**Politechnika Wrocławska**
**Wydział Informatyki i Telekomunikacji**

Kierunek: **TIN**
Specjalność: **TIP**

# PRACA DYPLOMOWA INŻYNIERSKA

## Implementacja modelu sztucznej inteligencji do gry w pokera

Michał Żarejko

Opiekun pracy
Dr. Inż Paweł Zyblewski

Słowa kluczowe: Deep CFR, Poker, Uczenie przez wzmacnianie, Drzewa decyzyjne

WROCŁAW 2022

# Abstract

The main goal of the thesis is to present author's implementation of the Deep CFR algorithm with the Heads Up Limit Poker Texas Hold'em. It is a modern method for creating artificial intelligence in large partial-observable games. Such environments have always been a great challenge and the main barrier to the development of machine learning. The thesis will show this problem by implementation of Deep CFR and analysis of the results. For this purpose, five recognition models were created every 10 iterations of the algorithm. The next step was to create games with that models. The results allowed to select the best playing models and to see how Deep CFR learned over time. Additionally, the quality of the models was tested against a simple program which simulate beginner player.

# Contents

# Chapter 1

# Introduction

Machine learning has been in development for a long time. As early as the 40s, books and programs related to artificial intelligence [1] have been writting. In 1940 Donald Hebb created theoretical foundations used in modern neural networks [1]. Despite such early progress, the branch of science, has begun to achieve great success about 10 years ago. This is due to the fact that many algorithms need a lot of computational power [1].

Today, there are many tools under development which are related with this topic. For example voice assistants, language translators, models to display elements on websites, video games or smart cars. Additionaly, artificial intelligence is heavily used in companies, on production halls, in transportation, medicine or cyber security.

Despite many possibilities and applications, AI is currently gaining the most popularity media through competitive games, where the main task is to show the superiority of algorithms over humans. Today, there are many such events, where world champions in a given game, lost to recognition models.

In 2016, there was an organized match between Fan Hui, the European champion in the Chinese game Go and the algorithm AlphaGo. The model created by the DeepMind team won against the opponent. Until now, the game was widely considered to be complicated and difficult to solve.

In 2019, a model OpenAI Five as first AI in the world, beat Team OG in the e-sport competition in the game Dota 2. The event was heavily covered in the media because of the first such achievement in this sport. Additionally, Dota 2 was a very complex environment. For example, the game Go solved a few years earlier, contained 150 possible moves per turn, Dota 2 could have 20,000 of them in less than an hour [2].

There are many such events. They show that today's artificial intelligence can surpass humans in strategic thinking. Additionally, it prove that AI topic is still in developed with more and more interest each year.

## 1.1 Genesis of the topic

Often in developing ML programs, a major challenge is the level of complexity of the game. This depends on whether the environment is deterministic or stochastic, how dynamic game is or whether the dimensional space is discrete or infinite. Currently, however, one of the major problems of such programs is the insufficient extent of available information about the environment [5].

Artificial intelligence, in order to win, needs a large number of unambiguous inputs for subsequent learning. An example of a game that meets such requirements is chess. AI makes moves based on information such as the arrangement of pawns in a given turn along with previous settings. The change in the environment on the opponent's side is immediately visible so model can easier bind actions and observations.

An example of a game hard to teach AI that does not meet this condition is Poker Texas Hold'em. It exhibits an incomplete set of information, despite knowledge of the cards in the hand and on the table, the player has no knowledge of opponent cards. In this case, two seemingly identical states of the environment in in reality may differ. In addition, the environment is characterized by very high randomness which is hard to predict good moves. Because of these characteristics, most popular algorithms such as DQN (*Deep Q Learning*), Actor-Critic or AlphaZero become useless and do not give good results.

This paper presents a way to possibly solve such a problem using the Deep CFR algorithm [3]. This is a popular method for creating models of recognition in card games.

## 1.2 Goal

The main goal of this paper is to implement a Deep CFR algorithm that will create 5 models of recognition in the game HULH (*Heads Up Limit Texas Poker Hold'em*). This is a popular version of 2-player play where participants cannot choose the amount of the raise on their own. It is limited by a set value. This environment minimizes possible moves to 3 actions, making it a simpler base for machine learning. The resulting models will then be used to create games consisting of all combinations of the two models, where each game will be repeated 200 times. The second stage will be to calculate the average pool won and lost by each AI along with the distribution of of moves made. This process will determine which model performs best and what strategy it uses to play the game.

The work has been divided into 5 stages for this purpose. The next section is a theoretical introduction to the implementation of the program. It describes possible solutions to the problem, analysis of the game Poker Texas Hold'em and the required theory for understanding Deep CFR. The third section presents the written program along with the technologies used. The final two chapters discuss the learning results, the results model games, and a summary of the work.

# Chapter 2

# Analysis of existing solutions

W ciagu ostatnich 15 lat powstało wiele algorytmów rozwiazujacych różne wersje gry Poker. Miedzy innymi CFR (*Counterfactual Regret Minimization*) [5], XFP (*Extensive-Form Fictitious Play*) [4] lub NFSP (*Neural Fictitious Self-Play*) [6]. Pierwszy z wymienionych, CFR powstał w 2007 roku. Był pomyślna prób rozwiazania abstrakcyjnego środowiska Poker Texas Hold'em [5]. Na jego podstawie utworzono wiele nowoczesnych algorytmów, które daja szanse rozwiazać takie gry jak HULH [5].

Z wymienionych metod zaimplementowanym rozwiazaniem w niniejszej pracy jest CFR rozszerzony o sieci neuronowe, czyli Deep CFR z gra HULH. Pozwala on na szybsze trenowanie modeli w środowisku typu zero-sum, dodatkowo lepiej rozwiazuje gry o dużych rozmiarach [3].

W tym rozdziale zostana przedstawione profesjonalne sposoby wyboru strategii w grze Poker Texas Hold'em. Określa one cechy, jakimi powinien charakteryzować sie prawidłowo utworzony model rozpoznawania. Nastepnie rozdział przedstawi istniejace sztuczne inteligencje, które wykorzystały podstawy metody CFR do zwyciężania z profesjonalnymi graczami. Ostatnim etapem jest teoria zwiazana z algorytmem Deep CFR.

## 2.1 Analiza Texas Hold'em Poker

Jest to jedna z najpopularniejszych gier rywalizacyjnych w kasynach, dodatkowo jest to dominujaca gra hazardowa. Można ja scharakteryzować duża stochastycznościa oraz cześciowa obserwowalnościa, tab. 2.1 [7].

Przez takie cechy gra była od zawsze tematem sporów, czy na jej wynik ma wiekszy wpływ losowość, czy umiejetności. Dużym aspektem pomagajacym w osiagnieciu zwycięstwa jest panowanie nad emocjam w celu ukrycia informacji o posiadanych kartach przez przeciwnikiem. Druga zasada jest umiejetność decydowania kiedy grać agresywnie, a kiedy pasywnie.

Table 2.1: Charakterystyka wybranych gier

| | środowisko deterministyczne | środowisko niedeterministyczne |
|---|---|---|
| pełny zestaw informacji | szachy<br>Go | Monopoly<br>Tetris |
| niepełny zestaw informacji | Saper<br>Mahjong | Poker<br>Makao |

Dodatkowo ważnym elementem jest obserwacja gry oraz wybór prawidłowej strategii. Można ja wybrać na bazie dostepnych kart i dotychczasowego zachowania oponenta.

Kolejna ważna zasada jest obserwacja przeciwnika oraz zapamietywanie jego poprzednich akcji. Przez to można określić, czy gra on agresywnie, czy pasywnie i dobrać do niego odpowiednia strategie. W tym celu dokonuje sie klasyfikacji przeciwników na bazie czestotliwości wykonywanych ruchów [8].

Każdego gracza można podzielić na cztery grupy, Loose Aggressive, Losse Passive, Tight Passive oraz Tight Aggressive [8]. Prawidłowe rozpoznanie danego stylu gry może zadecydować o wyborze prawidłowej strategii i zwyciestwie. Poniżej opisano każdy z nich oraz porównano je z rys. 2.2, który pokazuje, w jakim stopniu profesjonalny gracz powinien używać każdego z nich [8].

### Loose Passive

Osoba, która bardzo czesto wchodzi do gry niezależnie czy posiadane karty daja jej wysokie szanse na wygrana. Taki sposób grania nie jest dobra strategia, ponieważ można łatwo sie do niego dostosować przez używanie tylko mocnych kart [8]. Taki styl jest czesto używany przez niedoświadczonych graczy.

### Loose Aggressive

Osobe, która czesto przebija stawke nawet w rundzie *pre-flop* i czesto wchodzi do gry [8]. Strategia ma stworzyć przekonanie wśród oponentów, że gracz ma bardzo duże szanse wygranej od samego poczatku. Okazuje sie ona jednak nieefektywna, jeśli przeciwnicy nie pasuja w poczatkowych etapach [8].

### Tight Passive

Uczestnik gry wchodzi tylko z dobrymi kartami, wykonujac czesto akcje *call*. Ostatecznie pasuje przy spotkaniu z graczem agresywnym. Taka osoba gra bardzo dokładnie tak, aby mało ryzykować, przez co czesto traci wiele okazji gdzie mogła by wygrać [8]. W rundach

traci ona bardzo małe stawki ale pomimo tego, bardzo rzadko wygrywa. W rezultacie stopniowo traci żetony przy grze z bardziej doświadczonymi graczami.

### Tight Aggressive

Graja podobnie do typu Tight Passive w poczatkowych etapach gry, a nastepnie zmieniaja swój styl na mniej lub bardziej agresywny tak aby wygrać runde po mimo wiekszego ryzyka [8]. Rys. 2.1 pokazuje, że jest to najlepsza wersja strategii, jaka można grać, dlatego czesto jest ona wybierana przez profesjonalnych graczy.

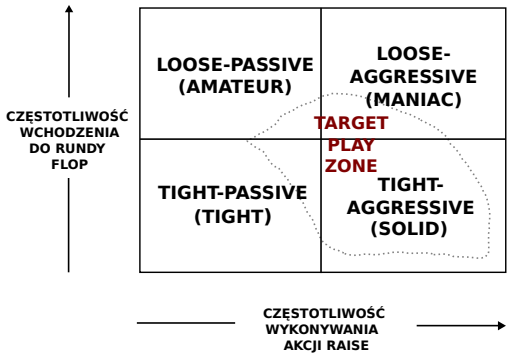


Figure 2.1: Podział graczy w grze Poker Texas Hold'em [8].

Jak wynika z powyższych kategorii, gra Poker Texas Hold'em zawiera wiele elementów niezwiazanych z losowościa, gdzie obserwacje i dobieranie odpowiedniej strategi do typu gracza pełni kluczowa funkcje. Korzystajac z tych informacji, bedzie można określić poziom zaawansowania i styl gry utworzonych modeli, które sa opisane w rozdziale 4.

## 2.2 Uczenie przez wzmacnianie

Jest wiele sposobów na stworzenie sztucznych inteligencji w grach. Miedzy innymi można użyć technik uczenia nadzorowanego pod warunkiem, jeśli przygotuje sie odpowiednie zbiory danych. W pracy jednak zdecydowano sie na stworzenie modelu, korzystajac z uczenia przez wzmacnianie, czyli rozwiazania gdzie AI nie potrzebuje wstepnej bazy uczacej. Wynika to z faktu, że jest mało publicznych zapisów gry Poker Texas Hold'em z profesjonalnymi graczami, które mogłyby posłużyć jako zbiory danych.

Algorytmy należace do wybranego działu powinny być w stanie polepszać swoje wyniki na podstawie interakcji ze środowiskiem bez korzystania z zewnetrznych materiałów. Wiele istniejacych metod należacych do wybranej techniki zakłada, że środowisko można opisać przez model matematyczny MDP (*Markov Decision Process*). Określa ona sekwencyjnie podejmowane decyzje wraz z uzyskiwana nagroda od środowiska [9]. W każdym z nowych

stanów, w jakich znajduje sie agent (AI), wykonuje on pojedyncza akcje $a$, zyskujac od środowiska informacje o nowym stanie $s$ oraz nagrodzie $r$. Elementem wyjściowym zasady powinien być zbiór strategii w postaci modelu, rys. 2.2.

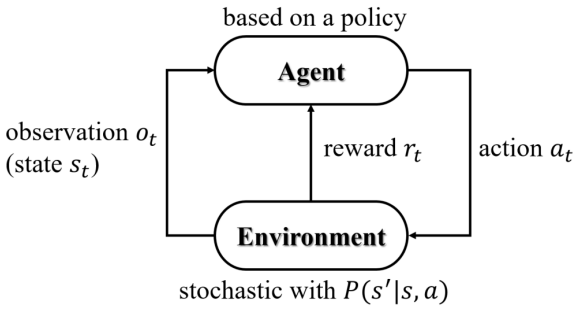


Figure 2.2: Schemat modelu matematycznego MDP [9].

MDP może opisywać jedynie środowiska z pełnym zakresem informacji, w przypadku algorytmu bedacego tematem pracy, głównym zadaniem jest rozwiazanie środowiska z niepełnym zestawem danych. Wtedy należy rozpatrywać zasade POMDP (*Partially Observable Markov Decision Process*) [9].

W przeciwieństwie do poprzedniego modelu w tym przypadku agent nie zna aktualnego stanu, w którym sie znajduje [9]. Przez takie okoliczności musi połaczyć zależnościa wykonywane akcie i obserwacje, a nie stany. Wiekszość gier karcianych można zakwalifikować do tego typu problemów [9].

## 2.3 Teoria Gier

Aby zrozumieć działanie algorytmów CFR, Deep CFR, MCCFR itd. należy zapoznać sie z podstawami działu matematyki o nazwie Teoria Gier. Bada on optymalne zachowanie w środowiskach, gdzie wystepuje konflikt [10]. W przypadku gry Poker Texas Hold'em zostały opisane terminy Równowaga Nasha oraz postać ekstensywna. Sa one obowiazkowe do zrozumienia algorytmu Deep CFR.

### Gra w postaci ekstensywnej

Gry w formie ekstensywnej można przedstawić jako drzewo decyzyjne, gdzie każdy wezeł rozgałezia sie na możliwe akcje oraz identyfikuje aktualny stan gracza przez zestaw informacji. Stany końcowe drzewa określaja zysk lub strate nagrody wybranego gracza [3]. Jest to sposób na uproszczenie opisu gry, gdzie ruchy sa wykonywane nierównocześnie.

Na rys. 2.4 przedstawiono przykład gry 'Papier-Kamień-Nożyce' w formie ekstensywnej, gdzie gracze P1 i P2 eksploruja 3 akcje w swoich wezłach. Każdy z nich uzyskuje wyniki danych ścieżek oraz zapamietuje dotychczasowa historie, co może zostać potem wykorzystane do znalezienia najbardziej opłacalnych strategii.

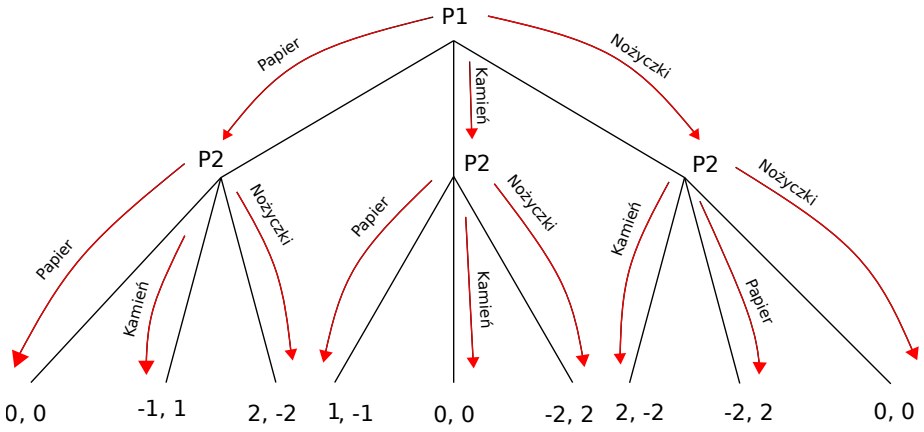


Figure 2.3: Opis gry w postaci ekstensywnej.

### Równowaga Nasha

W grach to twierdzenie określa perfekcyjny stan wyboru akcji, gdzie wszyscy gracze wykorzystuja najlepszy zestaw strategii, którego zmiana przyniesie tylko straty. Oznacza to, że nie jest możliwa zmiana ruchów na lepsze oraz zwiekszenie uzyskanej nagrody przy spełnieniu tej zasady [10].

Dobrym przykładem prezentujacym taki stan jest "Dylemat Wieźnia" [10]. Takie środowisko zawiera dwóch przestepców, którzy sa przesłuchiwani w odseparowanych pomieszczeniach co oznacza, że każdy z nich ma ograniczony zestaw informacji. Maja oni dwie opcje, przyznać sie do zarzutów lub tego nie robić. Każda z kombinacji akcji jest zaprezentowana w tab. 2.2, gdzie wartości określaja lata spedzone w wiezieniu po danym ruchu. Posługujac sie Równowaga Nasha, można stwierdzić, że najlepsza opcja dla obu uczestników bedzie przyznawanie sie za każdym razem [10]. Wynika to z faktu, że uczestnicy wtedy nie ryzykuja.

Głównym zadaniem wiekszości algorytmów gier karcianych bazujacych na metodzie CFR jest znalezienie takiego stanu. Deep CFR odkrywa zbiory strategii, które sa bliskie Równowadze Nasha w grach karcianych [3].

Table 2.2: Wyniki akcji środowiska "Dylemat wieźnia".

| | wieźnia A<br>sie przyznaje | wiezień A<br>kłamie |
|---|---|---|
| wieźnia B<br>sie przyznaje | 1<br>1 | 5<br>0.5 |
| wiezień B<br>kłamie | 0.5<br>5 | 0<br>0 |

## 2.4 Historia modeli Texas Hold'em Poker

Bazujac na Teorii Gier oraz innych twierdzeniach powstało wiele rozwiazań różnych wersji gry Poker. Pierwsze dokumenty naukowe omawiały bardzo proste środowiska rozwiazaywane przez algorytmy jak CFR [3].

Dopiero w 2015 roku utworzono znana sztuczna inteligencje "Cepheus" rozwiazujaca problem HULH [11]. Było to pierwsze takie osiagniecie w historii. Kolejnym etapem były prace nad algorytmami mogacym rozwiazać problem gry HUNH (*Heads Up No-limit Texas Hold'em*). Pierwszy powstały model w 2017 roku nazwano "DeepStack" [13]. Mieszał on sieci neuronowe z technikami algorytmu CFR. W podobnym czasie utworzono kolejne, bardziej zaawansowane AI, Libratus [12]. W 2019 roku pierwszy raz udało sie rozwiazać problem gry HUNH składajacej sie z 6 graczy. Zbudowano model o nazwie "Pluribus", który był pierwszym AI rozwiazujacym standardowa wersje pokera [19].

Pomimo takich osiagnieć tworzone modele do 2019 potrafiły grać tylko w środowiskach składajacych sie maksymalnie z 2 osób typu zero-sum [12] [11] [13]. Wynika to z poziomu skomplikowania gier cześciowo-obserwowalnych. Sposoby na jego rozwiazanie zaczeły powstawać od niedawna, a pierwsze dwa duże osiagniecia w grze HUNH miały miejsce dopiero w 2017 roku.

#### Cepheus

AI powstałe w celu wygrywania w grach HULH. Był to pierwszy model rozwiazujacy duża wersje gry Texas Poker Hold'em w historii [11]. Wykorzystał on nowsza wersje techniki CFR, która nazwano CFR+ [11]. W wyniku dwóch miesiecy nauki i testów nowa metoda zbiegała sie znacznie szybciej do Równowagi Nasha niż bazowy CFR [11]. Powstały model jest udostepniony publiczne, każdy może go przetestować.

#### DeepStack

Model DeepStack rozwiazał HUNH przez połaczenie metody CFR, sieci neuronowych wraz z dodatkowymi elementami. W rezultacie AI zaczeło osiagać bardzo dobre wyniki. Przetestowano go na 33 profesjonalnych graczach w wielu iteracjach gry. Model w wiekszości przypadków wygrał [13]. Była to pierwsza wygrana AI z człowiekiem w normalnej wersji gry Poker Texas Hold'em z taka czestotliwościa.

#### Libratus

Sztuczna inteligencja, która jest wykorzystywana w grach HUNH. Jak wynika z testów wygrywa znacznie cześciej niż DeepStack z profesjonalnymi graczami [12]. Przetestowano go z mistrzami gry, Dong Kim, Dan McAulay, Jimmy Chou i Jason Les [12]. AI wygrało z nimi, z ogromna przewaga.

## 2.5 Counterfactual Regret Minimization

### 2.5.1 Regret Matching

Jest to nieodłaczna metoda uczenia AI w grach karcianych. Polega ona na liczeniu najlepszej strategii pod warunkiem, że znany jest wektor żalu (*regrets*) w weźle [5]. Taki wektor opisuje sie jako tablice wag o długości równej liczbie możliwych akcji gracza. Każda z tych wag opisuje, jak dużym błedem bedzie niewykonania danego ruchu.

Poniżej przedstawiono wzór wynikajacy z tej metody, gdzie $R^t(a)$ jest omawianym wektorem [5]. Nastepnie, aby uzyskać nowa strategie, usuwa sie wartości ujemne, czyli takie, których gracz nie żałował (formuła nr 2.2). Potem sprawdzana sie, czy ich suma jest wieksza od zera w celu wybrania odpowiedniego wzoru, formuła 2.1. W zależności od tego warunku otrzymuje sie określony rozkład prawdopodobieństwa wykonania każdej z akcji.

$$p_i^t(a) = \begin{cases} \frac{R^{T,+}(a)}{\Sigma_{a' \in A} R^{T,+}(a')} & \text{if } \Sigma_{a' \in A} R^{T,+}(a') > 0; \\ \frac{1}{|A|} & otherwise. \end{cases} \tag{2.1}$$

$$R^{t,+}(a) = max(R^t(a), 0) \tag{2.2}$$

Proces ten jest powtarzany wielokrotnie, tak, aby przy każdej iteracji rozkłady prawdopodobieństwa ruchów były stopniowo poprawiane.

W przypadku algorytmu Deep CFR zachodzi modyfikacja formuły nr 2.1. Strategia jest liczona na dodatnich wartościach żalu podzielonego przez prawdopodobieństwo dostania sie do tego stanu $D^T(I, a)$ [3]. Jeśli suma jest ujemna, to zostaje wybrana akcja z najwyższa wartościa $D^T(I, a)$ [3].

$$\sigma_i^{t+1}(I, a) = \begin{cases} \frac{D^{T,+}(a)}{\Sigma_{a' \in A} D^{T,+}(a')} & \text{if } \Sigma_{a' \in A} D^{T,+}(a') > 0; \\ \text{argmax}(D^T(I, a)) & otherwise. \end{cases} \tag{2.3}$$

### 2.5.2 Counterfactual Regret

Algorytm CFR do znanych wcześniej metod dodał termin 'Immediate Counterfactual Regret' oznaczany przez $R^T_{i,imm}(I)$, czyli żal przydzielony do wezła I. Do obliczenia takiego parametru została zdefiniowana wartość "counterfactual utility" $u_i(\sigma, I)$. Oznacza ona przewidywa opłacalność stanu I gdzie wszyscy gracze używaja strategi $\sigma$ [5]. Dodatkowo $\pi^\sigma(h, h')$ oznacza prawdopodobieństwo dostania sie z historii h do nowego stanu h' przy strategii $\sigma$ [5].

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi^\sigma_{-i}(h) \pi^\sigma(h, h') u_i(h')}{\pi^\sigma_{-i}(I)} \tag{2.4}$$

Na podstawie równania 2.5 można wyliczyć końcowa wartość żalu w algorytmie CFR.

$$R^T_{i,\text{imm}}(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi^{\sigma^t}_{-i}(I)(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I)) \tag{2.5}$$

Powyższe 2 równania można doprowadzić do formuły nr 2.6. Wartość $\pi^\sigma(h,h')$ została zastapiona przez liczbe 1, ponieważ CFR zakłada, że dla $u_i(\sigma^t|_{I\to a}, I)$, gracz wykonuje zawsze akcje $a$ z całkowita pewnościa [5].

$$R^T_{i,\text{imm}}(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi^{\sigma}_{-i}(h) \sum_{h\in I, h'\in Z} (1 * u_i(h') - \pi^\sigma(h,h')u_i(h')) \tag{2.6}$$

Po uzyskaniu $R^T_{i,\text{imm}}(I,a)$ można wykorzystać metode "Regret Matchning" i zaktualizować strategie. Poniżej przedstawiono przykład obliczeń pojedyńczego wezła P2 oraz wyniki dla gry "Papier-Kamień-Nożyce" przy ustawionych nagrodach i karach w stanach końcowych jak na rys. 2.4, 2.5, 2.6.

$$\pi^\sigma(h,h')u_i(h') = (\frac{1}{3}\cdot 0) + (\frac{1}{3}\cdot -1) + (\frac{1}{3}\cdot 2) = \frac{1}{3}$$

$$T * R^T_{i,\text{imm}}(I,a) = ((0-\frac{1}{3}),(-1-\frac{1}{3}),(2-\frac{1}{3}))\cdot\frac{1}{3} = (-\frac{1}{9},-\frac{4}{9},\frac{5}{9})$$

$$\sigma_i^{t+1}(I,a) = (0,0,1)$$

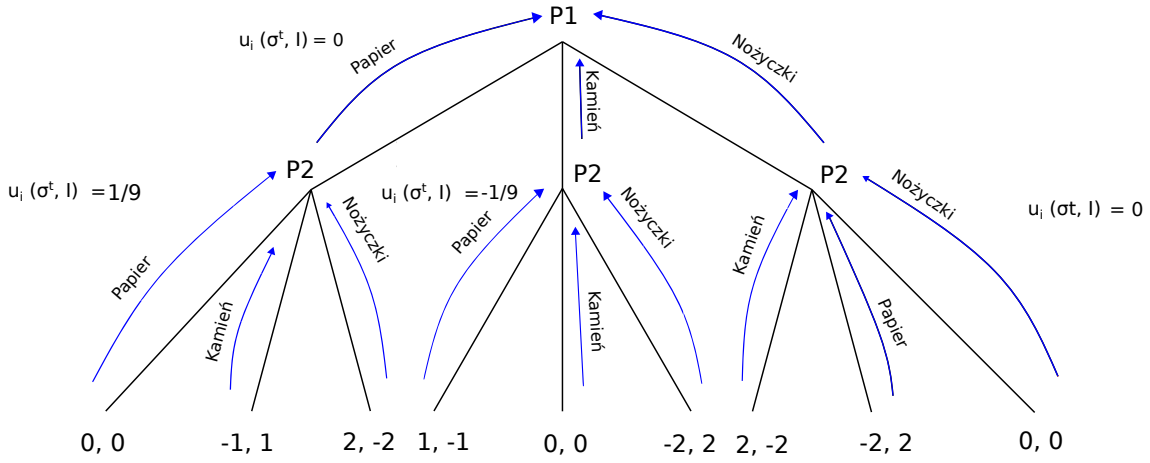


Figure 2.4: Przykład obliczonych wartości 'counterfactual utility'.

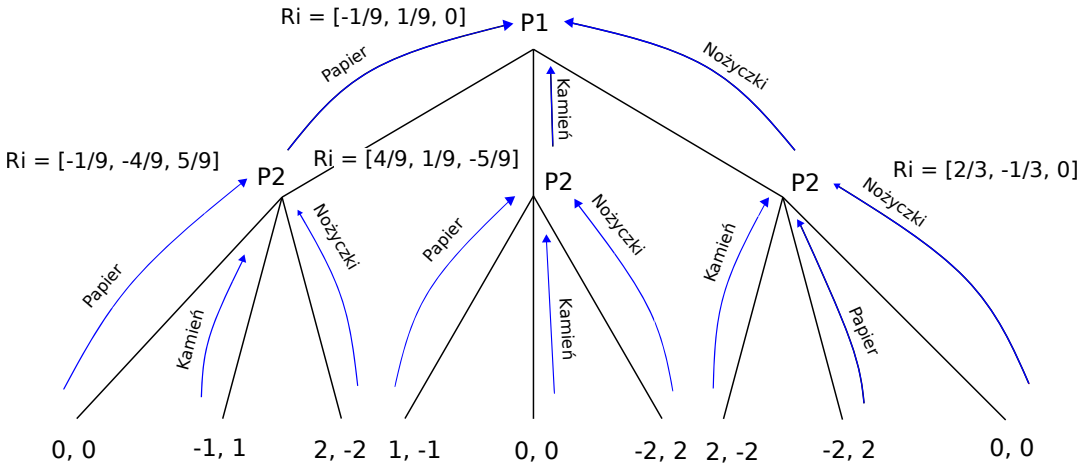


Figure 2.5: Przykład obliczonych wartości 'Immediate Counterfactual Regret'.

Na podstawie rys. 2.4 widać, że gracz P2 bedzie miał stan o najwyższej wartości "counterfactual utility" w weźle $u_i(\sigma^t, I) = \frac{1}{9}$, a najniższej dla $u_i(\sigma^t, I) = -\frac{1}{9}$.

Powyżej zaprezentowano wektory $R^T_{i,\text{imm}}(I, a)$. Gracz P1 grajac, najbardziej bedze żałował nie wykonania akcji *Kamień*, a najmniej *Papier*.

Rys. 2.6 przedstawia wektory określajace jakimi rozkładami akcji powinni sie kierować gracze, aby osiagnać najlepsze wyniki. Sa to wektory wyliczone po pierwszej iteracji, w praktyce eksploracja drzewa i powyższe obliczenia sa powtarzane wielokrotnie. Końcowym etapem algorytmu CFR jest policzenie średniej strategii, która ma reprezentować Równowage Nasha [5].

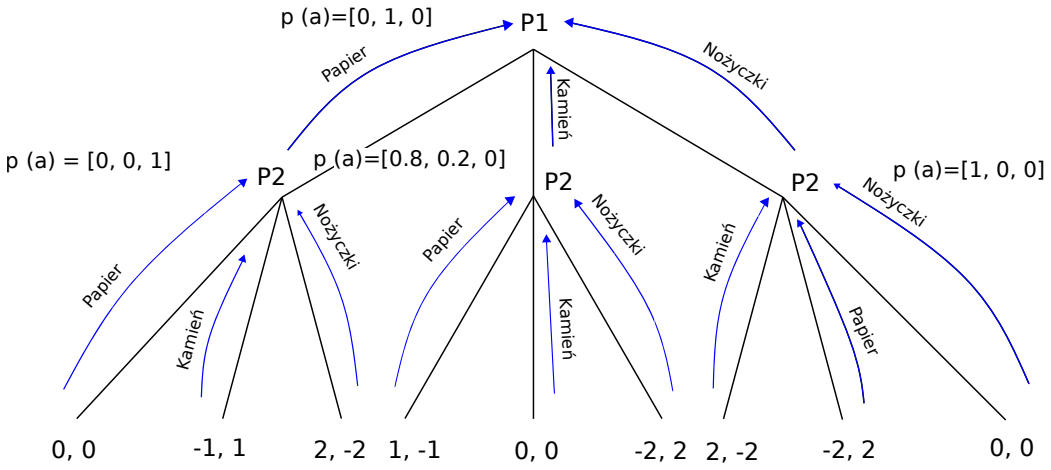


Figure 2.6: Przykład obliczonych strategii.

## 2.6 Monte Carlo Conterfactual Regret Minimization

Algorytm CFR eksploruje całe drzewa decyzyjnego w jednej iteracji, co tworzy wymagania na duża moc obliczeniowa i długi czas uczenia. Dla małych gier takie rozwiazanie jest akceptowalne, ale w przypadku wiekszych środowisk jest nieefektywne. Spowodowało to powstanie nowszej wersji algorytmu CFR, MCCFR (*Monte Carlo Conterfactual Regret Minimization*) , który w każda iteracje eksploruje tylko cześć drzewa [14].

Metode można podzielić na dwie odmiany, Outcome-Sampling oraz External-Sampling [14].

W pracy został przedstawiony sposób MCCFR ES (*Monte Carlo Conterfactual Regret Minimization External Sampling*), ponieważ taki jest zaimplementowany w algorytmie Deep CFR. MCCFR ES przed eksploracja drzewa wybiera kolejno spośród graczy jednego uczestnika, którego oznacza sie jako "traverser" [3]. Eksploruje on wszystkie odpowiedzi ze swoich akcji w danym weźle. W miedzyczasie inni uczestnicy wykonuja pojedynczy ruch na podstawie swojej najlepszej strategii [14].

Na rys. 2.7 przedstawiono przykład cześci drzewa HULH, gdzie gracz P2 został wybrany jako "traverser". Jak można zauważyć, tylko jego wezły rozgałeziaja sie na wszystkie możliwe ścieżki. Dodatkowo w drodze powrotnej obliczono $u_i(\sigma, I)$ dla wszystkich stanów używajac wzoru 2.5.

Jest to przykład z jednej eksploracji gry, w praktyce powtarza sie ten proces wielokrotnie,

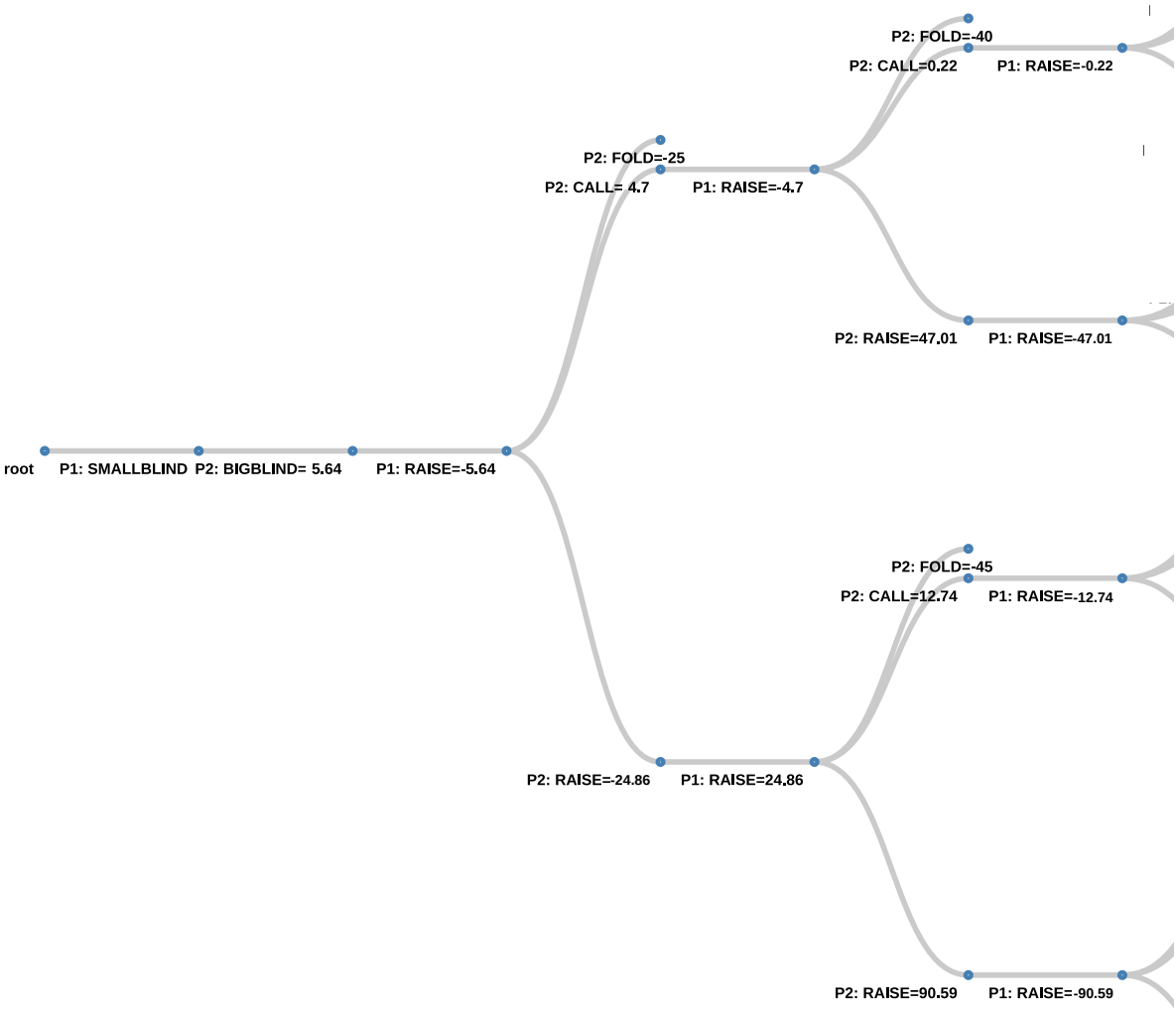


Figure 2.7: Cześć drzewa decyzyjnego MCCFR ES z graczem P2 jako "traverser".

uzyskujac różne wersje drzew.

Majac takie wyniki, można przystapić do liczenia $R^T_{i,imm}(I)$ oraz poprawiania wartości przez "Regret Matching". MCCFR ES spełnia swoja funkcje, ale wymaga wielu iteracji, aby uzyskać dobre wyniki. Z tego powodu w dalszym rozdziale zostanie przedstawiona metoda Deep CFR, która przyspiesza proces uczenia przez użycie sieci neuronowych.

## 2.7 Deep CFR

Algorytm Deep CFR opisany w [3] rozwija podstawowa wersje metody CFR o sieci neuronowe. Taka modyfikacja była wymagana, aby utworzyć AI, które może rozwiazać duże gry jak HULH znacznie szybciej. Liczy on wektory w drzewie decyzyjnym przez algorytm MCCFR ES. Dodatkowo Deep CFR okazał sie lepszy niż popularny algorytm NFSP z 2016 roku [3].

Deep CFR wykorzystuje sieci neuronowe do przewidzenia wartości $D^T$(I, a) w podobnych

obserwacjach, potem na podstawie predykcji liczy strategie ze wzoru nr 2.3. Nastepnie liczona jest zaktualizowana wersja wektora żalu, formuła nr 2.5. Obliczone wyniki sa dodawane do buforów ($B_1$, $B_2$) $\in B_p$, a strategia do zbioru $B_s$.

Po wielu iteracjach rozpoczyna sie nauka sieci z zebranej bazy. Poniżej przedstawiono dokładny opis Deep CFR.

**Algorithm 1:** Deep CFR

**Wejście:** $B_p$, $B_s$, $\theta_s$, $\theta_p$, $P$, $N$, $K$
**Wyjście:** $\theta_s$

```
for n in N do
    h, t ← Nowa gra
    for p in P do
        for k in K do
            MCCFRES(θ_p, θ_{p−1}, p, t, h, B_p, B_s)
        θ_p ← TRAIN(B_p, θ_p, t)        ▷ loss: 1/N_batch ∑(y_i − ŷ_i)^2 · t_i
θ_s ← TRAIN(B_s, θ_s, t)                ▷ loss: 1/N_batch ∑(y_i − ŷ_i)^2 · t_i
return θ_s
```

Algorytm na wejściu dostaje argumenty przystosowane do gry 2-osobowej. Pierwszymi elementami sa bufory graczy ($B_1$, $B_2$) oraz kontener na strategie $B_s$. Dodatkowo metoda potrzebuje liste uczestników $P$, z których bedzie wybierany cyklicznie gracz eksplorujacy drzewo decyzyjne. Ostatnimi argumentami sa iteracje gry $N$ oraz liczba cykli $K$ metody MCCFR ES.

Podsumowujac należy ustawić trzy petle wraz z nowa runda, uzyskujac poczatkowa historie oraz krok gry t. Kolejnym etapem jest wykonanie funkcji *MCCFRES*. Przyjmuje ona na wejściu sieci neuronowe obu graczy, liste uczestników p, numer kroku t, historie rundy h oraz bufory. Po k powtórzeniach nastepuje uczenie sieci neuronowej wybranego wcześniej gracza eksplorujacego drzewo.

Model używa zmodyfikowanej wersji funkcji MSE (*Mean Square Error*) do liczenia błedu predykcji. Każdy wynik $(y_i - \hat{y}_i)^2$ mnożony jest przez krok $t_i$ w, którym uzyskano $y_i$ [3]. Dodatkowo jak wynika z badań, algorytm osiaga lepsze wyniki jeśli sieci neuronowe sa trenowane od poczatku [3]. Dlatego należy przed funkcja *TRAIN* wyczyścić model.

Po wszystkich powtórzeniach i zebraniu całej bazy bufora $B_s$, rozpoczyna sie trenowanie sieci neuronowej $\theta_s$ w ten sam sposób jak inne modele. Elementem wyjściowym Deep CFR jest sieć $\theta_s$.

**Algorithm 2:** Implementacja MCCFRES kożystajac z sieci neuronowych

**Function** MCCFRES($\theta_p$, $\theta_{p-1}$, $p$, $t$, $h$, $B_p$, $B_s$):
  $t_r \leftarrow h$ ▷ sprawdzenie czyja jest aktualnie tura
  **if** *h jest stanem końcowym Z* **then**
    **return** $u_p(h)$
  **else if** $p = t_r$ **then**
    $\hat{D}(I) \leftarrow$ utworzenie wektora $\hat{D}(I)$ z sieci $\theta_p$
    $\sigma(I) \leftarrow$ obliczenie strategi $\sigma_{t_r}(I)$, kożystajac z $\hat{D}(I)$ i wzoru 1.1
    **for** *a* ***in*** *A(h)* **do**
      $u_{t_r}$(ha) $\leftarrow$ MCCFRES($\theta_p$, $\theta_{p-1}$, $p$, $t+1$, $h+a$, $B_p$,$B_s$)
    $u_{r_r}(\sigma, I) \leftarrow \sum(u_{t_r}(ha) \cdot \sigma(I))$
    $R^T_{t_r}$(I) $\leftarrow$ ($u_{t_r}(ha) - u_{t_r}(\sigma, I)$)
    $B_{t_r} \leftarrow$ dodanie próbki [$R^T_{t_r}$(I), h, t] do bufora
  **else**
    $\hat{D}(I) \leftarrow$ utworzenie wektora $\hat{D}(I)$ z sieci $\theta_p$
    $\sigma(I) \leftarrow$ obliczenie strategi $\sigma_p(I)$, kożystajac z $\hat{D}(I)$ i wzoru 1.1
    $B_s \leftarrow$ dodanie próbki $[\sigma(I), h, t]$ do bufora
    a $\leftarrow \sigma(I)$
    **return** MCCFRES($\theta_p$, $\theta_{p-1}$, $p$, $t+1$, $h+a$, $B_p$, $B_s$)

Implementacja MCCFR ES przedstawiona powyżej przy zadanych argumentach rozpoczyna sie od sprawdzenia, który gracz rozpoczyna dana ture. Na podstawie tej informacji bedzie wykonywana dalsza cześć algorytmu.

Pierwszym krokiem jest sprawdzenie, czy stan gry jest końcowym etapem. W przypadku prawdziwego warunku zwracana jest wygrana lub przegrana wartość stawki. Jeśli powyższy etap jest fałszywy, algorytm sprawdza, czy gracz jest oznaczony jako "traverser". Wtedy program używajac sieci neuronowej gracza, otrzymuje wektor $\hat{D}(I)$ przez wprowadzenie do modelu informacji o widocznych kartach oraz dotychczasowej historii gry. Korzystajac ze wzoru 2.3, liczy strategie, odczytuje wektor $u_{t_r}$ (ha) wykonujac rekurencje. Ostatnim krokiem jest wyliczenie wektora żalu i dodanie go do bufora.

Jeśli powyższy warunek był fałszywy, gracz liczy nowy rozkład akcji i dodaje go do zbioru strategii. Nastepnie korzystajac z tej sieci neuronowej i otrzymanej dystrybucji wykonuje nowa akcje.

## 2.8 Podsumowanie

Środowiska z niepełnym zestawem informacji i brakiem deterministyczności sa trudne do rozwiazania. Algorytmy tworzace modele dla takich gier sa czesto skomplikowane i obciażajace obliczeniowo. Przez takie cechy dopiero od niedawna zaczeły powstaja algorytmy, zdolne pokonać ludzi w dużych grach karcianych jak "Cepheus", "DeepStack", "Libratus" lub "Pluribus". Metody zdolne tworzyć takie AI dalej sa rozwijane i aktualizowane z roku na rok. W taki sposób w 2007 roku powstał CFR, potem MCCFR i

po para latach zastapiono je przez CFR+, aż zaczeto wykorzystywać sieci neuronowe w takich algorytmach jak Deep CFR bedacy tematyka pracy.

Rozdział dokładnie opisał działanie omawianego algorytmu przez przedstawienie terminów należacych do działu matematyki Teoria Gier. Miedzy innymi pokazał, że opis środowiska przez drzewa decyzyjnych pozwala na wiele uproszczeń i możliwości śledzenia gry. Dodatkowo przedstawiono problem poszukiwania stanu Równowagi Nasha, którego znalezienie jest celem wiekszości algorytmów sztucznej inteligencji gier karcianych.

Dobrze zaimplementowany algorytm Deep CFR przy prawidłowej parametryzacji i odpowiednio dużej liczby iteracji powinien zbiec sie do punktu bliskiego takiego stanu. Pomimo tak optymistycznych założeń utworzonie AI potrzebuje bardzo długiego czasu nauki do znalezienia dobreych strategii gry w HULH i HUNH.

# Chapter 3

# Implementacja algorytmu

Program Deep CFR został napisany w niniejszej pracy, korzystajac z narzedzi, pozwalajacych na zredukowanie nadmiarowości kodu oraz na prosta implementacje. W tym rozdziale skupiono sie na opisaniu wybranych technologii, parametrów oraz funkcji, które znalazły sie w implementacji algorytmu.

Baza do napisanego kodu jest jezyk programowania, Python 3.8. Zawiera on wiele technologii wspomagajacych uczenie maszynowe i rozległa społeczność wspierajaca jego rozwój. Poniżej wylistowano i opisano główne narzedzia użyte w pracy.

**TensorFlow** Jest to wysokopoziomowe API dostepne dla takich jezyków jak Python, JavaScript, C++ lub Java [21]. Wykorzystuje je sie głównie do zadań głebokiego uczenia maszynowego. Przez swoja prostote, dostepność i dobra dokumentacje stał sie jednym z najpopularniejszych narzedzi wykorzystywanych do tworzenia sztucznych inteligencji. Dodatkowo od niedawna biblioteki technologi Keras stały sie cześcia Tensorflow. Daje to możliwości znacznego zredukowania kodu przy prostych problemach, które czesto sa rozwiazane przez funkcje w tym module.

W przypadku niniejszej pracy głównie korzystano z funkcji zawartych w bibliotekach Keras. Wyjatkiem sa nieliczne wiersze w kodzie gdzie np. było wymagane wykonanie obliczeń na tensorach.

**Numpy** Duża biblioteka do naukowych obliczeń na wielowymiarowych tablicach. Jest nieodłacznym elementem przy pisaniu programów uczenia maszynowego, zwłaszcza jeśli korzysta sie z bibliotek Tensorflow. Wynika to z faktu, że wiele funkcji tego API, jako argumenty przyjmuje typy danych powiazane z Numpy [21].

**Tqdm** Małe narzedzie w jezyku Python pozwalajace na wyświetlenie postepu procesów w działajacym programie. Przydatne w celach testowych. W pracy zostało użyte do śledzenia iteracji drzew decyzyjnych algorytmu Deep CFR.

**TensorBoard** Moduł należacy do API Tensorflow. Wizualizuje postepy uczenia sieci neuronowych oraz ich jakość przez przedstawienie odpowiednich wykresów.

**PyPokerEngine** Biblioteka wspomagajaca symulacje gry Poker Texas Hold'em wraz z dodatkowymi elementami. Użyto jej jako podstawe do napisania środowiska HULH.

## 3.1 Implementacja sieci neuronowych

Algorytm Deep CFR do działania wymaga dwóch typów sieci neuronowych, jedna ma rozpoznawać strategie $\sigma^t$, a druga przypisana do określonego gracza przewiduje wartości $D_p^t$. Dodatkowo każdy z tych elementów jest trenowany na podstawie cyklicznie aktualizowanych buforów $B_s$ i $B_p$. W tym rozdziale zostanie przedstawiona dokładna implementacja modeli, proces ich uczenia, struktura zbiorów danych oraz budowa środowiska, z którym jest wykonywana interakcja. W tab. 3.1 i 3.2 zamieszczono podstawowe informacje o parametrach sieci. Dalsza cześć rozdziału dokładnie opisuje te parametry i dodatkowe elementy, ważne podczas uczenia modelu.

Table 3.1: Podstawowe parametry sieci neuronowej $\theta_s$.

| parametr | użyte wartości |
|---|---|
| predkość uczenia | 0.0001 |
| końcowa funkcja aktywacyjna | *softmax* |
| rozmiar bufora | 600 000 |
| maksymalna liczba iteracji | 5 000 |
| rozmiar *minibatch* | 500 |
| parametr *patience* | 40 |

Table 3.2: Podstawowe parametry sieci neuronowej $\theta_p$.

| parametr | użyte wartości |
|---|---|
| predkość uczenia | 0.0001 |
| końcowa funkcja aktywacyjna | *linear* |
| rozmiar bufora | 300 000 |
| maksymalna liczba iteracji | 5 000 |
| rozmiar *minibatch* | 500 |
| parametr *patience* | 40 |

### 3.1.1 Architektura modelu

W algorytmie zaimplementowano trzy sieci neuronowe $\theta_1$, $\theta_2$, $\theta_s$ o nieskomplikowanej architekturze. Z tego powodu wykorzystano biblioteke Keras, która do takich przypadków jest dobrym rozwiazaniem. Dodatkowo założono, że wszystkie modele beda miały podobna budowe poza ostatnia warstwa z innymi funkcjami aktywacyjnymi. Wynika to z faktu, że algorytm Deep CFR korzysta z sieci, które dostaja dane wejściowe o tej samej strukturze, ale zwracaja $D_p^t$ lub $\sigma_p^t$.

Architektura sieci neuronowych składa sie z 7 elementów, wejścia, wyjścia, bloku o nazwie *Flatten* [21], trzech warstw ukrytych zakończonych normalizacja i wyjściem w postaci wektora o trzech polach. Poczatkowo sieci dostaja tablice o wymiarach (3, 52), czyli kolumne kart ze stołu, z reki oraz historie ostatniej rundy gry. Model w dalszych obliczeniach musi przekształcić takie dane do formy jedno-wymiarowej (None, 156), co robi warstwa *Flatten*. Kolejne dwa elementy składaja sie z 512 wag, trzecia warstwa ukryta posiada ich 256. Na trzech wymienionych elementach ustawiono funkcje *Relu*. Całość kończy sie wyjściem wektora reprezentujacego możliwe akcje gry HULH. Dodatkowo dane sa poddawane normalizacji przez warstwe o nazwie *BatchNormalization* [21].

Wyjście modeli, aby zwracało prawidłowe liczby, używa innej funkcji aktywacyjnej niż poprzednie elementy. W przypadku predykcji strategii $\sigma_p^t$ wybrano *softmax*. Sieci neuronowe $\theta_1$, $\theta_2$ używaja funkcji *linear*. Dokładna architektura tych obiektów jest zaprezentowana na rys. 3.1. Dodatkowo sieci korzystaja z funkcji optymalizujacej Adam. Predkość uczenia jest równa 0,0001, co powoduje powolne uczenie, ale minimalizuje szanse na ominiecie minimum globalnego.

Jak wynika z dokumentu prezentujacego algorytm Deep CFR, uzyskuje on lepsze wyniki, jeśli sieci neuronowe sa uczone za każdym razem od poczatku przy losowo ustawionych zerach w wagach [3]. W tym celu dodano do każdej warstwy funkcje *RandomUniform* [21], która tworzy wartości losowo w przedziałach od -0,005 do 0,005.

Ostatnim elementem sieci jest funkcja liczaca błąd predykcji w trakcie uczenia. Jak wynika z opisu algorytmu Deep CFR, wymaga on zmodyfikowanej wersji MSE (*Mean Square Error*). Zostało to wykonane przy pomocy funkcji matematycznych na tensorach, jakie udostepnia Tensorflow [21]. Wszystkie te elementy zostały uwzglednione w funkcjach klasy *Poker_network* przedstawionej na rys. 3.4.

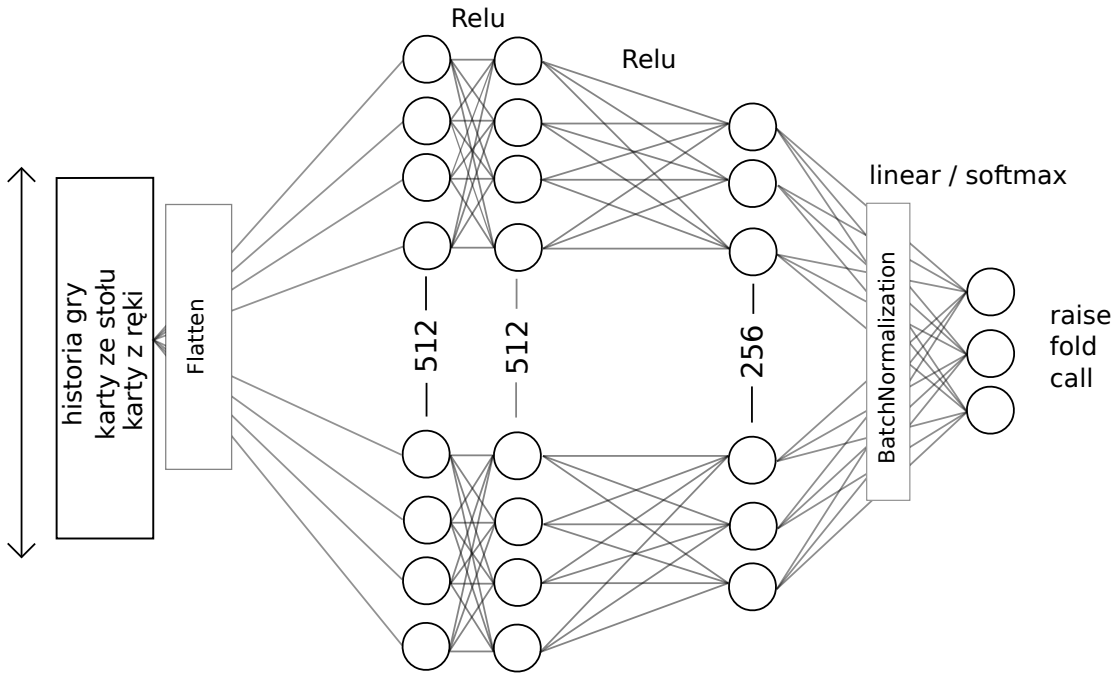


Figure 3.1: Architektura sieci neuronowych wykorzystana w programie.

### 3.1.2 Budowa zbiorów danych

Jak wynika z algorytmu Deep CFR, w programie musza być zawarte trzy bufory. Maksymalna pojemność kontenerów $B_1$ i $B_2$ jest równa 300 000 próbek. Bufor $B_s$ może posiadać ich 600 000. Każdy z dodawanych elementów do bufora składa sie z trzech połaczonych wektorów o rozmiarach 52. W tym celu zaimplementowano klase *Memory* zarzadzajaca tymi zbiorami. Działaja one jak kolejki i posiadaja typ danych *deque* co oznacza, że w przypadku przepełnienia jest usuwany najstarszy wpis.

Majac uzupełnione dane w tablicach, program rozpoczyna przygotowanie zbiorów danych do uczenia wybranych sieci neuronowych. Każdy z buforów zostaje losowo przetasowany i podzielony na dwa podzbiory. Pierwszy z nich to zbiór uczacy zajmujacy 80% całej bazy. Wykorzystuje sie go do poprawiania wag modelu sekwencyjnie. Drugim zbiorem sa dane walidacyjne. Dokonanie takiego podziału było wymagane, aby przeciwdziałać stanowi przetrenowania modelu. Zbiór walidacyjny jest nadzorowany i na jego podstawie można określić moment od którego model przestaje dobrze działać. Schemat tego podziału przedstawiono na rys. 3.2. Dodatkowo w trakcie nauki sieć neuronowa aktualizuje swoje wagi w każdym kroku przez użycie elementu o nazwie *minibatch*, bedacym parametrem określajacym mały podzbiór bazy użyty do trenowania sieci w pojedynczym kroku. Jego liczebność jest równa 500 próbek.

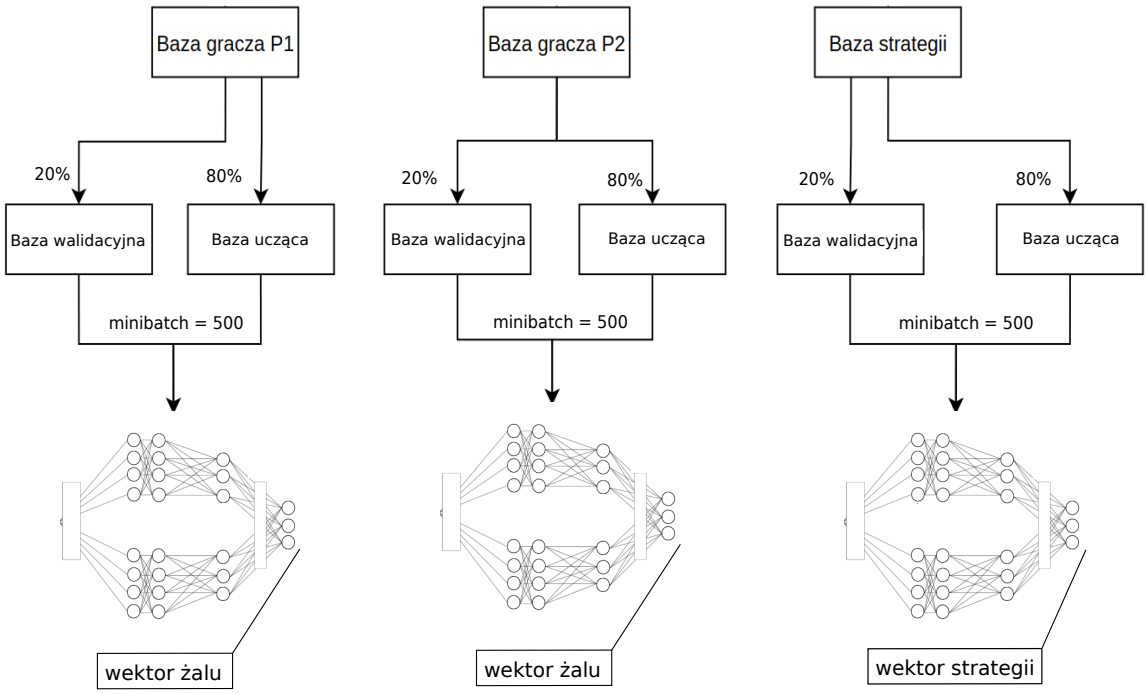


Figure 3.2: Podział danych.

### 3.1.3 Proces uczenia

Algorytm Deep CFR do gromadzenia obserwacji w buforach wykorzystuje metode MCCFR ES, która eksploruje w jednej iteracji wiele razy drzewo decyzyjne. Po zakończeniu wszystkich powtórzeń zachodzi etap uczenia jednej z sieci neuronowych $\theta_1$, $\theta_2$ wykonujac maksymalnie 5 000 aktualizacji wag pod warunkiem jeśli wcześniej nie dojdzie do dużych oscylacji błedu predykcji w zbiorze walidacyjnym. Taki schemat działania jest wykonywany dla obu graczy i powtarza sie wielokrotnie. Końcowym zadaniem jest wytrenowanie sieci neuronowej $\theta_s$. Wszystkie te elementy zostały połaczone przez klase *Brain*, agregujaca obiekty *Memory* oraz *Poker_network*, rys. 3.4.

W celu śledzenia i zatrzymania oscylacji wyników uczenia, które mogły by doprowadzić model do przetrenowania, użyto funkcji EarlyStopping [21]. Zatrzymuje ona iteracje modelu w przypadku kiedy bład predykcji na zbiorze walidacyjnym wzrośnie odpowiednio wysoko. Sprawdza sie to na podstawie argumentu *patience*, który określa próg, po którym nauka sie kończy. Taka procedura została zastosowana, aby polepszyć umiejetność generalizacji wyników przez model, a wraz z tym zwiekszyć jego jakość [15]. Rys. 3.3 prezentuje przykładowy punkt zakończenia nauki modelu w momencie przekroczenia progu *patience*.

Do samego trenowania sieci neuronowych wybrano urzadzenie GPU GeForce GTX 1050. Wynika to z uzyskiwania szybszych rezultatów w przeciwieństwie do innej możliwości, jaka jest wykorzystanie CPU.

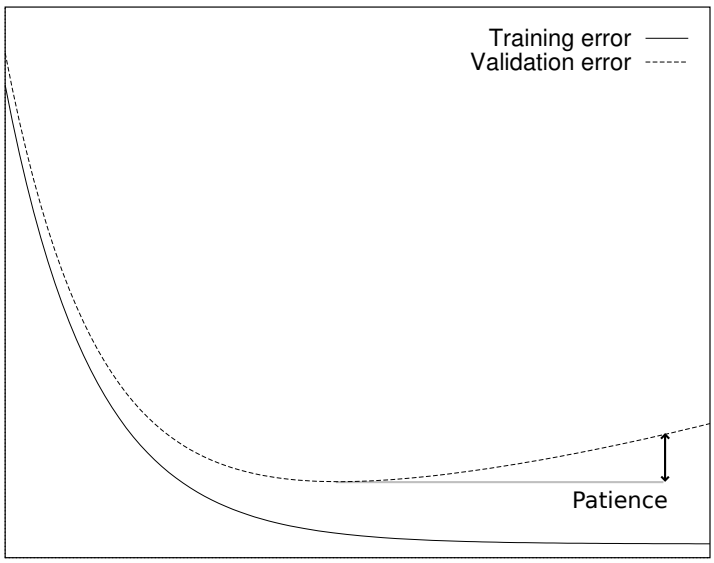


Figure 3.3: Przykład działania funkcji *EarlyStopping* [15].

## 3.2 Implementacja środowiska

Do symulacji gry HULH z która model bedzie sie komunikował, użyto biblioteki PyPokerEngine [20]. Klasa *HULH_Emulator* zaimplementowana w programie pozwala na utworzenie obiektu zarzadzajacego taka gra. Przy tworzeniu go zostaja zdefiniowane nazwy graczy *P1* i *P2*, które pełnia role identyfikatorów. W dalszej cześci działania algorytmu sa używane do śledzenia historii gry oraz wykonywania wszelkich akcji graczy. W tabeli nr 3.3 wylistowano parametry ustawione w programie, do rozpoczecia nauki.

Table 3.3: Parametry gry HULH.

| parametr | wartość |
|---|---|
| ante | 0 |
| mała w ciemno | 5 |
| duża w ciemno | 10 |
| liczba rund po których resetuje sie środowisko | 1 |
| udział każdego z graczy (stock) | 80 |
| liczba graczy | 2 |

Zaimplementowane środowisko po każdej wprowadzonej akcji zwraca dwa elementy, obiekt *State* określajacy stan gry oraz historie gry w formie słownika. Obie wartości sa używane do śledzenia postepu gry oraz pozycji w drzewie decyzyjnym. Dzieki takim obiektom *HULH_Emulator* może zwrócić graczowi informacje o dostepnych kartach, historii gry lub wygranych stawkach z dowolnego stanu wprowadzonego przez model.

Zaimplementowana gra symuluje HULH, czyli działa w taki sposób, że każde podbicie stawki jest wielkości dużej w ciemno. Dzieki takiemu założeniu gra nie kończy sie zbyt szybko pod warunkiem, że żaden z graczy nie wykona akcji *fold*. Dodatkowo ustalono, że po każdej rundzie środowisko jest resetowane. Taki proces został ustawiony na czas uczenia. Powodem jest powstawanie mniejszych drzew decyzyjnych, co pozwala na szybsza nauke modeli oraz krótszy czas poświecany na eksploracje gry przez MCCFR ES. W etapie rozgrywek miedzy modelami gra bedzie kończyć sie dopiero po wyczerpaniu sie żetonów po jednej ze stron.

## 3.3 Implementacja Deep CFR

Klasa *DCFR* zawiera implementacje algorytmu Deep CFR. Przy tworzeniu obiektu powstaja razem z nim trzy elementy klasy *Brain*, gracz, oponent oraz strategia $\sigma$ . Do tego ustawiane jest środowisko *HULH_Emulator* przez referencje.

Cały proces powstawania modelu rozpoczyna sie od funkcji *iterate*, która wykonuje trzy petle *for*, dla iteracji algorytmu po 50 razy oraz powtórzeń eksploracji drzewa dla każdego z gracza po 270 razy. W pierwszej z nich środowisko tworzy nowa gre, w ostatniej petli jest wykonywana funkcja *__traverse*, działajaca według metody MCCFR ES. Dostaje ona argumenty *state* czyli obiekt określajacy stan gry, *events* - dotychczasowa historie oraz *timestep*. Ostatni argument *verbose* jest opcjonalny i służy tylko do wizualizacji powstałego drzewa decyzyjnego. Wszystko działa przez rekurencje w celu stworzenia drzewa decyzyjnego. Po zakończeniu działania MCCFR ES rozpoczyna sie uczenie sieci neuronowej $\theta_p$. Po wszystkich powtórzeniach rozpoczyna sie trenowanie sieci $\theta_s$ oraz zapisanie jej w formie pliku. Aby spełnić warunek utworzenia pieciu modeli, kolejno ustawiono w funkcji *iterate* opcjonalny argument *checkpoint*. Przyjmuje on bazowa wartość *None*. Przy uruchamianiu programu zmieniono ja na liczbe 10, co oznacza, że co 10 iteracji bedzie trenowana sieć $\theta_s$ z uzbieranego bufora i zapisywana.

Po ustawieniu wszystkich parametrów uruchomiono program, który wykonywał sie przez trzy dni. Tab. 3.4 prezentuje podstawowe parametry zaimplementowanego algorytmu.

Table 3.4: Parametry algorytmu Deep CFR.

| parametr | wartość |
|---|---|
| liczba powtórzeń eksploracj drzewa | 270 |
| liczba iteracji algorytmu | 50 |
| co ile iteracji wytrenować i zapisać model | 10 |

## 3.4 Podsumowanie

W tym rozdziale zaprezentowano implementacje algorytmu wraz z użytymi technologiami. Program nastepnie został uruchomiony z zadanymi parametrami jak 50 iteracji po 270 eksploracji drzewa. Wykonywał sie on przez okres czterech dni, używajac GPU. Postepy tworzonych pieciu modeli były na bieżaco analizowane i zapisywane. Przedstawiono miedzy innymi strukture sieci neuronowych, budowe zbioru danych, działanie środowiska HULH oraz funkcje zwiazane z uczeniem algorytmu. Pokazano cel użycia takich metod jak *EarlyStopping* lub *Flatten*. Na rys. 3.4 przedstawiono dokładny schemat zaimplementowanego algorytmu. Nastepny rozdział przedstawi zapisane wyniki ilustrujace jakość utworzonych modeli oraz sprawdzi, który z nich bedzie najlepiej grał w HULH.

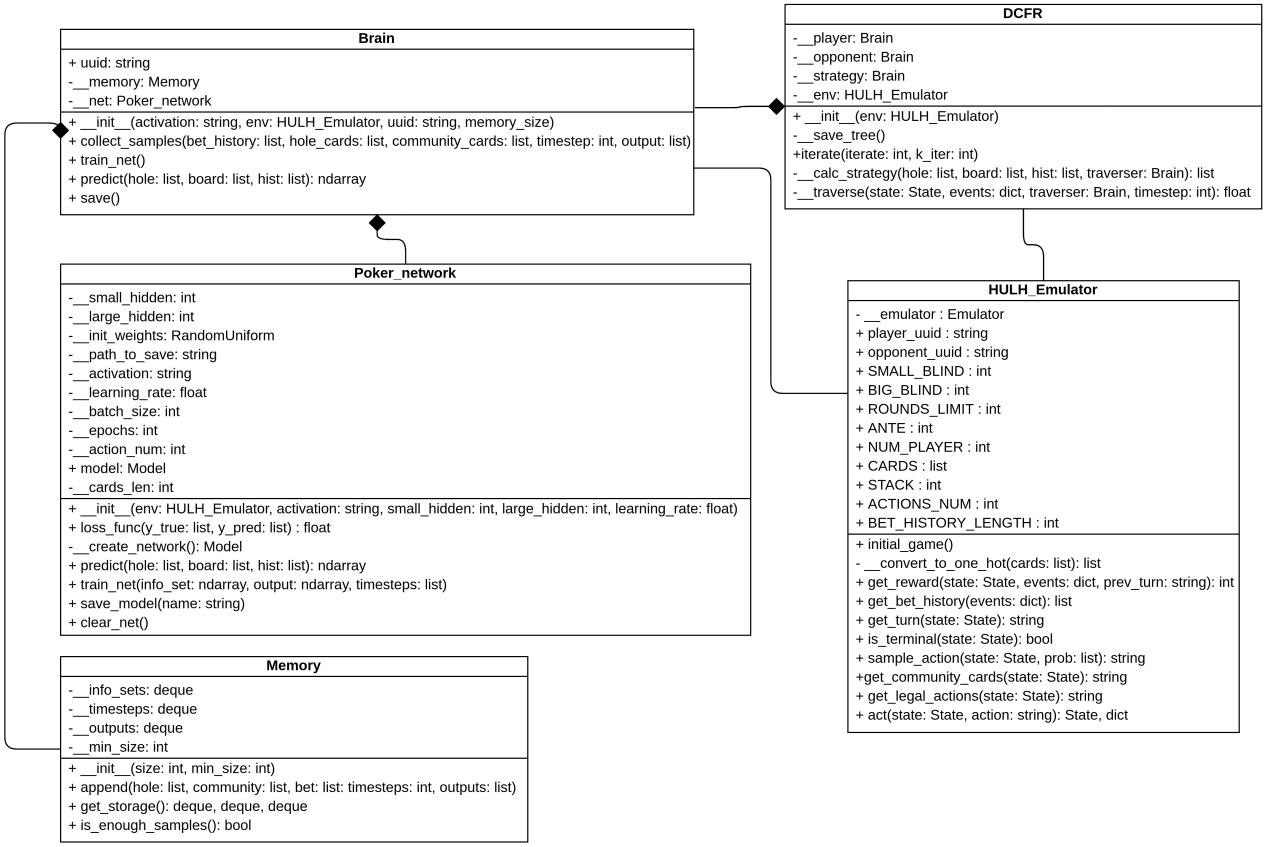


Figure 3.4: Diagram UML projektu.

# Chapter 4

# Wyniki

Rozdział przedstawia rezultaty powstałe po około czterech dniach nauki modelu. Celem rozdziału jest przedstawienie wyników uczenia oraz rezultatów wszystkich rozgrywek miedzy modelami. Dodatkowo, aby przedstawić jakość algorytmu, najlepsze modele zagraja 200 razy z prostym programem zawartym w bibliotece PyPokerEngine, który symuluje gracza nigdy nieblefujacego i niewykonujacego akcji *raise*. Zostana tutaj zaprezentowane różnice miedzy utworzonymi AI wraz z ich przewidywana jakościa korzystajac z odpowiednich wykresów bazujacych na danych pobranych z modułu TensorBoard. Dodatkowo zostanie dokonana próba wyboru najlepszego modelu na podstawie wszystkich etapów oraz klasyfikacji jego stylu gry.

## 4.1 Proces uczenia modelów rozpoznawania

Algorytm Deep CFR zdołał utworzyć pieć modeli rozpoznawania, gdzie każdy proces nauki był śledzony przez funkcje biblioteki *TensorBoard*. Uzyskane dane nastepnie zostały użyte do utworzenia czytelnych wykresów ilustrujacych zależność błedu predykcji od liczby iteracji.

Pierwsze utworzone AI powstał po 10 iteracjach metody Deep CFR, co zajeło około 5 godzin. Nastepnie rozpoczeła sie nauka sieci $\theta_s$ ze zbiorem danych $B_s$ zapełnionym przez około 300 000 próbek. Można zauważyć na rys. 4.1, że wartość błedu po 40 krokach uczenia nie zmieniła sie mocno dla obu zbiorów. W przypadku danych walidacyjnych sieć neuronowa przy dokonywaniu predykcji powodowała duże oscylacje wartości, pozostajac na poziomie około 2,2, co przyczyniło sie do zakończenia procesu przedwcześnie przez *EarlyStopping*. Przy takim okresie model zmniejszył błąd predykcji zbioru uczacego wynoszacy poczatkowo 0,2 do wartości bliskiej 0.

Analizujac taki wykres, można stwierdzić, że model nauczył sie z zebranych danych zależności miedzy wprowadzanymi obserwacjami a zwracanymi akcjami. Dodatkowo rys. 4.6 pokazuje, że takie AI gra lepiej wzgledem wielu utworzonych później modeli.

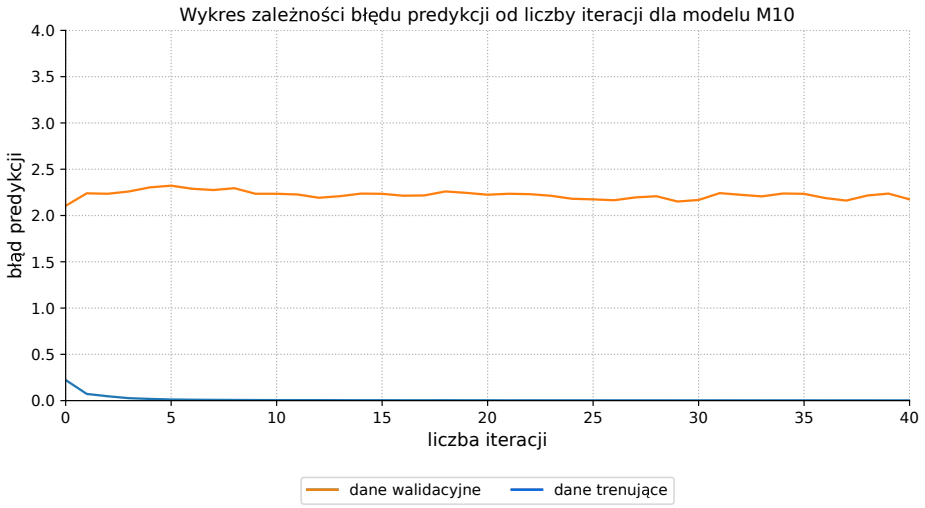


Figure 4.1: Wyniki uczenia modelu po 10 iteracjach Deep CFR.

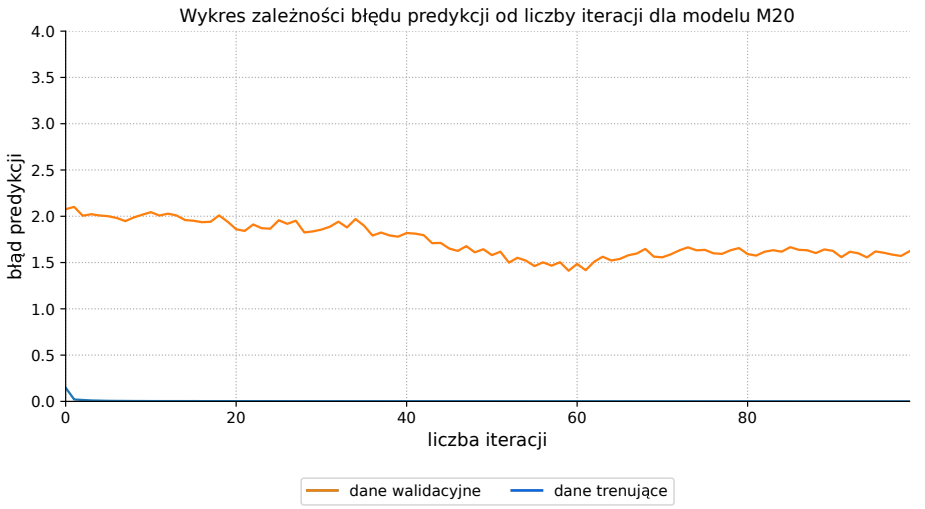


Figure 4.2: Wyniki uczenia modelu po 20 iteracjach Deep CFR.

Po 20 powtórzeniach algorytmu Deep CFR utworzono drugi model rozpoznawania. Nastapiło to po około 12 godzinach. Algorytm do nauki użył prawie całego zapełnionego buforu $B_s$. W zbiorze znajdowały sie nowe próbki oraz te, które zebrano przed nauka pierwszego AI. Proces uczenia przebiegał poczatkowo w podobny sposób jak w poprzednim etapie. Model zaczał od błedu predykcji 2,2 dla zbioru walidujacego i błedu 0,2 dla buforu uczacego. Pierwsza różnica wzgledem poprzedniego etapu jest lepsza nauka na podstawie buforu walidacyjnego, po 10 powtórzeniach wartości zaczeły mocno maleć aż do błedu predykcji wynoszacej około 1,4. Nastepnie pojawił sie krótki wzrost wartości i utrzymywanie sie na tym samym poziomie z lekkimi oscylacjami co spowodowało zakończenie procesu. Wykres 4.2 dobrze pokazuje cel używania biblioteki *EarlyStopping*. Prawdopodobnie przy wiekszej liczbie iteracji model zaczałby uzyskiwać coraz gorsze wartości predykcji zbioru walidacyjnego. Podsumowujac, model uzyskał lepsze wyniki wzgledem etapu pierwszego, co może wynikać z wiekszego zbioru danych. Rys. 4.6 pokazuje, że utworzony model

wygrywa nieznacznie z AI M10 przy 200 powtórzonych grach.

Kolejne AI zostało wytrenowane po 30 iteracjach. Był to drugi dzień działania algorytmu Deep CFR. W tym momencie bufor $B_s$ był całkowicie zapełniony, przez co nowe próbki dodawane do zbioru usuwały najstarsze wpisy. Na tak dużej bazie model uczył sie przez około 100 powtórzeń. Nauka modelu na podstawie zbioru uczacego była podobna jak w poprzednich etapach. Główna różnica jest bufor walidacyjny, dla którego błąd poczatkowo wyniósł 3,5 i zmalał do 1,7.

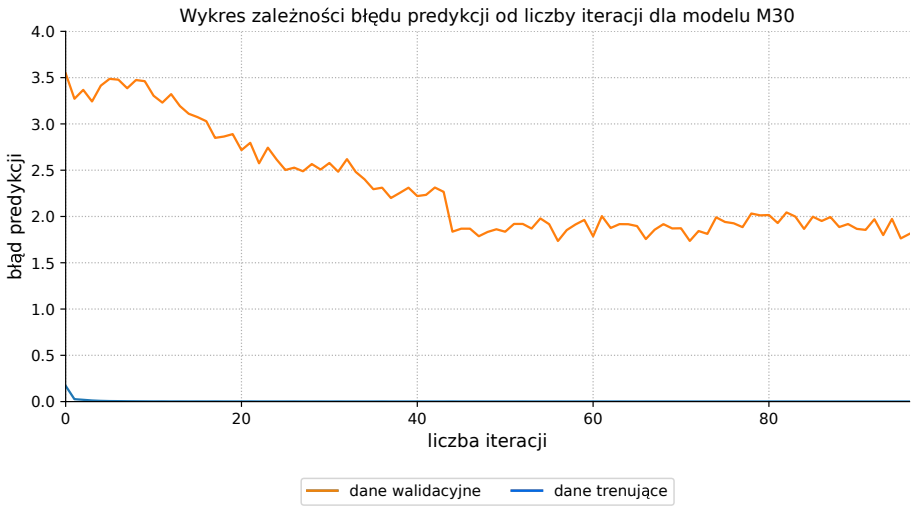


Figure 4.3: Wyniki uczenia modelu po 30 iteracjach Deep CFR.

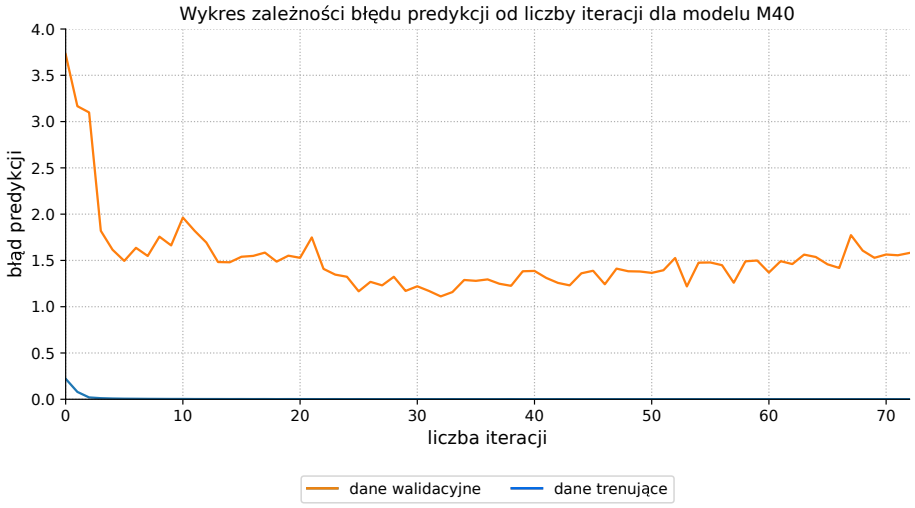


Figure 4.4: Wyniki uczenia modelu po 40 iteracjach Deep CFR.

Model nr 4 powstał po 40 iteracjach. Wytrenował sieć neuronowa po około 75 potwórzeniach gdzie wartości błedu predykcji prezentuja sie w podobny sposób jak w poprzednim etapie. Jedyna róznica jest gwałtowny spadek błedu predykcji w pierwszych 5 iteracjach.

Ostanie AI utworzono po 4 dniach. Rys. 4.5 pokazuje, że proces nauki przebiegał początkowo z niewielkim błedem predykcji i powoli sie zmniejszał do wartości 1,9 w przypadku zbioru walidacyjnego. Drugi zbiór doprowadził do podobnych rezoltatów jak we wszystkich etapach.

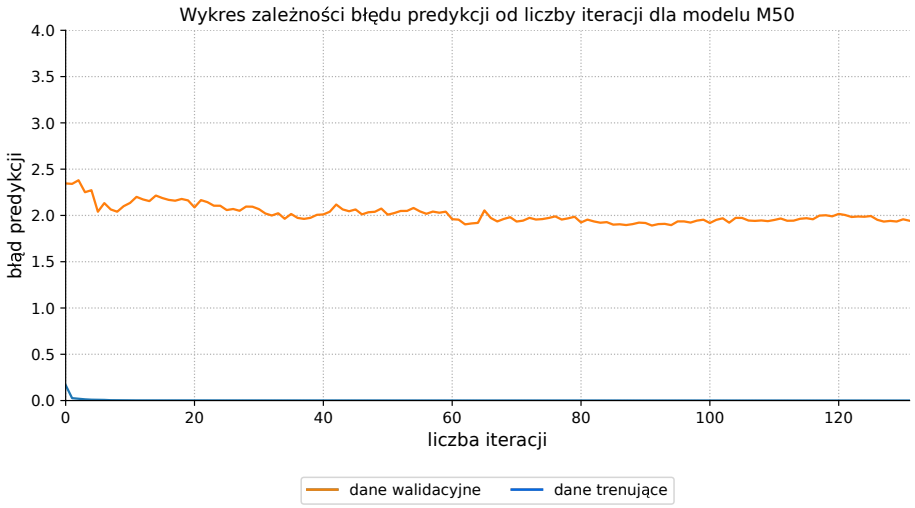


Figure 4.5: Wyniki uczenia modelu po 50 iteracjach Deep CFR.

Podsumowujac powyżesze wyniki można stwierdzić, że zbierane dane w buforze $B_s$ przez Deep CFR sa łatwe w znajdowaniu zależności. Główna przyczyna moga być dane wejściowe i wyjściowe o niewielkim rozmiarze wraz z prostym środowiskiem. Prawdopodobnie przy grach dłuższych niż jedna runda, gdzie gracz przegrywa dopiero jak straci wszystkie żetony, oraz przy wiekszym zbiorze informacji wejściowych proces uczenia by nie przebiegał tak szybko. W takim przypadku model by musiał uwzglednić dodatkowe czynniki jak numer rundy, liczba pozostałej sumy żetonów gracza lub przewaga przeciwnika.

Table 4.1: Lista utworzonych modeli.

| nazwa modelu | liczba iteracji Deep CFR | liczba aktualizacji wag $\theta_s$ |
|---|---|---|
| M10 | 10 | 40 |
| M20 | 20 | 98 |
| M30 | 30 | 92 |
| M40 | 40 | 73 |
| M50 | 50 | 131 |

## 4.2 Wyniki rozgrywek modeli

W celu przetestowania jakości utworzonych modeli przeprowadzono 10 rozgrywek gdzie każda z nich to pojedyncza kombinacja dwóch AI. Każda gre powtórzono 200 razy z wieloma rundami tak, aby zminimalizować czynnik losowości i zakończyć gre w momencie wyczerpania sie żetonów po jednej ze stron. Nastepnie sporzadzono cztery wykresy ilustrujace wyniki tych rozgrywek. Rys. 4.6 przedstawia każda kombinacje gier z przypisana liczba wygranych każdemu z modeli. Rys. 4.7 i 4.8 maja za zadanie pokazać średnia wygrywanych i przegrywanych pul przez graczy, a rys. 4.9 pokazuje rozkład wykonanych akcji. Takie wykresy pozwola stwierdzić, który z modeli cześciej wygrywa, ale też cześciej ryzykuje, przegrywajac wiecej, a który gra ostrożniej.

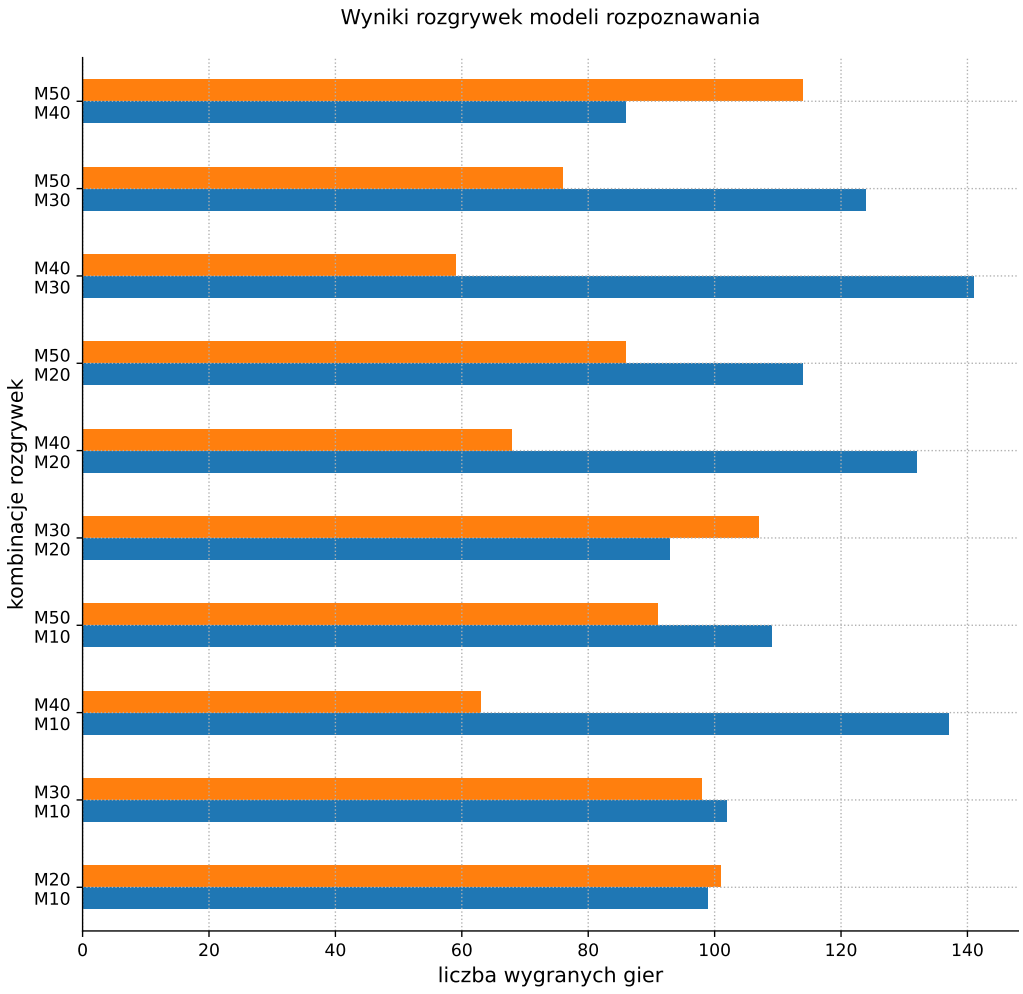


Figure 4.6: Kombinacje rozgrywek miedzy utworzonymi modelami.

Analizujac rys. 4.6 można zauważyć, że model 1 wygrywał z wiekszościa utworzonych AI najczęściej. Najwieksza przewage zdobył wzgledem modelu M40, około 140 wygranych wzgledem 60 porażek oraz około 110 zwyciestw z M50. Rozgrywki z reszta uczestników, cechuje sie niewielkimi przewagami gracza. Wyjatkiem jest gracz M20, którego wybrane strategie okazały sie skuteczne przeciwko oponentowi M10. Przy wielokrotnym powtarzaniu eksperymentu, modele M30 i M20 zdobywały lepsze lub gorsze wyniki niż M10 przez wystepujacy w grze czynnik losowości. Podsumowujac, gracz pierwszy nieznacznie cześciej wygrywał w porównaniu M30 i M20 przez co cieżko stwierdzić czy jest od nich lepszy.

Kolejny obiekt zdobywajacy najlepsze wyniki to M30. Wygrał on z modelem M20 z duża przewaga. Oznacza to, że wybierane strategie przez niego sa skuteczne w rozgrywce z takim typem przeciwnika. Dodatkowo osiagnał on tak samo duża różnice miedzy wygranymi a przegranymi z modelem M40. Na podstawie takich informacji można stwierdzić, że AI powstałe po 40 iteracjach przegrywa najczęściej, a utworzone w 1 i 3 etapie najrzadziej. W dalszej cześci rozdziału zostanie zbadana możliwa przyczyna takich wyników.

Problemem wykresu rys. 4.6 jest pokazywanie tylko czestotliwości zwyciestw modeli, ale nie zawiera informacji o sposobie gry każdego z graczy. Taka wiedza pozwoliłaby na stwierdzenie możliwych przyczyn powstałych wyników. W tym celu zebrano wszystkie wygrane wartości przez graczy z każdej rundy, a nastepnie obliczono ich średnie. Rys. 4.7 przedstawia uśrednione wyniki przegrywanych puli przez każdego z graczy, rys. 4.8 prezentuje odwrotna ceche. Analizujac oba rysunki, można zauważyć, że model nr 1, który posiada najwiecej wygranych, zyskuje i traci najwieksza pule w rundach. Rys. 4.9 pokazuje też, że sztuczna inteligencja wykonuje równie czesto akcje *call* i *raise*, ale bardzo rzadko akcje *fold*. Jest drugim najbardziej zrównoważonym graczem, który uzyskuje przy tym najbardziej skrajne wyniki. Zaleta modelu jest to, że średnie wartości sa na podobnym poziomie, co oznacza, że wygrywa i traci tyle samo. Bardzo podobnej strategii używa gracz M50, ale pomimo tego przegrywa z omawianym modelem. Gracz M30 okazał sie graczem zyskujacym i tracacym mniejsze wartości. Może to wynikać z czestego wykonywania akcji *call*. Modele M20, który znacznie cześciej wykonuje ten ruch, zdobywa najniższa pule wygrana i przegrana. Najgorsze wyniki ma gracz M40, który też najcześciej pasował. Uzyskał on znacznie wieksza średnia wartość wygranej wzgledem przegranej, ale najrzadziej wygrywał.

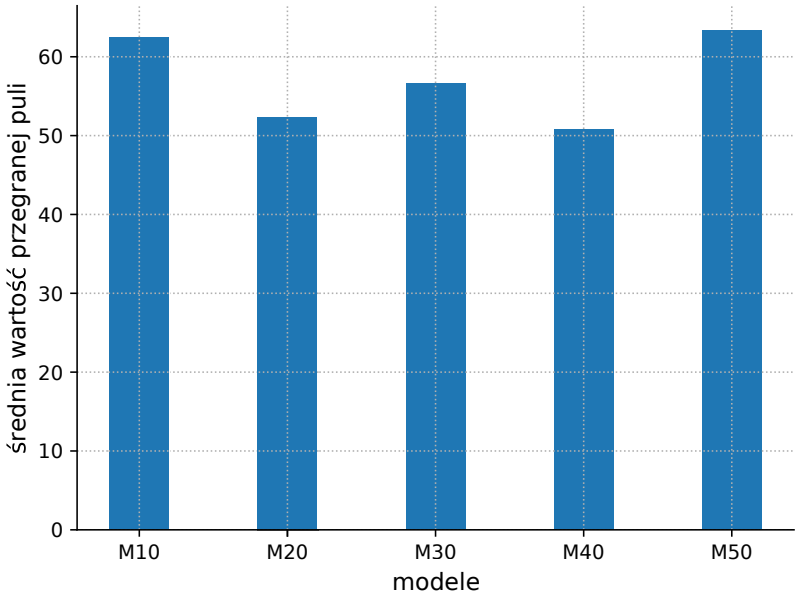


Figure 4.7: Uśrednione wartości traconych żetonów przez modele.

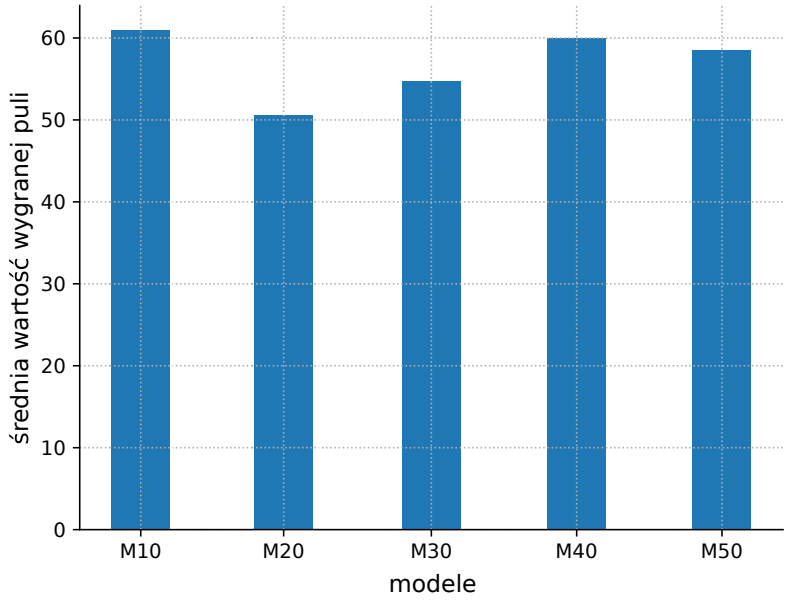


Figure 4.8: Uśrednione wartości wygrywanych żetonów przez modele.

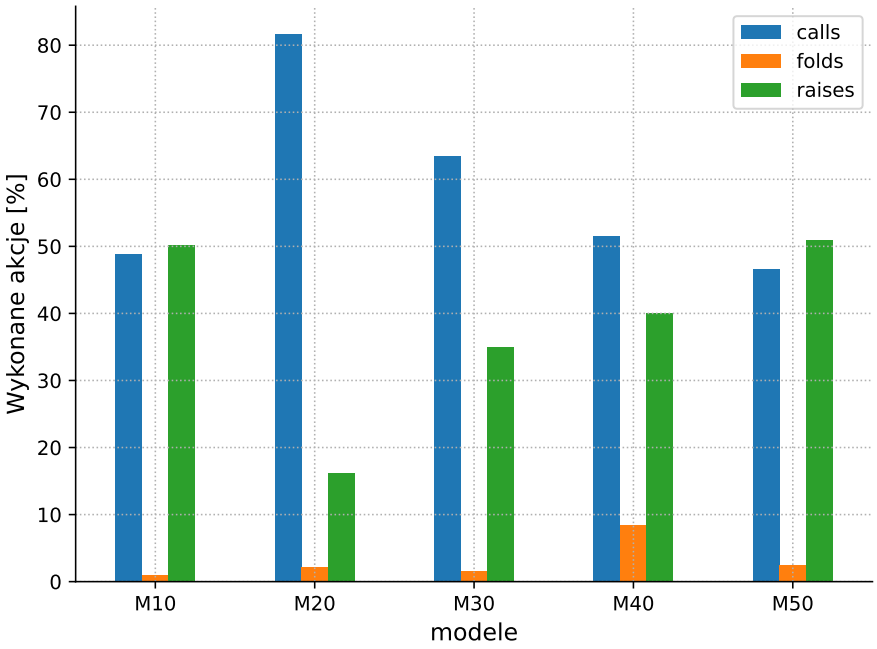


Figure 4.9: Rozkład wybieranych akcji przez modele podczas gry.

Table 4.2: Dokładne wyniki uśrednionych wartości zdobywanych lub traconych żetonów przez modele.

| model | średnia wygrana | średnia przegrana |
|---|---|---|
| 1 | 61 | 65 |
| 2 | 49 | 52 |
| 3 | 53 | 58 |
| 4 | 61 | 50 |
| 5 | 60 | 61 |

## 4.3 Porównanie modeli z programem HP

Końcowym etapem oceny jakości algorytmu Deep CFR jest wykonanie gry miedzy najlepszymi utworzonymi modelami a graczem HP (*HonestPlayer*). Jest to program symulujacy gre w HULH, wykonujac tylko akcje opierajace sie na sile dostepnych kart. W tym celu wykorzystano funkcje *estimate_hole_card_win_rate* [20]. Wykonuje ona określona liczbe iteracji mozliwych wersji gier, a nastepnie liczy szanse wygrania z dostepnymi kartami. W zalezności od zwracanej wartości nastepnie wykonuje akcje *fold* lub *call*.

Rozegrano 2 gry po 200 powtórzeń, gdzie każda z nich składała sie z maksymalnie 10 rund. Na rys. 4.10, 4.11, 4.12, 4.13 zaprezentowano wyniki uzyskanych rozgrywek. Pierwszy z wykresów przedstawia czestotliwośc wygrywania modeli z graczem HP. Wyniki pokazuja ogromna przewage utworzonych modeli z prostym programem. Pomimo tego, analizujac nastepne wykresy można zauważyć, że gracz HP wygrywa wieksza średnia

pule wzgledem modeli i przegrywa mniejsza stawke. Prawdopodobnie wynika to z faktu wykonywania bardzo czesto ackji *fold* oraz *call*. Jak widać taka strategia nie jest dobrym rozwiazaniem przez czeste tracenie szansy na wygrana.

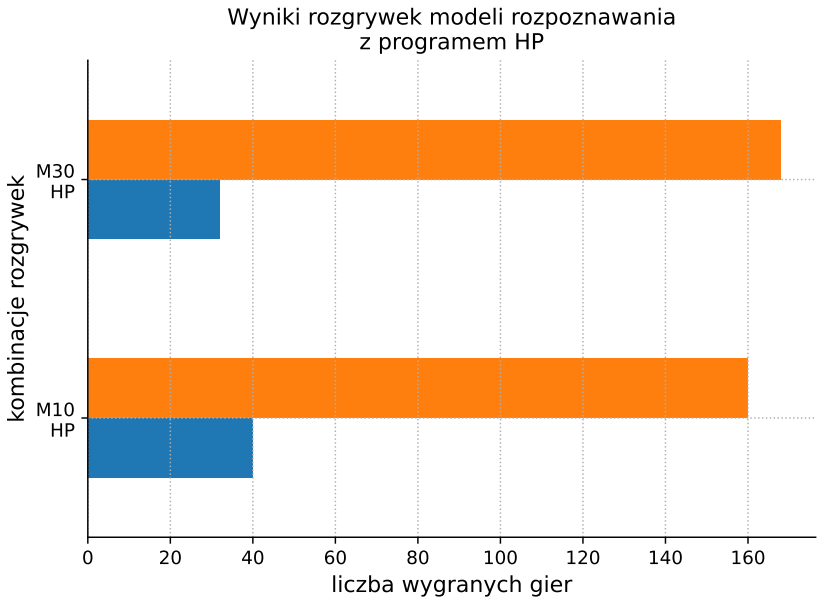


Figure 4.10: Wyniki rozgrywek miedzy modelami M10, M30 i graczem HP.

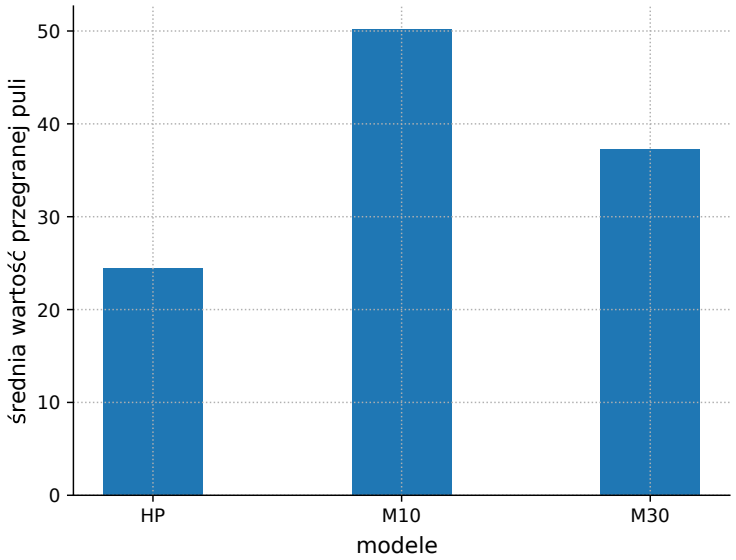


Figure 4.11: Wyniki przegrywanych pul miedzy modelami M10, M30 i HP.

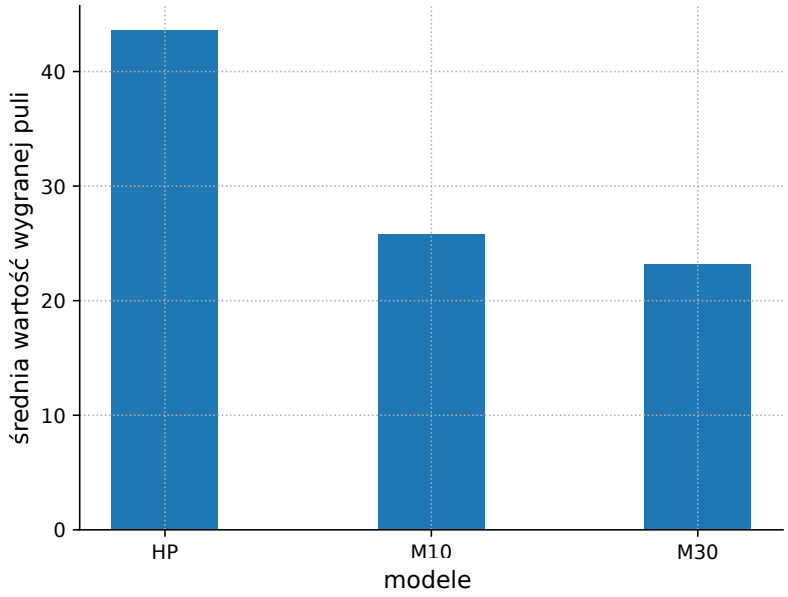


Figure 4.12: Wyniki wygrywanej pul miedzy modelami M10, M30 i graczem HP.

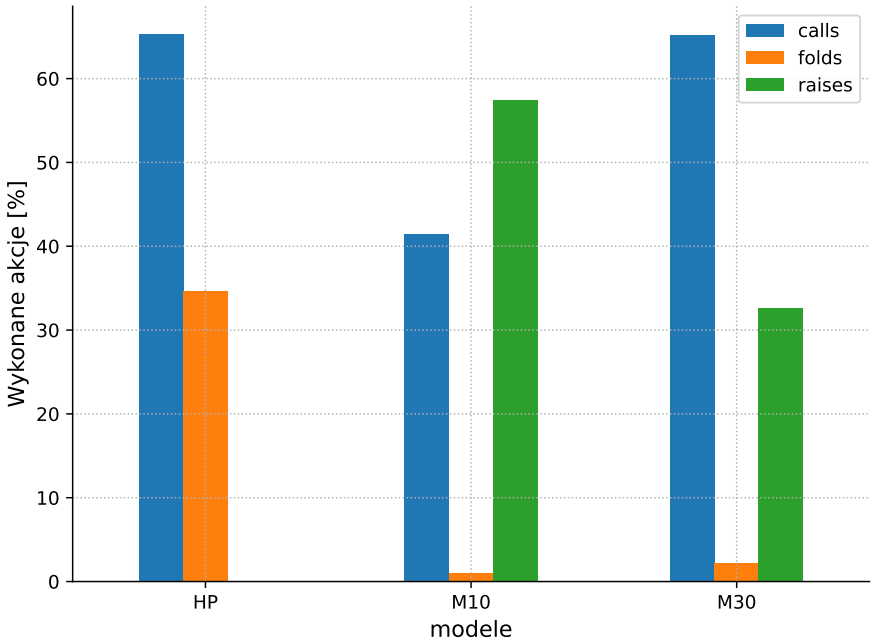


Figure 4.13: Rozkład wykonywanych akcji przez modele w trakcie rozgrywek.

## 4.4 Podsumowanie

Rozdział przedstawił wyniki utworzonych AI. Rezultaty okazały sie mało optymistyczne. Model wytrenowany w ostatniej iteracji nie uzyskuje najlepszych wyników, a przedostatnie AI przegrywa najczęściej. Pomimo tego utworzony zbiór AI gra lepiej od prostych programów jak zaimplementowany gracz HP. Dodatkowo zauważono, że strategia polegajaca na wykonywaniu tylko akcji *call* i *fold* nie daje dobrych rezultatów.

Model M10, który wygrywał najczęściej, posiada zrównoważony rozkład wykonywanych akcji *call*, *raise* oraz bardzo rzadko pasuje gre. Dodatkowo z taka strategia uzyskał najbardziej skrajne wyniki uśrednionych pul wygrywanych i przegrywanych spośród utworzonych modeli. Korzystajac z informacji zawartych w rozdziale 2, można sklasyfikować graczy na bazie ich rozkładów akcji. Wszystkie AI oprócz modelu M40 wykonywały bardzo rzadko akcje *fold*, co oznacza, że wchodziły prawie zawsze do gry. Dodatkowo modele M20 i M30 wykonywały czesto akcje *call*, a M10 i M50 cześciej *raise*. Średnia pula tracona przez te obiekty była powyżej 45 żetonów. Oznacza to, że można je prawdopodobnie sklasyfikować do grupy pomiedzy Loose Passive i Loose Aggressive.

# Chapter 5

# Wnioski i podsumowanie etapów pracy

Praca przedstawiła problem uczenia przez wzmacnianie w środowiskach cześciowo obserwowalnych. Rozdziały 1 i 2 omówiły źródło takich problemów oraz poziom skomplikowania gry Poker Texas Hold'em. Pokazały, że należy korzystać ze skomplikowanych algorytmów, które potrafia powiazać obserwacje z najlepszymi akcjami w danym stanie. W przypadku gier karcianych w tym celu korzysta sie z algorytmów bazujacych na metodzie CFR.

Zaimplementowany algorytm w niniejszej pracy, Deep CFR usprawnił proces znany w metodzie CFR, przez użycie sieci neuronowych. Taka konstrukcja przyspiesza zbieganie sie AI w dużych grach jak HULH do Równowagi Nasha [3]. Program użyty w pracy nie pozwolił na określenie stopnia bliskości do takiego stanu. Wynika to z faktu, że algorytmy używajace metody CFR korzystaja z metryki *Exploaibility* bazujacej na wartości BR (*Best Response*) do śledzenia postepów wybieranych strategii przez AI [16]. Taki element głównie sie implementuje w grach abstrakcyjnych z powodu bardzo dużej obciażalności obliczeniowej. Duże gry stosuja mniej dokładna metryki jak LBR(*Local Best Response*), która z dużym przybliżeniem zwraca stopień bliskości modelu do Równowagi Nasha [16]. Charakteryzuja sie one dużym skomplikowaniem implementacyjnym.

Z tego powodu do oceny jakości Deep CFR użyto prostszej metody. Rozegrano wiele gier HULH miedzy utworzonymi modelami, a nastepnie przeanalizowano wyniki i wybrano najlepszego z nich na bazie uzyskanych cech. Rezultaty okazały sie mało optymistyczne w porównaniu do czasu poświeconego na czas uczenia modeli. Aktualny rozdział przedstawia możliwe przyczyny takich efektów oraz wnioski po dokonaniu wszystkich poprzednich etapów pracy wraz z mozliwymi poprawami parametów. Końcowa cześć pracy przedstawi dalsza historia algorytmu Deep CFR. Dodatkowo zostanie przedstawiony możliwy kierunek rozwoju uczenia maszynowego w środowiskach cześciowo-obserwowalnych.

## 5.1 Wnioski

Gra Limit Poker Texas Hold'em jest bardzo trudnym środowiskiem do uczenia sztucznych inteligencji, wynika to z cześciowej obserwowalności. Z tego powodu model Cepheus, zdolny pokonywać ludzi w takim środowisku powstał dopiero w 2015 roku. Od tamtej pory rozpoczał sie nagły rozwój metod uczenia maszynowego coraz wiekszych gier karcianych. Przykładami sa DeepStack, Libratus lub Pluribus rozwiazujace gre Heads Up No Limit Texas Poker Hold'em i pokonujace profesjonalnych ludzi.

Głównym zadaniem niniejszej pracy była próba zmierzenia sie z takim środowiskiem. W celu uproszczenia zadania wybrano gre HULH. Nastepnie zaimplementowano nowoczesny algorytm Deep CFR i wytrenowano pieć modeli rozpoznawania. Pierwszy problem, jaki napotkano to, powolna nauka sieci $\theta_p$, wraz z dużym błedem predykcji. Podczas uczenia wartości nie spadały poniżej 600. Możliwym rozwiazaniem jest zwiekszenie parametru *learning rate* oraz dopracowanie architektury sieci neuronowej co przyczyni sie do niewpadania wartości w minima lokalne. Dodatkowa przyczyna takich rezultatów moga być zbiory danych o słabej jakości. Prawdopodobnie nauka by przebiegała lepiej przez zwiekszenie zawartości obserwowalnych informacji wejściowych np. liczba żetonów w grze. W taki sposób sieć neuronowa by miała wiecej informacji, które by zostały użyte do zwracania właściwego wektora $D_p^t$.

Kolejnym problemem, który może mieć duże znaczenie w grze to konstrukcja wprowadzanych informacji do modelu. Dane wejściowe użyte w implementacji to karty widziane od strony gracza oraz historia z jednej rundy. Wada takiej architektury jest to, że w praktyce rozgrywki Poker Texas Hold'em moga odbywać sie dłużej. Wtedy gracz musi dodatkowo używać takich informacji jak, wybrana strategia przeciwnika w poprzednim etapie, czy grał ostrożnie, agresywnie albo blefował. Kolejnym czynnikiem jest sposób zmieniania sie gry zależnie od liczby pozostałych żetonów w puli oraz od numeru rundy. Gracze moga podejmować bardziej ryzykowne i nierozważne ruchy, bedac w stanie bliskim porażki. Takie informacje mogłyby być kluczowe w osiagnieciu zwyciestwa. Prawdopodobnie przy uwzglednieniu tych elementów w sieci neuronowej, utworzone modele lepiej by dobierały strategie do określonych stanów gry. To by wymagało wykonywania eksploracji MCCFR ES na znacznie wiekszych drzewach decyzyjnych obejmujacych wiele rund. Dodatkowo dane wejściowe sieci neuronowej byłyby wieksze. W takim przypadku możliwym zbiorem informacji mógłby być zestaw składajacy sie z widocznych kart, historii z wielu rund, liczby żetonów każdego z graczy oraz numeru gry. W taki sposób model nauczyłby sie lepiej dostosowywać sposób gry do obserwacji.

Deep CFR spełnił funkcje i utworzył modele, które wygrywaja z prostymi programami symulujacymi gre Poker Texas Hold'em jak przedstawiony gracz nieblefujacy, HP w rozdziale 4. Pomimo dobrych rezultatów dużym problemem okazał sie proces powstawiania sztucznych inteligencji. Pozornie można by było oczekiwać, że algorytm bedzie tworzył lepsze AI wraz z dłuższym czasem działania. W pracy doszło do odwrotnej sytuacji. Sztuczne inteligencje utworzone w iteracjach 10 i 30 wygrywały z późniejszymi obiektami. Cieżko określić przyczyne takich rezultatów. Pomocne okazałoby sie użycie metryki *Exploiability* do śledzenia postepów AI pomimo zwiekszenia wymaganej mocy obliczeniowej przez algorytm. Taki element pozwoliłby na stwierdzenie czy algorytm ominał punkt o najlepszej jakości i w dalszym procesie uzyskuje podobne lub coraz gorze efekty. Pozwoliło by to na zatrzymanie procesu uzyskujac najlepszy możliwy model z implementacji.

## 5.2 Dalszy rozwój algorytmów bazujacych na metodzie CFR

Algorytm powstały w 2017 roku dawał dobre rezultaty, ale przez wykorzystanie dwóch sieci neuronowych tworzył duża wariancje wyników [17]. Z tego powodu powstał jego nastepca Single Deep CFR. Po wykonanych testach uzyskał nie znacznie lepsze wyniki. Algorytm dalej nie był perfekcyjny, z tego powodu w 2020 roku powstała metoda uczenia maszynowego o nazwie DREAM [18]. Jest to bardzo dobry sposoby na tworzenie AI w środowiskach cześciowo-obserwowalnych. Pomimo tego wada takich algorytmów jest to, że wymagaja od gry, aby wartość wygranej i przegranej sumowała sie do zera. Taka cecha określa sie środowiska zero-sum [3]. Przez to algorytmy sa mało adaptacyjne do innych środowisk. Kolejnym problemem tych metod jest przeprowadzenie testów schodzenia sie do Równowagi Nasha tylko w grach 2-osobowych [3]. Wiele środowisk jak Poker Texas Hold'em standardowo uwzglednia wieksza ilość uczestników. Ten problem udało sie rozwiazać przez model Pluribus dopiero w 2019 roku. Na podstawie takich informacji można stwierdzić, że zaimplementowany Deep CFR wraz z jego nastepcami nie wyczerpały tematu środowisk gier karcianych. Nawet nowsze AI, które pokonuja profesjonalnych graczy jak DeepStack lub Libratus musza trzymać sie tych warunku [12] [13]. Oznacza to, że takie gry to bardzo trudne środowiska, które prawdopodobnie beda jeszcze długo badane pod wzgledem możliwych rozwiazań.

# Bibliography


[1] Haenlein, Michael, and Andreas Kaplan. "A brief history of artificialintelligence: On the past, present, and future of artificial intelligence." California management review 61.4 (2019): 5-14.

[2] Berner, Christopher, et al. "Dota 2 with large scale deep reinforcement learning." arXiv preprint arXiv:1912.06680 (2019).

[3] Brown, Noam, et al. "Deep counterfactual regret minimization." International conference on machine learning. PMLR, 2019.

[4] Heinrich, Johannes, Marc Lanctot, and David Silver. "Fictitious self-play in extensive-form games." International conference on machine learning. PMLR, 2015.

[5] Zinkevich, Martin, et al. "Regret minimization in games with incomplete information." Advances in neural information processing systems 20 (2007): 1729-1736.

[6] Heinrich, Johannes, and David Silver. "Deep reinforcement learning from self-play in imperfect-information games." arXiv preprint arXiv:1603.01121 (2016).

[7] Teófilo, Luís Filipe Guimarães. "Building a poker playing agent based on game logs using supervised learning." (2010).

[8] Félix, Dinis Alexandre Marialva. "Artificial intelligence techniques in games with incomplete information: opponent modelling in Texas Hold'em." (2008).

[9] Xiang, Xuanchen, and Simon Foo. "Recent Advances in Deep Reinforcement Learning Applications for Solving Partially Observable Markov Decision Processes (POMDP) Problems: Part 1—Fundamentals and Applications in Games, Robotics and Natural Language Processing." Machine Learning and Knowledge Extraction 3.3 (2021): 554-581.

[10] Nogal-Meger, P. (2012). Dylemat wieźnia jako przykład wykorzystania teorii gier. Prace i Materiały Wydziału Zarzadzania Uniwersytetu Gdańskiego, 10(4, cz. 2), 87–95.

[11] Bowling, Michael, et al. "Heads-up limit hold'em poker is solved." Communications of the ACM 60.11 (2017): 81-88.

[12] Brown, Noam, and Tuomas Sandholm. "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals." Science 359.6374 (2018): 418-424.

[13] Moravčík, Matej, et al. "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker." Science 356.6337 (2017): 508-513.

[14] Lanctot, Marc, et al. "Monte Carlo sampling for regret minimization in extensive games." Advances in neural information processing systems 22 (2009): 1078-1086.

[15] Prechelt, Lutz. "Early stopping-but when?." Neural Networks: Tricks of the trade. Springer, Berlin, Heidelberg, 1998. 55-69.

[16] Lisy, Viliam, and Michael Bowling. "Eqilibrium approximation quality of current no-limit poker bots." Workshops at the Thirty-First AAAI Conference on Artificial Intelligence. 2017.

[17] Steinberger, Eric. "Single deep counterfactual regret minimization." arXiv preprint arXiv:1901.07621 (2019).

[18] Steinberger, Eric, Adam Lerer, and Noam Brown. "DREAM: Deep regret minimization with advantage baselines and model-free learning." arXiv preprint arXiv:2006.10410 (2020).

[19] Brown, Noam, and Tuomas Sandholm. "Superhuman AI for multiplayer poker." Science 365.6456 (2019): 885-890.

[20] https://github.com/ishikota/PyPokerEngine GitHub.GitHub repository.

[21] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.