BME-AUT MI tehetséggondozás szeminárium

Unsupervised és reinforcement learning

BME AUT
Department of Automation and Applied Informatics

A mai alkalmon ...

- Gépi tanulási paradigmák
- Unsupervised learning
- Reinforcement learning bevezető
- Bellman-egyenlet
- Megoldási módszerek
- DQN
- Demó (Q-learning, Sarsa-learning, DP)

Gépi tanulási paradigmák

A 3 paradigma

Supervised

- összegyüjtött adat
- bementi és kimeneti párok
- reprezentációt tanul a bemenet és kimenet között
- legnépszerűbb

Unsupervised

- összegyűjtött adat
- csak bemenet van, elvárt kimenet nélkül
- adat belső struktúrájának feltárása a cél, például klaszterezés
- relatíve gyakran használt

Reinforcement

- nincs összegyűjtött adat
- környezet van, amivel egy ágens kölcsönhat, az interakciókból történik a tanulás
- viselkedést tanul
- ritkán használt, de a trendek alapján egyre gyakrabban

Unsupervised learning

Bevezetés

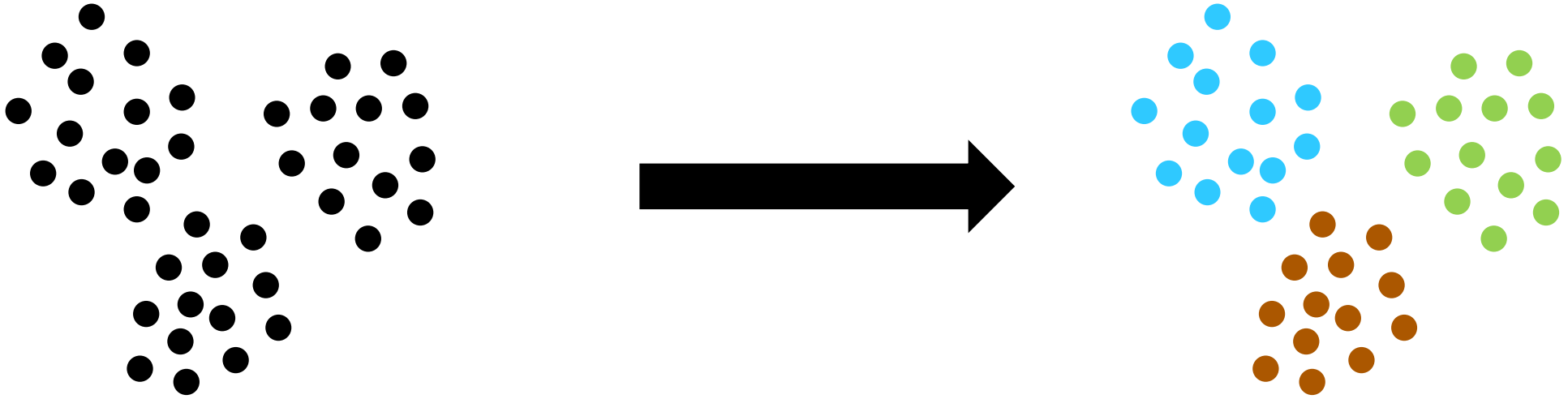
- Nincsenek labelek, a mintákat, összefüggéseket kell megtalálni az adatban
- Feladatok:
  - Klaszterezés
  - Association mining
  - Feature extraction (kisebb látens teret készíteni)
  - Dimenzió csökkentés
  - Sok más pl. mélység becslés HD képen

Bevezetés

- Klaszterezés (leggyakoribb feladat)
- Típusai:
  - Exclusive (partitioning) (pl. k-means)
  - Agglomerative (pl. hierarchical)
  - Overlapping (pl. Fuzzy methods)
  - Probabilistic (pl. Mixture of Gaussians)

Klaszterezés – k-means

A feladat:



Ez a bemeneti adat

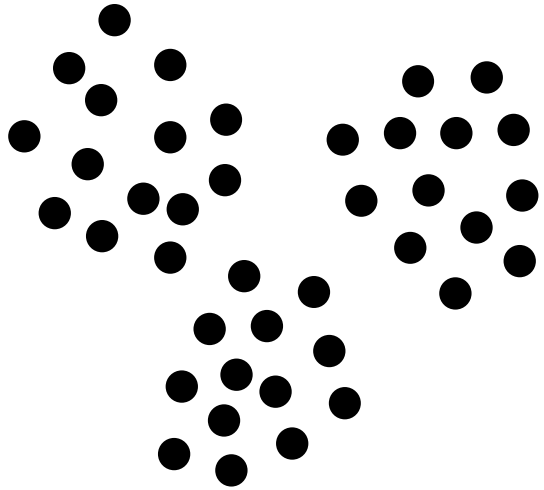
3 db klaszter

Klaszterezés – k-means

A k-means működése:

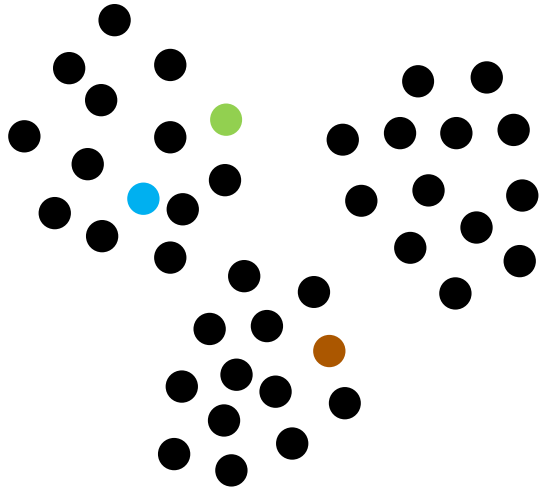
1. Bemenet: a k hiperparaméter, adat pontok
2. Inicializálunk k db klaszter középpontot (pl. random)
3. Minden adat pontot besorolunk egy klaszterbe az alapján, hogy a k db klaszter középpont közül melyikhez van legközelebb.
4. A létrejövő klaszterek alapján új klaszterközéppontokat határozunk meg.
5. Ismételjük, amíg nincs már változás, vagy sok az iteráció

Klaszterezés – k-means



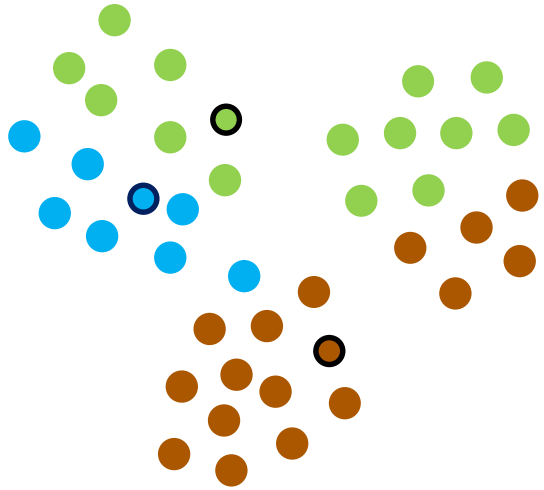
Lépés 1: k= 3

Klaszterezés – k-means



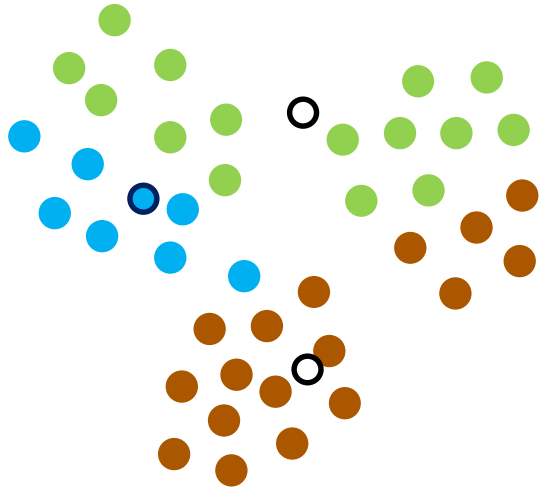
Lépés 2: klaszter centrumok inicializálása

Klaszterezés – k-means



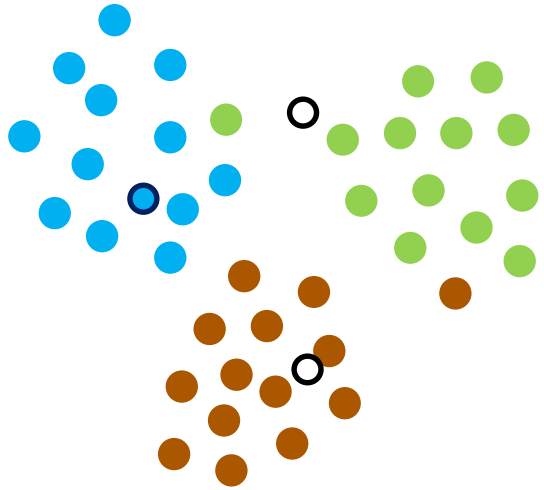
Lépés 3: adatpontok centrumokhoz rendelése (a távolság most euklideszi)

Klaszterezés – k-means



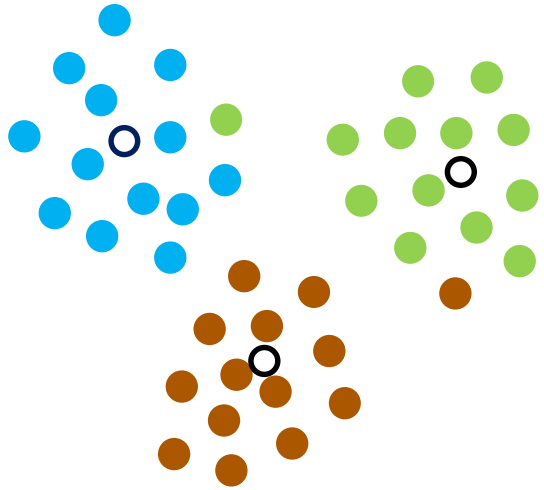
Lépés 4: új klaszter középpontok kiválasztása, az új centrumok nem új adatpontok

Klaszterezés – k-means



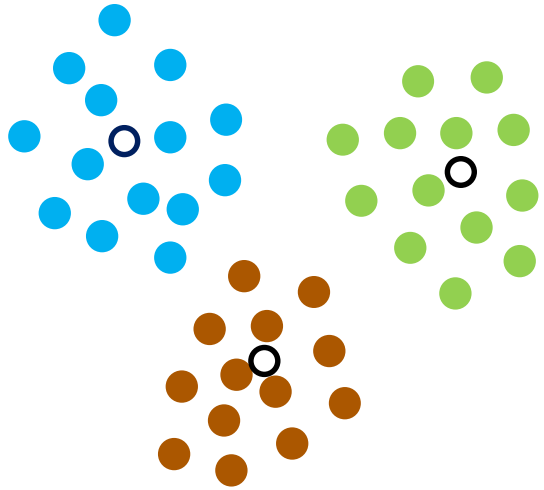
Lépés 3 újból: klaszterek meghatározása

Klaszterezés – k-means



Lépés 4 újból: klaszter középpont újra számolás

Klaszterezés – k-means



Lépés 3 újból: klaszterek újra osztása

Ezután már nem fog változni, így itt véget ér az algoritmus futása!

Klaszterezés – k-means

Részletek:

- Adatpont megadása, mint egy vektor (ha n attribútum van):

$$\vec{x} = (x_1, x_2, \dots, x_n)$$

- Távolság két pont között, a klaszter kiosztáshoz:

$$d_{ab} = \|\vec{x}_a - \vec{x}_b\|_2$$
$$= \sqrt{(x_{a1} - x_{b1})^2 + (x_{a2} - x_{b2})^2 + \cdots + (x_{an} - x_{bn})^2}$$

- Új középpont meghatározása (tömegközéppont):

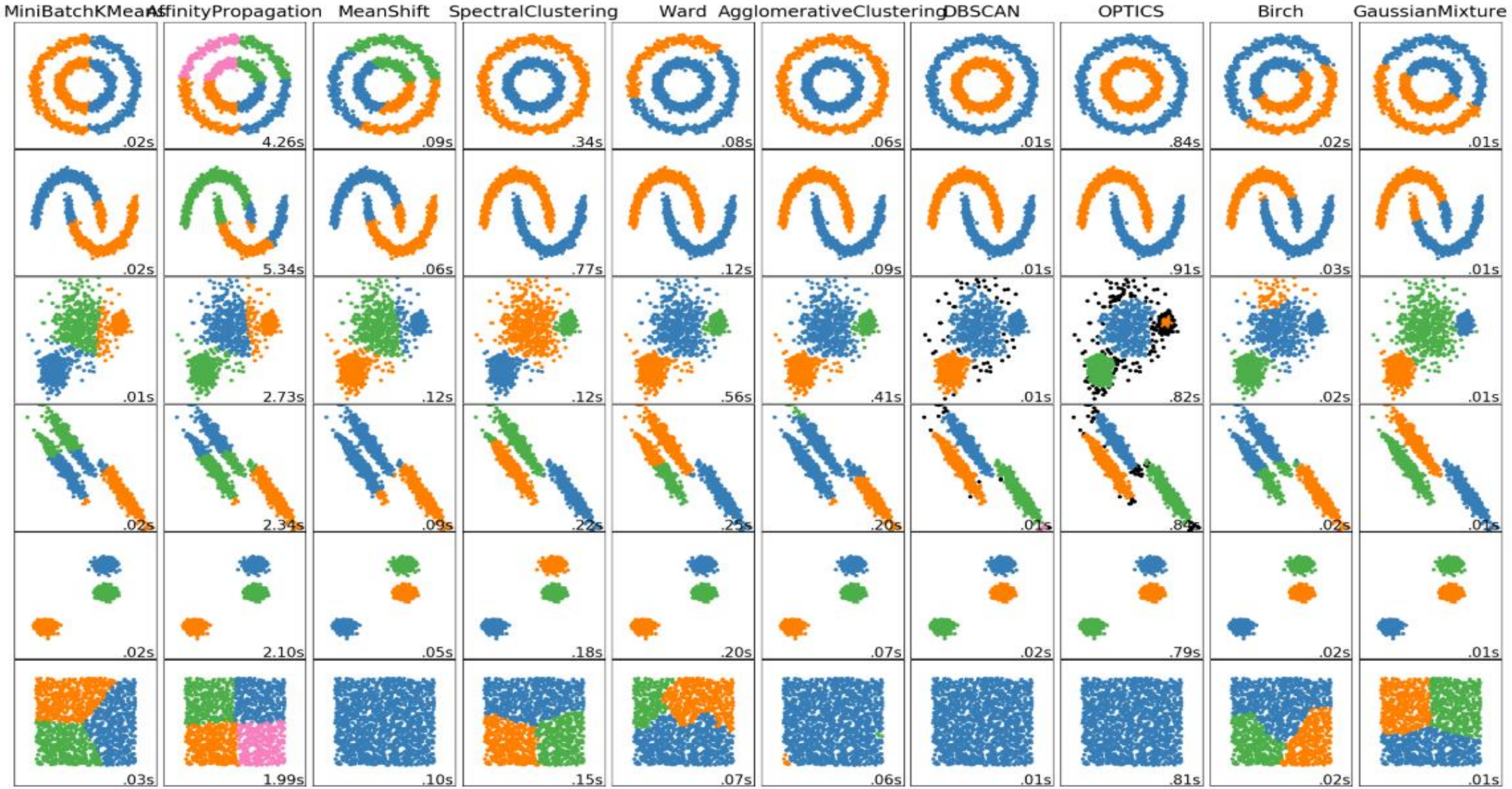
$$\vec{c} = \frac{1}{s} \sum_{i=1}^{s} \vec{x}_i$$

Klaszterezés – k-means

Részletek:

- A konvergencia garantált az euklideszi távolság és tömegközéppont számítás esetén
- Ha más távolság metrika van, akkor más center számítás kell
- A végállapot nem egyértelmű, függ az inicializációtól

Klaszterezés – más algoritmusok



Előnyök, hátrányok

Előnyök:

- Nem kell annotált adat

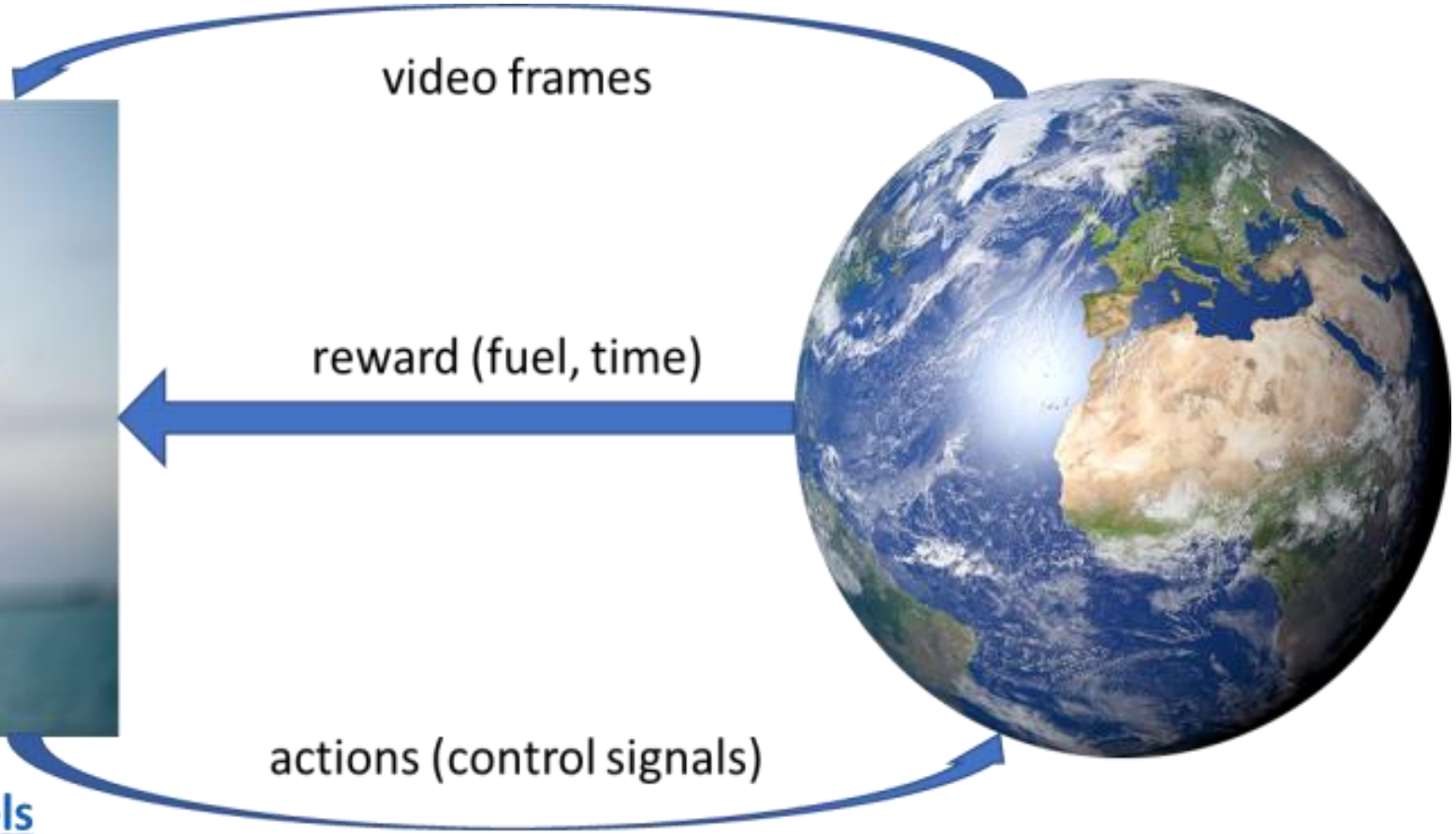
Hátrányok:

- Nagyobb számítási kapacitás
- Kevésbé megbízható eredmények

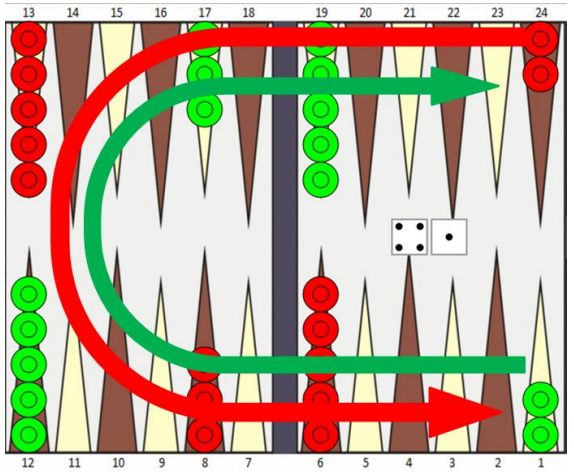
Reinforcement learning

Bevezetés

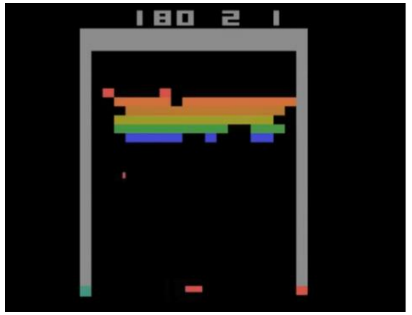
Photo by Josh Sorenson from Pexels



Bevezetés - sikerek



- Backgammon (1992); komplex játék
- Helikopter repülni tanul (Peter Abbeel, 2008)
- Atari játékok (2014 környéke)
  - > Áttörést jelentett
  - > Deep RL kezdete
  - > Human-level performance sok játékon



- AlphaGo
- Dota2, Starcraft

Ezek mind játékok. Nagyon sok testbed (benchmark) az RL-ben játékokon alapul, hiszen ezeket könnyű lekódolni, olcsók, biztonságosak.

Bevezetés - alkalmazások

- Nascar (RL alapú döntéstámogatás)
- Osaro (robotika)
- Covariant (robotika)
- Bonsai (RL platform)
- JP Morgan (foreign exchange)

Bevezetés - háttér

- Néhány kiemelt pont:
  - Law of effect (pszihológia)
  - Optimal control (mechanikai rendszerek)
- Galambok pingpongoznak (law of effect)
- RL pingpongozik

Bellman-egyenlet

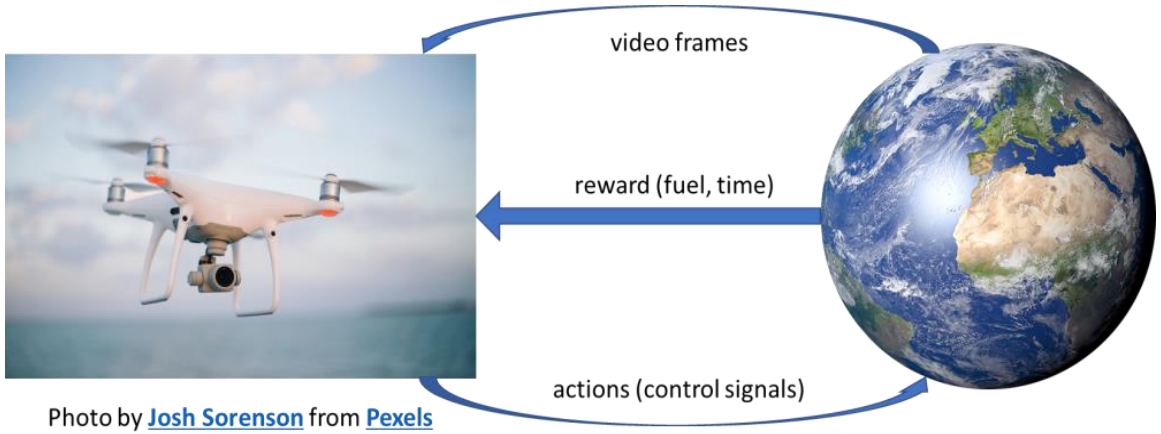


Photo by Josh Sorenson from Pexels

- ágens (pl. drón)
- környezet
- state (távolságok, sebességek)
- action (mozgás irány megjelölése)
- reward
- policy (viselkedést ír le)
- Cél: olyan policy-t találni, ami a legtöbb rewardot gyűjti

Bellman-egyenlet

Policy

- $\pi: S \rightarrow A$ *determinisztikus*
- $\pi: S \rightarrow P(A)$ *sztochasztikus*

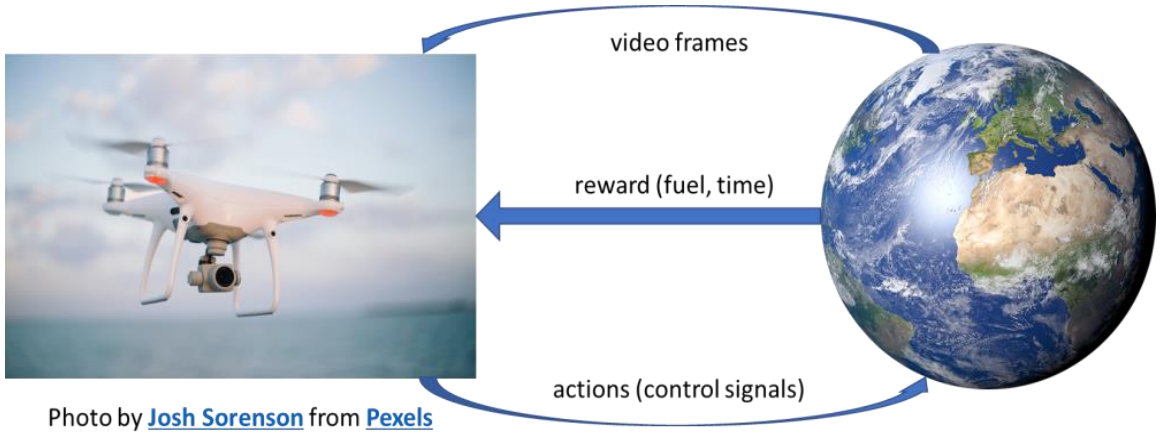


Photo by Josh Sorenson from Pexels

- A policy tehát az ágens elemi döntéseit adja meg
- Elemi, mert minden lépésben (interakciónál) dönteni kell

Bellman-egyenlet

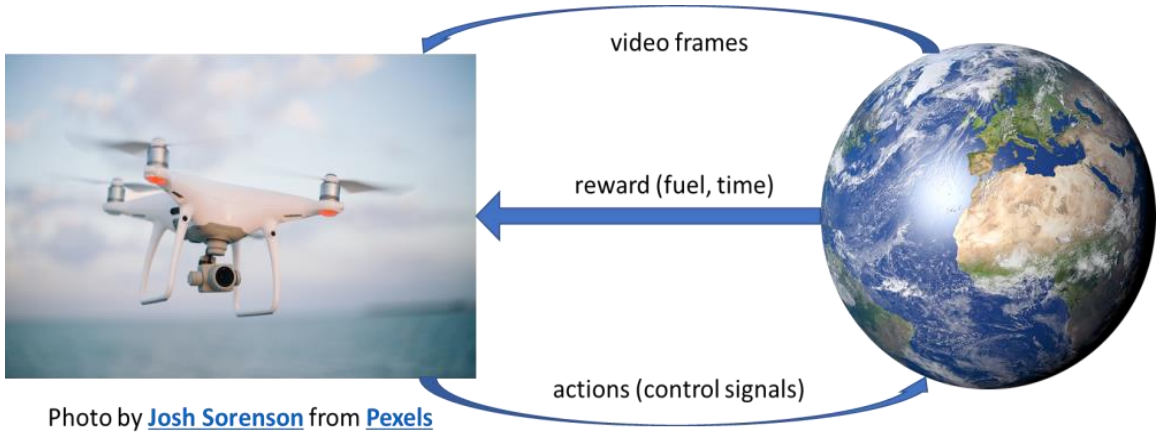


Photo by Josh Sorenson from Pexels

Reward

- Az ágens rewardot kap
- Minden lépés után

döntés → reward → döntés → reward → döntés → reward → döntés → vége

- A teljes szekvencia hasznossága (return):

$$G = \sum_t \gamma^t r_t \, , 0 < \gamma < 1$$

- gamma: discounting factor. A közeli reward értékesebb.

Bellman-egyenlet

Átmenet mátrix

- A környezet dinamikája

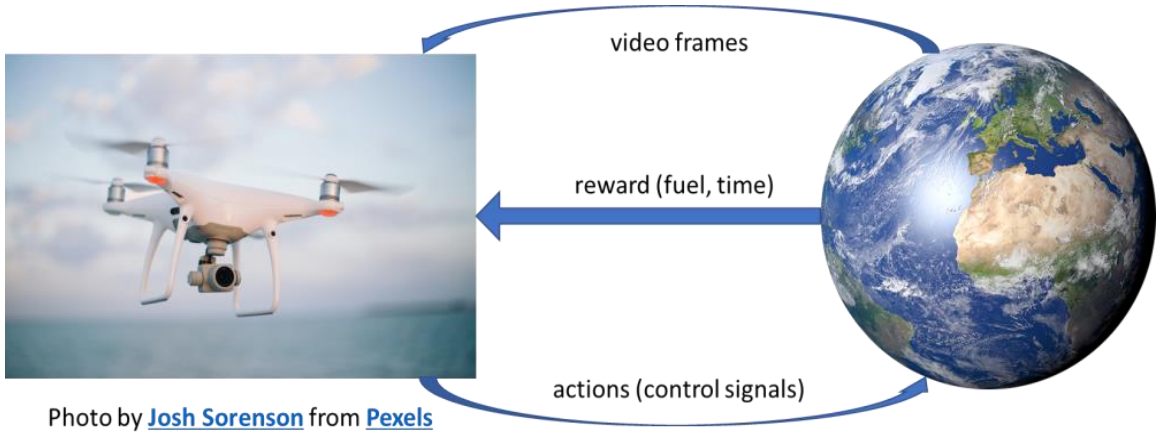
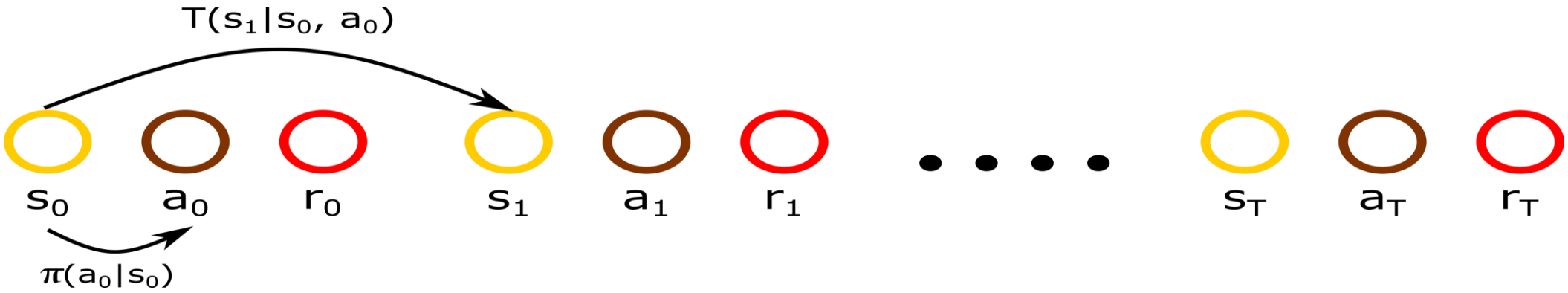


Photo by Josh Sorenson from Pexels

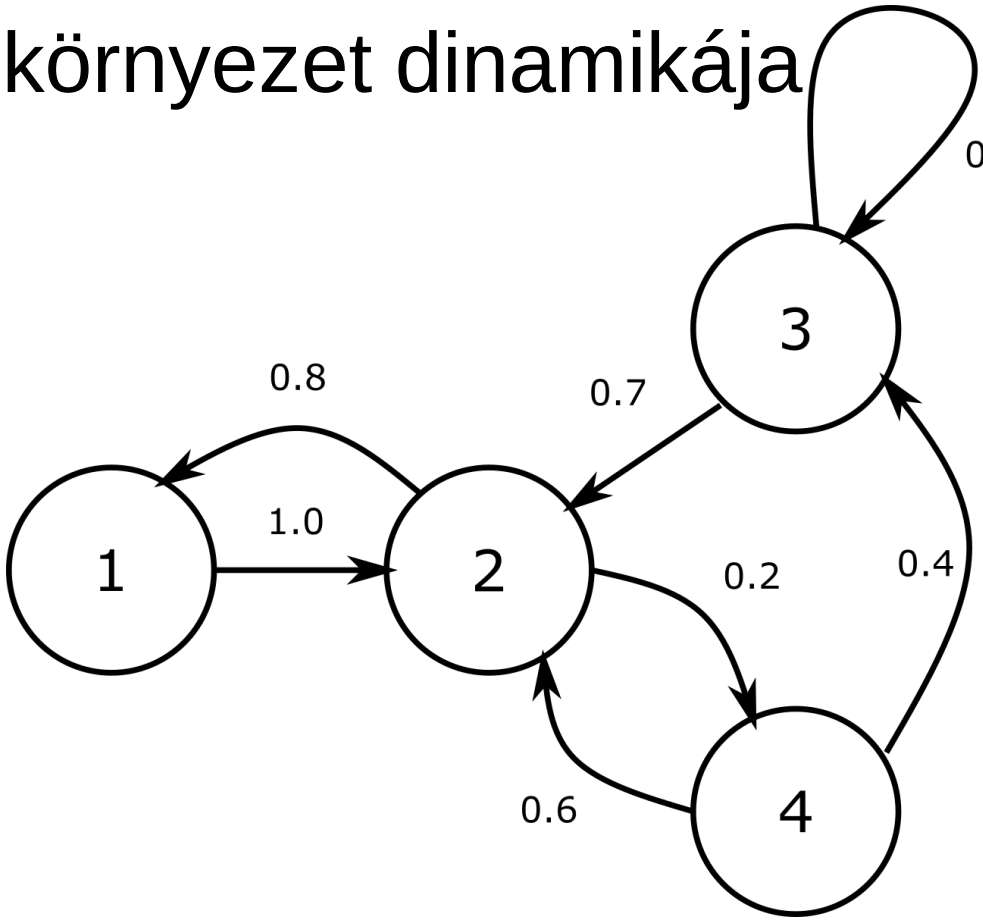


- Erre később úgy hivatkozunk, majd hogy **trajektória**

Bellman-egyenlet

Átmenet mátrix

- A környezet dinamikája



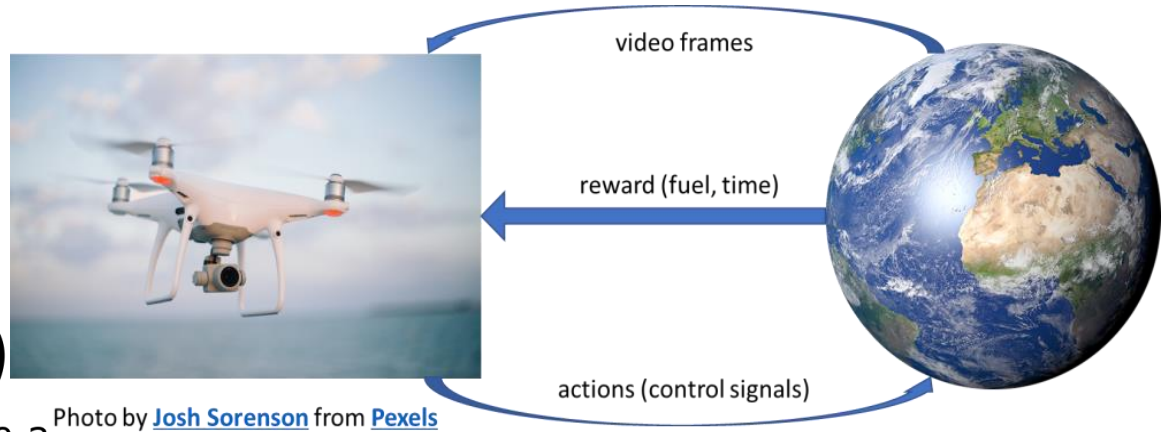


Photo by Josh Sorenson from Pexels

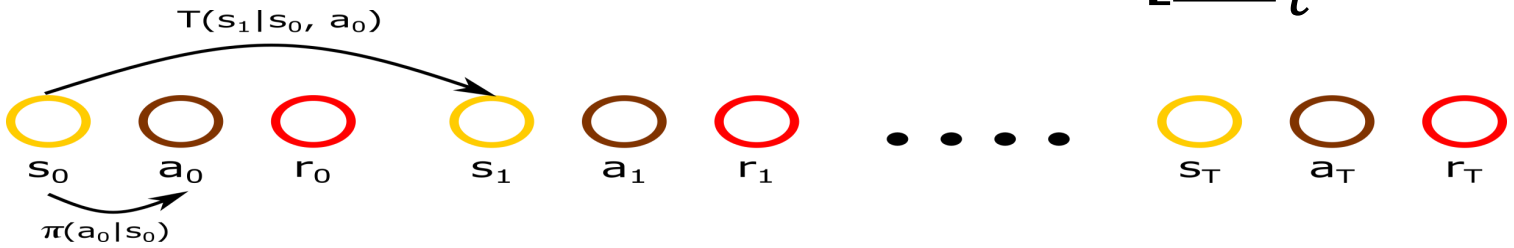
$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.8 & 0.0 & 0.0 & 0.2 \\ 0.0 & 0.7 & 0.3 & 0.0 \\ 0.0 & 0.6 & 0.4 & 0.0 \end{bmatrix}$$

Bellman-egyenlet

Action-value function

- Megmondja, hogy egy adott állapotból, ha teszünk egy lépést, majd utána követjük a policy-t, akkor mekkora lesz a várható hasznosság (G).

$$Q^{\pi}(s,a) = E_{\tau}\left[\sum_{t}\gamma^{t} r_t \,|\pi, r_i \in \tau\right]$$



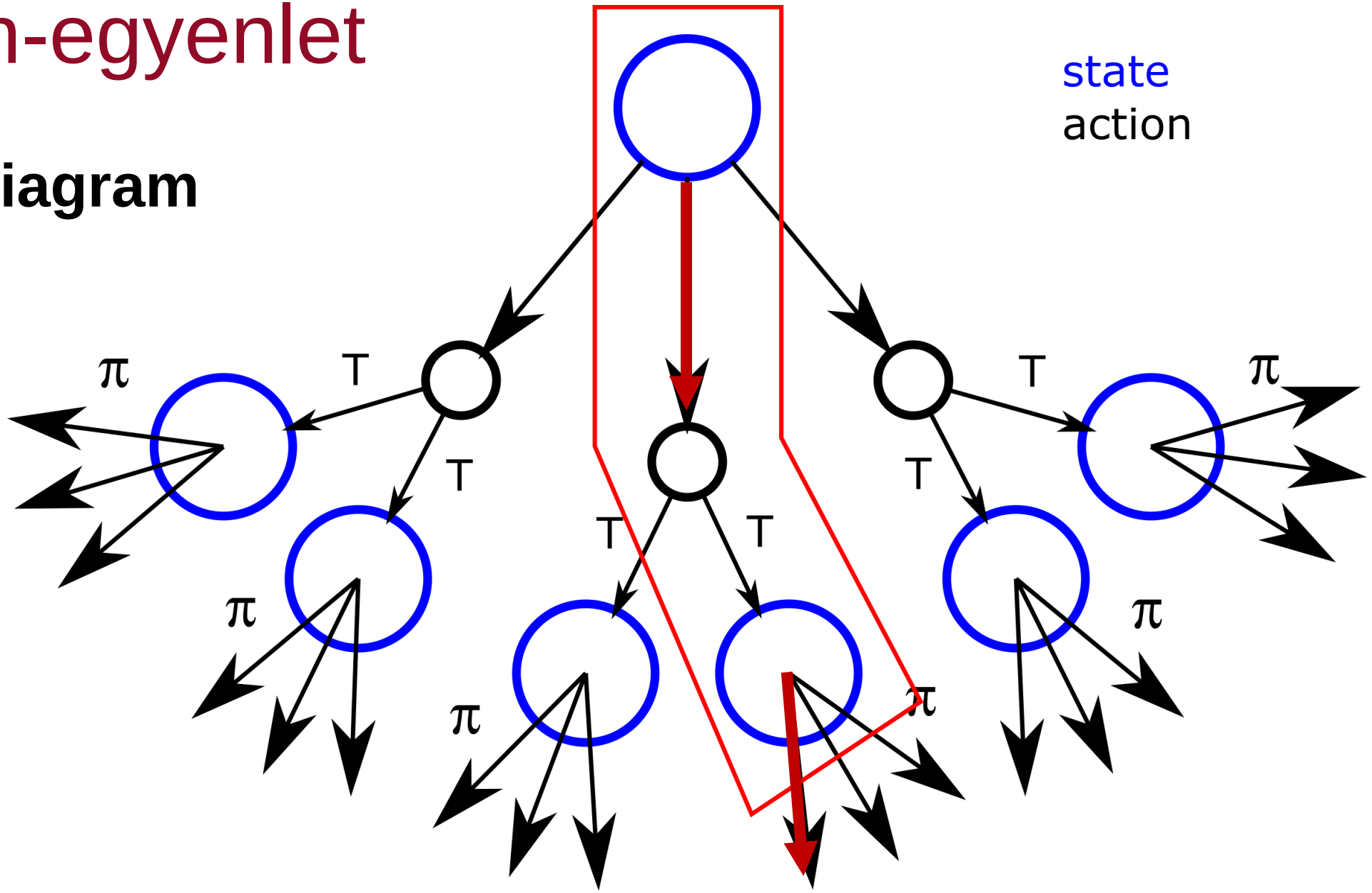
Bellman-egyenlet

A Bellman-egyenlet a Q függvényre felírt rekurzív formula alapján adódik. Lényegében egy dinamikus programozáson alapuló összefüggésről van szó.

Dinamikus programozás: *0-1 Hátizsák probléma*

Bellman-egyenlet

Backup-diagram



Bellman-egyenlet

Backup-diagram

Elemi értéke egy átmenetnek:

$$q_{sa}(s', a') = r(s, a, s') + \gamma Q(s', a')$$

Valószínűsége egy átmenetnek:

$$p_{sa}(s', a') = T(s, a, s') \cdot \pi(s', a')$$

Innen:

$$Q^{\pi}(s, a) = \sum_{s', a'} p_{sa}(s', a') q_{sa}(s', a') = \sum_{s', a'} \pi(s', a') \cdot T(s, a, s')[r(s, a, s') + \gamma Q^{\pi}(s', a')]$$

Bellman-egyenlet

Összefoglalva

$$Q^{\pi}(s,a) = \sum_{s',a'} \pi(s',a') \cdot T(s,a,s')[r(s,a,s') + \gamma Q^{\pi}(s',a')]$$

Bellman-egyenlet

Mire jó az action-value function?

Ha ismerném a tökéletes Q-t, azaz az optimális policy-hez tartozó Q-t, akkor azonnal megvan a policy is:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Optimalitás jele gyakran: *-gal (pl. Q*)

Bellman-egyenlet

Az optimális Bellman-egyenlet:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]$$

A max oka, hogyha ez az optimális policy-hez tartozik, akkor az optimális policy döntése az argmaxxal adódik, lásd előző dia.

Bellman-egyenlet megoldása (1)

- Tehát a kérdés, hogy lehet megtalálni Q*-ot!
- Belátható, hogy iteratívan megkereshető (successive approx.) -> sok ismétlés után nem változik

$$Q^*_{t+1}(s,a) = \sum_{s'} T(s,a,s') \left[r(s,a,s') + \gamma \max_{a'} Q^*_t\ (s',a') \right]$$

- Q_0 tetszőlegesen inicializálható!
- Itt a Q egy táblázatként van megadva!

DP

Bellman-egyenlet megoldása (2)

- Tehát a kérdés, hogy lehet megtalálni Q*-ot!

1. Inicializáljuk Q_0-t tetszőlegesen
2. Adott s állapotban kiválasztjuk a-t epszilon-greedyvel
3. Végrehajtjuk a cselekvést. Fogadjuk s'-t, r-t.
4. Alkalmazzuk az alábbi formulát az aktuális átmenetre

$$Q^*_{t+1}(s,a) = (1-\alpha)Q^*_t(s,a) + \alpha(r + \gamma \max_{a'} Q^*_t\,(s',a'))$$

Olyan mint a learning rate

Q-learning

Bellman-egyenlet megoldása (3)

- Tehát a kérdés, hogy lehet megtalálni Q*-ot!

1. Inicializáljuk Q_0-t tetszőlegesen és a-t.
2. Végrehajtjuk a cselekvést. Fogadjuk s'-t, r-t. Választunk a'-t epsilon-greedy alapján.
3. Alkalmazzuk az alábbi formulát az aktuális átmenetre
4. a=a', majd ismétlés 2-től

Sarsa-learning

$$Q^*_{t+1}(s,a) = (1-\alpha)Q^*_t(s,a) + \alpha(r + \gamma Q^*_t\,(s',a'))$$

Bellman-egyenlet megoldása

- A DP megoldás gyors és egzakt lehet, de sajnos igényli a T ismeretét (és a reward függvényt is)
- A Q-learning és a Sarsa-learning módszerek sehol nem igénylik a T-t, hiszen nincs benne a frissítési formulában, sem pedig a policy kiszámításában!

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a)$$

DQN

DQN

- Deep Q-network
- Q-learningen alapul
- A Q fv-t egy neurális háló becsli
- Ezért kell egy loss függvény a Q-hoz:

$$L_\theta = \sum_i \left(r_i + \gamma \max_{a'} Q_{\tilde{\theta}}(s,' a') - Q_\theta(s,a) \right)^2$$

DQN

- A probléma az instabilitás, melyre megoldás:
  - experience replay
  - delayed update
- DQN paper

Demó

Lásd notebook …