Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

UCZENIE PRZEZ DEMONSTRACJĘ NA PODSTAWIE INFORMACJI OBRAZOWEJ W ŚRODOWISKU 3D

Wojciech Kopeć, 101675

Promotor
dr inż. Krzysztof Dembczyński

Poznań, 2017 r.

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wstęp

Kolejna rewolucja przemysłowa, której nadejście często upatrywane jest w gwałtownie rozprzestrzeniającym się użyciu narzędzi bazujących na uczeniu maszynowym, ma pociągnąć za sobą uwolnienie ludzkości od żmudnych i niewymagających kreatywności prac. Mimo że do prawdziwej, świadomej i twórczej sztucznej inteligencji jest ciągle daleko, to powszechne wykorzystanie komputerów do dalszej automatyzacji codziennych, powtarzalnych zadań wydaje się być w zasięgu ręki. Aby to osiągnąć, programy muszą być świadome otaczającego je środowiska i płynących z niego bodźców, adaptować swoje zachowanie do zmieniającej się sytuacji, zamiast stosować z góry ustalone reguły, oraz uczyć się nowych zadań bez potrzeby dokładnego opisu pożądanego zachowania.

W 2012 roku autorzy pracy [Krizhevsky et al., 2012] zrewolucjonizowali widzenie komputerowe. Stworzona przez nich głęboka sieć konwolucyjna osiągnęła drastyczny skok skuteczności w klasyfikacji obrazów wysokiej rozdzielczości do jednej z tysiąca ogólnych kategorii. Od tego czasu rozwiązania bazujące na ich architekturze wyrównały i przewyższyły skuteczność ludzi w tym samym zadaniu. W 2015 autorzy pracy [Mnih et al., 2015] połączyli klasyczny algorytm *uczenia ze wzmocnieniem*, *Q-learning* z głębokimi sieciami neuronowymi, co pozwoliło stworzyć program, który na podstawie surowych obrazów 2D sam nauczył się grać w klasyczne gry Atari, dla wielu z nich osiągając wyniki znacznie przewyższające ludzkie. Te dwa odkrycia otworzyły drogę do programów uczących się działać w środowisku 3D, a dalej w realnym świecie, na podstawie surowej informacji obrazowej.

Wiodące metody uczenia ze wzmocnieniem, z *Q-learningiem* na czele, mają jednak dwa poważne mankamenty.

Po pierwsze, dopiero po dłuższym czasie nauki programy zaczynają się zachowywać sensownie. Systemy, które po długim treningu zaczynają przewyższać człowieka, by jeszcze później zachwycić stosowaniem niespotykanych wcześniej strategii rozwiązywania znanych problemów, zaczynają naukę od długiego czasu poświęconego na losowe i bezsensowne działania oraz wielokrotne powtarzanie tych samych, nieskutecznych strategii. W zastosowaniach takich jak gry planszowe lub komputerowe, gdzie dostępne są dokładne symulatory rozgrywki, takie zachowanie podczas uczenia nie jest przeszkodą, ale dla realnych problemów, dla których nie ma możliwości symulacji albo symulacja jest zbyt wolna, takie zachowanie jest nie do zaakceptowania. Metoda, w której podczas nauki helikopter rozbija się w komputerowej symulacji tysiąc razy może być użyteczna. Metoda wymagająca rozbicia tysiąca prawdziwych helikopterów zdecydowanie nie.

Po drugie, wiodące metody uczenia ze wzmocnieniem są bardzo wolne. Nauka programów rozwiązujących aktualnie badane problemy trwa najczęściej dziesiątki godzin i wymaga sprzętu komputerowego o potężnych możliwościach obliczeniowych. Dodatkowo, jak wspomniano wcześniej,

duża część tego czasu poświęcana jest na zrozumienie przez trenowane programy podstawowych zasad rządzących środowiskiem, w którym się znajdują, które dla człowieka są oczywiste.

Właśnie na wykorzystaniu wiedzy i prezentacji człowieka oparte są metody *uczenia przez demonstrację*. W ramach działania tych metod ekspert, będący najczęściej człowiekiem potrafiącym skutecznie wykonywać dane zadanie (lub zewnętrzny, specjalizowany program komputerowy), demonstruje programowi jak się zachować, żeby osiągnąć oczekiwany rezultat. Program ma zadanie zbudować klasyfikator, który minimalizuje rozbieżności pomiędzy akcjami wykonywanymi w danej sytuacji przez program i przez eksperta.

Metody *uczenia przez demonstrację* pozwalają na uzyskanie poprawnie zachowujących się w środowisku programów w czasie zdecydowanie krótszym niż w przypadku uczenia ze wzmocnieniem. W niektórych zastosowaniach konieczność dostępu do eksperta i brak umiejętności „kreatywnego" generowania rozwiązań przez program może być przeszkodą, ale dla znacznej części problemów szybkie uzyskanie programu nieznacznie ustępującemu ludzkiemu ekspertowi jest w zupełności wystarczające. Co najważniejsze, dzięki *uczeniu przez demonstrację* program potrafi wykonywać stawiane przed nim zadania bez konieczności wcześniejszego kontaktu ze środowiskiem, czyli bez konieczności rozbijania helikopterów. Opublikowana niedawno praca [Hester et al., 2017] pokazuje też, że *uczenie przez demonstrację* można połączyć z *Q-learningiem*, uzyskując metodę, która pozwala szybko uzyskać dobrze działający program nauczony na podstawie pokazów eksperta, który potrafi dalej samemu udoskonalać swoje działanie, osiągając umiejętności znacznie przewyższające przedstawione demonstracje.

Celem pracy jest zweryfikowanie, czy przy pomocy metod *uczenia przez demonstrację* możliwe jest szybkie uzyskanie wyników porównywalnych z wynikami *Q-learningu* w środowisku opartym na informacji obrazowej 3D.

W ramach pracy zostaną zaimplementowane na bazie głębokich sieci neuronowych metody *kopiowania zachowań (ang. Behavioral Cloning)* i *agregacji zbioru danych (ang. Dataset Aggregation, DAgger)*. Ich działanie będzie przeanalizowane pod względem wygody dla eksperta udzielającego demonstracji, a ich zachowanie będzie przetestowane na wybranych scenariuszach środowiska VizDoom i porównane z *Q-learningiem* pod względem osiąganych wyników i czasu nauki.

Struktura pracy jest następująca. W rozdziale 2 opisano podstawy teoretyczne i podstawowe metody *uczenia ze wzmocnieniem* i *uczenia przez demonstrację*. W rozdziale 3 opisano środowisko VizDoom, w którym przeprowadzono wszystkie eksperymenty, oraz aktualne metody rozwiązywania problemów uczenia ze wzmocnieniem na podstawie informacji wizualnej. W rozdziale 4 opisano zaimplementowane w ramach pracy algorytmy. W rozdziale 5 opisano konfigurację i wyniki przeprowadzonych eksperymentów. Rozdział 6 stanowi podsumowanie pracy.

Rozdział 2

Podstawy teoretyczne

W poniższym rozdziale przedstawione są metody uczenia ze wzmocnieniem i uczenia przez demonstrację, stojące za nimi motywacje oraz podstawy teoretyczne. Dalej omówione są najważniejsze metody oraz aktualny stan wiedzy.

2.1 Słownik pojęć

W dalszej części pracy używane będą następujące terminy:

- *agent/gracz* - program sterujący kontrolowanym obiektem poruszającym się w badanym środowisku,
- *ekspert* - człowiek lub zewnętrzny program, który zna optymalną (albo przynajmniej skuteczną) politykę działania i potrafi udzielić agentowi informacji na temat realizacji tej polityki i oceniać politykę agenta,
- *epizod/gra* - ciąg interakcji agenta lub eksperta z otoczeniem, od stanu początkowego do stanu terminalnego.
- *krok/klatka/krotka/doświadczenie* - pojedyncza informacja s, a, s', r zebrana przez agenta lub eksperta w ramach jednego kroku interakcji ze środowiskiem.
- *ocena/odpytanie eksperta* - zarządzanie przez program agenta, aby ekspert określił jaką decyzję podjąłby w danym stanie.
- *trajektoria* - uporządkowana lista krotek pochodząca z jednego epizodu.
- *uczeń* - agent (w algorytmach uczenia przez demonstrację), najczęściej w kontekście odpytywania eksperta.

2.2 Proces decyzyjny Markowa

Środowisko, w którym porusza się program, albo inaczej agent, jest matematycznie zamodelowane jako proces decyzyjny Markowa *(ang. Markov decision process, MDP)* [Bellman, 1954]. Oznacza to, że środowisko ma w każdym momencie czasu określony stan i umożliwia wykonanie określonych akcji, za które agent może otrzymać pozytywną lub negatywną nagrodę. Rezultatem wykonania akcji w danym stanie jest przejście do następnego stanu. Zakłada się, że nowy stan jest zależny tylko od stanu poprzedniego i wykonanej akcji. Takie środowisko ma właściwość braku

pamięci (lub inaczej, *własność Markowa*). Celem agenta jest zgromadzenie nagród o jak największej sumie wartości. Im bardziej odległe w przyszłości nagrody, tym mniej są wartościowe (są dyskontowane).

Formalnie, proces decyzyjny Markowa definiujemy jako piątkę $(S, A, T(\cdot,\cdot), R(\cdot,\cdot), \gamma)$, gdzie:

- S jest skończonym zbiorem możliwych stanów środowiska,
- A_s jest skończonym zbiorem akcji możliwych w stanie s,
- $T_a(s, s')$ - funkcja przejść, która reprezentuje prawdopodobieństwo trafienia do stanu s' po wykonaniu akcji a w stanie s,
- $R_a(s, s')$ - funkcja nagrody, która określa nagrodę (lub wartością oczekiwaną nagrody, obie mogą być negatywne) otrzymywaną po wykonaniu akcji a w stanie s i trafieniu do stanu s',
- $\gamma \in [0, 1]$ jest współczynnikiem dyskontowym, obniżającym wartość nagród uzyskanych w przyszłości.

Celem jest maksymalizacja zdyskontowanej sumy nagród

$$\sum_{t=0} \gamma^t R(s_t, s_{t+1}),$$

gdzie kolejne t są kolejnymi momentami czasowymi. Ponadto:

- Polityką (strategią) π, realizowaną przez agenta, nazywamy funkcję $\pi : S \to A$, która określa, jak agent powinien się zachować w danym stanie w celu osiągnięcia maksymalnej możliwej nagrody.
- Funkcja użyteczności $U(s)$ lub wartości *(ang. Value)* $V(s)$ określa maksymalną oczekiwaną nagrodę, jaką agent może osiągnąć znajdując się stanie s i postępując dalej zgodnie z aktualną polityką. Poniższe równanie oparte jest na równaniu Bellmana [Bellman, 1954].

$$U(s) = V(s) = \max_{a \in A(s)} \sum_{s'} T_a(s, s')(R_a(s, s') + \gamma U(s'))$$

- Funkcja $Q(s, a)$ określa maksymalną oczekiwaną nagrodę, jaką agent może osiągnąć wykonując w stanie s akcję a i postępując dalej zgodnie z aktualną polityką.

$$Q(s, a) = \sum_{s'} T_a(s, s')(R_a(s, s') + \gamma \max_{a' \in A(s)} Q(s', a'))$$

Znając funkcje przejść możliwe jest iteracyjne określenie optymalnej polityki działania agenta.

W badanym problemie środowisko VizDoom jest *częściowo obserwowalnym procesem decyzyjnym Markowa*, co oznacza, że stan obserwowany przez agenta nie zawiera pełnej informacji o środowisku. Środowisko to jest stochastyczne, co oznacza że skutki działań agenta nie są deterministyczne - wielokrotne wykonanie tej samej akcji w tym samym stanie może przynieść różne rezultaty.

2.3 Uczenie ze wzmocnieniem

W przypadku, kiedy funkcja przejść $T_a(s, s')$ albo funkcja nagród $R_a(s, s')$ nie jest znana (albo wielkość przestrzeni stanów S sprawia, że analityczne podejście nie jest możliwe), mamy do czynienia z *uczeniem ze wzmocnieniem (ang. Reinforcement learning, RL)*.

Intuicyjnie, uczenie ze wzmocnieniem opisuje sytuację, w której agent porusza się w nieznanym środowisku. Na podstawie obserwowanych rezultatów swoich działań buduje wiedzę o środowisku, pozwalającą na określenie strategii działania optymalną dla danej wizji środowiska. Postępując według tej strategii zdobywa dalsze doświadczenia, pozwalające dalej uaktualniać wiedzę o środowisku i politykę agenta.

2.3.1 Uczenie ze wzmocnieniem a uczenie nadzorowane

W rozdziale 2.2 określono politykę π jako funkcję $\pi : S \to A$, określającą optymalną akcję do wykonania a dla każdego stanu s. Odpowiednikiem w uczenia nadzorowanego byłoby określenie π jako klasyfikatora $S \to A$. Do uczenia ze wzmocnieniem nie stosuje się jednak technik uczenia nadzorowanego, ponieważ, bez rozwiązania całego problemu, nigdy nie są znane „poprawne" akcje a dla danych stanów s. Mimo, że środowisko dostarcza czasem informacji zwrotnej w postaci nagród lub kar, to wykonanie danej akcji w danym stanie jest najczęściej konsekwencją całej poprzedniej sekwencji ruchów - określenie, który z ruchów w sekwencji był faktycznie kluczowy dla uzyskania określonego rezultatu jest nietrywialne.

Przykładowo, w partii szachów zwycięski ruch jest najczęściej konsekwencją konkretnych zagrań lub błędów popełnionych wiele ruchów wcześniej - zadaniem algorytmu uczącego jest określenie, które z ruchów były decydujące dla końcowego wyniku. Ruchy, które doprowadzają do zwycięstwa mogą krótkodystansowo przynosić straty (np. poświęcenie figury) - algorytm uczący powinien zrozumieć, że mimo negatywnej informacji zwrotnej dany ruch był pożądany.

2.3.2 Zalety i zastosowanie

Uczenie ze wzmocnieniem jest narzędziem, które świetnie sprawdza się w sytuacjach, w których środowisko jest zbyt skomplikowane, żeby analitycznie znaleźć optymalną politykę działania. Dzięki temu, że model przejść i model nagród opisane są rozkładami prawdopodobieństwa, a nie stałymi zależnościami, uczenie ze wzmocnieniem radzi sobie bez problemu z modelowaniem zachowań w bardzo niepewnym świecie. Dzięki współczynnikowi dyskontowemu γ, możliwe jest balansowanie pomiędzy optymalizacją krótko i długoterminowych zysków.

Najważniejsza jest jednak możliwość działania bez żadnej wiedzy i silnych założeń na temat środowiska, w którym znajduje się agent. RL zakłada brak wiedzy o modelu świata, a wszystkie informacje czerpane są z doświadczeń na temat interakcji ze środowiskiem. Jest to kluczowe założenie, ponieważ dla wielu praktycznych problemów, które adresuje RL stworzenie dokładnego modelu świata jest niemożliwe (np. stworzenie dokładnego matematycznego modelu aerodynamiki i zachowania śmigłowca), albo mimo znajomości modelu matematycznego przestrzeń możliwych stanów jest zbyt wielka, by analitycznie otrzymać rozwiązanie (np. szachy, go).

Uczenie ze wzmocnieniem stosuje się z powodzeniem do sterowania robotami [Mataric, 1994] i windami [Crites and Barto, 1996], grania w gry planszowe (backgammon[Tesauro, 1992], go [Silver et al., 2016], warcaby [Samuel, 1959]) i gry komputerowe [Mnih et al., 2015].

2.4 Metody

Podejścia stosowane do uczenia ze wzmocnieniem możemy podzielić na trzy rodzaje, w zależności od typu informacji na której bazuje agent [Russell and Norvig, 2009].

1. Agent z polityką - uczy się polityki $\pi : S \to A$. Przykłady:
   - algorytmy ewolucyjne,
   - uczenie przez demonstrację.
2. Agent z funkcją użyteczności U. Przykłady:
   - adaptatywne programowanie dynamiczne *(ang. adaptative dynamic programming, ADP)*,
   - metoda różnic czasowych *(ang. temporal difference learning, TDL)*.
3. Agent z funkcją użyteczności Q. Przykłady:
   - Q-learning,
   - SARSA *(ang. State, Action, Reward, State, Action)*

Poniżej przedstawione zostaną najważniejsze metody.

2.4.1 Metoda różnic czasowych

Metoda różnic czasowych *(ang. Temporal difference learning, TDL)* [Sutton, 1988] opiera się na uaktualnianiu stanu wiedzy agenta na podstawie różnicy pomiędzy spodziewanym a zaobserwowanym wynikiem.

Agent trafiający do stanu s' po wykonaniu akcji a w stanie s może uaktualnić stan swojej wiedzy: $U(s) \leftarrow U(s) + \alpha(R(s,s') + \gamma U(s') - U(s))$ lub $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,s') + \gamma(\max_{a'} Q(s',a') - Q(s,a)))$, gdzie α jest współczynnikiem prędkości uczenia. Jeżeli α w odpowiedni sposób zmniejsza się w czasie, to TDL gwarantuje zbieżność do optimum globalnego.

2.4.2 Funkcja U, Q i SARSA

Implementację metody różnic czasowych można oprzeć na różnych funkcjach, modelujących wiedzę agenta. Przykładowo:

- Funkcja U (patrz: rozdział 2.2) opisuje użyteczność stanu,
- Funkcja Q (patrz: rozdział 2.2) opisuje użyteczność wykonania danej akcji w danym stanie,
- SARSA stanowi wariację metody Q. W Q-learningu wartość funkcji Q jest aktualizowana na podstawie wartości Q dla najlepszej akcji do wykonywania w stanie s' ($\gamma \max_{a'} Q(s',a') - Q(s,a)$)), natomiast w SARSIE na podstawie wykonanej przez agenta akcji ($\gamma(Q(s',a') - Q(s,a))$)), czyli przebytej przez agenta trajektorii $s \to a \to s' \to a'$. Aktualizacja TD w SARSA-ie wygląda następująco:

  $$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s,s') + \gamma(Q(s',a') - Q(s,a)))$$

  SARSA może dla niektórych problemów zachowywać się nieznacznie lepiej niż Q-learning, ale w większości przypadków będzie się uczyła wolniej bez wpływu na jakość agenta.

Mimo podobnych wzorów i definicji nauka funkcji Q ma jedną, diametralną przewagę nad nauką funkcji U - funkcja Q nie wymaga znajomości modelu świata do wyboru najlepszej akcji do wykonania. Zbiór dostępnych akcji A jest znany agentowi. Przy wyborze najlepszej akcji a w stanie s:

- Agent z funkcją Q wybiera akcję $a = \arg\max_{a \in A} Q(s, a)$.
- Agent z funkcją U wybiera akcję, która maksymalizuje $U(s')$ - wartość stanu, do którego trafi agent: $a = \arg\max_{a \in A} \sum_{s'} T_a(s, s')U(s')$. Obliczenie tego wyrażenia wymaga znajomości modelu przejść $T_a(\cdot, \cdot)$, czyli modelu świata. Można przyjąć, że dla trudniejszych i realnych problemów model świata nie jest dostępny.

Z tego powodu większość wiodących rozwiązań w dziedzinie uczenia ze wzmocnieniem oparta jest na Q-learningu. Dalsza część pracy przyjmuje Q-learning jako obowiązującą metodę rozwiązywania problemu uczenia ze wzmocnieniem.

Niezależnie od metody, dla realnych i interesujących naukowo problemów uczenie ze wzmocnieniem jest wymagające obliczeniowo i czasowo. Mimo wspomagających agenta technik, nauka sprowadza się najczęściej do interakcji ze środowiskiem metodą prób i błędów - potrzeba wiele prób i błędów, zanim agent zacznie pojmować zasady rządzące środowiskiem w którym się znajduje, a potem dużo dalszych zanim znajdzie dla danego środowiska satysfakcjonującą skuteczną politykę działania.

2.5 Aproksymatory funkcji Q

Zdefiniowana w podrozdziale 2.4 przykładowa reguła aktualizacji wartości funkcji Q wygląda następująco:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, s') + \gamma(\max_{a'} Q(s', a') - Q(s, a)))$$

Wynika z niej, że po wykonaniu ruchu wartość funkcji Q dla poprzedniego stanu aktualizujemy na podstawie otrzymanej nagrody i wartości funkcji Q dla stanu aktualnego. Oznacza to, że dla każdego stanu, który analizujemy, konieczna jest znajomość jego wartości funkcji Q dla wszystkich możliwych akcji, a dalej oznacza to, że dla dokładnego przedstawienia funkcji $Q(s, a)$ konieczne jest zapamiętanie $|S| \cdot |A|$ wartości. Co więcej, aby uzyskać sensowne wartości tej funkcji konieczne jest odwiedzenie każdego ze stanów wiele razy, zanim aktualizowana stopniowo wartość funkcji Q będzie bliska prawdziwej. Wiele z tych stanów jest też bardzo podobnych do siebie nawzajem, więc wiedza wyniesiona dla jednego stanu powinna się w pewien sposób generalizować na podobne stany.

Backgammon, jedna z gier planszowych służących jako benchmark algorytmów uczenia ze wzmocnieniem, ma 10^{20} możliwych stanów, a szachy 10^{40}. Jeden obraz 90x60 pikseli w skali szarości, używany jako zapis stanu w problemie rozwiązywanym w ramach poniższej pracy może przyjąć 256^{5400} różnych kombinacji. Wiele realnych problemów opisanych jest wartościami ciągłymi, nie dyskretnymi, a praktyczna liczba ich możliwych stanów rośnie wykładniczo wraz ze wzrostem dokładności pomiaru.

Rozważanie i zapamiętanie każdego stanu z osobna dla bardziej skomplikowanych problemów jest niemożliwe i niepraktyczne ze względu na liczbę możliwych stanów i podobieństwo wielu stanów. Rozwiązaniem jest wykorzystanie *aproksymatora funkcji Q* - niestablicowanej, parametrycznej funkcji pary (stan,akcja) $\hat{Q}_\theta(s, a)$, gdzie θ jest wektorem parametrów funkcji.

Aproksymator (za [Jaskowski, 2016]):

- musi być łatwo obliczalny,
- kompresuje dużą przestrzeń stanów w znacznie mniejszą przestrzeń parametrów,
- uogólnia wiedzę na temat podobnych stanów,
- w większości przypadków przyspiesza uczenie w stosunku do wersji stablicowanej ze względu na uogólnianie wiedzy.

Jako jedne z pierwszych i prostszych aproksymatorów stosowano funkcje liniowe, opierające się na ręcznie zdefiniowanych cechach: $\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$, gdzie wektor $x = (x_1, x_2, \ldots, x_n)$ jest wektorem cech. Przykładem zastosowania może być gra w warcaby, opisana w [Samuel, 1959]. Zaletami liniowego aproksymatora są prostota i łatwość interpretacji, a także szybkość obliczania i nauki. Dalszym krokiem było wykorzystanie sieci neuronowych jako aproksymatorów w grze Backgammon [Tesauro, 1992]. W pierwszej wersji algorytmu wykorzystano ręcznie zaprojektowane cechy, w kolejnych wykorzystano prawie surową informację o rozkładzie pionków na planszy. Sieć neuronowa jest bardziej skomplikowanym i trudniejszym do nauczenia aproksymatorem, ale jest w stanie zamodelować znacznie bardziej złożone funkcje.

2.6 Eksploracja

W uczeniu ze wzmocnieniem agent posiada umiejętność uczenia się na podstawie zdobytych doświadczeń. Na początku każdej nauki jest jednak zupełnie nieświadomy zasad świata, w którym się znajduje i nie jest w stanie podejmować sensownych działań. Konieczna jest metoda pozwalająca na zdobywanie nowych doświadczeń przy jednoczesnej możliwości szlifowania i ulepszania opracowanych wcześniej przez agenta sposobów. Proces nakładania agenta do zbadania nieznanych jeszcze obszarów przestrzeni stanów nazywany jest eksploracją.

Przetarg pomiędzy eksploracją nowych stanów a zgłębianiem znanych jest problemem nietrywialnym i przedmiotem wielu badań.

2.6.1 Algorytm e-zachłanny

Podstawowym i często używanym podejściem do eksploracji jest algorytm ϵ-zachłanny, w którym agent z zadanym prawdopodobieństwem ϵ zamiast akcji optymalnej względem aktualnej polityki wykonuje akcję losową. Takie zachowanie jest mało wydajne, szczególnie kiedy optymalne zachowanie agenta wymaga zaplanowania złożonych lub dalekosiężnych planów.

2.6.2 Algorytm R-max

Prostym, ale skutecznym i posiadającym teoretyczne gwarancje zbieżności algorytmem jest zaproponowany w [Brafman and Tennenholtz, 2002] R-max, realizujący ideę optymizmu wobec niepewności. Podstawą R-maxa jest optymistyczna inicjalizacja – przed rozpoczęciem uczenia funkcja aproksymacyjna powinna zwracać maksymalną nagrodę dla wszystkich stanów i akcji. W ramach działania agent będzie uaktualniał (czyli obniżał) spodziewaną nagrodę w odwiedzonych stanach.

Największa spodziewana nagroda będzie zwracana dla zachowań, które agent odkrył już jako zyskowne i dla zachowań jeszcze nieodkrytych (dla których funkcja aproksymacyjna nie jest jeszcze poprawiona). Ten prosty zabieg powoduje, że algorytmy uczenia ze wzmocnieniem naturalnie balansują pomiędzy eksploracją i intensyfikacją przeszukiwania bez dodatkowych modyfikacji.

Od strony teoretycznej zaletą R-maxa jest duża ogólność zastosowania – algorytm wymaga spełnienia bardzo luźnych założeń, badany proces nie musi być nawet procesem decyzyjnym Markowa.

2.6.3 Przewidywanie przejść za pomocą autoenkodera

W [Stadie et al., 2015] autorzy zaproponowali rozwiązanie, które pozwala ocenić, w jakim stopniu odwiedzony stan jest dla agenta nowością. Opiera się ono na stworzeniu generatora, którego zadaniem jest przewidywanie, jaki stan osiągnie agent po wykonaniu danej akcji w danym stanie. Predykcja porównywana jest z faktycznie osiągniętym stanem, a wielkość błędu jest wyznacznikiem nowości stanu – im większy błąd predykcji, tym bardziej nieznany stan, za co przyznawana jest większa nagroda eksploracyjna. Jak w większości opisywanych publikacji, w [Stadie et al., 2015] rozwiązywano problem uczenia agenta grania w gry zręcznościowe na podstawie surowego obrazu z wykorzystaniem Q-learningu i głębokich sieci neuronowych.

Pierwszą kwestią do rozwiązania przy implementacji pomysłu jest metryka pozwalająca określić podobieństwo stanów. Próby predykcji wartości konkretnych pikseli opisane przez autorów nie przyniosły efektów, generując tylko szum. Zamiast tego trenowano autoenkoder oparty o głęboką sieć neuronową i wykorzystano jedną z ukrytych warstw o mniejszej liczbie jednostek tej sieci jako enkoder stanu, który przenosi surowy obraz do przestrzeni o znacznie mniejszej liczbie parametrów. Za miarę podobieństwa między stanami przyjęto odległość kartezjańską parametrów uzyskanych z zakodowania dwóch stanów. Zakodowane stany używane były do wytrenowania właściwego, prostszego aproksymatora, za pomocą którego którego określano nowość stanu. Dla każdego przejścia między stanami przyznawano sztuczną nagrodę zależną od nowości odwiedzonego stanu.

Potencjalnym problemem związanym z tym podejściem jest to, że Q-learning stara się nauczyć funkcji, która jest niestacjonarna. Autorzy piszą, jednak, że w praktyce nie stanowiło to problemu.

2.6.4 Bootstrapowane DQN

Inną taktykę dywersyfikacji przeszukiwania przy wykorzystaniu głębokiej sieci neuronowej zaprezentowano w [Osband et al., 2016]. Podobnie jak w [Stadie et al., 2015] uczono sieć funkcji Q, jednak zamiast pojedynczej funkcji Q trenowano jednocześnie K funkcji Q, przy czym każda trenowana była tylko na podzbiorze przykładów uzyskanym za pomocą techniki bootstrapingu. Każda funkcja Q reprezentowana była przez jedną z K „głów" wspólnej wielopoziomowej sieci.

Dla każdego z epizodów wybierana była losowo jedna głowa, czyli funkcja Q i przez cały epizod agent kierował się polityką optymalną dla tej funkcji Q.

Dzięki temu zabiegowi każda z sieci Q była nauczona na podstawie nieco różnych doświadczeń i prezentowała nieco inną politykę działania. Nowe informacje o pożądanych zachowaniach były prędzej czy później propagowane do każdej z głów, ale jednocześnie różnorodność zachowań była wystarczająca, żeby utrzymać eksplorację.

Autorzy raportują spowolnienie uczenia o zaledwie 20% w stosunku do normalnej, pojedynczej sieci Q, ale w przeprowadzonych w ramach tej pracy eksperymentach uczenie było znacznie wolniejsze.

2.7 Uczenie przez demonstrację

Uczenie ze wzmocnieniem jest bardzo kosztowne obliczeniowo, a trudniejsze problemy wymagają nauki na podstawie tysięcy lub milionów kroków agenta. Znaczną część tego czasu program spędza na początkowym poznawaniu możliwości i zasad rządzących środowiskiem, albo żmudnym,

stopniowym poprawianiu suboptymalnych zachowań, ewoluujących powoli w skuteczną politykę agenta.

Ludzie i zwierzęta, których zachowanie często stanowi inspirację i motywację dla nowych rozwiązań algorytmicznych, potrafią odtwarzać czynności i zachowania na podstawie samej obserwacji wykonania danej czynności przez innych. Czerpiąc z tego przypadku, wskazane jest konstruowanie algorytmów, które będą potrafiły uczyć się zaawansowanych polityk nie na bazie milionów prób i błędów, ale na bazie obserwacji kilku, lub nawet jednego, powtórzenia wykonania docelowego zadania. Taki cel stawiany jest przed uczeniem przez demonstrację.

W większości przypadków ekspertem jest człowiek, który potrafi wykonać zadanie postawione przed agentem i potrafi sterować agentem w celu jego wykonania. Wykorzystanie takiego eksperta pociąga za sobą poważne konsekwencje:

- **czas eksperta może być znacznie droższy niż czas maszynowy** - jeżeli dla badanego problemu dostępne jest wiarygodne i wydajne środowisko symulacyjne, to wykorzystanie klasycznego uczenia ze wzmocnieniem może być bardziej opłacalne niż zdobywanie prezentacji eksperta,

- **zachowanie eksperta może być nieoptymalne** - sposób realizacji zadania przez eksperta może być suboptymalny. Uczenie ze wzmocnieniem wyprzedziło ekspertów w wielu dziedzinach, na przykład w grze w szachy, backgammona i go. Opierając się na samym odtwarzaniu zachowań eksperta agent nie osiągnie lepszych niż on wyników.

- **zachowanie eksperta może być niespójne** - realizacje tego samego zadania przez eksperta cechuje najczęściej pewna wariancja. Co więcej ekspert odpytywany o zachowania dla tego samego zadania podczas prezentacji w różnych warunkach może prezentować niespójne ze wcześniejszymi zachowania. W przypadku częściowo obserwowalnych problemów decyzyjnych ekspert może mieć dostęp do większej ilości informacji niż agent, i podświadomie korzystać z nich podczas prezentacji.

- **konieczne jest zrozumienie „idei" za zachowaniami eksperta** - jak w każdym problemie uczenia, konieczna jest generalizacja wiedzy. W uczeniu przez demonstrację duży nacisk kładziony jest na użycie niewielkiej ilości danych uczących do zrozumienia skomplikowanych zadań, przez co odporność na przeuczenie i umiejętność generalizacji są wyjątkowo istotne.

Omawiane poniżej publikacje wykorzystują często ekspertów komputerowych - za ekspertów służą oddzielne, niezależne programy (przykładowo znające model środowiska i potrafiące podejmować optymalne decyzje). Przy nauce od komputerowych ekspertów nie występują opisane powyżej problemy, (w szczególności koszt i niespójność), co znacznie ułatwia i przyspiesza badania nad algorytmami uczenia przez demonstrację. Z drugiej strony, prawidłowości zaobserwowane przy eksperymentach z ekspertem komputerowym mogą nie zachodzić przy użyciu ludzkiego eksperta i odwrotnie.

Eksperci komputerowi mogą się za to sprawdzić w sytuacji, w której jeden wielozadaniowy i uniwersalny agent uczy się wykonywania wielu zadań na podstawie prezentacji wielu wyspecjalizowanych do danego zadania programów.

Głównym celem stawianym przed uczeniem przez demonstrację jest szybkie uzyskiwanie zadowalających wyników lub uzyskiwanie ich bez potrzeby kontaktu ze środowiskiem w trakcie treningu.

2.7.1 Kopiowanie zachowań

Najprostszym podejściem do uczenia przez demonstrację, nazywanym kopiowaniem zachowań *(ang. Behavioral cloning)* i opisanym między innymi w [Schaal, 1999], jest traktowanie problemu jak każdego innego problemu uczenia nadzorowanego. W kopiowaniu zachowań, w przeciwieństwie do maksymalizowania nagrody agenta w uczeniu ze wzmocnieniem, minimalizowana jest różnica pomiędzy polityką wyuczonego agenta a polityką eksperta.

To podejście zakłada jednak, jak każda metoda uczenia nadzorowanego, że dane uczące i testowe są niezależne i mają jednakowy rozkład, podczas gdy przy uczeniu przez demonstrację nauczona polityka ma bezpośredni wpływ na osiągane później stany, na podstawie których dana polityka będzie sprawdzana - intuicyjnie, trajektorie eksperta będą przedstawiały dobre zachowania i będą odwiedzały tylko dobre stany leżące na ścieżce optymalnej polityki. Gdy klasyfikator popełni błąd w odwzorowywaniu polityki eksperta, agent najprawdopodobniej trafi do stanu nieodwiedzonego przez eksperta i w którym nie będzie wiedział jak się zachować. Z dużym prawdopodobieństwem oznacza to popełnianie następnych błędów, ponieważ uczeń nie miał jak nauczyć się „podnoszenia się" po błędach. Jak dowiedziono w [Ross and Bagnell, 2010] wynikający z tego błąd rośnie kwadratowo w stosunku do czasu trwania epizodów.

2.7.2 Uczenie w przód

Pierwszym podejściem opisywanym przez [Ross and Bagnell, 2010] jest uczenie w przód *(ang. Forward Training)*. Podejście opiera się na przeprowadzeniu kilku powtórzeń uczenia, gdzie w każdym kroku następuje uczenie się jednej polityki w jednym, konkretnym, momencie. Jeżeli uczenie będzie przeprowadzone po kolei dla każdego kolejnego kroku w czasie, to próbka uzyskanych stanów, na których prowadzone jest dalsze uczenie odpowiada dystrybucji stanów testowych, a algorytm może odpytać eksperta o właściwe działanie w osiągniętych stanach, dzięki czemu ekspert ma okazję zaprezentować jak „podnosić się" po popełnieniu błędów przez klasyfikator. Powyższe podejście działa tylko dla zadań o skończonym horyzoncie czasowym, wymaga dużej interakcji z ekspertem, możliwości zrestartowania stanu środowiska oraz dokładnego odtworzenia uzyskanego wcześniej stanu, co w wielu przypadkach nie jest możliwe do zrealizowania.

2.7.3 Stochastyczny mieszający iterowany algorytm

W celu wyeliminowania tych ograniczeń [Ross et al., 2010] proponują stochastyczny mieszający iterowany algorytm *(ang. Stachastic Mixing Iterative Learning, SMILe)*. Opierając się na algorytmie iterowania polityki, algorytm w każdym kroku stosuje nową stochastyczną politykę wybierając z zadanym prawdopodobieństwem pomiędzy wykonywaniem polityki wyuczonej w poprzednim kroku i konstruowanej w danej iteracji nowej polityki. Prawdopodobieństwo wyboru nowej polityki jest niewielkie. Algorytm zaczyna od dokładnego wykonywania akcji eksperta. W każdej kolejnej iteracji prawdopodobieństwo odpytania eksperta jest coraz niższe i zbiega do 0.

Opisane rozwiązanie zostało z powodzeniem przetestowane na przykładzie grania w proste gry, gdzie danymi wejściowymi były surowe dane obrazowe z ekranu. Wadą tego podejścia jest brak odrzucania nieskutecznych polityk podczas iteracji, co może prowadzić do niestabilnych wyników.

Wykorzystanie analogicznego rozwiązania proponują [Bengio et al., 2015]. Ich propozycja zakłada wybieranie z prawdopodobieństwem ϵ polityki eksperta i z prawdopodobieństwem $1 - \epsilon$ polityki wyuczonej. Początkowa wartość ϵ powinna wynosić 1, aby klasyfikator mógł nauczyć się odtwarzać politykę eksperta. Wraz z postępem nauki ϵ powinno stopniowo maleć do 0, aby klasyfikator miał szanse nauczyć się stanów nieodwiedzonych przez eksperta.

2.7.4 Agregacją zbioru danych

W kolejnej publikacji [Ross et al., 2010] prezentują nowe podejście, nazwane agregacją zbioru danych *(ang. Dataset Aggregation, DAgger)*. W uproszczeniu, podejście to jest następujące: w pierwszej iteracji algorytm zbiera dane testowe stosując politykę pokazaną przez eksperta, po czym trenuje klasyfikator odwzorowujący zachowanie eksperta na danym zbiorze danych. W każdej kolejnej iteracji algorytm stosuje politykę wygenerowaną w poprzedniej iteracji i dodaje dane uzyskane podczas jej stosowania do zbioru danych, po czym trenuje klasyfikator tak, by odwzorowywał zachowanie eksperta na całym zbiorze danych. Podobnie jak w poprzednim algorytmie, żeby przyspieszyć uczenie na pierwszych etapach algorytmu, dodano opcjonalną losową możliwość odpytania eksperta o decyzję dla wybranego stanu. Uzyskane z pomocą tej metody wyniki są wyraźnie lepsze od wyników metody SMILEe.

Metodę testowano z wykorzystaniem klasyfikatorów SVM na przykładzie gry wyścigowej 3D Mario Cart (ekspert wybiera pomiędzy skręcaniem w lewo i w prawo) oraz z komputerowym ekspertem na przykładzie gry Mario.

2.7.5 Podążanie za ekspertem a przewyższanie eksperta

Dla wielu praktycznych problemów polityka eksperta może nie być optymalna. Algorytm, który stara się tylko i wyłącznie odwzorować politykę eksperta będzie generował w takiej sytuacji nieoptymalne wyniki, które w wielu praktycznych sytuacjach mogą znacznie odbiegać od optimum. Prostym rozwiązaniem tego problemu przedstawionym w [Chang et al., 2015] jest stosowanie e-zachłannej strategii – w każdym ruchu algorytm może wybrać z małym prawdopodobieństwem e wykonanie losowej akcji zamiast akcji optymalnej według wyuczonej polityki. Dzięki temu algorytm może znaleźć lokalne optimum bliskie polityce eksperta. Warto zauważyć, że wymusza to posługiwanie się całościową nagrodą (kosztem) wykonania zadania jako celem optymalizacji, w przeciwieństwie do prostszego minimalizowania różnicy pomiędzy wynikami wyuczonej polityki a polityki eksperta.

2.8 Odwrócone uczenie ze wzmocnieniem

Użycie kopiowania zachowań i wywodzących się z niego metod pozwala na wytrenowanie agenta, który dla danego stanu będzie potrafił określić optymalną akcję na jeden krok do przodu. Taki agent zna optymalną politykę, ale nie jest świadomy jej powodów. Odwrócone uczenie ze wzmocnieniem *(ang. Inverse reinforcement learning, IRL)* opiera się na założeniu, że w ogólności optymalna polityka agenta nie stanowi najlepszego i najbardziej zwięzłego opisu zadania podstawionego przed agentem - najbardziej precyzyjnym opisem, pozwalającym na większą dowolność i adaptację jest znajomość funkcji nagród $R_a(\cdot,\cdot)$.

Oczywiście, bez modelu środowiska funkcja $R_a(\cdot,\cdot)$ nie jest dostępna, dlatego zadaniem postawionym przed odwróconym uczeniem ze wzmocnieniem jest odtworzenie $R_a(\cdot,\cdot)$ na podstawie dostarczonych trajektorii eksperta.

Zadanie odwróconego uczenia ze wzmocnieniem jest znacznie trudniejsze od zwykłego uczenia ze wzmocnieniem. Przede wszystkim, IRL musi się zmierzyć z 2 problemami:

- Niejasność $R_a(\cdot,\cdot)$ - w większości przypadków, dla danych trajektorii eksperta istnieje nieskończenie wiele pasujących $R_a(\cdot,\cdot)$. Formalnie oznacza to, że IRL nie ma zdefiniowanego poprawnego rozwiązania.

- Złożoność obliczeniowa - samo uczenie ze wzmocnieniem jest bardzo wymagające obliczeniowo. W odwróconym uczeniu ze wzmocnieniem, sprawdzanie $R_a(\cdot,\cdot)$ uzyskanych w kolejnych krokach wymaga każdorazowego rozwiązywania problemu uczenia ze wzmocnieniem na podstawie aktualnej funkcji $R_a(\cdot,\cdot)$, co oznacza, że IRL wymaga większego o rząd wielkości kosztu obliczeniowego niż RL.

Przykładowe zastosowania odwróconego uczenia ze wzmocnieniem to parkowanie samochodu [Abbeel et al., 2008], nawigacja na podstawie obrazów satelitarnych [Ratliff et al., 2006] albo wykonywanie ewolucji helikopterem[Coates et al., 2009].

Rozdział 3

Uczenie na podstawie informacji obrazowej

W poniższym rozdziale opisano podejście zastosowane w pracy [Mnih et al., 2015], które pozwoliło na skuteczną naukę na podstawie surowych danych obrazowych. Dalej przedstawiono wybrane techniki zwiększające skuteczność praktycznego uczenia ze wzmocnieniem. Na końcu opisano środowisko 3D VizDoom, oraz scenariusze na których przeprowadzono eksperymenty obliczeniowe.

3.1 Uczenie na podstawie surowych danych obrazowych - Atari 2600

Jednym z największych przełomów uczenia ze wzmocnieniem ostatnich lat była praca [Mnih et al., 2015], w której autorzy wykorzystali głębokie sieci neuronowe do stworzenia agenta potrafiącego grać w klasyczne gry z Atari 2600 na poziomie porównywalnym z człowiekiem, wykorzystując jako reprezentację stanu jedynie surowy zapis obrazu 2D. Dotychczas, jak w poprzednich przykładach, algorytmy uczenia ze wzmocnieniem opierały się na manualnie stworzonej reprezentacji stanów. W [Mnih et al., 2015] pokazano, że możliwe jest stworzenie rozwiązania, które samo będzie potrafiło ekstrahować wysokopoziomowe cechy z niskopoziomowych danych. Zaproponowana architektura, jak również pomysłowe usprawnienia zwiększające stabilność uczenia zaproponowane w artykule, a opisane w rozdziale 3.2 stanowią obecnie podstawę i punkt odniesienia dla większości dalszych badań na temat uczenia ze wzmocnieniem.

Jako aproksymator funkcji Q wykorzystano głęboką sieć neuronową. Z tego powodu opisywane podejście określa się często skrótem DQN *(ang. Deep Q Network)*, czyli głęboka sieć Q. Analogicznie jak w obecnie stosowanych architekturach rozpoznawania obrazu, pierwsze warstwy sieci to warstwy konwolucyjne, które wykrywają kolejno nisko i wysokopoziomowe cechy obrazu. Dalsze warstwy, w pełni połączone, łączą informacje z warstw konwolucyjnych we wnioski na temat stanu świata, na podstawie których następne warstwy mogą określić wartość funkcji Q.

3.1.1 Warstwy konwolucyjne

Warstwy konwolucyjne *(ang. convolutional layers)* są podstawowym elementem sieci neuronowych służących do analizy obrazów. W przeciwieństwie do warstw w pełni połączonych *(ang. fully connected)*, w których każdy neuron łączy się z wyjściem każdego neuronu z warstwy poprzedniej, warstwy konwolucyjne analizują obraz z podziałem na małe, niezależne fragmenty. Warstwa konwolucyjna składa się z wielu filtrów. Każdy filtr rozpoznaje jakiś wzorzec w małym fragmencie obrazu (na przykład o wielkości 6x6 pikseli). Filtr „przykładany" jest kolejno do kolejnych fragmentów obrazu (z przesunięciem równym wartości *kroku*). Każdy z filtrów wykrywa jakąś cechę.

W pierwszych warstwach mogą to być na przykład krawędzie, w dalszych bardziej złożone cechy. Wyjście warstwy konwolucyjnej można intuicyjnie postrzegać jako informację, gdzie na obrazie wykryto dane cechy.

3.2 Q-learning - usprawnienia

Skuteczność i stabilność Q-learningu może zostać drastycznie polepszona dzięki zastosowaniu następujących technik.

3.2.1 Pamięć powtórek

Szkielet uczenia ze wzmocnieniem opiera się na zbieraniu doświadczeń i uaktualnianiu na ich podstawie stanu wiedzy agenta. W praktyce, doświadczenia zbierane bezpośrednio po sobie są silnie skorelowane - przykładowo agent uczący się na podstawie obrazu jazdy samochodem w kolejnych klatkach widzi niemal identyczne obrazy i wykonuje najczęściej te same akcje. Oznacza to, że aktualizowanie wiedzy agenta na podstawie nowych doświadczeń, czy to pojedynczo czy w paczkach, będzie skutkować funkcją obciążoną w kierunku tych, nowych doświadczeń.

Aby temu zapobiec w [Mnih et al., 2015] zaproponowano metodę pamięci powtórek *(ang. Replay memory)*. Metoda opiera się na zapamiętywaniu znacznej ilości najnowszych doświadczeń. Po każdym kroku nowe doświadczenia dodawane są do pamięci (w przypadku braku miejsca zastępując najstarsze), a następnie z pamięci wybierana jest losowa próbka doświadczeń, na podstawie których aktualizowana jest wiedza agenta. Dzięki tej technice dane użyte do nauki przez agenta są nieskorelowane i niezależne. Dodatkowo, dzięki dostępowi do starszych danych agent jest mniej podatny na obniżanie jakości gry na skutek krótkotrwałych spadków wyników.

Dalsze rozszerzenia metody mają na celu np. priorytetyzowanie używania do nauki najważniejszych doświadczeń [Schaul et al., 2015].

3.2.2 Zamrażanie docelowej sieci

Podobnie jak pamięć powtórek, zamrażanie docelowej sieci *(ang. Target network freezeing / fixed target network)* zostało zaproponowane w [Mnih et al., 2015] i służy zmniejszeniu skutków obciążenia rozkładu danych uczących zebranych przez agenta, a wynikającego ze sposobu zbierania próbek. Zamrażanie sieci zakłada utrzymywanie dwóch funkcji Q - starej i nowej. Agent działa na podstawie nowej funkcji, ale wartości Q „docelowych" stanów używanych do aktualizacji wartości Q (2.4.1) pobierane są ze starej funkcji. Co jakiś czas do starej funkcji Q przepisywana jest nowa funkcja Q.

Technika ma na celu zniwelowanie oscylacji i ustabilizowanie zachowań agenta. Dzięki wykorzystaniu „zamrożonych" wartości do nauki funkcji Q zerwane jest sprzężenie zwrotne pomiędzy zebranymi danymi a wartościami docelowymi.

3.2.3 Kształtowanie

W wielu zadaniach stawianych przed uczeniem ze wzmocnieniem osiągnięcie celu jest bardzo trudne, a agent dostaje nagrody dopiero po osiągnięciu stanów terminalnych, albo na zaawansowanym etapie zadania. Agent uczący się na podstawie prób, błędów i losowych akcji nie jest najczęściej w stanie wykonać wystarczająco dużej części zadania, żeby dostać informację zwrotną w postaci nagrody, a więc nie ma jak się uczyć lub uczenie następuje bardzo wolno.

Kształtowanie *(ang. Shaping)* ([Mataric, 1994]) zakłada sztuczne wprowadzenie do środowiska dodatkowych nagród, które agent będzie dostawał po wykonaniu etapów pośrednich zadania. Przykładowo, przy grze w szachy, w której agent dostaje nagrodę tylko za wygraną lub przegraną (1 lub -1) można byłoby wprowadzić nagrodę 0.1 za zbijanie figur przeciwnika.

Kształtowanie wymaga możliwości ingerencji w środowisko albo percepcję agenta (rozpoznawanie, kiedy agent powinien dostać sztuczną nagrodę i ingerowanie w odczyty nagrody dokonywane przez agenta). Co ważniejsze, wymaga wiedzy eksperckiej na temat zadania wykonywanego przez agenta (możliwość określenia sensownych etapów zadania, na których agent miałby dostać sztuczną nagrodę) i wiedzy na temat środowiska, w którym agent się porusza (wysokość sztucznej nagrody musi być dopasowana do prawdziwych nagród, które może dostawać agent). Dodatkowo, kroki określone przez eksperta mogą wymuszać nieoptymalną politykę działania i powstrzymać agenta przed odkryciem optymalnych strategii.

3.2.4 Dropout

Technika *dropoutu*, opisana w pracy [Srivastava et al., 2014] jest narzędziem regularyzacji głębokich sieci neuronych i służy zapobieganiu zjawisku przeuczenia sieci.

Dropout polega na losowym traktowaniu wybranych neuronów jak usuniętych z sieci, na czas pojedynczych iteracji uczenia. W każdej kolejnej iteracji nauki inne neurony są losowo wygaszane. Dzięki temu wiedza na temat konkretnych cech jest rozpropagowana między wieloma neuronami i nie ma nadmiernej specjalizacji neuronów. Na etapie testowania sieci żadne z neuronów nie są usuwane, a aktywacje są normalizowane, żeby sumaryczna siła aktywacji odpowiadała sumarycznej sile aktywacji neuronów podczas treningu.

W używanych narzędziach *dropout* zaimplementowany jest najczęściej jako oddzielny typ warstwy sieci, przepuszczający do nastepnej warstwy tylko wybrane aktywacje neuronów z poprzedniej warstwy, a resztę aktywacji propagując jako 0. Warstwa dropoutu parametryzowana jest prawdopodobieństwem p, które oznacza z jakim prawdopodobieństwem neuronów zostanie zachowany.

3.3 Środowisko VizDoom

Środowisko VizDoom, przedstawione w [Kempka et al., 2016], jest narzędziem do testowania algorytmów sterowania na podstawie surowych danych o obrazie 3D. Środowisko bazuje na klasycznej grze Doom, w której gracz widzi trójwymiarowy świat z perspektywy pierwszej osoby i strzela do potworów. W stosunku do nauki w środowisku 2D, takim jak Atari 2600 [Mnih et al., 2015], nauka w środowisku 3D jest wielkim krokiem naprzód i stanowi znacznie lepsze przybliżenie nauki w realnym świecie.

VizDoom oferuje wygodny interfejs, który doskonale wpisuje się w standardowy szkielet metod uczenia ze wzmocnieniem. VizDoom potrzebuje mało zasobów, może działać bez środowiska graficznego, jest wydajny, pozwala na uruchamianie wielu instancji równolegle oraz na wygodne tworzenie nowych scenariuszy dopasowanych do potrzeb konkretnych problemów badawczych.

Vizdoom udostępnia interfejs programistyczny dla Pythona, Javy, C++ i Lua. Preferowany i najbardziej rozwijany jest Python.

Przykładowe minimalne użycie środowiska VizDoom przedstawione jest poniżej (prezentowany kod jest zmodyfikowanym przykładem *scenarios.py* dołączonym do środowiska):

```
1 from vizdoom import DoomGame
2
3 game = DoomGame()
4
```

```
game.load_config("../../scenarios/basic.cfg")

game.set_screen_resolution(ScreenResolution.RES_640X480)
game.set_window_visible(True)
game.init()

# Creates possible actions depending on how many buttons there are.
actions = prepare_actions(game.get_available_buttons_size())

episodes = 10

for i in range(episodes):
    game.new_episode()
    while not game.is_episode_finished():
        # Gets the state and possibly to something with it
        state = game.get_state()
        # Makes a random action and save the reward.
        reward = game.make_action(choose(state, actions))
        new_state  = game.get_state()
        learn(state,game.get_last_action(), reward, new_state)
```

3.4 Wykorzystane scenariusze

Eksperymenty przeprowadzono na następujących scenariuszach, reprezentujących łatwy, średni i wysoki poziom trudności. W ostatecznych porównaniach nie wykorzystano scenariusza podstawowego, ponieważ ma zbyt niski poziom skomplikowania żeby zobrazować zależności pomiędzy metodami.

3.4.1 Podstawowy (ang. Basic)

Środowisko składa się z prostokątnego pomieszczenia. Agent startuje w jednym końcu pomieszczenia, po środku ściany, a w losowym miejscu pod przeciwległą ścianą znajduje się pojedynczy, nieruchomy przeciwnik.

Agent może strzelać do przeciwnika oraz poruszać się w prawo lub w lewo. Agent ma ograniczoną amunicję i dostaje punkt za trafienie przeciwnika.

Racjonalna strategia polega na przesunięciu się w kierunku przeciwnika i oddaniu do niego pojedynczego strzału.

Rysunek 3.1: Scenariusz podstawowy

3.4.2 Obrona środka (ang. Defend the center)

Środowisko składa się z kolistej areny. Agent znajduje się na środku areny, a na jej krańcach losowo pojawiają się przeciwnicy, którzy poruszają się w stronę agenta, a po dotarciu do niego zadają mu obrażenia. Są dwa rodzaje przeciwników różniących się wyglądem i szybkością poruszania.

Agent może strzelać i kręcić się wokół własnej osi w lewo i prawo. Agent ma ograniczoną amunicję i dostaje punkty za każde trafienie przeciwnika.

Racjonalna strategia polega na kręceniu się w jedną stronę w kółko, ignorowaniu odległych przeciwników i strzelaniu do bliskich, priorytetyzując szybszych przeciwników. Ignorowanie dalekich przeciwników jest konieczne, żeby w czasie strzelania do nich inni przeciwnicy nie zaszli agenta od tyłu - optymalna strategia wymaga zajmowania się najpierw najbliższym zagrożeniem.

Rysunek 3.2: Scenariusz obrona środka

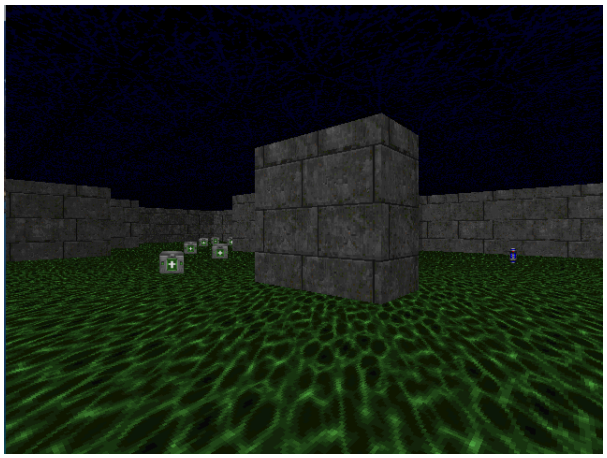
3.4.3 Trudne zbieranie apteczek (ang. Health gathering supreme)

Środowisko składa się z labiryntu, którego podłoga jest pokryta kwasem. Agent startuje w losowym miejscu labiryntu. W tym scenariuszu nie ma ruchomych przeciwników. Na podłodze labiryntu pojawiają się losowo apteczki, które dodają agentowi punkty życia i miny, które zabierają agentowi punkty życia. Kwas na podłodze nieustannie odbiera agentowi punkty życia.

Agent może poruszać się do przodu, na ukos w prawo lub w lewo oraz kręcić się wokół własnej osi w prawo lub w lewo. Agent dostaje punkty za pozostawanie przy życiu - im dłużej potrwa gra, tym większą sumę punktów zdobędzie agent.

Racjonalna strategia polega na chodzeniu po labiryncie, niewchodzeniu na miny i zbieraniu apteczek, preferując kierowanie się do dużych skupisk apteczek. Wskazane jest unikanie zbierania pojedynczych, odizolowanych apteczek, gdyż liczba punktów życia uzyskana z takiej apteczki może być mniejsza niż liczba punktów życia straconych na dotarcie do apteczki. Wskazane jest niepozostawanie w jednym obszarze labiryntu, ponieważ nowe apteczki mogą nie pojawiać się wystarczająco szybko, żeby utrzymać agenta przy życiu.

Co istotne, w tym scenariuszu bardzo często optymalna decyzja nie jest jednoznaczna - sensownie działający ekspert i agent mogą często wybierać pomiędzy wieloma poprawnymi drogami i zachowaniami.



Rysunek 3.3: scenariusz trudne zbieranie apteczek

Rozdział 4

Rozważane podejścia i ich analiza

W poniższym rozdziale przedstawione są badane i zaimplementowane w pracy podejścia. Każde z rozwiązań działa w ramach wspólnego szkieletu, bazującego na przykładowych rozwiązaniach towarzyszących środowisku VizDoom. Dzięki temu możliwe jest bezpośrednie porównanie zachowania różnych podejść przy zmianie tylko kluczowych algorytmów przy zachowaniu niezmienności pozostałych czynników.

4.1 Kopiowanie zachowań

Kopiowanie zachowań (ang. *Behavioral Cloning*) stanowi najbardziej podstawowe podejście do uczenia przez demonstrację. Na podstawie zebranych trajektorii eksperta uczony jest klasyfikator, który przyjmując na wejściu stan s ma za zadanie przewidzieć, jaką akcję a wykonałby w danej sytuacji ekspert. Mimo że dla wielu problemów kopiowanie zachowań jest nieskuteczne (powody opisane są w rozdziale 2.7.1), to wyniki osiągane na badanych scenariuszach VizDoom są bardzo zadowalające. Kopiowanie zachowań jest podstawą bardziej zaawansowanych technik opisanych w dalszych punktach.

Ważną różnicę w stosunku do Q-learningu stanowi fakt, że czas trwania metody ogranicza się w znaczącej części do czasu zbierania przykładowych trajektorii przez eksperta. Czas trenowania klasyfikatora na zebranych danych powinien być pomijalny w stosunku do czasu zbierania. Oznacza to, że uczenie agenta za pomocą kopiowania zachowań, wliczając w to zbieranie przykładowych trajektorii, trwa znacznie krócej niż za pomocą np. Q-learningu, w którym, dla wielu praktycznych problemów, agent musi grać przez przez wiele milionów klatek, by osiągnąć zadowalające wyniki.

4.1.1 Implementacja

Kopiowanie zachowań sprowadza się do klasycznego problemu uczenia nadzorowanego z wieloma klasami (możliwymi akcjami), z których dla danego stanu tylko jedna etykieta na raz jest poprawna. Dane wejściowe stanowią obrazy przedstawiające stan, wynikiem jest etykieta akcji, którą należy wykonać. Jako klasyfikatora użyto głębokiej sieci neuronowej.

Różnica pomiędzy architekturami sprowadza się do uczenia i interpretacji wyników. Przy Q-learningu sieć musi przewidywać wartość funkcji Q dla wszystkich akcji, a przy kopiowaniu zachowań wystarczy określenie najbardziej pasującej akcji. Wykonane i przewidywane akcje są zakodowane za pomocą *one-hot encoding*, a wynik uzyskiwany jest przez zastosowanie funkcji *softmax* na wartościach Q z architektury Q-learningu.

Co istotne, dla większości scenariuszy problem uczenia stan $\rightarrow$ akcja charakteryzuje się niezbalansowanym zbiorem danych. Akcje „strzelaj” występują znacznie rzadziej niż akcje ruchu.

Oznacza to, że klasyfikator naiwnie nauczony na niezmienionym zbiorze danych, mimo dobrej teoretycznej trafności, będzie zupełnie nieskuteczny (np. nie wybierze nigdy akcji „strzelaj").

Problem ten został rozwiązany przez zrównoważenie zbioru danych przy użyciu metody *oversampling* która polega na wielokrotnym uwzględnieniu w zbiorze uczącym przykładów mniej licznych klas w taki sposób, żeby liczność przykładów dla każdej z klas w uzyskanym zbiorze danych była podobna. W zastosowanej implementacji dla każdej akcji przetrzymywany jest oddzielny zbiór danych, a użyte do uczenia próbki składają się w równych proporcjach z przykładów zastosowania każdej akcji.

Warto zauważyć, że architektura Q-learningu wymaga, żeby każda możliwa akcja była zdefiniowana oddzielnie, łącznie z akcjami stanowiącymi złożenie innych, podstawowych akcji. Przykładowo akcje „lewo", „prosto" i „lewo i prosto" są dla modelu zupełnie niezwiązane, mimo że często można byłoby stosować je zamiennie. W przypadku kopiowania zachowań możliwe byłoby stworzenie klasyfikatora wieloetykietowego stan → akcja. Taki klasyfikator mógłby zamiast wybierać pomiędzy „lewo", „prosto" i „lewo i prosto" zdecydować „lewo" - tak i „prosto" - tak, uzyskując „lewo i prosto". Jednakże takie rozwiązanie nie zostało w tej pracy zbadane.

4.1.2 Sieć neuronowa

Zastosowana architektura sieci neuronowej opiera się na tej zaproponowanej w pracy [Mnih et al., 2015] i wykorzystanej dalej w przykładach środowiska VizDoom i w pracy [Kempka et al., 2016]. Sieć składa się kolejno z:

1. Warstwy konwolucyjnej z 8 filtrami o wielkości 6x6, krokiem 3x3 i wypłaszczoną liniową funkcją aktywacji *(ang. rectified linear unit, ReLU)*, opisaną w pracy [Glorot et al., 2011]
2. Warstwy konwolucyjnej z 8 filtrami o wielkości 3x3, krokiem 2x2 i wypłaszczoną liniową funkcją aktywacji *(ang. rectified linear unit, ReLU)*
3. Warstwy dropoutu z prawdopodobieństwem 0.5 (opisanej w rozdziale 3.2.4)
4. Warstwy w pełni połączonej o wielkości 256 neuronów z wypłaszczoną liniową funkcją aktywacji *(ang. rectified linear unit, ReLU)*
5. Warstwy w pełni połączonej o wielkości 128 neuronów z wypłaszczoną liniową funkcją aktywacji *(ang. rectified linear unit, ReLU)*
6. Warstwy dropoutu z prawdopodobieństwem 0.5
7. Warstwy wyjściowej w pełni połączonej z wypłaszczoną liniową funkcją aktywacji *(ang. rectified linear unit, ReLU)*, w której każdy neuron wyjściowy odpowiada jednej akcji
8. Warstwy softmax

W stosunku do wersji z pracy [Kempka et al., 2016] zastosowana sieć jest rozszerzona o jedną dodatkową warstwę w pełni połączoną i warstwy dropout. Co istotne, wszystkie użyte warstwy są złożone z kilka razy mniejszej liczby neuronów. Na końcu sieci została też dodana warstwa *softmax*, która jest standardowym elementem przy zadaniach klasyfikacji. Jej zadanie polega na przekształcaniu i normalizowaniu wartości aktywacji wyjść sieci odpowiedzialnych za poszczególne akcje. Po przekształceniu suma aktywacji ze wszystkich neuronów wyjściowych wynosi 1, dzięki czemu można interpretować wartość aktywacji każdego neuronu jako prawdopodobieństwo wystąpienia danej klasy.

Liczba i wielkości warstw zostały dobrane eksperymentalnie. Pozostałe parametry sieci, w szczególności parametry filtrów warstw konwolucyjnych zostały wybrane na podstawie wzorców z literatury.

4.1.3 Techniczna implementacja

Zbieranie danych zostało zrealizowane za pomocą trybu SPECTATOR środowiska VizDoom, pozwalającemu agentowi obserwować grę człowieka. Podczas gry eksperta zapisywane są stany, akcje i nagrody dla każdej kolejnej klatki. Trajektoria eksperta serializowana jest do pliku za pomocą narzędzia *pickle* dostępnego dla języka python.

Eksperta gra przy rozdzielczości 640x480 pikseli, i takiej wielkości obrazy zapisywane są do pliku z trajektorią. Konsekwencją są bardzo duże rozmiary plików (4GB dla 6 tysięcy klatek). Obrazy nie są zmniejszane przed zapisem, żeby umożliwić swobodne manipulowanie wielkością obrazów używanych do uczenia klasyfikatora, bez konieczności generowania nowych trajektorii eksperta przy innych ustawieniach obrazu.

Tryb SPECTATOR ustawiany jest w następujący sposób.

```
game.set_window_visible(True)
game.set_mode(Mode.SPECTATOR)
```

Trajektoria eksperta zbierana i zapisywana jest następująco.

```
    game.new_episode()
    while not game.is_episode_finished():
        state = game.get_state()
        game.advance_action()
        next_state = game.get_state()
        last_action = game.get_last_action()
        reward = game.get_last_reward()
        isterminal = game.is_episode_finished()

        print("State #" + str(state.number))
        print("Game variables: ", state.game_variables)
        print("Action:", last_action)
        print("Reward:", reward)
        print("====================")
        memory.append((state.screen_buffer,last_action, next_state.screen_buffer, reward,
     isterminal))
```

Zapis trajektorii do pliku wygląda następująco.

```
with open('recorder_episode.pkl', 'wb') as f:
    pickle.dump(memory, f, 2)
```

4.1.4 Zachowanie

Eksperymenty były prowadzone na scenariuszach „Trudne zbieranie apteczek (ang. Health gathering supreme)" i „Obrona środka (ang. Defend the center)". W obu przypadkach kopiowanie zachowań nauczone już na podstawie 3 trajektorii eksperta (6 tysięcy klatek) osiągało wizualnie sensowne zachowanie agentów i zaskakująco dobre wyniki.

Dla scenariusza „Obrony środka" agentowi zdarzało się strzelać w nieodpowiednim momencie lub nie strzelać, kiedy było to potrzebne. Częste było też zachowanie w postaci strzelania do odległych przeciwników, podczas gdy inni przeciwnicy mogli podkraść się za plecy agenta zabijając go i kończąc grę.

W tym scenariuszu zwiększanie liczby trajektorii eksperta użytych do uczenia zwiększało wyniki agenta, który w dużej części gier osiągał wyniki bliskie maksymalnym i tylko sporadycznie dawał się na początku gry zajść od tyłu na początku gry, co skutkowało pojedynczymi niskimi wynikami.

Dla scenariusza „Trudnego zbierana apteczek" agentowi często zdarzało się blokować w rogach labiryntu, wpadając w nieskończoną pętlę akcji. Problem i rozwiązanie zostało opisane w 4.3. Po wyeliminowaniu problemu agent zachowywał się wizualnie sensownie i osiągał przyzwoite wyniki. Problemem jest tylko nauczenie agenta omijania min.

Na początku pracy ekspert uznał miny za mniejsze apteczki i nie zauważył spadku życia po wejściu w nie. Wyniki uzyskiwane przez eksperta wchodzącego czasami w miny były tylko nieznacznie lepsze od wyników agenta nauczonego na podstawie tych trajektorii.

W obu scenariuszach, po osiągnięciu pewnego poziomu, zwiększanie liczby trajektorii eksperta użytych do uczenia nie polepszało wyników agenta.

4.1.5 Wnioski

W badanych scenariuszach metoda kopiowania zachowań osiąga znacznie lepsze wyniki, niż sugerowałaby literatura i uzyskuje je w ciągu ułamka czasu potrzebnego klasycznym metodom uczenia ze wzmocnieniem. Uzyskani agenci w większości przypadków zachowują się sensownie, chociaż czasem popełniają systematyczne błędy. Kopiowanie zachowań wydaje się świetnym punktem startowym dla VizDooma i wydaje się wskazane, żeby inne metody rozszerzały to podejście, zamiast je zastępować.

4.2 Agregacja zbioru danych

Podejście Agregacji zbioru danych (ang. Dataset Aggregation) [Ross et al., 2010] zostało opisane we wcześniejszym rozdziale 2.7.4. Kluczowym założeniem metody jest odpytywanie eksperta o właściwe działanie w stanach, które nie były wcześniej przez niego pokazane (i nie należą do „poprawnych" trajektorii), a które zostały odwiedzone przez agenta na skutek jego nieoptymalnego zachowania.

W rzeczywistości, dla bardziej skomplikowanych zadań, odpytywanie eksperta o decyzję dla każdego odwiedzonego przez agenta stanu jest niepraktyczne. Ocenianie wielu kolejnych stanów może być drogie i nużące dla eksperta, co może przekładać się na obniżoną jakość decyzji. Ocena dokonywana przez eksperta może też w praktyce różnić się w zależności od tego, czy ekspert napotkał dany stan podczas normalnego działania, czy podczas oceny pojedynczych, wyrwanych z kontekstu stanów.

Aby zminimalizować ten problem, konieczne jest określenie mniejszego podzbioru stanów, dla których potrzebna jest ocena eksperta.

4.2.1 Implementacja

Zastosowana implementacja jest rozszerzeniem 4.1. Pierwszym krokiem jest załadowane przygotowanych wcześniej trajektorii eksperta do pamięci agenta (zestawu danych).

Następnie agent rozpoczyna działanie, bazując na swoim aktualnym stanie wiedzy. Po wystąpieniu określonych warunków, definiujących potrzebę odpytania eksperta, działanie programu zostaje wstrzymane, a sterowanie przekazane jest do eksperta. Aby dostosować się do ograniczeń ludzkiego eksperta, po przekazaniu sterowania program przechodzi w tryb synchroniczny - przed każdą kolejną klatką czeka na reakcję eksperta. Ubocznym skutkiem tej implementacji jest pomijanie akcji „nic nie rób", która jest wykonywana dopiero po wciśnięciu dedykowanego klawisza.

Po wystąpieniu określonych warunków, definiujących koniec potrzeby odpytywania eksperta, wszystkie stany i akcje odwiedzone w trakcie danej demonstracji dodawane są do pamięci agenta (trajektoria może być dodana do pamięci z większą wagą niż początkowe prezentacje - w przeciwnym wypadku dodanie nowych danych mogłoby nie być odczuwalne). Agent aktualizuje klasyfikator akcji na podstawie rozszerzonego zestawu danych, po czym przejmuje sterowanie od eksperta i wraca do normalnego działania bazując na uaktualnionym stanie wiedzy.

Po ponownym wystąpieniu określonych warunków, kontrola może ponownie zostać przekazana ekspertowi.

4.2.2 Przekazywanie sterowania

Jednym z najważniejszych problemów jest zdefiniowanie, kiedy przekazywać sterowanie pomiędzy agentem a ekspertem. Wybór sposobu będzie decydował o tym, jak często ekspert będzie odpytywany i na ile istotna będzie uzyskana wiedza. Sprawdzone zostały trzy następujące sposoby.

Losowe przekazanie sterowania

1. Przed wykonaniem każdej akcji agent z bardzo małym prawdopodobieństwem może zdecydować o przekazaniu sterowania ekspertowi.
2. Po każdej akcji eksperta program z większym prawdopodobieństwem może zdecydować o przekazaniu sterowania do agenta.

Losowe przekazywanie sterowania jest niepraktyczną metodą - dla analizowanych problemów agent nie potrzebuje pomocy eksperta przez większość czasu, więc losowo wybrane momenty przekazania sterowania w ogromnej większości nie dostarczają istotnej informacji. Zaletą jest natomiast automatyczność decyzji - program podczas gry agenta może działać w przyspieszonym tempie.

Analiza niepewności sieci

1. Przed wykonaniem każdej akcji sprawdzana jest niepewność sieci dla danego stanu, przybliżana za pomocą wartości aktywacji funkcji softmax dla danej akcji. W przypadku wystąpienia zadanej liczby kolejnych niepewnych akcji sterowanie przekazywane jest do eksperta.
2. Po każdej akcji eksperta sprawdzana jest akcja, którą wykonałby agent. Jeżeli przez zadaną liczbę kolejnych kroków agent postąpiłby identycznie jak ekspert, to sterowanie wraca do agenta.

Analiza niepewności sieci jest skuteczniejsza niż losowe przekazywanie sterowania. Wybrane tym sposobem okna działania eksperta częściej pokrywają się z oknami niepoprawnego działania agenta. W dalszym ciągu skuteczność metody nie jest zadowalająca - przyjęta miara niepewności powoduje, że agent może przekazać sterowanie do eksperta w obliczu sytuacji, dla której więcej niż jedna akcja jest sensowna. Porównywanie akcji agenta i eksperta przez zadaną liczbę kroków jest skuteczne dla problemów z niewielką liczbą akcji, ale nieskuteczne w sytuacji, w której podobny efekt można uzyskać za pomocą różnych sekwencji kroków (przykładowo dojście do danego punktu za pomocą permutacji akcji „lewo", „prosto" i „lewo i prosto"). Podobnie jak przy losowym podejściu, dzięki automatycznemu działaniu możliwe jest działanie programu w przyspieszonym tempie podczas gry agenta.

Przyjęta miara niepewności łączy niepewność wynikającą z niewiedzy sieci i niepewność wynikającą z równorzędności akcji, podczas gdy konieczne jest analizowanie tylko pierwszej z nich. Z uwagi na zachowanie całej metody lepsza miara nie była analizowana.

Decyzja eksperta

1. Ekspert obserwuje działanie agenta i przejmuje sterowanie kiedy uzna, że agent trafił do niepożądanego stanu.

2. Kiedy ekspert uzna, że agent nie jest już w niepożądanym stanie może oddać sterowanie agentowi.

Decyzja eksperta jest najskuteczniejszą metodą i jest używana w dalszych eksperymentach. Ekspert może sam stwierdzić, kiedy działanie agenta jest niezgodne z pożądanym, maksymalizując skuteczność odpytywania eksperta. Oczywiście, ekspert musi spędzić więcej czasu obserwując działanie agenta, ale obserwacja jest dużo mniej uciążliwa (a zatem tańsza), niż prezentowanie. Problemem w niektórych sytuacjach jest możliwość rozróżnienia, kiedy agent zachowa się niepożądanie i należałoby przejąć sterowanie - w wielu sytuacjach ekspert reaguje zbyt późno, żeby demonstracja była skuteczna.

4.2.3 Techniczna implementacja

Architektura sieci neuronowej jest identyczna z architekturą zastosowaną w 4.1.

Implementacja przekazywania sterowania do eksperta w środowisku VizDoom byłaby wymagająca i czasochłonna, dlatego zastosowano znacznie prostsze, chociaż mniej eleganckie rozwiązanie.

Przy odpytywaniu eksperta o akcje program oczekuje na następny znak, który pojawi się na standardowym strumieniu wejścia programu (następny znak wpisany w konsoli). Wybrane znaki są przypisane do indeksów wybranych akcji, wpisanie nieznanego znaku powoduje wybranie akcji o indeksie 0, czyli „nic nie rób”.

```
1
2     def get_expert_action(self):
3         fd = sys.stdin.fileno()
4         old_settings = termios.tcgetattr(fd)
5         try:
6             tty.setraw(sys.stdin.fileno())
7             move = sys.stdin.read(1)
8         finally:
9             termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
10        if move == 'j':
11            return 4
12        if move == 'l':
13            return 2
14        if move == 'a':
15            return 1
16
17        if move == 'i':
18            return 1
19        if move == 'u':
20            return 5
21        if move == 'o':
22            return 3
23        return 0
```

Przy metodzie decyzji eksperta konieczne jest asynchroniczne przetwarzanie działania eksperta. Program nie może oczekiwać na działanie eksperta, ale kiedy ekspert zarządza przekazania sterowania następna akcja powinna być już wykonywana przez niego.

W tym celu wykorzystano bibliotekę PyKeyboardEvent, która umożliwia reagowanie na systemowe informacje o wciśnięciu bądź puszczeniu klawiszy klawiatury. Poniższa klasa wywołuje zadaną funkcję po wciśnięciu lub puszczeniu zadanych klawiszy.

```
1 from __future__ import print_function
```

```
from pykeyboard import PyKeyboardEventwojciech_kopec_101675.pdf


class KeyMonitor(PyKeyboardEvent):
    def __init__(self, keys, keypress_handler):
        PyKeyboardEvent.__init__(self)
        self.keypress_handler = keypress_handler
        self.keys = set(keys)

    def tap(self, keycode, character, press):
        if character in self.keys:
            self.keypress_handler(character, press)
```

Wywoływana funkcja znajduje się poniżej. Klawisz 'p' przekazuje sterowanie pomiędzy ekspertem i agentem. Klawisze ',' i '.' zwalniają i przyspieszają działanie programu podczas gry agenta.

```
    def __toggle_user_input(self, character):
        if character == 'p':
            if self.expert_mode:
                self.learn_all()
            self.expert_mode = not self.expert_mode
            print ("Expert toggled: " + str(self.expert_mode))
        elif character == '.':
            self.framerate+=5
            print ("Framerate: " + str(self.framerate))
        elif character == ',':
            self.framerate -= 5
            print ("Framerate: " + str(self.framerate))
        return True
```

4.2.4 Zachowanie

Eksperymenty były prowadzone przede wszystkim na scenariuszu "Trudne zbieranie apteczek". Początkowe trajektorie eksperta były wygenerowane zgodnie z opisem w rozdziale 4.3.

Dla każdego z badanych scenariuszy uwzględnianie fragmentów trajektorii zaprezentowanych przez eksperta w trakcie gry obniża początkowo wyniki. Na skutek nauczenia się niespójnych zachowań eksperta agent zachowuje się mniej płynnie i częściej wpada w nieskończone pętle ruchów (przykładowo obracanie się na przemian w lewo i w prawo w rogu labiryntu), co prowadzi do osiągania niższych wyników.

W scenariuszu Trudne zbieranie apteczek (ang. Health gathering supreme) głównym problemem agenta opisanego w rozdziale Kopiowanie zachowań jest nieomijanie min i celem zastosowania podejścia DAgger jest wyeliminowanie tego problemu. Za każdym razem, kiedy agent zbliża się do min ekspert przejmuje kontrolę i omija miny bądź wybiera inną ścieżkę.

Najczęściej podczas pierwszych epizodów nauki wyniki uzyskiwane przez agenta zauważalnie się obniżają, a problem wchodzenia na miny nie jest wyeliminowany. Następne epizody nauki powoli poprawiają wyniki agenta, przywracając je do poziomu wyjściowego lub nieznacznie go przewyższającego. Agent rzadziej wchodzi w miny, ale problem w dalszym ciągu pozostaje obecny.

Kolejne epizody nauki regularnie doprowadzają do przeuczenia - wyniki obniżają się, a agent często wpada w nieskończone pętle ruchów. Wchodzenie w miny nie zostaje wyeliminowane.

Wyniki metody cechują się bardzo dużą wariancją. W pojedynczych przypadkach wytrenowany agent osiąga bardzo wysokie, innym razem bardzo niskie wyniki. Najczęściej jednak po 3 epizodach gry wyniki agenta są nieznacznie wyższe niż na początku działania metody.

4.2.5 Wnioski

Dla badanych problemów DAgger nie wydaje się być skuteczny. W VizDoomie decyzje podejmowane przez ludzkiego eksperta są bardziej skomplikowane niż w Mario Cart, przedstawianym w publikacji, co, na skutek niespójności przedstawianych przez eksperta zachowań, zamiast do podwyższenia wyników agenta prowadzi do obniżania jego skuteczności. Zastosowane głębokie sieci neuronowe mogą też znacznie skuteczniej uogólniać wiedzę zdobytą podczas pierwszej prezentacji eksperta niż prostsze klasyfikatory SVM, a co za tym idzie nawet bez użycia DAggera agent potrafi znaleźć sensowne wyjście z większości sytuacji. Uzyskiwanie oceny eksperta jest uciążliwe i kosztowne.

4.3 Świadomie prezentujący ekspert

W sekcji 4.1 opisano agenta budującego klasyfikator (stan → akcja) na podstawie trajektorii zebranych podczas gry eksperta. Uzyskany agent zachowywał się sensownie, ale problem stanowiło między innymi blokowanie się w rogach labiryntu i wchodzenie na miny. Główną praktyczną wadą metody 4.2, która miała na celu zaradzenie temu, jest niespójność zachowań eksperta podczas pierwszej (ciągłej) prezentacji i zachowań podczas krótkich prezentacji podczas gry agenta oraz uciążliwość obserwacji i przejmowania sterowania od agenta w trakcie gry.

Problem wchodzenia w ściany, dla przykładu, jest łatwo zauważalny podczas obserwacji działania agenta. Oczywistym jest też powód jego występowania - ekspert, w przeciwieństwie do agenta, pamięta jak dotarł do danego stanu i znajdując się w rogu pamięta, w którą stronę powinien z niego wyjść. Badani agenci mogą pamiętać tylko kilka ostatnich odwiedzonych klatek i nie pamiętają swoich trajektorii. Dlatego klasyfikator nauczony na trajektoriach eksperta nie ma wystarczającej informacji żeby rozróżnić konieczność wychodzenia z rogu obracając się w prawo bądź w lewo.

Rozwiązaniem jest powtórne zebranie trajektorii eksperta, kładąc przy prezentacji nacisk na zachowywanie się w sposób spójny i ułatwiający klasyfikatorowi skuteczną naukę. Możliwe jest też pokazywanie rozwiązań sytuacji, które wcześniej sprawiały klasyfikatorowi problem, w celu pokazania poprawnego zachowania w danej sytuacji.

Oczywiście, takie zachowanie eksperta skutkuje uzyskiwaniem przez niego nieoptymalnych wyników, a co za tym idzie wyniki możliwe do osiągnięcia przez idealnie odwzorowującego agenta też są niższe. W praktyce różnica pomiędzy wynikami eksperta i agenta powinna się zmniejszyć dzięki świadomej prezentacji, skutkując wyższymi wynikami osiąganymi przez agenta.

4.3.1 Algorytm

Algorytm działania jest nastepujący:

1. Naucz agenta na podstawie trajektorii, dla których uzyskano najlepsze wyniki
2. Przeanalizuj działanie agenta, szukając nieoptymalnych zachowań, które mogą wynikać z niedoskonałości prezentacji eksperta
3. Wygeneruj nowe trajektorie, eliminując zauważone niedoskonałości

Przykładowymi zachowaniami mogą być:

- Problem: agent blokuje się w rogach labiryntu. Przeciwdziałanie: będąc w rogu labiryntu ekspert zawsze odwraca się w lewą stronę.

- Problem: agent nie radzi sobie, gdy przeciwnicy podejdą zbyt blisko. Przeciwdziałanie: ekspert pozwala przeciwnikom podejść do siebie przed wyeliminowaniem ich.

4.3.2 Techniczna implementacja

Techniczna implementacja jest identyczna z opisaną w rozdziale 4.1.

4.3.3 Zachowanie

Eksperymenty były prowadzone na scenariuszach Trudne zbieranie apteczek (ang. Health gathering supreme) i Obrona środka (ang. Defend the center).

W scenariuszu Obrona środka ekspert podczas świadomej prezentacji powstrzymywał się od strzelania do odległych przeciwników i świadomie preferował strzelanie do szybszych przeciwników. Świadoma prezentacja zmniejszyła liczbę niepotrzebnych strzałów nauczonego agenta.

W scenariuszu Trudne zbieranie apteczek ekspert podczas świadomej prezentacji zawsze wychodził z rogów obracając się w tę samą stronę i czasami odwracał się od tras z minami. Świadoma prezentacja prawie całkowicie wyeliminowała wpadanie w nieskończone pętle ruchów w rogach. W niektórych sytuacjach zdarzało się, że agent zawracał za to w ciasnych, ale możliwych do przejścia korytarzach - było to zachowanie wyraźnie nieoptymalne, ale bez zauważalnego wpływu na osiągane wyniki. Niestety, świadoma prezentacja nie wyeliminowała wchodzenia w miny. Wynik punktowy agenta zwiększył się istotnie po zastosowaniu świadomej prezentacji.

To, jak ważna jest świadoma prezentacja widoczne było przy zwiększaniu wielkości trajektorii eksperta użytych do nauki klasyfikatora. Dla obu scenariuszy agent nauczony na podstawie małej liczby trajektorii świadomego eksperta przewyższał agenta nauczonego na większej liczbie trajektorii nieświadomego eksperta i agenta nauczonego na mieszance trajektorii.

4.3.4 Wnioski

Dla obu scenariuszy świadoma prezentacja eksperta jest prostym i bardzo skutecznym sposobem eliminowania części oczywistych błędów popełnianych przez agenta. Dla niektórych problemów i sytuacji może wypełniać zadanie postawione przed metodą agregacji zbioru danych w wygodniejszy i bardziej naturalny sposób.

Świadoma prezentacja nie jest formalną metodą, a raczej wytyczną. Dzięki temu można ją z powodzeniem stosować w połączeniu z innymi technikami uczenia z ekspertem.

Rozdział 5

Wyniki eksperymentalne

W celu zweryfikowania postawionych hipotez przetestowane zostały następujące metody:

- Klonowanie zachowań
- Klonowanie zachowań ze świadomie prezentującym ekspertem
- Agregacja zbioru danych

Testy zostały wykonane na następujących scenariuszach:

- Obrona środka
- Trudne zbieranie apteczek

Wyniki eksperymentów są przedstawione w poniższym rozdziale.

5.1 Platforma testowa

Wszystkie testy zostały przeprowadzone na komputerze z systemem Linux, na ośmiowątkowym procesorze Intel Core i7-6700K CPU 4.00GHz, i karcie graficznej GeForce GTX 1070 wyposażonej w 8GB pamięci RAM. Do testów zostało wykorzystane oprogramowanie w następujących wersjach:

- python - 2.7.12
- tensorflow - 1.0.1 dla GPU
- VizDoom - 1.1.0rc1

5.2 Gromadzenie trajektorii eksperta

Do zbierania trajektorii eksperta przeznaczony jest dedykowany program napisany w języku python, *spectator.py*, opisany wcześniej w rozdziale 4.1.3. Program powstał poprzez modyfikację programu o tej samej nazwie dołączonego do środowiska VizDoom.

5.2.1 Konfiguracja VizDoom

Instancja VizDoom inicjalizowana jest z następującymi ustawieniami:

```
1 game.set_window_visible(True)
2 game.set_mode(Mode.SPECTATOR)
```

Tryb SPECTATOR umożliwia komputerowi obserwowanie gry człowieka - eksperta. W tym trybie gra reaguje na wejście z klawiatury i myszki, jak przy normalnej grze w Dooma, oraz umożliwia odczytywanie aktualnego stanu, ostatnio wykonanej akcji i wartości ostatnio otrzymanej nagrody. Przejście do następnego stanu jest osiągane poprzez wywołanie metody *game.advance_action()*. Tryb SPECTATOR jest synchroniczny, co oznacza że gra oczekuje na wywołanie *game.advance_action()* przed przejściem do następnego stanu (zatrzymanie przetwarzania w programie skutkuje zatrzymaniem przetwarzania w grze). Nie oznacza to jednak, że w tym trybie gra oczekuje z wykonaniem każdej klatki, aż gracz wykona jakiś ruch - brak ruchu ze strony gracza również interpretowany jest jako ruch.

```
1          game.advance_action()
2          next_state = game.get_state()
3          last_action = game.get_last_action()
4          reward = game.get_last_reward()
```

Obrazy z gry zapisywane i przetwarzane są skali szarości, a nie w kolorze. Dzięki temu zajętość pamięciowa każdego ze stanów oraz wielkość początkowych warstw uczonej sieci neuronowej jest trzykrotnie mniejsza niż kolorowego odpowiednika. Według autorów platformy VizDoom i według krótkich eksperymentów przeprowadzonych w ramach tej pracy spadek jakości agentów w stosunku do kolorowego trybu jest pomijalny.

```
1  game.set_screen_format(ScreenFormat.GRAY8)
2  game.set_screen_resolution(ScreenResolution.RES_640X480)
```

Obraz oglądany i zapisywany jest w rozdzielczości 640 na 480 pikseli. Użycie mniejszego obrazu podczas gry sprawiłoby, że mniejsze szczegóły byłyby trudne do rozróżnienia, co znacząco utrudniałoby działanie ludzkiego gracza. Obrazy są zapisywane w pełnej rozdzielczości pomimo, że użyte algorytmy wykorzystują obraz w znacznie mniejszym rozmiarze. Dzięki temu raz zebrane trajektorie eksperta mogą być przeskalowywane do wielu różnych rozmiarów z minimalną stratą jakości obrazu. Skutkiem ubocznym jest bardzo duża wielkość plików z zapisanymi trajektoriami eksperta (do 4GB dla pliku z zapisem 6 tysięcy kroków gry).

5.2.2 Zapis i odczyt trajektorii

Zapis

W każdym kroku działania programu *spectator.py* odczytywane są wartości:

- stanu początkowego
- wykonanej akcji
- stanu, w którym znalazł się agent po wykonaniu akcji
- otrzymanej nagrody
- informacji, czy dany stan jest końcowy dla danego epizodu

Te pięć wartości łączone jest w jedną krotkę, a każda krotka dodawana jest do listy krotek.

Program *spectator.py* pozwala ekspertowi odbyć zadaną liczbę epizodów, po czym serializuje listę krotek z zapisem trajektorii do pliku za pomocą biblioteki *pickle*.

Dla scenariuszy *Trudne zbieranie apteczek* i *Obrona środka* przy jednym uruchomieniu programu zbierane są trajektorie 3 epizodów. Przy poprawnej grze każdy z epizodów składa się z

około 2 tysięcy kroków, których odbycie zajmuje od jednej do półtorej minuty. Pięć minut ciągłej prezentacji pozwala na wygenerowanie danych wystarczających do nauki i jest wystarczająco krótkim czasem, żeby nie znużyć eksperta.

Zestaw trajektorii uczących

Dla każdego scenariusza (*Obrona środka* i *Trudne zbieranie apteczek*) i dla każdego rodzaju eksperta (*zwykły ekspert* i *świadomie prezentujący ekspert*) wygenerowano po 5 zbiorów trajektorii, czyli odpowiednio po 15 epizodów, każdy składający się z około 2000 kroków.

Odczyt

Program agenta otrzymuje listę plików z paczkami trajektorii do wykorzystania i informację o maksymalnej liczbie klatek, jaką ma wczytać. Program przetwarza pliki w losowej kolejności. Z każdego pliku trajektorie odczytywane są po kolei, aż limit odczytanych klatek nie zostanie osiągnięty lub aż wszystkie pliki nie zostaną przetworzone. Dane w ramach jednego pliku nie są przetwarzane w losowej kolejności, ponieważ:

- agent mógłby zdobywać dodatkowe informacje na podstawie przebytej sekwencji stanów (np. odtwarzać wartości Q)
- liczba kroków z poszczególnymi akcjami może być silnie niezbalansowana, a losowe próbkowanie trajektorii mogło by skutkować zestawem danych zupełnie pozbawionym informacji o niektórych akcjach

Odczytane klatki zapisywane są bezpośrednio do pamięci powtórek agenta, po uprzednim przeskalowaniu obrazów-stanów do zadanej rozdzielczości.

Na każdym etapie uczenia program pomija klatki, w których nie została wykonana żadna akcja. To zachowanie opiera się na założeniu, że optymalne zachowanie agenta można osiągnąć bez wykonywania ruchu „nic nie rób", a klatki z akcją „nic nie rób" pojawiające się w trajektoriach eksperta są artefaktami spowodowanymi nieoptymalną grą eksperta lub nieoptymalnym interfejsem zbierania danych eksperta.

5.3 Wykorzystany agent - implementacja

Implementacja agenta, który posłużył do wykonania eksperymentów opisywanych w tym rozdziale znajduje się w pliku *actions_estimator_tf.py*. Agent w zależności od ustawień może realizować zarówno algorytmy *kopiowania zachowań* jak i *agregacji zbioru danych*. Implementacja agenta opiera się na głębokiej sieci konwolucyjnej modelującą funkcję wyboru $stan \rightarrow akcja$.

5.3.1 Sieć neuronowa

Architektura

Opis architektury znajduje się w rozdziale 4.1.2.

Pamięć powtórek

Dla części scenariuszy (*Obrona środka*) problemem przy uczeniu jest niezbalansowanie zbioru danych (niektóre akcje występują znacznie rzadziej niż inne). Zamiast popularnej metody *oversamplingu* zastosowano rozwiązanie, w którym dla każdej z akcji przetrzymywana jest osobna

pamięć powtórek. Przy wielkości batcha n i liczbie dostępnych akcji a z każdej z kolejnych pamięci powtórek pobierana jest losowa próbka o wielkości n/a. Uzyskane podpróbki łączone są w losowej kolejności i zwracane jako ostateczna próbka. Dzięki zastosowaniu takiego rozwiązania próbki danych uczących zawierają dla mniej wykonywanych akcji bardziej zróżnicowane rekordy, zamiast kilku powtórzeń tego samego doświadczenia.

Implementacja pamięci znajduje się w pliku *replay_memory.py*.

Pozostałe ustawienia

Wagi sieci aktualizowane są za pomocą metody RMSProp *(ang. Root Mean Square Propagation)*. Prędkość uczenia *(ang. learning rate)* została eksperymentalnie ustalona na 0.00025. Sieć jest uczona za pomocą paczek danych *(ang. batch)* o wielkości 64.

5.4 Kopiowanie zachowań - wyniki

Metoda *kopiowania zachowań* stanowi punkt wyjściowy i punkt odniesienia dla pozostałych metod.

Nauka agenta sprowadza się do nauczenia klasyfikatora (sieci neuronowej) na podstawie zestawu danych treningowych, czyli zapamiętanych trajektorii eksperta.

5.4.1 Zachowanie eksperta

Podczas prezentacji ekspert zachowuje się tak, jak zachowywałby się człowiek podczas normalnej gry. Działa na podstawie aktualnie widzianego stanu i pamięci na temat poprzednio odwiedzonych stanów (np. pamięta, że poza polem widzenia agenta został niezabity przeciwnik, lub że za rogiem labiryntu zostały niezebrane apteczki). W szczególności ekspert:

- w scenariuszu *Trudne zbieranie apteczek* wraca od regionów, w których zostały pominięte wcześniej apteczki
- w scenariuszu *Trudne zbieranie apteczek* znajdując się w rogu lub na ciasnym zakręcie wychodzi z zakrętu na podstawie zapamiętanego wcześniej kierunku ruchu
- w scenariuszu *Trudne zbieranie apteczek* ignoruje apteczki, jeżeli uważa że nie opłaca się ich zbierać
- w scenariuszu *Obrona środka* wbrew optymalnej taktyce okazyjnie strzela do odległych przeciwników
- W scenariuszu *Obrona środka* odwraca się do przeciwników, którzy nie są już widoczni na ekranie

5.4.2 Wyniki

Przy testowaniu scenariusza *Obrona środka* dla każdej konfiguracji zostało nauczonych 10 agentów, z których każdy został przetestowany odgrywając 10 epizodów. Wynikowa próbka danych składa się ze 100 wyników dla każdej z konfiguracji.

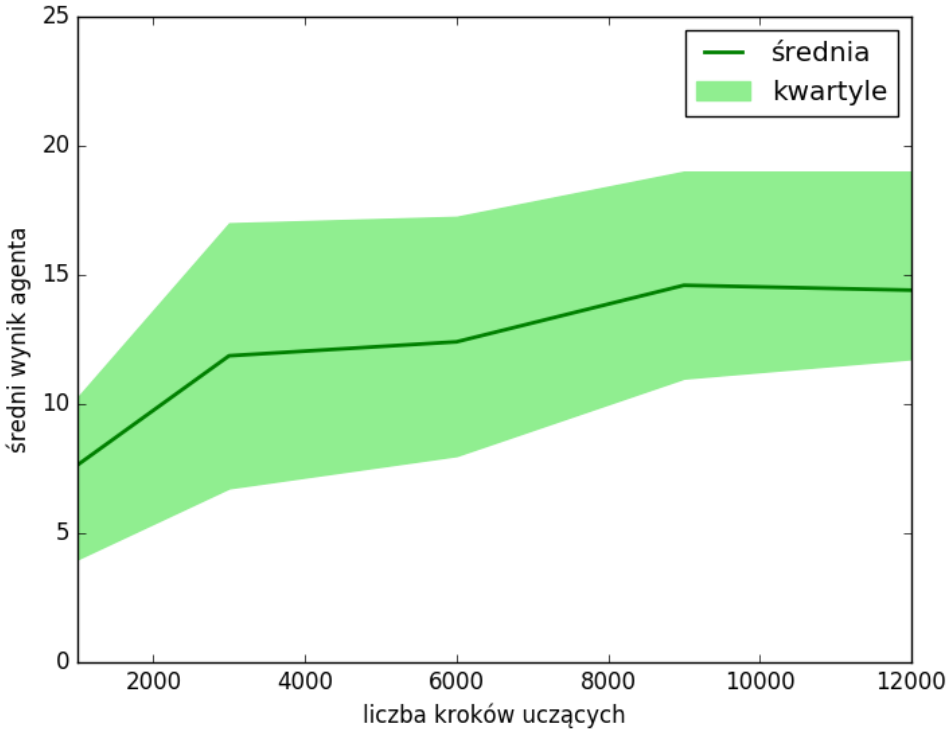
Przy testowaniu scenariusza *Trudne zbieranie apteczek* dla każdej konfiguracji zostało nauczonych od 82 do 111 agentów, z których każdy został przetestowany odgrywając 20 epizodów. Wynikowa próbka danych składa się z od 1640 do 2220 wyników dla każdej z konfiguracji.

Dla scenariusza *Trudne zbieranie apteczek* wykonano znacznie więcej powtórzeń eksperymentów niż dla *Obrony środka*, żeby upewnić się że różnice pomiędzy algorytmami są statystycznie istotne.

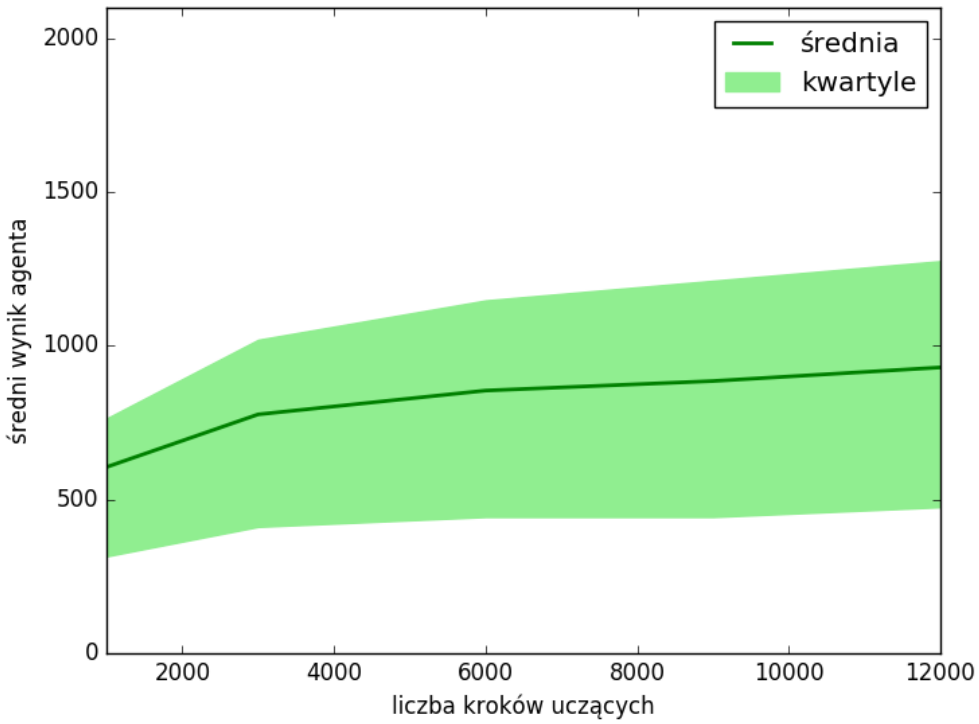
Uczenie sieci przebiega z prędkością około 100 kroków na sekundę, a jego prędkość skaluje się liniowo względem wielkości zestawu danych, tak więc nauka na podstawie trajektorii o rozmiarze 6000 kroków zajmuje około minuty, podobnie jak wykonanie 20 gier testowych.

Maksymalny wynik możliwy do osiągnięcia w scenariuszu *Obrona środka* to 25, a w scenariuszu *Trudne zbieranie apteczek* 2100. Ekspert w większości prezentacji osiąga maksymalne wyniki, ale zdarzają się też wyniki porównywalne z wynikami nauczonego agenta.

W tabelach 5.3 i 5.4 oraz na rysunkach 5.1 i 5.2 przedstawione są wyniki osiągnięte przez agenta w przeprowadzonych eksperymentach.



Rysunek 5.1: Wyniki agenta nauczonego na podstawie trajektorii *zwykłego eksperta* w zależności od liczby kroków uczących w scenariuszu *Obrona środka*



Rysunek 5.2: Wyniki agenta nauczonego na podstawie trajektorii *zwykłego eksperta* w zależności od liczby kroków uczących w scenariuszu *Trudne zbieranie apteczek*

Kroki uczące	1 kwartyl	Średnia	3 kwartyl	Odchylenie std.	Powtórzenia	Czas
1000	4	7.64	10.25	5.16	100	70s
3000	6.75	11.87	17	6.21	100	1.5min
6000	8	12.41	17.25	5.88	100	2min
9000	11	14.6	19	6.15	100	2.5min
12000	11.75	14.41	19	6.54	100	3min

Rysunek 5.3: Wyniki agenta nauczonego na podstawie trajektorii *zwykłego eksperta* w zależności od liczby kroków uczących w scenariuszu *Obrona środka*.

Kroki uczące	1 kwartyl	Średnia	3 kwartyl	Odchylenie std.	Powtórzenia	Czas
1000	316	605.8	764	393.1	1640	70s
3000	412	777.1	1020	485.01	1720	1.5min
6000	444	854.6	1148	521.79	2140	2min
9000	444	885.6	1212	527.5	2220	2.5min
12000	476	929.9	1276	551.93	1880	3min

Rysunek 5.4: Wyniki agenta nauczonego na podstawie trajektorii *zwykłego eksperta* w zależności od liczby kroków uczących w scenariuszu *Trudne zbieranie apteczek*.

5.4.3 Zachowanie agenta

Trudne zbieranie apteczek

Agent wykazuje pełną świadomość labiryntu i otoczenia - poprawnie porusza się po labiryncie, nie wchodzi w ściany i nie blokuje się w ślepych zaułkach. Poprawnie identyfikuje apteczki i zazwyczaj zbiera je zgodnie z oczekiwaniami. Agent bywa niedokładny, na przykład przechodzi

zbyt daleko od apteczki by ją zebrać. Agent nie omija min, ale zazwyczaj nie wchodzi w nie z premedytacją. Agentowi zdarza się zablokować pomiędzy dwoma równorzędnymi decyzjami - agent wykonuje wtedy „rezonujące" ruchy (na zmianę wykonuje akcje „lewo" i „prawo" bez ruszania się z miejsca). Najczęściej zdarza się to, gdy agent zajdzie do rogu i nie może zdecydować, w którą stronę powinien się obracać, żeby z niego wyjść.

Agent podczas poruszania się prosto często rozgląda się w lewo i w prawo - jest to artefakt i skutek uboczny wynikający z faktu, że na względnie otwartej przestrzeni wszystkie ruchy są potencjalnie dobre i jedynie szczegóły stanu, takie jak np. rozlokowanie apteczek decyduje o tym, gdzie najbardziej opłaca się udać. W praktyce jest to bardzo pomocne zachowanie, ponieważ przy rozglądaniu się agent może zauważyć korzystne do odwiedzenia obszary.

Polityka zbierania apteczek przez agenta jest najczęściej wystarczająco skuteczna, ale daleka od optymalnej. Agent często skupia się na arbitralnie wybranej apteczce, ignorując apteczki które są bliższe, ale nie ustawione centralnie przed nim. Agent często wybiera też pojedyncze apteczki zamiast zyskownych skupisk apteczek.

Obrona środka

Agent rozumie zasady rządzące środowiskiem i zachowuje się poprawnie. Popełnia sporadyczne błędy ignorując przeciwników albo strzelając w pustą przestrzeń. Agent bywa niedokładny, strzelając obok przeciwników. Strategia decydowania o strzelaniu bądź pozostawieniu danego przeciwnika nie jest optymalna. Agent regularnie traci zainteresowanie przeciwnikami, którzy po początkowym nietrafieniu agenta znaleźli się po stronie przeciwnej do obrotowego ruchu agenta. Agent zazwyczaj potrafi rozpoznać, że jest atakowany i obraca się szybko szukając bezpośredniego zagrożenia. Niestety często reakcja występuje zbyt późno.

W zachowaniu agenta nie ma wyraźnych błędów taktycznych, a jedynie niedokładności wyglądające na błędy klasyfikatora.

5.4.4 Analiza i wnioski

Nauczony agent mimo popełnianych błędów zachowuje się poprawnie. Jego zachowanie jest dalekie od optymalnego, ale udało mu się dobrze opanować wszystkie reguły rządzące środowiskiem, w którym się znajduje, oprócz zrozumienia idei min.

Biorąc pod uwagę, jak mało czasu i danych potrzebne było do wytrenowania agenta i porównując jego zachowanie i czas uczenia z zachowaniem i czasem uczenia agenta opisanego w pracy [Kempka et al., 2016] można stwierdzić, że metoda *kopiowania zachowań* jest bardzo potężnym narzędziem.

W scenariuszu *Trudne zbieranie apteczek* agent nie zwraca zazwyczaj uwagi na miny, a wejście w kilka min pod rząd jest najczęstszą przyczyną śmierci agenta. Konstrukcja scenariusza jest taka, że nawet przy mniej efektywnej taktyce zbierania apteczek agent mógłby przeżyć cały epizod, dlatego w praktyce częstotliwość wchodzenia w miny (i w mniejszym stopniu zakleszczania się w rogu labiryntu) jest czynnikiem determinującym średnie wyniki agenta. Warto zauważyć, że tego problemu nie udało się również wyeliminować w pracy [Kempka et al., 2016].

5.5 Świadomie prezentujący ekspert - wyniki

Podejście *kopiowania zachowań ze świadomie prezentującym ekspertem* nie jest podejściem algorytmicznym. Metoda jest identyczna z *kopiowaniem zachowań*, a jedyną różnicę stanowi za-

chowanie eksperta podczas generowania trajektorii uczących. Cały przebieg eksperymentu jest identyczny z opisanym w punkcie 5.4.

5.5.1 Zachowanie eksperta

Podczas prezentacji ekspert gra tak, żeby na podstawie jego trajektorii mógł zostać utworzony wewnętrznie spójny zestaw danych do nauki. Stara się nie działać na podstawie pamięci, żeby uzasadnienie każdej decyzji mogło zostać wywnioskowane tylko i wyłącznie na podstawie dostępnego klasyfikatorowi stanu. Ekspert dokłada dodatkowych starań, żeby jego działania były spójne, mimo upływającego czasu i wykonywania powtarzalnych czynności. Ekspert generuje i rozwiązuje sytuacje, w których wstępnie nauczony agent nie zachowywał się poprawnie. W szczególności:

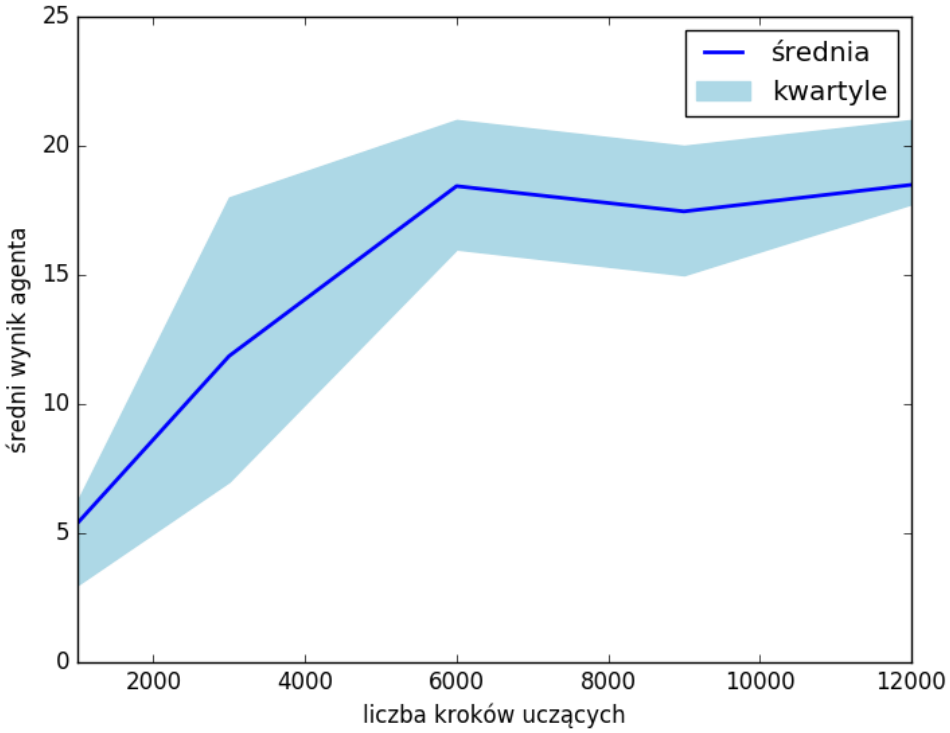
- w scenariuszu *Trudne zbieranie apteczek* znajdując się w rogu lub na ciasnym zakręcie zawsze wychodzi z niego obracając się w tę samą stronę
- w scenariuszu *Trudne zbieranie apteczek* znajdując się przed miną nie wymija jej, ale w miarę możliwości odwraca się i szuka innej drogi
- w scenariuszu *Obrona środka* pozwala przeciwnikom podejść bliżej (klatki z „czekaniem" nie są brane pod uwagę), żeby zaprezentować zachowanie w otoczeniu przeciwników
- W scenariuszu *Obrona środka* nie odwraca się do zapamiętanych przeciwników, którzy nie są widoczni na ekranie

5.5.2 Wyniki

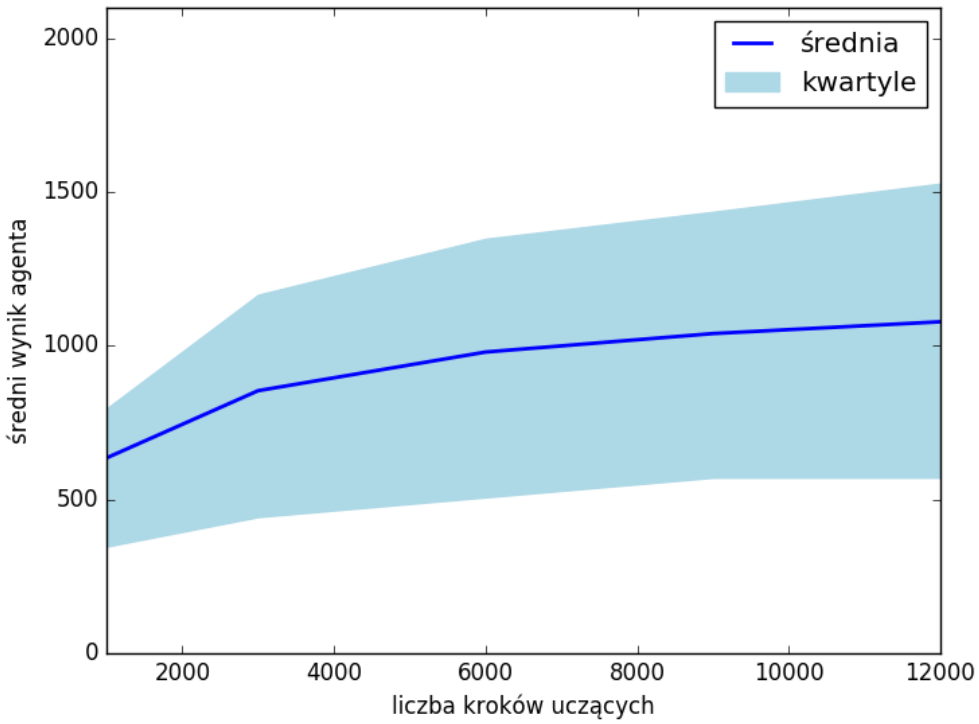
Przy testowaniu scenariusza *Obrona środka* dla każdej konfiguracji zostało nauczonych 10 agentów, z których każdy został przetestowany odgrywając 10 epizodów. Wynikowa próbka danych składa się ze 100 wyników dla każdej z konfiguracji.

Przy testowaniu scenariusza *Trudne zbieranie apteczek* dla każdej konfiguracji zostało nauczonych od 82 do 111 agentów, z których każdy został przetestowany odgrywając 20 epizodów. Wynikowa próbka danych składa się z od 1620 do 1720 wyników dla każdej z konfiguracji.

W tabelach 5.7 i 5.8 oraz na rysunkach 5.5 i 5.6 przedstawione są wyniki osiągnięte przez agenta w przeprowadzonych eksperymentach.



Rysunek 5.5: Wyniki agenta nauczonego na podstawie trajektorii *świadomie prezentującego eksperta* w zależności od liczby kroków uczących w scenariuszu *Obrona środka*



Rysunek 5.6: Wyniki agenta nauczonego na podstawie trajektorii *świadomie prezentującego eksperta* w zależności od liczby kroków uczących w scenariuszu *Trudne zbieranie apteczek*

Kroki uczące	1 kwartyl	Średnia	3 kwartyl	Odchylenie std.	Powtórzenia	Czas
1000	3	5.39	6.25	3.89	100	70s
3000	7	11.86	18	6.34	100	1.5min
6000	16	18.44	21	4.43	100	2min
9000	15	17.46	20	4.22	100	2.5min
12000	17.75	18.49	21	3.61	100	3min

Rysunek 5.7: Wyniki agenta nauczonego na podstawie trajektorii *świadomie prezentującego eksperta* w zależności od liczby kroków uczących w scenariuszu *Obrona środka*.

Kroki uczące	1 kwartyl	Średnia	3 kwartyl	Odchylenie std.	Powtórzenia	Czas
1000	348	635.4	796	410.54	1660	70s
3000	444	854.2	1166.2	510.7	1620	1.5min
6000	508	979.9	1348.2	564.33	1660	2min
9000	572	1040	1436	570.56	1700	2.5min
12000	572	1078.4	1528	590.68	1720	3min

Rysunek 5.8: Wyniki agenta nauczonego na podstawie trajektorii *świadomie prezentującego eksperta* w zależności od liczby kroków uczących w scenariuszu *Trudne zbieranie apteczek*.

5.5.3 Zachowanie agenta

Trudne zbieranie apteczek

Agent zachowuje się lepiej, niż odpowiednik nauczony na podstawie normalnych trajektorii.

Problem blokowania się w rogach został w praktyce wyeliminowany. Jedynie w pojedynczych przypadkach agentowi zdarza się zablokować w podobnych, ale nie identycznych sytuacjach i niejednokrotnie agentowi udaje się samemu z nich wydostać.

Mimo wysiłków eksperta nie udało się w zauważalny sposób zmniejszyć problemu wchodzenia w miny.

Obrona środka

Agent zachowuje się lepiej, niż odpowiednik nauczony na podstawie normalnych trajektorii. Osiągane przez niego wyniki znacznie wzrosły, a ich wariancja wyraźnie zmalała, co widać wizualnie w grze agenta. Agent rzadko popełnia wyraźne błędy, a jego niedoskonałości wynikają raczej z suboptymalnego zachowania - decyzje, czy dany przeciwnik powinien zostać zastrzelony czy powinno się go na razie zostawić nie zawsze są podejmowane poprawnie, ale jest to kwestia „dostrojenia" agenta.

5.5.4 Analiza i wnioski

Podejście *kopiowania zachowań ze świadomie prezentującym ekspertem* zgodnie z oczekiwaniami okazuje się skutecznym narzędziem, któremu udało się wyeliminować większość problemów metody bazowej. Eksperymenty potwierdzają hipotezę, że źródłem większości błędów metody *kopiowania zachowań* jest nieumiejętność klasyfikatora do rozróżnienia pewnych klas sytuacji na skutek braku pełnej informacji dostępnej ekspertowi, ale niedostępnej agentowi.

Jedyny problem, którego nie udało się wyeliminować, to wchodzenie w miny w scenariuszu *Trudne zbieranie apteczek*. Zastosowana metoda jest w stanie pokazać, jak agent powinien się zachować w określonej sytuacji, ale nie dostarcza wystarczających narzędzi, żeby pokazać że dane zachowanie jest niepożądane.

5.6 Agregacja zbioru danych - wyniki

Aby uruchomić metodę agregacji zbioru danych konieczne jest uruchomienie agenta z niezerowym parametrem *epochs*.

Na początku działania algorytmu agent jest inicjalizowany w identyczny sposób jak w metodzie *klonowania zachowań*, na zestawie trajektorii *świadomie prezentującego eksperta* o wielkości 6000 kroków. Następnie agent odbywa 3 iteracje metody agregowania zbioru danych. Każda iteracja to odegranie jednego epizodu o długości 2000 kroków, podczas których ekspert może w dowolnym momencie przejąć kontrolę nad agentem i poprawić jego zachowanie. Kroki wykonane przez eksperta są dodawane do *pamięci powtórek* 5 razy w celu zwiększenia wagi decyzji eksperta. Po zakończeniu każdego epizodu wagi sieci neuronowej są resetowane, a cała sieć jest uczona od początku na podstawie rozszerzonego zestawu danych (tzn. na podstawie początkowych trajektorii uczących i kroków wykonanych podczas gry przez eksperta). Po nauczeniu sieci wykonywanych jest 20 epizodów testowych.

Przejęcie kontroli nad agentem odbywa się poprzez wciśnięcie klawisza *p*. Po przejęciu kontroli środowisko przechodzi w synchroniczny tryb gry, każdorazowo czekając na następny ruch gracza. Ekspert kontrolujący agenta może przy użyciu dedykowanych klawiszy wykonywać dowolną kombinację ruchów, a po zakończeniu prezentacji oddać kontrolę agentowi ponownie używając klawisza *p*.

Testy zostały przeprowadzone wyłącznie na scenariuszu *Trudne zbieranie apteczek*, ponieważ zachowanie agenta w scenariuszu *Obrona środka* nie przejawiało żadnych wyraźnych błędów, za

pomocą poprawy których można było by osiągnąć poprawę w stosunku do wyników metody *klonowania zachowań* z zestawem trajektorii *świadomie prezentującego eksperta*.

Eksperyment został powtórzony 10 razy. Już po tej niewielkiej liczbie powtórzeń tendencja zachowania wynikowego agenta i osiąganych przez niego wyników była wyraźna, dlatego dalsze czasochłonne eksperymenty nie były kontynuowane. Testy z wykorzystaniem pozostałych technik przekazywania kontroli do eksperta nie zostały przeprowadzone, ponieważ zastosowanie innej metody nie wyeliminowałoby problemów badanego algorytmu.

5.6.1 Zachowanie eksperta

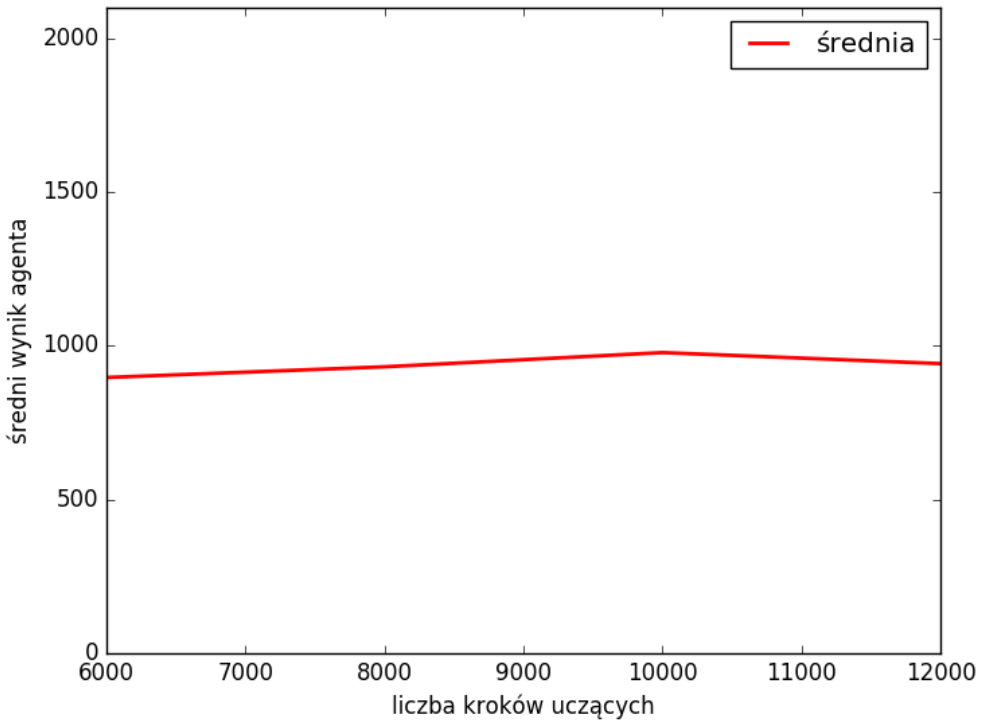
Ekspert skupia się na wyeliminowaniu problemu wchodzenia w miny. Ekspert przejmuje kontrolę gdy agent zachowuje się, jakby miał popełnić błąd. Ekspert sporadycznie reaguje na nieoptymalne, ale nie błędne zachowanie agenta. Wielokrotnie ekspert nie potrafi zareagować na działanie agenta wystarczająco szybko, żeby uniknąć popełnienia błędu. Ponieważ ekspert zawsze przejmuje kontrolę w sytuacjach awaryjnych, jego zachowania mogą być odmienne od zachowań, które przejawiałby w identycznej sytuacji podczas normalnej gry. W szczególności ekspert:

- znajdując się przed miną nie wymija jej, ale w miarę możliwości odwraca się i szuka innej drogi
- jeżeli agent zapętli się na niewielkim fragmencie labiryntu skierowuje go do innych rejonów
- pomaga wychodzić z krótkich pętli zachowań (np. zablokowanie w rogu) w optymalnym kierunku
- sporadycznie nakierowuje agenta na konkretne apteczki, jeżeli w danej sytuacji jego polityka wyboru apteczek jest błędna
- wielokrotnie przejmuje kontrolę nad agentem zbyt późno, żeby zapobiec wejściu w minę

5.6.2 Wyniki

W tabeli 5.10 i na rysunku 5.9 przedstawiono średnie wyniki uzyskiwane przez metodę *agregacji zbioru danych* podczas eksperymentów. Punktem startowym są wyniki uzyskane za pomocą metody *klonowania zachowań*, bez dodatkowych działań eksperta. W kolejnych wierszach tabeli przedstawione są średnie wyniki testów w kolejnych epizodach nauki. Ocenione kroki oznaczają orientacyjną liczbę kroków, które ocenił ekspert (wliczając to początkowe trajektorie uczące). Obejrzane kroki to liczba kroków wykonanych podczas nauki (wliczając to początkowe trajektorie uczące i kroki agenta, które obserwował ekspert). Zmiana wyniku i odchylenie standardowe wyniku oznaczają średnią i odchylenie standardowe zmiany wyniku testowego w stosunku do wyników testowych sprzed 1 epizodu.

Uczenie sieci na podstawie 6000 kroków trajektorii trwa około minuty, podobnie jak wykonanie 20 gier testowych. Każdy epizod gry agenta trwa około dwie minuty, ponieważ przejęcie kontroli nad agentem i decydowanie o optymalnym ruchu dla danego stanu jest w praktyce wolniejsze niż normalna, płynna gra. W sumie każda iteracja metody *agregacji zbioru danych* trwa około 4 minut.



Rysunek 5.9: Wyniki agenta nauczonego za pomocą metody *agregacji zbioru danych* w zależności od liczby kroków uczących w scenariuszu *Trudne zbieranie apteczek*

Epizody	Ocenione k.	Obejrzane k.	Wynik	Zmiana wyn.	Odch. std. Zmiany	Czas
0	6000	6000	897.15	0	0	2min
1	6100	8000	931.9	34.75	134.56	6min
2	6200	10000	978.16	81.01	184.24	10min
3	6300	12000	942.29	45.14	138.56	14min

Rysunek 5.10: Wyniki agenta nauczonego za pomocą metody *agregacji zbioru danych* w zależności od liczby kroków uczących w scenariuszu *Trudne zbieranie apteczek*.

5.6.3 Zachowanie agenta

Dla istotnej części powtórzeń eksperymentu wizualna ocena zachowania agenta jest odwrotna od oceny wynikającej ze zmiany wyników. Już po pierwszym epizodzie zaczynają się pojawiać objawy przeuczenia: agent często „wpada w rezonans" na rozwidleniach i w rogach labiryntu (na zmianę wykonuje akcje „lewo" i „prawo" bez ruszania się z miejsca), w stopniu znacznie większym niż w metodzie bazowej, oraz wielokrotnie utyka w jednym, małym obszarze labiryntu (np. wybiera na każdym rozwidleniu skręt w prawo i wraca po chwili do punktu wyjścia).

Chwilami można zaobserwować częściowe omijanie min, najczęściej jednak ten sam agent po kilku wzorowych ominięciach min wchodzi bez zawahania prosto w następną. Zdarza się też, że agent postawiony przed miną, przy braku jakichkolwiek apteczek w polu widzenia po chwili „rezonującego" wahania wchodzi z premedytacją prosto w tę minę. Po zakończeniu uczenia agent regularnie zacina się w losowych miejscach labiryntu, co nie zdarza się raczej na początku uczenia.

Kolejne iteracje metody w średnim przypadku prowadzą do zwiększenia wyników agenta, jednakże wzrost wyników w stosunku do przejrzanej liczby klatek wypada znacznie gorzej niż wzrost

wyników metody *kopiowania zachowań* przy takim samym zwiększeniu liczby klatek. Jeszcze gorzej wypada porównanie wzrostu wyników w stosunku do czasu trwania algorytmu.

5.6.4 Analiza i wnioski

Ekspert przejmuje kontrolę w tylko w sytuacjach awaryjnych, a jego decyzje muszą być dołączone do zbioru danych z większą wagą, żeby wywarły jakikolwiek wpływ na zachowanie agenta. Z pierwszego powodu zachowanie eksperta podczas iteracji metody może zasadniczo różnić się od zachowania które ekspert przejawiałby znajdując się w analogicznej sytuacji podczas normalnej, ciągłej gry.

Przy tak małym zestawie danych uczących zachowania niespójne z początkowymi trajektoriami uczącymi sprawiają, że agent silnie generalizuje „poprawki" eksperta, co pogarsza jego zachowanie w sytuacjach, w których zachowywał się wcześniej poprawnie. W praktyce metoda rozdmuchuje problemy *klonowania zachowań*, które miała eliminować.

Proces nauki jest długi i znacznie bardziej obciążający dla eksperta niż zbieranie trajektorii, z przeplatającymi się momentami oczekiwania na zakończenie przetwarzania i momentami skupionej kontroli zachowań agenta - dlatego czas trwania całego algorytmu, a nie kroki wykonane podczas nauki przez agenta powinny być przedmiotem porównania. Z punktu widzenia eksperta poświęcenie tego samego czasu na generowanie kolejnych trajektorii prezentacyjnych jest znacznie mniej obciążające niż obserwowanie agenta i rozważanie w każdym momencie, czy agent jest na skraju popełnienia błędu i konieczna jest reakcja.

Średni wynik uzyskiwany przez agenta w kolejnych iteracjach metody rośnie, ale wzrost wyników jest wolniejszy niż przy nauce z pomocą innej metody przy analogicznym zwiększaniu liczby kroków uczących. Znaczny rozrzut wyników sprawia też, że *agregację zbioru danych* trudno uznać za solidną.

5.7 Zestawienie wyników

Jak pokazały eksperymenty, zachowanie agentów nauczonych za pomocą każdej z metod wskazuje na zrozumienie podstawowych zasad i zależności rządzących środowiskiem w którym się znajdują. Każdy z uzyskanych agentów był w stanie wypełniać w zadowalającym stopniu stawiane przed nim zadania.

W tabelach 5.11 i 5.12 przedstawiono zestawienie wyników poszczególnych metod dla scenariuszy *Obrona środka* i *Trudne zbieranie apteczek*. W tabeli 5.12 kolumna „Wynik" oznacza średni wynik uzyskany przez agenta w przeprowadzonym eksperymencie, „Gen. trajektorii" oznacza czas potrzebny ekspertowi na wygenerowanie początkowych trajektorii uczących z daną liczbą kroków, „Algorytm" oznacza czas trwania samego algorytmu uczenia łącznie z testami skuteczności, a „Czas" to suma czasów z tych dwóch kolumn.

Konfiguracja	Średni wynik
Zwykły ekspert 6000 kroków	12.41
Prezentujący ekspert 6000 kroków	18.44
Zwykły ekspert 12000 kroków	14.41
Prezentujący ekspert 12000 kroków	18.49

Rysunek 5.11: Porównanie wyników poszczególnych metod dla scenariusza *Obrona środka*.

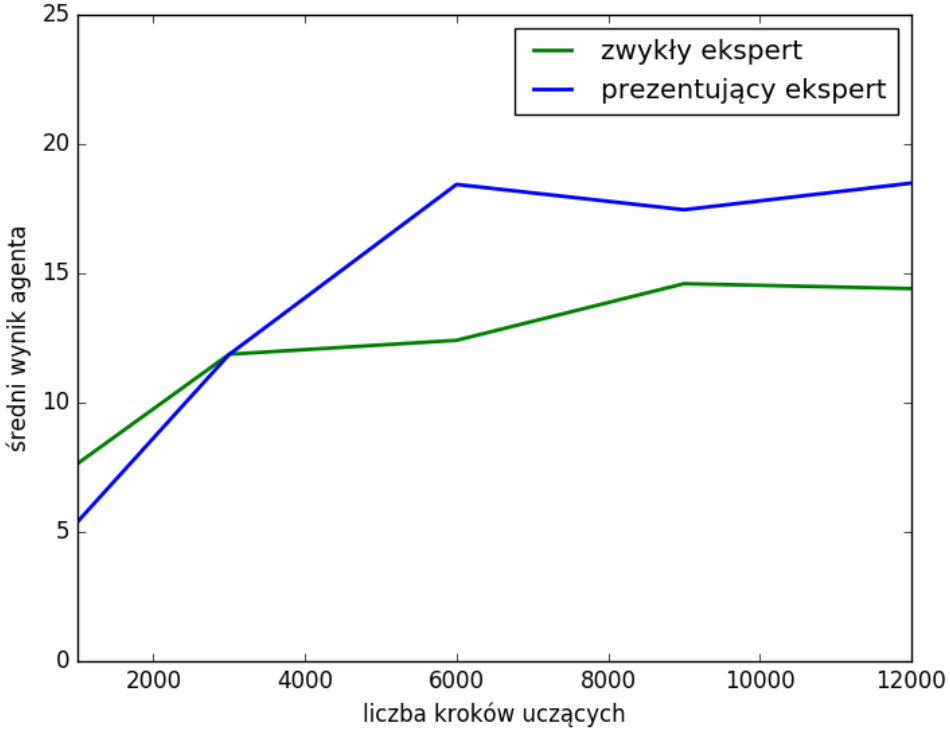
Konfiguracja	Wynik	Gen. trajektorii	Algorytm	Czas
Zwykły ekspert 12 000 kroków	929.9	10min	3min	13min
Prezentujący ekspert 12 000 kroków	1078.4	10min	3min	13min
Agregacja zbioru danych 12 000 kroków*	942.29	5min	14min	19min
Prezentujący ekspert 12 000 kroków **	979.9	10min	4min	14min
Prezentujący ekspert 18 000 kroków **	1078.4	15min	5min	20min
Q-learning 300 000 kroków***	1000	0min	9h	9h
Q-learning 1 000 000 kroków***	1300	0min	29h	29h

Rysunek 5.12: Porównanie wyników i czasów poszczególnych metod dla scenariusza *Trudne zbieranie apteczek*.

∗ Zgodnie z konfiguracją z rozdziału 5.6.2: trajektoria początkowa o długości 6000 kroków i 6000 obejrzanych kroków.
∗∗ Przy założeniu wykorzystania 6000 kroków zwykłego eksperta, przeanalizowania błędów agenta i nauce od początku na podstawie trajektorii prezentującego eksperta.
∗∗∗ Przybliżone wartości wyników zaprezentowanych w pracy [Kempka et al., 2016].

Obrona środka

Przewaga metody ze *świadomie prezentującym ekspertem* nad bazowym *kopiowaniem zachowań* jest szczególnie widoczna w scenariuszu *Obrona środka*: „świadomie" trenowany agent osiąga zdecydowanie wyższe średnie wyniki obarczone znacznie mniejszą wariancją, a jego wyniki stabilizują się na najwyższym poziomie już po 6000 kroków, w przeciwieństwie do 9000 kroków metody bazowej. Porównanie wyników zaprezentowane jest na rysunku 5.13.

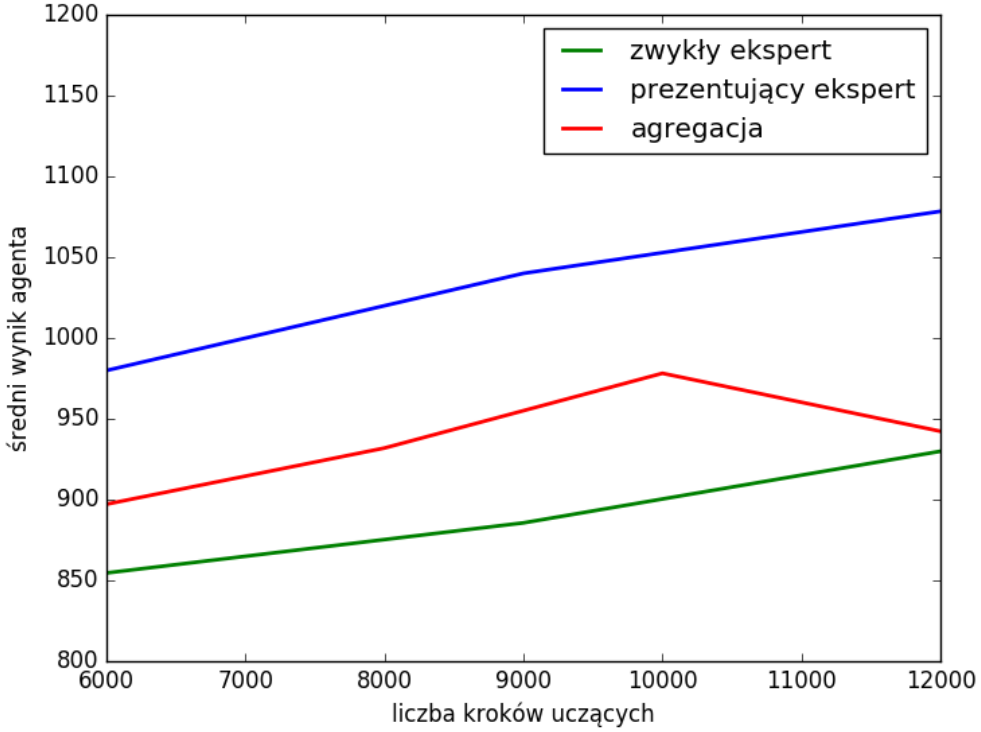


Rysunek 5.13: Porównanie wyników poszczególnych metod dla scenariusza *Obrona środka*

Z ostatecznego porównania wykluczono metodę *agregacji zbioru danych* ponieważ, jak wspomniano w dedykowanym jej podrozdziale, agent uzyskany za pomocą *świadomie prezentującego eksperta* nie przejawia żadnych zachowań, które *agregacja* miała poprawiać. Wstępne eksperymenty pokazywały też niezadowalające osiągi tej metody i problemy analogiczne jak dla scenariusza *Trudne zbieranie apteczek*.

Trudne zbieranie apteczek

W tabeli 5.12 i na rysunku 5.14 przedstawiono porównanie wyników badanych metod z wynikami uzyskanymi przez autorów środowiska VizDoom za pomocą tradycyjnych metod uczenia ze wzmocnieniem.



Rysunek 5.14: Porównanie wyników poszczególnych metod dla scenariusza *Trudne zbieranie apteczek*

Prezentujący ekspert występuje w zestawieniu trzykrotnie: pierwszy wpis traktuje metodę jako osobny i samodzielny przebieg. Kolejne wpisy zakładają, że najpierw konieczne było nauczenie agenta za pomocą *zwykłego eksperta* (6000 kroków). Zachowanie takiego agenta zostało przeanalizowane, a następnie *prezentujący ekspert* przeprowadził kolejną prezentację (6000 lub 12000 kroków) mając na uwadze błędy popełniane przez pierwszego agenta.

Najwyższy średni wynik, ze sporym zapasem, osiąga tradycyjny *Q-learning*, zaimplementowany i zaprezentowany przez autorów pracy [Kempka et al., 2016]. Ta metoda jest jednak, jak wielokrotnie wspomniano wcześniej, bardzo wymagająca czasowo: ostateczny wynik uzyskany jest po 29 godzinach, a rezultaty porównywalne z badanymi w tej pracy osiągane są dopiero po około 9 godzinach nauki.

Spośród metod *uczenia przez demonstrację* wyraźną przewagę ma *klonowanie zachowań ze świadomie prezentującym ekspertem*, a wygenerowane tym sposobem agenty zachowują się też

najlepiej przy ocenie wizualnej. Metoda *agregacji zbioru danych* ma średni wynik nieznacznie lepszy niż bazowe *klonowanie zachowań*, ale ogromna wariancja uzyskiwanych wyników, jak i znacznie większe wykorzystanie eksperta stawia ją na ostatnim miejscu.

Żadnej z metod, łącznie z *Q-learningiem* nie udało się wyeliminować największego problemu z zachowaniem agenta, czyli wchodzenia w miny.

5.7.1 Uczenie przez demonstrację a Q-learning

Jak wyraźnie widać w tabeli 5.12, bezpośrednie porównywanie metod *uczenia przez demonstrację* z *Q-learningiem* mija się z celem. *Q-learning* osiąga wyraźnie lepsze wyniki, ale nauka za jego pomocą zajmuje ponad 100 razy więcej czasu. Oznacza to, że dla badanego problemu *uczenie przez demonstrację* spełniło swoje zadanie: pozwoliło wygenerować przyzwoicie zachowującego się agenta w ułamku czasu potrzebnego na nauczenie tradycyjnymi metodami uczenia ze wzmocnieniem jego odpowiednika.

Na szczególną uwagę zasługuje czas, jaki jest potrzebny na wytrenowanie agentów. Wykorzystanie 10 minut czasu eksperta pozwala zaoszczędzić 30 godzin czasu maszynowego nauki. Oczywiście, w niektórych przypadkach, przy wystarczających zasobach, 29 godziny automatycznej nauki może być korzystniejsze i tańsze niż wykorzystanie ludzkiego eksperta przez 10 minut. Trzeba jednak zauważyć, że wdrożenie każdego praktycznego zastosowanie wymaga wielu iteracji poprawek i konfiguracji. Przy wykorzystaniu *uczenia przez demonstrację* czas zbierania trajektorii eksperta jest jedno lub kilku razowy. Na raz zebranych trajektoriach eksperta można wielokrotnie przeprowadzać relatywnie szybkie eksperymenty z wykorzystaniem innych parametrów i metod. W przypadku *Q-learningu* każda zmiana wymaga każdorazowo wielu powtórzeń wielogodzinnych testów.

Z drugiej strony, zachowanie uzyskanych agentów wskazuje, że dalsze poprawianie ich za pomocą metod *uczenia przez demonstrację* może być trudne, a uzyskanie zauważalnie lepszych wyników niemożliwe. Przejawiane problemy, czyli przede wszystkim niedokładność agentów i wchodzenie w miny, powinny być dużo łatwiejsze do wyeliminowania z użyciem uczenia ze wzmocnieniem, dlatego obiecującym rozwiązaniem jest połączenie obu algorytmów, co udało się skutecznie zrobić autorom pracy [Hester et al., 2017].

Rozdział 6

Wnioski i perspektywy rozwoju

Celem pracy było zbadanie skuteczności *uczenia przez demonstrację* na podstawie informacji obrazowej w środowisku 3D. Badania zostały zrealizowane na przykładzie popularnej gry 3D Doom, z wykorzystaniem platformy VizDoom.

Dzięki eksperymentom przeprowadzonym na najtrudniejszych scenariuszach zaproponowanych w ramach platformy VizDoom wykazano, że oparte na głębokich sieciach neuronowych agenty nauczone przy pomocy metod *uczenia przez demonstrację* uzyskują wyniki zbliżone do wyników uzyskanych za pomocą *Q-learningu*, przy ponad stukrotnie krótszym czasie nauki. Uzyskane agenty przejawiają wysoką świadomość środowiska 3D w którym się znajdują, poprawnie rozpoznając i zachowując się w stosunku do ruchomych i nieruchomych obiektów oraz w stosunku do przeciwników. Zachowanie agentów pozwala na skutecznie, chociaż nie bezbłędne, wykonywanie stawianych przed nimi zadań, a wizualna i punktowa ocena ich działań jest porównywalna z osiągami agentów uzyskanym za pomocą *Q-learningu*.

W ramach pracy skutecznie zaimplementowano i przetestowano metodę *klonowania zachowań (ang. behavioral cloning)*, która pozwoliła na nauczenie skutecznych agentów bez konieczności interakcji ze środowiskiem, na podstawie tylko kilkuminutowej prezentacji pożądanego zachowania przez ludzkiego eksperta.

Następnie zaimplementowano i przetestowano metodę *agregacji zbioru danych (ang. Dataset Aggregation, DAgger)*, której zachowanie i uzyskane wyniki są znacznie gorsze od oczekiwanych. Za możliwą przyczynę tego zachowania wskazano rozbieżność pomiędzy dystrybucją zachowania eksperta podczas początkowej fazy prezentacji i późniejszej oceny działań agenta w trakcie nauki. Przy wykorzystaniu ludzkiego eksperta do nauki nietrywialnych zadań niezgodność zachowań w obu fazach może być trudna do wyeliminowania, co obniża przydatność metody *agregacji zbioru danych* dla praktycznych zastosowań.

Ostatecznie zbadano i eksperymentalnie wykazano, że przy *uczeniu przez demonstrację* świadoma prezentacja eksperta, mająca na uwadze ograniczenia percepcyjne uczonego agenta, może skutecznie poprawić uzyskiwane przez niego wyniki. Agent nauczony na podstawie świadomej prezentacji wyrównuje wyniki bazowej metody *kopiowania zachowań* przy dwa razy mniejszej liczbie kroków uczących i osiąga zacznie lepsze i bardziej stabilne wyniki.

Agent uczony przez 4 minuty na podstawie 10 minutowej, świadomej prezentacji eksperta prześciga agenta uczonego przez 10 godzin za pomocą *Q-learningu* i osiąga wyniki zbliżone do wyników agenta uczonego przez 30 godzin.

Perspektywy rozwoju

Uzyskane agenty prezentują wysoki poziom skuteczności i świadomości otoczenia, ale w ramach żadnej z metod nie udało się wyeliminować wszystkich nieprawidłowych zachowań.

Jednym z obiecujących kierunków rozwoju jest znalezienie skutecznego sposobu umożliwienia ekspertowi oznaczania wybranych zachowań jako nieprawidłowe i generowanie na tej podstawie informacji uczących dla agenta.

Poprawienie drobnych nieprawidłowości i niedoskonałości agentów za pomocą metod *uczenia przez demonstrację* jest trudne do zrealizowania, dlatego innym kierunkiem rozwoju, zaproponowanym niedawno przez innych badaczy w pracy [Hester et al., 2017], jest łączenie metod *uczenia przez demonstrację* z *Q-learningiem*. Połączony algorytm byłby w stanie szybko uzyskać przyzwoicie zachowującego się agenta nauczonego na podstawie prezentacji eksperta, a następnie doskonalić jego zachowanie i eliminować niedoskonałości za pomocą klasycznych metod uczenia ze wzmocnieniem.

Literatura

[Abbeel et al., 2008] Abbeel, P., Dolgov, D., Ng, A. Y., and Thrun, S. (2008). Apprenticeship learning for motion planning with application to parking lot navigation. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, September 22-26, 2008, Acropolis Convention Center, Nice, France*, pages 1083–1090.

[Bellman, 1954] Bellman, R. (1954). The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515.

[Bengio et al., 2015] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099.

[Brafman and Tennenholtz, 2002] Brafman, R. I. and Tennenholtz, M. (2002). R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

[Chang et al., 2015] Chang, K., Krishnamurthy, A., Agarwal, A., III, H. D., and Langford, J. (2015). Learning to search better than your teacher. *CoRR*, abs/1502.02206.

[Coates et al., 2009] Coates, A., Abbeel, P., and Ng, A. Y. (2009). Apprenticeship learning for helicopter control. *Commun. ACM*, 52(7):97–105.

[Crites and Barto, 1996] Crites, R. and Barto, A. (1996). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pages 1017–1023. MIT Press.

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G. J. and Dunson, D. B., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings.

[Hester et al., 2017] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., and Gruslys, A. (2017). Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732.

[Jaskowski, 2016] Jaskowski, W. (2016). Uczenie ze wzmocnieniem — generalizacja i zastosowania (materiały wykładowe).

[Kempka et al., 2016] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaskowski, W. (2016). Vizdoom: A doom-based AI research platform for visual reinforcement learning. *CoRR*, abs/1605.02097.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Mataric, 1994] Mataric, M. J. (1994). Reward functions for accelerated learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189. Morgan Kaufmann.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Osband et al., 2016] Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. (2016). Deep exploration via bootstrapped DQN. *CoRR*, abs/1602.04621.

[Ratliff et al., 2006] Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 729–736, New York, NY, USA. ACM.

[Ross and Bagnell, 2010] Ross, S. and Bagnell, D. (2010). Efficient reductions for imitation learning. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy.

[Ross et al., 2010] Ross, S., Gordon, G. J., and Bagnell, J. A. (2010). No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686.

[Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229.

[Schaal, 1999] Schaal, S. (1999). Is imitation learning the route to humanoid robots?

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *CoRR*, abs/1511.05952.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

[Stadie et al., 2015] Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *CoRR*, abs/1507.00814.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44.

[Tesauro, 1992] Tesauro, G. (1992). Temporal difference learning of backgammon strategy. In *Machine Learning Proceedings 1992*, pages 451 – 457. Morgan Kaufmann, San Francisco (CA).

Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Wojciech Kopeć ",
    title = "{Uczenie przez demonstrację na podstawie informacji obrazowej w środowisku 3D }",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2017",
}
```