Универзитет "Св. Кирил и Методиј" - Скопје
Факултет за информатички науки и компјутерско инженерство

ДИПЛОМСКА РАБОТА

Дизајн на агент-играч за играта Connect4 базиран на учење со поттикнување

Студент: Вилијан Монев 151063
Ментор: проф. д-р Соња Гиевска

Членови на комисија
проф. д-р Андреа Кулаков
проф. д-р Ѓорѓи Мацаров

Скопје, Март 2020 година

Абстракт

Целта на оваа дипломска работа е да се дизајнира агент кој ќе биде способен да ја игра играта Connect4. Агентот ја совлада играта со употреба на учење со поттикнување. Однесувањето на агентот беше дефинирано со користење на следните алгоритми : Q-learning, DQN, minimax-DQN и Double-DQN кои припаѓаат на фамилијата Q-learning алгоритми. Агентите учат да ја совладуваат играта преку различни видови на интеракција со средината. Првиот вид на интеракција претставува играње против веќе дефиниран агент кој се третира како дел од средината. Вториот начин претставува само-игра каде што агентот игра епизоди од играта сам против себе притоа обидувајќи се од своите грешки да научи како да се подобри во истата.

Резултатите од дипломската работа укажуваат на можноста да се дизајнира агент кој ќе научи сам да ја игра играта Connect4 кажувајќи му ги само правилата на играта. Целта на користењето на учење со потикнување во решавање на најразлични игри е да се испита потенцијалот кој го поседува, притоа испитувајќи ги и правилните методологии на употреба. Откако би се совладале игрите, идејата и очекувањата се истите алгоритми да се скалираат и да почнат да се користат во реалниот свет каде што би му помогнале на човекот за решавање на разновидни проблеми.

Содржина

1. Вовед

Идејата дека учиме преку интеракција со околината во која се наоѓаме е една од првите идеи кои ни доаѓа на ум кога размислуваме за природата на учењето. Ова може да се забележи уште од најраните стадиуми на човековиот живот. Кога мало дете почнува да си игра и да ја оспознова околината нема конкретен надледувач, тоа прави најразлични движења надевајќи се дека ќе ја постигне посакуваната цел. Откога ќе ја постигне целта, движењата односно акциите кои му овозможиле да ја постигне истата ги памти како добри за во иднина да може да ги повтори. За секоја остварена акција која ни претставува некаков вид на интеракција со околината детето добива повратна информација. На овој начин се генерира големо податочно множество од извршени акции и информации за последиците од акциите. Ваквото множество го користиме за учење.

Овој начин на учење ги воспостави корените на учење со поттикнување кое е засновано на истите принципи. Главната цел е да се креира вештачка интелигенција која преку интеракција со околината ќе може да научи како да се однесува во истата за да постигне некоја претходно дефинирана цел.

Во последните години учењето со поттикнување завзема голем подем и истото постигна успешни резултати во најразлични области. Ваквиот тип на учење најпрвин се обиде да ги совлада стратешките игри како GO, Шах и Shogi. Овие игри се доста популарни кај луѓето уште многу одамна, истите се сметаат за доста комплексни и експертите во нив поминуваат значителен дел од животот усовршувајќи се. Во последната деценија беа создадени системи кои користат учење со поттикнување победувајќи ги најдобрите играчи во овие игри. Треба да се напомене дека DeepBlue[1] го победи Гери Каспаров во 1997 година, тогашниот најдобар играч во шах, но овој систем не користеше учење со поттикнување. Истиот беше наменет за решавање на еден конкретен проблем.

AlphaGo[2] во 2016 година го победи Ли Сидол најдобриот играч во играта Go. Во полето на учење со поттикнување ова се смета за едно од најголемите

достигнувања бидејќи компјутерот играјќи сам против себе успеа да ја усоврши играта која се игра повеќе од 3000 години.

Покрај стратешките игри, учењето со поттикнување наоѓа примена и во економијата, автономните возила, маркетинг и слично.

Овие достигнувања ме мотивираа да креирам агент-играч за играта Connect4. Станува збор за доста поедноставна игра која не бара голема процесирачка моќ при имплементација на алгоритмите. Во понатамошниот дел од дипломската ќе бидат презентирани резулатите од дизајнираниот агент и неговите фази во учењето. Евалуацијата на перформансите на агентот се извршува на натпревар кој е организиран на страната Kaggle[3].

2. Учење со поттикнување

Учење со поттикнување претставува учење во кое системот учи како да мапира дадени ситуации во соодветна акција, притоа обидувајќи се да максимизира некоја претходно дефинирана награда. На системот не му е кажано кои акции треба да ги извршува, негова задача е тој сам да открие преку проба и грешка.

Во некои поинтересни ситуации кои воедно се случуваат во релативно комплексни средини, моменталните акции кои се извршуваат не влијаат само на моменталната награда што ќе се добие туку и на сите награди кои понатаму се добиваат. Во играта која се обработува во овој дипломски труд се среќава така наречена ретка награда. Овој тип на награда претставува награда која агентот ја добива само на крајот од епизодата во зависност од исходот на истата, позитивна или негативна. За сите други акции кои ги извршува во меѓувреме не добива никаква награда. Бидејќи учење со поттикнување се заснова на учење од добиените награди за соодветните акции треба внимателно да се дизајнира агентот за да биде способен да научи од ретката награда.

Еден од предизвиците кои се јавува во овој тип на учење, а не се среќава на друго место во областа на машинското учење е размената односно воспоставување на баланс помеѓу истражување и експлоатација. Кога агентот се наоѓа во дадена

ситуација тој може да избере дали сака да ја изврши најдобрата акција која што мисли дека ќе му донесе најмногу награда или пак би сакал случајно избере некоја нова акција за која не е сигурен каква награда ќе му донесе. Доколку агентот избере да ја изврши најдобрата акција станува збор за експлоатација, а од друга страна пак доколку случајно избере акција станува збор за истражување. Балансот е многу битен бидејќи доколку извршуваме само експлоатирачки акции ја губиме можноста да научиме нешто ново за средината што можеби е подобро од тоа досега што го знаеме, а доколку цело време истражуваме не гарантираме никаков прогрес кон дадената цел.

Во учењето со поттикнување може да се идентификуваат четири главни карактеристики:

- *стратегија* - оваа карактеристика го дефинира однесувањето на агентот во дадена ситуација. Всушност претставува мапирање од дадена ситуација односно состојба во акција која што агентот би сакал да ја изврши. Стратегијата може да биде детерминистичка, каде што во тој случај точно се знае која акција да се изврши или пак стохастична каде што се добива дистрибуција од веројатности за секоја акција.
- *награда* - оваа карактеристика ја дефинира целта која се обидува да ја постигне агентот. За секоја извршена акција околината враќа соодветна награда, каде што целта на агентот претставува некаква функција од наградата. Наградата директно означува кои се добри а кои се лоши акции што значи истата влија врз стратегијата која што се користи.
- *функција на вредност* - оваа карактеристика ја означува очекуваната награда која агентот се надева дека ќе ја оствари од моменталната состојба до крајот на епизодата.
- модел - оваа карактеристика претставува креирање на модел кој ќе се однесува исто како средината во која што се наоѓа агентот. Со ваквиот модел може да се симулира средината при што би му овозможило на агентот да изврши планирање пред изборот на наредните акции. Методите од учење со поттикнување кои користат модел за симулација

на средината се нарекуваат модел базирани алгоритми додека пак тие што не користат се нарекуваат модел слободни. Во реалноста многу се ретки проблемите каде што може да се симулира средината.

3. Connect4

Connect4 претставува популарна игра за двајца играчи. Играта се игра на табла која што содржи 6 редици и 7 колони. Играчите наизменично поставуваат фигури во некоја од колоните, така што фигурата паѓа најдоле до првото слободно место во таблата. Играчот кој прв ќе спои четири фигури вертикално, хоризонтално или дијагонално победува. На *Слика1* е прикажана една епизода од играта Connect4 каде што играчот со црвените фигури остварува победа.

Слика1: Епизода од играта Connect4

Агентот кој се дизајнира во истражување на влез ќе добие некоја конфигурација на таблата и на излез ќе знае да одговори во која колона треба да ја постави неговата фигура на тој начин што би си ги зголемил шансите за победа.

Оваа игра е интегрирана во алатка наречена OpenAIGym[4]. Алатката содржи најразлични околини кои што овозможуваат интеракција помеѓу агенти и околина. Connect4 е една ваква околина каде што состојбата на играта е претставена преку

матрица $Board_{R,C}$. Во овој случај *R* го претставува бројот на редици и има вредност 6 додека пак *C* го го претставува бројот на колони кој има вредност 7. Секој елемент $Board_{i,j}$ има нумеричка вредност која што ја претставува фигурата која се наоѓа во i-тата редица и j-тата колона. Значењата на овие нумерички вредности се следните:

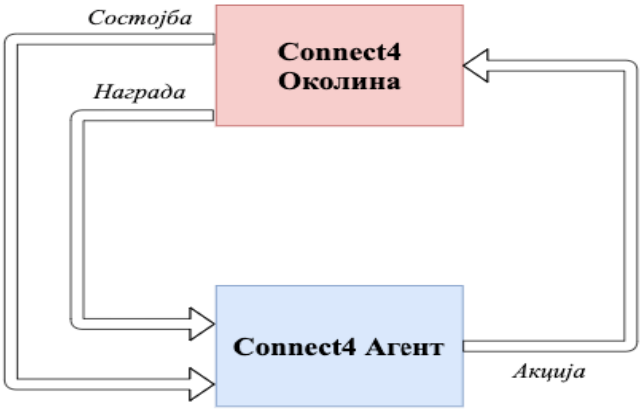
- *0* - означува дека на позицијата $Board_{i,j}$ моментално нема никаква фигура.
- 1 - означува дека на позицијата $Board_{i,j}$ има фигура која е поставена од играч 1.
- 2 - означува дека на позицијата $Board_{i,j}$ има фигура која е поставена од играч 2.

За интеракција со околината се користат следните функции:

- *reset()* - со оваа функција околината се обновува односно се почнува нова игра од почеток. После повикот моменталната состојба е празна табла односно сите вредности на $Board_{R,C}$ се нула.
- *step(action)* - со оваа функција агентот извршува акција во околината. По повикот на оваа функција, агентот на кој што му е редот да постави фигура поставува фигура во првата слободна редица во колоната која има индекс еднаков на параметерот *action*. По извршувањето на акцијата околината ја враќа моменталнта состојба на таблата и исто така соопштува колкава награда агентот добил за извршената акција.

Наградите кои се добиваат од околината Connect4 се следните: *0.5* за секој направен чекор, *1* доколку со извршената акција агентот победи, *0* доколку со извршената акција агентот изгуби и *0.5* доколку партијата заврши без победник.

На *Слика2* е претставена целата интеракција на агентот и околината во играта Connect4.



Слика2: Интеракција помеѓу околината и агентот

Од сликата може да се забележи дека за секоја остварена акција, агентот од околината ја добива новата состојба во која се наоѓа и исто така ја добива наградата за остварената акција.

Целта на учењето со поттикнување е од ваквата интеракција со околината да го научи агентот како треба да се однесува за да постигне што е можно поголема награда на крајот од играта. Една целосна партија од почетокот, кога таблата е празна, до крајот кога некој играч остварува победа се нарекува епизода од играта.

4. Функции на вредност

Функции на вредност претставуваат функции кои ја означуваат очекуваната награда што агентот треба да ја постигне доколку од дадена состојба S_t следи одредена стратегија дефинирана со π. Параметарот t претставува една временска точка од почетокот на епизодата. Има четири главни видови на функции на вредност:

- $V^{\pi}(s)$ функција на вредност со стратегија - претставува функција која ја враќа очекуваната награда која би ја освоил агентот доколку почне од состојба s и ја следи стратегијата π.

- $Q^{\pi}(s, a)$ функција на акција-вредност со стратегија - претставува функција која ја враќа очекуваната награда што би ја освоил агентот доколку почне од состојба s, изврши некоја произволна акција a, која не мора да доѓа од стратегијата π, и потоа продолжи да ја следи стратегијата π.
- $V^*(s)$ оптимална функција на вредност - претставува функција која ја враќа очекуваната награда која би ја освоил агентот доколку почне во состојба s и секогаш ја следи оптималната стратегија за таа средина.
- $Q^*(s, a)$ оптимална функција на акција-вредност - претставува функција која ја враќа очекуваната награда која би ја освоил агентот доколку почне од состојба s, изврши некоја произволна акција a и после тоа секогаш ја следи оптималната стратегија за таа средина.

За сите четири функции на вредност важат специјални равенки на самостојност наречени равенки на Белман[5]. Главната идеја на равенките укажува дека вредноста од почетната состојба е дефинирана како збир помеѓу наградата што агентот ќе ја добие наоѓајќи се во таа состојба и вредноста што очекува да ја добие од следната состојба во која што ќе заврши па се до крај.

5. Q learning

Алгоритмите кои се коирстат во овој дипломски труд се различни варијанти на основниот Q-learning алгоритам кој бил развиен во 1989 година од страна на Ваткингс[6]. Овој алгоритам се обидува да пронајде функција на акција-вредност која што ќе биде најблиску до оптималната. Алгоритамот е дефиниран со следната формула:

$$Q(S_t, A) \leftarrow Q(S_t, A) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A)]\,. \qquad (1)$$

На овој начин функцијата на акција-вредност која што агентот се обидува да ја научи Q, директно ја апроксимира Q^*, оптималната функција на акција-вредност, независно од стратегијата што се користи. Доколку се има добра естимација за $Q(S_t, A_t)$ може директно да се дефинира однесувањето на агентот во околината. Во секоја состојба во која би се наоѓал агентот се избира акцијата која ја дава најголема вредност од функцијата на акција-вредност за таа состојба.

Со терминот $[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$ од равенката за Q-learning се претставува привремената разлика од моменталната вредност $Q(S_t, A_t)$ и новата вредност која ја се добива после извршената акција. R_t ја претставува остварената награда за извршување на акцијата A кога агентот се наоѓа во состојбата S_t. Со $\max_a Q(S_{t+1}, a)$ се избира најголемата очекувана награда која што може да се добие од наредната состојба. Факторот на вреднување γ ја претставува важноста на наградите кои се очекуваат да се добијат во иднина. Доколку овој фактор има вредност 0, тоа означува дека агентот не го интересираат наградите кои ќе ги добие во иднина. Од друга страна пак доколку факторот за вредност е поголем од 1, Q функцијата може да дивергира. Најчесто на овој фактор му се доделува вредност малце помала од 1.

Откако ќе се пресмета привремената разлика, треба моменталната вредност на $Q(S_t, A_t)$ да ја се пренасочи кон таа насока. Ова се постигнува со параметарот α кој се нарекува параметар на учење. Овој параметар означува колкав процент од новата информација ја пребришува старата информација за дадена состојба. Кога $\alpha = 0$ агентот не учи ништо ново, додека пак кога $\alpha = 1$ агентот во предвид ги зема само новите информации.

Функцијата Q може да биде преставена на неколку различни начини. Доколку станува збор за доста едноставна околина со мал број на состојби и акции оваа функција може директно да се претстави како дводимензионална низа каде што редиците ќе ги означуваат сите можни состојби додека пак колоните ги

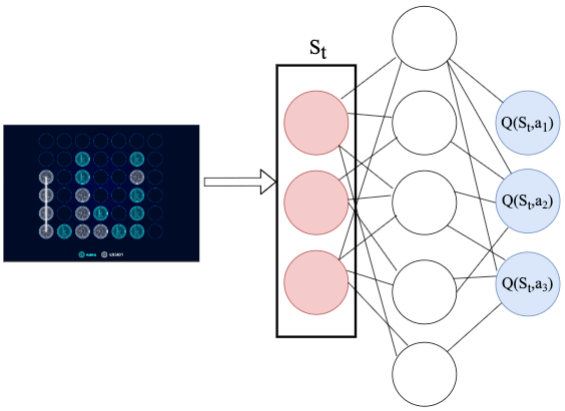
претставуваат возможните акции. Најчесто вакво имплементирање на функцијата не е можно, па затоа треба да се користат апроксиматори на функции. Едни од најпознатите апроксиматори на функции се вештачките невронски мрежи. Доколку Q функцијата се имплементира како невронска мрежа, на влез во мрежата се предава моменталната состојба на околината и на излез се добива очекуваната награда за секоја извршена акција. Со ваквата имплементација треба да се очекуваат приближно исти Q вредности доколку на мрежата се предадат слични состојби.

Докажано е дека алгоритамот Q-learning конвергира до оптималните вредности односно Q^*, доколку бројот на можни состојби е конечен и секоја врска помеѓу состојба-акција се посети доволно голем број пати[7].

5.1. Длабоки Q мрежи

Терминот длабоко учење со поттикнување прв пат се јавува во 2013 година кога тим на научници креира модел базиран на Q-learning[8] кој учи да игра 7 различни Atari[9] игри. Тие го користат истиот модел за учење на 7 игри, притоа моделот остварува подобри резултати во 6 игри до сите дотогашни модели кои биле дизајнирани за да ги играат овие игри. Дополнително во 3 од игрите успеваат да ги победат луѓето кои се сметаат за експерти во истите. Овие резултати го воспоставија почетокот на длабоко учење со поттикнување. Моделот користи конволуциска невронска мрежа каде што на влез добива неколку последователни слики од Atari играта што ја претставуваат состојбата на агентот и на излез ги враќа $Q(s, a_i)$ вредностите за секоја a_i акција која што може да ја изврши агентот во тој момент. Оттука потекнува името длабоки Q мрежи односно DQN.

На *Слика3* е прикажано како би изгледала длабока Q мрежа која би служела како акција-вредност функција на агент во играта Connect4.



Слика3: DQN мрежа кај агент во Connect4

Може да се забележи дека на влез невронската мрежа ја добива моменталната состојба која ја гледа агентот во околината обележана со S_t, а на излез ја апроксимира Q вредноста за секоја акција. Доколку агентот се одлучи на алчна стратегија, може за секоја состојба да ја избере акцијата која добива најголема Q вредност. Доколку агентот поседува добро истренирана ваква невронска мрежа во околината Connect4, тогаш истиот би бил способен да ја игра играта.

Тренирањето односно подесувањето на тежижините на мрежата претставува најголем предизвик. Мрежата се тренира со надгледувано учење. За овој тип на учење потребни се лабелирани податоци, односно податоци каде што за секој податок проследен на влез во мрежата се знае излезот кој што треба да го даде истата. На почеток мрежата се иницијализира со произволни тежени и користејќи ја оваа мрежа агентот почнува да игра повеќе епизоди од играта. Овие епизоди може да се играат против друг претходно дефиниран агент или пак агентот да ги игра против самиот себе што би значело дека истата невронска мрежа ги донесува одлуките за двата играчи. Секој изигран потег се бележи како (s_t, a_t, r_t, s_{t+1}) и додава во низа наречена репризна меморија која се бележи со D. Потегот претсавува подреден пар од моменталната состојба во која што се наоѓал агентот, акцијата што ја изршил, наградата што ја добил од таа состојба со таа изиграна акција и состојбата во која што завршил после изигранта акција. Изиграните потези уште се нарекуваат искуство на агентот.

Како метрика за ажурирање на тежините на мрежата се користи целната вредност $Q(S_t, a) \rightarrow R_t + \gamma \max_a Q(S_{t+1}, a)$ од Q-learning алгоритамот кој беше дефиниран претходно. Интуитивно ова би значело дека моменталната вредност $Q(S_t, a)$ која се добива од мрежата треба да се насочи малце кон вистинската вредност $R_t + \gamma \max_a Q(S_{t+1}, a)$.

Финално, тренирањето се извршува на тој начин што најпрво се избира подмножество на искуства E од репризната меморија. Потоа за секој елемент во множеството E се пресметува таргет вредноста. Користејќи пропагација кон назад се ажурираат тежините на мрежата на тој начин што би се намалила разликата помеѓу вредноста што ја враќа мрежата и предвидената таргет односно вистинска вредност. Доколку со θ_i се претставени тежините на мрежата при посетувањето на i-тиот примерок од репризната меморија тогаш ажурирањето се извршува со следната форумула:

$$\theta_i \leftarrow \theta_i - \alpha(Q_\theta - target)^2$$

каде што α го претставува параметарот на учење.

Ваквиот начин на учење има повеќе предности отколку обичниот Q-learning алгоритам. Најпрво бидејќи користи произволни множества за ажурирање на тежините секое искуство може да се искористи повеќе пати што би го направило моделот поточен. Учењето од последователни потези не е ефикасно бидејќи примероците се многу тесно поврзани, начинот на кој што се избира множеството E е комплетно случаен па ја укинува корелацијата на потезите при што ја намалува варијансата на ажурирањата.

5.2. Стратегии за истражување

Како што беше напоменато на почетокот, еден од проблемите кои се јавува во учење со поттикнување е воспоставување на баланс помеѓу експлоатацијата и истражувањето на агентот. Има различни стратегии кои го решаваат овој проблем. Во овој беше избрана ϵ-алчна стратегија, каде што со оваа стратегија се дефинира променлива ϵ која ја претставува стапката на истражување. Оваа стапка на истражување ја претставува веројатноста со која агентот ќе изврши истражувачка акција.

На почетокот од тренирањето на агентот ϵ има вредност 1 бидејќи агентот ништо не знае, па сака да изврши што е можно повеќе истражувачки акции. Со секоја извршена акција на агентот во околината се намалува вредноста на ϵ за некој одреден фактор на распаѓање кој е претходно дефиниран.

На овој начин откако агентот ќе стане подобар истиот почнува да извршува повеќе експлоатирачки акции. Оваа стратегија е претставена со функцијата $explore(start, end, decay, actions)$ која што ја враќа веројатноста со која што треба да се изврши истражувачка акција. Параметрите на функцијата го имаат следното значење:

- *start* - почетната вредност на ϵ
- *end* - крајната вредност на ϵ. Агентот секогаш сака да продолжи со истражување бидејќи никогаш не можеме да биде сигурен дали ги открил најдобрите акции.
- *decay* - фактор на распаѓање претставува за колку треба да се намали вредноста на ϵ по секоја извршена акција.
- actions - бројот на извршени акции на агентот.

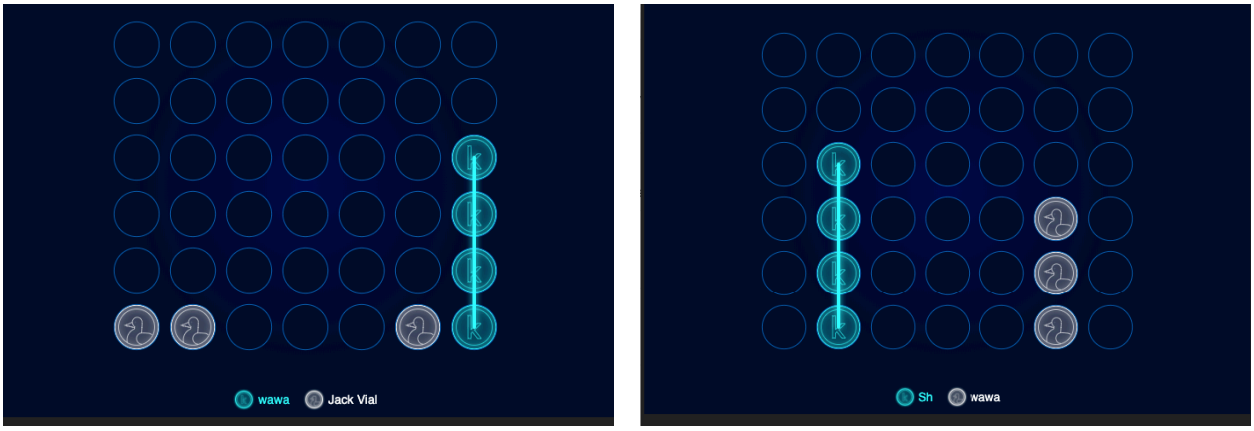
Стратегијата во овој труд беше искористена на тој начин што пред секое избирање на акција од страна на агентот се генерира случаен број, доколку тој број е помал од вредноста на функцијата $explore$ агентот извршува истражувачка акција во

спротивно експлоатирачка. На овој начин беше воспоставен баланс помеѓу експлоатација и истражување.

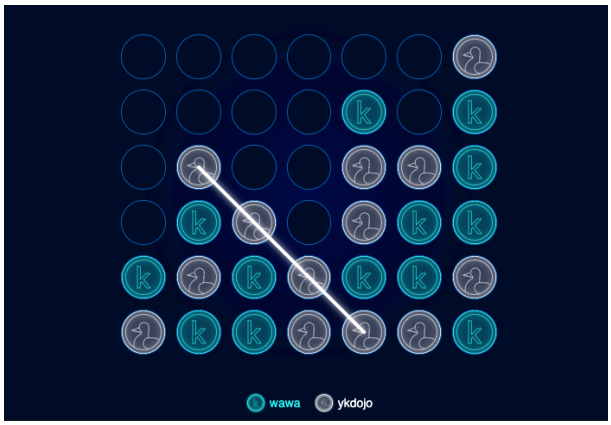
6. Connect4 агент трениран против случаен агент

Во првите истражувања кои беа изработени во овој труд беше креиран агент кој учи да ја игра играта користејќи го DQN алгоритамот опишан претходно. Играјќи против случаен агент, агентот научи да го совладува случајниот агент. Самата околина Connect4, која е овозможена од натпреварот на Kaggle, има имплементирано во неа случаен агент. Овој агент, при изборот на акција во средината не ја зема во предвид состојбата на таблата туку само по случаен избор бира некоја од можните акции за таа состојба. Играјќи против овој агент лесно би можел да се тестира напредокот односно учењето на агентот.

На следните слики се прикажани 3 партии кои беа изиграни агентот. Истиот моментално има 360 поени на ранг листата што е малце повеќе од случаен агент кој постигнува приближно 320 поени. Поените на странта Kaggle се евалуираат на тој начин што се овозможува агенти поставени од различни играчи да играат меѓу себе. После секоја партија на победникот му се додаваат поени, а на поразениот му се одземаат.



Слика4.1: Сценарио во кое агентот успева да победи следејќи ја планираната стратегија и сценарио во кое агентот не успева да ја сфати стратегијата на противникот.



Слика4.2 Епизода во која агентот не успева да ја следи планираната тактика.

Агентот прикажан на сликите беше трениран само на 1000 партии што е релативно мал број на партии. Од сликите и неговиот начин на играње на играта може да се забележи дека успеал да научи некоја тактика со која што најчесто успеава да го победи случајниот агент. Агентот научил тактика доколку ги поставува сите 4 фигури една по енда во иста колона има најмногу шанси да победи. Но, доколку е втор и игра против друг агент кој ја има научено истата тактика нема да биде во можност да сфати на време дека ќе ја изгуби партијата и никогаш не успева да оствари победа. Исто така од третата слика може да се забележи доколку случајниот агент му стави фигура во колоната каде што агентот ги реди фигурите следејќи ја тактиката, после тој потег истиот почнува да прави случајни потези бидејќи нема научено алтернативна тактика.

Тренирањето на агентот против случаен агент не е многу надежно бидејќи агентот ќе научи тактики со кои ќе може само да го победува случајниот агент, а нема да биде во можност да се снаоѓа во посложени ситуации затоа што нема да биде доведен до такви.

За да може Q-learning алгоритамот да конвергира до оптималната вредност, средината во која што агентот учи да се однесува оптимално треба да биде непроменлива. Ова значи дека за извршена акција од страна на агентот во дадена состојба околината секогаш ќе ја враќа наредната состојба според една иста

функција од овие два параметри. Случајниот агент на некој начин може да се третира како дел од средината бидејќи секогаш на ист начин ги избира акциите. Ова не ни дозволува да бидат поставени повеќе агенти во средината, кои ќе бидат од најдобри до најлоши, и потоа да го трениряме агентот да игра против нив. На овој начин ќе се наруши правилото за непроменливост на средината, и агентот во секој чекор од учењето ќе учи да игра само против еден од поставените агенти во околината.

7. Унапредување на длабоки Q мрежи

Во овој дел од трудот се претставуваат повеќе различни варијанти и комбинации на DQN алгоритамот. Во Rainbow[10] трудот кој е објавен во 2017 година се опишуваат кои од подобрувањата на DQN алгоритамот може да се комбинираат за да се подобрат перформансите на истиот. Начинот на комбинирање на DQN алгоритмите кој е препорачан во Rainbow трудот значително ги подобрува перформансите на истиот. Некои од тие комбинации беа имплементирани во агентот.

7.1. Целна Q Мрежа

Како што е напоменато во делот кога е објаснето функционирањето на длабоките Q мрежи, ажурирањето на тежините на мрежата се извршува на тој начин што се намалува квадратната разлика од моменталната вредност на Q мрежата и целната вредност. Ова се пресметува со следната формула

$$\theta \leftarrow \theta - \alpha(Q_\theta - (R_t + \gamma \max_a Q_\theta(S_{t+1}, a)))^2$$

каде што $R_t + \gamma \max_a Q_\theta(S_{t+1}, a)$ ја претставува целта кон која ја се насочуваат параметрите на мрежата.

Може да се забележи дека целта зависи од моменталната мрежа. Невронските мрежи функционираат како целина, па секое ажурирање на тежините околу една точка во Q функцијата исто така влијае на целата околина околу таа точка. Точките $Q(S_t, a)$ и $Q(S_{t+1}, a)$ се блиску една до друга бидејќи секој примерок претставува транзиција од S_t до S_{t+1}. Ова претставува проблем бидејќи со секое ажурирање на тежините во Q_θ мрежата, исто така тежините кои ја претставуваат целта се поместуваат. Компарацијата на овој проблем е со куче како пробува да си го фати опашот бидејќи мрежата сама ги поставува нејзините целни вредност и пробува да се доближи до истите. Проблемот се решава на тој начин што се воведува нова невронска мрежа која ќе ги предвидува целните вредности. Оваа мрежа се нарекува целна мрежа. После одреден број на чекори тежините од Q мрежата се префрлаат на целната мрежа. Со ова и се остава простор на Q мрежата да конвергира, затоа интервалот на кој се ажурираат тежините на целната мрежа треба да биде голем. Основниот DQN алгоритам се менува на тој начин што се дефинира еден хиперпараметар кој ќе го претставува бројот на чекори на кои што се менуваат тежините на целната невронска мрежа со тежините од Q мрежата. По воведувањето на целна мрежа, ажурирањето на тежините се извршува според формулата

$$\theta \leftarrow \theta - \alpha(Q_\theta - (R_t + \gamma \max_a Q_{\theta_1}(S_{t+1}, a)))^2$$

каде што со θ_1 се претставени тежините на целната Q мрежа. Но, за жал со ваквиот начин исто така се успорува учењето.

7.2. Minimax-DQN

Интуитивно DQN алгоритамот работи на тој начин што после секоја извршена акција на агентот, Q функцијата која што ја апроксимира ја приближува кон збирот од добиената награда од таа акција и очекуваната награда која се надева дека ќе ја добие од состојбата во која што ќе заврши по извршената акција. Користењето

на овој алгоритам во агент кој ја игра Connect4 играта би значело дека по секоја извршена акција на агентот тој очекува од наредната состојба да ја добие најголемата очекувана награда. Ова не е случајот тука бидејќи наредната акција која што се извршува во околината е од друг агент односно вториот играч. Вториот играч не би сакал во ниту еден случај да ја зголеми очекуваната награда на првиот играч па оттука може да се воочи проблемот кој се јавува доколку се користи основната варијанта на DQN алгоритамот.

Играта Connect4 може да се претстави како zero-sum[11] игра со двајца играчи. Овој тип на игри се претставени како ситуации во кои добивката или загубата на едниот играч се балансирани со добивката и загубата на вториот играч. Ова би значело доколку се соберат добивките на двата играчи и се одземат загубите би се добила вредност нула.

Во трудот[12] објавен од страна на Yang, се претставува модификација на DQN алгоритамот која овозможува истиот да се користи во zero-sum игри и притоа истиот да конвергира до оптималната вредност. Модификацијата на алгоритамот е претставена во следната равенка:

$$Q_{\theta}(S_t, a) \rightarrow R_t - \gamma \max_{a} Q_{\theta_1}(S_{t+1}, a)$$

каде што со θ_1 се претставени тежините на целната мрежа која што беше дефинирана во претходниот пасус. Со заменувањето на операцијата збир со разлика агентот добива интуиција дека во наредниот чекор вториот агент ќе се обиде да ја минимизира добивката на првиот агент.

Друг начин на имплементирање на minimax-DQN алгоритамот е доколку во последниот слој од невронската мрежа се користи сигмоидна активациска функција. Со користење на оваа активациска функција, Q вредноста за секоја акција ќе биде има вредност помеѓу нула и еден. Со следната равенка е прикажано како би се пресметувала Q вредноста на овој начин.

$$Q_\theta(S_t, a) \rightarrow R_t + \gamma(1 - \max_a Q_{\theta_1}(S_{t+1}, a))$$

Со имплементацијата на една од овие две мали промени во основниот DQN алгоритам значиетелно се подобруваат перформансите на агентот.

7.3. Double Q-learning

Алчното максимизирање на Q-вредностите од наредната состојба при изборот на моменталната акција воведува максимизирачка предрасуда во учењето на алгоритамот. Пример доколку за одредена состојба вистинската Q-вредност за сите акции е еднаква, естимираната Q-вредност од страна на алгоритамот ќе биде различна за секоја акција и тој ќе ја избира цело време најголемата. На овој начин алгоритамот ќе научи дека една акција е подобра од другите, но во конкрениот случај не е така бидејќи сите акции имаат иста вредност.

Решение кое ја намалува максимизирачката предрасуда е имплементирано во трудот објавен од Hado van Hasselt[13] во 2010 година преку претставување на Double Q-learning алгоритамот. Во основната варијанта на алгоритамот се користат две невронски мрежи за естимирање на Q-вредноста, притоа при секое бирање на акција, произволно со иста веројатност се избира една од мрежите да ја естимира Q-вредноста.

Во овој дипломски труд беше искористена модификација на основниот Double Q-learning алгоритам која е претставена во трудот[14] објавен во 2015 година од истиот автор. Наместо користење на два независни модели, тука се користи основната Q мрежа и целната Q мрежа. Естимациите на Q вредноста, користејќи Double Q-learnig, за еден изигран потег од страна на агентот се пресметуваат на следниот начин:

$$Q_\theta(S_t, a) \rightarrow R_t + \gamma Q_{\theta_1}(S_{t+1}, argmax_a Q_\theta(S_{t+1}, a)).$$

Воведувањето на овој алгоритам не ги подобрува секогаш перформансите, но значително го стабилизира процесот на учење. Ваквото стабилизирање му овозможува на агентот да учи покомплицирани задачи.

8. Connect4 агент трениран со само-игра

8.1. Претпроцесирање на состојбите

На почетокот беше објаснето дека во околината Connect4 моменталната состојба агентот ја перцепира како табла претставена со $Board_{R,C}$. Во секоја епизода од играта, агентот може да игра како прв или како втор играч. Ова претставува проблем затоа што агентот ќе има две различни перцепции за играта во зависност од тоа дали ја почнал епизодата како прв играч или не. Целта на агентот е на влез да добие состојба на таблата и на излез да даде соодветна акција. Состојбата која ја добива на влез не треба да зависи од фигурите со кои агентот ја игра играта, фигура 1 доколку прв ја почнува епизодата и фигура 2 доколку втор. За да се постигне ова се извршува пресликување $Board_{R,C} \rightarrow S_{R,C}$ каде што $S_{R,C}$ не зависи од фигурите со кои игра играчот. Ова е постигнато на тој начин што со 0 е прикажано празно поле, со 1 се прикажани моменталните фигури на играчот додека пак со -1 се прикажани фигурите на противникот.

8.2. Симетрија на состојбата

За поточно апроксимирање на Q функцијата агентот треба да посети што е можно поголем број на различни состојби. Оваа цел е постигната на тој начин што се користи симетричноста на состојбите на играта. Симетричност на состојбата претставува својство кое дозволува да се распределат фигурите на таблата на тој начин што од новоформираната состојба нема да се промени исходот на партијата доколку играчите играат според определените стратегии. Connect4 е вертикално

симетрична според средната колона. Ова дозволува да се креираат двојно повеќе состојби притоа што со користење на ова својство значително би се забрзало тренирањето.

За дадена состојба $S_{R,C}$ симетрична состојба $SS_{R,C}$ се креира на тој начин што се заменуваат позициите на првите три колони со последните три. Со матриците S и SS се прикажани две претпроцесирани симетрични состојби.

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & 1 & 0 \\ 1 & -1 & 1 & 1 & 0 & -1 & 0 \end{bmatrix} \quad SS = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 & 1 & -1 & 1 \end{bmatrix}$$

Независно во која состојба се наоѓа агентот дали во S или пак во SS исходот на епизодата ќе биде еднаков.

8.3. Дизајнирање на агентот

Како апроксиматор на Q-функцијата агентот користи длабока невронска мрежа. Во овој труд беа извршувани експерименти со различни дизајни на мрежата, за на крајот да се користи архитектура со 6 слоеви.

Првиот слој на мрежата има 42 неврони со кој што е претставена влезната состојба на табелата. Секоја состојба $Board_{R,C}$ која се добива од околината Connect4 најпрво се претпроцесира на начинот кој беше објаснет претходно, притоа што се добива пресликаната состојба $S_{R,C}$. Оваа состојба се претставува како низа S од 42 елементи на тој начин што првата редица од матрица се првите 7 елементи од низата, втората вторите 7 и така натаму до последната редица.

Значи, со S_t е претставена состојбата на епизодата после изиграни t потези од страна на играчите. Секоја ваква состојба S_t се праќа на влез во мрежата и мрежата одлучува која акција агентот треба да ја изврши.

Следните 4 слоеви ги претставуваат скриените слоеви во моделот. Само вториот од овие слоеви има 256 неврони додека пак останатите 3 имаат по 128 неврони. Слоевите се комплетно поврзани и излезот од секој од овие слоеви поминува низ ReLU активациска функција.

Последниот слој на мрежата има 7 неврони кои ги претставуваат $Q(S_t, a_i)$ вредностите за секоја акција која што може да ја изврши агентот во дадената состојба S_t. На излезот од последниот се користи сигмоидална активациска функција за да се имплементира втората варијанта на minimax-DQN алгоритамот. Бидејќи во некои ситуации некои акции се невалидни, пример кога некоја колона ќе биде целосно пополнета, овие $Q(S_t, a_i)$ вредности се препокриваат со најмала можна вредност со цел агентот да не може да селектира невалидна акција.

8.4. Само-игра

Целото тренирање на агентот односно учење како да ја игра играта Connect4 се одвива преку концепт на учење наречен само-игра[15]. Ова претставува интересен и мистериозен концепт на учење бидејќи невронската мрежа учи да ја игра играта на тој начин што пробува да се победи самата себе си. Подобрувањето има природен тек затоа што кога невронската мрежа ќе стане подобра во играњето на играта и противникот станува подобар што значи ќе треба да совлада нови предизвици за да го победи. Исто така само-игра претставува процес на конвертирање на процесирачка моќ во податочно множество. Извршувањето на голем број симулации од играта овозможува креирање на големо податочно множество кое што може да го искористи агентот со цел да стане подобар во играта. Треба да се внимава како се поставува учењето со само-игра затоа што не

секогаш овозможува подобрување на невронската мрежа. Во овој дипломски труд беше експериментирано со два различни начини на само-игра.

Првиот и поедноставен начин на само-игра е да се остави мрежата да игра цело време против моменталната верзија од себе си. Ова би значело дека не зависно кој играч е на ред, истата мрежа ја извршува наредната акција. Притоа од собраното искуство на ваков начин се ажурираат тежините на мрежата со цел агентот да стане подобар. Со ваквото поставување на само-игра може да се јави проблем во кои што мрежата ќе се врти во круг и нема ништо да научи. Пример за доведување на мрежата во ваква ситуација е следниот: Инстанца *A* од мрежата ќе научи да ја победува инстанца *B* од мрежата, после тоа инстанцата B ќе научи да ја победува инстанцата *C* но интанцата *C* не може да ја победи инстанцата *A.* Може да се забележи дека на овој начин првичната инстанца од мрежата *A* всушност ништо не научила.

Вториот начин на само-игра кој беше користен се заснова на играње против претходни верзии од себе си. На почетокот агентот игра одреден број на епизоди преку првиот начин на само-игра каде што на секоја N-та епизода се зачувуваат моменталните тежини на мрежата. Откако ќе се се соберат M зачувани модели агентот почнува да игра против претходни верзии од себе си. Моменталниот модел игра N епизоди каде што секоја епизода ја игра против произволен модел од претходните M зачувани модели. Доколку во овие N епизоди моменталниот модел постигне доволно голем процент на победа против претходните модели тогаш ги зачувуваме моменталните тежини на мрежата и се продолжува истата постапка одново. Во спротивно доколку агентот не оствари толкав процент на победа, истиот игра нови N епизоди против претходните верзии обидувајќи се да го постигне потребниот процент на победа.

8.5. Тренирање на агентот

Во сите екперименти кои беа извршувани за време на изработката на трудот агентот беше трениран преку еден од двата начини на само-игра.

Главниот алгоритам кои се користи за тренирање на агентот е варијанта на Q-learning која ги енкапсулира сите подобрувања кои беа опишани во секција 7. Овој алгоритам е претставен како minimax-DoubleDQN. Интуитивно алгоритамот функционира на тој начин што невронската мрежа почнува да ја игра играта сама против себе. Невронската мрежа одлучува која акција да ја изврши агентот користејќи ϵ-алчна стратегија. По извршената акција, секое искуство на агентот се зачувува во репризната меморија која што понатаму служи за ажурирање на тежините односно тренирање на невронската мрежа.

Ажурирањето на тежините на неворнската мрежа за едно искуство (s_t, a_t, r_t, s_{t+1}) се извршува со следната функција:

$$\theta \leftarrow \theta - \alpha(Q_\theta(S_t, a) - (R_t + \gamma[1 - Q_{\theta_1}(S_{t+1}, argmax_{a'} Q_\theta(S_{t+1}, a'))])^2 \quad (2)$$

Во оваа функција се вметнати сите подобрувања на Q-learning алгоритамот кој беа претходно детално објаснети. Преку оваа функција со користење на пропагирање кон назад алгоритамот се ажурираат тежините на невронската мрежа. На *Слика5* е прикажан псевдокодот од алгоритамот кој се го користи финалниот агент.

Симетријата на состојбата се употребува при генерирањето на подмножеството. Секое искуство од подмножеството се трансформира во истото симетрично искуство со 50% шанса.

TRAIN_STEP претставува променлива која означува дали треба да се изврши тренирање на мрежата или не. Променлива UPDATE_STEP одлучува после колку изиграни потези во играта треба да се заменат тежините од мрежата која се тренира θ со замрзнатите тежини на целната невронска мрежа θ_1.

minimax-Double DQN алгоритам за тренирање на агентот

Произволно иницијализирај ги вредностите на Q_θ и Q_{θ_1}
Иницијализирај ја репризната меморија D со капацитет N

за секоја епизода од 1 до M:
обнови ја околината за агентот да почне нова партија
додека трае епизодата:
Претпроцесирај ја моменталната S_t состојба од околината
Со веројатност ϵ избери произволна акција a_t во спротивност избери акција $a_t = argmax_a Q_\theta(S_t, a)$
Изврши ја акцијата a_t, земи ја наградата r_t и наредната состојба S_{t+1} од околината по извршената акција во истата
Зачувај го искуството (S_t, a, r, S_{t+1}) во репризната меморија D
Ако TRAIN_STEP:
Избереи произволно подмножество од D со BATCH_SIZE број на елементи
За секој елемент во подмножеството изврши gradient descent според равенката 2
Ако UPDATE_STEP:
Ископирај ги тежине θ во тежините на целната мрежа θ_1

Слика5: minimax-Double DQN

8.6. Подесување на хиперпараметрите

Во minimax-Double DQN алгоритамот кој беше опишан претходно има голем број на хиперпараметри кои може да се подесуваат. Бидејќи тренирањето на агент со едно множество на хиперпараметри одзема околу 20 саати немаше можност да се испитаат голем број на комбинации. Во следната табела се прикажани хиперпараметрите на агентот кој постигна најдобри резултати на натпреварот:

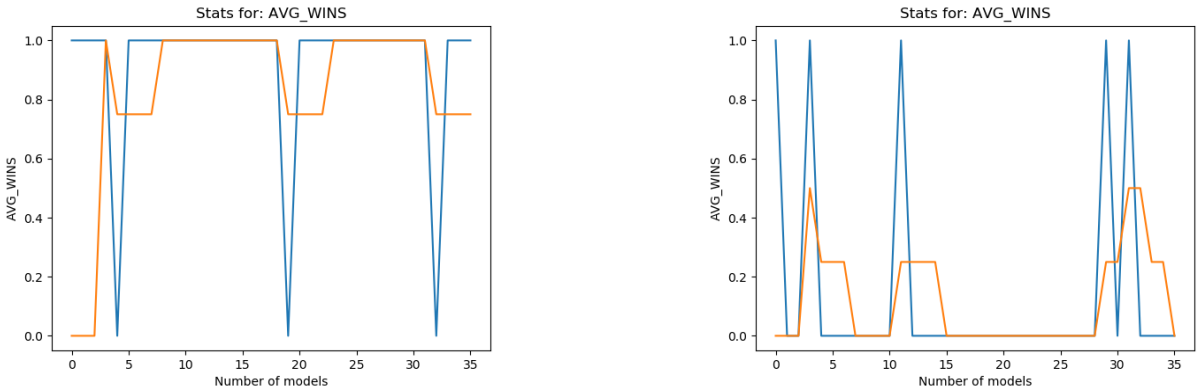
Хиперпараметар	Вредност	Опис
BATCH_SIZE	256	*големината на податочното множество во еден тренирачки чекор*
EPS_START	1	*почетна вредност на ϵ-алчната стратегија за истражување*
EPS_END	0.01	*најмала можна вредност на ϵ која што ќе гарантира истражувачки акции*
EPS_DECAY	0.001	*за колку се намалува ϵ вредноста по секој изгиран потег*
TARGET_UPDATE	700	*после колку чекори ги менуваме тежините на целната мрежа*
α	0.001	*параметар на учење*
MEMORY_SIZE	500000	*големината на репризната меморија однсно бројот на искуства кои може да ги собере*
M	3000000	*бројот на изиграни партии за време на тренирањето*
TRAIN_STEP	32	*го претставува бројот после колку извршени чекори на агентот ги ажурираме тежините на невронската мрежа*

8.7. Евалуација на агентот

За време на тренирањето на агентот, после секои изиграни 5000 епизоди од играта беа зачувани тежините на моменталната верзија на моделот. Евалуацијата на тренираните агенти беше извршена така што беа тестирани перформансите на секој од овие модели против агенти со најразлична јачина. Тренирањето на агентот се одвиваше на тој начин што најпрво беа изиграни 50000 епизоди играјќи сам против себе. Откако беа зачувани доволен број на модели започна тренирањето на агентот според вториот начин на само-игра. За да бидат зачувани тежините од моменталната генерација на моделот истиот треба после изиграни

5000 епизоди против 10-те претходни верзии од себе си да оствари победа барем во 65% од партиите.

На следните слики се претставени перформансите на секоја генерација од моделите против различни видови на агенти.



Слика6: Сценарио во кое секоја генерација од моделот игра против случаен агент и агент кој е помѓу првите 100 на натрепварот

Сините линии на сликите го претставуваат исходот на секоја партија, додека пак со портокаловите е претставен просекот на победа на последните три модели. На левата слика може да се забележи дека моделот речиси во сите партии го победува случајниот агент. На десната слика е претставен исходот на трениранот агент против агент кој се наоѓа во најдобрите 100 на ранг листата на натпреварот. Може да се забележи дека агентот многу поретко остварува победа, но фактот што истиот вушност победува укажува на учење на комплексни тактики. За да се зачуваат овие 35 модели беше потребно тренирање од околу 30 часа. Агентот успеа да се пласира помеѓу најдобрите 50% од сите агенти кои се прикачени на натпреварот.

9. Заклучок

Во овој дипломски труд беше прикажан целокупниот процес на дизајнирање на агент користејќи учење со поттикнување кој е способен да ја игра играта Connect4.

Процесот е претставен и имплементиран на генерички начин така што лесно би можел истиот да се искористи при дизајнирање на агенти за други игри или пак за решавање на некои конкретни проблеми од реалниот свет.

Алгоритамот кој се користеше за тренирање на агентот е наречен minimax-Double DQN и истиот претставува енкапсулација на повеќе различни екстензии на основниот Q-learning алгоритам. Ваквата комбинација на алгоритми покажа добри резултати, пласирајќи го агентот во најдобрите 50% од агентите на натпреварот притоа користејќи само техники од областа на учење со поттикнување. Исто така во истражувањето беа претставени различни начини на само-игра кои му овозможија на агентот да ја совлада средината. Ваквиот начин на тренирање сеуште претставува мистерија и предзвик бидејќи не е докажано зошто истиот постигнува добри резултати.

Во понатамошни истражувања би можеле да се испитаат и други фамилии на алгоритми од областа на учење со поттикнување како што се: модел базирани алгоритми, алгоритми кои користат стратегија и мешавина од наведените. Бидејќи играта не е толку комплексна исто така агентите би можеле да се дополнат со користење на алчно пребарување низ дрвото на состојби на секоја епизода. Зголемување на времето на тренирање исто така би допринело за подобрување на перформансите на агентот.

10. Литература

[1] Murray Campbell, A.Joseph Hoane Jr., Feng-hsiung Hsu, *Deep Blue*. Artificial intelligence, January 2002, 57-83

[2] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwiese,r Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, Demis Hassabis, *Mastering the game of Go with deep neural networks and tree search*. Nature, vol. 529(2016), 484-504

[3] Kaggle ConnectX, https://www.kaggle.com/c/connectx

[4] OpenAI Gym, https://gym.openai.com/

[5] Bellman equation, *Richard S. Sutton and Adnrew G. Barto*, Second edition 2018, page 227

[6] Watkins, C.J.C.H. (1989), *Learning from delayed rewards*. PhD Thesis, University of Cambridge, England.

[7] Watkins, C.J.C.H., Dayan, P. *Q*-learning. *Mach Learn* 8**,** 279–292 (1992). *Q-learning*

[8] Volodymyr Mnih and Koray Kavukcuoglu and David Silver and Alex Graves and Ioannis Antonoglou and Daan Wierstra and Martin Riedmiller, *Playing Atari with Deep Reinforcement Learning*. NIPS Deep Learning Workshop 2013

[9] Atari 2600, *https://en.wikipedia.org/wiki/Atari_2600*

[10] Matteo Hessel and Joseph Modayil and Hado van Hasselt and Tom Schaul and Georg Ostrovski and Will Dabney and Dan Horgan and Bilal Piot and Mohammad Azar and David Silver, *Rainbow: Combining Improvements in Deep Reinforcement Learning*, Artificial Intelligence (cs.AI); Machine Learning (cs.LG)

[11] Zero-sum game, *https://en.wikipedia.org/wiki/Zero-sum_game*

[12] Jianqing Fan and Zhaoran Wang and Yuchen Xie and Zhuoran Yang, *A Theoretical Analysis of Deep Q-Learning, Machine Learning (cs.LG)*; Optimization and Control (math.OC); Machine Learning (stat.ML)

[13] Hado van Hasselt, *Double Q-learning*, Advances in Neural Information Processing Systems 23 (NIPS 2010)

[14] Hado van Hasselt and Arthur Guez and David Silver, *Deep Reinforcement Learning with Double Q-learning*, Machine Learning (cs.LG)

[15] Fictious play, *Richard S. Sutton and Adnrew G. Barto*, Second edition 2018, page 442