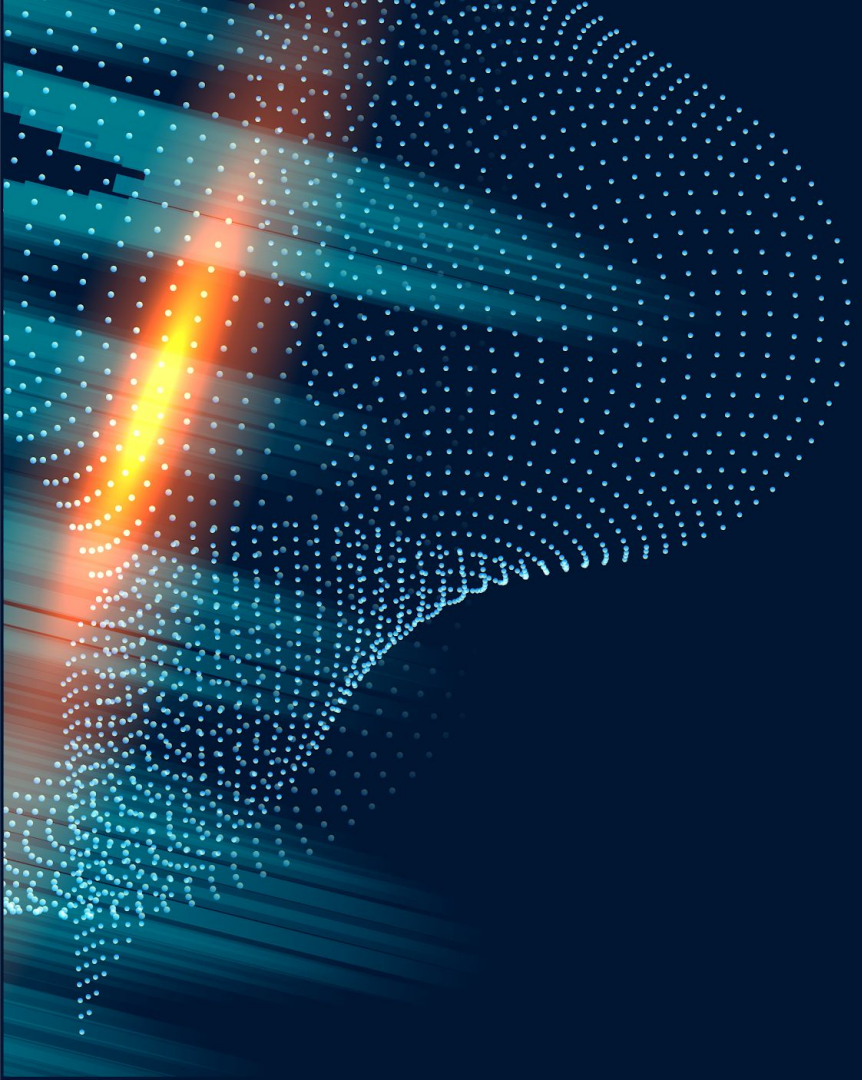
Artificial Intelligence project

Cinvestav Guadalajara

Carlos Cardenas
Emilio Tonix
Julia Abud



Part 6 - Final

Reinforcement learning
+ Neural networks (pytorch)

Simple64

TASKS

13 April 2021 - 30 April 2021

- Implement
  - Reinforcement Learning
  - Neural Network using pytorch
  - Simple 64 playing versus:
    - Random agent
    - Blizzard AI

Map: Simple64

Description

This map is consist in a map medium map size with four camps that contains minerals and vespene gas.

This map is for two players that try to destroy each other.

Initial State

- 1 command center
- 12 SCV

Rewards

- Win and keep your live

End Conditions

- Destroy the enemy

Time Limit

- No

Additional Notes

- We manually activated fog of war
- And visualize features

There were two general training approaches following the implementation of the Reinforcement Learning algorithm in pytorch:

- An agent was generated with the ability to learn how to compete against a **random-agent** (an agent that randomly selects what to do from a more limited range of actions).

- The agent was then put to the test against **StarCraft's AI in easy mode.**

Introduction

For this stage we keep using reinforcement learning using pytorch, but now the network should be in a more complex environment, where it has to deal with exploring the map, fog of war, allies items, and enemy known items.

The input network data is given by the pysc2 framework. It has the agent status of each unit and visible field, the output take actions to do task related with certain dependencies.

0	total score
1	idle_production_time
2	idle_worker_time
3	total value of units
4	total value structures
5	killed value units
6	killed value structures
7	collected minerals
8	collected vespene
9	collection rate minerals
10	collection rate vespene
11	spent minerals
12	spent vespene

Introduction

In general it was used the RELU function as activation function, for each layer, in total there are implemented 3 layers Input,Hidden and Output. Input vary on the implementation of agent, hidden is 64 long and finally output depends on implementation. The random factor epsilon is .1, and will decrease over time. The learning rate was constant at .33. Finally we save the trained weights every 100 matches.

```
class nnq():
    def __init__(self, input_dims, n_actions, lr, gamma=0.80, epsilon=.1, eps_dec=1e-5, eps_min=0.01):
        self.lr=lr
        self.input_dims = input_dims
        self.n_actions = n_actions
        self.gamma   = gamma
        self.epsilon = epsilon
        self.eps_dec = eps_dec
        self.eps_min = eps_min
        self.action_space = [i for i in range(self.n_actions)]
        self.Q = LinearDeepQNetwork(self.lr,self.n_actions,self.input_dims)
```

```
class LinearDeepQNetwork(nn.Module):
    def __init__(self, lr, n_actions, input_dims):
        super(LinearDeepQNetwork,self).__init__()

        self.fc1 = nn.Linear(input_dims, 64)
        self.fc2 = nn.Linear(64, n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.loss    = nn.MSELoss()
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu')

        self.to(self.device)

    def forward(self,state):
        layer1  = F.relu(self.fc1(state))
        actions = self.fc2(layer1)

        return actions
```

VS Random agent

The neuron inputs and outputs are shown in the table below, note that all data is normalized considering max possible size for item. The NN is 21 input neurons that are read from the pysc2 framework, then 6 output neurons which represent the actions taken by the army.

VS Random agent

NET

```
class NNAgent(Agent):
  def __init__(self):
    super(NNAgent, self).__init__()
    self.NN_net = nnq(21,6,0.33) # 21 data in , 6 actions
    self.new_game()
```

Normalized Data

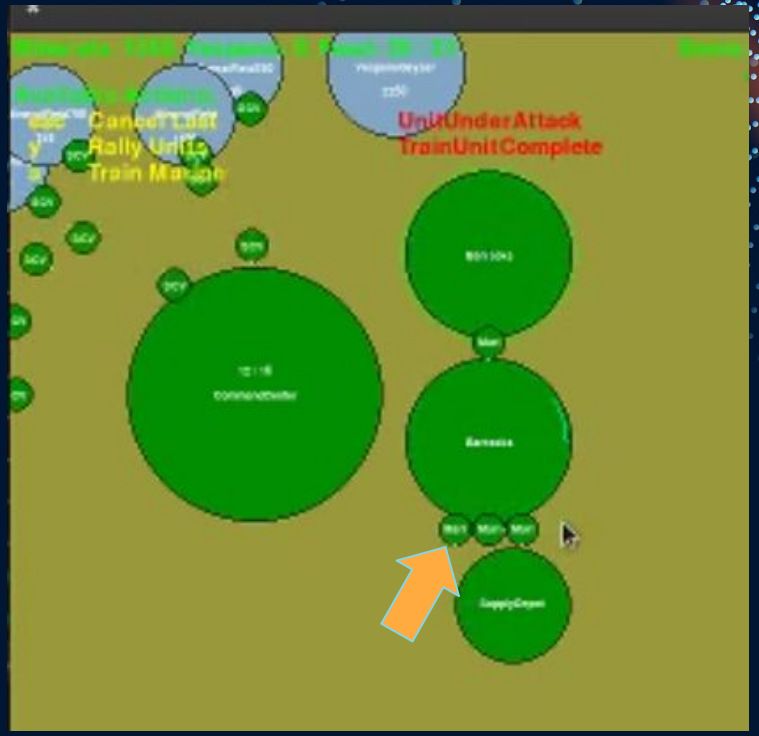
```
#data, normalized
return (len(command_centers)/COMANDCENTERS, #0
        len(scvs)/SCVS_NUM,  #1
        len(idle_scvs)/SCVS_NUM, #2
        len(supply_depots)/BUILDINGS, #3
        len(completed_supply_depots)/BUILDINGS, #4
        len(barrackses)/BUILDINGS, #5
        len(completed_barrackses)/BUILDINGS, #6
        len(marines)/MARINES_NUM, #7
        queued_marines/MARINES_NUM, #8
        free_supply/SUPPLY, #9
        can_afford_supply_depot, #10
        can_afford_barracks, #11
        can_afford_marine, #12
        len(enemy_command_centers)/COMANDCENTERS, #13
        len(enemy_scvs)/SCVS_NUM, #14
        len(enemy_idle_scvs)/SCVS_NUM, #15
        len(enemy_supply_depots)/BUILDINGS, #16
        len(enemy_completed_supply_depots)/BUILDINGS, #17
        len(enemy_barrackses)/BUILDINGS, #18
        len(enemy_completed_barrackses)/BUILDINGS, #19
        len(enemy_marines)/MARINES_NUM) #Se regresan todos nuestros valores necesarios. #20
```

Output

```
class Agent(base_agent.BaseAgent):
  actions = ("do_nothing",
             "harvest_minerals",
             "build_supply_depot",
             "build_barracks",
             "train_marine",
             "attack") #generación
```

VS Random agent

The neuronal network will fight against a marine team with the same possible actions as it, but the difference is that the enemy is a **random agent**, at first sight it must get over it, but it takes **1000 trainings** to almost win every match.



At **4000 it changes it's strategy**, **it waits** to produce more marines and send them in groups to upgrade performance, once it reached that point the agent stuck in the same winning point and strategy.

VS Random agent

The reward was given by data input gather from **pysc2 framework,** every time machine has to decide, the feedback was given by weighted criteria, it was compared the observations states, actual state versus previous one.If enemy increases their units or items **rewards is [-1], if all stills the same[0], and if the marines kill enemies or have more resources[1].**

```
def update_reward(self,prev_obs, current_obs):

  #NEGATIVE REWARD
  if  current_obs[ITEM.MARINES_LEN] < prev_obs[ITEM.MARINES_LEN] :
    self.ai_reward-=1
  if current_obs[ITEM.SCVS_LEN] < prev_obs[ITEM.SCVS_LEN] :
    self.ai_reward-=1
  if current_obs[ITEM.ENEMY_MARINES_LEN] > prev_obs[ITEM.ENEMY_MARINES_LEN]:
    self.ai_reward-=1
  if current_obs[ITEM.BARRACKSES_LEN] < prev_obs[ITEM.BARRACKSES_LEN]:
    self.ai_reward-=1
  if current_obs[ITEM.ENEMY_COMPLETED_SUPPLY_DEPOTS_LEN] > prev_obs[ITEM.ENEMY_COMPLETED_SUPPLY_DEPOTS_LEN]:
    self.ai_reward-=1
  if current_obs[ITEM.IDLE_SCVS_LEN] > prev_obs[ITEM.IDLE_SCVS_LEN]:
    self.ai_reward-=1
```

```
#POSITIVE REWARDS

if current_obs[ITEM.CAN_AFFORD_MARINE] or current_obs[ITEM.QUEUED_MARINES]:
  self.ai_reward+=1
if current_obs[ITEM.SUPPLY_DEPOTS_LEN] > prev_obs[ITEM.SUPPLY_DEPOTS_LEN]:
  self.ai_reward+=1
if current_obs[ITEM.BARRACKSES_LEN] > prev_obs[ITEM.BARRACKSES_LEN]:
  self.ai_reward+=1
if current_obs[ITEM.MARINES_LEN] > prev_obs[ITEM.MARINES_LEN]:
  self.ai_reward+=1
if current_obs[ITEM.ENEMY_MARINES_LEN] <  prev_obs[ITEM.ENEMY_MARINES_LEN] :
  self.ai_reward+=1
if current_obs[ITEM.ENEMY_BARRACKSES_LEN] < prev_obs[ITEM.ENEMY_BARRACKSES_LEN]  :
  self.ai_reward+=1
if current_obs[ITEM.ENEMY_SCVS_LEN] < prev_obs[ITEM.ENEMY_SCVS_LEN]:
  self.ai_reward+=1
```


Applications
dom, abr 25 8:18 PM
Simple64_agent.py - workspace (Workspace) - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
sc2_env.py
Simple64_agent.py M
inteligencia.py M
csvFileData.py
MachineLearning_Project > Minigames > FinalAgent > Simple64_agent.py
WORKSPACE (WORKSP...
FinalAgent
Project
.gitignore
TODOS: TREE (11)
pysc2
MachineLearn...
OUTLINE
TIMELINE
StarCraft Viewer
HoldPosition
Move
Patrol
Stop
Sending ResponseJoinGame
Sending ResponseJoinGame
19425 20:16:05.020960 140546787112768 sc2_env.py:314] Environment is ready
19425 20:16:05.038172 140546787112768 sc2_env.py:507] Starting episode 1: [terran, terran] on Simple64
Select Python Interpreter
Ln 517, Col 1
Spaces: 2
UTF-8
LF
Python
Go Live

VS Blizzard AI

Blizzard AI is pretty good playing, it has different levels. However **medium is really hard,** so it was thought to firstly analyse game strategy in **easy**, it was observed that game uses **other resources** and items that we was not using, so we decided to l**evel up the complexity** of the network.

It was added to NN other factors, such as **Vespene gas** collection, and **marauders** characters, which are stronger and bigger, **laboratories.**

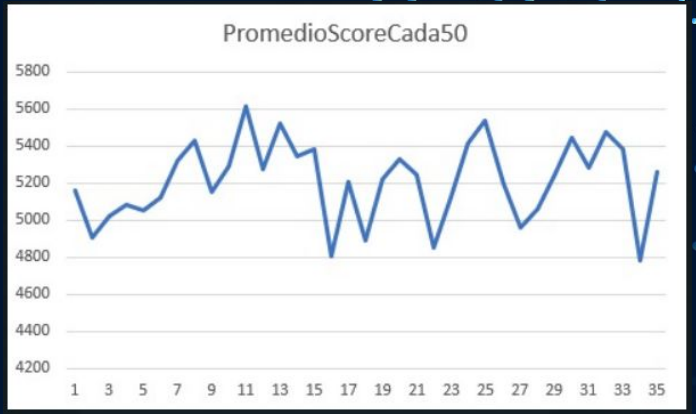
```
class Difficulty(enum.IntEnum):
  """Bot difficulties."""
  very_easy = sc_pb.VeryEasy
  easy = sc_pb.Easy
  medium = sc_pb.Medium
  medium_hard = sc_pb.MediumHard
  hard = sc_pb.Hard
  harder = sc_pb.Harder
  very_hard = sc_pb.VeryHard
  cheat_vision = sc_pb.CheatVision
  cheat_money = sc_pb.CheatMoney
  cheat_insane = sc_pb.CheatInsane
```

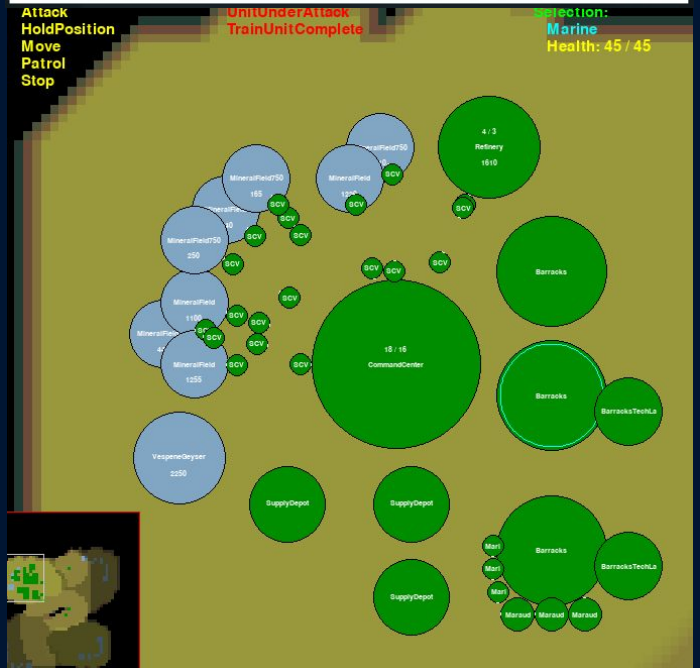
```
len(enemy_marauder)/20,
len(enemy_TechLab)/10,
len(marauders)/10,
len(TechLabs)/3) #Se re
```

```
class Agent(base_agent.BaseAgent):
  actions = ("do_nothing",
             "harvest_minerals",
             "build_supply_depot",
             "build_barracks",
             "train_marine",
             "attack") #generación
```

VS Blizzard AI

As final result our agent didn't won vs **easy**, it does against **very_easy**, however between some matches it does a higher score.





Conclusion

There were several factors and points that can be concluded from what was done in this project:

- The full decision tree with all the possible actions would be needed for the agent to be able to beat the AI from the game. In this project the actions were limited to some of those of the Terran race.
- The more actions available, the slower the decision making is, thus lowering the efficiency of the agent.
- When the list of actions is expanded, the quantity of possible strategies grows exponentially.
- The match tends to last longer when the agent makes good decisions and by consequence the training also takes more time.

The training time is an important factor because sometimes it is impossible to recognize if our agent is learning as expected until several hundred episodes have been played. This means that we need to take into consideration the amount of time consumed by the training, as we have limited time and resources to develop this project.