Creating AI Agent for Starcraft II: A Deep Reinforcement Learning Approach

Denis Barkar(dbarkar@stanford.edu), Peter Lapko(plapko@stanford.edu)

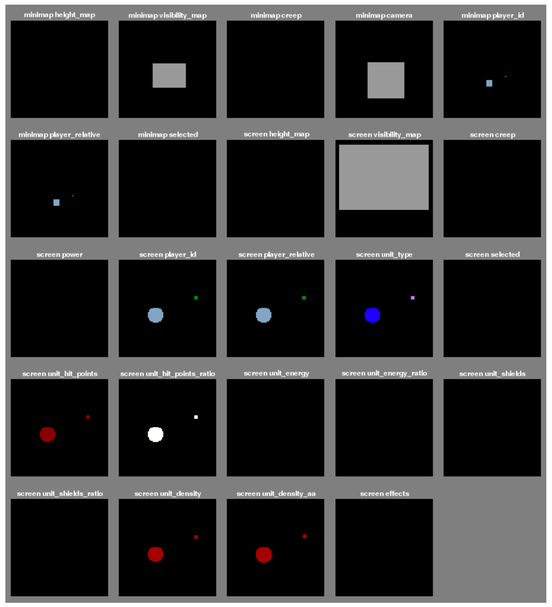
Motivation and Goals

In the recent years there are were a lot of success in applying DL to a large variety of simple games (like Pong or Brakout) and now people with some success try to use it for playing more complex games like Dota 2. We wanted to try using DL to create an AI agent for Starcraft II.

Problem definition and challenges

The action and state spaces of StarCraft 2 are very large (hundreds of actions and enormous variations of created units, their positions e.t.c.) making the creation of a good model for the game very time- and effort consuming. Thus we limited ourselves to creation a model for a mini-game where the player must make a marine collect as many as possible beacons, which spawn one after another, by walking over them within set time limit. The example of the game screen can be seen on the left image.

Environment



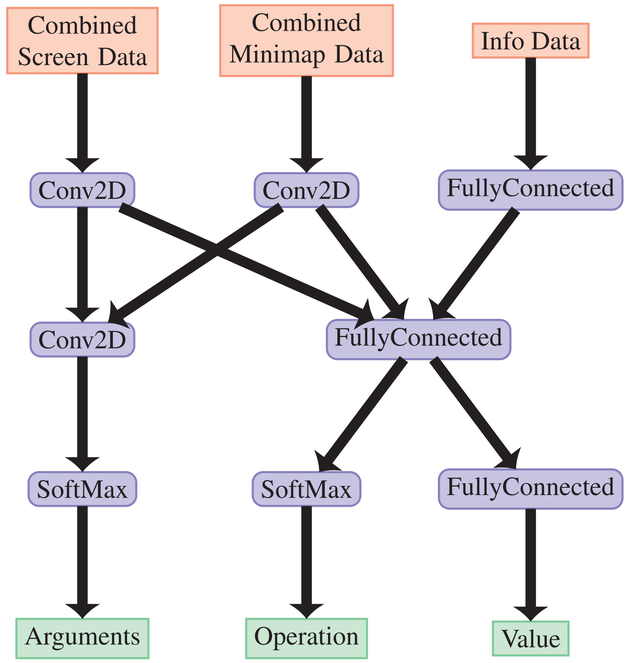
For the interaction with the game we used PySC2 (DeepMind's Python component of the Star-Craft II Learning Environment based on Blizzard's StarCraft II Machine Learning API).

PySC2 generates a set of feature layers, each represents something specific in the game (unit type, hit points, owner, or visibility).

Action space is designed to mimic the human interface. For instance, action **[select rect, [[select add], [x1, y1], [x2, y2]]]** selects multiple units by drawing a rectangle.

PySC2 defines two different reward structures: ternary 1 (win) / 0 (tie) / -1 (loss) received at the end of a game, and Blizzard score. The ternary win/tie/loss score is the real reward that we care about. The Blizzard score is the score seen by players on the victory screen at the end of the game, it is computed as the sum of current resources and upgrades researched, as well as units and buildings currently alive and being built.

Approach



To tackle the problem we decided to use the AC3 method as it shows good result for a variety of 2D and 3D games. On the left is the model of the network used for approximation of policy and value function where:

- The **Info Data** is the one-zero vector corresponding to available action indices for the current enviroment state
- The **Combined Minimap Data** is the sum of matrices representing enviroment layers on minimap: terrain height levels, absolute ids of unit owners, unit type ids, e.t.c.
- The **Combined Screen Data** is the same as the **minimap** but for the screen area
- The **Value** is a scalar value and the approximation of the value function.
- The **Operation** is a vector representing probabilities of the actions according to the current state.
- The **Arguments** is a vector representing the probabilities of the combined arguments (value of the first argument multiplied by the maximum possible value for the first argument plus the value of the second argument) for the current action.

Also to vary the policies and improve the learning process we perform ε-exploration with decreasing ε (decreasing randomness) as the number of epochs increases.

Results

With our approach we managed to successfully train models that show performance better that average shows average person:

Player	Average score
Human	20
Perfect bot	26
Our Model	25

Currently our model can be successfully trained in $\approx$20% cases of total train attempts.

Analysis and Future Work

Our results show that current approach is very unstable in terms of training and requires further improvement. Possible approaches include simplification and optimization of current model, usage of the human prepared replays.
Also we consider testing this model for other mini-games and trying other DL algorithms.