Zaawansowane Metody Inteligencji Obliczeniowej

Wykład 5: Matody przybliżone

Michał Kempka Marek Wydmuch Bartosz Wieloch

21 marca 2022

Fundusze Europejskie
Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

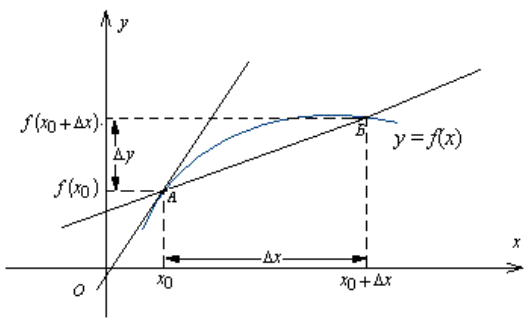
"Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)",
projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20

Plan wykładu

Pochodna

- Pochodna funkcji — miara szybkości zmian wartości funkcji względem zmian jej argumentów.
- Dla funkcji rzeczywistej $y = f(x)$ ($f : \mathbb{R} \to \mathbb{R}$ — funkcja o dziedzinie $\mathbb{R}$ i przeciwdziedzinie $\mathbb{R}$), pochodna w punkcie x_0 (o ile istnieje):

$$\lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}.$$



Pochodna

- Pochodna oznaczana jest często jednym z poniższych oznaczeń:

$$\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}\,,\, \frac{dy}{dx}\,,\, \frac{d}{dx} f(x)\,,\, f'(x)\,.$$

- Jeśli pochodna funkcji $f : (a, b) \to \mathbb{R}$ istnieje w każdym punkcie przedziału (a, b), funkcja f jest różniczkowalna w przedziale (a, b).
- Różniczkując funkcję f otrzymujemy pierwszą pochodną f', która może być również różniczkowalna w przedziale (a, b), różniczkując pierwszą pochodną f' otrzymujemy drugą pochodną f''.

Pochodna funkcji złożonej

- Pochodna funkcji złożonej obliczamy w następujący sposób:

$$h(x) = f(g(x))$$
$$h'(x) = (f(g(x)))' = f'(g(x)) \cdot g'(x)$$

- Stosując inną notację. Jeśli zmienna z zależy od zmiennej y, która z kolei zależy od x:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

Pochodna cząstkowa

- Pochodna cząstkowa — dla danej funkcji wielu zmiennych jest to pochodna względem jednej z jej zmiennych przy ustaleniu pozostałych, oznaczana często jednym z poniższych oznaczeń:

$$\frac{\partial f}{\partial x}, f'_x \,.$$

Gradient

- Gradient — wektor pochodnych cząstkowych funkcji $f : \mathbb{R}^n \to \mathbb{R}$ oznaczany jako $\nabla f(\boldsymbol{x})$, który jest funkcją $\mathbb{R}^n \to \mathbb{R}^n$:

$$\nabla f(\boldsymbol{x}) = \nabla f \left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}$$

Hesjan (Macierz Hessego)

- Hesjan (ang. Hessian), Macierz Hessego — symetryczna, kwadratowa macierz drugich pochodnych funkcji $f : \mathbb{R}^n \to \mathbb{R}$ oznacza jako $\nabla^2 f(\boldsymbol{x})$ lub $H(\boldsymbol{x})$, która jest funkcją $\mathbb{R}^n \to \mathbb{R}^{n \times n}$

$$H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x}) = \nabla^2 f \left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial^2 x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial^2 x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial^2 x_n} \end{bmatrix}$$

Znajdowanie minimum funkcji różniczkowalnej

- Dwukrotnie różniczkowalna funkcja $f : \mathbb{R}^n \to \mathbb{R}$ ma minimum lokalne w punkcie x^* jeśli $\nabla f(x^*) = 0$, hesjan jest dodatnio określony (tj. $\forall_{x \neq 0} x^T H(x^*) x > 0$).

Regresja liniowa

- Macierz $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{n \times m}$, gdzie wiersz i jest wektorem $\boldsymbol{x}_i$ reprezentującym pojedynczy punkt w danych i zawierającym m wartości określających jego cechy.
- Wektor $\boldsymbol{y} \in \mathbb{R}^n$, w którym wartość y_i odpowiada cechą w wektorze $\boldsymbol{x}_i$.
- Chcemy znaleźć **model** (często nazywany hipotezą i dlatego oznaczany jako $h(\boldsymbol{x})$), która najlepiej przewiduje wartość y_i ($\forall_{i \in \{1,2 \ldots n\}} h(\boldsymbol{x}_i) \approx y_i$).
- Jak nazwa wskazuje w wypadku regresji liniową naszym modelem jest funkcja liniowa o współczynnikach $\boldsymbol{w}$ i wyrazie wolnym b (alternatywnie często bias jest omijany i zamiast tego dopakowana jest kolumna do macierzy X, zawierająca 1 dla każdego wiersza, dla uproszczenia my będziemy stosować to podejście).
- By sformalizować ten problem najczęściej używany jest błąd średnio-kwadratowy (ang. mean square error (MSE)):

$$\arg\max_{\boldsymbol{w} \in \mathbb{R}^m} \mathcal{L}(\boldsymbol{w}) = \frac{1}{2n} \sum_{i=1}^{n} (\overbrace{\boldsymbol{x}_i^T \boldsymbol{w}}^{h(\boldsymbol{x}_i)} - y_i)^2 = \mathbb{E}[(X\boldsymbol{w} - y)^2]$$

Regresja liniowa — rozwiązanie analitycznie

$$\nabla \mathcal{L}(\boldsymbol{w}^*) = \nabla \frac{1}{2n} \sum_{i=1}^{n} (\boldsymbol{x}_i^T \boldsymbol{w}^* - y_i)^2 = 0$$

$$\frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i^T \boldsymbol{w}^* - y_i) \boldsymbol{x}_i = 0$$

$$\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^T \boldsymbol{w}^* = \sum_{i=1}^{n} \boldsymbol{x}_i y_i$$

$$\boldsymbol{w}^* = \Big(\sum_{i=1}^{n} \overbrace{\boldsymbol{x}_i \boldsymbol{x}_i^T}^{\text{macierz m}\times\text{m}} \Big)^{-1} \sum_{i=1}^{n} \boldsymbol{x}_i y_i \quad = (X^T X)^{-1} \sum_{i=1}^{n} \boldsymbol{x}_i y_i$$

Metoda Spadku Wzdłuż Gradientu

- Niestety, wiele problemów optymalizacji nie posiada (lub nie zostały jeszcze znalezione) rozwiązań analitycznych.
- Jedną z najpopularniejszych metod na znalezieni $\arg\max_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}); f : \mathbb{R}^n \to \mathbb{R}$ dla problemów ciągłych jest Metoda Spadku Wzdłuż Gradientu (ang. Gradient Descent (GD)).
- W metodzie GD, w każdym kroku poruszamy się w kierunku wskazywanym przez gradient.
- Wielkość o jaką się poruszamy jest regulowana przez wartość nazywaną wielkością kroku (ang. step-size, albo learning rate) i jest oznaczana często jako α albo η.

$$x_{t+1} = x_t - \alpha \nabla f(x_t),\ \alpha \in \mathbb{R}^+$$

- W najprostszej postaci α jest stała dla każdej współrzędnej. W różnych rozszerzeniach/wariantach metody GD (np. AdaGrad, Adam, Momentum) różne wartości są używane dla różnych współrzędnych.

Metoda Spadku Wzdłuż Gradientu

Algorytm 1: (Batch) Gradient Descent

Argumenty:
Zbiór danych $X, \boldsymbol{y}$ o długości n
Parametry (wagi) W
Funkcja straty (błędu) $\mathcal{L}(y_i, h(\boldsymbol{x}_i, W)) \in \mathbb{R}$
Ilość iteracji $T \in \mathbb{N}$
Zbiór malejących wartości wielkości kroku: $\{\alpha_0, \alpha_1 \dots \alpha_{T-1}\}$
for $t \leftarrow 0$ **to** $T - 1$ **do**
    $g_t \leftarrow 0$
    for $i \leftarrow 0$ **to** $n - 1$ **do**
        $g_t \leftarrow g_t + \frac{1}{n} \nabla \mathcal{L}(y_i, h(\boldsymbol{x}_i, W))$ (względem W)
    $W \leftarrow W - \alpha_t g_t$

Stochastyczny spadek wzdłuż gradientu

- Jedno z najczęściej stosowanych odmian GD jest Stochastyczny Spadek Wzdłuż Gradientu (ang. Stochastic Gradient Descent (SGD))
- Zamiast oblicza $\nabla\mathcal{L}$ dla wszystkich punktów w danych, w danym kroku oblicza $\nabla\mathcal{L}$ tylko dla pojedynczej pary $\boldsymbol{x}_i, y_i$.
- Obliczenie obserwacji na podstawie jednej obserwacji jest prostsze, szybsze, nie wymaga dostępu do całej macierzy X w momencie liczenia gradientu.
- Zazwyczaj proces optymalizacji podzielony jest na epoki, w jednej epoce dokonuje się jednej aktualizacji dla każdego punktu w danych.

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha\nabla(\boldsymbol{x}_i^T\boldsymbol{w}_t - y_i)^2 \\ &= \boldsymbol{w}_t - 2\alpha(\boldsymbol{x}_i^T\boldsymbol{w}_t - y_i)\boldsymbol{x}_i\end{aligned}$$

Stochastyczny spadek wzdłuż gradientu

Algorytm 2: Mini-batch Stochastic Gradient Descent

Argumenty:
Zbiór danych $X, \boldsymbol{y}$ o długości n
Parametry (wagi) W
Funkcja straty (błędu) $\mathcal{L}(y_i, h(\boldsymbol{x}_i, W)) \in \mathbb{R}$
Ilość iteracji $T \in \mathbb{N}$
Zbiór malejących wartości wielkości kroku: $\{\alpha_0, \alpha_1 \dots \alpha_{T-1}\}$
Wielkość mini-batcha $b \in \mathbb{N}$
for $t \leftarrow 1$ **to** T **do**
    Pomieszaj kolejność wierszy w $X, \boldsymbol{y}$
    for $b_i \leftarrow 1$ **to** $\lceil \frac{n}{b} \rceil$ **do**
        $g_{tb_i} \leftarrow 0$
        for $j \leftarrow b_i \cdot b$ **to** $\max((b_i + 1) \cdot b, n) - 1$ **do**
            $g_{tb_i} \leftarrow \frac{1}{n} \nabla \mathcal{L}(y_i, h(\boldsymbol{x}_i, W))$ (względem W)
        $W \leftarrow W - \alpha_t g_{tb_i}$

Sieci neuronowe i propagacja wsteczna

- Sieci neuronowe to po prostu złożenie wielu funkcji, jak np:

$$
\begin{aligned}
f^1(\boldsymbol{x}) &= \sigma^1(\boldsymbol{x}W^1) \\
f^2(\boldsymbol{x}) &= \sigma^2(\boldsymbol{x}W^2) \\
&\dots \\
f^L(\boldsymbol{x}) &= \sigma^L(\boldsymbol{x}W^L) \\
h(\boldsymbol{x}) &= f^L(f^{L-1}(\dots f^2(f^1(\boldsymbol{x}))\dots))
\end{aligned}
$$

- Każdą z takich funkcji nazywamy zazwyczaj warstwą sieci neuronowej, a σ funkcją **aktywacji** (ang. activation function), która jest nieliniowa.
- Możemy policzyć dla każdej pary $\boldsymbol{x}_i, y_i$ składową gradientu $\nabla\mathcal{L}(y_i, h(\boldsymbol{x}_i))$ – pochodną cząstkową dla każdej wagi $\frac{\partial\mathcal{L}}{W^l_{jk}}$, dzięki znajomości pochodnych funkcji złożonej.

Sieci neuronowe i propagacja wsteczna

- Oznaczmy $\boldsymbol{z}^l = W^l \boldsymbol{a}^{l-1}$ i $\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$.
- Teraz wyznaczmy pochodne cząstkowe względem W^{1-L}:

$$\frac{\partial \mathcal{L}}{\partial W^1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^L} \cdot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \cdot \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \cdot \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \cdot \ldots \cdot \frac{\partial \boldsymbol{z}^3}{\partial \boldsymbol{a}^2} \cdot \frac{\partial \boldsymbol{a}^2}{\partial \boldsymbol{z}^2} \cdot \frac{\partial \boldsymbol{z}^2}{\partial \boldsymbol{a}^1} \cdot \frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{z}^1} \cdot \frac{\partial \boldsymbol{z}^1}{\partial W^1}$$

$$\frac{\partial \mathcal{L}}{\partial W^2} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^L} \cdot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \cdot \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \cdot \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \cdot \ldots \cdot \frac{\partial \boldsymbol{z}^3}{\partial \boldsymbol{a}^2} \cdot \frac{\partial \boldsymbol{a}^2}{\partial \boldsymbol{z}^2} \cdot \frac{\partial \boldsymbol{z}^2}{\partial W^2}$$

$$\ldots = \ldots$$

$$\frac{\partial \mathcal{L}}{\partial W^{L-2}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^L} \cdot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \cdot \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \cdot \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \cdot \frac{\partial \boldsymbol{z}^{L-1}}{\partial \boldsymbol{a}^{L-2}} \cdot \frac{\partial \boldsymbol{a}^{L-2}}{\partial \boldsymbol{z}^{L-2}} \cdot \frac{\partial \boldsymbol{a}^{L-2}}{\partial W^{L-1}}$$

$$\frac{\partial \mathcal{L}}{\partial W^{L-1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^L} \cdot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \cdot \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \cdot \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \cdot \frac{\partial \boldsymbol{z}^{L-1}}{\partial W^{L-1}}$$

$$\frac{\partial \mathcal{L}}{\partial W^L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^L} \cdot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \cdot \frac{\partial \boldsymbol{z}^L}{\partial W^L}$$

Sieci neuronowe i propagacja wsteczna

- Liczenie gradientu bezpośrednio w taki sposób jest nieefektywne!
- Zauważmy, że do wyliczenia pochodnej cząstkowej dla wag w każdej warstwie, używamy tych samych pochodnych cząstkowych.
- Możemy więc wyliczyć pochodną cząstkową $\mathcal{L}$ w kolejnych warstwach rekurencyjnie:

$$\frac{\partial \mathcal{L}}{\partial W^l} = \overbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^L} \cdot \frac{\partial \boldsymbol{a}^L}{\partial \boldsymbol{z}^L} \cdot \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{a}^{L-1}} \cdot \frac{\partial \boldsymbol{a}^{L-1}}{\partial \boldsymbol{z}^{L-1}} \cdot \ldots \cdot \frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{a}^l}}^{\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^l}} \cdot \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{z}^l} \cdot \frac{\partial \boldsymbol{z}^l}{\partial W^l}$$

$$\frac{\partial \mathcal{L}}{\partial W^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^l} \cdot \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{z}^l} \cdot \frac{\partial \boldsymbol{z}^l}{\partial W^l},$$

gdzie:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{l+1}} \cdot \frac{\partial \boldsymbol{a}^{l+1}}{\partial \boldsymbol{z}^{l+1}} \cdot \frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{a}^l}$$

Taki sposób liczenia gradientów w sieciach neuronowych nazywamy **propagacją wsteczną** (ang. backpropagation).

Plan wykładu

Metody przybliżone

- W większości praktycznych problemów przestrzeń stanów jest ogromna:
  - problem z pamięcią (zapamiętanie wartości dla każdego stanu)
  - problem z czasem (większość stanów nigdy nie będzie w praktyce odwiedzona nawet jeden raz!)
- Potrzeba generalizacji wiedzy/doświadczenia ze stanów w pewien sposób podobnych.
- Aproksymacja funkcji — metody uczenia nadzorowanego

Aproksymacja funkcji stanu

- Będziemy przybliżać funkcję stanu v_π dla znanej polityki π.
- Zamiast wersji tablicowej (tak jak na wcześniejszych wykładach tj. zapamiętywanie wartości dla każdego stanu z osobna) będziemy używali funkcji parametryzowanej wektorem wag $\boldsymbol{w} \in \mathbb{R}^d$:

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$$

  gdzie $d \ll |\mathcal{S}|$
- Funkcja $\hat{v}$ może mieć formę:
  - funkcji liniowej,
  - sieci neuronowej,
  - drzewa decyzyjnego,
  - itd.
- Aktualizacja wartości dla jednego stanu powoduje zmianę dla wielu (potencjalnie wszystkich) stanów

Aproksymacja funkcji stanu

- Aktualizacja wartości stanu $s \mapsto u$ (aktualizacja wartości stanu s w kierunku wartości u)
  - Monte Carlo: $S_t \mapsto G_t$
  - TD(0): $S_t \mapsto R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t)$
  - DP: $s \mapsto \mathbb{E}_\pi \left[R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t) | S_t = s \right]$
- W podejściu tablicowym aktualizacja wartości pojedynczego stanu o zadany krok w danym kierunku.
- Aproksymacja funkcji – uczenie nadzorowane:
  - zbiór uczący to zbiór par: (stan, oczekiwana wartość stanu)
  - każda wykonywana aktualizacja $s \mapsto u$ jest przykładem uczącym
  - dowolna metoda ale mamy problem niestacjonarny (cel może zmieniać się w czasie)

Funkcja celu

- Mamy znacznie więcej stanów niż wag — oznacza to, że poprawienie estymaty dla jednego stanu będzie powodowało pogorszenie estymat dla innych
- Błąd kwadratowy:
$$\overline{VE}(\boldsymbol{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left[\hat{v}(s, \boldsymbol{w}) - v_\pi(s)\right]^2$$
$\mu(s)$ — rozkład określa jak bardzo zależy nam na danym stanie s: $\mu(s) \geqslant 0$, $\sum_s \mu(s) = 1$
- Będziemy zakładali, że $\mu(s)$ jest proporcjonalny do czasu jaki agent spędza w danym stanie (ang. on-policy distribution)
- Chcielibyśmy znaleźć optymalne wagi $\mathbf{w}^*$ takie że $\overline{VE}(\mathbf{w}^*) \leqslant \overline{VE}(\boldsymbol{w})$ dla wszystkich $\boldsymbol{w}$.

Stochastic Gradient Descent (SGD)

Metoda stochastycznego spadku wzdłuż gradientu — ang. Stochastic Gradient Descent (SGD)

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \tfrac{1}{2}\alpha\nabla\left[\hat{v}(S_t,\boldsymbol{w}_t) - v_\pi(S_t)\right]^2 \\ &= \boldsymbol{w}_t - \alpha\left[\hat{v}(S_t,\boldsymbol{w}_t) - v_\pi(S_t)\right]\nabla\hat{v}(S_t,\boldsymbol{w}_t) \end{aligned}$$

gdzie $\nabla f(\boldsymbol{w})$ to wektor pochodnych cząstkowych, α — rozmiar kroku (prędkość uczenia)

- metoda spadku wzdłuż gradientu: aktualizacja proporcjonalna do negatywnego gradientu funkcji celu
- stochastyczna: aktualizacja tylko po jednym przykładzie (ale po wielu minimalizujemy średnią wartość celu)

Stochastic Gradient Descent (SGD)

Wróćmy do przypadku gdy nie znamy $v_\pi(s)$.

- U_t — wartość docelowa t-go przykładu uczącego $S_t \mapsto U_t$ czyli potencjalnie losowe przybliżenie $v_\pi(S_t)$
- U_t może być np.
  - zaszumioną wersją $v_\pi(s)$
  - wartością docelową zależną od $\hat{v}$ (bootstrapping)

Ogólny wzór SGD:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \left[\hat{v}(S_t, \boldsymbol{w}_t) - U_t\right] \nabla \hat{v}(S_t, \boldsymbol{w}_t)$$

- Jeśli U_t jest nieobciążoną estymatą (czyli $\mathbb{E}[U_t|S_t = s] = v_\pi(s)$ dla każdego t) to mamy gwarancję zbieżności do optimum lokalnego przy malejącym α.

Tabularyczny TD(0)

Algorytm 3: Ewaluacja polityki za pomocą algorytmu TD(0)

Argumenty: Dana dowolna polityka π, $\alpha > 0$
Inicjalizacja:
Dowolnie zainicjalizowane $V(s)$ dla wszystkich $s \in \mathcal{S}$
$V(s) \leftarrow 0$ jeśli s jest stanem terminalnym.
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*;

TD(0) z aproksymacją

Algorytm 4: 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*;

Aproksymator liniowy

- $\hat{v}(s, \boldsymbol{w})$ — liniowa funkcja (iloczyn skalarny):

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

  gdzie $\boldsymbol{x}(s)$ to wektor cech opisujący stan s (liczba wag musi być taka sama)

- Gradient takiego przybliżenia ze względu na wagi wynosi:

$$\nabla \hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)$$

- Aktualizacja wag:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left[U_t - \hat{v}(S_t, \boldsymbol{w}_t) \right] \boldsymbol{x}(S_t)$$

||||||||||||||||||||

Aproksymator liniowy — przykład

- plansza 10×10, start w punkcie $(0, 0)$, celem dojście do punktu $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$

Pytanie

Jeśli aktualne wagi $\boldsymbol{w}_t^T = [0.5, 0.2, 0.3]$, to ile wynosi $\hat{v}(s, \boldsymbol{w}_t)$ dla stanu $x = 2$, $y = 3$?

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

Aproksymator liniowy — przykład

- plansza 10×10, start w punkcie $(0, 0)$, celem dojście do punktu $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$

Pytanie

Jeśli aktualne wagi $\boldsymbol{w}_t^T = [0.5, 0.2, 0.3]$, to ile wynosi $\hat{v}(s, \boldsymbol{w}_t)$ dla stanu $x = 2$, $y = 3$?

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

$\hat{v}(s, \boldsymbol{w}_t) = 0.5 \cdot 2 + 0.2 \cdot 3 + 0.3 \cdot 1 = 1.9$

Aproksymator liniowy — przykład

- plansza 10×10, start w punkcie $(0, 0)$, celem dojście do punktu $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$
- aktualne wagi: $\boldsymbol{w}_t^T = [0.5, 0.2, 0.3]$
- $x = 2$, $y = 3$, $\hat{v}(s, \boldsymbol{w}_t) = 1.9$

Pytanie

Jeśli wartość docelowa przykładu wynosi $U_t = 2$ to jak zmienią się wagi dla $\alpha = 0.1$?

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left[U_t - \hat{v}(S_t, \boldsymbol{w}_t) \right] \boldsymbol{x}(S_t)$$

Aproksymator liniowy — przykład

- plansza 10×10, start w punkcie $(0, 0)$, celem dojście do punktu $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$
- aktualne wagi: $\boldsymbol{w}_t^T = [0.5, 0.2, 0.3]$
- $x = 2$, $y = 3$, $\hat{v}(s, \boldsymbol{w}_t) = 1.9$

Pytanie

Jeśli wartość docelowa przykładu wynosi $U_t = 2$ to jak zmienią się wagi dla $\alpha = 0.1$?

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left[U_t - \hat{v}(S_t, \boldsymbol{w}_t) \right] \boldsymbol{x}(S_t)$$

$$\boldsymbol{w}_{t+1} = \begin{bmatrix} 0.5 + 0.1 \cdot (2 - 1.9) \cdot 2 \\ 0.2 + 0.1 \cdot (2 - 1.9) \cdot 3 \\ 0.3 + 0.1 \cdot (2 - 1.9) \cdot 1 \end{bmatrix} = [0.52, 0.23, 0.31]^T$$

Aproksymator liniowy — przykład

- plansza 10×10, start w punkcie $(0, 0)$, celem dojście do punktu $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$
- aktualne wagi: $\boldsymbol{w}_t^T = [0.5, 0.2, 0.3]$
- $x = 2$, $y = 3$, $\hat{v}(s, \boldsymbol{w}_t) = 1.9, \boldsymbol{w}_{t+1} = [0.52, 0.23, 0.31]^T$

Pytanie

Ile wynosi $\hat{v}(s, \boldsymbol{w}_{t+1})$ dla stanu $x = 2$, $y = 3$?

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

Aproksymator liniowy — przykład

- plansza 10×10, start w punkcie $(0, 0)$, celem dojście do punktu $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$
- aktualne wagi: $\boldsymbol{w}_t^T = [0.5, 0.2, 0.3]$
- $x = 2$, $y = 3$, $\hat{v}(s, \boldsymbol{w}_t) = 1.9, \boldsymbol{w}_{t+1} = [0.52, 0.23, 0.31]^T$

Pytanie

Ile wynosi $\hat{v}(s, \boldsymbol{w}_{t+1})$ dla stanu $x = 2$, $y = 3$?

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

$\hat{v}(s, \boldsymbol{w}_{t+1}) = 0.52 \cdot 2 + 0.23 \cdot 3 + 0.31 \cdot 1 = 2.04$

Uwaga: „przeskoczyliśmy" docelową wartość przykładu $U_t = 2$ — zbyt duża wartość α

Generalizacja

- W poprzednim przykładzie aktualizacja wag spowodowała zmianę wartości dla stanu $x = 2$, $y = 3$
- Ale zmieniła się również wartość dla innych stanów, np. $x = 4$, $y = 1$
- Agent z aproksymatorem może generalizować i w konsekwencji uczyć się szybciej
  - gra w szachy: ok. 10^{47} liczby stanów
  - gra Go: ok. 10^{170} liczby stanów
  - gra w tryktrak/backgammon (ok. 10^{20} stanów): odwiedzenie tylko co 10^{12} stanu umożliwiło nauczenie „mistrzowskiego" agenta

Opis stanu — cechy

- Cechy opisujące stan środowiska mają bardzo istotny wpływ na skuteczne działanie RL
- Odpowiedni wybór cech (spośród dostępnych oryginalnie lub ich pochodnych) — sposób na dodanie wiedzy dziedzinowej

Przykłady:

- nawigacja robotem: jego pozycja, stan baterii, odczyt z czujników (detektor przeszkód), itp.
- balansowanie tyczką: pozycja, kąt nachylenia, prędkość kątowa, itp.
- gra w szachy: liczba pionów, liczba figur w centrum, czy mamy hetmana/królową, itd.

Opis stanu — aproksymator liniowy

Aproksymator liniowy:

- gwarancja zbieżności
- wydajny (zarówno pamięciowo jak i obliczeniowo)
- wada: nie potrafi uwzględniać interakcji pomiędzy cechami
  - np. cecha a jest korzystna tylko przy jednoczesnym braku cechy b
  - balansowanie tyczką: duża prędkość kątowa (a) i kąt nachylenia (b) — gdy blisko pionu duża prędkość kątowa może oznaczać ryzyko upadku/wybicia z równowagi (niekorzystne), ale przy małym kącie może oznaczać powrót do pionu (korzystne)
- potrzebujemy nowych cech łączących cechy bazowe

Opis stanu — przykład

Zadanie:

- plansza 10×10
- nagroda za dojście do $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$

Aproksymator liniowy działa dobrze.

Pytanie

A co się stanie gdy nagroda będzie za dojście do punktu $(5, 5)$?

Opis stanu — przykład

Zadanie:

- plansza 10×10
- nagroda za dojście do $(10, 10)$
- cechy: $\boldsymbol{x}(s) = [x, y, 1]^T$

Aproksymator liniowy działa dobrze.

Pytanie

A co się stanie gdy nagroda będzie za dojście do punktu $(5, 5)$?

Liniowy aproksymator nie będzie w stanie poprawnie oddać funkcji wartości stanu.
Potrzebna jest nowa cecha, np.: $\sqrt{(x-5)^2 + (y-5)^2}$

Tworzenie cech — wielomiany

- Załóżmy, że stan jest dwuwymiarowy, reprezentowany przez dwie wartości: s_1 i s_2 (są to wartości które oryginalnie zwraca nam środowisko)
- Możemy zdecydować, że reprezentujemy stan jako $\boldsymbol{x}(s) = [s_1, s_2]^T$
- Problemy:
  - jak reprezentować zależności między tymi wymiarami?
  - dla $s_1 = 0$ i $s_2 = 0$ wartość stanu wg. aproksymatora będzie równa zeru.
- Rozwiązanie, np.:
  - $\boldsymbol{x}(s) = [1, s_1, s_2, s_1 s_2]^T$
  - $\boldsymbol{x}(s) = [1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2]^T$

Tworzenie cech — wielomiany

- Można uogólnić na wielomiany wyższego rzędu i dla dowolnej liczby wymiarów.
- Wielomiany wyższego rzędu — umożliwiają dokładniejsze przybliżenie bardziej skomplikowanych funkcji

Uwaga

Aproksymator jest nadal liniowy pomimo, że cechy nie są liniowe.

Tworzenie cech — inne podejścia

- cechy bazujące na szeregach Fouriera

$$x_i(s) = \cos(\pi \mathbf{s}^T \mathbf{c}^\mathbf{i})$$

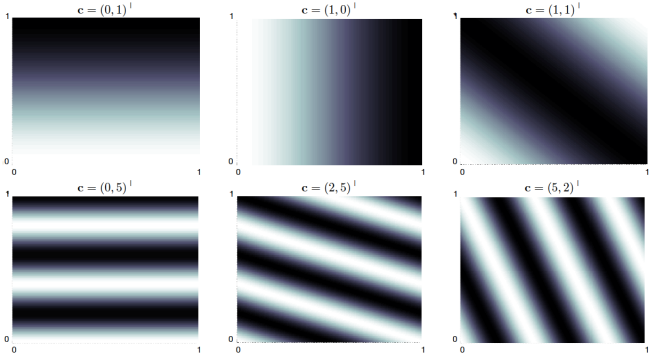


Figure 9.4: A selection of six two-dimensional Fourier cosine features, each labeled by the vector $\mathbf{c}^i$ that defines it (s_1 is the horizontal axis, and $\mathbf{c}^i$ is shown with the index i omitted). After Konidaris et al. (2011).

Tworzenie cech — inne podejścia

- kodowanie zgrubne (ang. Coarse Coding)
- dzielimy oryginalną przestrzeń na (pokrywające się) obszary (np. koła, elipsy, itp.) reprezentujące nowe cechy
  - jeśli s leży wewnątrz koła to dana cecha ma wartość 1, jeśli nie to 0

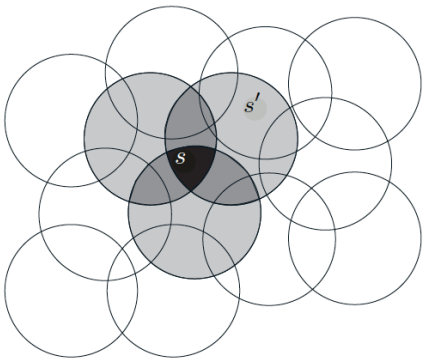


Figure 9.6: Coarse coding. Generalization from state s to state s' depends on the number of their features whose receptive fields (in this case, circles) overlap. These states have one feature in common, so there will be slight generalization between them.

Tworzenie cech — inne podejścia

- kodowanie kafelkowe (ang. Tile Coding)
  - wydajny wariant Coarse Coding
  - podział przestrzeni np. na kwadraty
  - aby kafelki się pokrywały tworzy się przesunięte (offset ułamkiem szerokości kafelka) podziały
  - możliwe różne nieregularne podziały przestrzeni

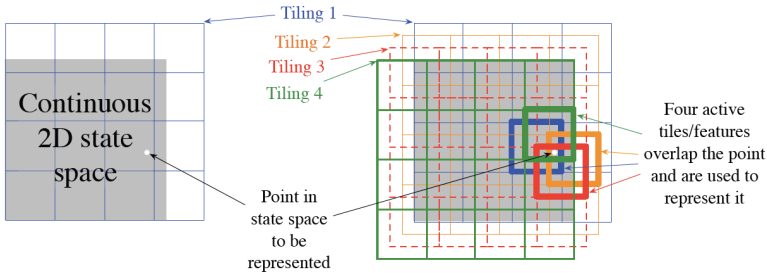


Figure 9.9: Multiple, overlapping grid-tilings on a limited two-dimensional space. These tilings are offset from one another by a uniform amount in each dimension.

Tworzenie cech — inne podejścia

- radialne funkcje bazowe (ang. Radial Baisis Functions)
- uogólnienie na wartości ciągłe z przedziału $[0, 1]$
  - np. funkcja gaussowska:

$$x_i(s) = \exp(-\frac{\|s - c_i\|^2}{2\sigma_i^2})$$

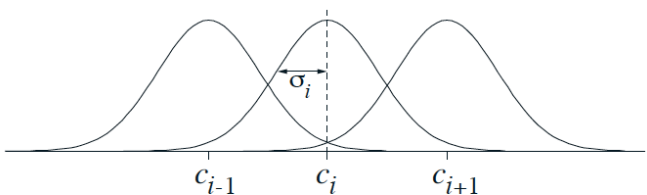


Figure 9.13: One-dimensional radial basis functions.

Aproksymator — sztuczne sieci neuronowe

- W ogólności funkcja liniowa ma ograniczoną siłę ekspresji.
- 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.
- Powszechnie stosuje się w tym celu sieci neuronowe, które same potrafią nauczyć się użytecznych cech.
- Podejście, w którym stosuje głębokie sieci neuronowe, nazywamy głębokim uczeniem ze wzmocnieniem (ang. Deep Reinforcement Learning).

SARSA z aproksymacją

Wejście:

- różniczkowalna funkcja $\hat{q}(s, a, \boldsymbol{w})$
- rozmiar kroku α

1. Zainicjalizuj wektor wag $\boldsymbol{w} = \mathbf{0}$
2. Powtarzaj dla każdego epizodu:
   S — stan początkowy
   A — początkowa akcja (używając np. metody eksploracji ϵ-zachłannej)
   Dla każdego kroku w epizodzie:
   1. wykonaj akcję A, zaobserwuj nagrodę R i kolejny stan S'
   2. jeśli S' jest stanem terminalnym:
      $$\boldsymbol{w} = \boldsymbol{w} + \alpha \left[R - \hat{q}(S, A, \boldsymbol{w})\right] \nabla \hat{q}(S, A, \boldsymbol{w})$$
      idź do kolejnego epizodu
   3. wybierz akcję A' ze stanu S' używając polityki wyprowadzonej z $\hat{q}(s, a, \boldsymbol{w})$ (używając np. metody eksploracji ϵ-zachłannej)
   4. $\boldsymbol{w} = \boldsymbol{w} + \alpha \left[R + \gamma \hat{q}(S', A', \boldsymbol{w}) - \hat{q}(S, A, \boldsymbol{w})\right] \nabla \hat{q}(S, A, \boldsymbol{w})$
   5. $S = S'$ oraz $A = A'$

SARSA

Algorytm 5: Pseudokod dla algorytmu SARSA

Inicjalizacja:
Dowolnie zainicjalizowane $Q(s, a)$ dla wszystkich $s \in \mathcal{S}, a \in \mathcal{A}$
$Q(s, \cdot) \leftarrow 0$ jeśli s jest stanem terminalnym
Dana dowolna polityka π (np. ϵ-greedy) zgodna z początkowym Q
repeat
    if S *nieustawiony lub terminalny* **then**
        Rozpocznij nowy epizod
        $S \leftarrow S_0$
        $A \leftarrow \pi(S)$
    Wykonaj akcję A, zaobserwuj nagrodę R i następnik S'
    $A' \leftarrow \pi(S')$
    $Q(S, A) \leftarrow Q(S, A) - \alpha(Q(S, A) - (R + \gamma Q(S', A'))$
    Zaktualizuj π zgodnie z Q
    $S \leftarrow S'$
    $A \leftarrow A'$
until *WarunekStopu*;

SARSA z aproksymacją

Algorytm 6: Pseudokod dla algorytmu SARSA z aproksymacją

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
        $S \leftarrow S_0$
        $A \leftarrow \pi(S)$
    Wykonaj akcję A, zaobserwuj nagrodę R i następnik S'
    $A' \leftarrow \pi(S')$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left[\hat{q}(S, A, \boldsymbol{w}) - (R + \gamma \hat{q}(S', A', \boldsymbol{w}))\right] \nabla \hat{q}(S, A, \boldsymbol{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$
until *WarunekStopu*;

Expected SARSA

SARSA

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \\ &\alpha \left[\hat{q}(S_t, A_t, \boldsymbol{w}_t) - (R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}_t))\right] \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)\end{aligned}$$

Expected SARSA

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \\ &\alpha \left[\hat{q}(S_t, A_t, \boldsymbol{w}_t) - (R_{t+1} + \gamma \textstyle\sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \boldsymbol{w}_t))\right] \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)\end{aligned}$$

Q-Learning

Algorytm 7: Pseudokod dla algorytmu Q-Learning

Inicjalizacja:
Dowolnie zainicjalizowane $Q(s, a)$ dla wszystkich $s \in \mathcal{S}, a \in \mathcal{A}$
$Q(s, \cdot) \leftarrow 0$ jeśli s jest stanem terminalnym
Dana dowolna polityka π zgodna z początkowym Q

repeat
    if S *nieustawiony lub terminalny* **then**
        Rozpocznij nowy epizod
        $S \leftarrow S_0$
    $A \leftarrow \pi(S)$
    Wykonaj akcję A, zaobserwuj nagrodę R i następnik S'
    $Q(S, A) \leftarrow Q(S, A) - \alpha(Q(S, A) - (R + \gamma \max_{a \in \mathcal{A}} Q(S', a))$
    Zaktualizuj π zgodnie z Q
    $S \leftarrow S'$
until *warunek stopu*;

Q-Learning z aproksymacją

Algorytm 8: Pseudokod dla algorytmu Q-Learning z Parametryczną Aproksymacją

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*;

Q-Learning z pamięcią

Algorytm 9: Pseudokod dla algorytmu Q-Learning z pamięcią

Inicjalizacja:
Różniczkowalna funkcja $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, wagi $\boldsymbol{w} \in \mathbb{R}^d$, polityka π zgodna z $\boldsymbol{w}$
Zainicjalizuj pamięć $\mathcal{M} = \emptyset$, parametr I (ilość kroków uczenia się z pamięci), parametr J (ilość doświadczeń wybieranych 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*;

Plan wykładu

Częściowe podsumowanie

Omówiliśmy kilka różnych, częściowo niezależnych aspektów:

- bezpośrednia aktualizacja wartości stanu vs użycie aproksymatorów
- sposób wyznaczania wartości stanu (DP, MC, TD)
- metody on-policy vs off-policy

Funkcja tablicowa vs aproksymator

Funkcja tablicowa:

- uczymy/poprawiany bezpośrednio estymaty wartości stanu konkretnych stanów
- musimy pamiętać wartość dla każdego stanu
- musimy aktualizować każdą wartość dla każdego możliwego stanu
- stan możemy utożsamiać z jego identyfikatorem

Aproksymator:

- uczymy/poprawiamy parametry aproksymatora który przybliża wartość zadanego stanu
- generalizacja: zmiana parametrów „pod jeden stan" aktualizuje wartości aproksymatora dla innych stanów
- musimy mieć dostęp do cech opisujących stan (argumenty aproksymatora)

Sposób wyznaczania wartości stanu

- Programowanie Dynamiczne (DP):
  - wartość stanu wyznaczana z równania Bellmana
  - „globalne" spojrzenie: jedna wartość zależy od wszystkich pozostałych oraz modelu świata
- Monte Carlo (MC):
  - spodziewany zysk ze stanu znany dopiero na koniec epizodu
  - zysk to zdyskontowana suma nagród od stanu **do końca epizodu**
- Uczenie Różnicowe (TD):
  - nowe oszacowanie zysku ze stanu po wykonaniu **pojedynczego kroku**
  - spodziewany zysk to bezpośrednia nagroda plus zdyskontowana wartość przyszłości (najbliższa przyszłość znana – nowy stan; jej wartość – dotychczasowe oszacowanie wartości tego stanu)

on-policy vs off-policy

Metody on-policy:

- uczą się funkcji wartości dla polityki która generuje obecne epizody (interakcję ze środowiskiem),
- np. SARSA uczy się funkcji wartości bazując na wartości kolejnego stanu oraz **akcji wynikającej z bieżącej polityki**
- zakłada, że bieżąca polityka będzie kontynuowana

Metody off-policy:

- uczy się funkcji wartości dla optymalnej polityki, niezależnie od tego jaka generuje obecne epizody (interakcję ze środowiskiem)
- np. Q-learning uczy się funkcji wartości bazując na wartości kolejnego stanu oraz zachłannej akcji wynikającej z bieżącej polityki
- zakłada, że zachłanna polityka będzie kontynuowana