Team Modeling with Deep Behavioral Cloning for the RoboCup 2D Soccer Simulation League

Felipe V. Coimbra and Marcos R. O. A. Máximo

***Abstract*—Soccer is still considered an open problem by the AI community due to its complex stochastic real-time multiagent nature. The RoboCup Soccer Simulation 2D League has been used as a testbed for new ideas and techniques for many subjects, including team modeling. However, even though team modeling has been an indispensable part of the best league participants to date, in practice it typically consists of ad-hoc heuristics encoded as rules. This requires time-consuming manual work, does not scale to multiple teams, and does not work well at unaccounted scenarios. This paper presents a data-driven method for modeling teams by training Deep Neural Networks with large amounts of data with an Imitation Learning formulation. We demonstrate the approach by training a deep model of the Japanese team Helios using $57{,}578{,}668$ state-action pairs of players. The resulting model achieved 84.5% accuracy on action selection and a small error on regression of the action parameters. The network is shown to be an effective movement predictor of Helios field players and have negligible degradation when Helios is evaluated against adversaries not seen at training time.**

***Index Terms*—Team Modeling, Deep Learning, Imitation Learning, RoboCup, Robot Soccer**

I. Introduction

Robot Soccer is a complex multiagent game where agents must work cooperatively against adversaries to win. It remains an unsolved problem both in the real world and simulated environments. Soccer players typically face noisy and incomplete localized information about the world and must act in real-time under stochastic settings and limited communication bandwidth. For this reason, it has served as an important testbed for new developments in Robotics, Control Systems, and countless subfields in AI.

The Robot Soccer World Cup (RoboCup) is a scientific community and robotics competition founded in 1996 that aims to advance, promote and popularize robotics and AI research. Its mission is that "By the middle of 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup." [1].

The RoboCup Soccer Simulation 2D (RCSS2D) League was RoboCup's first research environment (Fig. 1) and the one we will explore in this work. In RCSS2D, autonomous agents are virtual players that act, observe, and communicate in a 2D world by means of a central simulator server. Sensorial information is local and unique to each player, besides being discretized, incomplete, and stochastically served. However, there is also a special information mode available, called *Fullstate*, in which teams receive perfect and complete information about all players and the ball. The simulator also allows accelerating matches by means of a *Synchronous* mode.

Felipe V. Coimbra and Marcos R. O. A. Máximo are with the Autonomous Computational Systems Laboratory, Instituto Tecnológico de Aeronáutica, e-mail:felipecoimbra97@gmail.com.

Fig. 1. Typical visualization of a RCSS2D match.

Players can take several actions, transmitted via server commands. Some minor commands can be issued multiple times per turn, but we will consider action commands that are sent once per unit time. For the purposes of this work, we are concerned with four types, from which the player must pick one:

1) ***dash***: linearly accelerates the issuing player with a given *dash power* in a given *dash direction*.
2) ***turn***: rotates the issuing player's body by a given *turn moment* angle. The rotation movement has zero-order dynamics (no rotational velocity), so the command changes the body's direction immediately.
3) ***kick***: allows a player to kick the ball with given *kick power* and *kick direction*.
4) ***tackle***: allows a player to clear the ball away in a given *tackle direction*. The tackling power is not controllable. Mainly used for stealing the ball from opponents as it can be used farther from the ball.

Because these high-level server commands abstract low-level actuators, work on RCSS2D has traditionally happened in a higher-level space. Some commonly pursued research topics include building models of individual and collective behavior, team strategies, planning algorithms, etc [2]. In particular, agent models have been extensively used offline and online in the RCSS2D research space and have been an essential component of competitive teams [3].

The typical procedure for obtaining agent models involves manually writing code to recognize patterns and hardwire policies. Although simple and effective at narrow contexts,

it is not trivial to encode individual and collective behavior in this manner. As the league evolves, it becomes harder to capture more complex interactions with codified rules. At the same time, existent hand-written logic is typically too brittle to adapt to newly seen behavior and rewrites take time.

A widely investigated alternative has been around the use of data-driven paradigms. Here, agent models leverage data directly, extracting and storing identified patterns algorithmically to later use them for inference. This has grown in popularity in the RCSS2D League with advances in Machine Learning (ML) techniques, such as Neural Networks [4]. The data-driven approach is very suitable to simulated environments where it's typically easier to collect data. A well regularized ML team model displays adaptation when the team is subject to unaccounted situations. Also, a single training pipeline can be reused to obtain a model of any team.

Some studies have also applied Decision Trees [5]–[7] and Case-Based Reasoning [8]–[11] for a range of problems such as coach modeling and player policy reconstruction. More modern approaches have been inspired by the Deep Learning (DL) framework, applying it for use cases such as improving tree search heuristics used for online planning [12], [13].

The Machine Learning field of Imitation Learning (IL), has been extensively employed for building models in Robotic Soccer [14]–[17]. In IL, the target objective is learning behavior from an expert's demonstrations. Within this space, the Behavioral Cloning (BC) technique consists of learning a direct mapping of states to expert actions. It formulates the problem as a straight application of Supervised Learning with the expert's examples being the supervised dataset [18].

The combination of the IL formulation with the DL framework has led to substantially good results. Deep LSTMs have been used for imitating an agent in a simplified offensive scenario [19], [20] and Deep Feedforward Networks have been used to learn defensive positioning and further improve it with Reinforcement Learning (RL) [21].

The investigations of this work embody the use of BC to teach a Deep Neural Network to imitate the field players of a RCSS2D team. The Neural Network (NN) receives the raw world state as input and outputs both the actions the players should take and the parameters of those actions. We focus on modeling field players, as we believe goalkeepers have very specific behavior (e.g. special actions) and deserve separate treatment.

The novel contributions of this work are:

- The proposal of a reproducible pipeline for generating datasets and building deep imitation models of RCSS2D teams
- An appropriate NN architecture for this task, with suggested parameters and baseline results
- A quantitative evaluation of the approach's generalization capabilities

Our methodology differs from previous work by:

- Learning the agent model using fully featured RCSS2D matches, instead of limited scenarios [19], [20], [22].
- Choosing to represent the model with a network of multiple layers trained with tens of millions of data points – a scale not reported in the literature yet to the best of our knowledge.
- Proposing a NN architecture designed with the nature of the task as well as RCSS2D environment constraints in mind [23].

We organize our work as follows: Section II explains our dataset generation pipeline, the BC state and action spaces, the NN architecture, and the training procedure; Section III characterizes the generated dataset and shows the results of the proposed NN; Section IV concludes our work with a brief recapitulation and possible directions. Readers can find more details on the grounding of our proposal in [23].

II. Methodology

We use a dataset of observed states and the corresponding actions taken, as typical of BC. Once built, the dataset can be used for training a NN via typical supervised learning procedures. This section explains the dataset generation, our proposed NN architecture, and the training setup. All code is openly available, including a game log parser [24], a command log parser [25], and the pipeline and training logic [26].

A. Dataset and Data Pipeline

Our main dataset was built from 1400 matches of the Helios2019 team released at RoboCup 2019 [27] against 8 different opponent teams from the RoboCup 2019 and Latin American Robotics Competition (LARC) 2020 competitions. Each Helios' opponent played 200 matches with 6000 game cycles each, generating about 9.2M raw frames (simulator states) and 83.3M commands from Helios players.

The version 16.0.0 of the RCSS2D simulator was used with *Synchronous* mode turned on and *Fullstate* information given to both sides of the field. At every match, Helios2019 played on the right side of the field without loss of generality.

Game and command logs were processed by an automated pipeline as shown in Fig. 2. Feature extraction removed data not useful for this work, such as information about other commands and sensors. Domain normalization projected each extracted field into a $[-1, 1]$ interval. The SQL ingestion condensed all matches into a common schema (described in [23]), making it easier to clean, shuffle, and bind game states to actions. Data cleaning of the SQL database removed all game states that did not represent normal gameplay (such as corner-kick and offside) and all commands issued by non-Helios players or goalkeepers.

In the end of the pipeline, the prepared data was split into a training dataset with $57{,}578{,}668$ data points plus a dataset for validation and testing with $3{,}030{,}372$ state-action pairs. Each data point consisted of 119 input features and 7 outputs, as summarized in Tables I and II.

The input features define the BC state space, an aggregation of static and dynamic information on the ball and all players. Because all players are represented by the same NN, the redundant information on the command issuer is necessary to specify to the network which player it should be predicting. We assume the Markov Property for team modeling, which in

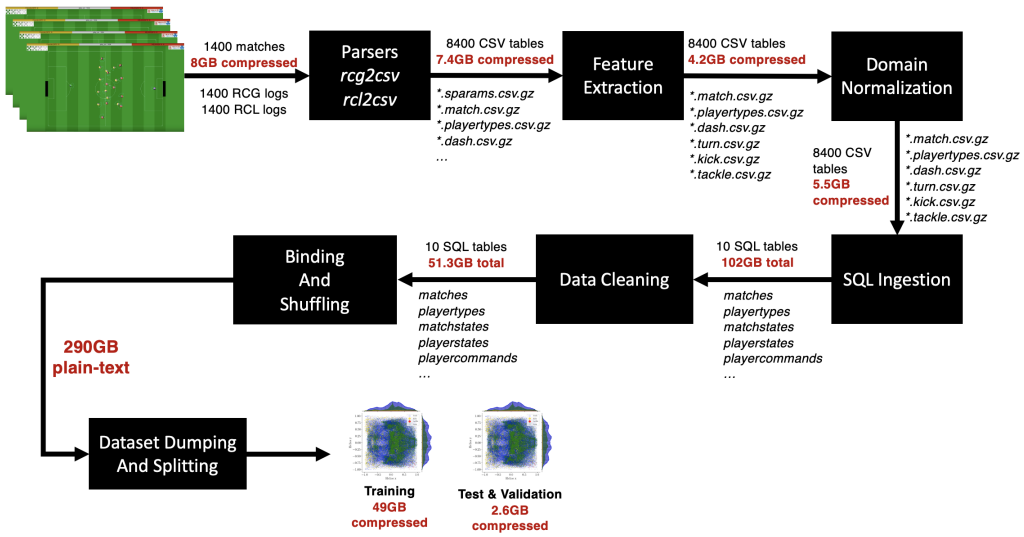


Fig. 2. Pipeline stages in the preparation of a dataset.

TABLE I
INPUT FEATURES USED AT TRAINING.

Object	Feature	Quantity	Description
ball	x	1	Ball's x-coordinate
ball	y	1	Ball's y-coordinate
ball	vx	1	Ball's velocity x-coordinate
ball	vy	1	Ball's velocity y-coordinate
player	x	22	X-coordinate of every player
player	y	22	Y-coordinate of every player
player	body	22	Body orientation of every player
player	vx	22	Velocity's X-coordinate of every player
player	vy	22	Velocity's Y-coordinate of every player
player	x	1	X-coordinate of command issuer
player	y	1	Y-coordinate of command issuer
player	body	1	Body orientation of command issuer
player	vx	1	Velocity's X-coordinate of command issuer
player	vy	1	Velocity's Y-coordinate of command issuer

TABLE II
OUTPUT COLUMNS USED AT TRAINING.

Output	Domain	Description
comm_type	$\{dash, turn, kick, tackle\}$	Command issued
dash_power	$[-1, 1]$	Dash power
dash_dir	$[-1, 1]$	Dash direction
turn_mom	$[-1, 1]$	Turn moment
kick_power	$[-1, 1]$	Kick power
kick_dir	$[-1, 1]$	Kick direction
tackle_dir	$[-1, 1]$	Tackle direction

BC terms means that it is enough to only include information from the latest cycle in our state formulation [23].

The outputs, on the other hand, define the BC action space. Although the action space has dimensions for all action parameters, the actual meaningful dimensions are determined by the value of *comm_type*. The remaining dimensions indicate other less likely predictions.

One last dataset was created via the same pipeline to be used for evaluating the final trained network on out-of-distribution data. It contained 60 matches of Helios2019 against 6 unseen teams that also participated in RoboCup2019. Each adversary contributed equally to the dataset with 10 matches. The 60 matches generated $2{,}486{,}573$ state-action data points after being processed by our pipeline.

B. Neural Network

The NN was designed to decide what action a single agent should take and the values for the action's parameters. By using a single NN to imitate any Helios field player, we assume that a single network can represent any field player. In other terms, we assume players' actions are independent from their roles given the current world state.

This approach allows the network to leverage 10 times more data than it would if every player was imitated by a different NN. Another way of seeing this design decision is by considering it as training 10 neural networks but keeping a strict weight-sharing constraint between all of them. By predicting one player at a time, the network can model the complete team after 10 predictions or a subset of it with fewer forward passes.

We further extend this architecture to force weight-sharing between layers that predict which action to take and what action parameters to use. The final result is illustrated by Fig. 3 as a NN with 2 output layers (two-head NN). There is some experimental indication that this architecture improves action selection accuracy [23].

A *softmax* layer is used for the classification head, which is

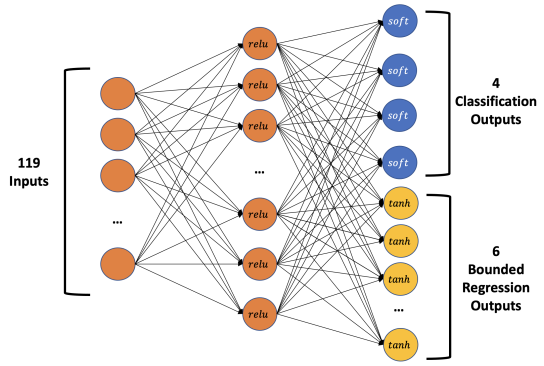


Fig. 3. Single-layer illustration of the Neural Network used in this work.

TABLE III
HYPERPARAMETERS USED DURING TRAINING.

Hyperparameter	**Value**
Number of Epochs	400
Batches per Epoch	300
Batches per Validation	200
Batch Size	4096
Optimizer	Adam
LearningRate	0.001
β_1	0.9
β_2	0.999
Hidden Activation	*relu*
Architecture	$[127, 512, 256, 128, (4, 6)]$
Input Normalization Layer	Yes
Classification Output Encoding	1-hot
Regression Output Encoding	*tanh*-bounded
Classification Loss Function	Categorical Cross-entropy
Regression Loss Function	Mean Squared Error

adequate given that players can only choose one of the actions considered in this work at each decision cycle. Because action parameters have bounded domains, we use *tanh* activation functions at the regression output layer to automatically limit values to the $[-1, 1]$ domain.

Although forward propagation generates parameter predictions for all actions, the truly relevant regression outputs are the ones linked to the command type with highest *softmax* output – the truly predicted command. The remaining regressed values suggested parameters for the remaining command options. The *softmax* values for those options express the probability of them being followed as other alternatives.

Classification output data is 1-hot encoded into 4-dimensional arrays. Regression values for parameters used in actions not issued at a state-action data point are set to zero. Table III shows the values of the hyperparameters selected for the network and the training procedure.

Since the dataset was quite large, considering an epoch to be a full pass would take very long and result in few validation checkpoints, hindering model selection. Therefore, we randomly shuffled and sliced the complete training dataset and limited each epoch to a full pass over a single slice. At the start of a new epoch, a new slice gets picked in a cyclic round-robin fashion. The slice is shuffled and then used to generate batches of data points for gradient updates until the slice was depleted. At the end of each epoch, we ran a validation pass on the network with a smaller number of batches, as validation should only use enough data to attain low errors and fluctuations.

The number of hidden layers and hidden units at each layer were defined after an iteration-bounded beam search on powers of 2 that optimized classification accuracy. More details can be found at [23]. The training error was defined as the sum of the heads' errors without any weighting.

III. RESULTS

In this section, we discuss characteristics of our training dataset: the final output of our data generation pipeline. We also exhibit our trained model's performance results and describe how it could be used to support decision-making. The trained model and the code used to evaluate the model are openly available [26].

A. Data Characterization

The generated dataset captured a good picture of Helios, showing good state space coverage. This can be seen in Fig. 4 for position coordinates through a sample of 1,000,000 commands issued by different Helios' players across the field. Nonetheless, there is a clear class imbalance between the different action types, as shown by Fig. 5, with a clear majority of *dash* and *turn* actions. This is expected because, at every cycle, at most one player tends to be close enough to the ball to either *kick* or *tackle* it. Also, both commands are mostly executed to push the ball away from the holder (dribbling, passing, or shooting).

Furthermore, raw match data has also presented semantic defects. Fig. 6 shows the distribution of *turn_mom* parameters used in *turn* commands. There is a sky-high number of *turn* commands issued with a zero value, which is essentially equivalent to doing nothing.

B. Model Results

Training and evaluation were conducted on a single local machine with a 2.6 GHz 6-Core Intel Core i7 9750H and 16 GB 2667 MHz DDR4 RAM. We used the Tensorflow 2 library to train and evaluate models. No GPU acceleration was used, but Tensorflow made use of AVX2 instructions available. Training sessions for hyperparameter tuning were 2 hours long, but the final reported model was allowed to train for 24h.

The final model obtained 84.5% accuracy on action selection and a mean absolute error of 0.069 on parameter regression. For visualization, this absolute regression error would be equivalent to about 7 units of dash or kick power and a deviation of 12° in dash or kick directions. This estimation is worst-case, assuming all regression error concentrates on the selected action, indicating that the real error should be lower.

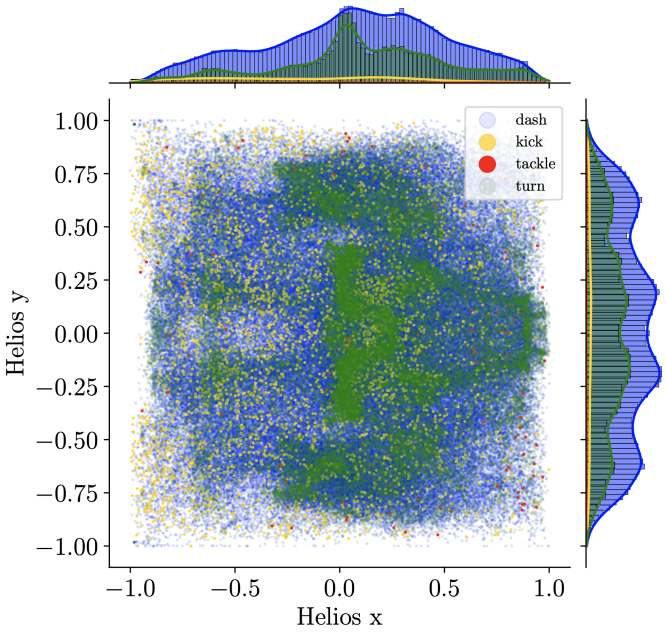


Fig. 4. Different actions issued across the field, colored by type.

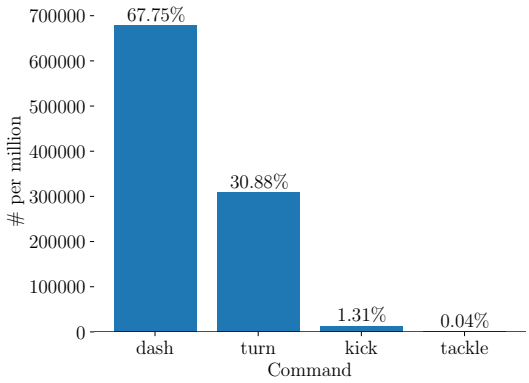


Fig. 5. Distribution of issued commands by type.

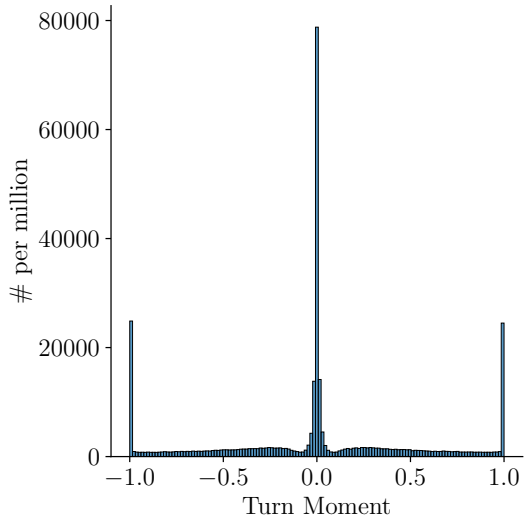


Fig. 6. Histogram of *turn moments* used by Helios field players.

As shown by Fig. 7, the model displayed no signs of overfitting during training. On the contrary, it seems longer training could unlock further improvements, which was not possible due to resource usage limitations. This could be achieved with more computational power, which was not available during our experiments. Each NN head achieved different error regimes by the end of the procedure: there are signs of convergence for the regression target, whereas the network might be underfitting for the classification output.

Table IV breaks the classification results down to each action. The *scaled accuracy* is calculated by dividing the accuracy by the fraction of the same action in the dataset. The *scaled accuracy* can be used to measure the room for improvement at each action – the closer to 100%, the less room exists for improving accuracy by improving that particular action.

TABLE IV
CLASSIFICATION METRICS FOR EACH COMMAND TYPE. THE MOST IMPORTANT RESULTS ARE HIGHLIGHTED IN BOLD.

Metric	**Dash**	**Turn**	**Kick**	**Tackle**
Accuracy	**62.63%**	20.91%	1.00%	0.0084%
Scaled Accuracy	92.44%	**67.71%**	76.34%	21%
Precision	85.60%	80.82%	77.64%	53.36%
Recall	**92.26%**	**66.59%**	**6.21%**	**0.054%**
Specificity	68.25%	92.77%	99.66%	99.99%

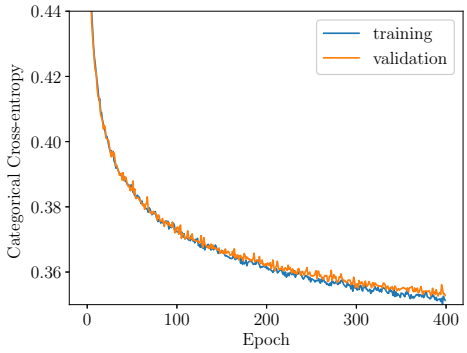
Table IV shows that the accuracy is greatly dominated by *dash* and *turn* commands. The low *scaled accuracy* of *turn* commands might be a result of the dataset defects previously discussed, which presents a clear path for improvement. The low recall for *kick* and *tackle* actions makes the model better suited for movement prediction only, instead of a full agent imitation. This is a consequence of training the network with a highly imbalanced dataset.

This imbalance is a natural characteristic of the underlying distribution of commands, and is not related to our Deep Behavioral Cloning formulation in specific. Any data-driven approach for modeling will need to deal with the difficulties that originate from the distribution's skewness. These difficulties can be counteracted with techniques such as class oversampling, class undersampling, data augmentation, and cost function biasing, but also by architecture adjustments, such as employing an ensemble [28].

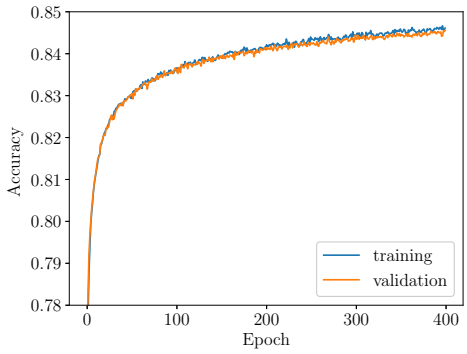
When evaluated against unseen teams, the model showed almost no degradation in both classification and regression outputs. Fig. 8 shows a comparison between the evaluation against teams in the test dataset and outside the training distribution. This result indicates that the final model learned to imitate Helios well enough to generalize at new scenarios against new styles of play.

Figure 9 shows 20,000 samples of the network's response time for a single prediction for random input. The NN takes on average 1.4 ms to output the action and its parameters with a standard deviation of 0.2 ms.

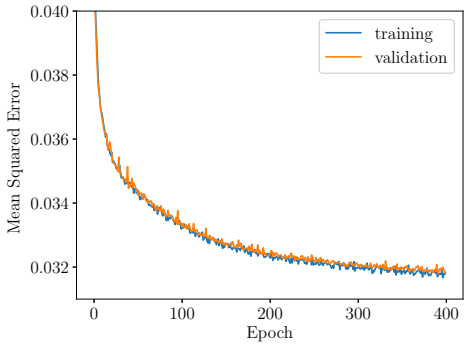
Official RCSS2D matches are played with 100 ms cycles and are ran on commodity hardware. Considering that half



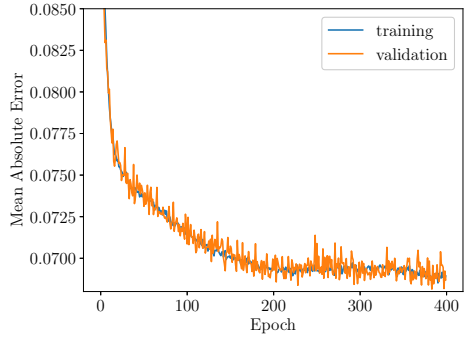
(a) Categorical Cross-entropy Loss



(b) Classification Accuracy



(c) Mean Squared Regression Error



(d) Mean Absolute Regression Error

Fig. 7. The evolution of classification and regression metrics during training.

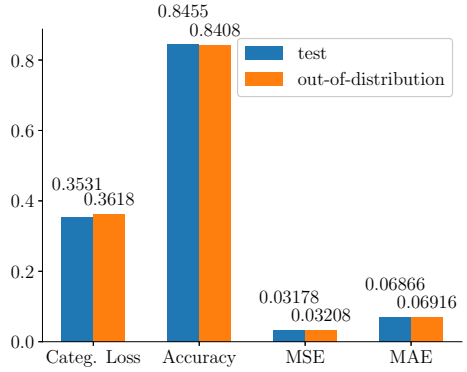


Fig. 8. Comparison of model evaluation against the test dataset and the out-of-distribution dataset.

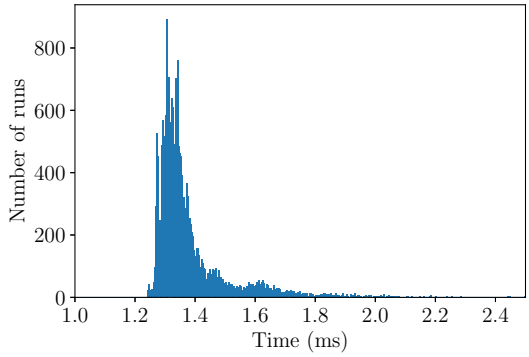


Fig. 9. Histogram of 20,000 measurements of the Neural Network's prediction runtimes.

the time is spent on decision-making, it would be possible to input the network with at least 35 states at every cycle. This is fast enough to represent the actions of an opponent team completely at every frame during online play.

The model could also be used to support online planning in more specific scenarios that do not require predicting all players, such as zone-marking or pressuring the ball holder. In these situations, the model need only predict the movement of free opponent players and of the opponent ball holder, a scenario where it has demonstrated good results.

IV. Conclusions

Encoding team models with hardcoded rules in complex multiagent scenarios such as robotic soccer is error-prone, requires greats amounts of work, and usually fails to generalize when the team plays against different opponents. In this work, we proposed replacing this approach with a data-driven paradigm. We demonstrated the construction of an end-to-end data preparation pipeline and the use of the generated datasets to build a Deep Neural Network that imitates the behavior of any field player, namely its action selection and action parameter regression. The trained network can be used to model any subset of team agents.

Our results show that the model can imitate action selection with an accuracy of 84.5%, although excelling mostly on movement prediction. Combined with an average runtime of 1.5 ms per prediction, it is possible to use the obtained

model for online inference. We also point clear improvement directions by using dataset class-balancing techniques, more rigorous action data cleaning, and longer training. The model showed strong generalization capabilities by having negligible performance degradation when facing teams not used during training.

Future directions of our work include using models obtained with the presented pipeline for online opponent modeling, imitating multi-step behavior, and improving training in the presence of imbalanced datasets.

ACKNOWLEDGMENTS

We thank all collaborators and colleagues of ITAndroids and the Autonomous Computational Systems Laboratory for the longstanding support and fruitful collaboration. We also thank the sponsors of the ITAndroids team for backing this research: ITAndroids: Altium, Cenic, Intel, ITAEx, Mathworks, Metinjo, Micropress, Neofield, Polimold, Rapid, Solidworks, STMicroelectronics, WildLife, and Virtual Pyxis.

REFERENCES

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: the robot world cup initiative," *Proceedings of the International Conference on Autonomous Agents*, 04 1998.

[2] S. V. Albrecht and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems," *Artificial Intelligence*, vol. 258, pp. 66–95, 2018.

[3] S. Pourmehr and C. Dadkhah, "An overview on opponent modeling in robocup soccer simulation 2d," in *RoboCup 2011: Robot Soccer World Cup XV* (T. Röfer, N. M. Mayer, J. Savage, and U. Saranlı, eds.), (Berlin, Heidelberg), pp. 402–414, Springer Berlin Heidelberg, 2012.

[4] W. Plant and G. Schaefer, "An overview of neural networks in simulation soccer," *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 1620–1625, 2009.

[5] P. Riley and M. Veloso, *On Behavior Classification in Adversarial Environments*, pp. 371–380. Tokyo: Springer Japan, 2000.

[6] A. Ledezma, R. Aler, A. Sanchis, and D. Borrajo, "Predicting opponent actions by observation," in *RoboCup 2004: Robot Soccer World Cup VIII* (D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, eds.), (Berlin, Heidelberg), pp. 286–296, Springer Berlin Heidelberg, 2005.

[7] A. Ledezma, R. Aler, A. Sanchis, and D. Borrajo, "Ombo: An opponent modeling approach," *Ai Communications*, vol. 22, no. 1, pp. 21–35, 2009.

[8] M. Ahmadi, A. K. Lamjiri, M. M. Nevisi, J. Habibi, and K. Badie, "Using a two-layered case-based reasoning for prediction in soccer coach," in *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications. MLMTA'03, June 23 - 26, 2003, Las Vegas, Nevada, USA* (H. R. Arabnia and E. B. Kozerenko, eds.), (Las Vegas, Nevada, USA), pp. 181–185, CSREA Press, 2003.

[9] T. Steffens, "Adapting similarity-measures to agenttypes in opponent-modelling," in *Workshop on Modeling Other Agents from Observations at AAMAS*, (1730 Massachusetts Ave., NW Washington, DCUnited States), pp. 125–128, IEEE Computer Society, 2004.

[10] T. Steffens, "Similarity-based opponent modelling using imperfect domain theories.," *CIG*, vol. 5, pp. 285–291, 2005.

[11] M. W. Floyd, B. Esfandiari, and K. Lam, "A case-based reasoning approach to imitating robocup players," in *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA* (D. Wilson and H. C. Lane, eds.), (Coconut Grove, Florida, USA), pp. 251–256, AAAI Press, 2008.

[12] T. Fukushima, T. Nakashima, and H. Akiyama, "Mimicking an expert team through the learning of evaluation functions from action sequences," in *RoboCup 2018: Robot World Cup XXII* (D. Holz, K. Genter, M. Saad, and O. von Stryk, eds.), (Cham), pp. 170–180, Springer International Publishing, 2019.

[13] T. Fukushima, T. Nakashima, and H. Akiyama, "Evaluation-function modeling with multi-layered perceptron for robocup soccer 2d simulation," *Artificial Life and Robotics*, vol. 25, no. 3, pp. 440–445, 2020.

[14] L. C. Melo, M. R. Maximo, and A. M. da Cunha, *Imitation learning and meta-learning for optimizing humanoid robot motions*. PhD thesis, Master's Thesis, Instituto Tecnológico de Aeronáutica, Sao José dos Campos ..., 2019.

[15] R. Aler, J. M. Valls, D. Camacho, and A. Lopez, "Programming robosoccer agents by modeling human behavior," *Expert Systems with Applications*, vol. 36, no. 2, Part 1, pp. 1850–1859, 2009.

[16] O. Aşık, B. Görer, and H. L. Akın, "End-to-end deep imitation learning: Robot soccer case study," in *RoboCup 2018: Robot World Cup XXII* (D. Holz, K. Genter, M. Saad, and O. von Stryk, eds.), (Cham), pp. 137–149, Springer International Publishing, 2019.

[17] S. Raza, S. Haider, and M.-A. Williams, "Teaching coordinated strategies to soccer robots via imitation," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1434–1439, 2012.

[18] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *arXiv preprint arXiv:1811.06711*, 2018.

[19] A. Hussein, E. Elyan, and C. Jayne, "Deep imitation learning with memory for robocup soccer simulation," in *Engineering Applications of Neural Networks* (E. Pimenidis and C. Jayne, eds.), (Cham), pp. 31–43, Springer International Publishing, 2018.

[20] A. Hussein, *Deep learning based approaches for imitation learning.* PhD thesis, Robert Gordon University, 2018. COMPLETED.

[21] Q. D. Nguyen and M. Prokopenko, "Structure-preserving imitation learning with delayed reward: An evaluation within the robocup soccer 2d simulation environment," *Frontiers in Robotics and AI*, vol. 7, 2020.

[22] T. Nakashima and H. Ishibuchi, "Mimicking dribble trajectories by neural networks for robocup soccer simulation," in *2007 IEEE 22nd International Symposium on Intelligent Control*, (Singapore), pp. 658–663, IEEE, IEEE, 11 2007.

[23] F. V. Coimbra, "Opponent Modeling by Imitation in Robot Soccer," Bachelor's thesis, Instituto Tecnológico de Aeronáutica, São José dos Campos, November 2021.

[24] F. V. Coimbra, "rcg2csv, an RCG parser based on librcsc-4.1.0." https://github.com/FelipeCoimbra/librcsc, 2021.

[25] F. V. Coimbra, "rcl2csv, an RCL parser for the RoboCup Soccer Simulation 2D League." https://gitlab.com/FelipeCoimbra/rcl2csv/, 2021.

[26] F. V. Coimbra, "RCSS2D Deep Imitation." https://gitlab.com/itandroids/open-projects/rcss2d-opp-imitation, 2021.

[27] H. Akiyama, T. Nakashima, T. Fukushima, Y. Suzuki, and A. Ohori, "Helios2019: Team description paper," 2019.

[28] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

Felipe Vieira Coimbra has a BSc degree in Computer Engineering from the Aeronautics Institute of Technology (ITA), Brazil, with distinctions from the Physics Department, and is a MSc candidate in Computer Science in the Computer and Electronic Engineering Department. He has 5 years of experience in Robotic Soccer, developing agents for the RoboCup Soccer Simulation 2D League. His research interests are Multiagent Systems, Optimization, and Artificial Intelligence.

Marcos R. O. A. Maximo received the BSc degree in Computer Engineering (with Summa cum Laude honors) and the MSc and PhD degrees in Electronic and Computer Engineering from Aeronautics Institute of Technology (ITA), Brazil, in 2012, 2015 and 2017, respectively. Maximo is currently a Professor at ITA, where he is a member of the Autonomous Computational Systems Lab (LAB-SCA) and leads the robotics competition team ITAndroids. He is especially interested in humanoid robotics. His research interests also include mobile robotics, dynamical systems control, and artificial intelligence.