Playing Soccer with Unknown Teammates

João Francisco Lopes Pirralha

Supervisor: José Alberto Rodrigues Pereira Sardinha

ABSTRACT

Robotic soccer allows researchers to attempt to solve many challenges in the field of artificial intelligence. One such challenge is collaboration with unknown teammates, without any sort of pre-coordination, which is known as ad hoc teamwork. Advances in ad hoc teamwork enable collaboration in multi-agent systems to be more robust and versatile compared to traditional coordination mechanisms, as it addresses situations such as collaboration with agents developed by different people, with legacy agents that cannot be modified and even with humans. Some current work in the literature attempts to address this challenge by reusing experience with past teammates to adapt to new ones, for example by acting using previously learned policies. This thesis extends the state-of-the-art approach in order to also deal with unknown teammates that might be significantly different from past teammates, while still leveraging what was previously learned. To achieve this, a current unidentified team is detected either as being a known team or unknown, by observing if the team's behavior is consistently similar to the past behavior of a known team. If it is detected as unknown, the agent selects the previously learned policy whose team it considers to be most similar to the unknown team, which is then improved online, as a source for parameter sharing transfer learning.

Author Keywords

Artificial intelligence, ad hoc teamwork, autonomous agents and multiagent systems, reinforcement learning

1. INTRODUCTION

1.1 Motivation

As autonomous agents increase in number, robotic or virtual, more situations arise where multiple agents need to cooperate as a team in order to achieve their objectives. However, it is not always possible to know in advance the teammates which an agent will end up with. As such, it is desirable to have a dynamic and versatile mechanism for collaboration that does not rely on pre-coordination.

One such situation in the human world is pick-up soccer, where players, that do not belong to a fixed team, gather to play a match. Players often play with others whom they had limited or no contact, so they need to quickly observe each other's capabilities and adapt. This situation could also happen in robotic soccer. Robotic soccer teams usually have communication and coordination mechanisms, but we can consider a challenge where multiple independent robots are gathered to play a soccer match with random teammates, lacking common mechanisms, e.g. the RoboCup drop-in challenge [20]. Advances in such a challenge could be relevant for other domains, such as rescue robots from different sources that need to work together during a catastrophe, which is a prominent example in the literature[25]. Such advances could also be useful for having robust systems where agents can collaborate with older and limited agents that cannot be modified, or even collaborate with humans.

These situations are framed within the problem of ad hoc teamwork[25], where agents need to cooperate with unknown teammates without relying on previously developed mechanisms for coordination or communication. These agents, often called ad hoc agents, must be able to quickly find an adequate strategy to act efficiently. Unlike multiagent learning, ad hoc agents are developed independently from their teammates.

Some of the current work in the literature [4] attempts to address the ad hoc teamwork problem by leveraging prior knowledge about past teammates. However, new teammates may be arbitrarily different from past ones; thus, using previously learned behavior may be ineffective. We believe that the ad hoc teamwork approach is still useful in this situation, as it allows to reuse knowledge about the dynamics of the environment and teammates, enabling to learn to coordinate faster than just a pure reinforcement learning [27] approach.

1.2 Problem

This thesis addresses a research problem where an ad hoc agent may encounter unidentified teammates which may be known or unknown, when performing a task. The task involves playing a game of Half Field Offense (HFO) [17], described in Section 2. In this environment, one team of agents attempts to score a goal and another defends. The possible teams are: `AUT MasterMinds`, `Base`, `Cyrus`, `Gliders`, `HELIOS` and `YuShan`. These teams are those used in [4], with one missing exception (`Axiom`) as there were technical difficulties in successfully executing it. The ad hoc agent joins an unidentified teammate belonging to one of those teams in the offense and both play against two defensive `HELIOS` players, one of which is a goalkeeper.

1.3 Contributions

The contributions of this thesis are the following:

1. Two methods for identifying known teams: one based on comparing predictions of the next state according to historical data of each of the known teams to the real next state (similar to the presented in [4], but using a model based on a Multi-Layer Perceptron (MLP)); and a second based on discriminating each team directly also according to historical data.
2. An extension to the methods of identifying known teams to also detect unknown teams, where the agent observes if the unidentified team is consistently identified as being the same known team across multiple observations – if not, then the team probably is unknown;
3. The use of parameter sharing transfer learning techniques in order to adapt a policy learned from one team to a different one, online. As far as we are aware, this is a novel use of transfer learning within the field of ad hoc teamwork.

2. HALF FIELD OFFENSE

RoboCup [18] is a soccer competition played by autonomous agents that has several leagues. It has several environments for researchers to empirically test and validate ideas, such as algorithms and agent architectures. While it is best known for its robotic leagues, it also provides simulated ones, namely the 2D subleague of the RoboCup Simulation League.

To provide a more tractable problem, Stone *et al.* proposed a subtask of the 2D Simulation League called Keepaway [26]. In this task, one team attempts to keep the ball from the other team, which attempts to take it, within a small region. This task is episodic – the episodes end if the taking team takes the ball or if it leaves the region. In addition to the smaller state space than the full league, it also has less subobjectives, making it more suitable for directly comparing methods.

In order to test the scalability of algorithms that have good performance in Keepaway to more complex environments, without going directly to the full league, Kalyanakrishnan *et al.* proposed the Half Field Offense (HFO) [17]. Similarly to Keepaway, which it extends, HFO is also a subtask of the 2D Simulation League. In this task, one offense team attempts to score a goal, while a defense team attempts to prevent it. It is played within one half of the soccer field and it is an episodic task – the episodes end if the offense scores a goal or if the ball gets captured by the defense or leaves the playing area. HFO can be viewed as problem for both the offense and the defense, but the authors only focus on the former.

Hausknecht *et al.* [14] released an open source[1] implementation of the HFO. Previous work on the HFO (such as Kalyanakrishnan *et al.* [17]) did not release source code, making it inaccessible for many potential users. It is built on top of the RoboCup 2D simulation platform and it is the environment used throughout this thesis. The authors proposed a primary evaluation metric, Goal Percentage, which is defined as the percentage of trials with a goal scored. They also proposed a secondary one, Time to Goal (TTG), which is defined as the number of time steps required to score a goal, for the trials that end with one scored. This release allows to develop a partial team and already includes two teams that can be used for automated teammates: `HELIOS` (specifically the 2013 release [2]) and `HELIOS Base` [1] (also known as `Agent2D`). An instant of a match in this HFO implementation can be observed in Figure 1.

This HFO release provides access to both low and high-level state spaces. The low-level state space is an egocentric viewpoint based on `HELIOS`' world model. It contains features related to the agent such as its position and orientation, as well as other features pertaining to the ball and other agents. For a 2 VS 2 match, it has 86 features. The high-level state space is a more compact representation and may allow the agent to generalize easier. For a 2 VS 2 match, it has 24 features. When using a Deep Q-Network (DQN) with this environment, it should only have fully-connected layers, as the state is not based on raw data such as an image. Agents cannot directly observe each others' actions, as there are no such features in any of the state spaces.

[1] `https://github.com/LARG/HFO`

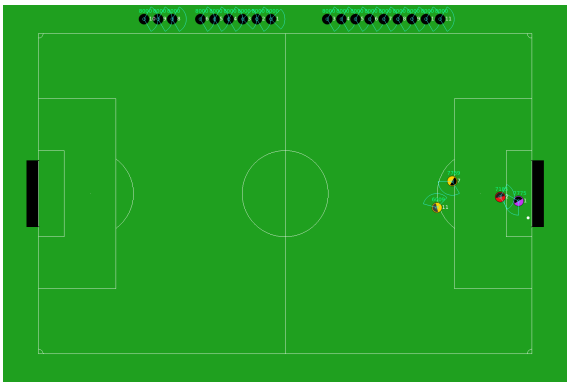

Figure 1: A match of HFO played by two offensive players (in yellow) against two defensive players (including a goalkeeper in purple).

The actions are also provided in different levels of abstraction: low, medium and high. There are four low-level actions (`dash`, `turn`, `tackle` and `kick`) and they are parameterized by power and angle. The medium-level actions (`kick to`, `move to`, `dribble to` and `intercept`) are still parameterized (by coordinates and speed). The high-level actions (`move`, `shoot`, `pass`, `dribble`, `catch`, `reduce angle to goal`, `defend goal`, `go to ball` and `mark player`) are discrete and thus suitable for an algorithm such as DQN (`pass` and `mark player` are parameterized but by a discrete argument identifying a player). They are composed by low level actions, parameterized following `Agent2D`'s strategy. According to the release's manual, the applicable high level actions for the offense team are `shoot`, `pass` and `dribble` when the agent has the ball; and `move`, `go to ball` and `reorient` when it does not.

3. RELATED WORK

Barrett *et al.* [4], based on Barrett's PhD thesis [3], presents an approach to ad hoc teamwork where the ad hoc agent reuses experience with previous teammates when encountering new ones. For this purpose, they proposed a general-purpose algorithm called Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation (PLASTIC). This algorithm assumes that there are similarities between past teammates and new ones (while this may not always be true, the authors consider that often there are some). Given enough time, PLASTIC enables the ad hoc agent (but not necessarily the team) to act optimally if its past experience correctly predicts the new teammates and their behavior is fixed.

The authors focus on a limited version of the ad hoc teamwork problem where they make the following assumptions:

- the ad hoc agent has previous experience with past teammates;
- all agents in the team share a common objective;
- there are no explicit communication protocols;
- the teammates do not learn and their behavior does not change.

PLASTIC is fundamentally based on:

- acquiring knowledge of past teammates either by learning about them or receiving hand-coded knowledge;
- forming beliefs about the current teammates (about which acquired knowledge best fits them);
- using those beliefs and the knowledge of past teammates for planning;
- updating the beliefs by observing the current teammates' behavior.

This general-purpose algorithm has two realizations, PLASTIC-Model and PLASTIC-Policy. As suggested by their names, PLASTIC-Model is a model-based approach that learns models of past teammates (offline, via supervised learning [5] methods), while PLASTIC-Policy is a model-free approach that learns policies to cooperate with past teams. PLASTIC-Model then selects the models that best predict the current teammates for planning, while PLASTIC-Policy similarly selects the policy that best reflects the current team for acting.

The authors argue that PLASTIC-Policy is more adequate for complex problems with large state spaces, as it can use function approximation techniques - they state that it is more effective in HFO as planning in PLASTIC-Model needs many samples and has problems with inaccuracies in the environment modeling. As such, the solution of this thesis is based on PLASTIC-Policy – it extends it to also consider the possibility of an unknown team and to use a special policy improved online for that case. However, they state that PLASTIC-Model, by being able to use a model for each teammate (instead of each team), would be more appropriate for heterogeneous teams (as PLASTIC-Policy selects a policy that best reflects the entire team, not considering individual teammates). Additionally, they also suggest it would be better suited for teams that change behavior (not explored in their work).

3.1 PLASTIC-Policy

PLASTIC-Policy directly collects samples from the environment and uses them to learn a policy for a team, using existing reinforcement learning algorithms. The authors suggest using Fitted Q Iteration (FQI) [10] for learning the policies. This thesis uses DQN to learn the policies as it is more efficient than FQI.

PLASTIC-Policy uses the Polynomial Weights Algorithm (PWA) [6] (1) to update the belief of each known team given the current team's behavior in the observed transition. The loss for PWA is given by 1 minus the probability of the observed behavior, for each team. However, it is not possible to directly obtain this probability, as it is not given by the policies. To address this, the authors use instead a model to estimate it, in particular via a nearest neighbor based approach. For each state s and its next state s' that the agent observes, it finds the past sample $\langle \hat{s}, \hat{a}, \hat{r}, \hat{s}' \rangle$ whose state $\hat{s}$ is most similar to s and then uses the difference between s' and $\hat{s}'$ to estimate a probability. In contrast, this thesis uses models based on neural networks to estimate this probability without searching all of the stored previous samples at each inference. This possibly also allows for better models.

$$belief \leftarrow \frac{belief \cdot (1 - \eta \cdot loss)}{\sum belief \cdot (1 - \eta \cdot loss)}, \eta \leq 0.5 \qquad (1)$$

4. LEARNING, IDENTIFYING AND ADAPTING TO TEAMMATES

The problem (Subsection 1.2) is addressed with an architecture (Figure 2) based on Barrett's PLASTIC-Policy [4], comprised of a training and an execution phase. In the training phase, the agent builds a library of policies, with one policy for each of the known teams. The agent can collect and train on as much experience as needed. In the execution phase, the agent needs to identify the current team either as one of the known teams or as an unknown team. If the team is known, the agent will ideally use the policy previously trained with it, else it will use a policy that is improved online. A team is considered to be unknown if the agent has determined it is not one of the teams it knows. It is considered unidentified if the agent has not yet determined if it is one of the known teams or unknown.

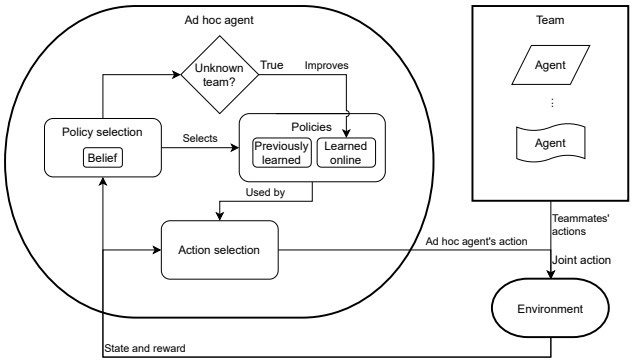

Figure 2: Overview of the architecture (execution phase), inspired by PLASTIC-Policy.

4.1 Policies for known teams

Policies for known teams are learned in the training phase using DQNs. It is appropriate for this problem as it approximates Q-values for discrete actions. This subsection describes how the policies are modeled and how the hyper-parameters are adjusted through a non-extensive search.

This adjustment is performed using an `HELIOS` teammate as a representative. This teammate was selected as it is one of those with the fastest execution times. While potentially biased, this simplification is performed in order to reduce the number of policies needed to be learned for this purpose. In the case of subjectively similar results, the value closest to the initial hyper-parameters (according to literature) is chosen.

4.1.1 State and action spaces

The HFO environment provides both low level and high level state feature sets. Both feature sets are experimented. As usual when employing DQN, the most current states are concatenated for its input. The 4 most recent states will be used, as often used in the literature (e.g. Mnih *et al.* [21]). This concatenation may allow the agent to extract more complex features, such as more insights in the behavior of other agents.

The high level action set provided by the HFO environment is used, as the lower level action sets are parameterized by continuous values and thus unsuitable for DQN. When it does not have the ball, the agent can move (according to team `Base`'s [1] strategy), go to the ball and reorient (the

agent stops and recovers stamina). When it has the ball, it can dribble, shoot and pass it. To model these conditions for the actions, the DQN algorithm is modified according to Lanctot *et al.* [19] such that the DQN does not consider illegal actions, by forcing the probability of illegal actions to be zero when acting and ignoring them when determining $\max_{a' \in \mathcal{A}(s_{t+1})}$ during the update.

4.1.2 Reward function

The reward function (2) is used. It positively rewards the agent when the offense team scores a goal and is neutral otherwise. There is no reward shaping, so the agent will not learn behavior that exploits the reward function in unintended ways. Barrett's [4] reward function (3) presented some issues in terms of the agent attempting to play too defensively until the maximum time step limit, even if scaled to $[-1, 1]$.

$$r(s,a) = \begin{cases} 1 & \text{if the offense team scores a goal} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$r(s,a) = \begin{cases} 1000 & \text{if the offense team scores a goal} \\ -1000 & \text{if the defense team scores a goal} \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

4.1.3 Initial hyper-parameters

The hyper-parameters/modeling from Table 1 are used as a starting point. They are based on the work by Mnih *et al.* [21], with some adjustments to the target network update frequency and final exploration rate by Hasselt *et al.* [28] which are also used in later extensions of DQN. The optimizer is also changed to Adam, as used in Hessel *et al.* [15].

Hyper-parameter/ modeling	Value
DQN extensions	None
Hidden layers (fully connected)	One layer with 512 hidden units with Rectified Linear Unit (ReLU) activation
Minibatch size	32 samples
Replay memory size	1 million samples
Agent history length	4 states
Target network update frequency	7500 updates
Discount factor	0.99
Action repeat	None
Update frequency	4 time steps
Optimizer	Adam
Learning rate	0.0000625
Initial exploration (ϵ-greedy)	1
Final exploration	0.01
Final exploration time step	1 million time steps
Replay start size	50,000 samples

Table 1: Initial hyper-parameters based on literature. Refer to Mnih *et al.* [21] for a detailed description.

4.1.4 Hyper-parameter and modeling adjustment

Some of the hyper-parameters and modeling approaches from Table 1 are adjusted through a non extensive search.

1. Using the low level feature set, the first hyper-parameter to be adjusted is the train frequency. A higher train frequency (that is, lower number of time steps between each training) may allow to train the policies in less wall clock time, which would be saved during the other hyper-parameter/modeling searches. A train frequency of 1 is compared to the train frequency of 4 from Table 1.

2. The low level feature set is compared to the high level feature set, to determine which modeling approach provides better performance.

3. A deeper topology of the Q-network is experimented, consisting of two hidden layers of 256 and 64 ReLU respectively. This topology is not extensively fine-tuned. It may ease transfer learning in Subsection 4.3 by not updating the weights of the first or even both layers.

4. Some extensions to the DQN algorithm are evaluated, in terms of modeling: Double DQN [28] and Dueling DQN [29]. All fully connected layers are doubled for the Dueling DQN. However, neither of these extensions showed to be beneficial in this problem in Subsection 5.2.

4.1.5 Policy training and effectiveness

Policies are trained for every known team and each trained policy is then evaluated with all known teams (Subsection 1.2). Ideally, for each agent the best policy should be the one that was trained with it. With such result, it will be beneficial to identify the team the agent is playing with, in order to select the most optimal policy.

4.2 Policy selection

PLASTIC-Policy needs to identify the team in order to select the policy to be used during the execution phase. This is performed using belief losses that are calculated for each observed transition, which are then used to update the belief. The following subsubsections present how these beliefs are calculated.

4.2.1 Identifying known teammates

PLASTIC-Policy's approach involves scanning known transitions for each team in order to find the past transition whose initial state is most similar to the initial state of the observed transition. Then it uses the difference between the next state of the known transition and the observed new state as a loss. This scanning procedure may have a significant impact in execution performance, so this thesis focuses on creating models for the belief loss during the training phase, which are then used in the execution phase. The samples contained in the DQN replay buffers are used to create these models.

4.2.1.1 Predicting the next state

One alternative is to predict the next state, which includes changes dependent on the team. For each known team, an MLP is trained (hyper-parameters in Table 2): given an observation consisting of 4 concatenated states and an action (one-hot encoded), the output is the difference between the new state s' and the previous state s ($s' - s$ is predicted instead of s' since, as pointed out in [16], the relative transitions may be similar for many state-action pairs). This is an adaptation of the method employed in PLASTIC-Policy,

Hyper-parameter	Value
Hidden layers (fully connected)	Four layers with 256, 192, 128 and 96 units respectively with ReLU activation and He initialization
Output layer	Linear (no activation) with the size of the used state feature set, using uniform He initialization
Loss	Huber
Optimizer	Adam
Learning rate	10^{-2} for 750 iterations, 10^{-3} for 150 iterations, 10^{-4} for 75 iterations and 10^{-5} for 25 iterations, for a total of 1000 training iterations
Batch size	2^{14} (adjusted for training speed)
Regularization	L1 and L2 of 10^{-6}

Table 2: Hyper-parameters for the predictive approach (not extensively adjusted).

Hyper-parameter	Value
Hidden layers (fully connected)	Four layers with 256, 128, 64, 32, 16 and 8 units respectively with ReLU activation and He initialization
Output layer	Softmax activation with as many outputs as there are known teams, using uniform Glorot initialization
Loss	Categorical cross-entropy
Optimizer	Adam
Learning rate	10^{-2} for 350 iterations, 10^{-3} for 75 iterations, 10^{-4} for 50 iterations and 10^{-5} for 25 iterations, for a total of 500 training iterations
Batch size	2^{14} (adjusted for training speed)
Regularization	L1 and L2 of 10^{-6}

Table 3: Hyper-parameters for the discriminative approach (not extensively adjusted).

where an MLP is used to obtain s' instead of scanning past transitions directly.

During execution, for each known team:

1. The difference is predicted and added to the previous state;
2. The resulting predicted new state is clipped to the maximum and minimum values of the features;
3. A loss is measured between the real new state and the predicted one using the Huber loss.

This loss vector is then normalized.

4.2.1.2 Discriminating teams

Another alternative is to discriminate between the known teams. This is modeled as a multi-class classification problem. An MLP is trained (hyper-parameters in Table 3), where the inputs correspond to the previous 4 concatenated states, an action (one-hot encoded) and the new state. The outputs are one-hot encoded vector representations of the known teams, where the first position denotes the transition was drawn from the first team's replay buffer and so on. The training data is naturally balanced, as each team's policy has an associated replay buffer of equal size. The output of this MLP can then be used to obtain a loss vector as follows: $loss\ vector = 1 - output\ vector$.

4.2.1.3 Advantages and disadvantages of each alternative

The alternative based on discriminating teammate behavior is an easier problem and as such should result in a better loss. However, every time a new team is added to the library, it needs to be retrained. For development and testing purposes of this thesis, it means one MLP needs to be trained for each set of possible known teams that are explored. The alternative based on predicting teammate behavior is more versatile, being able to just add a new MLP for a new team. However, it is slower, as multiple MLPs need to be inferred during execution. The discriminative approach is shown as having better results in Subsection 5.4.

4.2.2 Belief updating

To update the agent's belief during execution, algorithms such as the Polynomial Weights Algorithm (PWA) [6] (1) (used in Barrett's work) or the update rule used in Exponentially Weighted Forecasters (EWFs) [7] (4) can be used. Higher values of η make the belief converge more quickly. EWF seemed to initially have better results than PWA in Subsection 5.4. A Bayesian update is not used for the discriminative approach (which gives probabilities) since, as pointed out in [4], a single update may set a belief to zero.

$$belief \leftarrow \frac{belief \cdot e^{-\eta \cdot loss}}{\sum belief \cdot e^{-\eta \cdot loss}}, \eta > 0 \tag{4}$$

4.2.3 Belief loss for unknown teams

It is not feasible to train the described MLP's during execution with samples acquired from the current team, as they are unlabeled data. As such, the belief loss for an unknown team must be obtained by other means.

Considering that the vector of belief losses for known teams sums to 1, if there is a team whose loss for the observed transition is smaller than 0.5 (in other words, more than half of the probability) then the agent probably is playing with that team. Otherwise, if there is no such team, then the agent probably is playing with an unknown team. As such, a simple heuristic approach is to use 0.5 as the loss for the unknown team: if there is a team where the model is confident across multiple transitions it should be identified, otherwise the agent should recognize the team as unknown. These results are smoothed across multiple transitions by the algorithms used for the belief update.

If the vector of losses of known teams summed to another value, the loss for an unknown team would thus be half of that value. This approach needs at least two known teams: with only one known team it would always have a loss of zero.

4.3 Policy for an unknown team

This subsection presents our approach to learning a policy for unknown teams via transfer learning. In face of an unknown team, the agent can start by acting with a policy learned within its library. Such policy will probably be better than acting randomly or starting to learn a new policy from scratch. One heuristic is to use the policy whose team the agent considers to be most similar to the unknown team. To do that, the agent can keep a second belief vector where

the loss of an unknown team is not considered (to avoid numerical issues in the main belief, such as individual beliefs possibly being zero). It can then act with the policy whose team has the highest value in that second belief vector. This is equivalent to ignoring the possibility of an unknown team.

Assuming the agent has better performance while playing with a teammate using a policy trained for it than a policy trained for another teammate, this strategy will produce non-optimal behavior. However, training a policy from scratch is also not desirable, as it would produce worse performance for an extended period of time. Ideally the agent would leverage its past training while improving by observing the team's behavior.

4.3.1 Adapting a selected pre-trained policy

One approach is to select a pre-trained policy and adapt it to the unknown team, using it as a source for parameter sharing transfer learning [31]. This selection can be done by using the second belief vector and selecting the team whose belief is highest after a certain amount of time steps (e.g. 1000). This policy is then used dynamically like the remaining.

However, training this policy online may result in some instability as the outputs of the network might change abruptly with just a few updates. As such, the following possible improvements to this transfer learning approach are also tested, in order to possibly improve the stability:

1. Use larger batch sizes, which may result in more similar gradients between updates;
2. Freeze the weights of the first hidden layer of the DQN, resulting in less weights to update; this is usual practice when employing parameter sharing transfer learning [31];
3. Reduce the learning rate, resulting in smaller updates;
4. Use samples from the selected known team, possibly avoiding overfitting in the initial replay buffer.

In order to perform exploration to learn more about the unknown teammate, the agent acts with the same 0.01 ϵ-greedy exploration presented in Table 1 when using this policy for acting, during PLASTIC-Policy's execution.

5. RESULTS

This section presents results that were used to validate the solution described in the previous section.

5.1 Evaluation procedure

This subsection presents the tasks and metrics used to evaluate the solution.

5.1.1 Task

The main task consists in repeatedly playing HFO episodes, where an ad hoc agent plays with a teammate belonging to one of the six possible teams (Subsection 1.2) in the offensive against two defensive HELIOS players, including a goalkeeper. This task varies in the teammate's team and in the ad hoc agent's behavior:

1. For training a policy where the ad hoc agent knows the teammate, it plays for up to 400 thousand episodes and creates a policy using the observed transitions (10 runs).
2. For testing the policy selection mechanisms, scenarios involving different subsets known teammates are evaluated (these subsets are: all teams known; an example with three known teams; an example with five known teams) (100 runs).
3. For testing the policy improved online for an unknown team, the agent is forced to treat the unidentified team as unknown and to adapt from a pre-determined policy (10 runs). The resulting policy is evaluated over 50 thousand episodes.

To obtain the policies that are used in further subsections after the policy hyper-parameter and modeling adjustment (Subsection 5.2), for each team the policy that scores the highest among the last 5 evaluations of each run is selected[2]. Each run uses a random seed.

5.1.2 Metrics

The fraction of episodes ending in goal is used to evaluate the performance of policies. It is equivalent to the goal percentage metric in Hausknecht *et al.* [14]. In order to evaluate our approach during the training phase, the agent is tested every 5000 episodes with the policy learned until that point (that is, with static DQN weights and no exploration) during 500 episodes. Each policy is trained for 10 independent runs (ideally more runs would have been performed).

The fraction of trials where the correct team has maximum belief (among all teams in the belief vector) is used for evaluating the policy selection. It is evaluated over the first 1000 to 10,000 time steps after meeting an unidentified teammate. This evaluation is performed for 100 trials.

The online policy improvement also uses the fraction of episodes ending in goal as a metric. It is evaluated without exploration every 1000 episodes, for 1000 episodes with static DQN weights. Each experiment is performed for 10 independent runs.

The plots used to present the results also show 95% confidence intervals, obtained using the bootstrap method [9] with 1000 resamplings[3].

5.2 Policy hyper-parameter and modeling adjustment

In this subsection, the learning of policies for known teams is adjusted according to Subsection 4.1. An HELIOS teammate is used as a representative to adjust hyper-parameters and the modeling.

Increasing the train frequency from every 4 time steps (Figure 3) to every time step (Figure 4) (less time steps between updates results in more frequent training) has no negative impact in the performance, but allows to train a policy in a smaller amount of episodes (400 thousand for the former and 200 thousand for the later). This also may result in some wall clock time savings, depending on the number of episodes used (e.g. around 33 hours instead of 36-38 for each run of Figure 4 and Figure 3, respectively).

The high level HFO state feature set (Figure 5) has significantly lower performance than the low level set (Figure 4). This may be caused by insufficient or inadequate features for DQN. One hypothesis is the angles: in the low level feature set they are encoded as a sinus-cosinus pair, avoiding

[2]Due to time constraints, the policies were selected only among the first 4 runs.

[3]As there are few runs, the results for a run are not averaged first – this results in smaller plotted errors than expected.

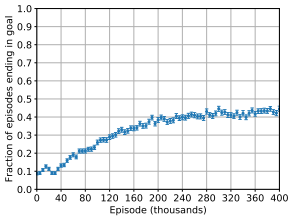

Figure 3: Train frequency of 4 time steps (low level state feature set; 512 ReLU).

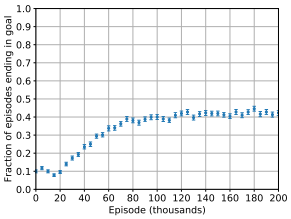

Figure 4: Train frequency of 1 time step (low level state feature set; 512 ReLU).

discontinuities, while in the high level feature set they are a single scalar.

The deeper topology with two hidden layers of 256 and 64 ReLU respectively (Figure 6) had improved results over a single layer of 512 ReLU (Figure 4) – the latter reached a performance of 0.4 to 0.45 starting at around episode 100 thousand, while the former barely went over 0.4. In general, deeper networks allow to learn a representation that is composed by simpler representations [13], which may present an advantage.

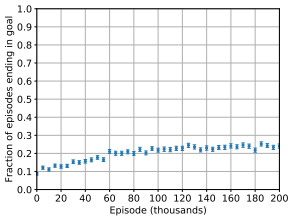

Figure 5: High level feature set (train frequency of 1 time step; 512 ReLU).

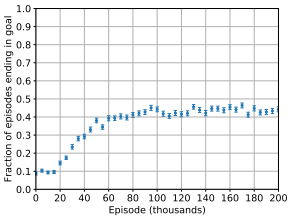

Figure 6: Hidden layers with 256 and 64 ReLU (low level state feature set; train frequency of 1 time step).

Regarding extensions to the original DQN [21], neither Double DQN [28], Dueling DQN [29] nor their combination improved the performance in this environment. Therefore, these extensions are not used in the following subsections.

In summary, relatively to Table 1 are used: two fully connected hidden layers of 256 and 64 ReLU respectively; and an update frequency of 1 time step.

5.3 Learning and evaluating policies

Team `Base` and `Gliders` reached an average fraction of episodes ending in goal of around 0.3 to 0.4 using the adjusted hyper-parameters and modeling, `AUT MasterMinds`, `Cyrus` and `HELIOS` a fraction of around 0.4 to 0.5 and `YuShan` around 0.5.

The policy that is used for each team in the following subsections was selected among the last 5 evaluations of each of the first 4 runs, as described in Subsubsection 5.1.1. Each policy was then compared against all teams for 1000 episodes for each combination. The results are present in Table 4. As can be observed, the most effective policy for each teammate is often the policy it was trained with, with the exception of

		Policy (the team it was trained with)					
		AUT	Base	Cyrus	Gliders	HELIOS	YuShan
Team	A	**0.481**	0.334	0.359	0.375	0.408	0.330
	B	0.336	**0.392**	0.367	0.363	0.298	0.389
	C	0.451	0.403	**0.481**	0.395	0.432	0.432
	G	0.351	0.380	0.375	**0.391**	0.365	0.390
	H	**0.467**	0.411	0.375	0.411	0.456	0.356
	Y	0.343	0.399	0.384	0.402	0.361	**0.530**

Table 4: Policy effectiveness (given by fraction of episodes ending in goal) when playing with all possible teams. The most effective policy for a given team is highlighted in bold. The rows are identified by the initial letter of the teams. Each combination played for 1000 episodes.

`HELIOS`. This result may suggest that some teams do not allow to explore certain plays as effectively as others.

5.4 Evaluating the policy selection within the ad hoc agent

This subsection presents the results of the policy selection mechanisms described in Subsection 4.2.

5.4.1 *Only known teams*

Once again, an `HELIOS` teammate is used as a representative to adjust the belief updating, between PWA and EWF (Subsubsection 4.2.2). For this purpose, the discriminative approach (Paragraph 4.2.1.2) is used, as it is more computationally efficient. The results with PWA can be observed in Figure 7. The results with the update rule used in EWFs can be observed in Figure 8 for $\eta = 1$ and Figure 9 for $\eta = 10$. The update rule used in EWFs with $\eta = 1$ seemed to perform the best up to the first 300 time steps, while PWA seemed to have slightly better results up to 700 time steps. As the first time steps are of higher concern in ad hoc teamwork, the EWF with $\eta = 1$ approach was chosen.

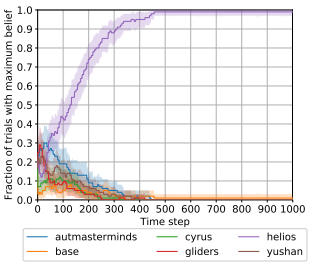

Figure 7: Discriminative approach, using PWA with $\eta = 0.5$.

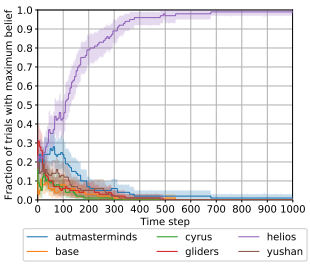

Figure 8: Discriminative approach, using EWF with $\eta = 1$.

The loss approach is then decided between the predictive approach (Paragraph 4.2.1.1 – Figure 10) and discriminative approach (Paragraph 4.2.1.2 – Figure 8), again using `HELIOS` as a representative. As can be observed, the discriminative approach identifies the correct team significantly faster (the predictive approach is plotted for 10 thousand time steps). This may be due to it being easier to learn compared to the predictive approach.

For the remaining teams, by around 400 time steps the agent correctly identified the team at least for 90% of the trials, using the update rule used in EWFs with $\eta = 1$ and the discriminative approach.

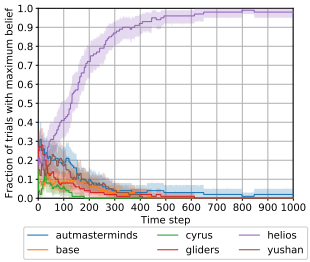

Figure 9: Discriminative approach, using EWF with $\eta = 10$.

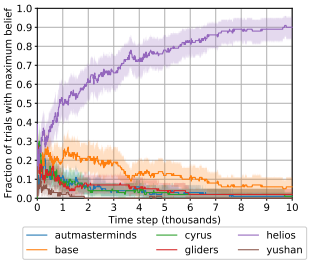

Figure 10: Predictive approach, using EWF with $\eta = 1$.

5.4.2 *Considering the possibility of an unknown team*

Considering an example of three known teams (the first three by alphabetical order - `AUT MasterMinds`, `Base` and `Cyrus`) and three unknown teams (`Gliders`, `HELIOS` and `YuShan` – identifiable as "unknown"), the agent can correctly identify all teams (tested for all six possible teams individually) for the majority of trials after 1000 time steps, except for `Gliders`, which it misidentifies as `Base` (Figure 11). One possibility is that the discriminative model used for the belief loss failed to create strict enough boundaries, such that most `Gliders` transitions fall within the boundaries of `Base` in the model and are classified as such. The agent correctly identified every known team at least for 90% of the trials by around 200 time steps, but the unknown teams it correctly identified were for only around 50% to 65% of the trials by around 1000 time steps.

Considering another example where five teams are known (all except for `Gliders`), there is an increase in identification performance for unknown teams (Figure 12). The agent correctly identified every team for 80% to 90% of the trials by around 500 time steps. Compared to three known teams, the agent believes a known team is unknown for more steps in the beginning (Figure 13 for three known teams and Figure 14 for five known teams). This may be due to having more possible teams, so the model gives more varied predictions, causing it to take longer to converge to a known team and having a greater bias for the unknown possibility. The results for the remaining known teams are similar to Figure 14.

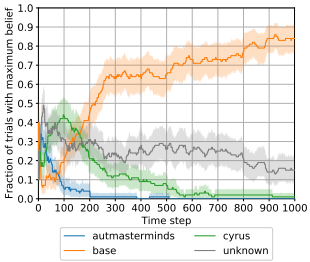

Figure 11: Three known teams (team `Gliders`).

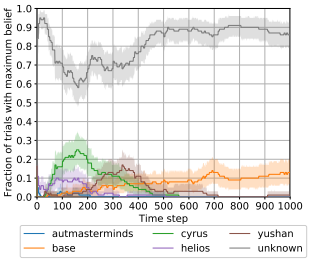

Figure 12: Five known teams (team `Gliders`).

5.5 Policy for an unknown team

In this subsection are presented the results of the approach for adapting a policy to an unknown team, based on parameter sharing transfer learning [31]. A case where the agent is playing with a `YuShan` teammate and only knows `AUT MasterMinds` is used for illustrative purposes (this is the case with the biggest possible growth according to Table 4). Adapting a pre-trained policy often results in achieving better performance faster (Figure 15) when compared to training a new policy from scratch (Figure 16), particularly in the initial episodes.

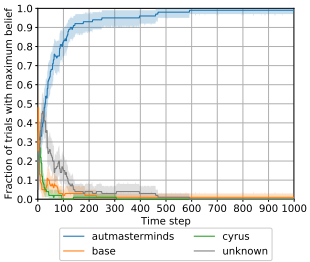

Figure 13: Three known teams (team `AUT MasterMinds`).

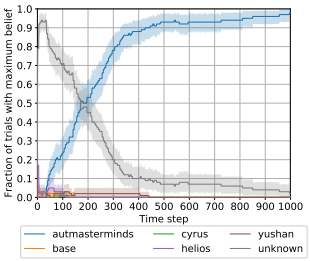

Figure 14: Five known teams (team `AUT MasterMinds`).

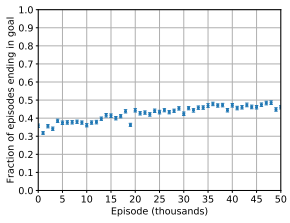

Figure 15: Adapting a policy from `AUT MasterMinds` to `YuShan`.

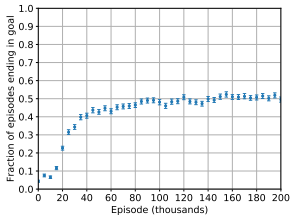

Figure 16: Training a policy for `YuShan` from scratch.

Some training instability can be observed when performing straightforward parameter sharing transfer learning (Figure 17 illustrates a particularly bad run of Figure 15). The following possible improvements to this approach are thus tested using the same `YuShan` case:

1. Using a larger batch size of 256 instead of 32. However, in order to compensate for the added computational cost and possibly reduce overfitting, the DQN updates are only performed every 4 time steps. Figure 18 presents a seemingly more steady growth than Figure 15, particularly in the first 20 thousand episodes, with only a small performance decrease (possibly related to the more infrequent updates).

2. Freezing the weights of first hidden layer of the DQN prevented the agent from improving. This may be related to the particular DQN topology that was used. The layer was kept frozen for the entire duration of the runs.

3. Reducing the learning rate to a fourth ($0.0000625/4$) resulted in smaller updates and thus a slower improvement, while still displaying some instability.

4. Initially using samples from the source policy did not help.

As such, we conclude that only using a larger batch size may present an advantage in this scenario.

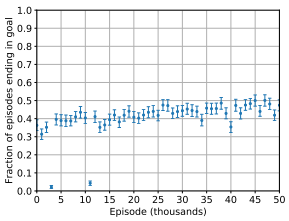

Figure 17: Adapting a policy from `AUT MasterMinds` to `YuShan`: a single run illustrating training instability.

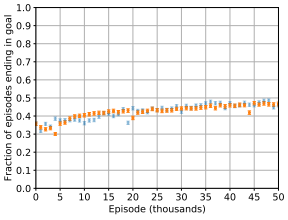

Figure 18: Adapting a policy from `AUT MasterMinds` to `YuShan`: batch size of 256 and training frequency of 4 time steps (Figure 15 repeated in blue).

6. CONCLUSION

This thesis aimed to address the problem of ad hoc teamwork when teammates may be unknown (never used for training). It was explored in the context of a two versus two match of HFO, where the ad hoc agent plays in the offense along with a teammate. For this purpose, an architecture based on PLASTIC-Policy [4] was developed. In this architecture, policies are learned for known teams via DQNs and added to a library of policies. A policy is then selected during execution with an unidentified team via a belief. Compared to PLASTIC-Policy, this architecture uses DQN instead of FQI for learning the policies, uses different mechanisms for identifying teammates (based on MLPs), extends these mechanisms to also identify if the team is unknown and adapts a policy from a known team to an unknown one.

To obtain a belief loss for a transition, two approaches were proposed: one based on predicting the next state (which depends on the team) and another based on directly discriminating its behavior among the known teams. These approaches are based on building a model using the transitions stored in the replay buffers of the DQNs. The discriminative approach proved to perform better, possibly for being easier to learn. However, it needs to be retrained every time a new team is added to the library. These belief losses are then combined across multiple transitions using an algorithm such as the update rule used in EWFs [7].

However, the original PLASTIC-Policy is unable to identify whether the unidentified team is in the library of known teams or not. To obtain the belief loss of a team being unknown these approaches cannot be used, as the transitions currently being observed are unlabeled (the agent does not know if they belong to previously encountered team or an unknown one). A simple approach based on an heuristic artificial belief was introduced, where the belief loss of the team being unknown is half of the sum of the known teams' belief losses: if a team has less than half of the total loss/more than half of the probability then the model is confident in it; otherwise, if all teams have less than half of the probability, then the model is not confident in any of them and the team probably is unknown. However, this approach is flawed since as more teams are introduced, there is less probability of the model identifying the same team consistently across multiple transitions, creating a bias for unknown.

If the ad hoc agent identifies a team as unknown, it then switches to a special policy improved online. As learning a new policy from scratch would be impractical in this setting, the ad hoc agent uses an existing policy as a source for parameter sharing transfer learning [31]. This source can be selected as the known team the agent considers to be most similar to the unknown team according to its belief (however, in Subsection 5.5 were chosen combinations of teams that allowed to observe the performance growth more easily). Several possible improvements were also tested, where increasing the batch size resulted in a more steady growth but in slightly reduced performance (possibly related to the more infrequent training that was used to compensate for the additional computational cost).

6.1 Limitations and future work

The first main limitation of this work is the simplistic unknown team detection. While it appears to work with some success in our experiments, its robustness has to be thoroughly tested. When there are many known teams it is expected that the model will have more varied predictions across multiple transitions, biasing the agent for the team being unknown. Future work on this area could involve applying more robust and well-studied techniques, namely in the field of anomaly detection [8]. Another possibility could be approaches based on the entropy of the transitions' belief losses.

The second main limitation of this work is the online policy adaptation. While the policy can remain reasonably stable, there was no meaningful observed improvement in the first moments (e.g. the first 1000 episodes), which are of greater importance in this setting. Future work in this area could involve studying some of the work mentioned in Yang Yu [30], namely using better exploration techniques [22] [12], using model-based methods for augmented DQN features [23] [24] and exploring meta-learning approaches [11].

7. REFERENCES

1. Akiyama, H., and Nakashima, T. Helios base: An open source package for the robocup soccer 2d simulation. In *RoboCup 2013: Robot World Cup XVII*, S. Behnke, M. Veloso, A. Visser, and R. Xiong, Eds., Springer Berlin Heidelberg (Berlin, Heidelberg, 2014), 528–535.

2. Akiyama, H., Nakashima, T., and Yamashita, K. Helios2013 team description paper. *RoboCup* (2013).

3. Barrett, S. *Making Friends on the Fly: Advances in Ad Hoc Teamwork*. PhD thesis, The University of Texas at Austin, Austin, Texas, USA, December 2014.

4. Barrett, S., Rosenfeld, A., Kraus, S., and Stone, P. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence* (October 2016).

5. Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

6. Blum, A., and Mansour, Y. *Learning, Regret Minimization, and Equilibria*. Cambridge University Press, 2007, 79–102.

7. Cesa-Bianchi, N., and Lugosi, G. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

8. Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM Comput. Surv. 41*, 3 (July 2009).

9. Efron, B. Bootstrap methods: Another look at the jackknife. *Ann. Statist. 7*, 1 (01 1979), 1–26.

10. Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res. 6* (dec 2005), 503–556.

11. Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR (International Convention Centre, Sydney, Australia, 06–11 Aug 2017), 1126–1135.

12. Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. Noisy networks for exploration, 2019.

13. Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

14. Hausknecht, M., Mupparaju, P., Subramanian, S., Kalyanakrishnan, S., and Stone, P. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop* (May 2016).

15. Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning, 2017.

16. Hester, T., and Stone, P. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine Learning 90*, 3 (Mar 2013), 385–429.

17. Kalyanakrishnan, S., Liu, Y., and Stone, P. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup-2006: Robot Soccer World Cup X*, G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, Eds., vol. 4434 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, 2007, 72–85.

18. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H. Robocup: A challenge problem for ai. *AI Magazine 18*, 1 (Mar. 1997), 73.

19. Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., Vylder, B. D., Saeta, B., Bradbury, J., Ding, D., Borgeaud, S., Lai, M., Schrittwieser, J., Anthony, T., Hughes, E., Danihelka, I., and Ryan-Davis, J. Openspiel: A framework for reinforcement learning in games, 2020.

20. MacAlpine, P., Genter, K., Barrett, S., and Stone, P. The robocup 2013 drop-in player challenges: Experiments in ad hoc teamwork. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), 382–387.

21. 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. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (Feb. 2015), 529–533.

22. Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration, 2018.

23. Pong*, V., Gu*, S., Dalal, M., and Levine, S. Temporal difference models: Model-free deep RL for model-based control. In *International Conference on Learning Representations* (2018).

24. Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc. (2017), 5690–5701.

25. Stone, P., Kaminka, G. A., Kraus, S., and Rosenschein, J. S. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence* (July 2010).

26. Stone, P., Sutton, R. S., and Kuhlmann, G. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior 13*, 3 (2005), 165–188.

27. Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*, second ed. Adaptive Computation and Machine Learning series. MIT Press, USA, 2018.

28. van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning, 2015.

29. Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. Dueling network architectures for deep reinforcement learning, 2015.

30. Yu, Y. Towards sample efficient reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization (7 2018), 5739–5743.

31. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. A comprehensive survey on transfer learning. *Proceedings of the IEEE 109*, 1 (2021), 43–76.