

# Üzleti Intelligencia

## 5. Előadás: $Q$ -tanulás

Kuknyó Dániel  
Budapesti Gazdasági Egyetem

2023/24  
1.félév

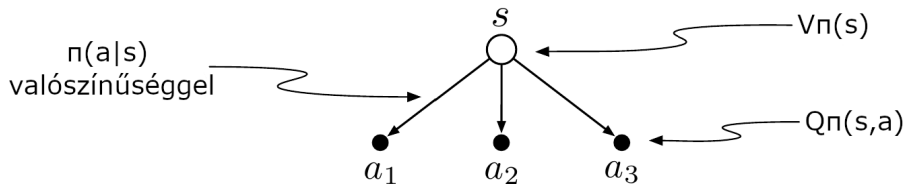
- 1 Bevezetés
- 2 Függvény illesztés
- 3 Gradiens ereszkedés
- 4 Neurális hálózatok
- 5 Mély  $Q$ -tanulás

- 1 Bevezetés
- 2 Függvény illesztés
- 3 Gradiens ereszkedés
- 4 Neurális hálózatok
- 5 Mély  $Q$ -tanulás

# Bevezetés a $Q$ -tanulásba

A megerősítéssel tanulásban a  $Q_\pi(s, a)$  minőségfüggvény megadja, hogy mennyire jövedelmező az ügynöknek, ha  $s$  állapotban állva,  $a$  cselekvést végrehajtva majd onnan  $\pi$  politikát követve mekkora a várható hozam.

Ha adott minden állapotra és cselekvésre az optimális  $Q(s, a)$  érték, onnan levezethető a  $\pi_*$  optimális politika, amelynek várható hozama maximális.



$$Q_\pi(s, a) = E_\pi [G_t | s_t = s, a_t = a] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right]$$

# Q-tanulás: politikafüggetlen TD irányítás

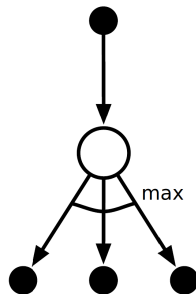
A megerősítéses tanulásban az egyik nagy áttörést egy politikafüggetlen TD algoritmus kifejlesztése hozta el.

Ebben az esetben a becsült állapot-cselekvés minőség függvény,  $Q$ , közvetlen becsüli meg  $q_*$  optimális állapot-cselekvés minőség függvényt a követett politikától teljesen függetlenül.

## Q-tanulás

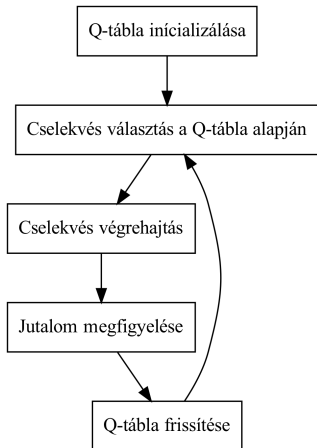
$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Az egyetlen különbség a SARSA algoritmusától, hogy a referencia a következő állapotból elérhető legjobb minőségű cselekvés értéke:  $\max_a Q(s_{t+1}, a)$ .



# $Q$ -tanulás eljárása

## A $Q$ -tanulás folyamata



## Példa $Q$ -táblára

	$a_0$	$a_1$	$a_2$	$a_3$
$s_0$	0.76	0.41	0.92	-0.14
$s_1$	-0.65	0.31	-0.07	0.55
$s_2$	0.23	-0.99	0.67	-0.43
$s_3$	-0.81	0.79	-0.58	0.17
$s_4$	0.62	-0.28	0.96	-0.72
$s_5$	-0.36	0.08	-0.51	0.64

A  $Q$ -tábla tárolja el a  $Q(s, a)$  értékeket minden  $s \in S$  és  $a \in A$  párosra.

---

**Algoritmus 1:**  $Q$ -tanulás algoritmus  $\pi \approx \pi_*$  megbecslésére

---

**Input:**  $\alpha$  tanulási sebesség;  $\varepsilon > 0$  hibahatár

```
 $Q(s, a) \leftarrow \text{random}()$  for  $s \in S, a \in A$ ;      /*  $Q$  értékek inicializálása */  
 $Q(s_T, \cdot) \leftarrow 0$ ;                        /* Terminális állapot 0-ra állítása */  
for  $i = 0 \rightarrow \max_i$  do  
     $s \leftarrow s_0$ ;                            /*  $s$  inicializálása */  
    while  $s \neq s_T$  do  
         $a \leftarrow \pi(s)$ ;      /* Cselekvés választása  $\pi$  szerint, pl.  $\varepsilon$ -mohó */  
         $a$  végrehajtása,  $r, s'$  megfigyelése;  
         $Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$ ;  
         $s \leftarrow s'$ ;                            /*  $s$  frissítése */  
    end  
end
```

---

A politikát a  $Q$ -tábla határozza meg.

## Dupla Q-tanulás

A kettős tanulás ötlete természetesen kiterjed a teljes MDP algoritmusaira. A  $Q$ -tanulásban a becsült  $Q$ -értékek torzítottak lehetnek, ha alacsony a minta számossága, vagy zaj van a rendszerben. Egy módja a  $Q$ -tanulás regularizálásának, ha egy helyet **két  $Q$ -táblát tart nyilván az algoritmus**,  $Q_1$ -et és  $Q_2$ -t.

A  $Q$ -tanulással analóg dupla  $Q$ -tanulás nevű kettős tanulási algoritmus két részre osztja az időlépéseket, **minden lépésnél egy érmét feldobva**. Ha az érme fejre esik, a frissítés a következő:

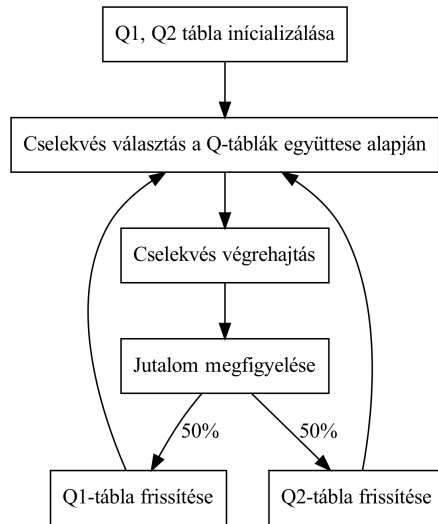
$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha \left[ r + \gamma Q_2 \left( s', \underset{a'}{\operatorname{argmax}} Q_1(s', a') \right) - Q_1(s, a) \right]$$

Ha az érme pedig írásra esik, akkor ugyanez a frissítés  $Q_1$  és  $Q_2$  felcserélésével történik, így  $Q_2$  frissül. A két közelítő értékfüggvényt teljesen szimmetrikusan kezeli az algoritmus. Például egy  $\varepsilon$ -mohó politika a dupla tanulás esetében az egyes cselekvési értékbecslések **átlagára vagy összegére épülhet**.



# Dupla $Q$ -tanulás eljárása

Mivel két  $Q$  táblát kell nyilván tartania, az algoritmus memóriaigénye megkétszereződik. A becsült értékek viszont jóval torzítatlanabbak lesznek, mint az egyszeres  $Q$  tanulás esetében. A lépésenkénti számításigény nem növekszik az extra  $Q$ -tábla bevonásával.





- 1 Bevezetés
- 2 Függvény illesztés
- 3 Gradiens ereszkedés
- 4 Neurális hálózatok
- 5 Mély  $Q$ -tanulás

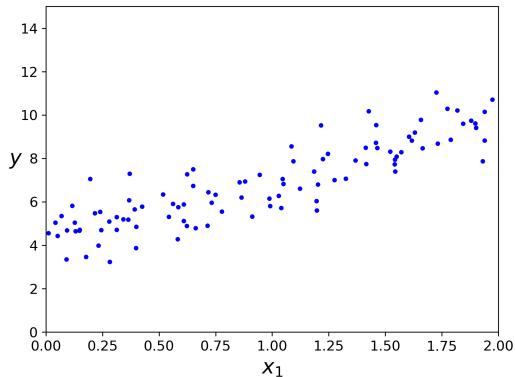
# Függvény illesztés alapjai

A függvény illesztés eljárása szerint valamely  $x$  **független változóból** vett minta alapján szeretnénk előre jelezni egy  $y$  **függő változó** értékét azért, hogy feltárja az adatpontok közötti mintázatokat.

Két eljárása ismert:

- **Regresszió:** tárgya egy folytonos változó
- **Osztályozás:** tárgya egy diszkrét változó

A függvény illesztés eredménye a **modell**.



# Függvény illesztés alapjai

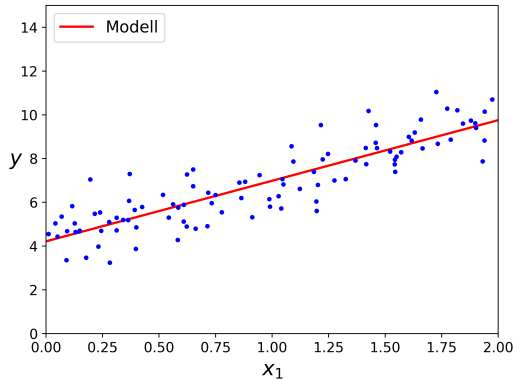
Az illesztett modell alakját és viselkedését a **paraméterei** határozzák meg, amelyek együtthatókként viselkednek a modell egyenletében. A lineáris modell egyenlete:

$$y_i = \theta_0 + \theta_1 x_i + \varepsilon_i$$

Ahol:

- $\theta_0$ : az  $y$  tengely metszéspontja, vagy eltolás
- $\theta_1$ : az egyenes meredeksége
- $\varepsilon$ : a véletlen hiba, amit a modell nem tud előre jelezni

$\theta = [\theta_0, \theta_1, \dots, \theta_n]$  a paraméterek vektora.

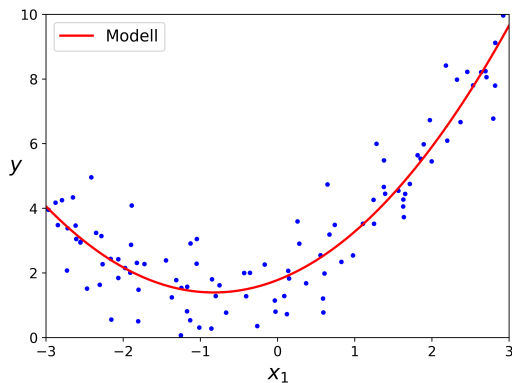


# Függvény illesztés több paraméterrel

Függvényt lehetséges nemlineáris adatokra is illeszteni. Ebben a példában a minta adatpontok kvadratikusak, **nem írhatók le egy lineáris egyenlettel**. A modellnek ebben az esetben 3 paramétere van:  $\theta_0$ ,  $\theta_1$  és  $\theta_2$ . Az illesztett modell egyenlete:

$$y = 1.78 + 0.93x + 0.56x^2$$

Tehát ebben az esetben  $\theta_0 = 1.78$ ,  $\theta_1 = 0.93$  és  $\theta_2 = 0.56$ .



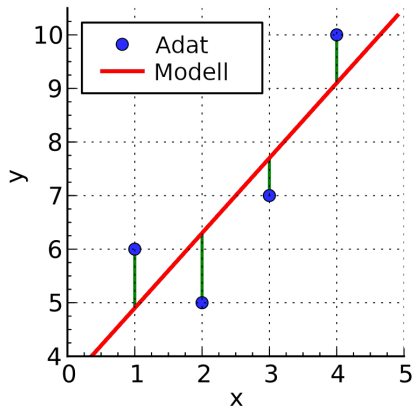
- 1 Bevezetés
- 2 Függvény illesztés
- 3 Gradiens ereszkedés
- 4 Neurális hálózatok
- 5 Mély  $Q$ -tanulás

# A költségfüggvény

A modell illesztő algoritmusok mindegyike úgy találja meg az optimális függvényt, hogy valamilyen költségfüggvényt minimalizál. A leggyakoribb ilyen költségfüggvény az **átlagos négyzetes hiba**:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - f(x)_i)^2$$

- $y_i$ : Megfigyelt adatpont
- $f(x)_i$ : Modell által adott becslés





# A gradiens

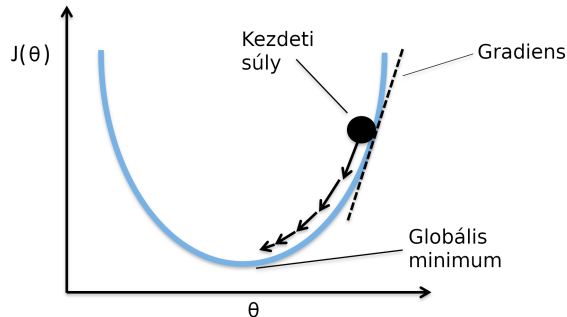
A függvény illesztés célja, hogy megtalálja azt a modellt, ami a legjobban illeszkedik az adatpontokra, tehát **minimalizálja a költségfüggvényt (MSE)**.

## Gradiens

Olyan vektor, amely megmutatja hogyan változik a függvény, és megadja a legnagyobb változás irányát minden dimenzióban.

$$df = \nabla f * dx$$

A gradiens segítségével meg lehet határozni, merre és mennyivel érdemes változtatni a paramétereket a célfüggvény értékének csökkentése érdekében.

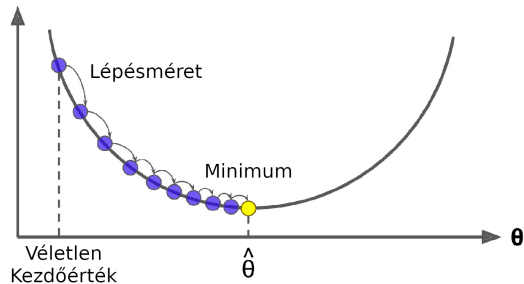


# Gradiens ereszkedés

A gradiens ereszkedés egy iteratív minimalizálási algoritmus egy tetszőleges függvény lokális minimum helyének megtalálására. Az algoritmus lépésről lépésre mozog a függvény értékének csökkentése érdekében.

## Tanulási sebesség ( $\alpha$ vagy $\eta$ )

A tanulási sebesség meghatározza, mennyire nagy lépéseket tesz a gradiens ereszkedés az optimalizációs folyamat során.

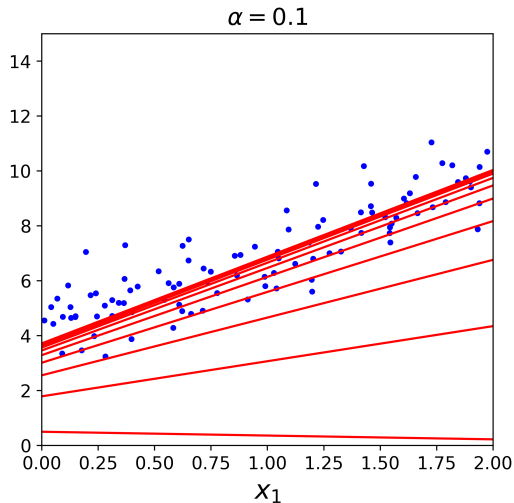


# Gradiens ereszkedés

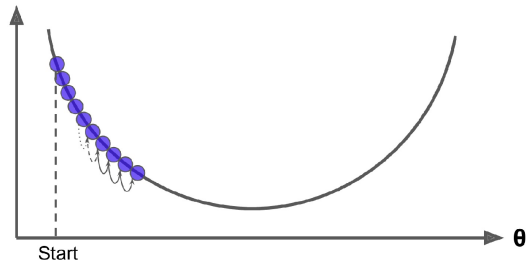
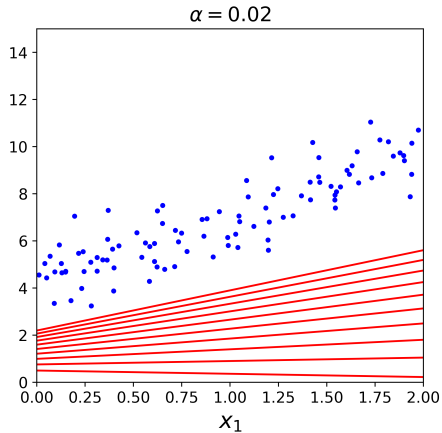
A gradiens ereszkedés paraméter frissítése:

$$\theta' \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$$

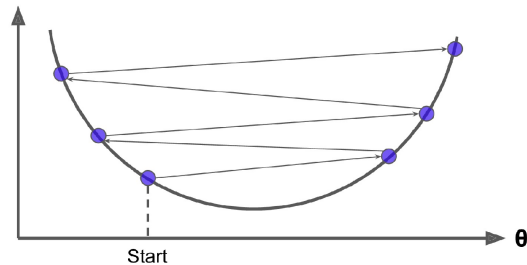
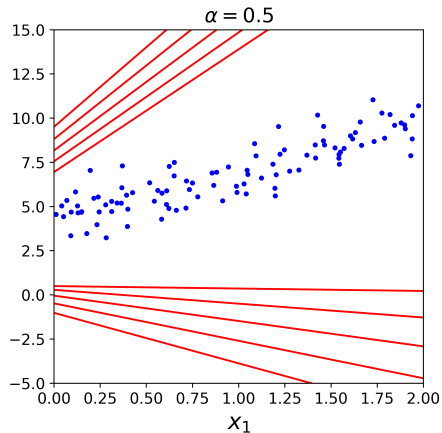
- $\theta = [\theta_0, \theta_1, \dots, \theta_n]$ : Paraméterek vektora
- $\alpha \in [0, 1]$ : Tanulási sebesség
- $\nabla_{\theta}$ : Költségfüggvény gradiense ( $\theta$  szerinti derivált)
- $J(\theta)$ : Költségfüggvény  $\theta$  szerint (pl. MSE)



# Túl alacsony tanulási sebesség



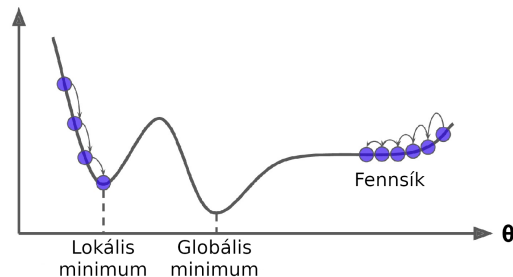
# Túl magas tanulási sebesség



# Egyéb problémák

A gradiens ereszkedés nem garantálja, hogy elér egy globális minimumot, hiszen a gradiens operátora csak lokális változásokat vesz figyelembe. Ebből adódóan az **algoritmus beragadhat egy lokális minimumon**.

Vannak esetek, amikor a gradiens nem meghatározható (szaturál), például amikor **elér egy fennsíkot a függvényen**. Ebben az esetben a futás vagy nagyon lassú lesz, vagy nem halad semerre.

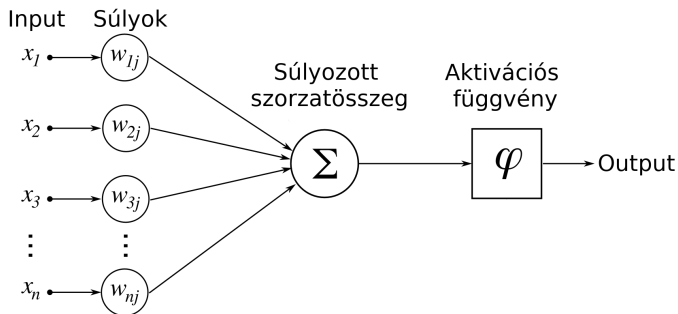


- 1 Bevezetés
- 2 Függvény illesztés
- 3 Gradiens ereszkedés
- 4 Neurális hálózatok**
- 5 Mély  $Q$ -tanulás

# A neuron

A neurális hálózatot **neuronok** **összessége** alkotja. Az egyes neuronok egyszerű elemi műveleteket végeznek. A neuronnak több inputja (kapcsolata) van, és mindegyikhez egy súly tartozik.

A neuron kiszámítja az inputjainak a súlyozott összegét ( $z = x_1w_1 + x_2w_2 + \dots + x_nw_n$ ), majd ezt az értéket behelyettesíti egy aktivációs függvénybe ( $h = \varphi(z)$ ).





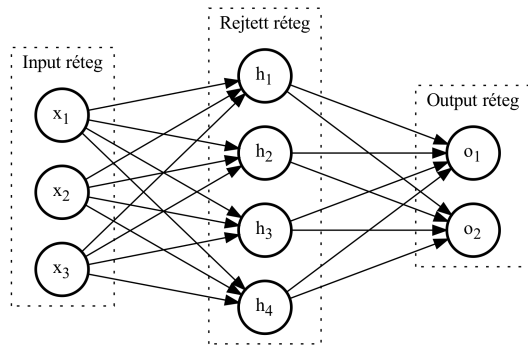
# Többrétegű hálózatok

Ebben az esetben a neuronok rétegekben foglalnak helyet. A kapcsolataik az előző réteg kimeneteivel állnak összeköttetésben. A legelső réteg neuronjai a bemeneti adattal állnak összeköttetésben. Minden bemeneti jellemzőhöz egy neuron tartozik.

## Teljesen becsatolt neuronréteg kimenete

$$h_{W,b}(X) = \varphi(XW + b)$$

- $X$ : Input jellemzők mátrixa
- $W$ : Kapcsolati súlyok mátrixa
- $b$ : Torzítások vektora
- $\varphi$ : Aktivációs függvény



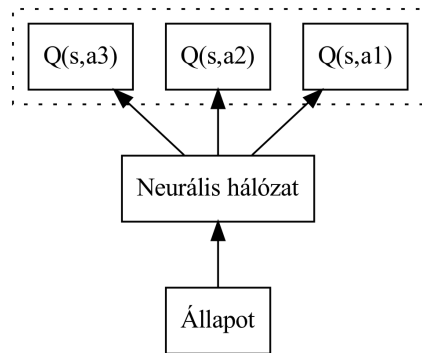
- 1 Bevezetés
- 2 Függvény illesztés
- 3 Gradiens ereszkedés
- 4 Neurális hálózatok
- 5 Mély  $Q$ -tanulás

# A $Q$ -hálózat (DQN)

A  $Q$ -hálózat egy természetes kiterjesztése a hagyományos  $Q$ -tanulásnak. A naív  $Q$ -hálózat inputja a **környezetet leíró változók** vektora vagy mátrixa, és az outputja pedig az ügynök számára elérhető **cselekvések  $Q(s, a)$  értéke** minden  $a_1, a_2, \dots, a_n$  cselekvéshez tartozóan.

A cselekvés választáshoz az ügynök kiválasztja a legnagyobb becsült  $Q$  értéket, és az ahhoz tartozó cselekvést fogja végrehajtani.

Miután először bemutatták 2013-ban, hamar kiderült hogy a naív  $Q$ -hálózat nem elég stabil, gyakran eredményez torz politikát a túltanulás miatt.

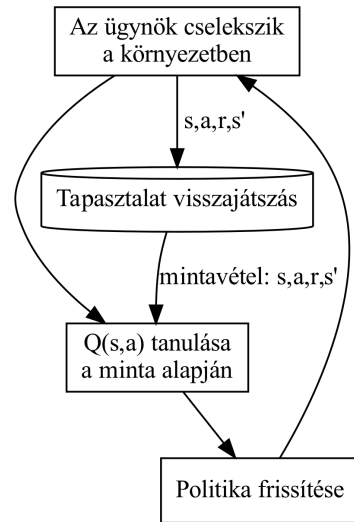


# Tapasztalat visszajátszás

A tapasztalat visszajátszás az egyik változtatás, ami a  $Q$ -hálózat problémáján hivatott segíteni.

A tapasztalat memória ( $D_{replay}$ )  $[s, a, r, s']$  **négyeseket tartalmaz**. Minden alkalommal amikor az ügynök cselekszik, az általa tapasztalt  $s_t, a_t, r_t, s_{t+1}$  elmentődik a tapasztalat memóriába.

Amikor tanulásra kerül a sor, az ügynök **véletlen és rendezetlen mintát** (miniköteget) kap a tapasztalat memóriából, melynek számossága megegyezik a kötegmérettel. Eszerint fogja kiszámolni a költségfüggvényt majd frissíteni a neurális hálózat paramétereit.



# Költségfüggvény és frissítési szabály

## A DQN költségfüggvénye

$$J(\theta) = E_{s,a,r,s' \sim D} \left[ \left( r + \gamma \max_{a'} Q_{\theta}(s', a') - Q_{\theta}(s, a) \right)^2 \right]$$

A költségfüggvény a **négyzetes Bellman hiba várható értéke**. Ahol  $s, a, r, s'$  a  $D$  tapasztalat visszajátszásból vett minta.  $Q_{\theta}$  megkeresi az  $s'$  következő állapothoz tartozó legnagyobb cselekvés értéket (**célérték**), és lekérdezi  $s$  aktuális állapothoz és  $a$  aktuális cselekvéshez tartozó értéket.

## A DQN paraméter frissítése

$$\theta' \leftarrow \theta + \alpha J(\theta) \nabla_{\theta} Q_{\theta}(s, a)$$

Ahol  $\theta$  a paramétervektor,  $\alpha$  a tanulási sebesség,  $J(\theta)$  a költségfüggvény  $\theta$  szerint és  $\nabla_{\theta}$  a költségfüggvény gradiense  $\theta$  szerint.