To appear in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (AAAI-15), Austin, Texas, USA, January 2015.

UT Austin Villa 2014: RoboCup 3D Simulation League Champion via Overlapping Layered Learning

Patrick MacAlpine and **Mike Depinet** and **Peter Stone**
Department of Computer Science
The University of Texas at Austin
Austin, TX 78701, USA
{patmac, msd775, pstone}@cs.utexas.edu

Abstract

Layered learning is a hierarchical machine learning paradigm that enables learning of complex behaviors by incrementally learning a series of sub-behaviors. A key feature of layered learning is that higher layers directly depend on the learned lower layers. In its original formulation, lower layers were frozen prior to learning higher layers. This paper considers an extension to the paradigm that allows learning certain behaviors independently, and then later stitching them together by learning at the "seams" where their influences overlap. The UT Austin Villa 2014 RoboCup 3D simulation team, using such overlapping layered learning, learned a total of 19 layered behaviors for a simulated soccer-playing robot, organized both in series and in parallel. To the best of our knowledge this is more than three times the number of layered behaviors in any prior layered learning system. Furthermore, the complete learning process is repeated on four different robot body types, showcasing its generality as a paradigm for efficient behavior learning. The resulting team won the RoboCup 2014 championship with an undefeated record, scoring 52 goals and conceding none. This paper includes a detailed experimental analysis of the team's performance and the overlapping layered learning approach that led to its success.

1 Introduction

Task decomposition is a popular approach for learning complex control tasks when monolithic learning (trying to learn the complete task all at once) is difficult or intractable (Singh 1992; Whitehead, Karlsson, and Tenenberg 1993; Whiteson et al. 2005). Layered learning (Stone 2000) is a hierarchical task decomposition machine learning paradigm that enables learning of complex behaviors by incrementally learning a series of sub-behaviors. A key feature of layered learning is that higher layers directly depend on the learned lower layers. In its original formulation, lower layers were frozen prior to learning higher layers. This can be restrictive, however, as freezing lower layers limits the combined behavior search space over all layers. Concurrent layered learning (Whiteson and Stone 2003) reduced this restriction in the search space by introducing the possibility of learning some of the behaviors simultaneously by "reopening" learning at the lower layers while learning the higher layers. A potential drawback of increasing the size of the search space, however, is an increase in the dimensionality and thus possibly the difficulty of what is being learned.

This paper considers an extension to the layered learning paradigm, known as *overlapping layered learning*, that allows learning certain behaviors independently, and then later stitching them together by learning at the "seams" where their influences overlap. Overlapping layered learning aims to provide a middle ground between reductions in the search space caused by freezing previously learned layers and the increased dimensionality of concurrent layered learning. Additionally, for complex tasks where it is hard learning one subtask in the presence of another, it reduces the dimensionality of the parameter search space by focusing only on parts responsible for subtasks working together.

The UT Austin Villa 2014 RoboCup 3D simulation team, using overlapping layered learning, learned a total of 19 layered behaviors for a simulated soccer-playing robot, organized both in series and in parallel. To the best of our knowledge this is more than three times the number of layered behaviors in any prior layered learning system. Furthermore, the complete learning process is repeated on four different heterogeneous robot body types, showcasing its generality as a paradigm for efficient behavior learning. The resulting team won the RoboCup 2014 championship with an undefeated record, scoring 52 goals and conceding none.

Primary contributions of this paper are twofold. First, we introduce and motivate general scenarios for using the overlapping layered learning paradigm. Second, we provide a detailed description and analysis of our machine learning approach, incorporating overlapping layered learning, to create a large and complex control system that was a core component of the 2014 3D simulation league championship team.

The remainder of this paper is organized as follows. Section 2 specifies and motivates the overlapping layered learning paradigm while contrasting it with traditional and concurrent layered learning. In Section 3 we introduce the RoboCup 3D simulation domain this research takes place in. Section 4 details the overlapping layered learning approach of the 2014 UT Austin Villa team and in Section 5 we provide detailed analysis of its performance. Section 6 discusses related work while Section 7 concludes.

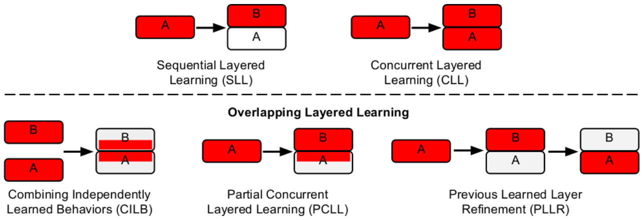

Figure 1: Different paradigms for layered learning with layers or parts of layers being learned shown in red.

2 Overlapping Layered Learning

Layered learning is a hierarchical learning paradigm that enables learning of complex behaviors by incrementally learning a series of sub-behaviors (each learned sub-behavior is a layer in the learning progression). Higher layers depend on lower layers for learning. This dependence can include providing features for learning, such as seed values for parameters, as well as a previous learned layer's behavior being incorporated into the learning task for the next layer to be learned. In its original formulation, layers are learned in a sequential bottom-up fashion and, after a layer is learned, it is frozen before beginning learning of the next layer.

Concurrent layered learning, on the other hand, purposely does not freeze newly learned layers, but instead keeps them open during learning of subsequent layers. This is done so that learning may enter areas of the behavior search space that are closer to the combined layers' optimum behavior as opposed to being confined to areas of the joint layer search space where the behaviors of previously learned layers are fixed. While concurrent layered learning does not restrict the search space in the way that freezing learned layers does, the increase in the search space's dimensionality can make learning slower and more difficult.

Overlapping layered learning seeks to find a tradeoff between freezing each layer once learning is complete and leaving previously learned layers open. It does so by *keeping some, but not necessarily all, parts of previously learned layers open during learning of subsequent layers.* The part of previously learned layers left open is the "overlap" with the next layer being learned. In this regard concurrent layered learning can be thought of as an extreme of overlapping layered learning with a "full overlap" between layers.

The following are several general scenarios, depicted in the bottom row of Figure 1, for overlapping layered learning that help to clarify the learning paradigm and identify situations in which it is useful:

Combining Independently Learned Behaviors (CILB): Two or more behaviors are learned independently in the same layer, but then are combined together for a joint behavior at the subsequent layer by relearning some subset of the behaviors' parameters or "seam" between the behaviors. This scenario is best when subtask behaviors are too complex and/or potentially interfere with each other during learning, such that they must be learned independently, but ultimately need to work together for a combined task. *Example: A basketball playing robot that must be able to dribble the ball across the court and shoot it in the basket. The tasks of dribbling and shooting are too complex to attempt to learn them together, but after the tasks are learned independently they can be combined by re-optimizing parameters that control the point on the court at which the robot stops dribbling and the angle at which the robot shoots the ball.*

Partial Concurrent Layered Learning (PCLL): Only part, but not all, of a previously learned layer's behavior parameters are left open when learning a subsequent layer with new parameters. The part of the previously learned layer's parameters left open is the "seam" between the layers. Partial concurrent learning is beneficial if full concurrent learning unnecessarily increases the dimensionality of the search space to the point that it hinders learning, and completely freezing the previous layer diminishes the potential behavior of the layers working together. *Example: Teaching one robot to pick up and hand an object to another robot. First a robot is taught to pick up an an object and then reach out its arm and release the object. The second robot is then taught to reach out its arm and catch the object released by the first robot. During learning by the second robot to catch the object, the part of the previously learned behavior of the first robot to hand over the object is left open so that the first robot can adjust its release point of the object to a place that the second robot can be sure to reach.*

Previous Learned Layer Refinement (PLLR): After a layer is learned and frozen, and then a subsequent layer is learned, part or all of the previously learned layer is then unfrozen and relearned to better work with the newly learned layer that is now fully or partially frozen. We consider re-optimizing a previously frozen layer under new conditions as a new learned layer behavior with the "seam" between behaviors being the unfrozen part of the previous learned layer. This scenario is useful when a subtask is required to be learned before the next subsequent task layer can be learned, but then refining or relearning the original learned task layer to better work with the newly learned layer provides a benefit. *Example: Teaching a robot to walk. First the robot needs to learn how to stand up so that if it falls over it can get back up and continue trying to walk. Eventually the robot learns to walk so well that it barely if ever falls over during training. Later, when the robot does eventually fall over, it is found that the walking motion learned by the robot is not stable if the robot tries to walk right after standing up. The robot needs to relearn the standing up behavior layer such that after doing so it is in a stable position to start walking with the learned walking behavior layer.*

3 Domain Description

Robot soccer has served as an excellent testbed for learning scenarios in which multiple skills, decisions, and controls have to be learned by a single agent, and agents themselves have to cooperate or compete. There is a rich literature based on this domain addressing a wide spectrum of topics from low-level concerns, such as perception and motor control (Behnke et al. 2006; Riedmiller et al. 2009), to high-

Figure 2: A screenshot of the Nao humanoid robot (left), and a view of the soccer field during a 11 versus 11 game (right).

level decision-making (Kalyanakrishnan and Stone 2010).

The RoboCup 3D simulation environment is based on SimSpark,[1] a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine[2] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints.

The robot agents in the simulation are modeled after the Aldebaran Nao robot,[3] which has a height of about 57 cm and a mass of 4.5 kg. The agents interact with the simulator by sending torque commands and receiving perceptual information. Each robot has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. Joint perceptors provide the agent with noise-free angular measurements every simulation cycle (20 ms), while joint effectors allow the agent to specify the torque and direction in which to move a joint. Although there is no intentional noise in actuation, there is slight actuation noise that results from approximations in the physics engine and the need to constrain computations to be performed in real-time.

In addition to the standard Nao robot model, four additional variations of the standard model, known as heterogeneous types, are available for use. The variations from the standard model include changes in leg and arm length, hip width, and also the addition of toes to the robot's foot. Figure 2 shows a visualization of the standard Nao robot and the soccer field during a game.

4 Overlapping Layered Learning Approach

The 2014 UT Austin Villa team used an extensive layered learning approach to learn skills for the robot such as getting up, walking, and kicking. This includes sequential layered learning where a newly learned layer is frozen before learning of subsequent layers, as well as overlapping layers where parts of previously learned layers are re-optimized as part of the current layer being learned.

In total over 500 parameters were optimized during the course of layered learning. All parameters were optimized using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm (Hansen 2009), which has been successfully applied previously to learning skills in the RoboCup 3D simulation domain (Urieli et al. 2011). A total of 705,000 learning trials were performed during the process of optimizing 19 behaviors. Optimization was performed on a Condor (Thain, Tannenbaum, and Livny 2005) distributed computing cluster allowing for many jobs to be run in parallel. Running the complete optimization process took about 5 days, and we calculated it could theoretically be completed in as little as 49 hours assuming no job queuing delays on the computing cluster, and all possible parallelism during the optimization process is exploited. Note that this same amount of computation, when performed sequentially on a single computer,[4] would take approximately 561 days, or a little over 1.5 years, to finish.

The following subsections document the overlapping layered learning parts of the approach used by the team. Due to space constraints full details of some of the learned behavior layers are omitted, however a diagram of how all the different layered learning behaviors fit together during the course of learning can be seen in Figure 3 with a brief description of each behavior provided in Figure 4.

4.1 Getup and Walking using PLLR

The UT Austin Villa team employs an omnidirectional walk engine using a double inverted pendulum model to control walking. The walk engine has many parameters that need to be optimized in order to create a stable and fast walk including the length and frequency of steps as well as center of mass offsets. Instead of having a single set of parameters for the walk engine, which in previous work we found to limit performance, walking is broken up into different subtasks for each of which a set of walk engine parameters is learned (MacAlpine et al. 2012a).

Before optimizing parameters for the walk engine, getup behaviors are optimized so that if the robot falls over it is able to stand back up and start walking again. Getup behaviors are necessary for faster learning during walk optimizations, as without the ability to get up after falling, a walk optimization task would have to be terminated as soon as the robot fell over. There are two such behaviors for getting up: *GetUp_Front_Primitive* for standing up from lying face down and *Getup_Back_Primitive* for standing up from lying face up. Each getup behavior is parametrized by a series of different joint angles and is evaluated on how quickly the robot is able to stand up (MacAlpine et al. 2013).

After learning both the *Walk_GoToTarget* and *Walk_Sprint* walk engine parameter sets, we re-optimize the getups by learning the *GetUp_Front_Behavior* and *GetUp_Back_Behavior* behaviors. *GetUp_Front_Behavior* and *GetUp_Back_Behavior* are overlapping layered learning behaviors as they contain the same parameters as the previously learned *GetUp_Front_Primitive* and *GetUp_Back_Primitive* behaviors respectively. The getup behavior parameters are re-optimized from their primitive behavior values through the same optimization as the getup primitives, but with the addition that right after completing a getup behavior the robot is asked to walk in different directions and is penalized if it falls over while trying to

[1] http://simspark.sourceforge.net/

[2] http://www.ode.org/

[3] http://www.aldebaran-robotics.com/eng/

[4] As measured on an Intel(R) Xeon(R) CPU E31270 @ 3.40GHz.

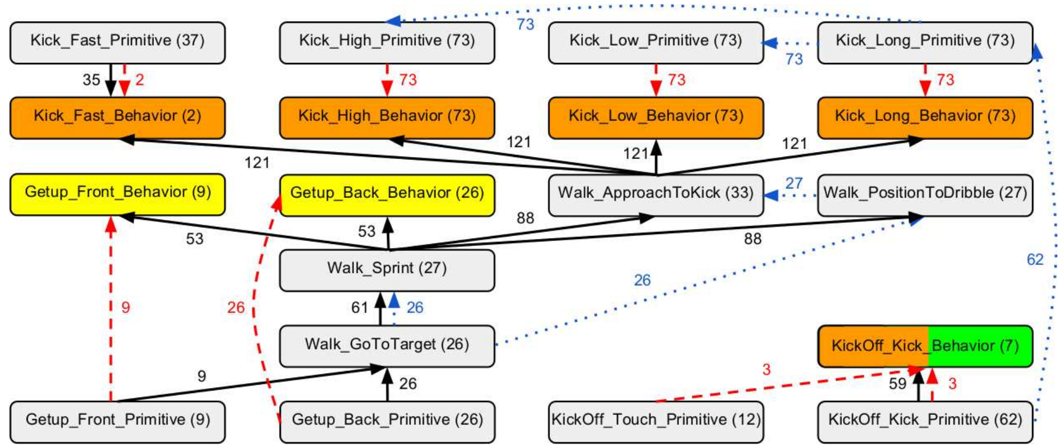

Figure 3: Different layered learning behaviors with the number of parameters optimized for each behavior shown in parentheses. Solid black arrows show number of learned and frozen parameters passed from previously learned layer behaviors, dashed red arrows show the number of overlapping parameters being passed and relearned from one behavior to another, and the dotted blue arrows show the number of parameter values being passed as seed values to be used in new parameters at the next layer of learning. Overlapping layers are colored with CILB layers in orange, PCLL in green, and PLLR in yellow. Descriptions of the layers are given in Figure 4.

do so. Unlike the getup primitives, which were learned in isolation, the getup behaviors are stable transitioning from standing up and then almost immediately walking. One might think that the walk parameter sets learned would be stable transitioning from the getups due to the getup primitives being used in the walk parameter optimization tasks, however this is not always the case as during learning walks become stable such that toward the end of optimizing a walk parameter set the robot almost never falls and thus rarely uses the getup primitives. Learning the getup behaviors is an example of previous learned layer refinement (PLLR).

Getup_Front_Behavior: Getup front primitive with walk.
Getup_Back_Behavior: Getup back primitive with walk.
Getup_Front_Primitive: Stand up when lying on front.
Getup_Back_Primitive: Stand up when lying on back.
KickOff_Kick_Behavior: Two agent behavior scores on kickoff with one agent touching ball before other kicks.
KickOff_Kick_Primitive: Single agent kick behavior that scores on a kickoff from a motionless ball.
KickOff_Touch_Primitive: Single agent behavior where agent lightly touches ball resulting in little ball motion.
Kick_Fast_Behavior: Fast kick primitive with walk.
Kick_Fast_Primitive: Fast and short kick.
Kick_High_Behavior: High kick primitive with walk.
Kick_High_Primitive: Kick for kicking over opponents.
Kick_Low_Behavior: Low kick primitive with walk.
Kick_Low_Primitive: Kick ball below goal height.
Kick_Long_Behavior: Long kick primitive with walk.
Kick_Long_Primitive: Long kick.
Walk_ApproachToKick: Walk parameters for stopping at a precise position behind the ball before kicking.
Walk_GoToTarget: General default walk parameters.
Walk_PositionToDribble: Dribbling walk parameters.
Walk_Sprint: Walk parameters for walking forward fast.

Figure 4: Description of layered learning behaviors in Figure 3.

4.2 Kicking using CILB

Four primitive kick behaviors were learned by the 2014 UT Austin Villa team (*Kick_Long_Primitive*, *Kick_Low_Primitive*, *Kick_High_Primitive*, and *Kick_Fast_Primitive*. Each kick primitive, or kicking motion, was learned by placing the robot at a fixed position behind the ball and having it optimize joint angles for a fixed set of key motion frames. Note that initial attempts at learning kicks directly with the walk, instead of learning kick primitives independently, proved to be too difficult due to the variance in stopping positions of the walk as the robot approached to kick the ball.

While the kick primitive behaviors work quite well when the robot is placed in a standing position behind the ball, they are very hard to execute when the robot tries to walk up to the ball and kick it. One reason for this difficulty is that when the robot approaches the ball to kick it using the *Walk_ApproachToKick* walk parameter set the precise offset position from the ball that the kick primitives were optimized to work with do not match that of the position the robot stops at after walking up to the ball. In order to allow the robot to transition from walking to kicking, full kick behaviors for all the kicks are optimized (*Kick_Long_Behavior*, *Kick_Low_Behavior*, *Kick_High_Behavior*, *Kick_Fast_Behavior*). Each full kick behavior is learned by having the robot walk up to the ball and attempt to kick it from different starting positions (as opposed to having the robot just standing behind the ball as was done when optimizing the kick primitive behaviors).

The full kick behaviors are overlapping layered learning behaviors as they are re-optimizing previous learned parameters. In the case of *Kick_Fast_Behavior* only the x and y kick primitive offset position parameters from the ball, which is the target position for the walk to reach for the kick to be executed, are re-optimized. The fast kick is quick enough that it almost immediately kicks the ball after transitioning from walking, and thus just needs to be in the correct posi-

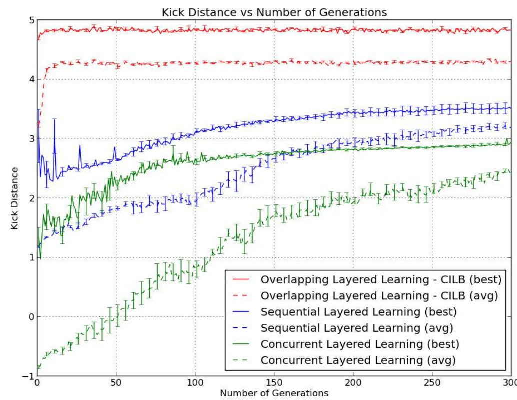

Figure 5: Performance of different layered learning paradigms across generations of CMA-ES when optimizing *Kick_Fast_Behavior*. Results are averaged across five optimization runs and error bars show the standard error.

tion near the ball to do so. A comparison of overlapping layered learning paradigms for learning *Kick_Fast_Behavior* is shown in Figure 5. The above overlapping layered approach of first independently learning the walk approach and kick, and then learning the two position parameters, does better than both the sequential layered learning approach where all kick parameters are learned after freezing the approach, and the concurrent layered learning approach where both approach and kick parameters are learned simultaneously.

For the other full kick behaviors all kick parameters from their respective kick primitive behaviors are re-optimized. Unlike the fast kick, there is at least a one second delay between stopping walking and kicking the ball during which the robot can easily become destabilized and fall over. By opening up all kicking parameters the robot has the necessary freedom to learn kick motions that maintain its stability between stopping after walking and making contact with the ball. Learning the kick behaviors by combining them with the kick approach walk behavior are all examples of combining independently learned behaviors (CILB).

4.3 KickOff using both CILB and PCLL

For kickoffs the robot is allowed to teleport itself to a starting position next to the ball before kicking it, and thus does not need to worry about walking up to the ball. Scoring directly off a kickoff is not allowed, however, as another robot must first touch the ball before it goes into the opponent's goal. In order to score on a kickoff we perform a multiagent task where one robot touches the ball before another kicks it.

The first behavior optimized for scoring off the kickoff is *KickOff_Kick_Primitive* in which a robot kicks the ball from the middle of the field. The robot is rewarded for kicking the ball as high and as far as possible as long as the ball crosses the goal line below the height of the goal. In parallel a behavior for another robot is learned to lightly touch the ball called *KickOff_Touch_Primitive*. Here a robot is rewarded for touching the ball lightly and, after ensuring that the robot has made contact with the ball, that the ball moves as little as possible. Finally an overlapping layered behavior called *KickOff_Kick_Behavior* is learned which re-optimizes x, y, and angle offset positions from the ball from both the *KickOff_Kick_Primitive* and *KickOff_Touch_Primitive* behaviors. Re-optimizing these positioning parameters together is important so that the robots do not accidentally collide with each other and also so that the kicking robot is at a good position to kick the ball after the first agent touches it. Learning *KickOff_Kick_Behavior* is another example of combining independently learned behaviors (CILB).

In addition to the positioning parameters of both robots being re-optimized for *KickOff_Kick_Behavior*, a new parameter that determines the time at which the first robot touches the ball is optimized. This synchronized timing parameter is necessary so that the robots are synced with each other and the kicking robot does not accidentally try to kick the ball before the first robot has touched it. As a new parameter is optimized along with a subset of previously learned parameters, learning *KickOff_Kick_Behavior* is also an example of partial concurrent layered learning (PCLL).

Further information about the kickoff, including how a seed for the kick was learned through observation, can be found in (Depinet, MacAlpine, and Stone 2015).

5 Results and Analysis

The 2014 UT Austin Villa team finished first among 12 teams at the RoboCup 3D simulation competition while scoring 52 goals and conceding none across 15 games. Considering that most of the team's strategy layer, including team formations using a dynamic role assignment and formation positioning system (MacAlpine, Price, and Stone 2015), remained unchanged from that of the previous year's second place finishing team, a key component to the 2014 team's improvement and success at the competition was the new overlapping layered learning approach to learning the team's low level behaviors.

After every RoboCup competition teams are required to release the binaries that they used during the competition. In order to analyze the performance of the different components of our overlapping layered learning approach we played 1000 games with different versions of the UT Austin Villa team against each of the top three teams from the RoboCup 2013 competition (at the time of writing this paper team binaries from the 2014 competition were not yet available). The following subsections provide analysis of game results when turning on and off the kickoff and kicking components learned though an overlapping layered learning approach. Additionally, to demonstrate the generality of our overlapping layered learning approach, we provide data that isolates the performance of our complete overlapping layered learning approach applied to different robot models.

5.1 Overall Team Performance

Table 1 shows the average goal difference across all games against each opponent achieved by the complete 2014 UT Austin Villa team. Against all opponents the team had a significantly positive goal difference and in fact out of the 3000 games played the team only lost one game (to AustinVilla2013). This shows the effectiveness of the team's overlapping layered learning approach in dramatically improving the performance of the team from the previous year

Table 1: Full game results, averaged over 1000 games. Each row corresponds to one of the top three finishing teams at RoboCup 2013. Entries show the average goal difference achieved by the 2014 UT Austin Villa team versus the given opponent team. Values in parentheses are the standard error. Total number of wins, losses, and ties across all games was 2852, 1, and 147 respectively.

Opponent	Average Goal Difference
Apollo3D	2.726 (0.036)
AustinVilla2013	1.525 (0.032)
FCPortugal	3.951 (0.049)

Table 2: Full game results, averaged over 1000 games. Each row corresponds to one of the top three finishing teams at RoboCup 2013. Entries show the average goal difference achieved by a version of the 2014 UT Austin Villa team *not attempting to score on a kickoff* versus the given opponent team. Values in parentheses are the standard error. Total number of wins, losses, and ties across all games was 2644, 5, and 351 respectively.

Opponent	Average Goal Difference
Apollo3D	2.059 (0.038)
AustinVilla2013	1.232 (0.032)
FCPortugal	3.154 (0.046)

in which the team achieved second place at the competition (and is now able to beat last year's team by an average of 1.525 goals).

Recent data released in (MacAlpine et al. 2015) showing the overall team's performance when playing against the released 2014 teams' binaries corroborates the overall team's strong performance. When playing 1000 games against each of the eleven 2014 opponents UT Austin Villa did not lose a single game out of the 11,000 played, and had at least an average goal difference of 2 against every opponent.

5.2 KickOff Performance

To isolate the performance of the learned multiagent behavior to score off the kickoff, we disabled this feature and instead just had the robot taking the kickoff kick the ball toward the opponent's goal to a position as close as possible to one of the goal posts without scoring. Table 2 shows results from playing against the top three teams at RoboCup 2013 without attempting to score on the kickoff.

By comparing results in Table 2 to that of Table 1 we see a significant drop in performance when not attempting to score on kickoffs. This result is not surprising as we found that the kickoff was able to score around 90% of the time against Apollo3D and FCPortugal, and over 60% of the time against the 2013 version of UT Austin Villa. The combination of using both CILB and PCLL overlapping layered learning garnered a large boost to the team's performance.

5.3 Kicking Performance

To isolate the performance of kicking learned through an overlapping layered learning approach we disable all kicking (except for on kickoffs where we once again have a robot kick the ball as far as possible toward the opponent's goal without scoring) and used an always dribble behavior. Data from playing against the top three teams at the RoboCup 2013 competition when only dribbling is shown in Table 3.

Table 3: Full game results, averaged over 1000 games. Each row corresponds to one of the top three finishing teams at RoboCup 2013. Entries show the average goal difference achieved by a version of the 2014 UT Austin Villa team *using a dribble only strategy* versus the given opponent team. Values in parentheses are the standard error. Total number of wins, losses, and ties across all games was 2480, 15, and 505 respectively.

Opponent	Average Goal Difference
Apollo3D	1.790 (0.033)
AustinVilla2013	0.831 (0.023)
FCPortugal	1.593 (0.028)

Table 4: Full game results, averaged over 1000 games. Each row corresponds to one of the top three finishing teams at RoboCup 2013. Entries show the average goal difference achieved by a version of the 2014 UT Austin Villa team *using different heterogeneous robot types* versus the given opponent team.

	Avg. Goal Difference per Robot Type				
Opponent	Type 0	Type 1	Type 2	Type 3	Type 4
Apollo3D	1.788	1.907	1.892	1.524	2.681
AustinVilla2013	0.950	0.858	1.152	0.613	1.104
FCPortugal	2.381	2.975	3.331	2.716	3.897

Here we saw another significant drop in performance when comparing Table 3 to Table 2. Kicking provided a large gain in performance, nearly doubling the average goal difference against FCPortugal, compared to only dribbling. This result is in stark contrast to when UT Austin Villa won the 2011 RoboCup competition, in which the team tried to incorporate kicking skills without using an overlapping layered learning approach, and found that kicking actually hurt the performance of the team (MacAlpine et al. 2012b).

5.4 Different Robot Models

At the RoboCup competition teams were given the option of using five different robot types with the requirement that at least three different types of robots must be used on a team and no more than seven of any one type. The five types of robots available were the following:

Type 0: Standard Nao model
Type 1: Longer legs and arms
Type 2: Quicker moving feet
Type 3: Wider hips and longest legs and arms
Type 4: Added toes to foot

We applied our overlapping layered learning approach for learning behaviors to each of the available robot types. Game data from playing against the top three teams at RoboCup 2013 is provided in Table 4 for each robot type.

While there are some differences in performance between the different robot types, likely due to the differences in their body models, all of the robot types are able to reliably beat the top teams from the 2013 RoboCup competition. This shows the efficacy of our overlapping layered learning approach and its ability to generalize to different robot models. During the 2014 competition the UT Austin Villa team used seven type 4 robot models as they showed the best performance, two type 0 robot models as they displayed the best performance on kickoffs, and one each of the type 1 and type 3 robot models as they were the fastest at walking.

6 Related Work

Within RoboCup soccer domains there has been previous work in using layered learning approaches to learn complex agent behaviors. Stone used layered learning to train three behaviors for agents in the RoboCup 2D simulation domain and specified an additional two that could be learned as well (2000). Gustafson et al. used two layers of learning when applying genetic programming to the keep away subtask within the RoboCup 2D simulation domain (2001). Whiteson and Stone later introduced concurrent layered learning within the same keepaway domain during which four layers were learned. Cherubini et al. used layered learning for teaching AIBO robots soccer skills that included six behaviors (2008). Layered learning has also been applied to non-RoboCup domains such as Boolean logic (Jackson and Gibbons 2007) and non-playable characters in video games (Mondesire and Wiegand 2011). To the best of our knowledge our overlapping layered learning approach, containing 19 learned behaviors, has more than three times the behaviors of any previous layered learning systems.

Work by Mondesire has discussed the concept of learned layers overlapping, and focuses on a concern of information needed to perform a subtask being lost or forgotten as it is replaced during the learning of a task in a subsequent layer (2014). Our work differs in that we are not concerned with the performance of individual subtasks in isolation, but instead are interested in maximizing the performance of subtasks when they are combined.

7 Summary and Discussion

This paper introduces and motivates general scenarios for using overlapping layered learning. The paper also includes a detailed description and experimental analysis of the extensive overlapping layered learning approach used by the UT Austin Villa team in winning the 2014 RoboCup 3D simulation competition.[5] Future work in this area includes the automated determination of appropriate subtasks for layered learning as well as automated identification of useful layer overlap or "seams" to use for overlapping layered learning.

References

Behnke, S.; Schreiber, M.; Stückler, J.; Renner, R.; and Strasdat, H. 2006. See, walk, and kick: Humanoid robots start to play soccer. In *Proc. of the Sixth IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids 2006)*, 497–503. IEEE.

Cherubini, A.; Giannone, F.; and Iocchi, L. 2008. Layered learning for a soccer legged robot helped with a 3d simulator. In *RoboCup 2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 385–392.

Depinet, M.; MacAlpine, P.; and Stone, P. 2015. Keyframe sampling, optimization, and behavior integration: Towards long-distance kicking in the robocup 3d simulation league. In *RoboCup-2014: Robot Soccer World Cup XVIII*, Lecture Notes in Artificial Intelligence. Berlin: Springer Verlag.

Gustafson, S. M., and Hsu, W. H. 2001. *Layered learning in genetic programming for a cooperative robot soccer problem.* Springer.

[5]Video highlights at `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/#2014`

Hansen, N. 2009. *The CMA Evolution Strategy: A Tutorial*. `http://www.lri.fr/~hansen/cmatutorial.pdf`.

Jackson, D., and Gibbons, A. 2007. Layered learning in boolean gp problems. In *Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 148–159.

Kalyanakrishnan, S., and Stone, P. 2010. Learning complementary multiagent behaviors: A case study. In *RoboCup 2009: Robot Soccer World Cup XIII*.

MacAlpine, P.; Barrett, S.; Urieli, D.; Vu, V.; and Stone, P. 2012a. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*.

MacAlpine, P.; Urieli, D.; Barrett, S.; Kalyanakrishnan, S.; Barrera, F.; Lopez-Mobilia, A.; Ştiurcă, N.; Vu, V.; and Stone, P. 2012b. UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

MacAlpine, P.; Collins, N.; Lopez-Mobilia, A.; and Stone, P. 2013. UT Austin Villa: RoboCup 2012 3D simulation league champion. In *RoboCup-2012: Robot Soccer World Cup XVI*, Lecture Notes in Artificial Intelligence. Berlin: Springer Verlag.

MacAlpine, P.; Depinet, M.; Liang, J.; and Stone, P. 2015. UT Austin Villa: RoboCup 2014 3D simulation league competition and technical challenge champions. In *RoboCup-2014: Robot Soccer World Cup XVIII*, Lecture Notes in Artificial Intelligence. Berlin: Springer Verlag.

MacAlpine, P.; Price, E.; and Stone, P. 2015. SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning. In *Proc. of the Twenty-Ninth AAAI Conf. on Artificial Intelligence (AAAI)*.

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

Mondesire, S. C. 2014. *COMPLEMENTARY LAYERED LEARNING*. Ph.D. Dissertation, Univ. of Central Florida Orlando, Florida.

Riedmiller, M.; Gabel, T.; Hafner, R.; and Lange, S. 2009. Reinforcement learning for robot soccer. *Autonomous Robots* 27(1):55–73.

Singh, S. P. 1992. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8(3-4):323–339.

Stone, P. 2000. Layered learning in multiagent systems: A winning approach to robotic soccer.

Thain, D.; Tannenbaum, T.; and Livny, M. 2005. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience* 17(2-4):323–356.

Urieli, D.; MacAlpine, P.; Kalyanakrishnan, S.; Bentor, Y.; and Stone, P. 2011. On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, volume 2, 769–776. IFAAMAS.

Whitehead, S.; Karlsson, J.; and Tenenberg, J. 1993. Learning multiple goal behavior via task decomposition and dynamic policy merging. In *Robot learning*. Springer. 45–78.

Whiteson, S., and Stone, P. 2003. Concurrent layered learning. In *Second Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, 193–200. New York, NY: ACM Press.

Whiteson, S.; Kohl, N.; Miikkulainen, R.; and Stone, P. 2005. Evolving soccer keepaway players through task decomposition. *Machine Learning* 59(1-2):5–30.