FRA-UNIted — Team Description 2018

Thomas Gabel, Philipp Klöppner, Eicke Godehardt

Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences
60318 Frankfurt am Main, Germany
{tgabel|godehardt}@fb2.fra-uas.de, kloeppne@stud.fra-uas.de

Abstract. The main focus of FRA-UNIted's effort in the RoboCup soccer simulation 2D domain is to develop and to apply machine learning techniques in complex domains. In particular, we are interested in applying reinforcement learning methods, where the training signal is only given in terms of success or failure. In this paper, we describe the implementation of our newest behavior: Wiretapping and decoding opponent communication using convolutional neural networks in TensorFlow.

1 Introduction

The soccer simulation 2D team FRA-UNIted is a continuation of the former Brainstormers project which has ceased to be active in 2010. The ancestor Brainstormers project was established in 1998 by Martin Riedmiller, starting off with a 2D team which had been led by the first author of this team description paper since 2005. Over the years, a number of sister teams emerged (e.g. the Tribots and Twobots) participating in real robot leagues. Our efforts in the RoboCup domain have been accompanied by the achievement of several successes such as multiple world champion and world vice champion titles as well as victories at numerous local tournaments throughout the first decade of the new millennium.

While the real robot teams mentioned were closed down entirely, the 2D team has been in suspended mode since 2010 and was re-established in 2015 at the first author's new affiliation, Frankfurt University of Applied Sciences, reflecting this relocation with the team's new name FRA-UNIted.

As a continuation of our efforts in the ancestor project, the underlying and encouraging research goal of FRA-UNIted is to exploit artificial intelligence and machine learning techniques wherever possible. Particularly, the successful employment of reinforcement learning (RL, [8]) methods for various elements of FRA-UNIted's decision making modules — and their integration into the competition team — has been and is our main focus.

Moreover, the extended use of the FRA-UNIted framework in the context of university teaching has moved into our special focus. So, we aim at employing the 2D soccer simulation domain as a fundament for teaching agent-based programming, foundations of multi-agents systems as well as applied machine learning algorithms.

In this team description paper, we refrain from presenting approaches and ideas we already explained in team description papers of the previous years. Instead, we focus on recent changes and extensions to the team as well as on reporting partial results of work currently in progress. We start this team description paper, however, with a short general overview of the FRA-UNIted framework. Note that, to this end, there is some overlap with our older team description papers including those written in the context of our ancestor project (Brainstormers 2D, 2005–2010) which is why the interested reader is also referred to those publications.

1.1 Design Principles

FRA-UNIted relies on the following basic principles:

- There are two main modules: the world module and the decision making module.
- Input to the decision module is the approximate, complete world state as provided by the soccer simulation environment.
- The soccer environment is modeled as a Markovian Decision Process (MDP).
- Decision making is organized in complex and less complex behaviors where the more complex ones can easily utilize the less complex ones.
- A large part of the behaviors is learned by reinforcement learning methods.
- Modern AI methods are applied wherever possible and useful (e.g. particle filters are used for improved self localization).

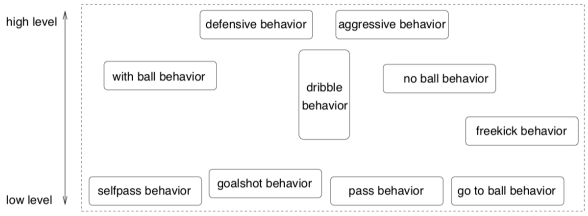

Fig. 1. The Behavior Architecture

1.2 The FRA-UNIted Agent

The decision-making process of the FRA-UNIted agent is inspired by behavior-based robot architectures. A set of more or less complex behaviors realize the agents' decision making as sketched in Figure 1. To a certain degree this architecture can be characterized as hierarchical, differing from more complex behaviors, such as "no ball behavior", to very basic, skill-like ones, e.g. "pass behavior". Nevertheless, there is no strict hierarchical sub-divisioning. Consequently, it is

also possible for a low-level behavior to call a more abstract one. For instance, the behavior responsible for intercepting the ball may, under certain circumstances, decide that it is better to not intercept the ball, but to focus on more defensive tasks and, in doing so, call the "defensive behavior" and delegating responsibility for action choice to it.

2 On the Shoulders of a Giant: Integrating TensorFlow into the FRA-UNIted Agent

The FRA-UNIted agent inherited Martin Riedmiller's N++ framework for neural networks in C++ [7]. In the past, it was used to train behaviors like the dribbling behavior mentioned in our 2017 Team Description Paper [3] or the aggressive defense behavior NeuroHassle [4] where in both cases we utilized reinforcement learning for training.

In 2017, we did research on interpreting and understanding opponent agent communication [5] using the TensorFlow [1] framework for machine learning in Python. The results of this publication motivated us to integrate TensorFlow into our agent to see how the knowledge gained can benefit our team's performance in a live game situation. So far, the integration was successful, but evaluation is still ongoing and details on our observations will be published in a paper that is currently under review.

2.1 Eavesdropping Opponent Communication

Communication between agents of a team has always played an important role in the 2D simulation league of RoboCup. By exchanging short messages, players are able to synchronize their observations with each other. Also, they are enabled to announce and request passes and other actions.

In our recent research, however, we focused on communication between opponent agents. We developed and implemented a neural network-based model that tries to predict whether an opponent message is a pass announcement or not. If we assume that it is, we try to get more information out of the message such as the movement vector of the announced pass by using a separate deep convolutional neural network. For more details on this approach we refer to [5].

2.2 Defining and Training Deep Neural Networks Using TensorFlow

Since the mentioned TensorFlow models were implemented in Python and our agent code base is implemented completely in C++, it was a natural fit to try to port these models to C++, using TensorFlow's C++ API. Leaving the models in Python and sending messages between an agent process and a Python process for regression would have been an option, but we assumed that the overhead of inter-process communication will make this approach unsuitable for real-time soccer simulation games.

We had to closely and thoroughly examine the TensorFlow C++ API, as it was neither as widespread nor as well documented as the Python API when we first started working with it. Defining a simple graph model in C++ using TensorFlow turned out to be very similar to the Python code for the same task. Unfortunately, the C++ API does not include any optimizer classes such as the Adam optimizer [6] used for training our models. We then decided to source out the graph definition to the Python API as TensorFlow offers means to effortlessly serialize, save and restore a graph definition using Protocol Buffers[1,2].

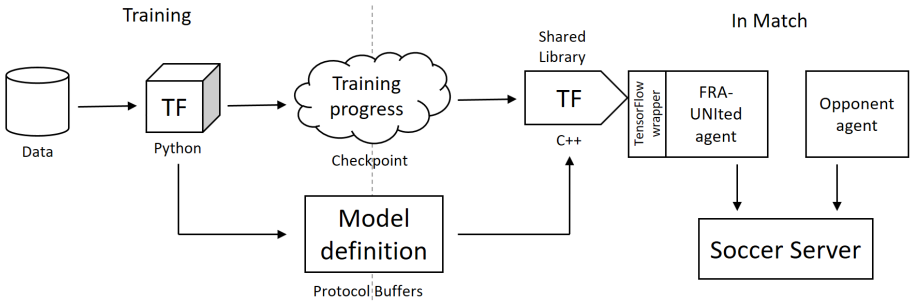

Fig. 2. The Python script defines the topology of the neural network model and uses previously gathered data to train it. Training progress and model definition are stored in separate files — the latter is generic for all teams, while the former is different for each opponent team we are facing. Using the TensorFlow C++ API, both files are then utilized for our agent.

As a consequence, our C++ agent code contains only few calls to the TensorFlow API. Upon initialization of our singleton TensorFlow wrapper class, we create a session[3] that uses the de-serialized graph definition from our model's ProtoBuf file. When an opponent's say message (string of up to 10 characters) is heard, it is forwarded to the TensorFlow Wrapper (see Figure 2). It first feeds the given message into the pass classification model that will output whether the given message is assumed to be a pass announcement or not. If it is, another model is fed with the same message, returning predicted information about the pass.

2.3 Creating a Shared TensorFlow Library

The recommended approach to building TensorFlow C++ projects is to use Bazel, which is a software build automation tool, similar to but on a higher abstraction level than the well known GNU Make. Our build process utilizes the

[1] Protocol Buffers (ProtoBuf) is a language-agnostic method of serializing data.
[2] Also, edge weights/variable values can be saved and restored in a similar manner to snapshot training progress. The resulting file is called a checkpoint.
[3] A session is used to hold the state of a TensorFlow model and run operations on it.

latter, which meant we had to find a way to integrate the library independently from Bazel. Fortunately, the TensorFlow source provides a Bazel build rule to compile a shared object file of the TensorFlow framework that can be linked to the agent source.

Static linking might have also been an option but we decided against it because this approach is not as well-documented as the shared object approach and is not officially supported. We assume that the potential performance advantage bears no relation to the required effort to engineer an adequate and enduring solution to this problem.

Using this shared object in combination with the corresponding header files[4] enables us to build the agent with TensorFlow support using our established GNU Make process. The size of our agent binary grew from 3.6 MB to about 6.0 MB when linking it to the shared library. The shared library itself is around 96 MB. On a modern 4-core i7 machine without GPU and with 22 agents, two coaches and a soccer server running all on this single machine, classification as a pass or no-pass requires 3–4 milliseconds. On a positive result, determination of detailed information about the pass takes 9–10 milliseconds on average.

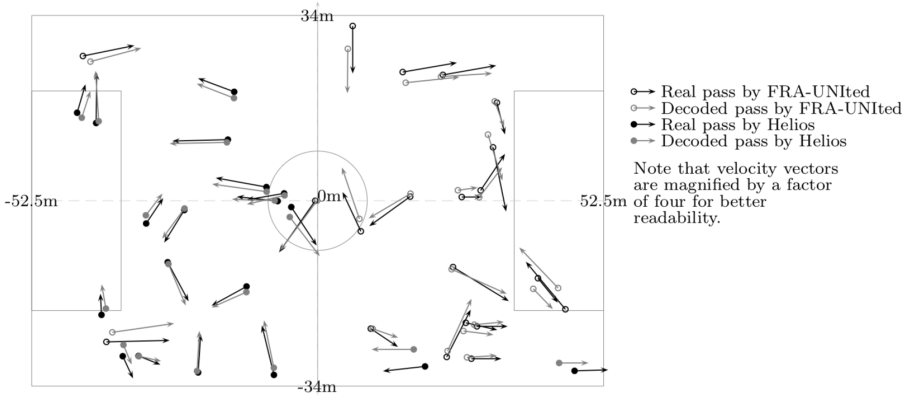

Fig. 3. For teams Helios and FRA-UNIted, 20 randomly chosen (not cherry-picked) passes are visualized, opposing the real passes played and the information extracted from pass announcing say messages that were sent by pass-playing agents and decoded using our learned models. (Figure taken from [5].)

2.4 Discussion

As remarked earlier, evaluation is still ongoing and a more elaborate discussion of our results will take place in a separate paper. Figure 3 shows the accuracy

[4] A TensorFlow project requires three classes of header files: The TensorFlow header files generated while building the shared object with Bazel and the header files of ProtoBuf and Eigen, two libraries TensorFlow relies on.

of pass predictions against team Helios 2017 [2]. Furthermore, the integration of TensorFlow lays the foundation for further research on advanced deep learning approaches in RoboCup 2D, as neural network models can now be tested and utilized in actual matches with little implementation effort.

3 Conclusion

In this team description paper we have outlined the characteristics of the FRA-UNIted team participating in RoboCup's 2D Soccer Simulation League. We have stressed that this team is a continuation of the former Brainstormers project, pursuing similar and extended goals in research, development as well as for teaching purposes. Specifically, we have put emphasis on our most recent research activities and practical implementation of our results.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015), http://tensorflow.org/, software available from tensorflow.org
2. Akiyama, H., Nakashima, T., Tanaka, S., Fukushima, T.: HELIOS2017: Team Description Paper (2017), www.robocup2017.org/file/symposium/soccer_sim_2D/TDP_HELIOS2017.pdf, supplementary material to RoboCup 2017: Robot Soccer World Cup XXI
3. Gabel, T., Breuer, S., Roser, C., Berneburg, R., Godehardt, E.: FRA-UNIted - Team Description 2017 (2017), Supplementary material to RoboCup 2017: Robot Soccer World Cup XXI
4. Gabel, T., Riedmiller, M., Trost, F.: A Case Study on Improving Defense Behavior in Soccer Simulation 2D: The NeuroHassle Approach. In: L. Iocchi, H. Matsubara, A. Weitzenfeld, C. Zhou, editors, RoboCup 2008: Robot Soccer World Cup XII, LNCS. pp. 61–72. Springer, Suzhou, China (2008)
5. Gabel, T., Tharwat, A., Godehardt, E.: Eavesdropping Opponent Agent Communication Using Deep Learning. In: Proceedings of Multi-Agent System Technologies (MATES 2017). pp. 205–222. Springer, Leipzig (2017)
6. Kingma, D., Ba, J.: Adam: A Method for Stochastic Optimization. In: Proceedings of the International Conference on Learning Representations (ICLR) (2015)
7. Riedmiller, M., Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In: Proceedings of the IEEE International Conference on Neural Networks (ICNN). pp. 586–591. San Francisco, USA (1993)
8. Sutton, R.S., Barto, A.G.: Reinforcement Learning. An Introduction. MIT Press/A Bradford Book, Cambridge, USA (1998)