Zaawansowane Metody Inteligencji Obliczeniowej

Lab 8: Metody aproksymacyjne – wprowadzenie

Michał Kempka Marek Wydmuch

29 kwietania 2021

"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20

1 Wprowadzenie

Dotychczas zajmowaliśmy się problemami w formie tabelarycznej, w której każdy ze stanów jest unikatowy i nieporównywalny z innymi stanami. Wiele jednak zadań do których chcielibyśmy użyć uczenia ze wzmocnieniem charakteryzuje się ogromną, jeśli nie nie skończoną przestrzenią stanów. W takim przypadku nie możemy oczekiwać, że znajdziemy optymalną funkcje wartości stanów i akcji oraz optymalną politykę. Zamiast tego naszym celem staje się znalezienie dobrej aproksymacji rozwiązania przy wykorzystaniu ograniczonej ilości danych i zasobów obliczeniowych. Przy ogromnej przestrzeni stanów, nie tylko wielkość tabeli jest problemem ale również równie ogromna ilość danych by móc wyestymować w miarę dokładie wartości. Żeby móc poradzić z takimi problemem, musimy być w stanie w przydatny sposób uogólnić doświadczenie pozyskaną z zaobserwowanych wcześniej stanów podobnych do nowo napotkanych. Na szczęście istnieje już wiele skutecznych metod generalizacji wiedzy z przykładów, nie musi więc wymyślać nowych metod, a raczej zastosować istniejące do problemu uczenia ze wzmocnieniem.

2 Aproksymacja funkcji

Na dzisiejszych laboratoriach zaczniemy od aproksymacji funkcji wartości stanów oraz stanów i akcji $\hat{v}(s, \boldsymbol{w}) \approx v^*(s), \hat{q}(s, a, \boldsymbol{w}) \approx q^*(s, a)$ gdzie w obu wypadkach $\boldsymbol{w} \in \mathbb{R}^d$ jest wektorem wag, a $\hat{v}, \hat{q}$ mogą być funkcją liniową reprezentacji stanów, siecią neuronową albo innym rodzajem estymatora. Generalnie chcemy by ilość wag była zdecydowanie mniejsza niż przestrzeń stanów ($d \ll |\mathcal{S}|$), a zmiana wag wpływała na estymaty dla wielu stanów lub par stanów i akcji.

Dotychczas wszystkie omawiane metody dokonywały aktualizacji estymat funkcji wartości, "przesuwajac" jej wartość kierunku zaobserwowanej wartości dla danego stanu lub pary stanu i akcji. Od teraz będziemy taką aktualizację oznaczać $s \mapsto u/s, a \mapsto u$, gdzie s oznacza stan, a akcje, a u cel aktualizacji. Naturalne więc wydaje się traktowanie krotki $(s/(s, a), u)$ jako przykład uczący dla metod uczenia nadzorowanego (ang. supervised learning), a dokładnie metod regresji. Niektóre z metod uczenia nadzorowanego bardziej niż inne nadają się do zastosowania w przypadku uczenia ze wzmocnieniem. Ważne jest by uczenie się estymatora mogło być przyrostowe (online) i odbywać się w trakcie interakcji agenta ze środowiskiem. Dodatkowo, uczenie ze wzmocnieniem wymaga estymatorów, które są w stanie uczyć się z niestacjonarnych danych (danych, które zmieniają się z czasem). Z powyższych powodów my skupimy się na aproksymatorach, opierających się o metodę spadku wzdłuż gradientu.

2.1 Wartość stanu i ewaluacja polityki

Tak jak poprzednio zacznijmy od estymacji wartości stanów $\hat{v}(s, \boldsymbol{w}) \approx v^*(s)$. W tabelarycznej formacji problemów byliśmy w stanie wyestymować funkcję wartości dla każdego stanu dokładnie. Dodatkowe estymata dla

każdego stanu była niezwiązana z estymatami dla innych stanów. W przypadku aproksymacji aktualizacja w jednym stanie wpływa na aproksymację dla innych stanów i nauczenie się dokładnych wartości funkcji wartości może nie być możliwe, dlatego potrzebujemy zdefiniować miarę jakości dla naszego problemu aproksymacji. Ponieważ mierzymy się z problemem regresji, przyjmiemy na razie jako naszą miarę jakości błąd kwadratowy:

$$MSE(\boldsymbol{w}) = \mathbb{E}\left[\frac{1}{2}(\hat{v}(S_t, \boldsymbol{w}_t) - v_\pi)^2\right] \tag{1}$$

$$= \sum_{s \in \mathcal{S}} P(s)\left[\frac{1}{2}(\hat{v}(S_t, \boldsymbol{w}_t) - v_\pi)^2\right], \tag{2}$$

gdzie $P(s)$ to prawdopodobieństwo stanu, $\sum_{s \in \mathcal{S}} P(s) = 1$, a $\hat{v}(S_t, \boldsymbol{w}_t) - v_\pi$ to różnica aproksymowanej wartości $\hat{v}(S_t, \boldsymbol{w}_t)$ a prawdziwej wartości v_π. Mając już wybraną miarę jakości, możemy wyznaczyć ogólną regułę aktualizacji wag:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla(\hat{v}(S_t, \boldsymbol{w}_t) - v_\pi)^2 \tag{3}$$

$$= \boldsymbol{w}_t - \alpha(\hat{v}(S_t, \boldsymbol{w}_t) - v_\pi)\nabla\hat{v}(S_t, \boldsymbol{w}_t) \tag{4}$$

Niestety wartość v_π nie są dla nas znane, dlatego by dokonywać aktualizacji musimy zastąpić v_π jago przybliżeniem.

Zaaplikujmy teraz naszą regułę aktualizacji do poznanego już algorytmu TD(0), w celu jako przybliżeniem v_π użyjemy $R_{t+1} + \gamma\hat{v}(S_t, \boldsymbol{w}_t)$, co da nam w efekcie poznany już błąd TD. Algorytm 1 przypomina jak wyglądał pseudocode tego algorytmu dla wersji tabelarycznej. Algorytm 2 prezentuję wersję zmodyfikowaną korzystającą z modelu aproksymującego funkcję wartości stanu.

Algorytm 1: Ewaluacja polityki za pomocą algorytmu TD(0)

Inicjalizacja:
Dowolnie zainicjalizowane $V(s)$ dla wszystkich $s \in \mathcal{S}$
$V(s) \leftarrow 0$ jeśli s jest stanem terminalnym.
Dana dowolna polityka π.
repeat
  if *S nieustawiony lub terminalny* **then**
    rozpocznij nowy epizod i zainicjalizuj
    $S \leftarrow S_0$
  Wykonaj akcję $\pi(S)$, zaobserwuj nagrodę R i następnik S'.
  $V(S) \leftarrow V(S) - \alpha(V(S) - (R + \gamma v(S'))$
  $S \leftarrow S'$
until *warunek stopu*;

Algorytm 2: Ewaluacja polityki za pomocą algorytmu TD(0) z aproksymacją

Inicjalizacja:
Różniczkowalna funkcja $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Zainicjalizowane wagi $\boldsymbol{w} \in \mathbb{R}^d$
Dana dowolna polityka π.
repeat
  if *S nieustawiony lub terminalny* **then**
    rozpocznij nowy epizod i zainicjalizuj
    $S \leftarrow S_0$
  Wykonaj akcję $\pi(S)$, zaobserwuj nagrodę R i następnik S'.
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\left(\hat{v}(S, \boldsymbol{w}) - (R + \gamma\hat{v}(S', \boldsymbol{w}))\right)\nabla\hat{v}(S, \boldsymbol{w})$
  $S \leftarrow S'$
until *warunek stopu*;

2.2 Metody tabelaryczne a aproksymacja

Pokaż, że tabelaryczne TD(0) jest specjalnym przypadkiem algorytmu TD(0) z liniową aproksymacją. Jaką reprezentację wektorową powinny mieć stany?

2.3 Wartość stanu i akcji oraz wyznaczanie polityki

Z aproksymacji funkcji wartości stanu poprzez analogie łatwo nam teraz przejść do aproksymacji funkcji wartości stanu i akcji $\hat{q}(s, a, \boldsymbol{w}) \approx q^*(s, a)$ i zastosować ją np. w algorytmie Q-Learning:

Algorytm 3: Pseudokod dla algorytmu Q-Learning

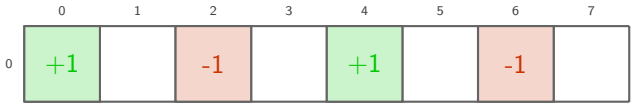
Inicjalizacja:
Różniczkowalna funkcja $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Zainicjalizowane wagi $\boldsymbol{w} \in \mathbb{R}^d$
Dana dowolna polityka π zgodna z początkowym wagami $\boldsymbol{w}$
repeat
  if S *nieustawiony lub terminalny* **then**
    Rozpocznij nowy epizod
    $S \leftarrow S_0$
  $A \leftarrow \pi(S, \hat{q}, \boldsymbol{w})$
  Wykonaj akcję A, zaobserwuj nagrodę R i następnik S'
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left(\hat{q}(S, A, \boldsymbol{w}) - \left(R + \gamma \max_{a \in \mathcal{A}} \hat{q}(S', a, \boldsymbol{w}) \right) \right) \nabla \hat{q}(S, A, \boldsymbol{w})$
  $S \leftarrow S'$
until *warunek stopu*;

2.4 Replay Memory

W czasie uczenia aproksymatora, krotka po krotce imitujemy schemat uczenia nadzorowanego Online Stochasitc Gradient Descent. Jest to wygodne, lecz niestety dziedziczy wady SGD: aktualizacje są ze sobą silnie skorelowane, a gradienty mają bardzo dużą wariancję. By zaradzić tym problemom powstała idea tzw. 'replay memory'. Po napotkaniu krotki s, a, r, s' algorytm nie uaktualnia parametrów funkcji, lecz zapisuje ją do pamięci. Pamięć ma oczywiście ograniczoną pojemność więc w przypadku wypełnienia całej pamięci, najstarsza krotka zostanie usunięta by pomieścić nową. Pamięć taka pozwala nam okresowo (np. co kilka przejść, lub co przejście) pobierać losową próbkę wielu przejść (liczba przejść wyznaczona przez hiperparametr), liczyć średni gradient i go aplikować. Na początku uczenia pamięć jest pusta więc typowo, pierwsze kroki (np. 5000, lecz zależy to oczywiście od problemu i preferencji) służy do wstępnego wypełnienia pamięci. Pseudokod 4 prezentuje działanie algorytmu q-learning z użyciem replay memory.
Pytanie: Wyobraźmy sobie sytuację gdzie w naszej pamięci są przejścia, które dostarczają nam cenniejszych informacji niż inne (np. mają większy błąd TD). Dobrym pomysłem mogłoby być ich prioretyzowanie względem mniej informatywnych przejść. Czy możemy to bezkarnie uczynić?

2.5 Grid World

Rozważmy środowisko z rysunku 1. Zakładając, że stany są reprezentowane przez współrzędną x, a do wyboru mamy 2 deterministyczne akcje (poruszanie się lewo/prawo), czy agent używający aproksymacji liniowej nauczy się optymalnej polityki? Załóż, że świat się zapętla (pole 0 jest obok pola 7) oraz nie ma stanów terminalnych.



Rysunek 1: TODO

Algorytm 4: Pseudokod dla algorytmu Q-Learning

Inicjalizacja:
Różniczkowalna funkcja $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Zainicjalizowane wagi $\boldsymbol{w} \in \mathbb{R}^d$
Dana dowolna polityka π zgodna z początkowym wagami $\boldsymbol{w}$
Zainicjalizuj pamięć $\mathcal{M} = \emptyset$
Parametr I określające co ile kroków uczymy się z pamięci
Parametr J określający ile doświadczeń wybieramy z pamięci
repeat
  for $i \leftarrow 1, \ldots, I$ **do**
    if S *nieustawiony lub terminalny* **then**
      Rozpocznij nowy epizod
      $S \leftarrow S_0$
    $A \leftarrow \pi(S, \hat{q}, \boldsymbol{w})$
    Wykonaj akcję A, zaobserwuj nagrodę R i następnik S'
    $\mathcal{M} \leftarrow \mathcal{M} \cup \langle S, A, R, S' \rangle$
    $S \leftarrow S'$
  for $j \leftarrow 1, \ldots, J$ **do**
    $\langle S, A, R, S' \rangle \leftarrow$ losowo wybrane z $\mathcal{M}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left(\hat{q}(S, A, \boldsymbol{w}) - \left(R + \gamma \max_{a \in \mathcal{A}} \hat{q}(S', a, \boldsymbol{w}) \right) \right) \nabla \hat{q}(S, A, \boldsymbol{w})$
until *warunek stopu*;

2.6 Ekstrakcja cech i Sieci Neuronowe

W ogólności funkcja liniowa ma ograniczoną siłę ekspresji (jak widać np. na przykładzie z Rysunku 1. Jednym ze sposobów na zaradzenie temu problemowi jest ręczne stworzenie cech, które pozwolą aproksymatorowi liniowemu lepiej odzwierciedlić faktyczną funkcję. W skrajnym przypadku, możemy wyobrazić sobie cechę, która jest wartością Q(s,a) - wtedy aproksymator musi się nauczyć funkcji identycznościowej. Niemniej, abstrahując od totalnego braku realizmu takiego scenariusza, podejście tworzenia cech posiada szereg wad. Stworzenie odpowiednich cech wymaga wiedzy na temat środowiska i musi być jeszcze bardziej dostosowane do konkretnego problemu. Alternatywą dla ręcznego tworzenia cech jest zaimplementowanie funkcji bardziej skomplikowanej niż liniowa np. sieci neuronowych - mając taką sieć liczymy, że warstwy nauczą się użytecznych cech samodzielnie.
Pytanie: Zaproponuj cechę, której dodanie sprawiłaby, że aproksymacja liniowa dla problemu z rysunku 1 zadziała lepiej niż aproksymacja liniowa?

Literatura

[1] Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

[2] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Fundusze Europejskie Polska Cyfrowa
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Rozwoju Regionalnego

"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20