AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism

Shuo Chen,[1,2] **Ewa Andrejczuk,**[2] **Zhiguang Cao,**[3] **Jie Zhang**[1]
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2]ST Engineering - NTU Corporate Laboratory, Nanyang Technological University, Singapore
[3]Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore
chen1087@e.ntu.edu.sg, {ewaa, zhangj}@ntu.edu.sg, isecaoz@nus.edu.sg

Abstract

In the ad hoc teamwork setting, a team of agents needs to perform a task without prior coordination. The most advanced approach learns policies based on previous experiences and reuses one of the policies to interact with new teammates. However, the selected policy in many cases is sub-optimal. Switching between policies to adapt to new teammates' behaviour takes time, which threatens the successful performance of a task. In this paper, we propose AATEAM – a method that uses the attention-based neural networks to cope with new teammates' behaviour in real-time. We train one attention network per teammate type. The attention networks learn both to extract the temporal correlations from the sequence of states (i.e. contexts) and the mapping from contexts to actions. Each attention network also learns to predict a future state given the current context and its output action. The prediction accuracies help to determine which actions the ad hoc agent should take. We perform extensive experiments to show the effectiveness of our method.

Introduction

In the majority of the multiagent systems (MAS) literature tackling teamwork, a team of at least two agents shares cooperation protocols to work together and perform collaborative tasks (Grosz and Kraus 1996; Tambe 1997). However, with the increasing number of agents in various domains (e.g. construction, search and rescue) developed by different companies, we can no longer coordinate agents' activities beforehand. In this case, we expect a team of agents to perform a task without any pre-defined coordination strategy. This problem is known in the literature as the *ad hoc teamwork* problem (Stone et al. 2010). This paper tackles the ad hoc teamwork problem where an ad hoc agent plans its actions only by observing other teammates.

Most of the existing approaches for the ad hoc teamwork focus on simple domains, i.e. they ignore the environmental uncertainty or they assume teammates' actions are fully observable (Agmon, Barrett, and Stone 2014; Ravula, Alkoby, and Stone 2019). To the best of our knowledge, PLASTIC-Policy (Barrett et al. 2017) is the only scheme proposed for more complex domains, i.e. with a dynamic environment and a continuous state/action space. PLASTIC-Policy learns a policy for each past teammate type. When working with new teammates, it chooses one of the policies based on the similarity between the new and past experiences and uses that policy to select the sequence of actions for the ad hoc agent. However, the selected policy only works well when new teammates behave similarly to the chosen teammate type. When new teammates' behaviour becomes far from the teammate type, the ad hoc agent needs to switch to another policy. Switching between policies to adapt to new teammates' behaviour takes time, which threatens the successful performance of a task. Given that, it is desirable to develop a new method that adapts to new teammates' behaviours in real-time for complex domains.

In this paper, we propose to *Achieve the Ad hoc Teamwork by Employing Attention Mechanism* (AATEAM). AATEAM uses attention-based recurrent neural networks to cope with new teammates' behaviour in real-time. Our workflow is as follows. First, we use reinforcement learning to learn policies to work with past teammates. Second, we make the ad hoc agent play with each type of teammate using the corresponding learned policy. We collect the states (*input*) and the ad hoc agent's actions (*label*) as training data. Third, we use this data to teach one attention network for each type of past teammates. The attention network works as follows. In every step, the attention network takes a state as input. The attention mechanism inside extracts the temporal correlation between the state and the states encountered previously, i.e. *context*. It further maps the context to an action probability distribution. In every step, the attention networks also output a metric that measures the similarity between past and new teammate types. The more similar the types, the more important the corresponding probability distribution over actions. We aggregate networks' probability distributions over actions by calculating the weighted sum of those probabilities and its importance. Finally, we output an action with the highest aggregated probability. Since AATEAM evaluates the similarity in every step, we can adjust to the new teammates' behaviour in real-time.

In summary, we make the following contributions: 1) we propose AATEAM, an attention-based method to cope with teammates' different behaviours in complex domains. To the best of our knowledge, this is the first time attention mechanism is applied to this problem. 2) Besides learning the mapping from contexts to probability distribution over ac-

tions, we train the attention network to measure the similarity between the corresponding past teammates and new teammates. This helps to adjust the action selection in real-time. 3) We perform extensive experiments in the RoboCup 2D simulation domain. The results demonstrate that the attention networks outperform PLASTIC-Policy when working with both known and unknown teammates.

Related Work

The majority of existing work in the ad hoc teamwork is proposed for simple domains. They either do not consider the environmental uncertainty or they assume the action space is fully observable.

For the first simplification, the authors (Agmon and Stone 2012; Chakraborty and Stone 2013; Agmon, Barrett, and Stone 2014) assume that the utilities of the team's actions are given. Also, Genter, Agmon, and Stone (2011) deal with the role assignment problem for the ad hoc agent where teammates' roles are known. However, the actions' utilities and teammates' roles may vary when the environment changes. Therefore, their problem setting is too simplistic.

As for the second simplification, assuming a fully-observable action space limits the potential on real-life applications. The researchers develop algorithms to plan the ad hoc agent's action either by using the past teammates' behaviour types (Wu, Zilberstein, and Chen 2011; Albrecht and Ramamoorthy 2013; Barrett et al. 2013; Melo and Sardinha 2016; Chandrasekaran et al. 2017; Ravula, Alkoby, and Stone 2019) or by inferring the responsibilities new teammates are taking (Chen et al. 2019). In both cases, they assume that teammates' actions are fully observable, which is a strong assumption when it comes to more complex domains (e.g. the actions are continuous and hence, the ad hoc agent cannot know the exact value of an action).

The only scheme for complex domains is PLASTIC-Policy (Barrett et al. 2017). PLASTIC-Policy uses reinforcement learning to learn a policy for working with each type of past teammates. To reduce learning complexity, PLASTIC-Policy first assumes the existence of a countable set of high-level actions. Each high-level action is a mapping from the state to a continuous action. Therefore, it limits the action space to be searched by reinforcement learning. The policy takes a continuous state as input and outputs a high-level action of the ad hoc agent. As mentioned before, in complex domains, the ad hoc agent cannot directly observe teammates' actions. Therefore, PLASTIC-Policy defines the state transition for continuous state space and observes teammates' behaviour based on state transitions. Given a state transition with new teammates, the ad hoc agent locates the most similar state transition with each type of past teammates. Using that information, it updates the probability of new teammates being similar to each type of past teammates, and applies the policy for the most similar past teammates. The main drawback of PLASTIC-Policy is that the learned policies are not necessarily optimal for new teammates. Additionally, the update based on state transitions may not be fast enough to adapt to new teammates' changing behaviour. Our paper addresses these drawbacks by using all attention networks to select the best action based on a context and the similarity between past and new teammates' type.

Preliminary

Markov Decision Process

In this paper, our objective is to find the best set of actions for the ad hoc agent. Following the state-of-the-art literature (Barrett et al. 2017), we model the ad hoc teamwork problem as the Markov Decision Process (MDP) (Sutton and Barto 2011). Specifically, we represent an MDP by a tuple $\langle S, A, P, R \rangle$, where S is a set of states; A is a set of high-level actions the ad hoc agent can perform; $P : S \times A \times S \rightarrow [0, 1]$ is the state transition function; That is, $P(s'|s, a)$ is the probability of transiting to the next state s', given the action a and current state s; and the reward function $R : S \times A \rightarrow \mathcal{R}$ specifies the reward $R(s, a)$ when the ad hoc agent takes the action a in the state s. Note that for continuous state space, the definition of state transition is domain-specific.

A policy $\pi : S \rightarrow A$ takes a state s as input and outputs an action a. After taking the action a in the state s, the maximum long term expected reward is computed as:

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} Q(s', a') \quad (1)$$

where $0 < \gamma \leq 1$ is the discount factor for the future reward. An optimal policy chooses the action a that maximizes the $Q(s, a)$ for all s.

Attention Mechanism

The attention mechanism is an approach for intelligently extracting contextual information from a sequence of inputs. Due to its great performance, the attention mechanism has become a popular technique in computer vision (Mnih et al. 2014; Xu et al. 2015) and natural language processing (NLP) (Bahdanau, Cho, and Bengio 2014; Vaswani et al. 2017).

The example attention network for translation is shown in Figure 1. The attention network has two recurrent neural networks called encoder and decoder respectively. The attention network first encodes a full sentence with its encoder. The last hidden state from the encoder is the initial

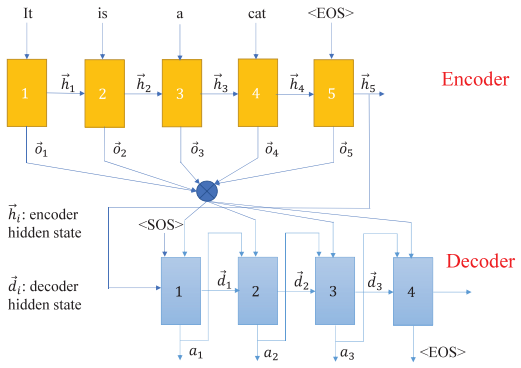

Figure 1: The attention network for translation

hidden state (input) to its decoder. To do the translation, the attention mechanism decides the importance (i.e. weights) of each input to the translation output. The weighted sum of encoder's outputs is the contextual information. The decoder's hidden state and the contextual information determine the output together. The method to compute the weight in the attention mechanism varies from problem to problem.

Note that in the teamwork, the ad hoc agent's action in the current step is correlated to the current state as well as to previous states. Given different teammates' behaviours (coming from different teammate types or different situations), the correlations of previous states to the current action are different. Therefore, the attention mechanism is a suitable approach that takes the sequence of states as input and outputs the correlations given different teammates' behaviours.

Ad Hoc Teamwork with Attention Mechanism

In this section, we explain the details of our method. First, we present the overall design of one attention-based neural network. Note that AATEAM contains multiple attention networks, one per past teammates' type. Second, we explain how AATEAM selects actions for the ad hoc agent. Finally, we describe how we train AATEAM.

The Design of the Attention Network in AATEAM

The overall architecture of our attention-based neural network is as shown in Figure 2. The architecture consists of two parts. That is, the encoder and decoder respectively. The encoder extracts important features for the ad hoc teamwork from the sequence of states. The decoder exploits the extracted features to output the probability distribution over actions for the ad hoc agent.

The encoder. The encoder consists of a linear layer and the gated recurrent unit (GRU)[1] (Cho et al. 2014) as shown in Figure 3. The initial hidden state of the encoder $\vec{h}_0$ is a vector of zeros $\vec{0}$. In every step t, the encoder takes the previous hidden state $\vec{h}_{t-1}$ and the state $\vec{s}_t$ as input. Let u denote the size of state vector, l denote the size of GRU's hidden state and $f_{x\to y}(\cdot)$ denote a linear layer that transforms a vector of size x into another vector of size y. A linear layer $f_{u\to l}(\cdot)$ transforms the state vector $\vec{s}_t$ into an embedding vector of length l. It does so, by multiplying the input by the weight matrix. Then, GRU uses the embedding vector and $\vec{h}_{t-1}$ to output a hidden state $\vec{h}_t$ and an encoder-output $\vec{o}_t$, i.e. the encoding of $\vec{s}_t$. The $\vec{h}_t$ and $\vec{o}_t$ then become the input of the decoder.

The decoder. The purpose of the decoder is to output a high-level action $a_t \in A$ for the ad hoc agent. Let $n = |A|$ denote the number of high-level actions. Next, we understand *an episode* as one instance of a task performance. We denote the signal of the start of an episode as $\langle SOE \rangle$ and set its index to $n+1$. The initial hidden state of the decoder $\vec{d}_0$ is the encoder's hidden state in step 1. $a_0 = \langle SOE \rangle$ and its index is input into the decoder in step 1 to signal the start of a task. In every step t, the decoder takes as input: 1) the set of encoder-outputs until step t, i.e. $\{\vec{o}_i | i = 1, ..., t\}$, 2) the previous hidden state $\vec{d}_{t-1}$, 3) the encoder's hidden state $\vec{h}_t$, and 4) the previous output action a_{t-1}. As shown in Figure 3, the decoder uses these inputs and the attention module to generate the inputs for its GRU. Then, the GRU outputs a hidden state $\vec{d}_t$ and a probability distribution over actions $\{P(a_t^k) | k = 1, .., n\}$ (we discuss how we select a_t from $\{P(a_t^k) | k = 1, .., n\}$ in the latter subsection).

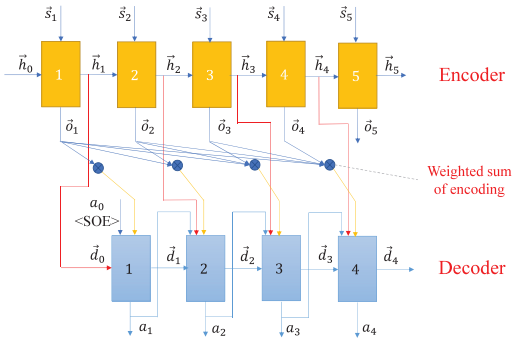

Figure 2: The attention network for the ad hoc teamwork

The attention module determines the weight of each $\vec{o}_i$ based on $\{\vec{o}_i | i = 1, ..., t\}$, $\vec{d}_{t-1}$, and $\vec{h}_t$ (we present the details of how to compute the weights in the following subsection). Given the attention weight set $\{w_{t,i} | i = 1, .., t\}$, the decoder produces the context vector $\vec{c}_t$ in step t as

$$\vec{c}_t = \sum_{i=1}^{t} w_{t,i} \cdot \vec{o}_i \tag{2}$$

Next, the decoder uses an embedding layer (that maps an action index to a vector of length l) and dropout layer (that reduces overfitting (Srivastava et al. 2014)) to generate the embedding of a_{t-1}, i.e. $Embed(a_{t-1})$. To produce the input in_t of GRU in step t, the concatenation (denoted by "$\|$") of $Embed(a_{t-1})$ and $\vec{c}_t$ goes through a linear layer $f_{2l\to l}(\cdot)$ and Relu layer. The Relu layer applies the rectified linear unit function to each element of the input vector:

$$Relu(x) = max(0, x) \tag{3}$$

We use the Relu function because, if the neuron gets activated, its gradient always equals to 1. This helps to train the network faster. Given in_t and $\vec{d}_{t-1}$, GRU outputs the decoder's hidden state $\vec{d}_t$ in step t and an action seed vector $\vec{a}_t$ of length l. To produce the probability distribution over actions in step t, i.e. $\{P(a_t^k) | k = 1, ..., n\}$, we first use a linear layer $f_{l\to n}(\cdot)$ that converts $\vec{a}_t$ into a vector $\{\tilde{a}_t^k | k = 1, .., n\}$. Second, we transform $\{\tilde{a}_t^k\}$ using a softmax layer. That is,

$$P(a_t^k) = \frac{e^{\tilde{a}_t^k}}{\sum_{m=1}^{n} e^{\tilde{a}_t^m}} \tag{4}$$

[1]The GRU is a popular variant of the recurrent neural network (RNN). As an alternative RNN, we could use the long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997). However, GRU has comparable performance to LSTM while having less parameters, which makes it easier to train (Chung et al. 2014).

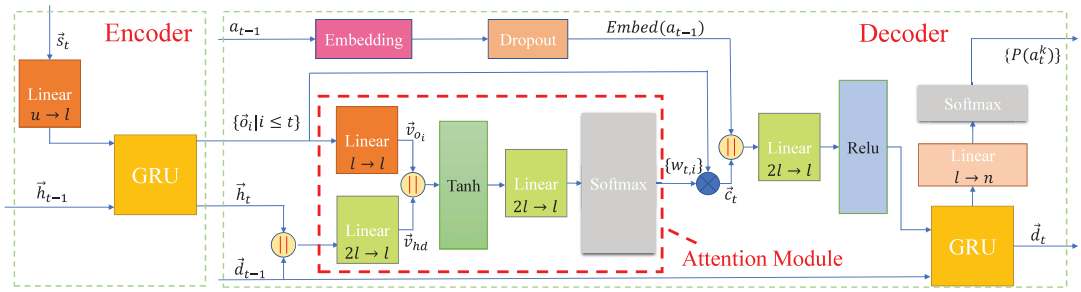

Figure 3: The detailed design of the attention network

where $P(a_t^k)$ represents the probability of taking action a^k in step t. **Attention Mechanism.** The attention module is shown in Figure 3. In every step t, the attention module takes the following input: 1) the set of encoder's outputs $\{\vec{o}_i | i \le t\}$; Each $\vec{o}_i$ naturally provides the information for the weights of $\{\vec{o}_i\}$; 2) The encoder's hidden state $\vec{h}_t$; The hidden state $\vec{h}_t$ contains the information of states until step t and thus, it helps to produce a new context; and 3) the decoder's previous hidden state $\vec{d}_{t-1}$; Note that $\vec{d}_{t-1}$ comes from a_{t-2} and $\vec{d}_{t-2}$. It also results from a_{t-3}, a_{t-2} and $\vec{d}_{t-3}$. Following this logic, $\vec{d}_{t-1}$ reflects the effects of the sequence of actions $\{a_i | i < t\}$ on $\vec{d}_0$, that is, on the information of initial state $\vec{h}_1$. The effects of previous actions on states are useful for generating $w_{t,i}$ since a_t is correlated with $w_{t,i}$.

Given the inputs, a linear layer $f_{2l \to l}(\cdot)$ transforms the concatenation of $\vec{h}_t$ and $\vec{d}_{t-1}$ into a vector $\vec{v}_{hd} = f_{2l \to l}(\vec{h}_t || \vec{d}_{t-1})$. Then, each $\vec{o}_i$ in $\{\vec{o}_i | i \le t\}$ goes through a linear layer to become a vector $\vec{v}_{o_i} = f_{l \to l}(\vec{o}_i)$. The concatenation of $\vec{v}_{o_i}$ and $\vec{v}_{hd}$ further goes through a Tanh layer (following Bahdanau, Cho, and Bengio (2014)) that applies the $tanh$ function to each element of the input vector:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

Next, the output of Tanh layer goes through a linear layer $f_{2l \to l}(\cdot)$ to output *the weight seed* $\bar{w}_{t,i}$ for $\vec{o}_i$ in step t. In summary, the weight seed $\bar{w}_{t,i}$ is computed as:

$$\bar{w}_{t,i} = f_{2l \to l}(Tanh(f_{2l \to l}(\vec{h}_t || \vec{d}_{t-1}) || f_{l \to l}(\vec{o}_i))) \tag{6}$$

Finally, a softmax layer takes the set of weight seeds $\{\bar{w}_{t,i} | i \le t\}$ as input and outputs the weight $w_{t,i}$ of each $\vec{o}_i$ in step t. That is,

$$w_{t,i} = \frac{e^{\bar{w}_{t,i}}}{\sum_{b=1}^{t} e^{\bar{w}_{t,b}}} \tag{7}$$

Selecting the Action in AATEAM

In the previous subsection, we described the design of one attention network. Each attention network outputs a probability distribution over actions. In this subsection, we discuss how to select the final action for the ad hoc agent given those probability distributions.

Formally, we denote the set of attention networks in AATEAM as $\mathcal{T}$. In every step t, we input the current state $\vec{s}_t$ and previous action a_{t-1} into each attention network. Note that each network has its own $\{\vec{o}_i\}$, $\vec{h}_{t-1}$ and $\vec{d}_{t-1}$ (computed by that network). Based on the input, each network j in $\mathcal{T}$ outputs a probability distribution over actions $\{P(a_t^k)\}_j$ as well as its $\vec{h}_t$ and $\vec{d}_t$. Note that the encoder's hidden state $\vec{h}_t$ results from $\{\vec{s}_i | i \le t\}$. Therefore, $\vec{h}_t$ contains the state information until the step t. Additionally, $\vec{d}_t$ reflects the effects of $\{a_i | i < t\}$ on $\vec{s}_1$ (as discussed in the Attention Mechanism subsection). Henceforth, we train each attention network such that $\vec{d}_t$ is the prediction of the state in step t based on previous states and actions (we discuss the details of the training in the next subsection). If $\vec{d}_t$ is close to $\vec{h}_t$, it means that the attention network's action affects the state as intended. Because the state also results from the teammates' behaviour, the distance between $\vec{d}_t$ and $\vec{h}_t$, i.e. $dist_j(\vec{h}_t, \vec{d}_t)$, implies the extent to which the attention network matches the teammates' behaviour. The more a network matches the teammates' behaviour, the more influential it should be to the action's selection. Henceforth, we can use the distance to weigh the probability distributions (one per attention network) and select the action with the highest weighted probability. We compute the weight of each network j as:

$$v_j = \frac{e^{-dist_j(\vec{h}_t, \vec{d}_t) \cdot 10}}{\sum_{z=1}^{|\mathcal{T}|} e^{-dist_z(\vec{h}_t, \vec{d}_t) \cdot 10}} \tag{8}$$

and the weighted probability distribution over actions as:

$$\{\bar{P}(a_t^k)\} = \sum_{j=1}^{|\mathcal{T}|} \{P(a_t^k)\}_j \cdot v_j \tag{9}$$

Finally, we select the action with the highest probability in $\{\bar{P}(a_t^k)\}$ as the action for the ad hoc agent in step t.

Training AATEAM

In this subsection, we discuss how we train one attention network j (for each teammate type j). First, we use reinforcement learning to learn a policy π_j for cooperating with past teammate type j. After that, we make the ad hoc agent use π_j to work with the teammate type j for a number of episodes. During the teamwork, we collect the state-action

pairs as training data. Let us denote $\hat{a}_t$ as the action that the ad hoc agent took in step t based on learned policy π_j. The state-action pair $(\vec{s}_t, \hat{a}_t)$ is the training instance of step t, where $\hat{a}_t$ is the label for the training. A training sample is a sequence of state-action pairs, formally $\{(\vec{s}_t, \hat{a}_t)\}_j$. For each state-action pair $(\vec{s}_t, \hat{a}_t)$ in a training sample, the attention network takes $\vec{s}_t$ as input and returns a probability vector over actions $\{P(a_t^k)|k = 1, ..., n\}$. The probability that the network selects the action $\hat{a}_t$ is $P(a_t^k = \hat{a}_t)$. We calculate the loss of an action using the negative log-likelihood:

$$loss_a = -log(P(a_t^k = \hat{a}_t)) \tag{10}$$

Notice that the higher the probability that the attention network selects $\hat{a}_t$, the smaller the loss of action $loss_a$.

Since $\{\hat{a}_t\}$ output by π_j fits the past teammate type j's behaviour, i.e. affects the state as intended, the attention network should learn to reduce the distance between $\vec{d}_t$ and $\vec{h}_t$. By doing so, we ensure that $\vec{d}_t$ makes good prediction of s_t given the teammate type j's behaviour. Hence, we compute the loss of distance as the square of $L2$-norm:

$$loss_d = \|\vec{d}_t - \vec{h}_t\|_2^2 \tag{11}$$

Then, the overall loss is an additive function of the above loss measures:

$$loss = loss_a + \beta loss_d \tag{12}$$

where β is the constant that adjusts the weight of $loss_d$.

We use the Adam optimizer (Kingma and Ba 2014) to update the network parameters to minimise the computed loss. Moreover, during the training, we apply the teacher forcing strategy (Williams and Zipser 1989), i.e. use $\hat{a}_t$ instead of a_t as the input to the decoder in the step $t + 1$. This is because the actual $\vec{s}_{t+1}$ results from $\hat{a}_t$ instead of a_t.

Comparison to the NLP domain. In the teamwork domain, the attention network needs to handle the sequence of inputs differently from the attention networks in the NLP domain (e.g. the Figure 1). This is because each domain has different information available. First, in the NLP domain, the encoder can encode the whole sentence before inputting anything into the decoder. In our domain, we do not know the states from the future. Hence, we can only use the state information until the current step. Additionally, the NLP decoder uses the last hidden state of the encoder as its initial hidden state such that the decoding is based on the whole sentence. In our domain, the decoder can only use $\vec{h}_1$ as its initial hidden state (since there is only $\vec{s}_1$ in the beginning of a task). Second, the attention network in NLP needs to learn when to end a sentence (i.e. output $\langle EOS \rangle$). However, in our domain, the end of episode is an external signal and thus, our network does not need to output it.

Experiments

In this section, we present the experiments we have done to demonstrate the effectiveness of AATEAM. First, we introduce the experiment domain. Second, we present how we collect the data. Finally, we pitch AATEAM against PLASTIC-Policy to show the advantage of our approach. In detail, we show the results of both algorithm when working with first, known and second, unknown teammates. Our results indicate that when working with both known and unknown teammates, in most cases our algorithm outperforms PLASTIC-Policy significantly.

Half Field Offence Domain

Following Barrett et al. (2017), we use a limited version of the RoboCup 2D domain, i.e. half field offense (HFO) domain (Hausknecht et al. 2016). HFO retains the complexity of the RoboCup 2D domain in terms of the state/action space and the uncertainty of actions' effects. The difference between both settings is that we play the game within a half of the field and the ad hoc agent always replaces one of the agents in the offence team. This simpler setting with single objective allows researchers to focus on developing algorithms that tackle the environmental complexity.

In this paper, we use the HFO version with two offensive agents that try to score a goal into opposite team's goal. The opposite team has two defenders including the goalie. For the defensive agents, we adopt the agent2D behaviour provided by Akiyama (2010). At the beginning of each episode, the ball is put in a random position close to the midline of full field. The initial position of the goalie is in the goal centre. The remaining agents start at random position within a given range. The ranges of offensive and defensive agents ensure that the offensive agents are initialised to be closer to the ball than the defender. In every step, each agent can observe the position of the ball as well as the others' positions. An episode ends in one of the following cases: 1) the offensive team scores a goal, 2) the ball leaves the field, 3) the defensive team captures the ball, or 4) the game lasts for 500 simulation steps. The success episodes are the episodes in which the offensive team scores a goal.

State. In every simulation step, the ad hoc agent gets the following state features:

- the ad hoc agent's (x, y) position and its body angle
- the ball's (x, y) position
- the distance and the angle from the ad hoc agent to the centre of the goal
- the ad hoc agent's largest opening angle to the goal
- the distance from the ad hoc agent to the closest defender
- the teammate's largest opening angle to the goal
- the distance from the teammate to the closest defender
- the opening angle for passing to the teammate
- the teammate's (x, y) position
- each defender's (x, y) position

Given our setting, i.e. one teammate and two opponents, the state is a vector of 18 elements. Note that we sort the defenders' position in the state vector based on their uniform number while the original simulator sorts those positions based on the distance between each defender and the ad hoc agent. This modification is to ensure that each element about a defender's position always corresponds to the same defender.

Action. The high-level actions that the ad hoc agent can perform are: 1) $Dribble$, 2) $Pass$, 3) $Shoot$, and 4) $Move$. In every step, the simulator informs the ad hoc agent if it is close enough to be able to kick the ball. If so, the ad hoc agent can select an action from $\{Dribble, Pass, Shoot\}$. When it is not close enough to the ball, it can only perform $Move$. Both learned policies and attention networks select high-level actions for the ad hoc agent when the agent can kick the ball. Given a high-level action, the simulator maps it to a continuous action to perform the simulation.

Collecting data for AATEAM. We use the State-Action-Reward-State-Action (SARSA) algorithm (Hausknecht et al. 2016) to learn a policy π_j for each past teammate type j. The ad hoc agent uses π_j to work with teammate type j for 200000 episodes. Within each episode, we collect the state-action pairs $\{(\vec{s}_t, \hat{a}_t)\}$ every time the ad hoc agent obtains the control of the ball and until it loses the ball. This sequence of $\{(\vec{s}_t, \hat{a}_t)\}$ is one training sample. We only store the state-action pairs from the success episodes.

Note that due to the continuous state space, the states of adjacent steps are very similar. Our initial experiments show that if we collect the state-action pairs in every step, the smoothness among the sequence of states impairs the attention network's performance. Henceforth, we set a threshold tr to detect the change of state. In every time step, we compare each element of current state (denoted as z') with the corresponding element of previous state (denoted as z). When at least one state element meets the condition $\frac{|z-z'|}{|min(z,z')|} > tr$, we collect the current state-action pair. Similarly, when using the trained networks to play, we select a new action when at least one state element meets the above condition. Otherwise, we continue the current action. Moreover, we remove the training samples that contain only one state-action pair. This is because attention networks learn to extract the temporal correlations of states. The samples with only one state have no temporal correlation inside and thus, hinder the learning performance.

Collecting data for PLASTIC-Policy. We collect state transitions (200000 episodes for each type) since PLASTIC-Policy uses them to update its probability distribution over the past teammate types. According to Barrett et al. (2017), a state transition $(\vec{s}, a, \vec{s}')$ means that either the possession of the ball changes from one agent to another agent or the episode ends when the ad hoc agent performs the action a. In the experiments, $\vec{s}$ is the state of the step when the ad hoc agent can kick the ball. $\vec{s}'$ is the state of the step either when another agent can kick the ball (this could result from intercepting, passing, or shooting) or the episode ends. Note that we cannot determine state transitions when the ad hoc agent cannot kick the ball because the simulator does not have a signal that indicates which specific agent holds the ball.

Teammate types. In this paper, we use the externally-created teammates. The teammates are not programmed to work with the ad hoc agent. We run the binary file released from the 2013 RoboCup 2D simulation league competition (RoboCup 2013). From these external agents, we use five types as past teammate types, i.e. aut, axiom, agent2D, gliders, and yushan. Additionally, we use five types as new teammate types, i.e. yunlu, utaustin, legend, ubc, and warthog.

Table 1: Experimental Parameters

Name	Value	Name	Value
β	0.1	tr	0.2
u	18	l	256
GRU Layer Number	5	Max length	60
Dropout rate	0.1	Learning rate	0.01

Experimental Parameters and Settings

We sum up the experiment parameters in Table 1. The number of hidden state layers for GRU is 5. We use the last layer of hidden states to compute the $dist_j(\vec{h}_t, \vec{d}_t)$ and $loss_d$. The maximum length of a state sequence that attention networks process is 60. When the state sequence's length exceeds 60, the networks drop the states from the beginning. The rate of dropout is 0.1. When training the attention networks, the default learning rate is 0.01 and the batch size is 32. In the experiments, we use AATEAM and PLASTIC-Policy to play with each teammate type for 10 trials where each trial contains 1000 episodes. We set the decision time limit for each step as 100 ms following Barrett et al. (2017). We evaluate the performance based on the scoring rate, i.e. how many episodes the offence team wins in every 1000 episodes. The error bars in Figure 4 and Figure 5 below are the standard deviations of scoring rates. Note that the results are discrete points. We add lines between them to improve clarity. We apply the binomial test to test the significance of our results (following Barrett et al. (2017)). We observe that the difference between AATEAM and PLASTIC-Policy is significant (95% confidence) except for the axiom and ubc types.

Playing with Past Teammates

In this subsection, we make the ad hoc agent play with the teammates it encountered before, i.e. past teammates. In detail, it plays with five past teammate types using first, AATEAM and second, PLASTIC-Policy. Also, we have one baseline and one upper bound here. The baseline is the performance of the original offence team, i.e. the team without the ad hoc agent. The upper bound is the performance of the ad hoc agent playing with a given teammate type following the learned policy for that teammate. Since each policy is learned only for the corresponding past teammate type, we expect the performance with the learned policies to be higher than any other approach. The results are shown in Figure 4.

In Figure 4, we see that both AATEAM and PLASTIC-Policy are better than the baseline. This is because the reinforcement learning algorithms managed to learn good policies for past teammate types (both PLASTIC-Policy and during the training of AATEAM). On top of that, the attention networks successfully learned to fit the behaviour of past teammates. Hence, both algorithms exhibit more intelligent behaviours.

When it comes to the upper bound, we can see that AATEAM's performance is closer to the upper bound than

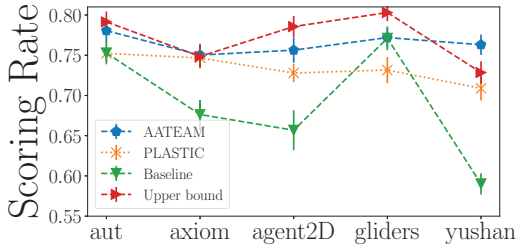

Figure 4: The performance with known teammates

PLASTIC-Policy in most of the cases. Also, AATEAM outperforms PLASTIC-Policy significantly. The reason is as follows. PLASTIC-Policy finds the corresponding policy for the known teammate type by evaluating the similarity between past experiences and current ones. However, it may not be able to identify the most similar past experience due to the decision time limit (i.e. 100 ms). Thus, the similarity evaluation may cause it to switch between different learned policies. This degrades its performance. In comparison, AATEAM extracts the context in every step and maps the context to the best action. Note that the extraction and mapping using trained attention networks are very efficient. Therefore, AATEAM can take advantage of the knowledge from the learned policies in a more flexible way. In detail, the ad hoc agent uses AATEAM to learn to cope with several different teammate types. Next, it plays with teammates of one of the known types. Each attention network outputs a probability distribution over actions. AATEAM aggregates the probabilities from all attention networks. The aggregation of the probability distributions enables a better usage of the previous knowledge (as we base our decision on more knowledge than the knowledge about only one past teammate type). Note that the originally learned policies (upper bounds) come from reinforcement learning and they may not be optimal. Given a teammate type, some policies for another types may suggest a better action than the policy for the given type. Therefore, AATEAM can perform better than the upper bounds as it uses more information from neural networks. In summary, we observe that AATEAM can easily adapt to the teammates' behaviour.

Playing with New Teammates

In this subsection, the ad hoc agent uses AATEAM and PLASTIC-Policy to work with the new teammates, i.e. the teammates it has never encountered before. Naturally, we have one baseline. That is, the baseline where an original team plays without the ad hoc agent. We cannot use the upper bound as the ad hoc agent did not learn the policies to work with the new teammates beforehand.

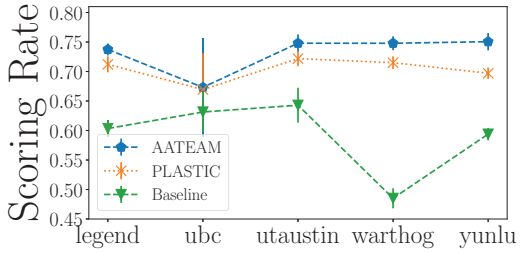

Figure 5: The performance with unknown teammates

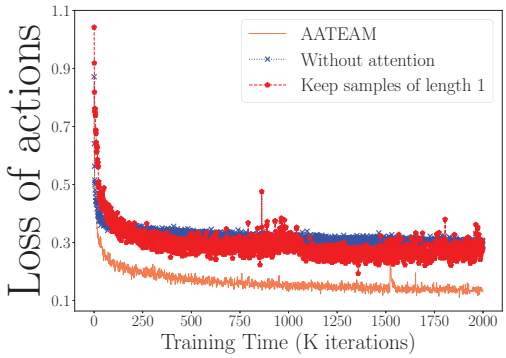

Figure 6: The loss of different training cases (using yushan type as example)

The results in Figure 5 show that both AATEAM and PLASTIC-Policy outperform the baseline as both algorithms behave more intelligently than the replaced agent. Moreover, in most cases AATEAM still significantly outperforms PLASTIC-Policy. The reason is that PLASTIC-Policy switches between policies based on the current behaviour of new teammates. Since switching takes time, PLASTIC-Policy cannot adapt to teammates' behaviours quickly. In addition, the performance of both AATEAM and PLASTIC-Policy is good except when the ad hoc agent plays with the ubc type. The reason is that the ubc agent does not often pass the ball to its teammate. Therefore, even though AATEAM and PLASTIC-Policy are smarter than its original teammate, they cannot help much. In summary, the experiments of playing with unknown teammate types demonstrate that AATEAM can adapt to different teammates' behaviours better than PLASTIC-Policy. This is crucial for the ad hoc teamwork in complex domains.

The Importance of Attention Mechanism

In this subsection, we show the importance of the attention mechanism in AATEAM. Besides the architecture shown in Figure 3, we also try to train the neural network without the attention mechanism, i.e. each o_i has the same weight. Additionally, we try to include the training samples of length 1 during the training. The loss of actions in different cases are shown in Figure 6. The results demonstrate that we can get a good network only when we adopt the attention mechanism and remove the training samples of length 1 (to prevent the attention module from being influenced). Note that the loss of actions directly affects the performance of neural network. Henceforth, the results in Figure 6 indicate that the attention mechanism is an essential part in AATEAM.

Conclusions and Future Work

This paper proposes a novel approach called AATEAM to solve the ad hoc teamwork problem in complex domains.

To the best of our knowledge, this is the first attempt to solve the ad hoc teamwork problem using attention-based recurrent neural networks. In detail, AATEAM contains a set of neural networks, each of them taught the behaviour of one different past teammate type. Using the probability distributions output by each neural network, AATEAM selects the best action for the ad hoc agent in real-time. We pitched our algorithm against PLASTIC-Policy that is the only scheme proposed for the ad hoc teamwork problem in complex domains. Our results indicate that when working with both known and unknown teammates, in most cases our algorithm outperforms PLASTIC-Policy.

One limitation of AATEAM is that it needs plenty of data to train the attention networks. Additionally, it is difficult to know if the architecture of neural networks is the most appropriate one. As future work, we will test AATEAM for more teammate types to show the robustness of our method. Furthermore, we plan to test the same idea for even more complex domains (with larger action and search space).

Acknowledgments

The research was partially supported by the ST Engineering - NTU Corporate Lab through the NRF corporate lab@university scheme.

References

Agmon, N., and Stone, P. 2012. Leading ad hoc agents in joint action settings with multiple teammates. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 341–348.

Agmon, N.; Barrett, S.; and Stone, P. 2014. Modeling uncertainty in leading ad hoc teams. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 397–404.

Akiyama, H. 2010. Agent2D base code release. https://osdn.net/projects/rctools/.

Albrecht, S. V., and Ramamoorthy, S. 2013. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 1155–1156.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Barrett, S.; Stone, P.; Kraus, S.; and Rosenfeld, A. 2013. Teamwork with limited knowledge of teammates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 102–108.

Barrett, S.; Rosenfeld, A.; Kraus, S.; and Stone, P. 2017. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence* 242:132–171.

Chakraborty, D., and Stone, P. 2013. Cooperating with a Markovian ad hoc teammate. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 1085–1092.

Chandrasekaran, M.; Doshi, P.; Zeng, Y.; and Chen, Y. 2017. Can bounded and self-interested agents be teammates? Application to planning in ad hoc teams. *Autonomous Agents and Multi-Agent Systems* 31(4):821–860.

Chen, S.; Andrejczuk, E.; A. Irissappane, A.; and Zhang, J. 2019. ATSIS: Achieving the ad hoc teamwork by sub-task inference and selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 172–179.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Genter, K. L.; Agmon, N.; and Stone, P. 2011. Role-based ad hoc teamwork. In *Proceedings of the AAAI Conference on Plan, Activity, and Intent Recognition*, 17–24.

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

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

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Melo, F. S., and Sardinha, A. 2016. Ad hoc teamwork by learning teammates' task. *Autonomous Agents and Multi-Agent Systems* 30(2):175–219.

Mnih, V.; Heess, N.; Graves, A.; and Kavukcuoglu, K. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, 2204–2212.

Ravula, M.; Alkoby, S.; and Stone, P. 2019. Ad hoc teamwork with behavior switching agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 550–556.

RoboCup. 2013. The binary file for the agents. https://archive.robocup.info/Soccer/Simulation/2D/binaries/RoboCup/.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Stone, P.; Kaminka, G. A.; Kraus, S.; and Rosenschein, J. S. 2010. Ad hoc autonomous agent teams: Collaboration without precoordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1504–1509.

Sutton, R. S., and Barto, A. G. 2011. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, 5998–6008.

Williams, R. J., and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1(2):270–280.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for ad hoc autonomous agent teams. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 439–445.

Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; and Bengio, Y. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, 2048–2057.