PokAImon Training: Competitive Multi-Agent RL with GAIL

Cole Sohn
Department of Computer Science
Stanford University
csohn@stanford.edu

Rita Tlemcani
Department of Computer Science
Stanford University
ritabt@stanford.edu

Abstract

We introduce a framework for using Generative Adversarial Imitation learning in a Reinforcement Learning setup to teach agents to compete in a sumo environment from human demonstrations. We train using Proximal Policy Optimization in a Unity Environment powered by the Unity ML-Agents toolkit. The agents used are Pokemon X/Y models custom rigged to interact with the Unity physics engine. We show that a fully-connected model trained with PPO with select features aided with a low-strength GAIL model performs best.

1. Introduction

Reinforcement Learning (RL) for competitive tasks is interesting because agents trained with relatively simple reward functions can exhibit complex emergent behavior that would otherwise require detailed environments or meticulously crafted reward functions. Emergent complex behavior is essential for many real-world RL applications such as autonomous vehicles, robotic manipulation, and enemy AI in games. Competitive RL agents have been shown to exhibit emergent behavior without the need for these hand-tuned steps.

For this project, we study and evaluate competitive RL by training Pokemon agents to autonomously compete in a sumo wrestling game to push the other off of a platform, with the goal of achieving good performance as a video game enemy AI. We use Generative Adversarial Imitation Learning (GAIL)[5] to aid the RL algorithm with human demonstrations. Using this setup for the sumo task, agents learn emergent sub-tasks such as locomotion, posture and stance, and strategy.

Our implementation and game link can be found at `https://github.com/ritabt/PokAImon`.

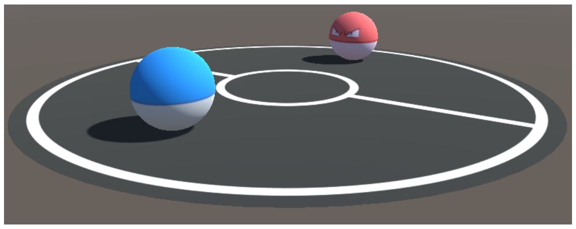
Figure 1. Illustration of Sumo Ball Agents, aka Voltorbs

2. Background

Physics-based 3D agents are trained to compete in a competitive sumo wrestling game. Agents can make observations about their environment such as position, rotation, velocity, and angular velocities of each limb, acceleration and angular acceleration of the body, and global position of the body. Agents learn policies that output target limb rotations in order to maximize reward by being the last agent on a platform. Agents are trained with self-play[11], meaning they compete against previous policies of their adversaries. This simulates practice with opponents of varying skill levels to stabilize training by ensuring no one agent dominates. We also experiment with GAIL[5], which is a flexible Imitation Learning[9] method that trains a neural network to evaluate how the agent's policy compares to human demonstrations.

We quantitatively evaluate our agent's performance and training using the ELO score[3]. The ELO score was introduced in the context of chess but is now a common way to quantitatively measure competitive RL models. The ELO score measures the improvement of the agent by making the latest version of the model compete against a previous iteration. We expect a log curve trend since towards the beginning of the training, the agents are learning at a higher rate thus winning more often against previous iterations. As the model converges and the training stabilizes we expect the ELO score to be more constant as competitors are at similar skill levels. We use other quantitative methods, see Section 4.1, to evaluate the impact of different training strategies.

We evaluate our agents qualitatively by inspecting emer-

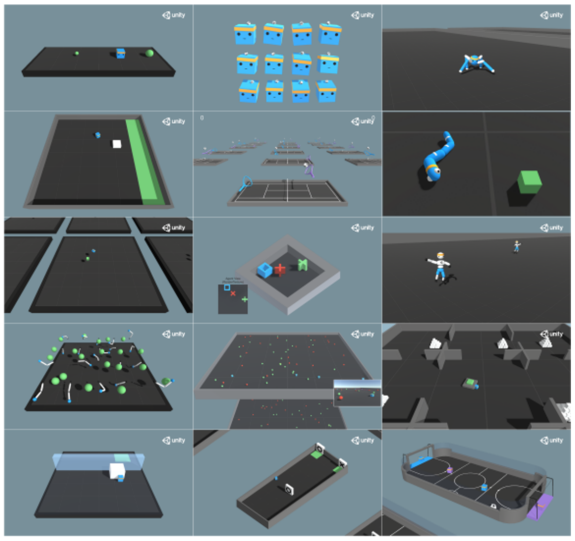
Figure 2. Example of environments available in Unity for Deep RL training - Image by Juliani et al. [7]

gent behaviors such as locomotion, stance, and strategies (such as trickery and building up velocity) during inference time. We also evaluate our agents qualitatively and inspect emergent behavior by piloting a human-controlled agent and observing the trained agent's reactions to user input.

2.1. Literature Review

We are using the Unity ML-Agents Toolkit developed and introduced by Juliani et al.[7] as an environment for our project. This Unity package offers a user-friendly API for simulated deep RL training. The API allows us to define different types of neural network architectures, different simulated environments and agents, as well as different training methods. The paper by Juliani et al.[7] introduced several example environments, see Fig.2, as well as a detailed study comparing different training and design strategies. Juliani et al. found that a PPO[10] trainer paired with an LSTM[6] network architecture performed best for training in 11 out of the 15 studied environments.

While the work by Juliani et al. gave us the tools and the background to work on our project, most of the strategies we are using are inspired by the work of Bansal et al.[2] from OpenAI. Bansal et al. did an extensive study of deep RL training of different agents in different simulated environments. They used the MuJoCo[12] simulator, which is a popular simulator for AI research but because of the limited visual rendering capabilities of the engine and the lack of complex lighting, textures, and shaders[7], we decided to use Unity as our simulator. Moreover, MuJoCo models are compiled which makes it impossible for us to set up a game-like environment where a human can fight against the AI.

Bansal et al.[2] also found that an LSTM[6] paired with PPO[10] worked best for many of the tested environments. One of the novel ideas they introduced is the concept of curriculum training. The idea is that it is very difficult to train a complex agent, such as a humanoid, to play a game like "You shall not pass" where one agent tries to block the other agent from reaching a goal. In curriculum training, the learning is done in steps. First, the agent learns how to walk and stand up, then the agent learns how to compete with another agent. This is done by using the same network in different setups as well as using a reward function that combines dense and sparse rewards. Dense rewards are calculated at every timestep and sparse rewards are calculated at the end of the episode. While dense rewards offer more detailed feedback to the agent and speed up the training, it is computationally expensive. Combining the two is achieved using a linear annealing factor α. So, at time-step t, if the exploration reward is s_t, the competition reward is R and T is the termination time-step, then the reward [2] is defined in Eq.1 below.

$$r_t = \alpha_t s_t + (1 - \alpha_t)\mathbb{1}[t == T]R \tag{1}$$

Another novel and interesting idea introduced by Bansal et al.[2] is the concept of distributed large scale self-play RL training. The idea here is to initialize many instances of one environment where each instance has two agents of the same type competing against each other. All of these instances are used to update the same network. This approach dramatically increases the learning speed by creating a batch of rollouts that can be used to update the network. Another main idea introduced with this concept is opponent sampling. With opponent sampling, the latest version of the trained network will randomly compete against an older version of the network. This prevents one agent type/role to overpower the competitor and can unstick training in some scenarios. This pipeline also allows us to compare an older version of the network to the most recent one which is necessary to evaluate the training with the ELO score [3].

Another influential paper in our work is the paper by Hester et al.[4] from DeepMind where they introduced Deep Q-Learning from demonstrations. The idea is that if we provide a RL training agent a human-made demonstration of what a "good" performance is, this can speed up the training by a lot and could even allow us to skip the curriculum training approach by Bansal et al.[2]. Hester et al. found that training a Deep Q-Network (DQN) from a demonstration first speeds up the training by at least 83 million steps (number of steps it took a regular DQN to catch up). Although we do not plan on using a DQN, we experiment with this strategy in the PPO approach. Juliani et al.[7] explored a very similar concept in their work and called it imitation learning support for self-play. Although Juliani et al. didn't extensively study the impact of incorporating hu-

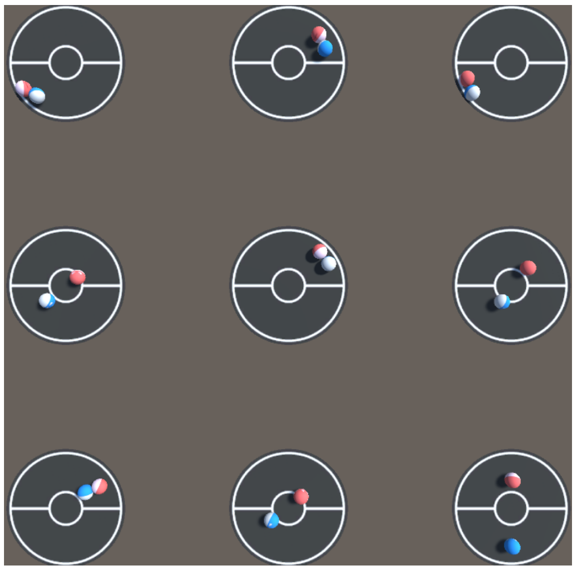
Figure 3. Illustration of our parallel training setup

man demonstrations in training, the tools they provide can be customized for this purpose.

Ho et al.[5] introduced the GAIL framework as a novel approach to imitation learning in reinforcement learning tasks. GAIL addresses the limitations of traditional behavior cloning methods by formulating imitation learning as an adversarial game between a discriminator and a generator. The generator aims to produce actions that mimic expert behavior, while the discriminator distinguishes between expert and generated actions. By leveraging the adversarial training process, GAIL enables the agent to learn complex behaviors from limited expert demonstrations while incorporating exploration. This framework provides an alternative to tediously handcrafted reward functions, allowing for more robust and flexible imitation learning in reinforcement learning settings.

2.2. Dataset

As this is a reinforcement learning problem, data is generated from interactions with the simulated environment. More specifically, as the competing agents take actions in the simulated world, this results in states and we use the action-state pairs as training inputs for our model. See Section 3.1 for more details on our observation/state vectors.

3. Technical Approach

3.1. Baseline

First, we train our baseline with a roller-ball agent (Voltorb) which can apply force in two directions to self-compete in a sumo environment. A visual of the setup can be seen in Fig.1. The ideas and strategies used in our baseline are from the work by Bansal et al.[2] adapted for the Unity[7] environment. We use self-play by sampling previous policy iterations to be opponents since Bansal et al.[2] showed that one agent will overpower the other if always using the latest policy when training both. We use a distributed large-scale setup for faster training, see Fig.3. We update agent policy with PPO [10], and a sparse reward function with win, lose, and stalemate states defined in Eq.2 below.

$$R(s) = \begin{cases} 1 \text{ , if } s = \text{won} \\ -1\text{, if } s = \text{lost} \\ -1\text{, if } s = \text{stalemate} \end{cases} \tag{2}$$

The observation/state vector used for training is defined in Eq.3 below.

$$S = \begin{bmatrix} t_{\text{remaining}} \\ D_{\text{self2edge}} \\ D_{\text{self2opponent}} \\ V_{\text{self}} \\ V_{\text{opponent}} \end{bmatrix} \tag{3}$$

where $t_{\text{remaining}}$ is the time remaining in the episode (max is $20s$), $D_{\text{self2edge}}$ is a 2D vector representing the radial displacement between the agent and the edge, $D_{\text{self2opponent}}$ is a 2D vector representing the displacement of the opponent from the agent, V_{self} is the 2D vector of velocities of the agent, and V_{opponent} is the 2D vector of velocities of the opponent.

3.2. Methods

To extend our baseline, we add Imitation Learning[9] components as part of the RL training. We experiment with adding GAIL, as described by Ho et al.[5], at different strength levels. The strength level is a factor that determines how much the reward function from the GAIL model impacts the overall reward of an episode. We also experiment with Behavioral Cloning [8] combined with GAIL since Juliani et al.[7] showed that this combination worked well in specific RL environments.

To train the imitation learning components of our project we modify our agents to make them human-controllable. We each control different agents and record the actions and observations at each time step as we compete in the Sumo environment. For the Voltorb agent we use 15 human demonstrations.

We train with more complex agents such as a crawler to compete against agents with the same morphology. An example of a 6-jointed crawler agent from The Models Resource[1] can be seen in Fig. 4. We modify the rigged model from The Models Resources by adding physics rigidbody components and capsule colliders on the main limb segments of the rig as well as defining joint behavior and controls.

We use a custom curriculum training method for the crawler agent. We train a locomotion model by adding a goal to the environment and setting the win in the reward function to be reaching the goal within the time limit. By training the agent to move towards goals in random directions the crawler learns how to move in the 8 cardinal directions. We use this trained model as a locomotion model so that the Sumo competition model only has to output a direction (up, right ..etc). This simplifies the Sumo competition model training since the agent does not need to learn locomotion and how to compete at once.

We evaluate the ELO[3] score during training using the self-play[11] component. Assuming we have two players, A and B, with initial scores of 1200, named R_A and R_B, the the score for player A is updated as in Eq. 4 and Eq. 5. The score for player B is updated similarly just by swapping the A and B in the formula.

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \tag{4}$$

$$R_A := R_A + K_A(S_A - E_A) \tag{5}$$

where K is a mastery factor whereas the most recent version of the model (higher skill agent) receives a lower factor and S is the score of the agent from the episode (1 for winning and -1 otherwise).

3.3. Novelty

To clarify novel methods, we itemized our unique contributions below:

- Use PPO and GAIL for competitive RL in Unity: Bansal et al.[2] found that PPO+LSTM worked best in many training scenarios and Hester et al.[4] found that human demonstrations (similar to imitation learning set up) can dramatically speed up RL training. We haven't seen any paper study the PPO+GAIL combination in competitive RL. Juliani et al[7] mentioned this concept in the context of reaching a goal (very simple environment/goal and no competitor) but no quantitative study has been done.

- Use Pokemon X/Y models[1] for training in a custom Unity environment, see Fig1. These are 3D rigged models with different morphologies and body types. These agents are more complex and more interesting than the models used by our reference papers [7][2]. Many of the papers in this field use the same OpenAI MuJoCo[2] models, so we wanted to introduce new models.

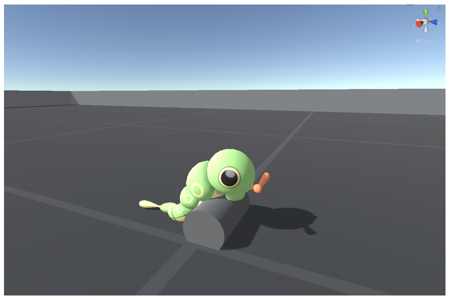
Figure 4. Example of a 6-jointed crawler agent aka Caterpie

4. Results and Evaluation

4.1. Results

We train our Voltorb agents for the Sumo task using all the methods mentioned in Section 3.2. We report our results for 4 training setups:

- Baseline: This method uses a 2-layer fully connected network, each layer of size 128 and PPO training. The baseline does not include any imitation learning.

- GAIL Strength 0.01: In addition to the baseline, this method uses a low-strength GAIL method.

- GAIL Strength 0.1: In addition to the baseline, this method uses a high-strength GAIL method reported for analysis purposes.

- GAIL with BC: In addition to the baseline, this method uses a low-strength GAIL combined with Behavioral Cloning reported for analysis purposes. Behavioral Cloning is only used for the first 10^5 alternatively with regular training.

To quantitatively evaluate our agent's performance and our model's training we plot the ELO scores in Fig. 5, the cumulative rewards in Fig. 6, the episode length in Fig. 7, and the GAIL model loss in Fig. 8. The muted lines in our plots refer to the the original non-smoothed data. Refer to Section 4.2 for a detailed analysis of these results.

Qualitatively, we observed many interesting emergent behaviors for such a simple agent and reward function. The first apparent strategy seen was for agents to ram into each other after building up momentum to push the other off the platform with more force. This resulted in agents spiraling around each other to be the first to push the other off the platform with more force. Another interesting behavior that occurred was one agent tricking the other. The agent would stay still at the edge of the platform, and quickly move out of the way as the other agent attempted to collide.

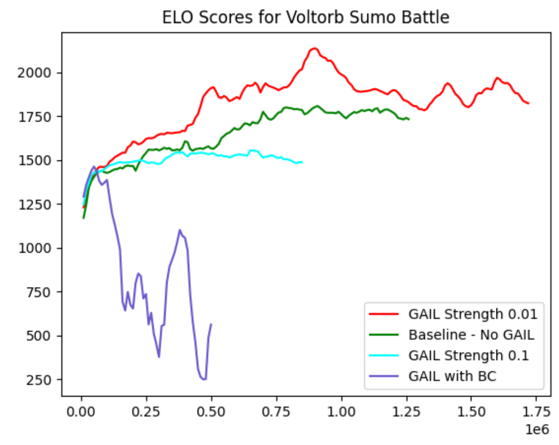

Figure 5. ELO Score per Iteration for Voltorb Agents

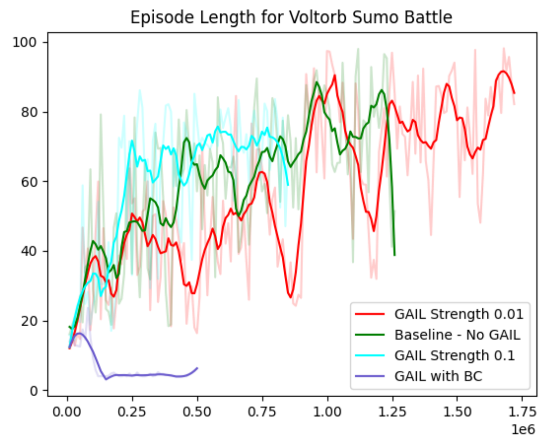

Figure 7. Episode Length per Iteration for Voltorb Agents

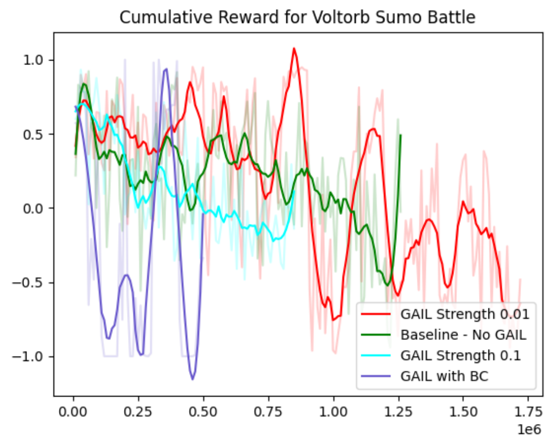

Figure 6. Cumulative Reward per Iteration for Voltorb Agents

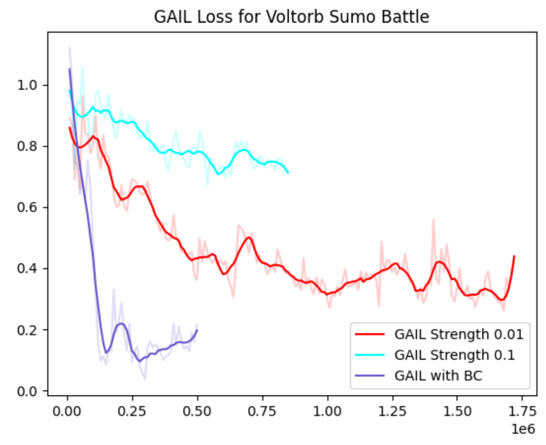

Figure 8. GAIL Loss per Iteration for Voltorb Agents

4.2. Analysis

4.2.1 ELO Scores

In our method, we initialize the ELO scores to 1200 and update the ELO score using our self-play pipeline with Eq. 4 and Eq. 5. We plot the ELO score per iteration in Fig. 5. We observe the best ELO scores trend with the low strength GAIL. This performs better than the baseline because training the GAIL model with human demonstrations helps the RL model to understand the task at hand and learn faster. After learning how to compete quickly, the agents have more training time to develop emergent behaviors and unique strategies.

The high-strength GAIL does not perform as well because we only provide 15 demonstrations and the human demonstrations are not perfect. Similarly, when adding behavioral cloning, which forces the model to follow the demonstrations even closer, the performance is much worse. While behavioral cloning and high-strength GAIL worked well for Juliani et al.[7] in environments such as driving a car on a race track in a video game, the competitive aspect of our setup adds a level of complexity that doesn't allow aggressive imitation learning methods to work well. The position and velocity of the agent, as well as the position and velocity of the opponent, can vary greatly, and high-strength imitation learning results in poor performance because it relies too much on the provided demonstrations instead of also learning from the current situation.

4.2.2 Cumulative Rewards

We use Eq. 2 to calculate the sparse reward at the end of each episode. We plot the cumulative rewards in Fig. 6. We see that the low-strength GAIL and the baseline both have a trend of converging towards -1 as the training continues. That is a good sign because an average reward of -1 means that the competing agents stall more often meaning they learn how to play the Sumo game and not lose.

The GAIL with BC approach oscillates around 0 and when observing the agents' behavior they just jump off the platform as soon as the episode starts and there is no visible behavior that the agents are learning a strategy.

4.2.3 Episode Length

The growing trend of episode lengths goes hand in hand with the conclusions from the cumulative rewards. We plot the episode length in Fig. 7. When the episodes last longer, this means that the agents learn how to fight and not lose thus resulting in more draws.

4.2.4 GAIL Model Loss

We observe from the plotted GAIL model loss in Fig. 8 that the Behavioral Cloning approach results in the lowest loss. This may seem like better training but it results in overfitting and bad performance as can be observed in the plotted episode lengths. Surprisingly, the high-strength GAIL did not result in an overfitting trend but more like underfitting. Low-strength GAIL seems to be the best approach where the model learns the right amount from human demonstrations.

5. Conclusion

From our work on this project, we conclude that imitation learning can greatly aid RL training, especially in competitive RL. While aggressive imitation learning proved to lead to worse performance and overfitting, further testing with better and more demonstrations can be done to determine if a higher-strength imitation learning approach can work well in an RL setup. We conclude however that low strength imitation learning is a good addition to traditional RL setups for faster training.

6. Future Work

We would like to experiment with dense rewards to create a more organic curriculum training as in the work done by Bansal et al.[2]. This can lead to better and faster training for more complex agents such as our Caterpie crawler. We would also like to add more agents to this setup by creating more custom rigs from the open-source 3D models database[1].

7. Contributions

Cole and Rita worked on all parts of this project together at the same time.

References

[1] The models resource - pokemon x/y. Available at `https://www.models-resource.com/3ds/pokemonxy/`.

[2] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. Emergent complexity via multi-agent competition, 2018.

[3] A. E. Elo. The proposed uscf rating system, its development, theory, and applications. *Chess Life*, XXII(8):242–247, August 1967.

[4] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep q-learning from demonstrations, 2017.

[5] J. Ho and S. Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

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

[7] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A general platform for intelligent agents, 2020.

[8] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[9] S. Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems*, volume 9, pages 1040–1046, 1997.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[11] 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.

[12] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.