The Understudy Approach

A Multi-Agent Reinforcement Learning Technique enabling Agents to perform complex tasks involving cooperation

Matt Russiello
University of Massachusetts
Amherst
mrussiello@umass.edu

Tejasvi Ravi
University of Massachusetts
Amherst
travi@umass.edu

Abstract

During the course of this project we introduced a new technique in reinforcement learning called the Understudy approach. This method is inspired by other techniques such as Curriculum and Imitation learning. It aims to train agents to learn complex tasks by combining the models of agents trained on simpler objectives. Specifically we explored using the Understudy approach to encourage collaboration between two agents whose objective was to collect multiple coins in a room as efficiently as possible. In this scenario only one of the agents is given the location of the coins. It must not only move to collect the coins itself but it must also send commands to another agent revealing the locations of the coins.

1. Introduction

Applying Reinforcement Learning [9] to games in order to create intelligent bots that can learn and perform better than established human players is on the rise. These bots are able to learn the rules of a game and develop winning strategies. In order to learn these strategies, the bots will play a video game thousands if not millions of times. Each move they make can trigger positive and/or negative rewards that provide feedback as to whether or not each move is beneficial. This process is very similar to the way our brains work when learning how to do a new task. We perform certain actions with the hope of certain rewards, be it short term or long term (rewards not immediately achieved but can be expected in the future). There have been many such intelligent bots trained in this fashion, perhaps the most exceptional being AlphaGO, Google's Deep Mind bot which can beat expert human players in the game of Go [7].
In games involving teamwork, co-operation is essential to developing useful strategies [8]. There are methods [4] which consider action policies of other agents and are able to successfully learn policies that require complex multi-agent coordination. Cooperation between agents is difficult because it requires agents to perform multiple tasks at once. They must learn how to accomplish the task of winning the game while also identifying how to effectively work with any teammates. These models can be very difficult to train from scratch. This paper focuses on implementing a new method for training complex reinforcement learning models that we introduce as the Understudy approach. This method is inspired by the Curriculum training method which trains agents on complex tasks by slowly increasing the task difficulty through various training iterations.

The Understudy approach is useful for when you would like to train an agent to perform multiple tasks at the same time. The basic idea is that you train agents on each task individually. After training is complete, move both agents to the same environment and have another agent called the Understudy learn from their actions. [1] This technique has the potential to be a valuable tool for engineers to train agents more efficiently on complex tasks.

2. Background

Markov decision processes (MDPs) [5] facilitate a mathematical framework which helps model decision making in situations where outcomes are sometimes random but are also partly influenced by the decisions made by the decision maker. Therefore MDPs come in handy when it comes to solving the optimization problems found in Reinforcement Learning.

Concretely, Markov Decision Processes try to find a policy π which specifies the action a, also given by $\pi(s)$, that needs to be taken by the decision maker while in the current state s. When an MDP is combined with policies in this way for giving rise to set of actions for each state, it forms a Markov chain.

Markov Games [3] allows one to widen the framework of MDP's to include multiple adaptive agents with interacting

[1] Code is available here at https://github.com/Yarlak/682$_{U}nderstudy$

or competing goals.

Selection of the policies in Markov games can be done via several techniques, but policy gradient methods stand out as they have better convergence properties, are effective in continuous action spaces and can learn stochastic policies.

However, they are highly sensitive to the choice of stepsizes and are easily overwhelmed by any noise while training. However the Proximal Policy Optimization [6] technique strikes a balance between sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

PPO has the following objective function:

$$L^{CLIP}(\theta) = \hat{E}[min(r(\theta)\hat{A}_t, clip(r_t(\theta), 1\epsilon, 1+\epsilon)\hat{A}_t)]$$

Where θ denotes the policy parameter, $\hat{E}$ denotes the Emperical Expectation over time steps, r_t is the ratio of the probability under the old and new policies, $\hat{A}_t$ is the estimated Advantage at time t, ϵ is usually a hyperparameter.

The essence of human intelligence is the understanding of the concept of co-operation for the greater long-term benefits. It's also a heavily researched topic in Multi-agent reinforcement learning. Co-operation requires communication of any kind between the agents, which is well explored in [4]. The paper also empirically shows that learning is faster when it is shared [2] and also how curriculum learning helps to train MARL systems faster.

We use these concepts and describe our approach below.

3. Methodology

Using reinforcement learning to train agents to accomplish difficult and complex tasks can be challenging to say the least. This is because agents that start training from scratch need to attempt the task a very large number of times before they can start to see a recognizable pattern between the observations from their environment and the correct actions to perform. If a task is very complex then it will take the agent a very large amount of time before such a pattern can be recognized. If there is an error in the reward function being used then it is possible the agent will never learn to accomplish the task. Training agents for difficult tasks can be made easier by methods like Curriculum learning which gradually increase the difficulty of the agents task by making changes to the environment during training. However, this method is somewhat brittle as it does not allow much variation in the number of observations or actions made by the agent throughout the course of training. Another method known as Imitation learning uses data collected from human players playing the game to train the agents. However, this can be very time consuming as complex games could require a large amount of training data requiring the human player to play the game for a very long time.

We propose a different method called the Understudy approach which is inspired by both Curriculum and Imitation learning. It can be used to gradually increase the difficulty and/or complexity of a task while allowing for changes in the observations and actions of the agent. This method also generates training data for the agent similar to imitation learning but does not require a human player to spend countless hours playing the game. The basic idea is to break up the difficult task into smaller less complex tasks and train agents to solve them (these agents will be called leads). All the trained leads will then be placed in the same environment with another agent called the understudy. The understudy will be trained by making the same observations as the leads and its reward function will solely be based upon how similar its actions are to those of the leads. This is similar to imitation learning with the exception that trained agents (the leads) are generating the training data for the understudy instead of human players. The goal is to train the understudy to act like the leads. If there are multiple leads then the understudy may be able to learn how to perform multiple tasks at the same time.

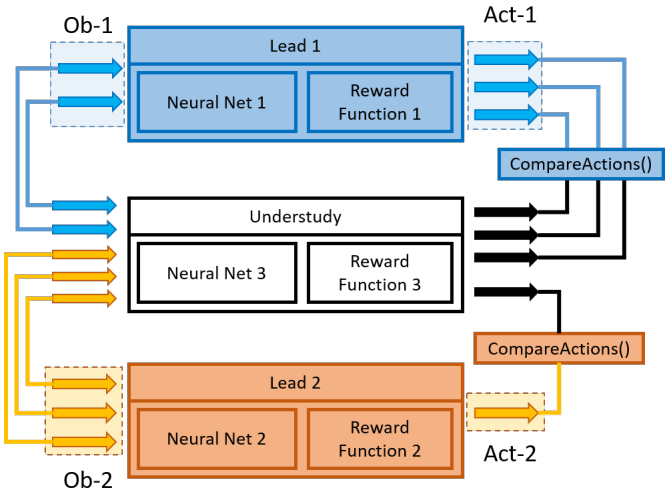

Figure 1. Outline for training an understudy using two Lead agents. Please note that both Leads 1 and 2 are fully trained before being used to train the understudy. The understudy's reward function is solely based on how closely its outputs resemble that of the Leads.

For example, lets consider the scenario depicted in Figure 1 in which we would like to train an agent to perform two tasks (Task A and Task B) at the same time. Using the Understudy approach we could train a different lead on each of the tasks (Lead 1 for Task A and Lead 2 for Task B). The observation and action spaces of the leads are Ob-1 and Ob-2 and Act-1 and Act-2 respectively. After successfully training on their tasks both leads are placed in the same environment with a new untrained agent (the understudy). The observation space of the understudy contains

most if not all of the observations in Ob-1 and Ob-2. The action space of the understudy contains all actions in Act-1 and Act-2. During training, the understudy will not interact with the environment. Its training will simply consist of trying to replicate the actions of both leads given the current environment. Its reward function will only consist of how closely its actions resemble those of the leads. In this manner, we may be able to train an understudy that can accomplish both Task A and Task B at the same time.

4. Experimentation

Our goal is to train two agents to work together to collect two coins randomly generated in an open room as quickly as possible. One agent takes the role of captain and the other the servant. Both the captain and the servant are able to move around the room and collect coins. However only the captain is shown the location of the coins and it is responsible for sending movement instructions to the servant.

4.1. Tools

All experiments for this project were completed using the Unity ML Agents package[1] created for the Unity 3D game engine. This package allows users to train reinforcement learning models with tensorflow and Proximal Policy Optimization (PPO). The code to train these models was included in the package and provided for us. We built the environments described below by modifying a demo environment provided as part of a tutorial from the Unity ML Agents package. The bulk of the programming performed as part of this project was focused on coming up with the best reward functions to accomplish the listed tasks and creating multiple environments that could be used for training and testing the various models and techniques.

4.2. Training from Scratch

We attempted to train both the servant and the captain at the same time in the same environment. The observations of the servant:

- Commands from captain
- Distance to all walls of the room

The action space of the servant:

- Movement in the x-direction
- Movement in the z-direction

The observations made by the captain:

- Relative position of the servant
- Status of the coins
- Relative positions of the coins
- Distance to all walls of the room

The action space of the captain:

- Movement in the x-direction
- Movement in the y-direction
- Commands to send to the servant

When one of the coins is picked up it will become ”inactive” meaning that it will no longer give points to any agent that touches it. For the rest of the paper we will refer to whether or not a coin is ”inactive” as the status of the coin. The reward function of the both agents is defined below:

- Existential reward of -1 at every time step to each agent
- Distance reward of -1 * distance to closest coin to agent
- Reward of +300 applied to both servant and captain if either picks up a coin
- Captain also gets the Distance Reward of the servant

The reward function for lead-1 is defined below:

- Existential reward of -1 at every time step to each agent
- Distance reward of -1 * distance to closest coin to agent
- Reward of +300 applied to both servant and captain if either picks up a coin

Even after training for 19 million time steps the models generated for both the servant and the captain were not able to consistently pick up the coins. This is most likely due to the overall system complexity. By training both agents from scratch we are forcing the following actions to be learned at the same time:

- Servant interprets commands sent by the captain
- Servant moves in response to captain’s commands
- Captain controls the servant with commands
- Captain chooses the best path for itself

What makes this system so complex is that some items listed above are dependent on others. For instance, the ability of the servant to interpret commands sent by the captain and the captain learning to control the servant are dependent on one another. Either the format of the commands or the way the servant responds needs to be consistent in order for any learning to occur. But because the system is trained from scratch neither are initially consistent. In order to simplify the problem we decided to train two leads to accomplish the following tasks:

- Lead 1 - move to collect a single coin given the coin's relative position
- Lead 2 - control two trained Lead 1's in order to collect two coins in a room

After both leads have been trained the understudy will observe the environment in which Lead 2 is controlling two Lead 1s. The reward function of the understudy will solely be based upon how closely its actions are to those of the trained leads.

4.3. Training Lead 1

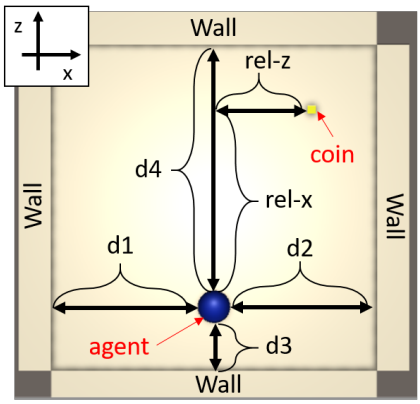

Figure 2. Environment and relevant observations for the Lead 1 agent trained by itself.

The environment used to train Lead 1 is shown in Figure 2 in which the agent makes the following observations:

- Relative position of the coin (rel-x and rel-z)
- Distances to all edges of the room (d1, d2, d3 d4)

The Lead 1 agent has two actions that it can perform during each time step:

- Movement in the x-direction
- Movement in the z-direction

Lead 1 was trained for $5.0e^5$ time steps using a fully connected neural network with 3 hidden layers and 64 hidden units at each layer.

4.4. Training Lead 2

The environment used to train Lead 2 is shown in Figure 3. Lead 2 does not move around the environment to collect coins. Instead, it sends messages to two fully trained Lead 1 agents which will act as servants. Instead of receiving the relative position of a coin, each Lead 1 agent will receive two numbers from the Lead 2 agent. Unlike when we tried

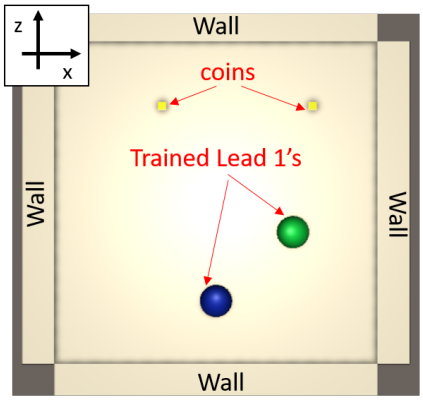

Figure 3. Environment used to train a Lead 2 agent. The Lead 2 agent is not visible as it does not collect coins itself (it commands the two Lead 1's to collect coins).

to train the system from scratch, now the way the recipients of commands will respond consistently because they have already been trained. This will make it easier for the agent sending the commands (in this case the Lead 2 agent) to learn what commands are effective in different situations.

The observations made by the Lead 2 agent are:

- Relative position of both Lead 1 agents
- Status of both coins (able to be picked up)
- Relative positions of both coins

The actions taken by Lead 2:

- Send two floating point numbers to each Lead 1

The Lead 2 agent was trained for $2.0e^6$ time steps using a fully connected neural network with 3 hidden layers and 64 hidden units per hidden layer.

The reward function for the Lead 2 agent consists of the following:

- Existential reward at every time step of -1
- Distance rewards for both Lead 1 agents
- Reward of +300 when either Lead 1 picks up a coin

4.5. Understudy Training

4.5.1 Understudy 1

We decided to first train an understudy on a single trained Lead 1 agent (environment similar to that pictured in Figure 2, but with an understudy agent) and compare its performance with that of the trained Lead 1 as a proof of concept. This understudy took in the following observations:

- Position of the coin relative to the Lead 1 agent

- Distances of the Lead 1 agent to the walls

After taking in the observations listed above the understudy calculates the following two actions:

- Movement in the x-direction
- Movement in the z-direction

The reward for the understudy at each time step is calculated using the following equation:

$$R = -1 * ((LA_1 - UA_1)^2 + (LA_2 - UA_2)^2) \quad (1)$$

Where UA_1 and UA_2 are the actions of the understudy and LA_1 and LA_2 are the target actions of Lead 1. Because the total reward is always negative, optimizing the reward from Equation 1 would mean minimizing the difference between the actions of Lead 1 and the actions of the understudy.

The understudy of Lead 1 was trained for $2.0e^6$ time steps with a neural network with 3 hidden layers and 64 hidden units per layer.

4.5.2 Understudy 2

In order to train an understudy as a captain, we insert an understudy into the environment where a Lead 2 controls two Lead 1's (similar Figure 3). However when the understudy is placed in the environment both the Lead 2 and Lead 1 agents are fully trained. The goal is to train the understudy to both send commands like the Lead 2 agent and be able to navigate to coins like the Lead 1 agents. Just like the proof of concept described above the understudy will not be interacting with the environment. It will merely be taking in observations and comparing its actions to the actions of the fully trained Leads. The understudy will essentially pick one of the Lead 1's (we will call Lead 1A) and learn how it moves around the map based upon the locations of the coins and the other Lead 1 (referred to as Lead 1B). The understudy takes the following observations from the environment:

- Location of Lead 1B relative to Lead 1A
- Status of the targets
- Location of the targets relative to Lead 1A
- Distance of Lead 1A to all of the walls

However, unlike a Lead 1 agent, the understudy will have four items in its action space:

- Movement in the x-direction
- Movement in the z-direction
- Two float point numbers sent to Lead 1B

The total reward the understudy receives will solely be based on how closely the understudy's actions resemble that of the Lead 1A and Lead 2 agents. The total reward is thus represented by the following equation:

$$R_{total} = R_{L1_A} + R_{L2} \quad (2)$$

Where R_{L1_A} and R_{L2} are the rewards calculated by comparing the understudy's actions to their counterparts in the L1A and L2 action spaces respectively and be calculated using Equation 1. The understudy was trained for $1.2e^7$ time steps using a neural network with 3 hidden layers, 64 hidden units per layer and a learning rate of $5.0e^3$.

5. Results

5.1. Scored Runs

In order to test the functionality of the models, we developed testing versions of the environments used to train the Lead 1 and Lead 2 agents. These environments are labeled Test 1 and Test 2. Each environment will spawn a set of coins whose locations have been pre-determined to ensure fairness and consistency in scoring the models. Test 1 spawns only one coin at a time while Test 2 spawns two coins at a time. New coins are only spawned after all the coins currently in the room have been picked up. Each time a new set of coins is spawned in the room a timer is set for 3 minutes. If the agents can't pick up all coins in the room in 3 minutes then the game is over and the model finishes with its current score. After picking up all the coins in the room the model's score is increased by the following equation:

$$Score = Score + T_{rem} \quad (3)$$

Where T_{rem} is the number of seconds left on the reset timer. Below is a table of the scoring results for all the models discussed in the experimentation section where Understudy 1 and 2 are the understudies trained in the Lead 1 and Lead 2 environment respectively.

Model	Environment	Score
Lead 1	Test 1	17,784.20
Understudy 1	Test 1	17,655.16
From Scratch	Test 2	0.00
Lead 2 and two Lead 1's	Test 2	17,771.02
Understudy 2	Test 2	178.02

As you can see from the table above, neither understudy is able to beat the models containing Lead 1 and/or Lead 2 agents. The Understudy 1 model was very close to the Lead 1 score on Test 1 but Understudy 2 was only able to collect 1 set of coins before time expired resulting in a score of

178.02. This is much lower than the score of 17,771.02 obtained by the model containing two Lead 1's and one Lead 2. The model trained from scratch was not able to collect any coins in the Test 2 environment and received a score of 0.

5.1.1 Understudy 1 vs. Lead 1

Figure 4 shows an agent from the Test 1 environment traveling to pick up a coin. Notice how the path created from the Understudy model is much more smooth than that of the Lead 1 path. Qualitatively it would seem that the understudy path should be faster, however, there are several periods during the test that Understudy 1 pauses and takes some time to locate the coin. The time taken by these pauses is ultimately what caused the Understudy 1 model to score less than the Lead 1 model. It is worth noting that training for Understudy 1 was stopped before the reward (calculated in Equation 1) reached zero. This means that Understudy 1 is not able to completely mimic the actions of Lead 1 hence the difference in paths taken to get the same coin and the strange pauses taken by Understudy 1.

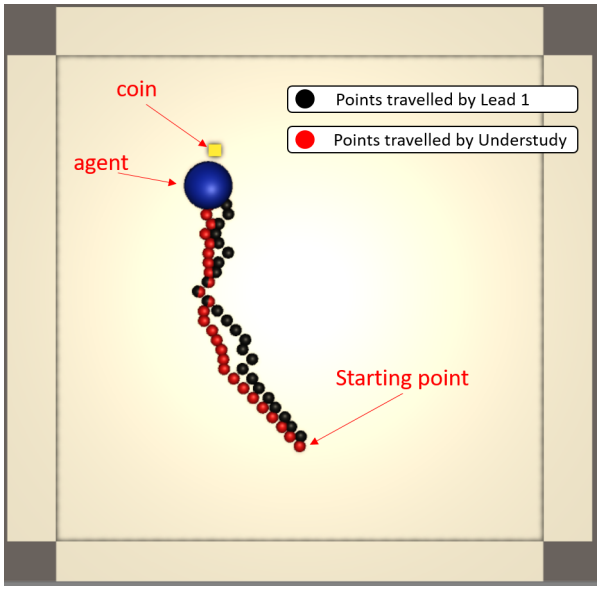

Figure 4. Paths generated by agents using the Lead 1 and Understudy 1 models.

5.2. Conclusion

As shown in the Results section, we were able to train an understudy to mimic the functionality of the Lead 1 Agent. However, this replication was not perfect which resulted in a lower score for Understudy 1 when running in the Test 1 environment compared to Lead 1's score. Although Understudy 2's score on the Test 2 environment was much lower than that of Lead 2 two Lead 1's, it still provided a noticeable improvement to training from scratch.

In the end, we were not able to effectively create a captain-servant model using the Understudy approach leaving much future work to consider. It is possible that a 3-layer fully connected network with 64 hidden units does not have large enough capacity to learn the model required by the captain. Future experiments could fine-tune the hyper-parameters of the network so as the achieve optimal performance. Another area of analysis that could benefit this project is to look into the reward functions used by the understudies. Perhaps simply taking the sum of the squared differences between the actions of the understudy and its leads is the best solution. Additionally, more training time analysis is required with respect to scaling of a number of tasks and/or number of agents to learn from. With improvements to he understudy approach, we believe that this technique can be generalized to multiple applications, where the trained agent can teach an understudy how to perform a combination of tasks and/or skills.

6. Appendix

Hyper-parameters used for experimentation in the coach model are given below.

Hyper-parameters	Lead 1	Lead 2
Train Iterations	5.0e5	2.0e6
Minibatch Size	1024	1024
Learning Rate	3e-4	3e-4

Hyper-parameters	Understudy 1	Understudy 2
Train Iterations	2.0e6	1.2e07
Minibatch Size	1024	1024
Learning Rate	5.0e-3	5.0e-3

All of the above models were tested with 64 hidden units and 3 layers.

Hyper-parameters used for experimentation in the model without the understudy are given below.

Hyper-parameters	Captain	Servant
Train Iterations	19.0e6	19.0e6
Minibatch Size	1024	1024
Learning Rate	3.0e-4	3.0e-4
Hidden Units	64	64
Layers	3	3
Epochs	3	3
Epsilon	0.2	0.2

References

[1] Unity ml agent environment. `https://github.com/Unity-Technologies/ml-agents/tree/master`.

[2] D. Garant, B. C. da Silva, V. Lesser, and C. Zhang. Accelerating Multi-agent Reinforcement Learning with Dynamic Co-learning. Technical report, 2015.

[3] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, pages 157–163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[5] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 10 2017.

[8] P. Stone, R. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. 13, 12 2005.

[9] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.