Programowanie Dynamiczne

w kontekście uczenia przez wzmacnianie

Bartosz Hanc

2.12.2025

O czym właściwie będziemy mówić?

"CLRS"

> ***Dynamic programming*** *typically applies to* ***optimization problems*** *in which we make a* ***set of choices*** *in order to arrive at an optimal solution. As we make each choice,* ***subproblems of the same form*** *often arise. Dynamic programming is effective when a given subproblem may arise from more than one partial set of choices; the key technique is to store the solution to each such subproblem in case it should reappear.*

R. Sutton, A. Barto, *Reinforcement Learning: An Introduction*

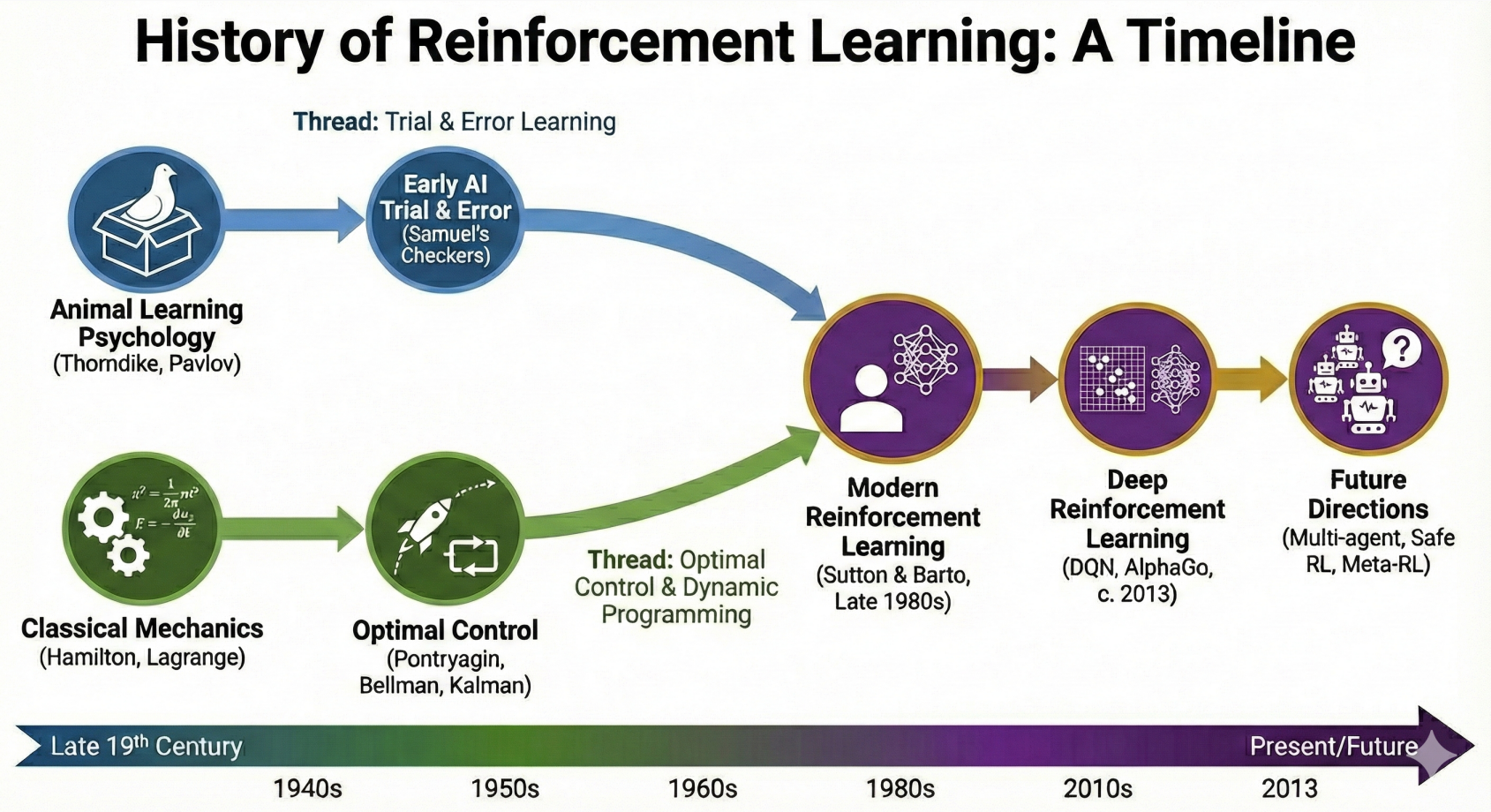
> *The term* ***dynamic programming*** *(DP) refers to a collection of algorithms that can be used to compute* ***optimal policies*** *given a perfect model of the environment as a* ***Markov decision process*** *(MDP).*

O czym właściwie będziemy mówić?

R. Bellman, Preface to *Dynamic Programming*

> *The purpose of this work is to provide an introduction to the mathematical theory of* ***multi-stage decision processes****. Since these constitute a somewhat formidable set of terms we have coined the term* ***"dynamic programming"*** *to describe the subject matter. [...]*
>
> *Let us suppose that we have a* ***physical system*** *S* ***whose state at any time*** *t* ***is specified by a vector*** *p. [...] Assume now that [...] we have a process in which we have a* ***choice of the transformations*** *which may be* ***applied to the system*** *at any time. A process of this type we call a [multi-stage] decision process.*

Tło historyczne



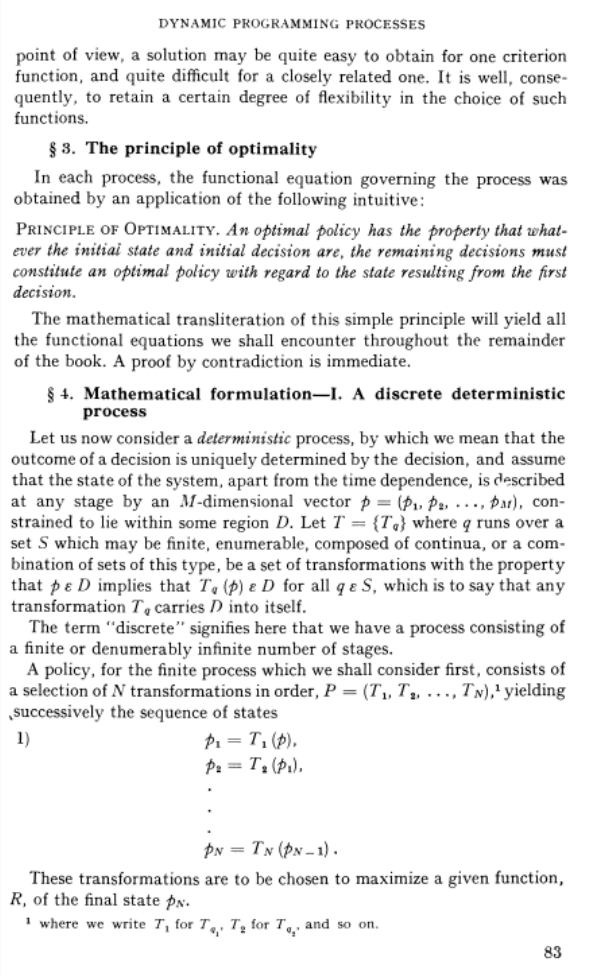
Tło historyczne

War Games, RAND Corporation, ∼1950

IBM 7090 – komputer tranzystorowy, 100 Kflop/s, 32K słów pamięci (36 bit / słowo), 1959

Tło historyczne

Richard E. Bellman



DYNAMIC PROGRAMMING PROCESSES

point of view, a solution may be quite easy to obtain for one criterion function, and quite difficult for a closely related one. It is well, consequently, to retain a certain degree of flexibility in the choice of such functions.

§ 3. The principle of optimality

In each process, the functional equation governing the process was obtained by an application of the following intuitive:

PRINCIPLE OF OPTIMALITY. *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

The mathematical transliteration of this simple principle will yield all the functional equations we shall encounter throughout the remainder of the book. A proof by contradiction is immediate.

§ 4. Mathematical formulation—I. A discrete deterministic process

Let us now consider a *deterministic* process, by which we mean that the outcome of a decision is uniquely determined by the decision, and assume that the state of the system, apart from the time dependence, is described at any stage by an M-dimensional vector $p = (p_1, p_2, \ldots, p_M)$, constrained to lie within some region D. Let $T = \{T_q\}$ where q runs over a set S which may be finite, enumerable, composed of continua, or a combination of sets of this type, be a set of transformations with the property that $p \varepsilon D$ implies that $T_q(p) \varepsilon D$ for all $q \varepsilon S$, which is to say that any transformation T_q carries D into itself.

The term "discrete" signifies here that we have a process consisting of a finite or denumerably infinite number of stages.

A policy, for the finite process which we shall consider first, consists of a selection of N transformations in order, $P = (T_1, T_2, \ldots, T_N)$,[1] yielding successively the sequence of states

$$
\begin{aligned}
1) \qquad p_1 &= T_1(p), \\
p_2 &= T_2(p_1), \\
&\cdot \\
&\cdot \\
&\cdot \\
p_N &= T_N(p_{N-1}).
\end{aligned}
$$

These transformations are to be chosen to maximize a given function, R, of the final state p_N.

[1] where we write T_1 for T_{q_1}, T_2 for T_{q_2}, and so on.

83

Dynamic Programming, R. E. Bellman, 1957

Uwaga dotycząca notacji

- Notacja dotycząca programowania dynamicznego nie jest spójna między RL i teorią sterowania.
- Nie jest nawet spójna w obrębie teorii sterowania.
- W tej prezentacji będziemy używać notacji zaczerpniętej z RLa.

Problemy deterministyczne ze skończonym horyzontem

Rozpatrujemy dyskretny układ dynamiczny, którego ewolucja jest opisana przez

$$s_{k+1} = f_k(s_k, a_k)\,, \quad k = 0, 1, 2, \ldots, K-1$$

dla pewnego **znanego** ciągu funkcji $(f_k)_{k=0}^{K-1}$.

- $s_k \in \mathcal{S}_k$ jest stanem układu w kroku k,
- $a_k \in \mathcal{A}_k(s_k)$ jest sterowaniem (inaczej akcją)
- zbiory $\mathcal{S}_k$ i $\mathcal{A}_k(s_k)$ wyznaczają dopuszczalne stany i akcje

Chcemy znaleźć ciąg akcji $(a_k)_{k=0}^{K-1}$, który maksymalizuje **zwrot** (sumę nagród), zakładając dany stan początkowy s_0

$$R^* = \max_{a_0, a_1, \ldots, a_{K-1}} \left[r_K(s_K) + \sum_{k=0}^{K-1} r_k\left(s_k, a_k\right) \right]\,,$$

gdzie ciąg funkcji nagród $(r_k)_{k=0}^{K}$ również jest znany.

Problemy deterministyczne ze skończonym horyzontem

Wprowadźmy ciąg funkcji

$$V_k(s_k) := \max_{a_k, a_{k+1}, \ldots, a_{K-1}} \left[r_K(s_K) + \sum_{j=k}^{K-1} r_j(s_j, a_j) \right]$$

Zauważmy, że $V_0(s_0) = R^*$. Jednocześnie

$$\begin{aligned}
V_k(s_k) &= \max_{a_k, a_{k+1}, \ldots, a_{K-1}} \left[r_k(s_k, a_k) + r_K(s_K) + \sum_{j=k+1}^{K-1} r_j(s_j, a_j) \right] \\
&= \max_{a_k} \max_{a_{k+1}, a_{k+2}, \ldots, a_{K-1}} \left[r_k(s_k, a_k) + r_K(s_K) + \sum_{j=k+1}^{K-1} r_j(s_j, a_j) \right] \\
&= \max_{a_k} \left\{ r_k(s_k, a_k) + \max_{a_{k+1}, a_{k+2}, \ldots, a_{K-1}} \left[r_K(s_K) + \sum_{j=k+1}^{K-1} r_j(s_j, a_j) \right] \right\} \\
&= \max_{a_k} \left[r_k(s_k, a_k) + V_{k+1}(s_{k+1}) \right]
\end{aligned}$$

Zasada optymalności Bellmana

- Zauważmy, że obliczenie ciągu funkcji (V_k) daje nam coś więcej niż tylko rozwiązanie początkowo postawionego problemu. Dla **dowolnego** stanu s i kroku k mamy bowiem **maksymalny zwrot, który można uzyskać zaczynając w kroku** k **w stanie** s.
- Dodatkowo zapamiętując akcję a, dla której występuje maksimum możemy również znaleźć optymalną **politykę**, czyli ciąg funkcji (π_k) określających optymalną akcję, którą należy podjąć będąc w kroku k w stanie s.
- Zamiast rozwiązania w postaci **pojedynczej** trajektorii $(a_0, a_1, \ldots, a_{K-1})$ dla **jednego** stanu początkowego w kroku 0, dostajemy rozwiązanie w postaci polityki $(\pi_0, \pi_1, \ldots, \pi_{K-1})$ dla **każdego** stanu początkowego w dowolnym kroku.

Zasada optymalności Bellmana

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

$$\boxed{\begin{aligned} V_k(s) &= \max_a \Big[r_k(s,a) + V_{k+1}\big(f_k(s,a)\big) \Big] \\ \pi_k(s) &= \arg\max_a \Big[r_k(s,a) + V_{k+1}\big(f_k(s,a)\big) \Big] \end{aligned}}$$

Przykład: Sad

„Bajtek zgłasza swój sad do konkursu. Aby spełnić wymogi komisji, suma wszystkich jabłek w sadzie musi być podzielna przez m. Bajtek chce ściąć minimalną liczbę drzew, aby spełnić ten warunek. Tablica T zawiera liczby jabłek na kolejnych drzewach."

$$
\begin{aligned}
\{0,1\} \ni a_k &:= \text{czy należy ściąć } k\text{-te drzewo} \\
[m] \ni s_k &:= \text{reszta z dzielenia sumy jabłek po } k \text{ krokach} \\
f_k(s,a) &:= (s - a \cdot T_k) \quad \textbf{mod } m \\
r_k(s,a) &:= -a \\
r_K(s) &:= 0 \text{ if } s = 0 \text{ else } -\infty
\end{aligned}
$$

```python
def orchard(T, m):
    K = len(T)
    V = [[inf for _ in range(m)] for _ in range(K + 1)]
    V[K][0] = 0

    for k in reversed(range(K)):
        for s in range(m):
            V[k][s] = min(V[k + 1][s], 1 + V[k + 1][(s - T[k]) % m])

    return V[0][sum(T) % m]
```

Przykład: Parking

„Inwestor ma n biurowców na pozycjach w posortowanej tablicy X i $m \geq n$ dostępnych działek pod parkingi na pozycjach w posortowanej tablicy Y. Chce wybudować po jednym parkingu dla każdego biurowca tak, aby suma odległości między biurowcem a jego parkingiem była minimalna. Przypisanie musi zachowywać kolejność: jeśli biurowiec i stoi przed biurowcem j, to parking dla i też musi znajdować się przed parkingiem dla j."

$\{0,1\} \ni a_k :=$ czy budujemy k-ty parking

$[n+1] \ni s_k :=$ ile parkingów mamy po k krokach

$\mathcal{A}_k(s) := \{0,1\}$ if $s < n$ else $\{0\}$

$f_k(s,a) := s + a$

$r_k(s,a) := -|X_s - Y_k|$ if $a = 1$ else 0

$r_m(s) := 0$ if $s = n$ else $-\infty$

```
def parking(X, Y):
    n, m = len(X), len(Y)
    V = [[inf for _ in range(n + 1)] for _ in range(m + 1)]
    V[m][n] = 0

    for k in reversed(range(m)):
        for s in range(n + 1):
            if s < n:
                V[k][s] = min(V[k + 1][s], abs(X[s] - Y[k]) + V[k + 1][s + 1])
            else:
                V[k][s] = V[k + 1][s]

    return V[0][0]
```

Problemy stochastyczne ze skończonym horyzontem

W celu uwzględnienia efektów losowych, do równania ewolucji układu wprowadzamy zmienne losowe w_k

$$s_{k+1} = f_k(s_k, a_k, w_k)\,, \quad k = 0, 1, 2, \ldots, K-1\,,$$

przy czym zakładamy, że zmienne losowe (w_k) są **niezależne** i **znamy** ich rozkłady $p_k(w_k)$. Wówczas interesuje nas znalezienie polityki $(\pi_0, \pi_1, \ldots, \pi_{K-1})$, która maksymalizuje **oczekiwany zwrot**

$$\sum_{w_0, w_1, \ldots, w_{K-1}} \left[r_K(s_K) + \sum_{k=0}^{K-1} r_k\big(s_k, \pi_k(s_k)\big) \right] p_0(w_0) p_1(w_1) \ldots p_{K-1}(w_{K-1})$$

Problemy stochastyczne ze skończonym horyzontem

Można pokazać, analogicznie jak dla przypadku deterministycznego (choć wymaga to większej „rzeźby” algebraicznej i rozważenia pewnych bardziej subtelnych problemów), że w przypadku stochastycznym równanie Bellmana przyjmuje postać

$$V_k(s) = \max_a \sum_w p_k(w)\Big[r_k(s,a) + V_{k+1}\big(f_k(s,a,w)\big)\Big],$$

gdzie $V_k(s)$ jest teraz maksymalnym **oczekiwanym** zwrotem, jaki można uzyskać startując ze stanu s w kroku k.

Problemy z nieskończonym horyzontem

W uczeniu przez wzmacnianie typowo formułujemy problem bez ustalonego horyzontu jako

$$\max_{\pi} \mathop{\mathbb{E}}_{w_0, w_1, \ldots} \left[\sum_{k=0}^{\infty} \gamma^k r\big(s_k, \pi(s_k)\big) \right],$$

gdzie $0 < \gamma < 1$ to tzw. *discount factor*. Przyjmujemy, że funkcje przejścia f, nagród r i rozkłady zmiennych losowych nie zależą od kroku k (tzw. *steady state*). Pod pewnymi warunkami (w szczególności w przypadku gdy $0 < \gamma < 1$ i funkcja nagrody jest ograniczona) można pokazać, iż optymalne wartościowanie stanów V^* nie zależy od kroku k is spełnia równanie Bellmana

$$V^*(s) = \max_a \mathop{\mathbb{E}}_{w} \Big[r(s, a) + \gamma V^*\big(f(s, a, w)\big) \Big]$$

Value Iteration

Dodatkowo w wymienionym przypadku można pokazać, że ciąg funkcji (V_k)

$$V_{k+1}(s) = \max_a \mathop{\mathbb{E}}_w \Big[r(s,a) + \gamma V_k\big(f(s,a,w)\big) \Big]$$

jest zbieżny do V^* dla dowolnej ograniczonej funkcji V_0, przy czym błąd $|V^*(s) - V_k(s)|$ jest ograniczony z góry przez $c\gamma^k$ dla pewnej stałej c. Jest to tzw. algorytm *value iteration*.

Ograniczenia programowania dynamicznego

- DP zakłada, że znamy dokładny model środowiska. W przypadku deterministycznym konieczna jest znajomość funkcji przejścia, natomiast w przypadku stochastycznym wymagamy dodatkowo rozkładów zmiennych losowych (ewentualnie symulatora, z którego moglibyśmy estymować wartości oczekiwane metodą Monte Carlo).
- W programowaniu dynamicznym nie ma **uczenia się** jako takiego.
- **Klątwa wymiarowości**. Wymagania obliczeniowe DP bardzo szybko rosną wraz ze złożonością problemu.

Bibliografia

- Richard Bellman, *Dynamic Programming*, Dover Publications, 1957.
- D.P. Bertsekas, *Dynamic programming and optimal control*, 2 ed., vol. 1 and 2, Athena Scientific, 2001.
- R. Larson, *A survey of dynamic programming computational procedures*, IEEE Transactions on Automatic Control **12** (1967), no. 6, 767–774.
- Richard S. Sutton and Andrew G. Barto, *Reinforcement learning: An introduction*, second ed., The MIT Press, 2018.