Gymnázium Christiana Dopplera, Zborovská 45, Praha 5

ROČNÍKOVÁ PRÁCE

SC2 umělá inteligence

Vypracoval: František Srb
Třída: 8.M
Školní rok: 2018/2019
Seminář: Seminář z programování

Prohlašuji, že jsem svou ročníkovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím s využíváním práce na Gymnáziu Christiana Dopplera pro studijní účely.

V Praze dne 20.2.2019 František Srb

Obsah

1. Úvod

Tato ročníková práce pojednává o implementování umělé inteligence do real-time strategie (dále RTS) Starcraft 2. Cílem je implementovat neuronovou síť do umělé inteligence, která by byla schopna vyrovnat se profesionálním hráčům. V této práci je vysvětlena problematika implementace kódu a neuronové sítě do RTS hry, vytvoření samotného bota obsahující umělou inteligenci, optimalizace bota a jeho schopnost se vyvíjet.

2. Starcraft II

Aby jsme lépe pochopili tuto práci tak je lepší znát zmíněnou hru. Cílem hry je zničit veškeré nepřátelské jednotky a struktury. Základem hry je mít prosté pracovníky, kteří sbírají suroviny(minerals, vaspene gas) a staví struktury, které dovedou vytvářet bojové jednotky a vylepšovat je. Ve hře jsou 3 různé rasy (Terran, Zerg, Protoss), které mají rozdílné jednotky a struktury. To znamená, že nemůžu svojí umělou inteligenci spustit pro rozdílné rasy, a tedy se soustředíme pouze na jednu(v tomto případě Terran)

2.1 Vstupní a výstupní data

Teď hru přeložíme do programovacího jazyka a musíme složité klikací úlohy přepsat na různé funkce. Hra běží v tak zvaných iteracích. Za jednu iteraci zavolá jednou funkci *on_step()*. Iterací proběhne zhruba 165 za minutu. Hra tedy zavolá 165x za minutu můj program, aby udělal tah, kde jako vstup dostane všechny známe informace, které jsou uloženy v proměnné *self*. Jsou tam uloženy například:

- jednotky(jméno, koho jsou, životy, útočná síla, pozice, zda-li se pohybuje či útočí)
- struktury(jméno, koho jsou, pozice, životy, jestli jsou postaveny popř. z jaké části jsou postaveny, zda něco produkují)
- suroviny
- uběhnuté iterace (neboli čas)

Ze vstupních dat musíme vyvodit výstupní data, což znamená zaúkolování jednotek a struktur. Pokud chci například poslat jednotky na nějaké dané místo, musím každou z nich označit a po jedné jim zadat úkol. A toto vše se musí stihnout v jedné iteraci. Naštěstí pokud náš Bot nestihne do té jedné iterace zareagovat(např. z důvodu hromady výpočtů) tak hra zamrzne a počká dokud program neskončí *on_step()* funkci.

2.2 Zprovoznění a spuštění programu

Program a příslušné soubory lze najít na Githubu, na který je odkaz v přílohách. Program stačí otevřít v libovolném prostředí podporující čtení souboru s koncovkou **.py**, poté nastavit počet her, které chceme aby program odehrál, do proměnné *number_of_games* na řáde 9 a program spustit. Pokud chcete program bezpečně ukončit dřív než doběhnou všechny hry, pak doporučujeme program zastavit či vypnout, když konzole vypíše – *Hra X z N* –, kdy program spí přesně 5 sekund a je bezpečné ho vypnout bez ztráty dat.

Dále je důležité aby v jednom adresáři byly následující soubory:

- weights_folder – adresář obsahující úspěšné neuronové sítě
- ranking.txt – textový soubor ukládající aktuální postup neuronové sítě
- score.txt – textový soubor ukládající výsledky
- terransc2.py – Program naspaný v jazyce Python
- time.txt – textový dokument ukládající počet odehraných her a odehraného času v sekundách (herního času)
- weights.txt – textový soubor obsahující aktuální váhy synapsí

3. Programovací prostředí

V této kapitole odůvodním výběr programovacího jazyka a knihovny.

3.1 Jazyk, knihovna a inspirace

Z hlediska programovacího jazyka jsme na výběr moc neměli. Mohli jsme využít pokud oficiální Blizzard knihovnu do jazyka **C++** [1], a nebo PySc2 [2] či Python-sc2 [3] , které využívají oficiální Blizzard knihovnu přeloženou do jazyka **Python**. Python-sc2 na rozdíl od ostatních knihoven obsahuje srozumitelnou dokumentaci a na internetu mají o hodně víc tutoriálů. Vybral jsem si samozřejmě Python-sc2, na jejich stránkách se dá najít také návod na instalaci knihovny.

Dále potřebujeme hru Starcraft 2, aby jsme mohli bota kde spouštět. Hra je ke stažení zdarma na oficiálních stránkách Blizzardu [4]. Tady je dobré podotknout, že na hru vycházejí neustále nové patche, které lehce upravují hru a mohlo by se stát, že by tento kód nefungoval na novějších verzích hry. Ročníková práce je dělána na **verzi 4.8.2**.

Jako inspiraci a tutoriál pro tuto práci jsem využil sérii videí od Youtube uživatele Sentdex[5]

3.2 Propojení programu se hrou

V kódu, na který je odkaz v příloze, je na prvním řádku *import sc2*, který propojí knihovnu PySC2 s kódem. Na řádcích 2-4 importujeme potřebné funkce a metody.

```
395 result = run_game(maps.get("CatalystLE"), [Bot(Race.Terran,
        SrBoTerran(mutate)),Computer(Race.Protoss, Difficulty.Easy)],
        realtime=False)
```

Na řádku 495 můžeme vidět funkci *run_ game()*, která spouští hru s danými argumenty:

- *maps.get("CatalystLE")* – který určí na jaké mapě se bude hrát. Mapy je důležité mít předem stažené.
- *[Bot(Race.Terran, SrBoTerran(mutate)),Computer(Race.Zerg, Difficulty.Easy)]* – druhý argument určuje hráče Bot/Computer/Player kde Bot je námi vytvořená funkce v kódu, Computer je AI vytvořené Blizzardem ve hře a Player umožní hrát člověku. Každý hráč musí mít určeno za jakou rasu hraje a v případě Bota odkaz na funkci obsahující umělou inteligenci a v případě Computera obtížnost.
- *realtime=false* – tento bool nám určí, zda hra pojede v reálném čase, aby měl člověk čas reagovat na události(true), anebo zda hra pojede v rychlejším stavu kterou může Bot i Computer stíhat(false)(tato možnost je cca 15x rychlejší než true)

Tato funkce taky vrací výsledek o tom jak dopadla hra (pokud Result.Victory nebo Result.Defeat)

4. Umělá inteligence

4.1 Funkce

Program má mnoho funkcí které může z funkce *on_step()* zavolat. Například funkce *create_marine()*, která vyrábí základní bojovou jednotku s názvem Marine:

```
348 async def create_marine(self):        #vytvarim mariny
349         if self.units(BARRACKS).ready.noqueue.exists and self.
                can_afford(MARINE):
350             await self.do(self.units(BARRACKS).ready.noqueue.
                    random.train(MARINE))
```

Řádek 349 nám kontroluje zda-li máme možnost tuto jednotku vůbec vytvořit (jestli máme strukturu, která má prázdnou úkolovou frontu a zda-li si můžeme Mariny dovolit z hlediska životního prostoru a surovin). Řádek 350 pak vybere náhodnou strukturu bez fronty a zaúkoluje ji vytvořením jednotky.

Další příklad je výňatek z funkce *army_management()*, která úkoluje bojové jednotky.

```
368 async def army_management(self):        #inteligence vojaku - brani
        okolo sebe
369         #MARINE
370         if (self.known_enemy_units-self.known_enemy_units(
                CHANGELINGMARINE)).exists:
371             for marine in self.units(MARINE).idle:
372                 if((self.known_enemy_units-self.
                        known_enemy_units(CHANGELINGMARINE)).
                        closer_than(70.0,marine).exists):
373                     await self.do(marine.attack((self.
                            known_enemy_units-self.
                            known_enemy_units(
                            CHANGELINGMARINE)).closest_to(
                            marine)))
```

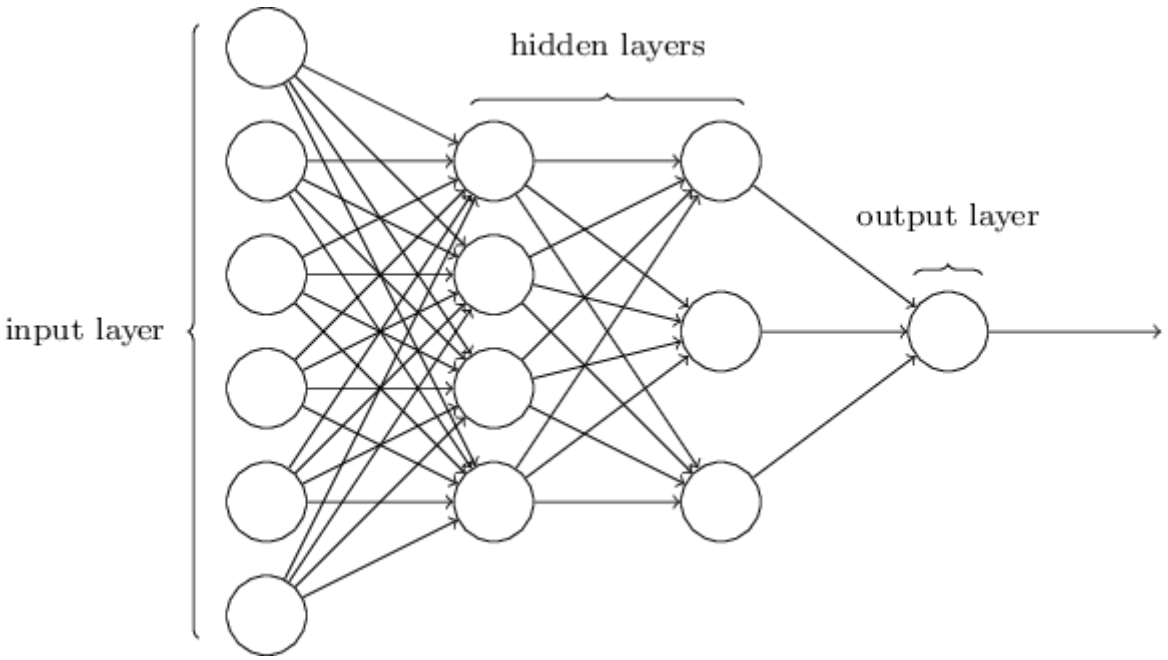
Tato funkce zkráceně zkoumá jestli Bot má informace o existenci nepřátelské jednotky a pokud nějaká existuje blíž než 70 jednotek vzdálenosti od označeného Marina, který není zaúkolovaný, tak daný Marine zaútočí na nejbližší nepřátelskou jednotku či strukturu.

4.2 Neuronová síť

V předchozí sekci jsme mohli vidět dvě rozdílné funkce. Logicky chceme, aby funkce úkolující jednotky běžela neustále, ale funkce tvořící bojové jednotky běžela jen ve chvíli kdy je vytvářet chceme. Proto implementujeme neuronovou síť.

4.2.1 Teorie

Neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci. Jejím vzorem je chování odpovídajících biologických struktur [6]. Neuronová síť je tvořena z řad neuronů. Vstupní řada (input layer), schovaná řada (hidden layer) a výstupní řada (output layer). Každý neuron je propojen se všemi (ale nemusí být se všemi) neurony z předchozí a následující řady pomocí synapsemi. Každá synapse má svojí váhu. Ve vstupní řadě uložíme do každého neuronu vstupní data. Každý neuron mimo vstupní řadu obsahuje vážený součet neuronů z předchozí řady (dále může obsahovat matematickou funkci než pošle výsledek dál). Výstupní řada nám pak předá výsledek celé neuronové sítě.



Obrázek 4.1: Neuronová síť

Zdroj: `https://chatbotslife.com/how-neural-networks-work-ff4c7ad371f7`

4.2.2 Vlastní implementace

V kódu je neuronová síť implementována následovně. Vstupní řada obsahuje 9 neuronů s následujícími vstupy:

1. čas v minutách

2. minerály ve stovkách (minerals/100)
3. vaspene gas po padesáti (vaspene/50)
4. pokrok v technologickém stromě - viz. program, řádky 232-244
5. počet jednotek SCV (workers)
6. počet jednotek Marine
7. počet jednotek Siege tank
8. počet jednotek Battlecruiser
9. počet základen

Poté následují dvě schované řady po 16 a 8 neuronech respektive. Poslední je výstupní řada obsahující 6 neuronů. Jelikož výstup z neuronové sítě by měl být pouze jeden, tak nakonec porovnáme čísla ve výstupní řadě a vybereme pozici neuronu s největší hodnotou (takže výstup je celé číslo mezi 0 a 5 včetně). Toto číslo pak později v kódu volá dané funkce tvořící úkoly a taktiky.

```
24 def neural_network(obs):  #nn
25         file = open("weights.txt","r")
26         if(file.readline())!="kontrolni_radek\n":
27                 print("Creating_new_weights")
28                 layers = [len(obs),16,8,6]
29                 neurons = []
30                 column = []
31                 file.close()
32                 for j in range(len(obs)): #INPUT LAYER
33                         column.append(0)
34                 neurons.append(column)
35                 for i in range(len(layers)-2):  #HIDDEN LAYERS
36                         column = []
37                         for j in range(layers[i+1]):
38                                 column.append(0)
39                         neurons.append(column)
40                 column = []
41                 for j in range(layers[3]): #OUTPUT LAYER
42                         column.append(0)
43                 neurons.append(column)
44                 weights = []
45                 for i in range(len(layers)-1):
46                         column = []
47                         for j in range(len(neurons[i+1])):
48                                 columnn = []
49                                 for k in range(len(neurons[i])):
50                                         x = random.uniform(-1,1)
51                                         columnn.append(x)
52                                 column.append(columnn)
53                         weights.append(column)
54                 return(layers,neurons,weights)
```

4.2.3 Vývoj

Některé neuronové sítě se dají učit, záleží však na tom, jestli jsme schopni neuronové síti říct správný výstup. Avšak v této práci nelze určit, jaký výstup je v daný iteraci správný. Máme pouze celkový výsledek zda neuronová síť uspěla či nikoli.

Proto jsem vytvořil systém hodnocení, která hodnotí danou neuronovou síť(tvořena ze synapsí) a porovnává s ostatními, popřípadě její velké úspěšnosti pošle proti těžším nepřátelům. Pokud však víme, že aktuální neuronová síť, která projevila úspěchy, selže, tak jí nezahazujeme, ale mutujeme jí.

```
87 def nn_mutation(nnl,nnn,nnw):    #MUTATION OF NEURAL NETWORK
88         for l in range(len(nnl)-1):
89                 l+=1
90                 #print(l)
91                 for n in range(len(nnn[l])):
92                         for w in range(len(nnw[l-1][n])):
93                                 a = randint(0,20)
94                                 if(a==3 or a==4):
95                                         nnw[l-1][n][w] += random.
                                            uniform(-0.2,0.2)
96                                 elif(a==5):
97                                         nnw[l-1][n][w] = random.
                                            uniform(-1,1)
98         return nnw
```

Samozřejmě nechceme úspěšné neuronové sítě zahazovat, a tak je v programu vytvořen systém ukládání pomocí funkce *save_nn_special(layers,neurons,weights)* na řádku 109.

4.3 Problémy při programování

Během programování jsem narazil na hodně malých chyb. Ale veškeré problémy byly čistě herního charakteru. Zde jsou vytknuty dvě chyby, které zabrali největší množství času.

Průzkumné jednotky Zergů Zergové mají specifické průzkumné jednotky, které na první pohled vypadají jako základní bojové jednotky nepřítele, avšak patří Zergům. I když jednotka vypadá přátelsky, náš Bot ji rozezná a určí jako nepřátelskou. Z tohoto důvodu se veškeré bojové jednotky snaží každou iteraci zaútočit na průzkumnou jednotku a každá nahlásí chybovou hlášku: na cíl nelze útočit. Tato chyba vedla ke výraznému zpomalení hry.

Problém je vyřešen tak, že kdykoliv hledáme nepřátelské jednotky tak zároveň z nich odebíráme nepřátelské průzkumné jednotky Zergů.

```
373 await self.do(marine.attack((self.known_enemy_units-self.
        known_enemy_units(CHANGELINGMARINE)).closest_to(marine)))
```

Techlab Budovy vytvářející útočné jednotky mají možnost přistavit další kus budovy, který přidává vylepšení. Tento kus budovy se jmenuje Techlab a zabírá místo poblíž budovy. Problém nastává, když toto místo není u budovy volné. V normálním případě by člověk přesunul budovu jinam, ale toto knihovna neumí a tak nám nezbývá nic jiného než problém vyřešit po svém.

Problém není úplně vyřešen. Bot se snaží budovy stavět dál od sebe a popřípadě postavit budovy navíc, ale nikdy nebudeme mít 100% jistotu, že tam místo na Techlab bude.

5. Výstupní data

V adresáři kde se nachází program se také nachází soubor **score.txt** Který v sobě obsahuje řádky obsahující text Result.Defeat/Result.Defeat.Time/Result.Victory. Každý řádek znamená odehranou hru a text v daném řádku je výsledek hry(Defeat = prohra,Defeat.Time = prohra z důvodu dlouhé hry(více jak 90 minut),Victory = výhra). Tento soubor využijeme k vytvoření závěru.

Pokud se chcete však podívat lehčeji jak se Botovi daří, doporučujeme si prohlédnout adresář **weights_folder**, který obsahuje textové dokumenty s názvem v tvaru "počet výher (max 3) - obtížnost nepřítele - čas uběhnutý od poslední epochy. Například Bot který porazil 3x za sebou nepřítele obtížnosti Easy 14.2.2019 12:31 bude mít název **3-Easy-1550143885.txt**.

6. Závěr

V této práci se nám povedlo implementovat neuronovou síť, která byla schopná hrát hru. Méně úspěšný už však je její pokrok. Ten je velice pomalý, až nulový. Největší chybou bude způsob učení neuronové sítě, který vsází na náhodu. Program mohl vytvořit skvělého hráče během pár pokusů, ale také by se k němu nemusel nikdy přiblížit. Během učení Bota jsme pozměnili neuronovou síť kde jsme v každém neuronu v schované řadě přidali funkci podobnou signum po vypočtení váženého součtu. Původním nápadem bylo to, že program by mohl dosahovat lepších výsledků kdyby pracoval s méně rozdílnými čísly. Tento efekt se však neprojevil. Při dopisování této práce program odehrál dohromady přes 2700 her z nichž vyhrál zhruba 6.5% her. Největším úspěchem jest výhra proti nepříteli s obtížností Medium.

Literatura

[1] Blizzard Entertainment, 2017. Github.com: sc2client-api [online]. Poslední změna 29.1.2019 [cit. 20.2.2019]. Dostupné z: https://github.com/Blizzard/s2client-api

[2] DeepMind, 2017. Github.com: pysc2 [online]. Poslední změna 21.9.2018 [cit. 21.2.2019]. Dostupné z: https://github.com/deepmind/pysc2

[3] Hannes Karppila, 2018. Github.com : python-sc2 [online]. Poslední změna 10.2.2019 [cit. 17.2.2019]. Dostupné z: https://github.com/Dentosal/python-sc2

[4] Blizzard Entertainment, 2019. Starcraft2.com [online]. [cit. 18.2.2019]. Dostupné z: https://starcraft2.com/en-us/

[5] sentdex, 2018. YouTube.com: Python AI in StarCraft II turoial [online]. 21.6.2018 [cit. 18.2.2019]. Dostupné z: https://www.youtube.com/watch?v=v3LJ6VvpfgI&list=PLQVvvaa0QuDcT3tPehHdisGMc8TInNqdq. Kanál uživatele sentdex.

[6] Kolektiv autorů, 2018. Wikipedia.org: Umělá neuronová síť [online]. Poslední změna 20.9.2018 [cit. 15.2.2019]. Dostupné z: https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A5

Přílohy

1. **Git repozitář s programem a příslušnými soubory:**

   `https://github.com/franke333/SC2-umela-inteligence`