Politechnika Wrocławska
Wydział Elektroniki

KIERUNEK: Teleinformatyka
SPECIALNOŚĆ: Projektowanie Sieci Teleinformatycznych

PRACA DYPLOMOWA INŻYNIERKSA

tu jest tytuł

tu tytuł w jezyku angielskim

AUTOR:
Michał Żarejko

PROWADZĄCY PRACĘ:
dr. mgr. Paweł Zyblewski

Wrocław 2021

Spis treści

Rozdział 1

Wstęp

Uczenie maszynowe jest zagadnieniem rozwijanym od dłuższego czasu. Już w latach 40 powstawały książki oraz programy powiązane ze sztuczną inteligencją [1]. Między innymi w 1940 roku Donald Hebb stworzył podstawy teoretyczne wykorzystane w późniejszych sieciach neuronowych [1]. Po mimo, wielu wczesnych pomysłów omawiany dział nauki, dopiero od niedawna zaczął osiągać wielkie sukcesy. Wynika to z faktu, że wiele algorytmów potrzebuje dużej mocy obliczeniowej i dopiero nowoczesny sprzęt był w stanie spełnić takie wymagania [1].

Dzisiaj można wymienić wiele rozwijanych narzędzi związanych z tym tematem. Między innymi asystenci głosowi, tłumacze językowe, modele wyświetlające elementy na stronach internetowych, gry wideo lub inteligentne samochody. Dodatkowo sztuczna inteligencja jest mocno wykorzystywana w firmach, na halach produkcyjnych, w transporcie, medycynie albo cyberbezpieczeństwie.

Pomimo tylu możliwości i zastosowań AI zyskuje aktualnie największą popularność medialną przez gry rywalizacyjne, gdzie głównym zadaniem jest pokazanie przewagi algorytmów względem ludzi. Dzisiaj można wymienić wiele takich wydarzeń, gdzie mistrzowie świata w danej grze, przegrywali z modelami rozpoznawania.

Między innymi w 2016 roku zorganizowano mecz między Fan Hui, mistrzem Europy w chińskiej grze Go oraz algorytmem AlphaGo [2]. Model utworzony przez zespół DeepMind osiągnął duży sukces przez wygraną z przeciwnikiem. Do tej pory gra była uważana powszechnie za skomplikowaną i trudną do rozwiązania.

W 2019 roku utworzono model zwany OpenAI Five czyli pierwsze na świecie AI, które pokonało zespół Team OG w rywalizacji e-sport w grze Dota 2 [3]. Wyzwanie polegało na rywalizacji 5-osobowych zespołów. Wydarzenie było mocno omawiane w mediach z powodu pierwszego takiego osiągnięcia w tej dziedzinie sportu. Dodatkowo Dota 2 była bardzo skomplikowanym środowiskiem. Przykładowo gra Go rozwiązana parę lat wcześniej, zawierała 150 możliwych ruchów na turę, Dota 2 mogła posiadać ich 20 000 w niecałą godzinę [3].

Istnieje wiele takich wydarzeń. Pokazują one, że w dzisiejszych czasach sztuczna inteligencja może przewyższać myśleniem strategicznym człowieka. Dodatkowo udowadniają one, że temat AI jest dalej rozwijany i zyskuje coraz większe zainteresowanie.

1.1 Geneza tematu pracy

Często w tworzeniu programów sztucznej inteligencji dużym wyzwaniem jest poziom skomplikowania gry. Zależy to między innymi od typu środowiska np. deterministycznego lub stochastyczne, od poziomu dynamiki gry lub od tego, czy przestrzeń wymiarowa jest dyskretna, lub nieskończona. Aktualnie jednak jednym z większych problemów takich programów jest niedostateczny zakres dostępnych informacji o środowisku [4].

Sztuczna inteligencja, aby zwyciężać, musi nauczyć się grać, więc potrzebuje dużej ilości danych wejściowych, które są rozróżnialne. Przykładem gry, która jest pozbawiona tego problemu, są szachy. AI wykonuje ruchy, bazując na informacjach jak np. ułożenie pionków w danej turze. Widoczna zmiana stanu środowiska przeciwnika jest zauważalna przez gracza, przez co może on łatwiej powiązać obserwacje z wykonywanymi akcjami.

Przykładem gry ciężkiej w uczeniu AI w której, występuje niepełny zestaw informacji, jest Poker Texas Hold'em, pomimo wiedzy o kartach w ręce i na stole, gracz nie posiada wiedzy o kartach przeciwników. W takim przypadku dwa pozornie identyczne stany środowiska w rzeczywistości mogą się różnić. Z powodu takich cech większość popularnych algorytmów jak DQN (*Deep Q Learning*), Actor-Critic lub AlphaZero staje się bezużyteczna i nie daje dobrych rezultatów.

W niniejszej pracy przedstawiono sposób możliwego rozwiązania takiego problemu przy pomocy algorytmu Deep CFR [5]. Jest to popularna metoda do tworzenia modeli rozpoznawania w grach karcianych.

1.2 Cel i zakres pracy

Głównym celem pracy jest implementacji algorytmu Deep CFR, który stworzy 5 modeli rozpoznawania w grze HULH (*Heads Up Limit Texas Poker Hold'em*). Jest to popularna wersja rozgrywki 2-osobowej, gdzie uczestnicy nie mogą wybrać samodzielnie kwoty podbicia stawki. Jest ona ograniczona przez ustaloną wartość. Takie środowisko minimalizuje możliwe ruchy do 3 akcji, co czyni go prostszą bazą do uczenia maszynowego. Uzyskane modele zostaną następnie wykorzystane do stworzenia rozgrywek składających się na wszystkie kombinacje dwóch modeli, gdzie każda gra zostanie powtórzona 200 razy. Drugim etapem będzie obliczenie średniej puli wygrywanej i przegrywanej przez każdy model wraz z rozkładem wykonywanych ruchów. Taki proces pozwoli określić, który model osiąga najlepsze wyniki i jakiej strategii używa do gry.

Praca została podzielona w tym celu na rozdziały opisujące każdy z etapów projektu. Następny rozdział jest wstępem teoretycznym do implementacji programu. Opisuje on możliwe rozwiązania problemu, analizę gry Poker Texas Hold'em oraz wymaganą teorię do zrozumienia Deep CFR. Trzecia część przedstawia implementacje programu wraz z użytymi technologiami. Końcowe rozdziały omawiają wyniki uczenia, rezultaty rozgrywek modeli oraz podsumowanie pracy.

Rozdział 2

Analiza istniejących rozwiązań

W ciągu ostatnich 15 lat powstało wiele algorytmów rozwiązujących różne wersje gry Poker. Między innymi CFR (*Counterfactual Regret Minimization*) [7], XFP (*Extensive-Form Fictitious Play*) [6] lub NFSP (*Neural Fictitious Self-Play*) [8]. Pierwszy z wymienionych, CFR powstał w 2007 roku. Był pomyślną prób rozwiązania abstrakcyjnego środowiska Poker Texas Hold'em [7]. Na jego podstawie utworzono wiele nowoczesnych algorytmów, które dają szanse rozwiązać takie gry jak HULH [7].

Z wymienionych metod zaimplementowanym rozwiązaniem w niniejszej pracy jest CFR rozszerzony o sieci neuronowe, czyli Deep CFR z grą HULH. Pozwala on na szybsze trenowanie modeli w środowisku typu zero-sum, dodatkowo lepiej rozwiązuje gry o dużych rozmiarach [5].

W tym rozdziale zostaną przedstawione profesjonalne sposoby wyboru strategii w grze Poker Texas Hold'em. Określą one cechy, jakimi powinien charakteryzować się prawidłowo utworzony model rozpoznawania. Następnie rozdział przedstawi istniejące sztuczne inteligencje, które wykorzystały metodę CFR do zwyciężania z profesjonalnymi graczami. Ostatnim krokiem jest przedstawienie zbioru informacji wymaganych do zrozumienia implementowanego algorytmu.

2.1 Analiza Texas hold'em Poker

Jest to jedna z najpopularniejszych gier rywalizacyjnych w kasynach, dodatkowo jest to dominująca gra hazardowa. Można ją scharakteryzować brakiem stanów deterministycznych oraz częściową obserwowalnością, tab. 2.1 [9].

Przez takie cechy gra była od zawsze tematem sporów, czy na jej wynik ma większy wpływ losowość, czy umiejętności. Jak wynika z badań dużym aspektem pomagającym w osiągnięciu zwycięstwa, jest panowanie nad emocjami, dokładna analiza stanu gry oraz umiejętność opóźnienia natychmiastowej nagrody [10].

Tabela 2.1: Charakterystyka gier

	środowisko deterministyczne	środowisko niedeterministyczne
pełny zestaw informacji	szachy Go	Monopoly Tetris
niepełny zestaw informacji	Saper Mahjong	Poker Makao

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

Kolejną ważną zasadą jest obserwacja przeciwnika oraz zapamiętywanie jego poprzednich akcji. Przez to można określić, czy gra on agresywnie, czy pasywnie i dobrać do niego odpowiednią strategię. W tym celu dokonuje się klasyfikacji przeciwników na bazie częstotliwości wykonywanych ruchów [11].

Każdego gracza można podzielić na cztery grupy, Loose Aggressive, Losse Passive, Tight Passive oraz Tight Aggressive [11]. Prawidłowe rozpoznanie danego stylu gry może zadecydować o wyborze prawidłowej strategii i zwycięstwie. Poniżej opisano każdy z nich oraz porównano je z rys. 2.2, który pokazuje, w jakim stopniu profesjonalny gracz powinien używać każdego z nich [11].

Loose Passive

Osoba, która bardzo często wchodzi do gry niezależnie czy posiadane karty dają jej wysokie szanse na wygraną. Ten typ gry nie jest dobrą strategią, ponieważ można łatwo się do niego dostosować przez używanie tylko mocnych kart [11].

Loose Aggressive

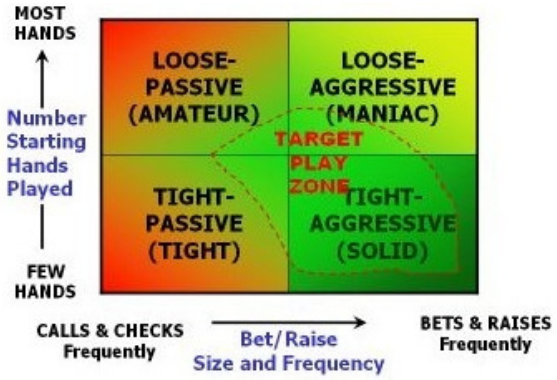
Osobę, która często przebija stawkę, zaczynając od rundy *pre-flop* pod warunkiem, że ma dobre startowe karty [11]. Strategia ma stworzyć przekonanie wśród oponentów, że gracz ma bardzo duże szanse wygranej od samego początku. Okazuje się ona jednak nieefektywna, jeśli przeciwnicy nie pasują w początkowych etapach gry [11].

Tight Passive

Uczestnik gry wchodzi tylko z dobrymi kartami, wykonując często akcję *call*. Ostatecznie pasuje przy spotkaniu z graczem agresywnym. Taka osoba gra bardzo dokładnie tak, aby mało ryzykować, przez co często traci wiele okazji, kiedy mogła, by wygrać [11].

Tight Aggressive

Grają podobnie do typu *Tight Passive* w początkowych etapach gry, a następnie zmieniają swój styl na bardziej agresywny [11]. Rys. 2.1 pokazuje, że jest to najlepsza wersja strategii, jaką można grać, dlatego często jest ona wybierana przez profesjonalnych graczy.



Rysunek 2.1: Podział graczy [11].

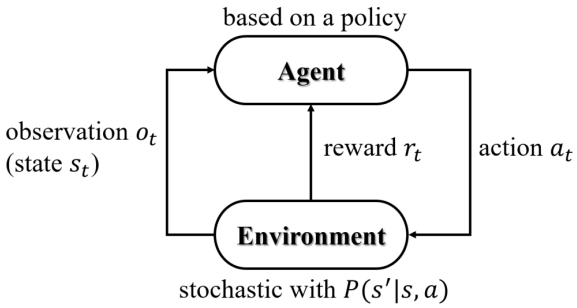
Jak wynika z powyższych kategorii, gra Poker Texas Hold'em zawiera wiele elementów niezwiązanych z losowością, gdzie obserwacje i dobieranie odpowiedniej strategi do typu gracza pełni kluczową funkcję. Korzystając z tych informacji, będzie można określić poziom zaawansowania utworzonych modeli, które są opisane w rozdziale 4.

2.2 Uczenie przez wzmacnianie

Jest wiele sposobów na stworzenie sztucznych inteligencji w grach. Między innymi można użyć technik uczenia nadzorowanego pod warunkiem, jeśli przygotuje się odpowiednie zbiory danych. W pracy jednak zdecydowano się na stworzenie modelu, korzystając z uczenia przez wzmacnianie, czyli rozwiązania gdzie AI nie potrzebuje wstępnej bazy uczącej. Wynika to z faktu, że jest mało publicznych zapisów gry Poker Texas Hold'em z profesjonalnymi graczami, które mogłyby posłużyć jako zbiory uczące. Algorytmy należące do wybranego działu powinny być w stanie polepszać swoje wyniki na podstawie interakcji ze środowiskiem bez korzystania z zewnętrznych materiałów.

Wiele istniejących metod należących do wybranej techniki zakłada, że środowisko można opisać przez model matematyczny MDP (*Markov Decision Process*). Określa ona sekwencyjnie podejmowane decyzje w niepewnym środowisku [12]. W każdym z nowych stanów, w jakich znajduje się agent (uczące się AI), wykonuje on pojedynczą akcję, zyskując od środowiska informacje o nowym stanie oraz nagrodzie. Elementem wyjściowym zasady powinien być zbiór strategii w postaci modelu, rys. 2.2.

MDP może opisywać jedynie środowiska z pełnym zakresem informacji, w przypadku algorytmu będącego tematem pracy, głównym zadaniem jest rozwiązanie środowiska z niepełnym zestawem danych. Wtedy należy rozpatrywać zasadę Partially Observable Markov Process, POMDP [12].



Rysunek 2.2: Schemat interakcji ze środowiskiem [12].

W przeciwieństwie do poprzedniej zasady tutaj agent nie zna aktualnego stanu, w którym się znajduje [12]. Przez takie okoliczności musi połączyć zależnością wykonywane akcie i obserwacje, a nie stany. Większość gier karcianych można zakwalifikować do tego typu problemów [12].

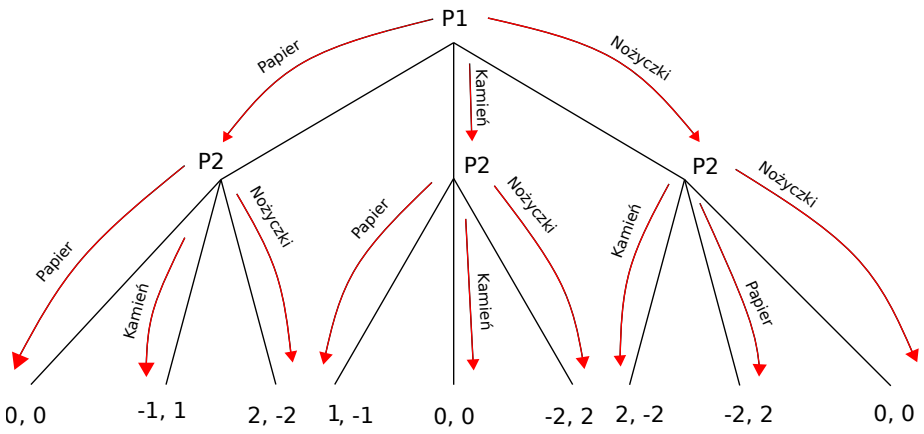
2.3 Teoria Gier

Aby zrozumieć działanie algorytmów CFR, Deep CFR, MCCFR itd. należy zapoznać się z podstawami działu matematyki o nazwie Teoria Gier. Bada on optymalne zachowanie w środowiskach, gdzie występuje konflikt [14]. W przypadku gry Poker Texas Hold'em zostaną wytłumaczone terminy Równowaga Nasha oraz postać ekstensywna. Są one obowiązkowe do zrozumienia algorytmu Deep CFR.

Gra w postaci ekstensywnej

Gry w formie ekstensywnej można przedstawić jako drzewo decyzyjne, gdzie każdy węzeł rozgałęzia się na możliwe akcje oraz identyfikuje aktualny stan gracza przez zestaw informacji. Stany końcowe drzewa określają zysk lub stratę nagrody wybranego gracza [14]. Jest to sposób na uproszczenie opisu gry.

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



Rysunek 2.3: przykład gry w postaci ekstensywnej.

Równowaga Nasha

W grach to twierdzenie określa perfekcyjny stan wyboru akcji, gdzie wszyscy gracze wykorzystują najlepszy zestaw strategii, którego zmiana przyniesie tylko straty. Oznacza to, że nie jest możliwa zmiana ruchów oraz zwiększenie uzyskanej nagrody [14].

Dobrym przykładem prezentującym taki stan jest "Dylemat Więźnia" [13]. To środowisko zawiera dwóch przestępców, którzy są przesłuchiwani w odseparowanych pomieszczeniach. Każdy z nich ma dwie opcje, przyznać się do zarzutów lub tego nie robić. Każda z kombinacji akcji uczestników jest zaprezentowana w tab. 2.2, gdzie wartości określają lata spędzone w więzieniu po danym ruchu. Posługując się Równowagą Nasha, można stwierdzić, że najlepszą opcją dla obu uczestników będzie przyznawanie się za każdym razem [13]. Wynika to z faktu, że wyniki przegranej są tam małe wraz z brakiem ryzykowania porażką, czyli 5 latami więzienia.

Głównym zadaniem większości algorytmów gier karcianych bazujących na metodzie CFR jest znalezienie takiego stanu. Deep CFR nie odnajduje go, ale odkrywa zbiory strategii, które są bliskie Równowadze Nasha [5].

Tabela 2.2: Wyniki akcji środowiska "Dylemat więźnia".

	przyznanie się więźnia A	więzień A kłamie
przyznanie się więźnia B	1 / 1	5 / 0.5
więzień B kłamie	0.5 / 5	0 / 0

2.4 Historia modeli Texas Hold'em Poker

Bazując na Teorii Gier oraz innych twierdzeniach powstało wiele rozwiązań różnych wersji gry Poker. Pierwsze dokumenty naukowe omawiały bardzo proste środowiska jak Poker Kuhn [5]. Dopiero w 2015 roku utworzono znaną sztuczną inteligencję "Cepheus" rozwiązującą problem HULH [15]. Było to pierwsze takie osiągnięcie w historii. Kolejnym etapem były prace nad algorytmem mogącym rozwiązać problem gry HUNH (*Heads Up No-limit Texas Hold'em*). Powstały model w 2017 roku nazwano "DeepStack" [17]. Mieszał on sieci neuronowe z technikami algorytmu CFR. W podobnym czasie utworzono kolejne, tym razem najbardziej zaawansowane AI w historii, Libratus [16].

Pomimo takich osiągnięć utworzone modele potrafią grać tylko w środowiskach składających się maksymalnie z 2 osób typu zero-sum [16] [15] [17]. Wynika to z poziomu skomplikowania gier częściowo-obserwowalnych. Sposoby na jego rozwiązanie zaczęły powstawać od niedawna, a pierwsze dwa duże osiągnięcia w grze HUNH miały miejsce dopiero w 2017 roku. Dodatkowo wszystkie wymienione modele bazują na metodzie CFR lub na jej nowszych wersjach.

Cepheus

AI powstałe w celu wygrywania w grach HULH. Był to pierwszy model rozwiązujący dużą wersję gry Texas Poker Hold'em w historii [15]. Wykorzystał on nowszą wersję techniki CFR, którą nazwano CFR+ [15]. W wyniku dwóch miesięcy nauki i testów nowa metoda zbiegała się znacznie szybciej do Równowagi Nasha niż bazowy CFR [15]. Powstały model jest udostępniony publiczne, każdy może go przetestować.

DeepStack

Model DeepStack rozwiązał HUNH przez połączenie metody CFR, sieci neuronowych wraz z dodatkowymi elementami. W rezultacie AI zaczęło osiągać bardzo dobre wyniki. Przetestowano go na 33 profesjonalnych graczach w wielu iteracjach gry. Model w większości przypadków wygrał [17]. Była to pierwsza wygrana AI z człowiekiem w normalnej wersji gry Poker Texas Hold'em z taką częstotliwością.

Libratus

Najbardziej zaawansowana sztuczna inteligencja, która jest wykorzystywana w grach HUNH. Jak wynika z testów wygrywa znacznie częściej niż DeepStack z profesjonalnymi graczami [16]. Przetestowano go z najlepszymi graczami na świecie, Dong Kim, Dan McAulay, Jimmy Chou i Jason Les [16]. AI wygrało z nimi, z ogromną przewagą.

2.5 Counterfactual Regret Minimization

2.5.1 Regret Matching

Jest to nieodłączna metoda uczenia AI w grach karcianych. Polega ona na liczeniu najlepszej strategii pod warunkiem, że znany jest wektor żalu w węźle. Taki wektor opisuje się jako tablicę wag o długości równej liczbie możliwych akcji gracza. Każda z tych wag opisuje, jak dużym błędem będzie niewykonania danego ruchu.

Poniżej przedstawiono wzór wynikający z tej metody, gdzie $R^T(a)$ jest omawianym wektorem [7]. Następnie, aby uzyskać nową strategię, usuwa się wartości ujemne, czyli takie, których gracz nie żałował (formuła nr 2.2). Potem sprawdzana się, czy ich suma jest większa od zera w celu wybrania odpowiedniego wzoru, formuła 2.1. W zależności od tego warunku otrzymuje się 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 się do tego stanu $D^T(I, a)$ [5]. Jeśli suma jest ujemna, to zostaje wybrana akcja z najwyższą wartością $D^T(I, a)$ [5].

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

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 węzła I. Do obliczenia takiego parametru została zdefiniowana wartość "counterfactual utility" $u_i(\sigma, I)$. Oznacza ona przewidywany wynik nagrody dla stanu I gdzie wszyscy gracze używają strategi σ [7]. Dodatkowo $\pi^\sigma(h, h')$ oznacza prawdopodobieństwo dostania się z historii h do nowego stanu h' przy strategii σ [7].

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

Na podstawie równania 2.5 można wyliczyć końcową 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 zastąpiona przez liczbę 1, ponieważ CFR zakłada, że dla $u_i(\sigma^t|_{I\to a}, I)$, gracz wykonuje zawsze akcję a z całkowitą pewnością [7].

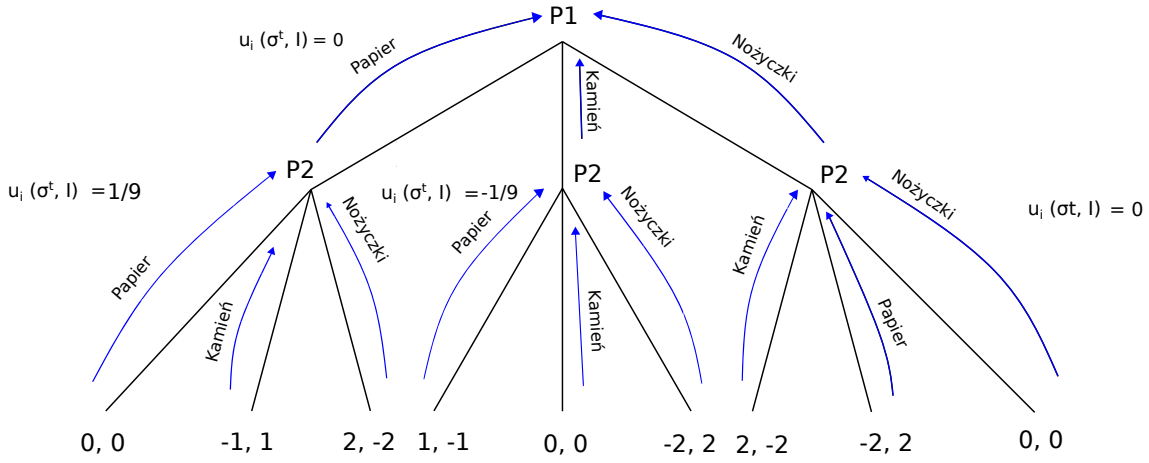
$$R^T_{i,\text{imm}}(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi^{\sigma}_{-i}(h) \sum_{h\in I, h'\in Z} (1 * u_i(h') - \pi^\sigma(h,h')u_i(h')) \tag{2.6}$$

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

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

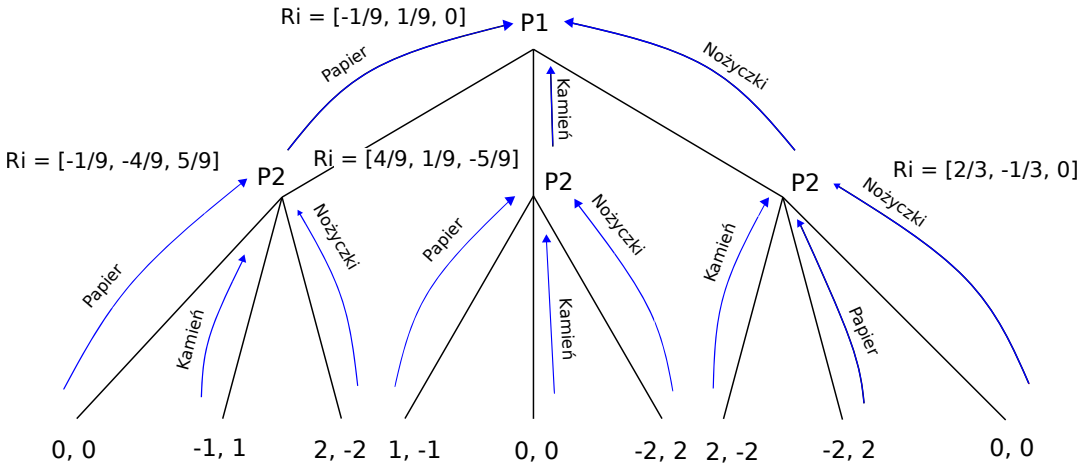
$$T * R^T_{i,\text{imm}}(I,a) = ((0-\frac{1}{3}),(-1-\frac{1}{3}),(2-\frac{1}{3}))\cdot\frac{1}{3} = (-\frac{1}{9}, -\frac{4}{9}, \frac{5}{9})$$

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



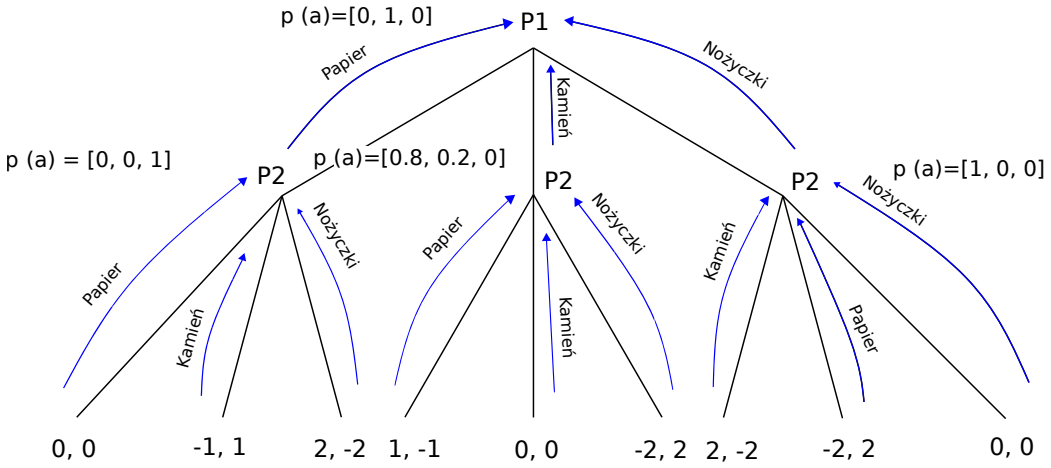
Rysunek 2.4: Przykład 'counterfactual utility'.

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



Rysunek 2.5: Przykład 'Immediate Counterfactual Regret'.

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



Rysunek 2.6: Przykład strategii.

Rys. 2.6 przedstawia wektory określające jakimi rozkładami akcji powinni się kierować gracze, aby osiągnąć najlepsze wyniki. Są to wektory wyliczone po pierwszej iteracji, w praktyce eksploracja drzewa i powyższe obliczenia są powtarzane wielokrotnie. Końcowym etapem algorytmu CFR jest policzenie średniej strategii, która ma reprezentować Równowagę Nasha [7].

2.6 Monte Carlo Conterfactual Regret Minimization

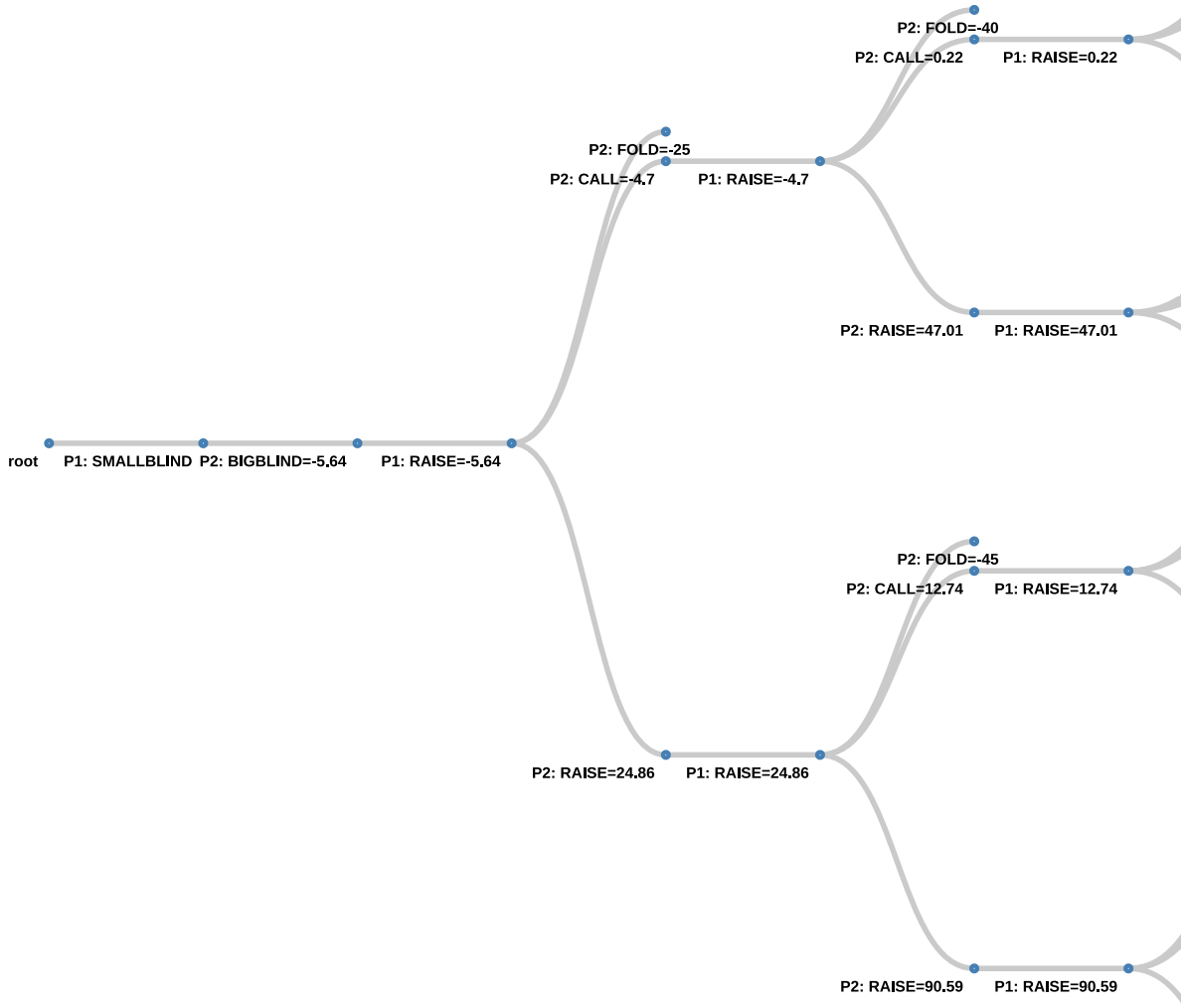
Algorytm CFR eksploruje całe drzewa decyzyjnego w jednej iteracji, co tworzy wymagania na dużą moc obliczeniową i długi czas uczenia. Dla małych gier takie rozwiązanie jest

akceptowalne, ale w przypadku większych środowisk jest nieefektywne. Spowodowało to powstanie nowszej wersji algorytmu CFR, MCCFR (*Monte Carlo Conterfactual Regret Minimization*) , który w każdą iterację eksploruje tylko część drzewa [19].

Metodę można podzielić na dwie odmiany, Outcome-Sampling oraz External-Sampling [19].

W pracy zostanie przedstawiony sposób MCCFR ES (*Monte Carlo Conterfactual Regret Minimization External Sampling*), ponieważ taki został zaimplementowany w algorytmie Deep CFR. MCCFR ES przed eksploracją drzewa wybiera kolejno spośród graczy jednego uczestnika, którego oznacza się jako "traverser" [5]. Eksploruje on wszystkie odpowiedzi ze swoich akcji w danym węźle. W międzyczasie inni uczestnicy wykonują pojedynczy ruch na podstawie swojej najlepszej strategii [19].

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



Rysunek 2.7: Część drzewa decyzyjnego MCCFR ES z graczem P2 jako "traverser".

Jest to przykład z jednej eksploracji gry, w praktyce powtarza się ten proces wielokrotnie, uzyskując różne wersje drzew.

Mając takie wyniki, można przystąpić do liczenia $R^T_{i,imm}(I)$ oraz poprawiania wartości przez "Regret Matching". MCCFR ES spełnia swoją funkcję, 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 [5] rozwija podstawową wersję metody CFR o sieci neuronowe. Taka modyfikacja była wymagana, aby utworzyć AI, które może rozwiązać nie tylko proste gry, ale też i duże jak HULH. Liczy on wektory w drzewie decyzyjnym przez algorytm MCCFR ES. Dodatkowo Deep CFR zbiega się do Równowagi Nasha szybciej niż popularny algorytm NFSP z 2016 roku [5].

Deep CFR wykorzystuje sieci neuronowe do przewidzenia wartości D^T(I, a) w podobnych obserwacjach, potem na podstawie predykcji liczy strategię ze wzoru nr 2.3. Następnie liczona jest zaktualizowana wersja wektora żalu, formuła nr 2.5. Obliczone wyniki są dodawane do buforów (B_1, B_2) $\in B_p$, a strategia do zbioru B_s.

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

Algorithm 1: Deep CFR

Wejście: B_p, B_s, θ_s, θ_p, P, N, K
Wyjście: θ_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 są bufory graczy (B_1, B_2) oraz kontener na strategie B_s. Dodatkowo metoda potrzebuje listę uczestników P, z których będzie wybierany cyklicznie gracza eksplorującego drzewo decyzyjne. Ostatnimi argumentami są iteracje gry N oraz liczba cykli K metody MCCFR ES.

W metodzie należy ustawić trzy pętle wraz z nową rundą, uzyskując początkową historię oraz krok gry t. Kolejnym etapem jest wykonanie funkcji *MCCFRES*. Przyjmuje ona na

wejściu sieci neuronowe obu graczy, listę uczestników p, numer kroku t, historię rundy h oraz bufory. Po k powtórzeniach następuje uczenie sieci neuronowej wybranego wcześniej gracza eksplorującego drzewo.

Model używa zmodyfikowanej wersji funkcji MSE (*Mean Square Error*) do liczenia błędu predykcji. Każdy wynik $(y_i - \hat{y}_i)^2$ mnożony jest przez krok t_i w, którym uzyskano y_i [5]. Dodatkowo jak wynika z badań, algorytm osiąga lepsze wyniki jeśli sieci neuronowe są trenowane od początku [5]. Dlatego należy przed funkcją *TRAIN* wyczyścić model.

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

Algorithm 2: Implementacja MCCFRES kożystając z sieci neuronowych

```
1  Function MCCFRES(θ_p, θ_{p−1}, p, t, h, B_p, B_s):
2      t_r ← h                                   ▷ sprawdzenie czyja jest aktualnie tura
3      if h jest stanem końcowym Z then
4          return u_p(h)
5      else if p = t_r then
6          D̂(I) ← obliczenie wektora D̂(I) używając h
7          σ(I) ← Obliczenie strategi σ_{t_r}(I), kożystając z D̂(I) i wzoru 1.1
8          for a in A(h) do
9              u_{t_r}(h) ← MCCFRES(θ_p, θ_{p−1}, p, t + 1, h + a, B_p,B_s)
10         u_{r_r}(σ, I) ←∑(u_{t_r}(h) · σ(I))
11         R^T_{t_r,imm}(I) ← (u_{t_r}(h) − u_{t_r}(σ, I))
12         B_{t_r} ← dodanie próbki do bufora [R^T_{t_r,imm}(I), h, t]
13     else
14         D̂(I) ← obliczenie wektora D̂(I) używając θ_p i stanu h
15         σ(I) ← Obliczenie strategi σ_p(I), kożystając z D̂(I) i wzoru 1.1
16         B_s ← dodanie próbki do bufora [σ(I), h, t]
17         a ←σ(I)
18         return MCCFRES(θ_p, θ_{p−1}, p, t + 1, h + a, B_p, B_s)
```

Implementacja MCCFR ES przedstawiona powyżej przy zadanych argumentach rozpoczyna się od sprawdzenia, który gracz rozpoczyna daną turę. Na podstawie tej informacji będzie wykonywana dalsza część algorytmu.

Pierwszym krokiem jest sprawdzenie, czy stan gry jest końcowym etapem. W przypadku prawdziwego warunku zwracana jest wygrana lub przegraną wartość stawki. Jeśli powyższy etap jest fałszywy, algorytm sprawdza, czy gracz jest oznaczony jako "traverser". Wtedy program używając sieci neuronowej gracza, otrzymuje wektor $\hat{D}(I)$ przez wprowadzenie do modelu informacji o widocznych kartach oraz dotychczasowej historii gry. Korzystając ze wzoru 2.3, liczy strategie, odczytuje u_{t_r} (h) wykonując rekurencję. 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. Następnie korzystając z tej sieci neuronowej i otrzymanej dystrybucji wykonuje nową akcję.

2.8 Podsumowanie

Środowiska z niepełnym zestawem informacji i brakiem deterministyczności są trudne do rozwiązania. Algorytmy tworzące modele dla takich gier są często skomplikowane i obciążające obliczeniowo. Przez takie cechy dopiero od niedawna zaczęły powstają algorytmy, zdolne pokonać ludzi w dużych grach karcianych jak "Cepheus", "DeepStack" lub "Libratus". Metody zdolne tworzyć takie AI dalej są rozwijane i aktualizowane z roku na rok. W taki sposób w 2014 roku powstał CFR, potem MCCFR i po para latach zastąpiono je przez CFR+, aż zaczęto wykorzystywać sieci neuronowe, przez co powstał Deep CFR będący tematyką pracy.

Rozdział dokładnie opisał działanie omawianego algorytmu przez przedstawienie terminów należących do działu matematyki Teoria Gier. Między 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 większości algorytmów sztucznej inteligencji gier karcianych.

Dobrze zaimplementowany algorytm Deep CFR przy prawidłowej parametryzacji i odpowiednio dużej liczby iteracji powinien zbiec się do punktu bliskiego takiego stanu. Pomimo tak optymistycznych założeń utworzone AI ("Cepheus", "DeepStack", "Libratus") potrzebowały bardzo wielu iteracji do nauczenia się dobrej gry w HULH i HUNH. Przykładem jest "Cepheus", który uczył się przez dwa miesiące, aby móc grać w HULH.

Rozdział 3

Implementacja algorytmu

Program Deep CFR został napisany w niniejszej pracy, korzystając z narzędzi, pozwalających na zredukowanie nadmiarowości kodu oraz na prostą implementację. W tym rozdziale skupiono się na opisaniu wybranych technologii, parametrów oraz funkcji, które znalazły się w implementacji algorytmu.

Bazą do napisanego kodu jest język programowania, Python 3.8. Zawiera on wiele technologii wspomagających uczenie maszynowe i rozległą społeczność wspierającą jego rozwój. Poniżej wylistowano i opisano główne narzędzia użyte w pracy.

TensorFlow Jest to wysokopoziomowe API dostępne dla takich języków jak Python, JavaScript, C++ lub Java [26]. Wykorzystuje je się głównie do zadań głębokiego uczenia maszynowego. Przez swoją prostotę, dostępność i dobrą dokumentację stał się jednym z najpopularniejszych narzędzi wykorzystywanych do tworzenia sztucznych inteligencji. Dodatkowo od niedawna biblioteki technologi Keras stały się częścią Tensorflow. Daje to możliwości znacznego zredukowania kodu przy prostych problemach, które często są rozwiązane przez funkcje w tym module.

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

Numpy Duża biblioteka do naukowych obliczeń na wielowymiarowych tablicach [23]. Jest nieodłącznym elementem przy pisaniu programów uczenia maszynowego, zwłaszcza jeśli korzysta się z bibliotek Tensorflow. Wynika to z faktu, że wiele funkcji tego API, jako argumenty przyjmuje typy danych powiązane z Numpy [26].

Tqdm Małe narzędzie w języku Python pozwalające na wyświetlenie postępu procesów w działającym programie. Przydatne w celach testowych. W pracy zostało użyte do śledzenia iteracji drzew decyzyjnych algorytmu Deep CFR.

TensorBoard Moduł należący do API Tensorflow. Wizualizuje postępy uczenia sieci neuronownych oraz ich jakość przez przedstawienie odpowiednich wykresów.

PyPokerEngine Biblioteka wspomagająca symulację gry Poker Texas Hold'em. Użyto jej jako podstawę do napisania środowiska HULH do interakcji z metodą Deep CFR.

3.1 Implementacja sieci neuronowych

Algorytm Deep CFR do działania wymaga dwóch sieci neuronowych, jedna ma rozpoznawać strategie σ^t, a druga przypisana do określonego gracza przewiduje opłacalność akcji 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 część rozdziału dokładnie opisuje dodatkowe elementy, ważne podczas uczenia modelu.

Tabela 3.1: Podstawowe parametry sieci neuronowej θ_s.

parametr	użyte wartości
rozmiar danych wejściowych	(3, 52)
rozmiar danych wyjściowych	(None, 3)
prędkość uczenia	0.0001
końcowa funkcja aktywacyjna	*softmax*
rozmiar bufora	600 000
maksymalna liczba iteracji	5 000
rozmiar *minibatch*	500
parametr *patience*	40

Tabela 3.2: Podstawowe parametry sieci neuronowej θ_p.

parametr	użyte wartości
rozmiar danych wejściowych	(3, 52)
rozmiar danych wyjściowych	(None, 3)
prędkość 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 θ_1, θ_2, θ_s o nieskomplikowanej architekturze. Z tego powodu wykorzystano bibliotekę Keras, która do takich przypadków jest dobrym rozwiązaniem. Dodatkowo założono, że wszystkie modele będą miały podobną budowę poza ostatnią warstwą z innymi funkcjami aktywacyjnymi. Wynika to z faktu, że algorytm Deep CFR korzysta z sieci, które dostają dane wejściowe o tej samej strukturze, ale zwracają D_p^t lub σ_p^t.

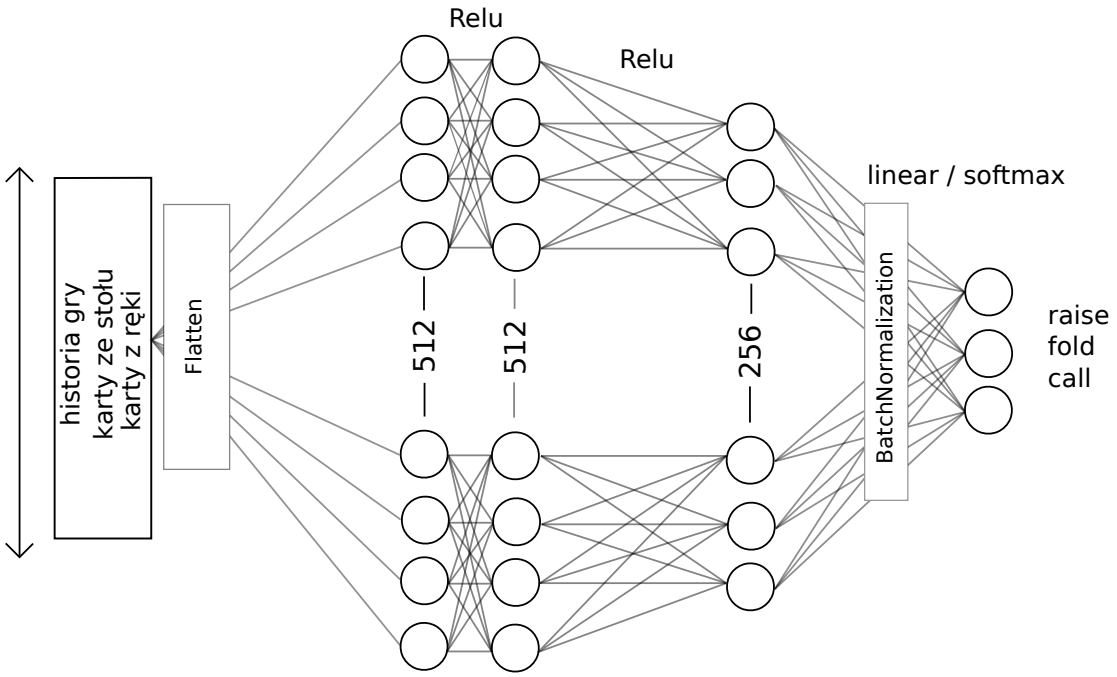
Architektura sieci neuronowych składa się z 7 elementów, wejścia, wyjścia, bloku o nazwie *Flatten* [26], trzech warstw ukrytych zakończonych normalizacją i wyjściem w postaci wektora o trzech polach.

Początkowo sieci dostają tablicę o wymiarach (3, 52), czyli kolumnę kart ze stołu, z ręki oraz historię ostatniej rundy gry. Model w dalszych obliczeniach musi przekształcić takie dane do formy jedno-wymiarowej (None, 156), co robi warstwa *Flatten*. Kolejne dwa elementy składają się z 512 wag, trzecia warstwa ukryta posiada ich 256. Na trzech wymienionych elementach ustawiono funkcję *Relu*. Całość kończy się wyjściem wektora reprezentującego możliwe akcje gry HULH. Dodatkowo dane są poddawane normalizacji przez warstwę o nazwie *BatchNormalization* [26].

Wyjście modeli, aby zwracało prawidłowe liczby, używa innej funkcji aktywacyjnej niż poprzednie elementy. W przypadku predykcji strategii σ_p^t wybrano *softmax*. Sieci neuronowe θ_1, θ_2 używają funkcji *linear*. Dokładna architektura sieci jest zaprezentowana na rys. 3.1.

W trakcie trenowania sieci korzystają z funkcji optymalizującej Adam. Prędkość uczenia jest równa 0,0001, co spowoduje powolne uczenie, ale zminimalizuje szanse na ominięcie minimum globalnego.

Jak wynika z dokumentu prezentującego algorytm Deep CFR, uzyskuje on lepsze wyniki, jeśli sieci neuronowe są uczone za każdym razem od początku przy losowo ustawionych zerach w wagach [5]. W tym celu ustawiono w każdej warstwie funkcję, która tworzy losowo wartości w przedziałach od -0,005 do 0,005.



Rysunek 3.1: Architektura sieci neuronowych wykorzystana w programie.

Ostatnim elementem sieci jest funkcja licząca 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 udostępnia Tensorflow [26].

Wszystkie te elementy zostały uwzględnione w funkcjach klasy *Poker_network* przedstawionej na rys. 3.4.

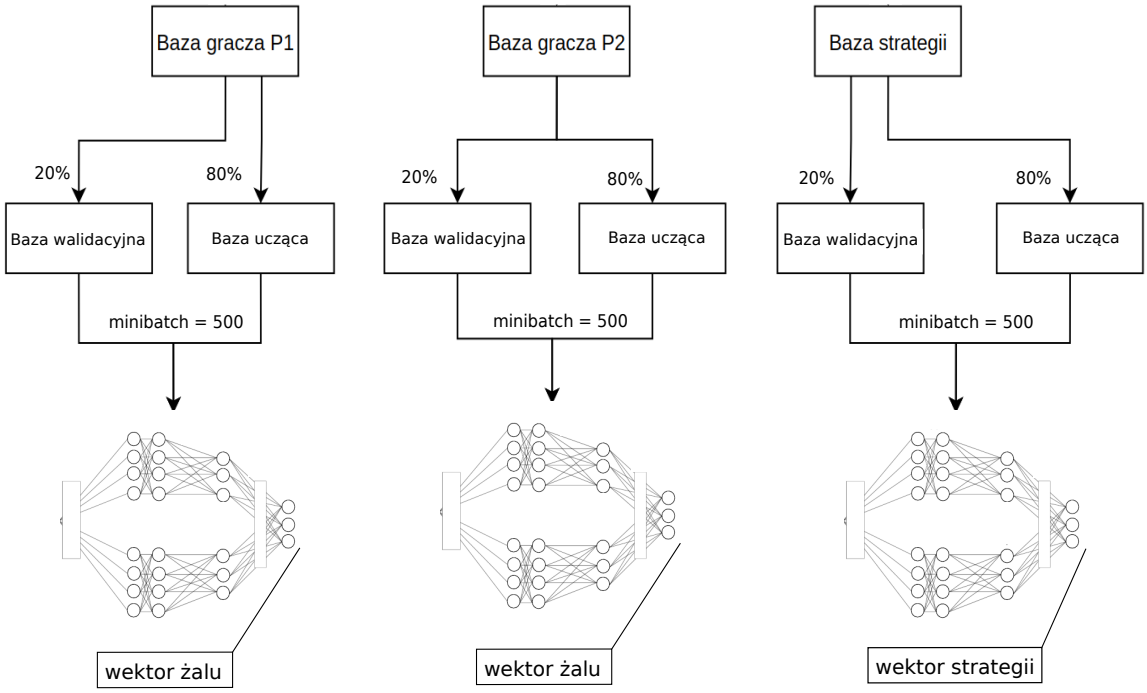
3.1.2 Budowa zbiorów danych

Jak wynika z algorytmu Deep CFR, w programie muszą być zawarte trzy bufory. Maksymalna pojemność kontenerów B_1 i B_2 jest równa 300 000 próbek. Bufor B_s może posiadać ich 200 000. Każdy z dodawanych elementów do bufora składa się z trzech połączonych wektorów o rozmiarach 52.

Zaimplementowano klasę *Memory* zarządzającą tymi buforami, które zachowują się jak kolejki *deque*, w przypadku przepełnienia jest usuwany najstarszy wpis.

Mając 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 uczący zajmujący 80% całej bazy. Wykorzystuje się go do poprawiania wag modelu sekwencyjnie. Drugim zbiorem są 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*, będącym parametrem określającym mały podzbiór bazy użyty do trenowania sieci w pojedynczym kroku. Jego liczebność jest równa 500 próbek.

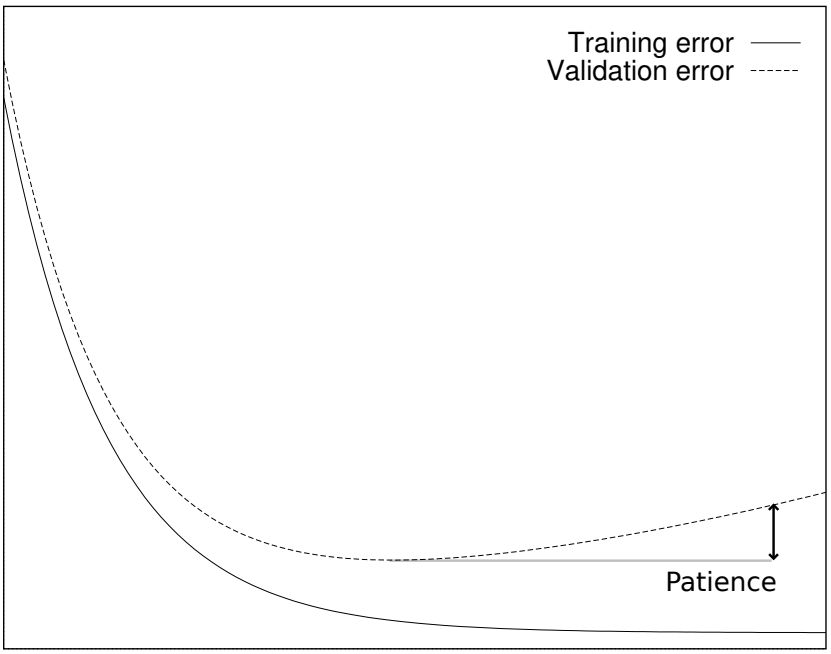


Rysunek 3.2: Podział danych.

3.1.3 Proces uczenia

Algorytm Deep CFR do eksploracji wykorzystuje metodę MCCFR ES, która eksploruje w jednej iteracji wiele razy drzewo dezyzyjne gromadząc przy tym próbki w buforach. Po zakończeniu wszystkich powtórzeń zachodzi etap uczenia sieci neuronowych θ_1, θ_2 wykonując maksymalnie 5 000 aktualizacji wag. Taki schemat działania jest wykonywany dla obu graczy i powtarza się wielokrotnie. Końcowym zadaniem jest wytrenowanie sieci neuronowej θ_s. Wszystkie te elementy zostały połączone przez klasę *Brain*, agregującą obiekty *Memory* oraz *Poker_network*, rys. 3.4.

Dodatkowo w celu poprawienia wyników uczenia użyto funkcji EarlyStopping [26]. Zatrzymuje ona iteracje modelu w przypadku kiedy błąd predykcji na zbiorze walidacyjnym wzrośnie odpowiednio wysoko. Jest to sprawdzane na podstawie argumentu *patience*, który określa próg, po którym nauka się kończy. Taka procedura została zastosowana, aby zminimalizować szanse na przetrenowanie modeli, a wraz z tym ich gorszą jakość [20]. Rys. 3.3 prezentuje przykładowy punkt zakończenia nauki modelu.



Rysunek 3.3: Przykładowy punkt zatrzymania się uczenia po zastosowaniu *EarlyStopping* [20].

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

3.2 Implementacja środowiska

Do symulacji gry HULH z którą model będzie się komunikował wykonując akcje, użyto blblioteki PyPokerEngine. Klasa *HULH_Emulator* zaimplementowana w programie pozwala na utworzenie obiektu zarządzającego taką grą. Przy tworzeniu obiektu zostają zdefiniowane nazwy graczy P1 i P2, które w dalszej części będą używane do śledzenia histori gry oraz wykonywania wszelkich akcji graczy. W tabeli nr 3.3 wylistowano parametry ustawione w programie, związane z zasadami grą.

Zaimplementowane środowisko każdej wprowadzonej akcji zwraca dwa elementy, obiekt *State* określający stan gry oraz historię gry w formie słownika. Obie wartości są używane do śledzenia stanu środowiska w drzewie decyzyjnym. Informacje jak karty na stole, historia gry lub wygrana gracza jest zarządzane przez obiekt.

Sama gra działa w taki sposób przy zadanych parametrach aby każde podbicie stawki było wielkości dużej w ciemno. Dzięki takiemu założeniu gra nie kończy się zbyt szybko podwarunkiem, że żaden z graczy nie wykona akcji *fold*. Dodatkowo ustalono, że po każdej rundzie środowisko jest resetowane. Taki proces powoduje powstawanie mniejszych drzew decyzyjne co pozwala na szybszą naukę modeli przez krutszy czas poświęcany na eksplorację gry przez MCCFR ES.

Tabela 3.3: Parametry gry HULH.

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

3.3 Implementacja Deep CFR

Klasa *DCFR* zawiera implementację algorytmu Deep CFR. Przy tworzeniu obiektu powstają w konstruktorze trzy elementy gracz, oponent oraz strategia σ z klasy *Brain*. Do tego ustawiane jest środowisko *HULH_Emulator* przez referencje.

Cały proces powstawania modelu rozpoczyna się od funkcji *iterate*, która wykonuje trzy pętle *for*, dla iteracji algorytmu po 50 razy oraz powtórzeń eksploracji drzewa dla każdego z gracza po 270 razy. W pierwszej z nich środowisko tworzy nową grę, w ostatniej pętli jest wykonywana funkcja *__traverse*, działająca według metody MCCFR ES. Dostaje ona argumenty *state* czyli obiekt określający stan gry, *events* - dotychczasową historię oraz *timestep*. Ostatni argument *verbose* jest opcialny i służy tylko do wizualizacji powstałego drzewa decyzyjnego. Funckja działa po przez rekurencję co oznacza, że aktywuje samą siebie wielkrotnie co pozwala na sprawdzenie każdej wersji gry. Po zakończeniu działania MCCFR ES, rozpoczyna się uczenie sieci neuronowej θ_p. I powtażanie powyrzszego procesu. Ostatnim etapem jest trenowanie sieci θ_s oraz zapisanie jej do pliku. Aby spełnic warunek utworzenia pięciu modeli kolejno ustawiono w funkcji *iterate* opciajlny argument *checkpoint*. Przyjmuje on bazowo wartość *None*. Przy uruchamianiu programu zmieniono ją na liczbę 10 co oznacza, że co 10 iteracji będzie trenowana sieć θ_s z uzberanego bufora i zapisywana do pliku.

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

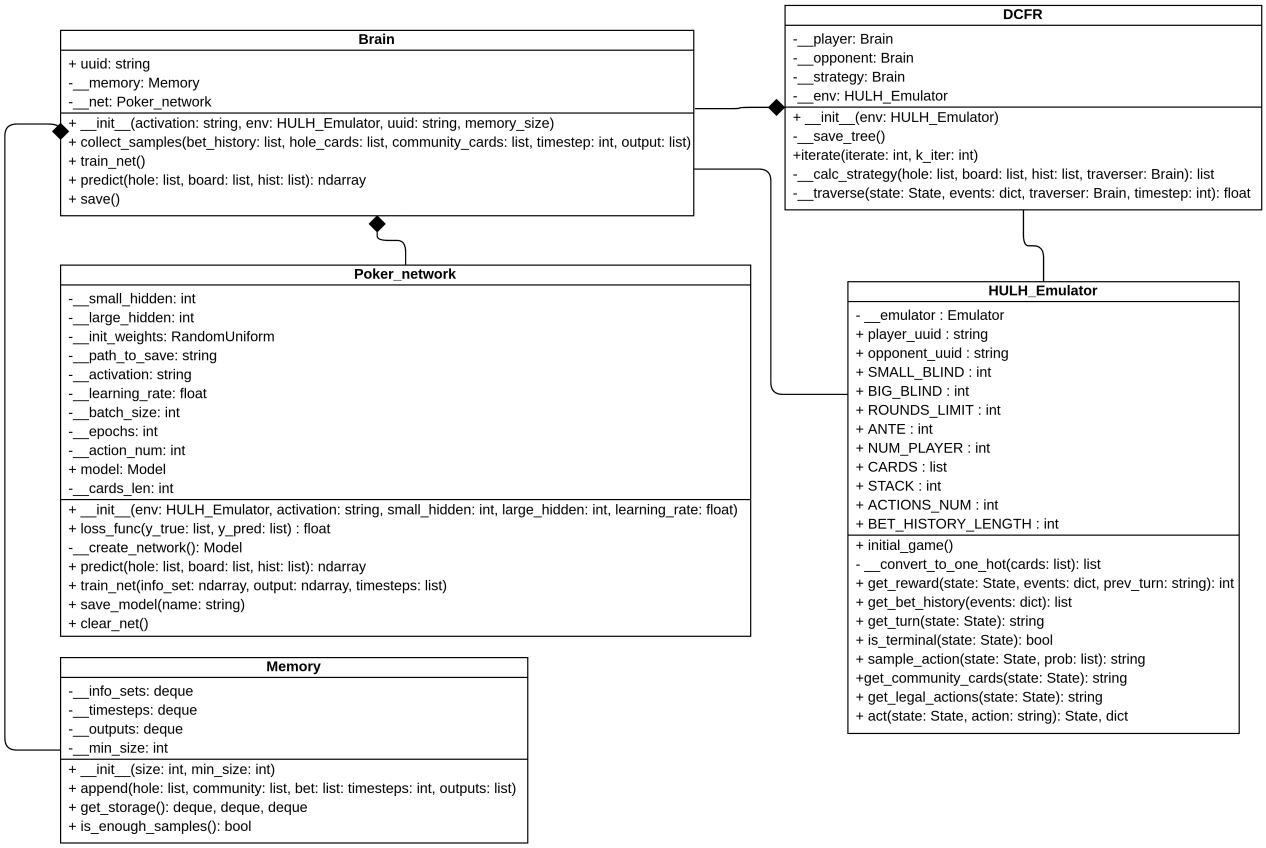
Tabela 3.4: Parametry algorytmu Deep CFR.

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 implementację algorytmu wraz z użytymi technologiami. Program następnie został uruchomiony z zadanymi parametrani jak 50 iteracji po 270 eksploracji drzewa. Wykonywał sie on przez okres trzech dni używając GPU. Postępy tworzonych pięciu modeli były na bieżąco analizowane i zapisywane. Przedstawiono między innymi struktórę sieci neuronowych, budowę zbioru danych, działanie środowiska HULH oraz funkcje związane z uczeniem algorytmu. Pokazano cel użycia takich metod jak *EarlyStopping* lub *Flatten*.

Następny rozdział przedstawi zapisane wyniki ilustrujące jakość utworzonych modeli oraz sprawdzi, który z nich będzie najlepiej grał w HULH.



Rysunek 3.4: Diagram UML projektu.

Rozdział 4

Wyniki

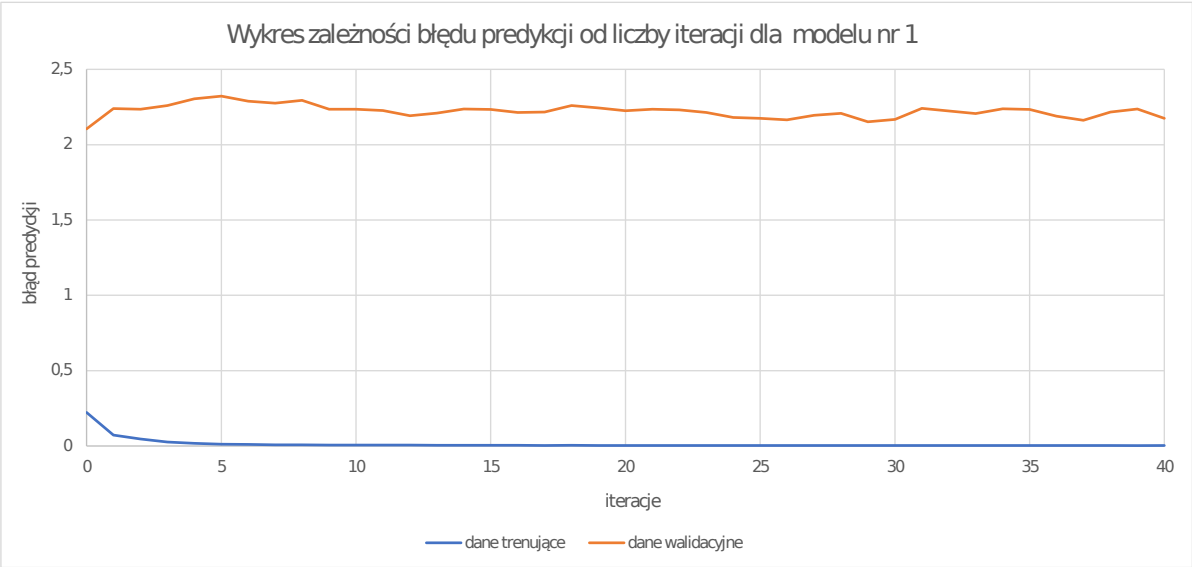
Rozdział przedstawia rezultaty powstałe po około czterech dniach nauki modelu, następnie pezedstawiono wyniki wszystkich kombinacji rzgrywek między modelami oraz średnie wartości wygrywanych pól. Zostaną tutaj zaprezentowane różnice między utworzonymi AI wraz z ich przewidywaną jakością korzystając z odpowiednich wykresów bazujących na danych pobranych z modułu TensorBoard. Dodatkowo zostanie dokonana próba wyboru najlepszego modelu na podstawie rozegranego turnieju oraz klasyfikacja jego stylu gry na podstawie pojedyńczego zapisu rozgrywki z drugim najlepszym modelem. Ostatnim etapem będze sprawdzenie o ile lepiej radzi sobie najlepsze AI z graczami wykonującymi tylko ruchy w pełni losowe oraz takim, który gra całkowicie szczerze.

4.1 Proces uczenia modelów rozpoznawania

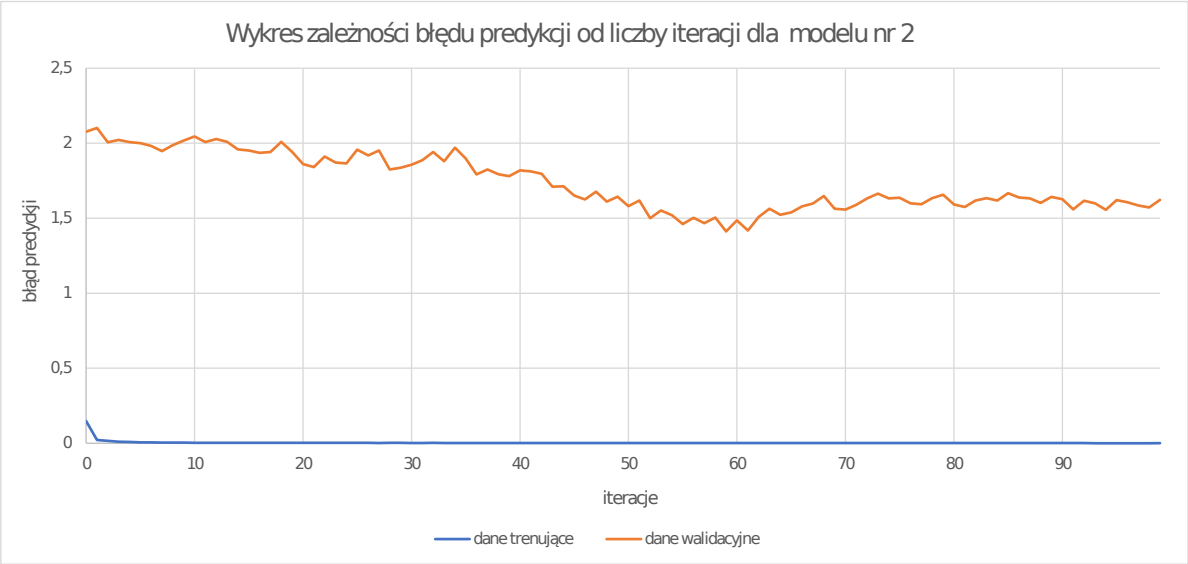
Algorytm Deep CFR zdołał utworzyć pięć modeli rozpoznawania, gdzie każdy proces nauki był śledzony przez funkcje biblioteki *TensorBoard*. Uzyskane dane następnie zostały użyte do utworzenia czytelnych wykresów ilustrujących zależność błędu predykcji od liczby iteracji.

Pierwsze utworzone AI powstał po 10 iteracjach metody Deep CFR co zajeło około 5 godzin. Następnie rozpoczęła się nauka sieci θ_s ze zbiorem danych B_s zapełnionym przez około 300 000 próbek. Można zaóważyć na rys. 4.1, że wartość błędu po 40 krokach uczenia nie zmieniła się mocno dla obu zbiorów. W przypadku danych walidacyjnych sieć neuronowa przy dokonywaniu predykcji powdowała duże oscylacje wartości pozostając na poziomie około 2,2 co przyczyniło sie do zakończenia procesu przedwcześnie przez *Early-Stopping*. Przy takim okresie model zmniejszył błąd predykcji zbioru uczącego wynoszący początkowo 0,2 do wartości bliskiej 0.

Analizując taki wykres można stwierdzić, że model nauczył się z zebranych danych zależności między wprowadzanymi obserwacjami, a zwracanymi akcjami. Dodatkowo rys. 4.6 pokazuje, że takie AI gra lepiej względem wielu utworzonych poźniej modeli.



Rysunek 4.1: Wyniki uczenia modelu nr 1.

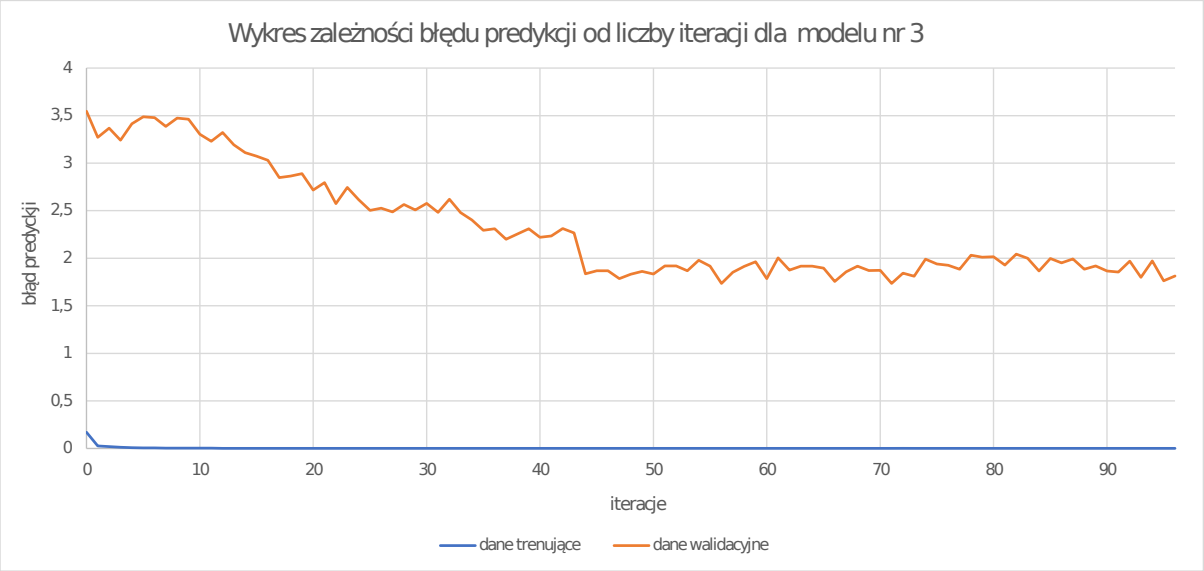


Rysunek 4.2: Wyniki uczenia modelu nr 2.

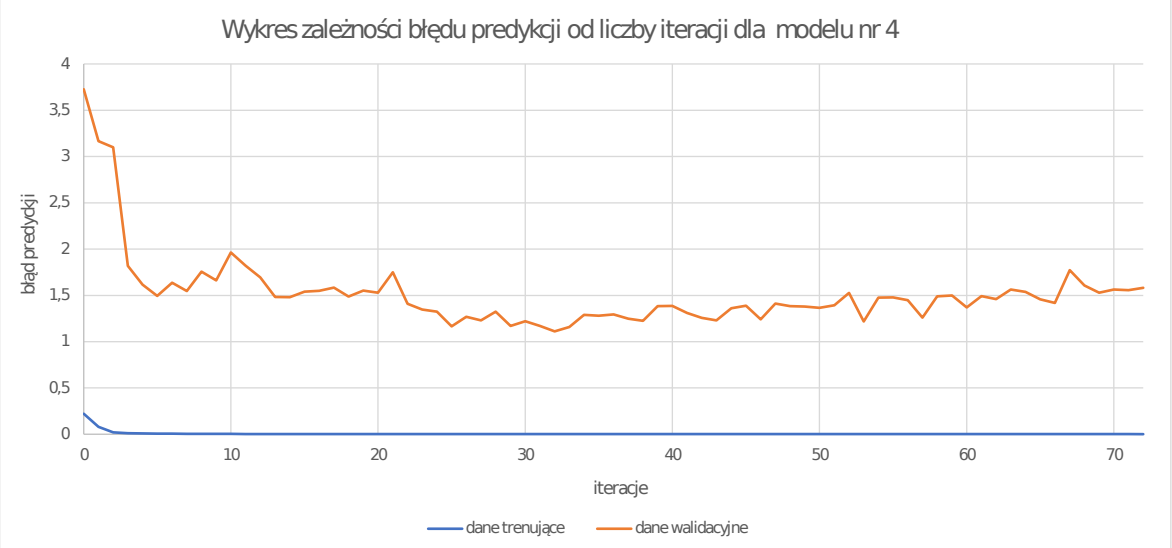
Po 20 powtórzeniach algorytmu Deep CFR, utworzono drugi model rozpoznawania. Nastąpiło to po około 12 godzinach. Algorym do nauki użył prawie całego zapełnionego buforu B_s. W zbiorze znajdowały się nowe próbki oraz te które zebrano przed nauką pierwszego AI. Proces uczenia przebiegał początkowo w podobny sposób jak w poprzednim etapie. Model zaczął od błędu predykcji 2,2 dla zbioru walidującego i błędu 0,2 dla buforu uczącego. Pierwszą różnicą względem poprzedniego etapu jest lepsza nauka na podstawie buforu walidacyjnego, po 10 powtórzeniach wartości zaczęły mocno maleć aż do błędu predykcji wynoszącej około 1,4. Następnie pojawił się krótki wzrost wartości i utrzymywanie się na tym samym poziomie z lekkimi oscylacjami co spowodowało zakończenie procesu. Wykres 4.2 dobrze pokazuje cel używania biblioteki *EarlyStopping*. Prawdopodobnie przy większej liczbie iteracji model zaczą by uzyskiwać coraz gorsze wartości

predykcji zbioru walidacyjnego. Podsumowując model uzyskał lepsze wyniki względem etapu pierwszego co może wynikać z większego zbioru danych. Po mimo takich zalet rys. 4.6 pokazuje, że utworzony model przegrywa z AI numer 1 przy 400 potwórzonych 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 uczącego była podobna jak w poprzenich etapach. Główną różnicą jest bufor walidacyjny, dla którego błąd początkowo wyniusł 3,5 i zmalał do 1,7.



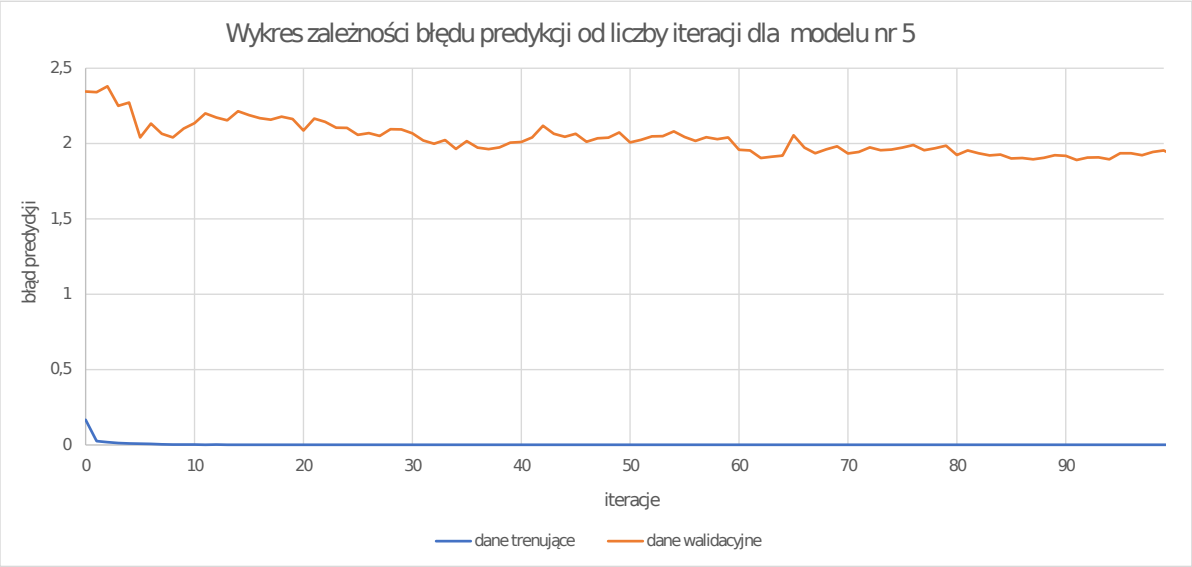
Rysunek 4.3: Wyniki uczenia modelu nr 3.



Rysunek 4.4: Wyniki uczenia modelu nr 4.

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

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

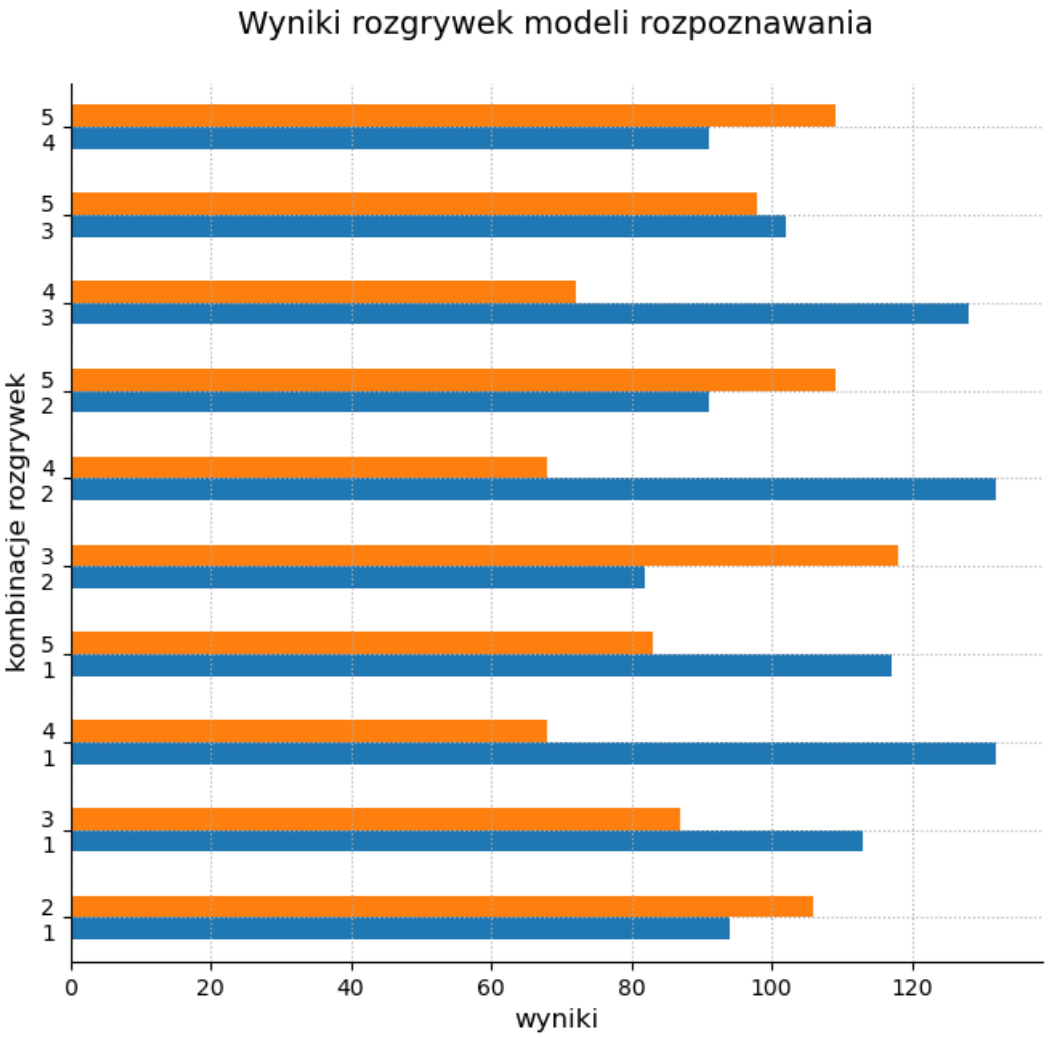


Rysunek 4.5: Wyniki uczenia modelu nr 5.

Podsumowując powyżesze wyniki można stwierdzić, że zbierane dane w buforze B_s przez Deep CFR są łatwe w znajdowaniu zależności. Główną przyczyną mogą 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 większym zbiorze informacji wejściowych proces uczenia by nie przebiegał tak szybko. W takim przypadku model by musiał uwzględnić dodatkowe czynniki jak numer rundy, liczba pozostałej sumy żetonów gracza lub przewaga przeciwnika.

4.2 Wyniki rozgrywek modeli

W celu przetestowania jakości utworzonych modeli przeprowadzono 10 rozgrywek gdzie każda z nich to pojedyncza kombinacja dwóch AI. Każdą grę powtórzono 200 razy po pięć rund tak, aby zminimalizować czynnik losowości i umożliwić sprawdzenie, kto statystycznie częściej wygrywa. Następnie sporządzono trzy wykresy ilustrujących wyniki tych rozgrywek. Rys. 4.6 przedstawia każdą kombinację gier z przypisaną liczbą wygranych każdemu z modeli. Rys. 4.7 i 4.8 mają za zadanie pokazać średnią wygrywanych i przegrywanych pul przez graczy, a rys. 4.9 pokazuje rozkład wykonanych akcji. Takie wykresy pozwolą stwierdzić, który z modeli częściej wygrywa, ale też częściej ryzykuje, przegrywając więcej, a który gra ostrożniej.

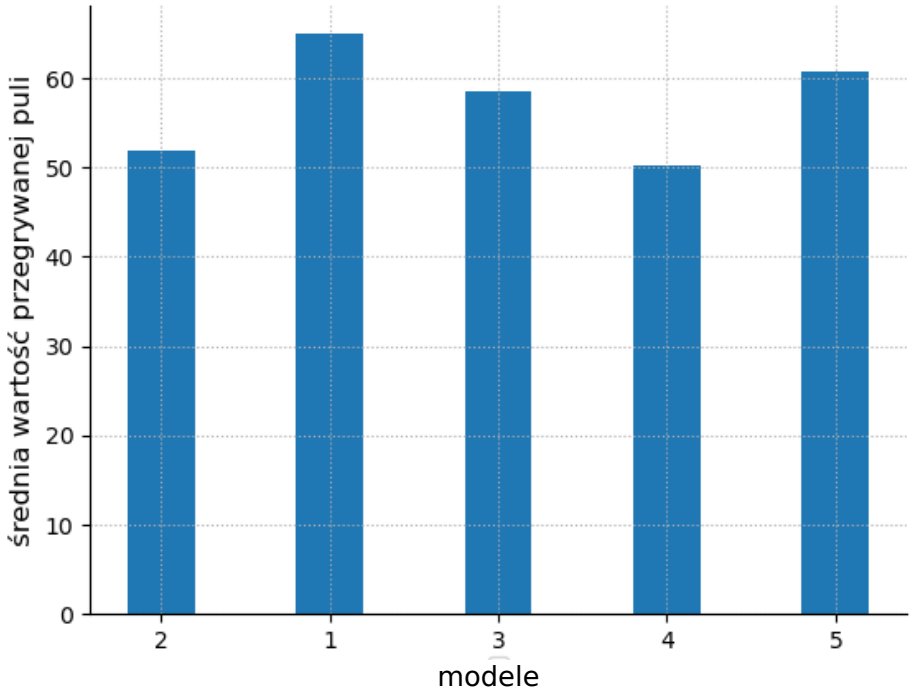


Rysunek 4.6: Kombinacje rozgrywek między utworzonymi modelami.

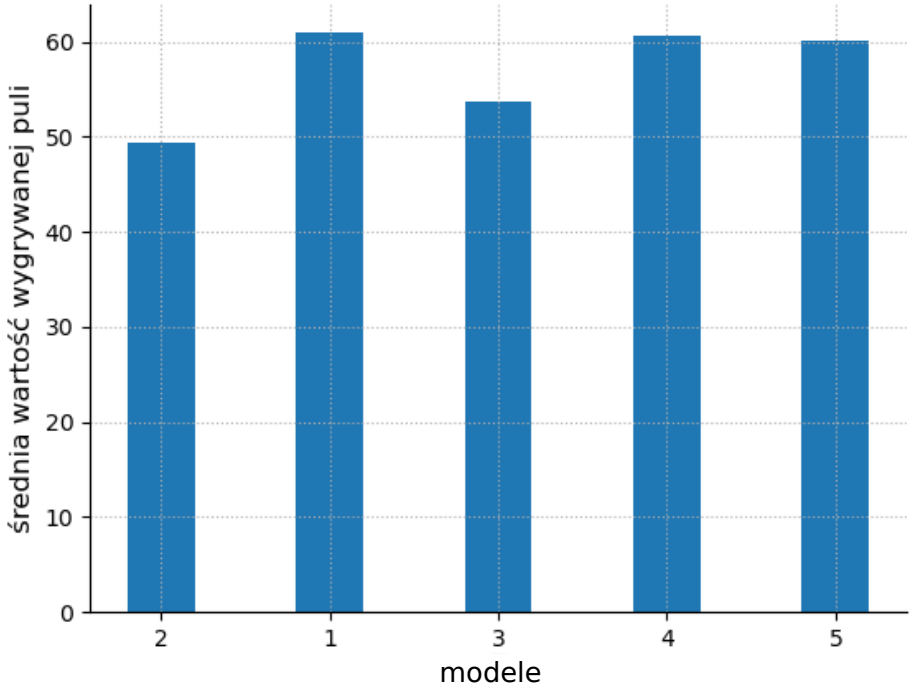
Analizując rys. 4.6 można zaóważyć, że model 1 wygrywał z resztą utworzonych AI najczęściej oprócz nr 2. Najwięszą przewagę zdobył względem modelu numer 4, około 130 wygranych względem 70 porażek oraz około 118 zwycięstw z nr 5. Rozgrywki z resztą uczestinków, cechuje się niewielkimi przewagami gracza. Przy wielokrotnym powtarzaniu eksperymentu nr 3 i nr 2 mogł by się okazać lepsze przez występujący w grze czynnik losowości, który ma wpływ na uzyskiwane karty.

Kolejny obiekt zdobywający najlepsze wyniki to nr 3. Osiągnął on tak samo dużą różnicę między wygranymi, a przegranymi z modelem nr 4. Na podstawie takich informacji można stwierdzić, że AI powstałe po 40 iteracjach przegrywa najczęściej, a utworzone w 1 i 3 etapie najżadziej. W dalszej cześci rozdziału zostanie zbadana możliwa przyczyna takich wyników. Powtarzając eksperyment uzyskiwano podobne rezultaty z niewielkimi zmianami, jak więsza częstotliwość zwycięstw modelu nr 2 i nr 3 względem nr 1 lub więszka liczba wygranych AI nr 5.

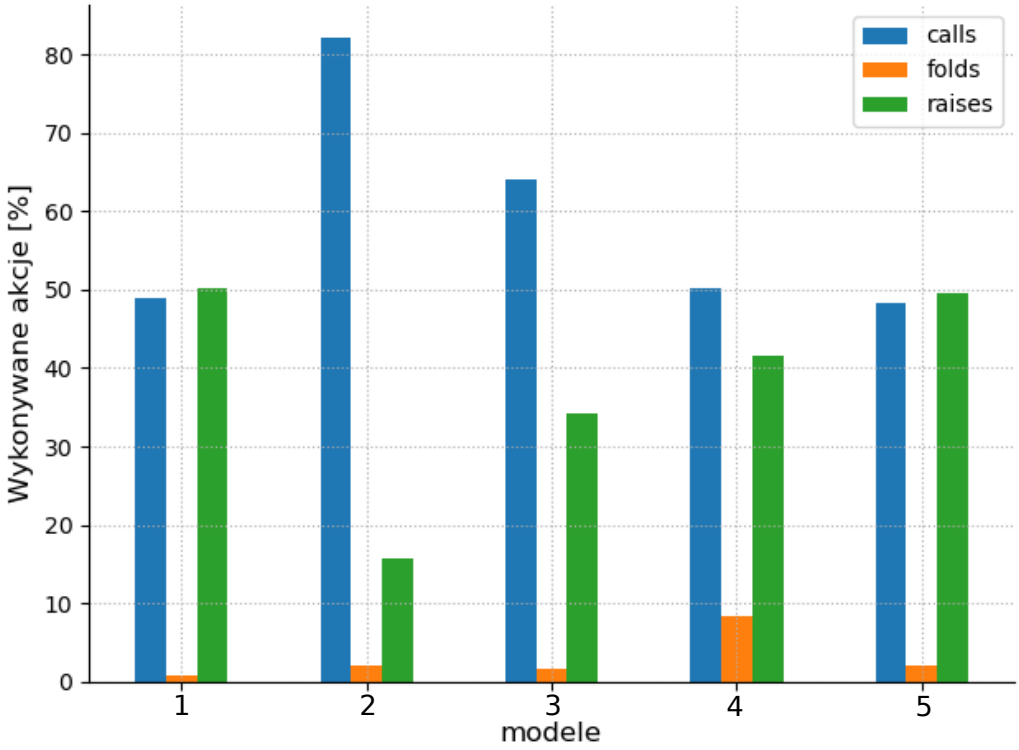
Problemem wykresu rys. 4.6 jest pokazywanie tylko częstotliwości zwycięstw modeli ale nie zawiera informacji o sposobie gry każdego z graczy. Taka wiedza pozwoliła by na stwierdzenie możliwych przyczyn powstałych wyników. W tym celu zebrano wszystkie wygrane wartości przez graczy z każdej rundy, a następnie obliczono ich średnie. Rys. 4.7 przedstawia uśrednione wyniki przegrywanych pulli przez każdego z graczy, rys. 4.8 prezentuje odwrotną cechę. Analizując oba rysunki, można zaóważyć, że model nr 1, który okazał się wcześniej najlepszym AI, wygyrywa i traci najwięcej w rundach, ale też najczęściej wygyrwa. Rys. 4.9 pokazuje też, że model nr 1 wykonuje tak samo często akcję *call* i *fold* oraz bardzo rzadko akcję *fold*. Jest najbardziej zrównoważonym graczem, który uzyskuje przy tym najbardziej skrajne wyniki. Zaletą modelu jest to, że średnie wartości są na podobnym poziomie, co oznacza, że wygrywa i tyle samo traci. Gracz nr 3, który równie często wygrywał, osiąga na tych wykresach gorsze wyniki. Jego średnia wartość przegrywanej puli jest dużo wieksza niż wygrywana. Model nr 5 gra podobnie do nr 1 co widać na rys. 4.9. Też ma równy rozkład wykonanych akcji *call* i *raise*. Dodatkowo jego częstotliwośc wygyrwania przedstawiona na rys. 4.6 pokazuje, że nie uzyskuje on tak dobrych wyników jak nr 1. AI nr 2 gra najmniej agresywnie, najczęściej wykonywał *call*. Może to być powód tak niskiej wygyranej puli. Najgorze wyniki ma gracz nr 4, który też najczęściej pasował. Uzyskał on znacznie większą średnia wartość wygranej względem przegranej przez taką strategię.



Rysunek 4.7: Wyniki przegrywanych pul przez modele.



Rysunek 4.8: Wyniki wygrywanych pul przez modele.



Rysunek 4.9: Rozkład wybieranych akcji przez modele.

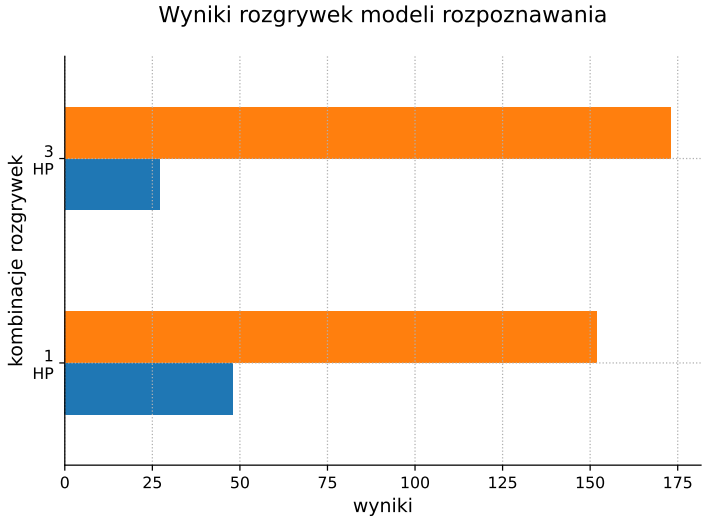
Tabela 4.1: Dokładne wyniki uśrednionych puli zdobywanych lub traconych przez modele.

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

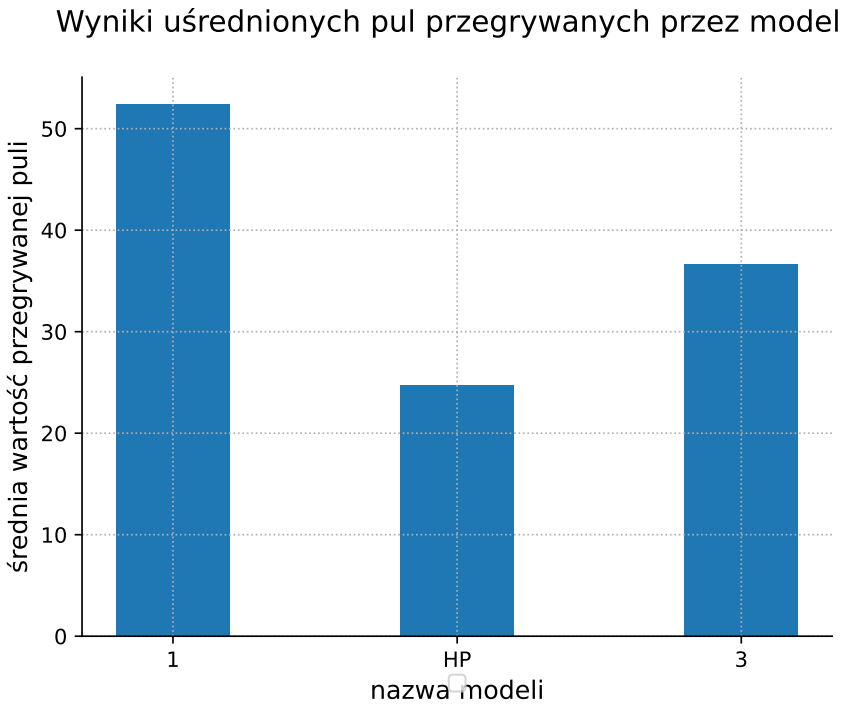
4.3 Porównanie modeli z graczem nieblefującym

Końcowym etapem oceny jakości algorytmu Deep CFR jest wykonanie gry między najlepszymi utworzonymi modelami, a graczem który nie blefuje. Jest to program symulujący grę w HULH, wykonując tylko akcje opierające się na sile dostępnych kart. W tym celu wykorzystano funkcję zawartą w bibliotece PyPokerEngine - *estimate_hole_card_win_rate*. Wykonuje ona określoną liczbę iteracji mozliwych wersji gier, a nastepnie liczy szanse wygrania z dostepnymi kartami. W zalezności od zwracanej wartości nastepnie wykonuje akcję *fold* lub *call*.

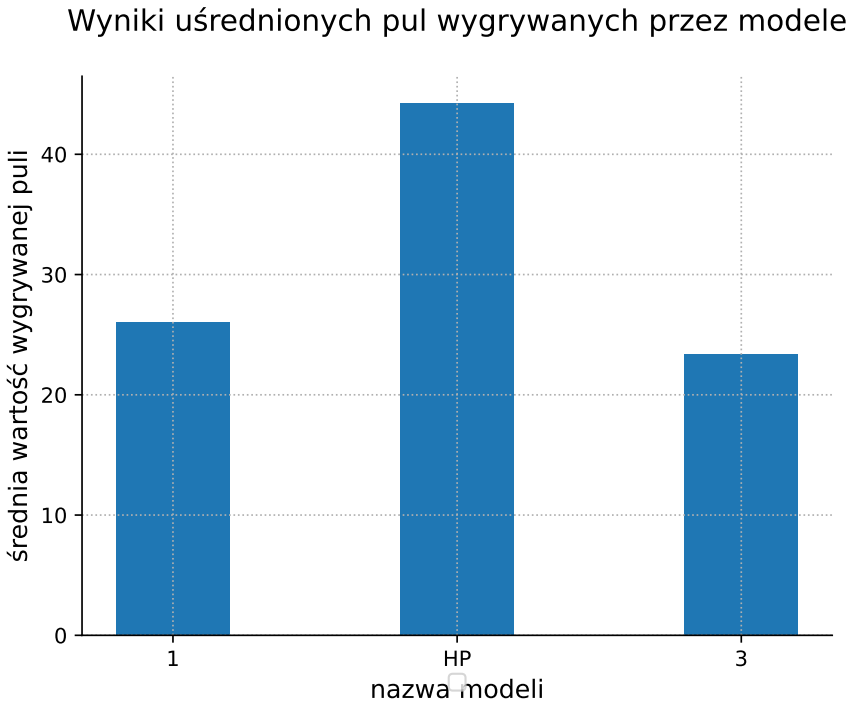
Rozegrano 2 gry po 200 powtórzeń, gdzie każda z nich składała się z 5 rund. Na rys. 4.9, 4.10, 4.11 zaprezentowano wyniki uzyskanych rozgrywek. Pierwszy z wykresów przedstawia częstotliwośc wygrywania modeli z graczem szczerym (HP). W przypadku nr 3 liczba zwycięstw jest równa liczbie porażek. Drugi z sztucznych inteligencji uzyskał nieznacznie lepsze wyniki. Dodatkowo analizując następne wykresy można zauważyć, że gracz HP przegrywa znacznie wiekszą średnią pulę względem modeli ale też wygrywa większą stawkę. Oznacza to, że po mimo braku wykonywanej akcji *raise* program częściej ryzykuje.



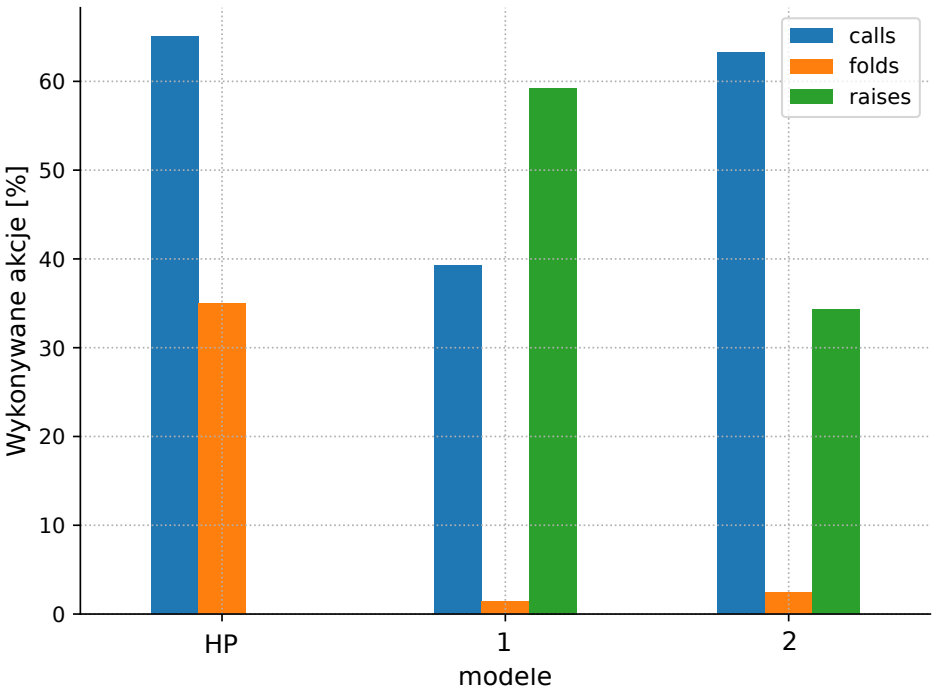
Rysunek 4.10: Wyniki rozgrywek między modelami nr 1, nr 3 i graczem nieblefującym.



Rysunek 4.11: Wyniki przegrywanych pul między modelami nr 1, nr 3 i graczem nieblefującym.



Rysunek 4.12: Wyniki wygrywanej pul między modelami nr 1, nr 3 i graczem nieblefującym.



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

4.4 Podsumowanie

Rozdział przedstawił wyniki utworzonych AI. Rezultaty okazały się mało optymistyczne. Model wytrenowany w ostatniej iteracji nie uzyskuje najlepszych wyników, a przedostanie AI przegrywa najczęściej.

Rozdział 5

Podsumowanie

Praca przedstawiła problem uczenia przez wzmacnianie w środowiskach częściowo obserwowalnych. Rozdziały 1 i 2 omówiły źródło takich problemów. Z tego powodu należy korzystać ze skomplikowanych algorytmów, które potrafią powiązać obserwacje z najlepszymi akcjami w danym stanie. W przypadku gier karcianych w tym celu korzysta się z algorytmów bazujących na metodzie CFR.

Zaimplementowany algorytm w niniejszej pracy, Deep CFR usprawnił proces znany w metodzie CFR, przez użycie sieci neuronowych. Taka konstrukcja przyspiesza zbieganie się AI w dużych grach jak HULH do Równowagi Nasha [5]. Program użyty w pracy nie pozwolił na określenie stopnia bliskości do takiego stanu. Wynika to z faktu, że algorytmy używające metody CFR korzystają z metryki *Exploaibility* bazującej na wartości BR (*Best Response*) do śledzenia postępów wybieranych strategii przez AI [21]. Taki element głównie się implementuje w grach abstrakcyjnych jak *Kuhn Poker* z powodu bardzo dużej obciążalności obliczeniowej. Duże gry stosują mniej dokładną metryki jak LBR(*Local Best Response*) lub RL_BR (*RL Best Response*), które z dużym przybliżeniem zwracają bliskość modelu do Równowagi Nasha [21] [22]. Charakteryzują się one dużym skomplikowaniem implementacyjnym.

Z tego powodu do oceny jakości Deep CFR użyto prostszej metody. Rozegrano wiele gier HULH między utworzonymi modelami, a następnie przeanalizowano wyniki i wybrano najlepszego z nich na bazie uzyskanych cech. Rezultaty okazały się mało optymistyczne w porównaniu do czasu poświęconego 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 dalszą historią algorytmu Deep CFR. Dodatkowo zostanie przedstawiony możliwy kierunek rozwoju uczenia maszynowego w środowiskach częściowo-obserwowalnych.

5.1 Wnioski

Gra Limit Poker Texas Hold'em jest bardzo trudnym środowiskiem do uczenia sztucznych inteligencji, wynika to z częściowej obserwowalności. Z tego powodu model Cepheus, zdol-

ny pokonywać ludzi w takim środowisku powstał dopiero w 2015 roku. Od tamtej pory rozpoczął się nagły rozwój metod uczenia maszynowego coraz większych gier karcianych. Przykładami są DeepStack lub Libratus rozwiązujące grę No Limit Texas Poker Hold'em i pokonujące profesjonalnych ludzi.

Głównym zadaniem niniejszej pracy była próba zmierzenia się z takim środowiskiem. W celu uproszczenia zadania wybrano grę HULH. Następnie zaimplementowano nowoczesny algorytm Deep CFR i wytrenowano pięć modeli rozpoznawania. Pierwszy problem, jaki napotkano to, powolna nauka sieci θ_p, wraz z dużym błędem predykcji. Podczas uczenia wartości nie spadały poniżej 600. Możliwym rozwiązaniem jest zwiększenie parametru *learning rate* oraz dopracowanie architektury sieci neuronowej. Dodatkową przyczyną takich rezultatów mogą być zbiory danych o słabej jakości. Prawdopodobnie nauka by przebiegała lepiej przez zwiększenie zawartości obserwowalnych informacji wejściowych przez np. liczbę żetonów w grze. W taki sposób sieć neuronowa by miała więcej informacji, które by zostały użyte do zwracania właściwego wektora żalu.

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. Wadą takiej architektury jest to, że w praktyce rozgrywki Poker Texas Hold'em mogą odbywać się 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 się gry zależnie od liczby pozostałych żetonów w puli oraz od numeru rundy. Gracze mogą podejmować bardziej ryzykowne i nierozważne ruchy, będąc w stanie bliskim porażki. Takie informacje mogłyby być kluczowe w osiągnięciu zwycięstwa. Prawdopodobnie przy uwzględnieniu tych elementów, utworzone modele lepiej by dobierały strategie do określonych stanów gry. To by wymagało wykonywania eksploracji MCCFR ES na znacznie większych drzewach decyzyjnych obejmujących wiele rund. Dodatkowo dane wejściowe sieci neuronowej byłyby większe. W takim przypadku możliwym zbiorem informacji mógłby być zestaw składający się z widocznych kart, historii z wielu rund, liczby żetonów każdego z graczy oraz numeru gry. W taki sposób model nauczyłby się lepiej dostosowywać sposób gry do obserwacji.

Deep CFR spełnił funkcję i utworzył modele, które wygrywają z prostymi programami symulującymi grę Poker Texas Hold'em jak przedstawiony gracz nieblefujący, HP w rozdziale 3. Pomimo dobrych rezultatów dużym problemem okazał się proces powstawiania sztucznych inteligencji. Pozornie można by było oczekiwać, że algorytm będzie tworzył lepsze AI wraz z dłuższym czasem działania. W pracy doszło do odwrotnej sytuacji. Sztuczne inteligencje utworzone w iteracjach 10 i 40 wygrywały z późniejszymi obiektami. Ciężko określić przyczynę takich rezultatów. Pomocne okazałoby się użycie metryki *Exploiability* do śledzenia postępów AI pomimo zwiększenia wymaganej mocy obliczeniowej przez algorytm. Taki element pozwolił by na stwierdzenie czy algorytm ominął pónkt o najlepszej jakości i w dalszym procesie uzyskuje podobne lub coraz gorze efekty. Pozwoliło by to na zatrzymanie procesu uzyskując najlepszy możliwy model z implementacji.

5.2 Dalszy rozwój algorytmów bazujących na metodzie CFR

Algorytm powstały w 2017 roku dawał dobre rezultaty, ale przez wykorzystanie dwuch sieci neuronowych tworzył duzą wariancję wyników [24]. Z tego powodu powstał jego następca Single Deep CFR. Po wykonanych testach uzyskał nie znacznie lepsze wyniki. Algorym dalej nie był perfekcyjny, z tego powodu w 2020 roku powstała metoda uczenia maszynowego o nazwie DREAM [25]. Są to bardzo dobre sposoby na wtorzenie AI w grach karcianych. Po mimio tego ich wadą jest to, że wymagają od gry aby wartość wygranej i przegranej sumowała się do zero. Taką cechą okresla się środowiska Zero-sum [14]. Przez to algorytmy są niemożliwe do użycia w wielu środowiskach. Kolejnym problemem tych metod jest przeprowadzenie testów schodzenia się do Równowagi Nasha tylko w grach 2-osobowych [5]. Wiele środowisk jak Poker Texas Hold'em standardowo uwzględnia wiekszą ilosć uczestników. Na podstawie takich informacji można stwierdzić, że zimplementowany Deep CFR wraz z jego następcami nie wyczerpały tematu środowisk gier karcianych. Nawet najnowsze AI, które pokonują profesjonalnych graczy jak DeepStack lub Libratus muszą trzymać sie tego warunku [16] [17]. Oznacza to, że takie gry to bardzo trudne środowiska, które prawdopodobnie będą jeszcze długo badane pod względem możliwych rozwiązań.

Bibliografia

[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] Gibney, Elizabeth. "Google AI algorithm masters ancient game of Go." Nature News 529.7587 (2016): 445.

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

[4] Brown, Noam, et al. "Combining deep reinforcement learning and search for imperfect-information games." arXiv preprint arXiv:2007.13544 (2020).

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

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

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

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

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

[10] Bouju, Gaëlle, et al. "Texas hold'em poker: a qualitative analysis of gamblers' perceptions." Journal of Gambling Issues 28 (2013): 1-28.

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

[12] 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.

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

[14] Myerson, Roger B. Game theory. Harvard university press, 2013.

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

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

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

[18] Davis, Trevor, Neil Burch, and Michael Bowling. "Using response functions to measure strategy strength." Twenty-Eighth AAAI Conference on Artificial Intelligence. 2014.

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

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

[21] 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.

[22] Eric Steinberger (2019). PokerRL in https://github.com/TinkeringCode/PokerRL GitHub.GitHub repository.

[23] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020)

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

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

[26] 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.