上海科技大学
ShanghaiTech University

Large action space end-to-end Reinforcement Learning

ShanghaiTech CS181 Final project | Final presentation

刘睿琪 Rui-qi Liu
liurq@shanghaiTech.edu.cn
86396172

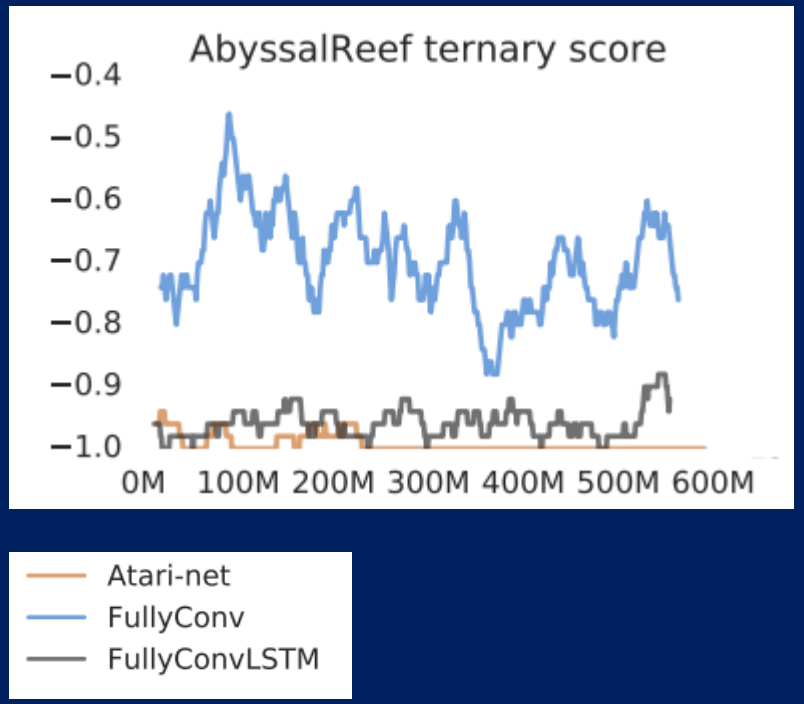
秦琦 Qi Qin
qinqi@shanghaiTech.edu.cn
49407951

徐寅韬 Yi-tao Xu
xuyt@shanghaiTech.edu.cn
53533324

袁可一 Ke-yi Yuan
yuanky@shanghaiTech.edu.cn
74638067

Motivation

From StarCraft II: A New Challenge for Reinforcement Learning
Baseline method indicates:



- Poor result sadly.. What is the reason?
  - large state space
  - Large action space
  - Poor independent assumption of the probability of action
- Motivation2: success of alphaGO

Our contribution

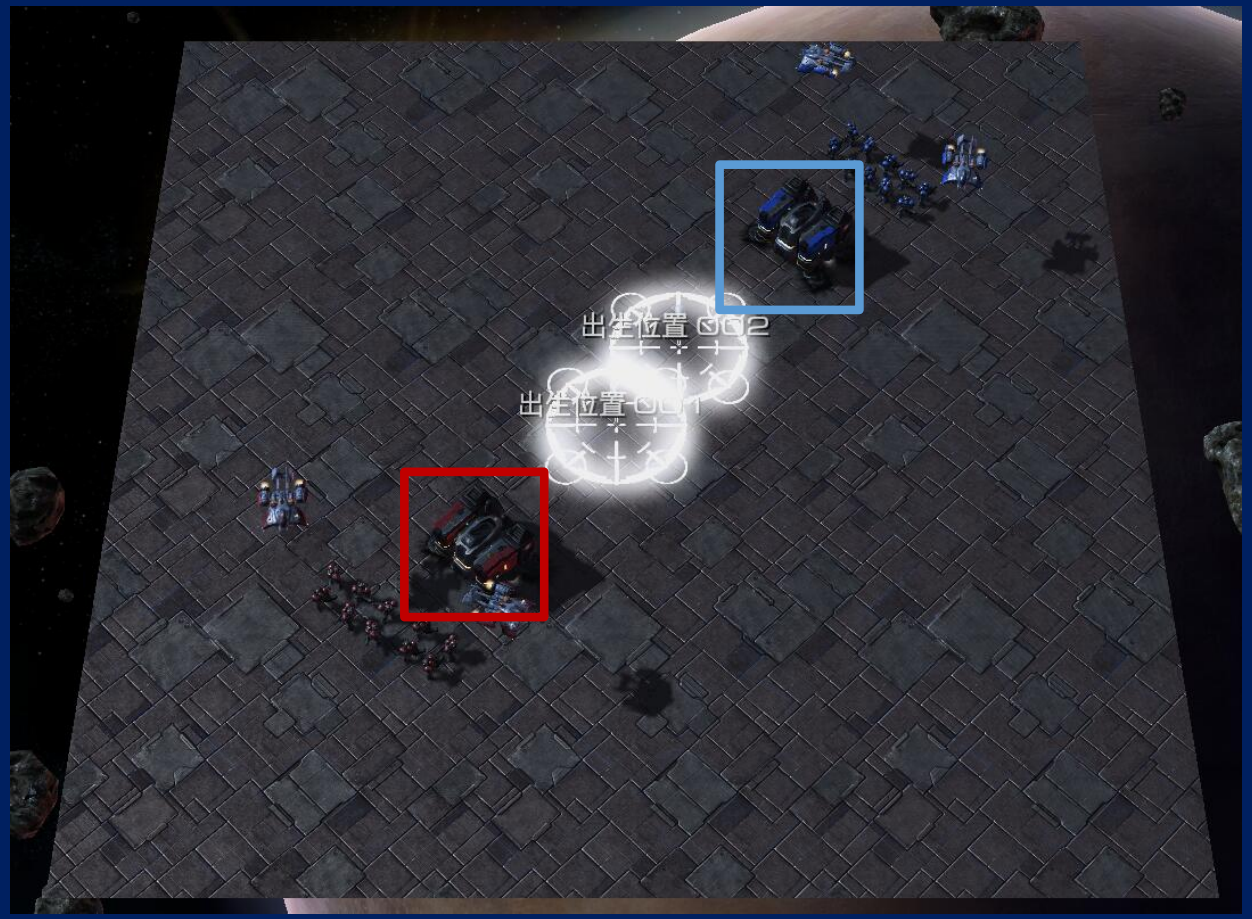
- Provide a PVP mini-game map basing on pysc2 API.
- Apply baseline DQN
- Handle the large action space

New task

raise a new practical mini-map, instead of extremely hard and complicated full sc2 game basing on slight modification of pysc2, a deep Mind API

- practical for RL-learning experiment
  - most of sc2 works base on how to hard-code good expertise knowledge(e.g: timing to rush, hard-code attack target evaluation function)
  - highly rely on human being replays
- a well-tested reward system
  - our baseline method shows that it works
- concentrate as much elements as possible
  - conserve attacking-training tradeoff

New task



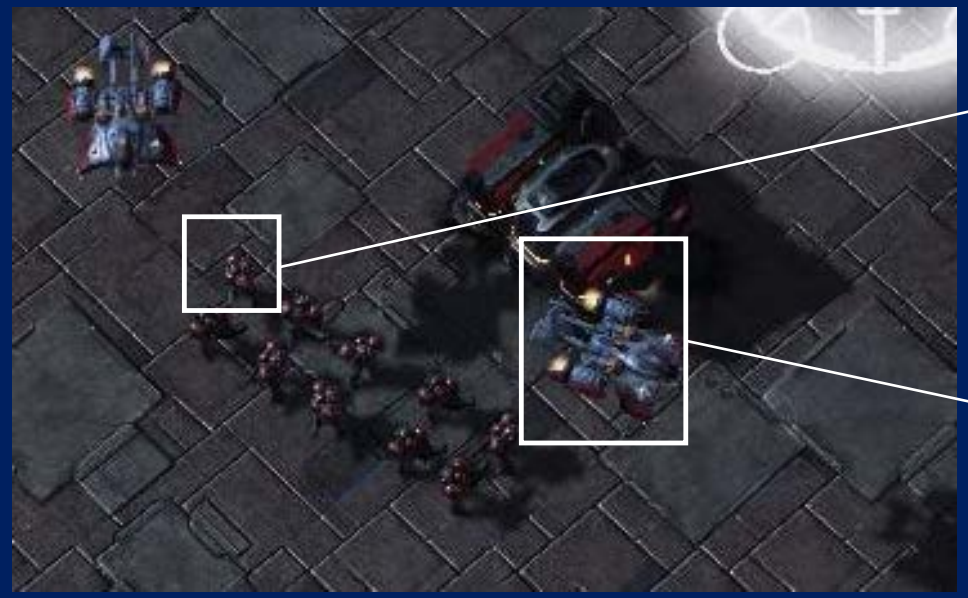
Chief goal:

- command your troop destroy other player's barracks

Game setting:

- Receive 10 minerals every second
- 50 minerals can be used to train a marine in one's barrack.
- everyone has his/her initial troop

New task



marine can cause damage

- only damage source in this sc2 minigame
- can be “produced” by mineral

medivac

- move faster
- cannot attack
- be able to heal the injured marines

Initial troop:

10 marines & 2 medivac

New task

Still many challenges

attack-or-train tradeoff

- shall AI focus on operations to attack others or train a new marine to enhance your troop

marine-or-medivac dilemma

- kill a marine can reduce other's damaging power
- Kill medivac can reduce other troop's endurance

Our contribution

- Provide a PVP mini-game map basing on pysc2 API.
- Apply baseline DQN
- Handle the large action space

DQN Overview

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Q approximation function
In DQN is a neuro network

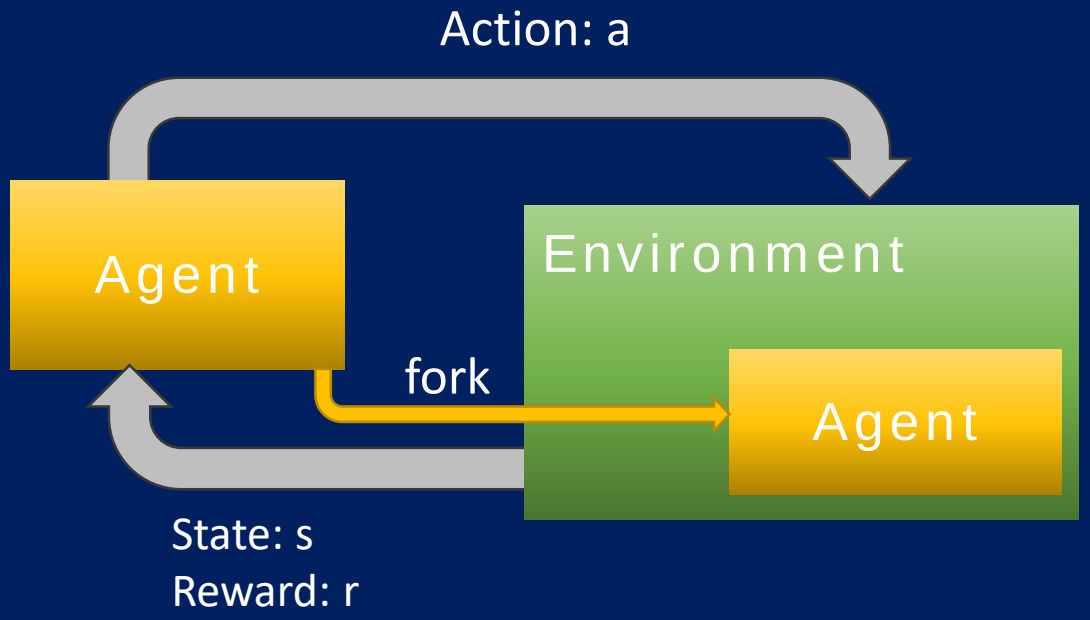
Let's first focus on the establishment
Of our neuro network

From DeepMind, Playing Atari with Deep Reinforcement Learning,2013

Motivation

Policy 2: self-game

AlphaGo zero opponent setting policy

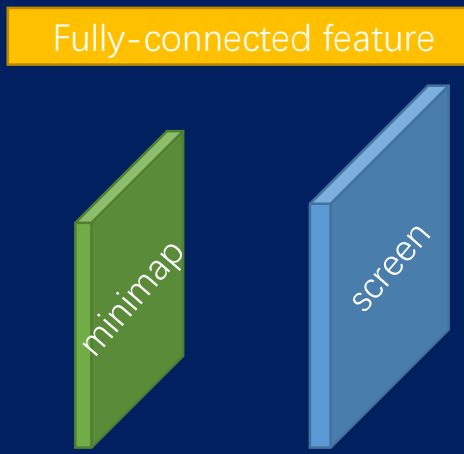


one training step

Network architecture

Inputs:

1. 1-D input[63x1]
   - current minerals
   - count of army
   - environment reward
   - 20x3 selected unit information
2. 2-D minimap information [64x64x3]
   - camera
   - player-relative information [hostile/relative]
   - selected unit info
3. 2-D screen information [84x84x8]
   - visibility
   - …



Experiment result

Episode 1

No command of spatial info
(the upper player hardly move)

No knowledge of who I am

Simply attack other's base

Remedy: switch side of training periodically

Experiment result

Episode 100

More conservative

Experiment result

Episode 200

More technical
Attack

Learn how to train

Experiment result

Episode 440

More fluent

More diversity of policies

Other tricks

Prioritized Experience Replay

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

Sampled by weight instead of uniformly.
Similar to explore function in lecture.

My Implementation:

1. Update weight of sample when it is selected to be trained
2. Epsilon-greedy method: w.p of ε, it picked sample in replay repository w.p. $\propto$ weight(by some function); w.p of 1-ε, it still pick sample randomly

Explore function(weight):

$$\frac{last\ training\ loss}{\#learned\ time + 1}$$

Motivation from, but not the same.

Prioritized Experience Replay

Tom Schaul, John Quan, Ioannis Antonoglou, David Silver

(Submitted on 18 Nov 2015 (v1), last revised 25 Feb 2016 (this version, v4))

Other tricks

Periodic Epsilon-greedy(not recommended)

With probability ϵ select a random action a_t

Motivation: we require consistency of operation in playing RTS game. Randomness of a trivial operation may not provide enough heuristics for exploration

My Implementation：

1. Two mode: sanity mode and random mode
2. In sanity mode, it pick action with largest Q value; Otherwise, act randomly
3. With some probability ε， switch to random mode; Otherwise, switch to sanity mode

Our contribution

- Provide a PVP mini-game map basing on pysc2 API.
- Apply baseline DQN
- Handle the large action space

Network architecture

Output:

Huge action space(a lot of Q values):

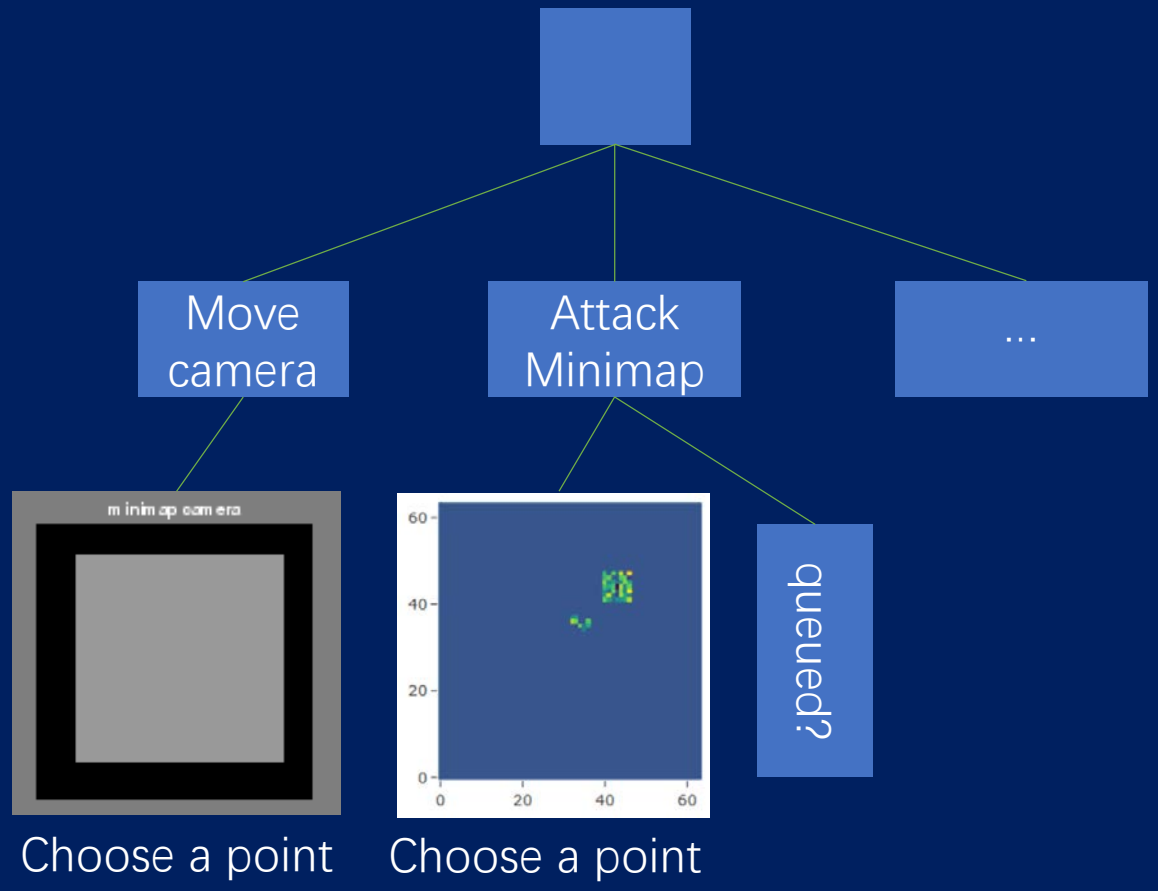
- no operator
- Move camera([0~63], [0~63])
- …

First, reframe it as a tree

Challenge:

Huge amounts of Q-value.

- From POV of tree, there are $O(b^d)$ Q-value(leaves),
  - where d is the depth of the tree, b is the maximum branch factor
  - b=84*84=7056, d=3, O(~3.51x10^{11}) operations
- Cannot be produced end-to-end by network



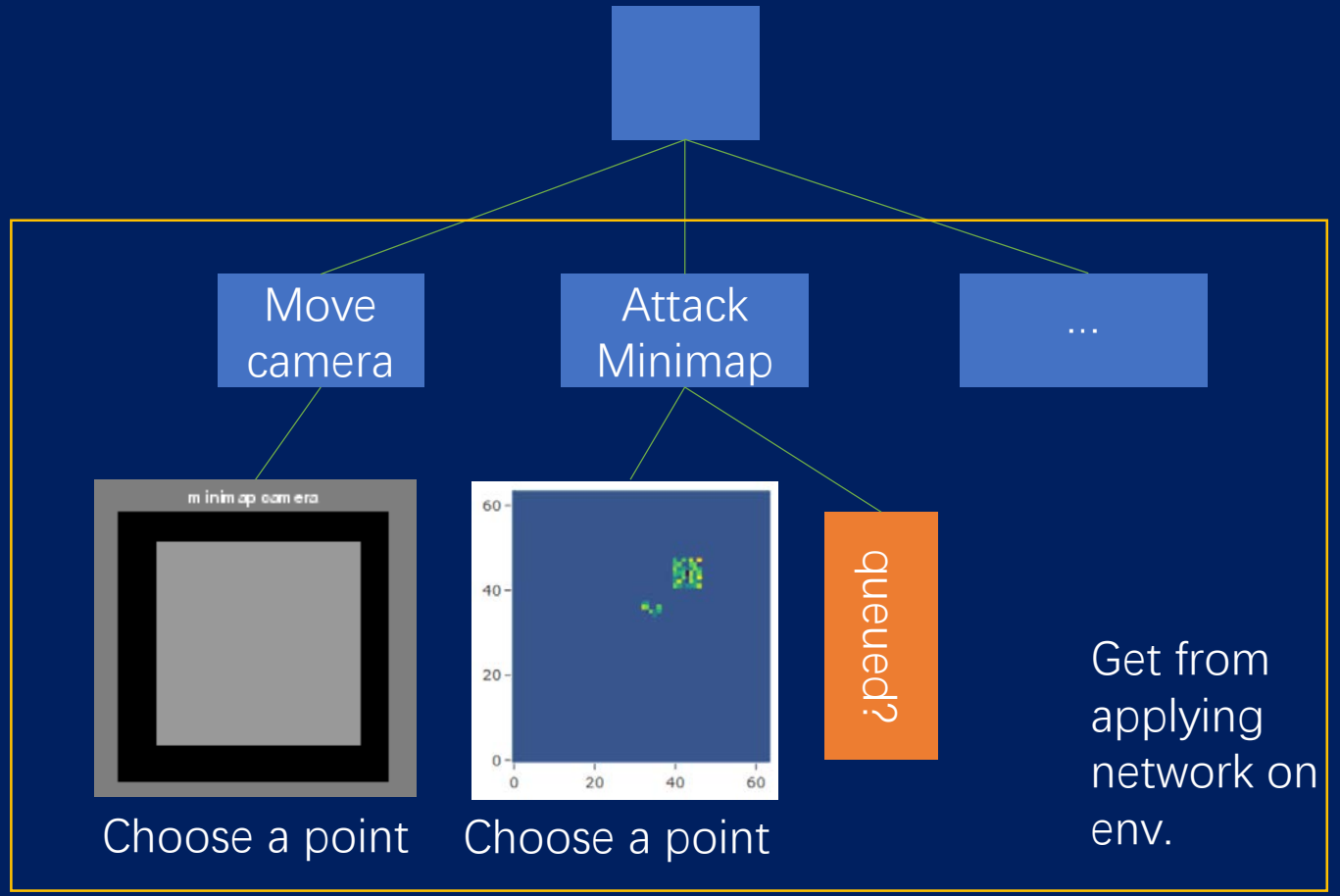
Network architecture

Challenge:
Huge amounts of Q-value.

- From POV of tree, there are $O(b^d)$ Q-value(leaves),
  - b=84*84=7056, d=3

My new implementation:
(TQGP) tree-leveled Q-value generating process

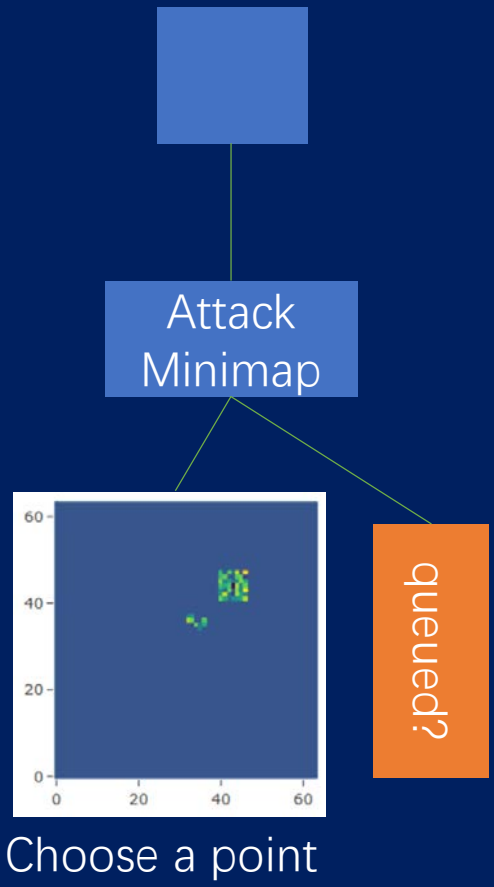
Intuition: network outputs importance value(IV), average the same importance but different semantics IV, promote parallel IVs, promote and average low levels IVs to its parent's IVs to compute a multi-leveled action. Top-level IV is Q-value



Tree-leveled Q-value generating process

My implementation:
(TQGP) tree-leveled Q-value generating process.
A method to handle large action space.
Network:
State -> Importance value (IVs).
TQGP:
(IVs, tree)->Q-Values
Three operations of TQGP:
(1) promote (2) average (3) promote and average

Example:
Compute Q-value of the operation
Attack to (10, 10) in screen and not queued
Sc2 allows queued operation (there is a FIFO queue for attack move)



Tree-leveled Q-value generating process

Example:

Compute Q-value of the operation

Attack to (10, 10) in screen and not queued

Sc2 allows queued operation (there is a FIFO queue for attack move)

Task1: find IV of choose a point(to attack)

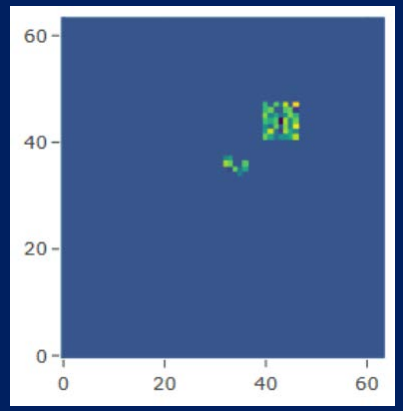
(1) Promote

Intuition: the importance of choosing all points are the same.

Details:

Network outputs a=[84x84] IVs for choosing a point to move camera, promote a[10, 10] since we're going to compute Q-value of the operation w.r.t. to attack [10,10] on the screen

Similarly, we can pick whether to queued by promote operation.



Choose a point

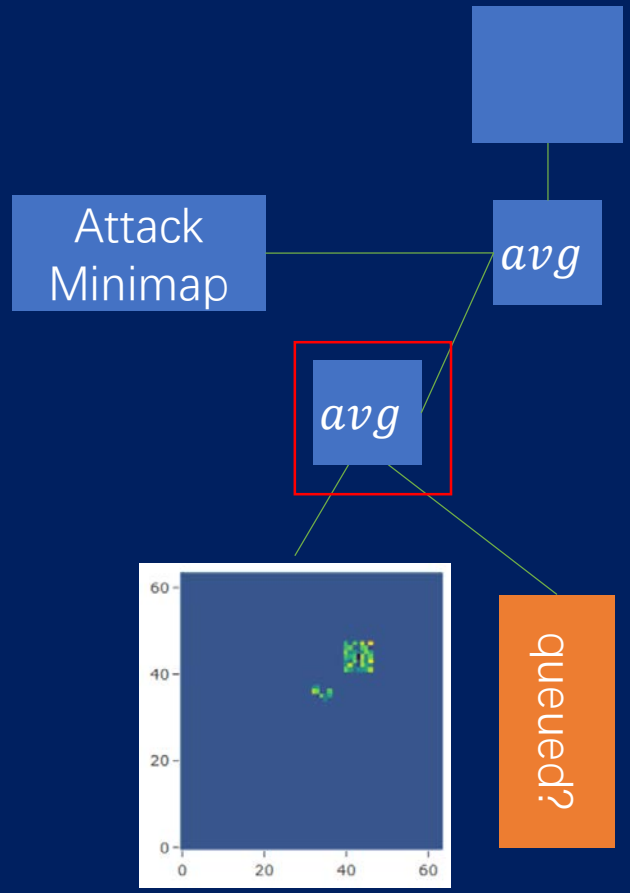
Tree-leveled Q-value generating process

Example:

Compute Q-value of the operation

Attack to (10, 10) in screen and not queued

Sc2 allows queued operation (there is a FIFO queue for attack move)



Task2: find IV the whole operation

(2) Promote

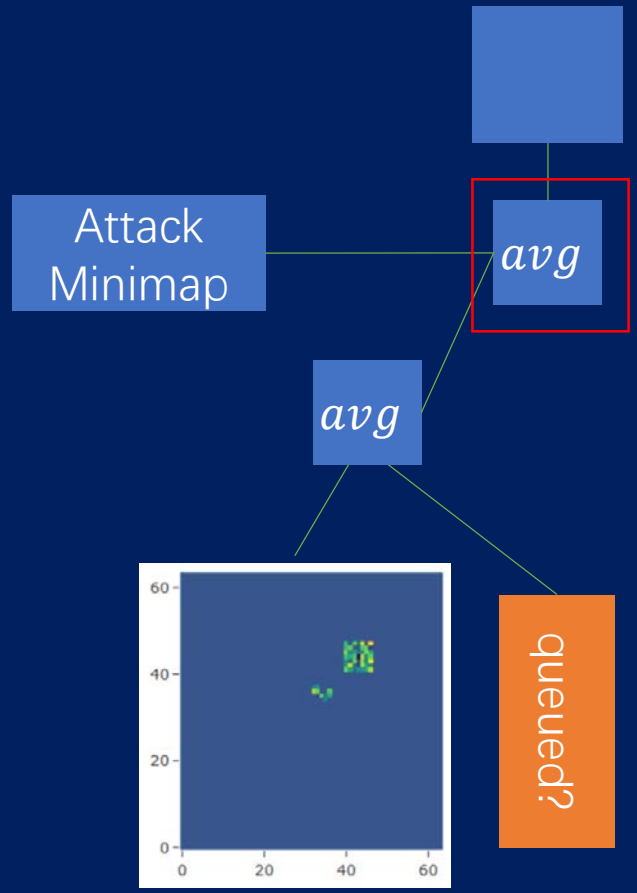
Intuition: the importance of choosing a point are the same and whether to queue roughly contain similar level of semantics(w.r.t. whether to attack such a general idea)

Details:

Average IV of whether to queue and IV of picking a point

Tree-leveled Q-value generating process

Example:
Compute Q-value of the operation
Attack to (10, 10) in screen and not queued
Sc2 allows queued operation (there is a FIFO queue for attack move)



Task2: find IV the whole operation

(3) Promote and average

Intuition: details(picking a point, whether to queue) and general ideas(whether to attack) shares the same importance. Promote IV of the set of details (gained by average) and to take average with higher level. Their relationship in three is parent and children

Details:

Average IV of whether to attack minimap with the average of picking a point and whether to queue. Set it as the IV of attack minimap. Now we are at top of the semantic level in tree(except the dummy root). It is the Q-value!

Tree-leveled Q-value generating process

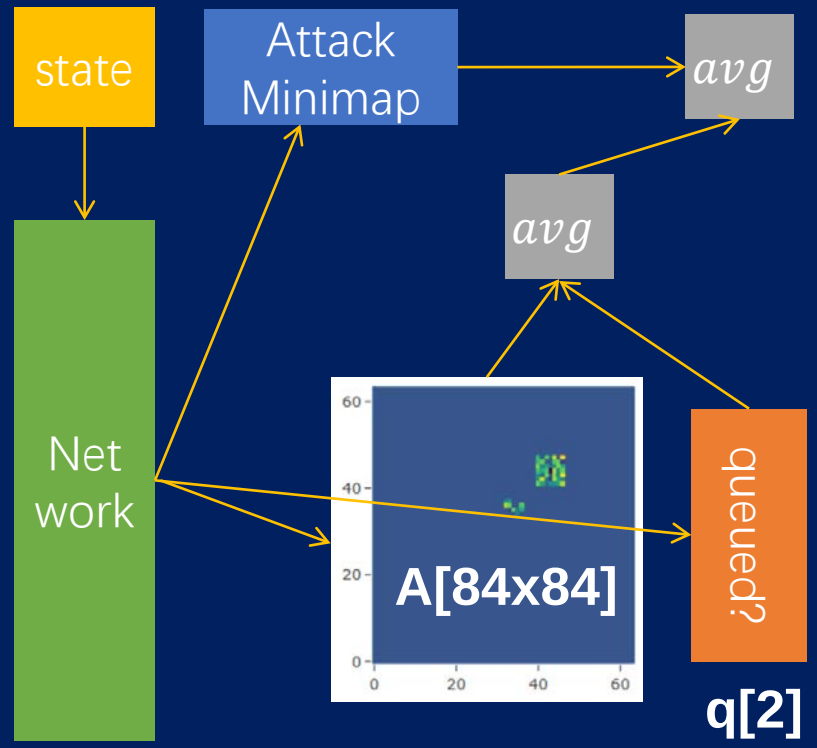
Example:
Compute Q-value of the operation
Attack to (10, 10) in screen and not queued
Sc2 allows queued operation (there is a FIFO queue for attack move)

Big formula

$$\frac{(A[10,10] + q[false])/2 + IV_{attack}}{2}$$

Fact: It is a convex combination of IVs.
Engineering hint: If you keep all IV in the same scaling, the final Q value is in the same scaling, which helps converge.

Compute graph



Tree-leveled Q-value generating process

(TQGP) tree-leveled Q-value generating process

Task now: Pick the maximum Q-value

$$\text{select } a_t = \max_a Q^*(\phi(s_t), a; \theta)$$

Solution: take the max Q-value at top level of Q-value Tree.

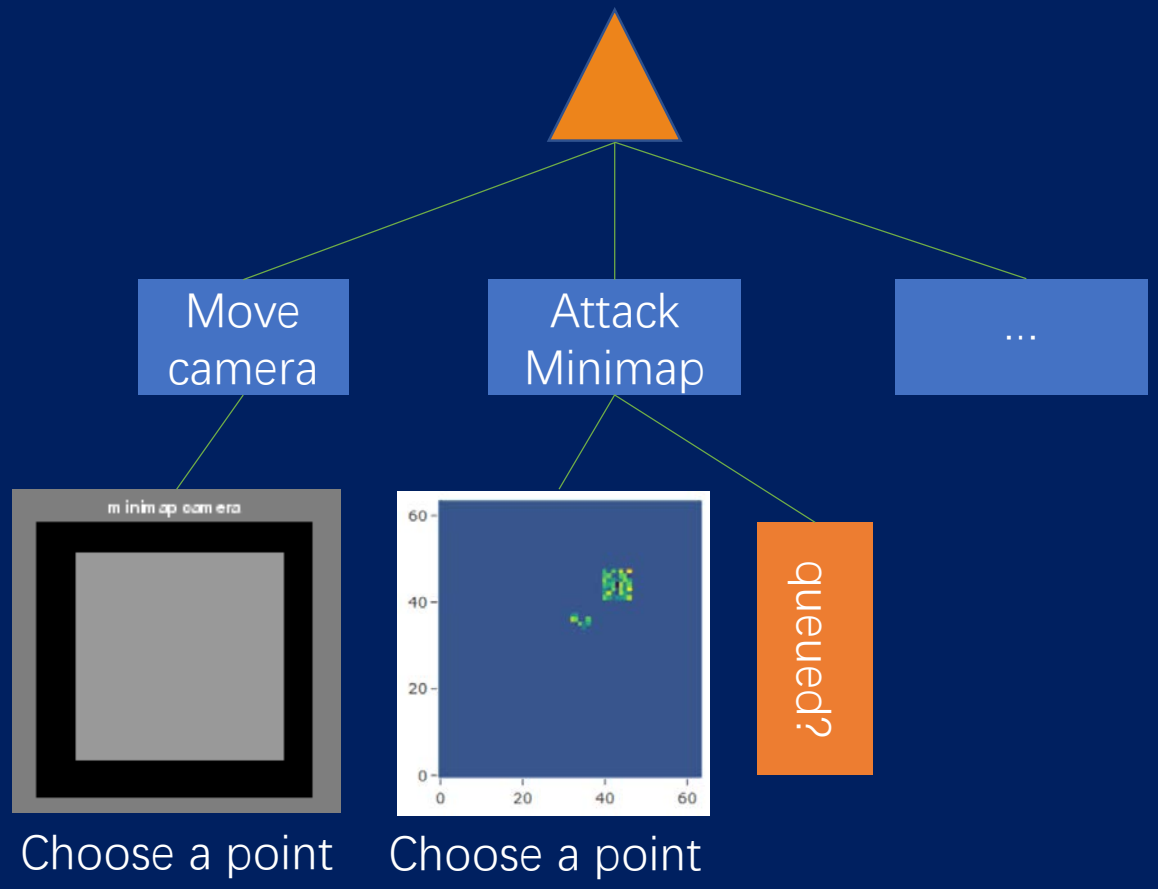
But challenge still exist:

Challenge:

Huge amounts of Q-value.

- From POV of tree, there are $O(b^d)$ Q-value(leaves),
  - b=84*84=7056, d=3

Solution: like min-max in lecture, move down the max operator



Tree-leveled Q-value generating process

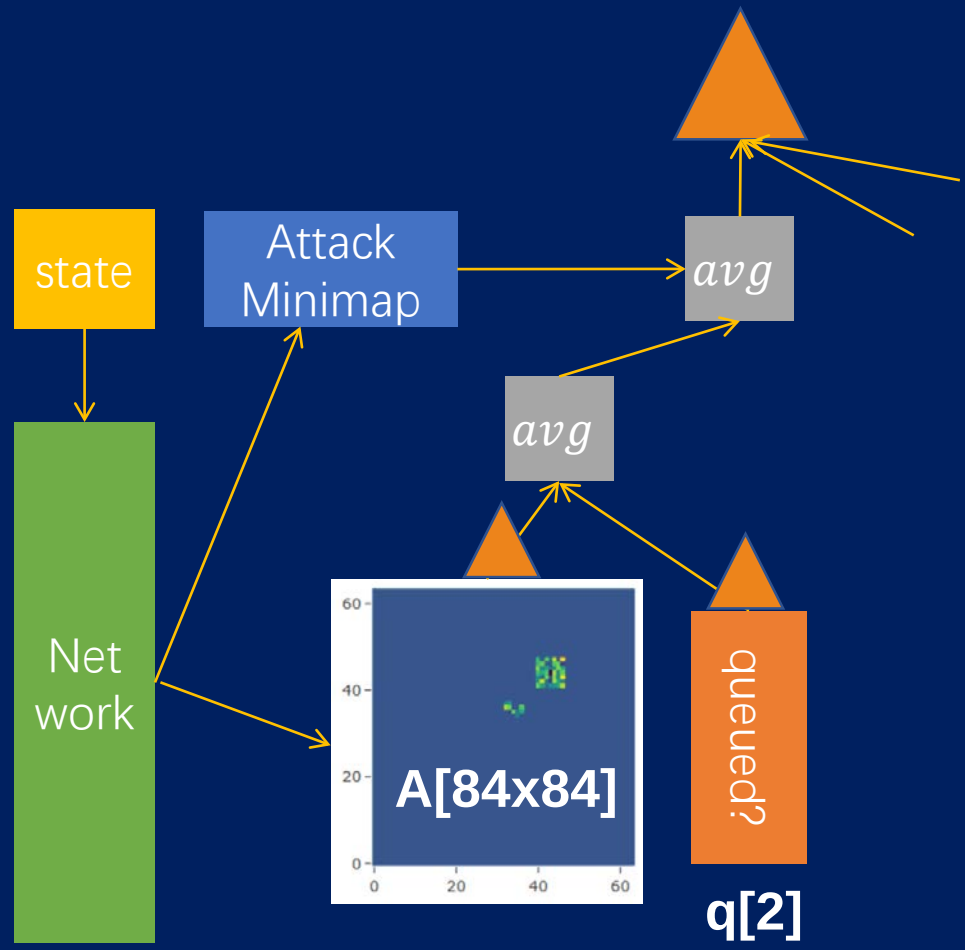
Task now: Pick the maximum Q-value
Solution: take the max Q-value at top level of Q-value Tree. Like min-max in class, move down the max operator.

Math trick here, for some set A:

$$max_{b,d \in A}(ab + cd)$$
$$= a \cdot max_{b \in A}(b) + c \cdot max_{d \in A}(d)$$

s.t: a, c > 0

“max can transfer on convex combination”
Average is a special convex combination



Tree-leveled Q-value generating process

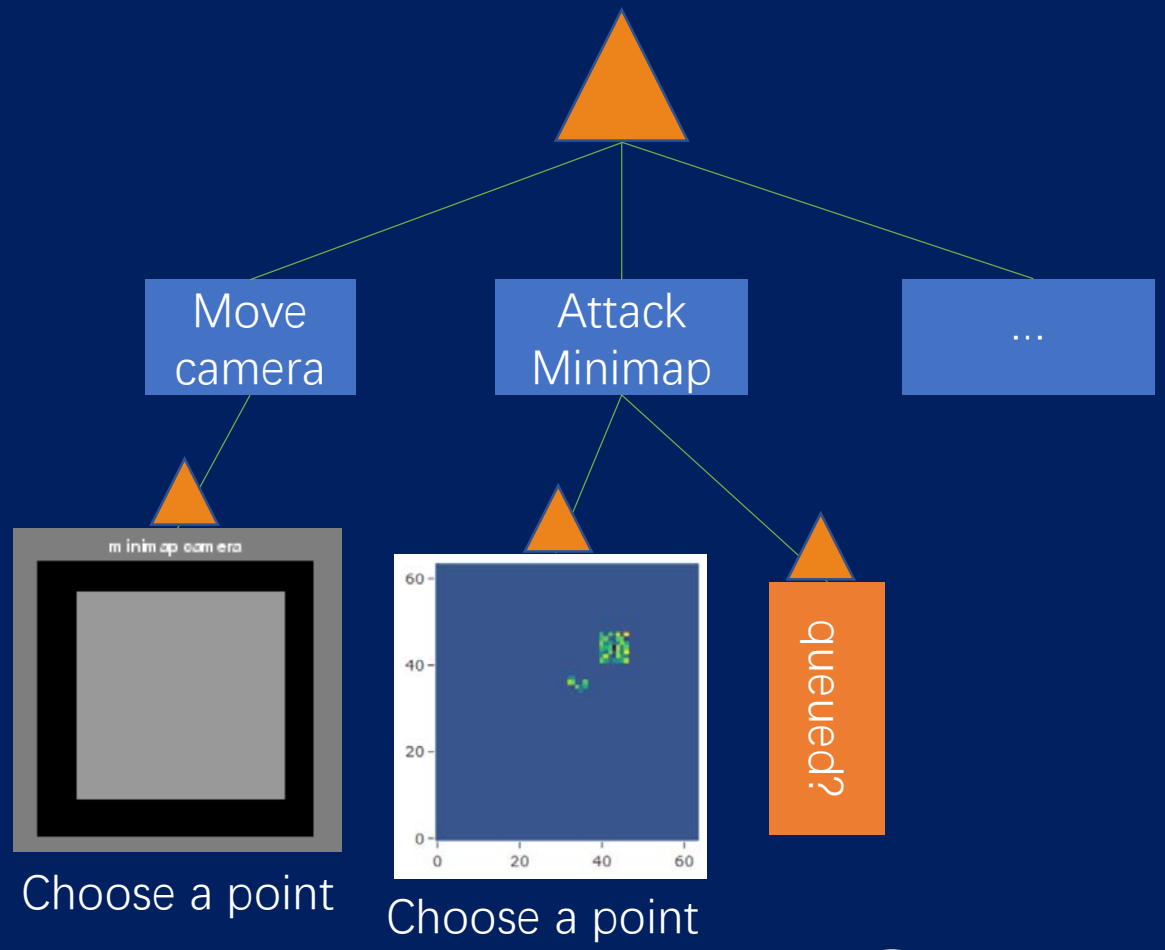
(TQGP) tree-leveled Q-value generating process

Pros:

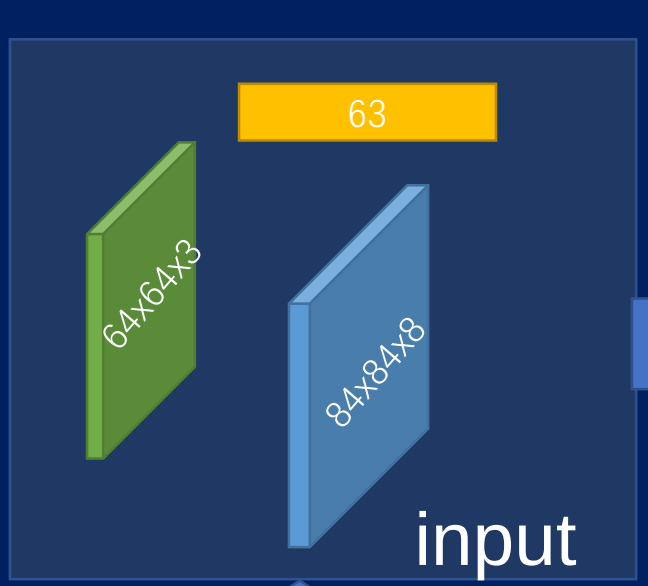
(1) Max only appear on average operator "layer" and top dummy layer, the time complexity reduce to $O(bd) \approx$ O(21,168).
(2) Encode prior knowledge of decision into tree architecture .
(3) Output of network is also $O(bd)$ now
In practice, 54,652 in our setting.

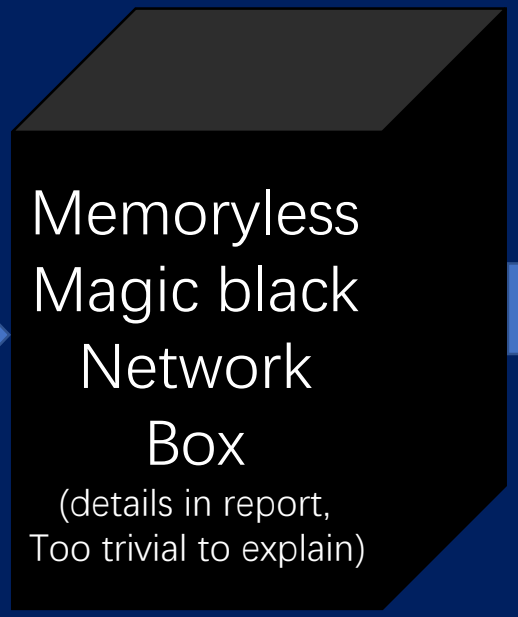
Motivations

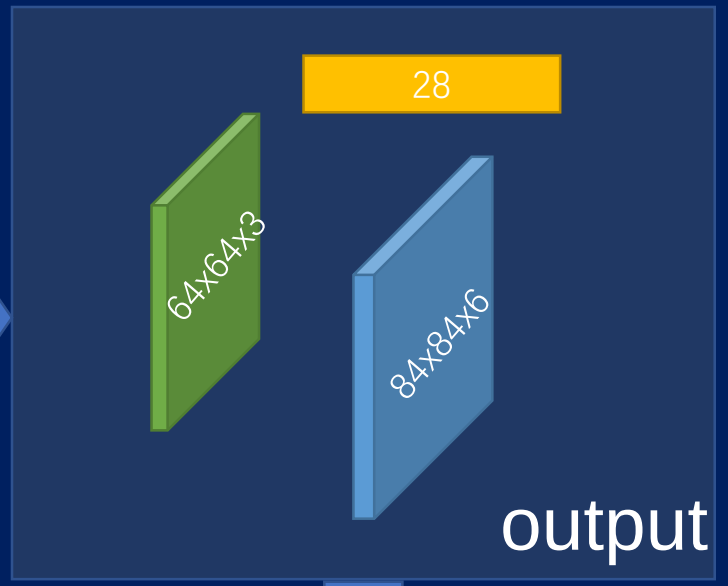
(1) min-max pruned
(2) decision tree



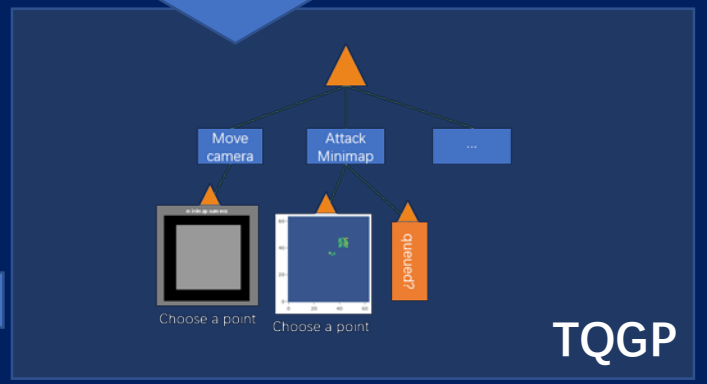
Network architecture







action



Experiment result

Network architecture

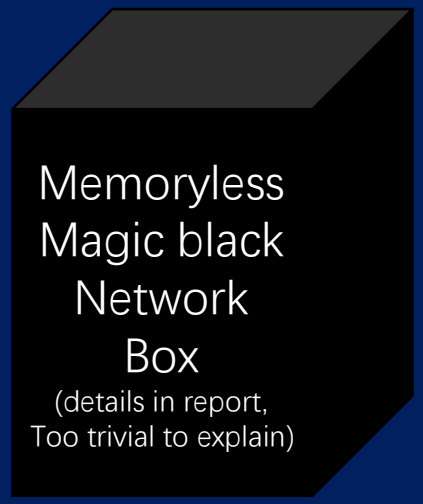
Remarks:

1. **Why memoryless?**

For this sc2 mini-task, the state sequence may be ~2000 long, classical method is impossible to compute(OOM).

One possible remedy is to hard code memory.(e.g.: output Q values by exponentially average the IV in TQGP)

Network architecture

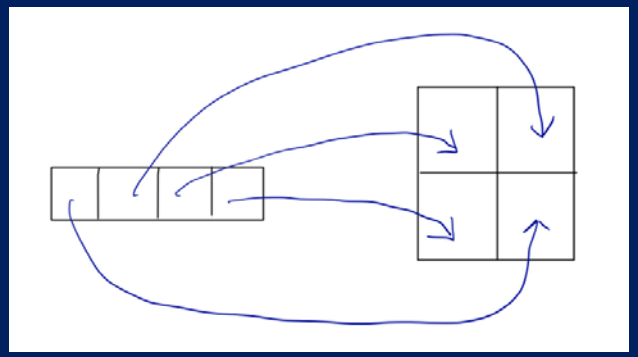


Remarks:

2. Any novelty?

One key problem is how to combine 1-D info(money, selected units) with 2-D info(minimap screen).

Popular method: reshape fc layer into convolutional layer.
Problem: fc does not contain spatial structure in architecture(require "learning" the structure). Huge amounts of parameter and learning cost.



Network architecture

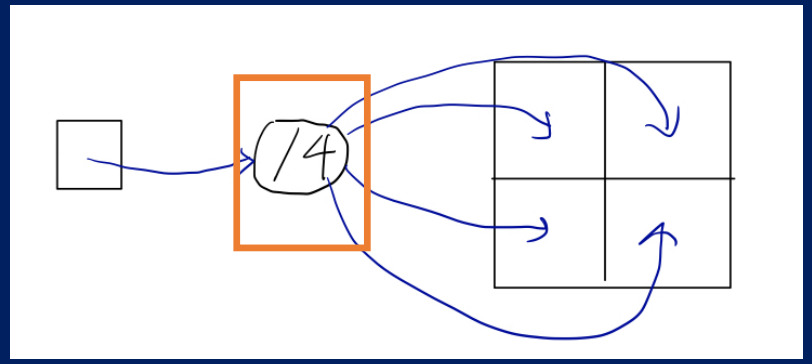
Memoryless
Magic black
Network
Box
(details in report,
Too trivial to explain)

Remarks:

2. Any novelty?

One key problem is how to combine 1-D info(money, selected units) with 2-D info(minimap screen).

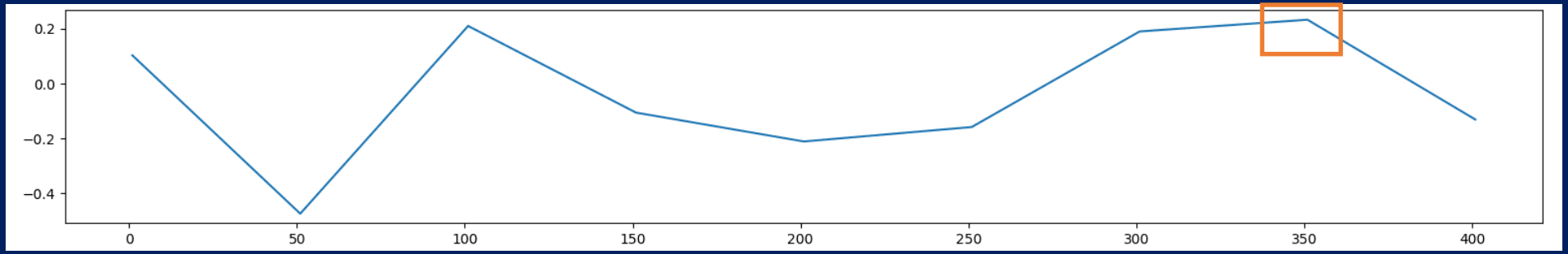
Our implementation: fork the fc feature into 2-D structure



Engineering trick: divided fc value by forked num. to avoid gradient explosion (Or NaN will come to find you!)

Experiment result

1. Training takes ~2 days, with ~400 epoxides(games) on GTX1080Ti
2. Speed: ~5fps
3. Compete with models with previous epoxides randomly(~20 matches), Pick the best model.



Experiment result

Ternary score: +1 win, -1 lost, 0 tie

RL vs. Bot(Easy in sc2)	RL vs. Random	RL vs. RL(pool)
-0.8	0.7	0.2

Hard-code robot takes better attack action:
attack one unit with consistency, where out robot(memoryless) cannot,
which determines the game.(Bright future for a model with memory ☺)

Summary

- Provide a PVP mini-game map basing on pysc2 API.
  - well-designed rewards
  - well-designed performance measurement metric
- Apply baseline DQN
  - self-game setting
  - Tricks1: prioritized Experience Replay
  - Tricks2: periodic epsilon-greedy(×)
- Handle the large action space
  - TQGP(tree-leveled q-value generating process)
  - Fork layer design details

Follow our project on GitHub!

AI-StarCraft-II