Received October 1, 2020, accepted October 18, 2020, date of publication October 26, 2020, date of current version November 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033725

Efficient Deep Learning Bot Detection in Games Using Time Windows and Long Short-Term Memory (LSTM)

MICHAIL TSIKERDEKIS, (Senior Member, IEEE), SEAN BARRET, RALEIGH HANSEN, MATTHEW KLEIN, JOSH ORRITT, AND JASON WHITMORE
Department of Computer Science, Western Washington University, Bellingham, WA 98225, USA
Corresponding author: Michail Tsikerdekis (michael.tsikerdekis@wwu.edu)

ABSTRACT Bots in video games has been gaining the interest of industry as well as academia as a problem that has been enabled by the recent advances in deep learning and reinforcement learning. In turn several studies have attempted to establish bot detectors in various video games. In this article, we introduce a bot detection model that can implemented in real-time and provide feedback on whether a player that is being observed is a bot or human. The model uses a limited feature set and amount of time of observation in order to be small and generalize easily to other domains. We trained and tested our model in a series of replays for Starcraft: Brood War and have yielded a higher accuracy than past studies and a fraction of detection time.

INDEX TERMS Video games, deep learning, bot, detection.

I. INTRODUCTION

As the video game industry continues to grow and the competition for online experiences becomes more fierce, the need for accurate detection of non-human players that present themselves as human players has grown as well. A game's economic lifespan can be curtailed by adversaries that use bots (computer controlled players) who can harass or be far superior to human players [1]. The prevalence of bots in online gaming necessitates and presents an opportunity to create and improve upon existing systems for bot detection. This study developed a bot detection method while also presenting an approach that can balance accuracy with computational overhead. Our research utilized datasets from *Starcraft: Brood War*, a real-time strategy game originally released by Blizzard Entertainment in 1998 and a remastered version followed in 2017. The game involves two players competing against each other by using a fictional civilization (e.g., Terrans, Protoss and Zergs) having the control of multiple buildings, units and research trees. The game has a top-down perspective and units can move freely on the map depending the mouse orders that a player gives to one or multiple units. The winner is the player that manages to destroy all of his or her opponents' units and buildings.

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

Presenting an accurate detection system for a real-time strategy game depends on an understanding of bot behavior during gameplay, which comes from the study of current state-of-the-art bot techniques. The use of statistical methods that compare player behavior to statistically common behaviors exhibited by bots has been found to be a successful strategy for bot detection [1], [2]. Self-similarity, the tendency for bots to repeat the same behavior, has also been noted as a useful means of distinguishing bots from human players [3], [4].

Over time, sophisticated attackers developed bots that have mechanisms for detection avoidance. AI competitions in StarCraft demonstrate an ability for bots to make intelligent, ad hoc decisions in the way a human might. Machine learning as well as human analysis have been used to study these AI bot techniques in order to provide varying perspectives on a sophisticated set of decision making methods [5]–[7]. The availability of large datasets have in turn aided in the development of bots capable of making complicated decisions [8]. For example, dealing with incomplete information and attempting to anticipate another player's actions [9]. The ability for complicated decision making without reliance on large data sets has also been achieved [10], [11]. As such, the ability to accurately distinguish humans from bots becomes an increasingly challenging task.

We have formulated the following research question: can we efficiently and accurately detect AI bots early in a strategy game?

Due to the aforementioned advances in bot AI, we have decided to apply machine learning for this nuanced classification task, which enables the option to be eventually automated. Taking what is known about the current state of bot detection in real time strategy games, we move forward to present an computationally efficient method for bot detection in *StarCraft: Brood War* using deep learning. We leverage information and techniques provided by earlier studies in order to develop a deep learning algorithm capable of accurately detecting machine controlled players.

Our model's contributions are the following:

- It uses a limited set of features in order to detect AI bot allowing it to potentially generalize easier to other games.
- It is computationally efficient requiring but a limited amount of gameplay in order to make accurate predictions.
- It can predict with high accuracy AI bots.

A source code is provided along with the paper and can be found at: https://gitlab.cs.wwu.edu/201940-starcraft/tsikerdekis/starcraft. The rest of the paper is organized as follows. In section II, we provide an overview of representative works in the field of AI bots and bot detection. Section III illustrates the steps for our proposed method for bot detection. Section IV introduces the experimental design that we have used to evaluate our method. We present our results in section V where we further contrast our method's performance and computational overhead with similar methods. Finally, in section VI we highlight some of the limitations as well as future challenges and opportunities in bot detection.

II. RELATED WORKS

Several examples of studies that use machine learning algorithms for detecting bots in MMORPGs exist with promising results in their accuracy scores [1], [3], [4], [12], [13]. For example, a study experimented with different classification architectures in order to ultimately discover a random forest technique that demonstrated the highest degree of accuracy [3]. The results from different works depend on an emphasis of precision over recall. For example, a study has demonstrated high precision scores in detecting bots in StarCraft [3]. While high precision is important for avoiding false identification of humans as bots, this can often come at the cost of being able to detect a majority of the bot population to improve the gaming experience (i.e., high recall). Low recall scores obtained in some studies [3], [12] present an opportunity for improvement of these bot detection methods.

However, high recall is not unattainable. Improved recall scores in Massive Multiplayer Online Role Playing Games (MMORPG) player classification have been demonstrated in a study using similar algorithms [1]. The approach resulted in such high precision and recall on a large test set demonstrating an ability to generalize well to unseen data. Additionally, classification was performed very efficiently in these high-accuracy models with predictions on new data often occurring in less than a second. Complex features such as player networks are often seen as being effective in producing high accuracy from these classifiers [4]. The drawback of such features is that they lead to a much greater amount of computation time during training [1]. As an alternative, achieving high accuracy scores while neglecting the use of such complex features is possible if one includes features beyond what can be taken from game replay data, effectively reducing the false positive rate [1]. Another study has demonstrated exceptional accuracy scores with a full detection framework comprised of several Bi-directional Long Short Term Memory (Bi-LSTMs) neural networks [13]. However, in addition to extensive data preprocessing, the computational overhead involved in training multiple supervised and unsupervised models is significant. A substantial amount of resources is required to implement and maintain such a system, which makes such systems less scalable for online gaming communities.

Current work demonstrates that training a classifier on a sequence of game state data has the capability of producing highly accurate detection systems. For example, studies have used player trajectory taken from game traces to detect bots [2], [14]. In this case, a model's learning is dependent upon the temporal relation between states. While this approach has proven to produce high precision and recall scores, traces are required to be up to ten minutes long for a model to produce accurate predictions. This is similar to other methods, where StarCraft replays are up to sixteen minutes long [12]. The length of a data sequence has a drastic impact on the computational expense of training a model. Therefore, an efficient approach which utilizes the richness of temporally related game data appears to be missing from the community.

While many of the methods described in related literature produce high classification results, we identified several key areas where improvements could be made.

First, there is a disparity in the aforementioned methods and their dataset sizes that were utilized in training and testing their predictive models. On the lower end, [14] and [2] used 519 player trajectories in the *Quake 2* game setting to train their models. On the upper end, [4] and [13] used billions of samples, which was achievable by having access to a wealth of data provided by the developers of the games themselves. More relevant to our work is [12], which aimed to detect bots in *StarCraft: Brood War* specifically. In terms of dataset size, the study used 1,139 Protoss vs Terran game logs to train models. We believe that a larger and more general dataset could be beneficial to model performance when using modern machine learning techniques.

Second, there is opportunity for improvement in terms of the frequency of predictions. The methods illustrated in some studies [1], [4], [12], [13] make new predictions only when certain player actions are taken (e.g., a player attacks another player). In the case of *StarCraft: Brood War* bot detection, the method proposed in [12] makes new predictions when players hit certain "milestones" in gameplay, such as building barracks, or expanding a base. Depending on the manner in which a player plays, these predictions can be sparse. Predictions that are not only accurate, but also occur more often can help identify bots earlier in the game, leading to efficiency benefits.

Third, the number of features used to detect bots could be improved. Most of the existing bot detection methods use a large number of features. For example, when testing the number of features, a study tested predictive models using 114, 62, and 6 features and found that as the number of features used decreases, the model performance suffered, but still retained impressive results even using only 6 features [3]. Another study was able to use 56 features to detect bots with good results inside of *StarCraft: Brood War* [12]. Although using a high number of features allows machine learning models to use feature extraction to isolate and interpret important information, fewer features provides efficiency benefits to both training and making predictions with a machine learning model.

III. PROPOSED METHOD

Our method predicts bot players inside of *StarCraft: Brood War* with a high degree of classification performance using machine learning algorithms. We also improve on other techniques used in literature by using a larger training dataset, utilizing fewer features, and having a higher prediction frequency.

A. DATA PARSING

We gathered *StarCraft: Brood War* replay files from large databases in order to get a sufficiently large training dataset. These files are separated by games played by human players and by bot players. We used over 6,000 professional human replays and 500 non-professional human replays [15], and 7,865 bot replays from a 2014 AI competition [16].

We then converted these replay files into JavaScript Object Notation (JSON) files. To do so, we used *screp*, an open source replay parsing tool [17]. This script was run with the argument -cmd in order to record player commands. JSON files remained separated by human or bot players.

We used *R* to convert, format, clean, and save the data from the JSON files as Comma Separated Value (CSV) files. We kept the CSV files separated by human and bot players. A unique CSV file was created for each player in the replay. The CSV files contained a record of the frame in which the player took an action, the number of units they have selected at that time, and their x and y mouse coordinates at that time.

Because there is little documentation for *screp* [17], we made the following assumptions on the extracted data. When a player has negative mouse coordinates, we translated this to mean that they are building units or constructing buildings. There also were occasionally instances where the same frame number appears in multiple consecutive rows of the CSV. This occured when multiple actions (usually building and unit production) are recorded at the same time.

In order to conserve memory, all CSV files were truncated so that none have frames above 15,000 (about 10 minutes of gameplay). The 10 minute decision was decided as a benchmark heuristically as well as a time threshold that has been used in the past for predicting the winner of a game [18]. Further, some players in a replay did not had any actions happen in the first 30 seconds of the game likely due to being a spectator or just a player that is away from their computers when the match starts. As such, any CSV files with initial frames higher than 360 (no action in about 15 seconds since the start of the match) were considered outliers and were deleted from the dataset. The final dataset consisted of 17,205 human players and 15,730 bot players that were saved locally in folders labeled "human" and "bot."

This training data provided us with unique information for our deep learning model. *StarCraft: Brood War* replays files contain enough input data to "reconstruct" a match. This includes the construction of buildings and units, without specifically describing information about them, while all using simple input data. This input data is recorded whenever a player performs any action, which means that any predictions about whether a player is a bot or human can be performed at a much higher frequency compared to other methods in the literature.

B. DATA PREPROCESSING

After data was parsed, each CSV file was loaded inside of a two dimensional (2D) Numpy array (a module used in scientific computing for Python).

We introduced an experimental variable, S, which was defined as the amount of time (in seconds) of player game inputs from the start of a match. We devised S as a means to determine what is the amount input data a model needs in order to make a determination on whether a player that the model is observing is a human or a bot. Smaller S values will inevitably lead to less computational overhead on the model and as such it was deemed critical to test variable values in order to demonstrate that the model is not just effective. Each 2D array had the number of rows limited based on this time constraint. Specifically, the last row of the split array was such that the frame number of the next row exceeds $24 * S$, where 24 is the number of game input frames that can occur per second. If an array did not contain S seconds of game input, no row limiting occurred.

After the number of rows were adjusted, numeric variables (e.g., time) are scaled to help speed up learning.

The first column of each row, the frame number (FN), was scaled to be the proportion of the game input length, S, that the frame occurs on. Mathematically, this scalar is $\frac{1}{24*S}$, since there are a possible 24 logical input frames per second.

The second column of each row, the number of units selected (SU), was scaled by $\frac{1}{24}$, since *StarCraft: BroodWar* only lets the player select a total of 24 units as an upper bound.

The third and fourth column of each row, the X and Y mouse positions on the game map (Mx and My respectively), are scaled by $\frac{1}{8192}$. Since the maximum boundaries of the game map vary depending on which map is being played, we used this general scalar to ensure that no matter how large the maps are, the values are mostly in the range of 0 and 1.

Figure 1 shows the process through which a replay was parsed and reprocessed.

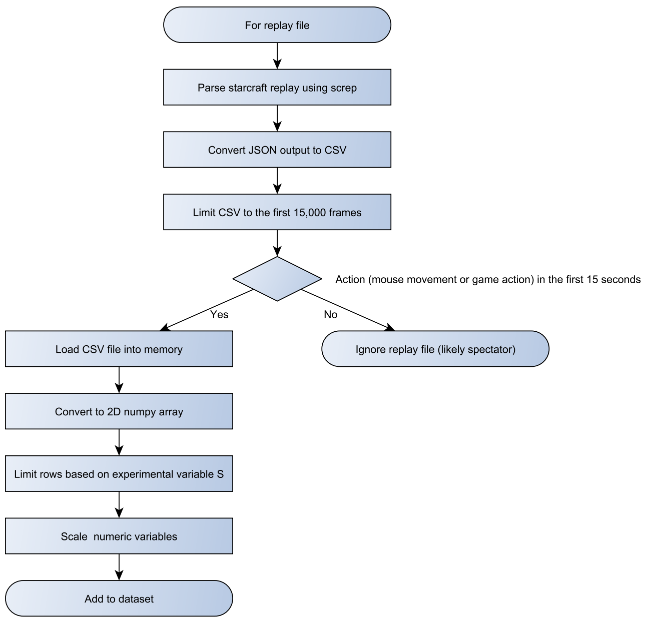
For replay file
Parse starcraft replay using screp
Convert JSON output to CSV
Limit CSV to the first 15,000 frames
Action (mouse movement or game action) in the first 15 seconds
Yes
No
Load CSV file into memory
Ignore replay file (likely spectator)
Convert to 2D numpy array
Limit rows based on experimental variable S
Scale numeric variables
Add to dataset

FIGURE 1. The replay preprocessing flow to produce a dataset that can be trained using deep learning.

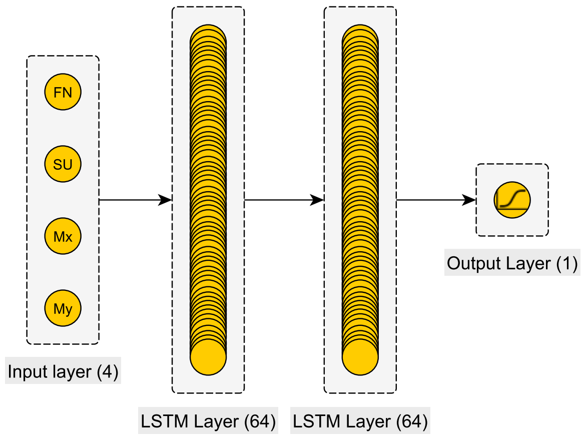

FIGURE 2. The deep learning binary classifier model that was used in this study. Output is a sigmoid function. Number in parentheses shows the number of neurons per layer. *FN* is frame number, *SU* is the number of selected units, *Mx* is the *X* mouse position, and *My* is the *Y* mouse position.

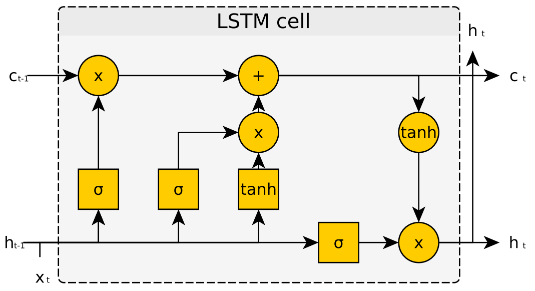

FIGURE 3. The Long Short-Term Memory (LSTM) cell [20]. σ is a sigmoid function. Squares represent network layers. Circles are pointwise operations.

C. DEEP LEARNING MODEL

We constructed a deep learning binary classifier model to detect bot players using sequences of game input information. Our model aimed to learn the mapping $Bot : G \mapsto \{0, 1\}$, where G is the set of all possible gameplay input sequences $g \in G$. In turn, the sequence of game frames, as a stream of player game input information extracted from the game is defined as $g = \langle f_1, f_2, \ldots, f_L \rangle$, where $f_t \in \mathbf{R}^4$ is a frame of game input from a player in real time and L is the index of the last frame which occurs within a specified time limit. This index is likely unique for every game, since frames occur whenever the player performs an action, rather than at regular timed intervals.

We chose to implement our deep neural network using a standard 4 layer architecture. Figure 2 summarizes the deep learning model. The input layer accepts vectors of size 4 which represent a game input frame, f_t. This input layer is then followed by 2 hidden Long Short-Term Memory (LSTM) layers, both with 64 units each.

We chose to use LSTM layers for our model primarily because they can process variable length sequences of input vectors. While standard Recurrent Neural Network (RNN) layers also have this ability, LSTM layers can more accurately interpret longer sequences due to the fact that they have input, output, and forget gates that manage the flow of information through the layer [19]. Also, LSTM layers avoid the "dying gradient" problem during training due to using vector addition, which prevents partial derivatives from getting close to 0 after successive applications of the calculus chain rule. Figure 3 depicts a single LSTM cell. The cell consists of several functions which operate from left to right. There is a "forget gate layer" that decides what information will be thrown away from the cell, an "input sigmoid layer" that decides which values to update, a tahn layer that create a vector candidate values and combines with the previous layer, and an "output layer" where the model decides what output to export using a sigmoid and a tahn function.

We chose to use 2 hidden LSTM layers because it allows the neural network to learn temporal hierarchical relationships in the training data. Using 64 units for the hidden layer size keeps the parameter count relatively small. Especially, since parameter counts grow $O(n^2)$ where n is the number of units in a hidden layer.

The final layer was a dense layer of one unit that uses the sigmoid activation function, which outputs values in the range of 0 and 1. Values close to 1 mean the model is predicting the player to be a bot, while values close to 0 mean a prediction that the player is human. During training, these output values were used directly, but were rounded to be either 0 or 1 for testing and validation.

Overall, our machine learning model had a total of 50, 753 parameters that had to be trained before predictions can be made. This parameter count is fairly small for a deep learning application, and when written to disk, the parameters occupied 220.1 kilobytes of memory. A small parameter count also provides speed benefits for both training and prediction. The matrix operations are applied to relatively small matrices, making both predictions and backpropagation (for finding gradients) much faster.

D. DETECTOR USAGE INTENT

Our detection method was designed to be well suited for a concurrent application environment. Like many other popular online games, *StarCraft: BroodWar* will have thousands of players participating in matches during peak hours. At any given moment, players may be currently inside of a match, waiting for an opponent during match-making, or leaving a completed game. This problem setting shares characteristics with concurrent tasks, where resource utilization is important to optimize.

A set of detectors can be considered to be a finite resource which can be "acquired" or "freed" very similarly to a semaphore. When a player enters a match, an open detector can be acquired by the player (although the actual detector exists on the server, not the client program). As the match progresses, the player input frames are sent to the detector for real time prediction and the LSTM layers of the model maintains a hidden state unique to the specific player.

Although a detector could be active for the entire match, it would be much more computationally efficient to make an accurate judgement as to whether or not the player is a bot in the least amount of time possible. In this scenario, a player that is suspected to be a bot early in the game can be flagged for further review and the detector can then be "freed" for the next player to acquire.

Using real time predictions to make early detection of bots provides several advantages. First, since fewer player input frames are required to make a decision on whether a player is a bot, fewer floating point operations would be required, which puts computing resources under less strain.

Second, a single detector would be able to observe more players in a given time, since it would only spend a fraction of a match's length making predictions on a player before being freed and moving onto the next player. Another benefit is that fewer detectors will be required to monitor an entire population of players, leading to even more efficient usage of computing resources.

IV. EXPERIMENTAL DESIGN

A. GAME INPUT SEQUENCE LENGTH EXPERIMENT

In order to demonstrate the accuracy, efficiency, and robustness of our detector, we performed multiple k-folds cross validation experiments across different game input sequence lengths to measure the effect that the amount of game input data has on model performance.

When loading the training data from disk, we omitted player input sequences that did not contain enough information. These data points often did not contain any input vectors, and as such could not be used for training. Once these outliers were removed, undersampling is performed to balance the dataset so that there are equal number of bot and human games. During all sequence lengths, 17,205 human and 15,730 bot games were used for training and testing the model.

For training, we chose to use the Adam optimizer [21] with a learning rate of 10^{-4}. To ensure training stability, we also applied a clip to each component of the gradient of 0.1. We chose to use a minibatch size of 64 training samples, which also helps with training stability in addition to faster training through computing gradients in parallel.

We used Binary Cross-Entropy (BCE) as our loss function. This loss function is well suited to this task, since the function output approaches infinity as the model output approaches the opposite of the training label. This unique property allows the model to train faster than a conventional loss function, such as mean squared error.

Keeping our training hyperparameters constant, we varied the game input length variable, S, to be 15, 30, 45, and 60 seconds. For each of these game input lengths, we performed k-folds cross validation with $k = 10$. Once a fold was complete, we evaluated our model's performance on F_1, F_2 scores, precision, and recall on the test set. These four metrics also provided us with information about the frequency of false positives, which are very important if a prediction of a bot could result in a player being banned.

B. REAL TIME PREDICTION

Once a model was trained, we evaluated its performance in a simulated real-time environment.

To accomplish this, an LSTM neural network model was created nearly identically as the one used during training. The difference being this new model was "stateful", meaning the internal state of the model was preserved rather than reset upon prediction. This was the real time detector. Since the model architecture is identical to the trained model, the learned parameters can be directly applied to the real time detector.

The goal of this experiment was to determine how predictions change over the course of the game. In order to do this, we ran each of the valid game input sequences into the real time model one frame at a time. These predictions were placed inside of an array of size $24 * S$. The index at which these predictions were placed is the unscaled logical game frame number. If multiple frames occurred on the same game frame number, the frame with the highest index in the sequence was used to make a prediction.

Since there are some logical game frames where the player takes no action, the prediction array had "gaps" in it. In these cases, the gaps were filled using a post processing algorithm that uses linear interpolation.

V. RESULTS

In this section, we describe the results of the experiments that we conducted.

A. SEQUENCE LENGTH EXPERIMENT

We highlight the effect of sequence length (S) in accuracy in figure 4 as well as table 1.

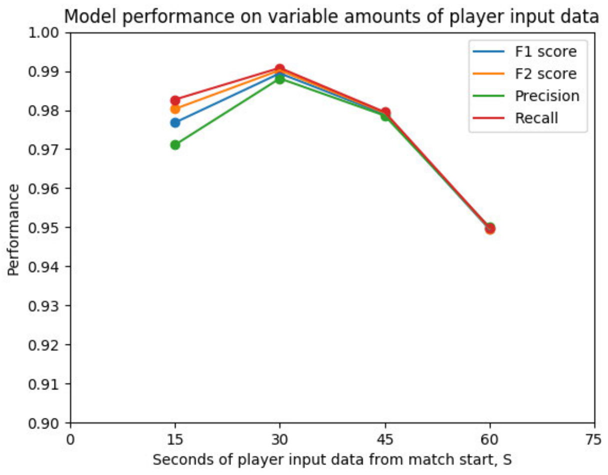

FIGURE 4. Sequence length experiment results in plotted form.

TABLE 1. Sequence length experiment results in tabular form (all entries rounded to 4 decimal places, all metrics calculated from test sets on each fold). Note: S: Number of seconds since the match started.

S	F1 score	F2 score	Precision	Recall	Loss
15	0.9768	0.9803	0.9711	0.9827	0.0970
30	0.9894	0.9902	0.9881	0.9908	0.0616
45	0.9790	0.9793	0.9786	0.9795	0.0971
60	0.9495	0.9497	0.9500	0.9499	0.1880

Overall, our model performed well irrespective of S. There is a general improvement in model performance as the amount of data provided to the model increased (i.e., the time the detector can observe a player increases). This expectation holds true for the 15 and 30 seconds tests, albeit the improvement is small.

However, performance dropped beyond that point. There are many reasons as to why this may be the case, but we speculate that the LSTM hidden layers of the model were more difficult to train on longer sequence lengths, which is a well known issue.

It appears that the optimal sequence length for detecting bots is around 30 seconds. Ideally, this model configuration would be chosen to be used in a real application. However, all 4 trained model configurations performed quite well, so it may be more beneficial to use the 15 second configuration, since it would perform slightly worse than the 30 second configuration, but make accurate predictions in half the amount of time.

B. REAL TIME PREDICTION RESULTS

Results of the real time predictions are shown on figure 5.

For general performance, all the real time detector models predicted classes were close to either 0 or 1 by the end of the gameplay sequence.

There are many things these results have in common. First, all of the models have the human and bot predictions very close to each other for the first quarter of the gameplay sequence. We believe this is due to both humans and bots having extremely similar behavior at the start of a match (selecting the initial workers and sending them to collect resources).

Second, all models are more confident in bot predictions than human predictions. This is clearly demonstrated on the 15 second model, where the detector has high confidence in bot predictions from nearly the start of the match, while human predictions are much farther from their supposed label of 0.

Third, the human prediction curve is much more stable across all models compared to the bot curve. It follows a strange trend where human predictions will start off as bot predictions before moving to 0 in a very smooth curve. Bot predictions have different types of curves, especially the 15 second model compared to the 60 second model.

Fourth, both bot and human predictions curves do not diverge until halfway through the gameplay sequence. This occurs with all 4 models, which is a unique observation since it is intuitive to believe that the prediction curves would diverge at roughly the same time from the start of the match.

Fifth, the variance in predictions across all prediction curves is somewhat high. This may be caused by the difficulty in classifying bots from humans. Some prediction curves, such as humans in the 15 second model, have a very high level of variance.

We believe that many of the shortcomings from this experiment (high variance, poor prediction curves) can be attributed to the difficulty in training LSTM layers in the model. Even though LSTM models are more resistant to vanishing gradient problems, their learning capacity is not infinite in terms of the sequences they can remember. It is possible that the abnormal behavior of bots is more likely to confuse even LSTM models and possibly that is why human prediction seems to be more stable in the later parts os a match's observation. Since all models performed more or less the same in a simulated real time setting, the 15 second trained model seems to be the best choice to use in an application, since reasonably good predictions can be made within 10 seconds of a match start, albeit with high variance in human predictions.

C. COMPARISON TO OTHER TECHNIQUES

We provide a comparison summary on table 2 for our method with studies that have developed bot detectors.

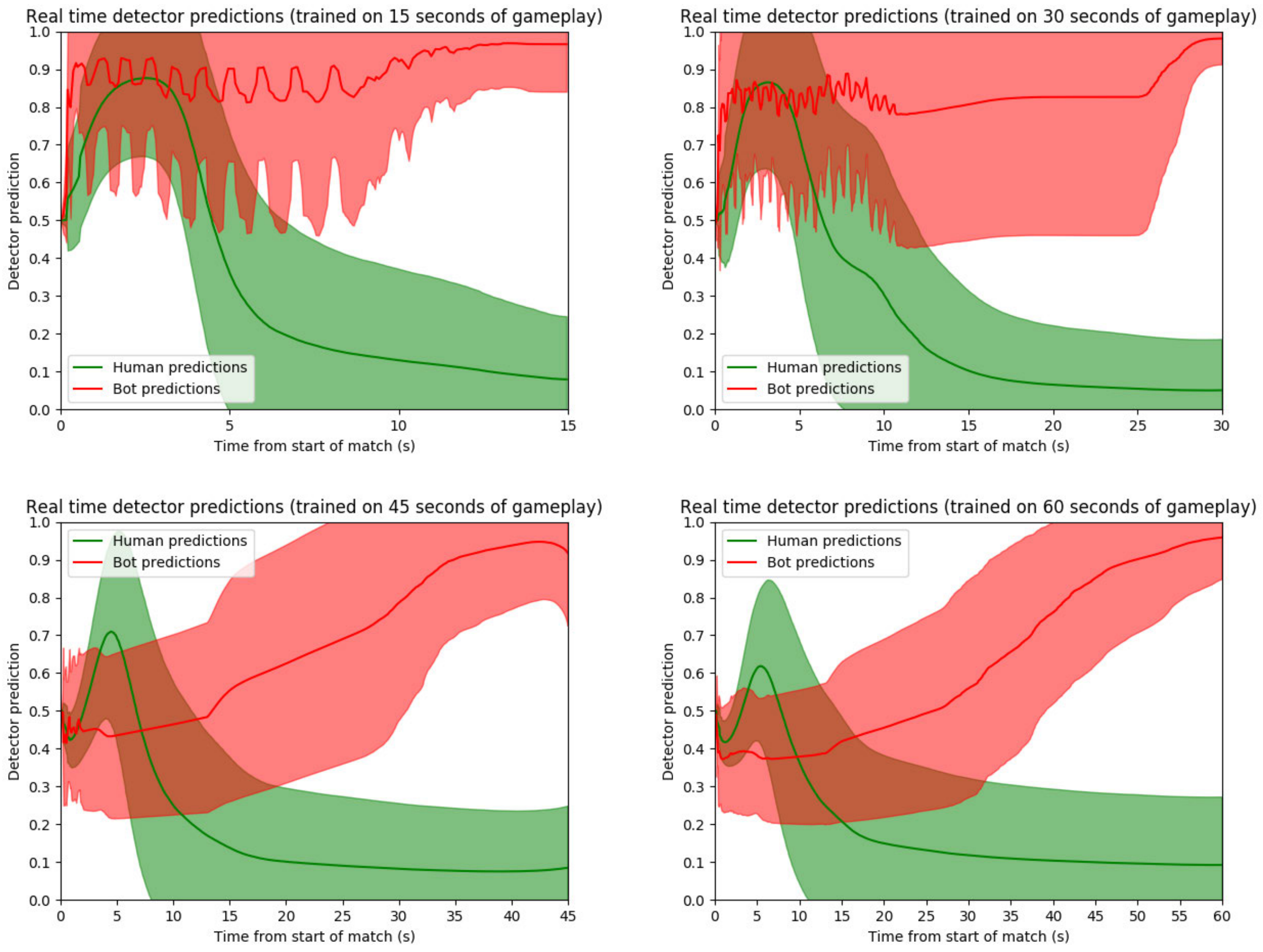

FIGURE 5. Real time detector predictions based on sequence length (*S*) of 15, 30, 45 and 60 seconds. Note: Shaded region denotes 1 standard deviation.

TABLE 2. Comparison of our detector with other similar bot detectors.

Technique	Game	Prediction time (S)	F1 Score	F2 Score	Precision	Recall
LSTM	*StarCraft: Brood War*	15	0.9727	0.9791	0.9625	0.9834
LSTM	*StarCraft: Brood War*	30	0.9843	0.9860	0.9817	0.9870
Non-linear SVM [15]	*Quake 2*	100	-	-	-	≈ 0.98
NNge [13]	*StarCraft: Brood War*	300	-	-	-	≈ 0.80

Many of the bot detection techniques described in the literature did not all provide the same classification performance metrics that we used. Recall, however, was provided, and we have included that metric in the table for these studies. Additionally, the other methods in the literature also use different video games as the setting for testing their technique, so the comparison is relative. On the one other source [12] that does use *StarCraft: Brood War* as the game setting, the train and test data set were not the same as in our technique.

Our technique meets or exceeds the recall score that the other techniques produce. In particular, compared to the other *StarCraft: Brood War* detector technique, our technique produces a significantly higher recall score and more importantly our precision scores is also just as high This is an important factor for administrators that will base their decisions in part based on the detector's classification.

The novelty of our technique is also shown in regards to prediction time. We were able to make accurate predictions in the fraction of the time at no cost to accuracy. The ramifications of this, is that a real-time implementation of our approach can determine whether a player is a bot early in the game and then proceed by examining another game. Further, players involved in a game will not have to expend effort for a long period of gameplay in order to be warned that they may be playing against an AI.

VI. LIMITATIONS

We introduce a few considerations that may apply to our method. First, the dataset that we have used for training our model is not a good representation of a typical *StarCraft: BroodWar* match. Specifically, the human labeled games are biased towards professional players. This is due to a lack of availability of amateur level replays on the internet, as there is a higher demand from the community for professional player replays for learning and entertainment purposes. Additionally, our dataset had a maximum of roughly 30,000 samples in it. Although our detector did generalize well with this amount of data, better results may be achieved with an even larger dataset, such as, as an extreme, the entirety of all *StarCraft: BroodWar* multiplayer matches. There are also considerations for the bot portion of the dataset. The bot data samples were collected from bot competitions that had participants developing bots to defeat primarily other bots. As such, bot behavior may not necessarily be close to a human player.

VII. CONCLUSION

We plan to expand our method in several ways in the future. This initially involves addressing some of the considerations discussed in section VI. Issues with how the human portion of the dataset being biased towards professional players could be addressed by crowdsourcing replay files from the *StarCraft: BroodWar* community. Replay files could then be acquired from players on a wide spectrum of skill levels in order to train a more generalized detector.

Acquiring bot replays of bots that attempt to behave similar to humans is more challenging, as it would require bot developers to redesign their bots to both play the game well and behave like a human. Instead, it might be worthwhile to use our detector as a component of a reward function for a reinforcement learning agent. Specifically, the detector's output multiplied by -1 could be used directly as a reward. If the bot is behaving like a human, the reward would be close to 0, and if the bot is easily detectable, the reward would be close to -1. Since our detector can make predictions every input frame, the rewards would not be sparse, which could help in training a reinforcement learning agent faster.

This idea could further be extended by having the detector train against the reinforcement learning agent by adding the agent's input frame sequences into the detectors training dataset. This would create an adversarial training environment where the detector would try and outsmart the agent, while the agent tries to outsmart the detector.

Advanced bots in video games has become a reality in the recent years with the proliferation of tools for reinforcement and deep learning. This article aimed to advance detection strategies that can be easily implemented and detect bots at a short amount of time compared to past methods. As such, we have demonstrated the feasibility of attaining high detection accuracy, with a small feature set and limited observations. Given the information asymmetry that typically exists between an attacker (bot developer) and a defender (bot detector developer), we believe that simpler models may be better at adapting in games where incomplete information may exist (e.g., where observing player actions in games).

APPENDIX

MODEL HYPERPARAMETERS

Hyperparameter	Value
Input vector size	4
Hidden layer type	LSTM
Number of hidden layers	2
Number of units per hidden layer	64
Output vector size	1
Output layer activation function	Sigmoid
Number of parameters	50,753

TRAINING HYPERPARAMETERS

Hyperparameter	Value
Optimizer	Adam
Learning rate	10^{-4}
Minibatch Size	64
Number of folds, k	10
Epochs per fold	8
Gradient clip (value)	0.1

SEQUENCE LENGTH EXPERIMENT RESULTS (TABLE FORM)

Seconds of player input data from match start, S	F1 score	F2 score	Precision	Recall	Loss
15	0.9727	0.9791	0.9625	0.9834	0.0860
30	0.9843	0.9860	0.9817	0.9870	0.0752
45	0.9761	0.9792	0.9709	0.9814	0.1030
60	0.9724	0.9707	0.9753	0.9696	0.1232

All metrics were calculated on each fold's test set during k-folds cross validation training.

REFERENCES

[1] M. L. Bernardi, M. Cimitile, F. Martinelli, and F. Mercaldo, "A time series classification approach to game bot detection," in *Proc. 7th Int. Conf. Web Intell., Mining Semantics*, Jun. 2017, pp. 1–11.

[2] H.-K. Pao, K.-T. Chen, and H.-C. Chang, "Game bot detection via avatar trajectory analysis," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 3, pp. 162–175, Sep. 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5560779/

[3] A. R. Kang, S. H. Jeong, A. Mohaisen, and H. K. Kim, "Multimodal game bot detection using user behavioral characteristics," *SpringerPlus*, vol. 5, no. 1, p. 523, Dec. 2016.

[4] E. Lee, J. Woo, H. Kim, A. Mohaisen, and H. K. Kim, "You are a game bot!: Uncovering game bots in MMORPGs via self-similarity in the wild," in *Proc. NDSS*, Internet Society, May 2017, pp. 1–15.

[5] M. Certicky, M. Sarnovsky, and T. Varga, "Use of machine learning techniques in real-time strategy games," in *Proc. World Symp. Digit. Intell. Syst. Mach. (DISA)*, Aug. 2018, pp. 159–164. [Online]. Available: https://ieeexplore.ieee.org/document/8490528/

[6] M. Certicky, D. Churchill, K.-J. Kim, M. Certicky, and R. Kelly, "StarCraft AI competitions, bots, and tournament manager software," *IEEE Trans. Games*, vol. 11, no. 3, pp. 227–237, Sep. 2019.

[7] S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in starcraft," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6637024/

[8] G. Synnaeve and P. Bessiere, "Multiscale Bayesian modeling for RTS games: An application to StarCraft AI," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 4, pp. 338–350, Dec. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7293159/

[9] G. Synnaeve and P. Bessì. (2012). *A Dataset for StarCraft AI and an Example of Armies Clustering*. [Online]. Available: http://code.google.com/p/bwta/

[10] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 3, no. 1, pp. 73–84, Feb. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8351991/

[11] Q. Gemine, F. Safadi, R. Fonteneau, and D. Ernst, "Imitative learning for real-time strategy games," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Sep. 2012, pp. 424–429. [Online]. Available: http://ieeexplore.ieee.org/document/6374186/

[12] K.-J. Kim and S.-B. Cho. (2011). *Server-Side Early Detection Method for Detecting Abnormal Players of StarCraft*. [Online]. Available: http://lmrb.net

[13] J. Tao, J. Xu, L. Gong, Y. Li, C. Fan, and Z. Zhao, "NGUARD: A game bot detection framework for NetEase MMORPGs," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 811–820.

[14] K.-T. Chen, H.-K.-K. Pao, and H.-C. Chang, "Game bot identification based on manifold learning," in *Proc. 7th ACM SIGCOMM Workshop Netw. Syst. Support Games NetGames*. New York, NY, USA: Association for Computing Machinery, 2008, pp. 21–26. [Online]. Available: https://doi.org/10.1145/1517494.1517498

[15] Starcraftai.com. (2016). *StarCraft Brood War Data Mining*. [Online]. Available: http://www.starcraftai.com/wiki/StarCraft_Brood_War_Data_Mining

[16] D. Churchill. (2015). *StarCraft AI Competition*. [Online]. Available: http://www.cs.mun.ca/~dchurchill/starcraftaicomp/media.shtml

[17] A. Belicza, B. Kutil, and P. Coles. (2020). *Screp*. [Online]. Available: https://github.com/icza/screp

[18] A. Álvarez-Caballero, J. J. Merelo, P. García Sánchez, and A. Fernández-Ares, "Early prediction of the winner in StarCraft matches," in *Proc. 9th Int. Joint Conf. Comput. Intell.*, Science and Technology Publications, 2017, pp. 401–406. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006587304010406

[19] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proc. 13th Annu. Conf. Int. Speech Commun. Assoc.* Portland, OR, USA: ISCA, 2012, pp. 194–197. [Online]. Available: https://www.isca-speech.org/archive/archive_papers/interspeech_2012/i12_0194.pdf

[20] G. Chevalier. (2018). *The LSTM Cell*. [Online]. Available: https://commons.wikimedia.org/wiki/File:The_LSTM_cell.pn

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

MICHAIL TSIKERDEKIS (Senior Member, IEEE) received the Ph.D. degree in informatics from Masaryk University. He is currently an Assistant Professor with the Department of Computer Science, Western Washington University. His research interests include deception, data mining, cybersecurity, and social computing.

SEAN BARRET is currently pursuing the B.Sc. degree in computer science with Western Washington University. His research interests include data parsing, research of current bot detection strategies, and testing.

RALEIGH HANSEN received the B.Sc. degree in computer science, with a minor in mathematics. She is currently pursuing the degree in computer science with Western Washington University. Her research interests include anomaly detection, transfer learning, and natural language processing.

MATTHEW KLEIN received the B.Sc. degree in computer science, with a minor in mathematics. He graduated from Western Washington University. He is currently a Quality Control and Analytics Specialist at Wompmobile, Bellingham, WA, USA. His research interests include data science, back-end development, and human–computer interaction.

JOSH ORRITT is currently pursuing the B.Sc. degree in computer science with Western Washington University. His research interests include social and consumer focused applications of computing, deep learning, and image and vision computing.

JASON WHITMORE received the B.Sc. degree in computer science, with a minor in mathematics. He graduated from Western Washington University. His research interest includes reinforcement learning, specifically actor-critic methods.

• • •