Player Imitation for Build Actions in a Real-Time Strategy Game

Nathan Partlan, Abdelrahman Madkour, Chaima Jemmali, Josh Aaron Miller, Christoffer Holmgård, Magy Seif El-Nasr
{nkp,a3madkour,chaimajemmali,joshm}@ccs.neu.edu, {holmgard,m.seifel-nasr}@northeastern.edu
Northeastern University

Abstract

Player imitation, in which an AI agent attempts to mimic a specific player's actions, can expand the possibilities for automated playtesting and adaptive AI. Prior work on player imitation, however, has been limited to relatively simple domains. Real-time strategy (RTS) games require complex, strategic decision-making, but previous research on them has not focused on player imitation. In this study, we compare player imitation using recurrent neural networks, random forests, and a random baseline model. We test these methods using replays from the popular RTS game *StarCraft*. We compare the results across methods and discuss how differences in the data sets affect performance, finding that RNNs are the most successful of the evaluated methods. We discuss takeaways for future research, including ethical considerations.

Introduction

"Player imitation," which attempts to mimic a particular player's gameplay decisions, could unlock new opportunities in automated playtesting, behavior modeling for procedural content and AI generation, and human-like agents.

By imitating a particular player, we might incorporate the quirks and preferences of that player's gameplay into an AI agent. Using the imitation as a training partner, we might train agents to compete or cooperate with that player.

Automated playtesting is a growing area of research, in which AI agents act in the role of a player in order to ease and expand playtesting (Holmgård et al. 2018). For independent game developers, or for games with large possibility spaces, such as those using procedural content generation, automated playtesting can make comprehensive testing less prohibitive. Player imitation could enable automated playtesting that explores how a player with a particular ability level, a specific disability, or a certain preferred play style would interact with the game.

Thus, player imitation, if sufficiently robust, could become a flexible and broadly applicable technique. So far, however, research has not developed player imitation with sufficient power and generality to meet these needs.

Real-time strategy (RTS) games present a particularly complex and difficult setting for player imitation. In these games, such as *StarCraft*, agents must decide what buildings and units to build, and when, among other tasks. In this domain, player imitation requires strategic action, precise timing, and mimicking of the player's particular strategic and tactical preferences while reacting to opponents' decisions.

To begin exploring models that might meet these requirements, we examine three methods for imitating player behavior in *StarCraft*, specifically focusing on strategic level actions: deciding what buildings and units to build and when to build them. We treat this as a supervised learning problem, rather than as indirect imitation learning (Hussein et al. 2017), due to the lack of a fast forward simulation and the focus on precise imitation of behavior in known situations.

Our approach compares a random baseline algorithm, random forests (RFs), and recurrent neural networks (RNNs). We measure the results using playtraces from two *StarCraft* players of the "Zerg" faction. We report results of and comparisons between these techniques in several data sets: single faction matchups or combining all factions, on a single map and on multiple maps. We also report the effects of the size of the data set on accuracy.

We show that, of the techniques presented here and compared against prior results by Justesen and Risi (2017), RNNs are the most successful in achieving player imitation. We discuss the meaning of our results for future research in player imitation, their limitations, and the ethical implications of this research, including potential abuses of player imitation and recommendations to avoid enabling them.

Related Work

Imitation Learning

The research area of "imitation learning" seeks to bootstrap AI task performance by imitating experts, providing a starting point for further learning (Hussein et al. 2017). State-of-the-art indirect imitation learning techniques, such as GAIL (Ho and Ermon 2016), usually require environmental interaction and an evaluation function for performance in order to improve. This is often distinguished from direct imitation, such as "behaviour cloning," which learns directly from training data without further learning for generalization (Ghasemipour, Gu, and Zemel 2019). While appearing human-like is often a goal of imitation learning, it also usually prioritizes "effective task performance," such as win-

ning a game, rather than pure imitation.

In games, several researchers have applied various forms of imitation learning, as surveyed by Hussein et al. (2017). Most recently, DeepMind employed imitation learning to bootstrap the AlphaStar AI for StarCraft 2 (Arulkumaran, Cully, and Togelius 2019), though, as in some other imitation learning, they combined data from multiple experts, rather than seeking to imitate a single, specific individual.

In this work, however, we do not assume access to a fast forward simulation. We also do not require an evaluation function for novel states, unlike indirect imitation learning. Instead, we seek to precisely predict the player's behavior in situations similar to those in the training data. This leads us to formulate the problem of player imitation as a supervised learning problem, more akin to behavior cloning than to indirect imitation learning, as further described below.

Player Imitation and Human-Like AI in Games

Researchers have long been interested in building human-like AI behavior for games, sometimes learning from human players, e.g. (Laird and Duchi 2000; Geisler 2002; Priesterjahn et al. 2005; Schrum, Karpov, and Miikkulainen 2011; Llargues Asensio et al. 2014; Holmgård et al. 2018). These efforts, however, were aiming for general human-like AI or imitating broad play-styles. It is unclear whether they would learn the specific strategies of a particular player, which would require generalizing from a smaller data set.

In a few cases, researchers have targeted imitation of a particular player. This work originally focused on racing games (Togelius, De Nardi, and Lucas 2006). The "Drivatar" system for *Forza Motorsport* imitated specific players in a commercial game (Muñoz, Gutierrez, and Sanchis 2010). From this success, van Hoorn et al. (2009) began researching "player imitation," followed by others (Cardamone, Loiacono, and Lanzi 2009; Muñoz, Gutierrez, and Sanchis 2010). Beyond racing, *Killer Instinct*'s Shadow AI imitated players in a commercial fighting game using N-grams (Neal and Hayles 2016). Racing and fighting games, however, feature simple game states and decision-making, making these prior approaches difficult to generalize. Our work is, to our knowledge, the first to explore the imitation of a specific player in more complex strategy games.

Learning from RTS Replays

In the RTS genre, researchers have explored techniques for learning from game replays, usually to model opponent behavior. For WARGUS (Ontañón et al. 2007) and *StarCraft*, researchers began with case-based reasoning to predict build and research actions (Hsieh and Sun 2008; Weber and Mateas 2009; Farouk, Moawad, and Aref 2017).

For predicting the order in which opponents will build units, researchers have studied Bayesian (Dereszynski et al. 2011; Synnaeve and Bessiere 2011) and constraint-based methods (Stanescu and Čertickỳ 2016). Random forests have shown promise for identifying strategies and players (Cho, Kim, and Cho 2013; Liu, Ballinger, and Louis 2013). In this work, we test random forests for imitating specific players. We do not use Bayesian or constraint-based models, as it is less clear how to apply them to player imitation.

For performing actions based on imitation of human replays, Bryant and Miikkulainen applied neuroevolution to learn a policy for a simple strategy game (2007), and Robertson and Watson (2015) have explored automatically building behavior trees. However, this resulted in an unwieldy number of branches. Justesen and Risi (2017) employed a 4-layer feedforward neural network to predict player build actions for a single faction matchup in StarCraft, resulting in a performance improvement. We report a comparison between our results and theirs, though the comparison is not exact because they were predicting generic player behavior, rather than imitating a specific player.

Methods

Our method consists of the following primary algorithmic components: a basic game state processing pass, followed by a transformation into a human-designed state representation, then training using this representation, and finally prediction. We compare three different methods: a simple random baseline, random forests, and recurrent neural networks.[1]

Data Processing

We use replays from the public data set "StarData," by Lin et al. (2017), consisting of 65,646 StarCraft replays, with game states recorded every 3 frames. We consider only games with exactly two players. Because we are imitating a particular player, we do not use the entire data set. We use replays associated with that player's anonymous username.

We extract and store the data in our desired format using TorchCraft (Synnaeve et al. 2016). Each data point is a pair (S_i, a_i) where $S_i = (f_{i1}, f_{i2}, \ldots, f_{im})$ represents a state and $f_{ij}, j = 0, ..., m$ are the features describing that state. Each state has a corresponding action a_i, which represents the first unit the player began building during that frame (or is "empty," represented by -1, if they built no unit). Because the pre-extracted replays do not contain action information, we separately extract the action labels from the raw replay files (also distributed in the StarData data set).

In this way, we can treat the problem of imitating the player's unit construction actions as a multi-class classification problem on a sequence of game states, where the inputs are the features S and the classes correspond to the possible actions a, including the empty action. By including empty actions, we require that the imitation must predict not only the order of actions but also their precise timing, since we feed our algorithms one state at a time.

As a first-pass processing step, we extract many low-level features, such as the map ID[2], the players' factions, their resources (such as gas and ore), their units, and the units' properties. Using domain knowledge, we encode these into more informative features. For example, we aggregate health percent across all units. Other features include: counts of workers, resource units, and buildings (and how many are under attack); past build actions — the last unit built, when, and a "build intensity" which increases with each build and

[1]The code and details of our data processing and prediction methods are available at https://hdl.handle.net/2047/D20323119.

[2]Categorical features were converted into enums.

decays over time; and aggregate positional data — the mean and std. dev. of units' positions, their average velocity, and the distance between our army's mean position and that of the enemy army. Note that we consider all enemy units, not only the ones visible to the player.

Instead of separate features for more than 100 different unit types, we assume players will use a single faction's units and re-use 35 slots among all three factions.[3] We further compress the features by grouping unit properties (or flags) into domain-specific categories, counting instances within the group. These include: attacking, moving, staying, debuff, building, gathering, and other. We similarly sum units' properties, such as armor, shields, and attack power. We do not currently encode map terrain data, beyond the map ID. In total, we compute 154 features: 8 for game meta-data; 2 for battle information; 4 for personal build information; then for each player: 6 for resources, 35 for units, 15 for unit properties, 3 for buildings and workers, and 11 for army status.

Models

Our **random baseline** algorithm simply chooses at random between predicting any of the units that the player could build, given the game state, taking into account the player's current buildings and resources. Though we do not expect this baseline to be at all successful in imitating a specific player, it can tell us how much the other approaches improve on random selection among all reasonable actions.

Random forests construct several decision trees at training time, minimizing correlations, and, when used for classification, output the class that is the mode of their predicted classes (Ho 1995). For our random forests implementation, we use the sklearn RandomForestClassifier.[4] This library provides several parameters including class weighting, number of trees, max-depth of trees, and others. We use the following non-default parameter values: "balanced" class weight, 20 max features, 200 estimators.

Our **recurrent neural networks** are built on the open-source pytorch framework.[5] We use a single recurrent layer, implemented as a Long Short Term Memory (LSTM) network (Hochreiter and Schmidhuber 1997). We apply a random dropout of 40 percent, as determined by a grid search on non-empty F1 score (20 and 60 percent were also tested), to reduce overfitting. A final output Softmax layer is shaped to the number of possible actions. We performed a grid search to determine the number of LSTM layers and nodes, with the best result being 1 layer (up to 4 layers were tested) with 600 nodes (500 and 800 were also tested). We did not test significantly larger values, in order to ensure that our model could be run on a single machine by a game designer.

For training, we use Cross-Entropy Loss, as it is suited to multi-class classification (Shore and Johnson 1981). We use ADAM for optimization and learning rate control (Kingma and Ba 2015), halving the rate when training loss plateaus for 100 epochs, with pytorch's "ReduceLROnPlateau."

[3]As an exception, the Dark Archon can Mind Control other units. However, this case is rare and therefore ignored for this work.

[4]https://scikit-learn.org/stable/index.html

[5]https://pytorch.org

StarCraft replays' states are highly skewed towards the correct action being "build nothing" (empty actions), and many specific actions are very rare. To counter this, we weight the loss for failing to predict each action. The weight is the inverse of the average percentage of the action in the training data: $w = \sum_{i=0}^{R} N_i/n_i$ where R is the number of replays, N is the total length of the replay, and n is the count of that action in that replay. We also normalize states, dividing by the maximum value in the training set for each input.

Experiment

We evaluated each algorithm by predicting actions in replays from two specific players, who we designate P1 and P2. We predict each action in the replay, based on its corresponding state, so this is a supervised learning, not indirect imitation learning, style evaluation. Each faction in StarCraft has different units and strategies, so we chose games played as the "Zerg" faction. For P1, we limited the data to replays from a single, specific map to further constrain the problem. For P2, we used games played on a variety of maps. Three of the four data sets for each player consisted of matches against a particular opponent faction. These were, for P1 and (P2): 152 (305) replays against "Zerg" opponents, 206 (217) against "Terran" opponents, and 204 (240) against "Protoss" opponents. Finally, we combined the three factions to form a full data set of 562 (762) replays. For the RNNs, we also tested a small data set of 50 randomly selected games of all factions from P1, in order to test performance with less training data.

These replays ranged in length from about 1,300 to 10,000 frames, depending on game length. We randomly selected a validation ($\sim 5\%$) and test set ($\sim 20\%$) from each data set. For the RNNs, we trained for 4000 epochs, which was sufficient to reach a plateau in their performance. We chose not to perform cross-validation, as training each RNN required approximately a day on a single machine.

We computed precision, recall, and F1 score for each game in the test set as micro-averages, meaning that we added the numerator and denominator terms for each action, then divided those sums. Precision indicates, averaged for all classes, the proportion of actions that, when predicted, were present in the ground truth replay (not false positives). Recall indicates, averaged for all classes, the proportion of actions that, when they appeared in the ground truth replay, were correctly predicted (not false negatives). F1 score mediates between these. For RNNs, we report Top-1 and Top-3 error (the % of predictions in which the correct action is not in the 1 or 3 most probable). Since the baseline and RFs do not rank actions by likelihood, we cannot report Top-3 error for them. We also computed the statistics for non-empty actions only, omitting the "do nothing" action.

On the test set, we report these metrics separately for each game, as well as "Time Warp Edit Distance" (TWED) (Marteau 2009). TWED measures the similarity of two timed sequences of actions, with a configurable penalty for incorrect timing. We chose to set this penalty to 0.1, since it should be minimally noticeable to act within an error less than human reaction time (approximately 200-500 milliseconds). We report the average TWED per non-empty action.

	Factions	Error Total	F1 Total	Error NE	Precision NE	Recall NE	F1 NE
Baseline	Zerg P1	0.91 ± 0.015	0.086 ± 0.015	0.93 ± 0.042	0.0016 ± 0.0010	0.075 ± 0.042	0.0032 ± 0.002
	Zerg P2	0.92 ± 0.019	0.082 ± 0.019	0.94 ± 0.038	0.0012 ± 0.0009	0.062 ± 0.038	0.0023 ± 0.0018
	Terran P1	0.93 ± 0.017	0.071 ± 0.017	0.95 ± 0.015	0.0017 ± 0.0006	0.05 ± 0.015	0.0033 ± 0.0011
	Terran P2	0.93 ± 0.031	0.07 ± 0.031	0.94 ± 0.028	0.0017 ± 0.0009	0.064 ± 0.028	0.0033 ± 0.0017
	Protoss P1	0.93 ± 0.016	0.072 ± 0.016	0.94 ± 0.023	0.0024 ± 0.0008	0.056 ± 0.023	0.0045 ± 0.0014
	Protoss P2	0.92 ± 0.016	0.076 ± 0.016	0.94 ± 0.017	0.0017 ± 0.0008	0.062 ± 0.017	0.0034 ± 0.0015
Random Forests	Zerg P1	$\mathbf{0.018 \pm 0.0049}$	$\mathbf{0.98 \pm 0.0049}$	1.00 ± 0.0003	$\mathbf{0.67 \pm 0.63}$	0.017 ± 0.023	0.033 ± 0.044
	Zerg P2	$\mathbf{0.016 \pm 0.0039}$	$\mathbf{0.98 \pm 0.0039}$	1.00 ± 0.0006	$\mathbf{1.00 \pm 0.50}$	0.020 ± 0.033	$\mathbf{0.039 \pm 0.065}$
	Terran P1	$\mathbf{0.032 \pm 0.0085}$	$\mathbf{0.97 \pm 0.0085}$	1.00 ± 0.0004	$\mathbf{0.78 \pm 0.34}$	0.012 ± 0.013	0.024 ± 0.025
	Terran P2	$\mathbf{0.025 \pm 0.052}$	$\mathbf{0.98 \pm 0.052}$	1.00 ± 0.0004	$\mathbf{1.00 \pm 0.00}$	0.013 ± 0.017	0.026 ± 0.033
	Protoss P1	$\mathbf{0.04 \pm 0.013}$	$\mathbf{0.96 \pm 0.013}$	1.00 ± 0.0004	$\mathbf{0.78 \pm 0.43}$	0.0092 ± 0.012	0.018 ± 0.023
	Protoss P2	$\mathbf{0.027 \pm 0.0078}$	$\mathbf{0.97 \pm 0.0078}$	1.00 ± 0.0003	$\mathbf{1.00 \pm 0.20}$	0.011 ± 0.011	0.021 ± 0.021
RNN	Zerg P1	0.41 ± 0.059	0.59 ± 0.059	$\mathbf{0.63 \pm 0.15}$	0.017 ± 0.0084	$\mathbf{0.37 \pm 0.15}$	$\mathbf{0.032 \pm 0.016}$
	Zerg P2	0.32 ± 0.070	0.68 ± 0.070	$\mathbf{0.64 \pm 0.17}$	0.018 ± 0.077	$\mathbf{0.36 \pm 0.17}$	0.034 ± 0.014
	Terran P1	0.34 ± 0.12	0.66 ± 0.12	$\mathbf{0.78 \pm 0.06}$	0.021 ± 0.0062	$\mathbf{0.22 \pm 0.06}$	$\mathbf{0.037 \pm 0.011}$
	Terran P2	0.22 ± 0.66	0.78 ± 0.065	$\mathbf{0.79 \pm 0.094}$	0.025 ± 0.011	$\mathbf{0.21 \pm 0.094}$	$\mathbf{0.045 \pm 0.020}$
	Protoss P1	0.58 ± 0.17	0.42 ± 0.17	$\mathbf{0.69 \pm 0.10}$	0.022 ± 0.0062	$\mathbf{0.31 \pm 0.10}$	$\mathbf{0.041 \pm 0.011}$
	Protoss P2	0.53 ± 0.10	0.47 ± 0.10	$\mathbf{0.71 \pm 0.075}$	0.014 ± 0.0050	$\mathbf{0.29 \pm 0.076}$	$\mathbf{0.027 \pm 0.0094}$

Table 1: The medians and interquartile ranges for the metrics of each method on the test sets split by faction, for each player. "NE" indicates that the statistic is for non-empty actions (where a unit was built) only. Bolded results are the best for a metric for each test set. Note that values of 1.00 indicate that the median is at or very near 1, but individual games still vary in performance.

RNN	TWED	Top3 Err Total	F1 Total	Top3 Err NE	Precision NE	Recall NE	F1 NE
Player 1 Replays							
All	$\mathbf{4.73 \pm 0.97}$	0.014 ± 0.016	0.60 ± 0.26	$\mathbf{0.39 \pm 0.14}$	$\mathbf{0.026 \pm 0.0095}$	$\mathbf{0.34 \pm 0.15}$	$\mathbf{0.048 \pm 0.017}$
50	5.82 ± 1.41	0.058 ± 0.088	0.41 ± 0.42	0.45 ± 0.12	0.017 ± 0.0085	0.30 ± 0.13	0.03 ± 0.016
P2	6.48 ± 1.45	$\mathbf{0.013 \pm 0.013}$	$\mathbf{0.61 \pm 0.21}$	0.45 ± 0.17	0.016 ± 0.0072	0.30 ± 0.13	0.031 ± 0.014
Player 2 Replays							
All	6.58 ± 1.67	0.018 ± 0.021	0.62 ± 0.21	$\mathbf{0.48 \pm 0.14}$	0.018 ± 0.0096	$\mathbf{0.31 \pm 0.13}$	$\mathbf{0.035 \pm 0.018}$
50	5.42 ± 1.64	$\mathbf{0.016 \pm 0.012}$	$\mathbf{0.78 \pm 0.11}$	0.52 ± 0.14	0.016 ± 0.0087	0.15 ± 0.073	0.029 ± 0.014
P1	$\mathbf{5.02 \pm 1.18}$	0.028 ± 0.029	0.61 ± 0.21	0.56 ± 0.17	$\mathbf{0.019 \pm 0.0099}$	0.23 ± 0.11	0.034 ± 0.019

Table 2: The medians and interquartile ranges for the metrics of the RNNs on the "all factions" combined data sets. For each player's test set, we report the results for the model trained on all training data (All), the model trained on 50 randomly selected games (50), and the other player's full model (P2 or P1). Bolded results are the best for a metric for each test set.

Results

The single-faction-matchup results are shown in Table 1. This table contains the metrics for each data set, for each player, separated by method. We report the medians and interquartile ranges for each metric, because all are non-normally-distributed for at least some data sets. For micro-averages over all classes in multi-class classification, precision, recall, and F1 score are all the same. Therefore, we report only the F1 score. In the case of non-empty action predictions, however, these are distinct, so we report all three.

In Table 2, we report the RNN's results on the combined all-factions data set, for each player. We report only the RNN because the other methods have already shown much lower performance on single factions (as discussed below), and because RFs ran out of memory on the larger data set. This table also shows the performance impact of data set size.

We also report tests of the RNN, trained on a particular player, against the test set from the other player. By comparing the cross-trained model with the regular one, we can determine how much of its predictive power is specific to the player. The results of this test are shown in Table 2. We performed a statistical analysis on the non-empty action F1 scores, since they represent a balanced comparison of precision and recall for the build actions. The F1 scores are non-normally-distributed, so we used the two-tailed Wilcoxon Signed Rank Test with paired samples, with $\alpha < 0.01$. We found that for both players, the test showed a significant difference in F1 scores ($Z = -7.527, p < 0.01$ for P1's replays and $Z = -5.179, p < 0.01$ for P2's). The effect size is large for P1 ($r = 0.72$) and medium for P2 ($r = 0.42$).

There is similarly a significant difference in non-empty action F1 scores for the models trained on 50 replays versus the full models ($Z = -7.891, p < 0.01$ for P1, $Z = -4.861, p < 0.01$ for P2) with large effect size for P1 ($r = 0.64$) and medium for P2 ($r = 0.39$).

Discussion

RNNs as a Starting Point for Player Imitation

As expected, the random baseline performs very poorly, since random guessing is unlikely to mimic a player. On the other hand, RFs achieve a very high overall F1 score, which might appear at first to be a positive result. They have, however, an extremely low recall (below even the baseline)

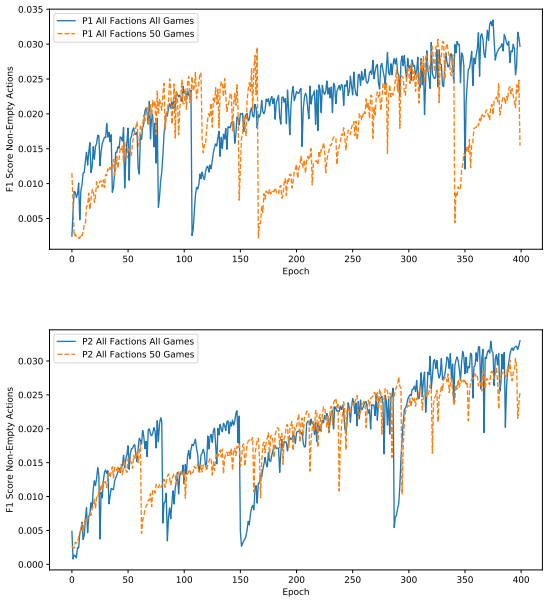

Figure 1: The non-empty action F1 score on the validation set during training for the all games vs. 50 games data sets, recorded every 10 epochs, for P1 (top) and P2 (bottom).

and high error rate for non-empty actions. This is because RFs fall victim to the class imbalance between empty and non-empty actions: they predict empty actions almost all the time, rather than risking a guess at a build action. Thus, the RF precisions are high because their extremely rare non-empty guesses tend to be very certain. This leads to a misleadingly high overall F1 score, when really the RFs are not modeling the player's actions well at all. High precision but low recall indicates that, on the rare occasion where they would choose to act, RFs would tend to match the player's action, but they would fail to act at all in most states. With 21-37% recall for non-empty actions while still achieving 45-78% for all actions, the RNNs much more effectively balance the prediction of specific actions against empty actions.

While RNNs are by far the most successful of the techniques tested here, their performance still leaves much to be desired. While 45-78% overall accuracy is much better than chance, it is far from a perfect mimicry of the player's strategic decision-making. This is especially evident in the low non-empty action precision scores, which indicate that the RNN repeatedly predicts an action incorrectly around the times when it is predicting it correctly. This may be partly due to the test setup: the replay is unchangeable. If the AI guesses a unit to build too early, it will not actually start that unit's construction. Thus, the next frame of the replay will look relatively similar, and the AI might guess the same action repeatedly. In a real game, the AI would be able to begin construction and immediately return to predicting empty actions. This hypothesis is borne out by the TWED scores, which show that, on average, each action is only 5-7 edits away from the correct sequence.

Moreover, the RNNs achieve approximately 2-5% top-3 error, which means that in all but a few cases, the player's actual action is within the top three predictions. Thus, the RNN could likely be combined with other decision-making to narrow the set of possible actions to the ones the player would be most likely to perform. Such hybrid decision-making could be a fruitful avenue for future work.

When running the RNNs trained for one player on the other's replays, we see a decrease in performance, as expected. Nonetheless, the two models are not entirely unsuccessful in predicting the other player's actions, which could indicate either that there are some similarities in how and when these players tend to build units in similar situations, or that the model is not yet fully capturing the individual differences in play, or some combination of the two.

In comparison to the neural-network-based predictions by Justesen and Risi (2017), our RNNs have a much smaller top-3 error ($\sim 2-5\%$, where they report $22.92\% \pm 0.09\%$). Additionally, they predict only non-empty actions based on the current game state, ignoring the sequence and timing, whereas we incorporate these. Our top-3 error for non-empty actions is higher (39 and 48% on the full data sets), but this is measured on the harder problem of predicting precise action timing. Finally, we predict the actions of a single player, whereas they draw their data from many players, providing them a larger training set of over 2000 replays.

We note that there is a drop in performance from training on a smaller data set, as shown in Table 2. Though the RNNs trained on 50 games are still somewhat able to imitate actions, they are unable to generalize as well as the full models. In training graphs of their non-empty F1 scores, shown in Figure 1, we can see that they almost match the full data set, only falling behind in the final stretch as the larger set fine-tunes its results. Further testing, preferably by expert review of games, would be necessary to determine how noticeable the difference in imitation quality is. It may well be that different goals for imitation, different games, and players' play styles will affect the required data set size.

We only tested imitation of two players, which prevents a claim of generalization to all players and factions. We do believe, however, that the size and diversity of the two tested data sets provide some indication of generalizability. Each data set contains a wide variety of maps, faction match-ups and game lengths. Therefore, we claim that this RNN-based approach can be useful as a starting point for future research.

Future work could explore improving on the game state features, iterating on the RNN structure, or switching to another learning technique such as inverse reinforcement learning (Abbeel and Ng 2004). Another idea might be to filter each frame's actions to those that are actually feasible.

Limitations

Several limitations temper these findings and should inform future study. First, as previously mentioned, the RNN's lack of precision in determining exactly when to build a unit is particularly in need of improvement. We have not tested our predictions in a complete AI agent, which would be necessary to determine whether its decisions actually appear player-like. Also, importantly, our replays contained all state

data, but a real agent might not have access to perfect information. Future work should incorporate hidden information.

Moreover, our test set is small. We focused on two specific players in order to test, in depth, similar numbers of replays from each, but this limits our claim to generality in our results. We cannot claim that our model would definitely generalize to additional factions, players, or strategies that did not appear in our data set. Future tests should include a larger number of players, and especially those who play other factions. We are also predicting only unit construction actions; to fully imitate players, we would need to predict movement orders, requiring additional considerations such as spatial reasoning. Finally, if we can solve the challenges of fast forward simulation of game states and of evaluating the resulting AI's similarity to the player's behavior, it would be useful to test imitation learning-based approaches.

RNNs require significant time and resources to train before they can be deployed – though still within the realm of possibility for a single developer with a sufficiently powerful computer. With these data sets, training over 4000 epochs required approximately a day of real time on a single computer with an NVidia^{TM} GTX 1080 or 980 GPU.

Ethical Considerations for Player Imitation

In performing this work, and especially in future work as techniques continue to improve, we have a responsibility to consider many ethical questions. We foresee potential misuses of player imitation: companies could use it to deanonymize a player's games under multiple aliases, to detect and ban accounts shared between friends or family, or to build technology that replaces or fails to adequately compensate playtesters or other workers. If applied outside of games, these misuses of imitation could have an even broader impact. While player imitation has beneficial purposes, if used ethically for more adaptive AI, inclusive game design for players with disabilities, and more, these do not absolve researchers from considering potential abuses.

Even when using player imitation for beneficent purposes, we must consider autonomy and justice as well (Metcalf and Crawford 2016). How Common Rule principles apply to data science is still in discussion in the research community, but the uncertain status of data collected through telemetry and public submission necessitates more ethical consideration, not less (Fairfield and Shtein 2014). When we collect data for player imitation, it is important that players are informed and consent, and that we enable them to rescind consent and delete their data. Current game license agreements do not provide sufficient protections (Canossa 2014). Perhaps the most effective option would be to develop player models that can be trained and deployed on the player's own computer, without providing data to a central server. Protecting privacy should be a focus of future imitation research.

Justice also requires that players who are imitated be drawn from many communities, backgrounds, and abilities. We must ensure even distribution of the work and risks to those who provide replays, and the benefits to players. Marginalized communities are at more risk of abuse and harassment, even when efforts are made to evenly represent them (Fairfield and Shtein 2014). Mitigating this requires intentionally reaching out to and working with them, as empowered political agents, not merely as subjects. It means fairly compensating those who provide their data, protecting their identities, and supporting them against any abuse that results from participation. It also means ensuring that the resulting game features benefit people in those communities. For a more comprehensive discussion of ethics in player modeling, see Mikkelsen, Holmgård, and Togelius (2017).

Finally, player imitation risks bias and harm from the data and the designers' modeling choices. Repeated analyses show that systems are biased by the input data and by its transformation into input features (Mikkelsen, Holmgård, and Togelius 2017). If player imitation is provided to all players for training, intentionally or unintentionally abusive players may cause the system to imitate their abusive behavior. Even barring abuse, designers may choose data or transformations that bias the imitation towards their own preferred features or play styles, leading to poor experiences for players with different preferences or abilities. Future research must interrogate and account for these biases.

Conclusion

Prior game AI research has sought to build generically human-like agents and to model play-styles based on replay data. For automated playtesting or for modeling agents on specific players, however, a more individualized player imitation approach is necessary. In automated playtesting, such a model could be valuable for designers to gain insight into how certain players might interact with their content. It could also inform an opponent or cooperative model to help AI agents develop a "theory of mind" about the player.

StarCraft, with its mix of high-level strategy and fast-paced, low-level unit control, is a difficult test for player imitation. In this study, we examined player imitation based on a random baseline, random forests, and recurrent neural networks. RNNs were partially successful in predicting the strategic actions of a specific player, even with limited data and in all faction matchups. There is still significant room for improvement in precision, however. Hybrid decision-making that better incorporates domain knowledge might lead to significant advances. We hope that these experiments will be a step on the path to robust player imitation techniques to make games more accessible and adaptive to people with various abilities and interests. We also encourage future research to grapple with the ethical implications of imitation, especially with issues of privacy and equity.

Acknowledgments

We would like to thank Professors Seth Cooper and Christopher Amato for their advice and support.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. ACM.

Arulkumaran, K.; Cully, A.; and Togelius, J. 2019. AlphaStar: An Evolutionary Computation Perspective. *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '19* 314–315. arXiv: 1902.01724.

Bryant, B. D., and Miikkulainen, R. 2007. Acquiring Visibly Intelligent Behavior with Example-guided Neuroevolution. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1*, AAAI'07, 801–808. Vancouver, British Columbia, Canada: AAAI Press.

Canossa, A. 2014. Reporting from the snooping trenches: Changes in attitudes and perceptions towards behavior tracking in digital games. *Surveillance & Society* 12(3):433–436.

Cardamone, L.; Loiacono, D.; and Lanzi, P. L. 2009. Learning drivers for TORCS through imitation using supervised methods. In *2009 IEEE Symposium on Computational Intelligence and Games*.

Cho, H. C.; Kim, K. J.; and Cho, S. B. 2013. Replay-based strategy prediction and build order adaptation for StarCraft AI bots. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*.

Dereszynski, E. W.; Hostetler, J.; Fern, A.; Dietterich, T.; Hoang, T.-T.; and Udarbe, M. 2011. Learning Probabilistic Behavior Models in Real-Time Strategy Games. In *AIIDE 2011*.

Fairfield, J., and Shtein, H. 2014. Big data, big problems: Emerging issues in the ethics of data science and journalism. *Journal of Mass Media Ethics* 29(1):38–51.

Farouk, G. M.; Moawad, I. F.; and Aref, M. M. 2017. A machine learning based system for mostly automating opponent modeling in real-time strategy games. In *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, 337–346.

Geisler, B. 2002. *An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a 'First Person Shooter' Video Game*. Ph.D. Dissertation, University of Wisconsin - Madison.

Ghasemipour, S. K. S.; Gu, S.; and Zemel, R. 2019. Understanding the Relation Between Maximum-Entropy Inverse Reinforcement Learning and Behaviour Cloning. In *ICLR 2019 Workshop on Deep Generative Models for Highly Structured Data*.

Ho, J., and Ermon, S. 2016. Generative Adversarial Imitation Learning. *arXiv:1606.03476 [cs]*. arXiv: 1606.03476.

Ho, T. K. 1995. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, 278–282. IEEE.

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

Holmgård, C.; Green, M. C.; Liapis, A.; and Togelius, J. 2018. Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics. *arXiv:1802.06881 [cs]*. arXiv: 1802.06881.

Hsieh, J. L., and Sun, C. T. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 3106–3111.

Hussein, A.; Gaber, M. M.; Elyan, E.; and Jayne, C. 2017. Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.* 50(2):21:1–21:35.

Justesen, N., and Risi, S. 2017. Learning Macromanagement in StarCraft from Replays using Deep Learning. *arXiv:1707.03743 [cs]*. arXiv: 1707.03743.

Kingma, D. P., and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd Annual International Conference for Learning Representations*. arXiv: 1412.6980.

Laird, J. E., and Duchi, J. C. 2000. Creating human-like synthetic characters with multiple skill-levels : A case study using the Soar quakebot. In *AAAI Fall Symposium Technical Report*.

Lin, Z.; Gehring, J.; Khalidov, V.; and Synnaeve, G. 2017. STARDATA: A StarCraft AI Research Dataset. *arXiv:1708.02139 [cs]*.

Liu, S.; Ballinger, C.; and Louis, S. J. 2013. Player identification from RTS game replays. *Proceedings of the 28th CATA* 313–317.

Llargues Asensio, J. M.; Peralta, J.; Arrabales, R.; Bedia, M. G.; Cortez, P.; and Peña, A. L. 2014. Artificial Intelligence approaches for the generation and assessment of believable human-like behaviour in virtual characters. *Expert Systems with Applications* 41(16):7281–7290.

Marteau, P.-F. 2009. Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. *IEEE transactions on pattern analysis and machine intelligence* 31:306–18.

Metcalf, J., and Crawford, K. 2016. Where are human subjects in big data research? The emerging ethics divide. *Big Data & Society* 3(1).

Mikkelsen, B.; Holmgård, C.; and Togelius, J. 2017. Ethical Considerations for Player Modeling. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

Muñoz, J.; Gutierrez, G.; and Sanchis, A. 2010. A human-like TORCS controller for the Simulated Car Racing Championship. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 473–480.

Neal, D., and Hayles, B. 2016. Designing AI for Competitive Games. The Game Developers Conference.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2007. Case-Based Planning and Execution for Real-Time Strategy Games. In *Case-Based Reasoning Research and Development*, Lecture Notes in Computer Science, 164–178. Springer, Berlin, Heidelberg.

Priesterjahn, S.; Kramer, O.; Weimer, A.; Weimer, E.; and Goebels, A. 2005. Evolution of Reactive Rules in Multi Player Computer Games Based on Imitation. In *Proceedings of the International Conference on Natural Computation*.

Robertson, G., and Watson, I. 2015. Building behavior trees from observations in real-time strategy games. In *International Symposium on Innovations in Intelligent SysTems and Applications*.

Schrum, J.; Karpov, I. V.; and Miikkulainen, R. 2011. UT^2: Human-like Behavior via Neuroevolution of Combat Behavior and Replay of Human Traces. In *Computational Intelligence in Games*.

Shore, J., and Johnson, R. 1981. Properties of cross-entropy minimization. *IEEE Transactions on Information Theory* 27(4).

Stanescu, M., and Čertickỳ, M. 2016. Predicting opponent's production in real-time strategy games with answer set programming. *IEEE Transactions on Computational Intelligence and AI in Games* 8(1):89–94.

Synnaeve, G., and Bessiere, P. 2011. A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft. In *Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011*.

Synnaeve, G.; Nardelli, N.; Auvolat, A.; Chintala, S.; Lacroix, T.; Lin, Z.; Richoux, F.; and Usunier, N. 2016. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*.

Togelius, J.; De Nardi, R.; and Lucas, S. M. 2006. Making racing fun through player modeling and track evolution. In *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*.

van Hoorn, N.; Togelius, J.; Wierstra, D.; and Schmidhuber, J. 2009. Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation*, 652–659.

Weber, B., and Mateas, M. 2009. A data mining approach to strategy prediction. In *Computational Intelligence and Games*.