Chapter 5

Learning an Individual Skill

As presented in Chapter 4, the initial layer in our simulated robotic soccer layered learning implementation is an individual skill, namely ball interception. In this chapter, I describe our ball-interception learning, the initial learning experiments in the RoboCup soccer server [Stone and Veloso 96c, Stone and Veloso 98c]. First, in Sections 5.1 and 5.2, I motivate the choice of ball-interception as our initial learned layer and describe the experimental setup. Then, in Section 5.3, I present our detailed empirical results demonstrating the effectiveness of the learned behavior. Sections 5.4 and 5.5 are devoted to discussion and related work.

As the experiments reported in this chapter were carried out in an earlier version of the soccer server—version 2—there were some differences from the current version 4 as described in Chapter 2. For example, when seeing an object, the agents only received information pertaining to its distance and angle from the agent: never velocity information. I point out this and other differences between server versions when they are relevant to the text.

5.1 Ball Interception in the Soccer Server

Just as young soccer players must learn to control the ball before learning any complex strategies, robotic soccer agents must also acquire low-level skills before exhibiting complex behaviors. In multi-agent systems in general, it is crucial that the agents are individually skilled: the most sophisticated understanding of how to act as part of a team is useless without the ability to execute the necessary individual tasks. Therefore, we defined an individual skill as the lower abstraction level to be learned in the layered learning approach.

The low-level skill we identified as being most essential to our soccer server agents was the ability to intercept a moving ball. This skill is ubiquitous in all soccer-type frameworks. Intercepting a moving ball is considerably more difficult than moving to a stationary ball both because of the ball's unpredictable movement (due to simulator noise) and because the agent may need to turn and move in such a direction that it cannot see the ball, as illustrated in Figure 5.1.

Intercepting a moving ball is a task that arises very frequently in the soccer server. Unless the ball has decelerated completely without an agent collecting it, this skill is a prerequisite for any kicking action. It is used by agents filling all the different roles: goaltenders and defenders must intercept shots and opponents' passes, while midfielders and forwards must frequently "intercept," or receive, passes to them from teammates.

Ball interception is a difficult task in the soccer server because of the noise in the ball's motion and because the agents have limited sensing capabilities. As presented in Section 2.2, each agent has a limited visible angle and it gets sensory information at discrete intervals (`send_step` = 250 msec in soccer server version 2). Often when an agent is trying to intercept the ball, the ball is moving roughly in the direction of the agent, the condition which causes the difficulty illustrated in Figure 5.1. The ball can move past the agent as it goes to where the ball used to be. But if the agent turns to move to where the ball *will* be, it may lose sight of the ball. Since the ball's motion is noisy, the agent cannot predict its motion while it is not visible.

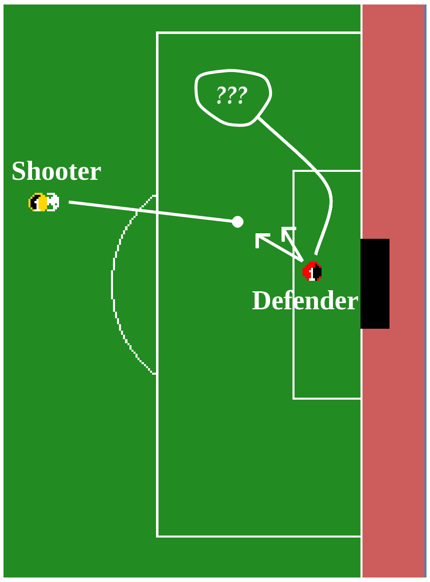


Figure 5.1: The challenge of ball interception. If the defender moves directly towards the ball (left arrow), it will miss entirely. If the defender turns to move in the appropriate direction (right arrow), it may no longer be able to see the ball.

There were two possible methods for equipping our agents with the ability to intercept a moving ball:

Analytically estimating the ball's velocity from its past positions (recall that velocity information was not directly available), and predicting its future motion based on this velocity.

Empirically collecting examples of successful interceptions, and using a supervised learning technique to create a general ball-interception behavior.

We implemented both approaches. For the empirical approach, we used neural networks (NNs) as the supervised learning technique.

As it turns out, the two approaches were roughly equal both in effort of implementation and in effectiveness (see Section 5.3). As the initial learning experiment in our layered learning implementation, we preferred the empirical approach for its appropriateness to our machine learning paradigm. Isolating a situation that requires this skill, we drilled the agents, providing the appropriate reinforcement, until they were able to learn to execute this skill reliably.

5.2 Training

In order to train the ball-interception behavior, we focus on a specific instance of the behavior: a defender blocking a shot from an opponent. The training setup is as follows:

- The *defender* starts at a distance of 4 in front of the center of the goal, facing directly away from the goal.
- The ball and *shooter* are placed randomly at a distance between 20 and 30 from the defender.
- For each training example, the shooter kicks the ball directly towards the defender with maximum power ($Power = 100$).
- The defender's goal is to save the shot. A *save* is a successful ball interception; a *goal* is an unsuccessful attempt in which the ball enters the goal; a *miss* is an unsuccessful attempt in which the ball goes wide of the goal.

Due to the noise in the simulator, the ball does not always move directly at the defender: if the defender remains still, the ball hits it only 35% of the time. Furthermore, if the defender keeps watching the ball and moving directly towards it, it is only able to stop the ball 53% of the time.

The defender's behavior during training is more complex than the shooter's. As we are using a supervised learning technique, it must first gather training data by acting randomly and recording the results of its actions. As its input representation for learning, the defender notices the ball's distance at time t ($BallDist_t$), the ball's relative angle at time t ($BallAng_t$), and the ball's distance at the time when the previous visual information string was received ($BallDist_{t-1}$). As no explicit velocity information is available in version 2 of the soccer server, $BallDist_{t-1}$ serves as an indication of the ball's rate of movement. It would have been possible to base the velocity estimate on several past ball positions. However, we found that a single previous position was sufficient to allow the agent to learn to intercept the ball. Figure 5.2 illustrates the experimental setup, indicating the starting positions of both agents and the ball.

The defender's goal is to determine its turn angle $TurnAng_t$ or the angle it should turn at time t relative to the ball's angle in order to intercept the ball. The defender acts randomly during training according to the following algorithm:

- While $BallDist_t > 14$, Turn($BallAng_t$)
- When $BallDist_t \leq 14$, set $TurnAng_t =$ Random Angle between -45^o and 45^o.

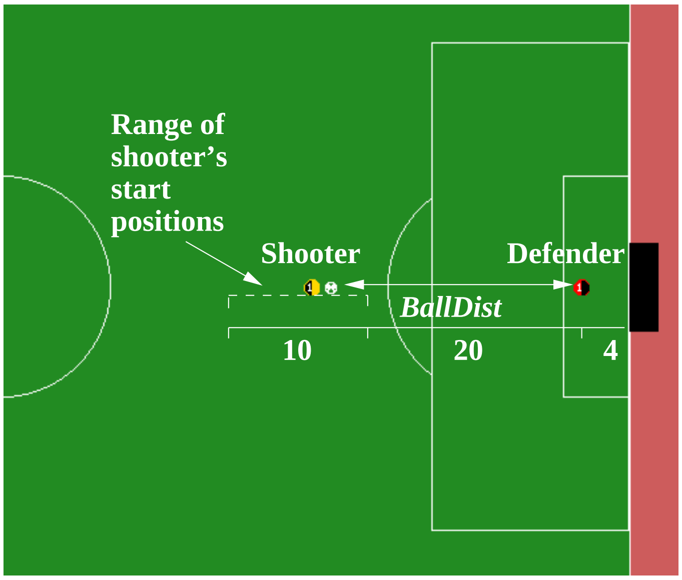


Figure 5.2: The range of training situations for learning ball interception. At the beginning of each trial, the defender starts at a distance of 4 from the goal, while the ball and shooter are placed randomly at a distance between 20 and 30 from the defender. *BallDist* is the distance from the defender to the ball. When the defender is looking straight at the ball, as in this figure, *BallAng* = 0.

- Collect the features of the training instance: $BallDist_t$, $BallAng_t$, $BallDist_{t-1}$, and $TurnAng_t$.
- Turn($BallAng_t$ + $TurnAng_t$).
- Dash forward.
- Gather classification as a successful training instance (save) or an unsuccessful instance (goal or miss).

Until the ball is within a given range (14), the defender simply watches and faces the ball, thus ensuring that all of the world features used for training are known. Then, once the ball is in range, the defender turns a random angle (within a range) away from the ball and dashes. This procedure of gathering data does not produce a successful training instance on every trial: only the saves correspond to the correct mapping from world features to the agent's action. Since the defender acts randomly during training, it often misses the ball (76% of the time). However, it can learn based on the successful training examples.

Our domain is characterized by an agent not always knowing the effects of its actions due to a large amount of hidden state in the world. If the defender is not facing the ball at the end of a trial, it does not know the ball's location, and therefore it does not know whether its interception attempt was successful or not. In order to automate the training process, we use an omniscient, omnipotent centralized agent. The centralized agent classifies each trial as a failure when the ball gets past the defender (a goal or miss) or as a success when the ball starts moving back towards the shooter (a save). Only saves are considered successful training instances and thus used for training. At the end of the trial, the centralized agent resets the positions of the defender, the shooter, and the ball for another trial.

The goal of learning is to allow the defender to choose the appropriate turn angle ($TurnAng_t$) based upon $BallDist_t$, $BallAng_t$, and $BallDist_{t-1}$. In order to learn the $TurnAng_t$, we chose to use a Neural Network (NN). NN's are appropriate for the task because of their ability to learn continuous output values from continuous inputs.

After a small amount of experimentation with different NN configurations, we settled on a fully-connected net with 4 sigmoid hidden units and a learning rate of 10^{-6}. The weights connecting the input and hidden layers used a linearly decreasing weight decay starting at .1%. We used a linear output unit with no weight decay. We trained for 3000 epochs. This configuration proved to be satisfactory for our task with no need for extensive tweaking of the network parameters.

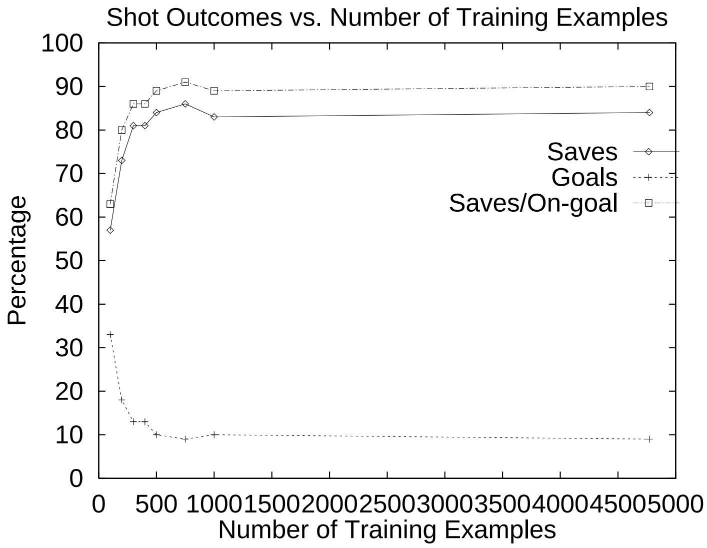
5.3 Results

5.3.1 Number of Training Examples

In order to test the NN's performance, we ran 1000 trials with the defender using the output of the NN to determine its turn angle. The behaviors of the shooter and the centralized agent are the same as during training. Thus, the testing examples are drawn from the same distribution as the training data. The results for NNs trained with different numbers of training examples are displayed in Figure 5.3. The misses are not included in the results since those are the shots that are far enough wide that the defender does not have much chance of even reaching the ball before it is past. The figure also records the percentage of shots *on-goal* (Saves+Goals) that the defender saved. Reasonable performance is achieved with only 300 training examples, and examples beyond about 750 do not improve performance. The defender is able to save almost all of the shots despite the continual noise in the ball's movement.

5.3.2 Effect of Noise in the Simulation

In order to study the effect of noise in the ball's movement upon the defender's performance, we varied the amount of noise in the soccer server (the `ball_rand` parameter). Figure 5.4 shows the effect of varying noise upon the defender when it uses the trained NN (trained with 750 examples) and when it moves straight towards the ball. The default `ball_rand` value in the soccer server is .05. The "straight" behavior always sets $TurnAng$=0, causing the defender to go directly towards where it last saw the ball. Notice that with no ball noise, both the straight and learned behaviors are successful: the ball and the defender move straight towards each other. As the noise in the ball's motion increases, the advantage of using the learned interception behavior becomes significant. The advantage of the NN can also be seen with no noise if the shooter aims slightly wide (by 4 degrees) of the goal's center. Then the defender succeeds 99% of the time when using the NN, and only 10% of the time when moving straight towards the ball.



Training Examples	Saves(%)	Goals(%)	$\frac{\text{Saves}}{\text{Goals+Saves}}$(%)
100	57	33	63
200	73	18	80
300	81	13	86
400	81	13	86
500	84	10	89
750	**86**	9	**91**
1000	83	10	89
4773	84	9	90

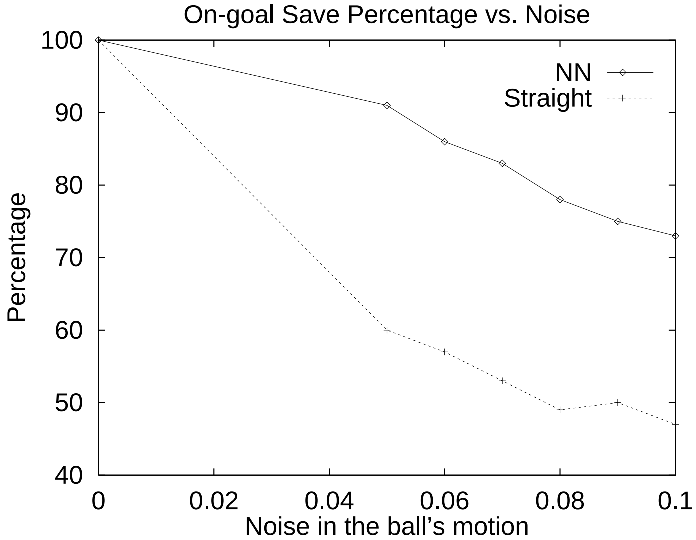
Figure 5.3: The defender's performance when using NNs trained with different numbers of training examples. A "save" is a successful interception of the ball, a "goal" is a failed attempt. The last column of the table indicates the percentage of shots that were "on goal" that the defender saved.

5.3.3 Comparison with other Methods

From an examination of the weights of the trained NN, it is apparent that the NN focuses primarily upon the ball's angle ($BallAng_t$). Consequently, we were curious to try a behavior that simply uses a lookup table mapping $BallAng_t$ to the typical output of the NN for that $BallAng_t$. We identified such outputs for $BallAng_t$ ranging from -7 to 7. Using this one dimensional lookup-table, the defender is able to perform almost as well as when using the full NN (see Table 5.1).

We also were curious about how well the NN would compare to analytical methods. As a basis for comparison, we used a behavior constructed by another student in the project whose goal was to create the best possible analytic behavior[1]. The resulting behavior computes the ball's motion vector from its current and previous positions and multiplies this vector by 3, thus predicting the ball's position two sensory steps (500 msec) into the future. The

[1]We thank Michael Bowling for creating this behavior. Michael Bowling joined the robotic soccer project as a full-time graduate student in 1998. He now focuses on the real robots [Veloso *et al.* 99a].



Noise	Behavior	Saves(%)	Goals(%)	$\frac{\text{Saves}}{\text{Goals+Saves}}$(%)
0	NN	100	0	100
	Straight	100	0	100
.05	NN	**86**	9	**91**
	Straight	**53**	35	**60**
.06	NN	75	13	86
	Straight	47	35	57
.07	NN	68	14	83
	Straight	40	36	53
.08	NN	59	16	78
	Straight	34	36	49
.09	NN	53	17	75
	Straight	32	33	50
.1	NN	49	18	73
	Straight	28	32	47

Figure 5.4: The defender's performance when using NNs and moving straight with different amounts of ball noise. A "save" is a successful interception of the ball, a "goal" is a failed attempt.

defender's $TurnAng_t$ is then the angle necessary to move directly towards the end of the lengthened vector. In particular, if (x_t, y_t) is the ball's current position, and (x_{t-1}, y_{t-1}) was its position at the time of the previous visual input (250 msec in the past), then the agent aims at the point $(x_{t-1}, y_{t-1}) + 3((x_t, y_t) - (x_{t-1}, y_{t-1}))$. Table 5.1 shows the results of the lookup table and analytic methods as compared to the learned NN.

Defender Behavior	Saves(%)	Goals(%)	Saves/(Goals+Saves) (%)
NN	**86**	9	**91**
Lookup Table	83	8	91
Analytic	82	13	86

Table 5.1: The defender's performance when using a NN, a one-dimensional lookup table, and an analytic method to determine the $TurnAng_t$.

5.4 Discussion

In this chapter I presented the initial layer of our layered learning implementation. We used a supervised learning technique to allow agents to learn a useful individual skill in a multi-agent domain. In addition to the successful learned method, we investigated other methods of achieving the same behavior functionality, verifying that the learning approach compares favorably with the other options.

Ball interception is a ubiquitous skill in robotic soccer. Not only our agents in the soccer server, but also all other agents in any real-time robotic soccer domain must be able to intercept the ball. Another ball-interception method for the soccer server that relies on knowing the instantaneous ball velocity is used by the AT Humboldt-97 team [Burkhard *et al.* 98]. It is similar to our own hand-coded ball-interception skill for version 4 of the soccer server described in Appendix B (Section B.1.4). Also in Appendix B (Section B.2.2), there is a description of the ball-interception behavior used by our real robotic agents.

As mentioned above, the learning described in this chapter is done in an early implementation of the soccer server in which agents do not receive any velocity information when seeing the ball. Thus the ball interception skill described in Appendix B is not applicable. Although we eventually hand-coded the ball interception skill for use in the new version of the server, the next learned layer—to be described in Chapter 6—uses the learned ball-interception behavior described in this chapter as part of its training behavior. Thus, the two layers link as advocated by the principles of layered learning.

Also in line with the principles of layered learning, the ball-interception subtask is an important initial skill for which it is possible to gather training data. Within the context of layered learning, the subtask is given as part of the task decomposition. We chose the NN machine learning method in order to exploit data to learn a continuous action parameter from three continuous features of the world state as inputs. Both of these choices—the subtask itself and the ML method—are heuristic choices. Other options may have worked equally well.

5.5 Related Work

Although the learned ball-interception described in this chapter was the first learning research conducted in the soccer server, individual behaviors had been previously learned in other robotic soccer systems. This section discusses these other early robotic soccer learning systems[2].

Prior to conducting the research reported in this chapter, we used neural networks to teach an agent to shoot a moving ball into a goal in a simulator based closely upon the Dynasim system [Sahota 96]. The simulator models the non-holonomic wheeled Dynamite robots [Sahota *et al.* 95]. In this scenario, we were able to use one neural network to time the agent's approach towards the ball, and another to aim the ball. The learned behavior successfully enabled the agent to redirect a moving ball with varying speeds and trajectories into specific parts of the goal. By carefully choosing the input representation to the neural networks so that they would generalize as much as possible, the agent was able to use the learned behavior in all quadrants of the field even though it was trained in a single quadrant [Stone and Veloso 98d].

Learning was applied in both simulated and real robotic, to hit a stationary ball into the goal using the concept of learning from easy missions [Asada *et al.* 94a]. By first placing the ball directly in front of the goal and then gradually moving it farther away, thus making the task harder, the learning time was reduced from exponential to linear in the size of the state space within an RL framework. The robots achieved 70% success in simulation and 40% success in the real world.

Another early example of learning an individual skill in the RoboCup soccer server involved using a neural network to enable an agent to learn when to shoot and when to pass [Matsubara *et al.* 96]. An agent was given the ball near the opponent's goal with a goaltender blocking the way and a teammate nearby. Based on the positions of the ball, the goaltender, and the teammate, the agent with the ball learned when it was best to shoot directly at the goal and when it was best to pass.

In a similar setup, using the same Dynasim-based simulator mentioned above, we used memory-based learning to allow an agent to learn when to shoot and when to pass the ball [Stone and Veloso 96a]. This learning was of an individual skill since the goaltender and the teammate used deterministic, fixed behaviors.

[2] A full discussion of related robotic soccer systems, including more recent learning approaches, appears in Chapter 9 (Section 9.2).