Fall 2018 CS7180 Final Project Report

AAAI Press
Peter Bernstein and Giorgio Severi

Abstract

AAAI creates proceedings, working notes, and technical reports directly from electronic source furnished by the authors. To ensure that all papers in the publication have a uniform appearance, authors must adhere to the following instructions.

Introduction

Reinforcement learning has been widely used to successfully solve a variety of games. Doom is a classic First-Person Shooter (FPS) game from 1993 created by Id Software. As one of the first FPS games to market, it is simple in its gameplay in comparison to contemporary videogames, but can still be significantly more complex than many other Atari games, such as Pong. Doom has the following components: a player is a marine in the central point of an invasion of demons. The agent can be placed in a variety of scenarios with a multitude of differently behaving demons. At any time the agent can navigate or shoot his weapon. The game state may not be completely visible at all times and obstacles often partially obscure the display. Most importantly for this project, the complexity of the game can be controlled.

The above features make Doom an interesting study for reinforcement learning because of the variety of settings with varying difficulties. Training an AI is possible in simple or complex settings variety of architectures. The focus of this project is only restricted to a simple game state because of the power of the machines used and the time we had to run them. The parameters in this project can easily be changed to train our models on a complex Doom environment.

Background

Related Work

Reinforcement learning and deep reinforcement learning in particular have a history of success on fully observable games for Atari, such as Pong or Space Invaders [al13]. Most of the best performing AI solvers were trained using deep recurrent Q-learning, weighted replay, or dueling. Google DeepMind recently developed an Asynchronous Advantage Actor-Critic (A3C) model that has also been used to train AI solvers for Atari games with success as well. The victory of AlphaGo over the best human player received wide media attention as a landmark. While we restrict our focus in this project to deep-Q learning and SARSA, more complicated models have proven to be more successful especially running on more advanced architectures.

Doom in particular has a history with using deep-Q networks to train solvers. Recent works on training the Doom AI have created models that outperform built-in bots and even some human players

Project Description

We chose a very simple setting of the Doom game to train our agent. The setting for our project AI allows for the agent at any time step to do one of only three things: move left, right, or shoot. For each game episode, there is a stationary demon that does not attack the agent. Thus the goal for the

Experiments

Sarsa (on-policy TD control)

To test the Sarsa algorithm on Doom, we needed to first tune the parameters of the experiment. Unfortunately it was determined that for significant learning to take place, large numbers of episodes (20,000) needed to be performed. Consider the example in figures and of how Sarsa performed over 3000 trials for values of the learning rate alpha set to 0.1 and 0.2. It is clear that learning hardly takes place if at all for either learning rate.

We decided to run the AI for a much larger number of episodes to see if we could witness learning taking place. We chose to do 120,000 episodes, where the $\epsilon-$greedy policy would anneal from $\epsilon = 1$, i.e. a completely random policy, to $\epsilon = 0$, i.e. a greedy policy over the first 100,000 episodes. The last 20,000 episodes show the reward when the agent greedily followed the learned policy. Unfortunately the average reward is still negative, indicating that the policy learned still had much room for improvement. Nonetheless, it is still evident that average reward did increase and then level off when the policy became completely greedy. Thus it is reasonable to infer that with much more training, the agent could learn a successful policy, albeit slowly.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 1: High level pseudocode description of sarsa algorithm for estimating Q

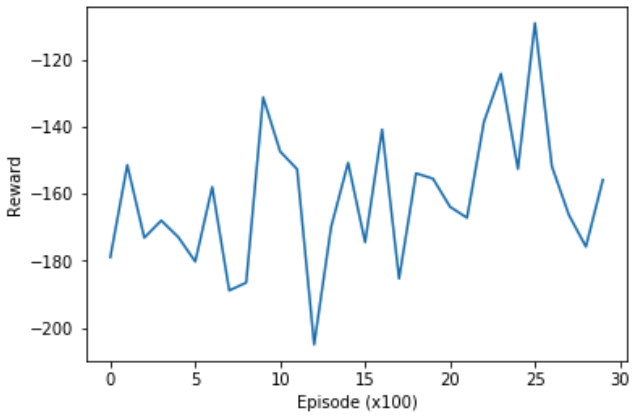

Figure 2: Running 120000 episodes of Sarsa with a smoothing window of 100 episodes. Learning rate was set to 0.1, ϵ-greedy policy selection was used with annealing steps from 1 to 0 over 2500 episodes and the last 500 episodes follow a completely greedy policy

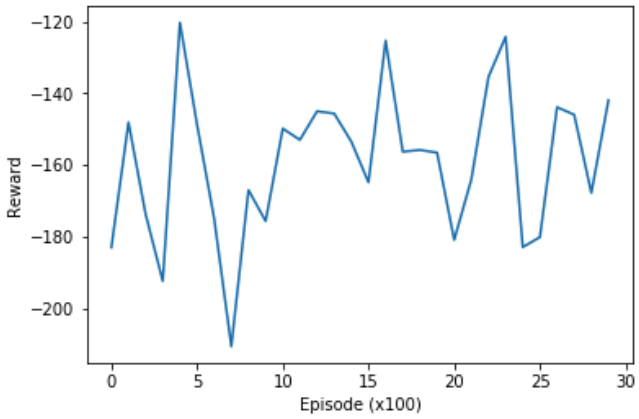

Figure 3: Running 3000 episodes of Sarsa with a smoothing window of 100 episodes. Learning rate was set to 0.2, ϵ-greedy policy selection was used with annealing steps from 1 to 0 over 2500 episodes and the last 500 episodes follow a completely greedy policy

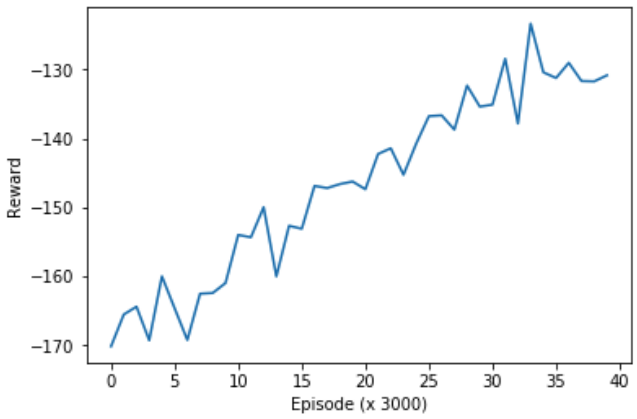

Figure 4: Running 120000 episodes of Sarsa with a smoothing window of 1000 episodes. Learning rate was set to 0.1, ϵ-greedy policy selection was used with annealing steps from 1 to 0 over 100000 episodes and the last 20000 episodes follow a completely greedy policy

Deep-Q Learning

Conclusion

References

[al13] Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *ArXiv preprint arxiv: 1312.5602* (2013).