**Build Order Optimization using Supervised Learning and Reinforcement Learning**

**Report**

Team: SwaggerBots

**Group Members**

Shraddha Agrawal, [shagrawal@ucdavis.edu](mailto:shagrawal@ucdavis.edu), 913001594

Axel A Hernandez, [aahern@ucdavis.edu](mailto:aahern@ucdavis.edu), 912268257

Aria Chang, [ariang@ucdavis.edu](mailto:ariang@ucdavis.edu), 999792759

Dolly Ye, [baoye@ucdavis.edu](mailto:baoye@ucdavis.edu) ,915511231

Rhys Li, [zjkli@ucdavis.edu](mailto:zjkli@ucdavis.edu) , 999285787

**Problem and Contribution Statement**

No matter how good you are at combatting or economy maximization you cannot win a game without good selection of units, structures and buildings. In Starcraft 2, the problem of production management, which includes how you spend your resources and what to build at what situations, is vital to achieve a better performance, and ultimately the victory of the game. In this paper, we consider the build order problem in RTS games in which we need to find concurrent action sequences that, constrained by unit dependencies and resource availability, to create a certain number of units and structures in the shortest possible time span.

There has been work done in the build order optimization area, documented in the research paper “Learning Macromanagement in StarCraft from Replays using Deep Learning”. The research conducted uses supervised deep learning to train a production agent capable of predicting the next best move. The lingering concern proposed at the end of the paper is whether reinforcement learning would help improve the results achieved by training on the UAlbertaBot, which was the subject of the paper. Our project addresses this concern and implements the build order problem using reinforcement learning on top of the UALbertaBot. We drew our inspiration from a preliminary research conducted in “StarCraft II: A New Challenge for Reinforcement Learning”. In the research conducted by the DeepMind and Blizzard team, they used reinforcement learning and successfully beat the built-in rule-based StarCraft AI on some minigames, but the bot is unable to defeat the built-in AI at full game. Furthermore, it is proposed that “learning from replays should be useful to bootstrap or complement reinforcement learning.” Therefore, our bot will be an upgrade of the UAlbertaBot.

**States**: units/building/tech status of the player at any given time for all feature layers of the map and camera view. This also includes action spaces.

**Actions**: set of all possible actions to take at each time step as well as action requirements for available actions within the feature layers

**Reward functions:** The hardest challenge which the StarCraft AI community faces is a good reward function. In our project, we plan to experiment with a couple of different kinds of reward functions, and compare and contrast the outcomes.

The easiest approach would be using the game score. We hypothesize that the score summary analyzed at the end of a match could provide insight to the game result to a certain degree. Therefore, to solve the problem of production management, we could implement a passive reinforcement learning agent that aims at maximizing the game score by watching the replay of other players. However, there are biases associated with the scoring evaluation and needs to be optimized. The Blizzard game score is computed as the sum of current resources and upgrades researched, as well as units and buildings currently alive and being built. One drawback from using this reward function is that the trained agent might be prone to maximizing production at every time step, and not getting the necessary reward for a combination of some strategic offensive attacking and essential temporary setback. We will experiment with adding a second component of a collection of reward functions, which is to punish the agent for letting resources rise faster than the agent builds buildings. To this end, we will measure the growth of the minerals and gas and calculate whether the rate of resource accumulation exceeds building establishment.

For the purpose of tackling on one goal at a time, we will focus on 1v1 replays. According to our research, We will obtain the input dataset from the replay. Specifically, we will extract the states from the replays, partitioned by each second as our time interval. On average,  The agent takes an, then learn the most effective action to maximize its score at each of the states through reinforcement learning. We plan to experiment with policy gradient, Q learning, and other RL methods.

* **Why is this problem important to solve?**

The build order optimization problem can be described as constraint resource allocation problem with makespan minimization, which features concurrent actions. Because of their practical relevance in RTS games, it is an important problem to solve. Furthermore, the size of the state space in RTS games is much larger than that of traditional board games such as Chess or Go. And, the number of actions that can be executed at a given instant of time is also much larger. Thus, standard adversarial planning approaches, such as game tree search are not directly applicable. Planning in RTS games can be seen as having multiple levels of abstraction and techniques that can address these large planning problems by either sampling, or hierarchical decomposition have not been fully developed yet.

* **Does there already exist a solution to this problem?**

The research paper “Learning Macromanagement in StarCraft from Replays using Deep Learning” has presented a supervised solution where macromanagement tasks are learned directly from replays using deep learning and the learned policy can be used to outperform the built-in bot in StarCraft.

The research paper “StarCraft 2: A New Challenge for Reinforcement Learning” introduces SC2LE (StarCraft 2 Learning Environment), a reinforcement learning environment based on the StarCraft 2 game. They present initial baseline results for canonical deep reinforcement learning agents applied to the StarCraft 2 domain.

* **Why is this problem technically interesting?**

The existing AI solutions are either only able to out perform built-in bots that are usually seen as weak player compared to human players or can play in mini-games environment. Our expectation is to train UAlbertaBot on deep-learning network and implement reinforcement learning so that it defeats the built-in bot almost every time. This will be an interesting problem because firstly it will give us a chance to extend on an existing research paper by implementing reinforcement learning. And secondly, we will experiment with policy gradient, Q learning, and other RL methods, explore SC2LE environment and try out different reward functions to optimize the results.

* **What possible AI approaches are there to this problem?**

From a high level, possible AI approaches include programming agents that are responsible for strategic planning from a macromanagement level, and unit control from a micromanagement level. According to Professor Mccoy’s research published in “An Integrated Agent for Playing Real-Time Strategy Games”, we see that the problem of programming an AI capable of defeating humans at StarCraft is broken down into sub research fields. Notably, there needs to be agents each responsible for economic maximization, build-order optimization, reconnaissance and more.

* **What is your solution and contribution to the solution of this problem?**

Our solution focuses primarily on learning and maximizing efficient build orders. We propose to solve this by training an agent with replays and summary data obtained at the end of games played by pro-players. It is quintessential that we prioritize early game build order, as it has been shown by multiple other researches that it is one of the biggest leading factors to a victory. The initial difficulties to solve include restricting the huge state space so that it can yield positive results at the end of the game as well as real-time strategy creation and counter measures for enemy strategies.

* **Why did you choose your solution given the possible alternatives?**

We drew our inspiration from a preliminary research conducted in “StarCraft II: A New Challenge for Reinforcement Learning”. In the research conducted by the DeepMind and Blizzard team, they used reinforcement learning and successfully beat the built-in rule-based StarCraft AI on some minigames, but the bot is unable to defeat the built-in AI at full game. Furthermore, they propose that “learning from replays should be useful to bootstrap or complement reinforcement learning.” We chose to implement this hypothesis to maximize our research success rate, bearing the assumption that the combined efforts of the creators of AlphaGo speak high volume on the design choice of a StarCraft AI.

**Design and Technical Approach**

* **What AI techniques are you proposing?**

We propose to implement a reinforcement learning agent using techniques such as Actor-Critic that focus on Adversarial Sequence Prediction for timed states within the game.  We are also combining Machine Learning techniques such as neural networks and Search Algorithms like A\*.

* **How do the techniques address the problem? Connect the levels of abstraction between problems space and technical solution. Architectural diagrams help.**

A technique we are considering for facilitating the reduction of state space is to separate the game into individual abstractions that each represent a smaller problem which can be solved separately. Search algorithms like Alpha-Beta or A\* could prove beneficial when taking care of which build order sequence to use for the trajectory of a game, although this would be rather troublesome when we reach the point of micro-managing our units. Lastly, through replays and summary data, as these are very useful training data, we hope to be able to implement an agent's ability to learn to create strategies that lead to victory and to effectively counter the enemy. All of this includes dealing with uncertainty, an uncertainty that we plan to solve through algorithms such as Actor-Critic or Adversarial Sequence Predictions where we are capable of searching the action space at different time intervals and reflecting on choices done during either real-time game learning or replay-game learning, by reviewing smaller sets of actions that will eventually lead to victory or defeat we are allowing the agent to adapt to more specific situations and therefore providing a better sense of choosing the best action and avoiding the worst at each step in the game, allowing the agent to also react positively to changes to the enemy strategies.

* **What is your technology stack?**
  + Numpy, pandas, pysc2, sc2le, tensorflow, scikit learn, PyTorch
  + sublime, eclipse
* **Why is your technology stack the appropriate choice?**
  + Using the pysc2 and sc2le libraries saves us the extra effort at implementing everything from scratch.They are provided by DeepMind and experienced researchers who provide the framework for any interested party to research and develop StarCraft AI bot.
* **What programming environments are you using?**
  + CSIF computers, MacOS High Sierra version 10.13.4.
* **How will you use GitHub?**
  + We will use GitHub for version control and resource sharing.
* **What code quality assurance tools are you using?**
  + We will be using code linting.

**Scope and Timeline**

4.22 - 4.29:

1. Understand what PySC2 API allows us to do by reading [PySC2 implementations](https://github.com/deepmind/pysc2)  and this [DeepMind paper](https://deepmind.com/documents/110/sc2le.pdf) and how to call the functions by typing every line of the code of the agents in this [Github repo](https://github.com/skjb/pysc2-tutorial). Accompany [Medium post](https://chatbotslife.com/building-a-basic-pysc2-agent-b109cde1477c) for the [Github repo](https://chatbotslife.com/building-a-basic-pysc2-agent-b109cde1477c).
2. Look at the [mini-game](https://github.com/deepmind/pysc2/blob/master/docs/mini_games.md).
3. Get all the agents from that repo up and running on our computer, meaning that we will understand the whole workflow of improving upon his agent
4. Finish the proposal
5. Understand how the  [Advantage Actor-Critic agent in this Github Repo](https://github.com/simonmeister/pysc2-rl-agents) works by reading this [blogpost](https://hackernoon.com/intuitive-rl-intro-to-advantage-actor-critic-a2c-4ff545978752).

**Documentation and Access**

**How are you sharing code? What about a readme? A website is recommended. A Github page or project is recommended.**

We have already created a github repo and will share the code through github. README.md contains a brief description of the problem we address in our project and directions and how to run the trained AI bot on user’s programming environment. We will showcase our final work on a website. The technology is undetermined at this point. We will choose to use github page or others depending on our members’ availability and skillset.

**Evaluation**

**Describe your criteria for success. How will you gather data to evaluate your solution? How will it be analyzed?**

Considering mini games as unit tests, our RL agent should be able to achieve human level performance on these games. Realistically we would not be able to develop an RL algorithm that succeeds on full game, as of now current state-of-the-art baseline agents haven't learned to win against the easiest built-in AI on the full game. Therefore, we will build additional mini-games to evaluate our agent in.

**Plan for Deliverables**

**How will you share your GitHub repo? Will you submit your code to tournaments or write a conference paper?**

After developing the code we will make make the repository public on github for future reference. We will definitely submit our code to tournaments if we are able to make even the slightest improvements compared to the rule-based built-in AI bot or the aforementioned AI bots. We will document our research in detail in paper. Hopefully, we could publish our findings.

**Separation of Tasks for Team**

<https://github.com/okdolly/SwaggerBot-StarCraft2/blob/master/Timeline.md>

Phase 1: (Everyone)

* Read API and setup a comprehensive AI agent to improve upon
* Setup PYSC2 environment: <https://github.com/deepmind/pysc2>
* Read paper from deepmind and get familiar with PYSC2 code: <https://deepmind.com/documents/110/sc2le.pdf>
* Walk through the code and tutorial of PYSC2 agent posted in group: <https://github.com/skjb/pysc2-tutorial> and <https://github.com/llSourcell/A-Guide-to-DeepMinds-StarCraft-AI-Environment>

Phase 2:

* Implement PYSC2 agent with passive reinforcement learning.
* Implement API to get necessary observation data from game environment : Axel & Aria
* Build a Passive agent RL network: Rhys & Dolly
* Implement API to execute the output of the network in the game: Shraddha

Phase 3: (Everyone)

* Project report and documentations.
* Integration: Entire team
* Unit Testing: Everyone with their code
* Inline Documentation: Everyone with their own code
* Deliverable Reports: Everyone with their portion

**Why integrating replay failed**

After analyzing the research paper *Learning Macro-management in StarCraft from Replays using deep learning*, and experimenting different methods with existing agents, we realized potential difficulties in using Replays with reinforcement learning to take decisions regarding build order optimization.

In the paper, a system learns macro-management from game replays using deep learning which is integrated into UAlbertaBot, an open source StarCraft bot.

First, data processing is a meticulous task and is a relatively long process. As StarCraft has several units, buildings for all 3 races, it increases the dimensionality of the data. In the paper the preprocessed dataset contains 2005 state-action files with a total of 789571 state-action pairs.

**Evaluating the results**