Training a Simulated Soccer Agent to Optimize Goal Scoring

Navid Kaihanirad
Department of Computer Science
University of Toronto
40 St George St, Toronto, ON
navidrad@cs.toronto.edu

Abstract

Scoring goal is the most fundamental objective in a simulated soccer game, likewise it is in its real-world counterpart. However, the very high level of uncertainty involved in this game, makes it impractical to find an optimal scoring policy using prevalent computational methods. This paper, however, presents a new approach toward optimal scoring policy in soccer simulation using machine learning techniques. Considering inherent constraints of soccer game and choosing a learning model which best matches these limitations, we could double accuracy of shooting the ball. This study has also shown that it's possible to have a more manageable source code by simply replacing several parts of code with a proper learning model. Moreover, this approach reduces computational cost significantly.

1 Introduction

RoboCup 2D Soccer Simulation League is the simulated model of the real world game soccer. It provides an environment where eleven autonomous agents, in a simulated soccer game, play against the opponent team to achieve the ultimate goal of winning the game. Here, scoring more goals is the key factor to success. In 2016, Gliders, champion of the year, won the final game vs Helios by scoring one goal more than its rival [1]. This is an example of how any slight improvement in the goal scoring policy can significantly change the results.

Although scoring goal is the main objective in soccer, it is very difficult to have a clear policy indicating when an agent has to attempt to score a goal. This difficulty comes from the fact that chance of scoring a goal is influenced by many factors of which many are unpredictable. The simulation server adds various forms of uncertainty to make the environment as close to the real world situations as possible [2]. Agent's limited and noisy perception of the environment, noise in objects movement, sensors and actuators are just to name a few. Besides various noises added to the environment, agent's action uncertainty should also be taken into account. In every cycle of the game, agent in a given scenario has to decide whether to attempt to score, or pass the ball to a teammate, dribble the opponent or take another action. Not to mention that for all these actions there is a high chance of failure, as well, depending on the scenario.

A common approach to address this problem is to implement several classes for each one of these actions where each class implements a different strictness level for the same action. An example of this is having several implementation of shoot such as "bhv_strict_check_shoot", "body_force_shoot", "shoot", and "shoot_generator" in HELIOS Base, the popular base codes for 2D soccer simulation [13]. Source codes developed with this approach, however, turn to be intricate and very difficult to maintain and develop. Moreover, every cycle of the game is limited to only 0.1 second by the server. This impose a harsh restriction on implementations. The simulation server will not respond to any command after this time frame. To meet this limit, participating teams have to discard many contributing features and conditions in order to keep computations light or they will end up with computation-intensive algorithms which will never get executed on time.

This study, however, propose use of learning models with inherent probabilistic characteristic.

This will enable us to eliminate needs of implementing several classes for each action which is meant to imitate probabilistic approach. Removing several thousand line of code and several classes in this way, not only will help to have a more maintainable source code, but it also reduces computational cost significantly and makes sure all decisions are made within the 0.1s timeframe. Any agent's action –shooting the ball score in this study- capable of being implemented with this approach can potentially benefit from these advantages.

Rest of this paper is structures as follow. In section two, we define the problem. We try to find a solution by breaking it in two sub-problems. In section three we mention how previous works attempted to address this issue. Next, in section four, we explain how machine learning techniques can help to address this problem. Section 5 elaborate on performance and finally section 6 explains about limitations of this work.

2 The optimal scoring problem

Problem of finding the optimal scoring policy can be formulated by the following statement: "*find the point in the goal where the probability of scoring is the highest when the ball is shot to this point in a given situation."*[3]

From the statement above, optimal scoring problem can be divided into two separate sub-problems:

- *Finding the best point in the goal that maximize chance of scoring if the ball shot to this point*
- *Finding the probability of scoring given the current state of the system*

A key point about this formulation of the optimal scoring problem is that both sub-problems are independent. Therefore, we address them independently and with two different approach.

To address the first sub-problem, finding the best point in the goal to aim for, several studies have previously used learning models [6]. Although they have been successful to choose a target point using a trained Neural Network, we address the first sub-problem with computational algorithms rather than learning models. The reason to choose computational algorithms over learning models for this specific problem is deterministic nature of computational algorithm. Knowing that it is highly likely to score a goal at a given moment, computational algorithms are capable of finding the exact point to maximize chance of scoring. Besides knowing that it is most likely that we end up scoring a goal given current state of the environment, it is then possible to implement a light-computation algorithm considering just a few parameters to find the optimal point, and yet very precise results. This algorithm is already implemented as part of the source code of HELIOS Base.

To solve the second sup-problem, however, we use supervised learning techniques. Knowing that, on average, each team has 5-8 opportunities to score a goal in a game, it is critical to save the ball unless it is very likely to score a goal. In fact, choosing another action such as passing the ball, dribbling an opponent or keep moving forward might increase the chance of scoring goal in future cycles of the game. Having a learning model with probabilistic view rather than a deterministic model enables us to choose the proper action with regard to chance of it being executed successfully.

3 Related work

Because of the Importance of scoring, and due to the unique opportunity that soccer simulation provides for researchers to push boundaries of artificial intelligence, many works have been carried out in this area.

One of the very first attempts towards an optimal scoring policy was done by J.R. Kok, R. de Boer and N. Vlassis [3]. Calculating the ball's deviation from its target due to noises imposed by the server, enabled them to precisely answer the first subproblem. This study, has also introduced the probabilistic approach to choose an action. However, there are many effective features which are not considered in this study. A simple example of such features can be an opponent capable of intercepting the ball before it reaches the goal.

A later study has tried to address these deficiencies by taking into account several more features [5]. New features include the angle formed between the line that goes from the ball to the goalkeeper and the line that goes from the ball to the desired point, the strength used by the player to kick the ball, angle that determines the direction of attacker's vision and opponents'

location and ball location, etc. More than thirty-one features have been considered to classify whether the ball will reach the goal or not. Although many of them increase accuracy of the classifier, many are not informatics at all.

Another study, tried to address the optimal scoring problem using Neural Network [6]. This study has proposed to divide the goal width into twenty-one separated regions. A trained neural network, then, determine which region is the best point to shoot the ball to, at each time. This implementation has noticeably increased the chance of scoring a goal by RoboSina's agent. However, this uses an undesired deterministic approach.

Some other work has implemented Q-learning techniques to optimize scoring problem [7]. This study propose to add two parameters, the body angle and the neck angle of the goalkeeper, to the scoring policy of UvA team [8]. This behavior was trained by reinforcement learning algorithm. Z. Zengrong proposed a mathematical approach to provide a probabilistic model and determine the optimal scoring policy [9]. F. Farahnakian and N. Mozayani in another work, applied dimensionality reduction techniques to choose the best features [10]. Finding the most informative features, they have, then, applied the C.45 tree to compare effectiveness of their proposed features.

4 The learning based solution

Our approach to address the optimal scoring policy, comprises of four part which are explained in detail in the following sections.

4.1 Data collection

First step toward finding an optimal scoring policy is to collect labeled data. A common approach to collect labeled data for the training purpose is to extract data from log files generated by the simulation server from the games played in the past. Although this is a common approached used in many previous studies [4], it comes with some strict limitations. First and for most, this approach is limited to the features logged by the server. However, in this work, we are considering new features, which are not being logged by server. Second, teams are evolving every year; agents are developed to be more intelligent and sensitive to their environment. Therefore, predictions based on data from several years ago are not reliable anymore.

In our approach to collect labeled data, we inject some snippets of code to the source to help us log features of interest. These code snippets are included only in the classes which are truly responsible for the desired action, shoot in our case. While this approach has the overhead of recompilation and codding, data collected this way are more precise, clean and reliable. Another issue that needs to be addressed is that reward, goal in this case, happens several cycles after related action is taken and thus it is logged separately. To solve this, we developed a Python application that matches each goal with its corresponding shoot based on the game cycle and label the rest of shoots as not scored.

Number of goals scored during each game varies, however, it barely exceed two for each team. Moreover, considering initialization time and overtime, each game takes around 15 minutes of which major part is spend in middle of the pitch. Sampling several hundred labeled shoots with common approaches take weeks. To overcome this, we initialized unfair games, keeping only opponent's defense line to increase chance of scoring a goal. We also developed a bash script program to automate game initialization process. Having a team of 11 simulated agents playing against a team of only 4 player, for over three hundred times, all automated. We could sample 3047 shoots, of which 1259 successfully scored goal. Table 1 shows details of sampled shots.

Table 1. Statistical evaluation of sampled goal shots

	Valid Games	**Total Samples**	**Scored Shots**	**Not-Scores Shots**
8 Features	89	806	339	467
16 Features	230	2187	920	1267

4.2 Feature selection

Aiming to optimize scoring policy, ball position and goalkeeper's location are key features effecting the chance of scoring goal. However, some other works could increase precision and accuracy of the classifier by adding other useful features [5, 10]. To incorporate previously considered features and still being able to add some new ones that had not been considered so far, we have collected two dataset with different features. First dataset, comprises of eight different features and the second dataset with 16 features. Some features have the exact same definition in both sets, while others provide a different perspective to the same concept. For instance, relative position of ball and goal can be expressed either as two set of (x, y) coordinates or as a pair of distance and angel. We then evaluate which interpretation results in a more accurate classifier. Table 2 and figure 1 demonstrate details of these features.

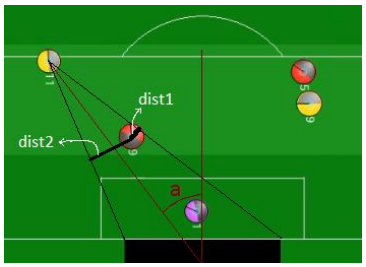

Figure 1: Illustration of some features

Table 2: Features used in two different datasets

	Description	1st Set	2nd Set
Ball Position	(x, y) position of the ball.	*	
Target Point	(x, y) point where the ball is headed to.	*	
Ball.vel	Ball velocity. It's limited to 3 m/s by server	*	
Goalie_line_dist	Distance of opponent goalie from the line formed by ball and its target point	*	
Goalie_ball_dist	Distance of opponent goalie from the ball	*	*
Opp_line_dist_i	Distance of the four closest opponent from the line formed by ball and its target point	*	
Opp_ball_dist_i	Distance of the four closest opponent from the ball	* {1,2,3,4}	* {1,2}
Goal_angle	The ∠a in the figure 1		*
Def_opps	Number of opponents in the ΔBCD in figure 1		*
Dist_ratio_i	Largest two A=$\frac{dist1}{dist2}$ as in figure 1. a measure of how much the opponent is covering the goal		*
Time	Indicate how many cycle has passed since game started. Each cycle is .1s	*	*

4.3 Modeling

Several models are selected to estimate chance of scoring. For each model two classifier have been trained; One to predict labeled data with 16 feature and another classifier for the 8-feature

dataset. Before training process, each dataset is shuffled and then splitted to two batches parts. First batch, including 66.66% of the original dataset, is used to train the model and second batch, which comprises the 33.33% of the rest, is used to test the model independently. Each model's hyper-parameters are tuned in order to produce the highest F1 score.

KNearest Neighbor (KNN) is one of the trained models. To optimize associated hyper-parameters we examined all k in range 1 to 10 and various different distance metrics. We found that K=8 is the optimum k for 8-feature dataset and equally k=5 for the 16-feature dataset. Among different distance metrics also, the Minkowski distance with p hyper-parameter equals to 2 has the best results.

Multi-layer Perceptron, as expected, was among the classifier with highest accuracy. For the 8-feature dataset, trying various structures and different activation functions, it turned out that the network with 4 layer each containing 7, 5, 4, 2 neuron respectively, has the best accuracy. Input layer had 9 neurons including one extra node for bias. For 16-feature dataset, however, a different structure had a better accuracy. A three layer network with 8, 4, 2 neurons in each layer respectively. Input layer had 17 neuron with one extra neuron for bias. Choice of activation function had a huge impact on accuracy rate and "tanh" turned out to be the best activation function for both datasets.

An SVM model with kernel also trained for both datasets. Although few different kernels examined, polynomial kernel with degree10 has the best accuracy for first dataset and a polynomial kernel with degree 5 was best for 16-feature dataset. The regularization parameter, C, is also set to 0.5 and 1.00 for 8-feature dataset and 16-feature dataset respectively.

Two linear classifier are also applied. Logistic regression and a Stochastic Gradient Descent linear classifier. In both cases, L2 loss and L1 regularization performed well. Table 3 compares performance of different models trained on both datasets.

Table 3: Comparison of different models
*S1: 8-Feature dataset. *S2: 16-Feature dataset

		Train Accuracy	Test Accuracy	Precision Score	Recall Score	F1 Score
KNeighbors	*S1	76.29	71.42	70.32	68.76	69.16
	*S2	78.77	69.66	69.19	69.42	69.26
Neural Network	S1	75.55	69.92	69.70	70.54	69.53
	S2	91.26	68.97	68.30	67.78	67.93
Polynomial Kernel	S1	62.03	58.64	48.75	49.70	41.50
	S2	67.23	62.46	64.28	57.80	54.82
Logistic Regression	S1	60.00%	57.51	53.010	52.270	51.003
	S2	67.57	64.54	63.72	63.58	63.63
Decision Tree	S1	100.00	66.54	65.07	65.02	65.04
	S2	100.00	67.86	67.32	67.49	67.38
Linear Classifier With SGD	S1	56.11	59.39	42.46	49.53	38.11
	S2	60.13	58.17	58.19	58.35	57.97
Gaussian Mixture	S1	56.85	60.15	30.07	50.00	37.55
	S2	58.36	57.06	28.53	50.00	36.33

4.4 Empirical experiment

Referring to results obtained in previous section, we chose the 16-feature dataset for implementation since almost every model perform better on this feature set. Moreover, we chose Logistic Regression as our learning model simply because it is the only model satisfies all constraints. Nearest Neighbor is way to slow in prediction phase to meet the .1s timeframe though it is predicting with high accuracy. Neural Network and Decision Tree are deterministic. Among the other models, Logistic Regression not only performs better, but also it has a probabilistic nature, which is well suited for our purpose, and it is extremely fast once it comes to prediction.

An experiment was carried out to compare the results with a learning-enabled soccer team. To have a fair comparison we used the HELIOS Base for both sides. Only our learning-enabled team has the learning model implemented in its shoot action. All the rest of agent' actions are the same for both side. We also used the same approach to collect data as explained in section 3.1 of this paper, data collection. Over hundred games, where scheduled automatically using a bash script program of which 89 had valid outcome. Results are then compared with the outcome of games played between same two teams without anyone having leaning model included. Source code and a few demos of this experiment are all available in GitHub [14]. Table 4 shows the empirical results of this experiment.

The empirical results show a significant improvement in saving the ball by not attempting to score when it is unlikely to succeed. Shoot accuracy, ratio of scored attempts to not-scored attempts, has a 97.2% increase meaning that attempting to score is now much more successful. Average score per game, however, is decreased by a factor of 25%. Many factors are contributing in reducing average score per game. One is the 47% reduction in total attempts to score. However, in section 6, limitation of our work, we have discussed about this in more details.

Table4: Empirical results of a learned soccer team

	Total Games	Total Attempts	Scored	Not-Scored	$\frac{Scored}{Total\ Attempts}$	$\frac{Scored}{Not-Scored}$
Base Team	319	2993	1259	1734	0.42	0.72
Learned Team	89	443	260	183	0.58	1.42

5 System evaluation

Many of the features that we used in this study have been used in previous studies as well. Features such as "goalie_distance_to_ball", "goal_angel", "ball_velocity" are among the most effectively distinguishing features used in almost all efforts to determine optimum scoring policy. However, analyze of data show these features are not informative enough to train a highly accurate model. Plotting density graph for all examined features in this study show that major portion of each variable's range cannot be used to effectively distinguish between classes. Figures 2 and 3 demonstrate density plot of the two most informative features. Data visualization using t-SNE and PCA techniques also lead to the same conclusion. Although there are some trends in data, but there is no clear separation between instances which scored a goal and those which not. Figures 4, 5 are the t-SNE and PCA plots for the 8-feature dataset.

Part of this vague separation is because of very noisy data. As mentioned earlier in introduction section, the simulation server imposes various type of noises. Therefore, neither the ball nor the players or other objects will always move in the direction expected. Another reason is limited perception of agents from their environment in order to make the soccer simulation as close to the real world game as possible. Therefore, agent's information about distances, position of the opponents, teammates, ball, etc., are not always accurate. Thus, scenarios like when an agent attempt to score because he has not seen an opponent blocking the goal is well possible. Recurring of these samples can cause unclear separation of classes.

Next issue in this domain is the correlation of features. Goal-keeper determine the best position to cover the goal with respect to his other teammates. Ball velocity decrease as the distance increase due to negative acceleration. Ball target point is determined with respect to all opponents

coordinates including goalkeeper. These are just a few examples of how these features are tied to each other. Tight correlation of features result in poor accuracy of learning models.

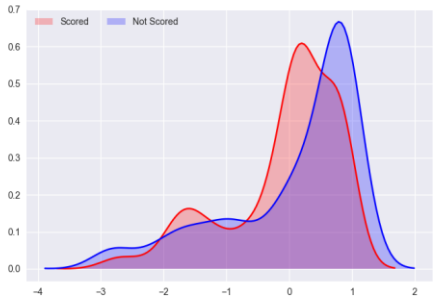

Figure 2: Density plot for Goal_angle

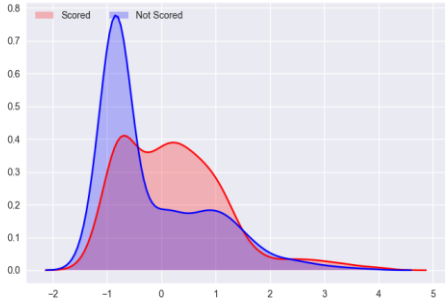

Figure 3: Density plot for Goalie_line_dist

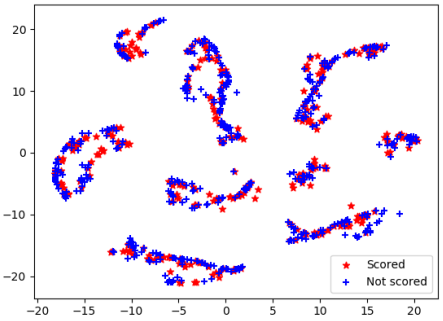

Figure 4: t-SNE visualization of dataset

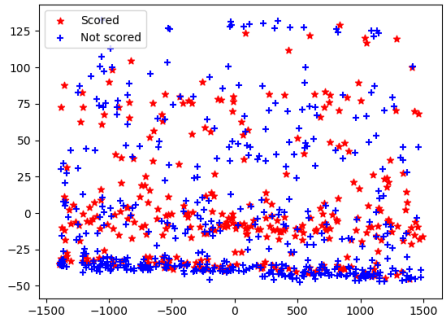

Figure 5: PCA visualization of dataset

6 Limitation

In a multi-agent system like soccer simulation, it is not possible to achieve a goal unless all agents are behaving in harmony. Not only agents should take actions according to one another, but also each agent's action should be well defined with regard to other actions. In our approach to optimize scoring policy, it only happens if other players are aware of this new policy and behave accordingly. Not only other agents should behave accordingly to this new policy but also one's actions should also change to match new behavior. Suppose a scenario where the player is in the penalty area and chance of scoring a goal is very low, keeping ball and not attempting to score will be helpful only if other teammates move to a position where they can receive a pass. Also the pass action should be implemented so that agent will pass the ball to the teammate who is located in the best position.

In our current implementation, coordination between teammates and between agent's actions are not yet applied. Although an agent is now capable of recognizing scenarios where scoring a goal not likely, neither he nor his teammates take a proper action. In such situations, agent correctly decides not to attempt to score a goal. On the other hand, now, teammates do not move to a position where they can receive a pass. At the moment, the pass action is also not implemented accurate enough for short distance passes in a congested areas like the penalty area. Therefore, in current implementation, saving ball in a situation that it is unlikely to score a goal, does not end up more scores. In contrasted, it usually end up losing the ball.

To address this issue, not only agent's behavior, basic move, pass, and all other actions need to be re-implemented, but also same probabilistic approach as the one we applied to the shoot action, should be applied to all other actions. Having a probabilistic implementation for all actions will enable us to develop an optimal decision making algorithm based on chance of successful execution of each action.

Finally, lack of powerful machine learning libraries with support for in C++ language is a strong barrier in implementing complex models for soccer simulation. A powerful library like “scikit-learn” does not provide a C++ API. Other libraries like TensorFlow, is another powerful machine-learning library by Google, provides C++API for some core learning models. However, any model developed with TensorFlow C++API has to be compiled with Google’s made compiler named Bazel [11]. On the other hand, source code developed for the simulation game are only compatible with GCC and not Bazel. Thus unable to use TensorFlow in this project. Shogun is another machine-learning library for C++ [12]. Shogun, however, does not have an active community neither a strong documentation and therefore it is very difficult to integrate it with any project.

7 Conclusion

In this work, we have proposed a new approach to address optimal scoring policy, which is a common challenge in the RoboCup Soccer 2D Simulation. Combining machine-learning techniques with computational algorithms, we have been able to find the optimal time and best point in the goal to maximize chance of scoring. We have been able to increase agent’s shoot accuracy by a factor of 97.2%. We have also been able to achieve a more maintainable source code by replacing major part of computational algorithms used to define agent’s shoot action with a more efficient learning model. Moreover, changing computational-intensive algorithms with an extremely fast learning model like logistic regression, agents are now more responsive in the predetermined timeframe of 0.1s.

Further improvements, however, can be made by applying probabilistic learning models to other actions such as ball passing, basic move, dribble, etc. Finding an optimum threshold to switch between possible actions base on each one’s chance of successful execution is still an unanswered challenge, which needs to be address.

Acknowledgments

This work is possible due to University of Toronto, and Department of Computer Science. I am also greatly thankful to Professors Ethan Fetaya, Emad Andrews and James Lucas for accepting to supervise this project and conducting this study.

References

[1] https://en.wikipedia.org/wiki/RoboCup_2D_Soccer_Simulation_League

[2] Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang and Xiang Yin. “Users Manual, RoboCup Soccer Server, for Soccer Server Version 7.07 and later”. February 11, 2003

[3] J.R. Kok, R. de Boer and N. Vlassis, "Towards an optimal scoring policy for simulated soccer agents" Proceedings of the International Conference on Intelligent Autonomous Systems, pp. 195-198, 2002.

[4] Achim Rettinger. “Learning from recorded games: A scoring policy for simulated soccer agents”. Proceedings of the ECAI 2004, 16th European Conference on Artificial Intelligence, Workshop 8: Agents in dynamic and real-time environments, August 2004.

[5] R. M. Oliveira, P. J. L. Adeodato, A. G. Carvalho, I. B. V. Silva, C. D. A. Daniel,and T. I. Ren, "A data mining approach to solve the goal scoring problem,” International Joint Conference on Neural Networks - IJCNN, 2009.

[6] M. H. Dezfoulian, N. Kaviani, A. Nikanjam, and M. Rafaie-Jokandan. “Training a Simulated Soccer Agent how to Shoot using Artificial Neural Networks”. Proceeding of the Iranian Researchers Conference in in Europe, 2005.

[7] Azam Rabiee, Nasser Ghasem-Aghaee, “A Scoring Policy for Simulated Soccer Agents Using

Reinforcement Learning". 2nd International Conference on Autonomous Robots and Agents December 13-15, 2004 Palmerston North, New Zealand

[8] Kok J., Vlassis N. & Groen F., "UvA Trilearn 2003 Team Description", Faculty of Science, University of Amsterdam, 2003.

[9] Zhao Zengrong, "How to Determine the Optimal Scoring Policy", Fourth International Conference on Intelligent Computation Technology and Automation, 2011.

[10] Fahimeh Farahnakian ,Nasser Mozayani, "Evaluating Feature Selection Techniques in Simulated Soccer Multi Agents System", International Conference on Advanced Computer Control

[11] https://www.tensorflow.org/api_guides/cc/guide

[12] http://www.shogun-toolbox.org/

[13] Akiyama H., Nakashima T. (2014) HELIOS Base: An Open Source Package for the RoboCup Soccer 2D Simulation. In: Behnke S., Veloso M., Visser A., Xiong R. (eds) RoboCup 2013: Robot World Cup XVII. RoboCup 2013. Lecture Notes in Computer Science, vol 8371. Springer, Berlin, Heidelberg9-5262.

[14] https://github.com/rad-navid/learned-agent2d