Automated Feature Engineering for Contextual Bandits via AATention Transfer Learning

Ethan Pedersen
Computer Science Department
Brigham Young University
Provo, United States
ethanp55@byu.edu

Jacob W. Crandall
Computer Science Department
Brigham Young University
Provo, United States
crandall@cs.byu.edu

***Abstract*—This paper explores transfer learning for contextual bandit problems using a method for automatically generating checker values used in Assumption-Alignment Tracking (AAT) [1]. In the contextual bandit problem, agents have access to various behavior generators which rely on certain assumptions. AAT tracks these assumptions to help an agent to evaluate how current conditions will impact its performance. While AAT has been shown to be effective in contextual bandit problems, the identification of assumptions and the effort required to create programs that check their veracity can be tedious and time-consuming. In this paper, we employ an attention-based neural network, which we call the AATention network, that uses domain transfer learning to generalize checkers created in one domain for use in other (previously unseen) domains. We evaluate this method in several multiagent environments. While imperfect, as any approach is, empirical results show that the AATention network can effectively facilitate transfer learning to previously unseen environments in multiple scenarios.**

***Index Terms*—AAT, AATention, Transfer Learning**

I. Introduction

At each time step of a task, bandit algorithms select an action from among N possible choices, often called "arms", with the goal of maximizing expected rewards (over time) when the consequences of each choice are stochastic and unknown beforehand [2]. While many traditional bandit algorithms offer performance guarantees, like any class of algorithm, previous work has identified key limitations of such approaches [2]. In addition to having shortcomings related to tuning on asymptotic performance measures (e.g., no regret [3], [4], [5]) and difficulties adapting to non-stationary domains (e.g., the CFR algorithm [6] in a repeated two-player prisoner's dilemma game, where the opponent might change strategies over time), one of the most glaring is the fact that traditional bandits ignore external information about the environment that might be beneficial in the decision-making process. Contextual bandits are designed to address these challenges.

Of particular interest to this work are contextual bandits with expert advice. Rather than model each arm as an action, the arms of these bandits are *experts*, or *generators*, designed to receive contextual information as input and generate actions as output. The bandit aims to follow the advice of the best generator given the current context at any particular time step. These bandits are arguably more generalizable than other contextual bandits, as the generators can represent complex rules, trained machine learning predictors, etc., which makes working with larger state and action spaces more feasible.

However, bandits with expert-advice are also imperfect. While the generators themselves use contextual information, the bandit typically does not use contextual information when choosing which generator to follow. This is arguably a significant weakness, as each generator likely only performs well in a subset of all possible contexts.

To boost the performance of contextual expert-advice bandits (hereafter referred to as "contextual bandits" for brevity) in complex, non-stationary environments, recent research has used Assumption-Alignment Tracking (AAT) [1]. AAT is a proficiency self-assessment algorithm in which the system designer identifies key assumptions an algorithm relies on and creates simple assumption checker programs that regularly estimate the veracity of those assumptions. The agent then uses these assumption estimates to predict the performance of an algorithm or generator. Recent work has found that AAT allows contextual bandit algorithms to frequently and effectively update their beliefs about actions as their environment changes. More specifically, AAT allows a contextual bandit to (1) use contextual information when selecting the best generator (and the generators also use contextual information) and (2) consider possible future rewards in its predictions. AAT is relatively new, but has shown positive results (e.g., [1], [7], [8]).

While promising, AAT suffers from the fact that it is essentially a principled (but manual) feature engineering framework (described more in the next section). This means that designers who wish to apply AAT in their problem(s) must invest significant upfront cost in terms of time and effort to properly set up AAT for the desired domain. To help address this problem, the focus of this paper is to explore the possibility of using a deep neural network to leverage AAT efforts done in one domain to other (previously unseen) domains.

In particular, we employ domain transfer learning using a deep learning mechanism. Transfer learning is a machine learning technique where a model developed for a specific task is reused as a starting point for another model in a second task [9]. Domain transfer learning can be viewed as a subset of transfer learning, where a model trained in one domain is adapted for use in a different but related domain. In recent

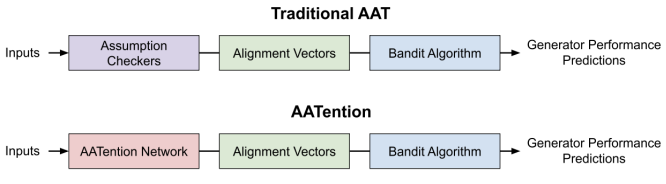


Fig. 1: Traditional AAT process compared to the new AATention process. Alignment vectors are produced from checkers (top) or AATention (bottom). The vectors are passed to contextual bandit algorithms, which use them to predict generator performance, which in turn guides generator selection.

years, domain transfer learning has found popularity in natural language processing and computer vision (e.g., [10]).

The work in this paper involves training an attention-based (multi-head) neural network, called AATention, in at least one domain with hand-crafted AAT alignment vectors. The AATention network is then used to generate alignment vectors to be used by contextual bandits in new domains. This proposed process, compared to the traditional AAT process, is shown in Figure 1. Note that the key difference between the two is the source of alignment vectors, wherein traditional AAT uses manually created *alignment checkers*, whereas the proposed AATention process uses output from the *AATention Network*. Our objective is to begin to understand when and how algorithms that use AATention outputs can compete with algorithms that use hand-crafted AAT vectors, as this would allow designers to more easily apply AAT to new domains.

We begin by reviewing AAT.

II. Background: Assumption-Alignment Tracking

Assumption-Alignment Tracking (AAT) [1] is a proficiency self-assessment algorithm originating in the robotics literature. AAT is used to predict the performance of a *generator*, which is an algorithm designed to perform a particular task. At each time step t, a generator G takes as input an encoding of the state of the world, denoted s_t^G, and outputs some action a_t to be executed in that time step.

AAT leverages the idea that a generator is created under certain assumptions about the environment and the agent. As such, the performance of the generator depends to a large degree on these assumptions being met. Thus, to predict the future effectiveness of a generator, the agent must be aware of the extent to which these assumptions are, and will continue to be, true. This can be accomplished by continually tracking the veracity of the assumptions upon which the generator relies, and then using these assessments to adjust performance predictions (using some machine learning model).

To track the veracity of the assumptions made in the construction of generator G, veracity assessments are calculated using alignment checkers. An alignment checker is a program that continually tracks the veracity of an assumption. Each assessment can be stored in a vector $\mathbf{x}_t^G$, the veracity assessment vector of generator G at time t. This makes AAT a principled feature engineering process, as assumption estimates are features that are estimated via checkers and passed to a learning model used to predict generator performance.

Despite being relatively new, AAT has been successful in multiple applications, including robotic navigation, anomaly detection, and selecting behaviors in repeated games [1], [7], [8]. However, a key weakness of AAT is the fact that a designer must manually identify assumptions and create alignment checkers for the domain they are working with. This can be a demanding, time-consuming, and somewhat subjective process, which is a common problem not only with AAT but with manual feature engineering in general [11]. Automating the checking of assumptions could improve AAT's useability.

Deep learning offers one potential solution to this challenge. At the core of deep learning lies the artificial neural network, a multi-layered structure inspired by the human brain [12]. One of the key strengths of deep learning is its ability to learn complex, nonlinear relationships between inputs and outputs [13]. Of particular interest for this work is the fact that deep learning models have been used for automatic feature selection. For example, convolutional neural networks (CNNs) extract image features from raw pixels and autoencoders aim to learn a compressed version of a dataset.

III. The AATention Network

Creating a generalizable neural network that can be used to automate AAT feature engineering across a variety of domains/problems via domain transfer learning requires that we solve two fundamental challenges.

The first challenge is that there are variations around how one might choose to represent environment state vectors. For example, two domains might be nearly identical, but designers could order the elements in their state representation vector s_t^G differently. Furthermore, the dimensions (i.e., the number of features) of s_t^G might vary depending on the designer and environment. The AATention network use two mechanisms to address this challenge. First, attention has been shown to help neural networks focus on the content of each element in their input vector(s) rather than the specific positions [14], [15]. Networks with attention can learn which elements are most relevant for predictions, regardless of order. Thus, we incorporate attention into our network. To help the network's attention mechanism learn effectively, we augment data by randomly changing the order of vector elements in the training data. This produced additional data samples for the network to learn from. Second, to address the problem of different vector dimensions, we allow for a maximum vector size of 300. Vectors with fewer than 300 dimensions are zero-masked (as is common practice employed in transformer text inputs). We feel that 300 is an ample size, allowing for more complex environments to be used in future projects, as existing work on AAT has traditionally only required small sizes (e.g., [1], [7], [8], [16]).

The second challenge to making the attention network generalizable is that the output dimension of the model (i.e., the number of assumption estimates to produce) can vary

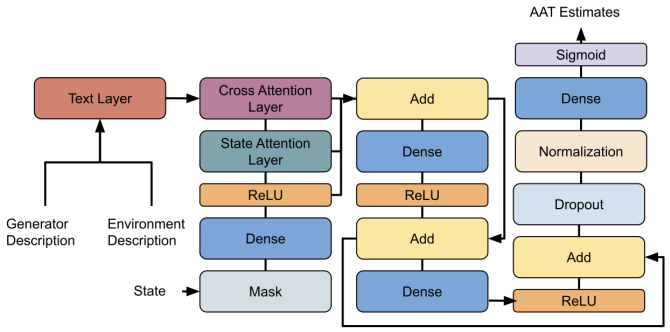


Fig. 2: The AATention network.

depending on the number of assumption checkers created by the designer. To combat this problem, we train the model to always output a fixed-size vector (maximum number of assumptions). We then use a post-processing step with a mask that sets irrelevant outputs (those beyond the actual number of assumptions for the specific task) to zero. We used an output dimension of 100. We again feel that this is more than sufficient, as previous work on AAT typically creates only a handful of checkers for each generator.

With these considerations in mind, we now present the AATention network, which is summarized in Figure 2. Implementation details (number of attention heads, dimension sizes, etc.) can be found in SM-1 and the supplied code. The AATention network contains 1,184,877 trainable parameters (approximately 4.52 MB of memory); compared to most modern deep neural networks that use attention (e.g., large language models), this is an extremely small size, allowing designers to quickly and efficiently use AATention.

The AATention network receives as input a string of text that represents the description of the generator G (for which the network should create an alignment vector) and a string of text that describes the domain/environment that the generator is operating in. These two strings are meant to provide the network with adequate context. The strings are passed to a custom text layer that processes both strings separately. This custom layer tokenizes and embeds the strings, adds positional encodings to both, performs self-attention on both, and outputs two self-attention matrices. The full architecture of this custom layer can be found in SM-2.

A numeric vector s_t^G, representing generator G's view of the current state of the environment at time t, is also passed as input. The vector is masked to account for varying dimension sizes, passed through a dense layer, and finally passed through a custom state attention layer. This custom layer applies positional encodings and self-attention to the masked and transformed state vector and outputs a self-attention vector. The architecture of this custom layer can be found in SM-3.

Once both strings and the state vector have been processed, they are passed to a custom cross attention layer. This layer performs cross attention between (1) the generator description self-attention matrix (one of the outputs from the custom text layer) and the state self-attention vector and (2) the environment description self-attention matrix (the second output from the custom text layer) and the state self-attention vector. The motivation behind these architectural decisions was to allow the network to learn how generator and environment descriptions relate to the current state. The vectors from the two cross-attention executions are added with the original state self-attention vector that was passed as input, and the resulting vector is returned. Details about this layer can be found in SM-4.

After the cross-attention layer, the remainder of the AATention network consists of a series of dense layers, skip connections, dropout, and normalization. The final dense layer uses a sigmoid activation function [17] to produce a 100-dimensional vector. AAT estimates are traditionally encoded with values between 0 and 1 to quantify how much an assumption is believed to be satisfied or violated. A sigmoid function is a convenient activation layer for this purpose, as it forces its output to fall in this range. A sigmoid function also has the advantage that the sum of the outputs does not need to be 1 (i.e., a probability distribution), which is not a requirement for AAT alignment vectors.

As a brief example of how AATention works and can be useful, imagine a repeated two-player prisoner's dilemma game. The first player wants to cooperate but the second player tends to defect consistently. The first player has a set of generators at their disposal that play different strategies and wants to use AAT to pick the best generator. Instead of needing to create alignment checkers for each generator, the first player can provide a description of a given generator, a description of the game, and a representation of the current state of the game (ideally capturing the fact that the second player often defects). AATention should produce alignment vectors that encapsulate the sentiment that generators that focus on cooperation likely will not perform well in this scenario due to the nature of the game and the behavior of the second player.

IV. Training and Testing

This section describes how we train the AATention network, the contextual bandit algorithms we evaluate in each domain, how we measure performance, and the domains in which we evaluate the algorithms.

A. Training the AATention Network

We trained the AATention network in two different domains: the Junior High Game (JHG) [18] and a repeated two-player chicken game (payoff matrix shown in Table I). We chose to train the ATTention network using these two domains as they represent different levels of complexity. The JHG is more complex, and thus requires a substantial number of alignment checkers when using AAT. On the other hand, the chicken game offers a more simplistic domain, thus requiring fewer checkers when using AAT. By seeing how the AATention network performs when trained in complex and simple domains, we can gain a better understanding of when and how well it generalizes to new domains.

TABLE I: Payoff matrix for chicken.

		Player 2	
		Swerve	*Straight*
Player 1	*Swerve*	$(0, 0)$	$(-1, 3)$
	Straight	$(3, -1)$	$(-5, -5)$

For both domains, we generated training data that contained state vectors s_t^G, veracity assessment vectors $\mathbf{x}_t^G$, simple text descriptions of each generator G, and simple text descriptions of the domain. To help with different possible orderings of individual s_t^G features, we performed data augmentation by randomly changing the elements of the s_t^G vectors. 30% of the training data was used for validation, where the best model weights were saved only if validation performance improved. We used the $\mathbf{x}_t^G$ vectors as labels, the Adam optimizer with an initial learning rate of 0.001, and calculated loss via the mean-squared error function; masking was required for the loss calculation, as the network outputs vectors with 100 elements and $\mathbf{x}_t^G$ vectors have different dimensions depending on the generator G. To help evaluate the impact of the training data, we performed experiments across three training sets: (1) data from only the JHG, (2) data from only the repeated chicken game, and (3) data from both the JHG and the repeated chicken game. Results of training on each of these three sets are described in Section V.

B. Contentual Bandit Algorithms

We created three contextual bandit algorithms that use alignment vectors generated from the AATention network and compared them to a fourth contextual bandit algorithm that uses regular AAT vectors (i.e., those generated from alignment checkers instead of the AATention network). The four algorithms are listed in Table II. AlegAATr, introduced in [8], is an expert-advice bandit that uses alignment vectors produced from manually created checkers to predict the performance for each of its generators. To make performance predictions, AlegAATr uses K-Nearest Neighbors (KNN) [19] with $k = 15$. AlegAAATr is identical to AlegAATr, but uses the AATention network in lieu of the manually created alignment checkers to form its context (the extra "A" stands for "AATention"). SMAlegAAATr is similar to AlegAAATr, but uses the AATention network and a fully-connected neural network "head" to directly predict generator performance (the "SM" stands for "single model", as it does not pass the alignment vectors to a KNN model for performance predictions). A simple diagram of this AATention and head combination is given in Figure 3. The penultimate dense layer of this network has an output dimension of 32 and the final dense layer has an output dimension of 1 (for performance predictions). Finally, AlegAAATTr is identical to AlegAAATr, except that it uses the fine-tuned AATention network piece (not the head) from SMAlegAAATr (the extra "T" stands for "tuned").

Each of these algorithms must be trained to make generator performance predictions from the alignment vectors they use in the domain in which they operate. AlegAATr, AlegAAATr, and AlegAAATTr train a KNN model for each of their generators, while SMAlegAAATr simply trains its neural network (since it uses a fully connected head). The specific AAT training processes for these algorithms are similar to [8] and is implemented in the code supplied in the supplementary material. We also give a brief summary of the AAT training processes for each test/evaluation domain in the next subsection.

TABLE II: Contextual-bandit algorithms compared in our studies.

Algorithm	**Description**
AlegAATr	Uses alignment vectors produced by alignment checker programs for context.
AlegAAATr	Uses alignment vectors from the AATention network for context.
SMAlegAAATr	Uses the AATention network for context and a fully-connected neural network "head" to directly predict generator performance (Figure 3).
AlegAAATTr	Uses alignment vectors from the fine-tuned AATention network from SMAlegAAATr for context (but does not use the fully-connected head).

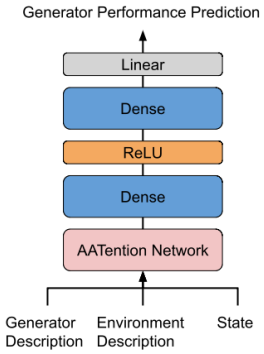


Fig. 3: SMAlegAAATr architecture (AATention with a head).

C. Evaluation Domains

We evaluate the performances of the contextual bandit algorithms in three domains: foreign exchange market day trading; repeated, two-player games; and a mixed-motive Grid Stag Hunt domain. Table III provides a brief description of each category.

The foreign exchange (forex) market is the global financial market in which individuals and organizations engage in currency trading. Widely regarded as the world's largest market, estimates suggest that the forex market has an average daily trading volume of approximately $6.6 trillion [20]. We trained and tested the algorithms on ten years of historical market data, spanning from November 2013 to November 2023, on the hourly (H1) EUR/USD currency pair. The generators and training and testing processes used in our experiments are identical to those found in [16].

Over the last several decades, playing repeated games has been theoretically interesting to many researchers, as they pose compelling learning challenges. Additionally, repeated games can abstractly model relevant scenarios, including wireless networks [21] and human-robot relationships [22]. Specifically, we trained and tested our algorithms in three two-player repeated games: prisoner's dilemma, chicken, and

TABLE III: Domains used in evaluations.

Domain	Description	AAT Training
Foreign Exchange Market simulator (forex)	A complex, realistic, zero-sum (or, arguably, negative-sum) domain in which currency pairs are traded.	Algorithms are trained over ten years of EUR/USD H1 data.
Repeated Games	Simple yet theoretically interesting domains that can have real-world applications.	Algorithms play against cooperators and defectors in three repeated games.
Grid Stag Hunt	A mixed motive domain in which three hunters must either compete for a hare or work together to capture a stag.	Algorithms are paired with simple expert-advice bandits and agents that either only hunt stag or only hunt hare.

coordination. Game payoff matrices can be found in SM-5 (the payoff matrix for the chicken game is shown in Table I). The duration of each game was set to 50 rounds. The specific generators and training and testing processes were the same as those found in [8] and can be found in our code. In short, the algorithms in Table II were trained against their generators, which included cooperators and defectors. The algorithms were then tested against cooperators, defectors, agents that change their strategies halfway through the game (round 25), agents that randomly cooperate and/or defect, self-play (i.e., a copy of the algorithm being tested), and other expert-advice bandits (BBL [23], EEE [24], and S++ [25]).

We also explored a two-dimensional, discretized grid environment called the Grid Stag Hunt, which is a modification of the ad hoc teamwork problem described in [26]. In essence, three hunters can either compete to capture a hare or work together to capture a stag. A stag is worth more than a hare, but a hunter must determine if they trust the other two to also pursue the stag instead of the hare. Specifics of the generators and training and testing processes are found in the supplied code. In short, the algorithms in Table II were trained against hunters that only hunt the stag, hunters that only hunt the hare, and simple expert-advice bandit hunters (they choose their target based on their generator performance predictions). The algorithms were tested against hunters that only hunt the hare, hunters that only hunt the stag, hunters that only hunt the hare but try to do so in a more efficient manner via planning, hunters that only hunt the stag but do so with planning, and self-play (i.e., all three hunters use independent instances of the algorithm being tested).

D. Metrics

To evaluate the performance of each algorithm under each test condition, we explored two metrics. The first is our primary metric and is simply average reward. We use this metric to compare the contextual bandits that use the AATention network (AlegAAATr, AlegAAATTr, and SMAlegAAATr) to those that do not (AlegAATr). If algorithms that use the AATention network can achieve similar performance to AlegAATr in domains in which the AATention network is not trained, we would argue that the AATention network is successful in learning generalizable alignment vectors.

The second metric is an adaptability score that is specific to mixed-motive domains. Mixed-motive domains are those in which agents are faced with both cooperative and competitive incentives and must choose between them. The repeated games and Grid Stag Hunt are examples of mixed motive domains; accordingly, we calculated an adaptability score, defined in detail in SM-6, for each algorithm in these domains (but did not for forex, as it is considered a zero-sum domain). The adaptability score is formed using two different measures of regret: *competitive regret* and *cooperative regret*. Competitive regret (denoted R_{comp}) measures how well an algorithm performs in scenarios when other agents are not inclined to cooperate, while cooperative regret (denoted R_{coop}) measures how well an algorithm performs when paired with other algorithms that are inclined to cooperate. Competitive and cooperative scores (denoted $\mathcal{S}_{\text{comp}}$ and $\mathcal{S}_{\text{coop}}$, respectively) are then calculated by normalizing R_{comp} and R_{coop}, respectively, to values between 0 and 1, where values closer to 1 indicate better performance. Finally, the *adaptability score* is calculated as $\mathcal{S}_A = \min(\mathcal{S}_{\text{comp}}, \mathcal{S}_{\text{coop}})$. We argue for the minimum (rather than, say, the average) in order to distinguish algorithms that adapt effectively to those that do well only in specific scenarios, as an agent that is adaptable should have high scores in both categories.

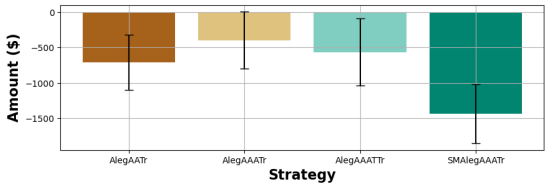
V. Results

We now evaluate the ability of the AATention network to generalize to previously unseen domains given different training mechanisms. Additional results and figures can be found in SM-7.

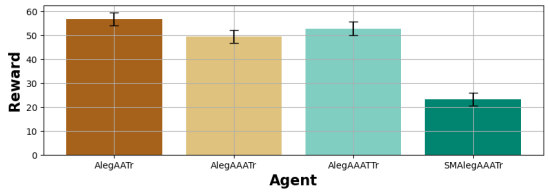
A. Training the ATTention Network on the JHG

We initially ran our experiments by training the AATention network using data from the JHG, as it is more complex than the repeated chicken game. Average performances for AlegAATr, AlegAAATr, AlegAAATTr, and SMAlegAAATr, respectively, in the forex market, repeated prisoner's dilemma, repeated chicken game, repeated coordination game, and Grid Stag Hunt, respectively, can be found in Figure 4. The results show that AlegAATr has higher average payoffs than the other algorithms in each domain (except forex). However, the differences between AlegAATr, AlegAAATr, and AlegAAATTr are only statistically significant in Chicken ($p < 0.001$ for all comparisons based on Tukey-Kramer tests). SMAlegAAATr consistently performs worse than the other algorithms (results are statistically significant in nearly every case). When considering AlegAAATr vs. AlegAAATTr, we observe that their performances are roughly equal in each domain.

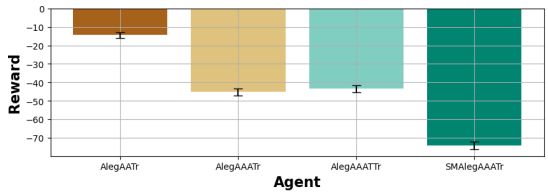
In short, when trained in the JHG, these results indicate that the performance of AlegAAATr and AlegAAATTr was on par with that of AlegAATr in four of the five previously unseen domains, with the only significant performance gap occurring in the repeated chicken game.



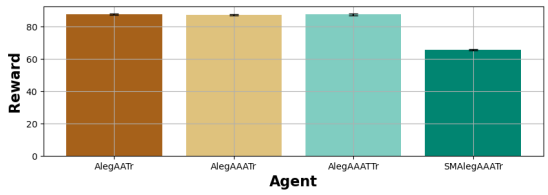
(a) Forex



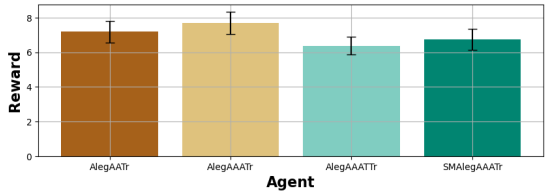
(b) Prisoner's



(c) Chicken



(d) Coordination



(e) Grid Stag Hunt

Fig. 4: Average results for each algorithm across the evaluation domains when AATention was trained only on JHG data. Black bars indicate standard error of the mean.

B. Training on the Chicken Game

Given AATention's seemingly poor performance in the repeated chicken game, we tested to see how training in that single domain would impact performance. The second section of Table IV shows the average rewards and standard errors for each algorithm in each domain when the AATention network is trained in the chicken game. The table shows that both AlegAAATr and AlegAAATTr improved substantially compared to when they were trained in the JHG (first section), though AlegAATr's average rewards remained significantly higher than both algorithms in the chicken game ($p < 0.001$). Interestingly, SMAlegAAATr did not change much. Furthermore, results for the other domains remained roughly the same. Overall, it appears that training on data from the chicken game either boosted or maintained performance across the domains; this is something we hoped to see specifically for the chicken game, as an algorithm should perform well in a domain in which it has been trained.

C. Training on the JHG and the Chicken Game

Considering the minor enhancements we saw when training on data from the repeated chicken game data, we felt it would be worthwhile to explore results when the AATention network is trained on data from both the JHG and the chicken game. The bottom section of Table IV shows that the performances for each algorithm in each domain were roughly the same as those when the AATention network is only trained on chicken game data (the middle section of the same table).

However, we also calculated each algorithm's defect, cooperate, and adaptability scores in each test domain (except the forex market) when the AATention network was trained on data from both the JHG and repeated chicken game. The algorithms' average cooperate and adaptability scores, respectively, in the repeated chicken game are shown in Figure 5 (figures for other domains can be found in SM-7). We chose to focus on the chicken game because that is the domain in which AATention-based algorithms struggled the most, and we feel that the cooperate and adaptability scores, as well as the game's payoff matrix, can shed some light as to why.

The figure shows that the bandit algorithms that use the AATention network achieved poor cooperative performance in this game (especially SMAlegAAATr). This translates into smaller adaptability scores. In other words, for this specific domain, the AATention network struggled to provide context that leads to cooperation in scenarios where it would have been beneficial to do so. They were not as adaptable/flexible as AlegAATr, which appears to contribute considerably to the performance gap in this game. If we recall the payoff matrix, found in Table I, we observe that the best form of cooperation given this payoff matrix results from alternating between swerving and going straight, for an average reward of $\frac{-1+3}{2} = 1$. This is a somewhat complicated strategy that is perhaps more difficult to recognize, as it requires alternating generator selection each round. While bandits that use the AATention network seemed to struggle with this, AlegAATr was able to use hand-crafted assumption checkers, tailored to the chicken game, that allowed it cooperate very effectively (achieving nearly a perfect score).

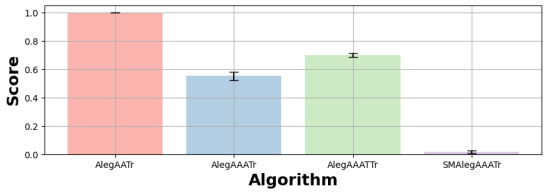
Do the AATention bandits struggle with adaptability in the other domains? Table V shows each algorithm's average compete, cooperate, and adaptability scores for every domain (except the forex market) when AATention was trained on both JHG and chicken game data. From this table, we can see that AlegAAATr and AlegAAATTr were as adaptable as AlegAATr in the other games on average. This reinforces that the AATention network has some ability to transfer learn to previously unseen domains.

TABLE IV: Results by domain, algorithm, and training condition. (±) values indicate standard errors.

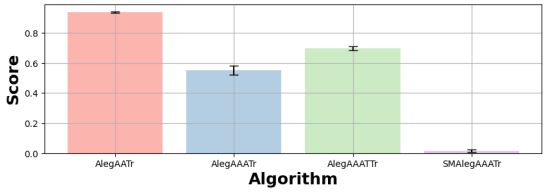
Training	Algorithm	Forex		Prisoner's		Chicken		Coordination		Grid Stag Hunt	
JHG	AlegAATr	-709.8	(± 386.9)	56.9	(± 2.7)	-14.4	(± 1.5)	87.6	(± 0.5)	7.0	(± 0.7)
	AlegAAATr	-395.6	(± 402.9)	49.5	(± 2.8)	-45.2	(± 1.9)	87.2	(± 0.5)	7.7	(± 0.6)
	AlegAAATTr	-564.2	(± 473.2)	52.9	(± 2.8)	-43.5	(± 1.8)	87.4	(± 0.5)	6.4	(± 0.5)
	SMAlegAAATr	-1437.6	(± 416.4)	23.2	(± 2.7)	-74.1	(± 2.0)	65.6	(± 0.3)	6.7	(± 0.6)
Chicken	AlegAATr	-709.8	(± 386.9)	56.9	(± 2.7)	-14.4	(± 1.5)	87.6	(± 0.5)	7.0	(± 0.7)
	AlegAAATr	-303.2	(± 664.6)	43.3	(± 3.0)	-26.9	(± 1.4)	87.3	(± 0.5)	8.2	(± 0.8)
	AlegAAATTr	-188.0	(± 637.1)	49.3	(± 2.8)	-24.1	(± 1.4)	85.8	(± 0.5)	7.9	(± 0.8)
	SMAlegAAATr	-1253.8	(± 625.9)	18.9	(± 2.7)	-72.8	(± 2.0)	64.9	(± 0.3)	6.6	(± 0.6)
Both	AlegAATr	-709.8	(± 386.9)	56.9	(± 2.7)	-14.4	(± 1.5)	87.6	(± 0.5)	7.0	(± 0.7)
	AlegAAATr	212.9	(± 463.8)	55.6	(± 2.9)	-31.5	(± 1.5)	86.6	(± 0.5)	9.2	(± 0.8)
	AlegAAATTr	-592.7	(± 610.8)	66.6	(± 3.0)	-23.9	(± 1.3)	86.3	(± 0.5)	8.7	(± 0.8)
	SMALegAAATr	-1434.6	(± 562.8)	21.2	(± 2.7)	-74.3	(± 2.0)	64.9	(± 0.3)	6.6	(± 0.6)

TABLE V: Competitive ($\mathcal{S}_{\text{comp}}$), cooperative ($\mathcal{S}_{\text{coop}}$), and adaptability ($\mathcal{S}_A$) scores for each algorithm in each domain.

	Prisoner's			Chicken			Coordination			Grid Stag Hunt		
	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$	$\mathcal{S}_{\text{comp}}$	$\mathcal{S}_{\text{coop}}$	$\mathcal{S}_A$
AlegAATr	0.929	0.691	0.691	0.937	1.00	0.937	0.980	1.000	0.980	0.233	0.417	0.100
AlegAAATr	0.818	0.843	0.800	0.869	0.552	0.552	0.980	0.971	0.971	0.083	0.717	0.017
AlegAAATTr	0.881	1.00	0.881	0.930	0.698	0.698	0.967	0.978	0.966	0.100	0.683	0.067
SMAlegAAATr	0.578	0.634	0.561	0.401	0.017	0.017	0.712	0.636	0.635	0.300	0.250	0.117

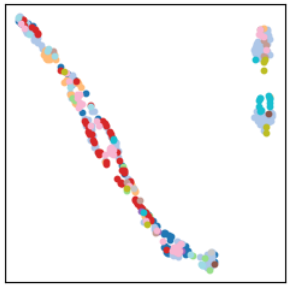


(a) Average cooperate scores, with standard errors.

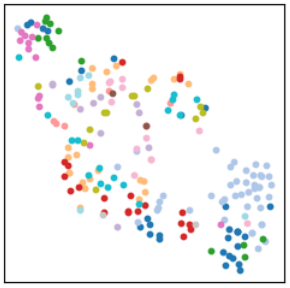


(b) Average adaptability scores, with standard errors.

Fig. 5: Average cooperative and adaptability scores, respectively, with standard errors (black bars), for each algorithm in the repeated chicken game.



(a) AlegAAATr vectors.



(b) AlegAATr AAT vectors.

Fig. 6: Compressed AAT vectors (via t-SNE [27]) for AlegAAATr and AlegAATr, respectively, when trading in the forex market. Colors represent which generator was selected.

VI. Discussion

Based on the results from the previous section, we feel it would be beneficial to highlight a few key observations. One important point is that the domain in which AATention is trained might have an impact on performance, as observed in some of the gains obtained by training on either the chicken game or both the chicken game and JHG. Regardless of training data, perhaps the most important outcome is for AATention to learn how to effectively map states and text descriptions to a unique vector space. For example, AlegAAATr had the highest average performance in the forex market (though, with no statistical significance). A comparison of its learned alignment vectors to those of AlegAATr (Figure 6) gives insights into why. While the groups/clusterings are different between the two, we can see that the AATention network seemed to learn an at least somewhat unique mapping from inputs to AAT vectors, which likely helps contribute to its performance in the forex market (and other domains).

Another important observation is that, across all evaluations, AlegAAATr and AlegAAATTr are essentially equal in terms of performance. Based on this, we would argue that spending the extra time and effort to fine-tune the AATention network on the domain in which one wishes to apply it does not seem worthwhile.

Additionally, AlegAATr was consistently the best performer (though, aside from the chicken game, there is little statistical significance). This is not terribly surprising, as AlegAATr operates on hand-crafted checkers created for the domain in question. These hand-crafted features allow AlegAATr to effectively achieve more difficult behavior for a specific domain, such as cooperation in the repeated chicken game. If obtaining the absolute "best" performance in a single domain

is a designer's primary concern, creating and using checkers in place of AATention is probably more promising. One also loses interpretability with AATention, as it is typically much easier to understand the output of alignment checkers than to understand the output of a deep neural network. If that is another significant concern, the traditional approach would again be more suitable.

Despite the arguments in favor of the traditional AAT process, the overall performance of AlegAAATr is competitive with that of AlegAATr in most of the domains. Deployment is also much easier and faster, as one does not need to plan and craft multiple alignment checkers for each domain. It also appears that, in most cases, adaptability is not a concern, unless certain strategies are complex and cannot be easily detected without hand-crafted features (e.g., cooperation in the repeated chicken game).

VII. Conclusion

In this paper, we explored the use of a deep neural network to automate the creation of Assumption-Alignment Tracking (AAT) alignment vectors via domain transfer learning. Instead of burdening a designer with identifying assumptions and creating alignment checkers in each unique domain, we train the AATention network from alignment checkers created in at least one domain, and then use the AATention network to produce context in previously unseen domains. Empirical evidence across three game categories indicates that, in the many cases, the AATention network allows algorithms to perform as well as they perform with hand-crafted (domain specific) alignment vectors. However, the AATention network seems to suffer when more complex strategies are needed; these are cases where tailored, hand-crafted features might be preferred. Future work is needed to tease out the effects of training domains and network architecture.

VIII. Supplementary Material

The technical appendix and code used in the experiments can be found at https://github.com/ethanp55/auto_aat/tree/ICMLA2025-SM.

References

[1] A. Gautam, T. Whiting, X. Cao, M. A. Goodrich, and J. W. Crandall, "A method for designing autonomous robots that know their limits," in *Proceedings of the International Conference on Robotics and Automation*, 2022, pp. 121–127.

[2] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: the adversarial multi-armed bandit problem," in *Proceedings of the 36th Symposium on the Foundations of Computer Science*, 1995, pp. 322–331.

[4] Y. Chang and L. P. Kaelbling, "Hedge learning: Regret-minimization with learning experts," in *Proceedings of the 22nd International Conference on Machine Learning*, 2005, pp. 121–128.

[5] M. Bowling, "Convergence and no-regret in multiagent learning," in *Advamces in Neural Information Processing Systems*, 2004, pp. 209–216.

[6] T. W. Neller and M. Lanctot, "An introduction to counterfactual regret minimization," in *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence*, vol. 11, 2013.

[7] X. Cao, J. W. Crandall, E. Pedersen, A. Gautam, and M. A. Goodrich, "Proficiency self-assessment without breaking the robot: Anomaly detection using assumption-alignment tracking from safe experiments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2023, pp. 12 714–12 720.

[8] E. Pedersen and J. W. Crandall, "AlegAATr the bandit," in *Proceedings of the* 26^{th} *European Conference on Artificial Intelligence*, 2023, pp. 1867–1874.

[9] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Proceedings of the 27th International Conference on Artificial Neural Networks*. Springer, 2018, pp. 270–279.

[10] G. Csurka, "A comprehensive survey on domain adaptation for visual applications," *Domain adaptation in computer vision applications*, pp. 1–35, 2017.

[11] Y. Jiang, N. Bosch, R. S. Baker, L. Paquette, J. Ocumpaugh, J. M. A. L. Andres, A. L. Moore, and G. Biswas, "Expert feature-engineering vs. deep neural networks: which is better for sensor-free affect detection?" in *Proceedings of the International Conference on Artificial Intelligence in Education*, 2018, pp. 198–211.

[12] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., 2022.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 2048–2057.

[16] E. Pedersen and J. Crandall, "Can a machine learning model consistently learn profitable trading strategies in the forex market?" in *2025 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2025.

[17] I. Goodfellow, "Deep learning," 2016.

[18] J. Skaggs, M. Richards, M. Morris, M. A. Goodrich, and J. W. Crandall, "Fostering collective action in complex societies using community-based agents," in *Proceedings of the 33rd International Joint Conference on Artificial Intelligence*, 2024, pp. 211–219.

[19] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

[20] L. Babypips.com, "Forexpedia forex (fx)," *babypips* – https://www.babypips.com/forexpedia/forex, 2023, accessed: 2023-11-13.

[21] D. T. Hoang, X. Lu, D. Niyato, P. Wang, D. I. Kim, and Z. Han, "Applications of repeated games in wireless networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2102–2135, 2015.

[22] M. Oudah, V. Babushkin, T. Chenlinangjia, and J. W. Crandall, "Learning to interact with a human partner," in *Proceedings of the 10th ACM/IEEE International Conference on Human-Robot Interaction*, 2015, pp. 311–318.

[23] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*. The MIT Press, 1998.

[24] D. P. De Farias and N. Megiddo, "Combining expert advice in reactive environments," *Journal of the ACM*, vol. 53, no. 5, pp. 762–799, 2006.

[25] J. W. Crandall, "Towards minimizing disappointment in repeated games," *Journal of Artificial Intelligence Research*, vol. 49, pp. 111–142, 2014.

[26] S. Barrett, P. Stone, and S. Kraus, "Empirical evaluation of ad hoc teamwork in the pursuit domain," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, 2011, pp. 567–574.

[27] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.