PLAYING STARTCRAFT II MINI-GAMES WITH DEEP RL

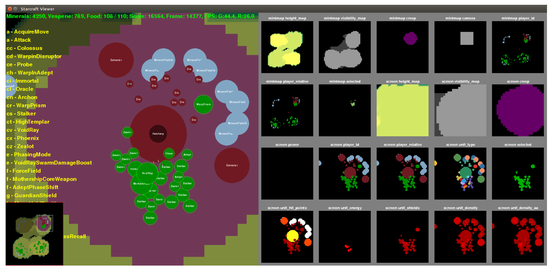
Haiyuan Mei
Stanford University, Department of Computer Science

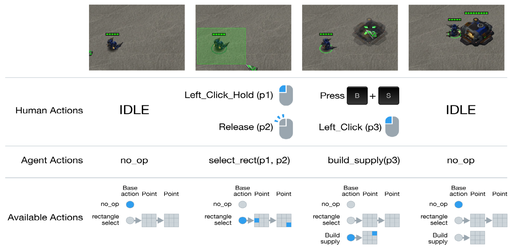
OVERVIEW

- **Motivation**: Explore RL application on large sparse state/action space and delayed credit assignment requiring long-term strategies.
- **Approach:** Replicate and experiment on an Asynchronous Advanced Actor-Critic algorithm, with a Fully connected CNN as regression function to learn MDP policy and value.[1]
- **Training and evaluation:** Two of the mini-games are selected in training and assessing the reinforcement learning: MoveToBeacon and FindAndDefeatZergling. After the training with A3C+FullConv, run evaluation using the model learned, and compare it with running on an Random model.

MODEL

- **States:** 1. Spatial observations **minimap**: feature layers representing the state of the entire world, each represents a specific aspect of the game (terrain height, fog-of-war, creep, camera location, player identity, etc.) 2. Spatial observations **screen**: A detailed view of a subsection of the world(the players on-screen view where actions are executed) 3. Non-spatial observations **info**: various non-spatial information such as gas and minerals collected, set of actions currently available, etc.





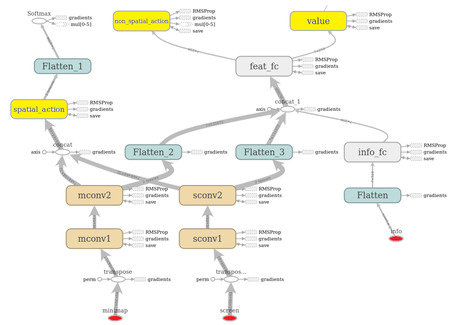
- **Actions:** Mimics the human interaction with keyboard and mouse. An action a is composed of a function ID and a list of arguments for that function. For example: [**select rect**, [[**select add**], [x1,y1], [x2,y2]]]. Approximately 300 action-function identifiers with 13 possible types of arguments available in game, but only a subset are valid in each game state.
- **Rewards:** Blizzard score: Blizzard scored at every step can be obtained by RL agent which can be used as the incremental 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.

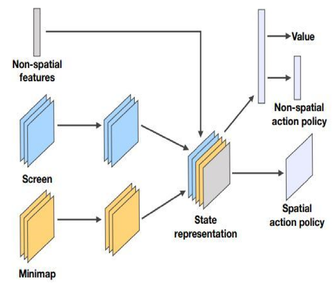
ALGORITHM

- **Policy Gradient:** Instead of learning value functions $Q_\theta^\pi(s)$ and $V_\theta^\pi(s)$, learn $\pi_\theta(s,a)$ directly
- **Vanilla policy gradient:** $\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(s_t,a_t)v_t$; v_t is available only at the end of each episode;
- **Actor-Critic**: Add a target: $Q_w(s,a)$, in this case it is a fully connected CNN.
  - **Policy gradient**: $\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(s_t,a_t)Q_w(s,a)$; Value target: $Q_w(s,a)$
  - **Value gradient**: $\beta(R_s^a + \gamma Q_w(s',a') - Q_w(s,a))Q_w(s,a)$
  - **A2C**: $\beta(Q_w(s,a) - V(s))Q_w(s,a)$, one-step TD error $\beta(r + \gamma V(s') - V(s))Q_w(s,a)$.
  - **A3C [2]:** a global_network θ, w, each worker thread has it's own copy, update separately.

FULLCONV

- **FullyConv:** The figure on the left is an implementation of the fullconv[2], the figure on the right is what proposed from SC2LE[1].
  - **1.** screen/minimap pass through separate 2-layer convnets with 16/32 filters of size 5x5/3x3.
  - **2.** state is formed by concatenation screen, minimap network outputs as well as the broadcast vector statistics (info)
  - **3.** non-spatial policies/value function: pass state through a fully-connected layer with 256 units and ReLU, followed by fully-connected linear layers
  - **4.** spatial policies is obtained using 1x1 convolution of the state with a single output channel



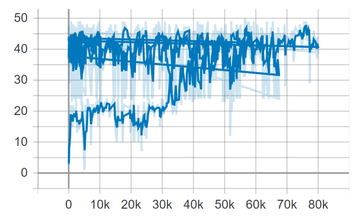


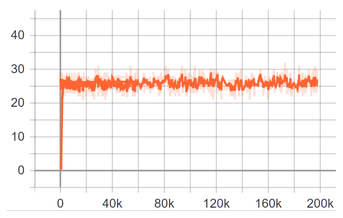
EXPERIMENT

- Comparison between my experiments and DeepMind Random policy, DeepMind baseline:

Experiments	MoveToBeacon	FindAndDefeatZergling
DeepMind Random	1	4
DeepMind Baseline	26	45
A2C Experiment	25.9	40.9
A3C Experiment	-	-

- Blizzard score curve(FindAndDefeatZergling (abnormal plot), MoveToBeacon (stable around 25)).





DISCUSSION

- A2C VS A3C: 12 core CPU+16G RAM+8G NVIDIA GPU, A3C seems a lot slower than A2C, need more investigation
- Future: Pretrain with replay data, full game.

REFERENCES

[1] Oriol Vinyals and Timo Ewalds et al. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.
[2] Volodymyr Mnih et al. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.