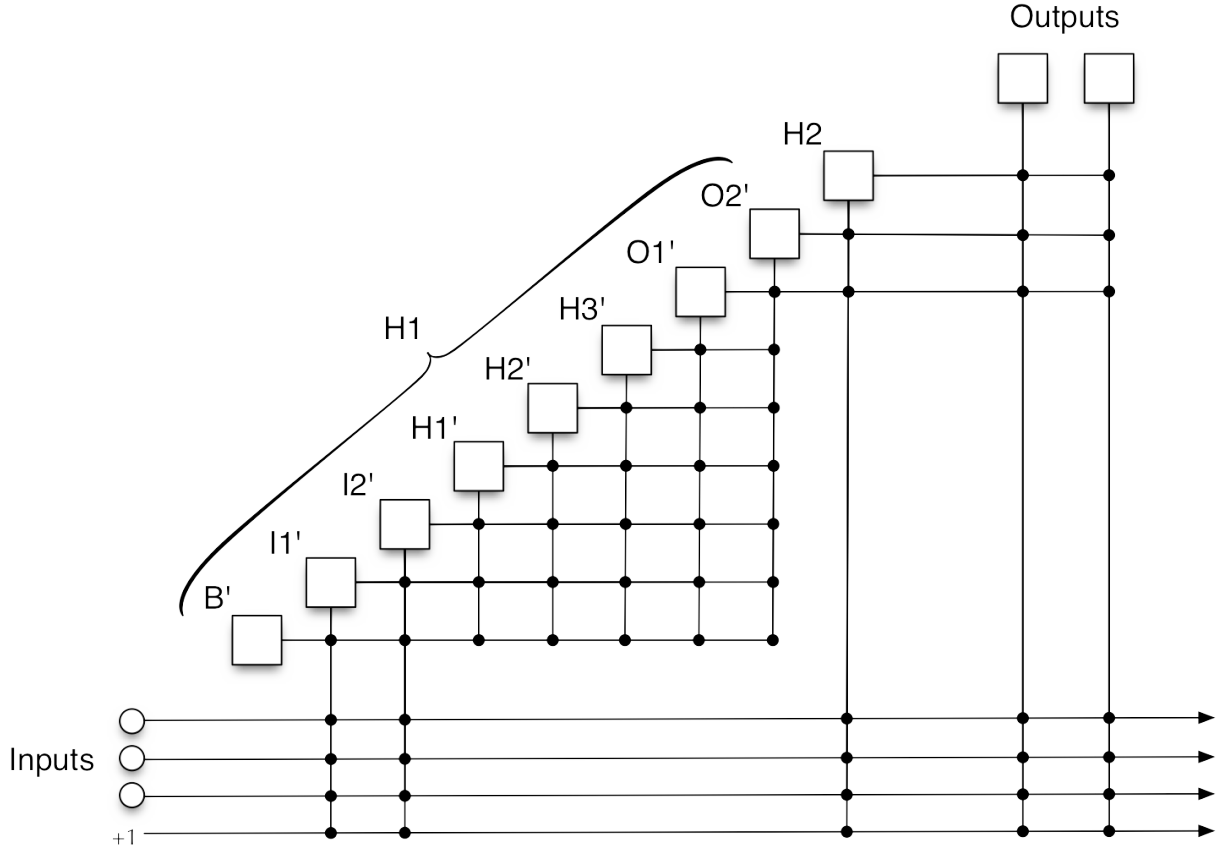


Figure 2.2: The structure of a KBCC Network. The sub-network H1 has weights connecting its inputs and outputs to the parent network, but its internal weights are not connected. The hidden layer node H2 is connected upstream to the inputs of the network and the outputs of the sub-network H1.

Figure 2.2 shows the connection scheme of a KBCC Network. The sub-network is connected through its inputs and outputs while its internal connections and nodes remain independent. When incorporating a sub-network there are two methods for connecting to the upstream nodes. The first method is to only connect the inputs of the sub-network to the corresponding inputs of the parent network with a weight value of 1. In this way the network is acting as if it is receiving the network input directly, and thus providing exact knowledge to the parent network. The second method is

to connect the sub-network to all upstream nodes in the same way as other candidates. In this configuration the sub-network can be adapted to fit the needs of the parent network. During the candidate training phase each sub-network has several copies initialized with random weights, and one copy directly connected.

An important feature of this connection scheme is that the internal networks do not have to have the same number of inputs and outputs as the parent network. This can be important if the sub-network is a sub-problem of the parent networks problem, or when networks have been trained on a subset of the problem input space.

The ability of KBCC to incorporate previously trained networks into future learning is a very powerful tool. This form of representational transfer will be useful in our life-long learning system.

2.3 Multitask Learning Literature

2.3.1 *Multi-task Learning*

Multi-task learning was developed by Rich Caruana as detailed in his thesis[13]. The primary motivation behind MTL is the idea that we do not learn tasks in isolation. When we learn new tasks we take advantage of our previous learning and we are learning many related sub-tasks at the same time. For example when we learn how to play tennis, we are learning how to run, how to swing the racket, how to predict the trajectory of the ball, and many other small but related tasks. If we were to try to learn each of those tasks in isolation we would most likely not learn the tennis task as effectively.

The theory of MTL is predicated by the idea that tasks can serve as a mutual source of inductive bias[12], that is to say that the data for each task could hold information that is useful to all related

tasks. Inductive bias is anything that causes an inductive learner to prefer one possible hypothesis over another. MTL is one source of inductive bias, other forms of bias can include the a priori connection weights, or higher level information used to alter the standard back-propagation update rules. In MTL the extra bias comes from the training data of the tasks. Training data is not normally considered a source of bias, but when the data contains information for multiple tasks, it is easy to see that from the perspective of any single task, the data from other tasks can serve as bias. For this reason MTL eschews the standard idea of breaking large tasks into smaller more manageable tasks, since learning the smaller tasks in isolation forgoes any information that can be gained from the relatedness of the tasks. In fact Caruana theorizes that more complicated tasks might be more difficult to solve once they have been broken down and the MTL bias is lost.

MTL can be better explained through an example. The following example is borrowed from [12]. Consider the following tasks:

Input: 8 bit string - ex. 10001101

Task 1 = $B_1 \text{ OR Parity}(B_2-B_8)$

Task 2 = $\sim B_1 \text{ OR Parity}(B_2-B_8)$

Task 3 = $B_1 \text{ AND Parity}(B_2-B_8)$

Task 4 = $\sim B_1 \text{ AND Parity}(B_2-B_8)$

In this example B_i represents the i^{th} byte and Parity is the number of bits set, mod 2. Given the bit string shown above T1 would be 1 and T4 would be 0.

The tasks in the example above share many similarities. First they all have the same inputs, this is essential for MTL, as the MTL method breaks down if the tasks don't share some inputs. Second they all share the Parity(B_2-B_8) sub problem. Also important to notice is that because of short circuiting the Parity part of the problem does not need to be computed for every task in every

case. For example, if $B1 = 1$ then parity only needs to be computed for tasks 2 and 3, while if $B1 = 0$ parity only needs to be computed for tasks 1 and 4. In this example MTL has advantages because once the Parity part of the function has been learned all of the tasks can take advantage of it, also the parity function will have 4 times as many examples available in training since each task's examples will carry information about it. Figure 2.3 shows the configuration of an MTL network compared to STL networks.

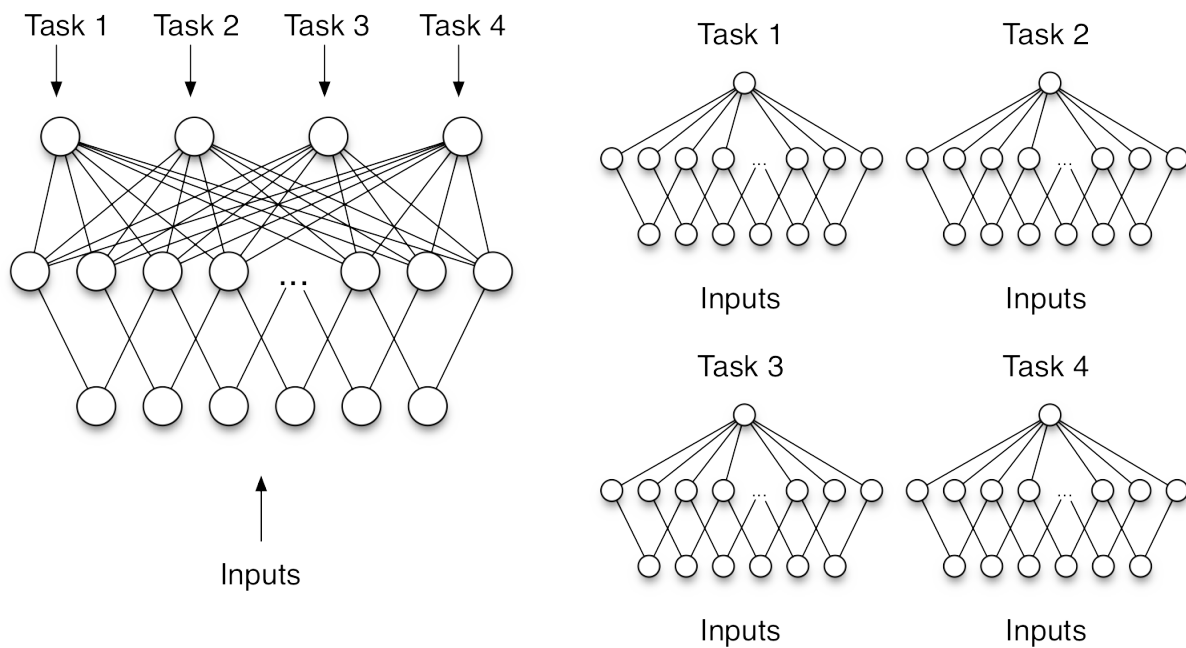


Figure 2.3: MTL vs STL networks: MTL and STL networks for a collection of tasks. The MTL network has a shared hidden layer connected to independent outputs for each task.

There are some disadvantages when using MTL. In most situations the MTL net will require more capacity than a network learning one task or the other. This is needed so that there is enough room for the multiple tasks to develop hidden layer representations that are useful. If there is not enough capacity it could cause the network to not be able to reach the desired level of generalization. This is one of the primary reasons for using a generative structure algorithm as it finds the correct

structure as it learns. Another potential problem could arise if the tasks being learned by MTL are competing or non complimentary. In this case the gradients computed for each task could interfere with each other and increase the amount of time needed to learn the tasks as well as decreasing the maximum level of performance achievable. This issue is overcome by η_{MTL} which is discussed in section 2.3.2. Also, when training multiple tasks it can be difficult to know when to stop the training so that some of the tasks do not become over trained. In some cases this is acceptable if you are only concerned with a target task, but if you are concerned with the performance of all tasks this could be an issue. Finally, MTL doesn't have the ability to use different learning rates for each task. This could be a problem because each of the outputs could be learning their tasks at slightly different rates. If the outputs are all learning with the same learning rate it means there is a good chance that the stopping point for one output will not be the optimal stopping point for a different task. This usually manifests as one or more of the outputs being slightly less optimal compared to its theoretical best if all of the outputs stop training at the same time.

The advantages of MTL are also numerous. If the tasks share weights then the MTL net could end up being smaller than the networks that learn each task combined. Also, even though the MTL aggregate gradient may be flatter it could also be more robust to noise, or point in better directions earlier in the search, because the direction benefits multiple tasks. Although it is possible for MTL to over fit some of the tasks during training, it does not have to be a problem. MTL requires tasks to be learned in context of other tasks, not that one network be designed that fits all the tasks equally. If all tasks are important then MTL can be used to produce networks that are peaked for each individual task.

In [12] the authors show that MTL can improve generalization over STL. They also provide explanations for this observation. MTL can improve generalization through the following means:

Data amplification Sub features shared by all tasks will be amplified by all training examples.

This means that the sub features will have many more examples to benefit from than if each task was learned in isolation

Eavesdropping Any feature developed by any tasks can be useful to other tasks. Since all the tasks share the hidden layer it is possible for all tasks to benefit for features developed there.

Bias networks toward novel configurations The internal representations developed by MTL that are by design useful to multiple tasks are probably not learnable through STL methods. This means that using the information available from multiple tasks could make it possible to reach network configurations that can not be reached using STL

Some of the disadvantages of MTL are solved by the modifications detailed below, but the importance of choosing the correct structure of the network is still left up to the user. This is one of the major benefits of using a generative algorithm for MTL.

2.3.2 η MTL

One of the primary drawbacks of MTL is the requirement that all tasks must be related. If unrelated tasks are chosen then the competing gradients of back-propagation will probably not reach the desired level of generalization. It is normally left up to the user which tasks to include, and this requirement places an additional burden on the user to select good tasks. In [57] the authors attempt to overcome this limitation with η MTL.

η MTL is a modified version of MTL that uses task specific learning rates η_k based on a measure of task relatedness. These learning rates are dynamic so that throughout the course of learning they can be adjusted according to their similarity to the target task. This technique allows MTL to overcome the burden of having to select related tasks a priori.

MTL achieves functional transfer due to the pressure of learning several related tasks in parallel. This pressure forces the hidden layer to assume a representation that is beneficial to all tasks. If all tasks are not strongly related then negative inductive bias can occur, and the hidden layer will be forced to assume a representation that is less optimal for some tasks. This will lead to slower training times and poor generalization. In order to decrease the effect of unrelated tasks η MTL uses a measure of task relatedness to relax the parallel learning constraint. This allows the algorithm to modify how much each individual task can influence the shared representation. Ultimately this allows the algorithm to weed out tasks that are detrimental to learning the target task, and limit their impact on the shared representation to almost zero.

η MTL uses a modified version of back-propagation that allows for multiple learning rates. The differences in the back-propagation are listed in the table below.

Table 2.1: Back-prop Definitions

MTL Back-prop	η MTL Back-prop
$\Delta w_{jk} = \eta \delta_k o_j$	$\Delta w_{jk} = \delta_k o_j$
$\delta_k = (t_k - o_k) o_k (1 - o_k)$	$\delta_k = \eta_k (t_k - o_k) o_k (1 - o_k)$
$\Delta w_{ij} = \eta \delta_j o_i$	$\Delta w_{ij} = \delta_j o_i$
$\delta_j = o_j (1 - o_j) \sum_k \delta_k w_{jk}$	$\delta_j = o_j (1 - o_j) \sum_k \delta_k w_{jk}$
	$\eta_k = f(\eta, R_k) = \eta \times R_k$

The major difference between standard back-propagation and η MTL back-propagation is the placement of the learning rate. Standard back-propagation uses a global learning rate, while η MTL use task specific learning rates. By altering these learning rates during training η MTL can alter the influence of any individual task on the shared weights. The learning rates are set using a measure of the tasks relatedness according to $\eta_k = f(\eta, R_k)$. The individual learning rates are all based off of a user selected global learning rate η . This way the user still selects the standard learning

rate. While the tasks are highly related their learning rate will remain close to the global rate. As training continues the individual learning rates will decrease in accordance to their relatedness to the main task.

The learning rate for T_0 is set to η so $\eta_0 = \eta$. The other learning rates are defined by:

$$\eta_k = f(\eta, R_k) = \eta \times R_k \quad (2.6)$$

Where $0 \leq R_k \leq 1$ so all learning rates are bounded above by the global learning rate. By altering the parameter R_k η MTL can generalize MTL and STL networks. With $R_k = 1$ all individual learning rates will be set to η and η MTL will form a standard MTL network. With $R_k = 0$ the individual learning rates will be 0 and η MTL will form an STL network.

The measure of task relatedness is defined as:

$$R_k = \tanh\left(\frac{a_k}{d_k^2 + \Psi} \times \frac{1}{RELMIN}\right) \quad (2.7)$$

Where $a_k = \frac{1}{S}SE_k$ $0 < a_k < 1$, the accuracy of hypothesis h_k for task T_k , and d_k is the weight space distance between the primary hypothesis, h_0 , and the task hypothesis, h_k . RELMIN is a tuning parameter that controls the decay rate of the tanh function from one, and Ψ is a small constant that prevents division by zero. The parameters a_k and d_k were chosen for the following reasons. It is assumed that all tasks begin with the same set of randomly chosen weights. As the learning progresses the task specific weights begin to diverge as they attempt to accommodate the differences in the tasks that can not be accounted for in the shared weights. The representations for tasks that are similar will tend to stay similar, while tasks that are very different will have much different representations. Therefore the distance parameter d_k varies inversely to the relatedness

between tasks, so R_k varies inversely to d_k . The accuracy parameter a_k is used to alter the effect of the d_k , for parallel hypothesis. For example if two hypotheses are found to be equidistant to h_0 , then the hypothesis with the highest accuracy should have the strongest impact. Therefore R_k varies proportionally to a_k . The R_k parameter was designed to be similar to inverse square laws seen in nature.

Testing showed that η MTL performed equivalently to MTL when all of the tasks are related, and performed much better when unrelated tasks are included. This is a useful technique especially in real domains where it is much more difficult to make a judgment a priori about which tasks will serve as good sources of inductive bias. This technique can not be used with cascade correlation in its current form because cascade correlation doesn't use learning rates to train the candidate units that become the hidden layer. They are trained based on their correlation to the output error directly rather than through a propagated error signal. However with some tweaks to the candidate training method η MTL can be applied to the CC MTL technique.

2.3.3 Task Rehearsal

In life-long learning one is often concerned with learning a sequence of tasks over time. In [57] η MTL is combined with a process called task rehearsal to create a lifelong learning system. One challenge of life-long learning is a deficiency of training examples. Sometimes it is difficult to collect enough data to correctly train a new task, or it might be important to learn a new task with as little experience as possible. The correct use of inductive bias can overcome these difficulties by reducing the hypothesis space, and by placing the neural network as close as possible to the new solution [40]. One such source of inductive bias could be previous task knowledge [8]. The task rehearsal method uses previous experience as its inductive bias, and η MTL as its method of knowledge transfer.

Task rehearsal is used to solve the problem of catastrophic forgetting in NN's. That is the problem of NN's losing the ability to represent old tasks as they are trained with data from a new task. In standard NN's the weights will begin to move towards the new tasks representation without regard for its ability to represent the old task. Task rehearsal retains previously trained NN's and uses them to generate virtual examples that are added to the new data set to force the NN to retain some of its ability to solve the old problems. In this scenario the transfer of knowledge happens at a functional level rather than a representational level. This means that the underlying functions of the data are what is transferred rather than the previous representations that solved them. In this way the relationship between the functions of the tasks are important rather than the representation of the task.

Task rehearsal stores previous knowledge by saving previously generated NN's, in this way the previous task knowledge is condensed. Figure 2.4 shows an MTL network setup to use task rehearsal. As new tasks are training TRM will generate new training data for the secondary tasks by applying the input pattern to the stored NN's and using their outputs as training targets. Using this data as training patterns for secondary tasks allows the TRM method to use MTL or ηMTL as a method to transfer knowledge to the new task. Using this method new tasks can be learned with fewer training examples.

Task rehearsal has two phases of operation:

Knowledge Recall and Training Phase: Learning the new task in the ηMTL network. Previous tasks are learned in parallel using virtual examples

Domain Knowledge Update Phase: After the successful learning of a new task, if the new task can reach a user defined threshold of accuracy, the representation of the task in the ηMTL network is saved to domain knowledge.

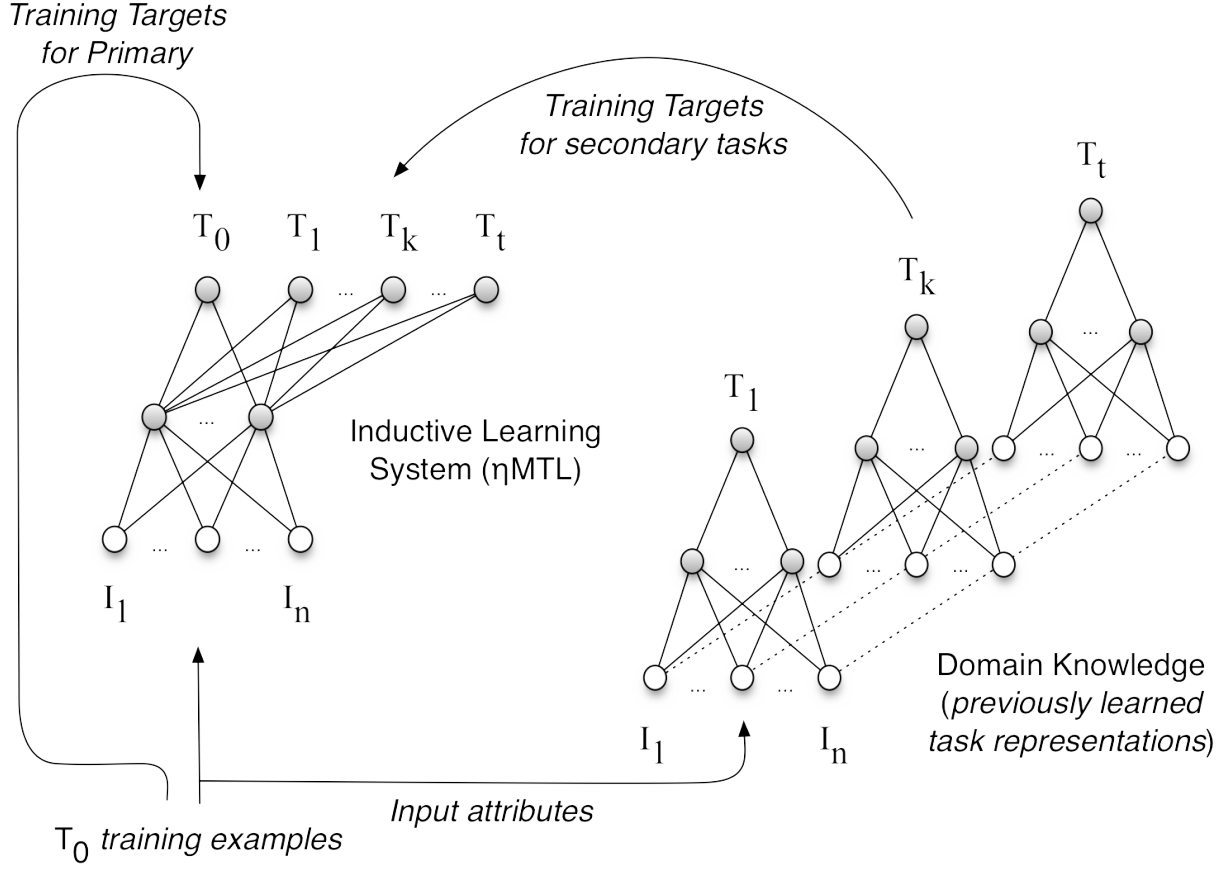


Figure 2.4: Task Rehearsal: Domain networks are used to provide secondary task targets for the primary networks MTL outputs through pseudo-examples. The η MTL algorithm is used to train the primary network. The domain networks allow the task-rehearsal system utilize the MTL technique without maintaining the previous tasks data.

The task rehearsal method has several benefits. First, it provides an efficient storage mechanism of training examples in the form of previously trained NN's. Second, it provides the freedom in choice of training examples since it will generate virtual examples for any previous task. And third, when the primary tasks data set is small it can overcome this problem by using additional data from secondary tasks as bias.

There are some negative aspects of TRM. The generation of virtual examples is vital to the success

of the technique and as such it is very important that the stored representations be accurate for the previous tasks. If the previous tasks were difficult to learn this will not be a trivial task and could require expertise on the part of the user to determine which existing networks provide the best knowledge to replicate. Another important aspect is to decide whether the networks will be used in real time, activating each domain knowledge network to acquire the desired output, or to batch the activations of the previous networks to recreate the training data before learning. This is essentially a trade off between computational time and storage space.

Task rehearsal is tested against STL NN's using data starved primary tasks. In each case TRM is able to learn the target task more efficiently by using the previous task knowledge. The success of the method is attributed to the functional knowledge stored in the domain networks and to the effective use of that knowledge by η MTL. It is important to note that the space requirements for TRM scale linearly with the number of tasks and the representation of the primary task. This means that each new task requires its own network to maintain its historical knowledge. This type of life long learning while successful, is somewhat limited by the fact that each new task requires a new long term network, and the fact that each new tasks network topology must be chosen by the user. Selection of each new networks topology becomes more difficult as the number of tasks increases, as the correct number of hidden nodes for an MTL network depend on how the tasks interact and how much representation they can share. If regularization is used it is generally safe to use a hidden layer sufficiently large to learn each task, but this is something that has to be found empirically for each task. This particular situation is handled elegantly by substituting cascade correlation networks for fixed structure, since the necessary number of hidden nodes will be found as training occurs.

Task rehearsal and η MTL while providing a serviceable life long learning system has several pit falls that our proposed system will over come. Our system will not need to maintain separate networks for each learned task, instead it will use two coupled networks and reverberating pseudo-

rehearsal to maintain the long term storage. Our system will also be able to take advantage of representational transfer by starting new tasks on the existing network, rather than starting from scratch with a new MTL network. Task rehearsal also requires that each new task get a new output, this leads to a system with redundant outputs, and hinders the inductive bias available to MTL to only the hidden layer. This is overcome by the CSMTL architecture detailed below. Our system will use this architecture as well and will get the same benefits.

2.3.4 *Context Sensitive MTL*

Context sensitive multitask learning [61] is a method of inductive transfer that uses a neural network with a single output and several contextual inputs to learn multiple tasks. MTL networks are feed forward multi-layer networks with an output for each task that is to be learned. They use back propagation to train all of the outputs at once. MTL training samples are composed of a set of input attributes and a desired output for each task, while CSMTL examples are composed of a set of input attributes with additional contextual inputs and a single shared output. Like MTL, CSMTL uses shared representation as the source of inductive bias. The more the tasks are related the more they will share representation and create positive inductive bias.

Silver and Poirier developed CSMTL to overcome shortcomings in the standard MTL architecture in relation to life long learning [61]. The first shortcoming is the requirement that all of the examples have output values for each input example. This can be impractical if the tasks examples are collected at different times, or if they have different combinations of input values. This shortcoming is especially evident in natural data sets as the likelihood of different tasks sharing exactly the same inputs is very low. One method to overcome this is to use an existing model of the task to generate examples for each tasks, which is possible if task rehearsal methods are being used, but this relies heavily on the accuracy of the model. Secondly the shared representation in MTL is lim-

ited to the hidden layer. This limits the information that can be shared between tasks, and ignores possible similarities between examples. Lastly, MTL systems will build up redundant outputs in a life long learning system as each new task data set requires an additional output, and it is possible that future data sets could be additional practice points for a previously learned task. Allowing contextual inputs to differentiate between tasks removes this possibility of redundant tasks.

The major differences in CSMTL are the shared output for all tasks, and the additional contextual inputs to indicate example context. Figure 2.5 shows the different configurations. This means that in CSMTL the entire representation is shared instead of only the hidden layer. This shifts the focus from learning a shared subset of the representation to learning a completely shared representation for the multiple tasks. The objective changes to predicting the output of similar combinations of contextual and example input values. Instead of relatedness being restricted to the task level it can now be exploited at the example level. For example if two tasks data sets have identical outputs for half of their training data, even if they aren't related at the task level, they are at least somewhat related at the example level. This could be visualized in the band database discussed later. In each of the band tasks the relatedness is determined by the area of the bands that overlap. However all of the tasks have some relatedness because they are all centered at the same place. This means that all of the task examples in the center of the range are going to share the same output. In this case CSMTL would be able to take advantage of this similarity directly where MTL would only benefit indirectly through the shared hidden layer.

Context sensitive MTL has several advantages over standard MTL. Silver shows in [61] that CSMTL has fewer free parameters than the MTL network for the same set of tasks. This is because of the mutually exclusive context inputs, which deactivate their weights while the tasks are not active. Deactivating these weights results in $(n-1)k$ fewer free parameters at any given time where k is the number of hidden nodes and n is the number of tasks. This is one possible explanation given for CSMTL's ability to learn the same set of tasks as MTL with fewer training examples. Experi-

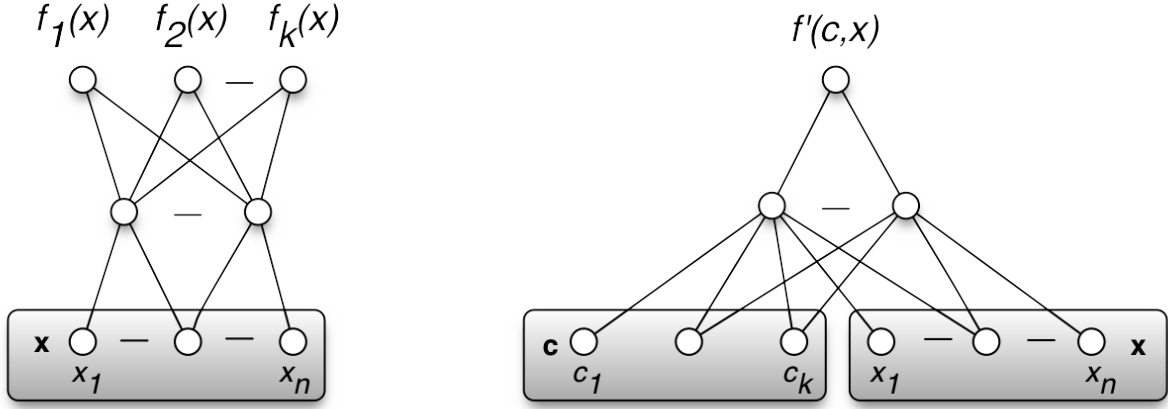


Figure 2.5: CSMTL vs MTL networks: CSMTL networks have additional context inputs rather than additional task outputs. Using context inputs allows for the tasks to share representation at the output layer in addition to the hidden layer as in MTL.

ments in [61] also show that CSMTL performs better under the ideal problem of perfectly identical secondary tasks, suggesting that CSMTL is better able to transfer knowledge between tasks, and that CSMTL better handles degradation of relatedness in secondary tasks. These advantages are most likely the result of the integration of the task specific bias. The special bias nodes allow each task to specialize each node in the network, whereas MTL networks only allow the output weights to specialize. These advantages make CSMTL a successful adaptation of MTL for the life long learning context. Combining CSMTL with CC should provide a robust foundation for a life long learning system.

2.4 Pseudo-Rehearsal

A major problem of using neural networks in a sequential learning task is the problem of catastrophic interference/forgetting. This problem was first formally introduced by McCloskey and Cohen in [39] and by Ratcliff in [48]. They noticed that neural networks had a tendency to com-

pletely forget previously learned examples when they were exposed to new examples, and they suggested that this was because of the distributed nature of knowledge in neural networks, ie. the feature responsible for NN's generalization performance and graceful degradation. Catastrophic forgetting is a severe manifestation of the more general stability-plasticity problem [32, 11, 10] of all memory models especially connectionist models. The stability-plasticity dilemma results from the competing goals of retaining all previously learned knowledge and remaining flexible enough to incorporate new information. A completely stable classifier would retain its performance on the data it was trained on but would not be able to learn new examples, while a completely plastic classifier would learn new information easily but at the expense of forgetting previously learned knowledge.

Catastrophic forgetting is a serious problem for neural networks especially when they are needed to learn sequentially in the way that humans do. Humans do not learn everything they know at once, they learn incrementally over time, and while forgetting in humans is natural it occurs gradually. Catastrophic forgetting would significantly impact the effectiveness of the life long learning system we intend to build so it is important to find a method to overcome this shortcoming.

Several techniques for addressing this issue have been tried throughout the years, a survey of techniques can be found in [29]. Of particular interest to this research is the idea of *pseudo-rehearsal*. Pseudo-rehearsal was first introduced by Robins in [52]. Previous to Robins it had been known that a trivial solution to catastrophic forgetting was to continually rehearse the previously learned data. Standard rehearsal however requires that all data be retained, meaning that the training set size would continue to grow. This solution is not practical because all of the data might not be available, or the training set size could grow to be very large. The solution introduced by Robins was to create pseudo-items from the trained neural network by providing random input activations to the network. In this way any number of rehearsal items could be created and interspersed with the new training data to overcome catastrophic forgetting. The fact that neural networks approxi-

mate the underlying functions of the data they learn allow this technique to generate examples that resemble the original training data. The obvious requirement being that the network accurately learned the underlying function of the training data. This is the exact mechanism used by Silver's Task-Rehearsal [58] which is the intended mechanism of short term to long term storage in the CSMTL life long learning system proposed here [59].

Pseudo-rehearsal was extended by Ans in [4] to incorporate the idea of reverberation. Ans showed that standard pseudo-rehearsal, while useful for combating catastrophic forgetting, was not as effective as it could be. He posits that random activations and a single pass through the network does not produce pseudo-examples that accurately reflect the nature of the network. In fact it is said that the single pass through could produce pseudo-items that are too noisy, and could block further learning. Ans method adds an auto-associative layer that learns to mimic the input patterns. During pseudo-item creation the random activation is allowed to reverberate by feeding the output of the auto-associative layer back through the network until a number of cycles have been completed. At the end of the reverberation process the output of the auto-associative layer and the output layer are used as the new pseudo-item. This method tends to move the pseudo-items to network attractors, and is more optimal at capturing the deep structure of the network. Additionally, instead of storing the pseudo-items in a population, Ans uses a dual network architecture. The idea of storing the pseudo-items in a connectionist system was also explored by Robins [53] and French [28]. Robins used a network with two weights per connection, one soft weight that updated quickly and one hard weight that updated slowly. French's solution was also a dual network solution with one network being the quick learner and the other network storing the long term knowledge in the form of pseudo-items. The major difference of Ans work from these two techniques is the use of the reverberation.

The system developed by Ans uses a two stage dual network approach. In describing the system I will use the nomenclature used in [4]. The networks will be called Net1 and Net2 and the Stages

will be Stage I and Stage II. Net1 and Net2 are both MLP networks with input, hidden, and output layers, with the addition of the auto-associative outputs. Net1 is the primary network and as such is the only network that communicates with the outside world. Net2 is meant to be the long term storage of the system's knowledge and it only communicates with Net1. Net1 is first trained on the initial training set as a standard MLP with the additional auto-associative target, meaning some of the outputs are trained to mimic the inputs. In Stage I Net1 is reverberated to create pseudo-items for Net2 to learn. The number of pseudo-items and the level to which they are learned by Net2 are parameters set by the user. After Stage I has completed Stage II begins with Net1 being presented with new data that is to be learned. This time however for every new item to be learned Net2 is used to generate pseudo-items to be rehearsed. The ratio of new items to pseudo-items and the number of reverberations are also parameters of the system. The use of Net2 to provide pseudo-items for Net1 to learn in addition to the new data allows this system to sequentially learn tasks while avoiding catastrophic forgetting. Figure 2.6 depicts the process graphically.

In tests performed by Ans the reverberating pseudo rehearsal significantly outperforms standard pseudo rehearsal in many cases maintaining nearly perfect performance on the initial training task even after learning a second sequential task. Another interesting outcome of their experiments is the suggestion that in the context of self-refreshing memory that the sequential learning of similar tasks outperforms the concurrent learning of those same tasks in relation to the performance on the last learned task when the tasks are related. This would suggest that if performance of the latest task is the primary concern it would be better learned sequentially rather than concurrently as it would be in MTL. The results were reversed however when the tasks were less related. In both cases using self-refreshing memory allowed the system to retain the learning of the previous task. Experiments were also done to show that the reverberating process creates new unseen examples in the pseudo-items rather than just producing copies of the previously learned items, implying that the reverberating process is creating items representing an optimal approximation of

the networks structure. The last important aspect of this technique is the statement that the pseudo-rehearsal works with networks with different topologies [53]. This is important for two reasons. First, it allows us to use the CC architecture and algorithm with the dual network pseudo-rehearsal technique even though the two networks could have different internal structure. Second, it means that we should be able to use this technique as part of the knowledge consolidation system envisioned by this research and discussed in section 3.2.3. Figure 2.7 shows a CSMTL network with the auto-associative outputs added. This would be the base unit of the proposed life-long learning algorithm.

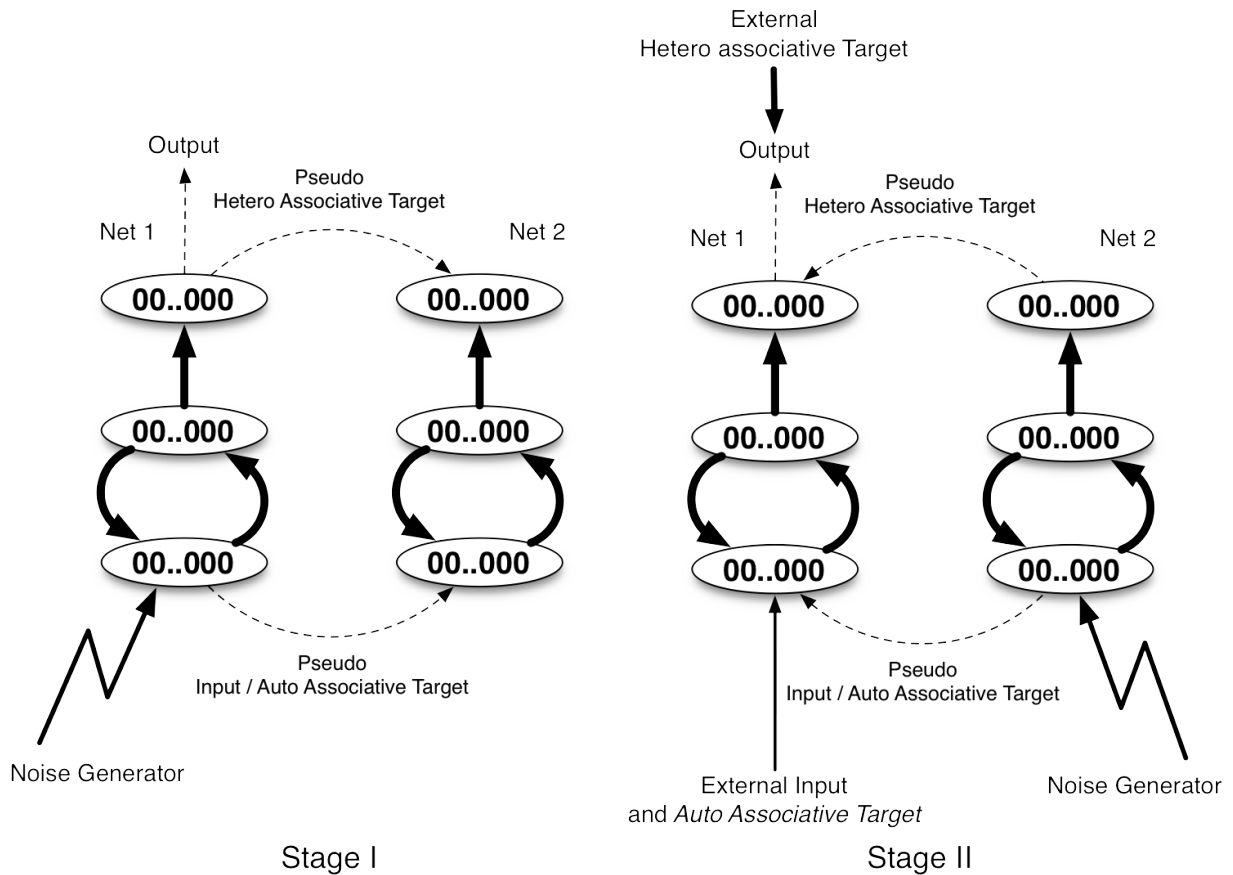


Figure 2.6: Stage I and Stage II of the dual network reverberating pseudo rehearsal system of Bernard Ans. In stage I the first network, which has already been trained on the first task, is activated with random data and reverberated to create pseudo-items. These items are used to train Net 2. In stage II Net 2 is used to create reverberated pseudo-items that are interspersed with the new task's training data. New tasks are only learned during stage II, while stage I is used to back up the knowledge from Net I into Net 2.

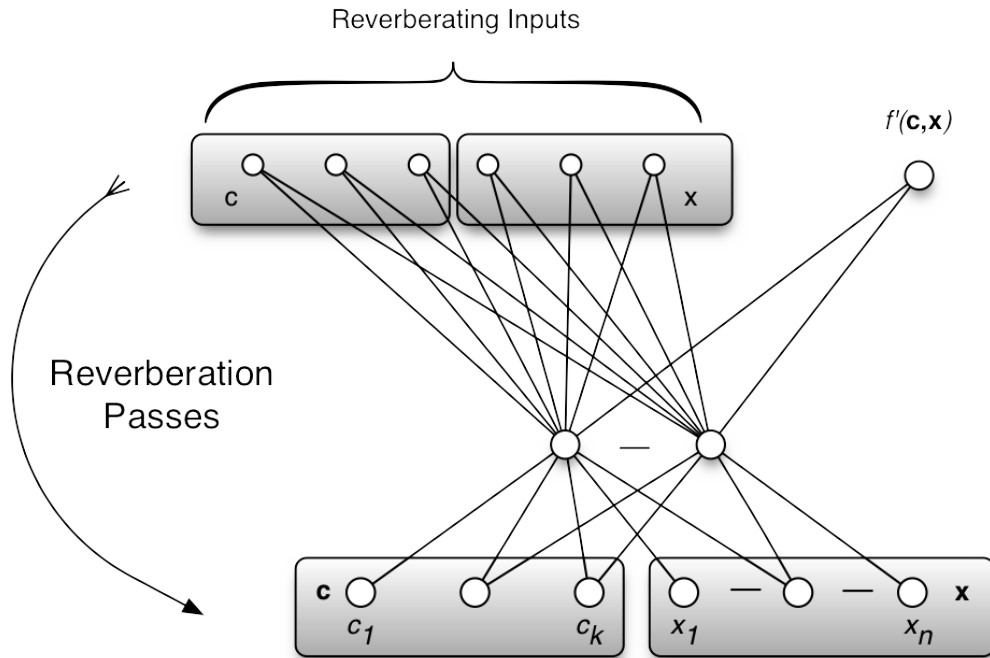


Figure 2.7: CSMTL network set up for self-refreshing memory. The only modification needed to the structure of the network is the addition of auto-associative weights. The most straight forward implementation of this is to add a special set of auto-associative outputs. They are then trained in the same manner as the normal outputs, using the input values as the target values.

2.5 Learning from Observation and Experience

One of the primary draw backs to learning behaviors is that in the search for optimal actions the agents can behave in ways that seem foreign and unintelligible to the human operators. This is often the case when neural controls are evolved. Evolution searches for the optimal behavior for the given fitness function, not for the behavior that is most optimal and also looks as we believe it should.

The development of human like agents is possible through hand coding and expert systems, but it is a tedious and complicated process. It is, however possible to learn human like behaviors through observation. FALCONET is such a system designed to create high performance human like agents through human observation. FALCONET provides an example of creating neural controls from human training that is the basis for the multiple teacher training we intend to use as the final application of our sparse life long learning algorithm. By combining the instruction of several human trainers we hope to get robust agents, while minimizing the burden on the individuals involved, and provide a mechanism to overcome individual trainers quirks.

2.5.1 *FALCONET*

Humans learn through several different processes. Learning through observation entails watching the process as performed by some other individual or agent. Learning through experience involves repetitive practice of the process with feed back on performance. Learning can and does occur under each process individually, but it is the combination of observation and experience that generally leads to the highest levels of performance. For instance when learning a new sport humans typically observe others already proficient in the activity before beginning to practice themselves. Observation bootstraps the learning process of experience enabling faster learning speed and higher

peak performance.

There is a long history in Machine learning of borrowing from biological systems. Examples include knowledge representations like neural networks, optimization algorithms like genetic algorithms and ant colony optimization, and learning paradigms like reinforcement learning. In this particular method discussed below the observational-experiential learning cycle is replicated in machine learning to achieve the same goals for simulated learning that are observed in biological learning.

FALCONET is a method of agent training that follows the biologically inspired cycle of observation and experiential learning. It was designed to enable the creation of high-performing, human like agents for real time reactionary control systems [64]. Typically the building of human like agents involves the complicated process of interviewing knowledge experts and then codifying that knowledge into a format that is machine readable. This process is complicated and time consuming and has led to the slow adoption of this technique in real world systems, despite the success that can be achieved. This problem is known as the “knowledge engineering bottleneck” [25]. FALCONET was designed to automate the agent creation process from human observation thereby sidestepping the bottleneck.

Previous work has been done using observational data alone to train agents, stopping once an acceptable level of performance is reached on training and validation sets [19] [38] [54] [33] [26]. While this might produce human like agents, it ignores the possibility that the observational agents will perform poorly in situations not covered in the observational data. The agents when presented with novel situations could perform in unpredictable and unintelligent ways. The experiential phase can fill in these gaps by providing feedback on the agents’ performance in novel situations.

The training in FALCONET follows a two phase training approach. First a supervised observa-

tional phase, followed by an unsupervised experiential phase. During the observational phase the objective of the learning is to be similar to the actions of a human trainer. Human trainers run through the selected tasks starting from different scenarios to generate the observational training set. The agents are then trained on this data set while being graded on how closely they mimic the decisions of the human. In the experiential phase the agents are trained further using a measure of performance on the task. In FALCONET all training is done by a hybrid GA PSO algorithm called PIDGION-alternate. This is a ANN optimization technique the generates efficient ANN controls from simple environmental feedback.

FALCONET has been tested showing that it can produce agents that perform as well or better than experiential training alone while incorporating human like behaviors. The results from FALCONET also state that unique human operator traits can be incorporated and evident in the final highest performance controls, that is to say that agents sourced from different trainers have slightly different behavioral quirks.

Human like behaviors in agents while sometimes less optimal are usually preferable for aesthetic and empathetic reasons. We are more likely to approve of the agent's behavior if it looks similar to how we would have performed the task.

As part of the validation of the FALCONET method experiments were conducted using only the experiential learning phase. High performance controls were created in this manner but they showed several "improper" quirks, that while more optimal in the performance metric, seem foreign to human operators. These quirks, like driving backward or slamming the controls left and right very quickly, can be programmed out by a human designer, but it requires the a priori knowledge of all "improper" behaviors that would be undesirable. The FALCONET method bypasses this need by bootstrapping the process with human training.

This type of system would be immediately applicable to the low level reactive controls of the sim-

ulated agents. The existing tests and experiments show very good results to driving and controlling agents with different levels of complexity. The benefit of creating controls in this manner is that you could simply have human experts practice the types of maneuvers and techniques that are desirable for the agents to have. It is much easier to collect this type of data through observation than to have a human expert try to explain the minutia of controlling an agent to a knowledge engineer, who then has to encode it into a program. In fact there are many unconscious skills that can be subtle but very important to the reflex response that are very difficult to articulate to the KE and thus transfer to the machine [38].

Using our method of life long learning in this type of problem would allow us to overcome of the minor issues that FALCONET found. First we would be able to more directly incorporate future training feed back from the instructor back into the network since we are able to do sequential learning using the same network resulting from the initial training rather than running a further round of evolution on the new data. Second, by collecting and merging networks trained by different instructors it should be possible to overcome the individual quirks of the instructors while still maintaining the human feel of the agents. Third by having the ability to merge different agents with different tasks from separate trainers, it would be possible for a team to coordinate in the training of an intelligent agent with several skills, with the trainers each working with their own network in their own environment.

2.5.2 *Multi-Agent Learning*

Multi-agent systems is a very active area of research. It is interesting because of the joint behaviors and the complexities arising from the interactions of agents with some degree of autonomy [44]. Early work in MAS dealt with planning and scheduling [37], and had very little to do with learning. It was primarily focused on developing protocols for interaction, and studying the effect

of various levels of agent communications. Early methods used scripted or rule based controls [31] or symbolic systems [20, 30]. However, because of the increased complexity of designing and building MAS machine learning quickly came into play. For an in depth survey of Machine Learning in MAS see [65], [55] or [44]. An overview of the major divisions of MAS learning follows.

MAS learning poses some significant challenges above and beyond standard machine learning. First the state space of the learning problem expands exponentially with the number of agents, second the dynamics of the environment become more intricate as the other agents add additional sources of change and complexity, and third the interaction of the agents can lead to emergent and unexpected behaviors. The majority of learning techniques employed in MAS tend to be reward based techniques like reinforcement learning and stochastic methods since knowing the correct answer in MAS is extremely difficult, thus precluding supervised methods [44]. Going further MAS learning can be broken down into two main approaches; team learning, where a single learner is used for the entire system, and concurrent learning, where individual agents learn their own behaviors.

Team Learning side steps the issue of unknown other agent actions, but has the negative side effect of blowing up the state space. While it doesn't have to deal with the moving target aspect of concurrent learning, it still has to deal with the possibility of *emergent behaviors*. The explosion of the state space in team learning can be a challenging problem for systems like RL that map the state space to actions, but it might not be as much of a problem for EC since it searches the space of behaviors. Team learning can be further divided into homogeneous and heterogeneous. Homogeneous learners develop a single behavior that is used by all the agents on the team, where heterogeneous learners can develop different unique behavior for each agent. It is noted by Balch [6] that domains that a single agent performs well in are suitable for homogeneous learners while domains that require specialization require heterogeneous learners. Hybrid systems are also possi-

ble by dividing up into squads and having heterogeneous squads of homogeneous agents.

The second cooperative MAS learning approach is *concurrent learning*. The advantage of this technique is that the state space of each agent will be smaller, while the downside is that each agent must learn in an environment that is not only changing but is adapting to it specifically. This comes from the fact that the other agents are learning as well and from each agent's perspective other agents are part of the environment. There are three main thrusts in concurrent learning research [44]. *Credit assignment*, which is how to assign the reward from the system between agents; *dynamics of learning*, which deals with the co-adaptation of the learning process; and *modeling other agents*, which tries to improve interaction and collaboration among the agents.

The larger application we have in mind to demonstrate the sparse life long learning system would in effect be a multi-agent learning system. We could have several agents in a simulation with the ability to learn and share their experiences. Our system would fall under the concurrent learning approach since each agent would have its own individual learning path, however the ability to merge the agents experiences and disperse them to the individual agents would provide some of the benefits of the team learning approach as well. Further research into how our use of life long learning in this application fits in with the existing MAL literature will be left to future work. Implementation of our system into an environment without the benefit of human training would require our system to be adapted to a reinforcement style learning technique, and is beyond the scope of this work.

CHAPTER 3: RESEARCH DESIGN AND METHODOLOGY

The previous section has detailed the previous work in life long learning, generative neural networks, MTL, Pseudo-rehearsal, and observational learning. Each of these areas are of particular interest to our proposed solution to the sparse life long learning environment.

A system that is able to do life long learning in a sparse simulated environment will require aspects of each of the areas detailed. The life long learning will come from the combination of representational transfer through the use of a dual CC network combining short-term and long-term knowledge, and functional transfer through CSMTL using reverberating pseudo-rehearsal. The use of the generative network technique of cascade correlation will allow our system to build the knowledge representation as it is needed, rather than having to set the storage of our system at the beginning or adjusting it after every new task. The use of observational traces will allow us to use our supervised life long learning technique to create simulated agents, and our method of merging separate traces will allow us to incorporate the training from multiple instructors.

The previous life long learning systems developed by Silver using η MTL [58] and CSMTL[59] with task-rehearsal are the inspiration for our life-long learning architecture, but have several limitations that we intend to overcome. The largest obstacle to these systems comes from having to choose the structure of their networks a priori. This limits the systems in several ways. First, the selection of neural network topology is a significant contributor to the performance of the network, and as such it usually requires an expert to make the decision. This is compounded in the MTL case and especially in the life long learning systems of Silver by the fact that multiple tasks are meant to share the representation. In the LLL systems it is compounded further by the need to choose the topology for the long term and short term networks. Second, the long term network poses a further problem in that it is meant to incorporate the knowledge of an unknown number of

tasks. Having a fixed structure for our long term storage network would either cripple the system in the long term by being too small, or would hinder training in the beginning by being far too large and requiring wasted computation with too many free parameters and be more susceptible to over-training. This might be alleviated by adjusting the long term network throughout the course of the life long learning scenario but at each decision point you are still faced with a difficult decision. Having a method to automatically adjust the size of the network as needed is paramount to building a system that is more generally applicable and more approachable to novice users.

The existing systems also have a shortcoming in the fact that they use task-rehearsal as their method of consolidating the long-term and short term knowledge. Task-rehearsal is a standard pseudo-rehearsal technique. Ans [2] showed that standard pseudo-rehearsal even when the original network adequately learned the original task was not able to completely overcome catastrophic forgetting. This would imply that using task-rehearsal to consolidate the long term and short term networks would not be able to transfer all of the necessary information back into the long term network. Our system by using reverberating pseudo-rehearsal, which showed nearly perfect recall [4], will have better recall of previously learned tasks.

Finally our system for life long learning will be able to take better advantage of the representation transfer mechanism than the CSMTL LLL system because of the way cascade correlation builds networks. Rather than having to provide connections between our long term and short term network to facilitate this transfer, our system will incorporate this transfer through the normal building cycle of CC. Our long term knowledge will be part of the frozen architecture of the CC network. In this way all new tasks will be learned on newly generated structure. Our consolidation step will also be faster and easier because we'll only need to use the reverberating pseudo-rehearsal to update our secondary network rather than starting from scratch with a new long term network.

3.1 Applications of Sparse Life long learning

This section discusses the larger real life applications that have helped direct our goals for this research, and have informed the decisions for each feature implemented in our life long learning system. The larger goal of our research is the extension of observational learning agent creation using several human teachers or a single user combining multiple agents. The ability to merge separate life long learning networks together provides the opportunity to use those networks to learn behaviors observationally from multiple experts. The benefits from using several human trainers to create agent behaviors could be manifold. First, it would allow us to potentially overcome some of the shortcomings of individual human trainers, by smoothing out the individual quirks in their example sessions, and by overcoming user fatigue. Second, it would allow us to create a system where a team of trainers could collaborate to create agents using different simulations and methods, using a system similar to version control. For example a user could start a chaser agent by training in an open field with a single target, then a second user could expand the agent by training in a simulation with obstacles. The changes made by the second user could then be merged back into the original agent. Third, it would allow us a straight forward mechanism for continuing the training of existing agents through further examples throughout the life of the agent. The benefits for a single user come from the ability to take existing agents with beneficial traits and merge them together to form a more complicated agent.

To illustrate this idea let's consider a game in which all of the non-player characters (NPCs) are agents created through observation of several players. For our purposes let's consider a real-time strategy (RTS) style game where the human player controls a collection of individual agents serving some purpose progressing the player forward according to some metric. Possible roles for the agents are resource collection, perimeter patrol, offensive attack, surveying, etc. Rather than have a fixed set of scripts for each of these behaviors this game would allow the players to have them

learn the behaviors from watching the human provide examples. This in of itself is not different from the existing research in observational learning, but by having the ability to use sparse life long learning, we could expand on this and allow the players to take existing agents already created by other players and extend them with their own training. They could add new experience to existing behaviors or they could extend the agent by adding new contexts with new behaviors. For example, a player could take an agent that has learned to patrol an area and teach it how to escort a moving target, or take an agent that has learned to gather resources and teach it how to appropriately respond to threats. Any modifications made to the agent could then be merged back into the previous agent or if the differences are too great and the performance level drops too far the user could spawn the agent off into a new agent strain. In this way over time the agents in the game could become more complex through the collaborative effort of the players. This idea has some similarities to the game NERO [62] where players adjust the fitness parameters to alter the agent's behavior. Nero uses a technique called mile-stoning to allow for the learning of more complex behaviors. To learn compound behaviors the agents are first trained according to a particular fitness function, and then once they've reached a desired level of competency the population is stored away and the second behavior is pursued. While training according to the new fitness function individuals from the first population are added back into the new population. In this way the behaviors of the first population are combined with the new desired behavior. This method allowed Nero to create more advanced agents, but in a round about way requiring the user of multiple fitness functions and relying on evolution to transfer behavior from a past population. Using life long learning and observational training the system envisioned would be a more direct route to complex useful agents. Instead of the user trying to manipulate agent behavior through adjusting parameters in a fitness function they would demonstrate the desired behavior through example. The use of the life long learning system would allow for behaviors to be trained in sequence building the new behaviors on top of the agents previous behaviors.

A second application of our ideas would be to use this system as a method of training and maintaining a collection of autonomous vehicles. Our methods would allow for the agents to be trained using human teachers, and would provide a method of updating the behaviors overtime as it was needed. In this scenario the agents could be dispersed throughout a large geographical area interacting with different scenarios as they are encountered in the real world. These encounters could be few and far between and if the agents can only learn from what they experience themselves it could be a very slow learning process. However, using the sparse life long learning technique each agent could contribute what they learn back to the others like it. It would be possible to mix this with observational training to allow for human operators to do after action training to improve an agent's response to a particular scenario. Once the new training has been added it could then be disseminated back to the other agents in the system. In this way intelligent human like responses could be learned by the agents in the system and learned more quickly through the merging of separate agents in the collection. Sparse life long learning could facilitate the creation of robust intelligent agents by combining the experiences of many agents across time and space.

Underlying both of these applications are two major types of scenarios. The scenario of a single user building agents, and the scenario of multiple users building agents. In each case both the life long learning and the merging of separate existing networks provide useful tools to achieve the goals of the user. Figure 3.1 depicts two scenarios for a single user of our system. In the first scenario the user trains an agent in sequential tasks of increasing complexity to achieve a particular goal. The user trains the agent through observational training in each of the scenarios training simple tasks first, then building on the previously trained networks, with additional observational training. The second single user scenario involves the user selecting existing networks with desirable traits to merge to achieve the desired result. The agents could be previously trained agents the user had created or they could come from agents that have been trained by others.

Figure 3.2 shows the multi-user scenario in a large environment. In a large environment it would

Single User Scenarios

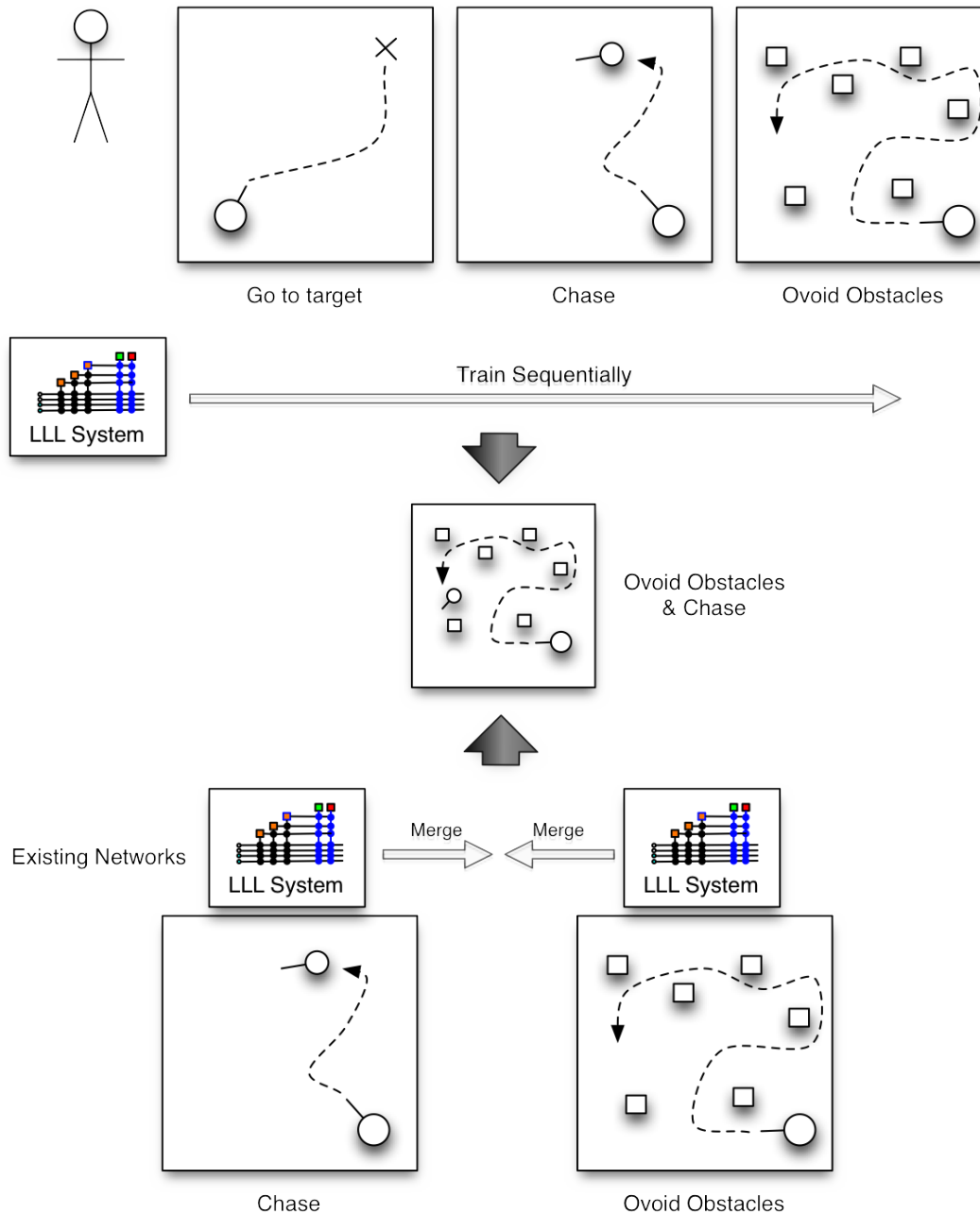
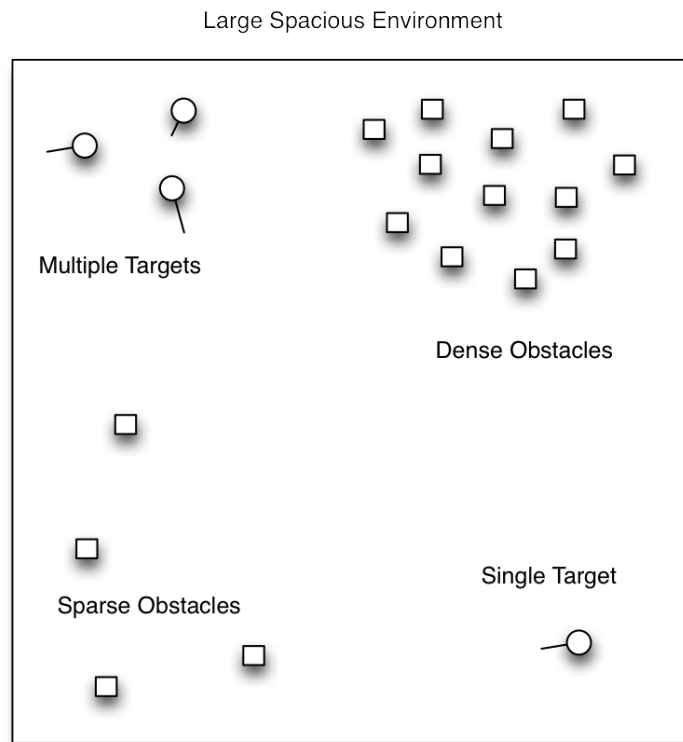
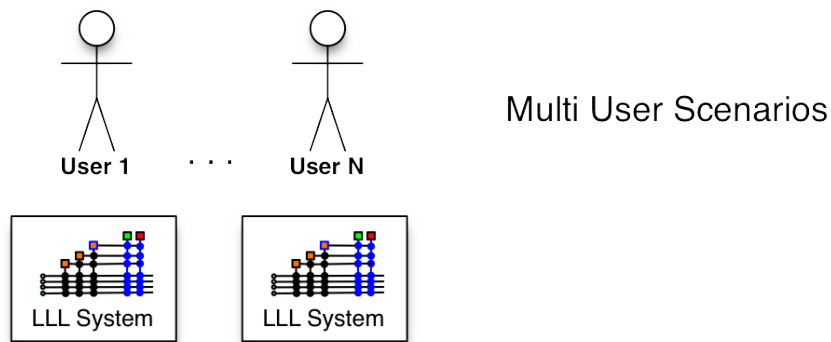


Figure 3.1: CC Life Long Learning scenarios for single users: Scenario 1: The single user could create an agent to chase a target while avoiding obstacles by first training it to approach a stationary target, then training it to chase a moving target, then finally training it how to avoid obstacles. Scenario 2: The user could create an agent for the same task by selecting existing agents with the chaser, and obstacle avoidance skills, then merging them together.

be unreasonable for a single agent to cover the entire state space, or for a single user to train the agent to handle every variance in the environment. Our system would allow multiple users to train several agents to spread the work across several trainers. Figure 3.2 shows a large environment with areas with differing characteristics. Trainers could split the environment up and create agents in each area. These agents could then be merged to share the experience across all of the agents.

Figure 3.3 shows two more multi-user scenarios, the first to consolidate practice of multiple users, and the second to combine the desirable traits of several users. Consolidating practice would allow multiple users to train an agent in the same task and environment. Using multiple users would allow for more thorough training without fatiguing human trainers, would provide better coverage of the tasks state space, and could overcome individual quirks. One issue of observational training is the fact that human trainers can tire. Methods of machine learning that require human intervention suffer from the fact that humans have a limited attention span and resources. Allowing several users to practice a task and then combining their resultant agents would allow a team of people to train an agent far more than any individual person could. A side effect of this would be better coverage of the tasks state space. When using observational training a potential downside is that the training data might not contain information about every possible situation that an agent could face. Having several users training in the same scenario decreases the likely hood that the agent is presented with a situation that doesn't have the benefit of human training data to provide the desired path. A final effect of combining several users efforts to train the agents would be the ability to smooth the agents responses by averaging out individual quirks. For example if one of the users had a tendency to make slow sweeping changes in directions while the other users made quicker more efficient turns, then combining the experience of all the users would keep the agent from learning the quirk.

The second scenario depicted in 3.3 would be the scenario to combine desirable traits from several users into an agent. For example if one user had a tendency to be very aggressive and fast in



Each user is responsible for training in an area
 Resulting agents are merged to create an agent
 that can handle the environment

Figure 3.2: CC Life Long Learning large environment multi-user scenario. The large environment would be split up between several users. Each user would train an agent in their area of responsibility. The resulting agents would then be merged to share their knowledge of the different areas.

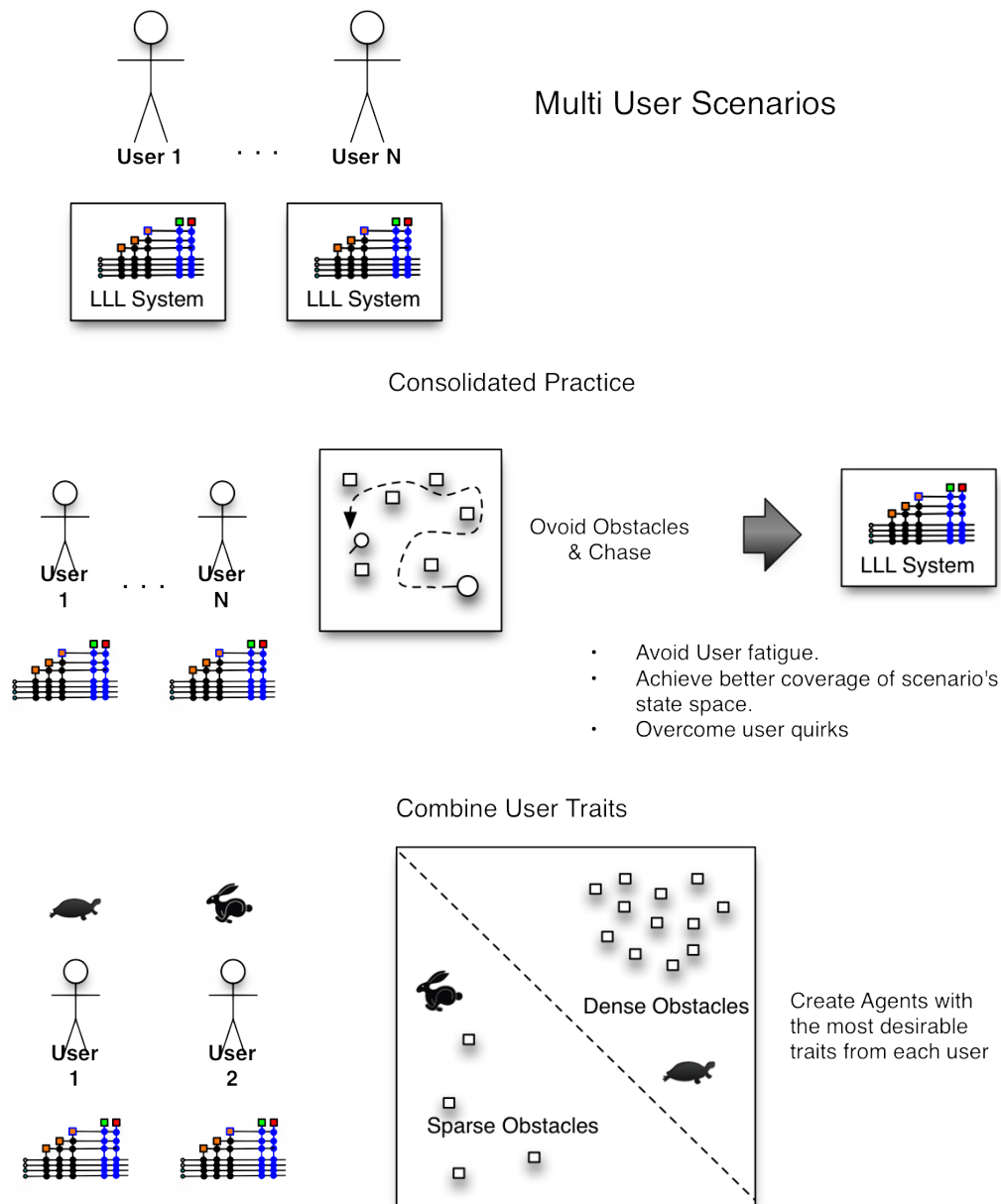


Figure 3.3: CC Life long Learning to consolidate practice and to combine user traits. Consolidating practice would allow several users to train an agent on an identical task. The goal would be to achieve a higher level of coverage of the agents state space, while minimizing user fatigue, and to overcome individual quirks of the human trainers. Combining user traits would allow a user architect to use multiple human trainers with different traits (aggressive or conservative) to create an agent with both traits.

its training, and the other was very slow and conservative, these traits could be combined in the appropriate way to create an agent that was fast and aggressive when it is desirable and slow and conservative when necessary. The figure shows an environment with two areas one area is sparsely populated with obstacles while the other area is densely populated. If the task were to chase a target through this environment, then having a strictly fast or strictly slow agent wouldn't be desirable. However if we combined the training of both trainers such that the conservative trainer takes the lead in the congested area and the fast trainer takes the lead in the open area we could create an agent that better fits the environment. In order to create an agent in this way it would require a human architect to direct which existing agent based on the human trainers should be used in which area of the environment. This is similar to the idea of cherry picking in source control. Cherry picking is the process of selectively choosing which modifications should be combined to create the final merged result. In this case the human architect is deciding which agents and which areas of the environment they should be responsible for. Once the areas of responsibility have been chosen the merging process would combine the networks by creating pseudo-items from each network in their respective areas. These pseudo-items would then be used to train the network resulting from the merged parent networks.

This section has discussed the goals that have motivated our research, and described the types of applications our life long learning system should ultimately be able to solve. In order to progress toward our larger goal we have designed and implemented several software test suites, and performed preliminary and advanced experiments to test the efficacy of our life long learning system. Our first steps were to implement the MTL, CSMTL, and reverberating pseudo-rehearsal algorithms adapted to the CC algorithm in order to determine that the unique benefits provided by each technique remain under the CC generative structure (Section 4.2.2). This was followed by implementing and testing methods for our system to incorporate new inputs in the form of new contexts or new features (Section 4.3.1). These two steps were then combined to implement a Reverberated

Dual CC learning algorithm meant to handle an arbitrary number of sequential tasks with minimal intervention (Section 4.3.2). The Reverberated Dual CC method is tested using both generated toy data and observational data from our OpenNERO human scenarios. The final implementation step was the integration of our Reverberated Dual CC method with the KBCC method of Shultz [56] to create a system capable of incorporating previously trained networks, as well as a method for merging existing networks in the absence of new training data. These techniques are then utilized to create simulated agents from human observation in a simulated environment. The following sections provide more details on our methods and on our simulated test environment.

3.2 Methods & Implementation

This section will cover the theory and methods behind our work. This will include details of our implementation of CC and MTL, and descriptions of our new input method, dual CC, and Dual KBCC life long learning systems.

3.2.1 *MTL Impoverished Task Implementation*

The experiments in section 4.2 focus on the impoverished primary task scenario of MTL. This scenario is of particular interest because of its relationship with life long learning, where new tasks might have few examples, or learning a new task more quickly is desired.

The impoverished task problem provides a minor problem to the standard MTL learning technique in that each output of an MTL example has to have a valid output. In order to use the extra tasks and to meet the constraints of MTL that all tasks have the same number of input and output values, and valid outputs for each input, we allow the networks to deal with unknown values. The method of unknown values used in our experiments is identical to the method used in the η MTL

experiments found in [57]. The impoverished task has its outputs filled in with a special unknown marker so that it has the same number of training points as the other tasks. In our case unknown is represented by the MAX DOUBLE value in C++. When the network sees an unknown output, the error for that output is considered 0. In this way it has no effect on that particular training pattern and is overlooked in the training algorithms. In our case we actually start with the same number of training points for the impoverished task and then randomly replace training outputs with unknown until we reach the desired level of impoverishment.

Unrelated Tasks

An important part of training impoverished tasks is how to deal with unrelated secondary tasks. During training unknowns are treated as having zero error for that output. Ultimately this leads to the primary impoverished tasks having a significantly lowered influence on the hidden layer learning. While this is to be expected, it becomes a problem when some of the tasks used as secondary knowledge are unrelated. These tasks pull the network away from configurations that help the primary task, and because they have more examples, they are much louder sources of error for the training algorithm to deal with.

η MTL deals with this by using a measure of task relatedness to alter the task specific learning rates. Tasks that are unrelated then have a shrinking influence over the course of the training. This technique can not be ported directly to CCMTL because of the way CC adds and trains new hidden nodes. In CC new hidden nodes are added one at a time after they have been trained outside of the network.

Because the error is not propagated back from the outputs we can not use the task specific learning rate to influence the training of the network.

Unrelated tasks and CC

We conjectured originally that CC might be less susceptible to the effect of unrelated tasks because it is less susceptible to the moving target problem, but this does not appear to be the case. Upon further reflection and investigation of the results of our experiments with unrelated tasks, it became apparent that the unrelated task problems exist in CC because of the way candidates are trained and selected. At each stage of candidate training the candidate is chosen that has the highest correlation to the existing error signal. Standard CC does not make a distinction at this stage between the primary task and the other tasks so each output error is used equally when calculating the correlation score. Fahlman states in [24] that the CC learning architecture tends to create new nodes that address the largest current source of error, with each subsequent node covering the remaining error. With this in mind if the unrelated tasks are the outputs with the greatest error signals, CC is going to generate hidden nodes attempting to compensate for those outputs. This is how the algorithm is meant to function, but in the case of an impoverished primary task this implies that the primary task is overshadowed by the other tasks, since the primary task has fewer examples providing the error needed to learn.

This means that the primary task could be mostly invisible to the mechanism of training and selecting candidate tasks. In cases where all tasks are mostly related this is less of a problem because good candidates for its peers are also good candidates for the primary task, but with unrelated tasks this could be an issue for the training of the primary task. Therefore, the problem still exists in CC, despite over coming the moving target problem.

To overcome this CC limitation we adapt the η MTL task relatedness measure as a way to dynamically adjust the weight each output has on the training. In order to use the task relatedness measure in our experiments we had to adjust how it is used. In η MTL it is used to adjust the task specific learning rates in the back-propagation algorithm (details in table 2.1). In CC the hidden layer

nodes are trained according to their correlation to the output error, thus the output gradients are not used during candidate training. This makes porting the η MTL technique to the CC algorithm more complicated.

In the CC algorithm we tested the relatedness measure in two different places. The first place is at the correlation score calculation. This is done by adding the R_k value defined in equation 2.7 to the candidate score calculation defined in equation 2.1. This redefines S as

$$S = \sum_o R_k \left| \sum_p (V_p - \bar{V})(E_{p,k} - \bar{E}_k) \right| \quad (3.1)$$

The second place is the candidate score derivative used in the quick-prop algorithm to adjust new candidate input weights (equation 2.2). This is accomplished by redefining equation 2.2 as

$$\delta S / \delta w_i = \sum_{p,o} \sigma_o R_k (E_{p,o} - \bar{E}_o) f'_p I_{i,p} \quad (3.2)$$

The first location allows the candidate weights to learn from all of the outputs but weighs the selection of the best candidate in favor of the primary task, while the second location moves the training of candidate units away from tasks deemed unrelated.

3.2.2 *New Input Methods*

In most life long learning methods the focus is on incorporating long term experience in a domain into an existing model. The parameters of the domain remain the same, having the same inputs and outputs, but the specific examples to be learned are new. However this is not always the case. In moving between tasks the shape or complexity of the problem inputs could change. Some previous work in knowledge transfer deals with learning sub-problems as a means of improving

performance on the primary problem. The robo-cup domain in particular deals with sub-problems including keep-away with 3v2, 4v3, etc.[68, 69]; additionally this survey of transfer learning by Taylor [67] has several examples of methods dealing with changing state spaces.

In the life long learning system we intend to build new inputs will arise from two possible reasons. First, as a result of using CSMTL as our method of functional transfer, each new task will require a new context input. In the case that the context inputs are mutually exclusive, the addition of these inputs is trivial. If the context inputs are allowed to take real values (to represent partial contexts) their addition is no longer trivial. Second, the addition of new features as a result of complexification. For example, the addition of new sensor inputs for a simulated agent.

Adding new inputs to an existing network will require training to adapt the weights connected to this input to the network. In a standard neural network this might entail adding the input and connecting it to all of the existing nodes with a small initial weight in order to keep it from significantly affecting the existing networks knowledge. The CC network doesn't have to worry about affecting the existing knowledge because of how it builds and freezes structure during training. Most of the existing weights of the network will not be affected by adding a new input into the network. Our problem then becomes how to connect a new input into the existing CC network and how to train it. The first possible solution to this problem would be to add the new input to the network and have it only affect new hidden layer nodes as they are added by CC. This would allow new nodes to take advantage of the new input as they are added to the network, but would mean that all existing nodes would not be aware of the new input. This method provides the most straight forward implementation, and doesn't require any additional training beyond the new task's training. However, it may mean that the new task is not able to get the full benefit that it could get by adapting the existing structure to the new input. Figure 3.4 gives an example of this technique.

The second solution would be to have an additional training step allowing the new task to adjust the

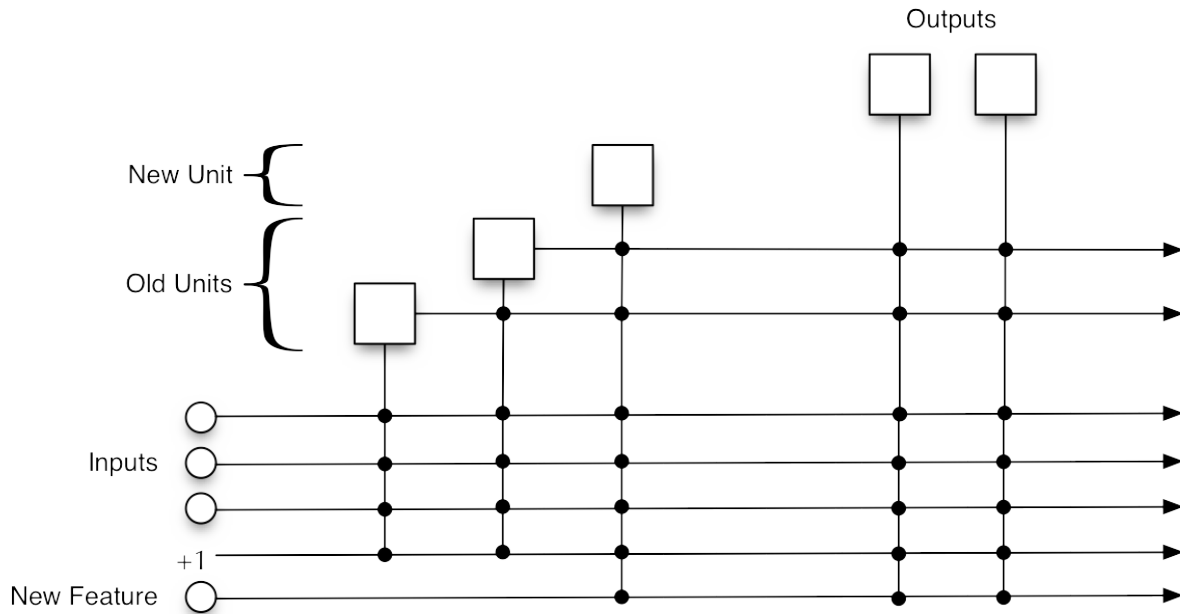


Figure 3.4: Adding a new input to an existing CC network. This depicts the new structure only method. This method only attaches the new input to newly developed structure as the training continues. This is the most straight forward method, and would require no additional training steps.

new input unit to the existing structure before any new structure is added. This method would allow for the best adaptation of the existing structure to be found before new structure is added. This additional training phase is achieved by first connecting the hidden layer nodes to the new input, then freezing all of the network weights except for the weights connected to the new input (The weights indicated in figure 3.5). Quick-prop is then used to train the free weights by propagating the error gradients back through the network. After the new input to hidden weights are trained they are frozen, and the normal output phase of CC is used to train all of the output weights of the network. The addition of weights to the existing hidden layer means that this method could alter previously learned knowledge. In the case of mutually exclusive context inputs this would not be an issue. For normal inputs this could be harmful or beneficial depending on how the new input

affects the previous tasks. These methods are tested and compared in section 4.3.1

Figure 3.5 highlights the structure difference between this technique and the previous one.

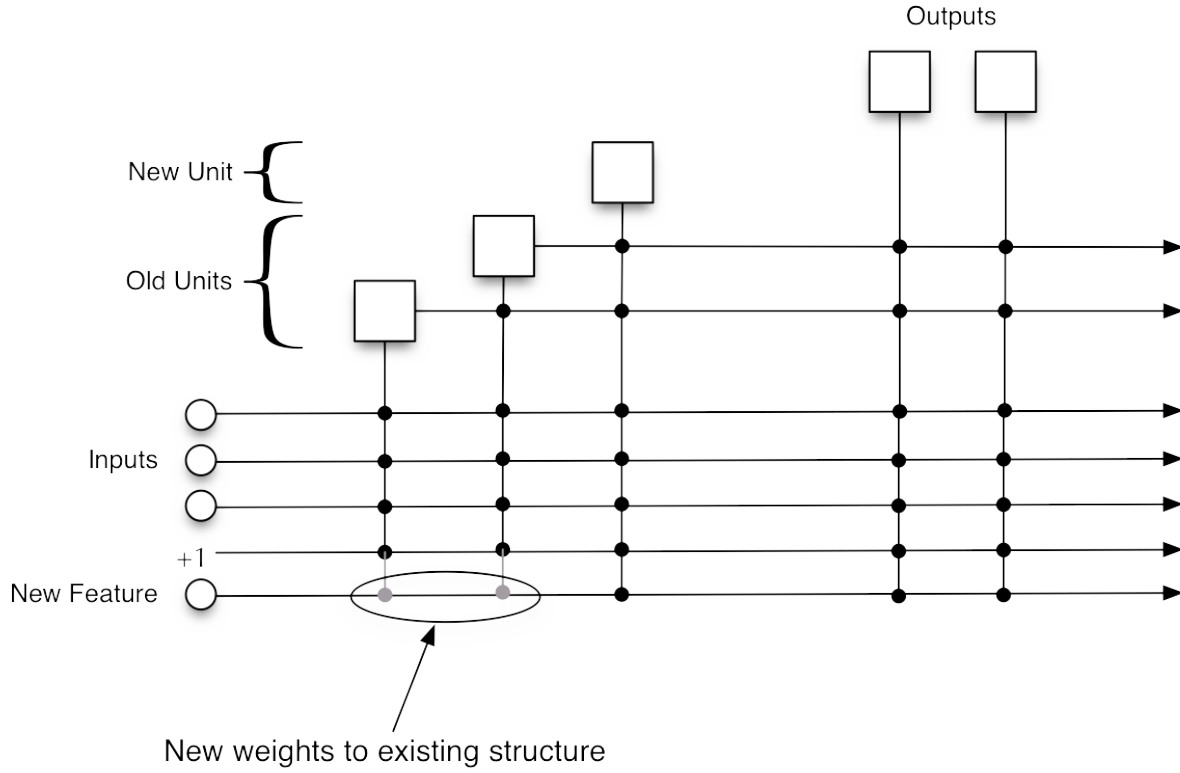


Figure 3.5: Adding a new input while also adding weights to the existing structure. This would allow the new input to adapt the existing structure. In the case of mutually exclusive context inputs we do not have to worry that the new weights would affect previous tasks. This would require an additional training phase to adjust the new weights.

3.2.3 Reverberated Dual CC Networks

This section describes one of our major contributions with this research; the development and testing of a life long learning system utilizing the functional transfer of CSMTL and Ans' Reverberated Dual Network Pseudo-rehearsal; and the representational transfer inherent in the CC

training method. The intended features of this life long learning system are the ability to handle an arbitrary number of tasks without requiring a user to adjust the structure of the network, and the ability to deal with new features as the problem becomes more complex.

Our system consists of two networks; an internal network that is used to reflect our previously held knowledge, and an external network that learns new tasks mixed with pseudo-examples from the internal network. In this way we consolidate the knowledge in the system as new tasks are learned, rather than as a separate process after new tasks are incorporated. Learning new tasks while consolidating long term knowledge will necessarily mean that learning new tasks will be slower than learning the new task in isolation, but if speed of learning is the primary concern then consolidation can be handled in a separate phase. In all of our experiments the new tasks are learned along with consolidated knowledge. The training of our system consists of two phases: new task learning, and mirroring. These mirror stage I and stage II of Ans [3] dual network system respectively. During the new task learning phase the external network is trained on the new task, incorporating any previous knowledge that exists in the internal network. The special case of learning the initial network is identical to learning a task with the standard CC algorithm. During the mirroring phase the external network is perturbed with random input data and reverberated to produce pseudo examples for the internal network to learn. Since both networks create structure as needed, and the existing structure is frozen each phase is taking advantage of the iterative learning ability of CC. Figure 3.6 depicts the dual network learning process.

Reverberated Cascade Correlation

Our dual network system also incorporates Ans' reverberated pseudo-rehearsal technique. In order to implement the reverberating pseudo-rehearsal technique in Cascade Correlation there are a few things that need to be addressed. First the in order to implement the auto-associative weights we

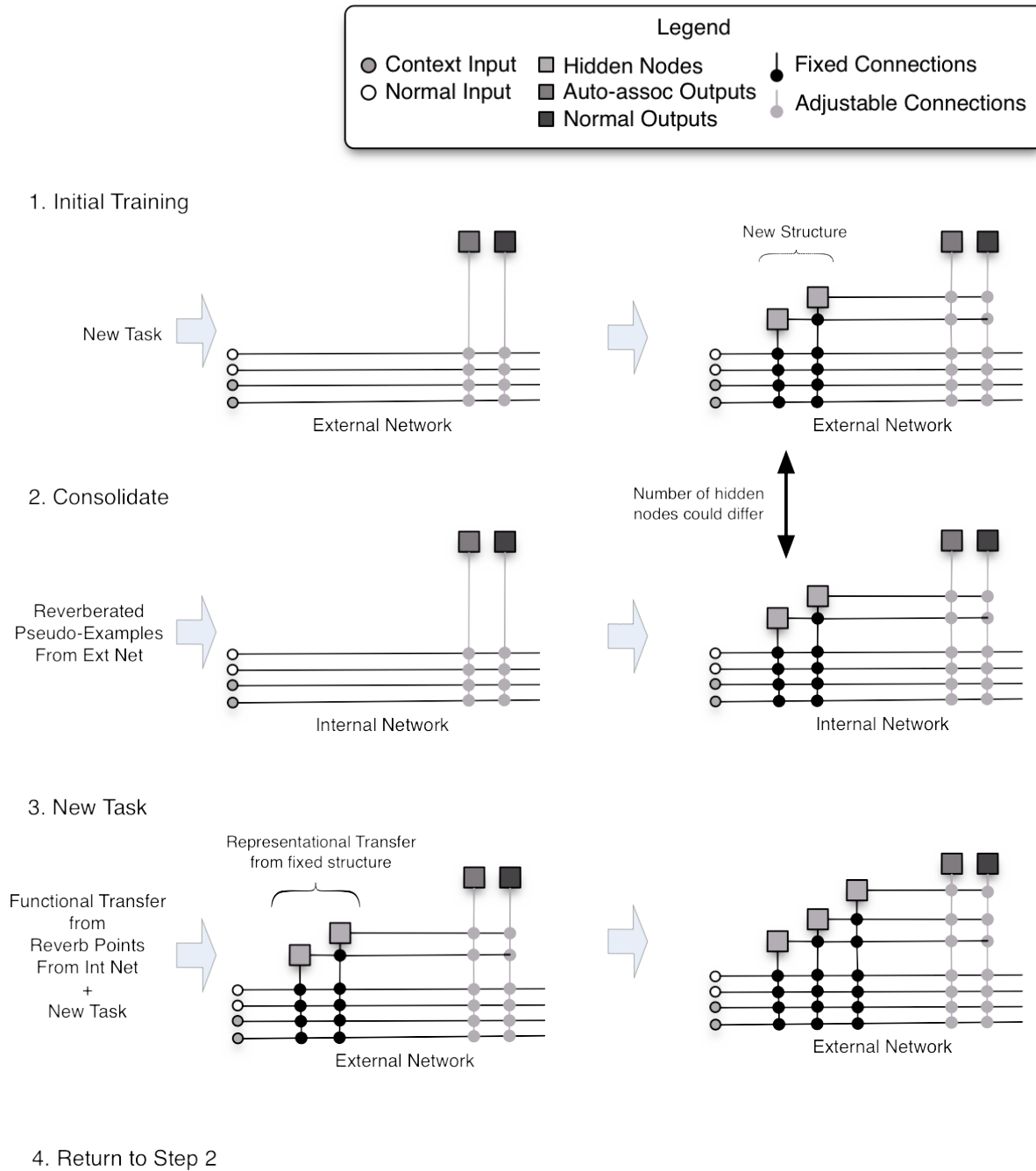


Figure 3.6: The process of the CC life long learning system as developed for this research. After the initial task is trained, the consolidate and new task training phases can be repeated as needed without having to modify the structure of the networks manually or creating any new networks.

create a second set of outputs. The additional outputs are sigmoid nodes since all of the inputs are already constrained between $[0, 1]$. One major caveat of using cascade correlation with reverberation is the fact that CC begins with a minimal network with no hidden layer. If the auto-associative outputs were connected directly to the inputs it would short-circuit the hidden layer. For this reason the auto-associative layer is created without any connections. This causes them to be ignored during the first output phase of the CC algorithm. Once the candidate training phase begins they are treated just like any other output. The candidates are connected to each of the regular and auto-associative outputs once they are inserted. In this way the auto-associative outputs are connected to all of the hidden layer nodes as they are added to the network. Figure 3.7 depicts a CSMTL network with the added auto-associative outputs.

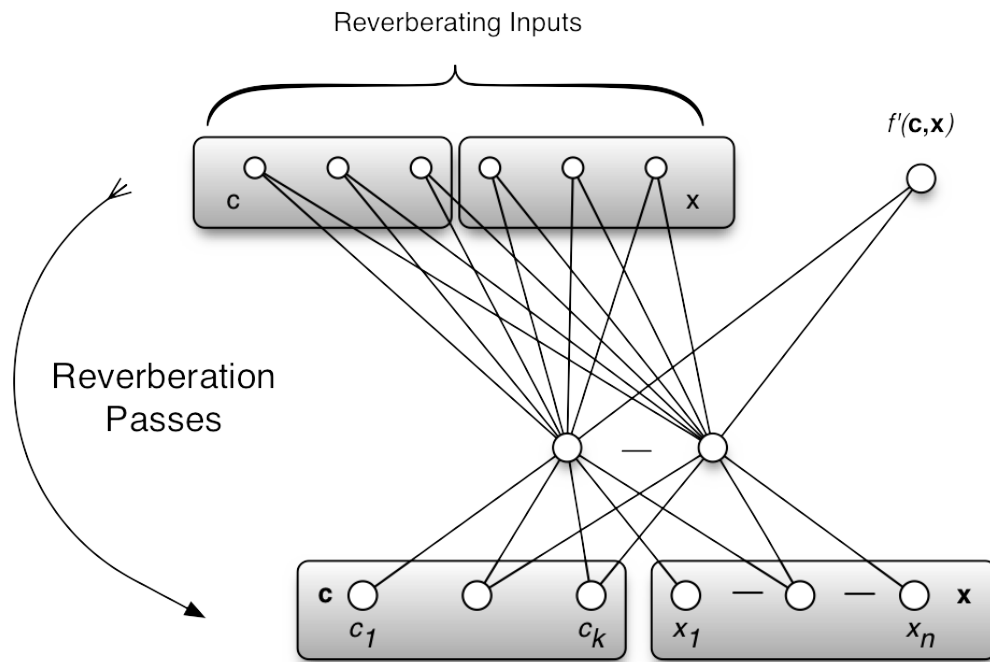


Figure 3.7: CSMTL network set up for self-refreshing memory

The dual network system is tested for its ability to recall previously learned tasks in section 4.3.2,

and for its ability to consolidate training session in section 4.3.3.

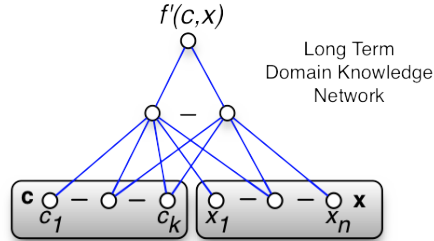
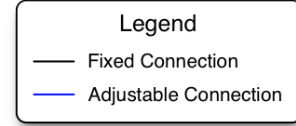
Our method of life long learning shares several features with the CSMTL for LLL of Silver [59], but our system was designed to overcome several shortcomings of Silver's method. First we will be able to grow the structure of the network as it is needed by the algorithm. This gets rid of the issue of having to choose the topology a priori for the first problem all the way to the last problem. In Silver's system it is necessary to have both a long term and short term network, and to have a second phase to integrate the new task with the long term storage network. The only way to achieve this integration with the long term network is to spawn a new long term network with the correct hidden topology and the new context inputs, then to use task rehearsal to retrain the new long term network using the short term and old long term networks. Using a specialized CC network with the reverberating pseudo-rehearsal we'll be able to do this in one step. Using the reverberating process of Ans [4] in cooperation with an adapted CC algorithm will allow us to learn the new tasks sequentially without the catastrophic forgetting that would normally occur. Instead of having a separate connected network learning the new task as in Silver's system, we would have the existing network incorporate the new task directly while rehearsing its existing knowledge. The frozen nature of the CC architecture along with the reverberating pseudo-rehearsal should allow us to learn new tasks sequentially without having to stop between each task to integrate a new task network with the long term storage network.

Table 3.1 list the major differences between our proposed system and the Silver life long learning system. Figure 3.6 and figure 3.8 depict the training processes of our proposed life long learning system, and Silver's CSMTL life long learning system respectively.

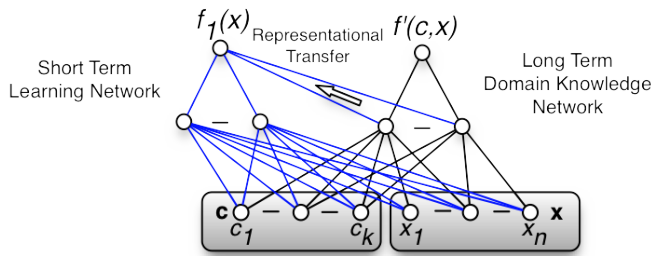
Table 3.1: Differences between Silvers life long learning system and the Dual CC life long learning system

Silver LLL System	Dual CC LLL System
<p>Dual Network Short term network and Long term network. Short term network learns new tasks using representational transfer weights</p>	<p>Dual Network Primary external network and internal reverberating network External network learns new tasks while practicing existing tasks. Internal network is updated after each new task. Representational transfer happens naturally as a side effect of using the single external network.</p>
<p>Consolidation Short term and long term network are used with task rehearsal to create a new long term network from scratch.</p>	<p>Consolidation This happens as new tasks are trained using reverberated pseudo-rehearsal. New structure is grown as needed to accommodate the tasks. Same networks are used throughout.</p>
<p>Functional Transfer Task rehearsal method A form of pseudo-rehearsal. Depends heavily on the effectiveness of the previously trained networks.</p>	<p>Functional Transfer Reverberated Pseudo-rehearsal. An advanced form of pseudo-rehearsal that has been shown to better encapsulate the knowledge contained in a network. Also depends on the effectiveness of the source network.</p>
<p>Topology Fixed and must be specified a priori. Both networks must be chosen before learning. During consolidation the new long term network must also be chosen.</p>	<p>Topology Created on the fly. Both networks will grow to fit the problem as they are trained. They are able to grow independently so they could have different topologies.</p>

1. Train Long Term Network
Choose network topology
Train on secondary tasks



2. Train Short Term Network
Choose network topology
Connect to long term network
Train on new task



3. Consolidate
Choose new long term network topology
Generate pseudo examples
from old LTN and STN

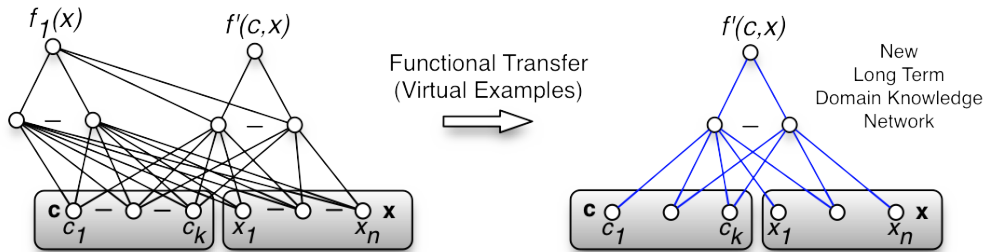


Figure 3.8: The process of Silver's CS MTL life long learning system. Between each new task and consolidation phase a new network must be created requiring the user to determine the appropriate topology. Additionally the system is only able to take advantage of the representational transfer mechanism during new task training and not during consolidation.

3.2.4 *Extending Dual CC Life Long Learning with KBCC*

The final major algorithmic contribution of our research is the extension of our Dual CC life long learning system with the Knowledge Based Cascade Correlation technique. Using the ability of the KBCC algorithm to incorporate previously trained networks into our system will provide us with an additional mechanism for representational transfer in our system as well as provide an elegant way to merge existing networks together.

The primary hardship in implementing the KBCC algorithm is the need to calculate the derivatives of entire neural networks as part of the candidate training process. In our system the existing CC network algorithm is extended to allow for special sub-network nodes that manage the lifetime of the sub-networks and handle the calculation of the networks partial derivatives. The derivatives of the sub-networks are calculated using a feed forward auto-differentiation method similar to the standard error propagation of the back-propagation algorithm. The partial derivative of each node in the network to the network inputs is calculated, starting at the inputs and feeding through to the outputs. Each layer of nodes is used to feed into the calculation of the next layers partial derivatives. Figure 3.9 shows a depiction of this process.

Merging Networks with Dual Reverberated KBCC networks

There are two scenarios where using the KBCC variant provide benefits over the standard dual CC architecture. The first scenario is when there are both existing networks and new training information for a new problem. An example of this would be out toy problems where individual networks have been trained on single tasks and we're interested in learning a both tasks. In this case the previous networks would be included in the KBCC Training algorithm, and the system would train on the combined training data. In this scenario the KBCC technique would be expected to

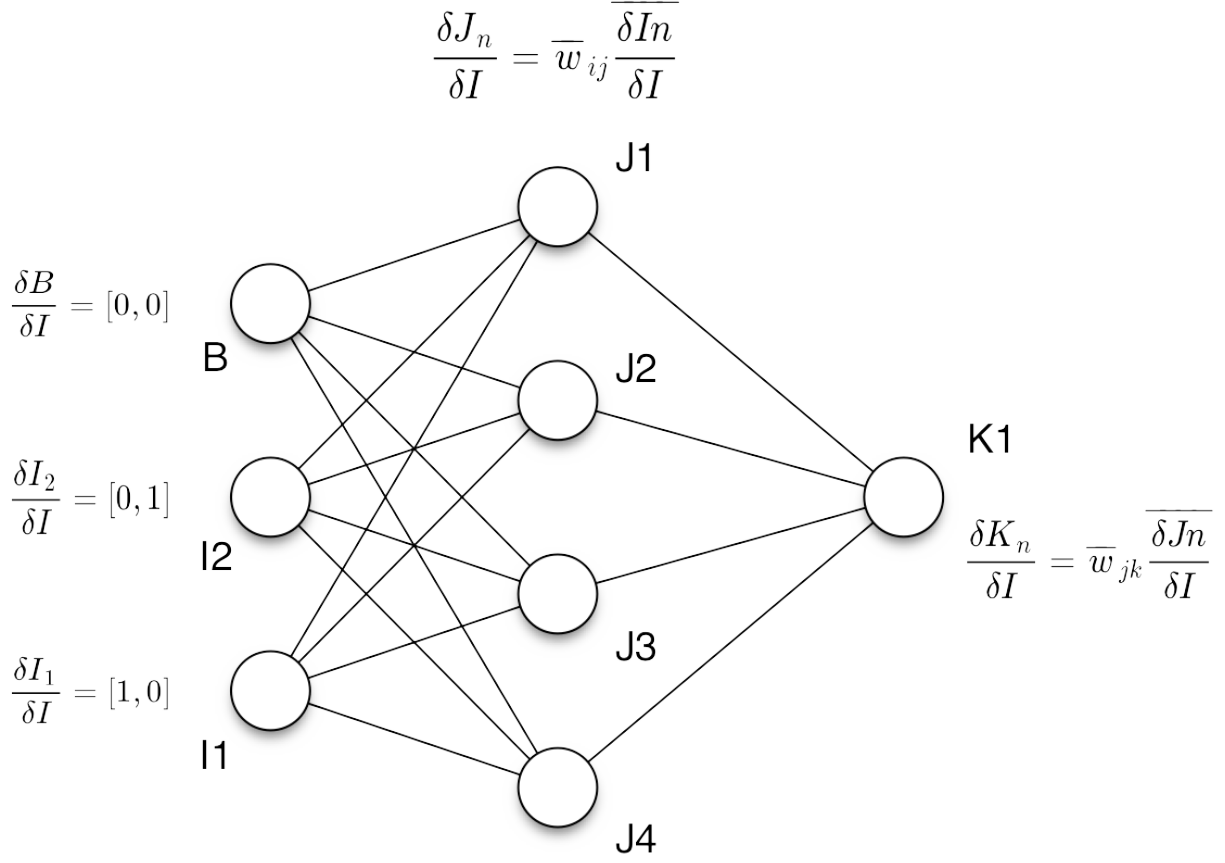


Figure 3.9: This figure shows a depiction of the auto-differentiation used in the KBCC algorithm. The partial derivatives are first calculated at the inputs and then propagated forward through each layer until the derivatives of the outputs have been calculated.

improve the speed at which the combined tasks are learned by incorporating the existing networks. Another example of this would be using networks trained on the simpler tasks in our simulation to improve the learning speed of the more complicated learning tasks. The second scenario is when there are only existing networks. In this case we use our dual reverberated system to generate training samples from the existing networks to create a training data set on which the new merged network will be trained. The existing networks are used as sub-networks during training. In this way we have both functional transfer (from the reverberated pseudo-rehearsal) and representational

transfer (KBCC) from our previous networks to our new merged network. Figure 3.10 shows a graphical depiction of this process.

The usefulness of the merging ability is best demonstrated by an example from our simulated agent scenarios. An agent is created that has learned to chase a target, this agent is then used to as the basis for an agent that chases through obstacles by one user, and an agent that chases and shoots by another user. These two agents could then be merged together to form an agent that chases through obstacles and shoots. This type of situation is experimented with in our later simulated experiments.

Figure 3.11 shows a possible lifetime path of an agent using our life long learning system.

3.3 Simulation Implementation.

The larger goal of our research is to build a system for life long learning that is user friendly from an end user stand point. One of the applications for our work is learning agent behaviors from human observation. We developed a simple game with several scenarios that a user can play. While the user is running through the scenarios the simulator is collecting training data that we later use as observational data sets to train neural networks. The game is implemented in the OpenNERO [66] framework. This framework was chosen because it provides the tools to generate simple games through the use of the python language, while providing the underlying graphics and user interface. This allows us to develop different test scenarios while keeping the programming burden relatively light. The use of the python language also allows us to incorporate our developed methods directly into the game through our own python interface. The following sections detail the simulation scenarios and the implementation details.

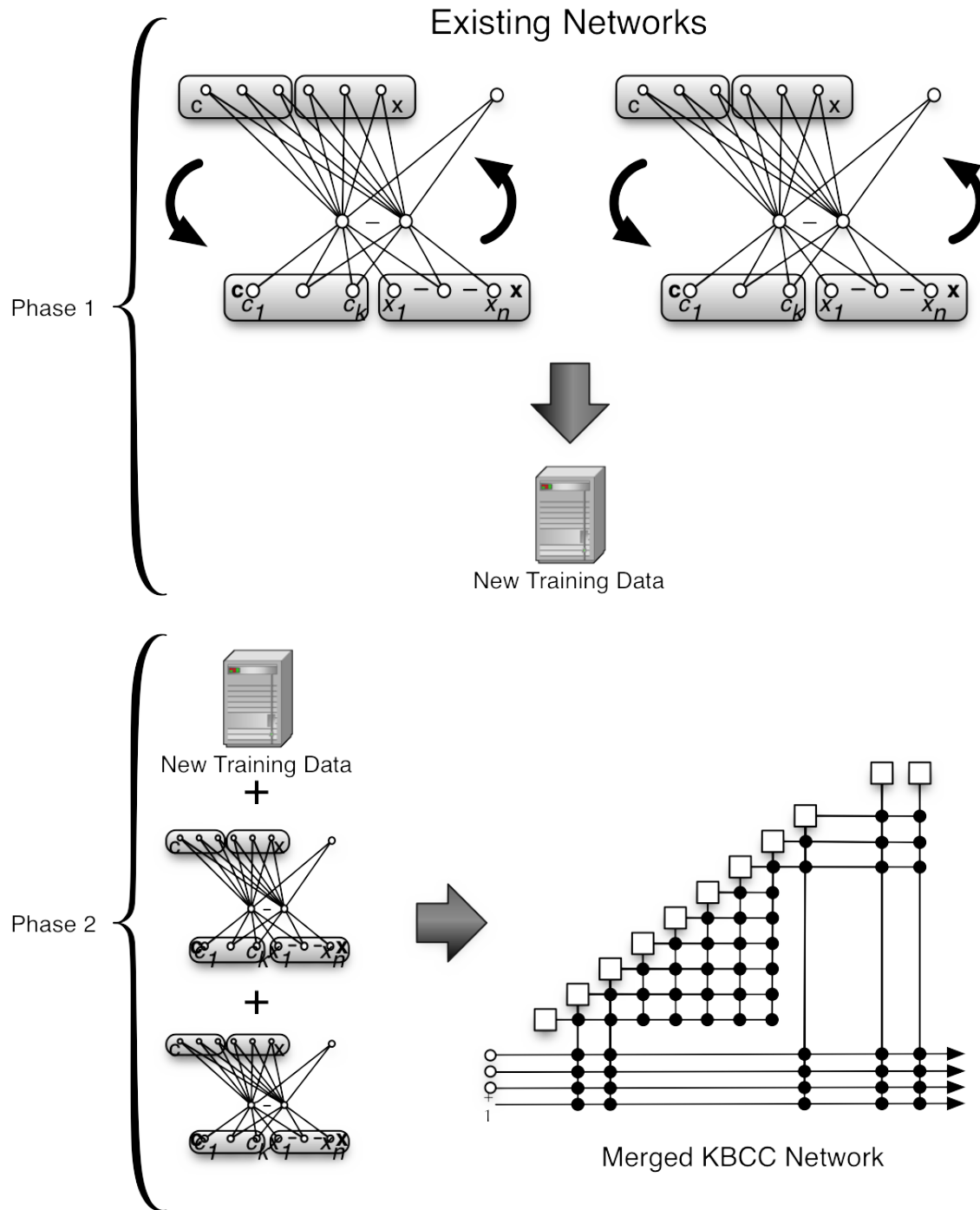


Figure 3.10: This figure depicts the merging process when only the original networks are available. The existing networks are reverberated to generate a new training set and then they are used as sub-networks to train a new dual KBCC network.

Merging Divergent Learning Tracks

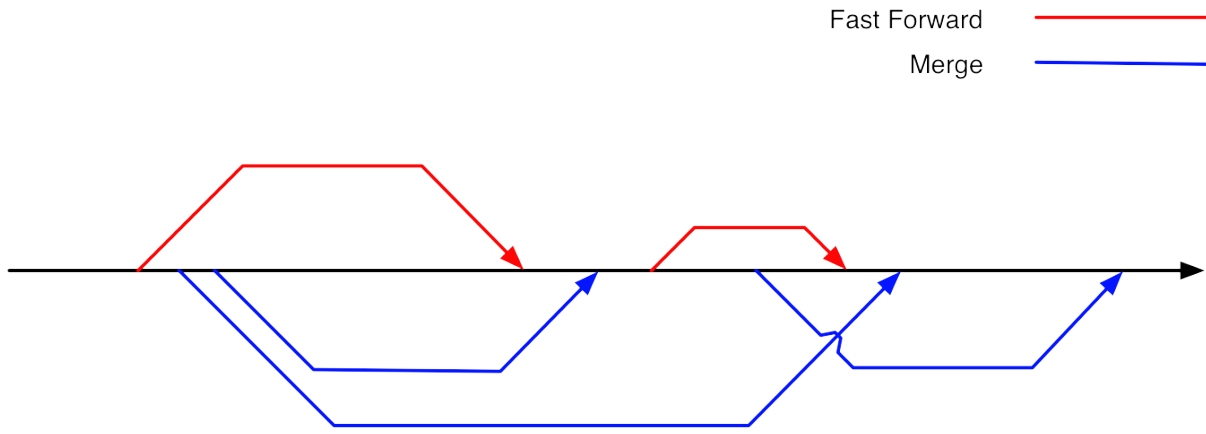


Figure 3.11: Depiction of the diverging and merging learning paths of the sparse life long learning system. Red tracks denote fast forward situations. These situations would be incremental learning situations handled by the standard life long learning system. The blue tracks represent instances where a merge would be necessary. These occur because the base network has been modified before the blue tracks have finished.

3.3.1 Game Scenarios

Chase Scenario

The chase scenario is the simplest of all the scenarios. The user and a scripted agent are placed in random starting locations inside a large square area with no obstructions. The object of the scenario is for the human operator to chase the scripted agent, which is programmed to avoid them, around the arena. The scenario lasts for 30 seconds during which the user attempts to remain as close as possible to the scripted agent.

Obstacles Scenario

The obstacles scenario places the user in the same arena this time filled with trees, and a center wall. The objective of this scenario is to maneuver around the arena while avoiding the obstacles. This scenario also lasts 30 seconds. Users were asked to approach near enough to obstacles to light up their sensors but to otherwise avoid running into them.

Chase-Obstacles Scenario

This scenario is a combination of the two previous. The user and scripted agent are again placed in random starting positions inside the arena, that is now filled with trees, and a center wall. The objective in this scenario is exactly the same as the first with the added complexity of avoiding the obstacles while chasing. This scenario is meant to be more difficult than the first.

Chase-Shoot Scenario

The Chase-Shoot scenario adds the complexity of trying to fire on the runner agent while chasing. Each shot landed adds points to the users score. This scenario while very similar to the Chase scenario proved somewhat more difficult as the shooting mechanics require a very precise shot to register.

Chase-Obstacles-Shoot

The Chase-Obstacle-Shoot scenario combines all of the previous features. Users must chase the runner through the obstacles simultaneously avoiding collisions and attempting to score shots on the runner. This is the most difficult scenario.

3.3.2 Scoring

For the chase and chase-obstacles scenarios the users are scored based on how well they are able to follow the moving target. The score is simply calculated as the length of time they were able to remain within a threshold distance of the agent. The threshold value chosen was 40 units. For comparison the arena is a 800x800 unit square. As each individual run lasts 30 seconds the maximum possible score would be 30. This would be difficult to achieve since both the user agent and scripted agent begin randomly placed within the arena. Scoring was not collected for the 'Obstacle' scenario as the scoring was based on a metric related to chasing only. Scoring for the chase-shoot and chase-obstacle-shoot scenarios are split into scores for chasing and scores for shooting. The shooting score is the total number of hits achieved during each 30 second episode.

Table 3.2 shows the scores for User 1 for all scored scenarios. It is interesting to note that the score for the obstacle scenario is higher for this user. It seems as if the obstacles hinder the runner agent enough to help the user stay within close proximity. The chase scores during the shooting scenarios are also improved for this user.

Table 3.3 shows the scores for User 2. User 2 starts strong in the chase scenario scoring the highest average, but seems to have issues performing in the Chase-Obstacles scenario. The scores improve again once the obstacles are removed for the Chase-Shoot scenario, while also achieving a good shooting score. In the Chase-Obstacles shoot scenario, the chasing score is again diminished, but the shooting score improves considerably.

Table 3.4 shows the scores for User 3. User 3 has the worst Chase scenario average of any of the users, and the Chase-Obstacle score is also low. The chasing score improves significantly throughout the shooting scenarios. User 3's performance in shooting was really low during the Chase-Shoot scenario, but picked up considerably during the Chase-Obstacles-Shoot scenario.

Based on these results User 3 should be avoided if the task is chasing in wide open field, but should be the first called upon when obstacles are introduced. This is in direct opposition to User 2. To learn shooting Users 1 and 2 would be good instructors, while User 3 could be useful as well so long as the obstacles are included.

These scores will be used later to compare with the scores of agents created from their observational data.

Table 3.2: User 1 scores. Scores for Chase are out of 30. The score for shooting is the number of his landed during the 30 second window.

	Chase	Chase-Obstacles	Chase-Shoot		Chase-Obstacles-Shoot	
	11.46	17.05	20.30	431.00	18.82	457.00
	10.03	17.64	21.00	506.00	25.20	522.00
	11.66	19.37	23.15	497.00	26.81	801.00
	10.40	11.51	21.14	589.00	12.95	556.00
	13.41	17.71	23.55	685.00	26.59	725.00
	12.78	16.60	22.53	709.00	24.44	732.00
	15.00	15.29	24.79	791.00	22.01	663.00
	11.32	15.07	22.16	697.00	20.34	621.00
	13.25	14.80	24.04	848.00	25.64	714.00
	11.06	17.64	17.91	583.00		
Mean	12.04	16.27	22.06	633.60	22.53	643.44
SD	1.54	2.19	2.03	134.84	4.55	113.10

3.3.3 Sensors and Actions

To generate the observational data sets the user's agent is equipped with several sensors which act as the input features of our data, and actions which act as the output features. The sensors are designed to provide the data set with information about the user's awareness of the situation. A description of the sensors and actions is provided in table 3.5. All of the sensor values are in the range of $[0, 1]$. The actions however range from $[-1, 1]$. The normal output range of our CC neural

Table 3.3: User 2 scores. Scores for Chase are out of 30. The score for shooting is the number of his landed during the 30 second window.

	Chase	Chase-Obstacles	Chase-Shoot		Chase-Obstacles-Shoot	
	23.81	12.47	20.27	154.00	16.74	646.00
	23.67	21.28	27.86	182.00	18.72	538.00
	23.40	1.73	20.11	238.00	22.18	705.00
	25.39	4.56	22.18	317.00	22.46	499.00
	23.43	19.27	23.59	400.00	17.70	583.00
	22.01	12.34	21.88	205.00	11.31	588.00
	23.46	22.00	27.04	325.00	20.95	574.00
	23.43	9.07	19.55	217.00	21.29	750.00
	21.82	13.95	14.26	639.00	16.59	681.00
	24.85		16.66	689.00		
Mean	23.53	12.96	21.34	336.60	18.66	618.22
SD	1.08	7.11	4.20	188.20	3.57	82.44

Table 3.4: User 3 scores. Scores for Chase are out of 30. The score for shooting is the number of his landed during the 30 second window.

	Chase	Chase-Obstacles	Chase-Shoot		Chase-Obstacles-Shoot	
	12.50	16.46	21.39	89.00	23.27	113.00
	9.18	19.71	21.18	69.00	24.94	75.00
	9.92	13.51	21.85	81.00	19.88	121.00
	8.81	11.72	24.48	66.00	18.94	212.00
	10.04	17.91	24.99	45.00	25.89	319.00
	12.22	15.78	24.26	83.00	20.71	188.00
	8.41	13.34	22.98	87.00	23.14	256.00
	13.20	7.01	23.35	90.00	25.86	309.00
	12.56	19.67	24.24	88.00	27.19	283.00
	7.25	15.27	22.17	97.00	23.39	329.00
Mean	10.41	15.04	23.09	79.50	23.32	220.50
SD	2.07	3.87	1.38	15.39	2.76	93.23

networks is from $[0, 1]$, so the outputs from the user data are distributed into multiple outputs in two different ways. The first way is for the negative ranges of each output ($[-1, 0]$) to be split off and turned into their own outputs in the range $[0, 1]$. This gives us 4 outputs; one each for forward, reverse, left, and right. The second method is for the left/right output to be split up into several discrete ranges. This method splits the turning signal into seven different outputs; three left ($[-1, -0.5]$, $(-0.5, -0.2]$, $(-0.2, -0.05]$), three right ($[0.05, 0.2]$, $[0.2, 0.5]$, $[0.5, 1]$), and one center ($(-0.05, 0.05)$). This is done to discretize the output and to provide a method for the turn signal to have different discrete intensities. These two methods were implemented to make the Observational data sets more palatable for the network. One last important note about the user data sets. There are two input methods available to the users. The first input method is the standard arrow keys on the keyboard. These provide maximum values for each of the four directions. In our experiments User 1 provided input using the keyboard. Users 2 and 3 played through the scenarios using an xbox controller. Their actions are therefore represented by continuous values representing the joystick positions along the two axis. This means that user 1's data is very discrete in its response to the inputs, while users 2 and 3 have smoother responses. The two different methods are used to test the differences in fidelity between the users and the simulation. We will see how this impacts our observational learning performance in later sections.

Table 3.5: Sensors and Actions of the user agent. These are used to create an observational data set from users running the scenarios.

Sensors	
Index	Description
0-4	Wall Ray Sensors: Each sensor represents whether or not a wall is within a certain distance of the agent. These are fanned out in front of the agent.
5-6	Obstacle Radar: These sensors project a cone and return whether or not there are obstacles in the cone. The strength of the sensor represents how close the obstacle is. There are two of these sensors facing in front of the agent.
7-18	Enemy Radar: These sensors form a circle around the agent and respond with the presence of the scripted agent and how close.
19-20	Enemy Angle and Distance: This sensor gives a relative heading and distance to the scripted agent. It performs a similar function to the enemy radar sensors, but with a much finer degree of accuracy.
Actions	
0	Forward / Reverse
1	Left / Right
2	Fire / Don't Fire (Only used during shooting scenarios)

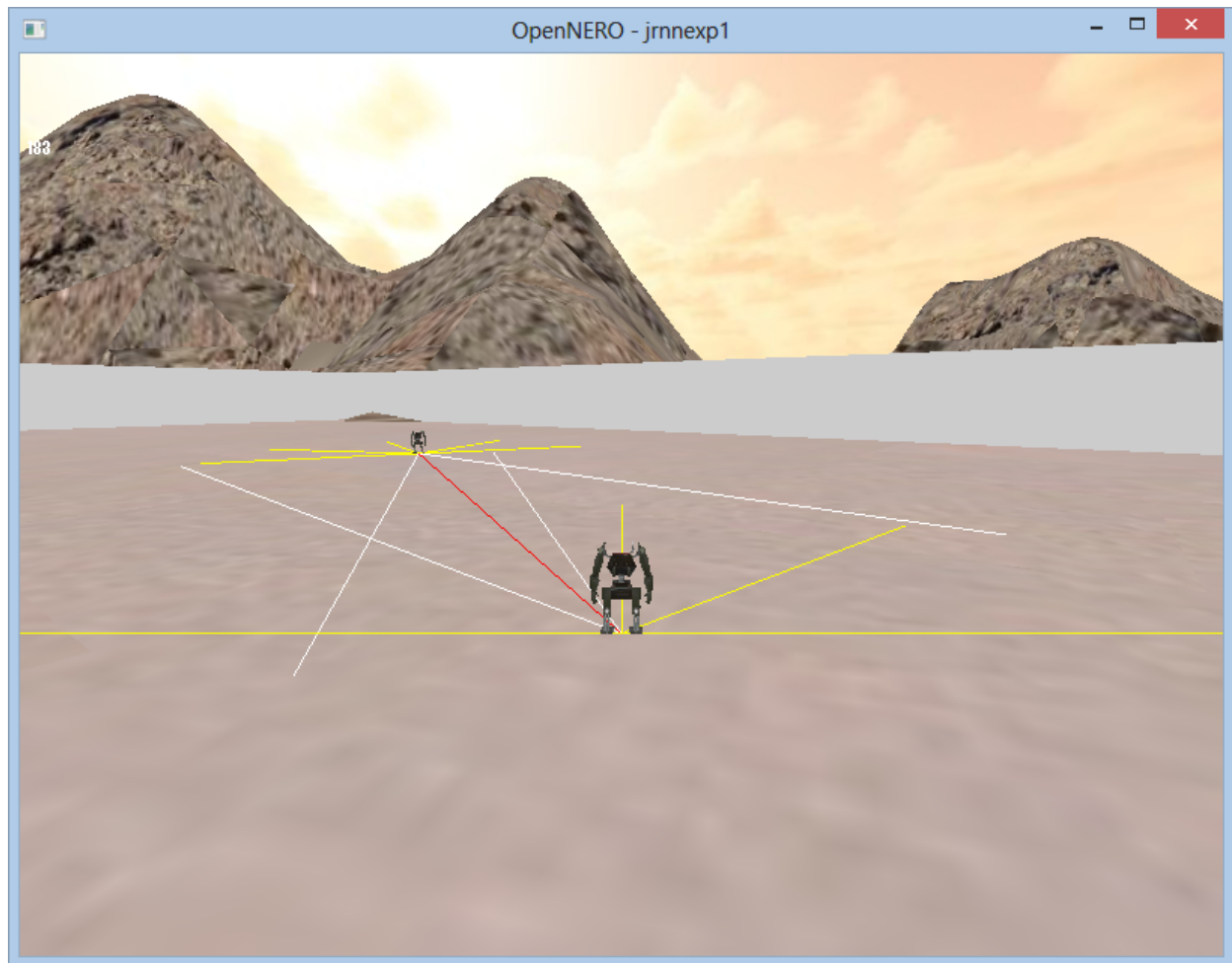


Figure 3.12: Screen shot of the OpenNERO Simulation. This scene depicts the chase scenario.



Figure 3.13: Screen shot of the OpenNERO Simulation. This scene depicts the obstacles scenario.

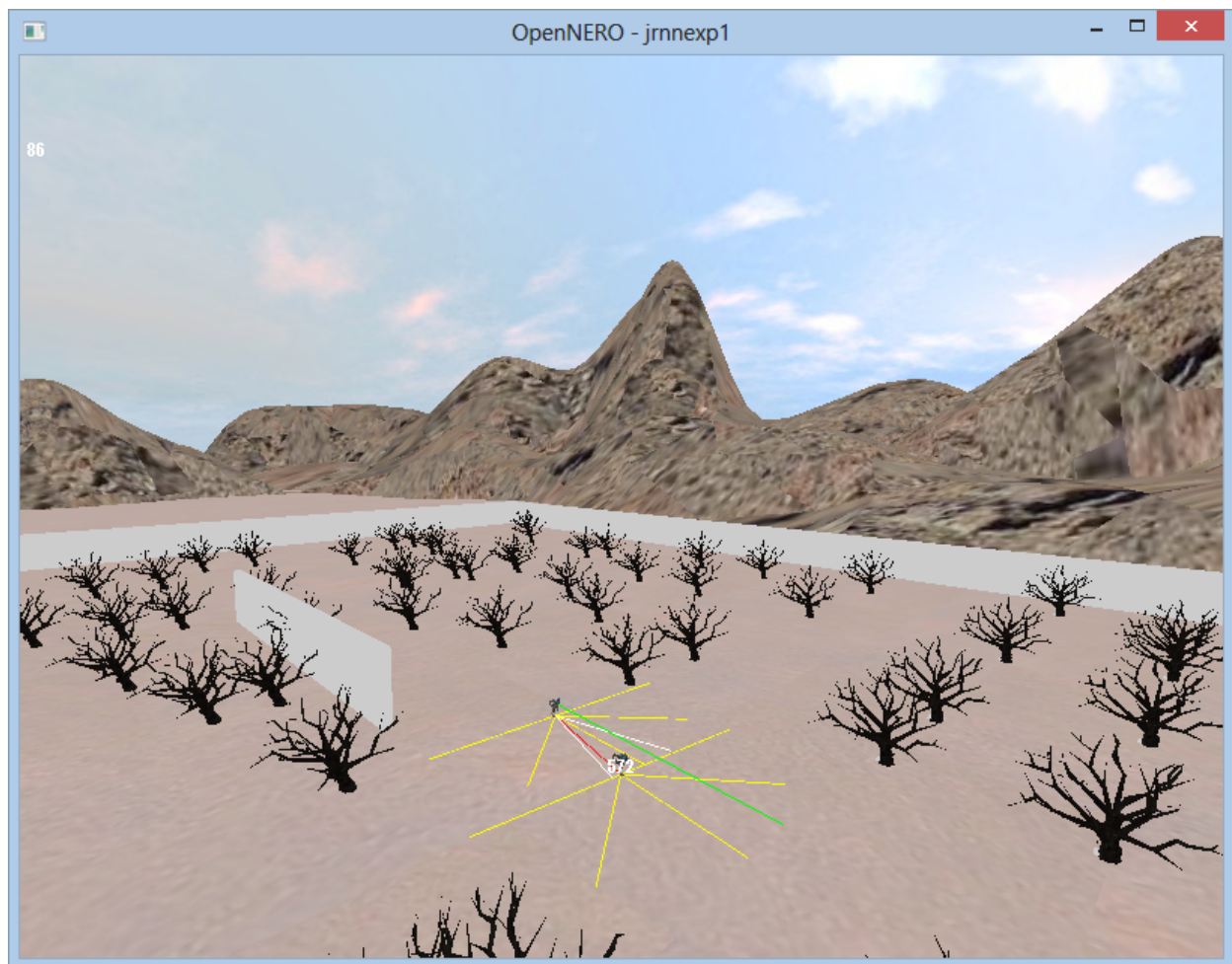


Figure 3.14: Screen shot of the OpenNERO Simulation. This scene depicts a replay of the chase-obstacles-shoot scenario

CHAPTER 4: RESULTS

This chapter details the experimentation and validation of our implemented ideas. Each of our contributions is tested under various conditions to evaluate their performance. The preliminary experiments with the MTL techniques are conducted with generated and real classification data sets to mimic the testing techniques in the literature while the dual network life long learning methods are tested on both the classification problems and our observational training sets derived from our human trainers.

The experiments detailed below fall into two categories. The first set of experiments are designed to test the efficacy of multi-task learning when the regenerative structure that the cascade correlation algorithm creates and trains, is used. We refer to this structure, for brevity, as CC. Several experiments are conducted to test different aspects of MTL and to verify that CC maintains all the same benefits that standard fixed structure neural networks do. The second set of experiments detail the use of a dual CC network life-long learning system using reverberated pseudo-rehearsal as well as a method for adding new inputs to the network after it has already learned a task.

Experiments are carried out using the JRNN neural network library available at [49]. Scripts controlling each of the experiments are included as well as scripts for parsing the resulting data. Initial experiments with the back-propagation algorithm did not provide an equal footing against the cascade correlation algorithm in terms of speed of learning so all fixed structure experiments were conducted using the RProp [50, 51] algorithm. RProp performed comparably to quick-prop and cascade correlation, and has a straight forward implementation that uses many of the same calculations used in back-prop. For comparison, to learn one of the band problems with back-propagation to a MSE threshold of 0.05 it took roughly 30,000 epochs, while the same feat is accomplished in RProp within 1000 epochs. There are two primary parameters of the RProp

algorithm, η_+ and η_- , which control the step size used to adjust the weights. For our experiments we used the suggested parameter values of 1.2 for η_+ and 0.5 for η_- .

Validation is used in many of the experiments below to restrain the networks from over learning the training set. It is used as a method of early stopping, which is recommended in the MTL literature to alleviate some of the issues of learning multiple tasks at the same time as well as several issues with neural networks in general. There are some differences in validation between RProp and Cascade correlation that bear mentioning. In RProp a validation epoch is done after each update epoch. If the error on the validation set stops improving for a defined number of epochs the network is reset back to the best set of weights found and training stops. In this way the RProp algorithm has a very fine level of control on stopping concerning the validation set. In Cascade Correlation the staged training of outputs and candidate units makes this type of validation impractical. Cascade correlation trains the output weights of the network until the training stagnates, reaches the desired threshold or times out. In each of these cases a various number of training epochs will have occurred each time this phase is completed. If the network has not reached the desired threshold a new candidate is trained and installed into the network and then output training occurs again. In Cascade Correlation the validation step is inserted between the output training phase and the candidate training phase. In this case validation checks to see how many candidate training cycles have been completed without an improvement on the validation set. Again when the validation patience has run out the network is set back to the best set of weights found thus far. This method of validation in CC is much more coarse than the validation used in RProp. This method is used because it is the method used in the Fahlman implementation of CC [24].

Our implementation of cascade correlation and quick-prop are based on the C implementation available at [21]. All of our experiments use a patience parameter of 12 for both the output and candidate phases. This means each phase will be considered stagnant if their performance hasn't improved after 12 epochs. The phases are considered to have improved if their error on the training

Table 4.1: RProp & CC Training Parameters

RProp Parameters		CC Parameters		
			Output	Candidate
$\eta+$	1.2	P	12	12
$\eta-$	0.5	CT	2%	3%
		ϵ	10	100
		μ	2	2

set has decreased by at least 2%. For the quick-prop algorithm we use a value of 2 for the μ parameter, and the values of 10 and 100 for the ϵ parameter for the output and candidate phases respectively. The epsilon parameter is scaled in the quick-prop algorithm according to the number of incoming weights for each node. These parameter choices come from the suggested values in the original CC [24] and quick-prop implementations [22].

Table 4.1 contains the primary parameters used for the RProp and the CC algorithms. The parameters P and CT represent the *patience*, and *change threshold* respectively. For both algorithms these parameters were selected from the suggested parameters in their respective publications.

4.1 Experimental Problems

For our experiments we have chosen a selection of real and generated data sets to get a broad view of the performance of CC and MTL under different conditions. Details for each of the data sets are given below.

4.1.1 Band Domain

The band task has been used to demonstrate the Task Rehearsal Method [58] as well as the CSMTL [61] method. It is a synthetic domain consisting of concept learning tasks with different levels of similarity. The relatedness of the tasks is controlled by the orientation of the band of positive examples. Any number of tasks can be created by changing the orientation and thickness of the positive bands. This domain is useful because it can generate tasks with differing levels of relatedness, which can then be used to measure how well an MTL, or life-long learning algorithm deals with related or unrelated tasks. For the purposes of our experiments 8 tasks have been created. The input space is two dimensional ranging from 0 to 1 in both axis. The bands are .4 units wide and centered on the coordinates (0.5, 0.5). Band orientations of 45, 30, 60, 50, 135, 120, 150 and 140 are used. These orientations give us two groups of related tasks that are unrelated with the other group.

4.1.2 Circle In Square

The circle in the square data set has been used in many neural network applications as an example of structure in structure data. The data set has been altered to create multiple tasks by changing the size of the circle and moving the circle around inside the square.

4.1.3 Linear Tasks

The linear data set is a two class problem like the band task. The two classes are separated by a linear discriminate where all points on the same side of the line are of the same class. The tasks in this data set differ in the angle of the discriminate line. Similar tasks have angles within a few degrees of each other while unrelated tasks have angles approximately 90 degrees apart.

4.1.4 *Glass Database*

The glass data set is the smallest data set we have used, it contains only 214 examples. It is also available from the UCI Repository, contains 9 attributes and 6 possible classifications. The goal is to classify the type of glass from the 9 attributes; possible types being building windows, car windows, containers, tableware, and headlamps. Once the data set has been normalized and converted to a collection of binary tasks we are left with 6 tasks, with 214 examples each.

4.1.5 *Dermatology Database*

The dermatology data set is used to classify 6 types of skin disease, it has the highest number of attributes used in any of our tests (33). The data set contains 366 examples which means our final data set contains 6 binary tasks with 366 examples each. Again each attribute value has been normalized between 0 and 1. The dermatology database is also available at the UCI Repository.

4.2 Experiments with Multi-task Learning and Cascade Correlation

These experiments are designed to test the use of the Cascade Correlation algorithm for the impoverished MTL learning scenario. The CC algorithm is compared against a fixed structure network in the same scenarios. We are looking for validation that the MTL benefits that are reported using standard networks are maintained while generating the structure on the fly. This is an important foundational step toward our goal of a generative life-long learning system using MTL and CSMTL as our method of functional transfer.

4.2.1 Preliminary Experiments

Two preliminary experiments were conducted to set the stage for our CC MTL experiments. The first preliminary experiment seeks to find the parameters for each data set which lead to impoverished training performance. These parameters are then used in the CC MTL experiments. The second preliminary experiment is an experimental validation of the effects of incorrect neural structure on the performance of the network.

Preliminary Grid Search

The objective of this experiment is to find the parameters that lead each data set to an impoverished training result. The two parameters of interest are the number of hidden layer nodes, and the number of training points.

Method

The parameters are found by performing a grid search over each of the two parameters of interest. For each data set the number of hidden units is varied from $[1, 2, 4, 6, \dots, 20]$, and the number of training points is varied from $[5, 10, 15, \dots, 100]$. Each run of the parameters is repeated 30 times and the average of the classification error rate is stored. The grid is then translated into a heat map showing the relative performance of the network at each combination of parameters.

Results

The heat maps were inspected visually to roughly find the points at which the networks could no longer function. Rather than looking for the exact defining point, we chose values that consistently

Table 4.2: Experimental values derived from grid search.

Data set	Training Points		Hidden Nodes		
	Impoverished	Normal	Impoverished	Normal	Oversized
Band	20	100	2	4	10
CirInSq	20	100	2	6	12
Derm	10	50	1	3	10
Glass	20	50	2	5	10
Linear	10	100	2	4	8

gave significantly worse results in order to be confident in the impoverished state of learning. The results are presented in Table 4.2. The chosen parameters are notable in that the normal values for the hidden layer are higher than what the underlying structure of each problem might find optimal. This is intentional as the selection was chosen to provide reasonable results across all of the training set sizes. It is shown in [15] that the number of hidden layer nodes larger than necessary are viable as long as a method of early stopping is used. These values are only used during the experiments in section 4.2

Preliminary Hidden Layer Size Experiment

This experiment shows the case of using a neural structure that is incorrectly chosen. This is an experimental validation of the pathological conditions of a network that is too small, and a network that is too large without validation. The results of these structure choices are well understood in the literature [15, 72], and the results of these experiments are presented here only as a validation of the parameters chosen in 4.2.1 and as a visual demonstration of the effects of choosing the wrong structure.

Method

Each data set is run 60 times with the normal, too small, and too large number of hidden layer nodes. The classification error rates are collected and the average is reported. Results for the CC network are also reported for comparison.

Results

Figure 4.1 illustrates the results of structure choice for the MTL technique, and figure 4.2 show the same results for the CSMTL technique.

4.2.2 MTL Impoverished Primary Task Experiments

The following sections detail our experiments incorporating cascade correlation with the MTL and CSMTL techniques. The objective of these experiments is to determine if the functional transfer effects of MTL and CSMTL are present when the structure of the network is developed during training. Each of the experiments in the section deal with the specific case of learning a primary task with impoverished training data using full secondary tasks.

Method

In all of the experiments, runs are repeated 60 times with random starting points, and with the training, validation, and testing sets re-sampled from the whole data sets. A stopping criteria of MSE of 0.05 or a failure to improve the validation score is used throughout these experiments. Each of the measurements reported are averaged over the 60 runs. The size of the training sets are chosen based on the results from the preliminary experiment in section 4.2.1, while the validation

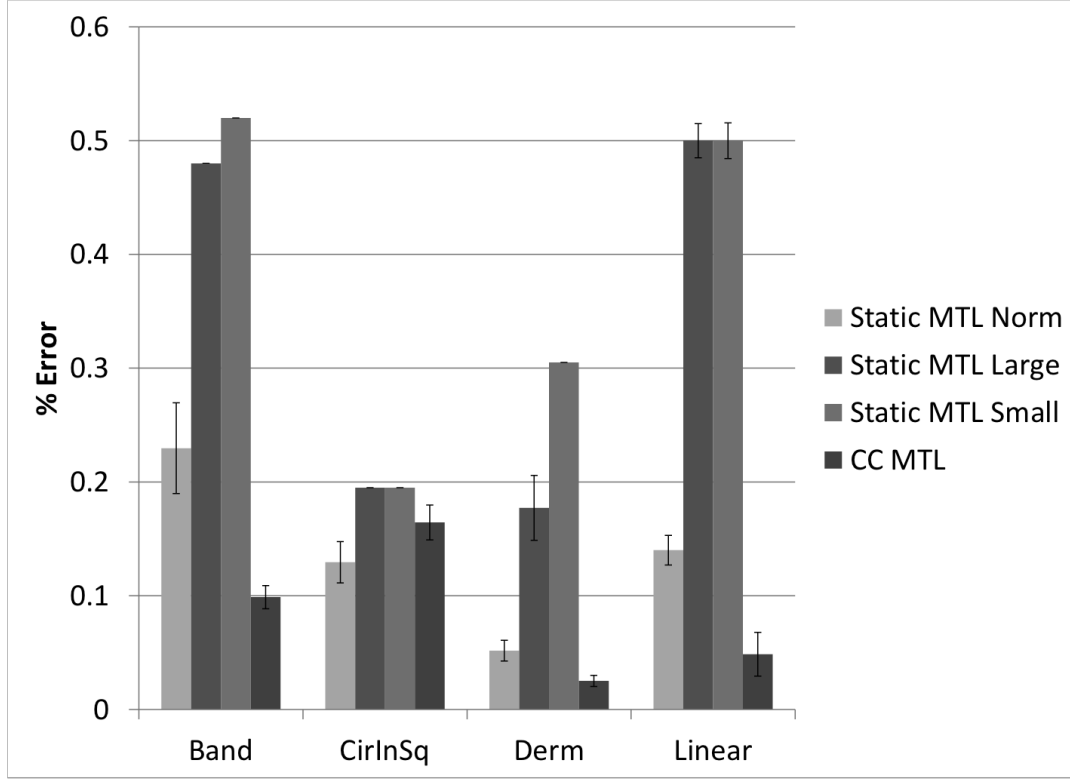


Figure 4.1: The effects of choosing the wrong hidden layer on MTL. In each data set the performance on the primary task is significantly degraded when a non optimal hidden layer size is chosen. The results are most pronounced in the band and linear data sets, where the networks perform no better than random selection.

and testing sets contain 200 examples each. The exception to this is the glass data set which had too few data points to support these set sizes. The glass data set uses a validation set size of 50, and a test set size of 100. In all of our experiments the training, validation, and testing subsets are selected from the entire data set at random while maintaining output class distributions.

For the hidden layer sizes of the MTL networks it is recommended in [14] to always use larger

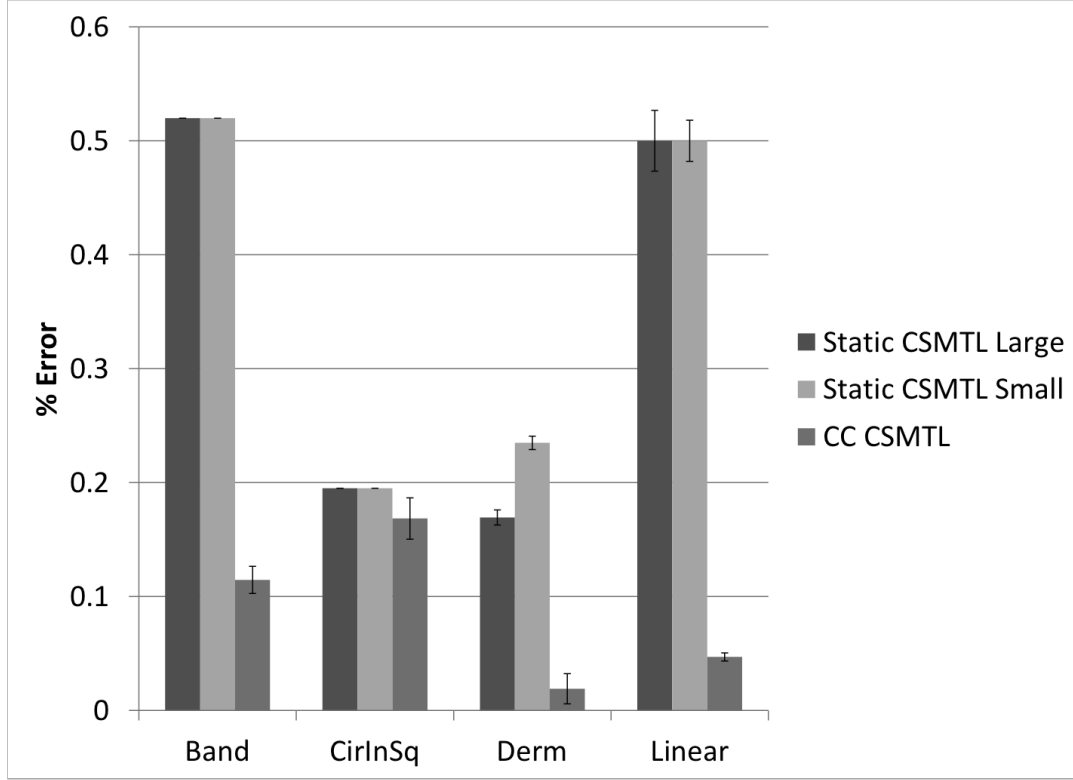


Figure 4.2: The effects of the wrong hidden layer size for static networks compared to CC network for CSMTL problems.

hidden layers than might otherwise be believed necessary, and in [57] it is recommended to use a multiple of the number of hidden layer nodes needed for each task singly. So if each individual task requires 2 hidden nodes it is recommended to use $2 * numOfTasks$ hidden nodes. This rule is followed in each of the MTL and CSMTL fixed networks.

MTL Impoverished Primary Task Results

This section details the results of the standard impoverished primary task problem for the static and CC networks. The results show the impoverished primary task trained on a single task (STL) network and again with an MTL network.

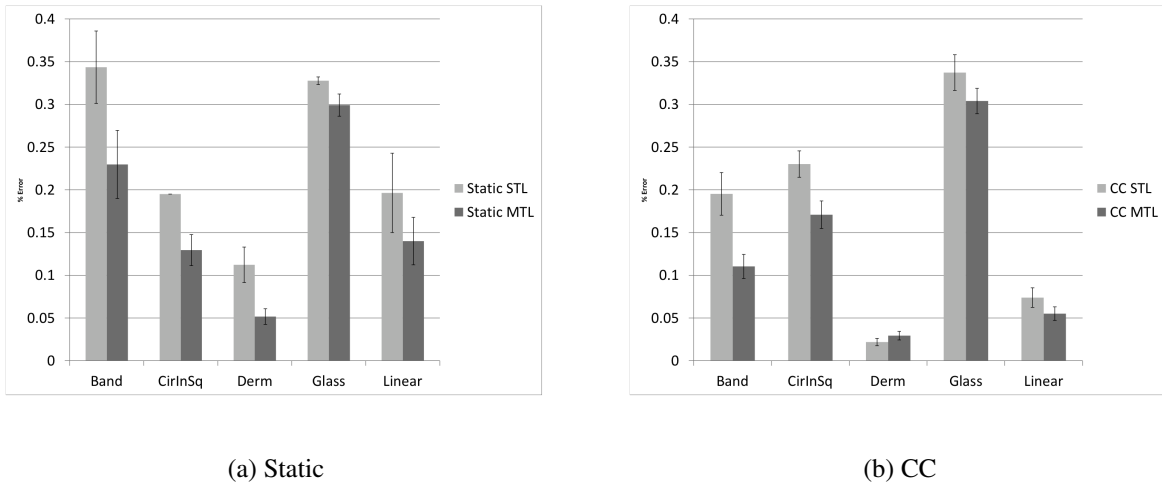


Figure 4.3: MTL Impoverished Primary Task: STL vs MTL with Static and CC networks. This result shows that the CC network is still able to achieve similar results to the static network using the MTL technique. This shows that we can generate the structure of the network while using MTL as a source of inductive bias.

Figure 4.3 shows that in most of the data sets the STL network has issues learning the primary task under the impoverished data set. The two cases where this is not as pronounced are the linear and the Dermatology data sets. These data sets are using very few primary task training points and still able to learn the problem to a small degree of error. This suggests that these data sets are very easy to learn and do not have much room for improvement from MTL. In all of the data sets where the impoverished primary task is difficult to learn the static (RProp) and generative (CC) MTL allows the network to improve its test performance significantly. This is the expected result, and it

shows that the generative network is still capable of utilizing the bias from MTL while generating the necessary structure. This is important because as we emphasized a number of times by now choosing the network structure is not an easy task. These problems are compounded in the MTL scenario because a delicate balance is needed between competing needs of the tasks. On one hand the inductive bias in MTL relies on the pressure of learning all tasks with a shared hidden layer [57], but at the same time if there is not enough room in the hidden layer even related tasks will cause issues for each other. In either case, this type of decision is only possible if the number of tasks to be learned and the correct number of hidden nodes is known from the beginning. In a system where tasks are meant to be learned sequentially as they arrive, choosing the representation a priori is not feasible. For this reason the result in figure 4.3 is encouraging because it means we can use CC to generate the network structure progressively, as needed, while still retaining the ability to use MTL as a source of inductive bias.

MTL Impoverished Primary Task Results with Unrelated Tasks

The experiments were repeated using one secondary task closely related to the primary task and one secondary task considered unrelated to the primary task. Figure 4.4a shows the results of using η MTL in the impoverished primary task with unrelated secondary tasks problem using the static neural network. In each of the data sets the η MTL technique provides some improvement over the standard MTL training. The use of the η MTL technique improves the performance in the face of an unrelated task.

To determine which of the two weighting approaches, mentioned in Section 3.2.1, is most effective in the CC algorithm we refer to the results in Figure 4.4b. The results show that using the η MTL weighting in the correlation score calculations provides the best improvement when training with unrelated tasks. This implies that the scheme of choosing the best candidate node for the primary

task provides more benefit than weighting the training algorithm in favor of the primary task. The candidates would still be trained using the error signals from all of the outputs but the candidate selection is weighted in favor of the candidates score with the primary task, and any tasks deemed related to the primary task.

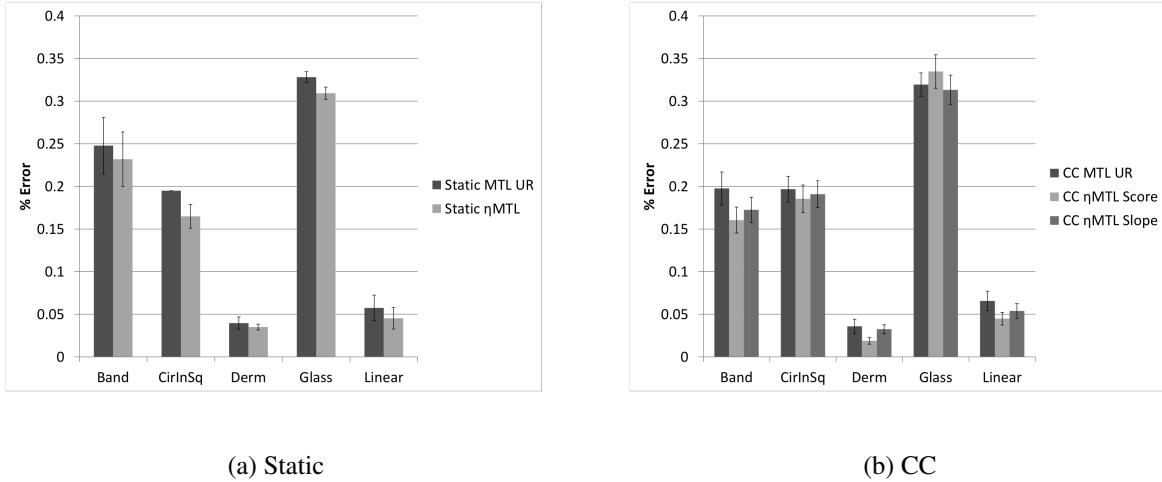


Figure 4.4: Results for η MTL in the impoverished primary task with unrelated secondary tasks. In the static case η MTL provides a benefit to each data set, with the benefit in the CirInSq and glass data sets being the most significant. In the results for the CC network using η MTL in the calculation of the candidate scores provided the most benefit in four of the five tasks. This would suggest that the weighting the candidate score based on how well it correlates with the primary and more related tasks is more beneficial than adjusting the gradients based on relatedness of the outputs.

4.2.3 CSMTL Impoverished Primary Task Experiment

The impoverished primary task problem was repeated using the same parameters as the previous MTL experiments, using CSMTL from section 2.3.4 as the method of functional transfer.

Method

In this experiment we use the same data sets and primary tasks as the MTL experiments in Section 4.2.2. The number of training points for the primary task is again taken from Table 4.2. For the CSMTL experiments the impoverished primary task is handled differently than in the MTL experiments. CSMTL uses a single output that is shared amongst all of the tasks, therefore it doesn't require the use of unknown values to fill in the primary tasks training set. However the primary task still requires special handling. The CSMTL network is trained in batch mode with all of the examples from each task trained together. In this situation the primary tasks proportion of the training set would be unequal to the proportion of examples from the secondary tasks. This would lead to the primary task having fewer weight updates than relative to the secondary tasks. This is handled by duplicating the impoverished primary task examples to match the number of examples provided by the secondary tasks. For each data set the primary and secondary tasks are combined, and the primary task examples are duplicated to reach the normal training set size. Each data set is run 60 times with a random starting network. Training is stopped when either an MSE of 0.05 is reached or performance on the validation set stops improving. The performance of the networks is averaged over all the runs. This process is repeated for both the static and CC algorithms.

Results

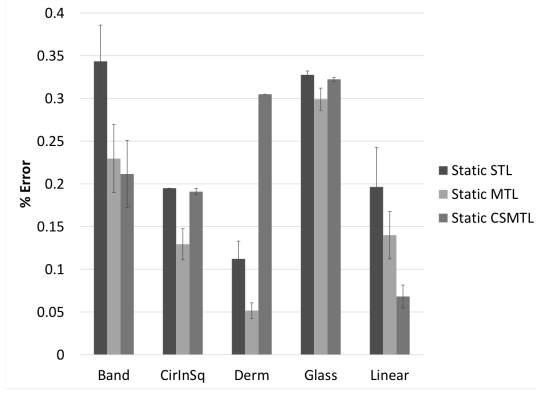
The results of the CSMTL technique are depicted in Figure 4.5. The results show that for the generated data sets (Band, Linear, Cirrus) the CSMTL technique improves the performance of the impoverished primary task. The band data set shows this improvement for both the static and CC networks, however the CirInSq and linear data sets each show improvement for only one of the techniques. The primary difference between the resulting CC network and the static network is one of depth. The CC network creates a new layer for each candidate node that is added to the network.

The results seem to indicate that CSMTL handles the deeper network structure of CC better in the CirInSq set than it does for the linear set, and vice versa for the flat structure of the static network. This is an observation that warrants further investigation.

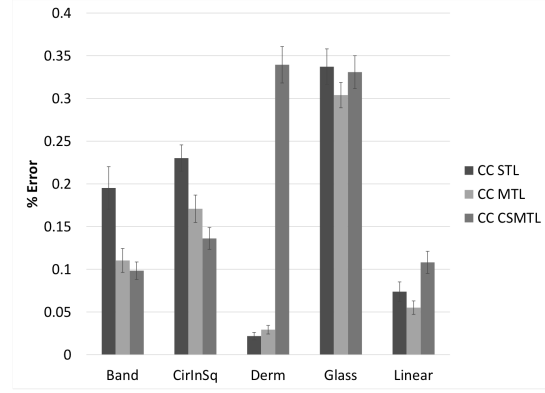
The results for the real data sets (Dermatology and Glass) show a different picture. For both the static and CC network CSMTL doesn't improve the primary tasks performance and in the dermatology data set it actually hinders performance. The difference in performance between the MTL and CSMTL for these data sets stems from the way these data sets were separated into tasks. Each of the tasks in these data sets are mutually exclusive, meaning that none of the examples have overlapping outputs. For the MTL technique this is not an issue as each task has its own output. For CSMTL this is an issue as each task is going to give conflicting outputs for each example input. The context input should allow for the network to differentiate, but in this experiment the results suggest that a single task specific bias does not allow the network to learn the impoverished primary task over the conflicting secondary tasks. For this reason these data sets are left out of the Dual CC experiments in Section 4.3.

4.2.4 CC MTL Conclusions

The results from this set of experiments provide a foundation for our future work by showing that we are still able to use the functional transfer techniques provided by MTL and CSMTL while generating the network structure on the fly. In the next set of experiments we continue this line of work by implementing a sequential learning system using dual CC networks and functional transfer through CSMTL to provide a framework for life long learning.



(a) Static



(b) CC

Figure 4.5: Results for CSMTL Impoverished Primary Task. These results show that the CSMTL technique performs slightly better than MTL on the band task in both the static and CC cases. For the CirInSq and Linear data sets the results are split between the static and cc cases. CSMTL performs better on the linear data set on with the static network, while it performs better on the CirInSq data set on the CC network. One possible reason for this result could lie in the sensitivity of the data sets to the cascaded structure of CC. The results suggest that in the CirInSq case CSMTL takes advantage of the cascaded architecture of CC while it is hindered by the flat hidden layer in the fixed network. The opposite effect is apparent in the linear data set CSMTL performs poorly for the Glass and Dermatology data sets for both the Static and CC networks.

4.3 Dual CC Life Long Learning Experiments

The experiments in this section move beyond the impoverished MTL and CSMTL scenarios and focus more on the sequential life-long learning scenario that is one of our main contributions. The first experiment in this section will test two methods for adding new inputs to an existing network without having to retrain the network from scratch. This ability will be useful for adding new context inputs for separate tasks, but could also be useful for adding new features as they become available. The second experiment will focus on the use of the dual CC network in learning two sequential tasks. It looks at the retention of the first task’s knowledge after learning the second task

and compares a single network, dual networks, and a dual network using reverberating pseudo-rehearsal. The final experiment will compare the same dual network system in a consolidated training scenario. Consolidated training is the situation where a large training set is divided up and training occurs in batches, or when training is carried out over time. We look at how the single network, dual network, and reverberated dual network perform for different problems. The experiments in this section are performed on several of the classification sets used in our previous experiments.

4.3.1 New Input Experiments

The first experiment in this section will test methods for adding new inputs to an existing network. This ability is important for a system intended for life-long learning for several reasons. First in our case of using CSMTL for functional transfer it is useful to be able to add new context inputs on the fly as new tasks arrive. Secondly it is also plausible that new features could be added to a problem later as they become available. To test this method we take the iris data set from the UCI repository [27] and two of the classification data sets from our earlier experiments, and attempt to learn them using a subset of their inputs. We then add the additional inputs to see how the network responds. We tested two methods for adding the additional inputs to our CC networks as described in section 3.2.2.

Method

In each of the following experiments a CC network is trained on a subset of the data sets inputs. The network is trained until performance on a validation set stops improving. After training with the subset the new inputs are added and the network is trained with the full input space of the problem. This process is repeated 30 times with random starting weights and the classification

error on a test set is averaged and reported for the 30 runs.

New Input Results

The first set of results comes from the 2 class Iris data set. This is a very well known data set in the classification literature, and it is commonly known that only two of the four inputs are needed for the classification task. This is because the classes are almost linearly separable using the Petal Length and Petal Width Attributes.

We start by training the network using only the sepal length and sepal width features. Once this is done we add new inputs representing the petal length and width and continue training. Each of the methods are tested to see how each affects the learning.

Figures 4.6 and 4.7 show the classification error rate, and the number of epochs respectively. The results from both methods are nearly identical. They both provide significant improvements in classification accuracy when the new features are added, and they both require similar numbers of epochs to incorporate the new features, however, method 2 takes slightly more. The number of pre-existing hidden nodes averages about 1.5 nodes increasing to an average of about 3 nodes in the final networks for both methods.

At this point it seems as if the differences between the two methods are negligible but the results for the Linear data set show some difference. Figures 4.8 and 4.9 show the results for the Linear data set. The error rate result shows that both methods provide for the same decrease in error rate moving from the single input to both inputs. The major difference between this set and the previous data sets is that this linear set is easy to classify even with the limited input set. The additional input provides a slight boost, but the error rate was very low to begin with. This is in contrast to the previous data set where the networks basically had to learn most of the problem from the

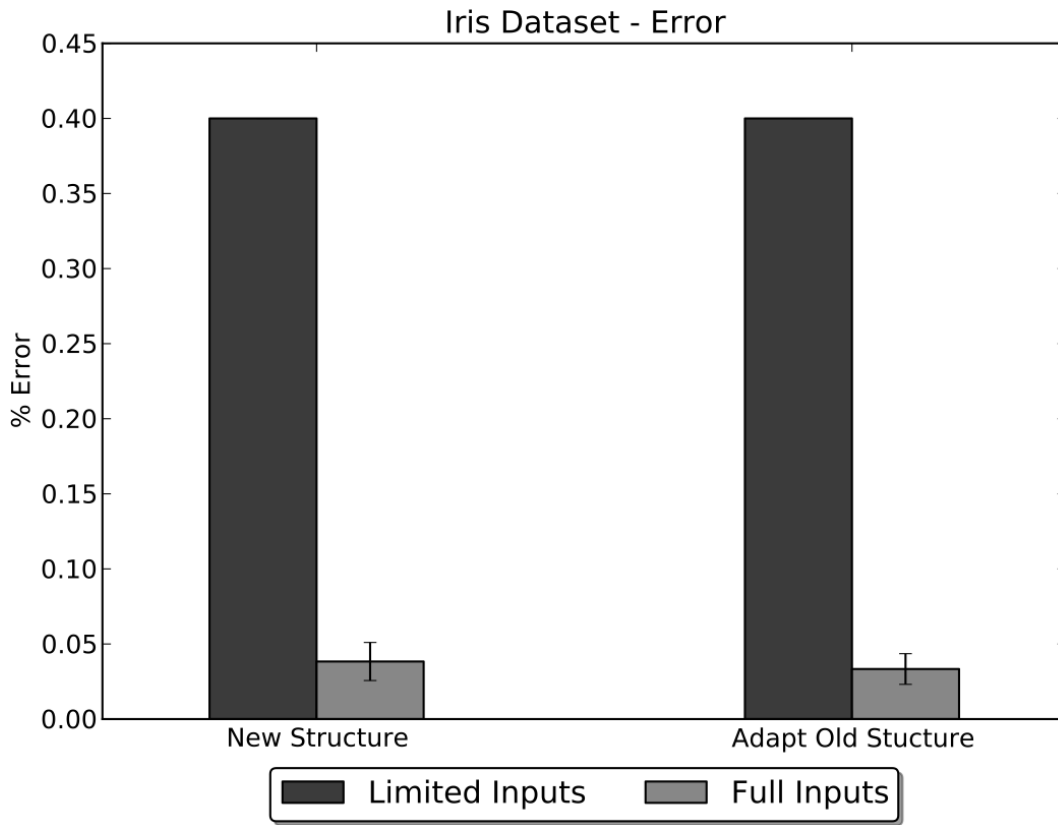


Figure 4.6: New Input Experiment: Classification error of the Iris data set using limited inputs and full inputs under two methods of adding new units. Both methods seem to provide the same level of improvement between the two feature sets.

additional inputs. The difference is most clearly seen in the number of epochs required. In both methods only a handful of epochs are required, but in method 2 all of those epochs occur during the adaptation of the existing structure. In both methods the number of hidden units before and after the new inputs were 1.

The results from these experiments while providing evidence that adding the new inputs to the CC network works well using either method, neither stands out as superior over the other. They

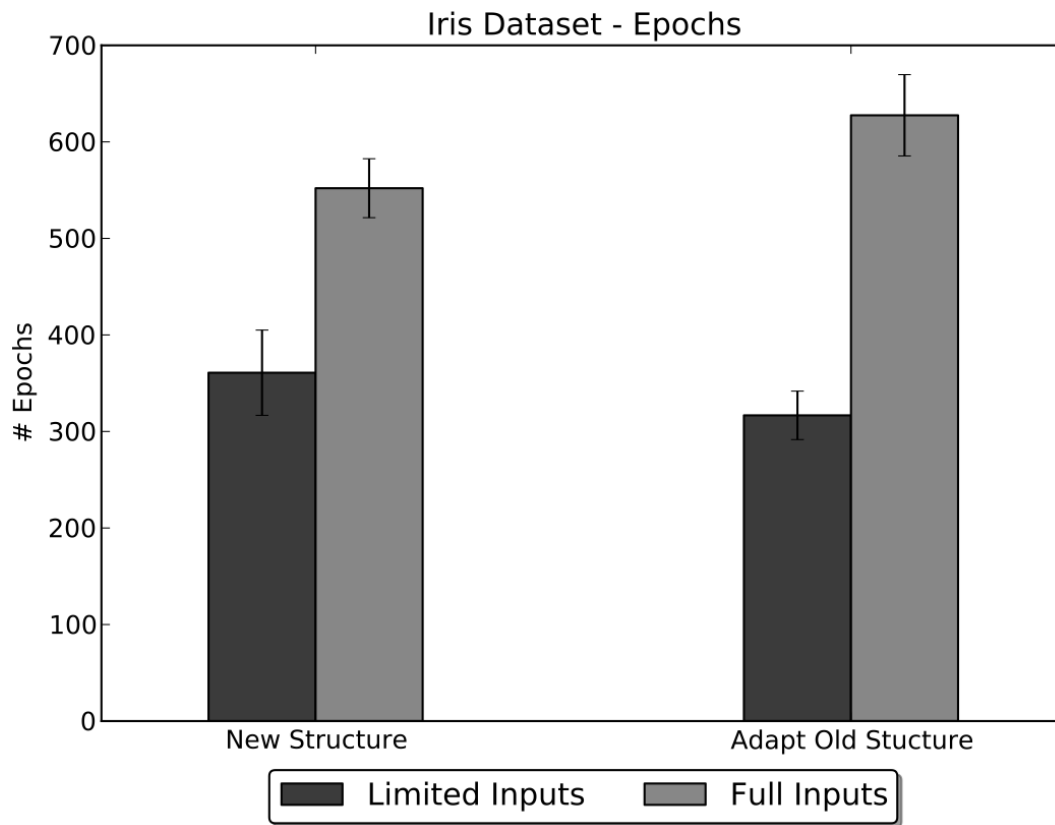


Figure 4.7: New Input Experiment: Number of epochs required to learn the added features. Both techniques are again very similar in the number of epochs required.

both improve performance equally well while requiring similar numbers of additional epochs and hidden nodes. The results from the linear data set suggest that when the network already has a good grasp of the problem, allowing the new inputs to adjust existing structure would require limited or no training after adaption. The benefit in epochs however is very small. In the case where multiple tasks are being learned and task recall is important it is possible that adapting the existing structure could harm the previous task recall by tampering with the hidden layer structure built to solve that task. The slight reduction in epochs seen in the Linear data set does not provide enough of a

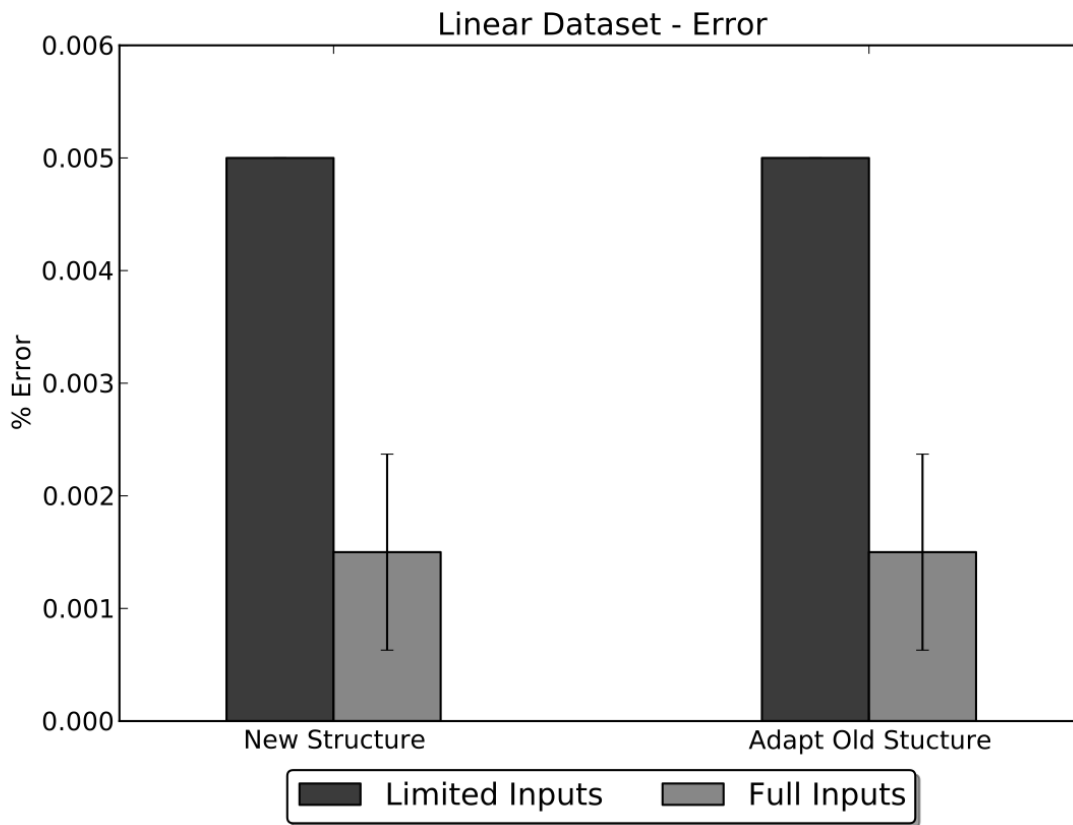


Figure 4.8: New Input Experiment: Classification error of the Linear data set using limited inputs and full inputs under two methods of adding new units. Both methods seem to provide the same level of improvement between the two feature sets. The performance from the limited set in the case is actually quite good. This is because the data set is almost separable using only the first input. This result shows the case of adding a new feature to an already decent classifier.

benefit to warrant the possibility of harming the previous task recall of the network. Considering how similarly both methods performed adapting new structure would be the preferred method.

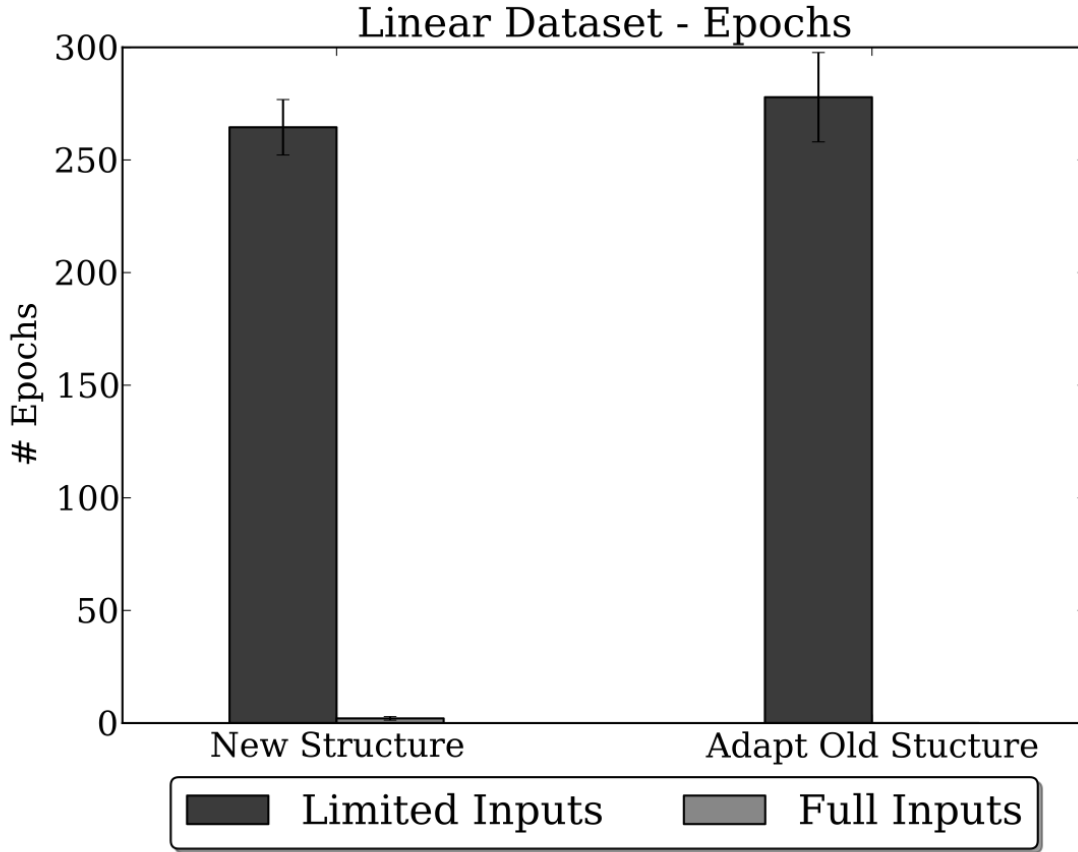


Figure 4.9: New Input Experiment: Number of epochs required to learn the added features. This shows the major difference between the two methods. In this set the first input was very good for the classification task. Adding the new input provided a modest boost to performance. In this case method 2 allowed the network to only adjust the existing structure to solve the problem requiring no additional hidden nodes.

4.3.2 Dual CC Recall Experiments

This set of experiments tests the reverberating dual network sequential learning system. This set of experiments is meant to compare the recall ability of our dual network system as described in section 3.2.3 with and without reverberation, and the single network case.

Method

The experiments are conducted by creating two training sets; Task 1 and Task 2. Each individual run begins by having a new network learn task 1, and then having the network learn Task 2. During training for task 2 we monitor the performance of the network on task 1. Training for task 1 is complete when an MSE of 0.05 is reached or if the MSE on the validation set stops improving. The MSE is calculated across both the normal outputs and the auto-associative outputs. In this way both sets of outputs are treated equally. The use of a validation set for early stopping differs from the work of Ans' where the first task training set is learned perfectly. We found in preliminary experiments with our real valued data sets, that using validation across both the normal and auto-associative outputs improved our secondary task performance without significant effects on the first task recall. The stopping criteria could be weighted differently for both sets of outputs, but that would lead to a multi-objective optimization problem.

The graphs presented in this section show the recall performance of the network during the training phase of task 2. The runs are repeated 30 times. In this experiment we are testing the effect of the number of reverberations on the recall performance. No Reverberations indicates that a single network is learning task 1 and task 2, while 1-6 reverberations indicates a dual network. The 1 reverberation case is special in that it denotes standard pseudo-rehearsal instead of reverberated pseudo-rehearsal. These experiments are run on our classification data sets as well as our observational data sets created from the users.

Dual CC Recall Results

Figure 4.10 shows the results for the band data set. The most obvious result is that the single network case loses first task recall very quickly as the second task is learned. The second thing to

note is that all of the pseudo-rehearsal cases keep the first task recall error at a low level. The single reverberation case performs the best, but the 2 and 4 reverberation levels are very close. The level with 6 reverberations however has the highest error and begins to degrade again toward the end of the training. This suggests that there is a level of reverberation that begins to hinder the training.

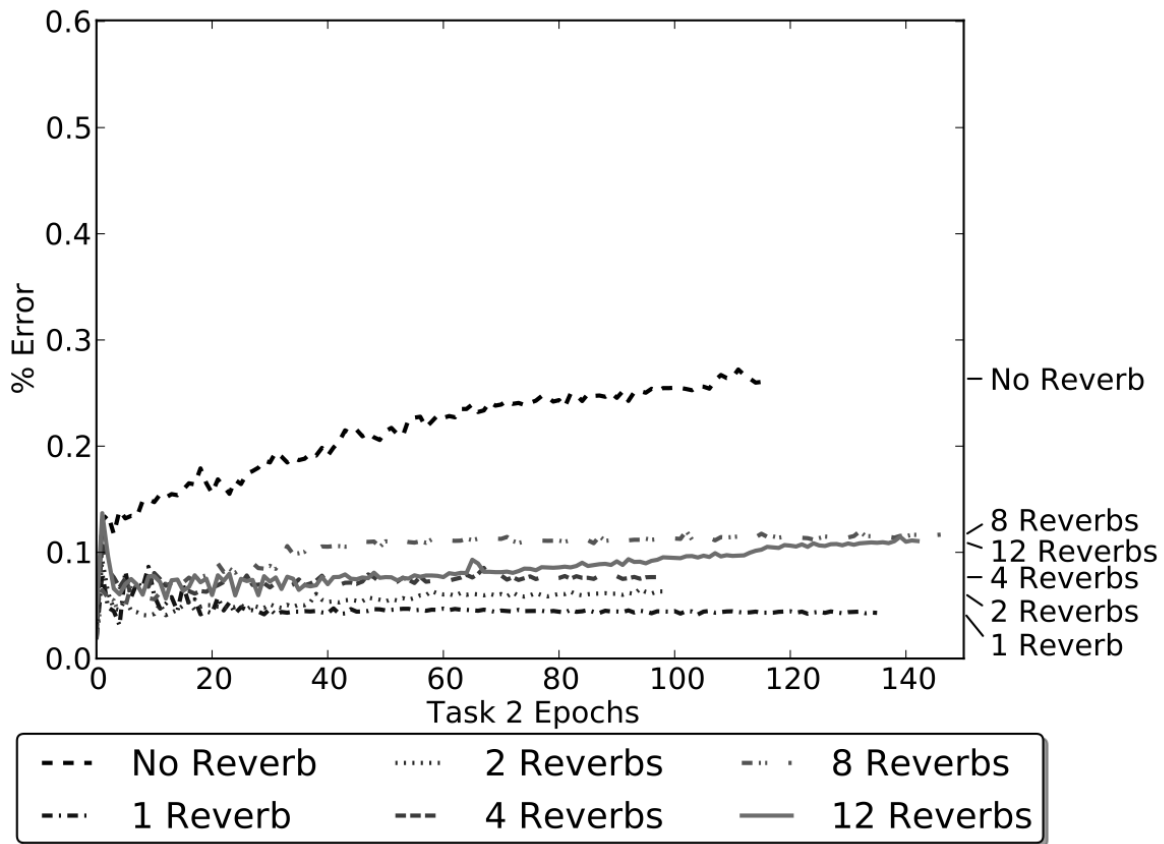


Figure 4.10: First task recall during second task training for the Band data set. The case with no reverberation is obviously losing performance, while pseudo-rehearsal, and reverberated pseudo-rehearsal keep the recall error low.

Figure 4.11 shows the results for the Circle in the Square data set. Again we see that the single network case loses recall very quickly and never recovers, and again the 1, 2, and 4 levels of

reverberation keep the recall error low and relatively the same. In this data set, however, the performance loss of the 6 reverberations case is more pronounced.

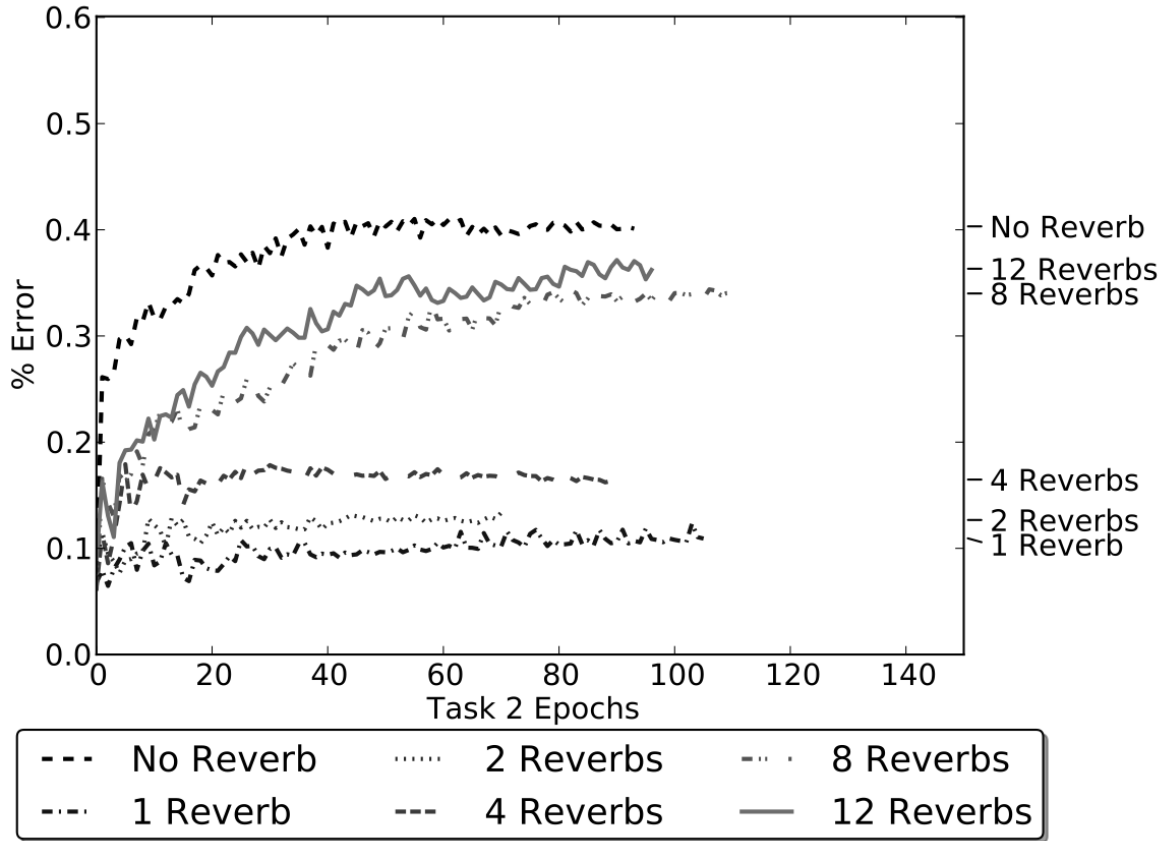


Figure 4.11: First task recall during second task training for the CirInSq data set. The case with no reverberation is obviously losing performance, while pseudo-rehearsal, and reverberated pseudo-rehearsal keep the recall error low. In this data set the 8 and 12 reverberations loss of performance is more pronounced.

Figure 4.12 shows the results for the Linear data set. The differences in performance for this data set between different levels of reverberation is much more pronounced. Again we see that the single network case loses performance very quickly, and that the 6 reverberation network also loses performance. Different from the previous two sets though the 4 reverberation network also

loses performance on par with the 6 reverberation network. Also interesting to note is that the standard pseudo-rehearsal and 2 reverberation networks, while achieving the lowest levels of recall error, also improve their recall performance over the course of the training. This differs from the previous two data sets where the performance remained level throughout most of the second task training, after an early period of adjustment. Again the single reverberation network achieves the best performance and does so at an earlier epoch than the 2 reverberation network.

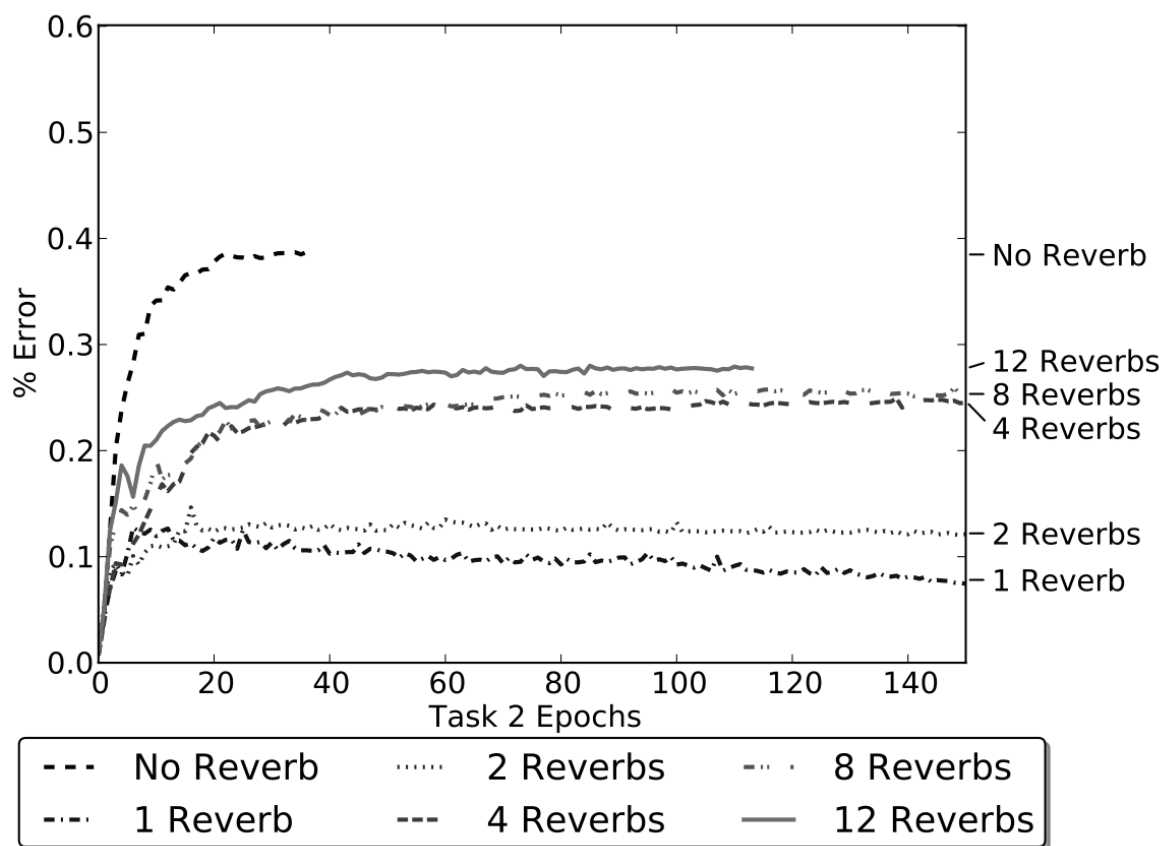


Figure 4.12: First task recall during second task training for the Linear data set. The case with no reverberation is obviously losing performance, while pseudo-rehearsal, and reverberated pseudo-rehearsal keep the recall error low. In this data set the 4, 8, and 12 reverberations loss of performance is more pronounced.

The relatedness between the first and second task differs between each of the three data sets. This is most clearly seen by the worst case performance of the no reverberation case. The band data set has an overlap between tasks of around 80%, meaning that task 1 and 2 share the same classification in 80% of the examples. The CirInSq data set has an overlap of 60%, and the linear data set has an overlap of 60%. The most obvious effect of the relatedness between the tasks is the performance spread between the no reverberations case and the best pseudo-rehearsal cases. On the Band data set the spread between the best and worst performance is much less than it is on the linear, and CirInSq data sets. This suggests that as the tasks become more unrelated the recall performance provided by pseudo-rehearsal becomes much more important.

The results from the classification tasks indicate that pseudo-rehearsal definitely increases first task recall while learning new tasks, however for these data sets reverberation doesn't seem to improve the recall performance over standard pseudo-rehearsal and for high numbers of reverberations it can be detrimental. Ans' experiments with reverberated pseudo-rehearsal focused on data-sets with binary inputs and outputs, and also required a high level of precision when learning the training sets, to the point of almost over-training. All of our data sets have real valued inputs, and while Ans postulated that reverberation could be transferred to real valued data, it seems to do so with some caveats. The likely explanation for the decreased performance as the number of reverberations increases, is due, perhaps to the way error propagates. The central idea behind reverberation is that reverberating the inputs will move the network to special network attractors; input patterns that have meaning to the network. All of the experiments in Ans' work [4] were performed using data with binary inputs and outputs representing binary encoded numbers. The examples in those tests had easily recognizable and meaningful combinations. Under those circumstances the important patterns are readily learned by the network as shown by Ans. When the examples have real valued inputs and outputs those patterns will be much harder to learn; stemming from the fact that the number of attractors could increase dramatically, and the differences between their representa-

tions could become very small. Additionally the requirement to learn the training data with high precision becomes more difficult as a result of the continuous examples. With all this in mind, it is likely that the reverberation amplifies any error that results from the networks difficulty in learning the more complicated examples resulting in increased error for higher levels of reverberation.

In the band and CirInSq data sets this didn't seem to happen until the number of reverberations reached 6, but in the linear data set it became an issue at 4 reverberations, and even 2 reverberations had a somewhat negative effect on the recall. Another important thing to keep in mind about these classification data sets is that they are artificially generated by applying some function to a random input vector, this means their input patterns are already pulled from a random distribution so in the course of generating pseudo-examples any random input pattern will be a valid input for these problems. Ans' reverberation, by moving randomly generated inputs to network attractors, will tend to generate pseudo-patterns that match the input pattern distribution of the original tasks, leading to pseudo-patterns that better represent the original data. This benefit of reverberation is lost on these randomly generated data sets

The next set of results come from running the same set of experiments on data collected from observing users running through the scenarios we described earlier. These data sets have several features that set them apart from our earlier data sets. One major difference is in the number of input features. Our classification data sets had two real valued inputs and a single binary output. These were increased to 4 inputs when the two contextual inputs are added for the tasks. The observational data sets have 21 real valued inputs describing the sensor responses seen by the users during the scenarios as well as the two contextual inputs for a total of 23 inputs. There are 9 outputs, since we are using the fanned output model discussed in 3.3.1, 2 of which are real valued and 7 of which are binary outputs. In the case of our dual reverberating networks this means that including the auto-associative outputs used for learning the input patterns, our networks for the observational data sets will end up with 23 inputs and 31 outputs, 9 regular outputs plus 23 auto-associative

outputs. Another feature of these data sets that is different from the classification data sets, is that the input patterns are not randomly generated. The inputs are recorded from the sensors in a simulated environment which means that the input patterns have meaning. The networks used to learn these observational data sets are much larger than the networks we've created in the previous experiments and the data sets are much more difficult to learn. For this experiment we will use data from our second and third users who used the joystick input device, and completed all three scenarios.

Figure 4.13 shows the results of training with user 2's observational data while figure 4.14 shows the results from user 3. In the experiment represented by these figures the first task is the 'Chase' scenario and the second task is the 'Chase-Obstacle' scenario. It is important to note that these two scenarios are extremely similar with respect to the user's behavior and the likely sensor inputs. The only differences come when the user approaches an obstacle, which is an intermittent phenomenon. For this reason these two tasks appear nearly identical to the training network. The effect of this can clearly be seen as the single network shows very little degradation of its first task recall even while learning the second task. This is in contrast to our earlier experiments with tasks that were different where the single network lost its first task recall very quickly while learning the second task. In both of these users' sets the single network retains the highest level of recall which suggests that if the tasks to be learned are very similar that the use of pseudo-rehearsal can have a detrimental effect on first task recall. An important thing to note from both figures is that in contrast to the classification tasks where it seemed that as the number of reverberations trended up so did recall error, the results from these two sets show that a reverberated network outperformed the standard pseudo-rehearsal in both cases. For user 2, four reverberations performed the best of all the pseudo rehearsals, while for user 3, two and four reverberations both perform better than standard pseudo-rehearsal. This could suggest that in these scenarios the reverberation process is helping the network generate slightly better pseudo-training points than simple random generation.

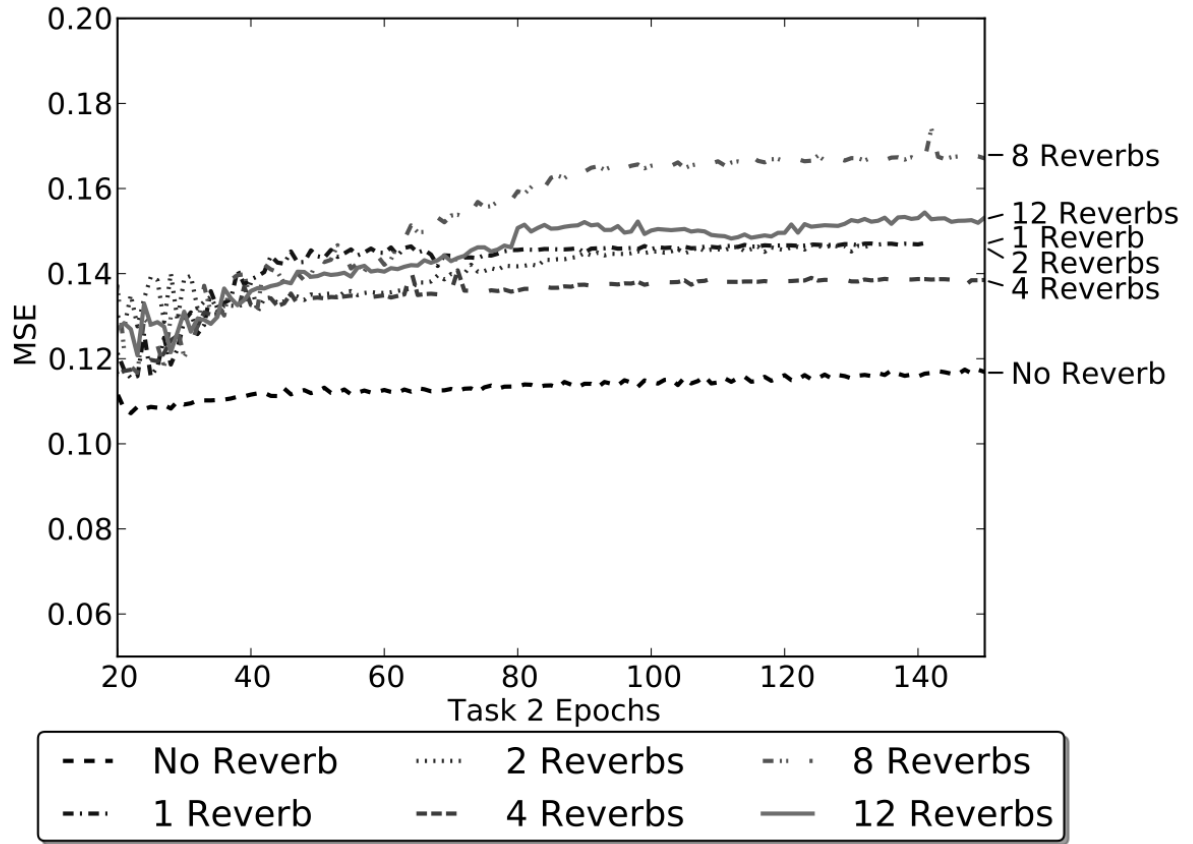


Figure 4.13: First task recall during second task training for the second user's data sets. The first task learned in this case is the 'Chase' scenario, the second is the 'Chase-Obstacle' scenario. The user behaviors in both of these scenarios is nearly identical with the only major difference being the presence of obstacles.

Experiments were also conducted while learning the 'Chase' and 'Obstacles' scenarios in sequence. These two scenarios are much different from each other in the behavior of the user participants. While the sensor and action inputs are the same, the users are either chasing a moving target in 'Chase', or maneuvering around an obstacle strewn arena in 'Obstacles'. Training these two scenarios in sequence is similar to our earlier classification experiments where the same input

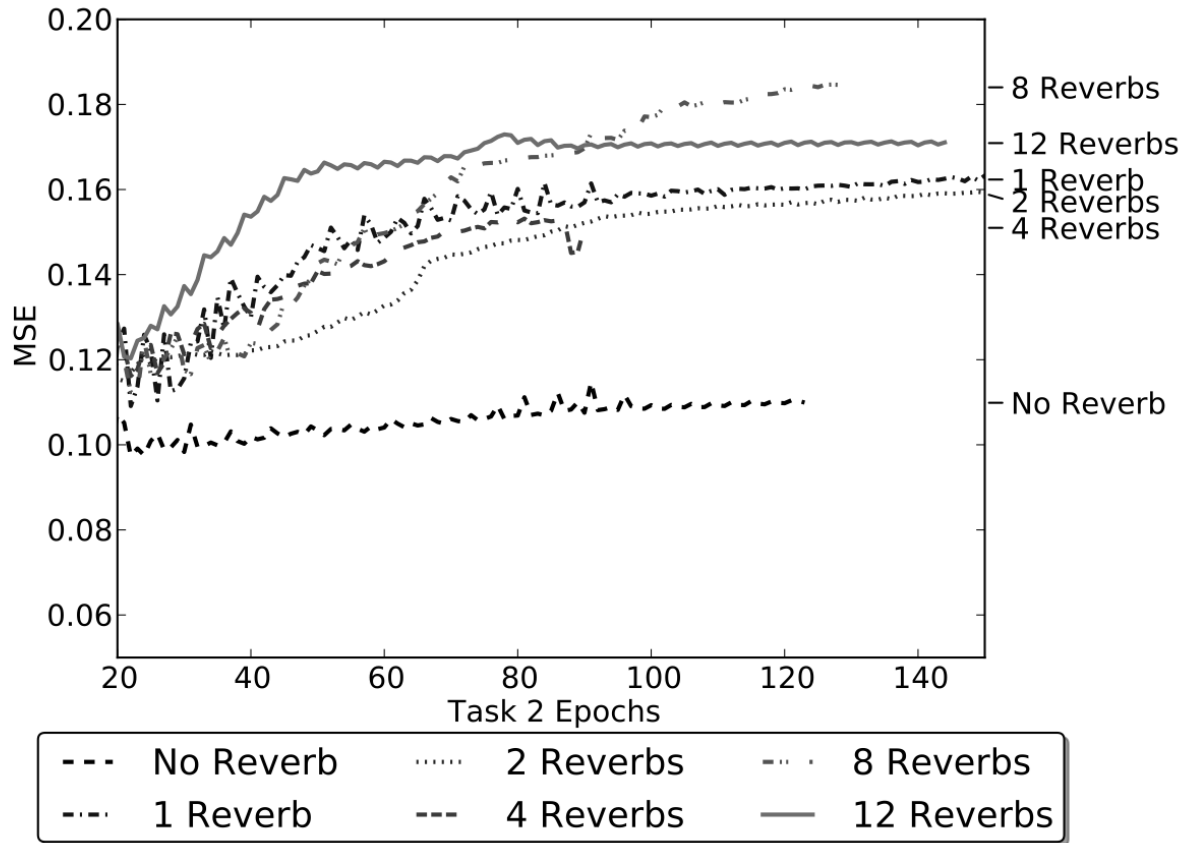


Figure 4.14: First task recall during second task training for the third user’s data sets. The first task learned in this case is the ‘Chase’ scenario, the second is the ‘Chase-Obstacle’ scenario. The user behaviors in both of these scenarios is nearly identical with the only major difference being the presence of obstacles.

pattern in one task could lead to a different correct action. Figures 4.15 and 4.16 show the results of first task recall when learning ‘Chase’ and ‘Obstacles’ in sequence. In these two cases the single network case is no longer the best performer; falling to last place for user 2, and 2nd / 3rd place for user 3. This reinforces our perception that pseudo-rehearsal is more beneficial when the tasks to be learned are less similar. These results also show a reverberated network achieving the best performance in both cases with 2 reverberations, while the standard pseudo-rehearsal finishes last

for both users. In fact the tests with user 2 showed all levels of reverberation remaining fairly close together. This could be the result of user 2's high level of consistency, implying that his reactions to similar stimuli were consistent leading to observational data with more structured inputs.

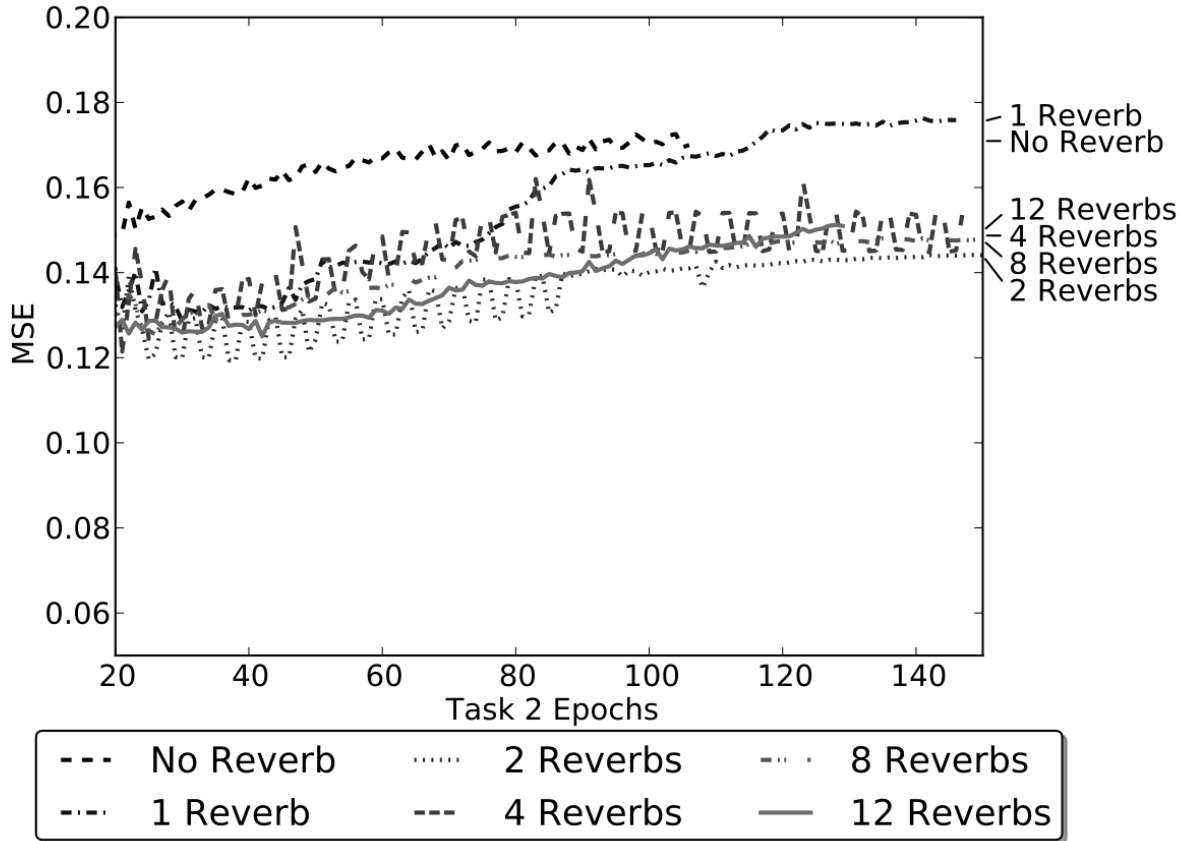


Figure 4.15: First task recall during second task training for the second user's data sets. The first task learned in this case is the 'Chase' scenario, the second is the 'Obstacle' scenario. The user's behavior in these two scenarios is quite different.

The experiments conducted in this section are focused on the recall of previous task knowledge while learning a new task. They compared the results of a single network, standard pseudo-rehearsal dual networks, and reverberated dual networks. The final results do not imply that there

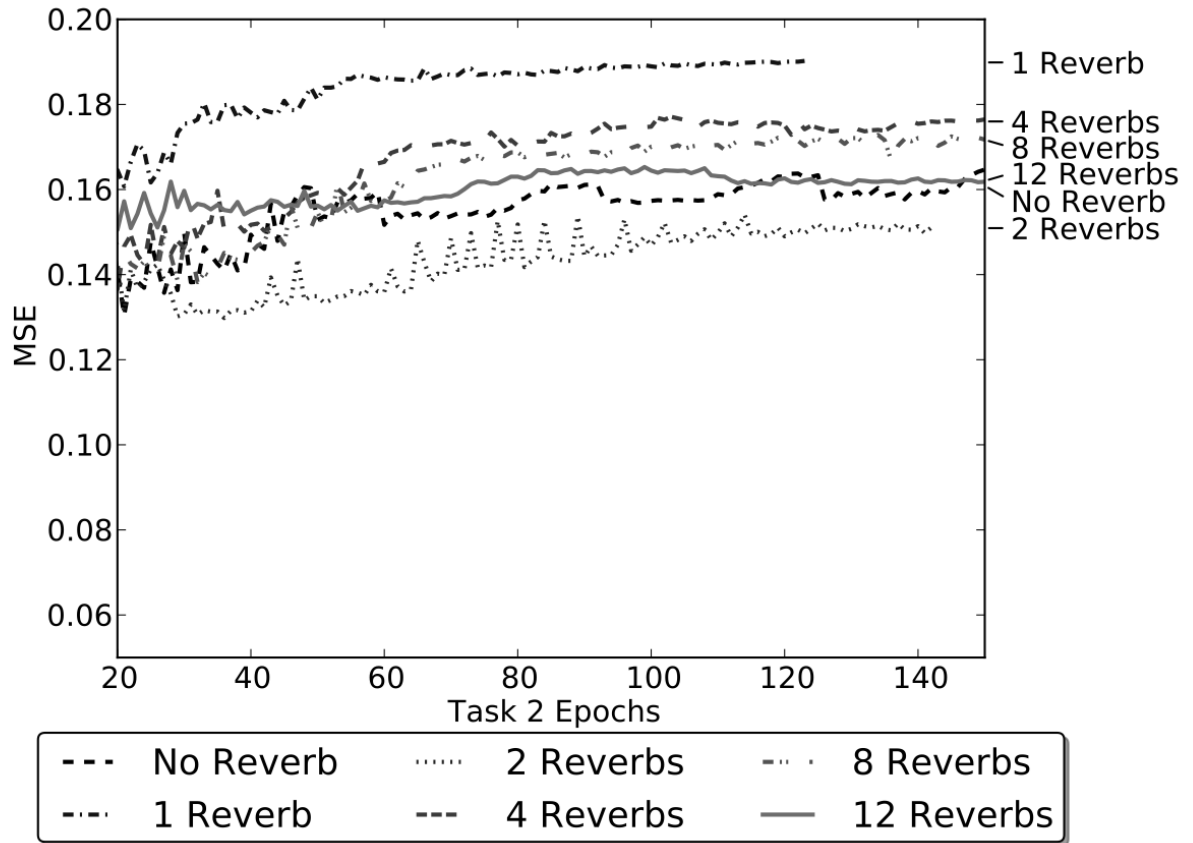


Figure 4.16: First task recall during second task training for the third user’s data sets. The first task learned in this case is the ‘Chase’ scenario, the second is the ‘Obstacle’ scenario. The user’s behavior in these two scenarios is quite different.

is a clear choice in every situation. The results from the classification data set tend to show that too much reverberation can begin to hinder the recall, in some cases almost as much as no pseudo-rehearsal at all. While, the observational sets seem to show that reverberated pseudo-rehearsal with a small number of reverberations performs better than standard pseudo-rehearsal, and for user 2 even larger numbers of reverberation performed well. These two findings seem to suggest that the performance of reverberation and pseudo-rehearsal rely somewhat on the dynamics of the input features. With reverberation performing better with structured inputs, and standard

pseudo-rehearsal performing better with random or noisy inputs. The second major finding is that pseudo-rehearsal can be detrimental to recall performance when the sequential tasks are very similar. This was shown when the ‘Chase’ and ‘Chase-Obstacles’ scenarios were trained in sequence.

4.3.3 Dual CC Consolidated Training Experiments

The last experiment focused on the recall performance of our system when learning two tasks in sequence. This will test the ability of our system to do consolidated training. Consolidated training is the continuation of learning of an existing network as new data is acquired. This is a useful skill for our life-long learning system to have as it will allow us to learn new skills over time and to polish existing skills as new data becomes available. This is particularly useful in our observational learning application where data is collected from users in real time. Having the ability to continue training as the users provide more runs allows us to break user training into segments and avoid user fatigue.

Method

The experiments in this section are performed by taking four data sets and treating the first three as training and the fourth as testing. Each of the classification data sets is divided into three smaller training sets and a single test set. The number of training points for each of the smaller training sets was chosen from our impoverished training point parameter as found in 4.2.1, in order to show how the network responds to training data sets that are individually too small to learn the task well. For the observational data sets we take the first four runs from each user in the ‘Chase’ scenario and divide them up into three training sets and one testing set. We also choose the number of training points for the observational data sets to simulate accumulating small data sets.

For each data set the networks are trained sequentially on the first three training sets and their performance on the test set is calculated after each run. The runs are repeated 15 times with the average classification error reported for each result. The error on the test set is reported after each successive training set in order.

Dual CC Consolidated Training Results

The results again reflect the use of a single network, pseudo-rehearsal dual network, and several reverberated dual networks for each data set. In the figures below the value of -1 for the number of reverberations indicates the single network, 1 indicates the standard pseudo-rehearsal, and 2+ indicate reverberated pseudo-rehearsal. Each figure will depict the test scores of the network on the fourth data set after training each of the first three data sets sequentially.

Figure 4.17 shows the results on the linear data set task 1. Each training set had 5 training points to simulate having to deal with small amounts of data over time. The results from this data set show that the reverberated networks with 2 and 4 reverberations perform the best and improve their performance on the test set after training with each of the first 3 training sets. The single network and standard pseudo-rehearsal network do not improve after each training session and in fact the single network gets significantly worse after the third training set is learned. From these results it seems as if the reverberated networks are keeping a better representation of the previous learning sets and building a better representation of the problem with each successive training phase. Each of the three training sets on their own do not provide a good model of the problem as can be seen from the results from the single network. This an ideal case showing the benefits of consolidated training.

Figure 4.18 shows the results for the Circle in the Square data set. It shows a similar trend to the results from the linear data set. The single network is not able to take advantage of the consolidated

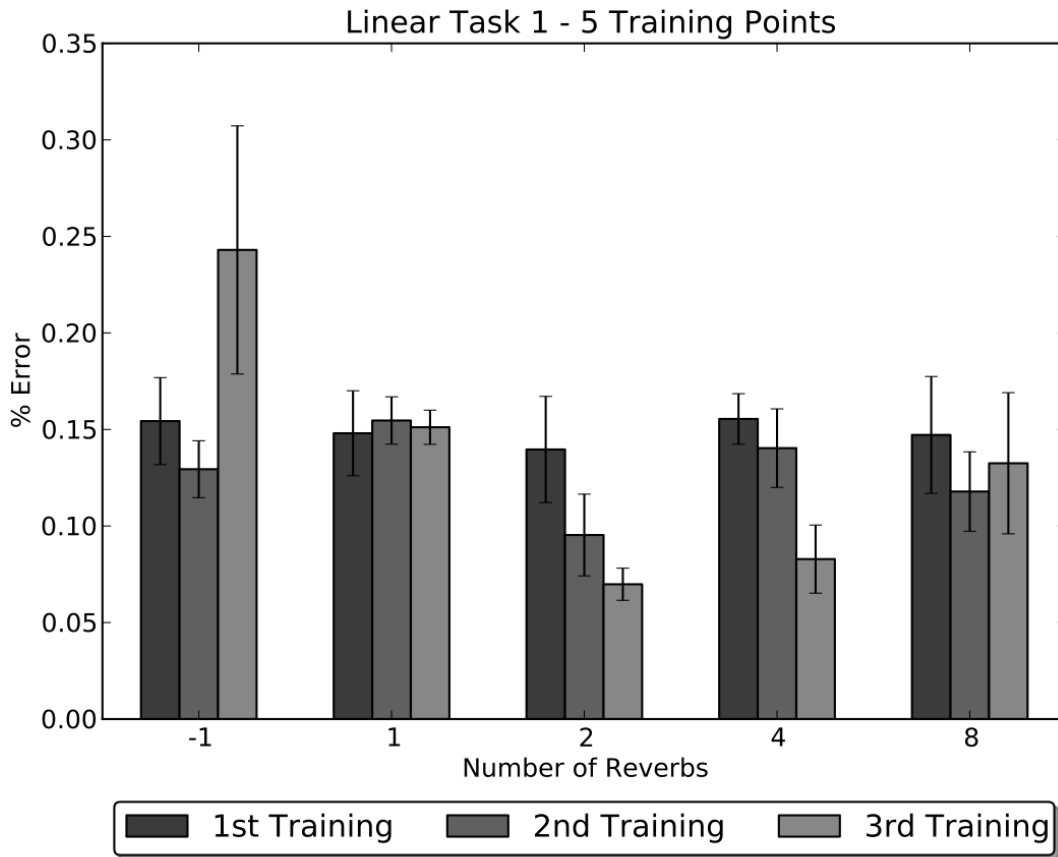


Figure 4.17: Results of consolidated training for the Linear data set task 1. The network using 2 reverberations performs the best with 4 reverberations achieving similar results. The single network, and standard pseudo-rehearsal network don't seem to benefit from the consolidated training.

training while the networks using pseudo-rehearsal are. In this case the standard pseudo-rehearsal is able to improve its performance after each phase though it doesn't do quite as well as the reverberated pseudo-rehearsal using 2 reverberations. While the pseudo-rehearsal case doesn't reach the same level of performance, it does exhibit much smaller confidence intervals implying more consistent performance. It is important to remember at this point that these results are averaged over 30 runs, meaning 30 different networks with random starting weights. The higher levels of variance in the reverberated networks, and the apparent increase in variance as the number of re-

reverberations go up suggest that the higher levels of reverberation increase the variability of the possible final networks. This suggests that if the final goal is maximum performance it would be worthwhile to run repeated trials and select the best performer. Figure 4.19 shows the effect of using reverberation on the number of epochs during training. As was mentioned previously the process of consolidating knowledge while learning new tasks would slow the training phase, and these results show that it is not an insignificant effect. The single task network uses slightly fewer epochs after each training phase, while all of the networks using pseudo-rehearsal take many more epochs after the first training phase. This shows that while pseudo-rehearsal improves the performance on the final task it comes at the cost of longer training times.

The results for the observational data sets are not as clear cut as the ones from the classification sets. The results from the user data sets are from networks using the 4 real valued outputs and reported in the mean squared error on the final test set. We use this configuration to see how closely the networks can predict the user's actions after training. Figure 4.20 shows the results from user 1. User 1 was the subject using keyboard input so his output actions will resemble binary outputs as each key press sends the maximum signal to the simulator. The binary nature of these outputs probably indicate why the results look similar to the classification results, in that pseudo-rehearsal and 2 reverberation networks both see improvements after each training phase while the single network and higher levels of reverberation seem to lose some performance after the final training phase. The results are somewhat different for user 2 and 3 as can be seen in figures 4.21 & 4.22. For both of these users the standard pseudo-rehearsal sees improvement through consolidated training but the reverberated pseudo-rehearsal does not. In the case of user 2 the reverberated networks actually lose performance after the second training phase though gaining most of it back in the third, while with user 3 the performance is mostly stagnated but with an increase in variability.

These results suggest that the dual network pseudo-rehearsal is able to perform consolidated learning across each of the data types. The utility of reverberation during consolidation, however, seems

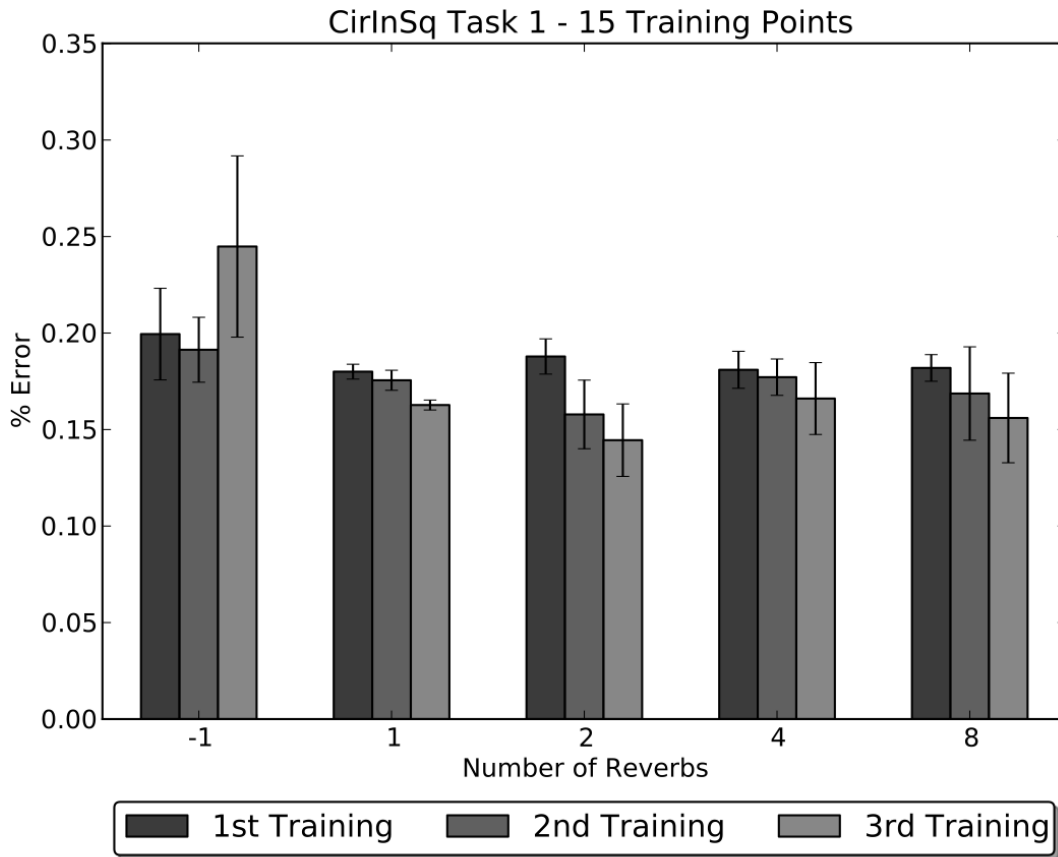


Figure 4.18: Results for consolidated training for the Circle in the Square data set task 1. The single network example denoted by the -1 column is not able to improve its performance with successive training tasks, while the standard pseudo-rehearsal and 2 reverberation networks improve after each training.

to be dependent on the type of problem being learned. From the results of our tests so far it appears as though the use of reverberation is more beneficial in improving generalization of the networks when the outputs are binary in nature, while not helping or actually hindering generalization when the outputs are real in nature.

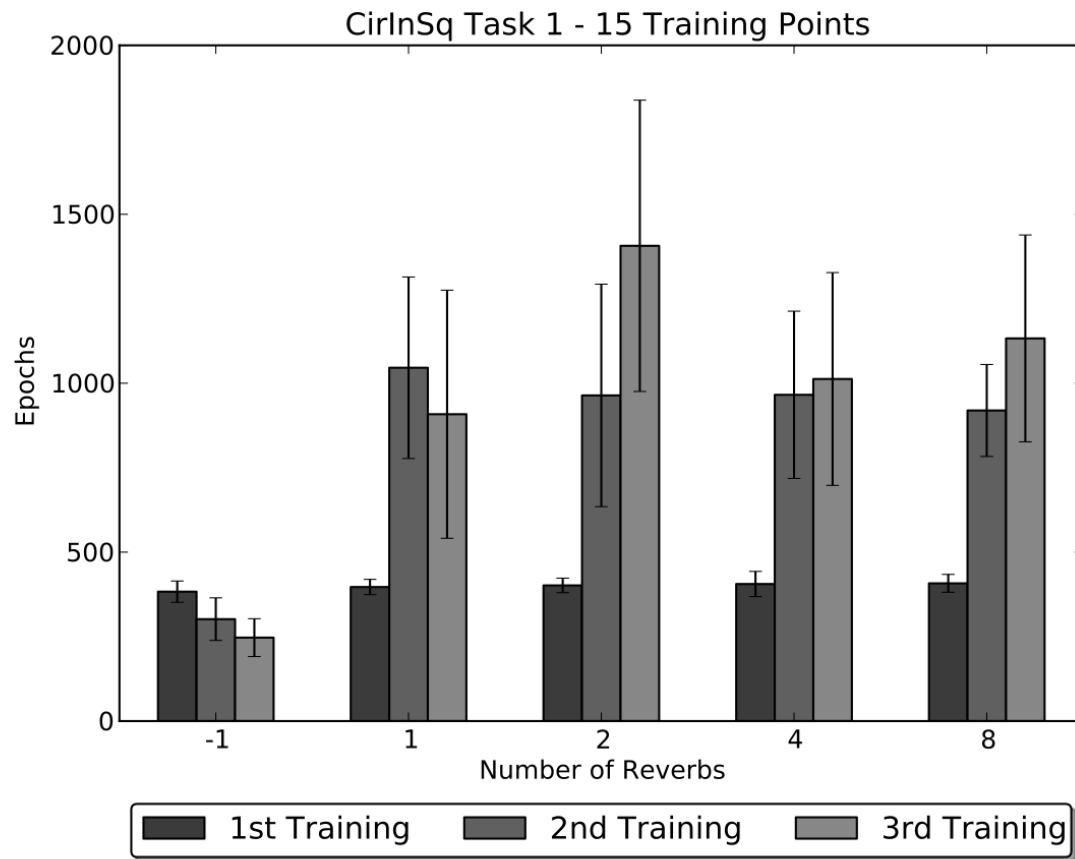


Figure 4.19: The number of epochs used during each training phase for the Circle in the Square data set. The increase in epochs during the pseudo-rehearsal and reverberated pseudo-rehearsal is readily apparent. This shows that while the performance is improving it comes at a cost of longer training times.

4.4 Generating Simulated Agents

The previous sections have used the observational data sets to train neural networks, but so far we have only reported classification scores and error rates. The more interesting results come from training the agents and utilizing them as participants in the simulation. In this section we'll report the results from generating agents from the user observational data and running them in the

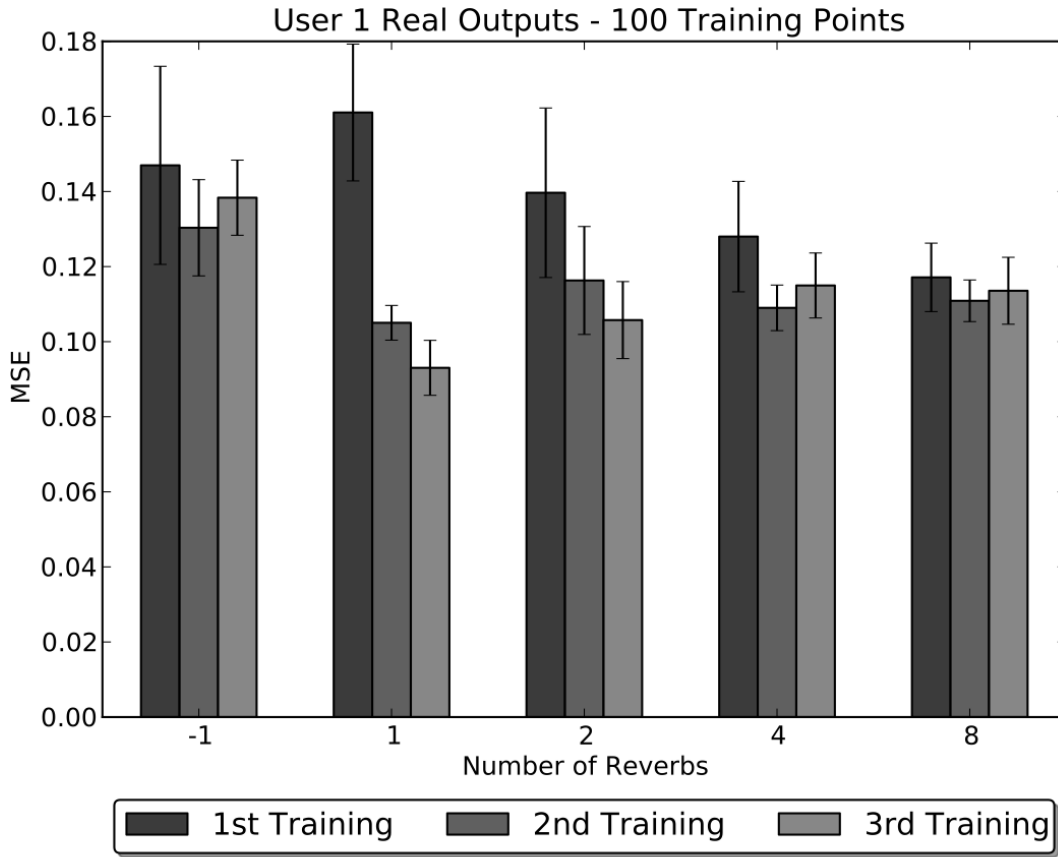


Figure 4.20: Consolidation training results for user 1. These results look similar to what we saw with the classification data sets. The pseudo-rehearsal and 2 reverberation networks both see improvement after each training phase while the single network and higher levels of reverberation get initial improvements then lose some performance after the final training set.

simulation.

4.4.1 Method

During the data collection phase each user sat down and ran through each of our scenarios for 30 seconds at a time. Each user participated in 10 episodes of each scenario. During their sessions

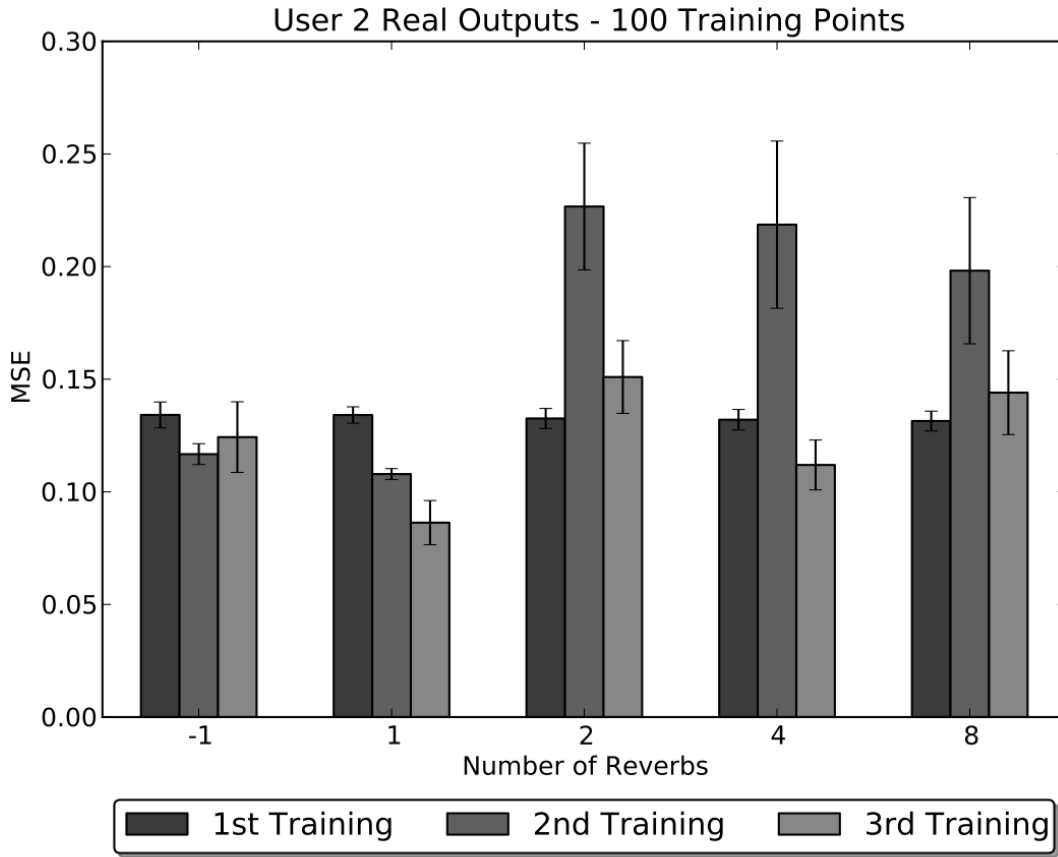


Figure 4.21: Consolidation training results for user 2. These results show a different picture than the results for user 1. Like user 1 the standard pseudo-rehearsal shows an improvement in performance after each training phase, but unlike user 1, each of the reverberated networks loses performance after training phase 2 then gains some of the performance back after training phase 3.

the sensors and actions from their controlled agents were recorded and collected as training data. These sets have been used in the previous sections already. We also collected history information about each entities position in the simulation. The episode history data allows us to go back and replay each users episodes in the simulation. For each individual training episode we used the Reverberated Dual CC algorithm to train 15 networks. The networks are sorted based on their classification score and the best network for each episode is chosen. In addition to these individual

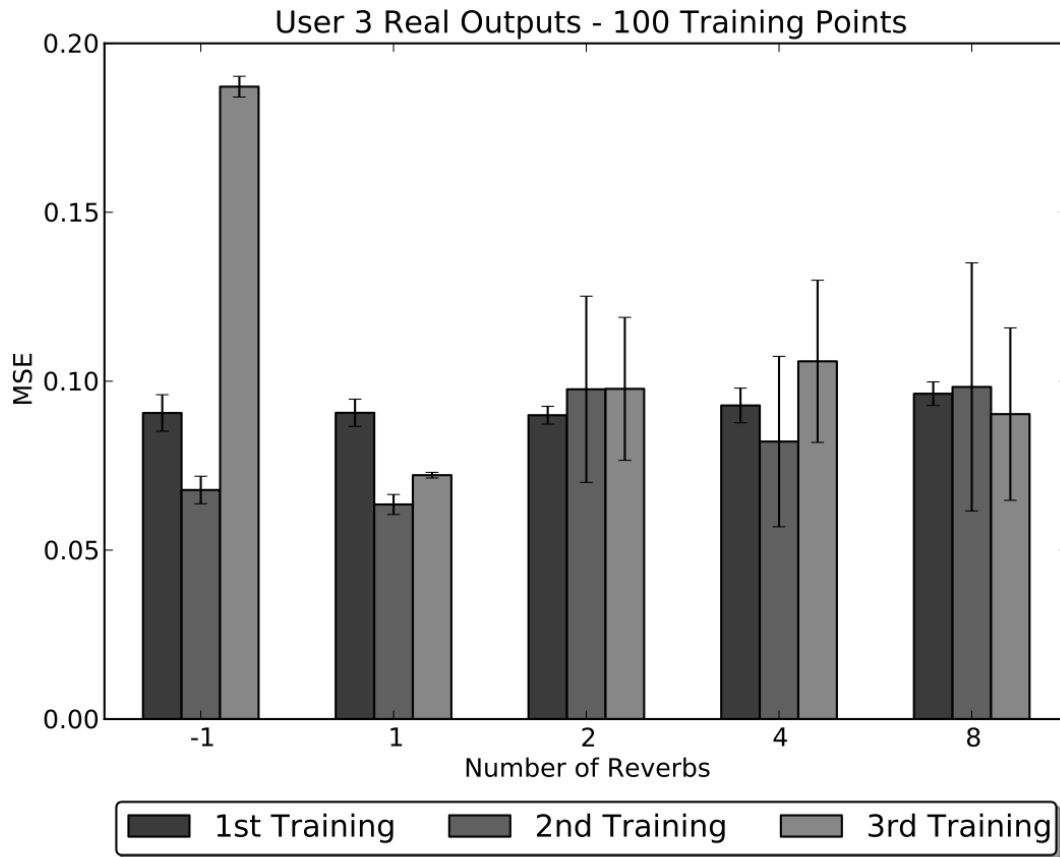


Figure 4.22: Consolidation training results for user 3. These results show a similar result to user 2. The standard pseudo-rehearsal performs the best, though it has a slight loss after the third phase, while the reverberated networks seem stagnate after each phase with a large spike in their variability. The single network gets a performance increase after the second training phase but loses it all and more after the third.

episode agents, we also train an agent that starts with the best episode 1 agent and then does consolidated training throughout the rest of the scenario. This generated a single agent that has trained on the entire scenarios data.

4.4.2 Simulated Results

These agents are then put through the simulation and their scores are calculated in the same way as they are for the original users. Each agent runs through the simulation 5 times with random starting points.

Table 4.3: User 1 Agent simulation scores. The average scores for the consolidated agent, individual episode agents, and the user are shown for each scenario

	Chase	Chase-Obstacles	Chase-Shoot		Chase-Obstacles-Shoot	
Consol	24.59	11.75	25.18	26	0	0
Avg Episode Agent	23.032	20.42	18.76	17.88	10.77	5.06
User Avg	12.04	16.27	22.06	633.6	22.53	643

Table 4.4: User 2 Agent simulation scores. The average scores for the consolidated agent, individual episode agents, and the user are shown for each scenario

	Chase	Chase-Obstacles	Chase-Shoot		Chase-Obstacles-Shoot	
Consol	15.29	1.34	22.12	0	2.5	91.6
Avg Episode Agent	16.01	9.54	15.79	2.82	9.73	16.32
User Avg	23.53	12.96	21.34	188.2	18.66	618.22

Table 4.5: User 3 Agent simulation scores. The average scores for the consolidated agent, individual episode agents, and the user are shown for each scenario

	Chase	Chase-Obstacles	Chase-Shoot		Chase-Obstacles-Shoot	
Consol	19.28	19.56	0	0	22.97	21.39
Avg Episode Agent	21.14	19.96	20.64	3.14	20.12	28.28
User Avg	10.41	15.04	23.09	79.5	23.32	220.5

Table 4.3 shows the simulation results for the agents created from user 1. There are a few interesting things to note. First for the Chase and Chase-Shoot scenarios the consolidated agent earns

a higher average score than the user does in the same scenario. This is reversed however in the Chase-Obstacles, and Chase-Obstacles-Shoot scenarios, with the extreme case of the Consolidated agent earning zero score for the C-O-S scenario. For the scenarios where the agents out perform the users, visual inspection of the agents running show that they tend to learn a good chase behavior and then chase the runner in circles. This is very hard for the user to do because their inputs can't be perfect, however the agents will output the same reaction for the same input parameters so if they get the runner in the right position they can achieve a high score. In the situations where the agents do poorly it tends to be the same problem in reverse. The runner gets in a spot they don't know how to respond to and they get stuck running the wrong direction. In fact it seems that in the chase scenario specifically the performance of the users and their agents is reversed. Users 1 and 3 (Table 4.5) performed more poorly, but a visual inspection of their replays show that they had long sections where they searched out the agent. These episodes provided lots of information on how to respond to the runner in different situations. User 2 (Table 4.4), however performed very well on the chase scenario with his agents performing slightly worse on average. Inspecting User 2's replays it shows the user very quickly acquiring the runner and sticking to his tail for nearly the entire episode. This behavior, while optimal from the score stand point, is less optimal from an observational training stand point.

The situations where the consolidated agents for User 1 and User 3 have zero score a visual inspection of the agents shows both agents tend to turn away from the runner and get stuck. This shows that while in most of the situations the consolidation technique performed well it is possible to corrupt the agents with bad data. This shows that selecting which episodes are used for training purposes can have an important impact on the resulting agent.

4.4.3 Subjective Results For Generated Agents

While reporting the simulation scores of the agents provides valuable information, this does not show the whole picture. The subjective value of the generated agents is also of interest. To analyze the subjective value of the agents user 1 and user 3 were asked to view a random selection of replays or agents. They were then asked to guess whether the episode they had watched was themselves, or an agent based on them. This is a highly simplified form of the Turing test, and while the ultimate goal is not necessarily human like agents, their similarity to their human counter parts is of interest from the observational learning stand point.

Method

Each user was shown 15 episodes with a 50% chance of being either human or agent. At the end of each episode they marked their decision. They were also asked to comment if an agent did particularly good or bad. After they had finished they were asked to judge their satisfaction with the resulting agents.

Results

Table 4.6: User 3 Subjective results. For all of the scenarios except Obstacles. The number of agents that passed for human and the total number of agents is reported.

	Chase	Chase Obstacle	Chase Shoot	Chase Obstacles Shoot
Passed as Human	4	7	0	5
# Agents	13	9	6	9

Table 4.6 shows the tally for the subjective tests for User 3. User 3's agents generally performed

well and he had a hard time differentiating himself from his agents. During the Chase Obstacle scenario they actually incorrectly identified two of their replays as agents. User 3 commented that for most of the scenarios the only way to tell was to catch the agent in an obvious bug or a flaw in the simulator (getting stuck to the runner). The Chase-Obstacles scenario was the obvious winner, with the chase obstacles shoot scenario close behind. In both of these scenarios the user remarked that it was much harder to tell as the obstacles kept the agent from trapping the runner in a loop. The stand out here is the chase shoot scenario, the agent in question was the agent that scored 0 in the previous section. This agent would run toward the agent until a certain distance was reached, and then it would turn its back and stand their. It did this consistently even on repeated starts from random locations. Overall User 3 rated the agents a 4 out of 5 in satisfaction, and a 4 out 5 in the agents behaving like he would.

Table 4.7: Subjective Results for User 1. For all of the scenarios except Obstacles. The number of agents that passed for human and the total number of agents is reported.

	Chase	Chase Obstacle	Chase Shoot	Chase Obstacles Shoot
Passed as Human	1	1	0	0
# Agents	6	11	7	11

Table 4.7 shows the subjective results for User 1. User 1 had a very different opinion and experience with the subjective tests. Early on the user picked out quirks in his own replays and used this to help determine which replays were agents or human. Two agents were able to slip by this user; one each in the chase and chase obstacles scenarios. The chase shoot scenario was easy to determine as User 1 was extremely good at the shooting task. The chase obstacles shoot scenario however the agent had a similar quirk as the chase shoot agent for User 3. It would turn away and get stuck. User 1 had several comments regarding the performance of the agents in the chase and chase obstacles. In general it was easy to determine the difference between user 1 and his agents because they maintained a much closer distance than he was able to maintain during his runs. He

also noticed that he tended to have a swerve in his turns that the agent ignored in favor of keeping very close to the runner. The chase obstacle scenario performed slightly better in his estimation with the choices becoming slightly harder even though he chose nearly perfectly. Overall the user was somewhat disappointed that the agent didn't pick up more of his quirks, and was not able to acquire his tremendous shooting skill. He rated the agent's ability to perform like him at 1 out of 5, but was more lenient and rated his satisfaction at 3 out of 5.

The subjective results for the agents is overall mixed. Each user is different and performed differently during each scenario, and each users utility as a training source differs as well, and not always in a way that's obvious from objective measures of performance. The consolidated agents performed well in most of the scenarios achieving good scores, though usually at the expense of having obvious none human abilities like forcing the runner into a circle. What is interesting is that the performance of the agent is not always directly tied to the performance of the human trainer. In fact the best performing agents in the chase scenario came from the users with the lowest score. The defining characteristic of their runs being somewhat wild behavior; thus providing the agent with a wide assortment of sensor information.

4.5 Merging with Reverberated KBCC

This section details experiments with the reverberated KBCC algorithm in merging existing networks to create a combined network. The description of our merging process is in Section 3.2.4.

4.5.1 *Method*

The first experiment in this section is a preliminary experiment testing the KBCC implementation on a band cross data set The band cross data set is a generated data set similar to the band data set It

is a collection of 3 similarly sized boxes placed at different locations with two of the boxes crossing. The point of this data set is to provide a mechanism to showcase the effect of using previously trained networks in training. The first experiment tests the situation of using a previously trained network that has learned the exact task you are currently learning, while the second shows using two previously trained networks to learn a combined task.

The rest of the experiments in this section test the merge process. This is the situation in which only the previously trained networks are available. The merge experiments are carried out by first generating single networks that have learned the two previous tasks that are to be learned. For each task 15 networks are created and the best scoring networks are used to generate a pseudo-item data set that is used for training a new Reverberated KBCC network on the combined task. The network then learns the combined task using the generated training set and the existing networks as sub-networks. These experiments are completed with the number of reverberations varying to show the effect of the number of reverberations on the merging performance.

4.5.2 Merging Results

This first set of graphs show the KBCC algorithm learning one of the three individual classes while using a previously trained network on the same task as a sub-network. Figure 4.23 shows the error rate and epochs for learning each individual task. Except for outBox3 we see that the KBCC algorithm achieves similar performance to learning the task in isolation. Outbox 3 suffers some loss in performance compared to learning the task in isolation. The number of epochs required to learn the problems using the sub-network is significantly reduced for outBox1 and outBox3 while maintaining a similar number of epochs in outBox2, though the variance between training runs is significantly less in the KBCC case.

The next set of results shown in Figure 4.24 show the performance of learning a combined data set

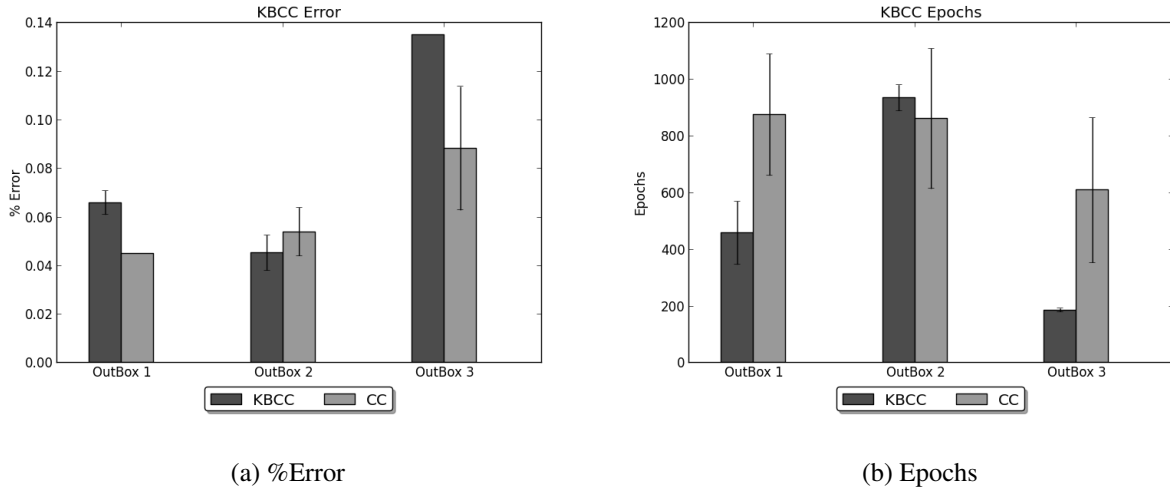


Figure 4.23: The classification error rate and the number of epochs to learn a single task using a previously trained network as a sub-network. The CC algorithm is used as comparison for learning the task without the sub-network.

of two tasks using networks previously trained on the sub tasks. In this more complicated problem we again see that the classification performance remains similar, and that we see a small drop in the number of epochs required in both cases. However, the difference in epochs is not as significant in this case.

Figure 4.25 shows the first results while using our merging technique. Two previously trained networks are used to generate a training set, and then are used as sub-networks during the training process. The results depicted are the error rate on the actual task test set, and the epochs needed to learn the simulated data set. The graph shows the merging process using a varying number of reverberations in the generation of the pseudo set. We saw in section 4.3 that the number of reverberations has a varied effect on training depending on the goal and the type of data being learned. We saw that for data sets with random inputs the reverberation could have a negative impact on performance. We see a similar effect during the merging process here. Merging using a single reverberation provides good classification performance on the actual combined task, but as the

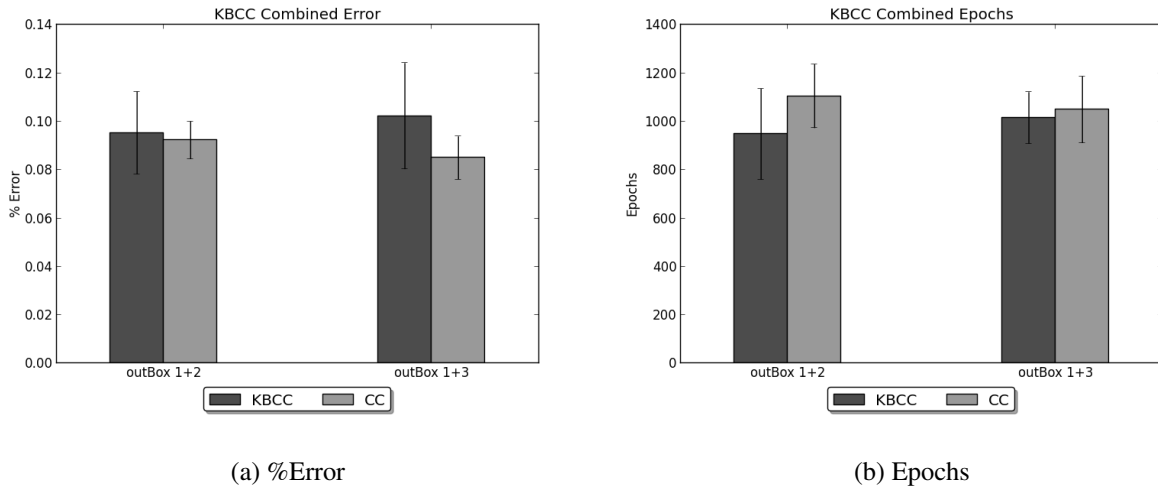


Figure 4.24: The classification error rate and the epochs to learn a combined problem utilizing two previously trained networks. The CC network is used to show the comparison to learning the new task without using sub-networks

number of reverberations increase the performance worsens, with the 2 reverberation case giving the worst performance. The 4 through 8 reverberation cases are odd in that their performance seems to improve, but their training epochs decrease significantly. This is likely because the reverberated data set ended up in a degenerate state with all the class labels the same.

Figure 4.26 shows the results of merging Task 1 and Task 5 of the band data set. These are the two band tasks that share the least similarity with each other. The results again show that the single reverberation case provides the best results for data sets with random input vectors. The performance on the combined task test set is very good especially since the two tasks are unlike.

Figure 4.27 shows the results of merging Task 1 and Task 5 of the CirInSq data set. These two tasks are circles of equal size but translated to opposite corners. We see the same effects here as in the other data sets. The merging process creates a network that performs well on the combined training set, when using a single pass through to create the artificial training set. The performance of the

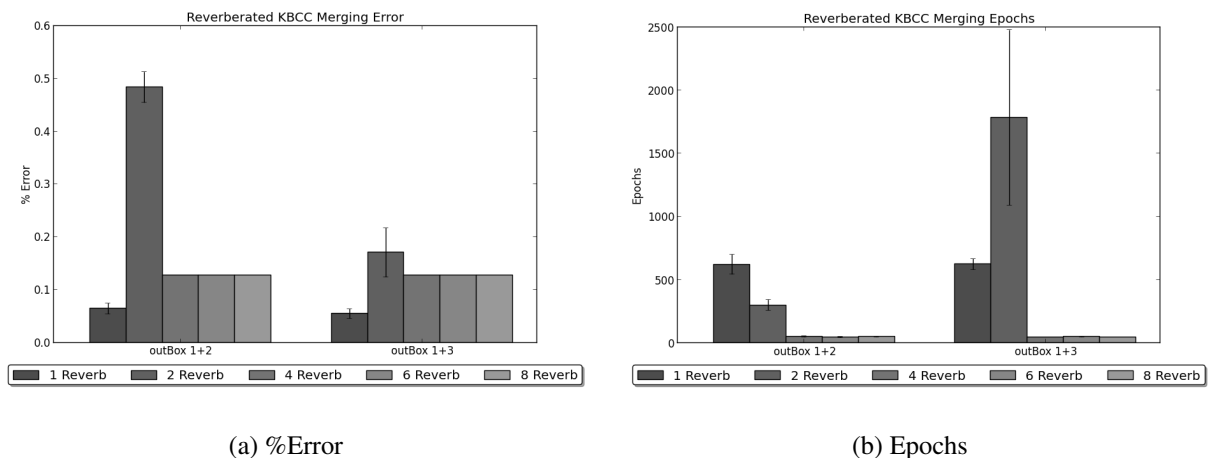


Figure 4.25: The classification error rate and the epochs to learn a merged task using only previously trained networks. The training data is generated by reverberating the sub-networks. These figures depict the out Box data set

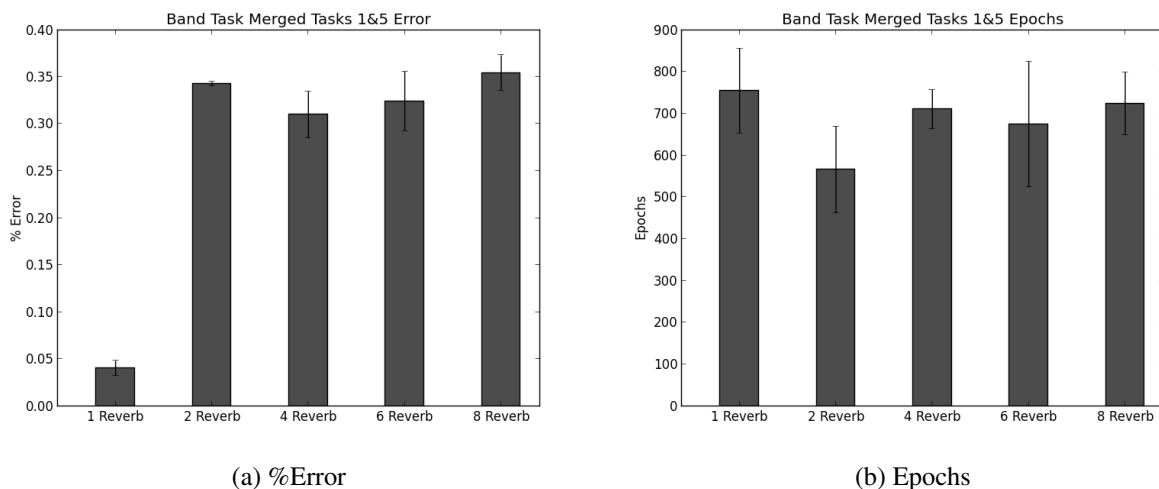


Figure 4.26: The classification error rate and the epochs to learn a merged task using only previously trained networks. The training data is generated by reverberating the sub-networks. These figures depict the Band data set

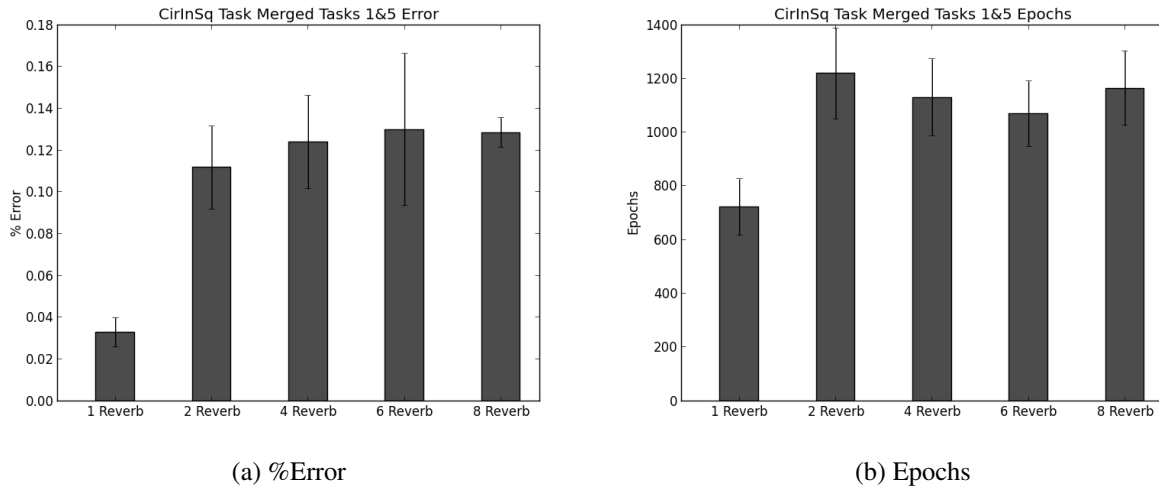


Figure 4.27: The classification error rate and the epochs to learn a merged task using only previously trained networks. The training data is generated by reverberating the sub-networks. These figures depict the CirInSq data set

higher reverberations was higher in this case but the difference is still significant.

The experiments in this section have shown that the merging process works well for these data sets when the number of reverberations is limited to a single pass through. This matches our earlier finding that the reverberation of networks trained on tasks with real random input vectors can cause a loss in performance. In these situations the random perturbations of the network are already valid inputs for the tasks these networks have learned. Reverberating through the network in this situation will only hinder the process of creating useful pseudo-items. The results from these experiments show that our merging technique is sound and is able to create new networks with good performance using only previously trained networks.

CHAPTER 5: CONCLUSION

In this research we have sought to create a sequential life-long learning system using generative neural networks, in the form of cascade correlation networks, and functional knowledge transfer, through MTL and CSMTL. In our first set of experiments (Section 4.2) we sought to test the efficacy of using generated neural networks with the MTL and CSMTL learning techniques, and to verify that the benefits of these functional knowledge transfer techniques were maintained while building the networks from scratch. We showed that using cascade correlation networks in MTL and CSMTL learning does in fact maintain the functional knowledge transfer effects by improving performance on impoverished training sets. We also showed that the choice of incorrect network structure using fixed networks can have a significant negative effect on performance, while the use of the CC architecture solves the problem completely. In the second set of experiments in section 4.3 we began by testing two methods for adding new inputs to existing CC networks to facilitate adding new task contexts and features on the fly, and determined that the CC network can gracefully incorporate new inputs using either technique. This allows us to build a more general system with the ability to add new tasks even when they require additional features. With this foundation in place we built and tested a dual network sequential learning system using CC networks and CSMTL. We tested the ability of this system to recall previously learned tasks, and to consolidate training for single tasks. We found that in most cases the dual network approach outperformed a single CC network, with the lone exception being the recall test when the two tasks were nearly identical. We also found that the utility of the Ans reverberation technique depends on the goal and type of data being trained. During recall it provided better performance when the input space was more structured, while in the consolidated training scenarios it provided better performance when the outputs were binary in nature. Our experiments also found that the use of the dual network technique while improving performance comes at the cost of increased training

epochs, though this was expected since we are consolidating long term knowledge while learning new tasks. New task performance could be improved by moving the consolidation phase later in the process, if speed of learning new tasks is the primary concern. We then applied our consolidated life long learning technique to build simulated agents from the observational data collected from users playing a simple game. We found that in general the consolidation technique allows for the creation of agents that perform well in the simulation given only limited training time from the human operators. Though there were cases where incorporating a bad training session hindered the training of the agent so care is needed when deciding the agents course of training. We received mixed subjective feedback from our users with one user very satisfied with the results, and one user disappointed that the agents didn't seem more like him. For all of our users the agents were able to pick up the skills to play the game. Finally we tested our network merging technique and found that we can successfully create merged networks with the knowledge of our previous networks.

5.1 Contributions

The primary goal of this research has been to build a life long learning system that is able to handle an arbitrary number of tasks and to apply it to the observational learning of human behaviors. Our Dual CC & KBCC reverberated system is able to learn tasks sequentially with minimal human intervention, and is able to incorporate previously trained networks or to merge multiple existing networks with the goal of continuing learning. The generative abilities of the CC technique allow it to grow our networks as they are needed so that our system can develop solutions of problems of different sizes. The major contributions of our research are listed below.

CC Life Long Learning System A life long learning system based on the Cascade Correlation algorithm for generative structure, CSMTL for inductive transfer, and reverberated pseudo-rehearsal for consolidation. This system has several benefits over Silver's LLL System.

1. A more elegant transition between new task learning and consolidation requiring less human intervention.
2. Functional and representational transfer in both the new task learning and consolidation steps, meaning we do not need to create a new network from scratch to consolidate each new task.
3. Better recall of previous tasks through the use of Ans reverberating pseudo-rehearsal, and representational transfer during consolidation, leading to less performance degradation of previously learned tasks.

Technique for merging separate learning paths A method of merging divergent learning paths, allowing agents to be spawned off into different scenarios and environments then recombining them to share their experience. This is useful as a method to combine the training from several users, or to allow for the use of multiple existing agents in the creation of a compound agent. This is accomplished through the extension of our Dual CC LLL system with the KBCC technique for incorporating sub-networks. This system uses the functional transfer of reverberated pseudo-rehearsal and the representational transfer of KBCC to combine networks together.

Open Source Library An open source library available at <http://jreeder.github.com/JRNN>. This library allows for other people to apply these methods to new applications and to extend the life long learning system with additional features. The bindings of this library into the OpenNERO project will also be included so that users can experiment with agent creation through observational training. The library also has a Python interface to allow for scripting and extension through the Python language.

5.2 Future Development

The development of the life long learning system developed for this research will continue in order to achieve the larger goals outlined in this manuscript. There are several short term tasks as well as longer term tasks planned. The short term tasks are focused on preparing the system for public release, while the longer term tasks are centered around expanding the system.

5.2.1 *Short-Term Tasks*

The first short term task is to clean up the software API (Application Programming Interface). The API is how other programmers interact with the software library to use our LLL system. Currently APIs exist for the C++ and Python programming languages. The first step required before public release is the improvement of our existing API. Currently the API is specific to the needs of our research and experiments. With some work the API can be cleaned up and made more general. This is an important step to making the system more useful to the greater research community since its adoption by other researchers is more likely if the system is easy to program. Once the API is ready additional language interfaces, such as java or ruby, could be added to further increase the appeal of our system.

The second short term task is the proper documentation of our software library. This goes hand-in-hand with the API clean up and will be completed simultaneously. Good documentation is essential for a software library to become more useful to the community. The best software in the world will not be useful if other people can't figure out how to use it. There are several tools available to facilitate the automatic generation of documentation from the comments in the source code. We will utilize the Doxygen (www.doxygen.org) system to generate the documentation for our library. It is capable of generating both a website and \LaTeX version of the documentation.

With a consistent API and good documentation the software library will be ready for public release. The next task will be to prepare the OpenNERO simulation environment for public release. There are several improvements that need to be made to the user interface to facilitate the creation of agents by end users. Currently the interface allows the user to create single episode agents, but doesn't provide an interface for merging or consolidating agents. These two methods will be exposed to the users through an improved user interface that allows the user to use all of the power of our LLL system. Another improvement to the simulation will be the addition of sharing features. The ability to import or export agents that can then be distributed to others using the simulator. The final improvement needed for the simulator is the creation of a simplified installer. Currently setting up the simulator requires several libraries and programs to be installed on the computer, and a good understanding of how the library dependencies work. In order for the simulator to be ready for public use a streamlined installer needs to be built so that users can install the simulator and LLL system like they would any other piece of software. Once the installation process has been streamlined the simulator will be released alongside the LLL software library.

5.2.2 Long-Term Tasks

The foundation provided by the open source library and simulation provide several avenues for longer term research. There are many areas ripe for improvement and extended study.

The existing simulation works well for individual users, but the power of our consolidation and merging techniques would be more useful in a scenario with many users collaborating. There are several features and tools that could be added to the simulation to assist user collaboration. The simulation could be extended to have a shared server back-end to facilitate the sharing and trading of agents between users. A tool for generating additional scenarios could be developed to allow users to create more complicated and interesting tasks. This could be taken even further by creating

a simulated multi-player game where users design and train agents on tasks, then trade them with other users.

The observational training of game agents is only one possible application of our LLL system. It could also be applied to other types of agent learning like reinforcement learning or multi-agent learning, but it doesn't have to be constrained to agent based techniques alone. The use of neural networks as our underlying model allows our system to be used in most situations where neural networks are already utilized. Applying our LLL system to many popular areas where neural networks are currently used would be a very large area for future research.

The use of Cascade Correlation, Knowledge-based Cascade Correlation, and Dual-network pseudo-rehearsal as our building blocks for our life long learning system is a design choice. The most fundamental and longest term area of research for our system going forward is the improvement of the underlying building blocks of the system. As research in the field continues, improvements in the areas of generative neural networks, or methods for dealing with catastrophic forgetting may emerge that could improve the efficacy of our system. The beauty of using an online open source library for our system is that any improvements made to the system are distributed out to those people using it. This also means that any improvements made by others can be integrated back into our system as well. The distributed nature of the open source community means that the more our system is used, and new improvements to its underlying parts are found, the quicker the system will improve as a whole. In this way our life long learning system can itself be life long learning.

LIST OF REFERENCES

- [1] YS Abu-Mostafa. Hints. *Neural Computation*, 1995.
- [2] B. Ans and S. Rousset. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'Académie des Sciences-Series III-Sciences de la Vie*, 320(12):989–997, 1997.
- [3] Bernard Ans. Sequential Learning in Distributed Neural Networks without Catastrophic Forgetting: A Single and Realistic Self-Refreshing Memory Can Do It. *Neural Information Processing-Letters and Reviews*, 4(2):27–32, March 2004.
- [4] Bernard Ans and Stephane Rousset. Neural networks with a self-refreshing memory: knowledge transfer in sequential learning tasks without catastrophic forgetting. *Connection Science*, 12(1):1–19, January 2000.
- [5] Bernard Ans, Stephane Rousset, R. M. French, and S Musca. Preventing Catastrophic Interference in Multiple-Sequence Learning Using Coupled Reverberating Elman Networks. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society*, 2002.
- [6] T Balch. *Behavioral diversity in learning robot teams*. PhD thesis, Georgia Institute of Technology, 1998.
- [7] J. Baxter. Learning model bias. *Advances in Neural Information Processing Systems*, pages 169–175, 1996.
- [8] Jonathan Baxter. Learning internal representations. In *Annual Workshop on Computational Learning Theory: Proceedings of the eighth annual conference on Computational learning theory*, pages 311–320, New York, NY, USA, 1995. ACM.

- [9] George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, July 1994.
- [10] G a Carpenter and S Grossberg. ART 2: self-organization of stable category recognition codes for analog input patterns. *Applied optics*, 26(23):4919–30, December 1987.
- [11] G.A. Carpenter, S. Grossberg, and J.H. Reynolds. ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural networks*, 4(5):565–588, 1991.
- [12] Rich Caruana. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Machine Learning, Proceedings of the Tenth International Conference*, volume 2, pages 41–48, Amherst, MA, USA, December 1993. Morgan Kaufmann Publishers.
- [13] Rich Caruana. *Multitask Learning*. Phd, Carnegie Mellon University, January 1997.
- [14] Rich Caruana. A dozen tricks with multitask learning. *Lecture Notes in Computer Science*, 1998.
- [15] Rich Caruana, S Lawrence, and L Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems 13: ...*, 2001.
- [16] H. Chen. Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Machine Learning*, 1995.
- [17] M.W. Craven and J.W. Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80. Citeseer, 1993.
- [18] AS d’Avila Garcez, K. Broda, and D.M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125(1-2):155–207, 2001.

- [19] Gerald Dejong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.
- [20] Aldo Franco Dragoni and Paolo Giorgini. Belief revision through the belief-function formalism in a multi-agent environment. *INTELLIGENT AGENTS III AGENT THEORIES, ARCHITECTURES, AND LANGUAGES*, 1193:103–115, 1997.
- [21] Scott E. Fahlman. CASCOR: Lisp and C implementations of Cascade Correlation. <http://www.cs.cmu.edu/Groups/AI/areas/neural/systems/cascor/0.html>.
- [22] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report 3, Carnegie Mellon University, Pittsburgh, PA, 1988.
- [23] Scott E. Fahlman. The Recurrent Cascade-Correlation Architecture. Technical Report 2, Carnegie Mellon University, Pittsburgh, PA, March 1991.
- [24] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation Learning Architecture. Technical Report 6, Carnegie Mellon University, Pittsburgh, PA, January 1991.
- [25] EDWARD A. FEIGENBAUM. Knowledge Engineering: The Applied Side of Artificial Intelligence. *Annals of the New York Academy of Sciences*, 426(1 Computer Culture: The Scientific, Intellectual, and Social Impact of the Computer):91–107, November 1984.
- [26] H K G Fernlund, A J Gonzalez, M Georgiopoulos, and R F DeMara. Learning tactical human behavior through observation of human performance. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(1):128–140, 2006.
- [27] A Frank and A Asuncion. {UCI} Machine Learning Repository. <http://archive.ics.uci.edu/ml>, 2010.
- [28] R.M. French. Pseudo-recurrent connectionist networks: An approach to the ‘sensitivity-stability’ dilemma. *Connection Science*, 9(4):353–380, 1997.

- [29] Rm French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, April 1999.
- [30] H Friedrich, O. Rogalla, and R. Dillmann. Communication and propagation of action knowledge in multi-agent systems. *Robotics and Autonomous Systems*, 29(1):41–50, 1999.
- [31] A. Garland and R. Alterman. Preparation of multi-agent knowledge for reuse. In *Proceedings of the Fall Symposium on Adaptation of Knowledge for Reuse*, volume 26, page 33, 1995.
- [32] S Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1988.
- [33] A E Henninger, A J Gonzalez, M Georgipoulos, and R F DeMara. The limitations of static performance metrics for dynamic tasks learned through observation. *Ann Arbor*, 1001:43031, 2001.
- [34] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–356, 1989.
- [35] A.K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.
- [36] K Kasabov and Others. On-line learning, reasoning, rule extraction and aggregation in locally optimized evolving fuzzy neural networks. *Neurocomputing*, 41(1-4):25–45, 2001.
- [37] C. Le Pape. A combination of centralized and distributed methods for multi-agent planning and scheduling. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 488–493. IEEE, 1990.
- [38] S. Lee and S. Shimoji. Machine acquisition of skills by neural networks. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 781–788. IEEE, 1991.

- [39] M. McCloskey and N.J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The psychology of learning and motivation*, 24:109–165, 1989.
- [40] Tom M. Mitchell. *Machine Learning*. McGraw Hill, New York, NY, USA, 1997.
- [41] David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22(1-3):11–32, 1996.
- [42] D.K. Naik and R.J. Mammone. Meta-neural networks that learn by learning. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, 1:437–442, 1992.
- [43] J O’Sullivan, T.M. Mitchell, and S. Thrun. Explanation based learning for mobile robot perception. In *Symbolic visual learning*, pages 295–324. Oxford University Press, Inc., 1997.
- [44] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.
- [45] Ryan Poirier and Daniel L. Silver. csMTL: a Context Sensitive Lifelong Learning System. In *NIPS 2005 Workshop on Transfer Learning*, 2005.
- [46] LY Pratt, SJ Hanson, and CL Giles. Discriminability-based transfer between neural networks. *Advances in Neural*, 1993.
- [47] Andrzej Pronobis, Luo Jie, and Barbara Caputo. The more you learn, the less you store: Memory-controlled incremental SVM for visual place recognition. *Image and Vision Computing*, 2010.
- [48] R Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285–308, April 1990.
- [49] John Reeder. JRNN: John Reeder’s Neural Network Library. <https://github.com/jreeder/JRNN>.

- [50] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. Ieee, 1993.
- [51] M. Riedmiller and H. Braun. Rprop-description and implementation details. Technical report, Citeseer, 1994.
- [52] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.
- [53] Anthony Robins. Maintaining stability during new learning in neural networks. In *Systems, Man, and Cybernetics, 1997.'Computational Cybernetics and Simulation'. 1997 IEEE International Conference on*, volume 4, pages 3013–3018. IEEE, 1997.
- [54] C Sammut, S Hurst, D Kedzier, D Michie, and Others. Learning to fly. In *Proceedings of the ninth international workshop on Machine learning*, pages 385–393. Citeseer, 1992.
- [55] S. Sen and G. Weiss. Learning in Multiagent Systems. In *Multiagent systems: A modern approach to distributed artificial intelligence*, chapter 6, page 259. The MIT Press, 1999.
- [56] Thomas R Shultz and Francois Rivest. Knowledge-based cascade-correlation : using knowledge to speed learning. *Connection Science*, 13(1):43–72, 2001.
- [57] Daniel L. Silver. The Parallel Transfer of Task Knowledge Using Dynamic Learning Rates Based on a Measure of Relatedness. *Connection Science*, 8(2):277–294, June 1996.
- [58] Daniel L. Silver and Robert E. Mercer. The Task Rehearsal Method of Life-Long Learning: Overcoming Impoverished Data. *Advances in Artificial Intelligence*, 2338:90–101, 2002.
- [59] Daniel L. Silver and R Poirier. Context-sensitive mtl networks for machine lifelong learning. In *Proceedings of the 20th Florida Artificial Intelligence Research Society Conference*, volume 18, pages 433–441, Orlando, FL, USA, October 2007. AAAI Press.

- [60] Daniel L. Silver and R Poirier. Requirements for machine lifelong learning. *Lecture Notes in Computer Science*, 2007.
- [61] Daniel L. Silver, Ryan Poirier, and Duane Currie. Inductive transfer with context-sensitive neural networks. *Machine Learning*, 73(3):313–336, December 2008.
- [62] Kenneth O. Stanley, B.D. Bryant, and Risto Miikkulainen. Real-Time Neuroevolution in the NERO Video Game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, December 2005.
- [63] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, May 2002.
- [64] G. Stein. *FALCONET: Force-feedback approach for learning from coaching and observation using natural and experiential training*. PhD thesis, University of Central Florida, 2009.
- [65] Peter Stone and Manuela Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective, 2000.
- [66] Retro Studios, Kramer Lane, Igor V Karpov, John Sheblak, and Risto Miikkulainen. Open-NERO : a Game Platform for AI Research and Education. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 220–221, 2008.
- [67] Matthew E Taylor and Peter Stone. Transfer Learning for Reinforcement Learning Domains : A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.
- [68] M.E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59. ACM, 2005.
- [69] M.E. Taylor, P Stone, and Y Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):21252167, 2007.

- [70] Sebastian Thrun and Tom M. Mitchell. Learning one more thing. *International Joint Conference on Artificial Intelligence*, 14:1217—1225, 1995.
- [71] Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 11(2):167–75, March 1995.
- [72] A Weigend. On overfitting and the effective number of hidden units. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 335–342. Lawrence Erlbaum Associates, 1993.