Politechnika Warszawska

WYDZIAŁ MECHANICZNY
ENERGETYKI I LOTNICTWA

Instytut Techniki Lotniczej i Mechaniki Stosowanej

Praca dyplomowa inżynierska

na kierunku Automatyka i Robotyka
w specjalności Robotyka

Zastosowania metod sztucznej inteligencji w grach z niepełną informacją

Piotr Tarasiewicz
Numer albumu 271278

promotor
Dr inż. Andrzej Kordecki

Warszawa, 2018

Streszczenie

Zastosowania metod sztucznej inteligencji w grach z niepełną informacją

Słowa kluczowe: sztuczna inteligencja, uczenie ze wzmocnieniem, poker, *counterfactual regret minimazation*

Poniższa praca zawiera implementację algorytmów sztucznej inteligencji do gier z niepełną informacją na przykładzie gry pokerowej znanej pod nazwą *5 Card Draw*. Zadaniem stworzonego algorytmu jest nauczenie się strategii gry w pokera jedynie na podstawie znajomości zasad oraz gry z samym sobą z wykorzystaniem uczenia ze wzmocnieniem.

W tym celu został użyty algorytm zaprezentowany przez naukowców z Univesity of Alberta w Kanadzie o nazwie *Counterfactual Regret Minimization* (CFR). Algorytm ten przeprowadza kolejne symulacje rozdań, w każdej symulacji przechodzi przez całe drzewo gry, w którego liściach znajdują się nagrody, czyli wyniki rozdań. Na podstawie tych nagród każdemu zagraniu przypisywana jest wartość żalu, który informuje nas, jaka strata wynika z niepodjęcia tego zagrania. Po każdej symulacji żal ten jest akumulowany, a mieszana strategia jest tworzona wprost proporcjonalnie do nieujemnych wartości tego żalu. Średnia strategia, uzyskana we wszystkich symulacjach, dąży do osiągnięcia stanu równowagi Nasha.

Ze względu jednak na rozmiar problemu, rzędu 10^{26}, nie jest możliwa nauka na pełnym drzewie gry. W celu optymalizacji złożoności zastosowano różne techniki abstrakcji. W pierwszej kolejności utożsamione zostały stany gry, które są sobie równoważne pod względem oczekiwanej nagrody. Konieczne było jednak ograniczenie głębokości przeszukiwania drzewa gry. Ponieważ w każdej sytuacji gracz ma do dyspozycji przynajmniej dwa różne zagrania, drzewo rośnie eksponencjalnie. Ograniczenie przeszukiwania drzewa jedynie do połowy jego głębokości pozwoli zatem na eksponencjalne ograniczenie złożoności problemu i sprawi, że algorytm będzie w stanie nauczyć się strategii korzystając z dostępnej mocy obliczeniowej.

Ograniczenie przeszukiwania głębokości drzewa zostało przeprowadzone w sposób mimikujący myślenie człowieka. Człowiek, z powodu swoich ograniczeń, nie rozważa wszystkich możliwych scenariuszy, aż do końca gry, ale jedynie kilka kolejnych zagrań naprzód (dzieje się tak w pokerze, ale również w grach z doskonałą informacją, jak szachy). Następnie szacowana jest wartość sytuacji, które można osiągnąć i wybierane zagranie, które prowadzi do tej najbardziej profitowej. W celu mimiki tego sposobu myślenia zastosowana została sztuczna sieć neuronowa. Do jej treningu został wykorzystany zestaw losowo wygenerowanych scenariuszy, rozwiązanych za pomocą algorytmu CFR. Następnie w czasie przeszukiwania drzewa, algorytm na odpowiedniej głębokości zaprzestaje dalszego schodzenia w głąb, a posiłkuje się wyjściem sieci neuronowej dla danego scenariusza, w którym się znalazł.

Użycie tych metod pozwoliło programowi nauczyć się strategii i wyraźnie pokonać za jej pomocą prawdziwych graczy na próbie ponad 50 tysięcy rozdań.

Abstract

Applications of artificial intelligence for games with imperfect information

Key words: artificial intelligence, reinforcement learning, poker, counterfactual regret minimization

This paper describes an implementation of AI algorithms in games with imperfect information, specifically in poker game called 5 Card Draw. The goal is to create an algorithm, which will be able to learn a winning strategy from self-play with a usage of reinforcement learning, on the basis of the knowledge of the rules of the game only.

In order to achieve this goal a Counterfactual Regret Minimization algorithm was used, which was originally presented by scientists from the University of Alberta in Canada. This algorithm learns the strategy by simulating poker hands. In each simulation, it traverses a whole game tree that contains the rewards in leaves. Rewards are essentially the results of the hands. With regard to these rewards, each action in the game tree is assigned with a regret value, which informs us how much a player will regret not taking this particular action. After each simulation, the total regret value is being accumulated and the strategy is updated proportionally to the nonnegative regret values. An average strategy obtained in all the simulations converges to the Nash equilibrium.

Due to the size of the problem, of an order of magnitude 10^{26}, it is impossible to use the complete game tree when solving the game. Different techniques of optimization and problem reduction are presented in this paper. The first abstraction that is used is combining game states, which are identical in terms of the expected reward. However, it was necessary to limit the depth, to which the game tree is being traversed. Given the fact that in every situation the player has at least two possible actions to choose from, the tree grows exponentially. Limiting the traversing to a half of the depth only will allow reducing the complexity of the problem also exponentially. It will result in the algorithm, which can be trained with the use of available resources.

The way in which the game tree traversing depth is being limited mimics the way people approach playing games. A human player, because of his own restrictions, does not analyze all possible scenarios, until the end of the game. Instead, he tries to think a couple of moves ahead and predict in what kind of a situation he will find himself in (similarly to a chess game). He then estimates the value of each situation and chooses the action, that allows him to reach the one with the highest expected reward. The way for mimicking human thinking process was the use of an artificial neural network. To create a training set a random set of scenarios solved explicitly by CFR was used. When the algorithm is traversing the tree at a certain depth level it replaces an expected reward that would come from the subtree, by the value obtained from neural network for the particular situation current node is corresponding to.

The use of methods described above allows the algorithm to learn a winning strategy and beat real players by a large margin on a sample of more than 50 thousand hands.

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że przedstawiona praca dyplomowa:

- została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami,
- nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego lub stopnia naukowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

....

data podpis autora (autorów) pracy

Oświadczenie

Wyrażam zgodę / nie wyrażam zgody*1 na udostępnianie osobom zainteresowanym mojej pracy dyplomowej. Praca może być udostępniana w pomieszczeniach biblioteki wydziałowej. Zgoda na udostępnienie pracy dyplomowej nie oznacza wyrażenia zgody na jej kopiowanie w całości lub w części.

Brak zgody nie oznacza ograniczenia dostępu do pracy dyplomowej osób:

- reprezentujących władze Politechniki Warszawskiej,

- członków Komisji Akredytacyjnych,

- funkcjonariuszy służb państwowych i innych osób uprawnionych, na mocy odpowiednich przepisów prawnych obowiązujących na terenie Rzeczypospolitej Polskiej, do swobodnego dostępu do materiałów chronionych międzynarodowymi przepisami o prawach autorskich. Brak zgody nie wyklucza także kontroli tekstu pracy dyplomowej w systemie antyplagiatowym.

....

data podpis autora (autorów) pracy

*1 - niepotrzebne skreślić

Spis treści

1. Wprowadzenie

1.1. Cel pracy

Celem niniejszej pracy było zaprojektowanie i stworzenie sztucznej inteligencji do gier o niepełnej informacji, która będzie w stanie wygrywać z ludźmi jedynie poprzez naukę w trakcie gry samej ze sobą (ang. *self-play*). Przykładem gry o niepełnej informacji użytej w tej pracy będzie bardzo popularna w Polsce odmiana pięciokartowego pokera dobieranego (ang. *5 Card Draw*), znana z takich filmów jak „Wielki Szu" czy „Żądło" oraz pewne jej uproszczenia.

Praca zawiera w pierwszej kolejności omówienie obecnego stanu rozwoju sztucznej inteligencji w grach oraz definicji gier o niedoskonałej informacji. Omówione zostaną również problemy, które można za pomocą takich gier modelować.

W rozdziale 2 znajduje się definicja gier pokerowych, zasad gry oraz używanej w pracy terminologii. Omówiona zostanie w tym rozdziale również złożoność pokera oraz postępy w jego rozwiązywaniu na tle innych znanych gier takich jak szachy, czy go. Na końcu zostanie przedstawiona matematyczna reprezentacja gry, która będzie używana w programie.

Rozdział 3 zawiera omówienie równowagi Nasha oraz podstawowego algorytmu użytego w programie jakim jest *Counterfactual Regret Minimazation* (CFR). Po teoretycznym przedstawieniu algorytmu zostanie on zaimplementowany na przykładzie bardzo prostej gry.

Rozdział 4 zawiera już implementację algorytmu do pełnej gry pięciokartowego pokera dobieranego z bardzo niewielkimi uproszczeniami oraz użytej przez program – przy pomocy między innymi sieci neuronowych – abstrakcji w celu ograniczenia rozmiarów problemu.

Wreszcie w rozdziale 5 znajdują się wyniki programu w grze przeciwko ludziom oraz przeciwko wyznaczonej przez ten sam algorytm optymalnej kontrstrategii, co pozwoli na oszacowanie odległości rozwiązania od równowagi Nasha.

W całej pracy będą zamieszczane fragmenty kodu programu, a cały kod jest dostępny w załączniku jak również w publicznym repozytorium.

1.2. Gry z niepełną informacją

Jednym z podstawowych podziałów gier jest podział na gry z pełną (doskonałą) informacją (ang. *perfect information*) i z niepełną informacją (ang. *imperfect information*) [1]. W literaturze anglojęzycznej wyraźnie zarysowany jest podział na *perfect* i *complete information*. W polskojęzycznej literaturze określenia doskonała oraz pełna informacją są używane zamiennie ze zdecydowaną przewagą tej drugiej, stąd też taka nazwa w tytule poniższej pracy.

W grach z pełną informacją (na przykład w szachach lub go), każdy z graczy w każdym momencie gry wie o tej grze wszystko, jakie są możliwe ruchy, jaka jest aktualna sytuacja, wszystkie przeszłe decyzje przeciwnika, itd. W grach z niepełną informacją natomiast (np. poker, brydż) część informacji jest przed każdym z graczy ukryta, na przykład jakie karty ma przeciwnik, jaka pojawi się następna karta.

Jednym z powodów, dla których gry często stają się tematem rozważań w badaniach naukowych, jest to, że można za ich pomocą modelować wiele zagadnień z prawdziwego życia. Najczęściej analizowanym zagadnieniem są prawa rządzące ekonomią [2], gdzie na przykład nie wiadomo jaką wartość przedstawia dany produkt dla różnych stron. Innym przykładem może być opieka medyczna i ubezpieczenia [3], gdzie pacjenci zwykle nie znają rzeczywistego kosztu usług, które otrzymują, a ubezpieczyciele nie mają wszystkich informacji o stanie zdrowia pacjentów. Dalsze przykłady można wskazywać w polityce, projektowaniu i w wielu innych dziedzinach. Tak naprawdę większość sytuacji z prawdziwego życia, które można modelować za pomocą gier, modeluje się jako gry z niepełną informacją.

W przypadku pokera mamy dodatkowo do czynienia z grą o sumie zerowej, co oznacza, że sumy nagród graczy, inaczej wygranych (przegrane interpretujemy jako ujemną wygraną), są zawsze równe 0. W grze dla dwóch graczy oznacza to, że jeżeli jeden z graczy wygrał 10, to drugi musiał przegrać 10 (wygrać -10).

1.3. Sztuczna inteligencja w grach

Bardzo ciężko określić początki sztucznej inteligencji w grach, czy też sztucznej inteligencji w ogóle, gdyż sam ten termin przysparza wiele trudności w dokładnym zdefiniowaniu terminu użytego po raz pierwszy w 1955 roku przez grupę amerykańskich naukowców [4]. Encyklopedia Brittanica definiuje sztuczną inteligencję jako „zdolność cyfrowego komputera lub kontrolowanego przez komputer robota do wykonywania zadań powszechnie wiązanych z inteligentnymi istotami" [5]. Jest to jedna z bardzo wielu niejasnych definicji.

Zastanówmy się nad przykładem gry w szachy. Zdecydowanie bardzo wysokie umiejętności gry w szachy są powszechnie kojarzone z inteligencją. Można sobie wyobrazić, że w przyszłości będzie istniał komputer, który będzie przechowywał w pamięci wszystkie możliwe stany gry w szachy oraz przypisany do każdego stanu ruch optymalny. Komputer taki, będzie grał w szachy idealnie. Czy jednak jest inteligentny program, który jedynie wykonuje proste instrukcje postaci: „w sytuacji A wykonaj ruch B"?

Z drugiej strony można myśleć o sztucznej inteligencji w kontekście algorytmów naśladujących działania ludzkiego mózgu. Są to jednak zwykle jedynie analogie, a za algorytmami stoją w rzeczywistości narzędzia statystyczne używane długo przed tym, zanim zbudowano na przykład sztuczne sieci neuronowe.

Można również wysuwać argumenty, że przecież i za ludzką inteligencją nie stoi żadna nadprzyrodzona siła i wszystko można sprowadzić jedynie do procesów fizycznych i chemicznych. Nie chcąc za długo zatrzymywać się nad samym zagadnieniem definicji pozostaniemy przy najbardziej powszechnej i intuicyjnej, pierwszej definicji z Encyklopedii Brittanica.

W jej świetle można wyróżnić pewne kamienie milowe sztucznej inteligencji w grach, chociaż wszystkie te programy używały bardzo różnych technik, od bardzo prostych algorytmów MiniMax używanych przez Deep Blue, przez algorytmy takie jak CFR, który został użyty w tej pracy, aż po głębokie sieci neuronowe używane przy projekcie AlphaGo. Oto zestaw tych najbardziej przełomowych momentów [5] [6]:

- 1994 – program o nazwie Chinook zdobywa tytuł mistrza świata w Warcaby,
- 1997 – Deep Blue pokonuje mistrza świata w szachach Garija Kasparova,
- 2011 – komputer o nazwie Watson wygrywa teleturniej *Jeopardy!,*
- 2016 – projekt Google'a o nazwie AlphaGo pokonuje mistrza świata Go Lee Sedol,
- 2017 – program DeepStack wygrywa z profesjonalnymi pokerzystami w grze *Heads Up No Limit Holdem,*
- 2017 – program stworzony przez Start Up należący do Elona Muska o nazwie OpenAI wygrywa z mistrzami świata w grę komputerową *Dota 2.*

Jak widać na powyższym zestawieniu, sztuczna inteligencja w ostatnich latach nabiera niesamowitego rozpędu i szturmuje coraz to kolejne bastiony ludzkiego umysłu. W tej niezwykłej rywalizacji projekt lowbotCFR stworzony w ramach poniższej pracy inżynierskiej ma być jedną z potyczek wygranych przez maszyny.

Zastosowania sztucznej inteligencji w grach nie są tylko wyzwaniem naukowym. Po pierwsze stymulują rozwój samych algorytmów, które są następnie używane w dużo szerszym zakresie, jak zostało wspomniane w poprzednim podrozdziale. Można porównać to do sytuacji zimnej wojny, w czasie której wyścig zbrojeń lub wyścig kosmiczny walnie przyczyniły się do rozwoju technologii cywilnych. Dużo bezpośrednim zastosowaniem są roboty służące do gry przeciwko ludziom. Maszyny służące do gry w proste odmiany pokera stoją już w coraz większej liczbie kasyn, zarabiając na ludziach, którzy starają się pokonać tajemniczą inteligencję.

2. Poker

2.1. Zasady gry

Różnych odmian gier pokerowych jest obecnie kilkadziesiąt, a wciąż powstają nowe, niekiedy wymyślane na potrzeby jednej tylko sesji. Zastosowane w tej pracy rozwiązania bardzo łatwo mogą zostać użyte w prawie każdej odmianie pokera. W tej pracy omawiać będziemy jedynie grę *5 Card Draw* w wersji *Heads Up, Pot Limit* oraz jej uproszczenia.

Określenie *Heads Up* oznacza, że w rozgrywce bierze udział jedynie dwóch graczy. W *5 Card Draw* gracze na początku każdego rozdania muszą wpłacić ciemne: małą w ciemno oraz dużą w ciemno. Ich wysokości ustalane są przed grą, duża w ciemno jest dwa razy wyższa niż mała w ciemno. Małą w ciemno oraz pozycję gracza, który ją wpłaca określa się jako SB (ang. *small blind*), natomiast dużą w ciemno i pozycję gracza, który ją wpłaca jako BB (ang. *big blind)*. Po każdym rozdaniu gracze zamieniają się pozycjami i SB staje się BB i odwrotnie.

Na początku rozdania po wpłaceniu ciemnych gracze otrzymują po pięć kart i następuje pierwsza tura licytacji. Po licytacji gracze kolejno wymieniają dowolną liczbę kart i następuję druga runda. Po jej zakończeniu gracze pokazują swoje karty i najlepsza ręka wygrywa całą pulę. W przypadku remisu gracze dzielą się pulą po połowie. Pierwszą turę licytacji rozpoczyna gracz na pozycji SB, następne etapy (wymianę kart i drugą licytację) rozpoczyna BB.

Podczas licytacji gracze mają do dyspozycji w swojej kolejce następujące akcje:

1. Czekanie – jeśli zakład nie został przebity, można czekać co przekazuję akcję dalej.
2. Pas – poddanie puli.
3. Sprawdzenie – wyrównanie zakładu przeciwnika.
4. Podbicie – przebicie zakładu przeciwnika.

Licytacja kończy się w momencie, gdy jeden z graczy spasował lub gdy każdy z graczy wykonał przynajmniej jedną akcję i obaj postawili tyle samo.

Określenie *Pot Limit* oznacza, że nie można przebić zakładu przeciwnika o więcej niż znajduje się w puli.

Bardzo ważne jest zrozumienie w jaki sposób liczona jest pula, przy obliczaniu maksymalnej wysokości podbicia. Niech to co znajduje się w puli przed licytacją (na przykład z poprzedniej tury) będzie oznaczone jako P_0, a to ile postawili gracze odpowiednio P_1 i P_2. Załóżmy, że $P_2 \geq P_1$ i pierwszy gracz ma wykonać teraz akcję.

Na początek należy zrozumieć różnicę pomiędzy *podbiciem o*, a *podbiciem do*. Gracz pierwszy *podbijając o* A gracza drugiego *podbija do* wysokości $P_2 + A$.

Obliczając pulę przy podbijaniu, najpierw myślowo gracz wyrównuje zakład przeciwnika i dopiero następnie zlicza całą pulę, oznaczmy ją jako $\widehat{Pula}$:

$$\widehat{Pula} = P_0 + 2P_2 \tag{1}$$

Zatem *podbijając o* całą pulę nowa wartość P_1 wynosi:

$$P_1 := P_2 + \boldsymbol{\widehat{Pula}} = P_0 + 3P_2 \tag{2}$$

Podbijając o na przykład pół puli otrzymujemy:

$$P_1 := P_2 + \frac{1}{2}\widehat{Pula} = \frac{1}{2}P_0 + 2P_2 \tag{3}$$

Dodatkowo na potrzeby pracy zostały przyjęte następujące zasady:

1. Gracz może przebijać jedynie o wartość puli w pierwszej licytacji oraz o wartość puli lub jej połowę w drugiej rundzie licytacji.
2. Podczas jednej tury licytacji mogą wystąpić maksymalnie trzy przebicia. Postawienie dużej w ciemno liczy się jako pierwsze przebicie.

Na końcu rozdania, jeśli żaden z graczy wcześniej nie spasował, obaj gracze pokazują karty i wygrywa najlepszy układ. Starszeństwo układów jest następujące:

1. Poker Królewski (kolor i strit do Asa jednocześnie, np. A♣ K♣ Q♣ J♣ T♣).
2. Poker (kolor i strit jednocześnie, np. K♣ Q♣ J♣ T♣ 9♣).
3. Kareta (cztery karty o jednakowej wartości, np. K♣ K♥ K♦ K♠ 7♠).
4. Ful (trzy i dwie karty jednakowych wartości, np. A♣ A♦ A♠ K♦ K♥, rycina 1).
5. Kolor (5 kart w jednym kolorze, np. A♣ Q♣ 9♣ 6♣ 4♣).
6. Strit (5 kart o kolejnych wartościach, np. K♣ Q♥ J♣ T♦ 9♣).
7. Trójka (3 karty o jednakowej wartości, np. K♣ K♥ K♠ T♠ 7♥).
8. Dwie pary (dwie pary kart tej samej wartości, np. K♣ K♥ T♠ T♥ 3♠).
9. Para (dwie karty tej samej wartości, np. K♣ K♥ T♠ 7♥ 3♠).
10. Wysoka karta (np. K♣ J♥ T♠ 7♥ 3♠).

Gdy gracze mają ten sam układ porównywane jest starszeństwo kart, na przykład kolor z asem jest mocniejszy od koloru z królem, strit do króla od strita do 9, para asów od pary dam, itd.

Rycina 1. Ful, asy na królach.

Przykładowe rozdanie:

- SB wpłaca małą w ciemno o wartości 0,5 (pula 0,5),
- BB wpłaca dużą w ciemno o wartości 1 (pula 1,5),
- SB otrzymuje rękę A♣ Q♣ T♣ J♣ 2♥,
- BB otrzymuje rękę K♣ K♥ T♠ T♥ 7♠,
- SB podbija do 3 (pula 4),
- BB podbija do 9 (pula 12),
- SB sprawdza (pula 18),
- BB wymienia jedną kartę 7♠ i otrzymuje T♦,
- SB wymienia jedną kartę 2♥ i otrzymuje 8♣,
- BB podbija o pulę do 18 (pula 36),
- SB podbija o pół puli do 45 (pula 81),
- BB podbija o pół puli do 99 (pula 162),
- SB sprawdza (pula 216),
- BB pokazuje fula i wygrywa z kolorem SB wygrywając 108 od SB.

2.2. Stan wiedzy

Teoria gry w pokera przeszła bardzo długą drogę od pierwszej książki legendy pokera Doyle Brunsona *Super System* z 1978. Po dziś dzień nazywana jest *Biblią pokera*. Jednak obecnie każdy gracz, który chce zostać wygrywającym graczem nawet po nią nie sięga. W zamian za to dołącza do internetowych platform treningowych, gdzie najlepsi gracze oferują kursy, których ceny wahają się od kilkudziesięciu do kilku tysięcy dolarów w zależności od klasy gracza, który je oferuje i stopnia zaawansowania przekazywanej wiedzy. Dodatkowo gracze inwestują pieniądze w oprogramowanie wspomagające zarówno grę przez Internet, jak i pracę nad swoją strategią.

Sposób przekazywania wiedzy nie jest główną zmianą, jaka zaszła w ciągu ostatnich lat. Kiedyś można było usłyszeć powiedzenie, że „nie gra się kartami, tylko przeciwnikiem". Oznaczało to, że ważniejsze od tego, jakie miało się karty było to, żeby domyślić się co ma przeciwnik (czy blefuje czy nie, jak mocny ma układ, itd.). Obecnie dzięki rozwojowi teorii gier gracze najpierw grają kartami, lecz już nie tylko swoimi, ale wszystkimi, które mogliby w niej mieć i tymi, które mógłby mieć przeciwnik. Starają się podejmować decyzję bazując nie na mierzeniu wzrokiem przeciwnika, ale na teorii gry optymalnej, GTO (ang. *game theory optimal*). Chcą, żeby ich zagrania były poprawne w matematycznym sensie, niezależnie od przeciwnika. Szczegółowo zostało to omówione w podrozdziale 3.1.

Podstawowym terminem używanym do szukania tych optymalnych rozwiązań jest *balans*. Każde zagranie musi być zbalansowane. Oznacza to, że należy tak tworzyć swoją strategię, żeby w każdej sytuacji mieć zarówno dobre ręce, z którymi chcemy wygrać dochodząc do końca rozdania i pokazania kart, jak i złe ręce, z którymi chcemy wygrać poprzez wypchnięcie przeciwnika z rozdania i zmuszenia go do spasowania lub poprzez polepszenie naszej ręki.

Można z tego wyprowadzić bardzo prymitywną, ale w ogólności słuszną zasadę:

- jesteśmy agresywni (podbijamy) z najlepszymi i najgorszymi rękami (dla blefu),
- jesteśmy pasywni (sprawdzamy, czekamy) ze średnimi rękami i z częścią najlepszych, które służą do zastawiania pułapki na przeciwnika,
- poddajemy pule z najgorszymi rękami.

Można dostrzec w tym wspomniany wcześniej balans. Na przykład, gdy jesteśmy agresywni, przeciwnik nie może zawsze poddawać puli, ponieważ czasem blefujemy z bardzo słabą ręką. Nie może też zawsze sprawdzać, ponieważ zwykle mamy bardzo silną rękę.

Gdy jesteśmy pasywni przeciwnik nie może zawsze nas atakować, żebyśmy spasowali, ponieważ czasem łapiemy go w pułapkę z bardzo silną ręką, ale nie może też zawsze sam być pasywny, bo pozwala nam wtedy w pełni zrealizować wartość naszych średnich rąk.

Tak w bardzo dużym uproszczeniu można przedstawić obecne trendy i stan wiedzy w pokerze. Oczywiście nie da się tak szybko nawet wyjaśnić podstaw strategii, które są zawarte w setkach i tysiącach godzin filmów szkoleniowych, ale ten wstęp pozwala na ogólny pogląd na strategie pokerowe i ich istotę – balans.

Jeśli chodzi o rozwój algorytmów obliczeniowych w pokerze, to ostatnie lata przyniosły bardzo duży postęp. Wpływ na to miały dwa podstawowe czynniki. Po pierwsze, po pokonaniu człowieka przez AI w większości gier z pełną informacją, gry pokerowe stały się często używanym polem doświadczalnym nowych algorytmów sztucznej inteligencji, a wyniki przeciwko ludzkim graczom są dobrym miernikiem postępów. Ten powód przyświecał również tworzeniu lowbotCFR. Drugim czynnikiem jest popyt na oprogramowania do wyznaczania optymalnej strategii. Poker jest grą nieodłącznie związaną z pieniędzmi, w której najlepsi gracze wygrywają dziesiątki milionów dolarów rocznie. W związku z tym niezwykle cenna jest również wiedza na temat optymalnych strategii. Przez to oprogramowania takie jak PIO Solver kosztują setki dolarów za pojedyncze licencje [7].

Wśród tworzonych rozwiązań aproksymujących strategie spełniające równowagę Nasha, dominują dwie metody. Pierwszą z nich jest używany również w lowbotCFR algorytm *Counterfactual Regret Minimization*. Został on użyty przez słynne programy DeepStack [8] oraz Libratus [9] służące do gry w *No Limit Hold'em*, oczywiście w wersji *Heads Up*. Drugą popularną metodą jest tzw. *Fictitious Play* [10], użyta na przykład w połączeniu z głębokim uczeniem w programie NFSP, stworzonym na londyńskim UCL [11]. Innym podejściem jest również stosowanie głębokiego uczenia ze wzmocnieniem jedynie do nauki gry przeciw danemu przeciwnikowi. Rozwiązania te nie pozwalają jednak dojść do równowagi Nasha, stąd nie są popularne wśród badaczy.

Prostsze metody rozwiązywania pokera, takie jak poprzez programowanie liniowe [12] pozwalają jedynie na osiąganie dobrych rezultatów bardzo ograniczonych odmianach pokera, takich jak *Khun Poker* czy *Leduc Holdem*. Nie są one w stanie poradzić sobie ze złożonością pełnych gier.

2.3. Reprezentacja matematyczna

Przed przystąpieniem do implementacji algorytmów, należy stworzyć matematyczną reprezentację gry, która będzie w tych algorytmach używana. W tym celu definiujemy następujące pojęcia i oznaczenia wraz z przykładami z gry pokerowej:

- Historia - ζ, czyli publiczne informacje.

  Historia gry zawiera wszystkie informacje publiczne, czyli takie, które są dostępne widzowi, który śledzi rozdanie z boku. W każdym momencie jest reprezentowana jako ciąg znaków (string). Zawiera on kolejne akcje graczy oznaczone w następujący sposób:

  Symbol „p” oznacza podbicie o całą pulę (ang. *pot size raise*), „r” oznacza podbicie o pół puli (ang. *raise*), „c” oznacza czekanie lub sprawdzenie (ang. *check/call*), a „f” pas (ang. *fold*). „-” oznacza stan, w którym żaden z graczy nie wykonał jeszcze akcji.

  Akcja wymiany kart jest w tym stringu oznaczona jako para liczb (*ab*), które oznaczają odpowiednio, ile każdy z graczy wymienił kart.

  Przykładowa historia: $\zeta =$ ”pppc(01)cprp”. Możemy z niej odtworzyć pełną historię rozdania:

  - SB wpłaca małą w ciemno o wartości 0.5 (pula 0.5) (niezawarte w historii, ale zawsze następuję ten moment),
  - BB wpłaca dużą w ciemno o wartości 1 (pula 1.5) (pierwsze „r” w historii),
  - SB podbija do 3 (pula 4),
  - BB podbija do 9 (pula 12),
  - SB sprawdza (pula 18),
  - BB nie wymienia żadnej karty,
  - SB wymienia jedną kartę,
  - BB czeka,
  - SB podbija o pulę do 18 (pula 36),
  - BB podbija o pół puli do 45 (pula 81),
  - SB podbija o pulę do 153 (pula 216).

  Tak reprezentowana historia rozdań po pierwsze jest bardzo prosta i zajmuje jak najmniej miejsca, a z drugiej strony, w każdym momencie, możemy z niej odtworzyć wszelkie potrzebne informacje, ile znajduje się pieniędzy w puli (216), który gracz podejmuje decyzję (BB), itd.

- Prywatne informacje gracza i - η_i.

  Prywatne informacje każdego z graczy zawierają wszystkie informacje, które są dostępne tylko jemu. Są to zatem jego karty, to które karty wymienił i jakie otrzymał na ich miejsce. Ponieważ wszystkie karty przed i po wymianie jednoznacznie określają, które karty zostały wymienione. Prywatne informacje będą przechowywane również w postaci stringa, zawierającego całą rękę przed

i – jeśli wymiana już nastąpiła – po wymianie. Na przykład $\eta_1 =$ „QcTs7s6h5h” oznacza, że gracz pierwszy otrzymał rękę Q♣ T♠ 7♠ 6♥ 5♥. $\eta_2 =$ „AsTs7s5s5dAsTs7s5s2h” oznacza, że drugi gracz otrzymał rękę A♠ T♠ 7♠ 5♠ 5♦, wymienił jedną kartę – 5♦ i otrzymał w zamian 2♥.

- Stan gry - g.

  Stan gry zawiera wszystkie dostępne informacje na temat rozdania w danym momencie, można zatem zapisać $g = \{\zeta, \eta_1, \eta_2\}$.

- Zbiór informacji – I.

  Zbiór informacji przy danej historii ζ zawiera dotychczasową historię rozdania oraz prywatne informacje gracza, który podejmuje decyzję. Można zatem zapisać $I = \{\zeta, \eta_i\}$, gdzie $i \in \{1, 2\}$ oznacza gracza podejmującego decyzję przy historii ζ. Jak łatwo zauważyć, każdemu zbiorowi informacji I może odpowiadać więcej niż jeden stan gry g, natomiast każdemu stanowi gry g odpowiada dokładnie jeden zbiór informacji I. Niech I_i oznacza taki zbiór informacji, dla którego następną decyzję podejmuje gracz i. $\{I_i\}$ oznacza rodzinę wszystkich możliwych zbiorów informacji I_i.

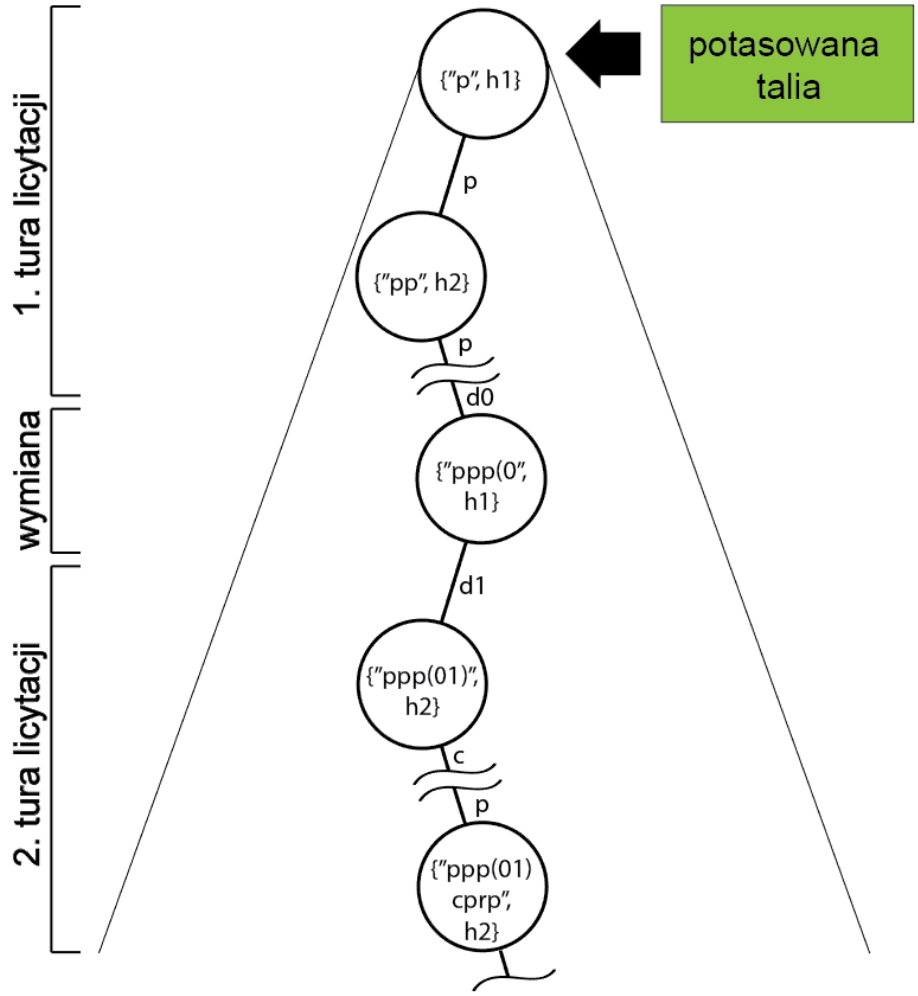
- Zbiór akcji – A, akcja – $a \in A$.

  Zbiór akcji oznacza wszystkie dostępne akcje graczy w ciągu całej gry. W przypadku rozpatrywanej gry $A = \{\mathrm{r}, \mathrm{p}, \mathrm{f}, \mathrm{d1}, \ldots, \mathrm{d5}\}$, gdzie „d$i$” oznacza wymianę i kart. Niech $A(I)$ oznacza zbiór wszystkich legalnych akcji dla zbioru informacji I.

- Strategia - $\sigma_i: \{I_i \times A(I_i)\} \rightarrow [0,1]$.

  Strategia σ_i, dla gracza i przyporządkowuje wszystkim zbiorom informacji I_i oraz wszystkim odpowiadającym im legalnym akcjom $a \in A(I_i)$ prawdopodobieństwo z jakim gracz i wybiera akcję a przy I_i. Wszystkie strategie określane jako *profil strategii* i oznaczane jako σ. Jako $\sigma_{I \rightarrow a}$ oznaczamy profil odpowiadający σ, jednak dla zbioru informacji I zawsze jest wybierana akcja a.

Każde rozdanie jest reprezentowane jako drzewo, którego każdy wierzchołek odpowiada pewnemu zbiorowi informacji I, a każda krawędź odpowiada akcji prowadzącej z jednego zbioru informacji do kolejnego. Na rycinie można zobaczyć przykładowy fragment drzewa (rycina 2). Korzeniem drzewa jest początek rozdania.



Rycina 2. Fragment drzewa gry.

Do przykładu posłużyło to samo rozdanie, które zostało opisane wcześniej. Na początku tworzona jest potasowana talia, która będzie używana w tej grze. W korzeniu znajduje się zbiór informacji $I = \{"p", \eta_1\}$, oznacza to, że dotychczasowa historia wynosiła „p”, czyli wpłacenie BB, a zatem obecnie jest kolej gracza 1, czyli SB, którego prywatne informacje znajdują się w I. Podejmuje on akcje p i w następnym węźle historia jest uaktualniona, a w zbiorze informacji znajdują się prywatne informacje drugiego gracza, itd.

2.4. Złożoność

Przed przejściem do rozważań na temat strategii należy określić rozmiar problemu [13].

Na początek obliczamy wszystkie możliwe scenariusze akcji stawiania zakładów w pierwszej i drugiej turze. Na wstępie wyróżnimy różne stany licytacji, które są fragmentami historii rozdania, zawierające jedynie informacje na temat trwającej rundy licytacji:

- stany pośrednie - stany, w których dana runda licytacji jeszcze się nie zakończyła,
- stany końcowe - zakończyło się całe rozdanie,
- stany kontynuacji - zakończyła się jedynie pierwsza licytacja, ale rozdanie trwa nadal (wymiana kart, druga tura licytacji).

W tabelach (tabela 1, 2) zostały wypisane możliwe stany w obu rundach licytacji oraz liczba wszystkich możliwych stanów każdego rodzaju. W drugiej turze każde podbicie może oznaczać podbicie o połowę lub o całą pulę (patrz rozdz. 1.1.). Sprawia to, że znacznie zwiększa się liczba możliwych stanów licytacji, żeby nie wypisywać ich wszystkich, wprowadzone zostało dodatkowe oznaczenie „x", które może przyjmować wartości ze zbioru {„r", „p"}.

Tabela 1. Stany w pierwszej licytacji.

Stany pośrednie	Stany końcowe	Stany kontynuacji
p, pc, pcp, pcpp, pp, ppp	pf, pcf, pcpf, pcppf, ppf, pppf	pcc, pcpc, pcppc, ppc, pppc
Liczba stanów: 6	Liczba stanów:6	Liczba stanów:5

Tabela 2. Stany w drugiej licytacji.

Stany pośrednie	Stany końcowe
-, c, x, cx, xx, cxx, xxx, cxxx	f, cf, cc, xcf, xc, cxf, cxc, xxf, xxc, cxxf, cxxc, xxxf, xxxc, cxxxf, cxxxc
Liczba stanów: 30	Liczba stanów: 59

Należy również przypomnieć, że otrzymując *k* kart z *n*-kartowej talii, możemy je otrzymać na $\binom{n}{k}$ sposobów.

Można teraz przejść do obliczenia dwóch charakterystycznych liczb określających rozmiar gry.

Niech $|\{I_i\}|$ oznacza liczbę wszystkich możliwych zbiorów informacji gry z punktu widzenia jednego gracza, jest to zatem liczba różnych scenariuszy, w których gracz musi podjąć decyzję.

$$|\{I_i\}| = \begin{pmatrix} liczba \\ możliwych \\ rąk \end{pmatrix}\left[\begin{pmatrix} liczba\ pośrednich \\ stanów\ pierwszej \\ licytacji \end{pmatrix} + \begin{pmatrix} liczba\ stanów \\ kontynuacji\ w \\ 1.licytacji \end{pmatrix} \times \begin{pmatrix} liczba \\ możliwych\ rąk \\ po\ wymianie\ kart \end{pmatrix} \times \begin{pmatrix} liczba \\ możliwych \\ wymian\ kart \\ przez\ przeciwnika \end{pmatrix} \times \begin{pmatrix} liczba\ stanów \\ pośrednich\ w \\ drugiej\ licytacji \end{pmatrix}\right] \tag{4}$$

$$|\{I_i\}| = \binom{52}{5} \times \left[6 + 6 \times \sum_{k=0}^{5} \binom{5}{k}\binom{47}{k} \times 6 \times 30\right] \cong 6.1 \times 10^{15} \tag{5}$$

Składnik sumujący pojawia się z powodu konieczności rozpatrzenia wszystkich możliwości wymiany kart przez gracza.

Ponieważ przy każdej decyzji mamy co najmniej dwie akcje do wyboru, musimy zatem przetrzymywać co najmniej jedną liczbę zmiennoprzecinkową typu *double*, która zajmuje 8 bajtów pamięci. Oznacza to, że do przetrzymywania pełnej strategii potrzeba co najmniej 48 000 terabajtów RAM.

Niech $|\{g\}|$ oznacza liczbę wszystkich możliwych stanów gry, a zatem wszystkich możliwych przebiegów rozdania. Analogicznie do $|\{I_i\}|$, tworzymy wzór na $|\{g\}|$, z tą uwagą, że oprócz stanów pośrednich dodajemy do ich liczby również stany końcowe, oraz obliczamy liczbę możliwych rąk obydwu graczy, a nie tylko jednego, jak to miało miejsce poprzednio (stąd dwa składniki sumujące).

$$|\{g\}| = \binom{52}{5}\binom{47}{5}\left[12 + 5\sum_{k=0}^{5}\binom{5}{k}\binom{42}{k} \times \sum_{j=0}^{5}\binom{5}{j}\binom{42-j}{k}89\right] \cong 6{,}6 \times 10^{26} \tag{6}$$

Oznacza to, że w celu nauki gry należałoby rozpatrywać 660 kwadrylionów możliwych scenariuszy rozdań.

Widać zatem, że rozmiar problemu zdecydowanie przerasta możliwości współczesnych komputerów i konieczne jest wprowadzenie abstrakcji w opisie stanów gry i rozdań, co zostało opisane w Rozdziale 4.

3. Counterfactual Regret Minimization

3.1. Równowaga Nasha

W celu zrozumienia co oznacza optymalna strategia, której ma się nauczyć algorytm, należy najpierw omówić pojęcie równowagi Nasha [14] (ang. *Nash Equilibrum*). Najkrócej równowagę Nasha można opisać jako taki profil strategii graczy σ^{eq}, w którym żaden z graczy nie może zwiększyć oczekiwanej nagrody poprzez odstąpienie od swojej strategii. W grze może istnieć wiele różnych punktów równowagi Nasha. Może również zdarzyć się, że w równowadze Nasha obaj gracze mogą przegrywać, choć przy innym doborze strategii obaj by zyskali. Sztandarowym przykładem takiej sytuacji jest znany dylemat więźnia [15].

Przypadek pokera *Heads Up* jest szczególny, ponieważ jest to dwuosobowa gra o sumie zerowej. W takiej grze jeden z graczy traci wtedy i tylko wtedy, gdy drugi z graczy zyskuje. W świetle równowagi Nasha oznacza to, że w punkcie równowagi każde odstąpienie od strategii jednego z graczy oznacza nie tylko zmniejszenie oczekiwanej nagrody przez tego gracza, ale również zwiększenie nagrody przez przeciwnika. Z tego powodu strategia, która spełnia równowagę Nasha nazywa się w pokerze strategią optymalną lub też strategią GTO (ang. *Game Theory Optimal*). Granie zgodnie ze strategią GTO gwarantuje, że nie będzie się przegrywać (w sensie wartości oczekiwanej), a remisować się będzie wtedy i tylko wtedy, gdy przeciwnik również będzie grał optymalnie.

Należy również zauważyć, że nie oznacza to, że granie według GTO zawsze będzie najbardziej profitową strategią. Może się zdarzyć, że jeśli przeciwnik odchodzi od optymalnej strategii można jeszcze bardziej zwiększyć swoją oczekiwaną nagrodę poprzez wykorzystywanie jego błędów samemu odstępując od GTO. Jednak ponieważ strategia przeciwnika nie jest znana, konieczne jest czynienie bardzo wielu założeń, co wiąże się z ryzykiem błędów i zmniejszenia swojej oczekiwanej nagrody. W tej pracy skupiono się tylko na poszukiwaniu strategii optymalnej.

Ponieważ w grach z niedoskonałą informacją nie są znane wszystkie czynniki, a jedynie można estymować rozkład prawdopodobieństwa nad możliwymi scenariuszami, strategia optymalna jest strategią mieszaną i nie jest strategią czystą [16].

Czysta strategia to taka, w której gracz podejmuję daną akcję z prawdopodobieństwem równym 1. W grach z doskonałą informacją strategia optymalna jest strategią czystą. Na przykład, jeśli w szachach jeden ruch daje największą szansę na wygranie, to należy go wykonać w 100% przypadków. W pokerze natomiast, podobnie jak w innych grach z niedoskonałą informacją, optymalna strategia nie jest czystą strategią i w danej sytuacji gra zgodnie z GTO może wymagać spasowania w 23%, sprawdzenia w 5% oraz podbicia w 72%. Jest to jeden z głównych powodów, dla których granie idealnie jest praktycznie niemożliwe dla człowieka. W rzeczywistości gracze starają się szacować, że w danej sytuacji na przykład zwykle powinni podbić, a czasem spasować

i losują liczbę z zakresu od 1 do 100. Jeśli wylosują mniejszą niż 30, to pasują, w innym wypadku podbijają.

To właśnie bardzo duże skomplikowanie strategii w grach z niedoskonałą informacją sprawia, że gry te są wręcz niemożliwe do mistrzowskiego opanowania i są tak interesujące z naukowego punktu widzenia.

3.2. Algorytm CFR

Podstawą działania stworzonej sztucznej inteligencji jest algorytm CFR (ang. *Countefactual Regret Minimization*). Wprowadzenie oraz szczegółowy opis algorytmu zostały zawarte w pracy *An Introduction to Counterfactual Regret Minimization* [17] napisanej przez Todda W. Nellera z Gettysburg College oraz Marca Lanctota z Maastricht University. Jest to algorytm uczenia ze wzmocnieniem.

Do opisu algorytmu należy zdefiniować kilka dodatkowych pojęć:

- Krok czasowy t (lub T) jest liczony względem kolejnych rozdań rozgrywanych przez algorytm licząc od 1. Po każdym rozdaniu t wzrasta o 1. Górnym indeksem t oznaczone będą wartości parametrów przy kroku czasowym t.
- $\pi^{\sigma}(\zeta)$ oznacza prawdopodobieństwo z jakim przy profilu σ gracze dochodzą do historii ζ. Natomiast $\pi^{\sigma}_{-i}(\zeta)$ oznacza *counterfactual probability*, czyli prawdopodobieństwo przy założeniu, że akcje gracza i prowadzące do historii ζ są podejmowane z prawdopodobieństwem 1. W dalszej kolejności jako *counterfactual*, również będziemy oznaczać wartości uzyskane poprzez założenie wbrew faktom, że gracz i dąży do historii ζ lub zawierającemu ją zbiorowi informacji I, z prawdopodobieństwem 1.
- Niech Z oznacza zbiór końcowych historii, $\zeta \sqsubset z, z \in Z$ oznacza niekońcową historię, będącą częścią historii końcowej z. $u_i(z)$ oznacza nagrodę (wygraną lub przegraną) gracza i przy historii końcowej z. Wówczas jako *counterfactual value* nie-końcowej historii ζ definiujemy:

$$v_i(\sigma, \zeta) = \sum_{z \in Z, \zeta \sqsubset z} \pi^{\sigma}_{-i}(\zeta)\pi^{\sigma}(z)u_i(z) \tag{7}$$

- Jako *counterfactual regret* nie podjęcia akcji a przy historii ζ jest definiowany jako:

$$r(\zeta, a) = v_i(\sigma_{I \to a}, \zeta) - v_i(\sigma, \zeta) \tag{8}$$

  Counterfactual regret przy zbiorze informacji I definiowany jest analogicznie:

$$r(I, a) = r(\zeta, a), \zeta \in I \tag{9}$$

- Jako *accumulated counterfactual regret* definiujemy:

$$R_i^T(I, a) = \sum_{t=1}^{T} r_i^t(I, a) \tag{10}$$

- Jako *nonnegative accumulated counterfactual regret* definiujemy:

$$R_i^{T,+}(I, a) = \max(R_i^T(I, a), 0) \tag{11}$$

Algorytm CFR opiera się na równaniu przedstawionym przez Harta i Mas-Collela w 2000 roku służącym do aktualizowania strategii w taki sposób, żeby średnia strategia zbiegała do równowagi Nasha [18]:

$$\sigma^{T+1}(I,a) = \begin{cases} \dfrac{R_i^{T,+}(I,a)}{\sum_{a\in A(I)} R_i^{T,+}(I,a)} & gdy \sum_{a\in A(I)} R_i^{T,+}(I,a) > 0 \\ \dfrac{1}{|A(I)|} & w\ pozostałych\ przypadkach \end{cases} \tag{13}$$

Posiadając już wszystkie potrzebne definicje, wzory i narzędzia, można przejść do przedstawienia podstawowego algorytmu CFR. Warto zauważyć, że po początkowym przetasowaniu talii, układ kart jest do końca rozdania zdeterminowany, zatem z samego układu talii wynika, jakie karty otrzyma każdy z graczy.

Algorytm

1: Inicjalizacja całkowitego żalu: $\forall_I\ r_I[a] \leftarrow 0$
2: Inicjalizacja całkowitych sum strategii: $\forall_I\ s_I[a] \leftarrow 0$
3: Inicjalizacja strategii: $\sigma^1(I,a) \leftarrow 1/|A(I)|$

4: **function** CFR(ζ, *t*, *talia*, π_1, π_2):
5: **if** ζ jest historią końcową **then**
6: **return** $u_i(\zeta)$
7: **end if**
8: $\eta_1, \eta_2 \leftarrow$ prywatne informacje graczy przy historii ζ i układzie talii $talia$
9: $i \leftarrow$ gracz podejmujący decyzję przy historii ζ
10: $I \leftarrow$ zbiór informacji zawierającym historię ζ oraz prywatne informacje η_i
11: $v_\sigma \leftarrow 0$
12: $v_{\sigma_I}[a] \leftarrow 0$ dla każdego $a \in A(I)$
13: **for** $a \in A(I)$ **do**
14: **if** *i* == 1 **do**
15: $v_{\sigma_I}[a] \leftarrow$ CFR(ζa, *t*, $talia$, $\sigma^t(I,a) \cdot \pi_1$, π_2)
16: **else if** *i* == 2 **do**
17: $v_{\sigma_I}[a] \leftarrow$ CFR(ζa, *t*, $talia$, π_1, $\sigma^t(I,a) \cdot \pi_2$)
18: **end if**
19: $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_I}[a]$
20: **end for**
21: **for** $a \in A(I)$ **do**
22: $r_I[a] \leftarrow r_I[a] + \pi_{3-i} \cdot (v_{\sigma \to a}[a] - v_\sigma)$
23: $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
24: **end for**
25: $\sigma^{t+1}(I) \leftarrow$ uaktualniona strategia zgodnie z równaniem (9)
26: **return** v_σ

27: **function** Solve():
28: **for** t = {1, 2, 3, …, T} **do**
29: $talia \leftarrow$ wylosowany układ kart w talii
30: CFR(r, $t, talia, 1, 1$)
31: **end for**

Na początku, po inicjalizacji początkowych strategii (linie 1-3) losowana jest talia (linia 30). Następnie inicjalizowany jest algorytm CFR w korzeniu drzewa gry, które następnie jest przechodzone algorytmem DFS. W trakcie powrotu od liści do korzeni aktualizowane są kontrfaktyczne wartości (linie 15, 17) zgodnie z definicją (4). Na podstawie tych wartości aktualizowane są skumulowany żal oraz strategia (linie 22, 23). Wreszcie wyznaczana jest strategia do kolejnego kroku czasowego (linia 25).

3.3. Zastosowanie

W celu wypróbowania działania algorytmu, przed przystąpieniem do rozwiązywania pełnej gry, został on zastosowany w bardzo uproszczonej wersji pokera. W tej grze gracze otrzymują po jednej karcie i występuje tylko jedna runda licytacji. Po niej gracze pokazują karty i lepsza ręka wygrywa.

Algorytm wykonał 10 mln iteracji w ciągu 20 minut. Przeanalizujemy strategię SB (Tabela 3.).

Tabela 3. Strategia lowbotCFR dla historii „p".

karty	pas	sprawdzenie	podbicie o pól puli	podbicie o całą pulę
A	0%	33%	38%	29%
K	0%	33%	27%	40%
Q	0%	48%	25%	27%
J	0%	10%	86%	4%
T	0%	96%	2%	2%
9	0%	94%	1%	4%
8	0%	100%	0%	0%
7	0%	99%	0%	0%
6	36%	51%	12%	1%
5	77%	0%	10%	13%
4	93%	0%	2%	4%
3	75%	0%	8%	17%
2	77%	0%	19%	3%

Możemy teraz spojrzeć na tę strategię pod kątem opisanego wcześniej obecnego stanu wiedzy pokera. Widzimy, że strategia, której nauczył się algorytm, jest zgodna z tą bardzo podstawową zasadą podaną w rozdziale 1.2. Zanalizujmy każdy z punktów tej zasady pod kątem tej strategii:

- Jesteśmy agresywni (podbijamy) z najlepszymi i najgorszymi rękami (dla blefu).

Widzimy, że jesteśmy agresywni w większości sytuacji, gdy mamy bardzo silne karty (J lub lepiej) oraz czasami, gdy mamy najsłabsze karty (5 i gorzej). Prawie nigdy nie jesteśmy agresywni ze średnimi kartami.

- Jesteśmy pasywni (sprawdzamy, czekamy) ze średnimi rękami i z częścią najlepszych, które służą do zastawiania pułapki na przeciwnika.

Widzimy, że najczęściej sprawdzamy mając średnie karty (6-T), ale czasem również z najsilniejszymi (łapanie w pułapkę). Nigdy nie sprawdzamy z najgorszymi kartami.

- Poddajemy pule z najgorszymi rękami.

Pulę poddajemy jedynie, gdy mamy najsłabsze karty (nigdy, gdy mamy 7 lub lepiej, czasem z 6, bardzo często z 5 i gorzej)

4. Optymalizacja

4.1. Abstrakcja

Jednym z największych wyzwań przy tworzeniu lowbotCFR było zmniejszenie rozmiaru problemu do takich rozmiarów, które umożliwiłyby trening algorytmu na dostępnych maszynach (docelowo w chmurze AWS – *Amazon Web Services*). Można to uczynić ograniczając liczbę możliwych stanów, utożsamiając niektóre stany z innymi wprowadzając abstrakcję. Wiąże się to jednak z utratą części informacji na temat stanu.

Na początku należy ograniczyć zbędne informacje, które w żadnym razie nie wpływają na optymalną strategię. Taką informacją jest rozróżnianie rąk pod względem rodzajów kolorów, choć już nie pod względem liczby kart w różnych kolorach. Kolejność kart na ręce również nie ma znaczenia, przecież można ułożyć karty jak tylko się chce.

Dla przykładu ręce: A♣ Q♣ T♣ 7♥ 2♥ oraz A♠ Q♠ T♠ 7♦ 2♦ są z punktu widzenia strategii identyczne, ponieważ, nie mając żadnych dodatkowych informacji, nie ma przesłanek do istnienia różnej wartości oczekiwanej ręki posiadającej 3 trefle od takiej posiadającej 3 piki, ponieważ w pokerze, w przeciwieństwie do na przykład brydża, nie występuje starszeństwo kolorów. Natomiast ręka A♥ Q♥ T♦ 7♦ 2♣ jest już inna niż te wymienione wcześniej, ponieważ nie mając żadnych trzech kart w jednym kolorze dużo ciężej będzie uzyskać kolor, który jest bardzo mocnym układem. Ręka 2♣ Q♣ T♣ 7♥ A♥ również będzie inna, ponieważ mając trzy trefle z asem, jeśli skompletujemy kolor będzie to najmocniejszy kolor do asa, a w tym przypadku nie jest to gwarantowane.

Abstrakcją będzie zatem funkcja $f: H \to H$, która każdej ręce ze zbioru *H* przyporządkowuje również rękę z tego zbioru w następujący sposób:

1. Posortowanie ręki *h* względem kolorów i ich liczebności
   (np. 7♥ Q♣ 2♥ T♣ A♣ → Q♣ T♣ A♣ 2♥ 7♥).
2. Posortowanie względem starszeństwa kart w obrębie kolorów
   (np. Q♣ T♣ A♣ 2♥ 7♥ → A♣ Q♣ T♣ 7♥ 2♥).
3. Zamiana pierwszego występującego koloru (najliczniejszego) na ♠, drugiego na ♥, trzeciego na ♦, czwartego na ♣
   (np. A♣ Q♣ T♣ 7♥ 2♥ → A♠ Q♠ T♠ 7♥ 2♥).

Kolejność przyporządkowywania kolorów jest w zupełności dowolna, musi być jednak arbitralnie przyjęta i nie może być zmieniana. W naszym przykładzie zachodzi zatem:

f (7♥ Q♣ 2♥ T♣ A♣) = *f* (T♥ 2♦ A♥ 7♦ Q♥) = A♠ Q♠ T♠ 7♥ 2♥

Od teraz każda ręka $h \in H$ nie będzie już reprezentowana w algorytmie przez samą siebie, lecz przez rękę $\hat{h} = f(h) \in H$.

W następnym podrozdziale będą omówione kolejne metody abstrakcji, które niestety eliminują część informacji, o których nie możemy powiedzieć, że na pewno nie mają wpływu na strategię. Z tego powodu od tego momentu będziemy mogli mówić jedynie o suboptymalnej strategii. Jednak jak pokazują testy, jest to ciągle strategia, która bez trudności wygrywa z ludźmi.

4.2. Sieć neuronowa

Pierwszym krokiem do znaczącego (wykładniczego) zmniejszenia rozmiaru problemu jest naśladowanie procesu symplifikacji, którego używa człowiek grając w skomplikowane gry takie jak szachy czy poker. Wykonując ruch w szachach, gracz rozważa na kilka ruchów do przodu i ocenia w jakiej znajdzie się sytuacji przy danym ruchu po tych kilku ruchach. Nie analizuje przebiegu gry aż do jej zakończenia. Podobnie grając w pokera i wybierając akcję, gracz rozważa w jakiej znajdzie się sytuacji w przyszłości i czy będzie to lepsza sytuacja od innych, a nie analizuje możliwych przebiegów rozdania aż do jego zakończenia. Intuicja podpowiada, że można podobną metodologię zastosować w tworzonej sztucznej inteligencji, jednak najpierw trzeba ją ubrać w język algorytmów.

Po pierwsze wprowadzimy niezależne rundy licytacji [19] (ang. *independent Betting round*). Pierwsza i druga runda licytacji będą rozpatrywane przez algorytm oddzielnie i oddzielnie będzie się uczona strategia dla każdej z tych rund. Tworzy to jednak następujący problem: w jaki sposób algorytm ma się nauczyć strategii pierwszej rundy licytacji nie wiedząc jak rozdanie się zakończy, a zatem jak będzie wartość oczekiwana jego akcji. Nie znając wartości oczekiwanej akcji algorytm nie będzie w stanie obliczać wartości żalu, co jest kluczowe w algorytmie CFR. Potrzebny jest mechanizm, który zastąpi tę intuicję człowieka, która pozwala mu przewidzieć, że znajdzie się w lepszej sytuacji wykonując akcją *A* niż wykonując akcję *B*. W tym celu została użyta sieć neuronowa.

Na początku należy zdefiniować pojęcie zakresu (ang. *range*) gracza. Zakresem będziemy nazywać wartość funkcji $\Gamma_i^{\zeta,\sigma}: H \to [0,1]$, która przyporządkowuje każdej ręce $h \in H$ prawdopodobieństwo z jakim gracz i, posiada rękę h, przy historii ζ, grając według profilu strategii σ. Zakresy dla danej historii i profilu strategii można stworzyć korzystając z metody Monte Carlo.

Ponieważ zbiór H posiada skończoną liczbę elementów, możemy ponumerować wszystkie jego elementy od 1 do $|H| = \binom{52}{5} = 2\,598\,960$. Dodatkowo przyjmujemy, że ręce są numerowane według siły, gdzie $h_1 =$ „7s5s4h3h2d”, jest najsłabszą ręką, natomiast $h_{|H|} =$ ”AsKsQsJsTs” jest najmocniejszą ręką. Wówczas zakres można reprezentować jako wektor, którego k-ty element jest równy $\Gamma_i^{\zeta,\sigma}(h_k)$.

Po pierwszej rundzie licytacji gracze posiadają odpowiednio zasięgi Γ_1 i Γ_2. Zasięgi te są reprezentowane przez wektor $[\Gamma_1, \Gamma_2]$. Sieć neuronowa miałaby jako wejście przyjmować taki wektor, natomiast na wyjściu chcemy otrzymać wartość oczekiwaną każdej z rąk. Niestety wektor wejściowy i wyjściowy miałby długość około 5 milionów. Wytrenowanie takiej sieci neuronowej również wykracza poza możliwości standardowych komputerów. W tym celu użyta zostanie kolejna abstrakcja. Jest to kubełkowanie (ang. *binning*). Na początku należy ustalić liczbę kubełków. W wyniku testów przyjęta została liczba 100 kubełków. Zbiór kubełków jest oznaczony jako B. Następnie ręce są sortowane pod względem siły i wkładane do odpowiednich kubełków w takiej kolejności, zatem kubełek B_1 będzie zawierał najsłabsze ręce, a kubełek B_{100} najmocniejsze. Należy również ustalić rozmiary kubełków, ponieważ

gdyby były równoliczne, to bardzo słabe ręce zajęłyby 90% kubełków, a silne ręce zostałyby zmieszane w tych samych ostatnich kubełkach. Rozmiary kubełków określa wzór:

$$|B_i| = \left\lceil \frac{|H|}{\sum_{k=0}^{n-1} \exp(-\lambda \frac{k}{n})} \cdot \exp(-\lambda \frac{i}{n}) \right\rceil \tag{14}$$

$$B_i = \left\{ h_k \in H : \sum_{j<i} |B_j| < k \leq \sum_{j<i} |B_j| + |B_i| \right\} \tag{15}$$

B_i - i-ty kubełek, $i = 0, 1, \ldots, n-1$, H – zbiór wszystkich rąk, n – liczba kubełków, λ – stała > 0

W ten sposób pierwsze kubełki z najsłabszymi rękami będą znacznie większe od ostatnich kubełków.

Kubełkowanie jest reprezentowane przez funkcję $b: H \to B$, gdzie każdej ręce $h \in H$ przyporządkowywany jest kubełek $B_i \in B$.

Dzięki temu zasięg może być reprezentowany jako rozkład prawdopodobieństwa nie posiadania danej ręki, ale ręki należącej do danego kubełka nad zbiorem wszystkich kubełków. Możemy zapisać teraz zasięg jako funkcje kubełka:

$$\Gamma_i^{\zeta,\sigma}(B_k) = \sum_{h_l \in B_k} \Gamma_i^{\zeta,\sigma}(h_l). \tag{16}$$

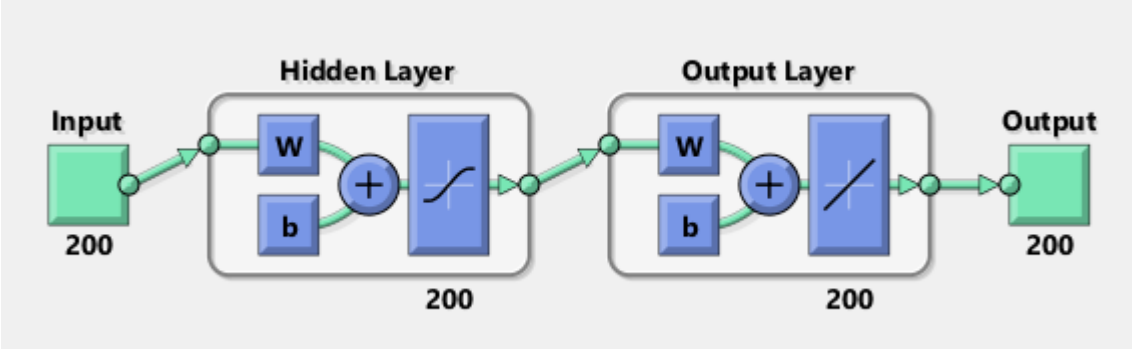
Wektor $[\Gamma_1, \Gamma_2]$w takim wypadku będzie miał długość jedynie 200, podobnie jak wektor wyjściowy.

Do wytrenowania sieci treningowej potrzebny jest jednak zbiór treningowy. Do stworzenia zbioru treningowego zostało użyte 20 000 losowo wygenerowanych wejść. Następnie dla każdego z tych wejść algorytm CFR znalazł optymalną strategię dla każdego z graczy oraz wartość oczekiwaną każdego kubełka przy założeniu, że gra się optymalnie. W ten sposób powstał zbiór treningowy zawierający 20 000 par (x_i, y_i), gdzie x_i to 200-elementowy wektor wejściowy zawierający zasięgi graczy, a y_i to 200-elementowy wektor zawierający wartości oczekiwane każdego kubełka z wektora wejściowego.

Do treningu został wykorzystany pakiet MATLAB i udostępnione w nim narzędzie *Neural Fitting*. Użyto sieci neuronowej typu *feedforward* z jedną warstwą ukrytą, z 200

neuronami w warstwie ukrytej. Algorytmem treningowym był Levenber-Marquardt. Schemat sieci można zobaczyć na rysunku (rycina 3).

Sieć neuronową możemy opisać jako funkcję $n: R \times R \to V \times V$, która parze zasięgów (Γ_1, Γ_2) przypisuję parę wektorów wartości oczekiwanych (V_1, V_2).



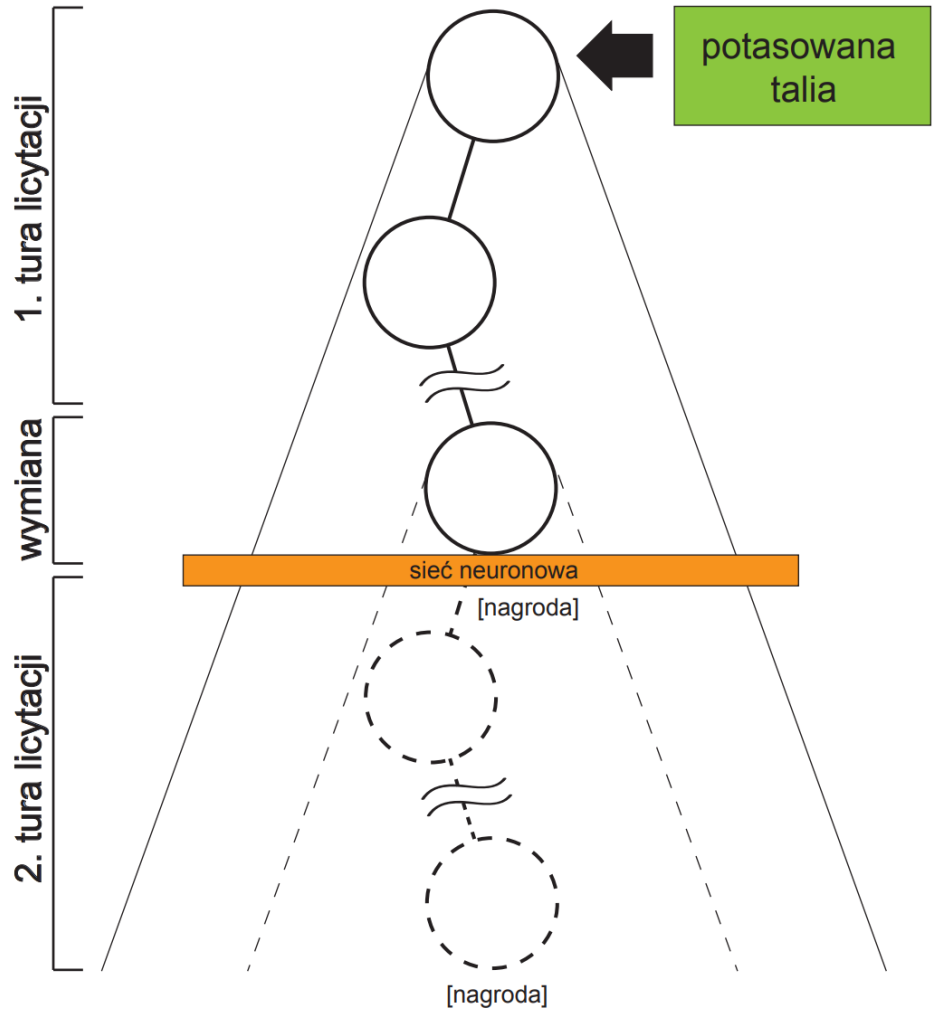
Rycina. 3 Schemat sieci neuronowej.

Posiadając funkcję $n(R_1, R_2)$, możemy zaimplementować ją do odpowiednio zmienionego algorytmu CFR, który pozwoli tym razem rozwiązać pełny problem. W tym celu należy odpowiednio zmodyfikować funkcję CFR z opisanego wcześniej algorytmu:

1: **function** CFR(ζ, *t*, *talia*, π_1, π_2):
2: **if** *h* jest stanem końcowym **then**
3: **if** trwa druga tura licytacji **then**
4: **return** $u_i(h)$
5: **else**
6: $R_1, R_2 \leftarrow$ zasięgi graczy przy historii *h* i strategii σ^t
7: *b* ⟵ numer kubełka, do którego należy ręka gracza *i* w stanie $\mathcal{I}$
8: **return** $n(R_1, R_2)[i \cdot 200 + b]$
9: **end if**
10: **end if**
11: $\eta_1, \eta_2 \leftarrow$ prywatne informacje graczy przy historii ζ i układzie talii $talia$
12: $i \leftarrow$ gracz podejmujący decyzję przy historii ζ
13: $I \leftarrow$ zbiór informacji zawierającym historię ζ oraz prywatne informacje η_i
14: $v_\sigma \leftarrow 0$
15: $v_{\sigma_I}[a] \leftarrow 0$ dla każdego $a \in A(I)$
16: **for** $a \in A(I)$ **do**
17: **if** *i* == 1 **do**
18: $v_{\sigma_I}[a] \leftarrow$ CFR(ζa, *t*, $talia$, $\sigma^t(I,a) \cdot \pi_1$, π_2)
19: **else if** *i* == 2 **do**
20: $v_{\sigma_I}[a] \leftarrow$ CFR(ζa, *t*, $talia$, π_1, $\sigma^t(I,a) \cdot \pi_2$)
21: **end if**
22: $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_I}[a]$
23: **end for**

24: **for** $a \in A(I)$ **do**
25: $r_I[a] \leftarrow r_I[a] + \pi_{3-i} \cdot (v_{\sigma \to a}[a] - v_\sigma)$
26: $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I, a)$
27: **end for**
28: $\sigma^{t+1}(I) \leftarrow$ uaktualniona strategia zgodnie z równaniem (9)
29: **return** v_σ

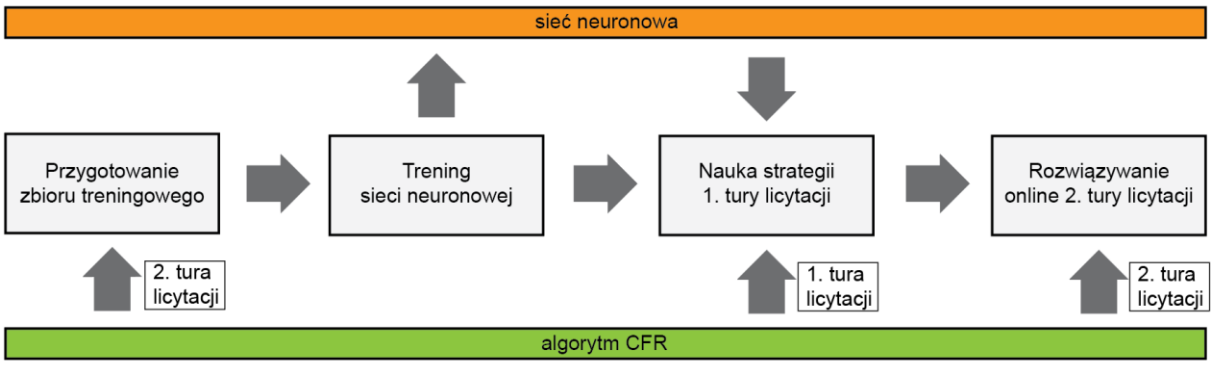
Algorytm zamiast za każdym razem schodzić do liści drzewa gry, dochodzi do jedynie do początku drugiej tury licytacji, po wymianie kart, i wówczas nagrodę, do tej pory liczoną poprzez przejście całego poddrzewa, zastępuję nagrodą odpowiadającą aktualnemu zasięgowi i aktualnej ręce, otrzymaną na wyjściu sieci neuronowej (rycina 4).



Rycina 4. Wykorzystanie sieci neuronowej w drzewie gry.

4.3. Architektura

Mając wszystkie komponenty można przejść do spięcia całości działania programu w całość. Poniżej pokazany jest schemat pełnej nauki lowbotCFR (rycina 5).



Rycina 5. Schemat pracy algorytmu.

Wyróżnione zostały następujące etapy:

- Przygotowanie zbioru treningowego. W tej części generowane jest 20 000 losowych par zasięgów graczy, które służą jako scenariusze po pierwszej turze licytacji oraz wymianie kart. Następnie za pomocą algorytmu CFR rozwiązywana jest druga tura licytacji przy danych scenariuszach. Na tej podstawie wyliczane są wartości oczekiwane każdego z kubełków dla obydwu graczy w danym scenariuszu. Powstają w ten sposób pary wejściowych zasięgów oraz wyjściowych wartości oczekiwanych, które tworzą zbiór treningowy.

- Trening sieci neuronowej. Na podstawie utworzonego w poprzedniej części oraz pakietu MATLAB, wraz z narzędziem *Neural Fitting*, trenowana jest sieć neuronowa do przewidywania wartości oczekiwanych dla dowolnej pary zasięgów wejściowych. Sieć została opisana w poprzednim rozdziale.

- Nauka strategii 1. tury licytacji. Posiadając dzięki sieci neuronowej funkcję, która pozwala przeszukiwać drzewo gry jedynie do fazy wymiany kart, lowbotCFR opracowuje strategię pierwszej tury licytacji przy użyciu algorytmu CFR. Odrzucenie całej części drzewa odpowiadającej za trzecią licytację pozwala ograniczyć problem na tyle, że możliwe staje się osiągnięcie rozwiązania.

- Rozwiązywanie *online* 2. tury licytacji. Część drzewa gry odpowiedzialna jedynie za drugą część licytacji w danym scenariuszu jest na tyle mała, że rozwiązanie jej jest możliwe w ciągu kilkunastu sekund. Problemem jest liczba możliwych scenariuszy (a zatem i poddrzew drugiej tury). Algorytm zatem rozwiązuje pojedyncze drzewa *online* w trakcie gry, bazując na zasięgach graczy, które są znane, ponieważ znana jest strategia w pierwszej turze licytacji.

Oprócz klas pomocniczych, program składa się z czterech podstawowych klas:

- Draw.cs. Klasa ta zawiera opis zasad gry. Zasady definiowane są za pomocą funkcji używanych przez algorytm CFR do odzyskiwania danych z historii oraz z talii, takich jak prywatne informację (linia 11), obecny gracz (linia 12), itd. W celu wykorzystania algorytmu do rozwiązania nowej gry wystarczy jedynie odpowiednio zmodyfikować tę klasę.

- Trainer.cs. Ta klasa zawiera opis nauki strategii przy użyciu algorytmu CFR. Znajduje się w niej również proces równoleglizacji nauki na dowolną liczbę wątków, przy użyciu biblioteki *System.Threading.Tasks*.

- Player.cs. Znajduje się tu algorytm służący do grania przeciwko lowbotCFR.

- Program.cs. Tutaj znajduje się opis schematu działania całości programu, opisany wcześniej. Ze względu na użycie zewnętrznego pakietu MATLAB program najpierw należy uruchomić z odpowiednimi opcjami w celu utworzenia zbioru treningowego, następnie skorzystać z pakietu MATLAB eksportując funkcję sieci neuronowej, a na koniec ponownie uruchomić lowbotCFR z innymi opcjami, który doprowadzi do końca proces nauki wykorzystując utworzoną w MATLAB-ie funkcję za pomocą interfejsu MATLABa do .NET.

Podstawowe metody znajdują się w załącznikach wraz z adresem repozytorium.

5. Analiza wyników

W testowaniu algorytmu wzięło udział dziesięciu graczy z kilkuletnim doświadczeniem gry w pokera. W poniższej tabeli (tabela 3) zawarte są informacje na temat ile rozdań rozegrali przeciwko CFR, jaki wynik liczony w bb/100 rozdań (bb oznacza tu wysokość dużej w ciemno) uzyskali, ile rozegrali rozdań oraz jakie było odchylenie standardowe również liczone w bb/100.

Tabela 3. Wyniki graczy przeciwko lowbotCFR.

Numer gracza	Liczba rozdań	Wynik (bb/100)
1	25549	-18
2	1155	-28
3	1016	-15
4	1004	-88
5	1000	-92
6	1000	-6
7	658	+40
8	451	-154
9	320	+15
10	250	-55

W sumie lowbotCFR rozegrał 32403 rozdania, z średnim wynikiem 23 bb/100 rozdań, co w porównaniu z typowymi wynikami nawet bardzo dobrych graczy, którzy wygrywają około 10bb/100 rozdań, jest wynikiem fenomenalnym. Można również zaobserwować, że gracze, którzy rozegrali więcej rozdań, a zatem, których wyniki są bardziej wiarygodne, wszyscy przegrali z lowbotCFR.

Choć te wyniki są więcej niż bardzo obiecujące, pozostaje problem bardzo dużego czynnika losowego w pokerze. Czasem zdarza się, że nawet wygrywający gracz przez kilkadziesiąt tysięcy rozdań przegrywa, lub odwrotnie, przegrywający gracz jest na plusie.

W celu jak największego ograniczenia czynnika losowego wygenerowane zostało losowo 10 000 rozdań. Następnie wymieniona wcześniej grupa 10 graczy rozegrała te rozdania dwukrotnie przeciw lobwotCFR. W czasie drugiej rundy zamieniając się z algorytmem pozycjami. LowbotCFR oczywiście nie zapamiętuje rozdań, które rozegrał, zatem nie było problemu ze znajomością kart. Gracze natomiast otrzymywali w każdej z rund otrzymywali różne rozdania. Wyniki zostały umieszczone w tabeli (tabela 4).

Tabela 4. Wyniki lowbotCFR przy powtórzonych rozdaniach.

Runda	Wynik lowbotCFR (bb/100 rozdań)
1	+56
2	+6

Algorytm zatem był w stanie wygrać niezależnie od kart, co pokazuje również, że poker jest pomimo dużego czynnika losowego, grą umiejętności, podobnie do takich gier jak brydż. LowbotCFR uzyskał zatem średni wynik 31 bb/100 rozdań. Eksperyment ten nie pozwala na uzyskanie pewności odnośnie wysokości przewagi algorytmu nad graczami w wymiarze bb/100 rozdań, lecz pozwala z całą pewnością stwierdzić, że z nimi wygrywa po wyeliminowaniu czynnika losowego.

Styl gry algorytmu nie odbiegał od oczekiwań, był perfekcyjnie zbalansowany, przez co gra przeciw niemu była niezwykle ciężka. Nie zauważono jednak bardzo nieortodoksyjnych ruchów, które kłóciłyby się mocno z intuicją i obecnym stanem wiedzy człowieka na temat strategii w *5 Card Draw*.

6. Podsumowanie

Uzyskane wyniki pozwalają śmiało stwierdzić, że w pokerze *5 Card Draw*, zwykli gracze nie mają szans w grze 1 na 1 przeciwko sztucznej inteligencji. Nie wiadomo jednak, jak algorytm poradzi sobie w grze przeciw najlepszym profesjonalistom na świecie. Jednak biorąc pod uwagę, że przewagi w pokerze są uważane za duże, jeżeli wynoszą około 5 bb/100 rozdań, można podejrzewać, że algorytm może prowadzić z nimi przynajmniej wyrównaną walkę.

LowbotCFR ma również możliwość ciągłego douczania się. W trakcie treningu przed grami testowymi, algorytm był trenowany równolegle na 30 wątkach przez 40 godzin. Istnieje jednak możliwość kontynuacji przerwanej nauki, zatem lowbotCFR, może się wciąż doskonalić

Architektura lowbotCFR została zaprojektowana w taki sposób, żeby był on jak najbardziej uniwersalny, zatem jedynie zmieniając klasę zawierającą zasady oraz modyfikując *pipeline*, można z łatwością przystosować program do nauki innych gier. Kolejnym wyzwaniem postawionym przed stworzoną inteligencją jest ta sama gra, jednak w wersji *Triple Draw*. W tej grze można trzykrotnie wymieniać karty oraz występują aż cztery rundy licytacji. Odmiana ta posiada prawie identyczną strukturę, jednak jest dużo bardziej złożona. Ze względu na to, nawet najlepsi gracze na świecie mają bardzo ogólną intuicję na temat tego, jaka strategia jest najlepsza. Może się zatem okazać, że algorytm nauczy się strategii, która będzie dużą niespodzianką i zmieni podejście do tej gry w przyszłości.

Bibliografia

[1] S. J. Brams, M. D. Davis, *Game theory, Encyclopedia Brittanica*, https://www.britannica.com/science/game-theory (20.02.2018), 2017

[2] OpenStax, *Principles of Economics,* Rice University, 2016

[3] J. G. James, *Imperfect Information in Medical Care, Wykłady*, Max Weber Programme Fellows' Websites, 2011/2012

[4] B.J. Copeland, *Encyclopedia Brittanica, Artificial Inteligence*, https://www.britannica.com/technology/artificial-intelligence (20.02.2018)

[5] G. Press, *A Very Shoir History of Artificial Intelligence*, Forbes, https://www.forbes.com/sites/gilpress/2016/12/30/a-very-short-history-of-artificial-intelligence-ai/#49b4d9f6fba2, (28.02.2017)

[6] A. Wagner, *A short history of AI schooling humans at their own games*, PBS News Hour, https://www.pbs.org/newshour/science/short-history-ai-schooling-humans-games, 28.02.2017r.

[7] www.piosolver.com (02.03.2018)

[8] M. Moravcik, M. Schmid, *DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker*, Science 356(6337), 508-513, 2017

[9] N. Brown, T. Sandholm, *Libratus: The Superhuman AI for No-Limit Poker,* Proceedings of 26th International Joint Conference on Artificial Intelligence, 5226 – 5228, 2017

[10] W. Dudziak*, Using Fictitious Play to Find Pseudo-Optimal Solutions for Full-Scale Poker*, International Conference on Artificial Intelligence, 374-380, 2006

[11] J. Heinrich, D. Silver, *Deep Reinforcement Learning from Self-Play in Imperfect-Information Games*, University College London, 2016

[12] http://wiki.ubc.ca/Poker_and_linear_programming_1, University of British Columbia (02.03.2018)

[13] M. Johanson, *Measuring the Size of Large No-Limit Poker Games*, University of Alberta, 2013

[14] *International Encyclopedia of the Social Sciences, 2nd Edition*, p. 540-541, 2008

[15] P. Nogal, *Dylemat więźnia jako przykład wykorzystania teorii gier*, Uniwersytet Gdański, Zarządzanie i Finanse 4(2), 87-95, 2012

[16] V. Li, *Pure vs. Mixed Strategies*, University of Waterloo, http://uweconsoc.com/pure-vs-mixed-strategies/ (20.02.2018), 2017

[17] T. W. Neller, M. Lnactot, *An Introduction to Counterfactual Regret Minimization*, Gettysburg College, 2013

[18] S. Hart, A. Mas-Colell *A Simple Adaptive Procedure Leading to Correlated Equilibrium*, Econometrica 68(5), 2000

[19] J. Shi, M. L. Littman *Abstraction Methods for Game Theoretic Poker*, International Conference on Computers and Games, 333-345, 2001

Spis rycin

Spis tabel

Załącznik A

- Funkcja implementująca algorytm CFR

```
private double CFR(string Deck, string History, string Hand1, string Hand2, double
Pot1, double Pot2, double p0, double p1)
        {
            int Player = Szu.GetCurrentPlayer(History);
            int Opponent = 1 - Player;
            string PlayerHand = (Player == 0) ? Hand1 : Hand2;
            string OpponentHand = (Opponent == 0) ? Hand1 : Hand2;

            double PlayerPot = (Player == 0) ? Pot1 : Pot2;
            double OpponentPot = (Opponent == 0) ? Pot1 : Pot2;

            string Actions = Szu.GetLegalActions(History);

            if (History[History.Length - 1] == ')')
            {
                double[] ranges = GetRanges(History);
                int b = GetBucket(PlayerHand);
                object result = null;
                MATLAB.Feval("nnet", 2, out result, ranges);
                double[] res = result as double[];
                return res[Player * NUM_BUCKETS + b];
            }
            if (Actions == Draw.TERMINAL_FOLD)
                return OpponentPot;
            if (Actions == Draw.TERMINAL_CALL)
                return OpponentPot * Szu.CompareHands(PlayerHand, OpponentHand);
            if (p0 == 0 && p1 == 0)
                return 0;
            string InfoSet = PlayerHand.Substring(PlayerHand.Length -
Draw.HAND_CARDS, Szu.HAND_CARDS) + History;

            Node Node = null;
            int NumActions;

            if (Actions == Draw.DRAW || Actions == Draw.LAST_DRAW)
                NumActions = (int)Math.Pow(2, Szu.HAND_CARDS);
            else
                NumActions = Actions.Length;
            Node = new Node();
            Node.Init(NumActions, Actions, InfoSet);

            if (!NodeMap.TryAdd(InfoSet, Node))
            {
                Node = NodeMap[InfoSet];
                Node.Count++;
            }
            double[] Strategy = GetStrategy(Node, (Player == 0) ? p0 : p1);
            double[] Util = new double[NumActions];
            double NodeUtil = 0.0;

        }
```

```
for (int i = 0; i < NumActions; i++)
            {
                string NextHistory;
                double NewPot = PlayerPot;
                string NewHand = String.Copy(PlayerHand);
                if (Actions == Draw.DRAW || Actions == Draw.LAST_DRAW)
                {
                    int NumDraw = Szu.DrawCards(History, Deck, PlayerHand, ref
NewHand, i);
                    if (Actions == Draw.DRAW)
                        NextHistory = History + "(" + Convert.ToString(NumDraw);
                    else
                        NextHistory = History + Convert.ToString(NumDraw) + ")";
                }
                else
                {
                    NextHistory = History + Actions[i];
                    if (Actions[i] == 'r')
                        NewPot = OpponentPot * 2;
                    else if (Actions[i] == 'p')
                        NewPot = OpponentPot * 3;
                    else if (Actions[i] == 'c')
                        NewPot = OpponentPot;
                }

                Util[i] = (Player == 0) ? -CFR(Deck, NextHistory, NewHand,
OpponentHand, NewPot, OpponentPot, p0 * Strategy[i], p1) : -CFR(Deck, NextHistory,
OpponentHand, NewHand, OpponentPot, NewPot, p0, p1 * Strategy[i]);
                if (Szu.GetCurrentPlayer(History) ==
Szu.GetCurrentPlayer(NextHistory))
                    Util[i] = -Util[i];

                NodeUtil += Strategy[i] * Util[i];
            }

            for (int i = 0; i < NumActions; i++)
            {
                double Regret = Util[i] - NodeUtil;
                Node.RegretSum[i] += Regret * ((Player == 0) ? p1 : p0);
            }

            Node.NodeUtil += NodeUtil;
            return NodeUtil;
        }
```

- Klasa *Node*, zawierająca dla każdego węzła drzewa wartości żalu oraz strategii wraz z metodami zwracającymi strategie.

```
public class Node
    {
        public int NumActions;
        public string Actions;
        public string InfoSet;
        public double[] RegretSum;
        public double[] Strategy;
        public double[] StrategySum;
        public double Realization;
        public double NodeUtil;

        public void Init(int num_actions, string ac, string info)
        {
            Count = 1;
            Realization = 0.0;
            NodeUtil = 0.0;
            NumActions = num_actions;
            Actions = ac;
            InfoSet = info;
            RegretSum = new double[NumActions];
            Strategy = new double[NumActions];
            StrategySum = new double[NumActions];
        }

        public double[] GetStrategy(double RealizationWeight)
        {
            double NormalizingSum = 0.0;
            for (int i = 0; i < NumActions; i++)
            {
                Strategy[i] = (RegretSum[i] > 0) ? RegretSum[i] : 0;
                NormalizingSum += Strategy[i];
            }
            for (int i = 0; i < NumActions; i++)
            {
                if (NormalizingSum > 0)
                    Strategy[i] /= NormalizingSum;
                else
                    Strategy[i] = 1.0 / NumActions;
                StrategySum[i] += RealizationWeight * Strategy[i];
            }
            return Strategy;
        }

        public double[] GetAverageStrategy()
        {
            double[] AvgStrategy = new double[NumActions];
            double NormalizingSum = 0.0;
            for (int i = 0; i < NumActions; i++)
                NormalizingSum += StrategySum[i];
            for (int i = 0; i < NumActions; i++)
            {
                if (NormalizingSum > 0)
                    AvgStrategy[i] = StrategySum[i] / NormalizingSum;
                else
                    AvgStrategy[i] = 1.0 / NumActions;
            }
            return AvgStrategy;
        }
    }
```

- Metody odtwarzające z historii informacje na temat rozdania

```
public string GetLegalActions(string History)
        {
            if (History == "")
                return "crp";
            string[] Split = History.Split('(', ')').Where(e => e != "").ToArray();
            string LastRound = Split[Split.Length - 1];
            if (Split.Length % 2 == 0)
            {
                if (LastRound.Length == 1)
                    return LAST_DRAW;
                else
                    return "fcrp";
            }
            if (LastRound[LastRound.Length - 1] == 'f')
                return TERMINAL_FOLD;
            if (Split.Length == 1 && LastRound == "rc")
                return "crp";
            if (LastRound[LastRound.Length - 1] == 'c')
            {

                if (LastRound.Length == 1)
                    return "crp";
                if (Split.Length / 2 < NUM_DRAWS)
                    return DRAW;
                else
                    return TERMINAL_CALL;
            }

            if (LastRound[LastRound.Length - 1] == 'r' ||
LastRound[LastRound.Length - 1] == 'p')
            {
                if (LastRound.Count(e => e == 'r') + LastRound.Count(e => e == 'p')
< CAP)
                    return "fcrp";
                else
                    return "fc";
            }

            return "fcrp";
        }

public int GetCurrentPlayer(string History)
        {
            if (History == "")
                return 1;

            string[] Split = History.Split('(', ')').Where(e => e != "").ToArray();
            string LastRound = Split.Last();
            string Actions = GetLegalActions(History);

            if (Actions == DRAW)
                return 1;
            if (Actions == LAST_DRAW)
                return 0;

            return 1 - LastRound.Length % 2;
        }
```

```
public double GetPotContribution(string History, int Player)
        {
            string[] Rounds = History.Split('(', ')').Where(e => e != "").Where((e,
i) => i % 2 == 0).ToArray();

            if (Rounds.Length == 1 && Rounds.First() == "rf" && Player == 0)
                return SMALL_BET / 2;
            if (Rounds.Length == 1 && Rounds.First() == "rcf" && Player == 1)
                return SMALL_BET;

            double Pot = 0;

            for (int i = 0; i < Rounds.Length; i++)
            {
                double Bet = (i < SB_ROUNDS) ? SMALL_BET : BIG_BET;
                double RoundPot = Rounds[i].Count(e => e == 'r') * Bet;
                if (i == Rounds.Length - 1 && Rounds.Last().Last() == 'f' && Player
== Rounds.Last().Length % 2 && RoundPot > 0)
                    RoundPot -= Bet;
                Pot += RoundPot;
            }

            return Pot;
        }

public int CompareHands(string Hand1, string Hand2)
        {
            Hand1 = Hand1.Substring(Hand1.Length - HAND_CARDS, HAND_CARDS);
            Hand2 = Hand2.Substring(Hand2.Length - HAND_CARDS, HAND_CARDS);
            int[] Value1 = GetHandValue(Hand1);
            int[] Value2 = GetHandValue(Hand2);
            for (int i = 0; i < Value1.Length; i++)
            {
                if (Value1[i] < Value2[i])
                    return -1;
                else if (Value1[i] > Value2[i])
                    return 1;
            }
            return 0;
        }


public string CreateInfoSet(string History, string Hand)
        {
            string InfoSet = Hand.Substring(0, HAND_CARDS);
            int index = HAND_CARDS;
            int p = 0;
            while (p < History.Length)
            {
                InfoSet += History[p];
                if (History[p] == ')' && index < Hand.Length)
                {
                    InfoSet += Hand.Substring(index, HAND_CARDS);
                    index += HAND_CARDS;
                }
                p++;
            }
            return InfoSet;
        }
```

```
public int DrawCards(string History, string Cards, string Hand, ref string
AfterHand, int ActionNumber)
        {
            string[] Draws = History.Split('(', ')').Where(e => e != "").Where((e,
i) => i % 2 == 1).ToArray();
            int LastCard = 2 * HAND_CARDS;
            foreach (char c in String.Join("", Draws))
                LastCard += (int)Char.GetNumericValue(c);

            string s = Convert.ToString(ActionNumber, 2);
            string Action = new string('0', HAND_CARDS - s.Length) + s;

            string OldHand = Hand.Substring(Hand.Length - HAND_CARDS, HAND_CARDS);
            string NewHand = "";

            for (int i = 0; i < HAND_CARDS; i++)
            {
                if (Action[i] == '0')
                    NewHand += OldHand[i];
                else
                {
                    NewHand += Cards[LastCard];
                    LastCard++;
                }
            }

            AfterHand = Hand + SortHand(NewHand);
            return Action.Count(e => e == '1');
        }
```

Całość kodu znajduje się w publicznym repozytorium pod adresem github.com/taraspiotr/lowbotCFR/.