Using Deep Learning to Automate Feature Modeling in Learning by Observation

Michael W. Floyd,[1] JT Turner,[1] David W. Aha[2]

[1]Knexus Research Corporation; Springfield, Virginia; USA

[2]Navy Center for Applied Research in AI; Naval Research Laboratory (Code 5514); Washington, DC; USA

{michael.floyd, jt.turner}@knexusresearch.com | david.aha@nrl.navy.mil

Abstract

Learning by observation allows non-technical experts to transfer their skills to an agent by shifting the knowledge-transfer task to the agent. However, for the agent to learn regardless of expert, domain, or observed behavior, it must learn in a general-purpose manner. Existing learning by observation agents allow for domain-independent learning and reasoning but require human intervention to model the agent's inputs and outputs. We describe Domain-Independent Deep Feature Learning by Observation (DIDFLO), an agent that uses convolutional neural networks to learn without explicitly defining input features. DIDFLO uses the raw visual inputs at two levels of granularity to automatically learn input features using limited training data. We evaluate DIDFLO in scenarios drawn from a simulated soccer domain and provide a comparison to other learning by observation agents in this domain.

1. Introduction

Learning by observation (LbO) agents learn to perform behaviors by observing an expert demonstrate those same behaviors. Whereas traditional methods for training an agent may involve computer programming or knowledge engineering, LbO only requires the expert to be able to perform a behavior. This shifts the knowledge-acquisition task from the expert, who would normally be responsible for formally encoding their knowledge or programming the agent, to the agent itself, thereby allowing LbO agents to learn from a variety on non-technical experts (e.g., healthcare professionals, disaster relief personnel, military commanders).

LbO is well-suited for situations where learning a particular behavior is more important than learning an optimal behavior. A sub-optimal behavior may be necessary if a user finds that behavior more trustworthy or preferable (Shapiro and Shachter 2002). For example, consider a self-driving car. A car driving at high speeds and taking sharp turns may reach the destination safely and efficiently, but a user may prefer a car that drives the speed limit and takes scenic routes. Since LbO does not optimize against a domain's predefined performance metrics (e.g., safety and efficiency for a self-driving car), it learns exclusively using an expert's demonstration of the behavior.

For an LbO agent to learn a behavior regardless of expert or domain, it should learn in a general, non-biased manner. We describe *Domain-Independent Deep Feature Learning by Observation (DIDFLO)*, a learning agent that overcomes several limitations of existing general-purpose LbO agents. Specifically, we remove the need for input features to be manually modeled for each domain. Instead, we use deep learning (DL) techniques (LeCun, Bengio, and Hinton 2015) to learn a feature representation from the agent's raw visual inputs. DIDFLO trains two DL models: one uses the agent's complete visual inputs while the other uses close-range visuals. The output of the two models are combined to select actions to perform in response to visual inputs during deployment (i.e., when the agent attempts to replicate the expert's behavior).

Our work has three primary contributions. First, we describe an application of deep feature learning in a learning by observation agent. While deep feature learning has been used with other learning techniques, we are unaware of any applications in LbO. Second, we demonstrate the performance of DIDFLO in a partially observable domain with limited training data and training time. These constraints result in significant design differences from existing deep feature learning systems that use large data sets and training times. Third, we provide a detailed comparison of three LbO agents that operate in a simulated soccer domain, highlighting the strengths and limitations of each.

We discuss related work in Section 2, with a specific focus on how DIDFLO differs from existing LbO and deep feature learning systems. Section 3 describes how DIDFLO observes, learns, and reasons. We evaluate our approach using scenarios defined in a simulated soccer domain in Section 4, and conclude with a discussion of future work in Section 5.

2. Related Work

Learning by observation has been used in a variety of domains (e.g., Grollman and Jenkins 2007; Ontañón et al. 2007; Rubin and Watson 2010). However, most existing LbO systems are designed to learn in a single domain. Substantial knowledge-engineering or redesign is necessary to deploy these systems in new domains or, in some situations, to learn non-standard behaviors (e.g., an expert that attempts to achieve a different set of goals). Two domain-independent approaches for LbO have been proposed: MMPM (Gómez-Martín et al. 2010) and jLOAF (Floyd and Esfandiari 2011). MMPM and jLOAF are similar in that they separate the agent's learning and reasoning from how it interacts with the environment. This is advantageous because it allows the development of general-purpose observation, learning, and reasoning components. Additionally, since algorithms are designed in a domain-independent manner, it helps prevent biasing them to any specific expert, behavior, or domain. However, before these systems can be deployed in a new domain they require the agent's inputs (i.e., what objects it can observe in the environment) and outputs (i.e., what actions it can perform) to be defined. Although this process only needs to be performed once, it still represents a non-trivial knowledge engineering task.

Floyd, Bicakci, and Esfandiari (2012) partially automate input and output modeling by using a robot architecture that allows sensors and effectors to be dynamically added or removed. Each component registers when connected to the robot or deregisters when disconnected from the robot, allowing the LbO agent to dynamically modify its input and output models. While this does not require human intervention before deployment in a new domain, it does require human intervention for each new type of sensor or effector the system can use. Our approach differs in that it does not require any knowledge engineering as long as the domain provides a visual representation of the environment.

Our feature learning method is inspired by the deep reinforcement learning work of Mnih et al. (2015). They learn input features from raw visual inputs as the agent plays a variety of Atari 2600 games. Their work differs from DIDFLO in the method of learning used and the amount of training time required. Reinforcement learning requires a reward function to be defined in each domain, thereby adding an additional knowledge engineering step than is not required in LbO. While the method they use to measure reward, the game score, is suitable for a variety of Atari games, a different reward function would be necessary for any domain that does not provide a score. Similarly, their system requires a significant amount of time for the agent to interact with the environment (i.e., exploration and exploitation). Such an approach would not be possible if the domain is not fully specified in advance and rapid training is necessary (e.g., a search and rescue domain). Deep reinforcement learning has also been used in simulated soccer (Hausknecht and Stone 2016). Unlike Mnih et al. (2015), their reward functions are more heavily biased to a specific domain and partially encode the behavior being learned (e.g., *move to ball* reward and *kick to goal* reward).

Deep LbO is used for initial training of AlphaGo (Silver et al. 2016), with subsequent training using deep reinforcement learning. However, their LbO methodology has several limitations that make it unsuitable for our requirements. First, they trained their system with over 30 million observations. Large datasets may be available for established games like Go, but little or no data may exist for less popular games or experts with novel behaviors/strategies. Second, such a large dataset requires months of training using datacenters composed of state-of-the-art hardware. This makes their approach impractical if an agent needs to be trained rapidly with limited computational resources. Finally, LbO is performed using images of a turn-based board game. This minimizes the influence of object occlusion (i.e., each Go piece is on its own square), observation error (e.g., due to erroneous or delayed responses by the expert), and provides the learning agent with full observability. Instead, we examine the feasibility of using deep learning by observation with limited observations and training time in partially observable, real-time domains.

3. Domain-Independent Deep Feature Learning by Observation

Domain-Independent Deep Feature Learning by Observation (DIDFLO) has four stages: *modeling*, *observation*, *learning*, and *deployment*. This section describes how each stage is implemented.

3.1 Modeling

Agents typically receive their sensory inputs in the form of periodic messages from a server (e.g., in games or simulations) or from a set of sensors (e.g., a physical robot). Although the set of possible inputs may be well-defined (e.g., in a game's user manual, in a robot's design document), some human intervention is required to encode the input definitions into a format that is understandable by

an agent. For example, in a simulated soccer domain it would be necessary to provide mechanisms for parsing sensory input messages and internally representing the observable objects (e.g., soccer balls, players, goal nets, boundary markers). Although modeling is only necessary the first time an agent is deployed in a new or modified environment, it is still a process that cannot be fully automated by the agent itself.

DIDFLO removes the need to explicitly model an agent's sensory inputs by using a raw visual representation of the environment (i.e., an image of what can currently be observed). This is beneficial because many domains provide sensory information in this format (e.g., a game's visualization, a robot's onboard camera). We assume that this information will be provided in a raw format without any additional processing or annotation (e.g., object identification or labeling). This allows the agent to be deployed in a new environment without modification, so long as the environment provides visual inputs. For example, an agent can transition between a soccer game, where the inputs represent the field, to a chess game, where the inputs represent the board, without any background information about what appears in the images.

We use the visual inputs at two levels of granularity: the *full* visual representation and the *zoomed* visual representation. Figure 1 shows an example of the full and zoomed representations in a simulated soccer game. The full representation V_{full} contains the player's entire field of vision, whereas the zoomed representation V_{zoom} contains an enlarged view of objects within a fixed-sized region surrounding the player. The visual representations contain what the player can currently observe, which may be a partial observation of the entire environment (i.e., if the player has a limited field of vision). By using the zoomed representation, the agent receives a higher fidelity view of nearby objects and is provided a better observation of objects that are partially occluded. However, since the zoomed representation only contains objects in a fixed-sized region surrounding the player, it may not contain as many objects as the full representation. Since the number of observations may be limited, storing each observation at two levels of granularity provides additional training data to use during learning and reasoning. Both V_{full} and V_{zoom} are stored as 256×256 pixel RGB images.

3.2 Observation

The observation process involves the expert acting in the environment (i.e., performing its behavior) and the learning agent observing the expert. During observation, the learning agent records the visual inputs received by the expert (i.e., V_{full} and V_{zoom}) and the resulting actions. The agent assumes that an action A is performed by the expert after reasoning about the most recent visual inputs, so observations as stored as input-actions pairs (i.e., $\langle V_{full}, A \rangle$ and $\langle V_{zoom}, A \rangle$). Over the course of observation, the learning agent collects all such input-action pairs and stores them in two observation sets, $\mathcal{O}_{full}$ and $\mathcal{O}_{zoom}$ ($\mathcal{O}_{full} = \{\langle V'_{full}, A' \rangle, \langle V''_{full}, A'' \rangle, \dots\}$ and $\mathcal{O}_{zoom} = \{\langle V'_{zoom}, A' \rangle, \langle V''_{zoom}, A'' \rangle, \dots\}$).

The observations serve as labelled training data that the agent can use to learn how to select actions in a similar manner as the expert (i.e., what action to perform in response to a given visual input). Additionally, in the case of an agent that learns exclusively by observation (i.e., no additional knowledge or feedback is provided by the expert), observations are the only source of information that is available to the agent.

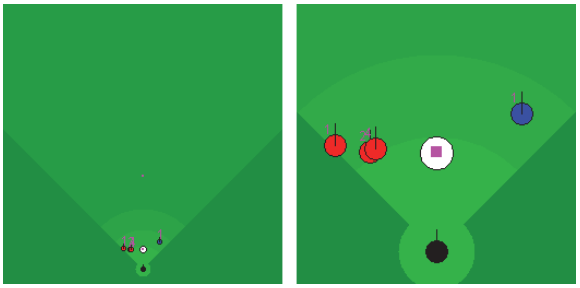

Figure 1: The full visual representation (left) and zoomed visual representation (right) in a simulated soccer game

3.3 Training

The training stage involves using the collected observations to learn models of the expert's behavior. DIDFLO trains two models, one model M_{full} that maps full visual inputs to actions and one model M_{zoom} that maps zoomed visual inputs to actions ($M_{full}: \mathcal{V}_{full} \rightarrow \mathcal{A}$ and $M_{zoom}: \mathcal{V}_{zoom} \rightarrow \mathcal{A}$, where $\mathcal{V}_{full}$ and $\mathcal{V}_{zoom}$ are the sets of all visual inputs, and $\mathcal{A}$ is the set of all actions). For both models, learning is performed using convolutional neural networks (CNN) (Krizhevsky, Sutskever, and Hinton 2012). The motivation for training two models is that each representation has certain strengths and weaknesses, so combining the models will produce better performance than either model in isolation. For example, a nearby soccer ball would be easier to detect in the zoomed image because it appears larger, whereas a goal net on the other side of the field can only be detected using the full image. The collected observation sets $\mathcal{O}_{full}$ and $\mathcal{O}_{zoom}$ are used to train the respective CNNs.

The CNN architecture used is a modified version of CaffeNet (Jia et al. 2014), containing an input layer, five convolution layers, five pooling layers, two fully connected layers, and one softmax loss layer. The CNNs take as input the pixel values of an image using all three color channels (i.e., red, green, and blue). Both V_{full} and V_{zoom} are 256×256 pixel images, resulting in each CNN having $256 \times 256 \times 3$ inputs. Each CNN outputs a set $\mathcal{C}$ containing the confidence values $c_1, \dots, c_n$ for each of the n possible actions

($\mathcal{C} = \{c_1, \ldots, c_n\}$, with each confidence value in the range $[0.0, 1.0]$). In the simulated soccer example, three actions are used[1]: *kick*, *dash* (i.e., move), and *turn*. This results in each CNN outputting three confidence values $(c_{kick}, c_{dash}, c_{turn})$.

DIDFLO was designed under the assumption that training data would be limited, so rather than fully train each CNN, a subset of layers are pretrained on other data sources. The convolution and pooling layers are extracted from a publicly available network trained on ImageNet data (Jia et al. 2014). This leaves only the fully connected layers and softmax loss layer to be trained using the collected observations. This approach has two primary advantages. First, the pretrained ImageNet layers can identify many important visual features already (e.g., lines, curves, shapes, objects). This removes the need to relearn common features by leveraging a preexisting network that was trained on a dataset containing millions of images across a variety of topics. Since the pretrained layers contain both low-level and high-level features across a variety of image categories, it does not bias DIDFLO to any particular domain. Second, the limited number of available training observations makes it impractical to train the entire network. Instead, training focuses exclusively on learning how existing features can be used to classify visual inputs. Both M_{full} and M_{zoom} are trained using an identical CNN architecture (i.e., they differ only in the observation sets used to train them).

3.4 Deployment

During deployment, DIDFLO is placed in the environment in a similar role as the expert (e.g., as a soccer player). Like the expert, DIDFLO receives sensory inputs from the environment and selects actions to perform. The primary goal of deployment is for the agent to select similar actions to the expert when presented with similar sensory inputs. Each received sensory input is used as input to both the full and zoomed CNNs, resulting in two sets of confidence values $\mathcal{C}_{full}$ and $\mathcal{C}_{zoom}$, that can be combined into an overall confidence value set $\mathcal{C}_{overall}$ ($\mathcal{C}_{overall} \leftarrow \mathcal{C}_{full} \cup \mathcal{C}_{zoom}$). Since the two CNNs use different input representations and are trained independently, they may select different actions to perform (e.g., one is most confident in *kick* and the other is most confident in *turn*). DIDFLO selects a single action to perform by selecting the highest confidence value c_{max} from $\mathcal{C}_{overall}$ and using its associated action ($\exists c_{max} \in \mathcal{C}_{overall}$ such that $\forall c_i \in \mathcal{C}_{overall}, c_{max} \geq c_i$).

By using this combined approach, DIDFLO leverages the strengths of each individual CNN during action selection. For example, we would expect the zoomed CNN to perform better when important objects are near the agent, whereas the full CNN would perform better when information from the entire field of vision is necessary.

[1] The actions can also be parameterized (e.g., turn direction, running speed) but for simplicity we only consider the class of action.

4. Evaluation

In this section we evaluate our claim that *DIDFLO can learn from observations without an explicit model of its inputs*. Our evaluation tests the following hypotheses:

- **H1**: Representing inputs at two levels of granularity will improve learning performance
- **H2**: DIDFLO will achieve comparable performance to other domain-independent learning by observation systems in a simulated soccer domain

4.1 Experimental Setup

To evaluate the performance of DIDFLO, we collected simulated soccer data from the RoboCup Simulation League (RoboCup 2016). DIDFLO observed a series of 5 vs. 5 soccer games where AI agents controlled each of the ten players. The specific agent used to control each player was a simple scripted agent named *Krislet*. Krislet performs simple soccer behavior that involves locating the ball, running towards the ball, and kicking the ball towards the opponent's goal. In each match, a single Krislet agent was used as the expert and had its inputs and actions recorded.

DIDFLO observed 10 full soccer matches resulting in approximately 40,000 observations (approximately 100 minutes of observation). The observations are highly imbalanced (approximately 73% dash, 26% turn, 1% kick), so random sampling was performed to create a training set of 1500 observations (i.e., 1500 full representation training examples and 1500 zoomed representation training examples). This process was repeated to create 25 randomly sampled training sets. Additional soccer matches were observed and a similar random sampling method was used to create 25 test sets of 1029 observations each. For each evaluation, a single training set was used to train the CNNs and a single testing set was used to evaluate the performance. The CNNs were trained using a base learning rate of 0.01, polynomial rate decay with a power of 3, and 13,000 training iterations.

4.2 Results

Table 1 shows the average F_1 score (and 95% confidence interval) over each of the 25 training runs. The F_1 score calculates the harmonic mean of the precision and recall values, with values ranging from 0.0 (low) to 1.0 (high). The table shows the performance at predicting each individual action along with the overall performance. In addition to the combined approach, we also evaluated the performance

when only the full or zoomed CNN was used for action prediction.

Table 1: Results of trained CNNs on RoboCup test data

Model	F_1 Kick	F_1 Dash	F_1 Turn	F_1 Overall
Full	0.66±0.03	0.60±0.02	0.48±0.01	0.58±0.01
Zoomed	0.89±0.01	0.60±0.01	0.56±0.01	0.68±0.01
Combined	0.90±0.01	0.63±0.01	0.58±0.01	0.71±0.01

The results show that the combined approach is a statistically significant improvement over using either the full or zoomed representation alone (using a paired t-test with $p < 0.001$). These results provide support for **H1**. For all three action types, the combined performance is better than any individual representation. This demonstrates that both the full and zoomed representation are providing a positive contribution to the overall performance. Although the zoomed representation significantly outperforms the full representation at selecting kick and turn actions, the improved performance of the combined approach at predicting those actions demonstrates that there are certain inputs the full representation is better at selecting actions for. For example, the zoomed representation is particularly good at selecting kick actions because the ball is larger and more visible. However, the presence or absence of the opponent's goal net is often only visible in the full representation (i.e., to differentiate between kicking towards the net and turning to see the net).

Overall, these results demonstrate that DIDFLO can learn a suitable model for action selection. Although each model is trained on relatively few training instance (i.e., 1500), DIDFLO is able to achieve reasonable performance. For example, a random baseline that randomly selects actions to perform has an overall F_1 score of approximately 0.33.

4.3 Comparison to Existing LbO Systems

RoboCup has been used as a domain in previous learning by observation work (Floyd, Esfandiari, and Lam 2008; Floyd and Esfandiari 2011; Young and Hawes 2015). Although differences in evaluation methodology do not allow for a direct empirical comparison, this subsection provides a comparison of the relative strengths of each approach.

Floyd et al. (2008) use *case-based reasoning* to perform learning by observation. Each observed state-action pair is stored as a *case* by the agent and during deployment the agent retrieves the case that is most similar to its current sensory input. This system is specifically designed to be domain-independent and has also been deployed in Tetris, obstacle avoidance, and robotic arm control (Floyd and Esfandiari 2011). This approach will be referred to as F&E. Young and Hawes (2015) use *Qualitative Spatial Relations* to modify how objects are represented in the environment. Rather than use the observed quantitative values, each object is represented by a set of qualitative relations to other objects. The resulting representation can then be used as input to a variety of existing classification algorithms. Although this system has only been deployed in a single domain, the authors minimize that amount of soccer-specific information by allowing the system to evaluate a number of possible spatial relationships when determining how to represent observations. This approach will be referred to as Y&H.

The primary strength of DIDFLO is that it requires less knowledge engineering than the other two approaches. Both F&E and Y&H require a definition of the possible objects in the environment, whereas DIDFLO learns an object model during training. Representation learning during a preprocessing stage is performed by F&E (i.e., learning important object types) and Y&H (i.e., learning the best qualitative representation), but both rely on an existing object model. However, although DIDFLO requires the least knowledge engineering, only F&E has been successfully deployed in multiple domains.

The strength of Y&H is that their primary contribution is a representation framework rather than a learning algorithm. This allows the qualitative representation to be used to train a variety of classifiers and the best performing classifier (or an ensemble) to be selected for deployment. The qualitative representation of Y&H could potentially be integrated with F&E (i.e., case-based reasoning) or DIDFLO (i.e., deep learning). Although the training data used and evaluation methodology differ, Y&H appears to have the highest action selection accuracy in the RoboCup domain. These results are what we would expect given that it is the most domain-dependent of the three approaches. Of the two domain-independent approaches, DIDFLO appears to outperform F&E in RoboCup. These results provide some support for **H2**.

The instance-based learning of F&E is its primary strength. Although it does not perform as well in RoboCup as DIDFLO or Y&H, it requires no training time after observations are collected (i.e., no generalization is performed). This is also advantageous because it allows newly acquired observations to be added without retraining. This could allow an agent to be rapidly created and deployed until the more computationally expensive models used by DIDFLO or Y&H are trained. Additionally, F&E have the only approach that has been successfully used to learn from state-based experts. (i.e., DIDFLO and Y&H can only learn from reactive experts). However, as the number of cases increase so too does the time required to retrieve a similar case. This places a limit on the number of training observations that can be used by the agent in real-time.

Overall, the three learning by observation systems used in the RoboCup domain have significant strengths based on

their design motivations. DIDFLO, F&E, and Y&H all aim to provide slightly different usages or functionality, thereby influencing how they approach the learning by observation problem. Each system has one or more elements that could potentially be integrated into the other systems or a novel LbO agent.

5. Conclusions and Future Work

We described Domain-Independent Deep Feature Learning by Observation, a learning by observation agent that can learn without being provided an explicit model of the objects it observes. Our approach uses observations of an expert's response to raw visual inputs to train a pair of convolutional neural networks. The CNNs differ in the granularity of observations they use, with one using the expert's entire field of vision and the other using a fixed-sized region surrounding the expert.

In our study, DIDFLO was able to learn from the expert and perform reasonably well at selecting a similar action to perform when presented with similar input. Our results demonstrated that combining the output of both CNNs resulted in significant improvements over either CNN in isolation. This indicates that even with limited training observations, partial observability, and noisy observations, it is possible to train an agent that can learn an expert's behavior without an explicit object model.

Several areas of future work remain. First, although DIDFLO removes the need to model observable objects, it still requires modeling possible actions. We plan to investigate methods for learning action models from observations. Second, our approach only reasons over visual inputs. In real-world scenarios it may be necessary to reason over additional inputs (e.g., audio). Third, we have only examined a two-CNN architecture. Future work will examine if further performance improvement is possible by training additional CNNs (e.g., other levels of granularity) or modifying how CNN outputs are combined. Fourth, although we have developed DIDFLO to be domain independent, further evaluation in other domains is necessary to validate this claim. Finally, we plan to examine how this approach can be extended to incorporate features of other learning by observation systems (e.g., learning from state-based experts, active observation acquisition, inferring high-level goals from observations).

References

Floyd, M. W., Bicakci, M. V. and Esfandiari, B. 2012. Case-based learning by observation in robotics using a dynamic case representation. In *Proceedings of the 25th International Florida Artificial Intelligence Research Society Conference*, 323-328. Marco Island, USA: AAAI Press.

Floyd, M. W., and Esfandiari, B. 2011. A case-based reasoning framework for developing agents using learning by observation. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence*, 531-538. Boca Raton, USA: IEEE Computer Society Press.

Floyd, M. W., Esfandiari, B., and Lam, K. 2008. A case-based reasoning approach to imitating RoboCup players. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference*, 251-256. Coconut Grove, USA: AAAI Press.

Gómez-Martín, P. P., Llansó, D., Gómez-Martín, M. A., Ontañón, S., and Ram, A. 2010. MMPM: A generic platform for case-based planning research. In *Proceedings of the International Conference on Case-Based Reasoning Workshops*, 45-54. Alessandria, Italy.

Grollman, D. H., and Jenkins, O. C. 2007. Learning robot soccer skills from demonstration. In *Proceedings of the IEEE International Conference on Development and Learning*, 276-281. London, UK: IEEE Press.

Hausknecht, M., and Stone, P. 2016. Deep reinforcement learning in parameterized action space. In *Proceedings of the International Conference on Learning Representations*. San Juan, Puerto Rico.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R. B., Guadarrama, S., and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, 675-678. Orlando, USA: ACM.

LeCun, Y., Bengio, Y. and Hinton, G. E. 2015. Deep learning. *Nature*, 521, 436-444.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. 2012. Classification with deep convolutional neural networks. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems*, 1106-1114. Lake Tahoe, USA.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518, 529-533.

Ontañón, S., Mishra, K., Sugandh, N., and Ram, A. 2007. Case-based planning and execution for real-time strategy games. In *Proceedings of the 7th International Conference on Case-Based Reasoning*, 164-178. Belfast, UK: Springer.

RoboCup. 2016. *RoboCup Official Site*. Retrieved from [http://www.robocup.org]

Rubin, J., and Watson, I. 2010. Similarity-based retrieval and solution re-use policies in the game of Texas Hold'em. In *Proceedings of the 18th International Conference on Case-Based Reasoning*, 465-479. Alessandria, Italy: Springer.

Shapiro, D., Shachter, R. 2002. User-agent value alignment. In *Proceedings of the Stanford Spring Symposium - Workshop on Safe Learning Agents*. Palo Alto, USA.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 484-503.

Young, J., and Hawes, N. 2015. Learning by observation using qualitative spatial relations. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems,* 745-751. Istanbul, Turkey: ACM.