DSTI project

Artificial neural network and introduction to Deep Learning

OPTION B

Creation of an Artificial Intelligence for the game Starcraft 2

June 2019

Author: Alexandre GENETTE

DSTI – Cohort Autumn 2018

INTRODUCTION

With this project I will study how to implement an AI for the game Starcraft 2. Deepmind, after the success of AlphaGo and AlphaZero succeeded to defeat human pro player at the game Starcraft 2 with their Ai named AlphaStar.

What is Starcraft 2 ?

Starcraft 2 is a Real Time Strategy Game (RTS) and it is considered as one of the most complete and complex videogames. Since the first edition in 1996 professional competitions exist all over the world.

The complexity from the AI point of view comes from the fact that not all informations are available to the AI, unlike Go or Chess where all players have the same information.

The AI Alphastar that defeated pro Player at Starcraft 2 was forced to mimic human limitation in terms of action per minute and time of reaction where computer has clearly a serious advantage over human being.

DEEP LEARNING ARCHITECTURES

Learning Environment:

Before coding any algorithm in Python we need a learning environment in order to make python and starcraft2 communicating together. This is made possible because of the **Pysc2** learning environment.

- **Pysc2** was developed by DeepMind with the collaboration of Blizzard (SC2 editor). PySC2 is an interface for some agents to interact with the videogame Starcraft 2 in order to get observations and to send actions. Having this kind of learning environment is mandatory in ordet to have full implementation.
  Source : https://github.com/deepmind/pysc2
  Command: pip install pysc2

- **Sc2** is an other learning environment**,** simpler than pysc2**.**
  https://github.com/Dentosal/python-sc2/
  command line: pip install sc2

- **Starcraft 2** must also be installed on the computer**.** There exists a free version of the game (this is the one I have on my PC)

- **Python** of course with **tensorflow/keras** packages wil be used of course.

Now we have set up our environment we can go further.

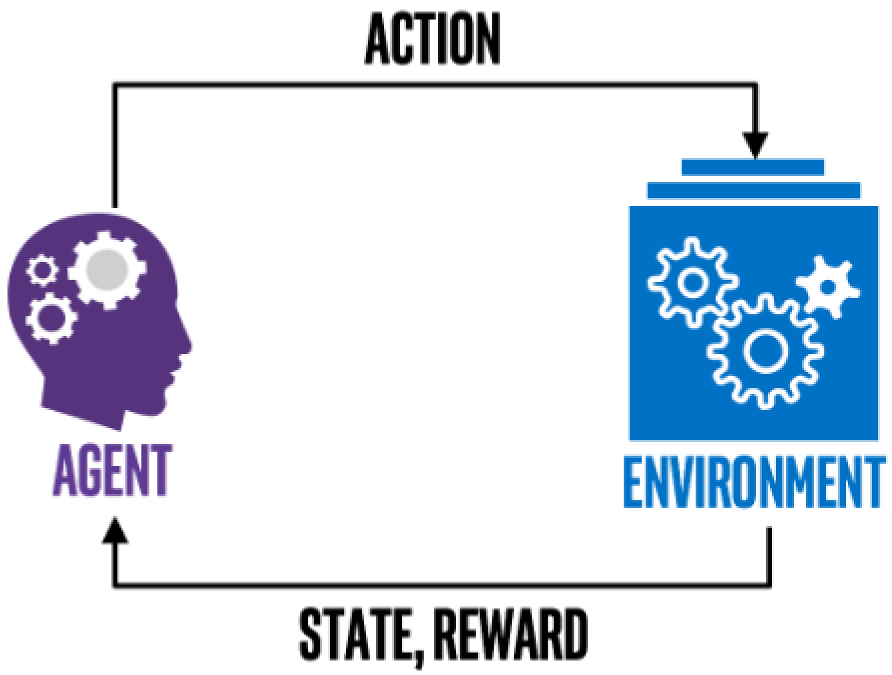
Deep Learning Architecture:

There exist several ways to create an AI for a game. Back in 90s Kasparov was defeated by Deepblue (IBM) using an **alpha-beta search** algorithm which is nothing alike a deep learning algorithm.

One of the most popular algorithm for this kind of task is called the **deep reainforcement learning** or **Q-Learning**. This method is powerful because it is an unsupervised method. The AI will learn from scratch with just basic rules and rewards system.

$$\underbrace{NewQ(s,a)}_{\text{New Q-Value}} = \underbrace{Q(s,a)}_{\text{Current Q-Value}} + \underbrace{\alpha}_{\text{Learning rate}}[\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{max\, Q'(s',a')}_{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)]$$

It can be graphically translated by :



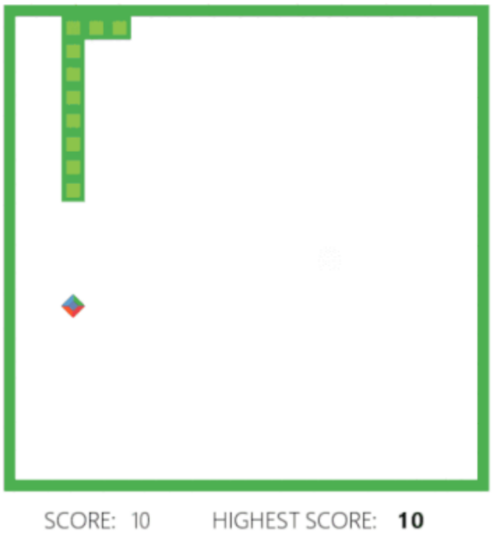
Iteratively, an agent does actions in its environment if it is a good choice it gets a reward and the neural network improves and then the agent will keep learning iterations after iterations.
The major drawback is that it requires a lot of power in terms of Memory and CPU/GPU.

> **For example,** you give the rules of chess to an agent, the environment is the chessboard with the chess pieces and the reward could be, win = 1 and loss = 0. (Very simple example)

For the reinforcement learning we can use what is called ***recurrent neural network*** (abbreviated in RNN).

Snake Example:

The snake is a famous videogame (even before the implementation in the Nokia 3310 😉)



We can train a **deep reinforcement learning agent** in order to make it better at the game with simple code.

I ran this one on my computer and after 150 iteration the agent was far better at the game than at the beginning.

Full code avalaible at the end ANNEXE 1

CORE CODE:

```
def run():
    pygame.init()
    agent = DQNAgent()
    counter_games = 0
    score_plot = []
    counter_plot =[]
    record = 0
    while counter_games < 150:
        # Initialize classes
        game = Game(440, 440)
        player1 = game.player
        food1 = game.food

        # Perform first move
        initialize_game(player1, game, food1, agent)
        if display_option:
            display(player1, food1, game, record)

        while not game.crash:
            #agent.epsilon is set to give randomness to actions
            agent.epsilon = 80 - counter_games

            #get old state
            state_old = agent.get_state(game, player1, food1)

            #perform random actions based on agent.epsilon, or choose the action
            if randint(0, 200) < agent.epsilon:
                final_move = to_categorical(randint(0, 2), num_classes=3)
            else:
                # predict action based on the old state
```

```
                prediction = agent.model.predict(state_old.reshape((1,11)))
                final_move = to_categorical(np.argmax(prediction[0]), num_classes=3)

            #perform new move and get new state
            player1.do_move(final_move, player1.x, player1.y, game, food1, agent)
            state_new = agent.get_state(game, player1, food1)

            #set treward for the new state
            reward = agent.set_reward(player1, game.crash)

            #train short memory base on the new action and state
            agent.train_short_memory(state_old, final_move, reward, state_new, game.crash)

            # store the new data into a long term memory
            agent.remember(state_old, final_move, reward, state_new, game.crash)
            record = get_record(game.score, record)
            if display_option:
                display(player1, food1, game, record)
                pygame.time.wait(speed)

        agent.replay_new(agent.memory)
        counter_games += 1
        print('Game', counter_games, '      Score:', game.score)
        score_plot.append(game.score)
        counter_plot.append(counter_games)
    agent.model.save_weights('weights.hdf5')
    plot_seaborn(counter_plot, score_plot)


run()
```

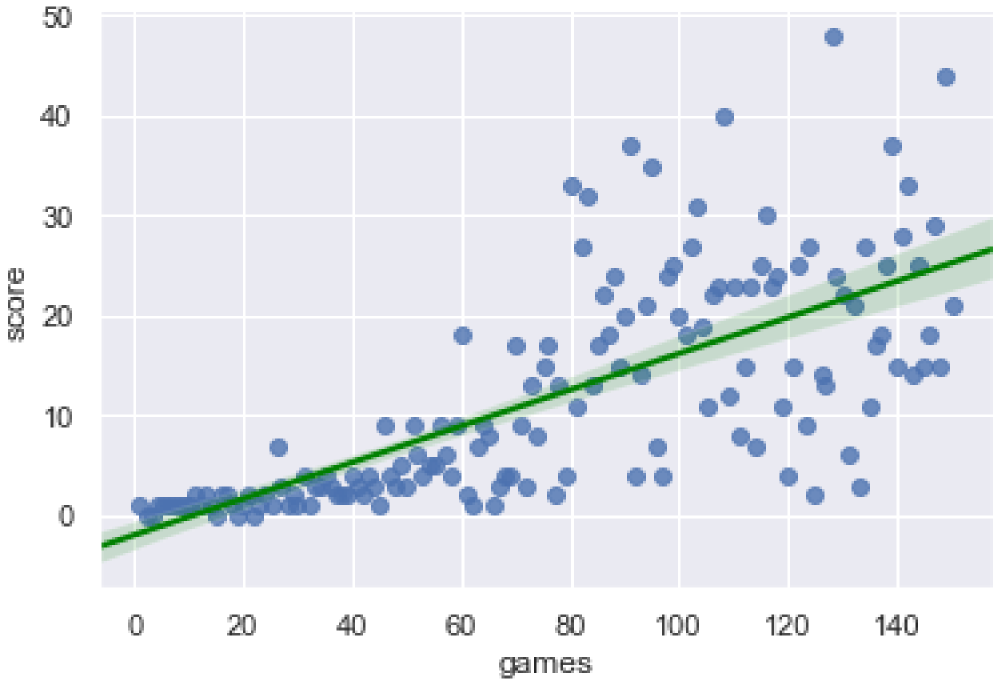
The first iterations the snake does not catch food very efficiently:
1 Game = 1 Iteration

```
WARNING:tensorflow:From C:\Users\Carcouss\Anaconda3\lib\site-packa
ges\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorf
low.python.ops.math_ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Use tf.cast instead.
Game 1         Score: 1
Game 2         Score: 0
Game 3         Score: 0
Game 4         Score: 1
Game 5         Score: 1
Game 6         Score: 1
Game 7         Score: 1
Game 8         Score: 1
```

At the end after more than 140 iterations the agent becomes far better at the game than at the beginning:

```
Game 140         Score: 15
Game 141         Score: 28
Game 142         Score: 33
Game 143         Score: 14
Game 144         Score: 25
```

```
Game 145        Score: 15
Game 146        Score: 18
Game 147        Score: 29
Game 148        Score: 15
Game 149        Score: 44
Game 150        Score: 21
```



It only takes 1298.113 seconds on my computer to reach the 150 games (~21 minutes).

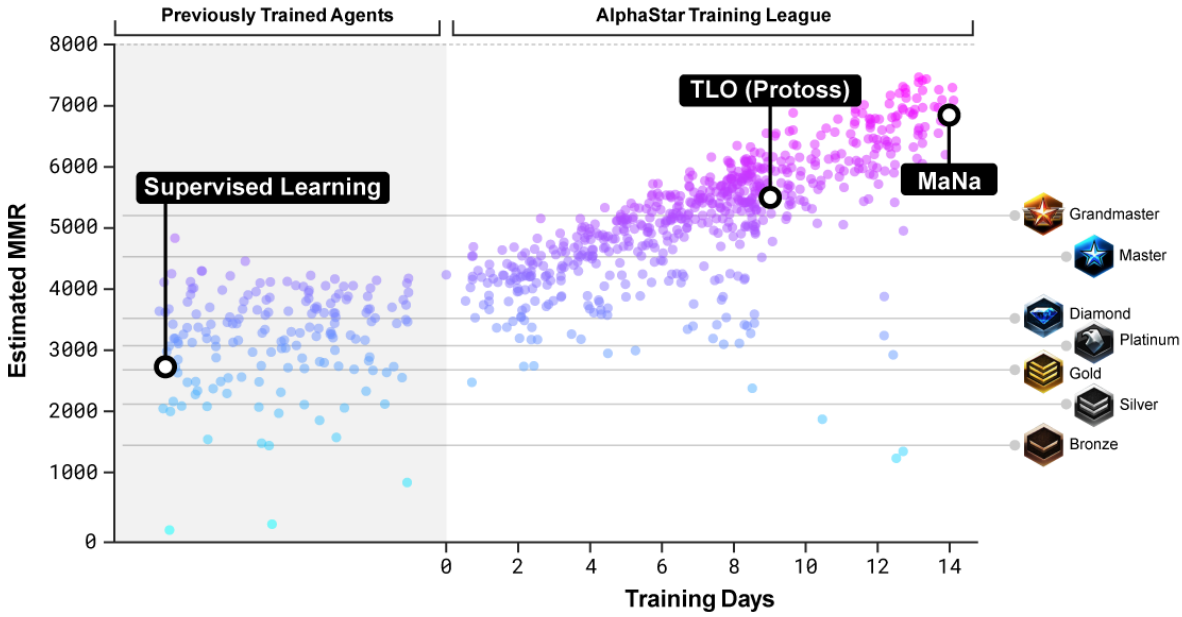
As a final example I will implement a code for an agent that is able to beat the game in Hard Mode most of the time.

SCIENTIFIC PAPERS & DOCUMENTATION

Alphastar is incredibly complex and uses the most advanced alogrithms.

First, they made supervised learning. The AI was first trained with thousands of human games recorded. The power of the AI is measured with MMR, Match Making Rank. It is the system Blizzard uses to classify players (like elo for chess). On the picture below we can see the MMR of the supervised learning. TLO is the MMR of a pro-player as well as MaNa.

Then, when the AI was trained with supervised learning, they created a league called AlphaLeague. In this league 600 of the best agents, with 16 TPUv3 units each, were trained during 14 days. For a total of 9600 TPU and equivalent of 60.000 years of Starcraft 2 games played.



Deepmind uses a lot of high-end algorithm and technology to achieve their goal, some papers can enlight their strategy.

Papers & useful information:

- **"Attention is all you need" (Vaswani et al, NeurIPS 2017)**
  The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism.
  https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

- **"Deep reinforcement learning with relational inductive biases"** (Vinicius Zambaldi, David Raposo & al…)
  We introduce an approach for augmenting model-free deep reinforcement learning agents with a mechanism for relational reasoning over structured representations, which improves performance, learning efficiency, generalization, and interpretability.
  https://openreview.net/pdf?id=HkxaFoC9KQ

- **A good abstract about Alphastar AI can be read on the Deepmind blog:**
  https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

- **“Pointer Networks” (Vinyals et al, NeurIPS 2015)**
  We introduce a new neural architecture to learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence.
  https://papers.nips.cc/paper/5866-pointer-networks.pdf

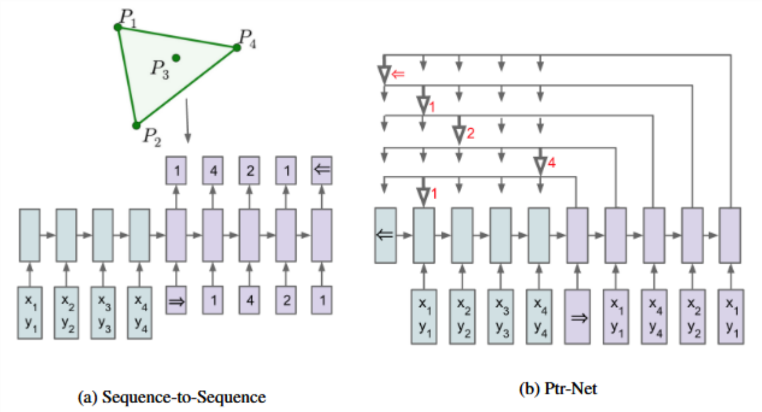


Diagram of PointerNet from original paper. A conventional RNN-based seq2seq model condtionally predicts output from the latent code. A PointerNet outputs attention vectors over its inputs.

- **Alphazero about reinforcement learning and unsupervised strategy**
  A general reinforcement learning algorithm that masters chess, shogi and Go through self-play (David Silver,1;2————————————— Thomas Hubert,1——)
  In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games.
  https://deepmind.com/documents/260/alphazero_preprint.pdf

CHALLENGE / SCIENTIFIC COMPETITION

Alphastar defeated 2 pro players TLO and MaNa:
https://www.engadget.com/2019/01/24/deepmind-ai-starcraft-ii-demonstration-tlo-mana/

Youtube Video of the games:
https://www.youtube.com/watch?v=HcZ48JDamyk&feature=youtu.be

IMPLEMENTATION

I cannot, for obviously reasons, runs on my computer an architecture close to Alphastar. But I 've found some interesting code here (https://pythonprogramming.net/starcraft-ii-ai-python-sc2-tutorial/) in order to create a basic AI ables to defeat the game in hard mode (which requires some skill for a human).

The algorithm is based on a standard Deep Learning convolutional architecture, functions are in ASYNC mode thanks to the **asyncio** package that enables parallelism of the code. The game is **accelerated** dozens of times comparing to the pace of a real game.

1) An agent is created from scratch that can only gather useful resources and attack randomly the enemy.

2) A dataset of training games is created for the supervised learning with games of 2 bots in hard mode. I used the dataset already trained by the author because it would have taken ages to do it on my i3.
   The trained dataset is a succession of action with the coordinates on the Map and the action performed.

3) The AI is trained with more than 8000 games
   Code in ANNEXE 2

   The agent can perform some basic actions like:

   def scout(self): Exploring the mapo in order to find the enemy base

   async def build_workers(self): to build units

   async def build_pylons(self):, async def build_assimilators(self):,…. : build constructions

   async def expand(self): create an expansion of the main base

Neural Network architecture:

   Convolution Layers processes its receptive field like a pattern and once trained is able to recognize the pattern and activates the neurons accordingly

   Pooling Layers reduce the dimensionality of the input

   Dropout is a regulation layer, randomly neurons are ignored during training in order to bring some randomness to the process of training

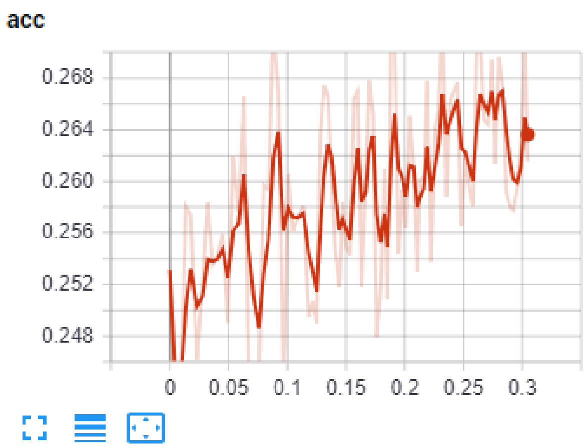
   Dense is a fully connected layer

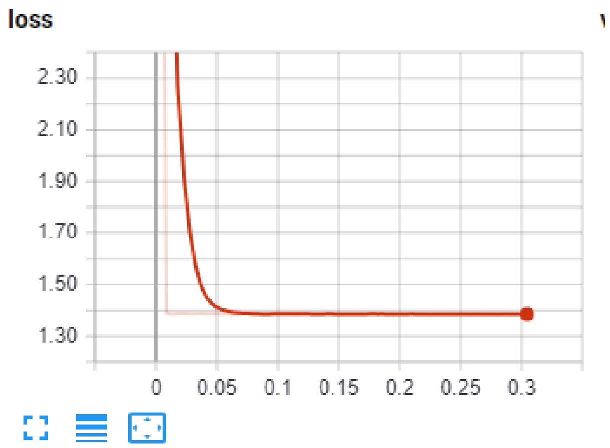
The output has 4 nodes for the 4 finals actions that can take the code:

- **no_attacks**
- **attack_closest_to_nexus**
- **attack_enemy_structures**
- **random.shuffle(attack_enemy_start**

```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=(176, 200, 3),
                 activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same',
                 activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3), padding='same',
                 activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

learning_rate = 0.0001
opt = keras.optimizers.adam(lr=learning_rate, decay=1e-6)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

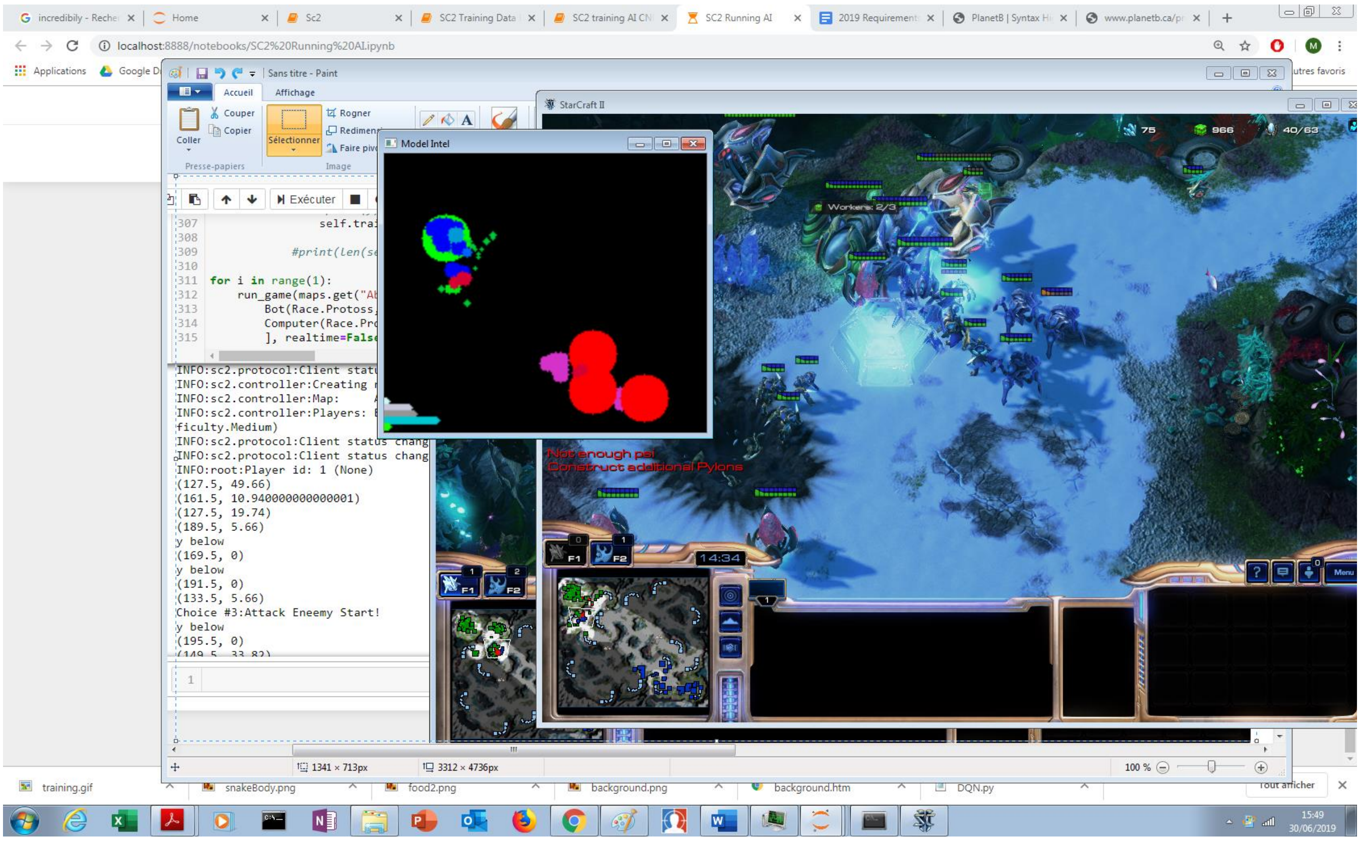
with tensorboard visualization:





4) The AI can be run in the game interface against the game in hard mode
Code in ANNEXE 4

The picture below is actually a screenshot taken on my computer, we can see the game running in a window, a minimap which is a simplified model of the game and in the jupyter log we can follow what actions are performed in the game.



5) More actions are added to the code (from 4 to 14) in order to create a “clever” AI.
Code ANNEXE 5

This time the last layer has 14 nodes corresponding to the 14 actions that can take the agent:

- build_scout",
- build_zealot
- build_gateway
- build_voidray
- build_stalker
- build_worker
- build_assimilator
- build_stargate
- build_pylon

- defend_nexus
- attack_known_enemy_unit
- attack_known_enemy_structure
- expand
- do_nothing

EVALUATION

Feeling:
Implementation of a competitive AI for Starcraft 2 is something really complex and that costs a lot of money. The cost of 9600 TPUv3 running during 14 days is estimated to 7Millions US dollars. Something that foir the moment only the largest company in the world can offer (Google/Deepminf, OpenAI, Apple, IBM, Microsoft…)

There are several project using stracraft learning environment but nothing that comes close to what deepmind achieved with Alphastar. Nevertheless, I assume that the complexity of the architecture is now well-known and with a consequent amount of money and enough time it can be implemented at industrial level.

Finally, now that the starcraft game is considered as solved the most interesting part of it would be to release it with the game in order to offer a real "human-like" challenge for the customer. It also can be implemented in all others RTS existing.

CONCLUSION

In this project I studied how to create an artificial intelligence that runs under the game Starcraft 2, and even if I am aware that I am still far of what Deepmind has been able to create there is a path I'd like to follow. Finally, an agent was trained and ran on my computer. I have sometimes played this game and consider it as one of the best game ever made so it was emotional to watch the game running by itself and being better than me 😉

ANNEXE 1 SNAKE

```
import pygame
from random import randint
from DQN import DQNAgent
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Set options to activate or deactivate the game view, and its speed
display_option = False
```

```python
speed = 0
pygame.font.init()


class Game:

    def __init__(self, game_width, game_height):
        pygame.display.set_caption('SnakeGen')
        self.game_width = game_width
        self.game_height = game_height
        self.gameDisplay = pygame.display.set_mode((game_width, game_height+60))
        self.bg = pygame.image.load("img/background.png")
        self.crash = False
        self.player = Player(self)
        self.food = Food()
        self.score = 0


class Player(object):

    def __init__(self, game):
        x = 0.45 * game.game_width
        y = 0.5 * game.game_height
        self.x = x - x % 20
        self.y = y - y % 20
        self.position = []
        self.position.append([self.x, self.y])
        self.food = 1
        self.eaten = False
        self.image = pygame.image.load('img/snakeBody.png')
        self.x_change = 20
        self.y_change = 0

    def update_position(self, x, y):
        if self.position[-1][0] != x or self.position[-1][1] != y:
            if self.food > 1:
                for i in range(0, self.food - 1):
                    self.position[i][0], self.position[i][1] = self.position[i + 1]

            self.position[-1][0] = x
            self.position[-1][1] = y

    def do_move(self, move, x, y, game, food,agent):
        move_array = [self.x_change, self.y_change]

        if self.eaten:

            self.position.append([self.x, self.y])
            self.eaten = False
            self.food = self.food + 1
        if np.array_equal(move ,[1, 0, 0]):
            move_array = self.x_change, self.y_change
        elif np.array_equal(move,[0, 1, 0]) and self.y_change == 0:  # right - going horizontal
            move_array = [0, self.x_change]
        elif np.array_equal(move,[0, 1, 0]) and self.x_change == 0:  # right - going vertical
            move_array = [-self.y_change, 0]
        elif np.array_equal(move, [0, 0, 1]) and self.y_change == 0:  # left - going horizontal
            move_array = [0, -self.x_change]
        elif np.array_equal(move,[0, 0, 1]) and self.x_change == 0:  # left - going vertical
            move_array = [self.y_change, 0]
        self.x_change, self.y_change = move_array
        self.x = x + self.x_change
```

```
        self.y = y + self.y_change

        if self.x < 20 or self.x > game.game_width-40 or self.y < 20 or self.y > game.game_height-40 or [self.x, self.y] in self.position:
            game.crash = True
        eat(self, food, game)

        self.update_position(self.x, self.y)

    def display_player(self, x, y, food, game):
        self.position[-1][0] = x
        self.position[-1][1] = y

        if game.crash == False:
            for i in range(food):
                x_temp, y_temp = self.position[len(self.position) - 1 - i]
                game.gameDisplay.blit(self.image, (x_temp, y_temp))

            update_screen()
        else:
            pygame.time.wait(300)


class Food(object):

    def __init__(self):
        self.x_food = 240
        self.y_food = 200
        self.image = pygame.image.load('img/food2.png')

    def food_coord(self, game, player):
        x_rand = randint(20, game.game_width - 40)
        self.x_food = x_rand - x_rand % 20
        y_rand = randint(20, game.game_height - 40)
        self.y_food = y_rand - y_rand % 20
        if [self.x_food, self.y_food] not in player.position:
            return self.x_food, self.y_food
        else:
            self.food_coord(game,player)

    def display_food(self, x, y, game):
        game.gameDisplay.blit(self.image, (x, y))
        update_screen()


def eat(player, food, game):
    if player.x == food.x_food and player.y == food.y_food:
        food.food_coord(game, player)
        player.eaten = True
        game.score = game.score + 1


def get_record(score, record):
        if score >= record:
            return score
        else:
            return record


def display_ui(game, score, record):
    myfont = pygame.font.SysFont('Segoe UI', 20)
    myfont_bold = pygame.font.SysFont('Segoe UI', 20, True)
    text_score = myfont.render('SCORE: ', True, (0, 0, 0))
    text_score_number = myfont.render(str(score), True, (0, 0, 0))
    text_highest = myfont.render('HIGHEST SCORE: ', True, (0, 0, 0))
```

```
        text_highest_number = myfont_bold.render(str(record), True, (0, 0, 0))
        game.gameDisplay.blit(text_score, (45, 440))
        game.gameDisplay.blit(text_score_number, (120, 440))
        game.gameDisplay.blit(text_highest, (190, 440))
        game.gameDisplay.blit(text_highest_number, (350, 440))
        game.gameDisplay.blit(game.bg, (10, 10))


    def display(player, food, game, record):
        game.gameDisplay.fill((255, 255, 255))
        display_ui(game, game.score, record)
        player.display_player(player.position[-1][0], player.position[-1][1], player.food, game)
        food.display_food(food.x_food, food.y_food, game)


    def update_screen():
        pygame.display.update()


    def initialize_game(player, game, food, agent):
        state_init1 = agent.get_state(game, player, food)  # [0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0]
        action = [1, 0, 0]
        player.do_move(action, player.x, player.y, game, food, agent)
        state_init2 = agent.get_state(game, player, food)
        reward1 = agent.set_reward(player, game.crash)
        agent.remember(state_init1, action, reward1, state_init2, game.crash)
        agent.replay_new(agent.memory)


    def plot_seaborn(array_counter, array_score):
        sns.set(color_codes=True)
        ax = sns.regplot(np.array([array_counter])[0], np.array([array_score])[0], color="b", x_jitter=.1, line_kws={'color':'green'})
        ax.set(xlabel='games', ylabel='score')
        plt.show()

    def run():
        pygame.init()
        agent = DQNAgent()
        counter_games = 0
        score_plot = []
        counter_plot =[]
        record = 0
        while counter_games < 150:
            # Initialize classes
            game = Game(440, 440)
            player1 = game.player
            food1 = game.food

            # Perform first move
            initialize_game(player1, game, food1, agent)
            if display_option:
                display(player1, food1, game, record)

            while not game.crash:
                #agent.epsilon is set to give randomness to actions
                agent.epsilon = 80 - counter_games

                #get old state
                state_old = agent.get_state(game, player1, food1)

                #perform random actions based on agent.epsilon, or choose the action
                if randint(0, 200) < agent.epsilon:
```

```python
                    final_move = to_categorical(randint(0, 2), num_classes=3)
                else:
                    # predict action based on the old state
                    prediction = agent.model.predict(state_old.reshape((1,11)))
                    final_move = to_categorical(np.argmax(prediction[0]), num_classes=3)

                #perform new move and get new state
                player1.do_move(final_move, player1.x, player1.y, game, food1, agent)
                state_new = agent.get_state(game, player1, food1)

                #set treward for the new state
                reward = agent.set_reward(player1, game.crash)

                #train short memory base on the new action and state
                agent.train_short_memory(state_old, final_move, reward, state_new, game.crash)

                # store the new data into a long term memory
                agent.remember(state_old, final_move, reward, state_new, game.crash)
                record = get_record(game.score, record)
                if display_option:
                    display(player1, food1, game, record)
                    pygame.time.wait(speed)

            agent.replay_new(agent.memory)
            counter_games += 1
            print('Game', counter_games, '      Score:', game.score)
            score_plot.append(game.score)
            counter_plot.append(counter_games)
        agent.model.save_weights('weights.hdf5')
        plot_seaborn(counter_plot, score_plot)


    run()
```

ANNEXE 2: training Neural Network

```python
import keras  # Keras 2.1.2 and TF-GPU 1.8.0
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, LSTM, TimeDistributed
from keras.layers import Conv2D, MaxPooling2D
from keras.callbacks import TensorBoard
import numpy as np
import os
import random


model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=(176, 200, 3),
                 activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same',
                 activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3), padding='same',
                 activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())


model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(4, activation='softmax'))




learning_rate = 0.0001
opt = keras.optimizers.adam(lr=learning_rate, decay=1e-6)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

tensorboard = TensorBoard(log_dir="logs/STAGE1")

train_data_dir = "train_data"


def check_data():
    choices = {"no_attacks": no_attacks,
               "attack_closest_to_nexus": attack_closest_to_nexus,
               "attack_enemy_structures": attack_enemy_structures,
               "attack_enemy_start": attack_enemy_start}

    total_data = 0

    lengths = []
    for choice in choices:
        print("Length of {} is: {}".format(choice, len(choices[choice])))
        total_data += len(choices[choice])
        lengths.append(len(choices[choice]))

    print("Total data length now is:",total_data)
    return lengths


# if you want to load in a previously trained model
# that you want to further train:
# keras.models.load_model(filepath)
hm_epochs = 10
import time
a = time.time()
print(a)
for i in range(hm_epochs):
    current = 0
    increment = 200
    not_maximum = True
    all_files = os.listdir(train_data_dir)
    maximum = len(all_files)
    random.shuffle(all_files)

    while not_maximum:
        print("WORKING ON {}:{}".format(current, current+increment))
```

```
    no_attacks = []
    attack_closest_to_nexus = []
    attack_enemy_structures = []
    attack_enemy_start = []

    for file in all_files[current:current+increment]:
        full_path = os.path.join(train_data_dir, file)
        data = np.load(full_path, allow_pickle = True)
        data = list(data)
        for d in data:
                   choice = np.argmax(d[0])
                   if choice == 0:
                       no_attacks.append([d[0], d[1]])
                   elif choice == 1:
                       attack_closest_to_nexus.append([d[0], d[1]])
                   elif choice == 2:
                       attack_enemy_structures.append([d[0], d[1]])
                   elif choice == 3:
                       attack_enemy_start.append([d[0], d[1]])

           lengths = check_data()
           lowest_data = min(lengths)

           random.shuffle(no_attacks)
           random.shuffle(attack_closest_to_nexus)
           random.shuffle(attack_enemy_structures)
           random.shuffle(attack_enemy_start)

           no_attacks = no_attacks[:lowest_data]
           attack_closest_to_nexus = attack_closest_to_nexus[:lowest_data]
           attack_enemy_structures = attack_enemy_structures[:lowest_data]
           attack_enemy_start = attack_enemy_start[:lowest_data]

           check_data()
           train_data = no_attacks + attack_closest_to_nexus + attack_enemy_structures + attack_enemy_start

           random.shuffle(train_data)
           print(len(train_data))
           test_size = 100
           batch_size = 128

           x_train = np.array([i[1] for i in train_data[:-test_size]]).reshape(-1, 176, 200, 3)
           y_train = np.array([i[0] for i in train_data[:-test_size]])

           x_test = np.array([i[1] for i in train_data[-test_size:]]).reshape(-1, 176, 200, 3)
           y_test = np.array([i[0] for i in train_data[-test_size:]])

           model.fit(x_train, y_train,
                     batch_size=batch_size,
                     validation_data=(x_test, y_test),
                     shuffle=True,
                     verbose=1, callbacks=[tensorboard])

           model.save("BasicCNN-{}-epochs-{}-LR-STAGE1".format(hm_epochs, learning_rate))
           current += increment
           if current > maximum:
               not_maximum = False

   b = time.time()
   print(b-a)
```