Playing Startcraft II with Reinforcement Learning

Team member(s): Haiyuan Mei (hmei0411@stanford.edu)

Progress

1. Debian+tensorflow+GPU environment on both google cloud and my own PC ready.
2. StarCraft environment running successful, training with existing A2C+FullConv on FindAndDefeatZerglings and MoveToBeacon works as expected (training on DefeatRoaches seems to have diminishing/exploding gradient issue. Beyond scope, look into it later).
3. Started working on and A3C, studying paper[2]

Abstract

Real Time Strategy (RTS) games such as StarCraft 2 has long been a major challenge in AI research. The study of AI in SC2 also opens a door to unparalleled opportunities to many challenging frontiers, both in AI research and in real life. This paper explores and replicates DeepMind's work of SC2LE (StarCraft II: A New Challenge for Reinforcement Learning) [1]. In this paper I will firstly introduces the model for the game environment, including specifications of observations, actions and rewards; then look into the details of an important RL policy gradient algorithm: Asynchronous Advanced Actor-Critic(A3C); and a fully connected convolutional neural network agent(CNN) which is used by A3C in making decisions for the next move. Lastly I would like to also explore more on deep q learning and make some comparisons between the policy based algorithm(A3C) and the value based algorithm (DQN). In order to train deep reinforcement learning models, TensorFlow is used for both A3C and DQN.

Motivation

Blizzard's Startcraft series game is one of the most challenging Real Time Strategy game, and has gained immense commercial and cultural success over the past 20 years. However, unlike its popularity, the study of AI in solving full SC games progresses somewhat slow and seems relatively far from finished, this is probably because of the challenge imposed by multi-agent interacting, imperfect information, large state/action space and delayed credit assignment requiring long-term strategies over thousands of steps. The most recent in this field was about Deepmind's AlphaStar in Jan/2019, which was defeated at the last round by Pro SCII champ Grzegorz "MaNa". Although a huge progress in this field, there are a lot more yet to be solved, one example is the full support of all races, the other example is to tackle the more complicated 2v2, 3v3 or 4v4 games.

In this paper, by using the PySC2 Library by DeepMind, the game states are represented as "images", and a fully connected Convolutional Neural Network (CNN) is used to produce the optimal actions for each given input state. Superimposed upon this network runs a policy gradient Reinforcement Learning algorithm Asynchronous Advanced Actor Critic (A3C), which can be used to learn the optimal policy. The second RL algorithm included in this paper is DQN, which has a fundamental different to A3C in that, the former is a policy based algorithm, learning directly an optimal policy π (s,a), while DQN is a value based algorithm, with the value function Q(s,a) as the goal of the learning.

Models

[1] StarCraft II: A New Challenge for Reinforcement Learning https://arxiv.org/abs/1708.04782

[2] Asynchronous Methods for Deep Reinforcement Learning https://arxiv.org/abs/1602.01783

[3] https://sergioskar.github.io/Actor_critics/

An finite-horizon Markov Decision Process (MDP) is used to model the problem. The MDP is defined as a 5 element tuple (States, Actions, {T(s,a)}, γ, Rewards) where each of the elements are defined as below:

- States

  StarCraft II is a 3D graphics game, the most intuitive model is to model the game as a times series with hundreds or thousands of graphical frames, each of the frame is a state which can be represented as an MxN vector; and exploit the latest deep learning achievement of CNN in image processing to predict the optimal policy based on the current state.

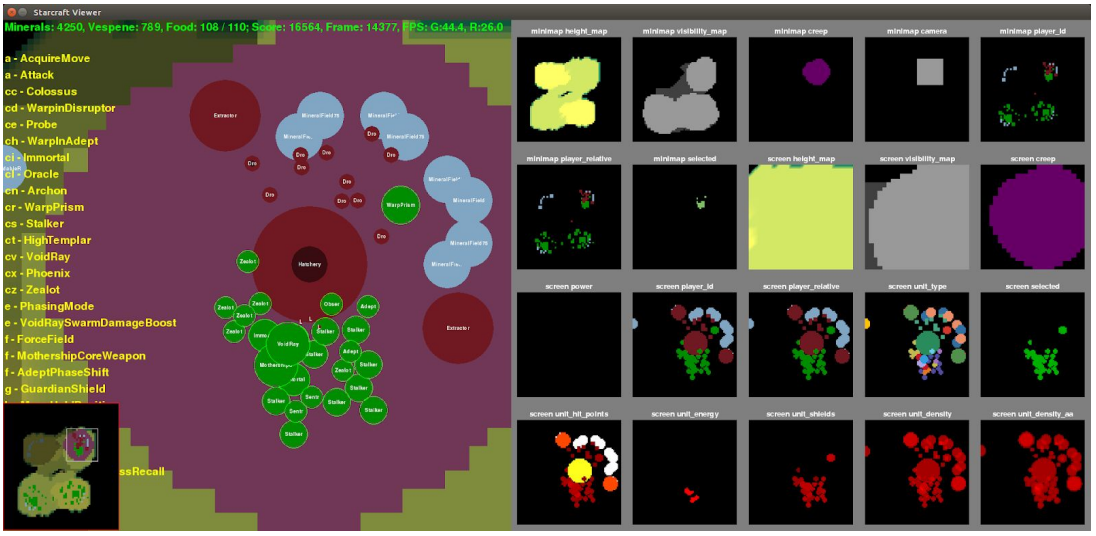

Figure 1. The PySC2 viewer shows a human interpretable view of the game on the left, and coloured versions of the feature layers on the right. For example, terrain height, fog-of-war, creep, camera location, and player identity, are shown in the top row of feature layers. A video can be found at https://youtu.be/-fKUyT14G-8.

In this paper, an abstracted RGB image seen during human play which is called “feature layers”, which maintains the core spatial and graphical concepts of SC2 (screen and minimap):

1. Spatial observations. As shown in the right of the above figure, there are 20 feature layers, each represents something specific in the game. For example, terrain height, fog-of-war, creep, camera location, and player identity, are shown in the top row of feature layers.

2. Non-spatial observations. On the left side of the above image is the human interface, various non-spatial information such as gas and minerals collected, set of actions currently available, etc, are shown in the human interface.

- Actions
  The environment action space is a close mimic of human interface with keyboard and mouse. Figure 2 shows an example of a sequence of actions.

[1] StarCraft II: A New Challenge for Reinforcement Learning https://arxiv.org/abs/1708.04782

[2] Asynchronous Methods for Deep Reinforcement Learning https://arxiv.org/abs/1602.01783

[3] https://sergioskar.github.io/Actor_critics/

An action a is composed of a function ID a0 and a list of arguments for that function: [a1, a2, ... aL]. For instance, consider selecting multiple units by drawing a rectangle. The action can be written as select_rect(select_add, (x1, y1), (x2, y2)). The first argument select_add is binary. The other arguments are integers that define coordinates. This action is fed to the environment in the form [select_rect, [[select_add], [x1, y1], [x2, y2]]].

There are approximately 300 action-function identifiers with 13 possible types of arguments (ranging from binary to specifying a point on the discretized 2D screen) defined in SC2LE[1]. See the environment documentation for a more detailed specification and description of the actions available through PySC2.

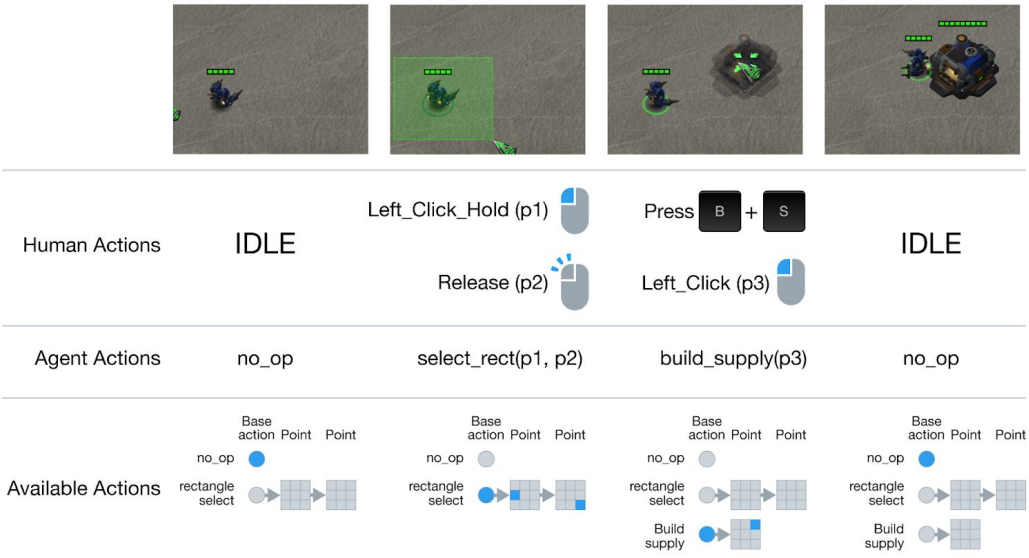

Figure 2: Comparison between how humans act on StarCraft II and the actions exposed by PySC2.

- Rewards
  There are two different rewards structures are described in SC2LE[1]: ternary 1 (win) / 0 (tie) / -1 (loss) received at the end of a game (with all-zero rewards during the game), and the Blizzard score. In this course project only the Blizzard score will be explored. The Blizzard score is what the players see on the victory screen at the end of the game, however via interacting with SC2LE environment, Blizzard scored at every step can be obtained by RL agent which can be used as rewards for the learning. This score is calculated by adding up different components: the current resources, the upgrades researched and units and buildings currently alive and being built. A players cumulative rewards accumulates and decreases as these 3 components accumulates and descreases.

Expriments:

In order to evaluate the project, the baseline will be played by a novice SC2 player and should be defeated by my agent; I myself as an intermediate SC2 player will be the oracle. I will initially pick 2 of the mini games in the project:

- MoveToBeacon: The agent has a single marine that gets +1 each time it reaches a beacon.

[1] StarCraft II: A New Challenge for Reinforcement Learning https://arxiv.org/abs/1708.04782

[2] Asynchronous Methods for Deep Reinforcement Learning https://arxiv.org/abs/1602.01783

[3] https://sergioskar.github.io/Actor_critics/

- FindAndDefeatZerglings: The agent starts with 3 marines and must explore a map to find and defeat individual Zerglings. This requires moving the camera and efficient exploration.

In order to work with video frames, CNN is an important part in this project. The reinforcement learning algorithm will be Asynchronous Advantage Actor Critic (A3C) as described by Mnih et al.[2] Based on how much time left I would also be very interested in applying q-learning and TD learning in this part.

[1] StarCraft II: A New Challenge for Reinforcement Learning https://arxiv.org/abs/1708.04782

[2] Asynchronous Methods for Deep Reinforcement Learning https://arxiv.org/abs/1602.01783

[3] https://sergioskar.github.io/Actor_critics/