Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Knoll Zsolt, Németh Marcell, Szűcs Gábor

MÉLY MEGERŐSÍTÉSES TANULÁS

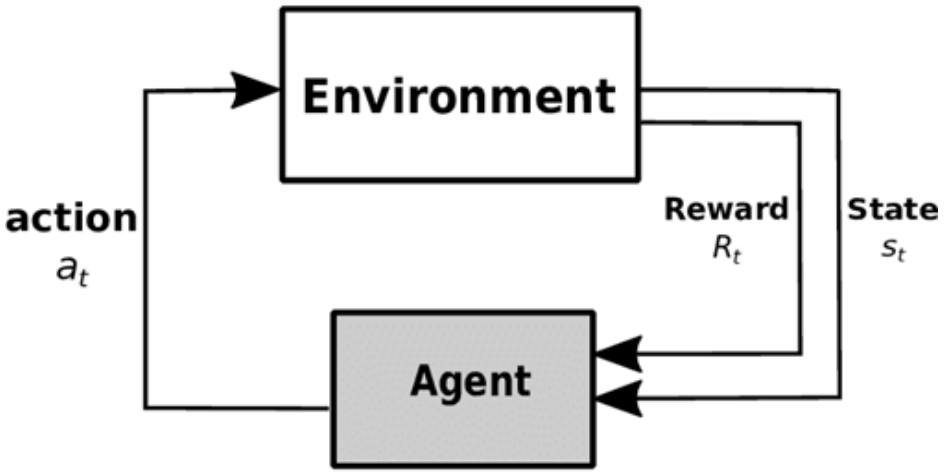
A tanulmány a Magyar Nemzeti Bank és a Budapesti Műszaki és Gazdaságtudományi Egyetem között létrejött Együttműködés keretében és finanszírozásával készült a Digitalizáció, mesterséges intelligencia és adatkorszak műhelyben.

BUDAPEST, 2021

Tartalomjegyzék

Vezetői összefoglaló

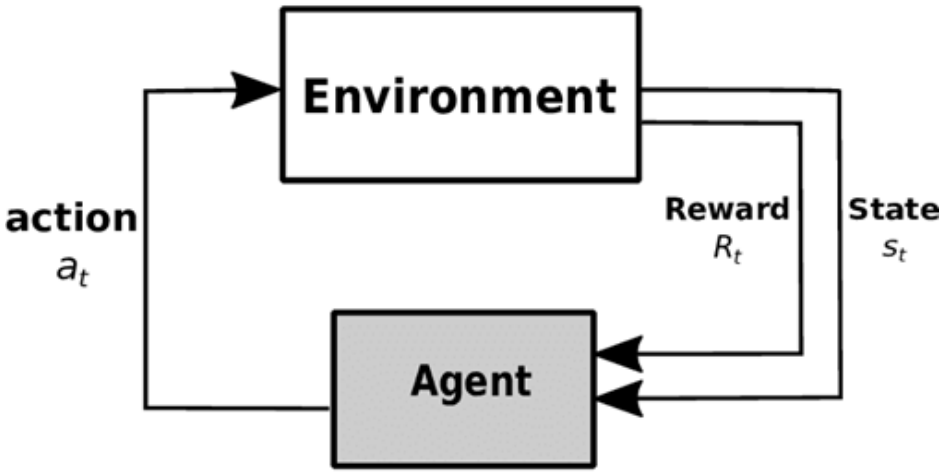
Jelen tanulmányunkban bemutatjuk a mesterséges intelligencia egy fontos ágát, a megerősítéses tanulást, valamint ennek a legnagyobb hajtóerőt biztosító mélytanuláshoz kapcsolódó részét. A neurális hálózatokon alapuló mély megerősítéses tanulásnak különböző módszerei, algoritmusai fontos alapköveket biztosítanak ezen a területen. Így ezek részletes bemutatása után az alkalmazási lehetőségekre mutatunk rá. A pénzügyi és gazdasági alkalmazások között megtalálható az algoritmikus tőzsdei kereskedés, a portfólió menedzsment, a likviditási stratégia és a különböző aukciók, így az utolsó fejezetünkben bemutatjuk hogyan használható ezeknél a mély megerősítéses tanulás.



1 Megerősítéses tanulás

1.1 Alapfogalmak

A gépi tanulásnak három alapvető paradigmája van: felügyelt tanulás, felügyelet nélküli tanulás és megerősítéses tanulás [18]. Felügyelt tanulás esetén a gépi tanulás célja, hogy megtanuljon egy függvényt, amely a bemeneti adatokból meghatározza a kimeneti adatokat, mindezt példákra alapozva. Ezzel ellentétben a felügyelet nélküli tanulás esetében előre nem meghatározott mintákat keresünk az adatokban. Megerősítéses tanulás esetén a gépi tanulási modellünket betesszük egy környezetbe (ahogy ezt az 1. ábrán láthatjuk), amelyben próbákon és hibákon keresztül a modell (az ábrában: agent, így magyar megfelelőjeként az ágens szót is szokták még használni) megtanulja hogyan maximalizálja a jutalmát (reward) az adott környezetben. A t-edik időpontban kapott jutalmat R_t-vel, az állapotot (state) S_t-vel és a cselekvést (action) pedig a_t-vel jelöljük.



1.1 ábra. A megerősítéses tanulás folyamata

Vannak olyan gépi tanuló algoritmusok, amelyek megpróbálnak egy modellt építeni a környezetről a róla begyűjtött információk alapján, és ezen modell alapján tervezik meg a viselkedésüket. A másik csoport a modell nélküli algoritmusok, amelyek egy modell építése nélkül próbálják meghatározni az optimális stratégiát a környezetben. A gépi tanulási modell alapján több kategóriába tudjuk sorolni a megerősítéses tanulási algoritmusokat. A hasznosságalapú modellek (utility-based vagy más néven value-based) az állapotokra alapozott hasznosságfüggvényt tanulják meg, és ennek alapján választják

ki azokat a cselekvéseiket, amelyekkel maximálják az elérhető hasznosság várható értékét. A stratégiaalapú (policy-based) modellek ehelyett megpróbálják meghatározni a cselekvésérték függvényt (action-value function) – vagy más néven Q-függvényt – amely segítségével már egyértelműen meghatározható, hogyan kell az adott állapotban cselekedni, hiszen a Q-függvény valamilyen várható hasznot tulajdonít egy adott állapotban egy adott cselekvésnek. A hasznosságalapú algoritmusoknak a környezet valamilyen modelljével is rendelkezniük kell, egy Q-tanuló algoritmus viszont össze tudja hasonlítani a lehetséges választásait anélkül, hogy tudná, mire vezetnek, így nincs szüksége a környezet modelljére. A cselekvő-kritikus (actor-critic) algoritmusok [21] egyszerre stratégia- és hasznosságalapúak, eltárolják mind a hasznosságfüggvényt, mind a cselekvésérték függvényt és mindkettőt felhasználják a legjobb döntés kiszámításához.

1.2 Markov döntési folyamat és a Bellman-egyenlet

Megerősítéses tanulás esetén a legjobban tanulmányozott eset, amikor a probléma megfogalmazható egy Markov döntési folyamatként [4] (Markov decision process, MDP). MDP esetén a gépi tanulási modell véges számú állapotot (state) képes meglátogatni, ahol *r* értékű jutalmat (reward) kap (negatív szám jelenti a büntetést). Minden *s* állapotból elérhetőek más állapotok *a* tevékenységek (action) végrehajtásával. Minden állapothoz tartozik egy állandóan változó érték, mely azt jelöli, hogy az adott állapotból kiindulva mennyi jutalmat tud a modell összegyűjteni, amíg eléri a környezet végállapotát (end state). Az állapotokban végrehajtandó tevékenységeket egy stratégia (policy) szerint határozzuk meg, amely szintén változhat a tanulási folyamat előrehaladtával. A modell célja, hogy a végrehajtott tevékenységekkel maximalizálja a jutalmát amíg eléri a környezet végállapotát.

A Markov döntési folyamatban található állapotok mindegyike egy-egy Markov állapot. Egy állapotot akkor nevezhetünk Markovnak, hogyha a jövő független a múlttól; így hogyha egy Markov állapotban tartózkodik a modellünk, akkor nem szükséges ismernünk a korábbi állapotok sorozatát, csak a jelenlegi állapotot. Matematikailag, egy s_t állapot (a t-edik időpontban) akkor és csak akkor Markov állapot, hogyha s_t állapot magában hordozza az összes előző állapot információját.

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \dots, s_t) \tag{1.2.1}$$

Egy *s* Markov állapot és egy *s'* utód állapot (successor state) között az átviteli valószínűség (transition probability) meghatározható a következő módon: $P_{ss'} = P(S_{t+1} = s'|S_t = s)$. Ezt eltárolhatjuk egy *P* mátrix formájában, ahol $P_{i,j}$ megegyezik s_i, s_j közötti átviteli valószínűség értékével, így *P* mátrix minden sorának az összege 1 lesz.

Egy Markov jutalmazási folyamat (Markov Reward Process, MRP) egy (*S, P, R, γ*) értéknégyes, ahol *S* a véges állapotok halmaza Markov állapotokkal, *P* az állapot átviteli valószínűségi mátrix, *R* a jutalmazási függvény (reward function) és *γ* a diszkont faktor (discount factor), ahol $\gamma \in [0, 1]$.

A jutalmazási függvény határozza meg, hogy mekkora azonnali jutalmat kapunk, hogyha elérünk egy adott állapotot. A modell célja, hogy maximalizálja G_t-t, ami a teljes diszkontált jutalom (total discounted reward), ez meghatározható *R* és *γ* segítségével a következőképpen:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.2.2}$$

Ez az érték mindenképpen véges, mivel *γ* értéke 0 és 1 között van. Abban az esetben, ha *γ* közel van nullához, akkor a modell az azonnali jutalmat preferálja a jövőbeli jutalmakkal szemben, míg hogyha közel van egyhez, akkor ez éppen fordítva történik.

A Markov döntési folyamat egy (*S, A, P, R, γ*) értékkötös, ahol (*S, P, R, γ*) egy MRP-t alkot, valamint *A* (action) a cselekvések (tevékenységek) véges halmaza. Ezen halmaz elemeinek felhasználásával tud a modell a környezet egyik állapotából egy másikba kerülni. A modell viselkedését, amellyel ezen tevékenységeket választja ki meghatározhatjuk a *π*, policy függvénnyel, ami felírható a következőképpen:

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{1.2.3}$$

A gépi tanulási modell célja, hogy maximalizálja az általa begyűjtött jutalmat. Ennek eléréséhez szükséges meghatároznunk az optimális értékfüggvényt (value function), melyet a Bellman-egyenlettel tehetünk meg [5]. A Bellman-egyenlet azt írja le, hogy egy állapot értéke (hasznosságértéke) felbontható az azonnali jutalom és az őt követő állapot diszkontált értékének összegeként:

$$V(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} \dots |S_t = s] = \tag{1.2.4}$$

$$= \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) | S_t = s]$$

Az értékfüggvény tehát megmutatja, hogy milyen jó a környezet egy adott állapotában tartózkodnia a modellünknek, azonban arról nem mond semmit, hogy melyik tevékenységeket válasszuk ki. Ehhez szükségünk van a cselekvésérték függvényre, mely azt fejezi ki, hogy milyen jó egy adott *s* állapotban egy *a* tevékenységet választanunk, és segítségével később meghatározhatjuk az aktuális állapothoz tartozó legjobb tevékenységet.

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \tag{1.2.5}$$

A legtöbb megerősítéses tanulási algoritmus célja a cselekvésérték függvény meghatározása, és abból az optimális cselekvés kiválasztása bármely állapotban (optimális azt jelenti, hogy a legnagyobb értékkel rendelkező tevékenységet kell választani, ahogy ezt az alábbi képlet mutatja):

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{1.2.6}$$

Ezen függvény segítségével tudjuk meghatározni a maximális jutalmat a π stratégiát követve. Könnyen beláthatjuk, hogyha ismerjük az optimális cselekvésérték függvényt akkor az optimális stratégia meghatározható a következőképpen:

$$\pi_*(a|s) = \begin{cases} 1 \; ha \; a = \; arg\max_{a \in A} q_*(s,a) \\ 0 \; egyébként \end{cases} \tag{1.2.7}$$

Az optimális cselekvésérték függvény rekurzívan felírható a Bellman optimális egyenlettel (Bellman optimality equation) a következő alakban:

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a') \tag{1.2.8}$$

A Bellman optimális egyenlet nem írható fel zárt formában, azonban léteznek iteratív megközelítések, amik alkalmazhatók, ilyenek például a SARSA és Q-tanuló algoritmusok, melyeket a következő fejezetben mutatunk be.

1.3 A megerősítéses tanulás legfontosabb algoritmusai

1.3.1 TD-tanuló

Temporal difference (TD) tanuló (időkülönbség tanuló) [22] a modell nélküli megerősítéses tanulási algoritmusok egy csoportját jelenti, azaz ezen algoritmusok nem alkotnak modellt a környezetükről, hanem mintákat vesznek a belőle, és ez alapján végeznek frissítést.

A legegyszerűbb algoritmusa a TD-tanulónak a TD(0) algoritmus. Ebben az esetben az értékfüggvényt minden mintavétel után frissítjük az azonnali jutalommal és a következő állapot diszkontált értékével. Ez az algoritmus azokban az esetekben hasznos, amikor a környezetnek nincs végállapota, mert az értékfüggvényt minden tevékenység után frissítjük a következő képlet alapján:

$$V(s_t) = V(s_t) + \alpha\big(R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big) \tag{1.3.1}$$

Itt az α paraméter a tanulási ráta, melynek értéke 0 és 1 között helyezkedik el. Ezzel a paraméterrel tudjuk meghatározni, hogy a modell milyen gyorsan tanuljon meg új információkat.

Másik népszerű TD-tanuló algoritmus a TD(1). Ebben az esetben az értékfüggvényt egy epizód után frissítjük, miután elértük a környezet végállapotát. Az értékfüggvény frissítését a következő képlet írja le:

$$V(s_t) = V(s_t) + \alpha\big(G_t - V(s_t)\big) \tag{1.3.2}$$

Abban az esetben, hogyha a környezetnek nincs végállapota, de nem akarjuk az értékfüggvényt minden lépés után frissíteni használhatjuk a TD(λ) algoritmust, ahol λ értéke 0 és 1 között van, és a frissítéshez használt útvonal hosszát határozza meg. Minél magasabb λ értéke, annál hosszabb a frissítéshez használt útvonal hossza.

1.3.2 SARSA

A SARSA (mely a State, Action, Reward, State, Action rövidítése) algoritmus [17] a TD tanulóval ellentétben nem az értékfüggvényt tanulja meg, hanem a policy függvényt. Az algoritmus neve arra utal, hogy a q-függvény értékét az aktuális s_t állapot, a kiválasztott a_t tevékenység, a begyűjtött r jutalom, a következő s_{t+1} állapot és az azt

követő a_{t+1} tevékenység határozza meg. A policy függvény frissítése a következő képlet szerint történik:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1.3.3)$$

Ezen algoritmusnál fontos a q-függvény kezdeti értékeinek meghatározása. A leggyakrabban használt módszer, hogyha a környezet felfedezése során az elsőként kapott jutalmat használjuk fel a q-értékének meghatározásához.

1.3.3 Q-tanuló

A Q-tanuló algoritmus [22] a SARSA-hoz hasonlóan q-függvényt határozza meg. A Q-tanuló algoritmus az off-policy algoritmusok csoportjába tartozik, ami azt jelenti, hogy nem pont ugyanazt a stratégiát tanulja meg, mint ami segítségével kiválasztja a cselekvést. (SARSA pedig az on-policy csoportba tartozik, azaz a stratégia, amit megtanul és amin a kiértékelés történik, az teljesen ugyanaz.)

A Q-tanuló egy úgynevezett q-táblát használ, melynek az állapotok számával megegyező számú sora, és az elvégezhető tevékenységekkel megegyező számú oszlopa van. A tábla egy értéke azt határozza meg, hogy mennyire jó az adott sornak megfelelő állapotban az adott oszlopnak megfelelő tevékenységet választani. A q-tábla értékeinek frissítése a következő képlet alapján történik:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * \left[r_{t+1} + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (1.3.4)$$

ahol a paraméterek ugyanazok, mint a SARSA esetén.

Az új q-érték meghatározása, az előző érték alapján, valamint az azonnali jutalom és a következő állapot maximális értéke alapján történik. A modellnek fel kell fedeznie az összes állapot-cselekvés párost az optimális q-függvény meghatározásához, és figyelnie kell, hogy ne ragadjon bele a modell egy lokális minimumba.

A modell a környezettel való interakciók alapján frissíti a q-tábláját. A modell kétféleképpen választhatja ki a következő tevékenységet: vagy azt a tevékenységet választja, amely az eddig összegyűjtött információ alapján a legjobbnak tűnik, tehát a legnagyobb érték tartozik hozzá a q-táblában, vagy véletlenszerűen választ egy tevékenységet, hogy felfedezze a környezetet. Az első módszert kihasználásnak

(exploiting), míg a másodikat felfedezésnek (exploring) hívjuk. Fontos, hogy a tanulási szakasz elején felfedezzük a környezetet, hogy megtaláljuk a lehető legjobb megoldást.

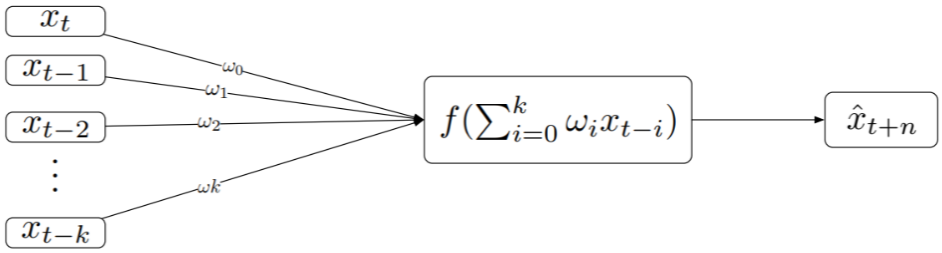
A kihasználás és a felfedezés arányát leggyakrabban az ε-mohó (ε-greedy) stratégia segítségével határozzuk meg. Ebben az esetben az ε paraméter (értéke 0 és 1 között van) határozza meg, hogy az esetek mekkora részében választ a modellünk véletlen tevékenységet, míg 1-ε valószínűséggel választja a legjobbnak tűnő akciót. Az ε értéket kezdetben 1 körülire szokás állítani, ezzel elősegítve, hogy a modell felfedezze a környezetét, és az idő, valamint a tanulási folyamat előrehaladtával exponenciális csökken az értéke egy előre meghatározott minimum szintig, ezzel elérve, hogy a modell maximalizálja a jutalmát.

2 Mélytanulás

Az elmúlt évtized felgyorsult technológiai fejlődése komoly hatással volt a számítógépes erőforrások, kiemelten a GPU-k teljesítményének növekedésére és szélesebb körű való hozzáférhetőségére. Az infrastruktúrális fejlesztések felgyorsították az egyre komplexebb neurális hálózatok fejlődését is, aminek köszönhetően a mélytanuló hálók az elmúlt években forradalmi sikereket értek el számos gyakorlati probléma megoldásában. A következő alfejezetekben nagyvonalakban bemutatásra kerülnek neurális hálózatok alapgondolatai és a leggyakrabban használt alkotóelemeik, fajtáik.

2.1 Többrétegű perceptron és előrecsatolt hálók

A neurális hálózatok alapegysége a perceptron, amely 3 fő részből áll: bemeneti, összegző/számító és a kimenetet reprezentáló csomópontok [18]. Gyakorlati szemléltetés gyanánt (ld. 2.1. ábra) tekintsük az *x* bemeneti vektor elemeit egy makroökonómiai indikátor *k* darab időben eltolt értékeinek, továbbá legyenek a ω súlyok (a $\omega_0 \ldots \omega_k$ élek súlyai az egyes bemenetek súlyait jelentik) kezdetben véletlenszerűen inicializálva.

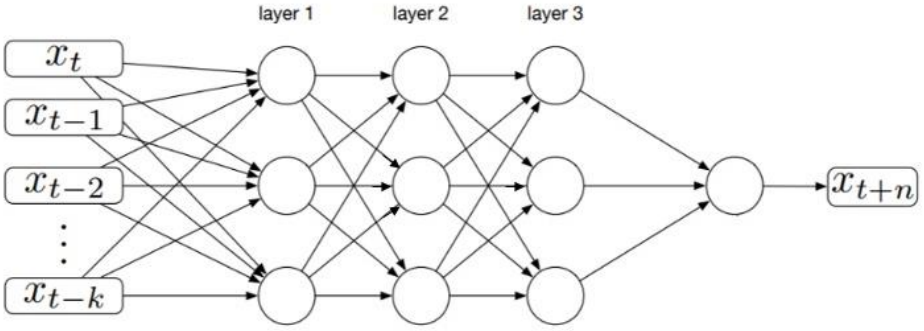


2.1 ábra. A perceptron egy idősoros feladathoz adaptálva [20]

A modell tanítása során az „előre” szakaszban (forward propagation) a bemeneti vektor és a súlyvektor lineáris kombinációja előáll az összegző csomópont bemenetén, majd egy alkalmasan megválasztott ún. aktivációs függvény segítségével transzformálva kapjuk a kimenetet. A kimenetet és a valós címkét (osztályozásnál) vagy folytonos értéket (regressziónál) végső lépésként összehasonlítjuk, majd a predikció jóságát egy alkalmas veszteségfüggvény segítségével visszamérjük és szükség esetén a súlyokat változtatjuk.

Több réteg csatolása esetén ún. teljesen összekötött (fully connected - FC) előrecsatolt hálókról beszélünk, amelyek a keresett függvény komplexebb, magasabb

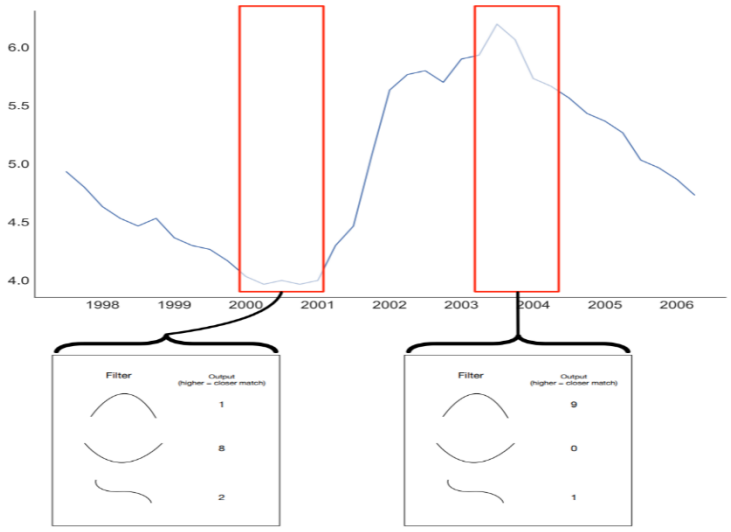
dimenziókban való közelítését teszik lehetővé. Az FC hálók tanítása a hiba visszaterjesztésével (backpropagation) történik, mely során a veszteségfüggvény parciális deriváltjait számoljuk ki az egyes rétegekre, majd a súlyokat a kapott gradiensek irányának megfelelően változtatjuk így közelítve a függvény optimumát.



2.2 ábra. Teljesen összekötött neurális hálózat [20]

2.2 Konvolúciós hálók

A konvolúciós hálózatok felépítésüket tekintve szintén előrecsatolt hálók, amelyek fő alkalmazási területe a mintafelismerésre épülő problémák, kiemelten a képfeldolgozás, de idősoros feladatoknál is használják. Fő elemeik a konvolúciós rétegek, amelyekben a teljes bemeneten végigfutó, mintázatokat kereső csúszóablakok találhatók. A konvolúciós rétegek képesek olyan mintázatok detekciójára, amelyeket előre betanítanak, hogy a pontosságot maximalizálják, ahogy az alábbi ábrán, egy idősorban való mintázat osztályozásnál is látszik.

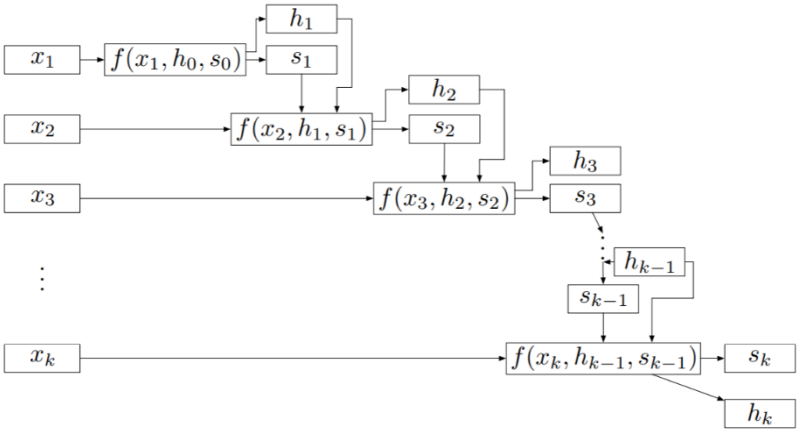


2.3 ábra. Idősorban való mintázat osztályozása [20]

2.3 Rekurrens hálók

A rekurrens hálózatok fő különbsége és előnye abban rejlik (az előrecsatolt hálózatokhoz képest), hogy a bemeneti adatok időbeliségét, szekvenciális felépítését is kihasználják az információk kinyerésére. Erre a célra ún. memóriacellákat alkalmaznak, amelyek a háló korábbi bemeneti adatokra adott válaszait tárolják el.

Számos rekurrens architektúra létezik, az egyik legszélesebb körben használt a Long-Short Term Memory (LSTM). Az LSTM hálózatok, ahogy nevük is mondja képesek mind hosszabb és rövidebb (időben aktuálisabb) időintervallumok adatait is eltárolni annak érdekében, hogy a bemenet időben összefüggő részeit egyben, összefüggően dolgozhassák fel.

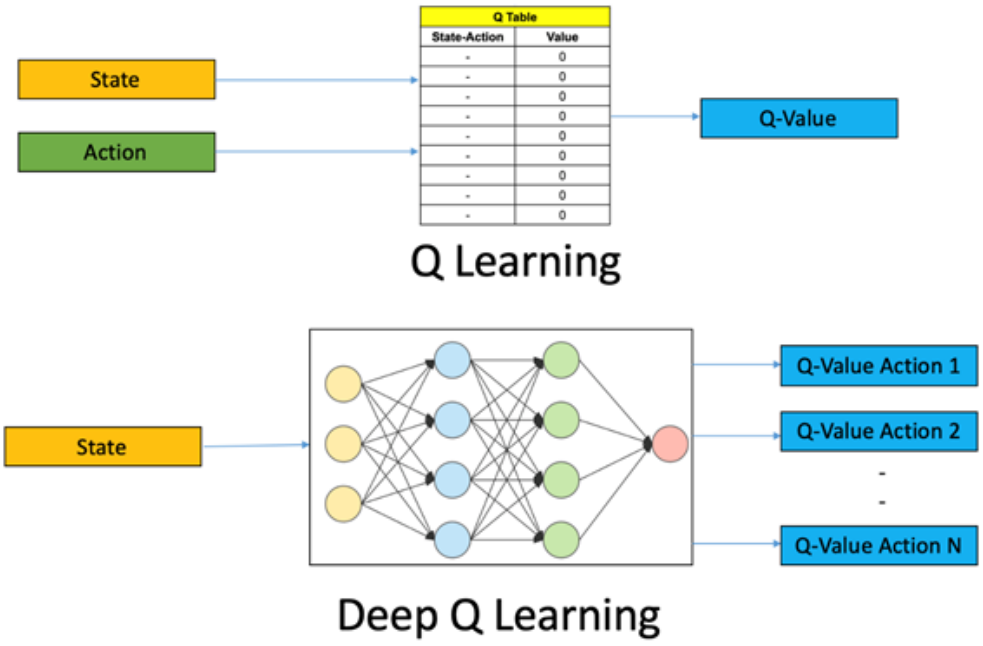


2.4 ábra. LSTM réteg felépítése [20]

3 Mély megerősítéses tanulás algoritmusai

3.1 Deep Q-learning

A Deep Q-Network (DQN) [15] a Q-tanuló egy speciális változata, amely akkor hasznos, hogyha a környezetben sok állapot és tevékenység figyelhető meg. Ebben az esetben ugyanis a Q-tanulóhoz szükséges Q-tábla mérete túl nagy lenne (pl. nem férne el a számítógép memóriájában). Ezen problémára nyújt megoldást a DQN, amely a Q-tábla helyett egy neurális hálót (Q-háló) használ a Q-értékek meghatározásához. A háló bemenetéül a környezet állapota szolgál, és a kimeneten a lehetséges tevékenységekhez tartozó Q-értékeket kapjuk, ahogy az a következő ábrán látszik.



3.1 ábra. Q-tanulás és a DQN közti különbség [16]

DQN esetén a Q-tanulónál látott képlet leegyszerűsödik a következőre:

$$Q(s_t, a_t) \leftarrow r_{t+1} + \gamma * \max_a Q(s_{t+1}, a) \qquad (3.1.1)$$

DQN esetén az jelenti a kihívást, hogy állandóan változó bemenetet próbálunk leképezni egy állandóan változó kimenetre. Ennek megoldására két módszer létezik, mellyel ideiglenes egyhelyben tartjuk a célváltozót. A célháló módszernél (target network) két neurális hálót használunk a tanításhoz: egy célhálót és egy főhálót. A célháló

által jósolt Q értékeket használjuk fel a főháló tanítására. A célháló paramétereit nem tanítjuk, azonban periodikusan szinkronizáljuk őket a főháló paramétereivel.

A másik megoldás ugyanezen problémára a tapasztalat visszajátszás (experience replay). Itt a környezet válaszát (az állapot, tevékenység, jutalom, következő állapot négyeseket) nem használjuk fel rögtön a háló tanítására, hanem egy pufferben eltároljuk. Egy bizonyos iterációszámot követően pedig ebből a pufferből választunk ki elemeket véletlenszerűen, és ezekkel az értékekkel tanítjuk a hálónkat. A véletlen kiválasztásnak azért van jelentősége, mert a környezet két szomszédos, vagy közeli állapotára általában hasonló dolgok a jellemzőek, így egymást követő két akció között nagy a korreláció, ami túltanuláshoz vezethet. Azzal, hogy a pufferből egyenletes mintavételezéssel választjuk ki az elemeket, tudjuk elkerülni az ebből fakadó problémákat.

3.2 Actor-critic módszerek

A hagyományos policy optimalizáláson alapuló, Monte Carlo szimulációkat használó módszerek egyik legnagyobb hátulütője a folyamatot jellemző magas variancia. Ezt a problémát igyekeznek megoldani az actor-critic módszerek [1], amelyek közös jellemzője, hogy neurális hálózatok paraméterezésével közelítik egyszerre az optimális Q-értékeket és a cselekvéseket is. A modell critic komponense felelős a Q-értékek tanulásáért, melyek felhasználásával frissíti szisztematikusan az actor komponens szabályterének (policy) paramétereit. A critic hálózat feladata továbbá az egyes állapot-cselekvés párosok végrehajtásával elérhető jutalom várható értékét is megbecsülni.

Az előző bekezdésben leírtak alapján megfigyelhető a módszer fő előnye, amelyet a két komponens összehangolt munkája tesz lehetővé: annak köszönhetően, hogy a critic folyamatosan ellátja az actor-t az egyes cselekvések végrehajtásának teljesítménymértékeivel, az actor úgy képes meghatározni a végrehajtandó tevékenységeket, hogy közben nincs szüksége a Q-függvényre.

A módszer természetéből adódóan a critic komponens jellemzően egy állapot-érték párost közelítő függvény. Minden egyes bemenetpárosra kiértékeli a függvényt, majd eldönti, hogy az új értékek javítottak vagy pedig rontottak a modellen:

$$A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{3.2.1}$$

ahol V az aktuális értékfüggvény. A 3.2.1 formulát TD (temporal difference) hibának is hívják, amely segítségével kiértékelhető egy a_t tevékenység végrehajtása s_t állapotban. Amennyiben a TD hiba pozitív az a_t tevékenység gyakoribb, amennyiben negatív az a_t ritkább végrehajtása javasolt a későbbi döntések során.

Algoritmus 3.2: Advantage Actor-Critic [14]

Paraméterek inicializálása: **s** (állapotok), **θ** (policy fv. paraméterei), **w** (Q fv. paraméterei)

for t = 1 ... T:

$\boldsymbol{r_t} \sim \boldsymbol{R}(\boldsymbol{s}, \boldsymbol{a})$ jutalom és $\boldsymbol{s'} \sim \boldsymbol{P}(\boldsymbol{s'} \mid \boldsymbol{s}, \boldsymbol{a})$ következő állapot mintavételezése

$\boldsymbol{a'} \sim \boldsymbol{\pi_\theta}(\boldsymbol{a'} \mid \boldsymbol{s'})$ (következő tevékenység) mintavételezése

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{a_\theta} \boldsymbol{Q_w}(\boldsymbol{s}, \boldsymbol{a}) \boldsymbol{\nabla_\theta} \boldsymbol{log}\, \boldsymbol{\pi_\theta}(\boldsymbol{a} \mid \boldsymbol{s})$ policy paraméterek frissítése

$\boldsymbol{\delta_t} = \boldsymbol{r_t} + \boldsymbol{\gamma} \boldsymbol{Q_w}(\boldsymbol{s'}, \boldsymbol{a'}) - \boldsymbol{Q_w}(\boldsymbol{s}, \boldsymbol{a})$ TD hiba számolása a korrekcióhoz

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{a_w} \boldsymbol{\delta_t} \boldsymbol{\nabla_w} \boldsymbol{Q_w}(\boldsymbol{s}, \boldsymbol{a})$ Q fv. paraméterek frissítése a TD hibával

frissítés: $\boldsymbol{a} \leftarrow \mathbf{a'}$ és $\mathbf{s} \leftarrow \mathbf{s'}$

end for

Deep Deterministic Policy Gradient módszer (DDPG)

Az actor-critic típusú modellek egyik legfejlettebb változata az összesen négy hálózatot felhasználó DDPG eljárás [12]. A módszer az előzőekben bemutatott Advantage Actor-Critic alapjaira épül, de kibővítésre került egy-egy célhálózattal is, amelyek az eredeti hálózatok periodikusan rögzített másolatai. A DDPG modell komponensei tehát a θ^Q és $\theta^{Q'}$ (Q-érték és cél Q-érték függvények), illetve a θ^μ és $\theta^{\mu'}$ (policy és cél policy függvények) hálózatok. A két extra hálózat használata stabilabban konvergáló tanulást tesz lehetővé, ugyanis a paraméterek időszakos mentése megoldást nyújt az optimalizációs folyamat közben fellépő divergenciára.

Az actor-critic hálózatok frissítésén kívül a DDPG modell három fő technikára épít: az tapasztalat visszajátszásra, a célhálózatok hatékony módosítására és a tevékenység- illetve paramétertérből való megfelelő mintavételezésre (exploration). Az elsőről már esett szó korábban, de a többi technikát érdemes kifejteni röviden. Az actor és critic hálózatok frissítése a Q-tanuló módszerhez hasonló módon történik, azzal a különbséggel, hogy a DDPG eljárás során a következő állapotok Q-értékeit a $\theta^{\mu'}$ és $\theta^{Q'}$ hálózatok adják meg:

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right) \tag{3.2.2}$$

Az új Q-érték meghatározása után a θ^Q hálózat által kiszámolt eredeti Q-érték felhasználásával megkaphatjuk a kumulált hibát, azaz a veszteséget:

$$\text{Veszteség} = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i \mid \theta^Q)\right)^2 \tag{3.2.3}$$

A cél hálózatok frissítése ennél egyszerűbben, a paramétereik másolásával történik:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{aligned} \tag{3.2.4}$$

Megerősítéses tanulás esetén az eseménytérből való következő tevékenység mintavételezése során fontos szempont az exploitation és exploration kiegyensúlyozása. A véletlenszerű választás mellett folytonos eseménytérből való mintavételezésnél magához a tevékenységhez adunk zajt az Ornstein-Uhlenbeck folyamat [23] felhasználásával:

$$\mu'(s_t) = \mu\left(s_t \mid \theta_t^\mu\right) + N \tag{3.2.5}$$

3.3 Clipped double Q-learning

A clipped double Q-learning algoritmus [7] a következőképpen néz ki:

Algoritmus 3.3: Clipped Double Q-learning

Neurális hálók inicializálása: $Q_{\theta 1}$, $Q_{\theta 2}$, puffer: D

for minden iterációra**:**

  for minden lépésre:

  s_t állapot megfigyelése és a_t kiválasztása $\pi(a_t, s_t)$ alapján

  a_t végrehajtása, és az s_{t+1} állapot, valamint az r_t jutalom megfigyelése

  (s_t, a_t, r_t, s_{t+1}) eltárolása a D pufferben

  end for

  for minden frissítésre:

    mintavételezés a bufferből: $e_t = (s_t, a_t, r_t, s_{t+1}) \sim D$

    cél Q-érték meghatározása:

$$Q^*(s_t, a_t) \approx r_{t+1} + \gamma * \min_{i=1,2} Q_{\theta_i}\left(s_{t+1}, \operatorname{argmax}_a Q_{\theta_i}(s_{t+1}, a)\right)$$

gradiens csökkentési lépés végrehajtása $\left(Q^*(s_t, a_t) - Q_\theta(s_t, a_t)\right)^2$

hálók paramétereinek frissítése: $\theta_{1,2} \leftarrow \gamma * \theta + (1 - \gamma) * \theta_{1,2}$

end for

end for

Elsőként 2 neurális hálót inicializálunk, majd a környezettel interaktálva egy buffer-ben eltároljuk az állapot, tevékenység, jutalom, következő állapot négyeseket, ahol a tevékenység kiválasztásához mindkét neurális hálót felhasználjuk.

A modell frissítésénél a bufferben eltárolt értéknégyesekből mintavételezünk, majd kiszámoljuk a cél Q-értékét a következő képlet segítségével:

$$Q^*(s_t, a_t) \approx r_{t+1} + \gamma * \min_{i=1,2} Q_{\theta_i}\left(s_{t+1}, \operatorname{argmax}_a Q_{\theta_i}(s_{t+1}, a)\right) \qquad (3.3.1)$$

Látható, hogy a két neurális haló közül annak a Q értékét használjuk fel, melynél kisebb értéket kapunk, ezzel elkerülve a Q-értékek túlbecslését. Ezután az update target-et felhasználva tanítjuk a két hálót.

Fujimoto emellett egy másik előnyét is bemutatja ennek a módszernek. A minimum operátor magasabb értéket biztosít azon állapotoknak, melyeknél kisebb a szórás becslési hibája [7]. Ez azt jelenti, hogy a minimalizálás olyan állapotokat részesít előnyben, melyek értékének kisebb a varianciája, amely egy biztonságosabb policy frissítéshez és stabilabb tanulási célhoz vezet.

4 Pénzügyi és gazdasági alkalmazások

A megerősítéses tanulás alkalmazásai között a különböző pénzügyi területek is gyakran szerepelnek. Ebben a fejezetben ezen alkalmazási lehetőségeket mutatjuk be.

4.1 Algoritmikus tőzsdei kereskedés

Az egyik talán legnépszerűbb terület az algoritmikus tőzsdei kereskedés. A nyereséges tőzsdei kereskedési stratégia létfontosságú a bankok, befektetési alapok számára, melynek sikeréhez a gépi tanulás által felismert összefüggések nagyban hozzájárulhatnak.

Ahhoz, hogy a megerősítéses tanulási algoritmusokat használni tudjuk algoritmikus tőzsdei kereskedésre, a problémát meg kell fogalmaznunk, mint egy Markov döntési folyamatként, és a kereskedési célunkat pedig egy maximalizálási problémaként. Ahhoz, hogy egy problémát Markov döntési folyamatként megfogalmazzunk szükséges definiálnunk a környezet állapotterét, a lehetséges tevékenységeket, valamint a jutalom függvényt.

4.1.1 Tőzsdei kereskedés egy részvénnyel

Elsőként vegyük azt az esetet, amikor csak egyetlen cég részvényeivel szeretnénk kereskedni a tőzsdén és ehhez hozzuk létre a modellünket. Itt a környezet állapotterét az adott cég részvényének az ára határozza meg, emellett pedig felhasználhatjuk a korábbi részvényárfolyamokat, valamint technikai indikátorokat is mint bemenő adatok.

A modell alapvetően három tevékenységet tud végrehajtani ebben a környezetben: elad, tart, illetve vesz részvényt, melyet az alábbi formában szokás megadni, rendre: {-1, 0, 1}. A tevékenységeket azonban úgy is definiálhatjuk, hogy a modell határozza meg vásárlandó részvények számát, például a {-10, ..., -1, 0, 1, …, 10} vektorral tudjuk definiálni azt az állapotteret, melyben a modellünk képes 1-10 részvény eladására, illetve megvételére [13].

Végül szükségünk van a jutalomfüggvény definíciójára, amely a modellünk ösztönzőmechanizmusa a jobb cselekvés megtanulására. Tőzsdei kereskedésben

egyszerűen meghatározhatjuk a jutalomfüggvényt azzal, hogy a portfólió új értékéből kivonjuk az előző állapotban vett értékét [13]. Formálisan ezt a következőképpen tudjuk felírni: $r(s, a, s') = v' - v$, ahol v jelenti a portfólió értékét (azaz az általunk birtokolt részvények értékének és a készpénzünk értékének összegét). A cél egy olyan kereskedési stratégia megtervezése, amely maximalizálja a portfóliónk értékének pozitív változását valós környezetben.

Egy lépésben a modell megkapja a környezet állapotát, amely tartalmazza az aktuális és korábbi részvényárat, valamint a használt technikai indikátorokat. Ez alapján a modell kiválaszt egy tevékenységet, és a következő részvényár alapján meghatározható a tevékenység jósága, amivel taníthatjuk a modellt.

4.1.2 Tőzsdei kereskedés több részvénnyel

Abban esetben, hogyha már nem csak egy részvénnyel szeretnénk kereskedni, hanem több cégnek a részvényeit is figyelembe vesszük, akkor változtatnunk kell a modellünkön, valamint figyelembe kell vennünk, hogy az állapottér, valamint a tevékenységtér mérete a részvények számával arányosan exponenciálisan nő.

Ebben az esetben az állapotteret már nemcsak egy cég részvényárfolyama határozza meg, hanem a számításba vett összes cég részvényárfolyama, valamint az ezekből számított technikai indikátorok. Általánosan felírva, hogyha c darab cég részvényével kereskedünk és minden részvényt l darab jellemzővel írunk le, akkor az állapotteret egy $(1 + l)^c + 1$ hosszú vektor határozza meg (minden cégre az általunk birtokolt részvények száma, az l darab jellemző, valamint a még rendelkezésre álló tőkénk) [13].

A tevékenységek tere is változik azzal, hogy több részvényt veszünk figyelembe. Egy részvénnyel kereskedve a *{-k, ..., -1, 0, 1, ..., k}* vektorral tudtuk leírni a végrehajtandó akciók terét, ahol k határozta meg, hogy egyszerre hány részvénnyel kereskedhet a modellünk. Általánosan c darab részvénnyel kereskedve minden céghez meghatározhatunk egy ugyanilyen halmazt, így $(2k + 1)^c$ hosszú vektor határozza meg a tevékenységek terét.

A jutalomfüggvény azonban nem változik abban az esetben, hogyha növeljük a részvények számát. Egyedül a portfólió értékének meghatározását kell módosítanunk,

ahol már az összes általunk birtokolt részvény számát kell összeszoroznunk a részvények árával és ehhez hozzáadnunk a rendelkezésre álló készpénzünket.

Egy lépésben a modell a cégek részvényárfolyamai alapján mindegyik részvényhez meghatározza, hogy mennyit vesz, ad el, vagy éppen tartja a részvényeket. Az új árfolyam alapján meghatározható a döntés jósága, amivel taníthatjuk a modellünket.

Az így meghatározott környezetet már felhasználhatjuk megerősítéses tanuláshoz. Több megerősítéses tanulási algoritmust is taníthatunk a környezetet felhasználva, melyeket tesztelve kiválaszthatjuk a legjobbat, amivel elkezdhetünk kereskedni a tőzsdén.

4.2 Portfólió menedzsment

Az algoritmikus részvénykereskedés mellett a portfólió menedzsment is egy népszerű pénzügyi terület a megerősítéses tanulás alkalmazására. Megerősítés tanulási algoritmust használva tudjuk megmondani, hogy a pénzünket milyen arányban osszuk fel a különböző pénzügyi eszközök között. Ezek között is a leggyakrabban vizsgált lehetőség, amikor különböző cégek részvényei között osztjuk el a pénzünket, melynek arányát bizonyos időközönként, például napi szinten tudjuk módosítani. A megerősítéses tanulás alkalmazásához ezt a feladatot is meg kell fogalmaznunk egy Markov döntési folyamatként.

A modell tevékenységtere azt adja meg, hogy a portfóliónknak mekkora részét teszi ki az adott részvény, tehát mindegyik részvényhez egy 0-1 közötti szám tartozik, melyek összege 1. Ennek eléréséhez például a softmax függvényt tudjuk használni. Ezután ismerve a rendelkezésünkre álló tőkét egy szorzással meghatározhatjuk, hogy az egyes részvényekből mekkora értékben kell vásárolnunk.

A környezet állapotterét a részvények ára és azok mozgása határozza meg. A részvények együttes mozgásából meghatározhatunk egy kovariancia mátrixot, mellyel jellemezhetjük a részvények együttes mozgását, illetve technikai indikátorokat is használhatunk, amelyek a modell döntését segíthetik a hatékony portfólió meghatározásához.

Az algoritmus célja a portfólió értékének maximalizálása, amihez leggyakrabban az átlagos kumulatív logaritmikus jutalomfüggvényt [9] szokás használni:

$$r = \frac{1}{n}\sum_{t=1}^{n+1} \ln(y_t w_{t-1}) \tag{4.2.1}$$

ahol y_t vektor tartalmazza a részvények árát a t-edik időpillanatban, míg w_t a t-edik részvény súlyát jelöli a portfólióban.

A kumulatív logaritmikus jutalomfüggvény azonban nem veszi figyelembe a portfolió kockázatát, egyedül a nyereséget próbálja növelni, így ennek használatával a modellünk nagyobb kockázatokat fog vállalni. Ahhoz, hogy a kockázatot is figyelembe vegye a modellünk, az egyik megoldás a Differential Sharpe Ratio-t (DSR) [6] használata.

Egy lépésben a modell a részvényárfolyamok és az ezekből meghatározható adatok alapján összeállít egy portfóliót, melynek jóságát a következő állapot és a jutalomfüggvény segítségével kiértékelve taníthatjuk a modellünket. A következő lépésben a modell az új adatok alapján újra osztja a részvények súlyát a portfólióban, amely alapján mi is átrendezhetjük a részvények arányát.

4.3 Market Maker

A market maker célja a likviditás biztosítása a piacon, és a piaci résztvevők közötti tranzakciók megkönnyítése. A nyereség érdekében a market maker-nek folyamatosan frissítenie kell az árajánlatokat, hogy tükrözzék a valós piaci körülményeket, és kezelniük kell a pozícióikat, hogy elkerüljék a piac egyirányú elmozdulását. Az árajánlatok és a készletszint fenntartásának optimalizálási problémája általi kihívás ad lehetőséget a megerősítéses tanulás alkalmazására.

A modell környezetét három résztér kombinációjaként határozhatjuk meg: a környezeti állapot tér (a limit order books, a trade flow imbalance és az order flow imbalances adataival); a modell állapot tér, amely kockázat- és helyzetindikátorokból áll; és az ágens cselekvési tér, amely az ágens legutolsó tevékenységét tartalmazza.

A modell négy általános műveletet hajthat végre egy adott lépésben: nem csinál semmit, aszimmetrikusan ad le megrendelést a limit order books-ba, torzítja a limit order books vételi vagy eladási megrendeléseit, vagy ellapítja a pozíciók leltárát.

Többféle jutalomfüggvényt is alkalmazhatunk a tevékenységek kiértékeléséhez. Az egyik a Positional PnL [19] (Profit and Loss), ahol a jutalom értéke megegyezik a modell pozíciójának változásával és az aktuális lépés nyereségének vagy veszteségének az összegével. Ha az aktuális lépésben nem zártak le pozíciókat, akkor ez az érték 0.

$$r_t = \Delta m_t \times ic_t + PnL_t^{realizált} \tag{4.3.1}$$

ahol $\Delta m_t = \frac{m_t}{m_{t-1}} - 1$ a hozam középértéke, valamint ic_t a t-edik időpillanatban tartott pozíciók száma.

A másik lehetséges jutalomfüggvény a Trade Completion [19], ahol a jutalom értékét -1 és 1 között határozzuk meg. Ez a függvény csak abban az esetben határoz meg 0-tól eltérő értéket, hogyha egy pozíciót lezár a modell és realizált PnL keletkezik. Ha a realizált PnL értéke nagyobb (vagy kisebb), mint egy előre meghatározott küszöbérték, akkor a jutalom értéke 1 (vagy -1), míg hogyha a két küszöbérték között van, akkor a jutalom a realizált PnL százalékban kifejezve.

$$r_t = \begin{cases} 1, & ha\ \ PnL^{realizált} > \varepsilon \\ -1, & ha\ PnL^{realizált} < -\varepsilon \\ PnL^{realizált}, & különben \end{cases} \tag{4.3.2}$$

Ilyen market maker alkalmazást készítettek kriptovalutáknál, ahol a megerősítéses tanulás nyereséges napi hozamot tudott termelni anélkül, hogy előzetesen ismerte volna a market making folyamatát [19].

4.4 Multiágens likviditási stratégia

A befektetési portfóliók módosításának, pontosabban a pozíciók felszámolásának egyik kellemetlen velejárója a főként nagy mennyiségű részvény eladását kísérő értékcsökkenés (shortfall). Így itt olyan mély megerősítéses tanuláson alapuló módszert mutatunk be, amely több ágens közreműködésével szimulálja a pozíciók eladásának optimális módját, amely a lehető legkevesebb negatív ráhatással bír a piac viselkedésére. A következőkben részletezett ágens (kereskedő) célja tehát olyan eladási stratégiát találni, amely minimalizálja a kereskedési költségek várható értékét (E(x)) az eszköz árának múltbeli adatainak és a hátralévő kereskedési napok számának ismeretében.

4.4.1 Szimulációs környezet

A szimuláció során az Almgren-Chriss modellel [2] írható le az eszköz (részvény) ára a folyamat során a következőképpen (ahol σ jelöli az eszköz volatilitását, ξ_k 0 várható értékű és 1 varianciájú véletlen változók, a g függvény a kereskedés átlagos aránya, n_k az eladható eszközök száma a t_{k-1} és t_k közötti időintervallumban, N a kereskedések teljes száma, $\tau = teljes\ kereskedési\ idő/N$):

$$P_k = P_{k-1} + \sigma\sqrt{\tau}\xi_k - \tau \cdot g\left(\frac{n_k}{\tau}\right) \quad k = 1,2,\dots,N \tag{4.4.1}$$

A kereskedési folyamatot a piac dinamikájára tekintettel modellezhetjük egy Markov döntési folyamattal [3], ahol az állapotokat az $s = [r, m, l]$ hármas írja le (r: log-return, m: hátralévő tranzakciók száma normalizálva, l: még el nem adott részvények száma normalizálva). Az állapotok közötti tevékenységeket (a) megfeleltethetjük a pozíció egyes hányadainak méretével (amekkora részt akarunk egy tevékenységben eladni). A jutalomfüggvényt a lambda kockázatvállalási hajlandóság (kockázatkerülési szint) és a maradék eladható részvény függvényében a következőképpen írhatjuk fel (x a kereskedési trajektória, x_t^* pedig az optimális kereskedési trajektória a t időpontban):

$$U(x) = E(x) + \lambda V(x) \tag{4.4.2}$$

$$E(x) = \sum_{k=1}^{N} \tau x_k g\left(\frac{n_k}{\tau}\right) + \sum_{t=1}^{N} n_k h\left(\frac{n_k}{\tau}\right) \text{ és } V(x) = \sigma^2 \sum_{k=1}^{N} \tau x_k^2 \tag{4.4.3}$$

$$R_t = U_t(x_t^*) - U_{t+1}(x_{t+1}^*) \tag{4.4.4}$$

Az R_t-t minden időpillanatra kiszámítva és ez alapján megválasztva a tevékenységet megkapjuk a $\pi(s)$ szabályt, azaz a likviditási stratégiát, amellyel leírható az s állapotban eladásra szánt részvények száma (pozíció hányada) a alapján. Ezek alapján a $Q_\pi(s, a)$ függvénnyel kiszámolható az a esemény várható jutalma s állapotban a π szabályt követve.

4.4.2 Multiágens megközelítés

A szimulációkban lehet alkalmazni a multiágens megközelítést is [3], amely – ahogyan a neve is mutatja – több piaci szereplőt (kereskedőt) használ, így szimulálva a „single-agent” módszerekhez képest valósághűbb környezetet a kereskedési folyamatnál

[11]. A szimulációk futtatása során a Q értékeket és a tevékenységeket az elméleti bevezetőben bemutatott actor-critic típusú DDPG módszer segítségével határozhatjuk meg.

Algoritmus 4.4: Multiágens DDPG módszer [3]

Paraméterek inicializálása: **M** (epizódok), **T** (időkeret), **N** (minibatch méret), **J** kereskedő

for $\mathbf{j = 1 \dots J}$**:**

critic: $\mathbf{Q_j}\left(\mathbf{O_{j,a}} \mid \boldsymbol{\theta}_\mathbf{j}^{\mathbf{Q}}\right)$ és actor: $\boldsymbol{\mu}_\mathbf{j}\left(\mathbf{O_j} \mid \boldsymbol{\theta}_\mathbf{j}^{\boldsymbol{\mu}}\right)$ hálók inicializálása

$\mathbf{Q'_j}$ és $\boldsymbol{\mu}'_\mathbf{j}$ target hálózatok inicializálása a critic és actor hálózatok súlyaival

end for

for $\mathbf{e = 1 \dots M}$**:**

Véletlen **N** exploration esemény indítása: $\mathbf{s_0}$ állapotba állás

for $\mathbf{t = 1 \dots T}$**:**

for $\mathbf{j = 1 \dots J}$**:**

$\mathbf{a_{j,t}} = \boldsymbol{\mu}_\mathbf{j}\left(\mathbf{0_j t} \mid \boldsymbol{\theta}_\mathbf{j}^{\boldsymbol{\mu}}\right) + \mathbf{N_t}$ tevékenység kiválasztása minden agent-nek

end for

$\boldsymbol{a_{j,t}}$ tevékenység végrehajtása: $\boldsymbol{s_{t+1}}$ állapotba állás, $\boldsymbol{r_{j,t}}$ és $\boldsymbol{O_{j,t+1}}$ meghatározása

for $\mathbf{j = 1 \dots J}$**:**

$\boldsymbol{B_j} \leftarrow \left(\mathbf{O_{j,t}}, \mathbf{a_{j,t}}, \mathbf{r_{j,t}}, \mathbf{O_{j,t+1}}\right)$ átmenetek eltárolása és N minibatch mintavételezése

critic hálózat frissítése a veszteséggel: $\mathbf{L} = \frac{\mathbf{1}}{\mathbf{N}}\sum_i \left(\boldsymbol{y_{j,i}} - \boldsymbol{Q_j}\left(\boldsymbol{Q_{j,i}}, \boldsymbol{a_{j,i}} \middle| \boldsymbol{\Theta_j^Q}\right)\right)^2$

actor hálózat frissítése: $\boldsymbol{\nabla_\theta \mu \pi} \approx \frac{\mathbf{1}}{\mathbf{N}}\sum_i \boldsymbol{\nabla_a}\, \boldsymbol{Q_j}\left(\mathbf{0}, \boldsymbol{a} \middle| \boldsymbol{\theta_j^Q}\right) \boldsymbol{\nabla_{\Theta^\mu}} \boldsymbol{\mu_j}\left(\boldsymbol{O_j} \middle| \boldsymbol{\theta^\mu}\right)$

target hálózatok frissítése: $\boldsymbol{\theta}_\mathbf{j}^{\mathbf{Q}'} \leftarrow \boldsymbol{\tau\theta}_\mathbf{j}^{\mathbf{Q}} + (\mathbf{1} - \boldsymbol{\tau})\boldsymbol{\theta j}^{\boldsymbol{Q'}}$ és $\boldsymbol{\theta}_\boldsymbol{j}^{\boldsymbol{\mu'}} \leftarrow \boldsymbol{\tau\theta}_\boldsymbol{j}^{\boldsymbol{\mu}} + (\mathbf{1} - \boldsymbol{\tau})\boldsymbol{\theta j}^{\boldsymbol{\mu'}}$

end for

end for

end for

A [3] cikkben részletezett mérések mintájára az itt bemutatott módszer segítségével meghatározható egy valósághű környezetben a pozíció eladásának optimális stratégiája a várható kereskedési költségekre való tekintettel. A módszer továbbá alkalmas különböző kereskedők kooperációs taktikájának vizsgálatára, amely

segítségével hatékonyabb, az aktuális célokhoz és megszorításokhoz alkalmazkodó kooperatív-kompetitív eladási stratégiák tervezhetők meg.

4.5 Deep hedging

Ebben az alfejezetben részvénypozíciók kockázatának optimális csökkentésére (hedging) mutatunk egy mély megerősítéses tanulást használó módszert. Legyen a kísérleti szcenáriónk egy kereskedési helyzet, amikor a piaci szereplő a short pozíciójának kockázatát szeretné csökkenteni egy call opció segítségével. Tételezzünk fel valós piaci feltételeket: a kereskedő Δt időközönként módosítja a pozícióját, amely jelentős költségekkel járhat.

4.5.1 Szimulációs környezet

A döntési folyamatok állapotait (S) három fő paraméter határozza meg [8]: a megtartott eszközmennyiség az előző időpont óta, az eszköz ára és a lejárati időpontig hátralévő idő. A *t* időpillanatban végrehajtandó tevékenység értéke legyen a megtartandó eszközmennyiség a következő időpillanatig. A döntési folyamatba bele kell számolni a tranzakciós költségeket, amelyek negatív jutalom formájában fognak megjelenni a modellben. Célunk tehát a hedging költség várható értékének minimalizálása:

$$Y(t) = E(C_t) + c\sqrt{E(C_t) - E(C_t)^2} \tag{4.5.1}$$

ahol C_t a teljes költség és a második tag a hedging költség szórását becsüli egy konstans segítségével a bias szimulálására.

4.5.2 Mély megerősítéses tanuló modell

A mély megerősítéses tanulásnál a célfüggvény komplexitása érdekében két külön Q függvényt használhatunk a várható költségek közelítésére. A Q_1 függvény az állapot-tevékenység párosítások várható költségeit, a Q_2 függvény pedig ezen várható költségek négyzetét becsüli meg. Diszkrét eseménytér esetén a költségfüggvény előáll a következő alakban:

$$\mathrm{F}(\mathrm{S_t}, \mathrm{a}) = \mathrm{Q_1}(\mathrm{S_t}, \mathrm{a}) + \mathrm{c}\sqrt{\mathrm{Q_2}(\mathrm{S_t}, \mathrm{a}) - \mathrm{Q_1}(\mathrm{S_t}, \mathrm{a})^2} \tag{4.5.2}$$

A Q_1 függvény frissítése:

$$Q_1(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q_1(S_{t+1}, a) - Q_1(S_t, A_t)\big) \tag{4.5.3}$$

A Q_2 függvény frissítéséhez felhasználjuk, hogy a várható értéke:

$$E(x) = [R_{t+1} + \gamma Q_1(S_{t+1}, a)]^2 \tag{4.5.4}$$

Ez alapján a paraméterek frissítése:

$$\begin{aligned} \mathrm{Q}_2(\mathrm{St}, \mathrm{A}_t) &\leftarrow \mathrm{Q}_2(\mathrm{S}_t, \mathrm{A}_t) \\ + \alpha\{\mathrm{R}_{t+1}^2 + \gamma^2 \mathrm{Q}_2(\mathrm{S}_{t+1}, \mathrm{a}) + 2\gamma \mathrm{R}_{t+1} \mathrm{Q}_1(\mathrm{S}_{t+1}, \mathrm{a}) - \mathrm{Q}_2(\mathrm{St}, \mathrm{A}_t)\} \end{aligned} \tag{4.5.5}$$

Egy folytonos eseménytér használata lehetővé teszi a hedging minél valósághűbb szimulálását (kiszűrhető a diszkretizálással járó torzítás) [24]. Egy ilyen megközelítéshez alkalmaznunk kell a korábban bemutatott DDPG módszert a két Q függvény, $Q_1(S_t, A_t, w_1)$ és $Q_2(S_t, A_t, w_2)$ közelítésére a veszteségfüggvényeik minimalizálásával. A DDPG algoritmusban a $\pi(S_t, \theta)$ policy függvény frissítése:

$$\theta \leftarrow \theta - \alpha \nabla_\theta F\big(S_t, \pi(S_t, \theta)\big) \tag{4.5.6}$$

Az itt bemutatott módszer alkalmazásával meghatározható egy olyan kockázatcsökkentő technika, amely optimális delta-hedging stratégiát tesz lehetővé. Segítségével elkerülhet az over- illetve underhedging, így csökkentve a felmerülő kereskedési költségeket és értékvesztést.

4.6 Aukcióelmélet ágensekkel

Az elmúlt években a játékelmélet alkalmazása a gazdasági kérdések, folyamatok számtalan fajtájában kapott egyre nagyobb szerepet. Ebben a fejezetben az aukcióelmélet eszközeit felhasználva piaci szereplők (ágensek) döntéseinek optimalizálására mutatunk módszereket, akik az aukciós események vételi oldalán állnak. Fontos megkötés a továbbiakra nézve, hogy az ágensek nem rendelkeznek korlátlan tőkével, a licitek során felhasználható eszközmennyiség korlátos (budget constrained bidding , BCB) [25].

Az előző alfejezetekben bemutatott problémákhoz hasonlóan, az aukciós folyamatok eseménytere is magas komplexitást és volatilitást mutat. A környezethez alkalmazkodva a BCB problémát célszerű a korábban már megismert Markov döntési folyamat feltételes változataként (Constrained MDP, CMDP) kezelni mély megerősítéses tanuló módszerek felhasználásával.

A módszer bővebb kifejtése előtt szükség van egy fontos fogalom az impression/return value bevezetésére. Ez az érték felfogható az adott licit tárgyának megtérülési értékeként egy tetszőlegesen választott skálán (hirdetési felületeknél lehet akár a várható megtekintések száma [10]). Kimutatták, hogy a megtérülési érték és az optimális licit között szoros kapcsolat van a v/λ formulán keresztül, ahol v a megtérülési érték, λ pedig egy skálázási faktor [26]. Az összefüggés a következőképpen értelmezhető: minél értékesebb a licit tárgya, annál magasabb árat érdemes (szükséges) fizetni érte és fordítva. A képlet egyszerű, a λ paraméter optimális meghatározás viszont nehéz feladat a korlátos tőke és az aukciós környezet sztochasztikus viselkedése miatt.

A nehézség ellenére célszerű az állapot átmenetek dinamikájának modellezése helyett a λ paraméter meghatározása az egyes időpillanatokban. Minden egyes t pillanatban az ágens s_t állapotban áll és egy a_t tevékenységet hajt végre, amelynek célja λ_{t-1} alapján λ_t korrigálása. Az ágens célja tehát egy olyan optimális λ policy-t meghatározni, amely maximalizálja az időpillanatokra számolt kumulatív $\sum_{t=1}^{T} r_t$ jutalmat tekintettel a rendelkezésre álló keretre és a licitek költségeire ($\sum_{t=1}^{T} c_t \leq B$).

Irodalomjegyzék

[1] Actor-Critic módszerek: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

[2] Almgren, R. and Chriss, N. (2001). Optimal execution of portfolio transactions. Journal of Risk, 3, pp. 5–40.

[3] Bao, W., & Liu, X. (2019). Multi-Agent Deep Reinforcement Learning for Liquidation Strategy Analysis. ArXiv, abs/1906.11046. https://arxiv.org/abs/1906.11046

[4] Bellman, R. (1957). A Markovian Decision Process. Journal of Mathematics and Mechanics. 6 (5): 679–684. JSTOR 24900506.

[5] Bellman, R. (1957). Dynamic Programming. Dover. ISBN 0-486-42809-5.

[6] Filos, A. (2019). Reinforcement Learning for Portfolio Management, ArXiv abs/1909.09571, 2019

[7] Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In International Conference on Machine Learning (pp. 1587-1596). PMLR.

[8] Jay C., Jacky, C., Hull, J.C. and Zissis, P. (2019). Deep Hedging of Derivatives Using Reinforcement Learning, http://dx.doi.org/10.2139/ssrn.3514586

[9] Jiang, Z., Liang, J. (2017). Cryptocurrency Portfolio Management with Deep Reinforcement Learning, ArXiv abs/1612.01277

[10] Jin, J., Song, C., Li, H., Gai, K., Wang, J., & Zhang, W. (2018). Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. Proceedings of the 27th ACM International Conference on Information and Knowledge Management. https://arxiv.org/abs/1802.09756

[11] Lee, J. W., J. Park, J. O, J. Lee and E. Hong, "A Multiagent Approach to Q-Learning for Daily Stock Trading," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 37, no. 6, pp. 864-877, Nov. 2007, doi: 10.1109/TSMCA.2007.904825. https://ieeexplore.ieee.org/document/4342801

[12] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. CoRR, abs/1509.02971. https://arxiv.org/abs/1509.02971

[13] Liu, X.Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., Wang, C.D. (2020). FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance, ArXiv abs/2011.09607

[14] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.

[15] Mnih, V.; Kavukcuoglu, Koray; Silver, David; Rusu, Andrei A.; Veness, Joel; Bellemare, Marc G.; Graves, Alex; Riedmiller, Martin; Fidjeland, Andreas K. (2015). Human-level control through deep reinforcement learning. Nature. 518 (7540): 529–533.

[16] Nováček, O., Vaiciukevičius, D., Koch, M. (2020). Connect X with DQN and PBT. Jun 15, 2020. (medium.com)

[17] Rummery, G. A., Niranjan, M. (1994). Online Q-learning using connectionist systems

[18] Russel, S. & Norvig, P. (2005). Mesterséges intelligencia modern megközelítésben. 2. kiadás. Panem.

[19] Sadighian, J. (2019). Deep Reinforcement Learning in Cryptocurrency Market Making, ArXiv abs/1911.08647,

[20] Smalter Hall, A., & Cook, T. R. (2017). Macroeconomic indicator forecasting with deep neural networks. Federal Reserve Bank of Kansas City Working Paper, (17-11).

[21] Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In NIPs (Vol. 99, pp. 1057-1063).

[22] Sutton, R., Barto, A. (1998). Reinforcement Learning. MIT Press. ISBN 978-0-585-02445-5. Archived from the original on 2017-03-30.

[23] Uhlenbeck, George E and Ornstein, Leonard S. (1930). On the theory of the brownian motion. Physical review, 36(5):823.

[24] Vittori, E., Trapletti, M., & Restelli, M. (2020). Option Hedging with Risk Averse Reinforcement Learning. ArXiv, abs/2010.12245. https://arxiv.org/abs/2010.12245

[25] Wu, D., Chen, X., Yang, X., Wang, H., Tan, Q., Zhang, X., & Gai, K. (2018). Budget Constrained Bidding by Model-free Reinforcement Learning in Display Advertising. Proceedings of the 27th ACM International Conference on Information and Knowledge Management. https://arxiv.org/abs/1802.08365

[26] Zhang, W., Yuan, S. and Wang. J. (2014). Optimal real-time bidding for display advertising. In 20th SIGKDD. pp. 1077–1086.

A szerzőkről

Knoll Zsolt

hallgató

Knoll Zsolt mesterszakos gazdaságinformatikus hallgató a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karán. Kutatási tevékenysége az adatelemzés és a gépi tanulás, ezen a területen 2. helyezést ért el a kari TDK konferencián.

Németh Marcell

hallgató

Németh Marcell mesterszakos mérnökinformatikus hallgató a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karán. Kutatási tevékenysége az adattudomány, az adatsorok elemzése, a képfelismerés, ezen a területen 2. helyezést ért el a kari TDK konferencián.

Dr. Szűcs Gábor, PhD

egyetemi docens, Távközlési és Médiainformatikai Tanszék

Szűcs Gábor 1994-ben végzett villamosmérnökként a BME VIK Karán, és ugyanitt 2002-ben informatikai PhD fokozatot szerzett. Kutatási területei az adattudomány, a mesterséges intelligencia, a mélytanulás, a tartalom alapú képkeresés és a gépi látás. Publikációinak száma meghaladja a 100-at. A HTE Mesterséges Intelligencia Szakosztályának elnöke, a DCLAB (Data Science and Content Technologies) kutatócsoport vezetője.