Machine Learning Engineer Nanodegree

Capstone Project

Ji Ma
August 23st, 2017

I. Definition

Project Overview

Recently, OpenAI's Artificial Intelligence beat the best player in popular strategy game on 1v1 match, stands for a oustanding milestone after the legend of Deepmind's Alpha Go. It stands for a new era of the AI powered bot in the game way more complex than Go. Also, what interested me so much is that Deepmind also released a brand new Reinforcement Learning Environment for the game StarCraft II, which is one of my favorate game. But the reality is I'm not very good at competing with human players in StarCraft II. Since I am a big fan of Deep Learning and Neural Network, I decided to use the Deep RL to implement the AI to achieve some RL tasks in this brand new handy environment by myself :)

Problem Statement

As declared in the paper along with the Learning Environment, the current best result worked by the most professional researchers from Deepmind could not beat a simple bot in Easy level. It takes so much work and computing powers to achieve the goals like that considering the complexity of the StarCraft II. So my goal should be realistic, tackle the Mini Game like MoveToBeacon and FindAndDefeatZerglings should be a good practice to get started.
Among all those mini games, I will specificly tackle **BuildMarines**, which described below according to pysc2 documents:

Description

A map with 12 SCVs, 1 Command Center, and 8 Mineral Fields. Rewards are earned by building Marines. This is accomplished by using SCVs to collect minerals, which are used to build Supply Depots and Barracks, which can then build Marines.

Initial State

12 SCVs beside the Command Center (unselected) 1 Command Center at a fixed location 8 Mineral Fields

at fixed locations Player Resources: 50 Minerals, 0 Vespene, 12/15 Supply ##### Rewards

Reward total is equal to the total number of Marines built

End Condition

Time elapsed

Time Limit

900 seconds

Additional Notes

- Fog of War disabled
- No camera movement required (single-screen)
- This is the only map in the set that explicitly limits the available actions of the units to disallow actions which are not pertinent to the goal of the map. Actions that are not required for building Marines have been removed.

Metrics

Reward total is equal to the total number of Marines built

II. Analysis

Data Exploration

The data I will first explore around is pysc2 itself.

According to it's source code:

Action space The action space is huge which is 524 actions for the regular game.

We could get valid actions dynamicly in each step

Status Space Depends on unit type, the status space could be enormous

1. screen*xy = 84 (the screen size will be screen*xy* screen_xy) we could change it later
2. unit*type*size = 1850 (The unit type size)
3. so, the status space would be screen*xy^2 * unit*type_size

In each Step in Deep RL, pysc2 provided a handy `TimeStep` class which will provide 4 major features for Deep RL:

- step_type: First, Mid, Last(useful to find the step progress in the episode)
- Reward
- Discount: the discound initialized in env
- Obervation: infomation contains
- screen(different type), unit_type screen will be used
- minimap
- etc

In order to know the performance of Random Agent and in the mean while get a sense of data generated in following aspects:

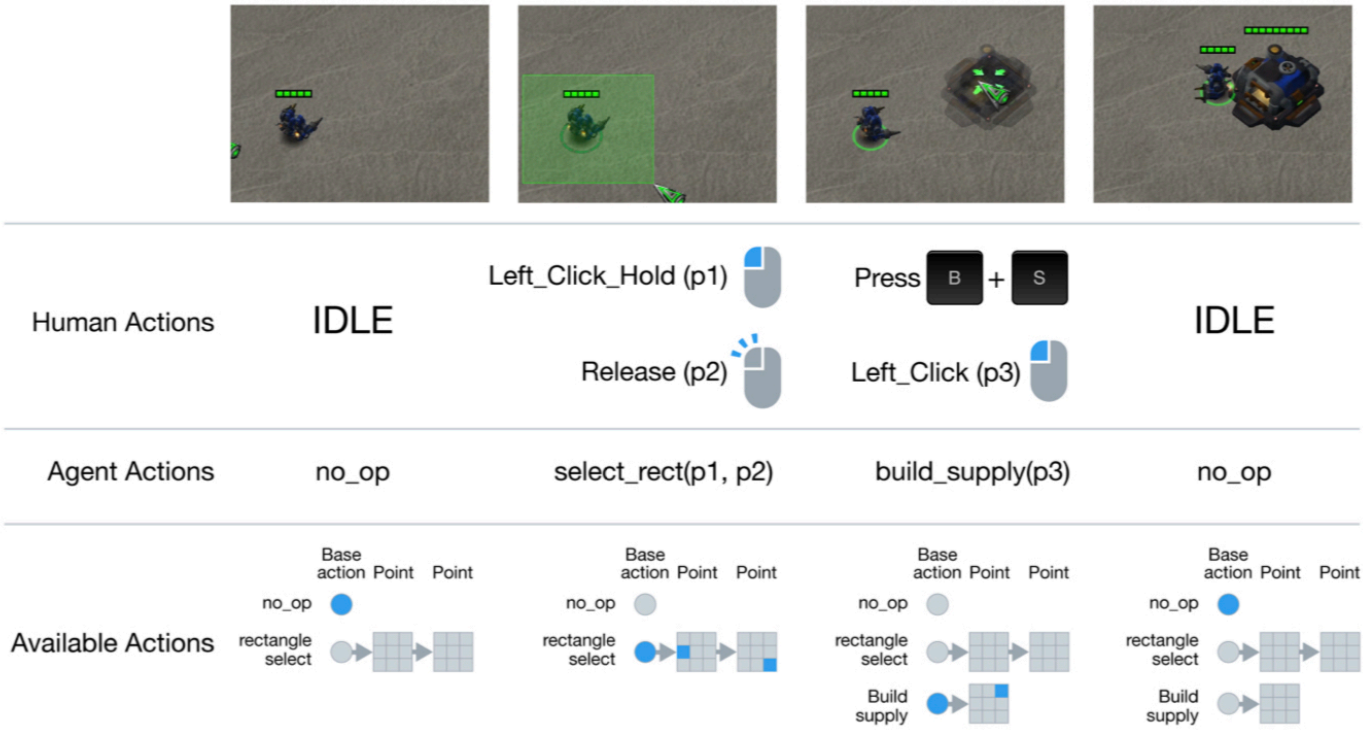
- screenList: the numpy array, will help me narrow down the unit type for minigame
- actionList: the numpy array, will help me narrow down the action space for minigame
- rList: the overall reward(performance matric)

Each of the Build Marine mini game has 15000 frames limit, and I used step_mul=8 as default(8 frames per action).
So, 15000 frames means 1875 steps per game.
I want 4000 games/episode sample data here, so I use 1875000 steps on 4 threads to collect data The result is similar to what has been described in the sc2le paper. Average 1 reward for random policy. Pretty bad.

Agent	Metric	MoveToBeacon	CollectMineralShards	FindAndDefeatZerglings	DefeatRoaches	DefeatZerglingsAndBanelings	CollectMineralsAndGas	BuildMarines
Random policy	mean	1	17	4	1	23	12	< 1
	max	6	35	19	46	118	750	5
Random search	mean	25	32	21	51	55	2318	8
	max	29	57	33	241	159	3940	46
DeepMind human player	mean	26	133	46	41	729	6880	138
	max	28	142	49	81	757	6952	142
StarCraft GrandMaster	mean	28	177	61	215	727	7566	133
	max	28	179	61	363	848	7566	133
Atari-net	best mean	25	96	49	101	81	3356	< 1
	max	33	131	59	351	352	3505	20
FullyConv	best mean	26	103	45	100	62	3978	3
	max	45	134	56	355	251	4130	42
FullyConv LSTM	best mean	26	104	44	98	96	3351	6
	max	35	137	57	373	444	3995	62

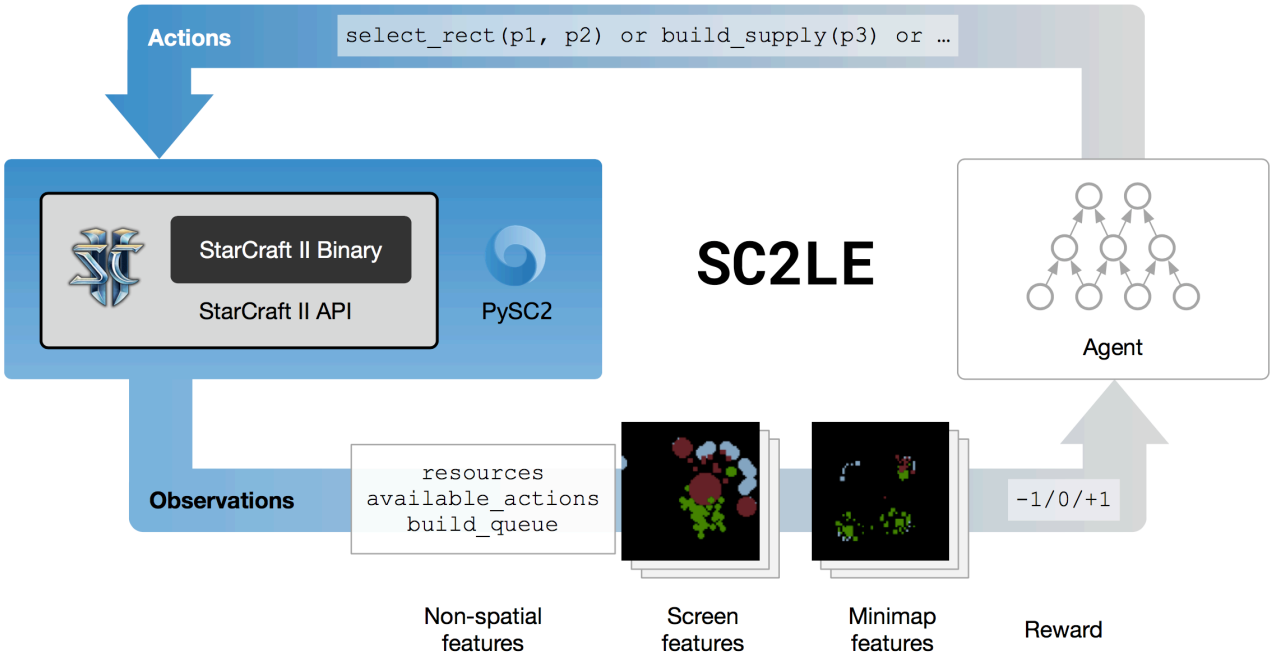
Exploratory Visualization

The input screen will be 64*64. Each entry will have a unit represenataion described above.

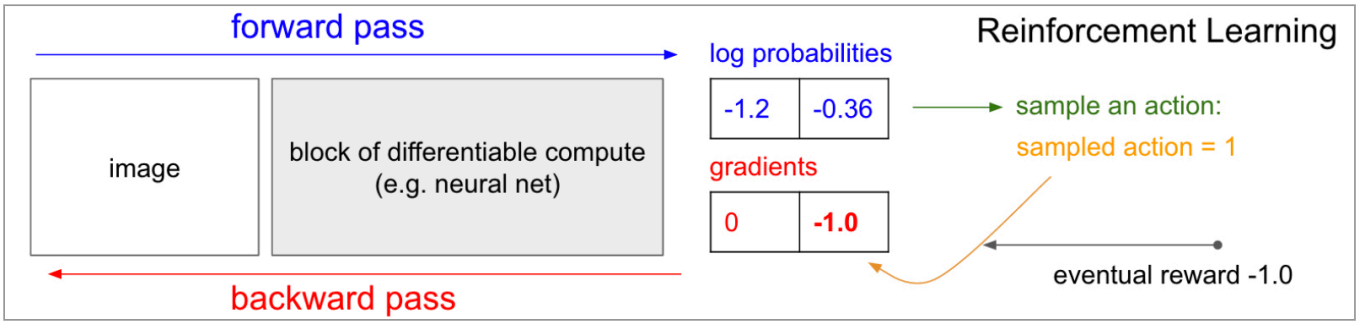


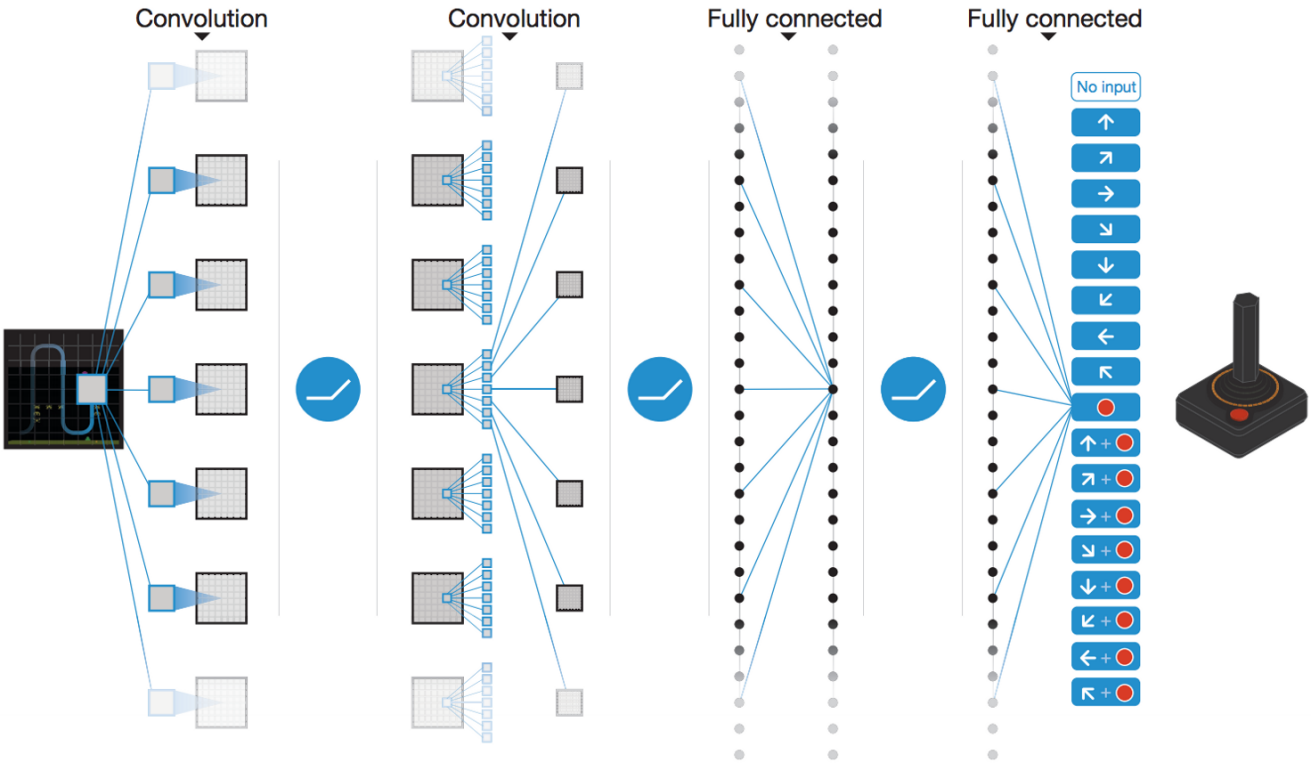
This is basically how the agent action would be executed.

Algorithms and Techniques



Deep Q Network





from Q learnig to DQN:

1. Going from a single-layer network to a multi-layer convolutional network.
2. Implementing Experience Replay, which will allow our network to train itself using stored memories from it's experience.
3. Utilizing a second "target" network, which we will use to compute target Q-values during our updates.

Bellman Equation:
$Q(s,a) = r + \gamma(\max(Q(s',a'))$

loss function
$Loss = \sum(Q\text{-target} - Q)^2$

Double DQN
$Q\text{-Target} = r + \gamma Q(s',\text{argmax}(Q(s',a,\Theta),\Theta'))$

Supervised Learning: learn both a **value function** (i.e., predicting the winner of the game from game observations), and a **policy** (i.e., predicting the action taken from game observations).

Benchmark

Random Agents

According to sc2le paper, they suggested two main random agents.
I will particularly use **Random Policy**
Random Policy will uniformly pick random action among all valid actions. (Samples in action space)

III. Methodology

Data Preprocessing

In walkthrough ipython notebook.

I derived the overall valid unit_type input and action input.

Valid unit type for Build Marines:

```
#unit frequency
[[       0 31386888]
 [      18  1517332]
 [      19  1073447]
 [      21   338735]
 [      45   636247]
 [      48     4367]
 [     341  1769464]]
```

Valid action type with ambiguous arguments for Build Marines:

```
{0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 10,
 11,
 12,
 13,
 42,
 91,
 264,
 269,
 274,
 331,
 332,
 333,
 334,
 343,
 344,
 451,
 452,
 453,
 477,
 490}
```

Not every single unit types/actions will be used in the mini games like this, so I reduced the space for both of them to reduce the training time.

Implementation

Deep Q-Learning Algorithm

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
    select an action a
        with probability ϵ select a random action
        otherwise select a = argmaxa’Q(s,a’)
    carry out action a
    observe reward r and new state s’
    store experience <s, a, r, s’> in replay memory D

    sample random transitions <ss, aa, rr, ss’> from replay memory D
    calculate target for each minibatch transition
        if ss’ is terminal state then tt = rr
        otherwise tt = rr + γmaxa’Q(ss’, aa’)
    train the Q network using (tt - Q(ss, aa))^2 as loss

    s = s'
until terminated
```

The conv2d layers are

- num*outputs=32, kernel*size=8, stride=4
- num*outputs=64, kernel*size=4, stride=2
- num*outputs=64, kernel*size=3, stride=1
- num*outputs=512, kernel*size=4, stride=1

for the deep q network's convolutional layers.

Refinement

I found that if we just use random agent as the experience buffer, it is very difficult to get the agent to get a reward higher than 1. So the learning process is going to be extremely long if we only utilized the random agent for experience buffer.

So, I gave a little customized push for the random agent:

if the barrack(the building that we need to build the marine) is available:

- if the barrack is selected in the screen: Build marines :)

- if barrack is not selected yet: Select all barracks

Here is the code for how I refined the random agent.

```python
avail_actions = timestep.observation["available_actions"]
                    # Choose an action by greedily (with e chance of random action)
                    if np.random.rand(1) < e or total_steps < pre_train_steps:
                        # Random actions
                        # if timestep.observation["single_select"]:
                        if _TERRAN_BARRACKS in s:
                            # The screen has the _TERRAN_BARRACKS
                            if not isBarracksSelected(timestep.observation):
                                # not selected yet.
                                location = getTerrainBarracksLocation(timestep.obser
                                # y goes first, and then x
                                unit_y, unit_x = (
                                timestep.observation["screen"][_UNIT_TYPE_VIEW] == _
                                target = [int(unit_x.mean()), int(unit_y.mean())]
                                a = _SELECT_POINT
                                args = [[_ALL_OF_TYPE_SELECT_POINT], target]
                            else:
                                # selected, good -> build marines
                                if _BUILD_MARINE in avail_actions:
                                    a = _BUILD_MARINE
                                    args = [[np.random.randint(0, size) for size in
                                             for arg in action_spec.functions[a].args
                                else:
                                    a = np.random.choice(avail_actions)
                                    args = [[np.random.randint(0, size) for size in
                                             for arg in action_spec.functions[a].args
```

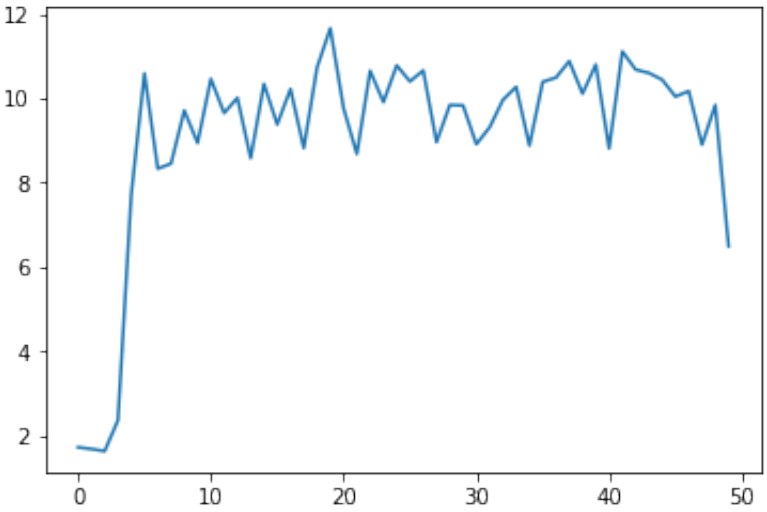
IV. Results

Model Evaluation and Validation

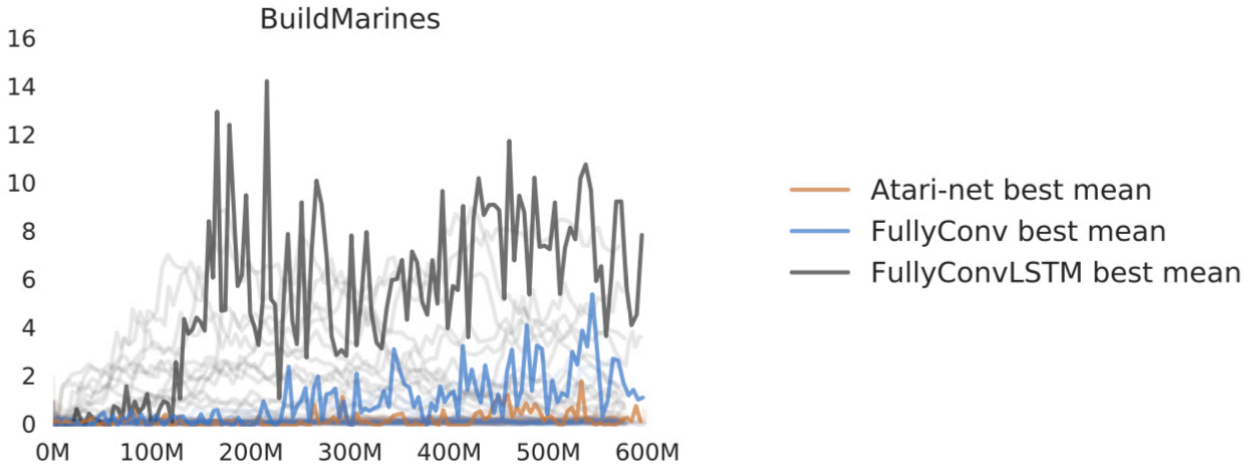
Since the major input for this game environment is the screen pixels input, a model with convolutional networks would be fit for dealing with the visual input like this. The deep q learning technique combines the beauty of q learning which is a classic reinforcement learning solution and ideas of the convolutional network, so Deep Q Learning is a good fit for the mini game like this.

Justification

The final result is definitely beating the previous random agent which only got around 1 average reward over time. The Deep Q Network has increased it's reward from nearly 0 to around average 10 reward per episode. This is far beyond the result from our previous random agent benchmark which only has average 1 reward.



Plus, when we compare our approach to this problem to the solution provided by sc2le paper, our result is pretty close to their performance and even much more stable:



V. Conclusion

Free-Form Visualization

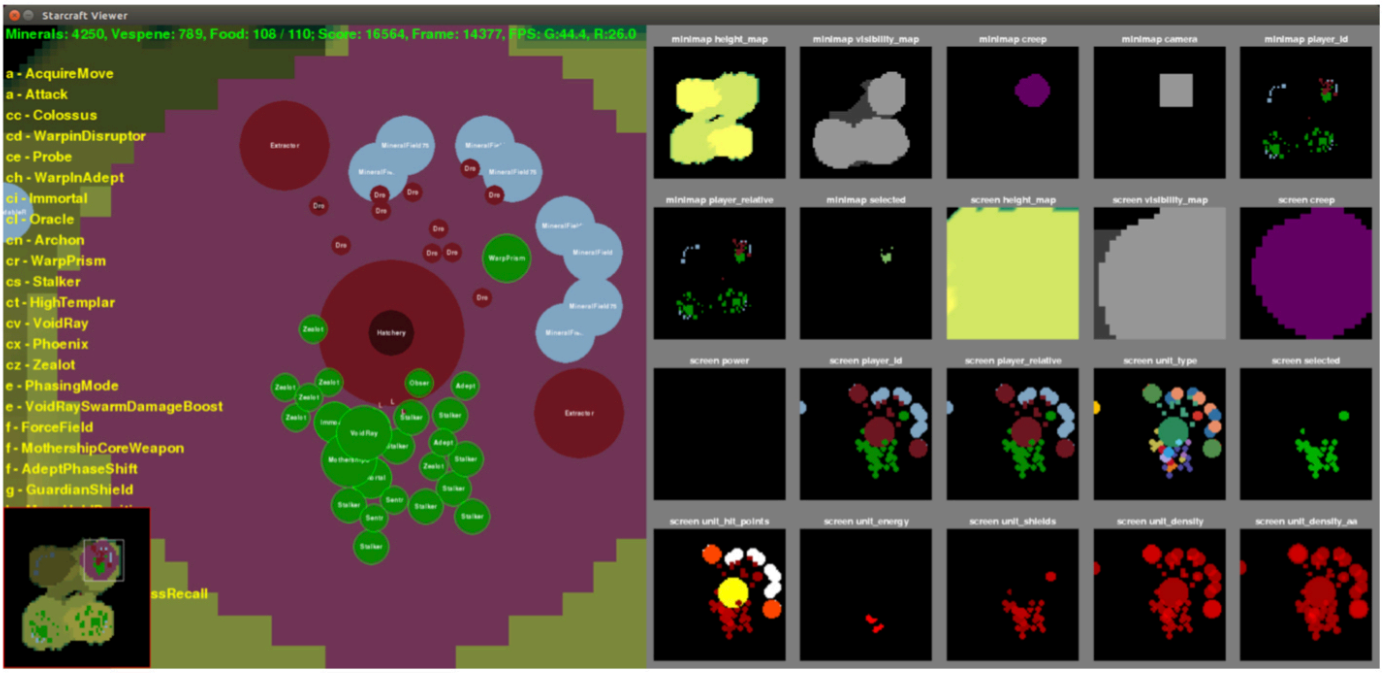


Figure 2: The PySC2 viewer shows a human interpretable view of the game on the left, and coloured versions of the feature layers on the right. For example, terrain height, fog-of-war, creep, camera location, and player identity, are shown in the top row of feature layers. A video can be found at `https://youtu.be/-fKUyT14G-8`.

Reflection

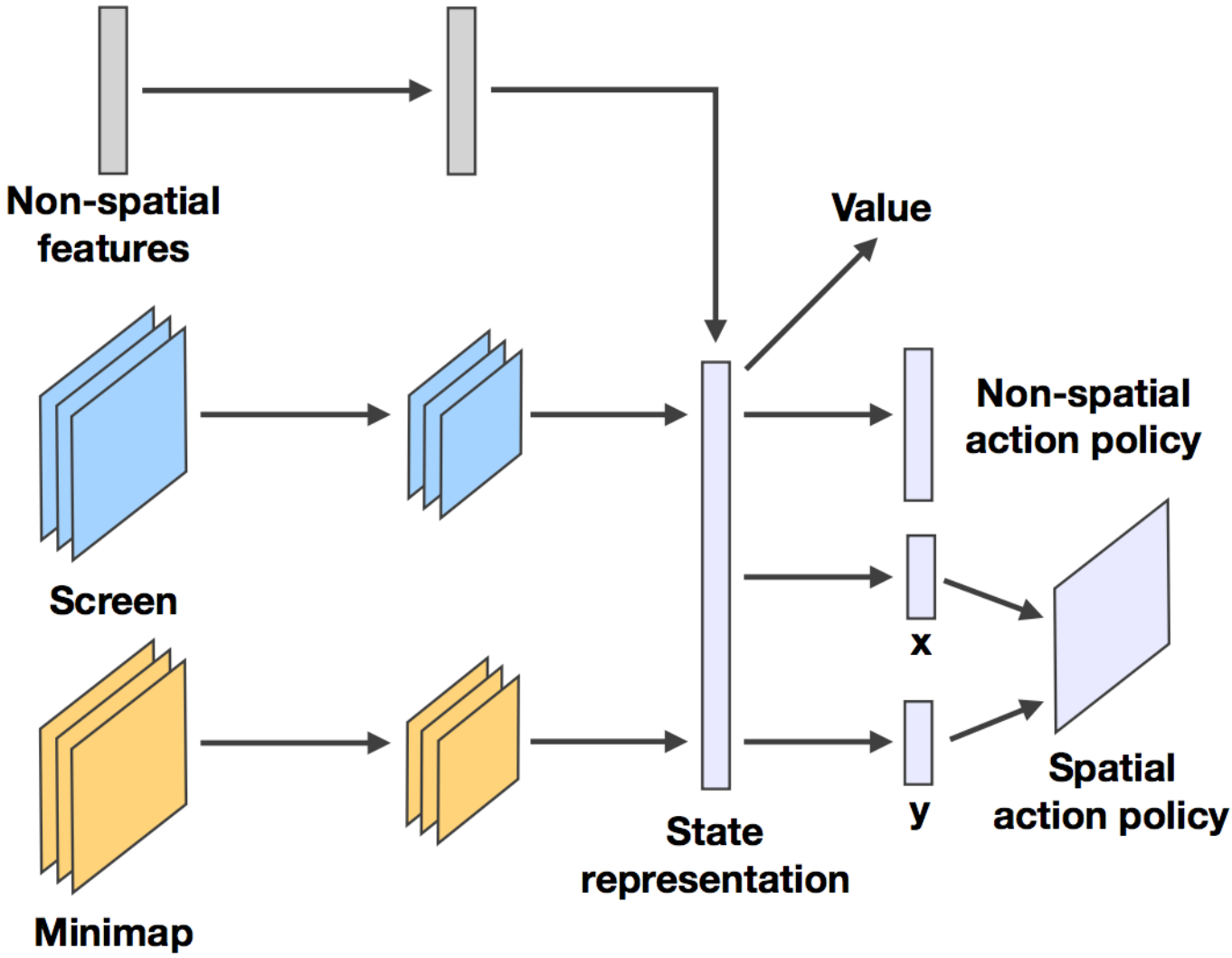
Doing this project is not a easy job at all. This Starcraft reinforcement learning environment just released in around 2 months. When I started to work on this project 2 months ago, I though it is going to be easy, however I was WRONG. DeepMind did NOT provide a well written API documentation at all for the learning environment. So, what I need to do is dive into the source code and figure out what exactly happened and make annotation by myself. And some of the discovery process(majorly debugging) process is painful and time consuming. I literally got stuck on a function related to the unit selection function just because the return value from this function are having a y,x order instead of x,y... Gosh, hope somebody write down a document for this learning environment.

Trying the new “toy” out there is hard, I tried to reached out Udacity mentor but they neither have any experience with this before. And the tutorial online is very limited. But finally, I made it. :)

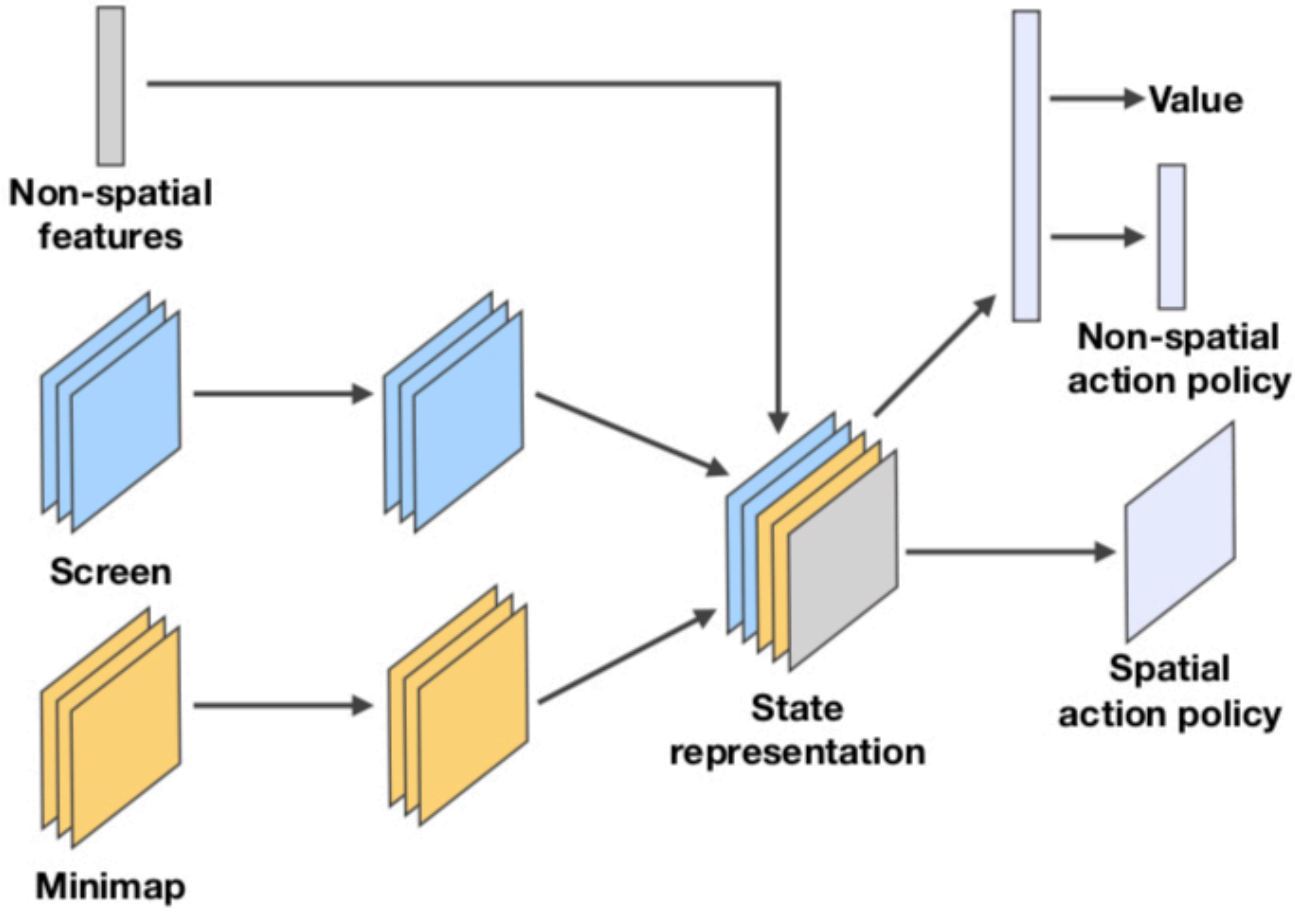
Improvement

I may try to improve the performance by adding the batch_normalization for each of the conv2d layer. Also, it would be beneficial to feed the network with more inputs like more feature layers(minimaps, height map, etc) and nonspacial information tensors(health, gas, money) provided in sc2. In sc2le paper, they implemented this instead of only one screen layer I used in my project. I assume a proper tweak from this implementation will generate a better result.

Atari-Net and Fully Conv



(a) Atari-net



(b) FullyConv

References

github.com/Blizzard/s2client-proto
github.com/deepmind/pysc2

Simple Reinforcement Learning with Tensorflow series by Arthur Juliani
Deep Reinforcement Learning: Pong from Pixels by Andrej Karpathy http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/

https://hackernoon.com/deep-learning-cnns-in-tensorflow-with-gpus-cba6efe0acc2

https://chatbotslife.com/building-a-basic-pysc2-agent-b109cde1477c