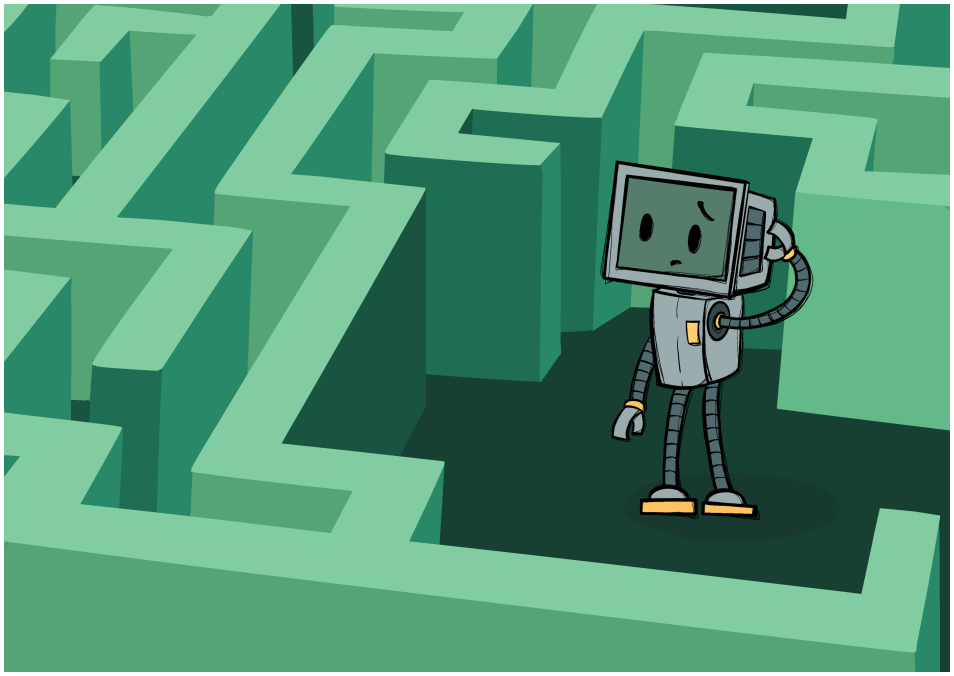
5. Modul

Megerősítéses tanulás

"Egy olyan útvesztőben, amelyben a robotot egy elem megtalálásáért jutalmazzák, megmutatja, hogy az RL nem olyan egyszerű, mint gondolnánk."



A modulról

Ez a modul a megerősítéses tanulásról (***RL***) szól. A cél az, hogy a hallgatók alapvető ismereteket szerezzenek arról, hogy mi az ***RL***, hogyan működik, és melyek a gyakori problémák és buktatók. A hangsúly inkább a gyakorlati feladatokon van, ezért a hallgatók egyrészt az önállóan tanuló algoritmusok szerepét veszik át, és megtapasztalják, hogyan működik a betanítási folyamat, másrészt pedig játszanak ellene, és tanulják képezni a mesterséges intelligenciát.

Célok

A tanulók képesek lesznek...

...elmagyarázni az ***RL*** alapgondolatát
...megértik az ***RL***-csomó lényegét és a Q-értékeken alapuló döntéseket
...megbecsülni, hogy egy alkalmazás használ-e ***RL***-t
...megnevezni az ***RL*** problémáit és korlátait

Időbeosztás

Idő	Tartalom
30 perc	Bevezetés
20 perc	Gyakorlat - Érmejáték
20 perc	Gyakorlat - Hexapawn
20 perc	Gyakorlat - Menace
30 perc	Gyakorlat - Labirintus
15 perc	Kvíz - Igaz vagy hamis?

Bevezetés

A modul célja, hogy a diákok megismerkedjenek a megerősítő tanulás (***RL***) alapfogalmaival és kifejezéseivel. A diákra épül, amelyek nemcsak a gyakori alkalmazásokat mutatják be, hanem azt az általános elképzelést is, hogy egy mesterséges intelligencia hogyan tanulhat önmagától, cselekvések végrehajtásával és kiértékelésével.

Valós világbeli példák

(2 - 6. dia)

A **prezentáció** az ***RL*** alapjaival kezdődik, és egy rövid **videó** segítségével az OpenAI önállóan tanuló bújócskázó mesterséges intelligenciáját mutatja be. Ha a diákok nem értenek elég jól angolul, akkor vagy automatikus feliratokat használhatunk, vagy a tanár fordíthatja/magyarázhatja a történéseket. A legfontosabb, hogy az ***RL***-ben a mesterséges intelligencia magától tanul, a programozók közvetlen irányítása nélkül.

A videó után a következő példa a **Leela Chess Zero** (***Lc0***), egy modern sakk mesterséges intelligencia, amelyet megerősített tanulással képeztek ki. Míg a hagyományos sakk-mesterséges intelligenciák statikus funkciókkal értékelték a táblaállásokat (például fix pontokat kaptak az alapvető különbségekért és a bábuállásokért), addig az ***Lc0*** több millió játszmát játszva megtanulta, hogy milyenek a jó pozíciók saját maga számára, és más játékosok ellen, és így jobb döntéseket tud hozni arról, hogy milyen lépéseket tegyen. Fontos leszögezni, hogy a modern sakkmotorok messze felülmúlják az emberi játékosokat, még a legjobb nagymestereket is meg lehet verni egy okostelefon segítségével.
A játékok azért elég népszerűek az ***RL***-kutatásban, mert jól megfogalmazott szabályok vannak, ami megkönnyíti a konkrét akciók és célok meghatározását, amelyek az ***RL*** működéséhez szükségesek. Ezt a példafóliák után részletesebben is kifejtjük.

Mivel a sakkmotorok még mindig többnyire hasonló módon számítják ki a lépéseket, mint a 90-es évek végén[3], amikor elkezdték az emberi játékosokat következetesen legyőzni, a játék "egyszerűsége" miatt néha lebecsülik őket. Az ***RL*** algoritmusok mégis fejlődtek, és ma már sokkal összetettebb forgatókönyvekben is képesek felvenni a versenyt az emberi játékosokkal. Ezért a harmadik példa a Google Deepmind kutatási részlegének mesterséges intelligenciájáról, az

AlphaStarszól, amely megtanult egy komplex modern számítógépes játékot játszani a legjobb emberi játékosokhoz hasonló szinten.[4] Tekintettel arra, hogy minden időpontban nagyjából 1026 lehetséges lépés létezik (szemben a sakkal, ahol az átlag kevesebb mint 40), megdöbbentő, hogy az AI képes kiválasztani a legjobbakat, miközben betartja ugyanazokat a korlátokat, mint az emberi játékos (korlátozott látás, reakcióidő).

Az utolsó egy gyakorlatiasabb példa az egyéni hirdetésekről (Ad). Bemutatja, hogy az ***RL*** sokféleképpen használható, ebben az esetben a bevételek optimalizálására azáltal, hogy megtanulja, mely hirdetésekre kattintanak a legtöbben (vagy húzzák fölé az egeret), és a jövőben megjelenő hirdetéseket úgy igazítja, hogy maximalizálja annak esélyét, hogy a felhasználó rákattintson egy hirdetésre.

Az ***RL*** felhasználási eseteire még sok más példa is van, mint például az önvezető autók képzése szimulációk segítségével, de ezek a példák elég jó kiindulópontot jelentenek ahhoz, hogy most azzal a kérdéssel foglalkozzunk, hogyan tanulhat egy algoritmus ténylegesen magától.

RL alapjai

(7 - 20. dia)

Ahhoz, hogy az ***RL*** alapjait megértsük, ismernünk kell legalább néhány definíciót, amelyeket az első néhány dián mutatunk be. Ezután a TicTacToe játékot használjuk példaként.

Ügynök

Az ügynök olyan intelligens valóságot ír le, amely kölcsönhatásba lép a környezettel, és ésszerű döntéseket kell hoznia. Ez a programnak az a része, amely magában foglalja a mesterséges intelligencia összes logikáját és tudását.

A TicTacToe-ban az ügynök az AI játékos lenne.

Környezet

Ahhoz, hogy egy ügynök tanulni tudjon, világosan meghatározott környezetre van szüksége. A környezet általában egy valós környezet valamilyen szimulációja (például egy szimulált utca egy autó számára, hogy megtanuljon vezetni) vagy egy játék.

A TicTacToe esetében a környezet a játéktábla, rajta a rajzokkal (X, O).

Akció

Az ügynöknek minden egyes időpontban el kell döntenie, hogy milyen cselekvést hajtson végre. A rendelkezésre álló cselekvéseket minden időpontban ismerni kell, az AI-nak ezután meg kell tanulnia kiválasztani a legjobbat.

A TicTacToe-ban minden üres lapkának megfelel egy olyan lépés, amely a lapkára egy (X, O) szimbólumot rajzol.

Állapot

Az állapot a környezet értelmes részeinek numerikus reprezentációja. Mivel a számítógépek csak számokkal dolgoznak, a környezet kritikus részei elvonatkoztathatók, hogy a mesterséges intelligencia számára érthetőek legyenek.

A TicTacToe-ban az állapot lehet egy szám a kilenc lapka mindegyikéhez (pl. 0 = üres, 1 = X, 2 = O), és esetleg annak száma, hogy ki van soron (1 vagy 2). Egy másik példában, az érmejátékban, amelyet a következő feladatban fogunk használni, az állapot egyszerűen az asztalon lévő érmék száma. Ez általában a legabsztraktabb definíció, a legtöbb ember számára a következő gyakorlat során válik majd érthetőbbé.

Jutalom

A jutalom általában egy numerikus érték, amely egy állapot minőségét vagy eredményét mutatja. Általában jó eredmény esetén pozitív, rossz eredmény esetén negatív.

A TicTacToe-ban a jutalom +1 lehet a győzelemért, -1 a vereségért és 0 minden másért.

Végül a tanulóknak át kell gondolniuk, hogy a következő példákban milyen ügynökök, állapotok, lépések és jutalmak szerepelnek. Ezt többféleképpen is meg lehet közelíteni, az egyszerű csoportos megbeszéléstől kezdve a kiscsoportos prezentációkig, a módszerek oldal további ötleteket tartalmaz.

Sakk

Az **ügynök** a sakkozó mesterséges intelligencia, az **állapot** a táblán lévő bábukat jelképező számértékekből áll. Az **akció** a lehetséges sakklépések, a **jutalom** pedig lehet egy nagyon magas/alacsony szám a győzelemért/veszteségért, és néhány szám a kettő között, ami azt jelzi, hogy az AI számára mennyire "jó" az állás.

Bújócska

Az **ügynökök** a rejtőzködők és a keresők, az **állapot** tartalmazza az ügynökök helyzetét, valamint az összes objektum helyzetét és azt, hogy azok le vannak-

e zárva vagy sem. A **cselekvések** közé tartozik a különböző irányokba történő mozgás, a forgás és a tárgyak megragadása/elengedése, valamint a tárgyak rögzítése. A **jutalom** lehet az a másodpercek száma, ameddig a rejtőzködőket nem találták meg, ami a keresők esetében negatív számként is használható.

Egyedi hirdetés

Az **ügynökök** lehetnek egyedi reklámspotok, az **aktuális** információk a reklám típusáról és arról, hogy a felhasználó rákattintott-e vagy sem. A **cselekvések** közé tartozhatna a konkrét témákra való váltás vagy a reklámok vagy reklámtípusok hirdetése. A **jutalom** lehet egy szám, amely jelzi, hogy a felhasználó rákattintott-e a hirdetésre vagy sem.

A valóságban nem könnyű, de nagyon fontos feladat megtalálni az állam és a rendelkezésre álló intézkedések megfelelő képviseletét. A rossz választás megnövekedett képzési időt és rosszabb teljesítményt eredményezhet, vagy akár meg is akadályozhatja az ügynök tanulását, mivel kulcsfontosságú információk hiányozhatnak. Ez azonban a következő feladatokban nem túl lényeges, mivel ott az állapotok és a cselekvések már egyértelműen definiálva vannak.

Q-Tanulás

(21 - 30. dia)

Az utolsó diák bemutatják a Q-tanulás fogalmát, valamint az ***RL*** interakciós hurkot. Ez az akció és reakció fogalmán alapul, ahol az ágens a környezetével kölcsönhatásba lép azáltal, hogy kiválaszt egy végrehajtandó akciót. Ez a cselekvés megváltoztatja a környezetet, és ennek eredményeként az ügynök megkapja az új állapot reprezentációját, valamint egy jutalmat, amely jelzi, hogy mennyire volt jó a cselekvés. Ezen új információ birtokában az ügynök ezután frissítheti viselkedését, majd új cselekvést választ a végrehajtáshoz. Ez a ciklus a cél eléréséig folytatódik.

Q-learning az ***RL*** interakciós hurkon alapul, és minden egyes **állapot** és állapotpárhoz tárol egy **minőségi értéket** (Q-érték). **művelet**. Ez az érték azt jelzi, hogy a műveletnek mennyire „jó” az aktuális állapota. Ezért egy állapot elérésekor a legmagasabb **Q-értékű** művelet választható az optimális viselkedéshez. Mivel a **Q-érték** nem biztos, hogy az elején helyes (sőt, gyakran minden művelethez véletlenszerű értékre van beállítva az elején), módosítható úgy, hogy növeli/csökkenti. a cselekvés végrehajtásáért kapott jutalomra tekintettel. Ha ezt a

folyamatot elég gyakran megismétlik, a **Q-értékek** elérnek egy pontot, ahol mindig a megfelelő műveletet hajtják végre.

Amíg nincs túl sok állapot-akció páros, ezeket az értékeket egy táblázatban lehet tárolni, ahol minden sor egy állapotból és az összes rendelkezésre álló akció Q-értékéből áll.

A 29. dián látható táblázat egy ilyen Q-táblázat egy kis részét mutatja be egy példaértékű jump-n-run játékhoz. A gyémánt elérését +1, a vízbe esést -1 jutalmazza. A következő, 30. dián látható táblázat ugyanezt a helyzetet mutatja, de egy fejlettebb táblázattal, amely a jövőbeni jutalmakat is beleveszi a cselekvésükbe. Ez azt jelenti, hogy egy olyan akció, amely egy akcióval később a vízbe esést eredményezheti, egy kis büntetést is kap.

A Q-értékek adaptálásának ezt a folyamatát nevezzük tréningnek. Bár néha megvalósítható, hogy egy mesterséges intelligenciát normál használat során képezzenek ki, a képzési folyamat gyakran független a végső felhasználástól, mivel több millió ismétlésre lehet szükség az értelmes értékek létrehozásához.

Mindezen elmélet után itt az ideje, hogy a tudást valós példákon teszteljük, ami a következő három, egymásra épülő feladathoz vezet.

Tananyag

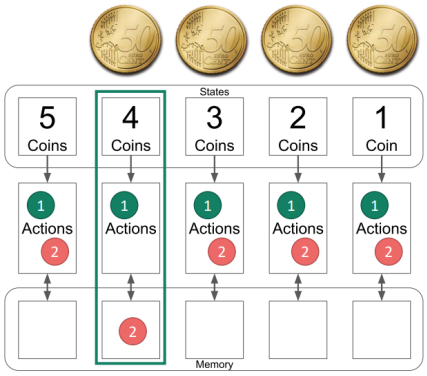
- RL - Introduction.pdf

Hivatkozások

1. https://openai.com/blog/emergent-tool-use/
2. https://lczero.org
3. https://www.chess.com/article/view/deep-blue-kasparov-chess
4. https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii

Érme játék

Ez a gyakorlat bevezetést nyújt az ***RL*** tanulásba az érmejáték segítségével. Az alapkoncepció megértéséhez a **diák** bevezetésként használhatók. A játékhoz valamilyen zsetonra, például érmékre, bábukra, jutalomfalatokra, ... van szükség, mégpedig nagy mennyiségben, hogy minden tanulópár legalább 5 darabhoz jusson. A játék úgy kezdődik, hogy 5 érme van az asztalon, és a játékosok felváltva elvesznek belőle egyet vagy kettőt. Az utolsó érmét elvevő játékos veszít.



1. A játék bemutatása

Mutasd be a játékot az 1-8. dia segítségével, és hagyd, hogy a tanulók párban játsszanak néhány mérkőzést.

2. Az AI bemutatása

Mutassuk be a mesterséges intelligenciát a 9-35. dia segítségével. Javasoljuk, hogy az anyagot (táblát és akciókat) előzetesen adjuk át, hogy a tanulók láthassák, hogyan néz ki minden a valóságban, és aktívan játszhassanak a példákkal.

3. Hagyjuk őket játszani

Ezután hagyjuk, hogy a diákpárok úgy játsszanak, hogy az egyikük az AI szerepét veszi át, a másikuk pedig normálisan játszik. Ahogy a játék folytatódik, egyre több és több akciót távolítanak el a mesterséges intelligenciától, amíg a győztes akciókon kívül más nem marad.

4. A legfontosabb tudnivalók

Végezetül beszéljünk a következő legfontosabb tudnivalókról:

Mi történik, ha bizonyos állapotokat soha nem érünk el (pl. a játékos mindig ugyanazokat az akciókat választja)?

Csak az elért állapotokból tanulunk. Ha a mesterséges intelligencia nem találkozik egy helyzettel a képzés során, előfordulhat, hogy nem találja meg a helyes lépést.

Ebben a példában az AI hol tanul többet, a győzelem vagy a vereség által?

Vesztéssel, ebben a példában a mesterséges intelligencia csak a vesztésért kap büntetést, a győzelemért soha nem kap jutalmat, ezért a győzelemnek nincs hatása.

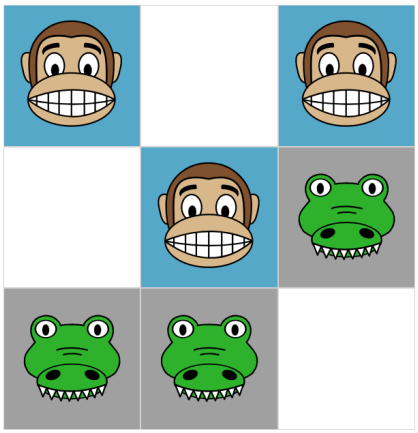
Megváltoztathatnád a játékot úgy, hogy a győzelem is adjon valamilyen jutalmat?

Igen, például új akciókövek hozzáadásával, ha a mesterséges intelligencia nyer. Akkor a mesterséges intelligencia nagyobb valószínűséggel választja majd a sikeresebb akciók egyikét. Ezt a következő feladatban fogjuk bemutatni.

5. Tananyag

- RL - Coin Game.pdf
- RL - Coin Game Board.pdf
- RL - Coin Game Actions.pdf

Hexapawn



Ez a feladat az előző (**érmejáték**) ismereteire épül, és a **https://www.stefanseegerer.de/schlag-das-krokodil/** weboldalon és a **Hexapawn** játékon alapul. A **Hexapawn** két játékos játszik egymás ellen egy 3x3-as táblán, kizárólag a sakkból ismert gyalogokat használva, amelyek vagy egy mezőt léphetnek egyenesen előre, vagy átlósan üthetnek le egy ellenséges gyalogot. A játékos akkor nyer, ha vagy egy gyaloggal eléri az ellenkező oldalt, vagy ha megakadályozza az ellenfél lépését (blokkolja az összes gyalogot).

1. A weboldal bemutatása

Először is mutassa be a weboldal képességeit, és ismertesse a játékszabályokat. Különösen a **Válaszidő** és a **Csak lehetséges lépések** beállításait kell elmagyarázni, mivel ezek segítenek a következő feladatokban. Továbbá elengedhetetlen, hogy a tanulók megértsék a színes pontok jelentését (azonos színű akciónak felelnek meg), és lássák, hogy ezek a pontok eltávolíthatók vagy hozzáadhatók.

2. Hagyjuk őket játszani

Most itt az ideje, hogy a diákok egyedül játsszanak. A cél az, hogy minél többször nyerjenek, mielőtt az AI, annyira tapasztalt nem lesz, hogy nem lehet legyőzni. Ez egyszerű feladatnak tűnik, de a diákok hamarosan rájönnek, hogy a 10 vagy akár 20 győzelem feletti eredmény eléréséhez jól meg kell érteniük a belső működését. **Vigyázz, ha az oldal újratöltődik, a győzelmek is törlődnek!**

3. A legfontosabb tudnivalók

A legfontosabb tanulságok hasonlóak az érmejátékhoz, de hangsúlyosabbak, mivel a tanulóknak kifejezetten olyan cselekvéseket kell végrehajtaniuk, amelyeket korábban nem használtak, hogy az AI-t ismeretlen területre kényszerítsék. Az is látható, hogy a lehetséges állapotok száma elég gyorsan növekszik a

rendelkezésre álló akciók számával. Könnyen elképzelhető, hogy egy nagyobb táblán (mint például egy sakktáblán) annyi lehetséges állapot van (nagyjából 10^{44})[1], hogy nem kivitelezhető egy mesterséges intelligencia kézzel történő betanítása, vagy akár az összes lehetséges állapot figyelembevétele.

> Ha a lehetséges állapotok száma túl nagy lesz, más módszereket kell alkalmazni, például az állapotok számának csökkentését értékelő függvények használatával, hogy kizárjuk a nemkívánatos állapotokhoz vezető műveleteket, vagy a Q-értékek **neurális hálózatok** segítségével történő közelítését.[2][3].

4. Tananyag

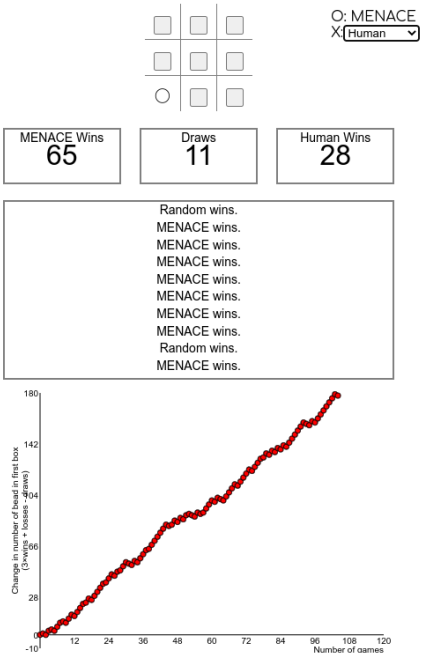
- https://www.stefanseegerer.de/schlag-das-krokodil/

5. Hivatkozások

1. https://tromp.github.io/chess/chess.html
2. https://www.chessprogramming.org/Evaluation
3. https://www.turing.com/kb/how-are-neural-networks-used-in-deep-q-learning

MENACE

Ez a gyakorlat ismét az előzőre (**Hexapawn**) épül, és egy még több állapotot tartalmazó játékot mutat be, mint a TicTacToe-t. Jó bevezetésként használható ennek a videónak az első része, **video from Matt Parker about MENACE**. Ez elmagyarázza a MENACE alapötletét, amely hasonló a korábbi Hexapawn példához, de ahelyett, hogy digitálisan dolgoznának, fizikai gyufásdobozokban tárolják a színes gyöngyöket.



Bár érdekes ötlet lehet egy ilyen rendszer létrehozása az osztállyal, valószínűleg elég sok időbe fog telni, nem csak a létrehozása, hanem a betanítása is, hogy értelmes eredményeket kapjunk. Ezért ebben a gyakorlatban a rendszer **online simulator** fogjuk használni.

1. A MENACE bemutatása

Először mutassuk be a MENACE rendszert, egy ***RL*** TicTacToe gépet, ehhez felhasználva **video from Matt Parker** első ~2:20 percét (vagy több), vagy azzal, hogy megbeszéljük (akár csoportmunkában), hogyan lehetne az előző példát módosítani a TicTacToe játékhoz.

2. A weboldal bemutatása

Ezután mutassuk be, hogyan kell használni a szimulátort a **https://www.mscroggs.co.uk/menace/**weboldalon, magyarázzuk el különösen a jobb oldali számok és rendszerek jelentését (állapotok és az akciók q-értéke, magasabb = nagyobb valószínűséggel választják, a tükrözött állapotok ki vannak zárva), és hogyan játszhatunk saját magunk, vagy hogyan engedhetjük az AI-t más AI-k ellen játszani (például a véletlenszerűen játszó AI vagy egy tökéletesen játszó AI ellen).

3. Hagyjuk, hogy a saját AI-t képezzék ki

A honlap használatára vonatkozó ismeretek birtokában a diákoknak most a saját mesterséges intelligenciájukat kell kiképezniük, amelynek a lehető legjobbnak kell lennie.

4. Beszéljük meg a problémákat

A diákok hamarosan rájönnek, hogy még ha ez könnyű feladatnak is hangzik, valószínűleg egyik megközelítés sem vezet tökéletesen játszó mesterséges intelligenciához. Ennek fő okai a következők:

- Az AI egyszerűen nem találkozik bizonyos állapotokkal a gyakorlás során, így nem tudja, hogy mi a jó lépés, amikor találkozik ezekkel. Ez történik például akkor, ha egy tökéletesen játszó AI ellen képezzük ki, ahol soha nem tanulja meg, hogyan nyerhet valójában, így nem fogja tudni, hogyan használja ki a hibákat, amikor azok elkövetésre kerülnek.
- Az AI megtanulja a rossz lépéseket, mert az ellenfél nem reagált megfelelően. Például megtanulhatta, hogy egy lépés általában győzelemhez vezet, mert az ellenség csak rossz (vagy véletlenszerű) lépéseket tett, de ha az ellenség helyesen játszik, az AI veszíteni fog.

A technikai probléma e mögött a feltárás és a kiaknázás között áll, amelyet a következő feladatban részletesebben is megvizsgálunk.[1]

5. Tananyag

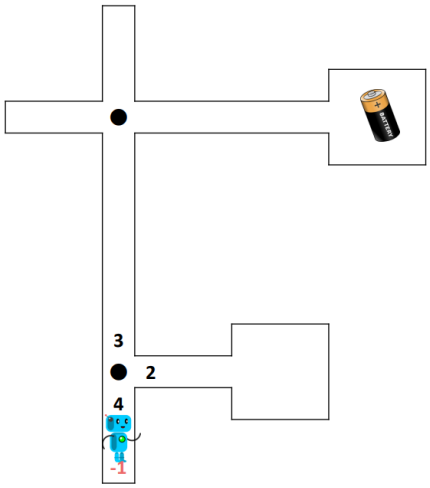
- https://youtu.be/R9c-_neaxeU
- https://www.mscroggs.co.uk/menace/

6. Hivatkozások

1. https://ai-ml-analytics.com/reinforcement-learning-exploration-vs-exploitation-tradeoff/

Útvesztő

Egy olyan **útvesztőben**, amelyben a **robotot** egy **elem** megtalálásáért jutalmazzák, megmutatja, hogy az ***RL*** nem olyan egyszerű, mint gondolnánk. A gyakorlat ezzel a **diával** kezdődik, amely egy labirintusban lévő robotot mutat be, amelynek egy elemet kell megtalálnia. Kezdetben a robot véletlenszerű fordulatokat tesz, majd idővel megtanulja a céljához vezető helyes utat. A második gyakorlatban a labirintusban több különböző értékű elem van, ami nem vezet optimális eredményekhez, ahogyan azt az utolsó néhány dián elmagyarázzuk.



1. A játék bemutatása

Először mutassuk be a diákoknak a játékot az 1-18. diasegítségével. Ezután adjunk minden diáknak (vagy diákpárnak) egy példányt az **RL Maze Basicből**, egy kockát és egy tollat.

2. Hagyjuk őket játszani

Ezután a tanulóknak több kört kell játszaniuk, amíg az akkumulátorhoz vezető út világosan ki nem rajzolódik. Az értékek frissítéséhez egyszerűen áthúzhatjuk, és a közelébe írhatjuk az új értéket. Néhány diáknak "szerencséje" lesz, és az útvonal nagyon gyorsan kialakul, míg másoknál a robot minden zsákutcába belemegy, mielőtt elérné a célját. Ha néhány tanuló túl gyorsan halad, akkor megpróbálhatják újra ugyanannak az útvesztőnek egy új lapjával, és megnézhetik, hogy másodszorra hogyan viselkedik másképp.

3. A haladó útvesztő bemutatása

Adjunk minden diáknak (vagy diákpárnak) egy példányt az **RL Labirintus haladó** változatából, és ismét hagyjuk, hogy betanítsák a robotjukat. Még ha ezúttal több elem is van, a robot akkor is visszamegy az elejére, amikor egy elemhez ér. A játéknak ismét vége, ha a robotnak szabad útja van az egyik elemhez, még akkor is, ha az nem a legmagasabb értékkel rendelkező elem.

4. A probléma feltárása és megoldási javaslatok megbeszélése

Amikor mindenki végez, a különböző robotok különböző pontszámokat érnek el. Most hozzunk létre kb. 4 fős csoportokat, és hagyjuk, hogy megvitassák, az egyes robotok miért értek el jobb eredményt, mint mások, és dolgozzanak ki egy lehetséges megoldást. Segíthet, ha a különböző pontszámú robotokkal rendelkező tanulókat egy csoportba soroljuk, így jobban szemléltethetik a problémát. Ezután minden csoportnak be kell mutatnia a megállapításait és a megoldásokat.

5. A probléma kifejtése

Végezetül a **19-26. dia** segítségével vitassuk meg a feltárt problémákat, és a felhasználás lehetséges megoldásait.

6. Választható: Különböző megoldások tesztelése

Adjunk új példányokat az **RL Maze Advanced** labirintusból, hogy a csoportok kipróbálhassák a különböző új megközelítéseket (például a különböző felfedezési sebességeket).

7. Tananyag

- RL - Maze.pdf
- RL - Maze Basic.pdf
- RL - Maze Advanced.pdf

8. Hivatkozások

1. https://ai-ml-analytics.com/reinforcement-learning-exploration-vs-exploitation-tradeoff/

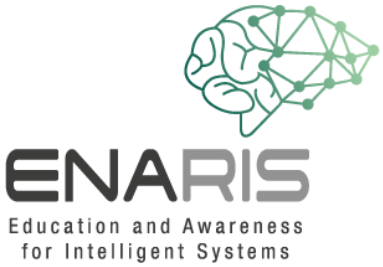
Igaz vagy hamis?

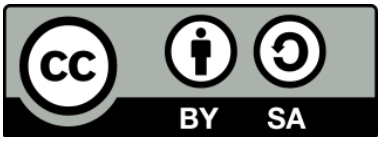
Ez az utolsó fejezet az egész modul összefoglalójaként szolgál. Tíz igaz vagy hamis kérdésből álló kvízre épül, és arra ösztönzi a tanulókat, hogy gondolják át a tanultakat.

Maga a **kvíz diákból áll**, amelyek egymás után mutatják be a kérdéseket, majd a helyes válaszokat. Ezért a diákok leírhatják válaszaikat, majd a végén kiszámíthatják pontszámukat. Erősen ajánlott minden kérdésnél a válaszok felfedése után szünetet tartani, hogy röviden megbeszéljük, miért helyes vagy helytelen a válasz. Alternatívaként ez a feladat csoportokban is elvégezhető, hogy tovább ösztönözzük a tanulók közötti vitát.

Tananyag

- RL - Quiz.pdf


ENARIS
Education and Awareness
for Intelligent Systems


BY SA


EUROPEAN UNION


Interreg
Austria-Hungary 2014-2020
European Union – European Regional Development Fund


TU
Graz


mobilis
interaktív kiállítási központ


OESTERREICHISCHE
COMPUTER GESELLSCHAFT
AUSTRIAN
COMPUTER SOCIETY