Politechnika Poznańska
Wydział Informatyki i Zarządzania
Instytut Informatyki

Praca dyplomowa magisterska

# IMITATION LEARNING

Wojciech Kopeć, 101675

Promotor
dr inż. Krzysztof Dembczyński

Poznań, 2017 r.

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

# Spis treści

# Rozdział 1

# Wstęp teoretyczny

## 1.1 Uczenie przez demonstrację a uczenie nadzorowane

Najprostszym podejściem do uczenia przez demonstrację jest traktowanie go jak każdego innego problemu uczenia nadzorowanego, przy czym w przeciwieństwie do minimalizowania kosztu działania agenta minimalizowana jest różnica pomiędzy pomiędzy polityką wyuczonego agenta a polityką eksperta. Najprostsze podejście zakłada jednak, że dane uczące i testowe są niezależne i mają jednakowy rozkład, podczas gdy przy uczeniu przez demonstrację nauczona polityka ma bezpośredni wpływ na osiągane później stany, na podstawie których dana polityka będzie sprawdzana. Jak dowiedziono w [**?**] wynikający z tego błąd rośnie kwadratowo w stosunku do czasu trwania epizodów – gdy klasyfikator popełni błąd w odwzorowywaniu polityki eksperta najprawdopodobniej trafi do stanu nieodwiedzonego przez eksperta, co z dużym prawdopodobieństwem oznacza popełnianie następnych błędów, ponieważ uczeń nie miał jak nauczyć się „podnoszenia się" po błędach.

Jednym ze sposobów radzenia sobie z tym problemem jest wprowadzanie małych zmian podczas iteracji polityki, dzięki czemu rozkład stanów dla nowej polityki jest bliski staremu. Idea polega na zaczynaniu od polityki całkowicie identycznej z polityką eksperta i stopniowym przechodzeniu na politykę wyuczoną. Aby to osiągnąć można wymagać, aby podczas uczenia uczeń mógł w każdej chwili zapytać eksperta, jakie akcje ekspert podjąłby w danym stanie. Dany układ wymaga większej interakcji, ale może być zrealizowany dla wielu z praktycznych przykładów wykorzystania uczenia przez demonstrację.

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

W celu wyeliminowania tych ograniczeń [**?**] proponują Iterowany Probabilistyczny Mieszający algorytm. Opierając się na algorytmie iterowania polityki algorytm w każdym kroku stosuje nową stochastyczną politykę wybierając z zadanym prawdopodobieństwem pomiędzy wykonywaniem polityki wyuczonej w poprzednim kroku i konstruowanej w danej iteracji nowej polityki, przy czym

prawdopodobieństwo wyboru nowej polityki jest niewielkie. Algorytm zaczyna od dokładnego wykonywania akcji eksperta. W każdej kolejnej iteracji algorytmu prawdopodobieństwo odpytania eksperta jest coraz niższe i zbiega się do 0. Opisane rozwiązanie zostało z powodzeniem przetestowane na przykładzie grania w proste gry, gdzie danymi wejściowymi był obraz z ekranu. Autorzy zdecydowali się na klasyfikator wybierający konkretne akcje dla danego stanu, zamiast częściej używanego w uczeniu ze wzmocnieniem klasyfikatora odwzorowującego funkcję kosztu. Wadą tego podejścia jest brak odrzucania nieskutecznych polityk podczas iteracji, co może prowadzić do niestabilnych wyników.

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

W kolejnej publikacji [**?**] prezentują nowe podejście, nazwane Agregacją Zbioru Danych. W uproszczeniu, podejście to jest następujące: W pierwszej iteracji algorytm zbiera dane testowe stosując politykę pokazaną przez eksperta, po czym trenuje klasyfikator odwzorowujący zachowanie eksperta na danym zbiorze danych. W każdej kolejne iteracji algorytm stosuje politykę wygenerowaną w poprzedniej iteracji i dodaje dane uzyskane podczas jej stosowania do zbioru danych, po czym trenuje klasyfikator by odwzorowywał zachowanie eksperta na całym zbiorze danych. Podobnie jak w poprzednim algorytmie, żeby przyspieszyć uczenie na pierwszych etapach algorytmu, dodano opcjonalną możliwość odpytania eksperta o jego wybór akcji. Uzyskane z pomocą tej metody wyniki są wyraźnie lepsze od wyników uzyskanych za pomocą metody opisanej w poprzednim paragrafie.

## 1.2 Podążanie za ekspertem a przewyższanie eksperta

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

## 1.3 Eksploracja

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

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

wana nagroda będzie zwracana dla zachowań, które agent odkrył już jako zyskowne i dla zachowań jeszcze nieodkrytych (dla których funkcja aproksymacyjna nie jest jeszcze poprawiona). Ten prosty zabieg powoduje, że algorytmy uczenia ze wzmocnieniem naturalnie balansują pomiędzy eksploracją i intensyfikacją przeszukiwania bez dodatkowych modyfikacji. Od strony teoretycznej zaletą R-maxa jest duża ogólność zastosowania – algorytm wymaga spełnienia bardzo luźnych założeń, badany proces nie musi być nawet procesem decyzyjnym Markowa.

W [**?**] autorzy zaproponowali rozwiązanie, które pozwala ocenić, w jakim stopniu odwiedzony stan jest dla agenta nowością. Opiera się ono na stworzeniu aproksymatora, którego zadaniem jest przewidywanie, jaki stan osiągnie agent po wykonaniu danej akcji w danym stanie. Predykcja porównywana jest z faktycznie osiągniętym stanem, a wielkość błędu jest wyznacznikiem nowości stanu – im większy błąd predykcji, tym bardziej nieznany stan, za co przyznawana jest większa nagroda eksploracyjna. Jak większość opisywanych publikacji, w [**?**] rozwiązywano problem uczenia agenta grania w gry zręcznościowe na podstawie surowego obrazu z wykorzystaniem Q-learningu i głębokich sieci neuronowych. Pierwszą kwestią do rozwiązania przy implementacji pomysłu jest metryka pozwalająca określić podobieństwo stanów. Próby predykcji wartości konkretnych pikseli opisane przez autorów nie przyniosły efektów, generując tylko szum. Zamiast tego trenowano głęboką sieć neuronową do przewidywania następnego stanu i wykorzystano jedną z ukrytych warstw tej sieci o mniejszej liczbie jednostek jako enkoder stanu, który przenosi surowy obraz do przestrzeni o znacznie mniejszej liczbie parametrów. Za miarę podobieństwa między stanami przyjęto odległość kartezjańską parametrów uzyskanych z zakodowania dwóch stanów. Zakodowanymi stany używane były do wytrenowania właściwego, prostszego aproksymatora, na podstawię błędy którego określano nowość stanu. Dla każdego przejścia między stanami przyznawano bonusową nagrodę zależną od nowości. Potencjalnym problemem związanym z tym podejściem jest to, że Q-learning stara się nauczyć funkcji, która jest niestacjonarna Autorzy piszą, jednak, że w praktyce nie stanowiło to problemu.

Innym taktykę dywersyfikacji przeszukiwania przy wykorzystaniu głębokiej sieci neuronowej zaprezentowano w [**?**]. Podobnie jak w [**?**] uczono sieć funkcji Q, jednak zamiast pojedynczej funkcji Q trenowano jednocześnie K funkcji Q, przy czym każda trenowana była tylko na podzbiorze przykładów uzyskanym za pomocą techniki bootstrapingu. Każda funkcja Q reprezentowana była przez jedną K „głów" wspólnej wielopoziomowej sieci . Dla każdego z epizodów wybierana losowo była jedna głowa – funkcja Q i przez cały epizod agent kierował się polityką optymalną dla tej funkcji Q.

# Rozdział 2

# Niepewność

## 2.1 Niepewność - uproszczony eksperyment

W celu porównania skuteczności Bootstrapa i Dropoutu w ocenianiu niepewności wyników sieci przeprowadzono prostszy eksperyment na lepiej znanych i kontrolowanych danych. Do eksperymentu wykorzystano zbiór MNIST [REF] zawierający odręcznie pisane cyfry. MNIST jest często używany jako przykładowy zbiór danych, służący do przystępnej prezentacji i porównań metod uczenia maszynowego. Najczęściej wykorzystywany jest w kontekscie klasyfikacji, jednak traktowanie wartości liczbowych cyfr jako etykiet (w przeciwieństwie do stosowanego w klasyfikacji one-hot encoding [REF]) w oczywisty sposób odpowiada problemowi regresji.

### 2.1.1 Eksperyment

W ramach eksperymentów uczono sieć neuronową na niezbalansowanych zbiorach danych bazujących na MNIST. Dobrane dystrybucje przykładów o konkretnych etykietach w danych uczących mają pokazać, jak poszczególne techniki zachowują się wobec zupełnie nieznanych danych i mało znanych danych.

Zbiór danych MNIST składa się z 60 tysięcy przykładów uczących i 10 tysięcy przykładów treningowych. Każdy obrazek przedstawia jedną czarno-białą cyfrę i ma rozmiar 28x28 pikseli. W eksperymencie etykiety zostały przetrasformowane liniowo z przedziału $[0, 9]$ do $[-0.5, 0.5]$.

Pierwszy, prostszy, eksperyment posłużył również do znalezienia optymalnych parametrów obu metod. Drugi eksperyment został przeprowadzony z wykorzystaniem znalezionych wcześniej parametrów. Każdy z eksperymentów został powtórzony 10 razy.

#### Rozpoznawanie nieznanych danych

Zachowanie wobec nieznanych danych zbadano ucząc sięć neuronową wyłącznie na przykładach parzystych cyfr. W wynikowej klasyfikacji stopień niepewności zwracanych wyników powinien być znacznie wyższy dla cyfr nieparzystych niż parzystych.

#### Ocena stopnia pewności wyników

Zachowanie wobec mało znanych danych zbadano ucząc sieć neuronową na niezbalansowanym zbiorze danych. Przykłady trafiały do zbioru uczącego z prawdopodobieństwem proporcjonalnym do przedstawianej cyfry, przykładowo 0 z prawdopodobieństwem $p = 0$, 5 z prawdopodobieństwem $p = 0.5$ i 9 z prawdopodobieństwem $p = 0.9$. W wynikowej klasyfikacji stopień niepewności zwracanych wyników powinien być zależny od prawdopodobieństwa trafienia do zbioru danej etykiety.

### 2.1.2 Implementacja bazowa

Bazą dla implementacji użytych w eksperymencie był przykład klasyfikacji zbioru MNIST za pomocą konwolucyjnych sieci neuronowych udostępniony z biblioteką Tensorflow. Użyta w przykładzie sieć składa się z dwóch warstw konwolucyjnych i dwóch warstw w pełni połączonych.

W ramach obu eksperymentów uczenie trwa 1 milion iteracji, a wielkość batcha wynosi 128. Prędkość uczenia wynosi 0.005, przy czym dla Bootstrapu ta wartość jest normalizowana przez średnią liczbę użyć każdego przykładu.

### 2.1.3 Miary jakości

Kluczowe dla określenia jakości obu metod jest zdefiniowanie miary jakości wyników. Docelowo przyjęta miara powinna dobrze oddawać przydatność do oceny stanu przez agenta DQN.

Za miarę niepewności przyjęto rozstęp międzykwartylowy próbek uzyskanych z sieci. O wyborze rozstępu międzykwartylowego zdecydowała większa od odchylenia standardowego odporność na skrajne wartości. Oprócz rozstępu międzykwartylowego sprawdzono eksperymentalnie również wariancję (która dawała nieznacznie gorsze wyniki) i różnicę między skrajnymi wartościami (zależność od skrajnych wartości uczyniła tą miarę bardzo niestabilną i mało skuteczną).

$$unc = q(75) - q(25)$$

#### Rozpoznawanie nieznanych danych

Liczba znanych i nieznanych etykiet w zbiorze tekstowym jest równa, dlatego za miarę jakości rozdziału danych znanych i nieznanych przyjęto stosunek sumy średnich niepewności dla kolejnych nieznanych etykiet do sumy niepewności dla kolejnych znanych etykiet. Wartości bliskie 1 oznaczają brak rozdziału danych. W eskperymentach wynikowe miary cząstkowe nie przekraczały wartości 2.

$$quality_{ND} = \frac{\sum_{l \in \{unknown\}} \overline{unc_l}}{\sum_{l \in \{known\}} \overline{unc_l}}$$

#### Ocena stopnia pewności wyników

Niepewność powinna być odwrotnie proporcjonalna do trafności klasyfikacji, dlatego za miarę jakości oceny niepewności wyników przyjęto wartość absolutną współczynnika korelacji pomiędzy średnią niepewnością a średnią trafnością klasyfikacji dla kolejnych etykiet. Użyty w eksperymencie współczynnik regresji jest liczony dla etykiet od 1 do 9, ponieważ zerowa trafność dla etykiety 0 jest wspólna dla obu metod i zaburza wyraźnie liniową zależność dla reszty etykiet.

$$quality_{OP} = |r_{unc\ acc}|$$

### 2.1.4 Dropout - konfiguracja

Dropout został dodany pomiędzy ostatnią warstwą konwolucyjną sieci a pierwszą w pełni połączoną oraz pomiędzy obiema w pełni połączonymi warstwami. Parametry modelu to prawdopodobieństwo zachowania neuronu w czasie treningu $p_{train}$, prawdopodobieństwo zachowania neuronu w czasie testu $p_{test}$ i liczbę odpytań sieci $n$ przy określaniu niepewności. W eksperymentach sprawdzano wartości $p_{train} \in \{0.25, 0.5, 0.75, 1\}$, $p_{test} \in \{0.25, 0.5, 0.75\}$ i $n \in \{10, 30, 50, 100\}$.

Zastosowana implementacja z wykorzystaniem Tensorflow wymaga każdorazowego przeliczenia wszystkich wartości przy każdym pojedyńczym odpytaniu.

### 2.1.5 Bootstrap - konfiguracja

Bootstrapowana sieć ma wspólne warstwy konwolucyjne. Z warstw konwolucyjnych wychodzi $n$ niezależnych "głów", składających się z dwóch warstwy w pełni połączonych wykorzystanych w przykładzie bazowym. Parametry modelu to liczba "głów" $n$ i prawdopodobieństwo uwzględnienia krotki danych przez głowę $p_{incl}$. W eksperymentach sprawdzano wartości $n \in \{5, 7, 10\}$ i $p_{incl} \in \{0.25, 0.5, 0.75, 1\}$.

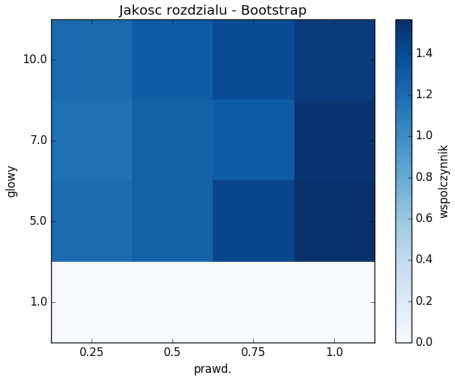
Zastosowana implementacja z wykorzystaniem Tensorflow decyduje o uwzględnianiu przez poszczególne głowy dla pełnych batchy danych, a nie dla pojedynczych krotek. Przy odpytywaniu kilku głów o ten sam przykład przeliczenie warstw konwolucyjnych następuje tylko jednokrotnie.

### 2.1.6 Niepewność - wyniki eksperymentu z nieznanymi danymi

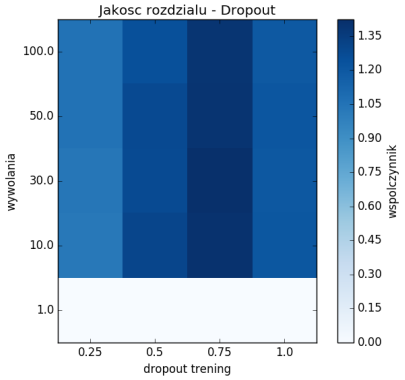
Na wykresach 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 przedstawiono średnie i odchylenia standardowe miary jakości $quality_{ND}$ uzyskane dla poszczególnych konfiguracji eksperymentów. Największa wartość $quality_{ND}$ uzyskana za pomocą Bootstrapa (1.565 dla 5 głów i prawdopodobieństwa 1) jest wyższa niż największa wartość uzyskana za pomocą Dropoutu (1.424 dla 30 wywołań i prawdopodobieństw dropoutu = 0.75). Wyniki bootstrapa dla tych parametrów cechują się trzykrotnie większą wariancją (0.148 a 0.049), ale mimo to sumarycznie wypadają korzystniej od Dropoutu.

Boostrap najlepiej wypada dla małej (5) liczby głów - jest to zaskakujące zachowanie, które może być artefaktem zbyt małej liczby powtórzeń eksperymentu. Podobne wyniki dla różnej liczby głów utrzymują jednak stałą przewagę nad Dropoutem. Zgodna z oczekiwaniami jest przewaga konfiguracji z prawdopodobieństwem uwzględnienia przez głowę próbki równym 1. Dzięki temu poszczególne głowy są lepiej dopasowane do znanych przykładów powiększając różnicę w stosunku do nieznanych danych.

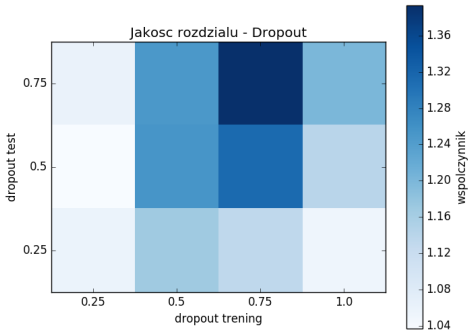
Dla Dropoutu zachodzi podobne zjawisko dla liczby wywołań jak dla głów w Bootstrapie - wbrew oczekiwaniom najlepsze wyniki osiągane są dla mniejszej liczby powtórzeń, przy zachowaniu niewielkich różnic między wartościami. Podobnie zgodnie z oczekiwaniami zachowuje się też drugi z parametrów - prawdopodobieństwo zachowania neuronu w czasie treningu. Wysokie prawdopodobieństwo pozwala "poznać" lepiej dane, a prawdopodobieństwo równe 1 uniemożliwia poprawne działanie dropoutu, ponieważ sieć nie jest przyzwyczajona do dropoutu pojawiającego się dopiero w teście.



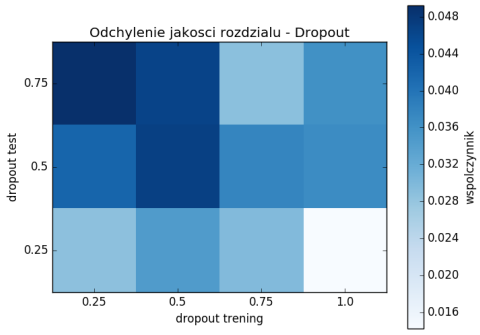

Rysunek 2.1: Bootstrap



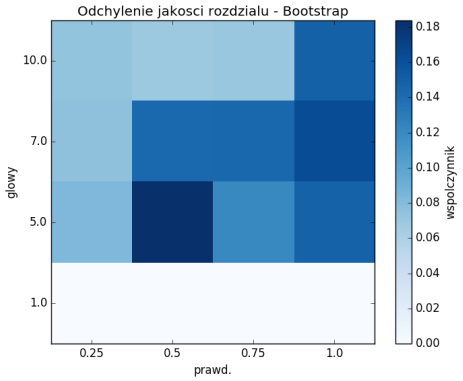

Rysunek 2.2: Dropout



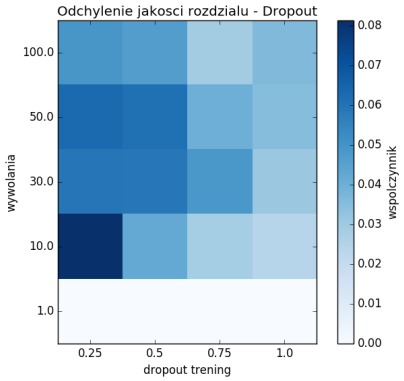

Rysunek 2.3: Dropout




Rysunek 2.4: Dropout

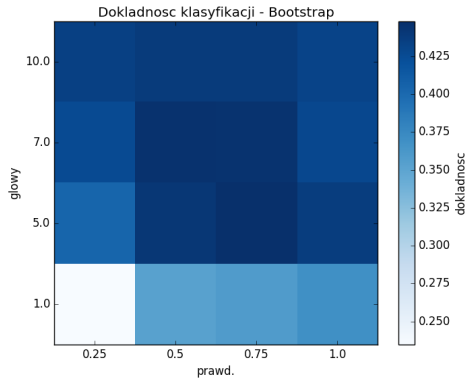



Rysunek 2.5: Bootstrap

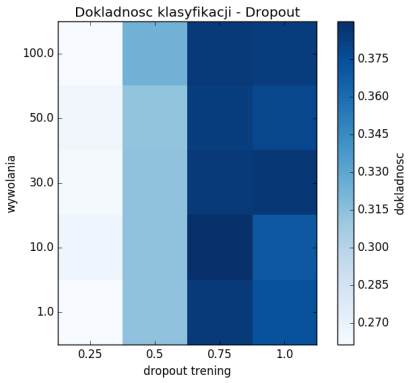



Rysunek 2.6: Dropout

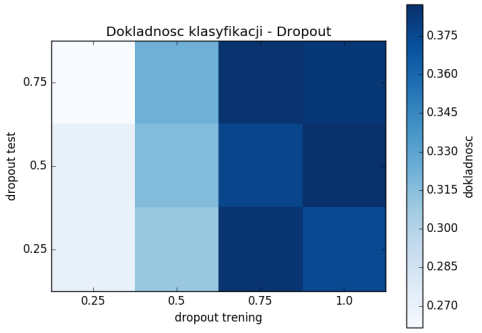
Na wykresach 2.7, 2.8, 2.9, 2.10, 2.12, 2.11 przedstawiono średnie i odchylenia standardowe dokładności klasyfikacji uzyskane dla poszczególnych konfiguracji eksperymentów. Wyniki przedstawiają się podobnie jak dla miary $quality_{ND}$. Lepsze wyniki osiąga Bootstrap (44.81% dla 5 głów i prawdopodobieństwa 0.75, 43.64% dla 5 głów i prawdopodobieństwa 1), a jego wyniki są podobne dla wszystkich sensownych parametrów. Wariancja jest minimalna. Wyniki Dropoutu oscylują dookoła 39% dla wszystkich sensownych parametrów, przy znacznie większej niż Boostrstrap wariancji (2%). Warto zauważyć, że dla obu metod wyniki są bardzo dodobne dla szerokich zestawów parametrów, i wyraźnie większe niż w przypadku braku bazowego rozwiązania (zaimplementowane w eksperymencie jako Bootstrap z jedną głową, osiągające 37% dokładności przy 5% wariancji. Gorszy wynik bazowej wersji wynika z braku odporności na przeuczenie - Bootstrap i Dropout działają jak regularyzatory).



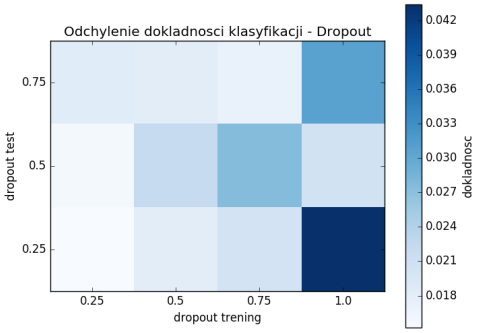

Rysunek 2.7: Bootstrap



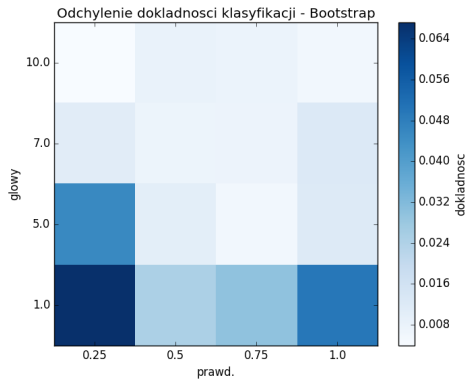

Rysunek 2.8: Dropout



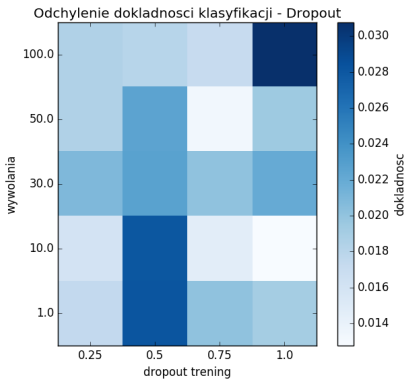

Rysunek 2.9: Dropout




Rysunek 2.10: Dropout

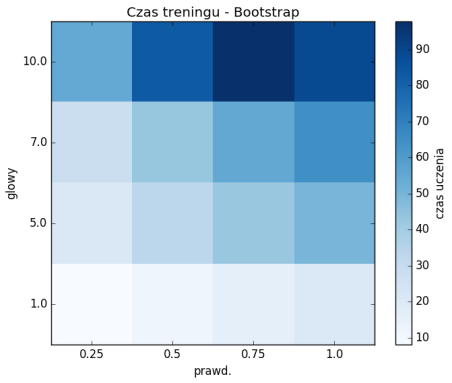



Rysunek 2.11: Bootstrap

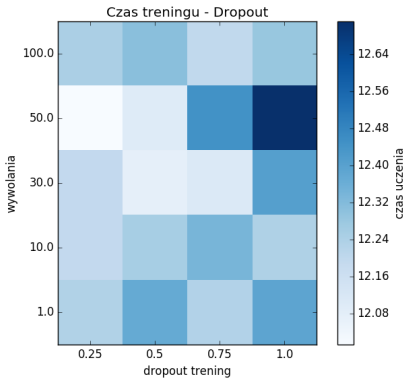



Rysunek 2.12: Dropout

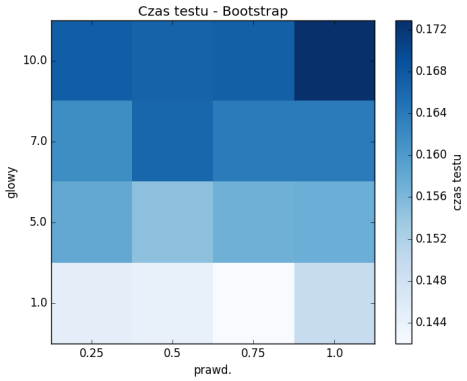
Na wykresach 2.13, 2.14, 2.15, 2.15 przedstawiono czasy treningu i testowania. Na etapie czasu testowania Boostrap wypada znacznie gorzej niż Dropout. 42 sekundy przy 5 głów i prawd=0.75 są 3.5 razy dłuższe od 12 sekund osiąganych przez Dropout na wszystkich parametrach, co jest znacznie większym narzutem niż 20% deklarowane przez autorów metody. Czas testowania ponownie korzystniejszy jest dla Bootstrapa (0.157s), ponad 5 razy mniej niż Dropout (0.84s). Czas testowania jest bardzo istotny w kontekscie Q-learningu, gdzie dla każdej klatki konieczna jest ocena jakości każdego z możliwych ruchów. Powody znacznych różnic czasowych mogą tkwić w szczółach implementacji obu metod.



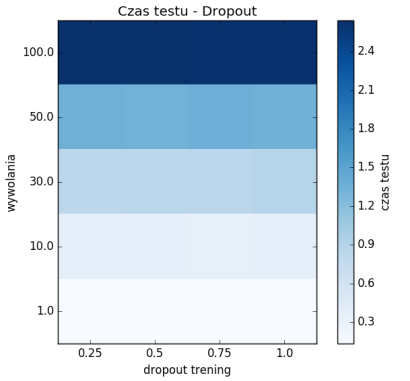

Rysunek 2.13: Dropout




Rysunek 2.14: Dropout




Rysunek 2.15: Dropout




Rysunek 2.16: Dropout

Na podstawie przeprowadzonego eksperymentu za najlepszą konfigurację Bootstrapa przyjęto 5 głów i prawdopodobieństwo 0.75, a dla Dropoutu 30 wywołań i oba prawdopodobieństwa równe 0.75

### 2.1.7 Niepewność - wyniki eksperymentu z oceną stopnia pewności

Dla wybranych w poprzednim punkcie parametrów metod współczynnik $quality_{OP}$ dla Bootstrapa wynosi 0.845 przy wariancji 0.06, a dla Dropoutu wynosi 0.925 przy wariancji 0.04.

### 2.1.8 Niepewność - wnioski

Bootstrap ma wyraźną przewagę na większości czynników. W drugim eksperymencie jego wyniki są nieznacznie gorsze, ale jako że obie metody osiągają bardzo wysoki współczynnik ten współczynnik jest pomijalny. Niestety, długi czas treningu Bootstrapa sprawia, że Dropout nie może być kategorycznie odrzucony.

# Rozdział 3

# Eksperymenty

## 3.1 Scenariusze

Eksperymenty przeprowadzono na następujących scenariuszach.

### 3.1.1 Basic

Sceneria składa się z prostokątnego pomieszczenia. Agent jest w jednym końcu pomieszczenia, a w losowym miejscu pod przeciwległą ścianą jest pojedynczy, nieruchomy przeciwnik. Agent może atakować i poruszać się bokiem w lewo i prawo. Strategia optymalna polega na przesunięciu się w kierunku przeciwnika i oddaniu pojedynczego strzału.

### 3.1.2 Defend the center

Sceneria składa się z kolistej areny. Agent jest na środku areny, a na jej krańcach losowo pojawiają się przeciwnicy, którzy poruszają się w stronę agenta, a po dotarciu do niego atakują. Agent może atakować i kręcić się w okół własnej osi w lewo i prawo. Strategia optymalna polega na kręceniu się w kółko, ignorowaniu odległych przeciwników i strzelaniu do biskich.

## 3.2 Bootstrapowane DQN - prosta implementacja

Celem tego eksperymentu było zaimplementowanie prostej wersji Bootstrapowanej DQN opisanej w [**?**] i sprawdzenie jej zachowania pod względem kierowania eksploracją i modelowania niepewności agenta.

### 3.2.1 Implementacja

Zaproponowana w [**?**] wersja ostateczna BDQN opiera się na jednej głębokiej sieci neuronowej, w której najwyższe warstwy „rozdzielały się" na $K$ róznych „głów" sieci, gdzie każda głowa odpowiadała jednej funkcji Q. Każda z funkcji Q jest uczona na podstawie osobnego próbkowanego zestawu danych, do którego z zadanym prawdopodobieństwem $p$ dołączane są wygenerowane podczas uczenia próbki. Na początku każdego epizodu losowo wybierana jest aktywna głowa, która w danym epizodzie będzie służyła za funkcję Q - dzięki temu zachowanie agenta w ramach każdego epizodu jest nieco różne, ale spójne.

Uproszczona wersja BDQN wspomniana w artykule i zaimplementowana w tym eksperymencie zakłada wykorzystanie $K$ niezależnych sieci zamiast jednej sieci z wieloma „głowami". Warto zwrócić uwagę, że to uproszczona implementacja dokładniej realizuje bootstraping, zapewniając

niezależność sieci - w łączonej sieci górne warstwy uczyły się na próbkowanych danych, natomiast dolne uczyły się na pełnym zbiorze - kosztem niezależności znacznie skrócono czas uczenia.

Miarę pewności sieci, mierzoną dla każdej podjętej decyzji, przyjęto jako liczbę głów zgadzających się z decyzją aktywnej głowy podzieloną przez liczbę głów $K$. Ostateczna miara pewności dla danego epizodu była uśrednioną wartością pewności wszystkich decyzji podjętych w tym epizodzie.

### 3.2.2 Ustawienia

Eksperyment przeprowadzono przy użyciu scenariusza *basic*. Punktem odniesienia był przykładowy agent dostarczony przez autorów VizDooma, opierający się na DQN. Kod agenta (a w szczególności architektura sieci neuronowej) posłużyła za podstawę eksperymentalnego agenta wykorzystującego BDQN. Dzięki temu jedynym zmiennym elementem w eksperymencie było badane bootstrapowanie danych dla sieci.

Eksperymenty przeprowadzano dla liczby podsieci $K = 5, 10$ i prawdopodobieństwa uwzględnienia próbki $p = 0.5, 0.75, 0.9, 1$. Agenci uczli się przez 20 epok po 2000 epizodów każda. Celem eksperymentu było wstępne zbadanie przydatności BDQN i zaobserwowanie ogólnych trendów, dlatego badania nie były wielokrotnie powtarzane.

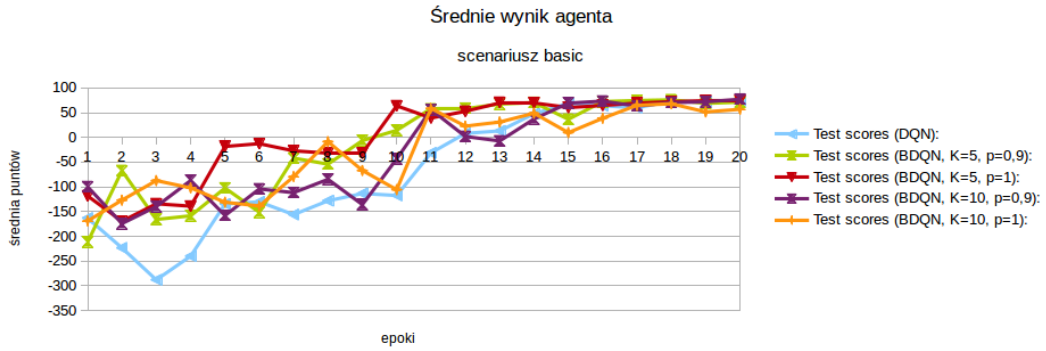
### 3.2.3 Wyniki

Eksperyment wykazał, że BDQN może uczyć się znacznie szybciej (w kontekscie liczby epizodów, nie czasu) niż zwykła DQN. Dodatkowo przyjęta miara pewności sieci wyraźnie koreluje z wynikami uzyskiwanymi przez agenta.

Jak widać na wykresie 3.1, wyniki BDQN przekraczają granicę 0 i dochodzą do ostatecznych wartości kilka epok wcześniej niż DQN. Wszystkie rozwiązania zbiegają się do porównywalnych wyników - empiryczna analiza zachowań agenta wskazuje, że nie jest to jeszcze zachowanie optymalne, ale sensowne. Akcje agenta sugerują niedokładność percepcji, co może wskazywać na zbyt prostą lub zbyt małą sieć. Zaimplementowane BDQN działa szybciej biorąc pod uwagę epoki, ale nie czas. Prostota implementacji sprawia, że odbycie każdej epoki zajmuje BDQN do trzech razy więcej czasu niż DQN. Spodziewane jest, że docelowa implementacja będzie porównywalna z DQN.

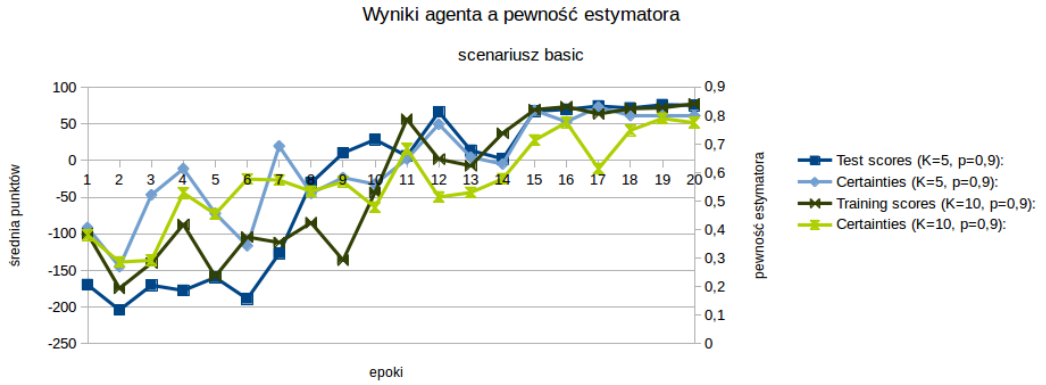
Ciekawe jest zachowanie BDQN dla różnych zestawów parametrów. Autorzy [**?**] osiągali najlepsze wyniki dla $p = 0.5$ i rosnące nieznacznie wraz ze wzrostem $K$. Badany agent dla $p = 0.5$ osiągał gorsze wyniki niż DQN. Jest to zrozumiałe zachowanie - w docelowej implementacji część sieci uczona jest w praktyce na wszystkich danych, podczas gdy badane rodzielne sieci dla $p = 0.5$ są w każdym momencie nauczone dwa razy mniejszym zestawie danych niż analogiczny agent DQN. Warto zwrócić uwagę, że dzięki losowej inicjalizacji wag nawet dla $p = 1$ uzyskane podsieci nie są identyczne.

Wyniki gorsze dla $K = 10$ niż dla $K = 5$ są natomiast sprzeczne z oczekiwaniami, przy czym to zachowanie nie wydaje się istotne przy badaniu na najprostszym scenariuszu.




Rysunek 3.1: Średnie wyniki agentów dla scenariusza *basic*

Jak widać na wykresie 3.2, pewność estymatora wyraźnie rośnie w miarę uczenia i koreluje z wynikami uzyskiwanymi przez agentów.




Rysunek 3.2: Średnie wyniki a pewność estymatora dla scenariusza *basic*

### 3.2.4 Wnioski

Eksperyment z użyciem prostej implementacji BDQN i prostego scenariusza *basic* wykazał, że BDQN ma potencjał prowadzenia skutecznej eksploracji samo w sobie, a co ważniejsze, umożliwia sensowne estymowanie stopnia pewności sieci, co stanowi podstawę do bardziej zaawansowanych technik kierowania eksploracją.



Instytut Informatyki, Wydział Informatyki
Politechnika Poznańska

Skład przy użyciu systemu LaTeX.

BibTeX:

```
@mastersthesis{ key,
    author = "Wojciech Kopeć \and ",
    title = "{Imitation Learning}",
    school = "Poznan University of Technology",
    address = "Pozna{\'n}, Poland",
    year = "2017",
}
```