MEGERŐSÍTÉSES TANULÁS, MÉLY Q-HÁLÓK

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

Megerősítéses tanulás

- A megerősítéses tanulás (reinforcement learning) egy speciális tanulási feladattípus
  - kilóg a hagyományos felügyelt/felügyeletlen gépi tanulási feladatok közül
  - Saját módszertana és irodalma van
- A gépnek nem egyetlen döntést, hanem döntések sorozatát kell meghozni
  - Az elemi döntések helyességére nem kap visszajelzést, csak a cselekvéssorozat végén kap jutalmat/büntetést
  - A formális modell bevezethet „jutalmat” az egyes lépésekért, de a lényeg ekkor sem az elemi lépések, hanem a lépéssorozat helyessége
- A fő cél tehát egy stratégia („policy”) tanulása
  - A gépnek próbálkoznia kell, hogy megtalálja a helyes lépéssorozatot
  - Járt út vagy járatlan?
  - Melyik cselekvés jó, milyen környezetben?
  - A múltbeli tudást hogyan tudja felhasználni?
- Alkalmazási példák: sakk, Atari játék, Go, Starcraft …

A megerősítéses tanulás változatai

- **Reflexszerű ágens**

Olyan stratégiát tanul, amely közvetlenül képezi le az állapotokat cselekvésekre, nem modellez hasznosságot. Az egyszerű reflexszerű csak az aktuális állapot alapján dönt, nem veszi figyelembe a korábbi állapotokat.

- **Hasznosságalapú ágens**

Az állapotokra alapozott hasznosságfüggvényt tanul, és az alapján választja ki azokat a cselekvéseit, amelyekkel maximálja az elérhető hasznosság értékét. Tehát állapotokhoz, vagy állapotok sorozatához rendeli a hasznosságot.

- **Q-tanuló**

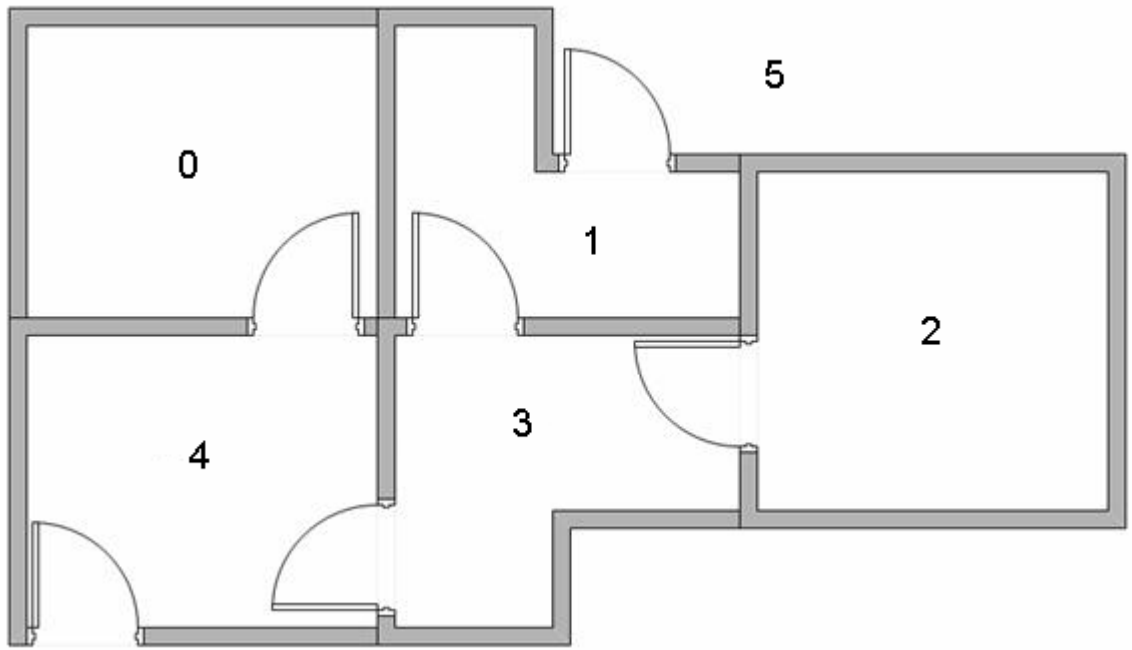
Egy függvényt – Q-függvényt (quality) – tanul, amely a várható hasznosságát becsüli meg egy adott helyzetben egy adott cselekvésnek. Tehát nem állapotokhoz, hanem állapot+cselekvés párokhoz rendel hasznosságot.

Q-tanulás

- Az egyik legegyszerűbb megerősítéses tanulási algoritmus
- Feltesszük, hogy a modellnek állapotai (states) vannak, és minden állapotban vannak lehetséges elemi cselekvések (action)
- Minden állapothoz és cselekvéshez tartozik egy jutalom (reward). Ez jutalmazhatja az elemi lépéseket is, de lehet, hogy csak a kívánt végállapot elérése esetén jutalmaz
- A Q-learning olyan stratégiára törekszik, amely az összes lépésre kapott jutalmak összegét (várható értékét) igyekszik maximalizálni
- A tanulás során összegyűjtött tudást egy ún. Q-táblában tároljuk
- A tanuláshoz nyilván próbálkozni kell lépéssorozatokkal. Ez történhet véletlenszerűen (exploring), vagy az eddig megtanult stratégiát követve (exploiting). Az exploiting/exploring lépések aránya is fontos lehet a módszer konvergenciájához (de ebbe itt nem megyünk bele)

Példa Q-tanulásra

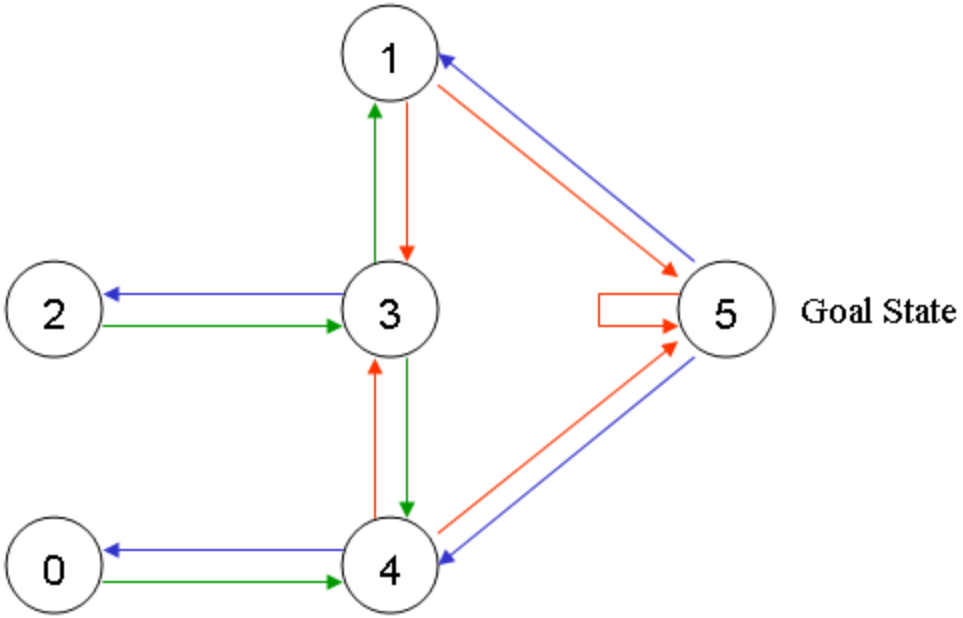
- Feladat: a házból kijutni minél gyorsabban (bármelyik szobából indulva)



Forrás: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

A feladat reprezentálása

- A szobákat állapotoknak, a cselekvések éleknek megfeleltetve a feladatot gráfként rajzolhatjuk fel



- Minden állapot-cselekvés párhoz tartozni fog egy jutalom
- Az 5-ös csúcsba mutató élekhez 100-as jutalmat rendelünk (eléri a célt)
- A többi élhez 0-át (nem tudjuk a cselekvés hasznosságát)
- Az 5-5 hurokél (technikai okból fog kelleni) jutalma szintén 100 lesz

A Q és R-táblák

- A kezdeti R jutalommátrix tehát (minden állapot-cselekvés párra):
  - -1 kódolja a hiányzó éleket

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Ezen felül létrehozunk egy Q táblát is, amelyben az adott állapotban adott akcióhoz tartozó becsült hasznosságot fogjuk tárolni.
  - Kezdetben Q minden eleme 0, (a tanulás célja Q helyes értékeinek megtalálása)

A Q-tanulási algoritmus

Válasszunk egy alfa értéket (0 és 1 közt), valamint töltsük ki a jutalmak R mátrixát.

Inicializáljuk a Q mátrixot nullákkal.

Minden tanulási epizódra:

Válasszunk kezdőállapotot véletlenszerűen.

Do While nem értünk a célállapotba

Az adott állapotban lehetséges akciók közül válasszunk véletlenszerűen.

Jelölje „next state” azt az állapotot, ahova a választott akció vinne

Keressünk meg Q maximumát a next state állapotban a ott lehetséges akciókra nézve

Frissítsük Q aktuális állapothoz és akcióhoz tartozó értékét:

Q(state, action) = R(state, action)+Alfa*Max[Q(next state, all actions)]

Menjünk át a „next state” állapotba.

End Do

End For

A Q-tanulási algoritmus

- Tanulási epizód: amíg egy véletlenszerű állapotból eljutunk a célállapotba; a sikeres tanuláshoz sok tanulási epizód kell
- Az algoritmus lényege az update szabály:
  - Q(state, action) = R(state, action) + Alfa* Max[Q(next state, all actions)]
- Az alfa „learning rate” hatása
  - 0 körüli érték: az aktuális jutalom a fontosabb, mint a hosszú távon várható jutalom (ezt becsüli Q)
  - 1 közeli érték: az aktuális jutalom helyett inkább a hosszú távú haszonra fókuszálunk
- Betanulás után a kiértékeléshez ugyanezt azalgoritmus használhatjuk, de
  - A Q mátrixot rögzítjük (nincs frissítés)
  - Az új állapotot nem véletlenszerűen választjuk, hanem a maximális Q érték alapján (exploring helyett exploiting-ot végzünk)

Példa tanulási epizódra - 1

- Alfa = 0.8, az 1. szobából kezdünk, R adott, Q üres

$$R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{\text{Action}} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array} \quad \text{State}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- R alapján az 1. szobából a 3. és az 5. szobába lehet menni. Tegyük fel, hogy a véletlenszerű választás az 5. szobába visz minket. Ekkor
  - Q(1, 5) = R(1, 5) + 0.8 * Max[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0.8 * 0 = 100
- Az új Q:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Példa tanulási epizódra - 2

- Alfa=0.8, a 3. szobából kezdünk, majd véletlenül az 1. szobába, onnan pedig az 5. szobába mentünk

$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Az első, majd a második lépésre:
  - Q(3, 1) = R(3, 1) + 0.8 * Max[Q(1, 3), Q(1, 5)] = 0 + 0.8 * Max(0, 100) = 80
  - Q(1, 5) = R(1, 5) + 0.8 * Max[Q(5,1), Q(5,1), Q(5, 5)] = 100 + 0.8 * 0 = 100
- Az új Q:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-tanulási példák

- Jó sok próbálkozás (tanulási epizód) után ezt kapjuk:

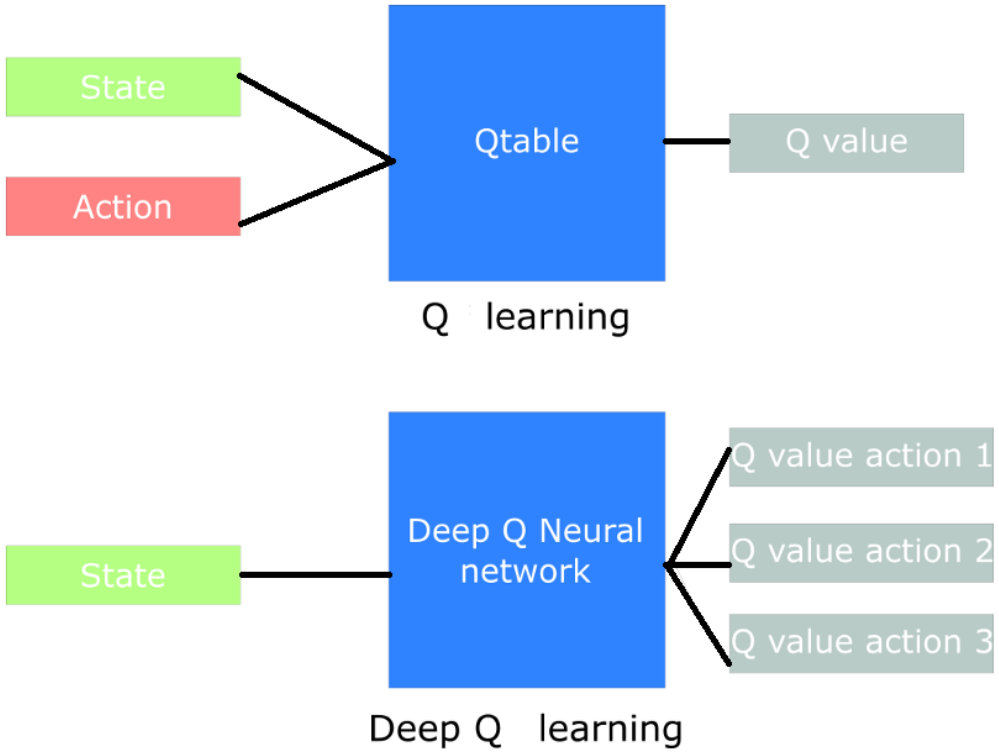
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

- Ezt lenormalizálhatjuk, hogy a legnagyobb érték 100 legyen:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

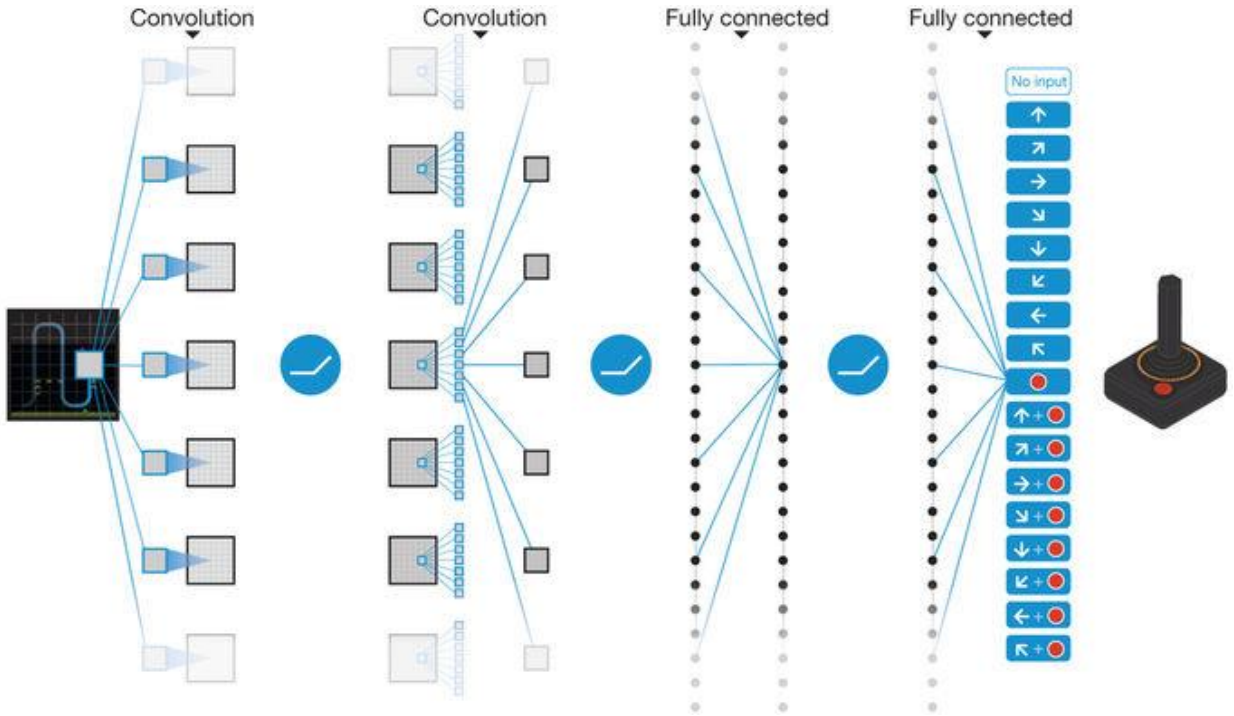
Mély Q-tanulás

- A Q hasznosságot egy egyszerű táblázat helyett egy neuronháló fogja tanulni
- A háló inputja az aktuális állapot, ez egyes kimenő neuronjai pedig a lehetséges akcióknak felelnek meg, azok hasznosságát becsülik meg:



Mély Q-tanulás

- Például ezzel a módszerrel tanították meg a gépet videójátékokat játszani
  - Input: az aktuális képőrnyőkép
  - Output: a lehetséges joystick-mozdulatok hasznossága
  - Háló: egy mély konvolúciós neuronháló



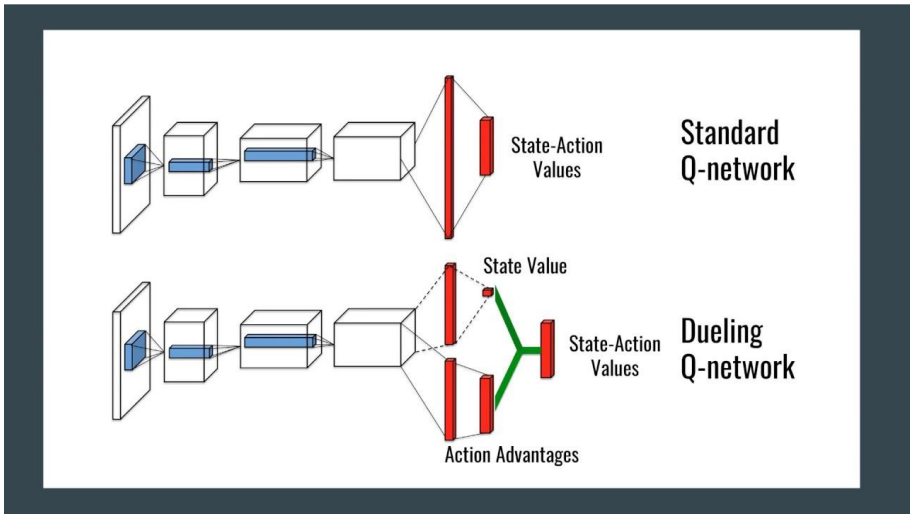
- Forrás: Human-level control through deep reinforcement learning (Nature)

Double Q-learning

- A Deep Q-learning egyik továbbfejlesztése
- Motiváció: a tanulás elején még semmit sem tudunk (a táblázatos modellnél a Q-tábla teljesen üres)
  - A Max[Q(next state, all actions)] függvény a tanulás elején félrevezető értékeket ad, ezért a tanulás nehezen indul be
- Egy helyett 2 neuronhálót tanítunk
  - Az első háló azt tanulja, hogy melyik állapotot érdemes választani (a maximális értéket adó helyett)
  - A második háló pedig az ehhez tartozó Q értéket becsüli
- A tapasztalatok szerint stabilabb tanulást, gyorsabb konvergenciát eredményez

Dueling Q-learning

- A deep Q-learning egy másik továbbfejlesztése
- Szintén 2 hálóra szedi szét az eredeti hálót, de kicsit más koncepció alapján
  - Az egyik háló azt becsüli, hogy mennyire jó egy adott állapotban lenni (state value), a másik pedig, hogy melyik akciót mennyire érdemes választani (state action)
  - Bár külön becsüli ezeket, később egy magasabb szinten összekombinálja

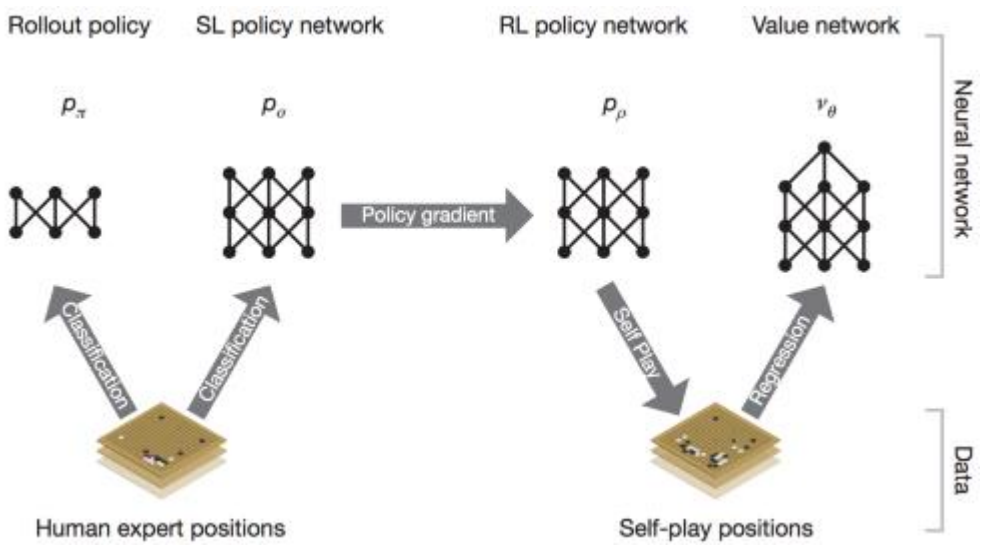


- Miért hatékonyabb ez a szétbontott becslés?
  - Bizonyos állapotok magukban is „jók”, bármit is lépünk (pl. nincs ellenség a közelben), illetve „rosszak” (mindegy, mit lépünk, meghalunk)
  - Az ilyen esetek kezelését könnyebben meg tudja tanulni a dueling háló

Alpha Go

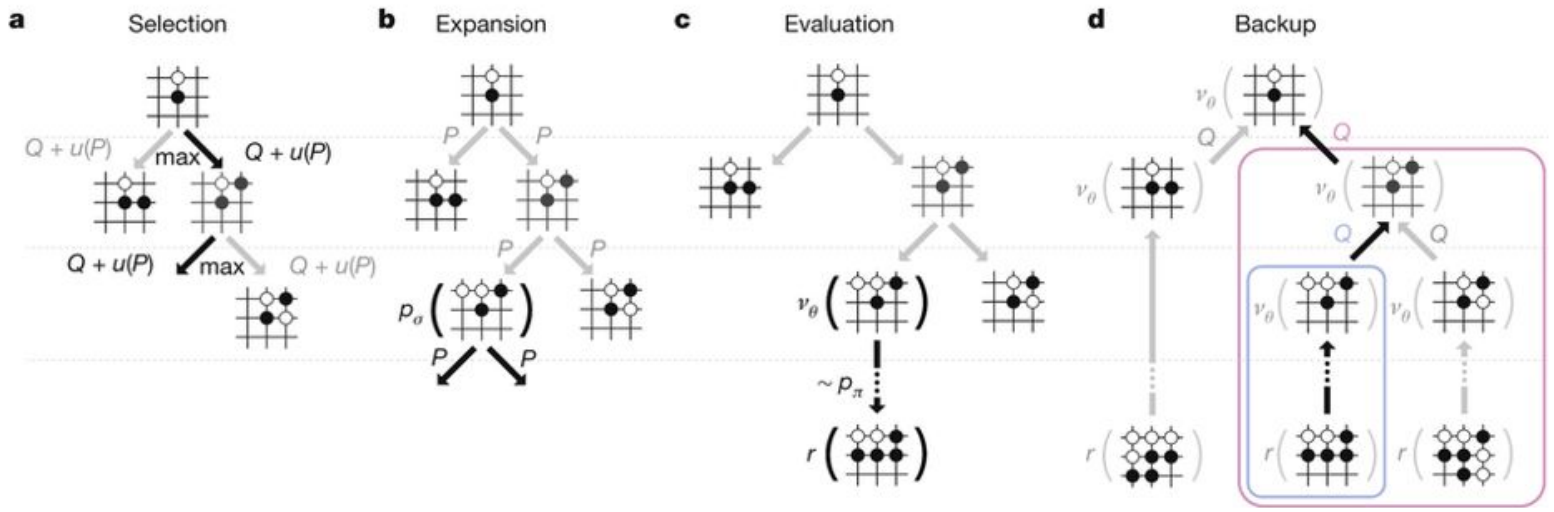
- A Google fejlesztése
- 2015-ben legyőzte a go világbajnokot
- Két mély hálót tartalmaz:
  - policy network: megbecsüli, hogy mit érdemes lépni
  - value network: megbecsüli a lépés hasznosságát
  - (mint az action és value hálózatok az előző dián)

- Először felügyelt módon (SL), 30 millió emberi lépésen betanítottak két hálót (bal oldali ábra)
- Ezután innen indítva, self-play módban tanították be a reinforce-ment learning (RL) policy és value hálóit



Alpha Go

- A hosszú távú stratégia megtalálásához a neuronhálók kombinálva vannak egy fabejáró keresőalgoritmussal
- E célra a Monte Carlo Tree Search (MCTS) nevű algoritmust használták:



a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action. (Mastering the game of Go with deep neural networks and tree search)

AlphaGo Zero

- Hasonló a „sima” Alpha Go-hoz, de már nem igényel ember által lejátszott partikon való „előtanítást” a betanuláshoz
- Önmaga ellen játszva tanul
- Módosítások az Alpha Go-hoz képest:
  - Egyszerűsítettek a tábla reprezentációján
  - A policy és value network egyesítve lett (dueling)
  - Konvolúciós háló helyett konvolúciós+reziduális háló
  - Egyszerűsített fakeresés az egyesített háló segítségével (nincs „rollout”)
- Az Alpha Go olyan új értelmes lépési stratégiákat is képes volt felfedezni, amelyeket az emberi játékosok nem is ismertek korábban

KÖSZÖNÖM A FIGYELMET!

A tananyag az EFOP-3.5.1-16-2017-00004 pályázat támogatásával készült.