Improvements in Imitation Learning for Overcooked

Duuk Niemantsverdriet[1]

Supervisor(s): Frans Oliehoek[1]**, Robert Loftin**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Duuk Niemantsverdriet
Final project course: CSE3000 Research Project
Thesis committee: Frans Oliehoek, Robert Loftin, Klaus Hildebrand

Abstract

Arguably the main goal of artificial intelligence is to create agents that can collaborate with humans to achieve a shared goal. It has been shown that agents that assume their partner to be optimal can converge to protocols that humans do not understand. Taking human suboptimality into consideration is imperative to perform well in a coordination task. One way to achieve this is imitation learning, where you train an agent on recorded data from a human playing optimally. I created several agents using different implementations of behavioral cloning, by reducing the dataset to state-action pairs and training a neural network on this. To evaluate their performance, I used an environment that poses a coordination challenge based on the popular game *Overcooked*. Neither expanding nor reducing the feature space that the agents are trained on yielded any significant improvement in the performance of the agents. In fact, expanding the feature space to include some historical data made the agent less generalizable and especially failed to perform when paired with agents with unfamiliar strategies. These limitations were mostly posed by the available dataset, which was not big enough to support more features and of too low quality of gameplay to create agents that perform exceptionally well.

1 Introduction

Over the last few years, there have been considerable advances in creating artificial intelligence that can play video games. These AI agents are often trained using self-play, where they are matched up against different versions of themselves. This method has yielded great results in various competitive zero-sum games such as Go [1; 2] and Quake [3]. Despite not encountering any accurate human behavior during their training, these agents still perform well when playing against a human. In most cases, they even perform better against humans. This tendency stems from the AI expecting its opponent to play optimally like itself, so when the human makes an unexpected move, that is usually in favor of the AI agent.

However, one of the main goals of artificial intelligence is to train agents that help people achieve what they want, and augment their capabilities [4]. Collaborative video games are excellent, low-stakes environments that we can use to test out different approaches to human-AI collaboration. They can provide us with interesting challenges that a team of a human and an AI have to overcome. It seems tempting to rely on self-play again. This has been successfully done on a few occasions, such as in Capture the Flag [3], but the high scores could also have resulted from the AI's sheer skill instead of collaboration. Pairing up artificial intelligence with a copy of itself fails to take into account human behavior, and incorporating human data or models in the training process yields significantly better results on collaborative tasks [5]. In that particular paper, they used a few different approaches to exploiting human data to train the AI agent, such as proximal policy optimization, coupled planning, and imitation learning, to train agents that could collaborate with humans in an environment that was based on the popular game *Overcooked* [6].

In this paper, I used imitation learning to train collaborative AI agents for *Overcooked*, to replicate and improve on the results of paper [5]. To this end, I formulated the following research question:

> How well can a collaborative AI agent that was trained by directly imitating human-generated data perform in *Overcooked*?

To guide me in the process of answering these questions, I compiled the following sub-questions:

- What properties should a good collaborative agent have?
- Which approach to imitation learning yields the best agent?
- How well does an agent that was trained using this approach perform?

To answer this question, I used the same simple environment based on *Overcooked* (Figure 1), which is designed specifically as a challenge for humans to coordinate in. I use this environment to compare experimental collaborative agents trained with different variants of behavioral cloning [7] against agents with different strategies. For the latter, I used another behavioral cloning agent, an agent trained using self-play, and a scripted agent.

I find that different variations of behavioral cloning, where the agents are trained on different features, do not significantly improve their results in the given challenge. Especially agents that are also trained on some features that are based on their previous actions perform considerably worse against agents whose strategy they are not familiar with. These agents also likely suffered from the curse of dimensionality, since the dataset that was available was quite small. Experiments that used smaller feature vectors did not markedly improve performance either. This is due to limitations imposed by the quality of the dataset.

2 Background

The Overcooked environment that I worked with [8] was originally made by the authors of [5]. It is a simplified version of the original game *Overcooked* [6], a game where you must work together to run a kitchen and serve as many dishes as possible within a time limit. It was specifically created to be a coordination challenge for humans. A simplified version of the game was created to shrink the state space, such that training an agent is achievable within a reasonable amount of time. This version of the game has discrete movement and time steps, as well as fewer distinct recipes. It is shown in Figure 1.

Imitation learning is an approach to machine learning where you train an AI agent to replicate the behavior of a human [9]. There are a few different approaches to imitation learning:

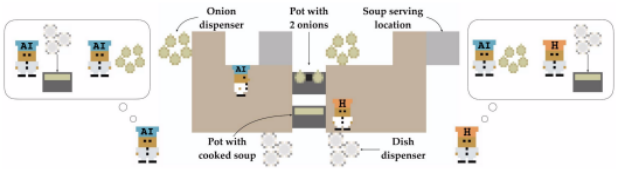

Figure 1: The simplified *Overcooked* environment. The goal is to place three onions into a pot (dark gray), wait for the soup to cook, and serve it at a soup serving location (light gray) to get points. Within a set amount of time, you need to get as many points as possible. Picture taken from [5].

- **Behavioral Cloning (BC)** exploits supervised machine learning algorithms to directly teach an AI agent human behavior [7]. It requires a dataset of expert demonstrations, which is then reduced to state-action pairs and used as training data for a supervised machine learning algorithm, commonly a neural network.
- **Direct Policy Learning** is an iterative version of behavioral cloning [10]. First, a policy is created using regular behavioral cloning. Then, an interactive expert gives feedback on the performance and the decisions made by the agent. Then the new data is incorporated into the agent, and this is repeated a few times.
- **Inverse Reinforcement Learning (IRL)** is an approach to imitation learning that uses the power of reinforcement learning. There are many tasks for which defining a reward function manually is too complex, such as autonomous driving. When doing IRL, only expert demonstrations for different situations are given, and the goal is to learn what the reward function is [11]. This function can then be used in a regular reinforcement learning algorithm.
- **Cooperative Inverse Reinforcement Learning (CIRL)** is a similar approach to IRL, but specifically tailored to collaborative settings. Instead of working under the assumption that the human would be optimal by itself and attempting to learn their reward function for itself, CIRL allows for easier value alignment due to behaviors of active teaching and learning [12].

The largest benefit of using imitation learning for coordination is that by using human examples, the AI agent is aware of how a human would play and will react to its partner's actions the same way a human would. On the other hand, training agents on human data, especially by means of behavioral cloning, could limit their performance at the skill level of the expert demonstrations, as the AI cannot ever discover strategies that may be better than what the expert has shown them.

3 Imitation Learning in Overcooked

This study is an extension of the one that originally worked with this Overcooked environment to develop collaborative AI, [5]. For that study, they used imitation learning to train a sort of "proxy human" that is supposed to behave similarly to a real human player. Next to each of their experiments, they ran a test where two behavioral cloning agents completed the same challenge as a control condition to compare the other results with. However, they did not attempt to exploit imitation learning for the development of a better AI agent.

The purpose of my research is to see if I can match or even improve on the results of [5] using an imitation learning approach. First, I reproduced the results that the original paper used as control conditions. This is important to create a baseline of performance. I used a dataset of human players playing the game that they collected, which has been anonymized. This data has been reduced to state-action pairs. A behavioral cloning agent is implemented using a neural network, with the state as input and the next action to take as output. However, the state needs to be transformed into a feature vector to be able to input it into a neural network. A featurization function is defined for this purpose. This function reduces each state to features that keep track of the player locations, the object that they are holding, how many onions are in the pots, etc.

To iterate on the application of behavioral cloning, I changed the featurization function. Since humans use their memory when making decisions on what to do next, I attempt to reflect that in my BC algorithm. There are two options. The first is to take into account one or more complete past states. Since every extra state that is included must double the size of the neural network, this can get overly computationally heavy. It is also subject to the curse of dimensionality, which tells us that in order to obtain a reliable result, the amount of data needed often grows exponentially with its dimensionality [13]. The other option is to select only a few features from past states, such as the past actions of the other player, and add those to our input vector. If present decisions always rely on the same kind of historical data, this can yield great results, since we can encode a lot of history if we do not need to incorporate every detail.

Additionally, since the provided dataset is somewhat limited, it is possible that the standard behavioral cloning agent already suffers from the curse of dimensionality. To test this, I also tried to alter the featurization function in such a way that we only take into account the features that are useful for the BC models to decide which action to take next. Although it seems counterproductive for a collaborative application, I tried to limit the number of features that gave information about the other player. I hypothesize that by being more selective about which features to keep and which to leave out, I can get the most mileage out of the limited data that I was provided.

I will evaluate each agent that I develop against an array of different agents, to somewhat emulate the fact that different human players can have radically different playstyles. In addition to the standard human proxy agent, I will test my agents against an agent that was trained using self-play and a manually programmed scripted agent.

4 Experimental Setup and Results

4.1 Environment and Data

The environment comes with 5 predefined layouts, which test different challenges that arise in cooperative settings. The

Figure 2: The 5 different experiment layouts in the Overcooked environment. From left to right: *Cramped Room* has mainly low-level challenges in movement planning, as it is easy to run into each other in this confined space. *Asymmetric Advantages* gives both players the opportunity to work on their own, but an optimal strategy would see the player on the right supply the onions, as they are closer to the pot on their side, and the player on the left should deliver the dishes. *Coordination Ring* presents more complex movement planning challenges since players cannot walk past each other easily. *Forced Coordination* is all about devising a high-level coordination strategy since neither player can complete an order by themself. Finally, *Counter Circuit* presents a non-obvious way to work together more efficiently, which involves passing onions over the counter instead of walking around.

different maps are displayed in Figure 2. Each approach that I tried was tested on all 5 of these maps, to test their performance under different challenges, ranging from low-level interactions and movement challenges to more high-level strategy challenges.

The authors of [5] put together a dataset of human players playing the game, which I used as my source of human data. It was crowdsourced through Amazon Mechanical Turk. They filtered out very suboptimal trajectories that got fewer points than a single player could achieve and ones that did not fit their minimum length of 1200 timesteps. Each trajectory with two agents is split into two single-agent trajectories, which contain information relative to each respective agent. In training, 85% of the data is used as training data and 15% for validation.

4.2 Agents

I trained my agents using behavioral cloning on the aforementioned dataset. Their performance was evaluated against three different types of other agents: the human proxy model, an agent trained using self-play, and a scripted agent.[1] The human proxy model is also a behavior cloning agent, trained to the same specifications that [5] used for their human proxy model. To train two distinct agents on this dataset, I split the data into two parts. One of these parts was used to train the human proxy models, while I used the other part for my experimental agents. I did this to prevent any advantage that the agents would have by being trained on the exact same examples, which simulates a situation where the human's strategy is unknown more accurately.

A behavior cloning agent has a neural network at its core, with two hidden layers of size 64. The learning rate was 1e-3 and the number of epochs differed per layout: 100 for *Cramped Room*, 120 for *Asymmetric Advantages* and *Coordination Ring*, 90 for *Forced Coordination* and 110 for *Counter Circuit*. These are the same parameters used in [5], and they account for the differing size of datasets across the maps. Fewer epochs were used if there was less data available to mitigate the effects of overfitting. The behavior cloning agents are trained on a manually created 96-dimensional featurization of each state, to incentivize the generalization of policies in spite of the limited available data. The standard featurization contains the relative positions from the player to the other player, the closest onion, dish, soup, serving location, and pot, the states of each pot in the layout, the absolute position of the player and which direction they're facing, and all of this information for the other player as well. Sometimes agents can get stuck in their coordination of low-level actions, such as continually running into each other. To correct this tendency, a hardcoded behavior was added where the model will choose a random action if it is stuck in the same position for three consecutive timesteps.

To train agents using self-play, I used Proximal Policy Optimization (PPO). Unlike behavioral cloning, this model was trained with a lossless state encoding consisting of 20 masks that had the same size as the grid of the map. Each mask contains information about a specific aspect of the state, including the positions and orientations of players and the locations of various types of objects. In order to speed up training, the reward function was shaped to give agents some reward when placing an onion into the pot, when picking up a dish while soup was cooking, and when picking up soup with a dish. The amount of reward shaping is reduced to 0 over the course of training, with a linear schedule. The policy is parameterized with a convolutional neural network with three convolutional layers of size 5×5, 3×3, and 3×3 respectively with 25 filters each, followed by 3 fully connected layers of hidden size 32. The hyperparameters used are reported in Table 1. These are the default parameters in the GitHub repository that I worked with and were optimized by an anonymous contributor.

The scripted agent uses a greedy approach to the challenge. Its default behavior is to deliver onions to the best possible place, which is either the pot that already contains the most onions or a counter to pass it to the other player if the soup is too far or unreachable. When a soup has finished cooking, the agent will try to pick up the soup and deliver it, unless the other player is clearly closer. If there is no plate in reach of a finished soup, this agent will try to bring a plate to a counter that is reachable from the pot.

4.3 Experiments

The first step was to establish a baseline for the performance of different agents. Two groups of five behavioral cloning agents were trained on each map. I picked one agent out of each group. One of the groups would produce the human proxy model for all of the experiments, and the other the BC agent that I test against all of the different agents as a baseline test. To simulate the tendency of imitation learning agents to produce an agent that is not quite as good as a real human player, I picked the agents such that the human proxy model is slightly better than the BC agent I compare to. Then I evaluated the performance of these agents. The results of this can be found in Figure 3. We can see that this agent especially lacks performance on the *Forced Coordination* and *Counter Circuit* layouts, both of which have mostly strategic challenges.

[1] I did not make this scripted agent myself. It was developed by Anton Mihai Cosmin for his research project, which we did alongside each other.

PPO hyperparameters					
Parameter	Cr. Room	Asym. Adv.	Coord. Ring	Forced Coord.	Counter Circ.
Training iterations	550	650	650	650	650
Learning rate	1.63e-4	2.1e-4	1.6e-4	2.77e-4	2.29e-4
Clipping	0.132	0.229	0.069	0.258	0.146
Gamma	0.964	0.964	0.975	0.972	0.978
Maximum gradient norm	0.247	0.256	0.359	0.295	0.229
Entropy coefficient	0.197	0.185	0.156	0.31	0.299
Lambda	0.6	0.5	0.5	0.6	0.6
Reward shaping horizon	4.5e6	5e6	5e6	4e6	5e6
VF loss coefficient	0.00995	0.022	0.00933	0.016	0.00992

Table 1: Hyperparameters for self-play PPO across the 5 layouts.

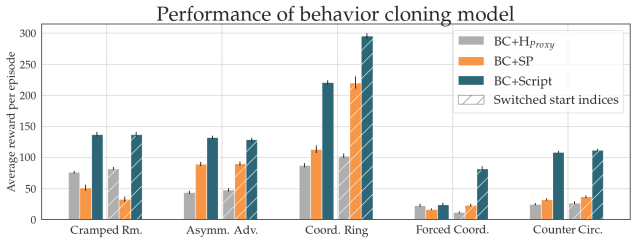

Figure 3: The results of a basic BC agent with the same specification as the human proxy agent. Shown is the mean reward with standard error over 100 simulated games.

To try to improve my agents, I started making changes to the featurization function. The standard function that was used for this contains all data from both players in this current state. I hypothesized that having fewer features to train the neural network on could lead to better performance since we have limited training data. I tried two different featurization functions that don't take all of the data of the other player. BC_1 only trains on data relative to itself (the complete set of all distances to pots, onions, dishes, etc., and the statuses of the pots) and the relative distance to the other player. BC_2 has four additional features that represent what the other player is holding so they can anticipate the other player's actions better. This is likely to pay off most in situations where strategy is important. The dimensions of the feature vectors are 50 and 54 respectively. Figure 4 shows the results of these experiments, grouped by the type of agent they were paired with.

It is visible that these two alterations to the function did not significantly improve the performance of these agents over the one that uses the standard featurization. Looking at relative differences in the means, the scores of both agents are rarely more than one or two standard errors off from the baseline. However, BC_2 seems to consistently have a slight edge over BC_1 across the board, which is especially visible in the human proxy trials. From this, we can conclude that at least some of the information of the other player is valuable knowledge for a BC agent.

On the other side of the spectrum, I trained some agents that have a larger featurization than the standard agent. I hypothesized that if an agent has more input, and therefore more variables, perhaps it can find patterns in data that it was not

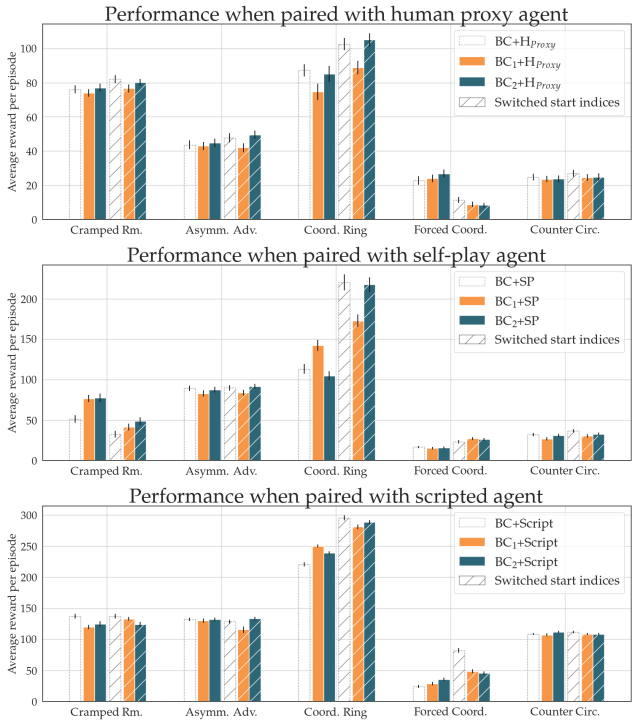

Figure 4: The results that agents BC_1 and BC_2 were able to obtain in the game when paired with a human proxy agent (top), a self-play agent (middle), and a scripted agent (bottom). Means and standard errors were computed from the rewards obtained in 100 simulated games. In orange are the results from the agent that trains only on data relative to itself, and in blue those of the agent that is also trained on what the other player is holding. In white is the comparison with the standard BC agent. Neither agent shows strong improvement, although BC_2 is consistently slightly better than BC_1.

getting before. Especially giving the agents some historical data to work with may yield more accurate representations of human players since humans tend to use their short-term memory when deciding what action to take next. However, it is possible that, by giving the neural network more features on such limited data, we might run into the curse of dimensionality, and the agent cannot distinguish which features are actually important to look at. I trained three more agents with different featurization functions. BC_3 is trained using the same set of features as BC_1, with additionally the actions taken by both players for the past 5 timesteps. BC_4 also trains on these past actions, but also on all the features of both players, much like the standard featurization function. Finally, BC_5 is trained on the standard featurization from both the current and the previous state, including the previous action that was taken by each player. The dimensionalities of these vectors are 114, 156, and 204 respectively. To account for these feature vectors being significantly larger than those from the standard featurization function, I opted to increase the number of epochs by 50%, 100%, and 150% respectively on each map. Figure 5 shows the results of these trials.

BC_3 and BC_4 do significantly worse when paired with the self-play and scripted agents. This can be explained by the fact that this agent was basing choices on the previous actions of the other player, which was always a human in the training data. Therefore, when paired with an agent with an unfamiliar strategy, who might take different actions to achieve something similar, this agent gets confused and makes incorrect decisions about the next action to take. This result suggests that the previous few actions add little useful information for deciding which action to take next. BC_5 did better than the other two experimental agents, but it also did not improve on previously attained results. This is likely caused by the overwhelming number of features, which this somewhat small dataset cannot support. Additionally, it seems that taking into account some historical data seems to harm the performance more than it helps it, due to becoming less generalizable.

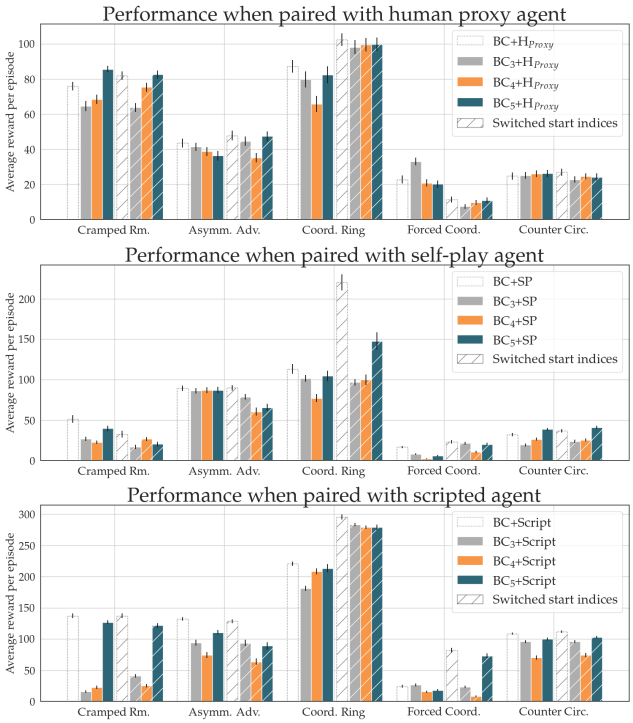

Figure 5: The results that agents BC_4 and BC_5 were able to obtain in the game when paired with a human proxy agent (top), a self-play agent (middle), and a scripted agent (bottom). Means and standard errors were computed from the rewards obtained in 100 simulated games. In gray are the results of the agent that on top of that also knows what actions were taken for the previous five timesteps. In orange are the results of the agent that had all the features of the standard agent plus the previous five actions of each player, and in blue are those of the agent that had a complete standard featurization of both the current and the previous state. These results are compared to the baseline test in white. BC_3 did much worse in every test and has likely suffered from seeking patterns in the wrong features. We see the same pattern in the performance of BC_4 confirming that basing decisions on past actions of both players only makes them less versatile. BC_5 was nearly as good as the baseline, but likely suffered from the curse of dimensionality to some extent.

5 Responsible Research

5.1 Human Data

The human player data that I had access to for this study was collected by the authors of [5]. No information was collected about the participants, besides their assigned anonymized identifier, which can be used to distinguish between two different playing styles if this is desired.

5.2 Reproducibility

All of the code that I used to produce my results is available at https://github.com/DuukPN/overcooked_ai. By using this code and following the descriptions in this paper, the results I got should be reproducible. Since the behavioral cloning agents are not deterministic, I evaluated each pair of agents 100 times to account for fluctuations in the obtained rewards.

5.3 Ethical Considerations

My artificial intelligence agents have no way of displaying undesired bias if they would be deployed. The most serious

effect that the discussed agents can have is that people with certain playstyles may obtain worse results in the game for which this agent was produced. The technologies discussed in this study are merely designed to learn from human behavior by trying to imitate it, so negative effects can only arise from unrepresentative or maliciously cherrypicked datasets.

6 Related Work

The paper on which this study expands, [5], is a study that investigated the importance of incorporating human data into training a collaborative AI agent. It compared the performance of different AI techniques, such as population-based training (PBT), proximal policy optimization (PPO), and coupled planning (CP), to that of classic self-play agents. They used behavioral cloning to create a human proxy model, which was used to measure the quantitative performance of the AI agents. There was also a separate behavioral cloning model which they paired with the human proxy model to create a baseline for how well a pairing of two humans would perform. However, they did not attempt to exploit imitation learning by itself as a technique to create a collaborative agent that rivals the performance of other well-performing agents that their study produced. There have been several other studies in this field that used the same environment, which investigated e.g. introducing diversity into training partners [14], testing the robustness of collaborative agents [15], and generating environments to train in [16].

7 Discussion

7.1 Reflection

None of the achieved results were significantly better than the basic behavioral cloning agent. It is clear that just changing the featurization function can only do so much to improve the performance of behavioral cloning. The lack of improvement with smaller feature vectors was likely due to the limited quality of the provided dataset. On the other hand, because the set is so small, adding more features is more likely a curse than a blessing. That can be most clearly seen in the subpar performance of agent BC_4. It also seems that including the previous few actions for each player makes the agents too specialized in working together with specific playstyles. Other approaches to collaborative artificial intelligence are considerably more effective [5].

7.2 Limitations

This study had some obvious limitations, the most prominent of them being time. The provided codebase was very difficult to work with, so it took several weeks before I could even train my first agent, and another few before I managed to evaluate it. It was only then that I could start altering the code to work with my experimental agents. I only obtained quantitative data, but to draw a more complete picture it would have been worth it to seek out some qualitative results obtained by manually playing with the agents.

However, even without qualitative analyses, it seems unlikely that we can improve the agents on the dataset that I was provided with. If the resources were available, I would have collected my own dataset of expert demonstrations. The increased volume of data would also allow me to choose stricter criteria to filter the already existing dataset, leading to higher-quality observations on which the BC agents can base their behavior.

Additionally, I wanted to investigate some other approaches to imitation learning that I discussed in Chapter 2. Especially cooperative inverse reinforcement learning (CIRL) seems like a promising approach. It does not assume that the human is optimal and follows a specific reward function, but instead tries to help the human in optimizing their (unknown) reward function. That, in addition to the active teaching and learning behaviors that this method can sometimes show, suits the challenges that *Overcooked* brings to the table.

8 Conclusions and Future Work

Trying to improve collaborative AI agents using behavioral cloning alone did not yield any significantly improved results. Shrinking the dimension of the feature vector by leaving out as many features of the other player as possible did not significantly change the performance of the agent. This is due to the somewhat low quality of human player data in the dataset that I received to work with. It shows that the quality of the expert demonstrations is the main limiting factor for behavioral cloning. On the other hand, encoding more information about the state of the field by including some historical data seemed to only hurt the performance of the agent, especially against agents with unfamiliar strategies. This is most likely caused by basing the next action to take on patterns that appear in the actions of the human players in the dataset, which has a tendency to confuse the agents when they are paired with a player that has a different strategy.

If this study was less limited in time, I would have tried to collect more, better data to train the agents on. The dataset was the single most important limiting factor for the performance of behavioral cloning agents. On better data, the BC agents would have had more of a chance to be competitive with other approaches to collaborative AI.

While I have shown that behavioral cloning has little potential to create good collaborative agents, some future work is required that investigates the use of other forms of imitation learning. One technique that especially stands out as having a lot of potential is cooperative inverse reinforcement learning. This technique does not assume the human is optimal but instead tries to align its values with its partner by trying to optimize their value function.

Acknowledgements

I would like to thank Robert Loftin and Frans Oliehoek for the guidance and feedback during the project, and Anton Mihai Cosmin for providing me with his scripted AI agent.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap,

M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," 2016.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[3] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," 2019.

[4] S. Carter and M. Nielsen, "Using artificial intelligence to augment human intelligence," *Distill*, vol. 2, Dec. 2017.

[5] M. Carroll, R. Shah, M. K. Ho, T. L. Griffiths, S. A. Seshia, P. Abbeel, and A. Dragan, "On the utility of learning about humans for human-ai coordination," *NeurIPS*, 2019.

[6] Ghost Town Games, "Overcooked." https://store.steampowered.com/app/448510/Overcooked/, 2018.

[7] C. Sammut, *Behavioral Cloning*, pp. 93–97. Boston, MA: Springer US, 2010.

[8] M. Carroll and N. Miller, "Overcooked-ai." https://github.com/HumanCompatibleAI/overcooked_ai, 2019.

[9] Y. Yue and H. M. Le, "Icml2018: Imitation learning tutorial." https://sites.google.com/view/icml2018-imitation-learning/, 2018.

[10] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15 of *Proceedings of Machine Learning Research*, pp. 627–635, PMLR, 2011.

[11] S. Russell, "Learning agents for uncertain environments (extended abstract)," *COLT*, 1998.

[12] D. Hadfield-Menell, A. Dragan, P. Abbeel, and S. Russell, "Cooperative inverse reinforcement learning," *NeurIPS*, 2016.

[13] R. E. Bellman, *Dynamic programming*. Princeton, NJ: Princeton University Press, Oct. 1957.

[14] R. Charakorn, P. Manoonpong, and N. Dilokthanakul, "Investigating partner diversification methods in cooperative multi-agent deep reinforcement learning," *ICONIP*, 2020.

[15] P. Knott, M. Carroll, S. Devlin, K. Ciosek, K. Hofmann, A. D. Dragan, and R. Shah, "Evaluating the robustness of collaborative agents," 2021.

[16] M. C. Fontaine, Y.-C. Hsu, Y. Zhang, B. Tjanaka, and S. Nikolaidis, "On the importance of environments in human-robot coordination," 2021.