Wrocław, 25. stycznia 2007

Zaawansowane Metody Sztucznej Inteligencji

Politechnika Wrocławska
Wydział Informatyki i Zarządzania
V rok studiów

Zastosowania algorytmu uczenia się ze wzmocnieniem

Temporal Difference Learning

na przykładzie konkretnego systemu

TD-GAMMON

Autor dokumentu: **STAWARZ Paweł**
Indeks: **125939**
Data wygłoszenia: **18.01.2007**
Data dostarczenia: **25.01.2007**

Prowadzący: **dr Maciej Piasecki**

Spis treści

Abstrakt

W niniejszej pracy zaprezentowany zostanie algorytm uczenia się ze wzmocnieniem *Temporal Difference Learning*. Po krótkim wprowadzeniu dotyczącym nadzorowanego i nienadzorowanego uczenia zajmiemy się opisem działania tego algorytmu jak również jego formalnymi podstawami. Zwrócimy także uwagę na jego korzyści. Następnie zapoznamy się ze spektakularnymi wynikami zastosowania tego algorytmu w praktyce na przykładzie gry backgammon.

Wprowadzenie

Algorytmy uczące się są jednymi z najbardziej popularnych podejść wykorzystania technik sztucznej inteligencji we współczesnym świecie. Rozwijane niemalże od kilkudziesięciu lat doczekały się już różnych swoich odmian. Idea powstawania tego typu algorytmów wzięła się z obserwacji procesu uczenia się człowieka. Poprzez analogię do rzeczywistych procesów poznawczych próbowano wypracować metody skutecznie uczące się określonych zadań. Wydzielono przy tym dwie zasadnicze klasy algorytmów bazujących na nadzorowanym albo nienadzorowanym uczeniu.

Pierwsza klasa algorytmów zakłada, że proces uczenia się jest w pewien sposób nadzorowany (np. przez nauczyciela). Chodzi tutaj o sytuacje, w których „uczeń" (algorytm) kształtuje swoje przyszłe zachowanie (sposób w jaki wykonuje powierzone zadania) na podstawie przedstawionych mu przykładów. Najczęściej algorytmy te wykorzystują w procesie uczenia zdefiniowany przez „nauczyciela" zbiór przykładów uczących. Każdy taki przykład składa się zazwyczaj z zestawu mierzalnych cech go opisujących oraz właściwego sposobu postępowania. Na przykład w procesie klasyfikacji obiektów, każdy obiekt ze zbioru uczącego jest już odgórnie sklasyfikowany, a zadaniem „ucznia" jest nauczenie się poprawnej (najlepiej stuprocentowej) klasyfikacji tych obiektów, aby na tej podstawie móc klasyfikować inne, niewidziane wcześniej, obiekty. Wykorzystuje się przy tym często tzw. zbiór przykładów testowych, aby zweryfikować stopień inteligencji wyuczonego algorytmu. Zbiór ten zawierałby, w przypadku klasyfikacji obiektów, nieznane „uczniowi" poprawnie sklasyfikowane obiekty.

Druga klasa algorytmów, bazujących na nienadzorowanym uczeniu, zakłada z kolei, że proces uczenia się nie jest w żaden sposób nadzorowany. Jest to pewnego rodzaju uczenie się na zasadzie prób i błędów. „Uczeń" nie dysponuje żadnymi poprawnymi przykładami zachowań. Jego zadaniem jest samodzielne ukształtowanie własnego prawidłowego postępowania. Jedynym założeniem jest fakt, że w trakcie nauki, w wyniku wykonywania określonych działań, środowisko zewnętrzne „ucznia" odpowiada pewnymi bodźcami – pozytywnymi i negatywnymi. Dzięki temu „uczeń" może czuć się „nagradzany" w sytuacji gdy zacznie osiągać cel, jak również przeciwnie – „karcony" w sytuacji gdy od tego celu zacznie znacznie zbaczać. Dlatego też mówi się również o tzw. algorytmach uczenia się ze wzmocnieniem. Wzmocnieniem jest w tym przypadku każdy bodziec zewnętrzny środowiska, w którym „uczeń" się uczy.

Bardzo łatwo znaleźć analogię pomiędzy wymienionymi wyżej klasami algorytmów uczących się a prawdziwym życiem. Już od wczesnych lat, każde dziecko podlega zarówno nienadozorowanemu jak i nadzorowanemu uczeniu. W procesie nadzorowanego uczenia bardzo często uczestniczą rodzice będący w roli nauczycieli. To oni stanowią m.in. wzorce uczące dla swojego dziecka, które podglądając je stara się jak najlepiej odwzorować. Natomiast nienadzorowane uczenie przejawia się najczęściej podczas samodzielnej zabawy, w której dziecko doznaje nowych osobliwych wrażeń. Na przykład wciskając przyciski swoich elektronicznych zabawek doświadcza efektów audio-wizualnych, które wzbudzają w nim upodobanie, a innym razem dotykając gorącego pieca doświadcza poparzenia, które wzbudza w nim strach przed ponownym takim doświadczeniem.

Temporal Difference Learning

Wstęp

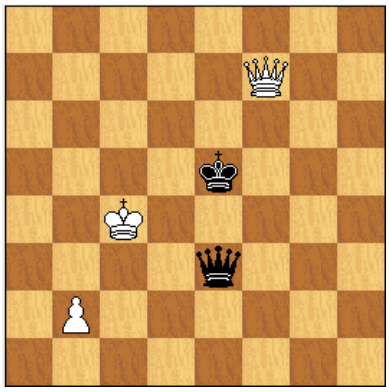
W dalszej części pracy zajmiemy się wybranym algorytmem uczenia się ze wzmocnieniem – *Temporal Difference Learning*. Algorytm ten znany również pod nazwami *TD-Learning* bądź *TD(λ)* zaliczany jest do klasy przyrostowych procedur uczących ukierunkowanych na predykcję w nie do końca poznanych systemach, tzn. takich, w których nie znany jest rzeczywisty sposób działania dla określonego stanu, w którym może się znajdować. Procedury te wykorzystują swoje przeszłe doświadczenie do przewidywania przyszłego zachowania obserwowanego systemu. Pojęcie ich przyrostowości zostanie lepiej przybliżone w dalszej części pracy. Po ewentualne szczegóły dotyczące zasad działania algorytmu *TD-Learning* warto zajrzeć do pracy R. S. Sutton`a [1], w której można znaleźć dość formalny jego opis. W pracy tej Sutton udowadnia między innymi, iż algorytm ten wykazuje lepsze wykorzystanie zdobywanego doświadczenia od innych znanych dotychczas metod uczenia oraz jest bardziej optymalny (pod względem szybkości obliczeń i zajętości pamięci) i szybciej zbieżny do docelowego rozwiązania będącego ostatecznym modelem obserwowanego systemu.

W celu rozjaśnienia mogących pojawić się we wstępie wątpliwości wyobraźmy sobie następujące nie do końca poznane przez nas systemy, w których omawiana dalej metoda uczenia mogłaby jak najbardziej znaleźć zastosowanie.

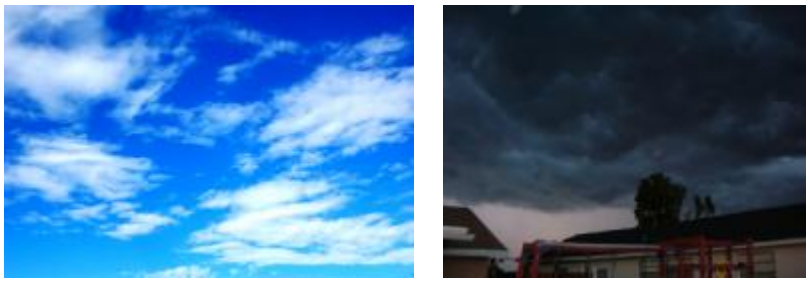
Pierwszym przykładem może być ocena pozycji szachowej (Rysunek 1.) pod kątem wygranej jednej ze stron. Niemalże nieskończoność królewskiej gry nie pozwala nam z pewnością ocenić milionów podobnych sytuacji, ponieważ potrzebowalibyśmy w tym celu rozwinąć z każdej takiej pozycji całe drzewo gry i dopiero na jego podstawie moglibyśmy stwierdzić czy przy najlepszej grze obu stron możliwa jest czyjaś wygrana. W większości przypadków jednak rozwinięcie całego drzewa gry graniczy z cudem ze względu na bardzo duży współczynnik rozgałęzienia. W przytoczonym przykładzie mamy do czynienia z pozycją, w której białe mają przewagę materialną (posiadają jednego pionka więcej). Okazuje się jednak, że stuprocentowa ocena tej pozycji nie jest do końca możliwa. Jeżeli założymy, że białe znajdują się na ruchu, to w prostu sposób mogą one doprowadzić do sytuacji, która uważana jest przez szachistów za wygraną (grając hetmanem z pola f7 na pole e7 z szachem zmuszają czarne do wymiany hetmanów doprowadzając do sytuacji, w której białe bez przeszkód mogą wykorzystać przewagę pionka wędrując nim do pola przemiany znajdującego się na ósmej linii). Jeżeli jednak założymy, że czarne znajdują się na ruchu, to sprawa nie jest już tak oczywista, ponieważ czarne mogą wykorzystać swojego hetmana w celu wiecznego szachowania białego króla i utrzymania np. remisu.

Kolejnym prostym przykładem może być prognoza pogody, a właściwie jej szczególny przypadek, w którym oceniany jest bieżący układ chmur (Rysunek 2.) pod kątem możliwości pojawienia się deszczu. Oczywiście w pewnych skrajnych sytuacjach jesteśmy w stanie stwierdzić czy będzie padać czy nie, ale mimo wszystko nie wszystkie przypadki będą dla nas tak klarowne. Właściwie prognozowanie deszczu dla pewnych układów

chmur może się sprowadzać jedynie do „wróżenia z fusów”. Jedno jest natomiast pewne – wszelkie zjawiska pogodowe wynikające z bieżących warunków atmosferycznych nie są przez nas do końca poznane.

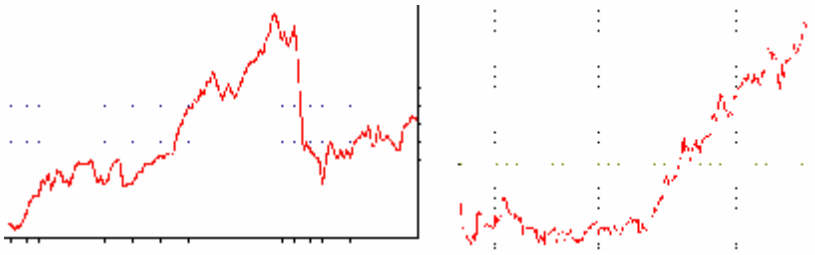


Rysunek 1. Czy dana pozycja szachowa może prowadzić do wygranej którejś ze stron?



Rysunek 2. Czy dany układ chmur może prowadzić do deszczu?

Innym przykładem może być także prognozowanie zachowań cen akcji na giełdzie wynikających z bieżących warunków ekonomicznych. Nigdy nie możemy być w stu procentach pewni czy nasze przypuszczenia są rzeczywiście słuszne.



Rysunek 3. Czy dane warunki ekonomiczne mogą prowadzić do poprawy warunków na giełdzie?

Modelowanie wszystkich powyższych systemów sprowadza się ostatecznie do nienadzorowanego uczenia, w którym wykorzystujemy wszelkie zdobyte doświadczenia w trakcie ich obserwacji do przyszłego przewidywania ich działania.

Warto jeszcze wspomnieć, iż algorytm *Temporal Difference Learning* w odróżnieniu od dotychczasowych klasycznych metod uczenia nie porównuje bieżącej predykcji z właściwą odpowiedzią dla danego przykładu, tylko porównuje kolejne predykcje aż do momentu uzyskania właściwej odpowiedzi – stąd też jego nazwa

temporal-difference-learning – co można dosłownie przetłumaczyć jako uczenie oparte na różnicach czasowych. Zasadniczym założeniem w tej metodzie jest fakt, że właściwa odpowiedź systemu jest obserwowalna dopiero po określonej sekwencji danych wejściowych. Dlatego też pomiędzy kolejnymi wejściami w procesie uczenia wykorzystywane są tylko i wyłącznie predykcje uczącego się algorytmu. Dla przykładu można przytoczyć ponownie problem prognozy pogody. Załóżmy, że naszym zadaniem jest przewidywanie w każdym dniu tygodnia czy w najbliższą sobotę będzie padał deszcz. W klasycznym podejściu predykcję z każdego dnia tygodnia porównalibyśmy z ostatecznym wyjściem (wzmocnieniem), które jest obserwowalne dopiero w sobotę i na tej podstawie wyniknęłyby ewentualne zmiany w uczonym systemie. W podejściu *TD-Learning* natomiast predykcję każdego dnia tygodnia porównywalibyśmy z predykcją z dnia następnego. W ten sposób w sytuacji gdy na przykład w poniedziałek przewidywalibyśmy, że w sobotę będzie padać z prawdopodobieństwem 50%, a we wtorek – z prawdopodobieństwem 80%, to drugie podejście spowodowałoby wzrost predykcji dla podobnych poniedziałków w przyszłości, natomiast to pierwsze spowodowałoby wzrost lub spadek w zależności od właściwego stanu na sobotę. W przykładzie tym uwidacznia się przy okazji jedna z korzyści płynąca ze stosowania algorytmu uczenia *TD-Learning* w porównaniu do dotychczas znanych – w procesie uczenia nie musimy czekać do ostatecznego wyjścia obserwowanego systemu, a co za tym idzie nie musimy składować w pamięci wszystkich obserwacji. Dzięki temu w systemach, w których właściwe wzmocnienie przychodzi po dość długim czasie, możemy zaoszczędzić sporo pamięci. Drugą zaletą jest sam fakt obliczeń – obliczenia wpływające na ewentualne zmiany systemu uczącego mogą być dokonywane w trakcie obserwacji kolejnych wejść (stanów obserwowanego systemu), a nie, tak jak w przypadku dotychczasowych metod, dopiero po otrzymaniu ostatecznej odpowiedzi.

Wszystkie zalety podejścia *TD(λ)* uwidaczniają się tak naprawdę w tzw. problemach wielokrokowych (*multi-step prediction problems*), w których informacja o prawidłowości poszczególnych predykcji uzyskiwana jest po więcej niż jednym kroku (każdy krok, to kolejna obserwacja systemu). W każdym takim kroku uzyskiwana jest tylko częściowa informacja odpowiadająca prawidłowości predykcji, na którą składa się nowa obserwacja wejściowa wraz z nową dla niej predykcją. W tego typu problemach *TD(λ)* wprowadza szereg niespotykanych dotychczas możliwości uczenia wyrażanych różnymi wartościami parametru λ. Natomiast w problemach jednokrokowych (*single-step prediction problems*) działanie omawianego algorytmu sprowadza się właściwie do działania dotychczas już znanych.

Metoda *TD(λ)* została bardzo pomyślnie wykorzystana w 1992 roku w programie TD-GAMMON grającym na bardzo wysokim poziomie w grę backgammon. Sukces tego programu doprowadził do zwiększenia zainteresowania tą metodą w zastosowaniu w różnych grach strategicznych. Realizacja tego programu zostanie jeszcze w miarę szczegółowo omówiona w ostatniej części pracy.

Definicja algorytmu

Załóżmy, że rozważamy wielokrokowy problem predykcyjny, w którym każde doświadczenie składa się z sekwencji *m* obserwacji $x_1, x_2, \mathbf{K}, x_m$ (gdzie każdy x_t jest wektorem rzeczywisto-liczbowych danych

wejściowych dotyczących obserwacji systemu w chwili t) zakończonych ostatecznym rzeczywisto-liczbowym wyjściem z (wzmocnieniem). Dla każdej sekwencji obserwacji „uczeń" generuje odpowiadającą im sekwencję predykcji $P_1, P_2, ..., P_m$ szacujących ostateczne wyjście z systemu. W ogólności każda predykcja P_t może być funkcją wszystkich poprzedzających ją obserwacji oraz wektora modyfikowalnych parametrów (wag) w – $P(x_1, x_2, \mathbf{K}, x_t, w)$. Jednakże dla uproszczenia dalszych rozważań łatwiej będzie założyć, że P_t jest funkcją obserwacji x_t oraz parametrów w – $P(x_t, w)$. Sutton stwierdza, że wszystkie inne przypadki mogą być zawsze sprowadzone do tego uproszczonego.

Każdy algorytm uczenia się definiowany jest poprzez regułę aktualizacji parametrów w:

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t , \tag{1}$$

gdzie Δw_t jest wektorem zmian wektora wag w wynikających z obserwacji w chwili t. W tym momencie zakładamy jednak, że każda aktualizacja parametrów w następuje dopiero po pełnej sekwencji obserwacji zakończonych ostatecznym wzmocnieniem. Jak się później okaże można rozważać również dwa inne przypadki: bardziej przyrostowy, w którym aktualizacja parametrów występuje po każdej obserwacji i mniej przyrostowy, w którym aktualizacja występuje po całym zbiorze uczącym, tj. wielu sekwencjach obserwacji.

Zanim wyprowadzimy ogólny wzór na przyrost Δw_t w podejściu *TD-Learning*, wyjdziemy najpierw od podejścia nadzorowanego uczenia i pokażemy, iż jest to szczególny przypadek całej rodziny procedur uczących $TD(l)$. Zauważmy, że w nadzorowanym uczeniu nasze doświadczenie wyrażone byłoby sekwencją par $(x_1, z), (x_2, z), \mathbf{K}, (x_m, z)$, a przyrost Δw_t zależałby od błędu między predykcją P_t a ostatecznym wyjściem z:

$$\Delta w_t = a\,(z - P_t)\nabla_w P_t , \tag{2}$$

gdzie a jest współczynnikiem uczenia (dodatnia liczba rzeczywista) i gradient $\nabla_w P_t$ jest wektorem pochodnych cząstkowych P_t względem kolejnych składowych wektora w. Zauważmy, że w każdej chwili t przyrost wag Δw_t zależy od wzmocnienia z, co jednocześnie oznacza, że poszczególne przyrosty nie mogą być wyznaczane dopóki wartość tego wzmocnienia nie zostanie poznana. Dlatego też wszystkie obserwacje i predykcje pojawiające się w trakcie doświadczenia muszą być skrzętnie gromadzone w pamięci do samego końca, w którym dopiero pojawia się cały proces obliczeniowy związany z aktualizacją wektora parametrów w – innymi słowy aktualizacja ta nie może odbywać się przyrostowo. Okazuje się jednak, iż procedura *TD-Learning* może doprowadzić do tego samego sposobu aktualizacji wag (2) w sposób bardziej przyrostowy. Zauważmy, że

$$(z - P_t) = \sum_{k=t}^{m} (P_{k+1} - P_k), \tag{3}$$

gdzie przyjmujemy z definicji, iż $P_{m+1} \overset{def}{=} z$. Wykorzystując równania (1)-(3) możemy w ten sposób wyprowadzić wzór na aktualizację wag w:

$$\begin{aligned} w \leftarrow w + \sum_{t=1}^{m} \Delta w_t = w + \sum_{t=1}^{m} a\left(z - P_t\right)\nabla_w P_t = w + \sum_{t=1}^{m} a \sum_{k=t}^{m} \left(P_{k+1} - P_k\right)\nabla_w P_t = \\ = w + \sum_{k=1}^{m} a \sum_{t=1}^{k} \left(P_{k+1} - P_k\right)\nabla_w P_t = w + \sum_{t=1}^{m} a\left(P_{t+1} - P_t\right)\sum_{k=1}^{t} \nabla_w P_k \end{aligned} \tag{4}$$

otrzymując ostatecznie następujący wzór na przyrost wag:

$$\Delta w_t = a\left(P_{t+1} - P_t\right)\sum_{k=1}^{t} \nabla_w P_k \,. \tag{5}$$

Jak widać obliczanie kolejnych Δw_t rzeczywiście może odbywać się w sposób przyrostowy, ponieważ zależy on od różnicy dwóch kolejnych predykcji oraz sumy wszystkich gradientów predykcji, która może być obliczana w sposób jak najbardziej przyrostowy (z każda kolejną predykcją).

Okazuje się, że w równaniu (5) przedstawiliśmy tylko szczególny przypadek procedury uczącej $TD(l)$ dla $l = 1$. Cała rodzina procedur uczących $TD(l)$ wyraża się następująco:

$$\Delta w_t = a\left(P_{t+1} - P_t\right)\sum_{k=1}^{t} l^{t-k} \nabla_w P_k \,, \tag{6}$$

gdzie $l \in [0;1]$. Jak widać w rodzinie tej uwzględnione jest wykładnicze ważenie ostatnich predykcji. Predykcje bliższe wzmocnienia są ważone z większą wagą niż te, które odbywały się na samym początku doświadczenia. Predykcje te mogą być ważone w sposób inny niż wykładniczy, aczkolwiek główną jego zaletą jest fakt, że powyższe wyliczenia mogą odbywać się w sposób jak najbardziej przyrostowy. Korzystając z równania (6), załóżmy, że

$$e_t = \sum_{k=1}^{t} l^{t-k} \nabla_w P_k \,, \tag{7}$$

a wtedy

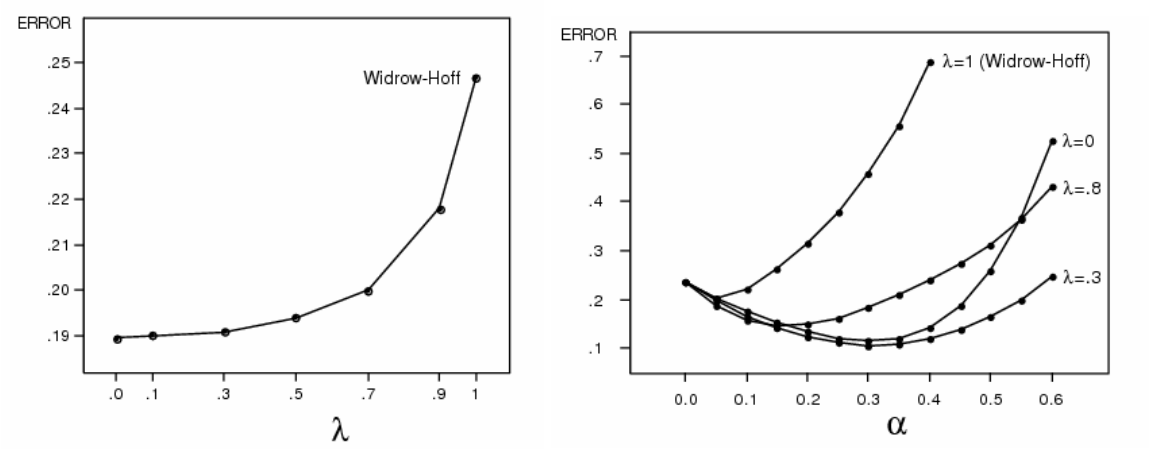
$$\Delta w_t = a\left(P_{t+1} - P_t\right)e_t \,. \tag{8}$$

Idąc dalej tym tropem otrzymujemy, że

$$e_{t+1} = \sum_{k=1}^{t+1} l^{t+1-k} \nabla_w P_k = l\left(\sum_{k=1}^{t} l^{t-k} \nabla_w P_k\right) + \nabla_w P_{t+1} = l\, e_t + \nabla_w P_{t+1}, \tag{8}$$

co oznacza, że kolejne wartości parametru e_t mogą być wyliczane w sposób przyrostowy na podstawie poprzedniej jej wartości oraz bieżącego gradientu predykcji $\nabla_w P_t$. Powyższy fakt w istocie w znaczny sposób ułatwia implementację omawianego tu algorytmu jak również sprawia, że wszystkie, wydawałoby się zagmatwane obliczenia odbywają się w bardzo krótkim czasie.

Sutton w swojej pracy [1] bardzo szczegółowo omawia wszystkie tajniki algorytmu *TD-Learning*. Wykazuje on między innymi w swoich eksperymentach (związanych ze spacerem losowym) zależności średniego błędu predykcji najlepiej wyuczonego systemu od faktycznych wartości obserwowanego systemu dla różnych wartości parametru a (współczynnika uczenia) oraz parametru l (parametru *TD-Learning*) – Rysunek 4.



Rysunek 4. Średni błąd predykcji w zależności od wartości parametrów α oraz λ.

Warto zauważyć, że eksperymenty Suttona wykazały, iż system uczący najlepiej odwzorowuje obserwowany system dla wartości parametrów $a = 0.3$, $l = 0.3$ (tzn. z najmniejszym błędem), natomiast najbardziej uogólnia ten system dla wartości $l = 1$ wraz z dużym współczynnikiem uczenia a.

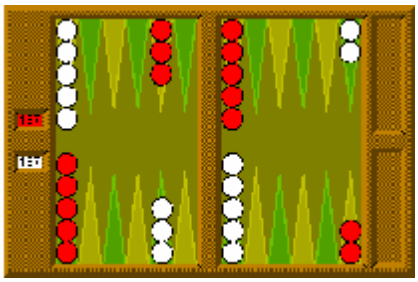
TD-Gammon

Wstęp

W niniejszej części pracy skupimy się już na praktycznym zastosowaniu algorytmu uczenia się ze wzmocnieniem *TD-Learning*. Jak to już wcześniej zostało wspomniane, TD-Gammon jest najbardziej spektakularnym wynikiem tego zastosowania. Wyniki eksperymentów przeprowadzonych przez twórcę, Gerarda Tesauro, przeszły jego nawet najśmielsze oczekiwania. TD-Gammon jest programem grającym w grę backgammon. Program nauczył się w nią grać dzięki metodzie uczenia *TD-Learning*. Zanim jednak przejdziemy do omówienia niektórych szczegółów projektowych tego programu, przytoczymy sobie najważniejsze zasady gry backgammon.

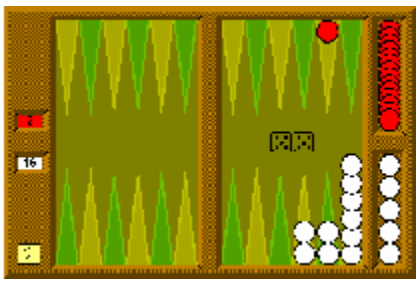
Gra backgammon jest jedną z najpopularniejszych (obok szachów) gier planszowych, w którą grają tysiące, a może nawet miliony osób na całym świecie. W grę backgammon rozgrywane są przeróżne międzynarodowe turnieje i regularne mistrzostwa świata. Podobno profesjonalnych graczy w backgammon

jest o wiele więcej niż profesjonalnych graczy w szachy. Jest to gra strategiczna nie pozbawiona jednak pewnego elementu losowości. W pozycji początkowej na planszy znajduje się 15 pionków jasnego koloru oraz 15 pionków koloru ciemnego (Rysunek 5.) – każdy z dwóch graczy dysponuje jednym kolorem pionków.



Rysunek 5. Pozycja początkowa gry backgammon.

Na planszy znajdują się 24 pola (w kształcie szpiczastych trójkątów), na których może znajdować się dowolna liczba pionków tylko jednego koloru. Gracze wykonują ruchy na przemian. Pierwszeństwo gry jest losowane przez rzut sześcienną kostką przez każdego z graczy – gracz z większa liczbą oczek zaczyna. Ruch gracza składa się na rzut dwoma sześciennymi kostkami oraz odpowiednie przemieszczenie pionków. Jasne pionki przesuwają się przeciwnie do kierunku ruchu wskazówek zegara, natomiast ciemne pionki przesuwają się zgodnie z tym kierunkiem. Liczba wyrzuconych oczek decyduje o tym, o ile pól do przodu (z perspektywy danego koloru) mogą zostać przemieszczone pionki. Jeżeli przykładowo wyrzucone zostaną liczby 6 i 3, oznacza to, że gracz może przemieścić jednego pionka o łączną liczbę 9 pól do przodu, albo dwa pionki – odpowiednio o 6 oraz 3 pola. Jeżeli na obu kostkach wyrzucona została ta sama liczba oczek, wówczas gracz przemieszcza swoje pionki, tak jakby wyrzucił cztery razy tę liczbę. Pionek może zostać przemieszczony na inne pole tylko i wyłącznie wtedy, gdy stoi na nim dowolna liczba pionków tego samego koloru albo co najwyżej jeden pionek przeciwnika. Jeżeli na danym polu znajdował się jeden pionek przeciwnika, to w wyniku tego ruchu, pionek ten jest zbijany i trafia na tzw. belkę – pozycję, z której rozpoczyna podróż dookoła planszy. Jeżeli na belce znajdują się jakiekolwiek pionki gracza, to jest on zobowiązany do ruchu nimi w pierwszej kolejności. Celem gry każdego z graczy jest przetransportowanie wszystkich pionków do ostatniej kwarty planszy (sześć ostatnich pól podróży), z którego następnie wykonywana jest ewakuacja pionów poza planszę (Rysunek 6.). Gracz, który jako pierwszy pozbędzie się wszystkich swoich pionków wygrywa grę.

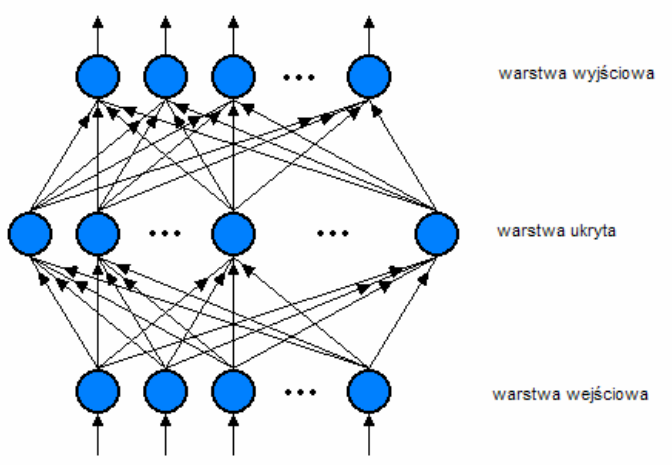


Rysunek 6. Końcówka gry backgammon.

Po przejrzeniu powyższych podstawowych zasad gry backgammon możemy sobie odpowiedzieć na pytanie: jak bardzo złożona jest ta gra? Oczywiste jest, iż liczba możliwych rozmieszczeń 30 pionków na 26 polach (24 + belka + miejsce poza planszą) jest bardzo duża. Ogromny jest również współczynnik rozgałęzienia analizowanego drzewa gry. Zakładając, że gracz będący na ruchu wyrzucił już odpowiednią liczbę oczek, to średnio rzecz biorąc może on wykonać swój ruch na około 20 różnych sposobów. Wchodząc jednak głębiej w drzewo gry można zauważyć, że z każdym poziomem pojawia się około 36*20=720 możliwych rozgałęzień (36 to liczba różnych kombinacji wyrzuconych oczek na obu kostkach). Chociaż pod koniec gry współczynnik rozgałęzienia zaczyna się znacznie zmniejszać, to jednak jego średnia wartość utrzymuje się w granicach 400. Łatwo zatem wyobrazić sobie, iż coraz głębsza analiza takiego drzewa gry staje się bardzo uciążliwa – a przede wszystkim czasochłonna. Dlatego też potrzebna nam jest odpowiednia heurystyka skutecznie oceniająca określony stan gry backgammon, aby wykorzystać ją na odpowiednim poziomie drzewa gry. Doskonałym sposobem odnalezienia tej heurystyki okazało się wykorzystanie w tym celu procedur uczących *TD(λ)* w połączeniu z trójwarstwową siecią neuronową uczoną z wykorzystaniem wstecznej propagacji błędów dla poprzedzających warstw.

Wykorzystanie TD-Learning

Gra backgammon w oczywisty sposób może być rozumiana jako wielokrokowy problem predykcyjny, w którym wzmocnienie (ostateczny wynik gry) następuje po sekwencji określonych ruchów zmieniających stan rozmieszczenia pionków na planszy. Jak to zostało pokazane w poprzednim rozdziale algorytm uczenia *TD-Learning* doskonale nadaje się do tego typu problemów. Algorytm ten został zaaplikowany w wersji TD-GAMMON 0.0 do uczenia trójwarstwowej sieci neuronowej (Rysunek 7.), w której warstwa wyjściowa składała się z 1-3 neuronów, warstwa ukryta z 40-80 oraz warstwa wejściowa ze 198. W uproszczeniu wyjście tej sieci miało oszacowywać ostateczny wynik gry dla określonych stanów gry (rozmieszczeń pionków na planszy). Liczba neuronów w warstwie ukrytej była najprawdopodobniej dobierana eksperymentalnie, natomiast w warstwie wejściowej zależała ściśle od sposobu kodowania aktualnego stanu gry.



Rysunek 7. Struktura trójwarstwowej sieci neuronowej z pełnymi połączeniami pomiędzy warstwami.

W pierwszej wersji (0.0) Gerard Tesauro wykorzystał najprostszy i najbardziej surowy sposób reprezentacji stanu planszy nie wykorzystujący żadnej dodatkowej wiedzy na temat dodatkowych cech gry backgammon. Z każdym z 24 pól planszy stowarzyszonych było 8 neuronów – po 4 neurony dla każdego koloru pionków. Zestaw 4 neuronów określał liczbę pionków danego koloru znajdujących się na tym polu w następujący sposób:

- jeśli na polu nie znajdują się żadne pionki, to każdy z neuronów przyjmuje wartość 0,
- jeśli na polu znajduje się dokładnie jeden pionek, to pierwszy neuron przyjmuje wartość 1, a pozostałe – wartość 0,
- jeśli na polu znajdują się dokładnie dwa pionki, to pierwsze dwa neurony przyjmują wartość 1, a pozostałe dwa – wartość 0,
- jeśli na polu znajdują się co najmniej trzy pionki, to pierwsze trzy neurony przyjmują wartość 1, natomiast ostatni – wartość $\frac{n-3}{2}$, gdzie n oznacza liczbę pionków znajdujących się na tym polu.

Powyższy sposób kodowania rozmieszczenia pionków na planszy zapewnia nam już w tej chwili 8*24=192 neuronów wejściowych. Dwa kolejne neurony wyznaczały odpowiednio liczbę jasnych oraz ciemnych pionków znajdujących się na belce przyjmując wartość $\frac{n}{2}$, gdzie n jest liczbą pionków na tej belce. Dwa kolejne neurony wyznaczały z kolei odpowiednio liczbę jasnych oraz ciemnych pionków, które zostały już szczęśliwie usunięte z planszy przyjmując wartość $\frac{n}{15}$, gdzie n jest liczbą usuniętych pionków. Dwa ostatnie neurony wyznaczały natomiast w sposób binarny kolor, który jest na ruchu.

Dla powyższego sposobu reprezentacji planszy, sieć neuronowa przyjmując odpowiednie wartości wejściowe obliczała swoje wyjście w klasyczny sposób. Z każdym połączeniem pomiędzy neuronami warstwy wejściowej a neuronami warstwy ukrytej, a także pomiędzy neuronami warstwy ukrytej a neuronami warstwy wyjściowej, utożsamiony jest pewien modyfikowalny parametr (waga) przyjmujący wartości rzeczywisto-liczbowe. Wyjście każdego neuronu warstwy ukrytej bądź wyjściowej wyznaczane jest jako wartość sigmoidalnej funkcji dla wartości argumentu będącego sumą iloczynów wyjść neuronów poprzedzającej warstwy z odpowiadającymi im wagami (na połączeniach). Po ewentualne szczegóły działania sieci neuronowych warto zajrzeć do [3].

Mając już zaprojektowaną sieć neuronową musimy się jeszcze zastanowić w jaki sposób będziemy ją uczyć. Tesauro zaproponował następujący sposób uczenia: sieć neuronowa gra jak najwięcej partii z samą sobą. W wersji 0.0 wybór ruchu ograniczał się do analizy tylko i wyłącznie pierwszego poziomu drzewa. Oczywiście wybierany był zawsze ten ruch, dla którego stan planszy po jego wykonaniu był najlepiej oceniany przez sieć. Każda gra traktowana była jako oddzielne doświadczenie składające się z sekwencji obserwacji (poszczególnych pozycji) oraz ostatecznego wzmocnienia (wyniku gry). Tesauro skorzystał także z najbardziej przyrostowego schematu uczenia *TD(λ)*, tzn. wszystkie aktualizacje wag następowały po każdym ruchu. Stan początkowy sieci, tzn. wszystkie modyfikowalne wagi zostały wybrane w sposób losowy

jako małe wartości rzeczywisto-liczbowe. Wynikiem tego była całkowicie losowa gra tego programu na początku. Jednakże już po kilkuset rozegranych partiach zaczął być zauważalny ciągły wzrost jego umiejętności.

Wyniki eksperymentów

Ku wielkiemu zaskoczeniu, po rozegraniu 300 000 gier z samym sobą, TD-Gammon 0.0 nauczył się grać równie mocno jak najlepszy znany do tej pory inny program grający w backgammon (wykorzystujący bardziej wyrafinowane cechy gry) – Neurogammon, program Tesaura oparty również na sieci neuronowej, ale nie uczony metodą *TD(λ)*. Najbardziej zadziwiający był fakt, że TD-Gammon bez żadnej wyszukanej wiedzy mógł rzeczywiście dorównać najlepszym swoim konkurentom. Oczywistym następstwem tego sukcesu była kontynuacja tych eksperymentów z wykorzystaniem wyspecjalizowanych cech gry backgammon w wersji TD-Gammon 1.0, które jak się można spodziewać zakończyły się jeszcze większym sukcesem. Siła gry TD-Gammon 1.0 była już nieporównywalnie większa od innych znanych programów grających w backgammon i zaczęła poważnie zagrażać prawdziwym arcymistrzom tej gry. Kolejne wersje tego programu były ulepszane pod kątem drzewa przeszukiwań (głębokość, wybór kolejności analizowanych węzłów) oraz odpowiednim doborem liczby neuronów w warstwie ukrytej. Wyniki eksperymentów poszczególnych wersji zostały zestawione w tabeli (Rysunek 8.).

Wersja	Liczba neuronów w w. ukrytej	Liczba gier treningowych	Przeciwnik	Wynik
TD-Gammon 0.0	40	300 000	Inne programy	dorównał najlepszemu
TD-Gammon 1.0	80	300 000	Robertie, Magriel…	-13 pkt / 51 gier
TD-Gammon 2.0	40	800 000	Różni arcymistrzowie	-7 pkt / 38 gier
TD-Gammon 2.1	80	1 500 000	Robertie	-1 pkt / 40 gier
TD-Gammon 3.0	80	1 500 000	Kazaros	+6 pkt / 20 gier

Rysunek 8. Wyniki eksperymentów przeprowadzane z kolejnymi wersjami programu TD-Gammon.

Kolejne wersje programu TD-Gammon wykazywały znaczny wzrost umiejętności gry. W dwóch ostatnich pozycjach tabeli z Rysunku 8. przedstawiono wyniki gier z dwoma najlepszymi graczami backgammon. Kazaros ówczesny mistrz świata po równej grze, ostatecznie przegrał sześcioma punktami. Warto jeszcze na zakończenie dodać jako ciekawostkę fakt, iż program TD-Gammon przyczynił się do zmiany sposobu dotychczasowego myślenia o grze początkowej. TD-Gammon nauczył się grać pewne otwarcia całkowicie inaczej od ówczesnych mistrzów. Okazało się, iż podejście tego programu zostało później dokładnie przeanalizowane, a wielcy mistrzowie zaczęli czerpać z niego wiedzę.

Literatura

[1] Richard S. Sutton. *Learning to Predict by the Methods of Temporal Differences*. Machine Learning 3: 9-44, 1988 Kluwer Academic Publishers, Boston.

[2] Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning: An Introduction. 11.1. TD-Gammon.* MIT Press, Cambridge, MA, 1998, A Bradford Book.

[3] Ryszard Tadeusiewicz. *Sieci neuronowe*. Warszawa, Akademicka Oficyna Wydawnicza RM, 1993.