



POLSKO-JAPONSKA AKADEMIA
TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Systemów Inteligentnych i Data Science

Data Science

Bartosz Żuk

Nr albumu s18174

Transfer wiedzy w nauce gier tekstowych
na przykładzie DRRN i TextWorld

Praca magisterska
Napisana pod kierunkiem
Dr. Michała Knapika

Warszawa, wrzesień, 2022

Streszczenie

Cele naukowe pracy obejmują zbadanie wybranych metod transferu wiedzy w problemie rozwiązywania gier tekstowych przy pomocy uczenia przez wzmacnianie. Gry tekstowe są złożonym środowiskiem o przestrzeni akcji i stanów określonej przez język naturalny. W pracy wykorzystano oprogramowanie badawcze TextWorld [10] (Microsoft) do stworzenia zbioru gier tekstowych, a następnie sprawdzono efektywność klasycznego transferu wag oraz zmodyfikowanej wersji Kickstartingu [42], dostosowanej do Q-learning [47].

Przedstawione metody wymagały treningu dwóch typów agentów: uczniów i nauczycieli. Nauczyciel, trenowany na zbiorze prostych gier, nadzoruje ucznia w nauce gier trudnych. Agenci oparci są o architekturę sieci neuronowej DRRN [22]. Zbadano cztery schematy metody transferu wiedzy, w tym dwie oryginalne. Wyniki eksperymentalne zostały poddane analizie; zarysowano również plan dalszych badań.

Słowa kluczowe

Uczenie przez wzmacnianie, Przetwarzanie języka naturalnego, Transfer wiedzy, Gry tekstowe, TextWorld, DRRN.

Spis treści

1 Wstęp	5
1.1 Gry tekstowe, ograniczenia NLP i uczenie przez wzmacnianie	5
1.2 Opis pracy	6
1.3 Zarys obserwacji eksperymentalnych	7
1.4 Plan pracy	7
2 Uczenie przez wzmacnianie	8
2.1 Problem uczenia przez wzmacnianie	8
2.1.1 Agent	8
2.1.2 Środowisko	9
2.1.3 Nagroda	10
2.1.4 Schemat problemu	10
2.2 Proces decyzyjny Markowa	11
2.3 Polityka, zwrot i użyteczność	12
2.3.1 Polityka	12
2.3.2 Zwrot	12
2.3.3 Użyteczność	13
2.4 Podstawowe metody uczenia przez wzmacnianie	15
2.4.1 Q-learning	15
2.4.2 Głęboki Q-learning	16
2.4.3 Policy gradient	17
2.5 Eksploracja kontra eksploatacja	18
2.5.1 Eksploracja ϵ -greedy	18
2.5.2 Eksploracja Boltzmanna	18
3 Gry tekstowe	20
3.1 Wyzwania gier tekstowych	20
3.2 Gry tekstowe jako POMDP	22
3.3 Przetwarzanie języka naturalnego	23
3.3.1 Tokenizacja	23
3.3.2 Zanurzenia słów	25
4 Transfer wiedzy w uczeniu przez wzmacnianie	29
4.1 Transfer wiedzy	29
4.1.1 Nauka z nadzorem	30
4.1.2 Nauka bez nadzoru	31

4.2	Kształtowanie nagrody	31
4.3	Transfer reprezentacji	33
4.4	Transfer polityki	35
5	Narzędzia	37
5.1	TextWorld	37
5.2	DRRN	38
5.2.1	Dwa warianty architektury DRRN	38
5.2.2	Algorytm uczenia	41
5.2.3	Uwagi do implementacji	44
6	Próby transferu wiedzy w grach tekstowych	45
6.1	Wygenerowane gry	45
6.2	Cztery metody transferu wiedzy	46
6.2.1	Nauczyciel uogólniony i wyspecjalizowany	46
6.2.2	Q-kickstarting i transfer wag	47
6.3	Przygotowanie do prób transferu wiedzy	48
6.4	Wyniki prób transferu	50
6.4.1	Nauczyciele wyspecjalizowani	51
6.4.2	Nauczyciel uogólniony	52
6.4.3	Środowisko eksperymentalne	55
7	Konkluzje	56
7.1	Napotkane trudności i ograniczenia	56
7.2	Wnioski i obserwacje	57
7.3	Dalsze badania	58

Rozdział 1

Wstęp

Ostatnie lata przyniosły imponujące postępy w zakresie przetwarzania języka naturalnego (ang. *Natural Language Processing, NLP*). Za najnowszymi osiągnięciami stoją potężne modele językowe, takie jak ELMo [36], BERT [13] czy GPT2 [37], analizujące statystyczne własności ogromnej ilości tekstu. Coraz częściej wskazywane są jednak ograniczenia podejścia czysto statystycznego [15]. Obecne modele językowe, fundamentalnie, uczą się języka na podstawie korpusu. Człowiek odnosi zaś abstrakcje słów i wyrazów do obiektów i zjawisk w świecie fizycznym, wykorzystując posiadane informacje do lepszego zrozumienia znaczenia odczytywanej treści. Ta obserwacja stoi u podstaw nowych trendów badań w dziedzinie NLP; zaproponowano szereg metod [23, 52, 9, 3] mających na celu uzupełnienie klasycznego podejścia do przetwarzania języka naturalnego poprzez uwzględnienie wiedzy na temat świata opisywanego przez język.

1.1 Gry tekstowe, ograniczenia NLP i uczenie przez wzmacnianie

Gry tekstowe są interaktywnymi symulacjami, w których język naturalny używany jest do opisu stanu gry oraz wydawanych komend. Opierając się wyłącznie na prezentowanej treści, gracz musi zrozumieć otaczający go świat gry - napotkane w nim obiekty i ich własności oraz możliwe działania i ich efekty. W rezultacie, gry tekstowe wymuszają analizę tekstu innego rodzaju niż analiza statystyczna korpusu. W celu określenia znaczenia słowa "klucz" niewystarczające jest poznanie jego statystycznych własności (częstotliwości występowania, rodzaju sąsiadujących wyrazów, itp.); wymagane jest też określenie czym jest klucz, co może mu się przydarzyć i co można z nim zrobić (np. otworzyć drzwi) [16].

Obok przetwarzania języka naturalnego, inteligentny agent musi posiadać umiejętności z zakresu eksploracji, długotrwałej pamięci, planowania oraz zdroworozsądkowego rozumowania. Unikalna i złożona problematyka gier tekstowych sprawiła że w ostatnim czasie zyskały one na popularności jako środowiska badawcze [10, 21, 22].

Najbardziej popularnym obecnie podejściem do problemu nauki gier tekstowych jest połączenie rozwiązań z zakresu przetwarzania języka naturalnego oraz uczenia przez wzmacnianie.

Nowe metody uczenia przez wzmacnianie osiągnęły w minionej dekadzie szereg spektakularnych sukcesów. W przełomowej pracy Mniha [32], inteligentny agent był w stanie

opanować gry typu Arcade na konsole Atari. Bardziej współcześnie, model AlphaStar [46] zyskał sławę rywalizując z najlepszymi graczami świata, w grę StarCraft II - złożoną strategię czasu rzeczywistego. Nauka obu wymienionych systemów, i wielu innych, oparta jest o tzw. głębokie uczenie ze wzmacnieniem (ang. *Deep Reinforcement Learning*), gdzie podstawą reprezentacji danych i efektywnych obliczeń są sieci neuronowe. Imponujące umiejętności agentów uczenia ze wzmacnieniem wymagają wykorzystania ogromnych zasobów czasowych i obliczeniowych (nauka AlphaStar trwała 14 dni, używając 16 jednostek obliczeniowych TPU).

Zapotrzebowanie na bardziej efektywne sposoby treningu agentów przekłada się na zwiększone zainteresowanie transferem wiedzy, czyli metodom korzystania w procesie nauki z wiedzy i umiejętności zdobytych na zadaniach pokrewnych. W niniejszej pracy badamy wybrane metody transferu wiedzy w nauczaniu przez wzmacnianie, sprawdzając jakie mają efekty na trening inteligentnych agentów w złożonych środowiskach gier tekstowych.

1.2 Opis pracy

Celem pracy jest zbadanie efektywności transferu wiedzy w nauce gier tekstowych.

Zbiór gier testowych. Wykorzystując oprogramowanie **TextWorld** [10], tworzymy zbiór treningowy składający się z 10 gier tekstowych podzielonych na dwa poziomy trudności. W każdej z gier zadaniem agenta jest skonsumowanie posiłku przygotowanego według przepisu ukrytego w świecie gry, określającego rodzaj i sposób przyrządzenia poszczególnych składników. Światy trudnych gier charakteryzują się większą niż światy gier łatwych liczbą pomieszczeń i obiektów oraz większą liczbą składników w przepisie. Dodatkowo, w grach trudnych niektóre obiekty, takie jak drzwi czy szafki, mogą być w stanie zamkniętym lub otwartym i gracz musi odkryć sposób ich otwarcia.

Badania transferu wiedzy. Eksperymentalnie zbadano klasyczne podejście do transferu wiedzy poprzez transfer wag pomiędzy modelami oraz zmodyfikowaną wersję Kickstartingu [42], dostosowaną do Q-learning.

Ścisłe, dla każdej metody wytrenowano parę agentów: ucznia i nauczyciela. Nauczyciel, trenowany na prostszych grach, nadzoruje ucznia w nauce gier trudniejszych. Oprócz metod transferu zbadano również dwie strategie treningu nauczycieli:

- W pierwszej, nauczyciel wyspecjalizowany trenowany jest na jednej grze prostej, a następnie jego wiedza wykorzystywana jest w trudniejszej wersji tej samej gry (obie gry mają zbliżony świat, zaś przepis w grze trudniejszej jest rozszerzeniem przepisu z gry łatwej). W transferze wykorzystujemy zatem wiedzę wyspecjalizowaną na temat danego typu gry.
- W drugiej, nauczyciel uogólniony trenowany jest na zbiorze wszystkich gier prostych a następnie nadzoruje agentów uczących się rozwiązywać gry trudne. W transferze wykorzystujemy więc ogólną wiedzę na temat zbioru gier.

Obie z wymienionych strategii testujemy dla wszystkich metod transferu wiedzy. Ponadto, dla każdej trudnej gry przeprowadzono trening wariantu bazowego (ang. *baseline*), tzn. agenta który nie korzysta z transferu wiedzy.

Struktura agentów oparta jest o sieci neuronowe **DRRN** [22] (opisane w Sekcji 5.2).

1.3 Zarys obserwacji eksperymentalnych

Przeprowadzone eksperymenty wskazują, że transfer wag z użyciem uogólnionego nauczyciela nie tylko przyspiesza trening ale też pozwala na osiągnięcie lepszych wyników.

Wersja z nauczycielami wyspecjalizowanymi wydaje się być zdecydowanie mniej efektywna: uczniowie nadzorowani tą metodą mają w naszych badaniach wyraźną tendencję do wcześniego blokowania się na poziomie 6 punktów (z 8 możliwych), nie mogąc poprawić wyniku do końca treningu.

Zjawisko to jest mniej widoczne u uogólnionego nauczyciela, co sugeruje, że ta strategia treningu lepiej dostosowuje się do zmian w środowiskach trudniejszych gier.

Spośród metod transferu wiedzy, Q-kickstarting okazuje się nieskuteczny. Co więcej, metoda ta osiąga wyniki zbliżone do wariantu bazowego dla uogólnionego nauczyciela i zauważalnie gorsze w przypadku nauczycieli wyspecjalizowanych. Przypuszczamy, że powodem nieefektywności Q-kickstartingu może być niewystarczająco szeroko zakrojona optymalizacja parametrów metody lub wrodzona trudność środowisk gier tekstowych (duża przestrzeń akcji, rzadki sygnał nagrody). W przyszłości zamierzamy przeprowadzić dalsze badania w celu weryfikacji powyższych hipotez.

1.4 Plan pracy

Praca podzielona jest na **7 rozdziałów**.

1. Rozdział pierwszy przedstawia tematykę pracy oraz formułuje cele badawcze.
2. Rozdział drugi wprowadza czytelnika w tematykę nauczania przez wzmacnianie, skupiając się na omówieniu i zdefiniowaniu pojęć wykorzystywanych w dalszej części pracy.
3. Rozdział trzeci poświęcony jest grom tekstowym oraz wybranym zagadnieniom z zakresu przetwarzania języka naturalnego.
4. Rozdział czwarty skupia się na omówieniu transferu wiedzy oraz pełni rolę przeglądu literatury z zakresu tego obszaru badań.
5. Rozdział piąty omawia wykorzystane narzędzia, w szczególności bibliotekę TextWorld, architekturę sieci DRRN i algorytm nauczania.
6. Rozdział szósty zawiera opis przebiegu prób transferu wiedzy oraz analizę otrzymanych wyników eksperymentalnych.
7. Ostatni rozdział poświęcony jest konkluzjom. Omówiono w nim napotkane trudności, wnioski i obserwacje oraz zaproponowano dalsze kierunki badań związane z transferem wiedzy w nauce gier tekstowych.

Rozdział 2

Uczenie przez wzmacnianie

Celem niniejszego rozdziału jest wprowadzenie kluczowych pojęć oraz zarysowanie intuicji związanych z uczeniem przez wzmacnianie. Początek rozdziału poświęcimy omówieniu problemu uczenia przez wzmacnianie w sposób ogólny, zaś w dalszej części wprowadzimy podstawowy formalizm matematyczny i zaprezentujemy wybrane metody uczenia. Zawartość rozdziału bazuje na książce *Reinforcement Learning: an Introduction* [45] autorstwa Richarda Suttona oraz Andrew Barto, a w szczególności użyta została spójna z nią notacja matematyczna.

2.1 Problem uczenia przez wzmacnianie

Uczenie przez wzmacnianie (ang. *Reinforcement learning, RL*) jest popularnym podejściem do uczenia maszynowego w którym występuje **agent** (inteligentna jednostka) funkcjonujący w **środowisku** i podejmujący akcje zgodne z pewną **polityką** w celu maksymalizacji swojej **funkcji nagrody**. Typowe podejście do optymalizacji funkcji nagrody polega na karaniu lub nagradzaniu interakcji agenta ze środowiskiem, np. wedle tego, czy przybliżają czy oddalają agenta od osiągnięcia celu. Na podstawie tych bodźców możemy modyfikować schematy zachowania agenta, dając do budowy coraz skuteczniejszej strategii postępowania (tzn. polityki).

Wprowadźmy wyżej wymienione pojęcia bardziej precyzyjnie.

2.1.1 Agent

Agent jest autonomiczną inteligentną jednostką obserwującą środowisko i oddziaływaną na nie poprzez swoje **akcje**.

Zbiór wszystkich dostępnych akcji nazywamy **przestrzenią akcji**. Przestrzeń akcji może być ciągła lub dyskretna. Na przykład, w grze “Papier, kamień i nożyce” akcje są dyskretne: mamy do wyboru jedynie trzy możliwości. Z drugiej strony, problem nauki nawigacji pojazdu autonomicznego może być rozważany w kontekście ciągłej przestrzeni akcji. Przykładowo, gdy w każdym stanie dopuśćmy decyzję skrętu kół pod dowolnym wymiernym kątem z zakresu $[0, 360]$.

Agent jest zarówno uczniem, jak i podejmującym decyzje. W dokonywaniu wyborów agent

kieruje się pewną strategią nazywaną **polityką** π . Polityka przyporządkowuje każdemu zaobserwowanemu, przez agenta, stanowi środowisku s dystrybucję $\pi(s)$ akcji przeznaczonych do wykonania. W szczególności agent może przyjąć politykę według której podejmuje decyzje w pełni losowo, tzn. każda akcja umożliwiona w danym stanie może być przez niego wybrana z tym samym prawdopodobieństwem.

Nauczanie agenta, przynajmniej w kontekście metod omawianych w tej pracy, polega na optymalizacji jego bieżącej polityki ocenianej w kontekście funkcji nagrody.

2.1.2 Środowisko

Środowisko jest otoczeniem które agent obserwuje i z którym wchodzi w interakcję. Akcje podjęte przez agenta mogą zmienić **stan** środowiska, zaś zbiór akcji dostępnych (lub *umozliwionych*) dla agenta może zależeć od stanu środowiska: na przykład w grze w szachy, każde poruszenie figury zmienia stan planszy i wpływa na możliwy zestaw ruchów.

Możliwości percepcyjne agenta (tzn. jego zdolności obserwacji i zapamiętywania konsekwencji uprzednio wykonanych akcji) i różne cechy środowisk prowadzą do szeregu klasyfikacji według różnych kryteriów. Poniżej przedstawiono kilka wybranych kategorii [39], które przydadzą się w omawianiu kolejnych zagadnień. Środowisko może być:

- **epizodyczne** lub **ciągłe**. W środowiskach epizodycznych możemy w łatwy sposób wyznaczyć początek i koniec interakcji, określając tak zwany **epizod**. Dzieje się tak w większości gier, gdzie zazwyczaj mamy jasno określone warunki wygranej. W innych problemach, na przykład we wspomnianej już nauce pojazdu autonomicznego, nie jest to tak proste. Agent powinien kontynuować naukę do momentu osiągnięcia zadowalających umiejętności sterowania pojazdem, ale określenie momentu przerwania treningu nie jest oczywiste. Środowiska gdzie nie możemy w naturalny sposób wyznaczyć epizodów kategoryzujemy jako środowiska ciągłe.
- **całkowicie** lub **częściowo obserwowe**. W środowiskach całkowicie obserwowańnych agent posiada pełną wiedzę o stanie środowiska. Na przykład gra w szachy zakłada iż obaj gracze obserwują całą planszę wraz z pozycjami wszystkich figur, jest więc to środowisko całkowicie obserwowe. Środowiska częściowo obserwowe charakteryzują się tym że dla agenta dostępna jest tylko częściowa informacja o pełnym stanie środowiska, nazywana **obserwacją**. Gra w statki, gdzie rozmieszczenie okrętów przeciwnika jest nieznane, jest przykładem środowiska częściowo obserwowanego. Podobnie, większość gier karcianych takich jak brydż lub poker jest częściowo obserwowańna.
- **stochastyczne** lub **deterministyczne**. W stochastycznych środowiskach efekt wykonania akcji zależy od pewnego rozkładu prawdopodobieństwa parametryzowanego stanem i akcją. Rozkład ten określa prawdopodobieństwo przejścia do stanów wynikowych. Gra w kości (Yahtzee) jest przykładem środowiska stochastycznego. W środowiskach deterministycznych w każdym stanie wykonanie danej akcji prowadzi zawsze do tego samego stanu wynikowego. Gra w szachy jest przykładem środowiska deterministycznego. Można zauważyć, że środowisko deterministyczne jest zdegenerowanym przykładem środowiska stochastycznego, gdzie w każdym stanie dla każdej akcji dokładnie jeden stan wynikowy ma w dystrybucji przyporządkowane prawdopodobieństwo 1.

- **dynamiczne** lub **statyczne**. Istotą środowisk statycznych jest niezmiennosć ich reguł (tzn. dystrybucji funkcji przejścia pomiędzy stanami i tego jak zależy ona od stanu i akcji). Zasady gry w szachy są takie same przez cały czas rozgrywki, jest więc ona statyczna. W dynamicznie zmieniających się środowiskach agent może wraz z upływem czasu być zmuszony dostosować się do zupełnie nowego otoczenia i reguł nim sterujących.

Granica oddzielająca agenta od środowiska nie jest oczywista: można np. zadać pytanie, czy pojazd w problemie nauki pojazdu autonomicznego jest agentem czy częścią środowiska. Oprogramowanie pojazdu dokonuje wyboru akcji i jest aktualizowane w trakcie nauki, stąd idealnie pasuje do definicji agenta. Z drugiej strony, stan techniczny pojazdu jest istotnym czynnikiem wpływającym na podejmowane decyzje: pojazd może być postrzegany jako system wieloagentowy, którego części (jak np. przebita opona) podlegają wpływom środowiska i mogą być modelowane jako jego część. W niniejszej pracy skłaniamy się do przyjmowania, że każda informacja, która wpływa na decyzje agenta pochodzi ze środowiska, zaś agenta traktujemy jako byt abstrakcyjny, odseparowany od otoczenia.

2.1.3 Nagroda

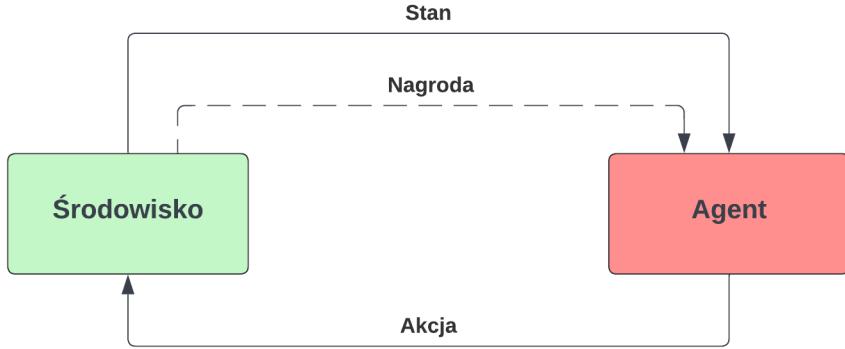
Nagroda (czasem zwana też *wypłatą*) jest obok stanu jedynym sygnałem zwrotnym przekazywanym ze środowiska do agenta. Zazwyczaj jest to liczba rzeczywista reprezentująca, jak dana akcja przybliża lub oddala agenta od osiągnięcia celu.

Uczenie przez wzmacnianie nie podsuwa agentowi bezpośrednio akcji które należy wykonać w danym stanie [45], w miejsce tego zmuszając go do eksperymentowania z ich wyborem i samodzielnej eksploracji środowiska. Sygnał nagrody (ang. *reward signal*) powinien więc być zaprojektowany w taki sposób, żeby (1) zdobycie przez agenta maksymalnej skumulowanej nagrody było równoznaczne z osiągnięciem celu; (2) iteracyjna poprawa polityki prowadziła do wzrostu skumulowanej nagrody; (3) efekty uczenia nie wymagały znajomości strategii wygrywającej a priori.

W szczególności, naturalne jest więc powiązanie sygnału nagrody z obserwacją stanu środowiska i pozostawienie schematowi uczenia dużej swobody w budowie polityki. Przykładowo, nagradzanie algorytmu nawigacji pojazdu autonomicznego wyłącznie za wybór tras zbliżonych do znanych już i obecnych w bazie danych jest niewskazanym podejściem, które może doprowadzić do budowy sztywnego schematu, nie radzącego sobie w dynamicznym środowisku miasta.

2.1.4 Schemat problemu

Podsumowując powyższą sekcję możemy określić podstawowy schemat problemu nauczania ze wzmacnieniem, zilustrowany na Rys. 2.1. Agent (obserwujący zarówno środowisko jak i wartość skumulowanej nagrody) ma za zadanie osiągnąć pewien cel poprzez interakcje ze środowiskiem. W każdym kroku tego procesu agent wykonuje akcję, wybraną świadomie lub losowo, na podstawie zapisu wcześniejszych interakcji, która zmienia stan środowiska. W wyniku tej zmiany agent otrzymuje nagrodę oraz informację o nowym stanie środowiska. Agent kontynuuje interakcje ze środowiskiem zgodnie z tym schematem do osiągnięcia uprzednio ustalonego punktu zatrzymania (czyli np. osiągnięcia celu, przekroczenia maksymalnej liczby iteracji, itp.).



Rysunek 2.1: Schemat interakcji pomiędzy agentem a środowiskiem

2.2 Proces decyzyjny Markowa

Do bardziej formalnego opisu problemów nauczania przez wzmacnianie używane są **procesy decyzyjne Markowa**, nazywane w skrócie MDP (ang. *Markov Decision Process*). W opisie MDP będziemy bazować na definicjach użytych w [45, 29], starając się ograniczyć do notacji z [45]. MDP jest krotką (S, A, R, T) gdzie:

- S jest zbiorem stanów $s \in S$, nazywanym również **przestrzenią stanów**.
- A jest zbiorem dostępnych akcji $a \in A$, nazywanym również **przestrzenią akcji**. Dodatkowo poprzez $A(s)$ będziemy oznaczać akcje dostępne w stanie s .
- $R \subseteq \mathbb{R}$ jest zbiorem nagród $r \in R$.
- T jest rozkładem prawdopodobieństwa warunkowego opisującego przejście pomiędzy stanami. Zapis $T(s', r|s, a)$ oznacza prawdopodobieństwo iż wykonanie akcji $a \in A(s)$ ¹ w stanie s doprowadzi do przejścia do stanu s' i wypłaty w wysokości r .

MDP jest procesem iteracyjnym, używamy więc subskryptu $t \in \mathbb{N}$ dla oznaczenia stanu procesu w kroku t . Zapis s_t, a_t odnosi się do stanu odwiedzonego w kroku t i akcji która została wykonana w tym stanie, natomiast r_t jest nagrodą która została wypłacona w wyniku wykonania akcji która doprowadziła do stanu s_t . Innymi słowy, nagroda r_{t+1} jest nagrodą zdobytą w wyniku wykonania akcji a_t w stanie s_t [45]. Dla poprawności formalnej można przyjąć, że $r_0 = 0$.

Kluczową cechą MDP jest **własność Markowa**. W systemie spełniającym własność Markowa przejście do kolejnego stanu s_{t+1} jest niezależne od jego historii. Zmiana stanu uwarunkowana jest jedynie ostatnio odwiedzonym stanem i wykonaną w nim akcją. Zależność ta wyrażona jest równaniem:

$$T(s_{t+1}, r_{t+1}|s_0, a_0, r_0, \dots, s_t, a_t, r_t) = T(s_{t+1}, r_{t+1}|s_t, a_t) \quad (2.1)$$

¹Zakładamy że $T(s'|s, a)$ dla $a \notin A(s)$ wynosi 0.

Własność Markowa mówi więc, że **przyszły stan MDP zależy wyłącznie od stanu teraz-niejszego**. To założenie znacząco upraszcza rozumowanie o dynamice procesów.

2.3 Polityka, zwrot i użyteczność

W poniższej sekcji opiszemy i formalnie wprowadzimy kolejne trzy kluczowe pojęcia. Pozwolą nam one precyzyjniej omówić, czym musi charakteryzować się optymalne rozwiązanie problemów nauczania przez wzmacnianie.

2.3.1 Polityka

W sekcji 2.1 wspomnieliśmy, że agent kieruje się polityką, czyli pewną strategią według której wybiera akcje w obserwowanych stanach. W literaturze rozważane są zwykle dwa rodzaje polityk: **deterministyczna** i **stochastyczna**. Polityka deterministyczna przyporządkowuje akcje stanom. Polityka stochastyczna przypisuje zaś każdemu stanowi rozkład określający dla każdej umozliwionej w nim akcji prawdopodobieństwo jej wykonania. W dalszej części odnosząc się do polityki będziemy w domyśle myśleli o polityce stochastycznej.

W formalnym ujęciu, polityka π jest rozkładem opisującym prawdopodobieństwo wykonania akcji w stanie. Zapis $\pi(a_t|s_t)$ oznacza prawdopodobieństwo wykonania akcji a_t w stanie s_t . Na końcu tej sekcji opiszemy warunki, jakie musi spełniać **polityka optymalna** (oznaczana π_*), która uzyskuje maksymalną skumulowaną nagrodę dla danego środowiska. W tym celu wprowadzimy kolejne pojęcia: zwrotu i użyteczności.

2.3.2 Zwrot

Zwrot pozwala nam określić **długoterminową** skumulowaną wartość stanu w którym znajduje się agent względem danego momentu w czasie i przy znanym ciągu zdobytych nagród. Znajomość ciagu nagród i stanów pozwala na budowę przypuszczeń dotyczących odwiedzonych stanów. Celem jest rozróżnienie stanów, które zaprowadziły agenta w ślepy zaułek, od tych które były kluczowe w naprowadzeniu go na wygrywającą trajektorię. Przykładowo, jeśli od czasu odwiedzenia pewnego stanu, wysokość kolejnych otrzymywanych nagród spada, to możemy przypuszczać że agent powinien tego stanu unikać.

Zwrot G_t jest funkcją operującą na ciągu nagród $r_{t+1}, r_{t+2}, r_{t+3}, \dots$ zdobytych od kroku t . W szczególności, gdy $t = 0$ ciąg ten zawiera wszystkie nagrody zdobyte w trakcie jednego (nieskończonego, lub ograniczonego pewnym horyzontem czasowym) przebiegu. Najbardziej naiwna wersja zwrotu została opisana w równaniu 2.2 i jest po prostu sumą ciągu.

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \dots \quad (2.2)$$

Równanie 2.2 nie sprawdza się dla środowisk w których liczba kroków jest nieograniczona, gdyż G_t może dążyć do nieskończoności [45]. Stąd używa się podejścia nazywanego **zdyskontowanym zwrotem** (ang. *discounted return*) opisanego równaniem 2.3.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (2.3)$$

Współczynnik dyskontowy $\gamma \in [0, 1]$, można interpretować jako dalekowzroczność agenta. Im γ jest mniejsze, tym mniej ceni on nagrody oddalone w czasie: agent staje się coraz bardziej "chciwy". Gdy γ zbliża się do 1, dalsze nagrody stanowią istotną część zwrotu, co sprawia że agent myśli bardziej przyszłościowo. Gdy $\gamma < 1$ oraz nagrody są nieujemne i ograniczone, zwrot G_t dąży do skończonej liczby. Pojęcie zwrotu zdyskontowanego można również zastosować do ograniczonego horyzontu czasowego, podmieniając ∞ przez stałą wartość w równaniu 2.3.

2.3.3 Użyteczność

Użyteczność (ang. *state-value function*) to oczekiwany zwrot jaki agent otrzyma podążając za wybraną polityką π . Innymi słowy, użyteczność jest miarą przy pomocy której możemy porównywać nie tylko poszczególne stany lub akcje ale też całe polityki. Równanie 2.4 definiuje wartość funkcji użyteczności w stanie s , przy ustalonej polityce π i zdyskontowanym zwrocie z parametrem γ .

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid S_0 = s \right] \quad (2.4)$$

Powyższy zapis jest półformalny, co wydaje się być popularne w literaturze. Bardziej ściśle:
(1) \mathbb{E}_π jest operatorem średniej wartości na przestrzeni wszystkich ciągów stanów s_0, s_1, \dots , przy czym funkcja prawdopodobieństwa jest w tej przestrzeni określona poprzez politykę π ;
(2) $\sum_{t=0}^{\infty} \gamma^t r_t$ jest zwrotem zdyskontowanym dla danego ciągu stanów (zadanego implicite).

Użyteczność akcji w danym stanie ocenia się przy pomocy funkcji użyteczności akcji w stanie $q_\pi(s, a)$ (ang. *state-action function*), zdefiniowanej dla akcji a umożliwionej w stanie s , przy ustalonej polityce π i zdyskontowanym zwrocie z parametrem γ poprzez równanie 2.5.

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid S_0 = s, A_0 = a \right] \quad (2.5)$$

Jak poprzednio, powyższy zapis jest półformalny i wiążą się z nim te same komentarze co w przypadku równania 2.4. Dodatkowo jednak $q_\pi(s, a)$ należy rozumieć jako obliczenie średniego zdyskontowanego zwrotu dla następujących przebiegów: (1) pierwszym stanem przebiegu jest s a pierwszą akcją a ; (2) kolejne akcje wybierane są zgodnie z polityką π .

Równania 2.4 i 2.5 można zapisać w pełni formalnie i obliczyć ich wartość bezpośrednio z definicji, jednak zwykle stosowane są metody rekurencyjne, jak przedstawiono poniżej.

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid S_0 = s \right] \\
&= \mathbb{E}_\pi \left[r_0 + \sum_{t=1}^{\infty} \gamma^t r_t \mid S_0 = s \right] \\
&= \mathbb{E}_\pi \left[r_0 + \gamma \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid S_0 = s \right] \\
&= \sum_a \pi(a|s) \sum_{s',r} T(s',r|s,a) \left(r + \gamma \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid S_0 = s' \right] \right) \\
&= \sum_a \pi(a|s) \sum_{s',r} T(s',r|s,a) \left(r + \gamma v_\pi(s') \right)
\end{aligned} \tag{2.6}$$

Zachodzi więc następująca zależność rekurencyjna, nazywana **równaniem Bellmana** [7].

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} T(s',r|s,a) \left(r + \gamma v_\pi(s') \right) \tag{2.7}$$

Jedyne nieoczywiste przejście w wyprowadzeniu 2.6 dotyczy linijek 3 i 4, gdzie wykorzystano twierdzenie o prawdopodobieństwie całkowitym i własność liniowości wartości oczekiwanej. Dokładniejsze wyprowadzenia równania można znaleźć w [45] oraz w oryginalnej pracy [7].

Własność opisana przez równanie Bellmana jest niezwykle istotna, ponieważ przy jej pomocy możemy sformułować **kryterium optymalizacji**. Optymalna funkcja użyteczności maksymalizuje nagrodę w każdym ze stanów s : $v_*(s) = \max_\pi v_\pi(s)$. Warto zauważyć [45], że może istnieć wiele polityk osiągających optymalną wartość funkcji użyteczności. Przekształconą postać kryterium optymalizacji, nazywanym też **kryterium optymalizacji Bellmana**, przedstawiono w równaniu 2.8.

$$v_*(s) = \max_a \sum_{s',r} T(s',r|s,a) \left(r + \gamma v_*(s') \right) \tag{2.8}$$

Analogiczne obliczenia można przeprowadzić dla funkcji użyteczności akcji. W tym celu wprowadza się rekurencyjną postać funkcji q_π (podobnie jak w równaniu 2.7), która jest następnie podstawą do formułowania warunku optymalności dla funkcji użyteczności akcji, czyli funkcji $q_*(s,a)$.

$$q_*(s,a) = \sum_{s',r} T(s',r|s,a) \left(r + \gamma \max_{a'} q_*(s',a') \right) \tag{2.9}$$

Wałącą cechą równań 2.8 oraz 2.9 jest ich **niezależność od polityki**. Fakt ten [45], oznacza że znalezienie optymalnych funkcji zależy tylko od dynamiki systemu sformułowanej poprzez MDP i sprowadza się do rozwiązania układu równań.

Dzięki pojęciu zwrotu i użyteczności możemy łatwo obliczyć **politykę optymalną** oznaczaną π_* . Wystarczy że w każdym stanie będziemy wybierać akcje która maksymalizuje funkcję q_* , tzn. $\pi_*(s) = \operatorname{argmax}_a q_*(s, a)$. Możemy powiedzieć, że polityka optymalna jest "chciwa" względem wartości q . Równoważnie optymalna polityka może być wyznaczona przy pomocy funkcji użyteczności stanów, poprzez wybór w danym stanie akcji która prowadzi agenta do stanu maksymalizującego wartość v_* .

2.4 Podstawowe metody uczenia przez wzmacnianie

Omawiając metody nauczania, ograniczymy się do dokładnego omawiania wyłącznie **Q-learning**, w szczególności w kontekście nauczania głębokiego (ang. *Deep learning*). Sukcesy nauczania głębokiego w aproksymowaniu złożonych, nieliniowych funkcji przyczyniły się do szybkiej popularyzacji metod z ich wykorzystaniem, kosztem bardziej tradycyjnych podejść, jak na przykład metody tabelaryczne, które omawiamy poniżej. Na koniec sekcji, jedynie zarysujemy intuicje drugiego, obok Q learning, prominentnego podejścia: **Policy Gradient**.

2.4.1 Q-learning

W końcowej części sekcji 2.3 wskazaliśmy, że optymalna polityka może zostać obliczona przy pomocy jednej z funkcji użyteczności. Oczywiście odszukanie faktycznej postaci takiej funkcji jest, dla większości praktycznych zadań, niezwykle trudne, stąd przeważająca część metod ogranicza się do jej aproksymacji. Nie inaczej jest z metodą Q-learning [47], której celem jest znalezienie funkcji Q będącej jak najlepszym przybliżeniem q_* . Skupienie się na użyteczności akcji, a nie stanu, pozwala uniezależnić naukę agenta od znajomości modelu środowiska.

Q-learning jest metodą **off-policy**. Oznacza to, że do nauki agenta nie jest wymagana znajomość explicite jego polityki. Należy jednak zaznaczyć, że polityka wciąż ma pośredni wpływ na trening agenta, odpowiedzialna jest bowiem za wybór akcji, które determinują odwiedzone stany i otrzymane nagrody. Tym samym generuje dane wykorzystywane do aktualizacji funkcji Q .

Reguła aktualizująca Q zapisana w równaniu 2.10 jest esencją algorytmu Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2.10)$$

Wyrażenie w nawiasach kwadratowych równania 2.10 można interpretować jako błąd pomiędzy wartością $q_*(s_t, a_t)$ estymowaną przy pomocy $Q(s_t, a_t)$ a jej dokładniejszą estymacją $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$ dostępną dopiero w kroku $t + 1$. Błąd ten nazywany jest **błędem TD** (ang. *Temporal difference error*). Rodzinę algorytmów optymalizujących błąd TD, do której zaliczamy Q-learning, nazywamy metodami TD lub **nauczaniem TD** (ang. *Temporal-Difference learning*).

Współczynnik $\alpha \in [0, 1]$ określa tempo zastępowania poprzedniej wartości $Q(s_t, a_t)$ nową, potencjalnie lepszą aproksymacją $q_*(s_t, a_t)$ uzyskaną po uwzględnieniu kolejnego kroku. By to zilustrować, przepiszmy równanie 2.10 w następujący sposób:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a)] \quad (2.11)$$

W podstawowej wersji, Q-learning implementowany jest w formie tabelarycznej. Zwyczajowo na pozycji przecięciu kolumny i i wiersza j notuje się bieżącą estymację użyteczności $Q(s_j, a_i)$. Tabela jest wstępnie inicjalizowana i, wraz z interakcją agenta ze środowiskiem, systematycznie aktualizowana przy pomocy reguły zapisanej w równaniu 2.10. Schemat tabelarycznego algorytmu *Q-learning* (zakładając środowisko epizodyczne) przedstawiono w Alg. 1.

Algorytm 1 Tabelaryczny Q-learning

```

1: Inicjalizuj losowo tabelę  $Q(s, a)$ 
2: for each epizod do
3:    $S \leftarrow$  stan początkowy  $s_0$ 
4:   for each krok epizodu do
5:      $A \leftarrow$  akcja wylosowana w stanie  $S$  na podstawie bieżącej wartości  $Q$ 
6:     Wykonaj akcję  $A$ 
7:      $R, S' \leftarrow$  nagroda i nowy stan zaobserwowane po wykonaniu  $A$ 
8:      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', A) - Q(S, A)]$ 
9:      $S \leftarrow S'$ 
10:    end for
11:  end for

```

Alg. 1 pozostawia otwartym wybór akcji w stanie S (linia 7 algorytmu). W sekcji 2.5 przedstawiamy dokładniej dwie popularne techniki jego dokonywania.

2.4.2 Głęboki Q-learning

Wraz ze wzrostem liczbą stanów i akcji metoda tabelaryczna staje się niepraktyczną obliczeniowo. Najczęściej współcześnie stosowane wydajne obliczeniowo i pamięciowo estymatory to (głębokie) sieci neuronowe.

Głęboki Q-learning oparty jest na kluczowym mechanizmie **bufora powtórek**. Alg 2, będący uproszczoną wersją Alg. 1 z pracy [32], wykorzystuje bufor \mathcal{D} w którym zapisywane są zaobserwowane przejścia w postaci $\sigma_t = (s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1}, a_t, r_t, s_t)$. Intuicyjnie, w buforze zapisywana jest informacja iż agent zdecydował się na wykonanie akcji a_t po tym jak w trakcie pewnej rozgrywki zaobserwował sekwencję stanów, akcji i nagród $s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1}$, zaś po wykonaniu tej akcji została mu wypłacona nagroda r_t i przeszedł do stanu s_t . Dla uzupełnienia dodajmy, że w oryginalnej pracy przyjmuje się naturalne założenie o tym iż bufor jest ograniczony a powyższe sekwencje obserwacji zastępowane są ograniczonymi reprezentacjami; te ograniczone reprezentacje mogą być w najprostszym przypadku postfiksami długości K , czyli odcinkami $s_{t-K}, a_{t-K}, r_{t-K}, \dots, s_{t-1}, a_{t-1}, r_{t-1}$.

Sieci neuronowe wykorzystywane w schematach głębokiego Q-learning przyporządkowują zwykle sekwencji obserwacji σ_t dystrybucję na akcjach umówionych w końcowym stanie s_t . Można więc powiedzieć, że mamy tu do czynienia z funkcją $Q(\sigma_t, a)$, czyli z przypadkiem bardziej ogólnym niż w tabelarycznej wersji Q-learning.

Algorytm 2 Głęboki Q-learning [32]

```
1: Inicjalizuj bufor  $\mathcal{D}$ 
2: Inicjalizuj losowo wagi  $\Theta$  sieci reprezentującej funkcję  $Q(\sigma_t, a; \Theta)$ 
3: for  $epizod = 1, \dots, E$  do
4:    $S \leftarrow$  stan początkowy  $s_0$       // Bieżący stan
5:    $\sigma \leftarrow S$                   // Bieżąca ścieżka
6:   for  $t = 1, \dots, T$  do // np. do zakończenia gry, czyli dotarcia do stanu finalnego
    // Wykonanie akcji i obserwacja wyników
7:      $A \leftarrow$  akcja wylosowana w stanie  $S$  na podstawie bieżącej wartości  $Q$ 
8:     Wykonaj akcję  $A$ 
9:      $R, S' \leftarrow$  nagroda i nowy stan zaobserwowane po wykonaniu  $A$ 
10:    Zapisz w  $\mathcal{D}$  przejście  $(\sigma, A, R, S')$ 

    // Uczenie na podstawie zebranych doświadczeń
11:    Wylosuj próbki przejścia  $(\sigma_j, A_j, R_j, S'_j)$  z bufora  $\mathcal{D}$ 
12:    Dla każdego przejścia z próbki oblicz  $y_j = R_j + \gamma \max_{a'} Q((\sigma_j, A_j, R_j, S'_j), a'; \Theta)$ 
13:    Na podstawie próbki bufora wykonaj krok uczenia (np. metodą spadku gradientu) aproksymując wartości  $y_j$  poprzez  $Q(\sigma_j, A_j; \Theta)$ 

    // Uaktualnienie bieżącego stanu
14:     $\sigma \leftarrow (\sigma \ A \ R \ S')$ 
15:     $S \leftarrow S'$ 
16:  end for
17: end for
```

W niniejszej pracy nie poruszamy kluczowego tematu architektury sieci. W sekcji 5.2 poświęconej modelowi **DRRN** [22] powrócimy do zagadnień związanych z Q-learning, przedstawiając bardziej szczegółowy algorytm nauczania dla konkretnego zagadnienia gier tekstowych.

2.4.3 Policy gradient

Rodzina algorytmów *Policy gradient* jest drugim, obok Q-learning, filarem na którym opiera się nauczanie przez wzmacnianie. W przeciwieństwie do Q-learningu, gdzie szukamy odpowiedniej aproksymacji funkcji q_* , metody Policy Gradient skupiają się na bezpośredniej optymalizacji polityki π . Polityka modelowana jest przy pomocy funkcji $\pi(a|s, \Theta)$, gdzie Θ to pewien zbiór parametrów (np. wagi i progi sieci neuronowej).

Intuicyjnie, trening metodą Policy Gradient opiera się na generowaniu przebiegów (wykorzystując ostatnią znalezioną politykę) na podstawie których estymowany jest zwrot poszczególny akcji. Następnie, korzystając z różnego rodzaju metod, modyfikuje się parametry Θ , tak aby zwiększyć prawdopodobieństwo akcji o dużym zwrocie.

W niniejszej pracy skupiamy się na metodach opartych o Q-learning, stąd pomijamy formalny opis Policy Gradient. Powyższe krótkie omówienie intuicji jest wystarczające dla zrozumienia reszty pracy. Zainteresowanych czytelników odsyłamy do książki Suttona i Barto [45] (w szczególności sekcji 13.1 i 13.2), gdzie można znaleźć dokładne omówienie teorii stojącej za Policy Gradient.

2.5 Eksploracja kontra eksplotacja

Jednym z unikalnych wyzwań z jakim mierzą się metody nauczania przez wzmacnianie jest balansowanie pomiędzy eksploracją (czyli badaniem nieznanej części środowiska) a eksplotacją (czyli wykorzystywaniem już poznanej części wiedzy o środowisku). Każda interakcja agenta ze środowiskiem dostarcza wiedzy na temat jego struktury i dynamiki. W szczególności agent zdobywa informacje na temat poszczególnych akcji i ich użyteczności, może więc albo **eksploatować** tę wiedzę, wybierając akcje które na ten moment są najbardziej efektywne, albo kontynuować **eksplorację** środowiska. Zarówno eksplotacja jak i eksploracja są konieczne do efektywnego nauczania przez wzmacnianie. Wymagane jest znalezienie pewnego kompromisu pozwalającego agentowi efektywnie wykorzystywać dotychczasowe najlepsze rozwiązanie i jednocześnie umożliwiając mu szukanie potencjalnych ulepszeń.

W algorytmach Policy Gradient konflikt pomiędzy eksploracją a eksplotacją jest w naturalny sposób rozwiązywany przez stochastyczną naturę polityki. Na początku, kiedy polityka jest jeszcze w dużym stopniu losowa, agent będzie intensywnie eksplorował środowisko. Jednak wraz ze zbieżnością algorytmu, dominować będzie eksplotacja. Inaczej sprawy się mają z metodami Q-learning, gdzie domyślnie wybieramy akcje w sposób deterministyczny - maksymalizując estymowaną użyteczność. W tym wypadku, konieczne jest zastosowanie technik, nazywanych **politykami eksploracji**, które umożliwią eksplorację środowiska w poszukiwaniu lepszych rozwiązań. Dwie takie techniki prezentujemy poniżej.

2.5.1 Eksploracja ε -greedy

Polityką eksploracji ε -greedy jest niezwykle prosta. Parametr ε określa prawdopodobieństwo z jakim wybieramy akcje losowo, zamiast stosując domyślną "chciwą" politykę. Z prawdopodobieństwo $1 - \varepsilon$ wybieramy akcje maksymalizującą wartość Q . Innymi słowy, ε określa stosunek eksploracji do eksplotacji. Częstą modyfikacją podstawowej wersji tej techniki jest stopniowe wygaszanie eksploracji, np. poprzez zmniejszanie wartości ε o 10% z każdym epizodem (w przypadku środowisk epizodycznych).

$$\varepsilon\text{-greedy} = \begin{cases} \underset{a}{\operatorname{argmax}} Q(s, a) & \text{z prawdopodobieństwem } 1 - \varepsilon \\ \text{losowa akcja} & \text{z prawdopodobieństwem } \varepsilon \end{cases} \quad (2.12)$$

2.5.2 Eksploracja Boltzmanna

Kolejną popularną polityką eksploracji jest eksploracja Boltzmanna, korzystającą z funkcji **Softmax** (patrz równanie 2.13) czyli wygładzonej wersji funkcji argmax (przy pomocy której implementujemy *chciwą* politykę eksploracji). Softmax przyjmuje za argument wektor liczb rzeczywistych, którego elementy mapowane są do przedziału (0, 1) w sposób proporcjonalny do wartości wejściowych. Dodatkowo elementy wektora sumują się do jedności, co pozwala interpretować wektor jako rozkład prawdopodobieństwa.

$$\operatorname{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (2.13)$$

Celem eksploracji Boltzmana jest normalizacja wartości Q poszczególnych akcji do rozkładu prawdopodobieństwa. Dzięki czemu eksploracja nowych akcji będzie proporcjonalna do ich obecnie estymowanej użyteczności. Prawdopodobieństwo wyboru akcji obliczane jest zgodnie z równaniem 2.14.

$$\text{Boltzmann}(s, a^i) = \frac{\exp(\tau \cdot Q(s, a^i))}{\sum_{a_j \in A(s)} \exp(\tau \cdot Q(s, a^j))} \quad (2.14)$$

$A(s)$ jest zbiorem akcji dostępnych w stanie s , a^i jest i-tą dostępną akcją, a τ jest parametrem zwyczajowo nazywanym **temperaturą**. Temperatura daje dodatkową możliwość kontrolowania stosunkiem pomiędzy eksploracją a eksplotacją. Im gorętsza temperatura (większe τ) tym bardziej wypłaszczyły rozkład prawdopodobieństwa, tym samym częściej wybierane będą nie "najlepsze" akcje, co doprowadzi do zintensyfikowanej eksploracji środowiska. Niższa temperatura sprawi że agent skupi się na eksplotowaniu obecnie najskuteczniejszego rozwiązania.

Rozdział 3

Gry tekstowe

Rozdział trzeci poświęcimy omówieniu gier tekstowych. Zdefiniujemy je, korzystając z pojęć wprowadzonych w poprzednim rozdziale, jako częściowo obserwowlane procesy decyzyjne Markowa. Dodatkowo, przedstawimy kilka konceptów z zakresu przetwarzania języka naturalnego, które okażą się przydatne w kontekście pracy z gram i tekstowymi.

3.1 Wyzwania gier tekstowych

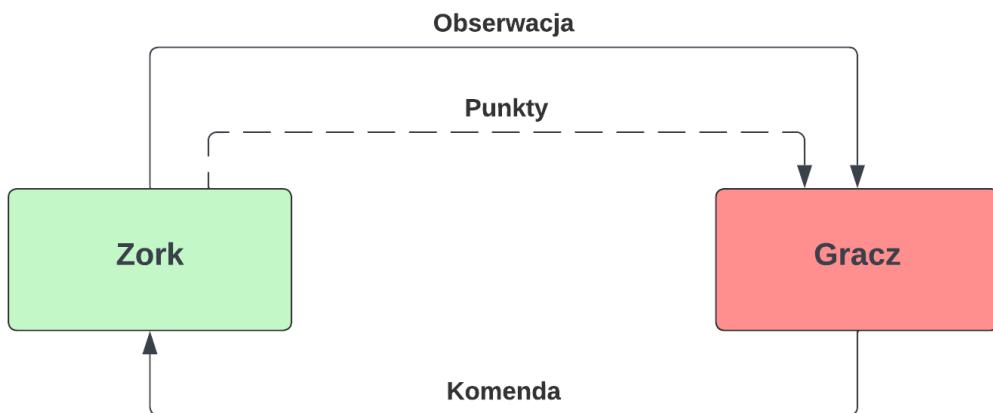
Gry tekstowe, nazywane też interaktywną fikcją (w skrócie IF), są rodzajem oprogramowania gdzie komunikacja z użytkownikiem odbywa się przy pomocy języka naturalnego. Graczowi prezentowany jest opis pewnego wirtualnego świata na który może wpływać wydając proste tekstowe komendy. Typowy dla gatunku jest brak jakiekolwiek oprawy graficznej oraz dźwiękowej. W zamian IF skupia się na rozbudowanej narracji i złożonych wieloetapowych zagadkach. Jedną z najstarszych gier tekstowych jest Zork [4], gdzie wcielamy się w bezimiennego poszukiwacza przygód przemierzającego podziemne labirynty. Rysunek 3.1 prezentuje początkowy opis świata w Zork wraz z wprowadzoną przez gracza komendą i stosowną odpowiedzią systemu.

```
Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved.  
ZORK is a registered trademark of Infocom, Inc.  
Revision 88 / Serial number 840726  
  
West of House  
You are standing in an open field west of a white house, with a boarded front door.  
There is a small mailbox here.  
  
> open mailbox  
Opening the small mailbox reveals a leaflet.  
  
> █
```

Rysunek 3.1: Wstęp do gry Zork

Gry tekstowe wpasowują się w schemat problemu nauczania przez wzmacnianie, który opisaliśmy w poprzednim rozdziale. Gracz (**agent**) otrzymuje tekstowy opis stanu świata

gry (**środowiska**) i na jego podstawie wydaje wybraną komendę (**akcję**). Najczęściej komenda składa się z rzeczownika wskazującego pewien obiekt w świecie gry (w tym samego gracza) oraz czasownika opisującego czynność jakiej wybrany obiekt ma być poddany. Przykładami komend z Zork są `open mailbox`, `take leaflet` czy `go north`. W przypadku wielu gier, za sygnał nagrody można wykorzystać wbudowany system punktacji. Jeśli gra takiego systemu nie posiada, nagrody muszą zostać zaimplementowane ręcznie.



Rysunek 3.2: Schemat interakcji (analogiczny do rysunku 2.1)

Rozbudowane gry tekstowe stanowią złożone środowiska które stawiają algorytmy nauczania przed szeregiem wyzwań. Wybrane z nich opisujemy poniżej.

Częściowa obserwowalność. Tekst prezentowany graczowi jest jedynie opisem wybranej lokalizacji lub obiektu, nie całego świata gry. Dodatkowo, treść opisu nie musi być jednoznacznie interpretowalna i może nie zawierać niektórych szczegółów. Patrząc na rysunek 3.1, widzimy że gracz początkowo nie jest poinformowany o zawartości skrzynki pocztowej: tę informację zdobywa dopiero po jej otworzeniu. Dodatkowo, jak zauważa [10], niektóre stany mogą być nieroróżnicalne. Założymy, że gracz znajduje się w pomieszczeniu z dwoma wyjściami, gdzie jedne drzwi prowadzą do kuchni, zaś drugie do salonu. Opis stanu następujący po otwarciu drzwi (na przykład `You opened the door!`) może być taki sam niezależnie od tego które wybrałyśmy. Powyższe przykłady obrazują częściową obserwowalność gier tekstowych. Zgodnie z terminologią wprowadzoną w sekcji 2.1.2, tekstowy opis stanu gry będziemy nazywali **obserwacją**.

Rozumienie affordancji. Afordancja (ang. *affordance*) jest terminem zaproponowanym przez Jamesa Gibsona [18], oznaczającym możliwe przeznaczenia obiektu. Bardziej precyzyjnie, affordancja mówi zarówno o użyciach danego przedmiotu jak i działaniach jakim może być on poddany. Przykładowo, klucz używany jest do otwierania skrzyni lub drzwi, może też być upuszczony lub podniesiony. Jasne jest też, że affordancja nie zachodzi zawsze. Raczej nie powiedzielibyśmy że klucz można pokochać. Na pewno nie powiedzielibyśmy też, że służy do jazdy albo pieczenia. Gry tekstowe wypełnione są obiektemi, które mają

różne zastosowanie w różnych kontekstach. Poprawne rozumienie afordancji i podejmowanie zdroworozsądkowych działań na ich podstawie jest niezwykle istotne, szczególnie przy wyborze komend.

Duża przestrzeń stanów i akcji. Już dla średniej wielkości gier tekstowych rozmiar przestrzeni stanów jest pokaźny. Rozważmy dodanie do gry nowego przedmiotu. Łatwo zauważać, że dla każdego elementu przestrzeni powstanie przynajmniej jeden kolejny stan w którym gracz posiada dodany przedmiot w ekwipunku. Powyższy przykład obrazuje kombinatoryczną naturę przestrzeni stanów gier tekstowych, stanowiącą istotne wyzwanie dla algorytmów nauczania. Przestrzeń akcji wprowadza dodatkowe utrudnienia. Po pierwsze, podobnie jak poprzednio, jest ona niezwykle duża. Mając pewien zbiór słów (nazywanym zwyczajowo **słownikiem**), komenda może być dowolnych ciągiem jego elementów. Po drugie, spośród wszystkich możliwości jedynie niewielka część komend będzie rozumiana i dozwolona przez silnik gry. Rozmiary obu przestrzeni, w większości przypadków, przekraczają możliwości metod tabelarycznych i wymuszają użycie bardziej wydajnych technik.

Rzadkość nagród. Kolejnym wyzwaniem gier tekstowych jest struktura sygnału nagrody. Dla większości gier tekstowych jedynie nieliczna część akcji jest punktowana. W ekstremalnym przypadku agent otrzymuje tylko pojedynczą nagrodę w momencie ukończenia rozgrywki. Brak częstych nagród, czy też **rzdaki sygnał nagrody** (ang. *sparse reward*) prowadzi do niewydajnej, losowej eksploracji środowiska i może znaczaco wydłużać czas treningu agenta. Rzadki sygnał nagrody jest istotnym problemem w nauczaniu przez wzmacnianie. Istnieje wiele metod mierzących się z tym zagadnieniem. Jedną z popularniejszych jest **kształtowanie nagrody** (ang. *reward shaping*). Wspomnimy o niej krótko, w kontekście transferu wiedzy, w kolejnym rozdziale.

3.2 Gry tekstowe jako POMDP

Wzorując się na [10], zdefiniujemy gry tekstowe jako częściowo obserwowlane procesy decyzyjne Markowa (ang. *partially observable Markov decision processes, POMDP*) [27]. POMDP jest krotką (S, A, R, T, Ω, O) , gdzie:

- S jest przestrzenią stanów
- A jest przestrzenią akcji
- R jest zbiorem nagród
- T jest rozkładem prawdopodobieństwa warunkowego przejść stanów
- Ω jest zbiorem obserwacji
- O jest rozkładem prawdopodobieństwa warunkowego obserwacji.

W każdym kroku POMDP agent wydaje pewną komendę c . Komenda nie jest równoważna akcji. W szczególności pojedynczej akcji może odpowiadać więcej niż jedna komenda. By to zilustrować, rozważmy akcję otwierania czerwonych drzwi: można się spodziewać, że zarówno komenda `open door` jak i `open red door` zostaną uznane za poprawne i wprowadzą takie same zmiany w stanie gry. Dla ułatwienia dalszego rozumowania, A zdefiniujemy

więc nie jako zbiór wszystkich dyskretnych akcji a , ale jako zbiór wszystkich słów z których możemy zbudować komendy c . W rezultacie, tak jak wspomnieliśmy w sekcji 3.1, zaledwie niewielki wycinek przestrzeni akcji będzie akceptowany przez silnik gry i będzie miał wpływ na stan środowiska.

Do jednego stanu przyporządkowanych może być wiele obserwacji. Weźmy za przykład komendę `inventory`, która wyświetla zawartość ekwipunku agenta. Samo wydanie tej komendy nie zmieni stanu gry, jest jednak nową obserwacją przedstawiającą stan posiadania gracza w kontekście danej lokacji. Dodatkowo, w niektórych grach, możemy mieć przyporządkowane po kilka obserwacji dla tego samego stanu i komendy. Przykładowo silnik gry może losowo zwracać jeden z kilku, znaczeniowo równoważnych, opisów pomieszczenia. Rozkład warunkowy O precyzuje zależności pomiędzy poszczególnymi stanami, komendami i obserwacjami. Precyzyjniej, O opisuje prawdopodobieństwo $O(o_t|s_t, c_{t-1})$ otrzymania przez agenta obserwacji $o_t \in \Omega$ pod warunkiem znajdowania się w stanie s_t oraz wydania komendy c_{t-1} .

3.3 Przetwarzanie języka naturalnego

Poświęcimy teraz uwagę omówieniu wybranych pojęć z zakresu przetwarzania języka naturalnego (ang. *Natural Language Processing, NLP*), a w szczególności metodach reprezentacji tekstu w postaci liczbowej (wektorowej). Standardowo, tekst przeznaczony do zakodowania jest w pierwszym kroku (nazywanym **tokenizacją**) dzielony na części, którymi najczęściej są słowa, ale mogą być to również poszczególne znaki, całe zdania lub inaczej zdefiniowane fragmenty tekstu. Następnie, dla każdej unikalnej części tekstu tworzy się oddzielną reprezentację matematyczną. W tym celu używa się zwykle wektorów liczb rzeczywistych. W kontekście NLP, przyjęło się określić wektory reprezentujące tekst jako **zanurzenia** (ang. *embedding*) - pojęcie oryginalnie zapożyczone z matematyki teoretycznej. W zależności od tokenizacji mówimy o zanurzeniach słów, wyrazów, zdań itd.

Poniżej, bazując na książce Jurafsky'ego i Martina [26], zarysujemy dwa wspominane, kluczowe koncepty: tokenizację i zanurzenia.

3.3.1 Tokenizacja

Tokenizacja jest procesem segmentacji tekstu. Tekst dzielony jest na pewne atomowe części, zwane **tokenami**. Tak jak wspomnieliśmy, w kontekście języka naturalnego tokenami najczęściej są słowa. W licznych zastosowaniach sens może mieć użycie innych jednostek podziału. W naszych rozważaniach ograniczymy się do omówienia tokenizacji na słowa. Podstawowym podejściem do tokenizacji jest wykorzystanie **wyrażeń regularnych** do wyznaczenia miejsc podziałów (tzw. separatorów). Niestrukturyzowana natura języka powoduje że liczba reguł i wyjątków, jakie musimy wziąć pod uwagę, szybko staje się przytłaczającą. Stąd popularność zyskały metody takie jak **BPE** (ang. *Byte-pair encoding*) [43]. Algorytm BPE automatycznie uczy się dzielić tekst na podstawie zaprezentowanych przykładów. Zaczniemy od omówienia prostej implementacji tokenizacji opartej na wyrażeniach regularnych. Następnie przejdziemy do zarysowania działania BPE.

Wyrażenia regularne. Przy dzieleniu tekstu na słowa naturalnymi miejscami podziału są znaki białe (ang. *whitespace*). Listing 3.1 prezentuje prostą implementację tego podejścia

w języku Python. Funkcja `re.split` przyjmuje dwa argumenty: tekst który należy podzielić oraz wyrażenie regularne przy pomocy którego znajdowane są separatory. Wykorzystajmy powyższy przykład do wskazania ograniczeń prostych metod opartych na wyrażeniach regularnych oraz zaproponujmy możliwe ulepszenia.

Listing 3.1: Prosta tokenizacja w języku Python

```
>>> import re
>>> text = 'Chytra gęś z Przemyśla Dolnego ukradła biało-czerwoną flagę!'
>>> re.split('\s+', text)
[''Chytra'', ''gęś'', ''z'', ''Przemyśla'', ''Dolnego'', ''ukradła'', ''biało-czerwoną'', ''flagę!'']
```

1. Pierwszym problemem jest brak wyodrębnienia znaków interpunkcyjnych jako oddzielnych tokenów. Przykładowo, zamiast dwóch tokenów `'flagę'` i `'!'` otrzymaliśmy jeden token `'flagę!'`. Problem interpunkcji jest o tyle złożony, że nie zawsze jej segmentacja jest wskazana. W przypadku wyrazu “biało-czerwoną” pozostawienie myślnika jako części tokenu wydaje się mieć sens, gdyż pozwala zachować oryginalne znaczenia słowa. Potencjalnym ulepszeniem może być zaprojektowanie dodatkowych reguł (wyrażeń regularnych), obejmujących oba wymienione przypadki. Niestety nie jest to zadanie proste i może wymagać specjalistycznej wiedzy eksperckiej.
2. Drugim problemem jest rozbijanie nazw własnych na oddzielne tokeny, co może doprowadzić do błędnej interpretacji zdań (dla ilustracji można zastanowić się np. nad interpretacją w powyższym przykładzie tokenu `'Dolnego'` jako przysłówka, wraz z jego możliwym bagażem znaczeniowym). Jednym z rozwiązań jest wyuczenie klasyfikatora by wykrywał nazwy własne i uniemożliwiał ich segmentacje. Niestety trening takiego klasyfikatora nie jest łatwy; wykrywanie nazw własnych (ang. *Named-entity recognition*, **NER**) jest złożonym i skomplikowanym zadaniem.

Bardziej generalnym problemem metod opartych na wyrażeniach regularnych jest potrzeba dostosowywania ich do konkretnych języków. Zupełnie inaczej wygląda tokenizacja dla języka angielskiego niż dla chińskiego: w języku chińskim poprawne określenie, która grupa znaków jest słowem, nie jest tak oczywiste jak w angielskim i w dużym stopniu jest zależne od kontekstu wypowiedzi. Problem jest na tyle duży, że jak zauważa [26, s. 17], w przypadku chińskiego, dla większości zadań NLP, bardziej efektywna okazuje się tokenizacja na znaki, nie słowa.

BPE. Tokenizacja jest pośrednim krokiem w procesie kodowania tekstu do reprezentacji liczbowej. Dla dużych korpusów nie jest możliwe reprezentowanie każdego poszczególnego słowa. Typowym podejściem jest stworzenie zbioru (słownika) najczęściej pojawiających się wyrazów. Co jednak zrobić, jeśli natrafimy w tekście na słowo spoza słownika?

Popularnym rozwiązaniem tego problemu jest tokenizacja tekstu na tzw. podsłowa (ang. *subwords*). W przypadku metody BPE podsłowa wybierana są automatycznie na podstawie częstotliwości ich występowania w tzw. korpusie treningowym: np. sama podstawa slowotwórcza “kot” oraz sam formant “-ek” w większości korpusów pojawią się dużo częściej niż ich połączenie w słowo “kotek”. BPE, oszczędzając miejsce w słowniku, zachowuje wskazane podsłowa przy pomocy których może odtworzyć znaczenia całego oryginalnego

Algorytm 3 BPE

```
V ← wszystkie unikalne tokeny w korpusie  $C$ 
for  $i = 0$  do  $k$  do
     $t_L, t_R \leftarrow$  najczęściej występująca para tokenów w  $C$ 
     $t \leftarrow t_L + t_R$ 
     $V \leftarrow V + t$ 
    Zamień wystąpienia  $t_L$  i  $t_R$  w  $C$  na  $t$ 
end for
```

wyrazu, dzięki czemu może poradzić sobie również z nieznanymi słowami konstruowanymi ze znanych podłów. BPE na podstawie zadanego korpusu treningowego tworzy słownik k najczęściej występujących podłów.

Schemat działania BPE prezentuje algorytm 3 (z pominięciem fazy pre-tokenizacji).

Praktyczny przebieg algorytmu jest następujący.

1. Początkowo tworzony jest słownik zawierający wszystkie unikalne symbole z korpusu, z pominięciem znaków białych. Dodatkowo, do słownika dodawany jest również specjalny token symbolizujący koniec wyrazu (np. symbol $_$).
2. W kolejnej fazie algorytmu, zwanej **pre-tokenizacją**, korpus jest wstępnie dzielony na wyrazy przy pomocy innej wybranej metody segmentacji tekstu. Tworzony jest zbiór wszystkich unikalnych wyrazów wraz z liczbą ich wystąpień. Do każdego wyrazu w zbiorze dodawany jest znak końca słowa. Następnie wyrazy dzielone są na poszczególne znaki.
3. Na podstawie otrzymanego zbioru, zliczane są wystąpienia par przylegających (zwłaszcza w obrębie jednego wyrazu) tokenów.
4. Najczęściej występująca para jest zastępowana nowym, niepodzielnym tokenem w całym tekście. Algorytm powraca $k - 1$ razy do kroku 1 (dokona się k złączeń).

Podstawowa implementacja BPE [17], wykorzystuje jednowymiarową tablice do przechowywania tekstu oraz tablice mieszającą (ang. *hash table*) do zapamiętywania tokenów wraz z ich częstotliwościami występowania. Przepisanie tekstu (tzn. zastąpienie starych tokenów nowym) jest operacją liniowo złożoną i proporcjonalną do długości tekstu N , liczoną w tokenach. Tym samym, wypełnienie słownika o rozmiarze V ma złożoność czasową $O(VN)$. Aktualizacja częstotliwości tokenów w tabeli mieszającej wykonywana jest wraz z procesem podmieniania tokenów w tekście. Stąd aktualizacja całej tabeli jest liniowo zależna od liczby podmian, których może być co najwyżej N . W rezultacie cały algorytm ma złożoność czasową $O(VN)$. Nowsze implementacje [44], umożliwiają, kosztem pamięci, redukcje złożoności czasowej do $O(V + N)$.

3.3.2 Zanurzenia słów

Reprezentacje wektorowe słów nazywa się często ich **zanurzeniami** [26]. Przedstawienie słowa w postaci elementu przestrzeni liniowej ma na celu odzwierciedlenie semantycznych relacji pomiędzy reprezentowanymi wyrazami przy pomocy pojęć geometrycznych. Jednym z podstawowych narzędzi są tu miary podobieństwa. Najczęściej używaną miarą

podobieństwa słów poprzez porównanie ich zanurzeń jest **podobieństwo cosinusowe**, określane jako stosunek iloczynu skalarnego wektorów do iloczynu ich norm euklidesowych (zob. równanie 3.1).

$$\cos \theta = \frac{A \cdot B}{\|A\| \|B\|} \quad (3.1)$$

Funkcja przyporządkowująca słowom ich zanurzenia powinna spełniać szereg własności. Naturalnym postulatem jest to, by znaczenie wyrazu było powiązane z sąsiedztwem jego zanurzenia. Stąd też np. zanurzenie słowa “szarlotka” powinno być bliższe zanurzeniu słowa “jabłko” niż zanurzeniu słowa “komunizm”. Ponadto, zanurzenia słów powinny kodować relacje pomiędzy synonimami i antonimami, np. podobieństwo pomiędzy wektorami synonimów “ciepły” i “gorący” powinno być dużo większe niż pomiędzy antonimami “ciepły” i “zimny”. W kontekście gier tekstowych ważną relacją, o jakiej wspominaliśmy, jest affordancia. Chcielibyśmy, żeby zanurzenia słów “piłka” i “mieszać” różniły się bardziej niż zanurzenia “piłka” i “kopać”.

Funkcje zanurzeń budowane są zazwyczaj w procesie uczenia maszynowego na zadanym korpusie, poprzez uwzględnienie szeregu postulatów zbliżonych do powyższych. Poniżej zarysujemy, bez wchodzenia w szczegóły, dwie popularne metody uczenia zanurzeń słów, bazujące na tym podejściu.

TF-IDF. TF-IDF (ang. *term frequency-inverse document frequency*) jest metodą w której zanurzenie określone jest na podstawie lokalnej i globalnej częstotliwości występowania wyrazu. Na wejściu algorytmu otrzymujemy korpus podzielony na N tekstów, nazywanych **dokumentami**, tokenizowanych wybraną metodą. Lokalną częstotliwość (ang. *term frequency*, **TF**) wyrazu t opisujemy poprzez liczbę jego wystąpień w konkretnym dokumencie d . Zwyczajowo, wartość TF jest dodatkowo skalowana logarytmicznie, zgodnie z równaniem 3.2.

$$TF_{t,d} = \log_{10}(count(t, d) + 1) \quad (3.2)$$

Globalność wyrazu (ang. *document frequency*, **DF**) to liczba dokumentów w których wystała. Na podstawie DF wartość **IDF** (ang. *inverse document frequency*) obliczana jest zgodnie z równaniem 3.3.

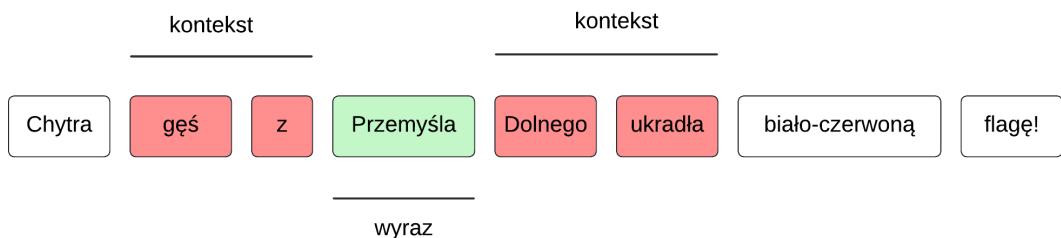
$$IDF_t = \log_{10} \left(\frac{N}{DF_t} \right) \quad (3.3)$$

Wartość **TF-IDF** definiujemy jako iloczyn TF i IDF (patrz równanie 3.4). Zanurzeniem wyrazu jest wektor, którego elementy są wartościami TF-IDF dla poszczególnych dokumentów (tym samym wymiar przestrzeni zanurzeń to N). Intuicyjnie, metoda TF-IDF stara się określić istotność wyrazu jako balans pomiędzy lokalną a globalną częstotliwością. Jeśli wyraz pojawia się często, ale w niewielkiej liczbie dokumentów, to możemy przypuszczać że jest lokalnie istotny i wnosi dużo informacji o dokumentach w których się znajduje.

Jeśli wyraz pojawia się często i w wielu dokumentach (jak np. spójniki), to prawdopodobnie jego znaczenie jest bardziej ogólne i słowo to nie jest kluczowe do zrozumienia danego dokumentu.

$$TF-IDF_{t,d} = TF_{t,d} \cdot IDF_t \quad (3.4)$$

Word2Vec. Word2Vec [31] jest pakietem oprogramowania¹ implementującym dwie metody nauki zanurzeń słów: **CBOW** (ang. *continuous bag-of-words*) oraz **Skip-gram**. Później przyjęło się odnosić do obu technik jako Word2Vec. Poniżej omówimy Skip-gram, co pozwoli nam zarysować główne pomysły leżące u podstaw obu metod.



Rysunek 3.3: Przykładowy wyraz i jego kontekst ($C = 2$)

Podstawowym pojęciem w omawianej metodzie jest kontekst słowa. Parametr C określa ile wyrazów następujących i poprzedzających dane słowo zawiera się w jego kontekście. Przykładowo, dla $C = 2$ zakres kontekstu obejmuje dwa wyrazy poprzedzające i dwa następujące po danym słowie (zob. rys. 3.3).

Wyraz	Etykieta	Klasa
Przemyśla	z	Pozytywna
Przemyśla	gęś	Pozytywna
Przemyśla	Dolnego	Pozytywna
Przemyśla	ukradła	Pozytywna
Przemyśla	Chytra	Negatywna
Przemyśla	biało-czerwoną	Negatywna
Przemyśla	flagę	Negatywna

Tablica 3.1: Dane treningowe dla słowa "Przemyśla" i zdania z rys. 3.3

Zarówno CBOW jak i Skip-gram są metodami **częściowo nadzorowanymi**. W uczeniu częściowo nadzorowanym model jest w stanie automatycznie nadać etykiety danym wejściowym. W celu zbudowania zbioru danych używany jest korpus treningowy, zaś z korpusu losowane są wyrazy i określone ich konteksty. Kolejnym krokiem jest etykietowanie:

- Etykieta klasy pozytywnej jest inny, zawierający się w kontekście, wyraz.
- Etykieta klasy negatywnej jest losowo wybrany wyrazy spoza kontekstu.

¹<https://code.google.com/archive/p/word2vec/>

Tablica 3.1 prezentuje przykład (etykietowanych) danych treningowych.

Nauka zanurzeń w Skip-gram polega na wytrenowaniu klasyfikatora oceniającego sąsiedztwo wyrazów. Najczęściej klasyfikatorem jest płytka (posiadająca 1-2 warstwy ukryte) sieć neuronowa. Sieć otrzymuje na wejściu pojedyncze słowo, na podstawie którego ma wskazać wyrazy mogące pojawić się w jego sąsiedztwie (traktowanym jako kontekst). Wyjściem sieci jest wektor o wymiarze odpowiadającym wielkości słownika. Każdy element wektora odpowiada poszczególnemu wyrazowi w słowniku. Funkcją aktywacji ostatniej warstwy sieci jest **Softmax** (patrz równanie 2.13), dzięki czemu wektor może być interpretowany jako rozkład opisujący prawdopodobieństwo, że dane słowo znajdzie się w kontekście wyrazu wejściowego. Celem treningu jest maksymalizacja wartości elementu odpowiadającemu etykietowanemu wyrazowi, z równoczesną minimalizacją wszystkich innych elementów. Po zakończeniu treningu, **wagi sieci** wykorzystywane są jako zanurzenia poszczególnych słów.

Rozdział 4

Transfer wiedzy w uczeniu przez wzmacnianie

W niniejszym rozdziale przyglądamy się bliżej problemowi transferu wiedzy w nauczaniu przez wzmacnianie; pełni on zarówno funkcję wprowadzenia w tematykę transferu wiedzy jak i przeglądu literatury.

4.1 Transfer wiedzy

Celem transferu wiedzy (ang. *knowledge transfer*), nazywanym również uczeniem przez transfer (ang. *transfer learning, TL*), jest wykorzystanie do wsparcia nauki nowego zadania informacji i doświadczeń zdobytych w trakcie procesu uczenia na innych zadaniach. Transfer wiedzy jest więc powiązany z kluczowym dla uczenia maszynowego problemem uogólniania modeli. W kontekście transferu wiedzy będziemy mówić o dwóch typach zadań:

- **Zadaniach źródłowych** (ang. *source tasks*): podczas ich rozwiązywania algorytm nauczania zdobywa “bazową” wiedzę.
- **Zadaniach docelowych** (ang. *target tasks*): podczas ich rozwiązywania “bazowa” wiedza używana jest (transferowana) do efektywniejszej nauki.

Kluczową zaletą nauki przez transfer jest możliwość wielokrotnego wykorzystania raz zdobytej wiedzy w różnych zadaniach docelowych. Wpływ transferu na proces nauki różni się w zależności od zastosowania; najczęściej obserwowanym efektem jest znaczące skrócenie czasu treningu.

Transfer wiedzy jest kluczowy dla wielu problemów nauczania przez wzmacnianie, zarówno praktycznych jak i badawczych. Przykładowo, bez możliwości transferu wiedzy z procesu uczenia w środowisku symulowanym do świata rzeczywistego trening pojazdu autonomicznego byłby niezwykle kosztowny i niepraktyczny.

Dalszą część rozdziału poświęcamy dokładnemu omówieniu różnych sposobów i podejść do transferu wiedzy w nauczaniu przez wzmacnianie. W pozostałej części tej sekcji zaprezentujemy krótko przykłady zastosowań w wybranych metodach uczenia maszynowego.

4.1.1 Nauka z nadzorem

W nauce z nadzorem transfer wiedzy umożliwił na znaczne zmniejszenie etykietowanych zbiorów danych wykorzystywanych przy nauczaniu nowych zadań. Jest to o tyle istotne, że etykietowanie jest niezwykle czasochłonnym i kosztownym procesem, często przeprowadzanym częściowo ręcznie. Poniżej omawiamy dwa popularne zastosowania transferu wiedzy w nauczaniu nadzorowanym.

1. **Wstępne trenowanie zanurzeń słów.** Bez transferu wiedzy nauka reprezentacji słów odbywa się wraz z treningiem zadania docelowego. Otrzymane w ten sposób zanurzenia powinny poprawnie oddawać znaczenie wyrazów w kontekście wykonywanego zadania. Na przykład reprezentacje wytrenowane na zadaniu analizy wydawnictwa będą odzwierciedlać sentyment wyrazów, pomijając inne zależności semantyczne. Taka specjalizacja może być zaletą, jednak każdorazowe trenowanie zanurzeń od zera jest niezwykle czasochłonne i wymaga ogromnej ilości danych treningowych.

Transfer wiedzy oferuje rozwiązywanie tych problemów w postaci wstępnie wytrenowanych zanurzeń słów (ang. *pretrained word embeddings*). W pierwszym kroku procesu wybrane zadania źródłowe wykorzystywane są do wstępnej nauki zanurzeń. Typ zadania uzależniony jest od metody wstępnego treningu: przykładowo, metoda Skip-gram wykorzystuje zadanie zgadywania kontekstu, opisanego w sekcji 3.3.2.

Wiedza zdobyta w trakcie wykonywania zadań źródłowych skrytalizowana jest w postaci modelu, często udostępnianego publicznie. Wstępnie wytrenowane zanurzenia mogą być wykorzystane bez zmian jako metody kodowania danych wejściowych, lub też dostrojone (ang. *fine tuned*) do nowego zadania.

Obok omawianego wcześniej Word2Vec, dużą popularnością cieszą się również metody FastText [8] oraz Glove [35]

2. **Ekstrakcja cech w rozpoznawaniu obrazów.** Kolejną dziedziną w której transfer wiedzy odniósł sukces jest rozpoznawanie obrazów. Aby lepiej zrozumieć zastosowaną metodę transferu przyjrzyjmy się ogólnej architekturze modeli używanych do pracy z obrazem. Model zwyczajowo składa się z dwóch komponentów: ekstraktora cech oraz tzw. głowy (ang. *model head*).

- **Ekstraktor cech** służy do wyboru nieredundantnych informacji (cech) kluczowych dla obrazu. Przykładowo, reprezentacja zdjęcia przy pomocy zbioru konturów lub kształtów może być bardziej efektywna niż traktowanie go jako mapy bitowej. Najczęściej ekstraktory są implementowane przy pomocy sieci konwojacyjnych.
- **Głowa modelu** wykorzystuje cechy otrzymane od ekstraktora do obliczenia finalnej odpowiedzi. Forma odpowiedzi różni się w zależności od zadania: w zadaniu klasyfikacji obrazów głową modelu może być klasyfikator, zaś odpowiedią przewidywana klasa.

Proces ekstrakcji cech zbliżony jest koncepcyjnie do zanurzeń słów, gdyż w obu przypadkach celem jest znalezienie reprezentacji skutecznie opisującej dany obiekt. Istnieje jednak zasadnicza różnica skali: w opisanych wcześniej metodach zanurzeń słów liczba reprezentacji jest ograniczona rozmiarem słownika. Dzięki temu mogą

być one raz wstępnie wytrenowane a następnie przechowane (także *explicite*, niekoniecznie w postaci modelu) do późniejszego użytku. W przypadku obrazu liczba możliwości jest znacznie większa, dlatego potrzebny jest model - ekstraktor który na bieżąco będzie tworzył odpowiednie reprezentacje

Stąd, w przeciwnieństwie do zanurzeń słów, zamiast konkretnych wytrenowanych wektorów transferujemy ekstraktor. Historycznie, popularnym było trenowanie ekstraktorów na zadaniach klasyfikacji obrazów przy wykorzystaniu popularnego zbioru danych ImageNet [12] (ponad 14 milionów zdjęć należących do 20 000 różnych klas).

Podobnie jak w przypadku zanurzeń słów, raz wyuczony ekstraktor może zostać użyty i dostrajany w całym szeregu zadań docelowych.

4.1.2 Nauka bez nadzoru

Sukces nauki przez transfer jest w obecnej chwili mniej spektakularny w uczeniu bez nadzoru, choć istnieją z powodzeniem wykorzystywane zastosowania. W szczególności, w zadaniach wykrywania anomalii wiedza zdobyta o strukturze pokrewnej dziedziny problemowej może zostać wykorzystana do lepszego znalezienia wartości odstających w drugiej. Przykładowo eksperyment przeprowadzony w [30] pokazał, że wiedza o anomalach może zostać transferowana pomiędzy różnymi typami silników odrzutowych.

4.2 Kształtowanie nagrody

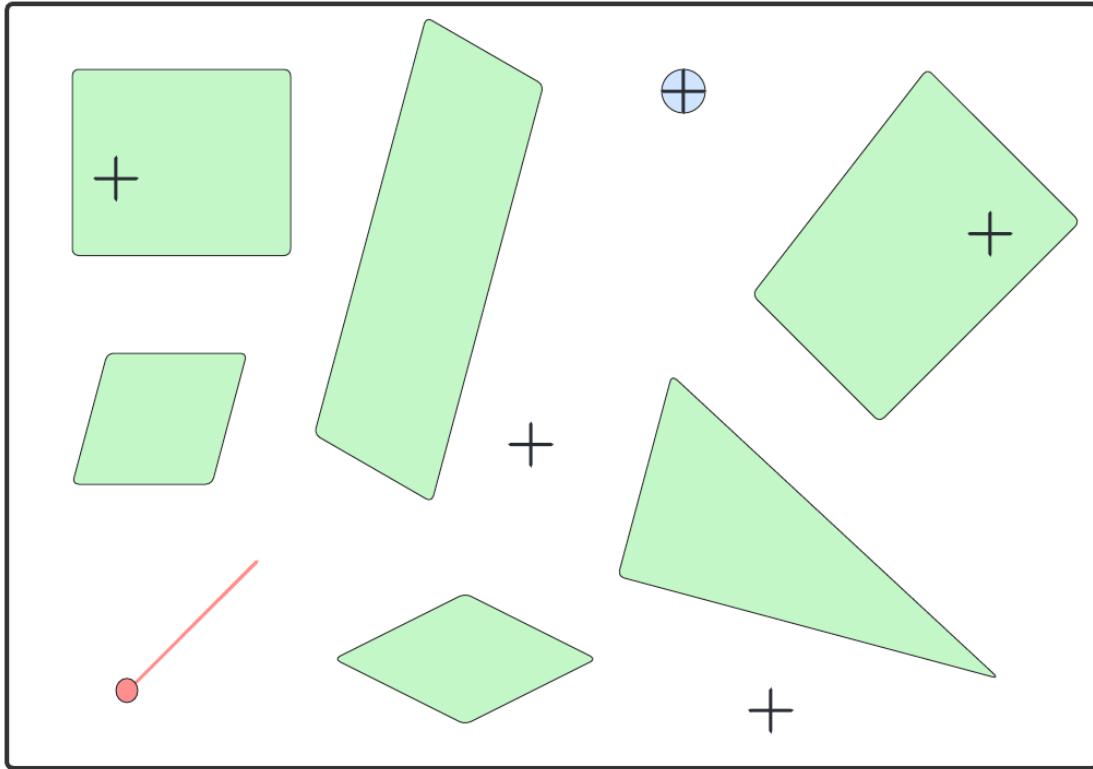
Omawiając wyzwania gier tekstowych (patrz sekcja 3.1), wspomnialiśmy o metodzie kształtowania nagrody (ang. *reward shaping*) [34]. Najczęstszym jej zastosowaniem jest uzupełnienie rzadko występującego sygnału nagrody o dodatkowe bodźce wspomagające agenta w eksploracji i eksploatacji środowiska.

Bodźce te, nazywane **nagrodami pośrednimi** (ang. *intermediate rewards*), są zwykle dobierane przez eksperta有条件的 daną dziedzinę problemową. Automatyczne kształtowanie nagrody jest aktywnym kierunkiem badawczym [24, 50, 49].

Sygnal nagrody powinien informować agenta o postępie zadania, unikając naprowadzania go na konkretny sposób rozwiązania. To podejście pomaga w zbudowaniu ogólnego rozwiązania i zbudowaniu wglądu w istotę środowiska i problemu. Dla kontrastu, agent otrzymujący głównie wskazówki pomagające jego zachowaniu zbliżyć się do już znanego rozwiązania może wyuczyć się polityki abstrahującej od celu wykonywanego zadania. Przykładowo, nagradzanie agenta za zbijanie figur w grze w szachy może doprowadzić do polityki skupiającej się nie na wygranej a na zdobyciu jak największej liczby figur.

W dalszej części sekcji przyjrzymy się wykorzystaniu kształtowania nagrody do transferu wiedzy w uczeniu przez wzmacnianie. Za przykład posłuży nam eksperyment przeprowadzony w publikacji Barto i Konidarisa [28], gdzie ukształtowany sygnał nagrody wyuczony na zadaniu źródłowym znacząco przyspieszył naukę zadania docelowego.

Zadanie źródłowe polega na pozycjonowaniu pręta znajdującego się w dwuwymiarowej przestrzeni opisanej przez współrzędne x i y poddane dyskretyzacji. Agent może przesuwać pręt o jedną jednostkę wzduż wybranej osi oraz wykonać jego rotację o 10° zgodnie lub przeciwnie do ruchu wskazówek zegara. Celem agenta jest doprowadzenia pręta do



Rysunek 4.1: Przykład zadania pozycjonowania pręta z [28]. Przeszkody oznaczone są jako zielone wielokąty, cel jako niebieskie koło, a radiostacje jako czarne krzyże.

określonego punktu w przestrzeni przy wykorzystaniu tych akcji, jednocześnie unikając jego kontaktu z rozmieszczonymi przeszkodami. Agent otrzymuje karę -1 za każdy wykonany ruch oraz nagrodę 1000 za dotarcie do celu. Źródłem informacji o środowisku (obok danych o położeniu i nachyleniu pręta) jest dla agenta ogólnie zdefiniowane "odczucie" (ang. *sensation*), obejmujące odczyty dotyczące środowiska oraz pomiary z pięciu radiostacji wysyłających naprowadzający sygnał (wyróżniający daną radiostację), słabnący z kwadratem odległości. Zakres współrzędnych oraz rozkład przeszkód są niezmienne w czasie trwania zadania. Radiostacje wysyłają sygnał informujący o odległości dzielącej je od agenta; jedna z nich jest zawsze zlokalizowana w punkcie celu, zaś rozkład pozostałych może posłużyć kształtowaniu nagrody. Rysunek 4.1 prezentuje przykładową wersję zadania.

Zadania źródłowe są losowo generowane. Agent rozwiązuje n zadań, każde z własną przestrzenią stanów notowaną jako S_j , gdzie $1 \leq j \leq n$. Poszczególne stany j -tego zadania określane są przez cztery atrybuty:

$$s_i^j = (d_i^j, c_i^j, r_i^j, v_i^j)$$

gdzie d_i^j jest dyskryminatorem pozwalającym rozróżnić stany w zadaniu; r_i^j i v_i^j są odpowiednio otrzymaną nagrodą oraz użytecznością stanu; atrybut c_i^j jest odczuciem agenta.

Agent trenowany jest przy pomocy algorytmu **SARSA** [38], tabelarycznej metody zbliżonej koncepcyjnie do Q-learning. W każdym zadaniu agent uczy się estymacji funkcji użyteczności V_j , przypisującej dyskryminatorowi użyteczność stanu.

$$V_j : d_i^j \mapsto v_i^j \quad (4.1)$$

Znaczenie i forma dyskryminatora d może zmieniać się w zależności od zadania. Rozkład przeszkołów pomiędzy zadaniami może się zmienić, więc dane koordynaty pręta w jednym zadaniu mogą być dozwolone a w innym nie. Stąd też funkcja V_j nie jest dobrym kandydatem do transferu wiedzy.

Znaczenie odczucia c ma być niezmienne; jest ono podstawą transferu wiedzy pomiędzy zadaniami. Założenie o uniwersalności znaczenia odczucia wynika z tego iż przyjmuje się, że polityka rozkładu radiostacji na planszy (a więc polityka kształtuowania nagrody) jest ustalona w grupach niezależnych eksperymentów. W celu dokonania bezpośredniego transferu wiedzy pomiędzy zadaniami korzysta się z funkcji L , estymującej użyteczność stanu na podstawie c :

$$L : c_i^j \mapsto v_i^j \quad (4.2)$$

W eksperymetach przeprowadzonych w [28] wykorzystano regresje liniową i nieliniową (model kwadratowy) w celu obliczenia funkcji L . Danymi treningowymi są (c_i^j, v_i^j) wygenerowane dla każdego stanu j -tego zadania, zaś budowa L odbywa się po zakończeniu treningu zadania źródłowego.

Zadanie docelowe jest podobne do źródłowego, jedyną różnicą jest większy rozmiar przestrzeni oraz większa liczba przeszkołów. Funkcja L wytrenowana na zadaniach źródłowych używana jest w zadaniu docelowym do wstępnej oceny użyteczności stanów.

Bardziej precyzyjnie: tabela używana przez SARSA w trakcie rozwiązywania zadania docelowego inicjalizowana jest wartościami otrzymanymi dzięki L . Wartości te mogą być interpretowane jako początkowe nagrody pośrednie, mogące ulec zmianie wraz z nauką zadania.

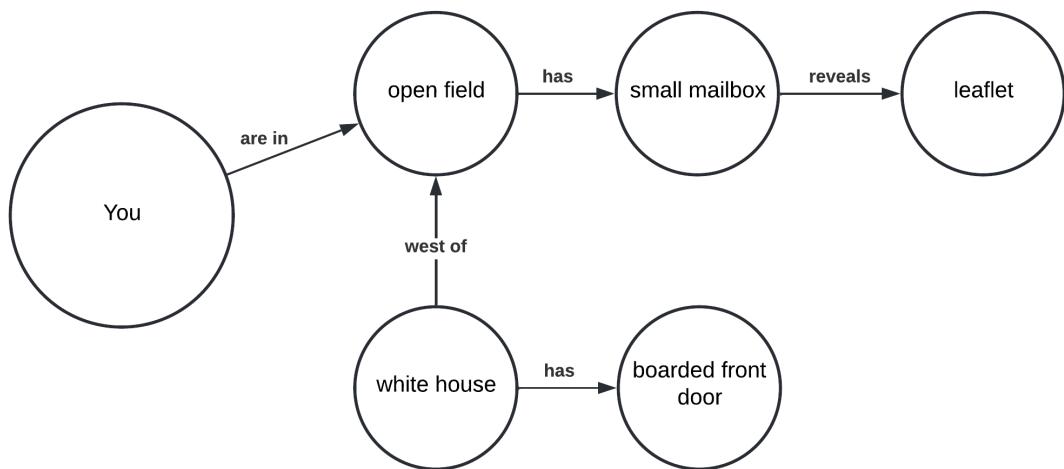
W eksperymetach przeprowadzonych w [28] agent trenowany był na 100 zadaniach źródłowych. Autorzy zauważyl, że transfer L miał największy wpływ na wczesne fazy treningu: w pierwszych epizodach agent osiągał cel znaczco szybciej (wykonywał mniej kroków). Jednocześnie metoda ta nie wpływała negatywnie na długotrwałą zbieżność algorytmu nauczania.

4.3 Transfer reprezentacji

Koncepcja transferu wiedzy poprzez transfer reprezentacji dotyczy grupy metod wykorzystujących wiedzę o strukturze środowiska zadań źródłowych do nauki zadań docelowych. Podstawą tego podejścia jest przyjęte założenie o niezmienności pomiędzy zadaniami części przestrzeni stanów, akcji, i/lub nagród [53], stąd też te elementy są głównym przedmiotem transferu. Z racji popularności metod nauczania opartych na estymowaniu funkcji użyteczności, często wykorzystywana metodą transferu reprezentacji jest przeniesienie

nie funkcji oceniającej użyteczność obserwacji z zadań źródłowych do zadań docelowych. Przykładowo, w Q-learningu transfer reprezentacji może opierać się na wykorzystaniu estymatora wyuczonego na zadaniu źródłowym do reprezentacji akcji (wykonywanych w danym stanie) w zadaniu docelowym.

Za przykład zastosowania transferu reprezentacji w zadaniach zbliżonych do tematu tej pracy rozważmy eksperyment przeprowadzony w publikacji [2]. Dziedziną problemową są gry tekstowe, zaś świat gry reprezentowany jest przy pomocy skierowanego, etykietowanego **grafu wiedzy** (ang. *knowledge graph*). Wierzchołkami grafu są obiekty napotkane w trakcie rozgrywki (np. przedmioty, pomieszczenia, sam gracz), a krawędziami relacje pomiędzy nimi. Można przyjąć, że graf definiowany jest poprzez zbiór trójelementowych krotek postaci: (**podmiot**, **relacja**, **obiekt**). Rysunek 4.2 przedstawia przykładowy graf dla wstępnej rozgrywki w Zork (zaprezentowanej w rysunku 3.1).



Rysunek 4.2: Poglądowy graf wiedzy dla wstępu do Zork (patrz rysunek 3.1)

Autorzy [2] pozyskują relacje tworzące graf przy pomocy technik **ekstrakcji informacji** (ang. *information extraction*) z tekstu. W tym celu posłużono się publicznie dostępnym narzędziem OpenIE [5] oraz zestawem ręcznie napisanych reguł, biorących pod uwagę specyfikę gier tekstowych. Budowanie grafu przebiega dwufazowo. Pierwsza faza, nazywana wysiewem (ang. *seeding*), korzysta z poradników do gier tekstowych w celu stworzenia wstępnego grafu. Poradniki te zawierają przykłady rozgrywki, informacje o często spotykanych przedmiotach i ich zastosowaniach oraz opis typowych zagadek z jakimi musi mierzyć się gracz. Druga faza obejmuje aktualizację grafu, przebiegającą wraz z nauką zadania źródłowego. Aktualizacja ta ma miejsce w każdym kroku rozgrywki; graf wiedzy rozszerzany jest o obserwacje stanu gry, czyli tekstu zwracanego przez silnik gry.

Grafy zbudowane na zadaniach źródłowych wykorzystano w [2] do lepszego zrozumienia świata gry w zadaniu docelowym. Co więcej, transfer grafów odbywa się w obrębie gier (oraz poradników) podobnych tematycznie. Przykładowo, wyróżnioną grupą tematyczną są gry typu "horror". Do nauki gier wykorzystano model **KG-DQN** (ang. *Knowledge Graph Deep Q-Network*) [1], dostosowany do pracy z grafami wiedzy. Graf wiedzy wykorzystano na dwa niezależne sposoby:

1. Po pierwsze, graf wiedzy zanurzany jest do postaci wektorowej przy pomocy techniki opisanej w [19] (której opis pomijamy). Tak zakodowany graf, rozumiany jako liczbowa reprezentacja wiedzy o świecie gry, użyty jest dalej przez sieci neuronowe w celu aproksymacji funkcji użyteczności.
2. Po drugie, graf wiedzy wykorzystuje się do oceny dopuszczalności komend. Komendy uznawane są za bardziej obiecujące (akceptowalne przez silnik gry), jeśli odpowiadająca im relacja znajduje się w grafie. Przykładowo, jeżeli krotka (`key`, `unlock`, `chest`) znajduje się w grafie wiedzy, to komenda `unlock chest with key` jest oceniona wyżej, niż `unlock banana with key`.

Wyniki eksperymentów przeprowadzonych w [2] wykazały, że zaproponowane metody skutecznie wspomogają naukę zadań docelowych. W eksperymetach tych transfer reprezentacji przyspieszył trening oraz pozwolił na osiągnięcie lepszych wyników niż miało to miejsce w bazowych przypadkach bez transferu grafu wiedzy. Co więcej, odkryto przypadki (gra *Anchorhead*), gdzie algorytm osiągał zbieżność tylko przy wykorzystaniu transferu reprezentacji.

4.4 Transfer polityki

Ostatnie omówione w niniejszej pracy podejście skupia się na wykorzystaniu wiedzy wcześniejszej wytrenowanego agenta (nazywanego **nauczycielem**) do wspierania treningu nowego agenta (nazywanego **uczniem**). Nauczyciel, wytrenowany na zadaniach źródłowych, przekazuje swoją wiedzę uczniowi, który wykorzystuje ją (pośrednio lub bezpośrednio) we własnej nauce. Najczęściej spotykana realizacja schematu nauczyciel-uczeń w uczeniu ze wzmacnieniem opiera się na przekazywaniu polityki. Zilustrujemy ją na przykładzie metody *Kickstarting* [42].

Kickstarting [42] jest metodą transferu polityki, która stara się upodobnić zachowanie ucznia do polityki nauczyciela. W tym celu funkcja błędu algorytmu uczącego modyfikowana jest o dodatkowy wyraz reprezentujący podobieństwo pomiędzy polityką uczniowską i nauczycielską. W pracy [42] używana jest miara **entropii krzyżowej** (ang. *cross-entropy*), opisana następującym równaniem:

$$H(p, m) = - \sum_{x \in X} p(x) \log m(x), \quad (4.3)$$

gdzie p i m to rozkłady prawdopodobieństwa określone dla zmiennej losowej X . Entropia krzyżowa [11] jest klasyczną miarą podobieństwa pomiędzy dwoma rozkładami, spełniającą nierówność $H(p) \leq H(p, m)$, gdzie $H(p)$ jest entropią (tj. średnią ilością informacji niesioną przez pojedyńczy sygnał) zmiennej losowej o rozkładzie p . Minimalizacja entropii krzyżowej polega więc na minimalizacji “zaskoczenia” odmiennością dwóch rozkładów.

Zaproponowana w [42] funkcja błędu algorytmu uczącego wyrażona jest równaniem 4.4:

$$\text{KickstartLoss}(\Theta) = \text{Loss}(\Theta) + \lambda_k H(\pi_T(a|s), \pi_S(a|s, \Theta)), \quad (4.4)$$

gdzie: (1) Loss jest dowolną funkcją straty; (2) π_T oraz π_S są odpowiednio polityką nauczyciela i ucznia; (3) Θ jest parametrem polityki ucznia; (4) $\lambda_k \in [0, 1]$ jest parametrem używanym do stopniowego usamodzielniania ucznia od nauczyciela.

Celem procesu jest minimalizacja wartości $\text{KickstartLoss}(\Theta)$ w funkcji parametru Θ . Dodatkowo, z każdym krokiem optymalizacyjnym k wartość λ_k jest zmniejszana tak aby po pewnym kroku t_0 uczeń był zupełnie uniezależniony od nadzoru nauczyciela ($\lambda_k = 0 \forall k > t_0$).

Kickstarting został przetestowany eksperymentalnie na zbiorze zadań **DMLab-30** [6] przy wykorzystaniu agenta **IMPALA** [14]. Przeprowadzone eksperymenty pokazały, że metoda znaczaco poprawia wyniki osiągane na zadaniach docelowych. Poprawa widoczna była w szczególności we wcześniejszych fazach treningu: agent korzystający z transferu polityki osiągał nawet o 50% lepsze wyniki niż odpowiednik trenowany od zera.

Rozdział 5

Narzędzia

Niniejszy rozdział stanowi wprowadzenie do narzędzi, które wykorzystujemy w Rozdziale 6 do budowy zestawu eksperymentów mających na celu zbadanie wybranych podejść do transferu wiedzy w grach tekstowych. Zaczynamy od przyjrzenia się **TextWorld** [10], narzędziu do generowania gier tekstowych oraz treningu i ewaluacji agentów. Następnie przedstawiamy dokładny opis architektury sieci Deep Reinforcement Relevance Network (DRRN), którą używamy do nauki agentów, zgodnie ze schematem głębokiego Q-learning.

5.1 TextWorld

TextWorld [10] jest biblioteką dla języka Python, umożliwiającą efektywne badanie generalizacji i transferu wiedzy w kontekście nauki gier tekstowych. Oprogramowanie umożliwia generowania zestawów podobnych, lecz nie identycznych gier oraz elastyczny interfejs programistyczny ułatwiający komunikację pomiędzy agentem a grą.

Domyślnie TextWorld oferuje implementacje generatorów dla czterech rodzajów gier: *Simple*, *Coin Collector*, *Cooking Game*, *Treasure Hunter*. Każdy generator przyjmuje szereg parametrów pozwalających na specyfikację gier, umożliwiając m.in. kontrolę liczby pomieszczeń i przedmiotów, rodzaju i długości zadań, a nawet częstotliwości sygnału nagrody (sygnał może być rzadki, gęsty lub zbalansowany). TextWorld umożliwia również prostą integrację własnego generatora gier.

Struktura tworzonych gier spełnia założenia procesów decyzyjnych Markowa, przy czym przejścia pomiędzy poszczególnymi stanami gry definiowane są przy użyciu **logiki liniowej** [39]. W niniejszej pracy pomijamy szczegóły dotyczące logiki liniowej, wspominając tylko iż za jej wyborem jako silnika wnioskowania stoi szczególna wersja implikacji, zaproponowana przez tę logikę, która w naturalny sposób pozwala specyfikować wykorzystanie zasobów. Silnik wnioskowania zapewnia możliwość efektywnego sprawdzenia osiągalności wszystkich stanów, w szczególności tych wygrywających. Generowane gry kompatybilne są z **Inform7** [33], szeroko przyjętym językiem programowania do tworzenia interaktywnej fikcji, co pozwala na ich użycie poza TextWorld.

TextWorld umożliwia również interakcje agentów z dowolną inną grą tekstową kompatybilną z Inform7. Ten sam interfejs programistyczny pozwala na naukę i ewaluację agentów zarówno na prostych, generowanych grach tekstowych jak i na pełnoprawnych tytułach,

takich jak Zork [4]. Listing 5.1 przedstawia przykład interakcji pomiędzy agentem a grą tekstową poprzez TextWorld.

Listing 5.1: Przykład interakcji agenta z grą w TextWorld

```
import textworld
environment = textworld.start('zork1.z5')
agent = textworld.agents.RandomCommandAgent()

state = environment.reset()
reward, done = 0, False

while not done:
    command = agent.act(state, reward, done)
    state, reward, done = environment.step(command)
```

5.2 DRRN

DRRN (ang. *Deep Reinforcement Relevance Network*) [22] jest architekturą sieci neuronowej przeznaczoną do aproksymacji funkcji użyteczności q_* . Architektura ta jest specjalnie dostosowana do zagadnienia gier tekstowych - środowisk o przestrzeni akcji i stanów określonej poprzez język naturalny. DRRN wykorzystano, w różnych wariantach, w szeregu publikacji [51, 21, 20]. Poniżej omawiamy oryginalną [22] wersję architektury oraz wykorzystany przez nas wariant DRRN zaproponowany w publikacji towarzyszącej oprogramowaniu Jericho [21].

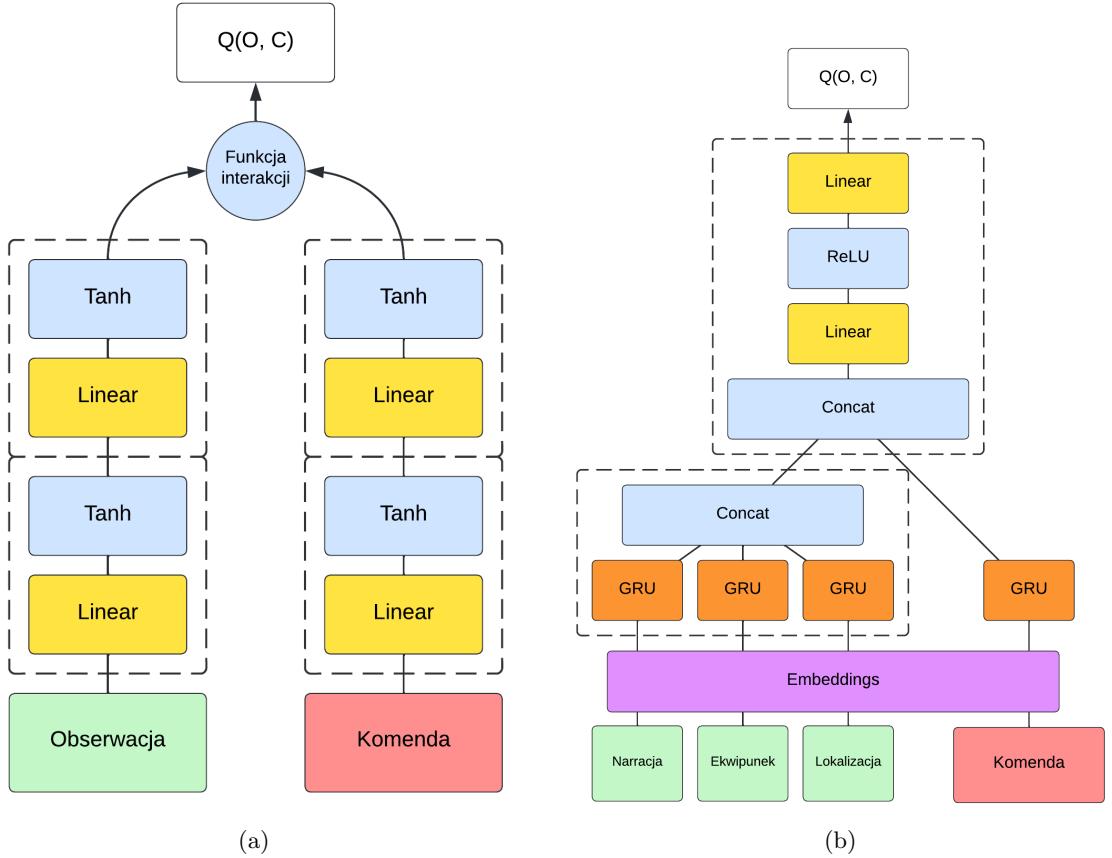
5.2.1 Dwa warianty architektury DRRN

DRRN działa dwuetapowo. W pierwszym kroku obserwacja stanu o oraz wybrana komenda c zanurzane są do postaci wektorowej, przy czym sieć wykorzystuje oddzielne zestawy zanurzeń do reprezentowania komend i obserwacji. W drugim kroku estymowana jest użyteczność $Q(o, c)$, przy wykorzystaniu otrzymanych zanurzeń i wybranej **funkcji interakcji**.

Oryginalna wersja DRRN. Oryginalny DRRN używa do zanurzania słów dwóch oddzielnich głębokich sieci neuronowych, zwyczajowo nazywanych **enkoderami**. Autorzy [22] nie specyfikują etapu wstępnego przetwarzania tekstu do postaci liczbowej, użytecznej dla sieci neuronowych. Jedno z typowych podejść do wstępnego przetwarzania tekstu opisujemy dalej, przy okazji przedstawiania zmodyfikowanej wersji architektury.

Schemat pierwotnej architektury DRRN zaprezentowano na Rys. 5.1(a). Każdy enkoder składa się z dwóch dwuwarstwowych bloków złożonych z warstwy gęsto połączonych neuronów (oznaczonej na schemacie jako *Linear*), której wyjście kierowane jest do funkcji aktywacji \tanh (gdzie $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$). Formalnie wyjście warstwy opisane jest równaniem $Y = f(WX + b)$, gdzie X jest macierzą wejściową a W macierzą wag, b wektorem progów, zaś f funkcją aktywacji.

Funkcja interakcji jest wymienna. W [22] wykorzystano w jej miejscu iloczyn skalarny, ale,



Rysunek 5.1: Architektura DRRN: schemat z oryginalnej publikacji [22] (a) oraz schemat zmodyfikowanego wariantu [21] (b)

jak zobaczymy dalej, może to być również sieć neuronowa. Wyjściem funkcji jest wartość skalarna reprezentująca estymację użyteczności $Q(o, c)$.

Zmodyfikowana wersja DRRN. Drugi wariant architektury [21] zaprezentowano na Rys. 5.1(b). Pierwszą widoczną różnicą jest liczba argumentów wejściowych. W tej wersji sieci obserwacje rozbite są na trzy składowe.

1. Narrację, czyli tekst zwracany przez silnik gry po wykonaniu poprzedniej komendy.
2. Ekwipunek, czyli opis zawartości ekwipunku, zwyczajowo otrzymywany przy pomocy komendy `inventory`.
3. Lokalizację, czyli opis lokalizacji w jakiej znajduje się gracz, zwyczajowo otrzymywany przy pomocy komendy `look`.

Tekst jest wstępnie przetwarzany w następujący sposób. W pierwszym kroku treść argumentów wejściowych jest tokenizowana. W naszej implementacji wykorzystaliśmy w tym celu publicznie dostępny tokenizator BPE¹. W kolejnym kroku tokeny zamieniane są na

¹https://huggingface.co/docs/transformers/model_doc/gpt2#transformers.GPT2Tokenizer

unikalne identyfikatory i kodowane do postaci **one-hot** (wektory mające zera na wszystkich poza jedną pozycją, gdzie znajduje się jedynka). Wektory one-hot agregowane są w macierzy kodowań K o wymiarach $\mathcal{K} \times \mathcal{V}$, gdzie \mathcal{K} to liczba tokenów, a \mathcal{V} to rozmiar słownika.

Kodowania przekazywane są do warstwy *Embeddings*, która przechowuje macierz zanurzeń E . Macierz ma wymiary $\mathcal{V} \times \mathcal{E}$, gdzie \mathcal{E} to rozmiar wektorów zanurzeń. Na początku treningu, E inicjalizowana jest losowo, a następnie modyfikowana wraz z procesem wstępnej propagacji błędu. Wyjściowa macierz, będąca zanurzeniem początkowego tekstu, otrzymywana jest wykonując mnożenie $K \times E$, co odpowiada wyborowi odpowiednich wierszy z macierzy zanurzeń.

Przekształcenia 5.1 prezentują przykładowy proces wstępnego przetworzenia i zanurzenia tekstu “Chytra gęś biega”.

$$\underbrace{\begin{bmatrix} \text{Chytra} \\ \text{gęś} \\ \text{biega} \end{bmatrix}}_{\text{Tokeny}} \rightarrow \underbrace{\begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}}_{\text{Id}} \rightarrow \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \end{bmatrix}}_{\text{Macierz kodowań}} \times \underbrace{\begin{bmatrix} 0.33 & 0.28 & 0.90 & 0.14 \\ 0.11 & 0.56 & 1.40 & 0.83 \\ 0.45 & 1.23 & 1.50 & 1.66 \\ 0.37 & 1.73 & 1.22 & 0.64 \\ \vdots & \vdots & \vdots & \vdots \\ 0.22 & 1.01 & 0.23 & 0.74 \\ 0.17 & 1.33 & 1.12 & 2.04 \end{bmatrix}}_{\text{Macierz zanurzeń}} = \underbrace{\begin{bmatrix} 0.11 & 0.56 & 1.40 & 0.83 \\ 0.33 & 0.28 & 0.90 & 0.14 \\ 0.37 & 1.73 & 1.22 & 0.64 \end{bmatrix}}_{\text{Zanurzenie}} \quad (5.1)$$

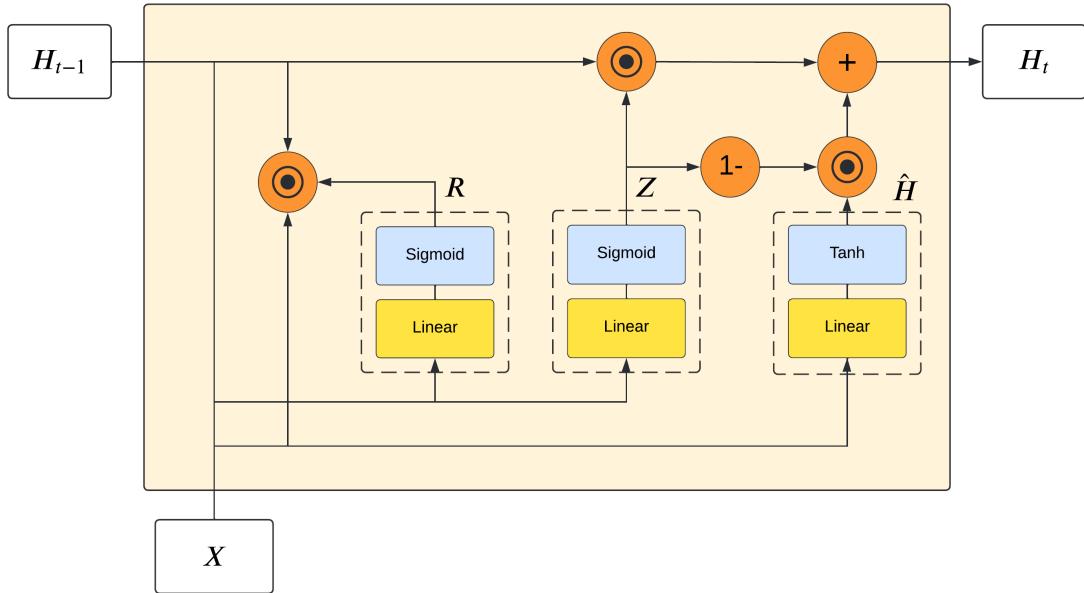
Zanurzona treść komendy, narracji, ekwipunku i lokalizacji przetwarzana jest przez odrebnego enkodery, implementowane przy pomocy warstwy **GRU** (opisanej dalej). Wyjście enkoderów jest łączone (warstwy *Concat*) a następnie przekazywane do funkcji interakcji. W opisywanej wersji architektury funkcja interakcji jest głęboką siecią neuronową, złożoną z dwóch warstw gęsto połączonych neuronów. Pierwsza warstwa używa funkcji aktywacji ReLU (ang. *Rectified Linear Unit*), zaś druga zwraca “surowe”, nieprzetworzone wyjście, reprezentujące estymacje użyteczności $Q(o, c)$.

GRU. Warstwa GRU (ang. *Gated Recurrent Unit*) jest przykładem sieci rekurencyjnej, gdzie wewnętrzny stan ukryty H , interpretowany jako długotrwała pamięć sieci, przekazywany jest z wyjścia z powrotem na wejście. Architektury rekurencyjne zaprojektowane zostały z myślą o pracy na danych sekwencyjnych, tzn. takich gdzie kolejność elementów ma znaczenie. Przykładem takich danych jest ciąg odwiedzanych stanów i wykonywanych akcji w procesie decyzyjnym Markowa.

Rys. 5.2 przedstawia schemat wewnętrznej struktury GRU. Warstwa składa się z trzech połączonych ze sobą bramek (jednowarstwowych sieci neuronowych).

- Bramka resetująca R odpowiedzialna jest za “zapominanie” informacji. Bardziej skrótnie, w każdym kroku sekwencji t bramka resetująca decyduje, które informacje z poprzedniego stanu ukrytego H_{t-1} oraz obecnego wejścia X_t powinny zostać pominięte w H_t .
- Bramka aktualizująca Z odpowiedzialna jest za zapamiętywanie informacji. Na podstawie poprzedniego stanu ukrytego H_{t-1} oraz obecnego wejścia X_t , bramka aktualizująca decyduje, które informacje powinny zostać zawarte w H_t .

- Bramka aktywacji \hat{H} , odpowiedzialna jest za generowanie “kandydata” na nowy stan ukryty. Stan generowany jest na podstawie przeszłego stanu ukrytego H_{t-1} , obecnego wejścia X_t oraz wyjścia bramki resetującej R_t .



Rysunek 5.2: Schemat warstwy GRU

Formalnie sieć GRU opisywana jest przy pomocy równań 5.2.

$$\begin{aligned}
 Z_t &= \sigma(W_z X_t + U_z H_{t-1} + b_z) \\
 R_t &= \sigma(W_r X_t + U_r H_{t-1} + b_r) \\
 \hat{H}_t &= \tanh(W_h X_t + U_h (R_t \odot H_{t-1}) + b_h) \\
 H_t &= Z_t \odot \hat{H}_t + (1 - Z_t) \odot H_{t-1}
 \end{aligned} \tag{5.2}$$

Gdzie W_n, U_n oraz b_n , dla $n \in \{z, r, h\}$, to macierze wag i wektorów progów poszczególnych bramek. Symbole Z_t , R_t , \hat{H}_t to odpowiednio macierze wyjściowe, w kroku t , dla bramki aktualizującej, resetującej oraz aktywacji. Macierz wejścia oznaczamy jako X_t , a poprzednią i nową macierz stanu ukrytego jako H_{t-1} oraz H_t . Operator \odot oznacza iloczyn Hadamard'a (iloczyn po współrzędnych), zaś $\sigma(\cdot)$ to sigmoidalna funkcja aktywacji, gdzie $\sigma(x) = \frac{1}{1+e^{-x}}$ dla $x \in \mathbb{R}$.

5.2.2 Algorytm uczenia

W niniejszej części rozdziału przedstawimy algorytmowi uczenia DRRN (zob. Alg. 4), będący zmodyfikowaną wersją Alg. 1 z pracy [22].

Algorytm 4 Algorytm uczenia DRRN^a

- 1: Inicjalizuj pusty priorytetowy bufor powtórek \mathcal{D} o pojemności N i parametrze η
- 2: Inicjalizuj **sieć aktywną**, reprezentującą funkcję $Q(o, c; \Theta)$, losowymi parametrami Θ
- 3: Inicjalizuj **sieć docelową**, reprezentującą funkcję $Q(o, c; \Theta')$, kopią parametrów sieci aktywnej $\Theta' \leftarrow \Theta$
- 4: **for** $epizod = 1, \dots, E$ **do**
- 5: Zresetuj grę
- 6: Odczytaj początkową obserwację o_1 oraz zbiór akceptowanych komend $C(o_1)$
- 7: **for** $t = 1, \dots, T$ **do**
 // Wykonanie komendy i obserwacja wyników
- 8: Wylicz wartości $Q(o_t, c; \Theta)$ dla wszystkich $c \in C(o_t)$
- 9: Wylicz $c_t \leftarrow \begin{cases} \text{losowa komenda z } C(o_t) & \text{z prawdopodobieństwem } \epsilon \\ \operatorname{argmax}_{c'} Q(o_t, c'; \Theta) & \text{w przeciwnym razie} \end{cases}$
- 10: Wykonaj komendę c_t
- 11: Odczytaj obserwację o_{t+1} , akceptowane komendy $C(o_{t+1})$ oraz nagrodę r_t
- 12: Dodaj przejście $(o_t, c_t, r_t, o_{t+1}, C(o_{t+1}))$ do \mathcal{D}
- 13: // Uczenie na podstawie zebranych doświadczeń
- 14: Wylosuj próbki $\mathcal{P} = \{(o_j, c_j, r_j, o_{j+1}, C(o_{j+1}))\}$ z \mathcal{D}^b
 Dla przejść w \mathcal{P} oblicz $y_j \leftarrow \begin{cases} r_j & \text{jeśli } o_{j+1} \text{ kończy grę} \\ r_j + \gamma \max_{c'} Q(o_{j+1}, c'; \Theta') & \text{w przeciwnym razie} \end{cases}$
- 15: Wylicz błąd HuberLoss($Q(o_j, c_j; \Theta), y_j$)
- 16: Zmodyfikuj parametry Θ wykonując krok uczenia dla obliczonego błędu^c
- 17: **end for**
- 18: **if** w kroku T uzyskano wynik wyższy od dotychczasowego maksimum **then**
- 19: Usuń wszystkie przejścia z priorytetowej części \mathcal{D}
- 20: **end if**
- 21: **if** w kroku T uzyskano wynik nie mniejszy od dotychczasowego maksimum **then**
- 22: Dodaj wszystkie przejścia (przebieg) epizodu do priorytetowej części \mathcal{D}^d
- 23: **end if**
- 24: Zmniejsz ϵ
- 25: Co U epizodów, aktualizuj parametry sieci docelowej $\Theta' \leftarrow \Theta$
- 26: **end for**

^abazujący na Alg. 1 z pracy [22]

^bwielkość próbki $|\mathcal{P}|$ jest parametrem algorytmu

^cnp. używając metody spadku gradientu

^dusuwając odpowiednią liczbę przejść, jeśli pojemność \mathcal{D} została przekroczena

Pierwszym krokiem jest inicjalizacja priorytetowego bufora powtórek \mathcal{D} o pojemności N (linia 1). W odróżnieniu od klasycznego bufora powtórek (zob. Sek. 2.4.2), \mathcal{D} ma wydzieloną część na priorytetowe przejścia. Pojemność części priorytetowej określana jest parametrem $\eta \in [0, 1]$ i wynosi $\eta \cdot N$ próbek.

Przykładowo, w przeprowadzonych eksperymentach przyjmujemy $\eta = 0.5$, co oznacza, że połowa bufora próbek przeznaczona jest na przejścia priorytetowe. Opisany \mathcal{D} jest uproszczeniem priorytetowego bufora zaproponowanego pierwotnie w [41], gdzie priorytet

przejścia określany jest bezpośrednio na podstawie błędu TD.

Kolejnym krokiem algorytmu jest inicjalizacja parametrów dwóch sieci: **sieci docelowej** (ang. *target network*), której parametry oznaczamy przez Θ' oraz **sieci aktywnej** której parametry oznaczamy przez Θ (linie 2 i 3). Sieć docelowa współdzieli strukturę z siecią aktywną i podczas inicjalizacji obie sieci otrzymują parametry o tych samych wartościach. Sieć docelowa używana jest do generowania wartości docelowych y , czyli potencjalnie lepszych estymacji funkcji użyteczności. Co U epizodów wartości parametrów sieci docelowej są aktualizowane wartościami parametrów sieci aktywnej (linia 25).

Nauka trwa E epizodów. Dla każdego epizodu emulator gry jest resetowany, a następnie odczytywana jest wstępna obserwacja o_1 oraz zbiór akceptowanych komend $C(o_1)$ (linie 5 i 6). Proces odczytania obserwacji składa się z trzech etapów.

1. Narracja odczytywana jest bezpośrednio z tekstu zwróconego przez emulator gry
2. Opis ekwipunku otrzymywany jest poprzez wykonanie komendę `inventory`
3. Opis lokalizacji jest efektem wykonania komendy `look`

Zbiór $C(o_t)$ zwracany jest bezpośrednio przez emulator gry. Należy zaznaczyć, że pominięcie etapu samodzielnego tworzenia akceptowanych komend jest znaczącym uproszczeniem problemu nauki gier tekstowych. Niemniej w literaturze [22, 21] uproszczenie to jest często wykorzystywane jeśli nie kłoci się z istotą badania. W eksperymentach przeprowadzonych na potrzeby niniejszej pracy skupiamy się na transferze wiedzy, stąd generowanie komend zostało pominięte.

W wewnętrznej pętli dla każdego kroku t , wyliczamy Q wartości dla wszystkich komend akceptowanych w bieżącej obserwacji o_t (linia 8). Następnie, przy pomocy polityki eksploracji ε -greedy wybieramy komendę c_t (linia 9). Pierwotna wersja algorytmu stosowała politykę eksploracji Boltzmanna. W implementacji wykonanej na potrzeby tej pracy dokonano zmiany metody eksploracji. Decyzję tę podjęto w wyniku obserwacji wstępnych eksperymentów: metoda ε -greedy dawała znacząco lepszą zbieżność algorytmu.

Po wyborze komendy jest ona wykonywana. Następnie odczytywana jest nowa obserwacja o_{t+1} , nowy zbiór akceptowanych komend $C(o_{t+1})$ oraz otrzymana nagroda r_t (linia 11). Utworzone przejście $(o_t, c_t, r_t, o_{t+1}, C(o_{t+1}))$ zapisywane jest w niepriorytetowej części \mathcal{D} (linia 12).

Po dodaniu nowego przejścia losowana jest próbka przejść \mathcal{P} , gdzie wielkość próbki $|\mathcal{P}|$ jest parametrem algorytmu (linia 13). Dla każdego wylosowanego przejścia obliczane są wartości docelowe y_j przy wykorzystaniu sieci docelowej (linia 14). Na podstawie wartości docelowych oraz predykcji $Q(o_j, c_j; \Theta)$, otrzymanych wykorzystując sieć aktywną, obliczana jest wartość funkcji błędu (linia 15). W omawianym algorytmie wykorzystano funkcje błędu Hubera z parametrem δ , widoczną w równaniu 5.3. Uczenie sieci aktywnej przebiega poprzez **minimalizację błędu Hubera**.

$$\text{HuberLoss}(p, y) = \begin{cases} \frac{1}{2}(y - p)^2 & \text{jeśli } |y - p| \leq \delta \\ \delta(|y - p| - \frac{1}{2}\delta) & \text{w przeciwnym razie} \end{cases} \quad (5.3)$$

Zaletą funkcji błędu Hubera w porównaniu do bardziej popularnego błędu średniokwadratowego jest to, że jest ciągła i różniczkowalna dla wszystkich wartości.

dratowego (ang. *MSE*) jest jego mniejsza wrażliwość na wartości odstające. Ponadto, w przeciwnieństwie do średniego błędu bezwzględnego (ang. *MAE*) jest też różniczkowalny w zerze. Własności te sprawiają że błąd Hubera umożliwia bardziej stabilną zbieżność algorytmu w porównaniu do wymienionych alternatyw [48]. Równanie 5.3 formalnie opisuje błąd Hubera pomiędzy wartościami docelowymi y i predykcjami p . Dla predykcji zbliżonych do wartości docelowych, wykorzystuje się błąd kwadratowy, w przeciwnym razie obliczany jest błąd absolutny. Parametr δ kontroluje wybór sposobu obliczania błędu.

Z każdym rozegranym epizodem, wartość ε jest liniowo zmniejszany (linia 24), poczynając od wartości ε_{start} a kończąc na ε_{stop} . Dodatkowo, jeśli agent osiągnął wynik lepszy od dotychczasowego maksimum, to z priorytetowej części \mathcal{D} usuwane są wszystkie przejścia. Jeżeli agent uzyskał wynik nie mniejszy od dotychczasowego maksimum, to cały przebieg epizodu (tzn. wszystkie jego przejścia) dodawane są do priorytetowej części \mathcal{D} .

5.2.3 Uwagi do implementacji

W ramach niniejszej pracy przygotowano pełną implementację Alg. 4. Kod naszej implementacji oparty jest na kodzie z repozytoriów²³ towarzyszącym publikacjom [21, 51].

Na potrzeby niniejszej pracy, oprogramowanie dostosowane zostało do interfejsu wykorzystywanego przez bibliotekę TextWorld, co wymagało istotnych zmian w jego strukturze. W szczególności, zmiany te objęły sposób odczytywania i kodowania (tokenizacji i zanurzania) obserwacji i komend. W odróżnieniu od bazowych implementacji, nasz kod wykorzystuje sieć docelową oraz metodę eksploracji ε -greedy z liniowo zmniejszaną wartością ε . Logika dla obu komponentów została zaimplementowana przez nas od podstaw. Dodatkowo, na potrzeby przeprowadzanych eksperymentów, priorytetowy bufor powtórek został zaadaptowany tak, aby umożliwić naukę wielu różnych gier w ramach jednego treningu.

²<https://github.com/microsoft/tdqn>

³<https://github.com/princeton-nlp/calm-textgame>

Rozdział 6

Próby transferu wiedzy w grach tekstowych

W niniejszym rozdziale przedstawiamy zaproponowane i zrealizowane przez nas metody transferu wiedzy w grach tekstowych. W szczególności, omawiamy zbudowany przez nas zbiór treningowy gier tekstowych oraz prezentujemy i omawiamy otrzymane wyniki.

6.1 Wygenerowane gry

Przy pomocy TextWorld wygenerowaliśmy zbiór treningowy gier typu *Cooking Game*, w których wspólnym celem rozgrywki jest zjedzenie posiłku przyrządzonego ze składników rozrzuconych po świecie gry. Przyjęto iż posiłek musi zostać przygotowany przez agenta zgodnie z przepisem, tzn. wszystkie składniki muszą zostać przygotowane zgodnie z instrukcją. Każdy ze składników może zostać zjadły albo upieczone (przy użyciu piekarnika) albo ugotowany (przy wykorzystaniu kuchenki). Aby wygrać, agent musi znaleźć w świecie gry odpowiednie składniki, przygotować je, a następnie przygotować (wykonując komendę `prepare meal`) i skonsumować (wykonując komendę `eat meal`) posiłek. Zjedzenie wymaganego składnika lub jego złe przyrządzenie, np. upieczenie marchewki która powinna być ugotowana, prowadzi do przegranej. Kolejność wykonywania instrukcji przepisu nie ma znaczenia.

Wygenerowano dwie grupy gier: o niższym i o wyższym poziomie trudności. Świat gier łatwiejszych zawiera 6 pomieszczeń, zaś przepis wymaga znalezienia dwóch konkretnych składników spośród 7 rozmieszczonych w różnych pokojach. W trudniejszym wariantie świata gry składa się z 9 pomieszczeń, a przepis wymaga przyrządzenia trzech składników spośród 9 rozmieszczonych w różnych pokojach. Trudne gry wymagają od gracza dodatkowej umiejętności otwierania niektórych obiektów, np. drzwi, lodówki i szafek kuchennych. Liczba obiektów i możliwych z nimi interakcji sprawia, że utworzone środowiska charakteryzują się niezwykle dużą przestrzenią akcji: już w prostszych grach można znaleźć stany, gdzie liczba akceptowanych komend przekracza 40.

W celu większej kontroli nad losowymi elementami gier użyto własnej, opartej na oryginalnej [10], implementacji generatora gier typu *Cooking Game*. W szczególności zapewniono, że gry generowane z tym samym ziarem losowości (ang. *seed*) mają podobne przepisy

niezależnie od poziomu trudności. Ścisłej: każdej grze łatwej przyporządkowano jej trudniejszy odpowiednik o własności takiej, że przepis w grze trudnej jest rozszerzeniem przepisu z gry łatwej. Przykładowo, jeśli przepis w prostszej grze wymaga upieczenia marchwi oraz ugotowania nogi kurczaka, to przepis w grze trudniejszej obejmuje te same instrukcje oraz jedną dodatkową (dla nowego składnika). Dzięki temu mechanizmowi zapewniono, że przestrzeń stanów i akcji gier trudnych i ich łatwiejszych odpowiedników przynajmniej częściowo się pokrywa.

W każdej z gier prostych do zdobycia jest 6 punktów, po dwa na każdy składnik (zebranie i przetworzenie) i po jednym za finalne przygotowanie i skonsumowanie posiłku. W grach trudnych uzyskać można 8 punktów (jeden dodatkowy składnik).

Zgodnie z powyższymi założeniami wygenerowano 5 gier łatwych i 5 ich trudniejszych odpowiedników. Ponieważ w naszych badaniach interesuje nas głównie transfer wiedzy, odbywający się w fazie treningu, pomineliśmy generacje zbioru walidacyjnego i testowego.

6.2 Cztery metody transferu wiedzy

Omówimy badane w naszych eksperymentach metody transferu wiedzy, dzieląc je ze względu na sposób treningu nauczyciela (dwa warianty) oraz ze względu na sposób nadzoru ucznia (również dwa warianty). W ramach przeprowadzonych eksperymentów przetestowano wszystkie możliwe konfiguracje. Dla każdego sposobu treningu nauczyciela sprawdziliśmy wszystkie sposoby nadzoru, co w sumie dało cztery różne metody transferu wiedzy.

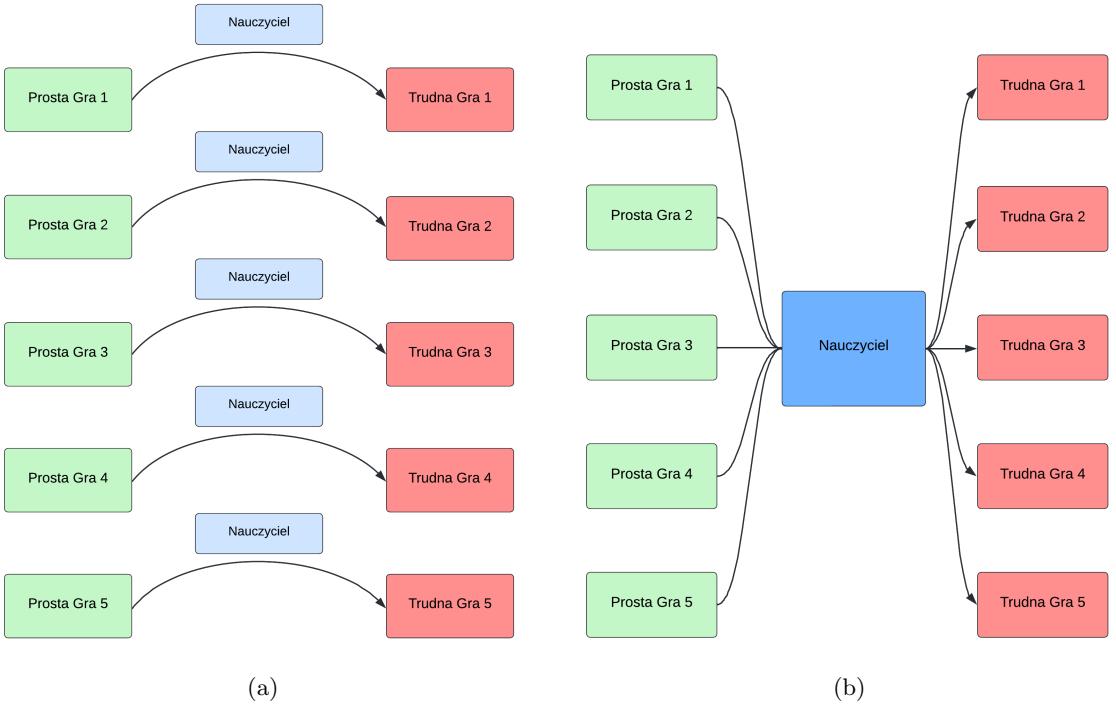
6.2.1 Nauczyciel uogólniony i wyspecjalizowany

W naszych eksperymentach rozróżniamy dwa sposoby treningu nauczyciela. Pierwszy sposób, nazwany przez nas **transferem wyspecjalizowanym**, polega na wytrenowaniu oddzielnego nauczyciela dla każdej prostej gry, a następnie użyciu go do nadzoru ucznia w jej trudniejszym odpowiedniku. Naszą wstępna hipotezą było to, że nauczyciele wyspecjalizowani, posiadający bardzo konkretną wiedzę o bazie przepisu z gier prostych, umożliwiają zwiększenie efektywności treningu na trudniejszych odpowiednikach. Z drugiej strony, podejrzewaliśmy iż brak generalizacji wiedzy nauczyciela może utrudniać jego sprawną adaptację do nowego środowiska.

Drugim testowanym przez nas sposobem jest **transfer uogólniony**, polegający na wytrenowaniu jednego nauczyciela na zestawie *wszystkich* gier prostych a następnie użyciu go do nadzoru treningu na grach trudnych. Naszą wstępna hipotezą było to, iż nauczyciel uogólniony może być w stanie generalizować zdobytą wiedzę, co z kolei pozwoli na porównanie go do nauczycieli wyspecjalizowanych.

Schematy obu podejść zilustrowano w Rys. 6.1.

Algorytm nauczania DRRN (zob. Alg. 4) został dostosowany do treningu uogólnionego nauczyciela poprzez zmodyfikowanie strategii korzystania z priorytetowego bufora powtórek. W pierwotnej wersji algorytmu cała część priorytetowa bufora jest czyszczona wraz z każdym nowym najlepszym wynikiem (linia 19 Alg. 4). Uznaliśmy iż ta strategia jest niepoprawna dla treningu agenta uogólnionego na zestawie gier. Prowadzi bowiem do usuwania z bufora kluczowych doświadczeń dotyczących już rozegranych gier, jeśli w grze



Rysunek 6.1: Schematy eksperymentów: transfer wyspecjalizowanych nauczycieli (a) oraz transfer jednego uogólnionego nauczyciela (b)

bieżącej zostanie uzyskany wyższy wynik. Przyjęto więc iż w tej wersji usuwane są z bufora tylko zdominowane przejścia z gry bieżącej.

Transfer wyspecjalizowany nie wymaga zmian w algorytmie nauczania.

6.2.2 Q-kickstarting i transfer wag

Przejdźmy do omówienia sposobów w jaki wytrenowani nauczyciele nadzorują uczniów.

Q-kickstarting [42] to modyfikacja metody Kickstartingu opisanej w sekcji 4.4. Kickstarting upodabnia politykę nauczyciela do ucznia poprzez wykorzystanie zmodyfikowanej funkcji błędu, gdzie do standardowego błędu modelu dodawana jest entropia krzyżowa pomiędzy rozkładami prawdopodobieństwa zwracanymi przez politykę ucznia i nauczyciela. W naszym algorytmie tak bezpośrednie podejście nie jest efektywne, gdyż estymacje polityk są znane tylko pośrednio, poprzez wykorzystanie otrzymanych estymacji użyteczności.

Z tego też powodu w ramach badań zaadaptowaliśmy Kickstarting do algorytmów nauczania opartych na aproksymacji funkcji użyteczności. W tym celu wykorzystano **Q-kickstarting**: metodę bazującą na technikach wykorzystywanych do destylacji polityki. **Destylacja polityki** [40] jest metodą transferu wiedzy, w założeniach bardzo podobną do Kickstartingu. Zasadniczą różnicą jest moment transferu: w destylacji polityki odbywa się *przed treningiem* ucznia a nie w jego trakcie. Ścisłe, tworzony jest zbiór danych złożonych z par (s_t, Q_t) , gdzie s_t to stan środowiska w kroku t , a Q_t to wektor aproksymacji

ci Q-wartości wszystkich akcji akceptowanych w stanie s_t uzyskanych przy wykorzystaniu nauczyciela. Dla tego zbioru uczeń rozwiązuje zadanie regresji, wykorzystując wybrane metody nauczania z nadzorem. Dopiero po ukończeniu tego wstępniego zadania, ustalającego jego wstępную politykę, uczeń przechodzi do faktycznego rozwiązywania problemu nauczania przez wzmacnianie.

W przeprowadzonych eksperymentach autorzy [40] przetestowali trzy funkcje błędu dla zadania regresji. Zdecydowanie najlepszą okazała się specjalnie dostosowana wersja **dywergencji Kullbacka-Leiblera**, opisana równaniem 6.1.

$$\text{KLDivLoss}(p, y) = \text{Softmax}(\tau \cdot y) \ln \frac{\text{Softmax}(\tau \cdot y)}{\text{Softmax}(p)}, \quad (6.1)$$

gdzie p to predykcje (Q-wartości zwrócone przez ucznia), y to wartości docelowe (Q-wartości zwrócone przez nauczyciela), a τ to temperatura, spełniająca analogiczną funkcję jak w eksploracji Boltzmanna (zob. sekcja 2.5.2).

Q-kickstarting wykorzystuje dywergencję Kullbacka-Leiblera do zmodyfikowania funkcji błędu używanej przez ucznia. Zmodyfikowana funkcja opisana jest równaniem 6.2, gdzie p_S i p_T to odpowiednio predykcje ucznia i nauczyciela (zwrócone Q-wartości), y to wartości docelowe, a Loss to pierwotna funkcja błędu. Parametr λ_k pełni taką samą rolę jak w oryginalnej wersji Kickstartingu, również będąc z czasem zmniejszany, tak aby w końcu całkowicie uniezależnić ucznia od nauczyciela.

$$\text{QKickstartLoss}(p_S, p_T, y) = \text{Loss}(p_S, y) + \lambda_k \text{KLDivLoss}(p_S, p_T) \quad (6.2)$$

Q-kickstarting zmienia sposób obliczania błędu w algorytmie nauczania DRRN (zmodyfikowana została linia 15 Alg. 4). Ścisłe, funkcja błędu HuberLoss zastąpiona jest funkcją QKickstartLoss przyjmującą dodatkowy argument p_T . Wartość tego argumentu to $Q(o_k, c_k; \Theta_T)$, gdzie Θ_T to parametry nauczyciela.

Klasyczny transfer wag jest drugim przyjętym przez nas sposobem transferu wiedzy. Podejście to opiera się na wykorzystaniu parametrów wytrenowanego nauczyciela Θ_T do inicjalizacji parametrów ucznia Θ . Adaptacja Alg. 4 do tej metody jest bardzo prosta: zmieniana jest wyłącznie faza inicjalizacji parametrów DRRN (linia 2).

6.3 Przygotowanie do prób transferu wiedzy

Aby ułatwić odtworzenie i weryfikację przeprowadzonych prób transferu, zamieszczamy pełny rozpis użytych parametrów. Omawiamy również pragmatyczne powody stojące za wyborem ich wartości.

Tabela 6.1 prezentuje wartości parametrów współdzielonych przez wszystkie eksperymenty. Stała ucząca α , współczynnik dyskontowy γ , pojemności i wielkości części priorytetowej bufora oraz wymiar zanurzeń i warstw ukrytych zostały bezpośrednio zaczerpnięte z wersji DRRN towarzyszącej publikacji o środowisku Jericho [21]. Wielkość próbki została zmniejszona z 32 do 16. Głównym powodem tej zmiany były ograniczone zasoby obliczeniowe. Mniejsza liczba przeliczanych przejść znaczco przyspieszyła trening, bez zauważalnego

Parametr	Symbol	Wartość
początkowa wartość ε	ε_{start}	0.6
końcowa wartość ε	ε_{stop}	0.05
stała ucząca	α	0.0001
współczynnik dyskontowy	γ	0.9
limit epizodów	E	300/400*
limit kroków	T	100
pojemność bufora powtórek	N	10000
część priorytetowa bufora	η	0.5
wielkość próbki bufora	-	16
wymiar wektorów zanurzeń	-	128
wymiar warstw ukrytych	-	128
częstotliwość aktualizacji parametrów Θ'	U	10^{**}

* w zależności od trudności gry

** liczona w epizodach

Tablica 6.1: Zestaw parametrów wspólny dla wszystkich eksperymentów

wpływ na jakość wyników. Wielkość ε_{start} , ε_{stop} oraz U zostały wybrane na podstawie wstępnych prób eksperymentalnych. Kontynuacja badań przedstawionych w tej pracy może objąć optymalizację tych parametrów.

Liczba rozgrywanych epizodów E zależy od ilości punktów do zdobycia, tzn. poziomu trudności gry. Przyjeliśmy, że na jeden punkt przypada 50 epizodów, stąd liczba epizodów E dla gier prostych (6 punktów) wynosi 300, zaś w grach trudnych 400 (8 punktów). Z analizy gier wynika iż optymalna liczba kroków wymaganych do wygranej wahę się pomiędzy 10 a 20, przyjeliśmy więc, że agent ma maksymalnie 100 kroków (około 5 razy więcej) na jedną rozgrywkę.

Tabela 6.2 prezentuje parametry wyłączne dla metody Q-kickstarting. Początkowa wartość współczynnika nadzory λ_{start} jest liniowo zmniejszana aż do osiągnięcia wartości λ_{stop} w epizodzie wyznaczonym przez czas nadzoru. Czas nadzoru określa w jakiej części treningu agent poddany jest nadzorowi. Przykładowo dla użytnej wartości 0.5 jest to połowa treningu ($E \cdot 0.5$).

Parametr	Symbol	Wartość
początkowy współczynnik nadzoru	λ_{start}	0.5
końcowa współczynnik nadzoru	λ_{stop}	0.0
czas nadzoru	-	0.5*
temperatura	τ	0.1

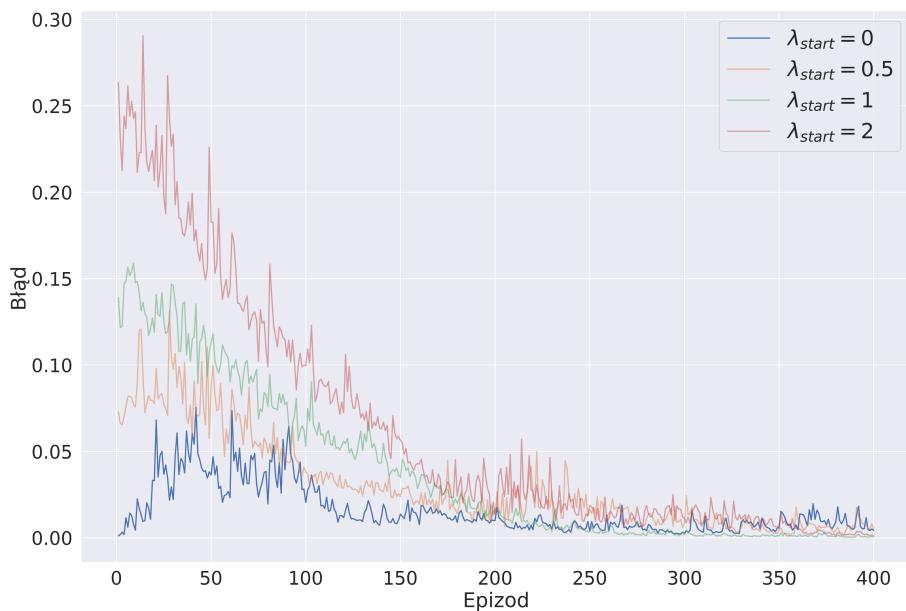
* λ_{stop} osiągana jest w epizodzie $E \cdot 0.5$ (połowie treningu)

Tablica 6.2: Zestaw parametrów wyłącznych dla Q-kickstartingu

W pracy [42] użyto wartości $\lambda_{start} \in \{1, 2\}$. W celu wyboru wartości tego parametru

przeprowadziliśmy wstępny eksperyment, trenując agenta na grze trudniejszej (gra nr. 2) i wykorzystując nadzór nauczyciela wyspecjalizowanego na łatwiejszym odpowiedniku tej gry. Rys. 6.2 pokazuje porównanie wykresów uśrednionego błędu w trakcie treningu ($\lambda_{start} = 0$ oznacza brak nadzoru). W trakcie eksperymentu zaobserwowaliśmy iż dla wartości $\lambda_{start} \in \{1, 2\}$ błąd dywergencji Kullbacka-Leiblera jest znaczaco większy niż błąd Hubera, co wydawało się negatywnie wpływać na jakość wyników. W szczególności, dla λ_{start} większego niż 0.5 najlepszy osiągnięty wynik to 6 punktów, choć dla pozostałych wartości agent uzyskiwał maksymalny możliwy wynik 8 punktów. Na podstawie tego wstępniego eksperymentu przyjęliśmy $\lambda_{start} = 0.5$ dla pozostałych prób transferu.

Należy zauważać (jak też poczynili autorzy [42]), że potrzeba ręcznego doboru odpowiedniego λ_{start} jest wadą tej metody. Jednym z zaproponowanych w artykule rozwiązań jest użycie metod ewolucyjnych aby na bieżąco dostosowywać wartość λ ; w [42] użyto w tym celu algorytmu PBT [25] (ang. *Population Based Training*).



Rysunek 6.2: Uśredniony błąd w trudnej grze dla różnych wartości λ_{start}

6.4 Wyniki prób transferu

Badania ekspermentalne rozpoczęto od treningu nauczycieli. Zgodnie z opisanyem wcześniej schematem wytrenowano pięciu nauczycieli wyspecjalizowanych (po jednym na prostą grę) i jednego uogólnionego. Wszyscy nauczyciele zdołali osiągnąć maksymalny wynik 6 punktów.

W kolejnym kroku realizowaliśmy wybrane metody transferu wiedzy nauczycieli używając Q-kickstartingu i transferu wag w grach trudnych.

Dodatkowo dla każdej gry wytrenowano agenta bazowego (ang. *baseline*), który nie korzystał z transferu wiedzy. Agent bazowy służy jako podstawa do oceny metod uczenia nadzorowanego i transferu wiedzy.

Do ewaluacji jakości wyników stosujemy miarę **średniej skumulowanej nagrody**, w przypadku rozważanym w niniejszej pracy będącą równoznaczną z ostatecznym punktowym wynikiem epizodu. Monitorowano również **względne odchylenie standardowe** nagrody (ang. Relative Standard Deviation, **RSD**), czyli stosunek odchylenia standardowego do średniej. W przeciwieństwie do zwykłego odchylenia standardowego, RSD jest miarą znormalizowaną, co umożliwia porównanie wyników pomiędzy różnymi grami i sposobami transferu. Finalnie, patrzmy też na liczbę gier dla których dana metoda zdołała znaleźć wygrywającą politykę.

6.4.1 Nauczyciele wyspecjalizowani

Pierwsza seria eksperymentów dotyczy nauczycieli wyspecjalizowanych. Tabela 6.3 prezentuje średnią skumulowaną nagrodę oraz względne odchylenie standardowe dla pojedynczych gier i sposobów nadzoru. Ścisłej, i -tej wiersz danych w tabeli przedstawia wyniki uzyskane w i -tej grze trudnej przez agenta nadzorowanego przez nauczyciela wyspecjalizowanego na i -tej prostej.

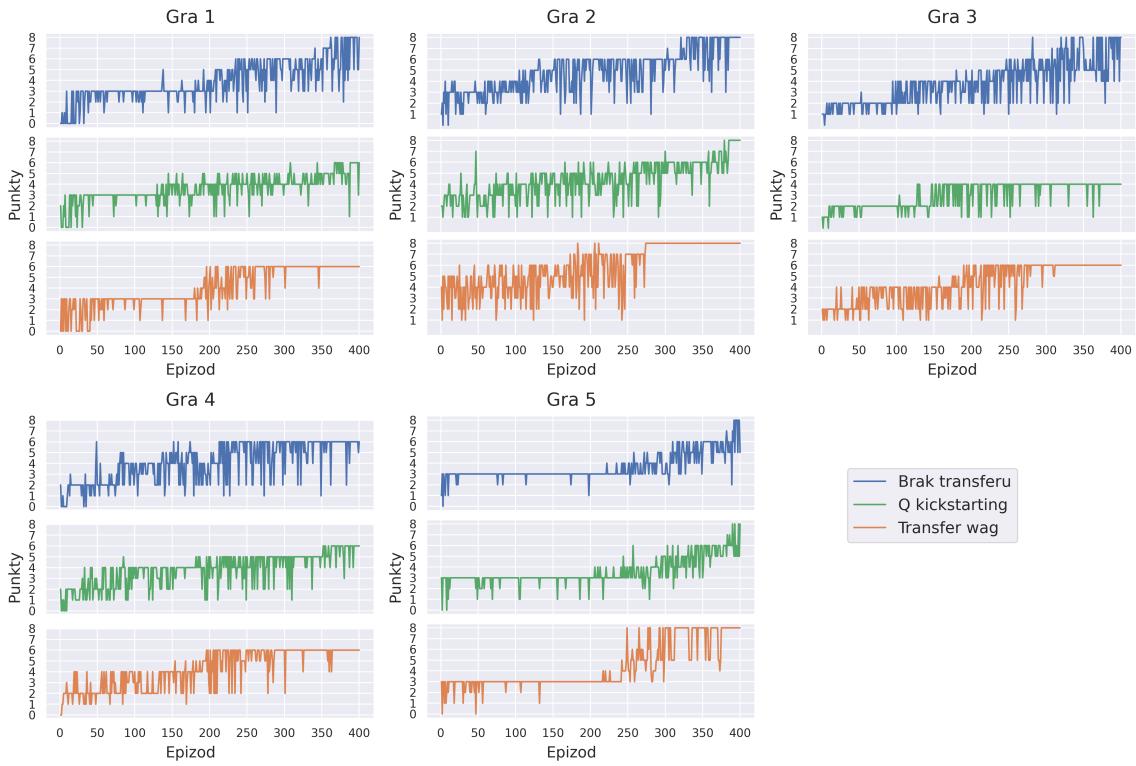
	Średnia skumulowana nagroda		
	Brak transferu	Q-kickstarting	Transfer wag
Gra 1	3.88 (0.48)	3.58 (0.34)	4.08 (0.43)
Gra 2	4.84 (0.40)	4.35 (0.38)	5.78 (0.36)
Gra 3	3.81 (0.52)	3.00 (0.38)	4.32 (0.39)
Gra 4	4.04 (0.44)	3.78 (0.39)	4.22 (0.41)
Gra 5	3.62 (0.34)	3.56 (0.35)	4.36 (0.49)
Sumarycznie*	4.04 (0.45)	3.65 (0.39)	4.56 (0.44)

* statystyki obliczone na próbce $400 \cdot 5 = 2000$ rozgrywek

Tablica 6.3: Nauczyciel wyspecjalizowany: średnia skumulowana nagroda i względne odchylenie standardowe (w nawiasach)

Dyskusja wyników. Pod względem średniej skumulowanej nagrody transfer wag wypada zdecydowanie najlepiej, osiągając najwyższy wynik we wszystkich grach. Agenci nadzorowani metodą Q-kickstartingu osiągnęli wyniki słabsze niż wariant bazowy. Q-kickstarting jest też najmniej wahliwą metodą ($RSD = 0.39$), co w połączeniu z niskimi wynikami może sugerować problem z eksploracją.

Możemy więc postawić hipotezę, iż agenci nadzorowani Q-kickstartingiem znajdują sub-optimalną politykę a następnie eksploatują ją, zamiast szukać akcji polepszających dotychczasowy wynik. W rezultacie skumulowana nagroda oscyluje relatywnie blisko średniej. Hipotezę wydaje się potwierdzać Rys. 6.3, gdzie zaprezentowano wykresy liniowe skumulowanej nagrody w poszczególnych epizodach treningów. Szczególnie w przypadku gry trzeciej, widzimy, że agent używający Q-kickstartingu natrafia na barierę 4 punktów, której nie jest w stanie przebić przez resztę treningu. Warto tu zauważyć, że cztery punkty odpowiadają wykonaniu części przepisu transferowanego z gry prostej, tzn. zebraniu i przyrządzeniu dwóch składników. Fakt ten, może sugerować, że wyspecjalizowany nauczyciel ma trudności z adaptacją do nowego środowiska, tak jak przypuszczaliśmy w



Rysunek 6.3: Nauczyciel wyspecjalizowany: wykresy skumulowanej nagrody.

sekcji 6.2.1.

Patrząc bardziej ogólnie, widzimy, że zarówno Q-kickstarting jak i transfer wag mają problem z osiągnięciem wyników wyższych niż 6 punktów. Udaje się to tylko w grze drugiej i piątej. Przypuszczamy iż główną barierą uniemożliwiającą osiągnięcie wyższych wyników jest to iż agent bazujący na nadzorze nauczyciela wyspecjalizowanego, próbuje przygotować posiłek po przyrządzeniu pierwszych dwóch składników, zgodnie z przepisem gry prostej ale błędnie w grze trudnej. Potrzeba “oduczenia” się tego zachowania może być istotną przeszkodą w treningu umożliwiającym agentom ukończenie gry. Za tą hipotezą przemawiają również wyniki wariantu bazowego, gdzie agent trenowany od zera osiąga maksymalny wynik we wszystkich grach, poza czwartą.

Wariant bazowy wypada również najlepiej pod względem sumarycznej liczby wygranych gier (pomimo iż nie osiąga najlepszego wyniku pod względem skumulowanej nagrody). Na drugim miejscu znajdują się ex-aequo Q-kickstarting i transfer wag, (po dwie wygrane gry).

6.4.2 Nauczyciel uogólniony

Druga seria eksperymentów sprawdza efektywność transferu wiedzy przy pomocy uogólnionego nauczyciela. Wstępne wyniki okazały się o wiele bardziej obiecujące niż w przypadku transferu wiedzy dla nauczycieli wyspecjalizowanych, stąd zdecydowaliśmy się na trzykrotne powtórzenie treningu każdego agenta. Miało to na celu lepsze uwierzytelnienie uzyskanych wyników.

	Średnia skumulowana nagroda		
	Brak transferu	Q kickstarting	Transfer wag
Gra 1	3.73 (0.44)	3.86 (0.43)	5.40 (0.45)
Gra 2	4.71 (0.36)	4.61 (0.40)	5.61 (0.37)
Gra 3	3.60 (0.49)	3.37 (0.44)	3.79 (0.42)
Gra 4	3.90 (0.42)	4.04 (0.48)	4.58 (0.41)
Gra 5	3.46 (0.33)	3.48 (0.30)	5.14 (0.43)
Sumarycznie*	3.88 (0.43)	3.87 (0.43)	4.91 (0.44)

* statystyki obliczone na próbce $400 \cdot 5 \cdot 3 = 6000$ rozgrywek

Tablica 6.4: Nauczyciel uogólniony: średnia skumulowana nagroda i względne odchylenie standardowe (w nawiasach) z trzech powtórzeń eksperymentu

Tabela 6.4 przedstawia średnią skumulowaną nagrodę oraz względne odchylenie standarde dla pojedynczych gier i sposobów nadzoru, przy czym miary te zostały uśrednione dla trzech powtórzeń. Dane znajdujące się w i -tym wierszu danych tabeli przedstawiają wyniki uzyskane w i -tej grze trudnej przez agenta nadzorowanego przez nauczyciela uogólnionego.

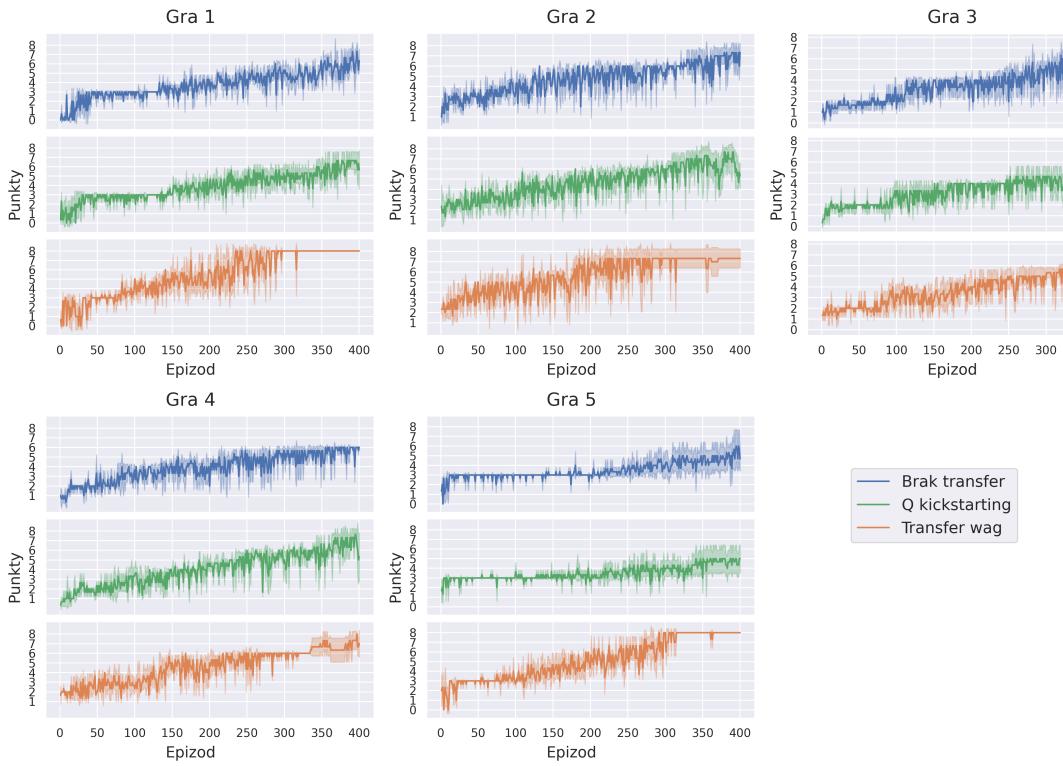
Dyskusja wyników. Zauważmy iż w tej serii eksperymentów wariant bazowy wypada gorzej niż w przypadku poprzedniej serii, otrzymując średnio 3.88 punktu.

Kolejną obserwacją jest to iż w tym przypadku ponownie najlepiej wypada transfer wag, uzyskując średnio ponad jeden punkt więcej niż pozostałe metody. Ponadto, Q-kickstarting korzystający z uogólnionego nauczyciela okazuje się bardziej skuteczny niż wariant wyspecjalizowany, osiągając średnią skumulowaną nagrodę porównywalną z nienadzorowanym agentem. Rozrzut wyników jest zbliżony dla wszystkich metod.

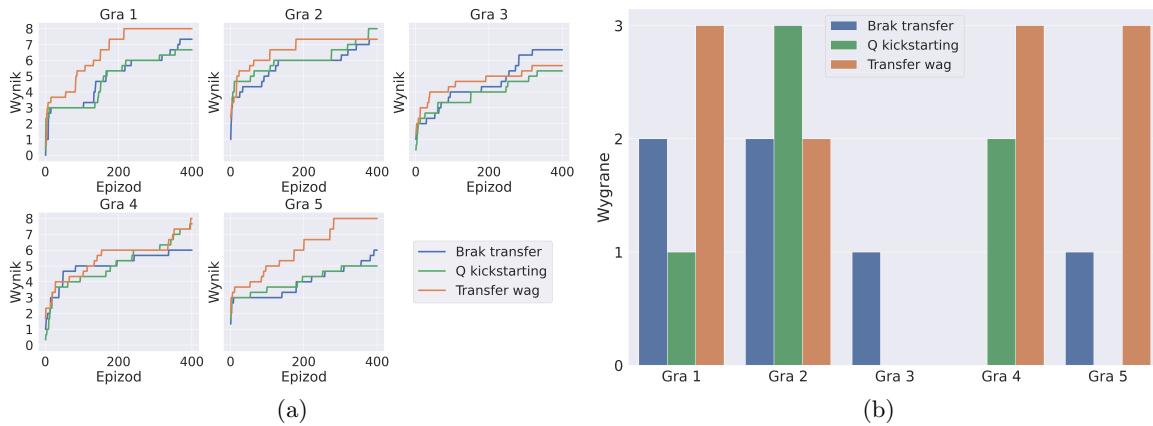
Rys. 6.4 przedstawia wykresy liniowe skumulowanej nagrody w poszczególnych epizodach, uśrednione dla trzech prób. W przeciwieństwie do poprzedniego eksperymentu, transfer wag regularnie przebija próg 6 punktów. Wspiera to naszą wcześniejszą hipotezę iż bariera metody opartej na nauczycielach wyspecjalizowanych jest brak generalizacji transferowanej wiedzy.

Dynamika osiągania nowych najlepszych wyników jest zilustrowane na Rys. 6.5, gdzie można zobaczyć, jak zmienia się najlepszy wynik na przestrzeni epizodów treningu. Dla wszystkich poza trzecią grą transfer wag jest w stanie (przynajmniej jednokrotnie) znaleźć i przyrządzić wszystkie wymagane składniki przed dwusetnym epizodem. Oznacza to, że druga połowa treningu może zostać spędziona w całości na nauce finalnego etapu przygotowania i skonsumowania posiłku. Tempo polepszania wyników Q-kickstartingu jest podobne do tempa wariantu bazowego. Wyjątkiem jest gra trzecia, gdzie oba sposoby transferu wiedzy wypadają gorzej niż wariant bazowy.

Wykres słupkowy z Rys. 6.5 (b) dodatkowo obrazuje zagadkową trudność gry trzeciej. Widzimy, że dla wszystkich trzech prób i każdego sposobu nadzoru, tylko wariantowi bazowemu udaje się wygrać (i to zaledwie raz). Bliższa analiza gry trzeciej nie odsłoniła



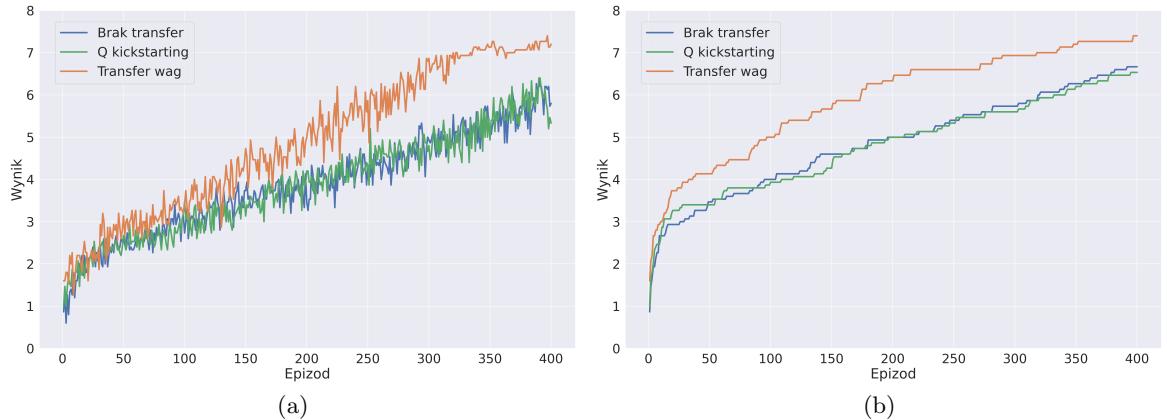
Rysunek 6.4: Nauczyciel uogólniony: wykresy średniej skumulowanej nagrody, wraz z oznaczonym odchyleniem standardowym z trzech powtórzeń eksperymentu.



Rysunek 6.5: Nauczyciel uogólniony: wykresy średniego najlepszego wyniku z trzech powtórzeń eksperymentu (a) oraz wykres słupkowy wygranych (b).

dla którego akurat ta gra stanowi wyzwanie dla metod użytych w tej pracy. Ścieżka, jaką gracz musi obrać aby zebrać składniki nie wyróżnia się na tle innych gier. Co więcej, liczba obiektów wymagających otwarcia jest mniejsza niż w grze piątej, gdzie gracz napotyka dwa rodzaje zamkniętych drzwi i lodówkę. Do tego przepis w tej grze jest prostszy niż w wielu innych: wszystkie trzy składniki wymagały usmażenia. Możliwe, że wyniki osiągnięte dla gry trzeciej są związane z rodzajem wiedzy wyniesionej z gier prostszych, weryfikacja tej

hipotezy wymaga jednak dodatkowej, pogłębionej serii eksperymentów powtarzających trening nauczyciela uogólnionego.



Rysunek 6.6: Nauczyciel uogólniony: wykres średniej skumulowanej nagrody z wszystkich pięciu gier i trzech powtórzeń eksperymentu (a) oraz wykres średniej najlepszej nagrody z wszystkich pięciu gier i trzech powtórzeń eksperymentu (b).

W omawianej serii eksperymentów transfer wag jest liderem pod względem sumarycznej liczby wygranych gier. Dodatkowo, wyniki otrzymane tą metodą są powtarzalne: dla trzech z pięciu gier wygrywająca polityka jest znajdowana w każdym przebiegu eksperymentu. Podobny rezultat osiągnięto jedynie dla Q-kickstartingu w grze drugiej, którą agent zdołał wygrać trzykrotnie; jednak dla pozostałych gier wyniki Q-kickstartingu są porównywalne lub gorsze niż brak transferu wiedzy. W szczególności należy zwrócić uwagę na brak jakichkolwiek wygranych dla gry piątej przy pomocy Q-kickstartingu (transfer wag wygrał ją trzykrotnie).

Rys. 6.6 (a) prezentuje wykres skumulowanej nagrody w poszczególnych epizodach, uśredniony dla pięciu gier i trzech powtórzeń eksperymentu. Rys. 6.6 (b) pokazuje wykres uśrednionego najlepszego wyniku w trakcie treningu. Oba wykresy ponownie wskazują na lepszą efektywności transferu wag w porównaniu do innych metod.

6.4.3 Środowisko eksperymentalne

Przeprowadzone eksperymenty wykonane zostały przy pomocy instancji obliczeniowej Google Cloud, wyposażonej w system operacyjny Ubuntu 20.04 LTS, kartę graficzną Nvidia Tesla T4, 4GB pamięci RAM oraz 10 GB pamięci dyskowej.

Rozdział 7

Konkluzje

W niniejszym rozdziale przedstawiamy wnioski i przemyślenia dotyczące badań zrealizowanych w pracy, podsumowując napotkane trudności oraz wskazując dalsze potencjalne kierunki badań związanych z transferem wiedzy w nauczaniu gier tekstowych.

7.1 Napotkane trudności i ograniczenia

Gry wygenerowane na potrzeby eksperymentów **nie są szczególnie złożone**, w porównaniu do pełnoprawnych tytułów takich jak Zork. Pomimo tego, trening agenta zdolnego regularnie wygrywać okazuje się większym wyzwaniem niż przypuszczaliśmy przed rozpoczęciem właściwej pracy.

Szczególnie dużą trudność sprawia **obszerna przestrzeń akcji**. Gry tekstowe są pod tym względem unikalne, gdyż (w odróżnieniu od środowisk takich jak gry arcade) akcje agenta określane są tu przy pomocy języka naturalnego. Stąd też liczba możliwych opisów akcji (komend) jest teoretycznie nieskończona - zakładając brak ograniczenia długości wpisywanej treści. Liczba akcji (czyli realizacji komend) jest również niespotykanie duża. Przykładowo, nawet w prostszych spośród wygenerowanych gier tekstowych agent może napotkać stany w których umożliwione jest ponad czterdzięciu akcji. Stąd też, nawet w tych grach gdzie silnik gry przekazuje zbiór akceptowanych w danym stanie komend (po jednej na akcję), agent wciąż postawiony jest przed ogromną liczbą wyborów. W rezultacie, większa część treningu poświęcana jest zwykle eksploracji środowiska; **agenci nie dokonują długotrwałej eksploatacji znalezionych rozwiązań**, co ogranicza możliwość doprecyzowania aproksymowanej funkcji użyteczności.

Powyższy problem potęgowany jest **rzadkim sygnałem nagrody**, co wydaje się być charakterystyczne dla gier tekstowych. W większości stanów nie występuje żaden sygnał nagrody. Stąd też we wczesnych fazach treningu agent zwykle nie zdobywa żadnych istotnych doświadczeń (nagradzanych przejść) z których może skorzystać w nauce. Skutkuje to **w pełni losowym przeszukiwaniem środowiska we wczesnych stanach**.

Połączenie rzadkiego sygnału nagrody i ogromnego zbioru możliwych akcji przekłada się na sporą wahliwość wyników. Częstą obserwacją w naszych eksperymentach jest to, że agenci potrafią w danym epizodzie osiągnąć maksymalne 8 punktów, w następnych uzyskując zaledwie 3 lub 2. Przypuszczamy, że brak czasu na dokładną aproksymację funkcji

użyteczności mógł sprawić, że w naszych eksperymentach zboczenie ze znanej ścieżki “gubiło” agenta. Inaczej mówiąc, użyteczność jest poprawnie estymowana dla niewielkiego wycinka przestrzeni stanów i akcji.

Pomimo obliczeń wykorzystujących GPU, nauka pojedynczej trudnej gry (bez transferu wiedzy) trwała około 30 minut. Agent poświęca 400 epizodów, wykonując w każdym maksymalnie 100 kroków, tylko po to aby nauczyć się sekwencji złożonej z kilkunastu komend. Obrazuje to, jak **kluczowym jest transfer wiedzy w kontekście omawianego zagadnienia**. Dla postępu badań konieczne jest znalezienie sposobów na efektywniejszy trening agentów.

7.2 Wnioski i obserwacje

Spośród badanych metod **tylko transfer wag korzystający z uogólnionego nauczyciela osiągnął zadowalające wyniki**. Metoda ta umożliwiła nadzorowanym agentom zarówno częstsze (w porównaniu do pozostałych rozważanych podejść) zwycięstwa, jak i szybsze tempo ich osiągania.

Nasze obserwacje sugerują, że **długość treningu determinowana jest w dużym stopniu momentem znalezienia pierwszego wygrywającego przebiegu**. Wraz z pierwszą wygraną, agent przechodzi do eksploatacji znalezionej rozwiązania (zachowanego w priorytetowej części bufora powtórek), zaś jego trening ogranicza się do powtórnego wykonywania optymalnego ciągu komend.

Powyzsze zjawisko dobrze ilustrują obserwacje w trakcie nauki gry pierwszej, gdzie transfer wag korzystający z uogólnionego nauczyciela osiąga maksymalną liczbę punktów już w okolicach połowy treningu (zob. Rys. 6.4). Skutkiem tego większość epizodów drugiej połowy treningu składa się z zaledwie 17 kroków, utrzymując optymalne rozwiązanie. Agenci korzystający z innych niż transfer wag metod spędzali najczęściej pełne 100 kroków starając się ukończyć grę. Trening korzystający z transferu wag uogólnionego nauczyciela trwał około 30% krócej, zarówno pod względem liczby kroków jak i czasu.

Q-kickstarting okazał się nieskuteczny, zarówno dla nauczycieli wyspecjalizowanych jak i uogólnionych. W przypadku tych pierwszych, nadzór miał negatywny efekt: agenci osiągali gorsze wyniki niż wariant bazowy. Określenie przyczyny tego zjawiska nie wydaje się oczywiste. Możliwe, że wybrany przez nas zestaw parametrów nie jest optymalnym i zastosowanie metod ewolucyjnych (zaproponowanych w pracy [42]) mogłoby zwiększyć efektywność tej metody. Równocześnie, nie można wykluczyć, że samo środowisko gier tekstowych nie sprzyja transferowi wiedzy poprzez Q-kickstarting. W [42] kickstarting przetestowany był na zbiorze gier DMLab-30, które charakteryzowały się gęstszym sygnałem nagrody oraz bardziej ograniczoną przestrzenią akcji (całość przestrzeni składała się z 8 akcji) niż rozważane w niniejszej pracy. Być może, w środowiskach tego rodzaju Q-kickstarting sprawdziłby się lepiej.

Patrząc na nasze wyniki eksperymentalne możemy stwierdzić **przewagę nauczyciela uogólnionego nad wyspecjalizowanymi**. Zarówno dla transferu wag jak i Q-kickstartingu nauczyciel wytrenowany na wszystkich 5 grach prostych umożliwia bardziej efektywną naukę agentów w grach złożonych. Zgodnie z naszymi przypuszczeniami, **nauczyciel wyspecjalizowany ma znaczące trudności w adaptacji do nowych środowisk**. Szczególnym wyzwaniem dla nauczyciela, wydaje się być dostosowanie się do nowego momentu przygotowania po-

silku (po przyrządzeniu trzech, a nie dwóch składników). Agenci uczeni przez nauczyciela wyspecjalizowanego zwykle nie byli w stanie (niezależnie od metody nadzoru) posunąć się dalej w rozgrywce po przyrządzeniu wszystkich składników i osiągnięciu 6 punktów. Miało to miejsce w trzech grach (zob. Rys. 6.3). Uogólniony nauczyciel był pod tym względem dużo bardziej efektywny, “utykając” w podobny sposób tylko w jednej grze.

7.3 Dalsze badania

W ramach dalszych badań chcemy zbadać powody dla których Q-kickstarting okazał się nieskuteczny. W tym celu warto zbadać efektywność oryginalnej metody Kickstartingu w grach tekstowych o złożoności podobnej do wygenerowanych na potrzeby tej pracy. Kickstarting wymaga użycia agenta uczonego metodą Policy Gradient, co pozwoliłoby dodatkowo na porównanie tego rodzaju algorytmów do głębokiego Q-learningu. Dostęp do większych zasobów obliczeniowych, umożliwi też na przeprowadzenie szerzej zakrojonej optymalizacji parametrów (w szczególności λ_{start}), zarówno dla zwykłego Kickstartingu jak Q-kickstartingu, np. wykorzystując algorytm PBT [25].

Naturalnym rozszerzeniem przeprowadzonych eksperymentów, jest wygenerowanie większej liczby gier, o jeszcze bardziej zróżnicowanych parametrach i poziomach trudności. Trening na zwiększonym zbiorze gier może uwarygodnić lub podważyć osiągnięte w tej pracy rezultaty. W szczególności, interesującym byłoby zaobserwowanie potencjalnych zmian w efektywności nauczyciela uogólnionego.

Dodatkową możliwością jest rozbicie gier na podzadania, np. poprzez stworzenie jednego zestawu gier w których jedynym zadaniem agenta jest znalezienie składników i drugiego zestawu, gdzie zadaniem agenta jest wyłącznie przygotowanie posiłku. Schemat ten może objąć dalsze wytrenowanie pary nauczycieli uogólnionych (po jednym dla każdego podzadania) i użycie obu w nadzorze agentów uczących się trudniejszych gier z pierwotnego zbioru.

Interesującym może być też zbadanie wpływu książki z przepisami na naukę agentów. W zamyśle, agent powinien wykorzystywać treść przepisu aby móc dynamicznie dostosować się do wymogów zadania. W praktyce może być zupełnie inaczej: w skrajnym przypadku agent może zignorować przepis, ucząc się metodą prób i błędów. Aby sprawdzić czy i jakie informacje agent wynosi z przepisu, można wygenerować zbiór gier kontrolnych, pozbawionych książek z przepisem. Ponadto można rozważyć gry zawierające niepoprawne przepisy, tzn. nieprowadzące do zwycięstwa. Porównanie wyników osiągniętych na oryginalnym i kontrolnym zbiorze gier może pozwolić na lepsze poznanie sposobu rozumowania agentów.

Bibliografia

- [1] P. Ammanabrolu and M. Riedl. Playing text-adventure games with graph-based deep reinforcement learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3557–3565, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [2] P. Ammanabrolu and M. O. Riedl. Transfer in deep reinforcement learning using knowledge graphs.
- [3] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. D. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. *CoRR*, abs/1711.07280, 2017.
- [4] T. Anderson, M. Blank, B. Daniels, and D. Lebling. Zork, 1980.
- [5] G. Angeli, M. J. Johnson Premkumar, and C. D. Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, Beijing, China, July 2015. Association for Computational Linguistics.
- [6] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Leg Francq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen. Deepmind lab, 2016.
- [7] R. Bellman, R. Corporation, and K. M. R. Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.
- [8] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.
- [9] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272, 2018.
- [10] M.-A. Côté, A. Kádár, X. Yuan, B. Kybartas, T. Barnes, E. Fine, J. Moore, R. Y. Tao, M. Hausknecht, L. E. Asri, M. Adada, W. Tay, and A. Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018.

- [11] D. R. Cox. The regression analysis of binary sequences (with discussion). *J Roy Stat Soc B*, 20:215–242, 1958.
- [12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [13] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
- [14] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu.
- [15] M. Forbes, A. Holtzman, and Y. Choi. Do neural language representations learn physical commonsense? *CoRR*, abs/1908.02899, 2019.
- [16] N. Fulda, D. Ricks, B. Murdoch, and D. Wingate. What can you do with a rock? affordance extraction via word embeddings. *CoRR*, abs/1703.03429, 2017.
- [17] P. Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, feb 1994.
- [18] J. J. Gibson. The theory of affordances. In J. B. Robert E Shaw, editor, *Perceiving, acting, and knowing: toward an ecological psychology*, pages pp.67–82. Hillsdale, N.J. : Lawrence Erlbaum Associates, 1977.
- [19] J. Guan, Y. Wang, and M. Huang. Story ending generation with incremental encoding and commonsense knowledge. In *AAAI*, 2019.
- [20] X. Guo, M. Yu, Y. Gao, C. Gan, M. Campbell, and S. Chang. Interactive fiction game playing as multi-paragraph reading comprehension with reinforcement learning. *CoRR*, abs/2010.02386, 2020.
- [21] M. Hausknecht, P. Ammanabrolu, M.-A. Côté, and X. Yuan. Interactive fiction games: A colossal adventure. *CoRR*, abs/1909.05398, 2019.
- [22] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1621–1630, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.
- [23] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. M. Czarnecki, M. Jaderberg, D. Teplyashin, M. Wainwright, C. Apps, D. Hassabis, and P. Blunsom. Grounded language learning in a simulated 3d world. *CoRR*, abs/1706.06551, 2017.
- [24] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *CoRR*, abs/2011.02669, 2020.
- [25] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

- [26] D. Jurafsky and J. H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, Upper Saddle River, N.J., 2009.
- [27] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1–2):99–134, may 1998.
- [28] G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. volume 148, pages 489–496, 01 2006.
- [29] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020.
- [30] G. Michau and O. Fink. Unsupervised transfer learning for anomaly detection: Application to complementary operating condition transfer. *Knowledge-Based Systems*, 216:106816, 2021.
- [31] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
- [32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.
- [33] G. Nelson. Inform7: a programming language for creating interactive fiction. 2022.
- [34] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.
- [35] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.
- [36] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.
- [37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.
- [38] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.
- [39] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.
- [40] A. A. Rusu, S. G. Colmenarejo, Çaglar Gülcöhre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *CoRR*, abs/1511.06295, 2016.
- [41] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

- [42] S. Schmitt, J. J. Hudson, A. Zídek, S. Osindero, C. Doersch, W. M. Czarnecki, J. Z. Leibo, H. Küttler, A. Zisserman, K. Simonyan, and S. M. A. Eslami. Kickstarting deep reinforcement learning. *ArXiv*, abs/1803.03835, 2018.
- [43] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.
- [44] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte pair encoding: A text compression scheme that accelerates pattern matching, 1999.
- [45] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.
- [46] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.
- [47] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
- [48] Wikipedia. Huber loss — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Huber%20loss&oldid=1077659242>, 2022. [Online; accessed 15-September-2022].
- [49] Y. Wu, M. Mozifian, and F. Shkurti. Shaping rewards for reinforcement learning with imperfect demonstrations using generative models. *CoRR*, abs/2011.01298, 2020.
- [50] B. Xiao, Q. Lu, B. Ramasubramanian, A. Clark, L. Bushnell, and R. Poovendran. FRESH: interactive reward shaping in high-dimensional state spaces using human feedback. *CoRR*, abs/2001.06781, 2020.
- [51] S. Yao, R. Rao, M. Hausknecht, and K. Narasimhan. Keep CALM and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8736–8754, Online, Nov. 2020. Association for Computational Linguistics.
- [52] X. Yuan, M. Côté, J. Fu, Z. Lin, C. J. Pal, Y. Bengio, and A. Trischler. Interactive language learning by question answering. *CoRR*, abs/1908.10909, 2019.
- [53] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey, 2020.