

# Temporal Difference Learning

# Temporal Difference Learning

- Równania Bellmana można rozwiązać analitycznie.
- Złożoność obliczeniowa sięga  $O(n^3)$  gdzie  $n$  oznacza liczbę stanów.
- W przypadku rozwiązywania dużych problemów istnieje wiele metod iteracyjnych:
  - Exhaustive search
  - Programowanie dynamiczne
  - Metody Monte Carlo
  - Temporal Difference Learning

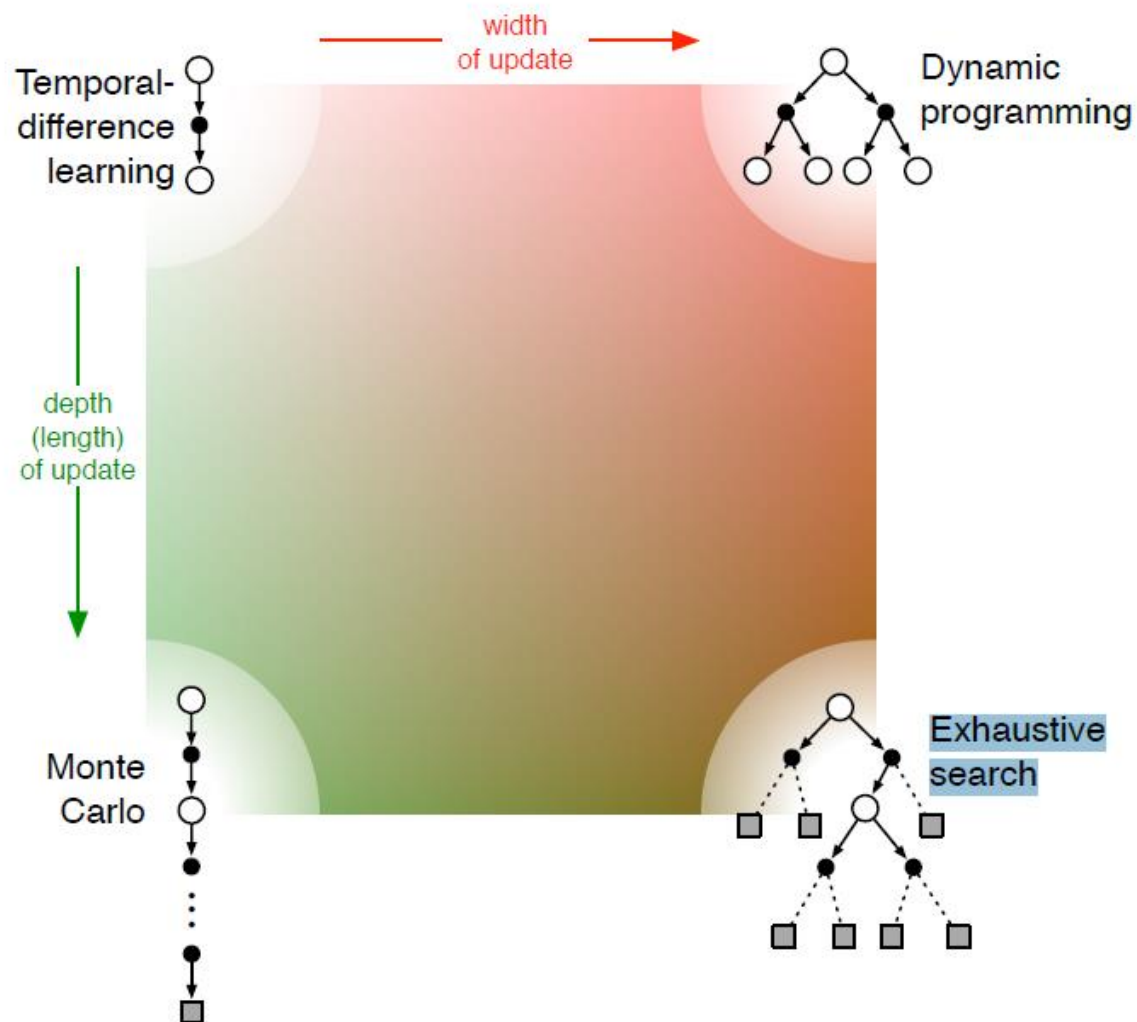
# Temporal Difference Learning

- Temporal difference learning (co możemy przetłumaczyć na uczenie oparte na różnicach czasowych) jest grupą algorytmów, które w pewnym sensie łączy ze sobą sposób działania metod Monte Carlo i podejścia związanego z dynamicznym programowaniem.

# Temporal Difference Learning

- Temporal difference learning (co możemy przetłumaczyć na uczenie oparte na różnicach czasowych) jest grupą algorytmów, które w pewnym sensie łączy ze sobą sposób działania metod Monte Carlo i podejścia związanego z dynamicznym programowaniem.
- Metody TD polegają na uczeniu się zarówno na podstawie **doświadczenia**, jak i **oczekiwań**.

# Temporal Difference Learning



Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

# Temporal Difference Learning

- Najprostszy algorytm ( $TD(0)$ ):

$$V_{\tau}(s_t) = V_{\tau-1}(s_t) + \alpha(r_{t+1} + \beta V_{\tau-1}(s_{t+1}) - V_{\tau-1}(s_t))$$

# TD vs. MC

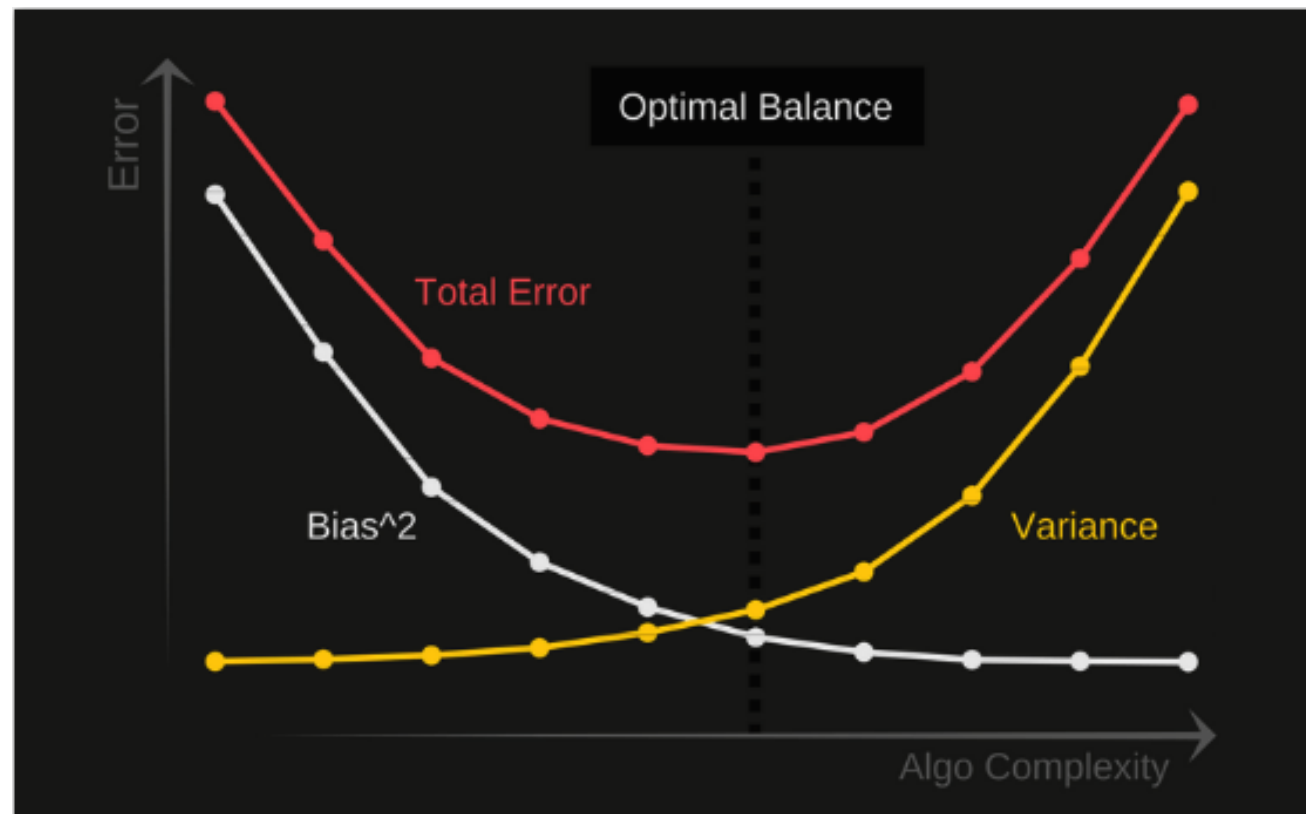
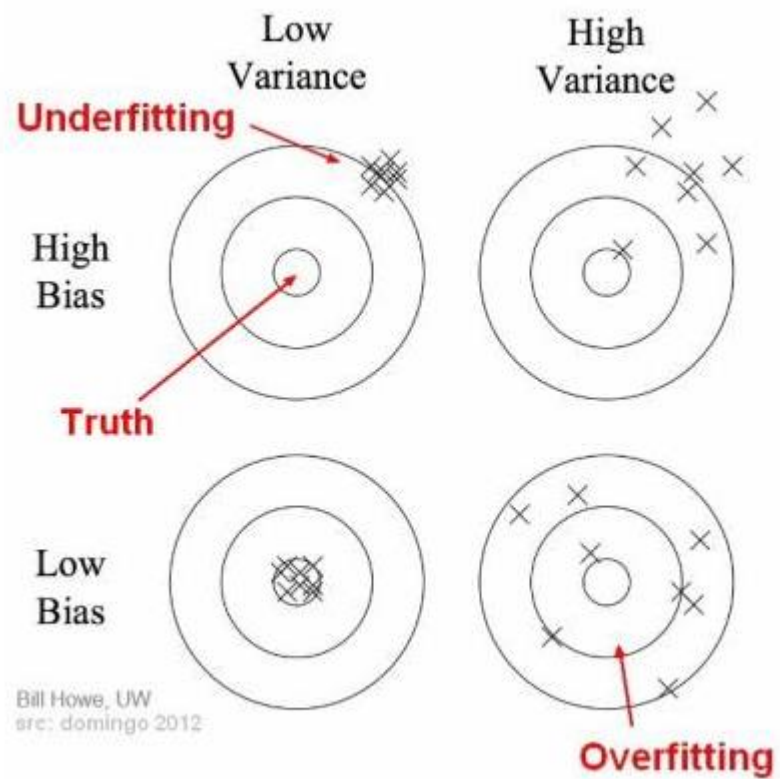
## Temporal difference learning:

- Uczy się w trakcie trwania epizodu.
  - Oznacza to, że metoda może się uczyć nawet w przypadku problemów z nieskończonym horyzontem czasowym
- Jest obciążonym estymatorem, posiada jednak niską wariancję .
- Dużo wrażliwszy na stan początkowy.
- Zbiega do estymatora największej wiarygodności (MLE) procesu decyzyjnego Markowa opisującego problem:
  - Estymata  $(S, A, \hat{P}, \hat{R}, \beta)$  najlepiej dopasowana do doświadczenia zbieranego w trakcie uczenia się
- Efektywnie wykorzystuje własność Markowa.

## Metody Monte Carlo:

- Uaktualnianie dzieje się po skończeniu epizodu.
- Ma bardzo wysoką wariancję, zerowe obciążenie.
- Niewrażliwy na stan początkowy,
- Zbiega do rozwiązania z najmniejszym błędem średniokwadratowym:
  - Dopasowuje się do obserwowanych w trakcie epizodów nagród: 
$$\text{MSE} = \sum_{k=1}^n \sum_{t=1}^{T_k} (R_t^k - V(S_t^k))^2$$
- Nie opiera się na własności Markowa, dzięki czemu możliwe jest rozwiązywanie problemów nie będących procesami decyzyjnymi Markowa.

# Bias-Variance Tradeoff



Źródło: <https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229>

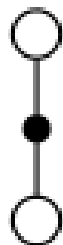


# Temporal Difference Learning

- Zdefiniowaliśmy algorytm TD dla najprostszego przypadku, gdy agent rozpatruje jedynie jeden okres w przód ( $TD(0)$ ):

$$V_{\tau}(s_t) = V_{\tau-1}(s_t) + \alpha(r_{t+1} + \beta V_{\tau-1}(s_{t+1}) - V_{\tau-1}(s_t))$$

TD (1-step)



Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

# Temporal Difference Learning

- Zdefiniowaliśmy algorytm TD dla najprostszego przypadku, gdy agent rozpatruje jedynie jeden okres w przód ( $TD(0)$ ):

$$V_{\tau}(s_t) = V_{\tau-1}(s_t) + \alpha(r_{t+1} + \beta V_{\tau-1}(s_{t+1}) - V_{\tau-1}(s_t))$$

- Co jeśli interesowałoby nas patrzenie  $n$  kroków przód?



Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

# Temporal Difference Learning

$$\begin{aligned} n = 1: & \quad R_t^{(1)} = r_{t+1} + \beta V_{\tau-1}(s_{t+1}) \\ n = 2: & \quad R_t^{(2)} = r_{t+1} + \beta r_{t+2} + \beta^2 V_{\tau-1}(s_{t+2}) \\ & \quad \vdots \\ n = \infty: & \quad R_t^{(\infty)} = r_{t+1} + \beta r_{t+2} + \beta^2 r_{t+3} + \cdots + \beta^{T-1} r_T \end{aligned}$$

- $TD(n)$ :

$$V_{\tau}(s_t) = V_{\tau-1}(s_t) + \alpha \left( R_t^{(n)} - V_{\tau-1}(s_t) \right)$$

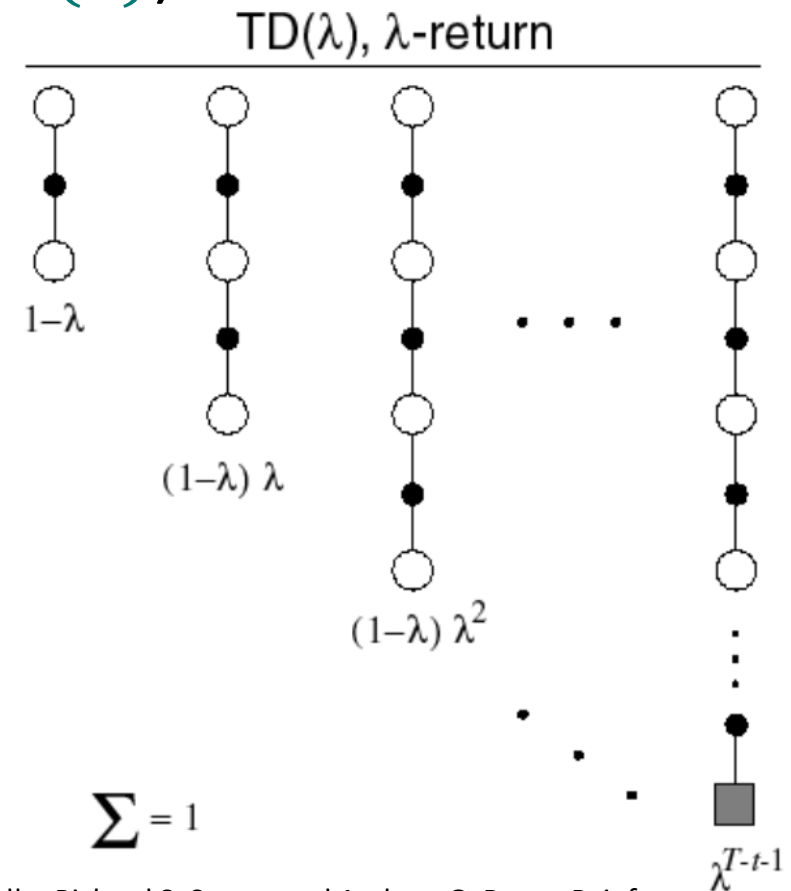
# Temporal Difference Learning

- Wiemy już, że możemy kontrolować długość doświadczenia z którego korzysta agent.
- Możemy też ważyć różne łańcuchy doświadczenia aby dostać efektywniejszą estymację funkcji wartości.

# Temporal Difference Learning

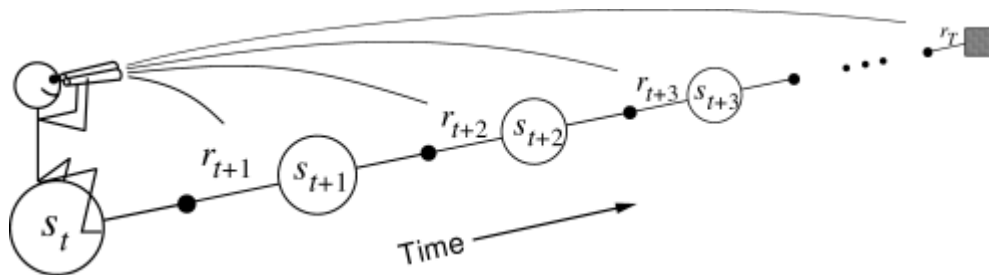
- Możemy też ważyć różne łańcuchy doświadczenia aby dostać efektywniejszą estymację funkcji wartości ( $TD(\lambda)$ ):

- $R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$
- $V_\tau(s_t) = V_{\tau-1}(s_t) + \alpha \left( R_t^\lambda - V_{\tau-1}(s_t) \right)$
- $\lambda = 0 \rightarrow TD(0)$
- $\lambda = 1 \rightarrow MC$



# Temporal Difference Learning

- **Forward view**

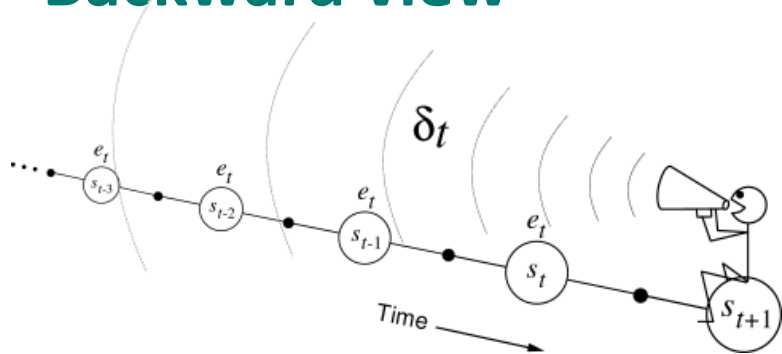


Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

- Uaktualnij funkcję wartości idąc w przód.
- Tak jak metody Monte Carlo może być stosowane jedynie pod koniec epizodu.

# Temporal Difference Learning

- **Backward view**

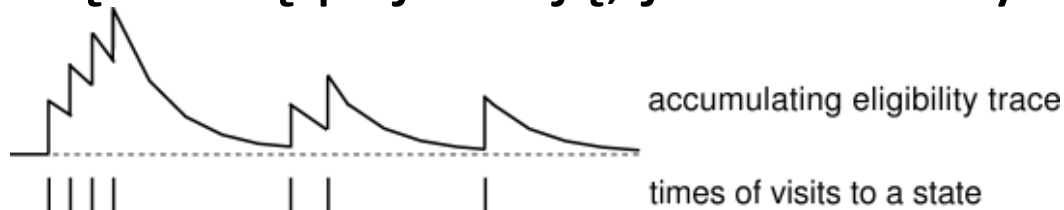


Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

- Uaktualnij funkcję wartości zaczynając od końca.
- Dzięki temu wartości funkcji  $V(s)$  mogą być uaktualniane w każdym kroku  $t$ .
- Jak jednak ważyć przeszłe stany?

# Temporal Difference Learning

- **Backward view**
- Można to zrobić za pomocą mechanizmu śladów wybieralności (*Eligibility traces*)
- Dzięki niemu jesteśmy w stanie oceniać stany ze względu na to jak często się pojawiają, jak i to kiedy ostatni raz w nich byliśmy:



Źródło: Richard S. Sutton and Andrew G. Barto, *Reinforcement learning, an introduction*, second edition

- Dzięki temu w każdym kroku  $t$  możemy uaktualnić ślad dla stanu  $s$ :

$$e_t(s) = \begin{cases} \beta \lambda e_{t-1}(s) & \text{gdy } s \neq s_t \\ \beta \lambda e_{t-1}(s) + 1 & \text{gdy } s = s_t \end{cases}$$



# Temporal Difference Learning

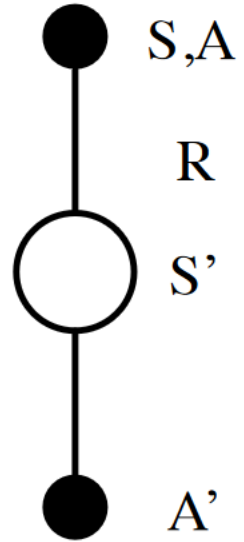
- **Backward view**
- Znając wartość  $e_t(s)$  w prosty sposób możemy uaktualnić  $V(s)$ :

$$\forall_{s \in S} V_{\tau}(s) = V_{\tau-1}(s) + \alpha e_t(s) (R_t - V_{\tau-1}(s))$$

# Temporal Difference Learning

- **On-policy**
  - SARSA
- **Off-policy**
  - Q- learning

# SARSA



$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

Źródło: [http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching\\_files/control.pdf](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/control.pdf)

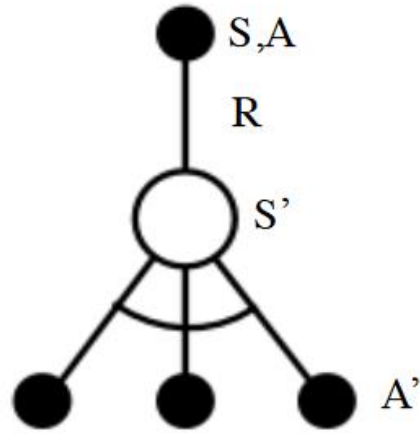
# SARSA

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np.  $Q_0(s, a) = 0 \forall s, a$  i  $0 < \epsilon < 1$ .
2. W każdej iteracji  $k$ :
  - Wyznacz stan początkowy  $S$  i akcję  $A$  za pomocą strategii wyprowadzonej z  $Q$  (np.  $\epsilon$ -zachłannej).
  - Dla każdego  $S, A$  należącego do epizodu:
    - Podejmij akcję  $A$  i zaobserwuj nagrodę  $R$  i stan  $S'$ .
    - Wybierz  $A'$  za pomocą strategii wyprowadzonej z  $Q$  (np.  $\epsilon$ -zachłannej).
    - Uaktualnij wartość funkcji  $Q_k(S, A)$ :
$$Q_k(S, A) = Q_{k-1}(S, A) + \alpha(R + \beta Q_{k-1}(S', A') - Q_{k-1}(S, A))$$
    - Przyjmij, że:  $S = S'$  i  $A = A'$ ; kontynuuj działanie aż osiągniesz stan terminalny.

# SARSA

- Algorytm SARSA można uogólnić:
  - ***SARSA( $n$ )***
  - ***SARSA( $\lambda$ )***

# Q-learning



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Źródło: [http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching\\_files/control.pdf](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/control.pdf)

# Q-learning

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np.  $Q_0(s, a) = 0 \forall s, a$  i  $0 < \epsilon < 1$ .
2. W każdej iteracji  $k$ :
  - Wyznacz stan początkowy  $S$
  - Dla każdego  $S$  należącego do epizodu:
    - Wyznacz akcję  $A$  za pomocą strategii wyprowadzonej z  $Q$  (np.  $\epsilon$ -zachłannej).
    - Podejmij akcję  $A$  i zaobserwuj nagrodę  $R$  i stan  $S'$ .
    - Uaktualnij wartość funkcji  $Q(S, A)$ :
$$Q_k(S, A) = Q_{k-1}(S, A) + \alpha(R + \beta \max_a Q_{k-1}(S', a) - Q_{k-1}(S, A))$$
    - Przyjmij, że:  $S = S'$ ; kontynuuj działanie aż osiągniesz stan terminalny.

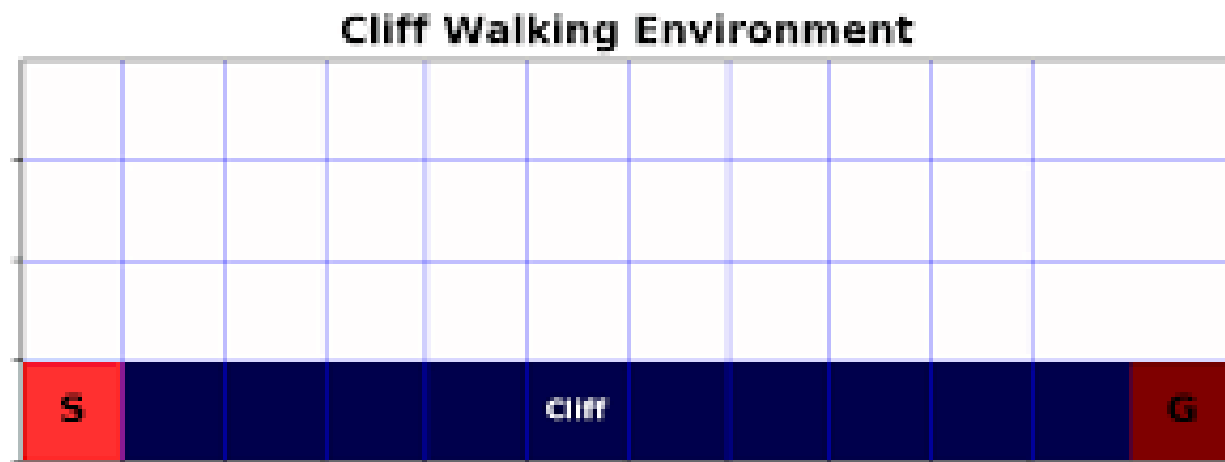
# Q-learning

- Możliwe modyfikacje:
  - **Watkin's**  $Q(\lambda)$
  - ***Opposition*-based**  $Q(\lambda)$



# Przykład

## Cliff Learning



Źródło: [https://yun-long.github.io/2017/11/RL\\_CW\\_TD/](https://yun-long.github.io/2017/11/RL_CW_TD/)

- Celem agenta jest przejść z punktu **S** do Punktu **G**.
- Agent może iść po białych polach musi unikać spadnięcia z klifu (pola granatowe).
- Tym razem świat jest w pełni deterministyczny.

# Warunki zbieżności

- Algorytm zbiega do rozwiązania optymalnego  $Q_*(s, a)$  gdy:

- *Greedy in the Limit with Infinite Exploration (GLIE):*

- Wszystkie pary  $s, a$  są odwiedzone nieskończoną liczbę razy:

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

- Strategia zbiega do strategii zachłannej:

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = I(a_* = \operatorname{argmax}_{a \in A} Q_k(s, a))$$

- np. gdy ustalimy, że  $\epsilon = \frac{1}{k}$

- Stopa uczenia się  $\alpha$  spełnia warunek Robbinsa-Monro:

$$\sum_{k=1}^{\infty} \alpha_k = \infty$$

$$\sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

# Współczynnik eksploracji $\epsilon$

- W przypadku uczenia bezmodelowego (metody Monte Carlo, uczenie różnicowe) chcemy zachować balans pomiędzy **eksploracją** a **eksploatacją**.
- Przyjęcie stałej wartości współczynnika eksploracji  $\epsilon$  jest dużym uproszczeniem z dwóch podstawowych powodów:
  - W początkowych fazach procesu uczenia chcemy żeby agent eksplorował możliwie jak najwięcej w celu zdobycia możliwie jak najpełniejszej informacji na temat otaczającego go świata.
  - Ale później, gdy jego strategia zaczyna zbiegać do strategii optymalnej, chcemy żeby jego zachowania były możliwie jak najbliższe optymalnej strategii – dzięki temu będzie on maksymalizował swoją użyteczność w każdym kolejnym epizodzie.
- Dlatego zazwyczaj przyjmujemy, że  $\epsilon$  maleje z czasem.

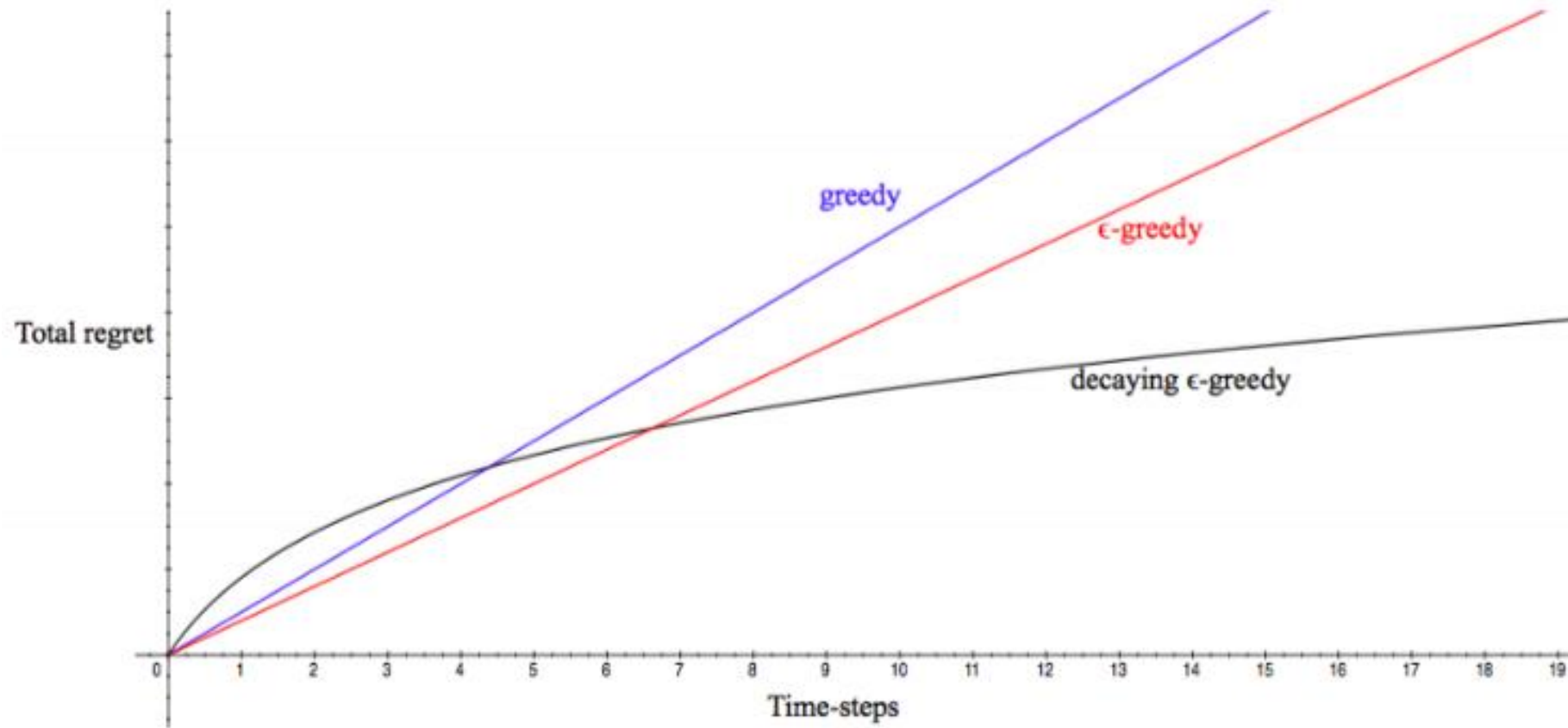
# Współczynnik eksploracji $\epsilon$

- Aby lepiej to zrozumieć zdefiniujmy pojęcie **żału**:

$$Reg_N = NE[R(A^*)] - \sum_{k=1}^N R_k(A_k)$$

gdzie  $A^*$  oznacza zbiór optymalnych akcji,  $A_k$  zbiór akcji podjętych w  $k$ -tym epizodzie a  $R(\cdot)$  skumulowaną nagrodę z danego epizodu.

# Współczynnik eksploracji $\epsilon$



Źródło: <https://towardsdatascience.com/exploration-in-reinforcement-learning-e59ec7eeaa75>

# Współczynnik eksploracji $\epsilon$

- Jak w takim razie kontrolować proces eksploracji?
  - $\epsilon = \frac{1}{k}$  - poprawny teoretycznie, w praktyce problematyczny
  - $\epsilon$  malejący liniowo – spada wolniej niż  $\epsilon = \frac{1}{k}$ , łatwiej go kontrolować
  - Dodać proces wypalania – agent przez pierwsze  $n$  iteracji zachowuje się całkowicie losowo, dopiero później zaczyna podejmować decyzję zgodnie ze strategią  $\epsilon$ -zachłanną:

$$\epsilon = \begin{cases} 1 & \text{gdy } k < n \\ \frac{1}{k} & \text{gdy } k \geq n \end{cases}$$

- Zamiast definiować  $\epsilon$  zastosować **górną granicę ufności** (*upper confidence bound*).

# Upper Confidence Bound

- **Górną granicę ufności** (*upper confidence bound, UBC*), dla epizodu  $t$ , stanu  $s$  i akcji  $a$  definiujemy jako:

$$U_t(s, a) = \sqrt{\frac{2 \ln t}{N_t(s, a)}}$$

gdzie  $N_t(s, a)$  jest licznikiem odwiedzin pary  $s, a$ .

- Przy tak zdefiniowanym  $U_t(s, a)$  agent wybiera akcję zachłannie maksymalizując:

$$a^* = \operatorname{argmax}_{a \in A} (Q_t(s, a) + U_t(s, a))$$