Jeffrey Barratt

jbarratt@
cs.stanford.edu

Deep Imitation Learning for Playing Real Time Strategy Games

Chuanbo Pan

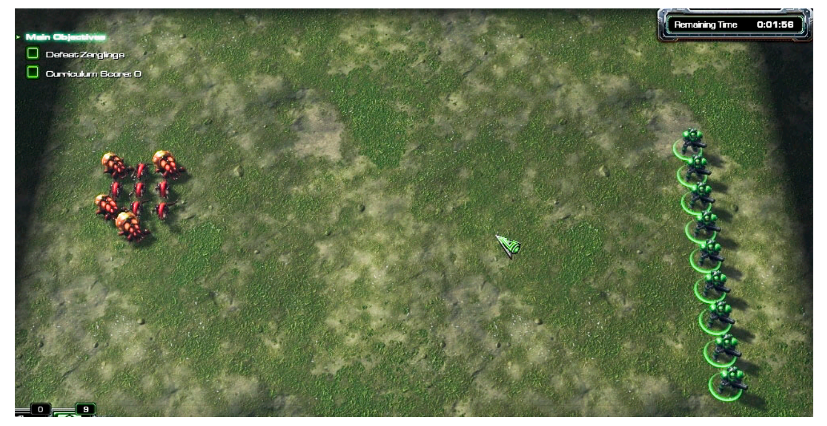
chuanbo@
cs.stanford.edu

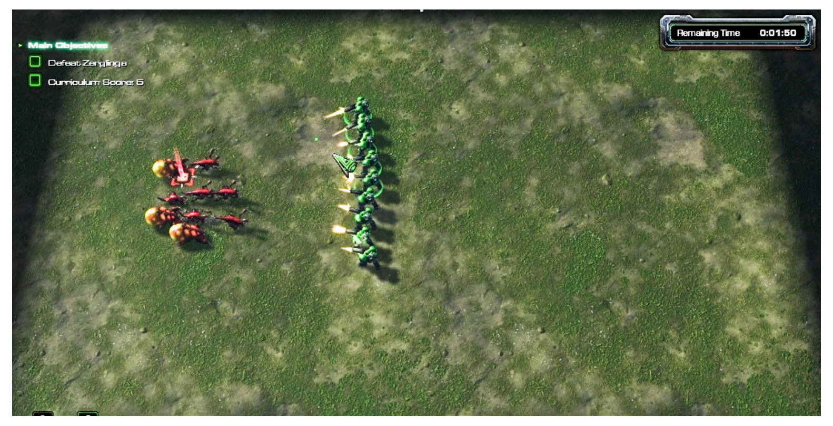
Motivation

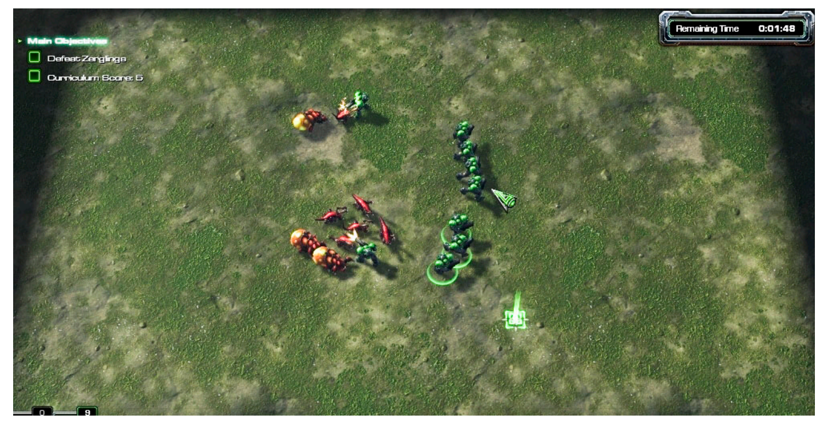
Competitive Computer Games, despite recent progress in the area, still remain a largely unexplored application of Machine Learning (ML), Artificial Intelligence (AI), and Computer Vision. Real Time Strategy games such as StarCraft II provide an ideal testing environment for AI and ML techniques, as:

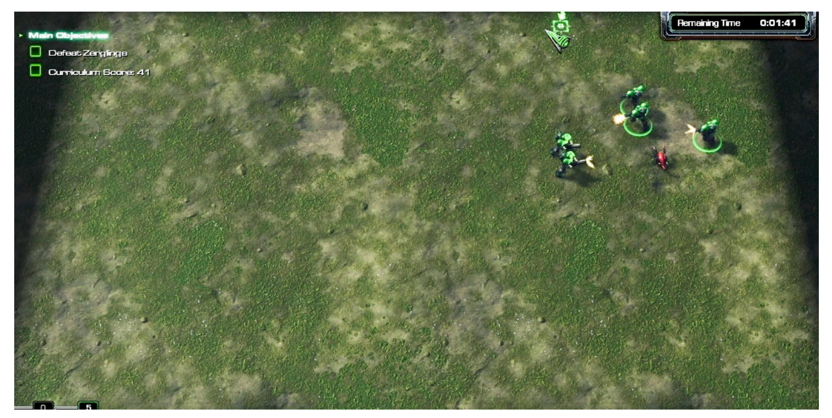
- They run in **real time** and involve **incomplete information** (the player cannot see the whole battlefield at once).
- Users must **balance** making units, controlling those units, executing a strategy, and hiding information from their enemy to successfully win a game of StarCraft II.
- Simple actions made early on in the game can greatly impact later stages of the game.

Problem Definition


1) The starting position of the PySc2-provided mini-game


2) The marines can start by attacking the more dangerous targets first


3) Splitting the marines mitigates the effect of splash damage


4) Kiting can be used to trade more efficiently with the Zerglings

We focus our efforts on one mini-game which involves “splitting”, a process of engaging area-of-effect units with a group of low-health ranged units, where the low-health units are moved apart to mitigate the area of effect damage to as few units at once, as well as other management tactics such as kiting and targeting. We collected data by playing the mini-game repeatedly and recording all actions taken and the state they were made in, yielding around 10,000 state-action pairs.

Challenges

Apart from the reasons mentioned in the motivation, StarCraft II is a challenging game because:

- The action space is huge – **$O(n^4)$** possible selections, and **$O(n^2)$** possible places to move to.
- The game is in real-time – can’t spend forever thinking!
- The game is **huge** and **multi-layered**.
  - Have to consider attacking, producing, gathering, etc.
  - Our mini-game already includes tasks such as moving to beacon (enemy), attacking, and moving back.
- **Overfitting** the data is an easy trap to fall into.

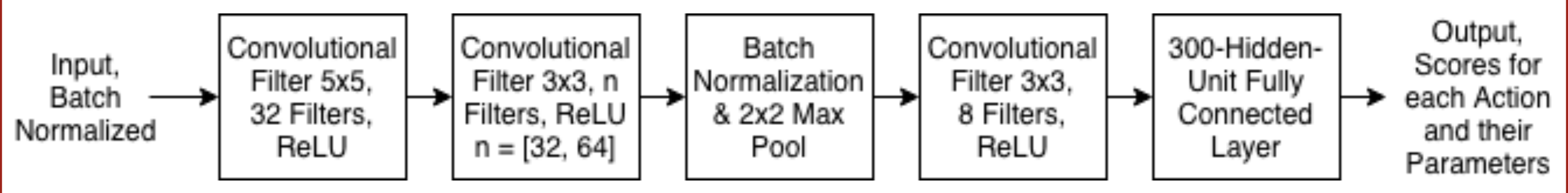
Features and Model

We use the newly-released PySc2[1] framework to interface with the StarCraft II environment. This API allows us the following channels to communicate with StarCraft II:

- A state **feature vector** that represents what the agent currently sees on the screen.
  - 17 84x84 feature layers, each representing a different aspect of the game (health, alliance status, etc.).
  - Because they preserve the spatial layout of the screen, feature layers act as “images” of what’s currently seen.
- A set of **available actions** that the agent can legally make in the current state.

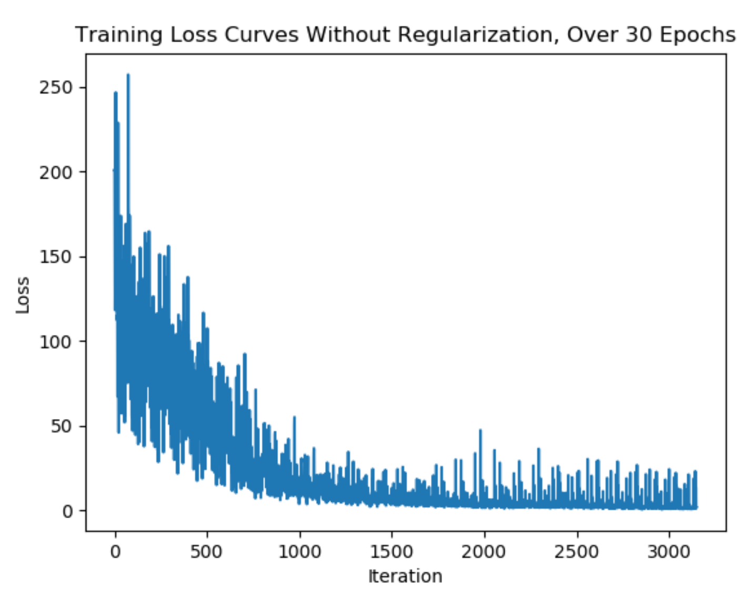
We wrote code to convert the features and actions from StarCraft II to the network and vice versa.

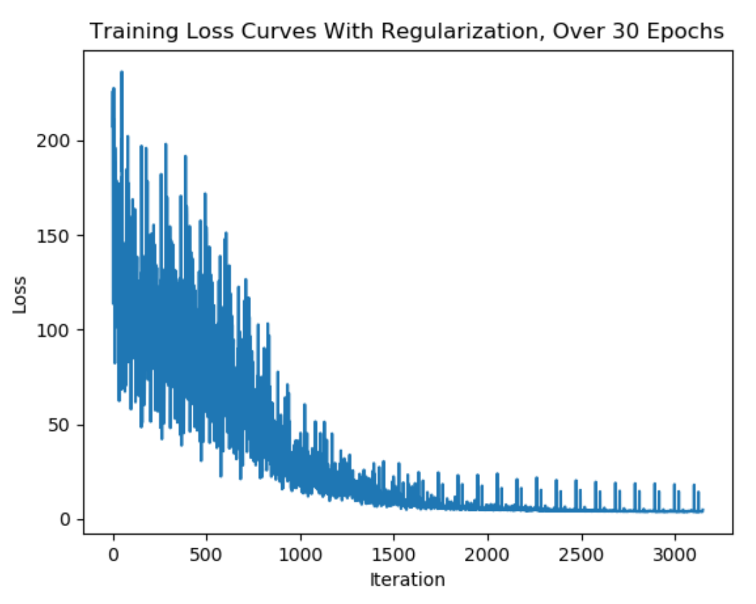
Our approach utilizes Deep Behavioral Cloning. We take advantage of the **image-like** nature of the state by using a Convolutional Neural Network (CNN).

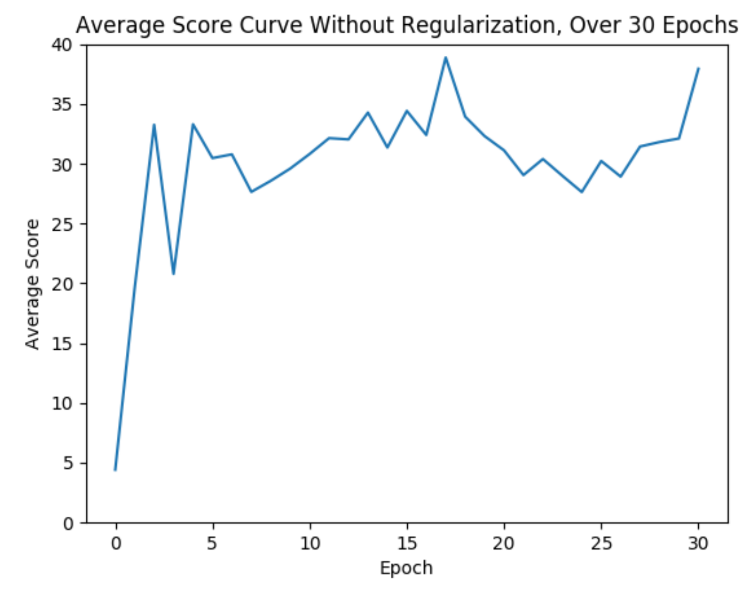

Input, Batch Normalized
Convolutional Filter 5x5, 32 Filters, ReLU
Convolutional Filter 3x3, n Filters, ReLU n = [32, 64]
Batch Normalization & 2x2 Max Pool
Convolutional Filter 3x3, 8 Filters, ReLU
300-Hidden-Unit Fully Connected Layer
Output, Scores for each Action and their Parameters

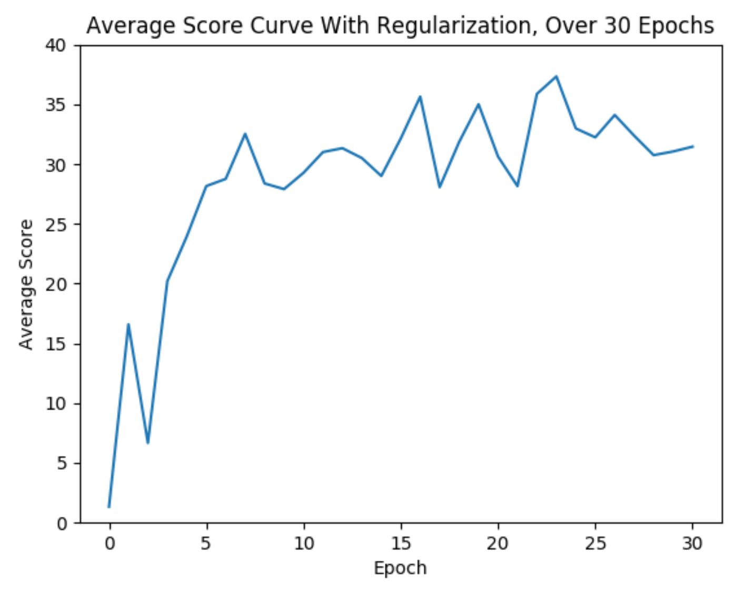
The network outputs scores for each possible action (select, attack, move, noop, etc.) as well as a best guess for each of their parameters for the current state, which will for the most part be coordinates of where to attack, select, move, etc.

Results









The graphs on the top show training loss during training with and without regularization. The graphs on the bottom show average scores for the mini-game during training.

Discussion and Future Work

We can observe several things from our results:

- Theoretical score ranges from -9 to 750+. Obviously, we are not near maximum potential.
- The learning curve indicates that we are improving our score from random but still suffer from overfitting.
- The agent sometimes gets stuck moving back and forth between two states.
- The agent doesn’t seem to have a full understanding of what it’s doing, just mimicking what looks right.

Future work, were we to have 6 months to work:

- Using a LSTM Recurrent Neural Network (RNN) to help determine actions, especially in sequence.
- Refining our abstraction to ensure stability during training
- Gather more training data to cover more possible cases.

References

[1] Vinyals, Oriol, et al., "StarCraft II: A new challenge for reinforcement learning." *arXiv preprint arXiv:1708.04782* (2017). [Online]. Available: https://arxiv.org/abs/1708.04782.