***Projektmunka I - II.***

*Mesterséges intelligencia készítése különböző játékszituációk megoldására*

***Szerző:*** *Mayer Barna*

Tartalomjegyzék

[*1. Bevezetés* 3](#_Toc102983455)

[*2. Mesterséges intelligencia* 3](#_Toc102983456)

[*2.1 Gépi tanulás* 3](#_Toc102983457)

[*3. Gépi tanulási módszerek a gyakorlatban* 4](#_Toc102983458)

[*3.1 Neurális Hálózatok* 4](#_Toc102983459)

[*3.1.1. Mesterséges- és biológiai neurális hálózatok kapcsolata* 4](#_Toc102983460)

[*3.1.2. Felépítésük* 6](#_Toc102983461)

[*3.1.3. Tanulási folyamat* 6](#_Toc102983463)

[*3.1.4. Egyéb népszerű neurális hálózatok* 7](#_Toc102983464)

[*4. Alkalmazható technológiák* 9](#_Toc102983465)

[*4.1. Megerősítéses tanulás keretrendszerek, eszköztárak* 10](#_Toc102983466)

[*4.1.1. OpenAI Gym toolkit* 10](#_Toc102983467)

[4.1.2 TensorFlow keretrendszer 10](#_Toc102983468)

[4.1.3. Theano keretrendszer 11](#_Toc102983469)

[4.1.4. Torch 12](#_Toc102983470)

[4.1.5. Egyéb machine-learning könyvtárak 13](#_Toc102983471)

[*4.2. A piacon jelenlévő Football Environment – ek* 13](#_Toc102983472)

[*4.2.1 Google Research Football API* 13](#_Toc102983473)

[*4.2.2 FIFA* 15](#_Toc102983474)

[*4.2.3 További elérhető környezetek* 15](#_Toc102983475)

[*4.3. Az ágensek általános tanító módszerei* 15](#_Toc102983476)

[*4.3.1 Q-Learning* 16](#_Toc102983477)

[*4.3.2 SARSA* 17](#_Toc102983478)

[*4.3.3 Multiagent learning* 17](#_Toc102983479)

[*4.4. Rendelkezésre álló, hasonló megoldások áttekintése* 19](#_Toc102983480)

[*4.4.1 Chintan Trivedi féle megoldás* 19](#_Toc102983481)

[*4.4.2.* Vishal *Bidawatka megoldása* 20](#_Toc102983482)

[5. Feladat specifikáció 21](#_Toc102983483)

[*5.2. A szimulációs környezettel szemben állított követelmények* 21](#_Toc102983484)

[*5.1. Az ágensekkel szemben támasztott követelmények* 21](#_Toc102983485)

[*Irodalomjegyzék* 21](#_Toc102983486)

# *1. Bevezetés*

*A projektmunkám célja, egy olyan rendszer implementálása, amely különböző labdarúgó játékszituációkra képes botokat betanítani mesterséges intelligencia használatával.   
Ahhoz, hogy a projekt megvalósításával kapcsolatban további részletekbe bocsátkozzak, alapos irodalomkutatás szükséges. Ennek keretein belül a célom, a kapcsolódó témakörökkel minél alaposabban megismerkedni, a gépi tanulási módszerek, algoritmusok felkutatása és elsajátítása, illetve ezeknek a használata, implementálása egy, a kutatás után választott szimulációs környezetben.*

# *2. Mesterséges intelligencia*

*Ennek a témakörnek a feldolgozásával temérdek mennyiségű szakirodalom foglalkozik, amelyből jól kivehető, hogy maga a téma sok szerteágazó tudományágat is magába foglal, definiálása pedig egyáltalán nem egyszerű, az évek során folyamatosan változott. [1] Definícióját tekintve négy különböző kategóriát is meghatároztak, melyek között különbséget téve különböző képességű rendszerek lettek meghatározva. Ilyen rendszerek: az Emberi módon gondolkodó rendszerek, Emberi módon cselekvő rendszerek, Racionálisan gondolkodó rendszerek és a Racionálisan cselekvő rendszerek.*

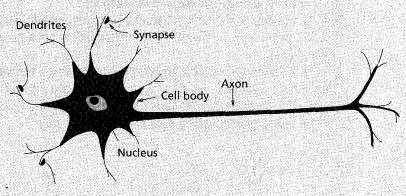
## *2.1 Gépi tanulás*

*A mesterséges intelligencia szerteágazó ágai közül az egyik a* ***gépi tanulás*** *(machine learning) amely a számítási algoritmusoknak egy olyan fejlődő iránya, amelyek arra lettek tervezve, hogy a környezet tanulmányozása révén, lemásolja az emberi intelligenciát, azaz a tapasztalatokból tudást generáljanak. [2] Ennek következtében ez a módszer kiválóan alkalmazható egy olyan játék-intelligencia készítéséhez, amely betanítása után képes sikeresen megoldani, a football-ban előforduló különböző játékszituációkat, legyen szó rögzített játékhelyzetről, ellentámadásról vagy sima támadásról.*

# *3. Gépi tanulási módszerek a gyakorlatban*

## *3.1 Neurális Hálózatok*

### *3.1.1. Mesterséges- és biológiai neurális hálózatok kapcsolata*

*A mesterséges neurális hálózat létrehozásának az ötlete a biológiai neurális hálózatok felépítésén alapszik. Ezek a biológiai hálózatok úgy nevezett neuronokból épülnek fel, amelynek mindegyike egy-egy információ feldolgozó egységnek tekinthető. [3] A neuronok működésüket tekintve, a dendritjein keresztül tudnak más idegsejtektől érkező jeleket, impulzusokat fogadni, amelyeket aztán a, a transzmittere mentén a sejttest irányába továbbítanak. Ezt követően ezeket az ingerületeket az axon azaz a nyúlvány a sejttesttől távolodó irányba vezeti, amely a végén szálakká, részszálakká bomlik szét. [3] Az emberi agy nagy számban tartalmaz ilyen neuronokat, amely idegsejtek egy másikhoz kapcsolódva – a szinapszisokon keresztül – nagyon összetett idegi hálózatot alkotnak. Köszönhetően annak, hogy ezek a hálózatok képesek a párhuzamos számítások elvégzésére, illetve a szétágazó információk feldolgozására, az embereknek megvan a képességük a tanulásra. [4]   
Az ezen a területen megszerzett tudást hasznosították a terület kutatói a mesterséges neurális hálózat koncepciójának megfogalmazásakor, annak reményében, hogy az emberi öntanuló képességet számítógépes A képen szöveg, óra látható

Automatikusan generált leíráskörnyezetben is hasznosítani lehessen [4] McCulloch és Pitts [5] a neuronok számítási modelljének leírásához bináris küszöbegységeket javasoltak. A model két részre osztható, egy beviteli és egy töntéshozói részre. Működésük egyszerűen úgy írható le, hogy a bemeneti részben definiált, bizonyos számú, súlyozott bemeneti jelre egy kimenetet adnak (y), ha ezeknek a jeleknek az összege meghaladja az előre definiált u értéket. Ellenkező esetben a visszatérési értékük 0. Ezt küszöbfüggvénnyel tudjuk ábrázolni. [3]   
A neuronok matematikai működését az alábbi egyenlettel tudjuk leírni:*

1. ábra: Egy biológiai neuron felépítésének vázlata, forrás: [3]

2. ábra McCulloch és Pitts neuron modellje, forrás: [3]

*ahol,* θ *az egységugrás függvényt jelenti a 0 helyen, xj \* wj a j-edik ingerfelvevő és az ehhez a bemenethez tartozó súlynak a szorzatát adja meg, u pedig a határértéket jelöli. [3] Jain egyébként említést tesz arról, hogy a -u határértéket w0 -ként is szokták jelölni az egyszerűség kedvéért, amelyre x0 = 1 értékű konstans bemenettel hivatkoznak. Továbbá leírja, hogy a különböző súlyoknak más hatása van a neuronok viselkedésére: a pozitív súlyok serkentő szinapszisoknak felelnek meg, míg a negatív súlyok a gátló szinapszisokat modellezik.[3] A McCulloch és Pits által leírt modell azonban nem volt tökéletes, mivel számos olyan egyszerűsítő felvetést tartalmaz, amely a valódi biológiai neuronok működését nem jellemzi. [3] A legfőbb probléma a modellel az volt, hogy nem vették figyelembe az időt, így az azonnal reagált a bemenetekre, míg ez a valós neuronoknál térben és időben történik. [6] Számos módon általánosították az évek alatt, a problémát pedig a küszöbfüggvények, aktivációs függvényekre való cserélésével orvosolták. Ilyen aktivációs függvények például a szakaszos lineáris, a Gauss, és a legelterjedtebb a szigmoid.[3]*

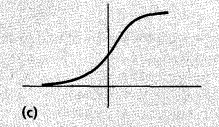
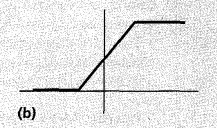
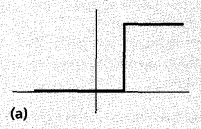
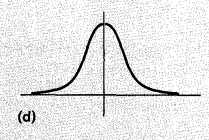
d) Gauss

c) Szigmoid

b) Szakaszos lineáris

a) Küszöbfüggvény

3. ábra Küszöbfüggvény és az említett aktivációs függvények, forrás: [3]

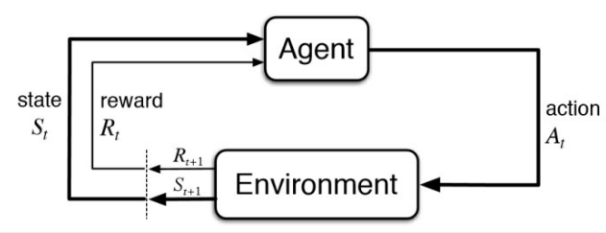


### *3.1.2. Felépítésük*

4. ábra A neurális hálózat 3 fő rétege, forrás [7]

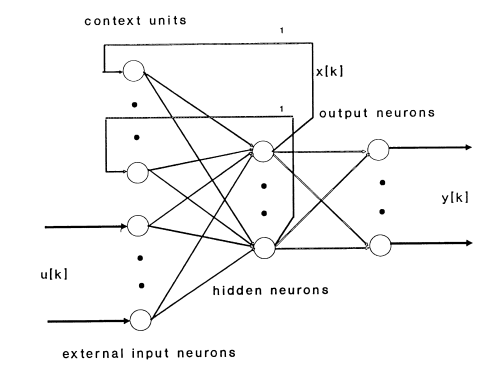
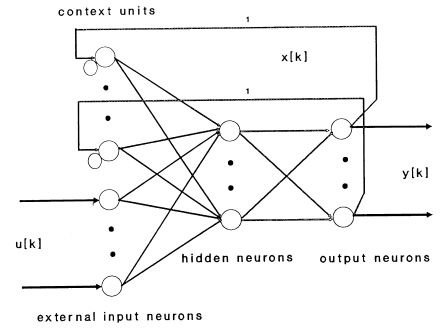
### *Felépítésüket tekintve a neurális hálózatok rétegekből állnak. Ezekben a rétegekben számos egymással kölcsönhatásban álló neuron tevődik össze, amelyek tartalmazzák a már említett aktivitási függvényt. A hálót rétegekre bontva, három fő réteg állapítható meg: a bemeneti, a köztes vagy rejtett és a kimeneti réteg. [6] A bemeneti rétegben olyan neuronok találhatóak, amelyek ismert információkból, vagy előre betáplált változókból állnak. [6] Ezek meghatározzák a hálózat bemeneteit, viszont semmiyen műveleteket sem végeznek.[7] Ahogy a 4.ábrán is jól látható, minden egyes bemeneti neuron kapcsolódik egy köztes réteg béli neuronhoz, amely kapcsolatokat a bemeneti neuronok fontossága szerint súlyozzák. [6] Ezeknek a súlykiosztása eléggé feladat specifikus, általában az alapján kerül kiosztásra, hogy az adott feladat szerint mely tulajdonságok a fontosak.ds A köztes rétegben is ugyanúgy neuronok kapcsolódnak neuronokhoz, viszont annyi különbség van a bemeneti réteghez képest, hogy a kapcsolatok súlya folyamatosan változik a neuronok eredményeitől függően. A súlyozott kapcsolatokat megtartva kimeneti réteg az eredményneuronokat tartalmazza, amelyből akár több is lehet. [6,7]*

### *3.1.3. Tanulási folyamat*

*Annak ellenére, hogy a tanulás pontos definícióját nehéz meghatározni ilyen mesterséges rendszereknél, az ANN esetében ez a folyamat lehet akár a hálózati architektúra és a csatlakozó súlyok frissítésének problémája is. [3] Ezek a hálózatok példákon, tanítómintákon keresztül tanulnak. Ehhez általában tartalmaznak valamilyen tanulási szabályt, algoritmust, amely a kapcsolati súlyokat változtatgatja a bemenet alapján. [3,6] Ha először látunk el egy hálózatot tanító mintával, akkor az először véletlenszerű találgatással keresi a lehetséges megoldást, amelynek eredményét összevetve a mintával, megfelelően átállítja a belső réteg súlyozását. [6] Röviden összefoglalva tehát, egy hálózat tanulási képességén a súlyok véletlenszerű változtatásait értjük.  
Betanításukat tekintve három féle környezeti modellt különböztetünk meg amelyekben operálnak. Ezeket a modelleket tanulási paradigmának is szokás hívni. A három legfontosabb ezek közül a felügyelt, felügyelet nélküli és ezeknek az ötvözete, a hibrid tanulás. [3] Ezen módszerek között a legfőbb különbségnek a tanító minta minőségbeli eltérése tekinthető. Felügyelt tanulás esetében a hálózat minden egyes bemeneti mintájára rendelkezésre áll a helyes válasz (kimenet), ez pedig az elérhető információmennyiséget figyelembe véve a legjobb. Ennek következtében a háló feladata a lehető legnagyobb pontossággal megközelíteni az elvárt eredményt, a súlytényezők állítgatásával. [3]   
Ennek a tanulási modellnek az egyik változata a megerősítésen alapuló tanulás, amelynél a hálózat nem tudja, hogy mi az elvárt eredmény és az optimális végrehajtás, viszont azt igen, hogy milyen feladatot kell végrehajtania és mi az ennek a megoldására szánt helyes művelet. [8] Magát a tanulás folyamatát jutalmazási és büntetési rendszer használatával valósítja meg, ezzel jelezve, hogy a megtett művelet az jó-e vagy sem. [8] Működését tekintve rendelkezésünkre áll egy tanuló ágens, ami a legmegfelelőbb döntések meghozataláért felel, a korábbi tapasztalatai, illetve a jelen helyzete alapján. Ez az ágens a gépi tanuláson belül lehet akár egy személy, gép vagy akár szoftver is. Egy működési ciklust figyelve az ágens figyelemmel kíséri a környezet állapotát, ehhez kiválasztja a végrehajtandó műveletet, majd végre is hajtja azt. Ezt követően állapota megváltozik, először megfigyel, majd a ciklust megismételve újra olyan műveletet választ, amely szerinte a legjobb. [8] A legmegfelelőbb művelet kiválasztása, a már korábban említett jutalmi rendszeren alapul, ennek megfelelően az ágens célja a megszerzett jutalom maximalizálása.   
Visszatérve a fő tanulási módszerekhez, a felügyelet nélküli tanulás esetében nem társítunk minden egyes bemenetéhez egy adathalmazt. [3] Feladatuk, hogy feltárják az adatok mögöttes szerkezetét, vagy az adatok mintái között lévő korrelációkat és ezeket kategóriákba rendezze. [3]  
Hibrid tanulás esetén, a neurális hálózat ötvözi a felügyelt, illetve, felügyelet nélküli tanulás módszerét. Míg a hálózat súlytényezőinek egy részének meghatározása továbbra is felügyelt tanulás használatával lesz lehetséges, addig a másik részét irányítás nélkül derítik fel. [3]  
Összefoglalva tehát, a hálózatok gyakorlati alkalmazása úgy írható le, hogy a már betanított háló olyan adatokat kap, amelyekkel még nem találkozott, de a betanításból eredő tapasztalatai alapján képes megoldani a feladatot.*

5. ábra Megerősítéses tanulás folyamata, forrás: [8]

### *3.1.4. Egyéb népszerű neurális hálózatok*

Az előző fejezetben bemutatásra került elméleti modellek mellett, a gyakorlatban való alkalmazáskor is több hálózatot különböztetjük meg, a tulajdonságaik alapján. Ilyen jellemzők a topológia, neuron terjedési irány, rétegek, valamint a neuronok összetétele.  
Egy hálózat topológiáján azt értjük, hogy abban hogyan helyezkednek el a neuronok és hogyan vannak azok összekapcsolva. Főbb különbséget a hálót alkotó rétegek számának (mélysége) eltérése jelenthet. [14]  
 A gépi tanulás esetében alkalmazott, nagy rétegszámmal jelenlévő hálózatokat, mély neurális hálóknak hívjuk. [15] A hálózat tartományán belül azt a területet, ahol több mint három neuron réteg található, mélytanuló rétegnek nevezik. [15] Általánosságban, a mélytanulásban használt rétegek manapság öttől - több mint ezerig is terjedhetnek. Ennek az adottságának köszönhetően a DNN-ek sokkal inkább képesek bonyolultabb magas szintű funkciók megtanulására, mint a sekélyebb idegi hálózatok. [15]  
Másik gyakran alkalmazott topológiai tulajdonság, ha a hálózat rendelkezik konvolúciós réteggel [17], ekkor pedig konvolúciós neurális hálózatról [16] beszélünk. Ezek a hálózatok (CNN – Convolutional Neural Networks) egészen eltérnek az eddig taglaltaktól, mivel elsősorban képfeldolgozási funkciókkal rendelkeznek, de emellett más típusú bemenetet is képesek kezelni (hang, videók). Ha működését röviden szeretnénk összefoglalni, akkor a hálózat bemenetként kapott adatait az input réteg egy tulajdonság map-re leképezi, a rejtett réteg (ahol egyébként a konvolúciós rétegek elhelyezkednek) ezekből a leképezésekből újabb map-eket állít elő, majd a kimeneti rétegben kerül megállapításra, hogy az így kapott leképezések, melyik előre definiált kategóriába esnek. [16] A továbbiakban ennek a hálózatnak a működését jelen írás nem részletezné, hiszen a gyakorlati feladat szempontjából nem lesz szükség képfeldolgozásra.   
Annak függvényében, hogy az információ milyen irányban terjed át a neuronokon, illetve milyen kapcsolat van a neuronok között, megkülönböztetjük az előrecsatolt, illetve a recurrent – azaz visszacsatoló hálózatokat. [3] Ha a hálózat nem tartalmaz kimeneti visszacsatolást a bemenet felé, tehát a neuronszerkezete körmentes gráfot alkot, akkor előbbi hálózatról beszélünk. Ekkor az információ a bemeneti neuronok irányából a kimenet irányába áramlik. [9] Ilyen feed-forward többrétegű rendszer látható a 4.ábrán. A hálózatra jellemző a backpropagation nevű eljárás, amely a hibákvisszafelé terjedését jelenti a neuronok között, gradiens süllyedés segítségével. [12] Az alapötlet a hálózat működéséből ered, amely az információ előre irányba való áramlásakor a hálózat megbecsül egy célértéket. Ezt követően kiszámítjuk, hogy az utolsó rétegben becsléseink milyen messze vannak a tényleges célpontoktól. Ez a már korábban említett hiba, amelyet rekurzív módon az előző rétegbeli neuronokra is kiszámítjuk, majd a hálózat súlyait ezeknek az értékeknek a függvényében változtatjuk. [13] Fő célja ennek az eljárásnak, hogy a rendszer hibáját minimalizálni tudjuk. Összefoglalva tehát, ezek a hálózatok iterációjuk során magukat tanítják meg, hogy mely elemek milyen fontosságban járulnak hozzá a probléma megoldásához. [13] Fő alkalmazási esetüket illetően, általában a csoportosítási és regressziós előrejelzés feladatok megoldásánál jelennek meg.  
Egyéb esetben az utóbbi hálózatról beszélünk, ahol a visszacsatolás két módon történhet. Az egyik ilyen megközelítés, az Elman által bevezetett megoldás, amelynél a visszacsatoló ág a rejtett rétegtől a bemeneti réteg kontextusrészéig van jelen. A másikat pedig, a Jordan féle recurrent neurális hálózatok valósítják meg, ahol a visszacsatoló ág a kimeneti rétegből vezet a bemeneti csomópontok felé. [10] Ezeknél a hálózatoknál az információterjedés hurkokon keresztül vezet, így egy neuron a lehetséges kimenet előállításakor képes figyelembe venni a korábbi kimenetre adott választ (memóriával rendelkeznek). [10]

6. ábra Az eredeti Elman-féle hálózat felépítése, forrás: [11]

7. ábra Az eredeti Jordan-féle hálózat felépítése, forrás: [11]

Végezetül érdemes megemlíteni, habár a gépi tanulásban a legelterjedtebb módszer a neurális hálózatok használata, léteznek más megoldásokon alapuló módszerek is.   
Ilyen megközelítés lehet, ha véletlenszerű erdőket használunk, amelyek döntési fák együtteséből állnak, és ezek mindegyikét a betanítási adatkészlet véletlenszerű részhalmazával képzik ki. [18]  
Másik népszerű felügyelettel végrehajtott tanulási módú megoldás a Support Vector Machines (SVM) [19], amelyek megpróbálják a bemeneti adatokat egy olyan térbe leképezni, ahol lineárisan szétválaszthatók különböző kategóriákra. Ebből következik, hogy lineáris klasszifikációs problémák megoldásánál szokás használni.   
További opció lehet a K-adik Legközelebbi Szomszéd algoritmus (KNN) [21] használata, amely a betanítási adatszerkezetben megkeresi azokat az értékeket, amelyek a legközelebb állnak egy új bemenethez, és ezekhez a legközelebbi szomszédokhoz társított célváltozókat egy új predikcióban egyesíti. A gyakorlatban akkor érdemes ezt a módszert választani, ha adott egy egyed párok távolságát/hasonlóságát leíró metrika vagy ha az egyes osztályokat elválasztó döntési felület nagyon bonyolult.  
Utoljára említést érdemel a szimbolikus regresszió analízis [20] módszerének alkalmazhatósága is. Ez olyan feladatoknál hasznos, ahol a cél az, hogy egy explicit matematikai képletet találjunk a hálózat bemenetei és kimenetei összekapcsolására.

Az előbbi összefoglalóból könnyen belátható, hogy egy football MI készítéséhez a legmegfelelőbb választás egy olyan neurális hálózat lehet, amely a tanulása szempontjából megerősítéses tanulás használatán alapul.

# *4. Alkalmazható technológiák*

Célszerű lehet a megoldandó feladat megvalósítási tervének elkészítése előtt átgondolni, hogy implementálás közben, milyen már létező technológiák állnak a rendelkezésünkre, azok közül melyek használhatóak fel. Habár egy szoftver megvalósításának az alapja maga a programozási nyelv, amelyben íródott, először érdemesebb lehet konkrét nyelvi implementációktól független keretrendszereknek utána járni, majd ezeknek az összeilleszkedését, kompatibilitását vizsgálni. Ezeknek függvényében pedig már egész egyszerű feladat, implementálási nyelvet választani.

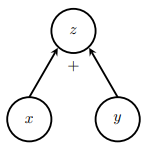
## *4.1. Megerősítéses tanulás keretrendszerek, eszköztárak*

A megerősítéses tanulás világában a fő programozási nyelvnek a Python tekinthető, mivel a legtöbb implementáció, és a legnépszerűbb keretrendszerek is ezen a nyelven íródtak. Manapság számos területen vannak jelen az ilyen tanuláson alapuló megoldások: Robotika, Önvezető járművek, Repülőgép irányítása stb. Általában ezeket a keretrendszereket nagy kutatócsoportok fejlesztik ki, a két legismertebb például ezek közül az Elon Musk által alapított OpenAI és a Google AI csapata, a DeepMind. Az elkövetkezendő fejezetekben, ezeknek az eszközöknek, keretrendszereknek a vizsgálata kerül bemutatásra.

### *4.1.1. OpenAI Gym toolkit*

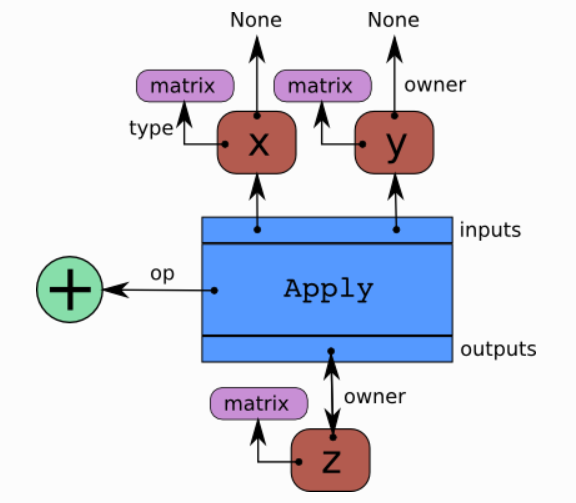
*A Gym [22] az OpenAI* nyílt forráskódú felülete, melynek könyvtára kiválóan alkalmas megerősítéses tanulási algoritmusok (ágensek) kidolgozására, összehasonlítására, mivel tesztproblémák – environments – sokaságát tartalmazza. Ebben rejtőzik legnagyobb előnye is, mivel arra a problémára nyújt egyszerű megoldást, hogy a meglévő RL környezetek nyílt forráskódú gyűjteményei nehezen beállíthatóak, használhatóak és még nem is elég változatosak. Jelenleg is használható környezetei témakör szerint: *Atari* [23], *Box2D* [24], *Classic Control* [25], *MuJoCo* [26], és a *Robotics* [27]. Fontos megemlíteni, hogy a feladatunkhoz szükséges football környezethez még csak hasonlót sem tartalmaz, viszont lehetőség van, saját környezet készítésére. Mivel ennek témája túlmutat jelen irodalmon ezért ezt a továbbiakban nem taglalom, viszont később bemutatásra kerülnek a jelenleg elérhető ilyen témájú környezetek. Programozási nyelvét tekintve Python nyelven íródott. Az eszköztár továbbá számos kompatibilitási előnyel jár, mivel támogatja bármely numerikus számítási könyvtár használatát, mint például a *Theanot*, vagy az iparág legelterjedtebb, a Google nyílforráskódú keretrendszerét, a *TensorFlow* keretrendszer családot.

### 4.1.2 TensorFlow keretrendszer

A Google 2015-ben adta ki *TensorFlow* [28] néven a nyíltforráskódú mélytanuló szoftverkönyvtárát, melyből mára az egész iparágat meghatározó eszköz lett, mivel használói egész széles spektrumon helyezkednek el: megtalálhatóak köztük az egyszerű szoftverfejlesztőktől kezdve egészen az MI kutatóiig bármilyen célcsoport is. További bizonyítéka lehet a népszerűségének, hogy több szoftvercsomaggal kiegészítették az évek folyamán. Megjelent a TensorFlow.js [33], amelyet a JavaScriptes gépi tanuláshoz hasznának, a TensorFlow Lite [34], amit mobilos neurális hálózatoknál alkalmaznak, de ezeken kívül még készültek más környezetekben használatos API-k is.   
A keretrendszer számos lehetőséget nyújt ahhoz, hogy könnyen gyorsan, és rövid kód implementálásával állítsunk össze és tanítsunk be neurális hálózatokat, TF-Agents nevezetű, előre letesztelt moduláris ágensei pedig megkönnyítik a mererősítéses tanuláson alapuló algoritmusok készítését. [32] Annak ellenére, hogy kezdetben C++ nyelven írodott a keretrendszer, ma már sok nyelvi implementáción keresztül is elérhető, mint például a Java és a C#, de a már a bevezetőben is említett ok miatt a legelterjedtebb ezek közül a Python használata.  
Működését tekintve, a gépi tanuló algoritmusok számításait és állapotait úgynevezett adatfolyam gráfokon értelmezi, ábrázolja a keretrendszer. Ez olyan gráf, melyben a csúcspontok számításokat, míg az élek az ezek között áramló adatokat reprezentálják. [29] Ennek fő előnye nem csak az intuitív (vizuális) kifejezése a számítási egységek közti függőségeknek, hanem az a tény is, hogy ezeknek a csomópontoknak a meghatározása teljesen általános lehet a gráfon belül. [29] Egyszerre több bemenete is lehet egy ilyen csomópontnak, melynek belső állapotai (mutuable state) folyamatosan változtathatóak, így akár nagy modellek használatakor, a megnövekedett paraméterszám is kezelhető marad.  
Említést érdemel, hogy habár a TensorFlow kompatibilis a már az előbbiekben kifejtett OpenAI Gym toolkit-tel, a mai gyakorlatban inkább egy másik magasabb szintű, Python implementációjú szoftvercsomagon keresztül használják azt, ami nem más, mint a *Keras*. [30] Annak ellenére, hogy nem kifejezetten a TensorFlow számára íródott és rajta kívül még számos más backend – et is támogat, ma már része a keretrendszernek. [31] Önmagában az API számos hasznos funkcióval is rendelkezik, mint például a hálókészítéshez szükséges aktivációs függvények elérése, nem mellesleg felhasználóbarát felülete megkönnyíti a vele való munkát.

8. ábra Olyan egyszerű adatfolyam gráf, amely a z = x + y műveletet valósítja meg, forrás: [29]

### 4.1.3. Theano keretrendszer

A *Theano* [35] egy olyan nyíltforráskódú Python könyvtár, amely lehetővé teszi a különféle matematikai kifejezések meghatározását, optimalizálását, kiértékelését, beleértve a többdimenziós tömbök hatékony kezelését is. Ennek eredményeképp bevezetése óta az ipar leggyakrabban használt CPU és GPU matematikai fórdítójának számít, különösen a gépi tanulás világban, habár említést érdemel, hogy régi keretrendszer lévén, a kutatók és tudósok már nem annyira népszerű, mint volt korábban. [36] A TensorFlow keretrendszerrel ellentétben a szimbolikus matematikai kifejezéseket nem adatfolyam gráfon, hanem aciklikus gráfokként valósítja meg, melyek két féle csomópontból épülnek fel: Változók csomópontja, amely az adatokat reprezentálja, és Alkalmazási csomópontok, amelyek a matematikai műveletek alkalmazását jelentik. Habár ez a gráf szerkezet nagyon hasonló az adatfolyam gráfokhoz, a legfőbb különbség, hogy ennél egyetlen köztes Változó csomópont több Alkalmazási csomópont bemenete is lehet egyszerre. [36] Ennek és a számítások C++ vagy CUDA nyelven való implementálásának negy szerepe van abban, hogy a keretrendszer gyorsan képes matematikai számításokat elvégezni. A keretrendszert úgy tervezték, hogy nagyfokú bővíthetőségge rendelkezzen, ezért ha nem tartalmazza az adott modellhez szükséges műveleteket, akkor lehetőség van Python, C++ vagy CUDA nyelven írt bővítményeket írni hozzá. Számos szoftvercsomag készült a Theano erősségeire építkezve, jobb grafikus felhasználói felületet kínálva, és sokkal specifikusabb problémák megcélzására. Hasonlóan a TensorFlow-hoz, ez a keretrendszer is része a Keras-nak, ezzel pedig megkönnyítve a mély tanulás architektúrájának kifejezését és a tanítóalgoritmusok létrehozatalát. [36]

9. ábra Theano matematikai kifejezéseinek gráf szerkezete

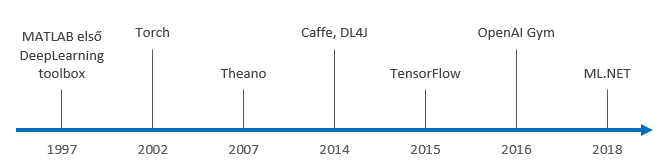
### 4.1.4. Torch

Az előbbiekhez hasonlóan a Torch [37] nyílt forráskódú - GPU támogatással rendelkező - keretrendszer is elérhető Python nyelven, PyTorch keretrendszer használatával, annak ellenére, hogy az alapját egy Cuda-C és Lua nyelvű implementáció adja. Felhasználási körét tekintve, elsődlegesen tudományos számítási keretrendszernek szánták, de indulása óta számos magánjellegű fejlesztésnek eszközbázisában megtalálható. (Microsoft, Facebook, Uber)

Felépítését tekintve a Torch moduláris megközelítést eszközöl, ezzel megkönnyítve a már létező algoritmusok módosításának lehetőségét, illetve az új gépi tanuló (megerősítéses tanuláson alapuló) metódusok implementálását. Ennek következtében a felépítése négy alapszerkezeten alapul [38]: a DataSet, amely osztály az adatok kezeléséért felel, a Machine osztály, amely olyan fekete-dobozt reprezentál, melynek feladata egy bemenetből és pár paraméterből, valamilyen belső működésen keresztül egy kimenet előállítása, a Trainer felel az optimális paraméterkiválasztásért és annak teszteléséért, az adott kritériumok és a DataSet alapján, míg végül a Measurer, a mérések során keletkezett adatok kiiratásáért felel (pl.: legnagyobb előfordulási valószínűség, átlagos négyzetes hiba, klasszifikációs hiba arány). [38] Működése röviden összefoglalva úgy írható le, hogy először a DataSet egy vagy több tanítómintát előállít, majd a Trainer ezt átadja a Machine számára. Ez az osztály kiszámít egy lehetséges kimenetet, amelyet aztán a Trainer arra használ, hogy a Machine paramétereit finom hangolja. Ezen folyamat közben egy vagy több Measurer – t használva lehetőségünk van a rendszer teljesítményének megfigyelésére. [38]

Említést érdemel, hogy míg Mac X és Ubuntu 12 + operációs rendszeren a meglévő támogatottságok miatt működése stabil, Windows rendszereken való futtatáshoz a hivatalos dokumentáció virtuális Ubuntu gép használatát javasolja. [39]

### 4.1.5. Egyéb machine-learning könyvtárak

Természetesen a fent részletezett AI keretrendszerek mellett számos másik is jelen van az iparágban és áll a felhasználók rendelkezésére. Ugyancsak mélytanuláshoz használható nyílt forráskódú könyvtár a Caffe [40], amely különösen konvolúciós neurális hálózatok fejlesztésére és tanítására alkalmas. Érdemes megemlíteni a horizontálisan skálázható Deeplearning4J [41] könyvtárat is, mely az egyetlen olyan keretrendszer, ami lehetővé teszi a modellek betanítását JAVA -ból, miközben együttműködik a Python ökoszisztémával. A .NET fejlesztőknek jelen van a Microsoft által fejlesztett ML.NET [43], mely nyílt forráskódú, több platformos és natív C# és F# támogatást élvez. Inkább ipari mérnököknek szánt keretrendszerek közé sorolható, a platformfüggetlen Matlab [42], amely ugyancsak rendelkezik gépi tanuláshoz alkalmazható eszközökkel. Persze a felsorolást lehetne tovább is folytatni, de mivel rengeteg ilyen megoldás létezik, így a teljesség igénye nélkül tovább lépünk.

10. ábra A taglalt keretrendszerek, eszköztárak megjelenésük szerint

## *4.2. A piacon jelenlévő Football Environment – ek*

Mint már a neurális hálózatok tulajdonságait taglaló részben megtudtuk, a megerősítéses tanuláson alapuló rendszerek az ágenseik által meghozott döntéseiket egy környezeten hajtják végre, amely aztán visszajelzést (megerősítések) ad az ágensnek jutalmak, illetve büntetések formájában. Tehát a környezet egy olyan feladat vagy szimuláció, amellyel az ágens kölcsönhatásba lépve, megpróbálja megoldani azt. A következő pontokban olyan környezetek vizsgálata következik, amelyek alkalmasak lehetnek labdarúgó botok betanítására.

### *4.2.1 Google Research Football API*

Egyik ilyen megoldás lehet a nyílt forráskódú, a Google kutatói által kifejlesztett Google Research Football Environment [45] használata, mely egy fizikán alapuló 3D-s futballszimuláció. A környezet számos kulcsfontosságú összetevőből áll össze: Magját egy fejlett futballszimuláció alkotja, a Football Engine, amely a Gameplay Football - Linuxra elérhető videójáték - erősen módosított változatán alapszik. Ennek alapja egy magasan optimalizált C++ kód, amely lehetővé teszi, hogy akár GPU nélküli gépeken is futtatható legyen a szimuláció. [45]. A játékmotor képes egy teljes mérkőzés leszimulálására, emellett pedig tartalmazza a futball olyan legáltalánosabb komponenseit, mint a szögletek, gólok, büntetőrúgások, leshelyzetek stb. Az ellenfél játékosainak irányításáért olyan szabály alapú botok felelősek, amelyeket még az eredeti Gameplay Football szimulátor biztosított. [45] Ezeknek a nehézségi szintjét egy θ, 0 és 1 közötti paraméterrel tudjuk módosítani, annak függvényében, hogy az ellenfeleket milyen reakcióidővel és döntéshozatali gyorsasággal szeretnénk felvértezni. A dokumentáció [45] által javasolt értékek a következők lehetnek: könnyű (θ = 0,05), közepes (θ = 0,6) és nehéz (θ = 0,95).

A képen szöveg látható

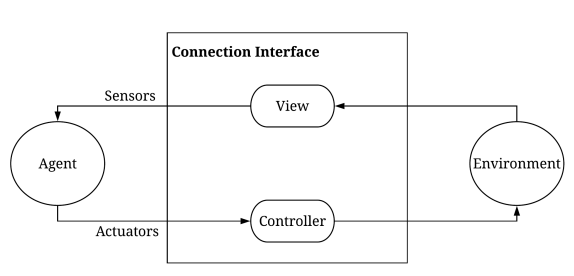
Automatikusan generált leírás Az ágensek számára több beépített, végrehajtható akciót is biztosít a környezet, melyek a 11. ábrában tekinthetünk meg. Ez tartalmazza a 8 irányba történő mozgási műveleteket, a különböző rúgási módokat (rövid, hosszú, magas passz, lövés), labdavezetést és szerelési módokat.

11. ábra Ágens műveleteinek listája, forrás [44]

Ami a végrehajtott akciók jutalmazását illeti, a Football Engine két félét valósít meg (egyedi jutalmazási funkciók hozzáadása is lehetséges), amely a SCORING és CHECKPOINT elnevezéseket kapták. [44] Röviden összefoglalva a működésüket, a SCORING egy rúgott gól esetében +1, egy bekapott gólnál -1 jutalmat ad. Mivel ennek állapotát a tanulás kezdeti fázisában nehéz megfigyelni, a CHECKPOINT ennek kiváltására szolgál. Ez annak tükrében jutalmaz, hogy egy adott művelet után a játékos közelebb, vagy távolabb került-e az ellenfél kapujától.

Annak érdekében, hogy minél egyszerűbben és gyorsabban lehessen az ágenseket betanítani, a környezet tartamaz egy Football Academy funkciót, amelyben különböző nehézségű előre meghatározott football szcenárió áll a fejlesztők részére. Ez alatt olyan szituációkat értünk, mint az ellentámadás, üres kapura való támadás, szögletek, szabadrúgások stb. Az érintett irodalom alapján [44], a meglévő szituációk mellé, egy egyszerű API-n keresztül saját forgatókönyvek is létrehozhatóak. Maga a környezet további olyan funkciókkal is rendelkezik, amelyek kifejezetten igazodnak a megerősítéses tanulás koncepciójához. Először is lehetőséget nyújt mind, szemantikai információkat tartalmazó (pl.: játékosok pozíciója), állapotábrázolásokból való tanulásra, mind nyers pixelek feldolgozásán alapuló tanulásra. [45] Továbbá a véletlenszerűség hatásának vizsgálata és a feladatok tetszőleges egyszerűsítése érdekében a környezet sztochasztikus (alapértelmezett, mesterséges intelligencia akcióiban van véletlenszerűség) vagy determinisztikus módban (a végrehajtandó művelet nem tartalmaz véletlenszerűséget) is futtatható. [45] Ami a kompatibilitási kérdést illeti, a környezet könnyen használható a széles körben használt Open AI Gym API-val.

### *4.2.2 FIFA*

Másik megoldás az EA Sports által fejlesztett FIFA [48] nevezetű játék használata, amely sztochasztikus környezetet tesz lehetővé, vagyis, ha ugyanazt a cselekvést, ugyanabból az állapotból hajtjuk végre, akkor az más eredményhez vezethet. [49] A játékban jelen vannak az ügyességi játékok, melyeket kiválóan lehetne használni, mint előre definiált játékszituációkat, taníttatás céljából. Ezekben az ügyességi játékokban megtalálható a szabadrúgások gyakorlására szánt forgatókönyvektől egészen a befejezések gyakorlására készített játékszituációkig minden. Ahhoz, hogy egyszerűen össze lehessen kapcsolni a tanuló modult (ágenseket) és a FIFA 3D-s játékkörnyezetet, interfészt kell használunk. Egyik ilyen használható API-t [50] Trivedi készítette a FIFA 18 kiadáshoz, azonban ez nagyon specifikus, csak szabadrúgást elvégző feladatokhoz használható. Egy sokkal generikusabb megoldást boncolgat az alábbi irodalom [49], amely Python nyelven lett implementálva és két fő modul különíthető el benne: egy Nézet és egy Vezérlő. A Nézet modul célja, hogy beszerezze a képernyőképeket a játékról és továbbítsa azt az ágens felé, ezzel pedig lehetővé téve a környezet aktuális állapotának észlelését és ennek megfelelően a döntés meghozatalát. Említést érdemel, hogy ez a megoldás a Google Research Football környezettel ellentétben nem képes a játék pillanatnyi adatai, információi alapján tanítani. A Vezérlőmodul fő feladata, hogy lehetőséget biztosítson az ágens számára a környezet módosítására. Ha ezt, vagy ehhez hasonló megoldást szeretnénk használni, akkor a fejlesztő feladata létrehozni egy olyan jutalmazó funkciót, amely érdemben nem befolyásolja a fent említett modulok működését. Ez a megoldás is kifejezetten az OpenAI keretrendszeren alapszik, amely széles körben használt kutatásoknál.

12. ábra A FIFA környezet és a tanuló modul közti kapcsolat, forrás: [49]

### *4.2.3 További elérhető környezetek*

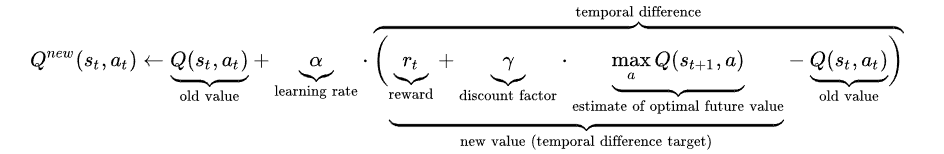
A fent megemlített környezeteken kívül, persze léteznek mások is. Ilyen a RoboCup Soccer Simulator [46] és a DeepMind MuJoCo Multi-Agent Soccer Environment [47], amelyek azonban, a magasabb szintű cselekvés helyett inkább a robotok fizikai szimulációjának, alacsony szintű vezérlésére koncentrálnak, ezért ezeknek részletezése nem tartalma jelen irodalomnak.

## *4.3. Az ágensek általános tanító módszerei*

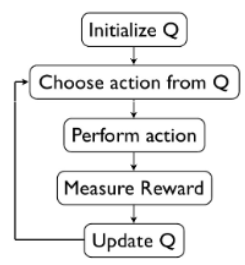
Az általam kiválasztott feladat megoldása következtében a mesterséges intelligenciának, a megerősítéses tanulás során játékszerű helyzettel kell szembenéznie, mely probléma megoldása során a *próba(*trial*)* és *hiba*(error) módszerét alkalmazza*.* Mint már korábban kifejtettem, ahhoz, hogy a gép azt tegye, amit maga a programozó, vagy a kívülálló felhasználó szeretne, az ügynök jutalmat, vagy büntetést kap az általa végrehajtott műveletekért. A következő alfejezetekben tehát olyan tanítási módszereknek, algoritmusok bemutatása következik, amelyek ezt a működést valósítják meg.

### *4.3.1 Q-Learning*

Az egyik legegyszerűbb és leggyakrabban alkalmazott megerősítéses tanulási algoritmus a Q-Learning, amely egy Q-függvény (quality) segítségével tanul. Ez az adott helyzetben végrehajtott cselekvésnek a várható hasznosságát becsüli meg, tehát az állapotokon kívül a cselekvésekhez is hasznosságot rendel. [51] Ezt egy *Q(s,a)* függvénnyel lehet leírni, amely megadja az *s*-ben az *a* végrehajtásának a jóságát, eredményét pedig mátrixokban, táblázatokban (Q-táblában) szokás eltárolni. [52] Ezt csak abban az esetben célszerű használni, ha a környezet egyszerű, ennél fogva a paraméterszám nem nagy.

A Q-Learning stratégiáját tekintve az összes lépésre adott jutalmak maximalizálására törekszik, ehhez pedig lépéssorozatokkal próbálkozik. Ez kétféle módon történhet: véletlenszerűen(exploring) vagy egy megtanult stratégia alapján (exploiting). [53] A Q-Learning algoritmusának fő alkotóeleme az *Update* szabály, amely a Bellman-egyenletet használva egyszerű érték iterációval frissíti a Q-táblázat megfelelő értékét, a régi érték és az új információ súlyozott átlagának felhasználásával [54]:

ahol, rt az st állapotból az st+1 állapotba való váltáskor kapott jutalmat, az α pedig a tanulás arányát jelöli (learning rate). Ez egy olyan 0 és 1 közti érték, amely meghatározza, hogy az újonnan beérkező információ milyen mértékben írja felül a régi információkat. 0 közeli értéknél az aktuális jutalom fontosabb, míg 1 körüli értéknél a fókusz a hosszú távú hasznon van. [53] Az állapot-cselekvés minősége három tényező összegéből alakul ki: az előbb taglalt súlyozott jutalomból (rt \* α), az aktuális minőség súlyozott értékéből [Q(st,at) \* (1-α)] és a maximálisan elérhető jutalom súlyozott értékéből.

A tanulási epizód során a cél, hogy egy véletlenszerű állapotból a célállapotba jussunk, viszont ehhez sok tanulási epizódra van szükség, ennek következtében pedig az algoritmus felépítése a következőképpen írható le: a Q tábla inicializálását követően, kiválasztásra kerül egy cselekvés ebből a táblából, amelyet végrehajtva jutalmazunk jósága szerint. Utolsó lépésben a kapott értékkel - a Bellman egyenletet használva - frissítjük a Q táblát, majd újra a választás fázisa következik. Ez az iteráció addig ismétlődik még az epizód végére nem ért az algoritmus [54].

13. ábra A Q-Learning algoritmus működési modellje

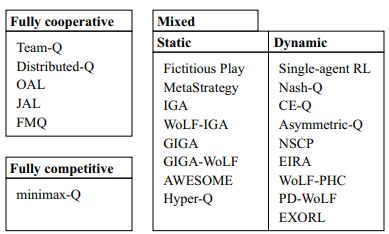
Abban az esetben, ha a lehetséges műveletek és állapotok száma megnő, a Q tábla méret növekedését kiküszöbölve, célszerű lehet, a táblázatot egy olyan neurális hálózatra cserélni, melynek bemenete az aktuális állapotot, kimenetei pedig a lehetséges akciókat jelentik. [52] Ezeket a hálózatokat nevezik Mély Q-hálózatoknak ( Deep Q-Networks – DQN ), felépítésüket tekintve pedig a gyakorlatban két neurális hálózatot tartalmaznak annak érdekében, hogy a tanulás gördülékenyen menjen. [52] Ezek közül az első a fő hálózat, amelyet a θ súlyvektor reprezentál és az s állapothoz tartozó *a* művelet, Q-értékének becslésére szolgál [Q(s,a, θ)]. A második a cél hálózat melynek felépítése teljesen megegyezik a fő hálózatéval, viszont az s’ és a’ állapotokat határozza meg. Tanulási folyamatát tekintve, csak a fő hálózat tanul, mégpedig azt, hogy melyik állapotot érdemes választani, majd ennek becsült értékei máslódnak át a célhálózatba, ezzel stabilabb tanulást lehetővé téve [52, 53] Ezt Dupla Q-Tanulásnak nevezik (Double Q-Learning).

### *4.3.2 SARSA*

Egy másik, a Q-Learninghez hasonló megerősítéses tanulás algoritmus a SARSA, melynek neve a **S**tate-**A**ction-**R**eward-**S**tate-**A**ction szavak kezdőbetűinek összeolvasásából ered. A mozaikszó lényegében a Q-érték frissítésének funkcióját írja le, amely függ az ágens jelenlegi állapotától „**St**”, az általa választott művelettől „**At**”, a művelet választásáért kapott jutalomtól „**Rt**”, az „**St+1**” állapottól, amelybe az ágens a műveletvégzés után lép be, és végül az új állapotban választott művelettől „**At+1**”. Mivel az algoritmus a Q-Learning módszer továbbfejlesztett verziója, ezért iterációját tekintve a lépései megegyeznek, azonban a legfőbb eltérés a két módszer között, hogy a SARSA a mohó politika helyett, a Q-értékek frissítését a ténylegesen végrehajtott művelet alapján végzi el. [55] Ennek megfelelően az Update szabálya a következőképpen írható le:

, ahol az állapotműveletek Q értéke egy olyan hiba értékkel frissülnek, amelyek egy, a már korábban megismert tanulási aránnyal (learning rate) vannak korrigálva. [54]

### *4.3.3 Multiagent learning*

A több ágensen alapuló tanulás (MARL – Multi Agent Reinforcement Learning) a megerősítéses tanulás egyik olyan ágazata, amely a megosztott környezetben szimultán létező ágensek viselkedésének tanítására összpontosít. [56] Mivel minden ügynököt a saját jutalma motivál és a saját érdeke előre mozdítása érdekében cselekszik, ennek következtében viszont ezek az érdekek nem mindig egyeznek, érdekütközések alakulhatnak ki a cél elérése érdekében. A koherens együttműködés szempontjából tehát mindegyik ágensnek figyelembe kell vennie a másikat és viselkedésüket össze kell koordinálni. A probléma megoldására használható algoritmusokat az irodalom [56] típusaik szerint négy fő csoportba különíti el: teljes mértékben együttműködő, teljes mértékben versenyképes, vegyes feladatokat megoldó, illetve statikus vegyes feladatokat megoldó algoritmusok. Az elkövetkezendőkben ezeknek a típusoknak a bemutatása lesz olvasható, jelen irodalom nem tér ki a konkrét algoritmusok jellemzésére.

14. ábra MARL algoritmusok, típusaik szerint csoportosítva, forrás: [56]

A teljesen együttműködő ágensek esetében a tanulás fő célja, a közösen megoldott művelet hasznosság értékének a maximalizálása. Egy darab központi vezérlő esetében a feladat a Markov féle döntési folyamatra redukálódik le, melynek cselekvési tere a közös cselekvési tér, a cél pedig ennek a térnek az értékeinek az optimális tanulásával érhető el. Erre jól használható a már korábban megismert Q-Learning algoritmus. Abban az esetben, ha az ágensek mind független döntéshozók, akkor felmerülhetnek koordinációs problémák, még akkor is, ha közös Q-függvényt használnak a közös tanulás párhuzamosítására. Erre jelenthetne megoldást a mohó szabály alkalmazása, azonban ezzel az a probléma, hogy a kialakult holtpontot véletlenszerűen szüntetné meg, és az ebből eredő közös művelet nem lenne optimális. Megoldására több megközelítés is említ az irodalom [56]: a koordináció-nélküli módszerek esetében az algoritmus úgy oldja meg a problémát, hogy feltételezi, hogy a közös műveletek egyediek (ezek ritkán fordulnak elő). Ez csak determinisztikus, azaz véletlenszerűséget nem tartalmazó környezetben érvényes felvetés. Egy általánosabb megközelítés lehet a probléma megoldására, ha garantáljuk, hogy a holtpontot mindegyik ágens ugyanúgy törje meg, ehhez viszont valamilyen szinten szükség van a cselekvési döntések összehangolására, szociális konvenciókkal [57] vagy koordinációs gráfokkal [58]. Ezt nevezik közvetlen koordinációs módszernek. Ennek ellentettjei, a közvetett koordinációs módszerek, ahol a cselekvések kiválasztását a jobb értékeket ígérő, és ezáltal jobb értékeket hozó cselekvések irányába tolják el az ágensek.

Teljesen versenyképes esetben, két ügynök esetén kiválóan alkalmazható a minimax elv, ami az egyik ágens hasznát úgy próbálja maximalizálni, hogy feltételezi, hogy az ellenfél annak minimalizálására törekszik. [56] Ez felfogható a minimális nyereség maximalizálása mellett, a maximális A képen szöveg látható

Automatikusan generált leírásveszteség minimalizálásaként is. Ezt a működést az alábbi algoritmus írja le:

, ahol **m1** a minimax kimenete az 1. ágensnek. Ezzel a megoldással még akkor is egy ellenféltől független működés érhető el, ha a minmax optimalizáció során több megoldás is lehetséges, mivel ezek közül mindegyik eléri a minimum megtérülést.

A képen szöveg látható

Automatikusan generált leírás A vegyes feladatokat megoldó algoritmusokat [56] két csoportra lehet osztani. Statikus, ismétlődéseken alapuló játékok esetében a tanulás egyik legfontosabb eleme az újonnan bevezetett negatív jutalom, a veszteség (loss). Mivel a tanulás problémája továbbra sem állandó az ágens dinamikus viselkedése miatt, a kategóriában jelenlévő megoldások továbbra is figyelembe veszik a többi ágenshez való adaptációt. [56] Ilyen környezet esetén, az ágensek betanítására kiválóan használhatóak, az ellenfelek viselkedési modelljéhez alkalmazkodó ágenskövetési módszerek (agent-tracking method), vagy az ágens-tudatos módszerek (agent-aware method), melynek fő erénye, hogy az algoritmus alkalmazkodik a többi ágenshez, ha azok mozdulatlanok, egyéb esetben pedig a játékelméletben ismert Nash-egyensúlyra törekszik. Nash-egyensúlynak nevezik azt az egyensúlyi állapotot, amikor egyik játékosnak sem éri meg változtatni a stratégiáján. A másik esetben, amikor dinamikus (sztochasztikus) környezetről van szó, a fő kihívást az negatív jutalmak, a nem állandó állapotok és az ütköző célok jelentik. Habár ezeknél a környezeteknél az egy-ágensű megerősítéses tanulás alkalmazható több ágens esetében is, a MARL nonstacionaritása miatt a Single-Agent RL során megtanult elmélet legtöbbjét érvényteleníti, illetve inkonzisztens működést is eredményezhet, ha az ágensek súlyosan zavarják egymást. [56] Mindezek ellenére, mivel ez a megoldás elég egyszerű, talált alkalmazásokat: [59], [60] Ennél sokkal elterjedtebbek az ágens-független módszerek, amelyek a Q-Learning-en alapulnak, értékeik állapotát és szabályaikat pedig játékelméleti megoldókkal számolják. [56] Ez az alábbi egyenlettel írható le:

, ahol {Q·,k(x, ·)} az x játékállapotban, az összes ügynök által a k időpillanatban keletkező Q-függvényeket jelöli, a *solvei* pedig az i. ágens megoldó stratégiáját adja meg. [56]

## *4.4. Rendelkezésre álló, hasonló megoldások áttekintése*

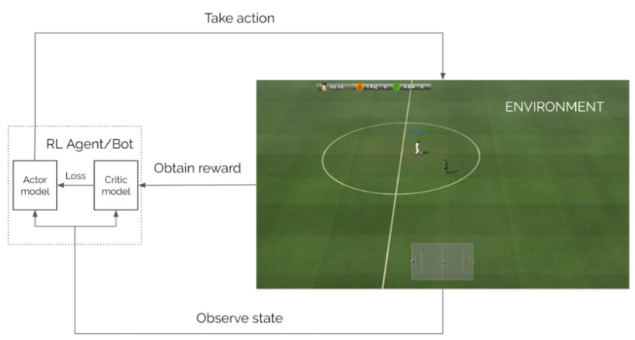
A fejezet célja, olyan már meglévő, elkészített projektek felkutatása és értékelése, melyek hasonlóan jelen irodalom támájához, football AI betanítással foglalkozik.

### *4.4.1* [*Chintan Trivedi*](https://medium.com/@chintan.t93?source=user_profile-------------------------------------) *féle megoldás*

Egyik ilyen vizsgálható megoldás a [Chintan Trivedi](https://medium.com/@chintan.t93?source=user_profile-------------------------------------) által készített [61], Keras keretrendszeren alapuló, Python implementációja. A projekt leginkább a proximális szabályzatoptimalizáláson alapuló megerősítéses tanulás algoritmus, ágensek betanítására alkalmazott módszerét hivatott bemutatni. Működését tekintve [62] ez a fajta tanulás nagy számú kötegeket (megerősítéses tanulás esetén egy kötegben az állapotot, műveletet, jutalmat és a t + 1 állapotot szokás eltárolni) igyekszik összegyűjteni, amelyből aztán egy megbízhatósági metrikát számolva meghatározza, hogy az adott szabályzat módosítása jelenthet-e teljesítménynövekedést a tanulás során. Ez a módszer biztosítja, hogy a szabályzat módosításakor ne változtasson túlzó mértékben a régi szabályon, ezzel gördülékenyebb tanulást elérve. Ennek eredményeképpen pedig megakadályozható, hogy az ágens értelmetlen és eredménytelen cselekedetet hajtson végre. A módszert az OpenAI [63] kutatói alkották meg, kijavítva az addig létező politikai gradiens módszerek hibáit: még az egyszerű feladatok megtanulásához is rengeteg időlépés kellett a gyenge mintavételi hatékonyság miatt, illetve a tanulás lépésméretének meghatározása is nehézkes lehetett.

Ez az implementáció a már korábban bemutatott Google Research Football API – t használja szimulációs környezetként, viszont kompatibilitását tekintve bármilyen olyan más környezet is használható jelen megoldáshoz, amely rendelkezik OpenAI Gym API Python támogatással.

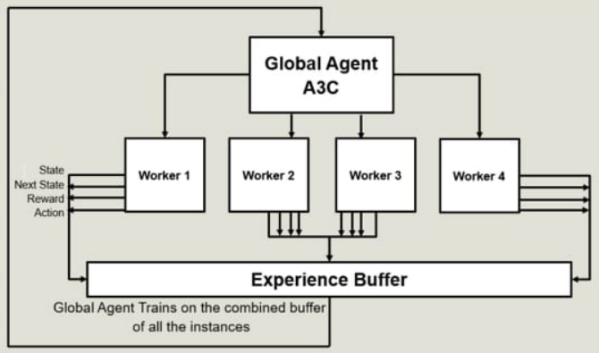
Jelen feladat egyetlen egy ágens betanítására fókuszál, amelyet a környezet „üres kapu” elnevezésű szcenáriójában végez el. Ennek a forgatókönyvnek a lényege, hogy a játékosnak a félpályáról indulva, egy ellenféllel szemben üres kapura kell gólt lőnie. A környezet által biztosított két tanítási mód közül (állapotábrázolásokból, pixelekből) az ágens állapotának megfigyelése jelen esetben képkockánkénti RGB képernyőképekből történik.

Az ágens modellje két különálló mély neurális hálózatra bomlik, melyeket „Critic” és „Actor” néven illet a dokumentáció [64]. Az Actor modell felelős azért, hogy a környezet adott megfigyelt állapotában, feldolgozva a bemenetként érkező képet, kimenetként előállítsa a végrehajtandó műveletet (passz, csel, lövés stb.). Ezt a műveletet végrehajtva, a környezet jutalom formájában negatív, illetve pozitív visszajelzést adhat, annak függvényében, hogy jósága szerint milyen volt az elvégzett cselekvés. Ezt a jutalmat kapja meg a Critic modell, feladata pedig megállapítani, hogy az elvégzett művelet jobb állapotba hozta a környezetet vagy sem. Kimenetként a művelet Q-értékét adja vissza, melyet az Actor összevet a régi értékkel, így eldöntve, hogy hogyan tud jobb lépéseket tenni.

15. ábra A projekt felépítése, forrás: [64]

Ahhoz, hogy az adott időpillanatokban összegyűjtött jutalmakból ezt el lehessen dönteni, minden egyes művelethez ki kell számolni, hogy annak végrehajtásával mekkora előnyt (advantage) lehet szerezni, nemcsak rövid, de akár hosszútávon is. Erre Trivedi a GAE, azaz a Generalized Advantage Estimation algoritmust használta.

### *4.4.2.* Vishal *Bidawatka megoldása*

Bidawatka megoldása [65] elméleti összetételében nagyon hasonlít Trivedi implementációjára, azonban egy lényeges különbség, hogy a sima Actor-Critic algoritmus helyett, annak aszinkron változatát, az Aszinkron Actor-Critic (A3C) algoritmust használja az ágensek tanítására. Az A3C előnye, hogy több dolgozó (worker) van jelen, amelyek mindegyike külön környezetben, a többitől teljesen függetlenül, párhuzamosan hajtja végre az utasításokat. Az epizódok végén mindegyik worker egy tuple adatszerkezettel tér vissza, amelyben eltároljuk a jelenlegi és a következő állapotot, a jutalmat, illetve egy bool változót, amely megadja, hogy az epizód a végére ért vagy sem. Ezeket összegyűjtve egy buffer összegzi a workerek eredményeit, amelyből aztán a globális ágens elvégzi a tanítást. Köszönhetően annak, hogy a dolgozók külön szálon futnak, párhuzamosan, rendkívüli sebességnövekedés érhető el a tanítás során.

16. ábra A3C felépítése, forrás: [65]

# 5. Feladat specifikáció

A megismert technikák, technológiák alapján egy olyan rendszer tervezése és implementálása a cél, amely egy szimulációs környezetet megfigyelve, megerősítéses tanulást használva helyesen képes végrehajtani a kijelölt feladatát.

## *5.1. A szimulációs környezettel szemben állított követelmények*

Szimulációs környezetként a feladatban a Google Research Football API használata legyen jelen, köszönhetően annak, hogy egyszerű használata mellett, több rendelkezésre álló funkcióval rendelkezik, a többi vizsgált környezethez képest. A feladatot a környezet „ellentámadás” szcenáriójára kell végrehajtani. Ezenkívül a tanításnak a környezet állapotábrázolásából kell, hogy történjen, mivel ezzel gyorsabb tanítás érhető el, mintha ugyanezt képfeldolgozással csinálnánk.

# 6. Megvalósítás

Kezdésnek egy olyan megvalósítás összeállítása volt a cél, amely egy egyszerű játékhelyzetre képes megtanítani az agent-et. Ennek oka, illetve alapgondolata, hogy ezáltal sokkal gyorsabban képet lehet kapni a fejlesztés közbeni állapotokról, részmegoldások működéséről, így az esetleges javításokra, módosításokra hamarabb sor kerülhet. Ennek értelmében a fejlesztés ütemét két részre osztottam: egy egyszerűbb, illetve az eredetileg is célként megjelölt „nehéz-ellentámadás” szituáció megtanítására.

## *6.1. Egyszerű játékhelyzetre való tanítás*

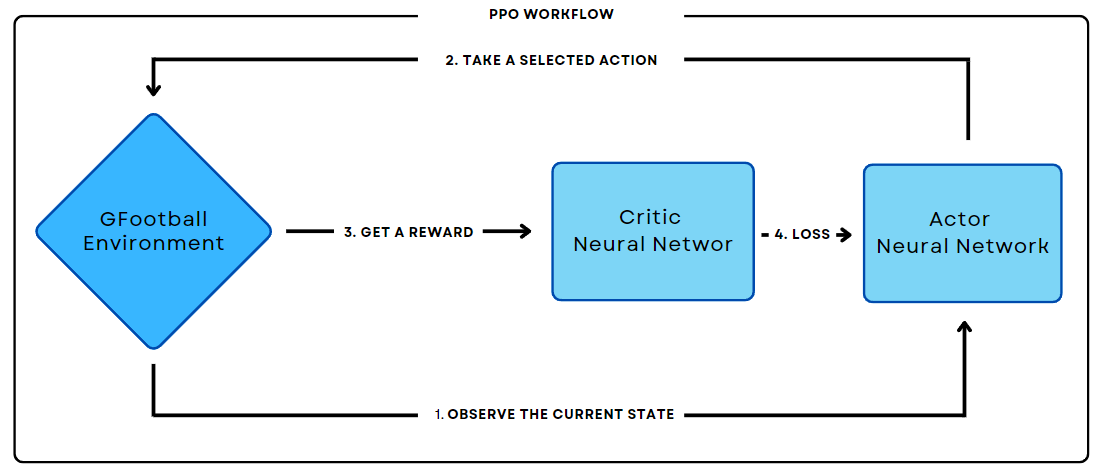
### *6.1.1.* Játékhelyzet ismertetése

Jelen szituációban egy támadó játékos vesz, akinek a feladata, hogy a félpályáról, valamilyen módon a jobb oldali kapuba juttassa a labdát. Ezt a környezet által előre definiált interakciókon keresztül érheti el (lövés, futás, passzolás), azonban a mi feladatunk megtanítani ezt neki. Fontos megemlíteni, hogy a játékossal szemben kapus, illetve védekező játékos nem helyezkedik el, ebből adódóan ez a játékhelyzet az egyszerűbbek közé sorolható.

17. ábra A környezet futás közben

### *6.1.2.* A tanítás elméleti megvalósítása

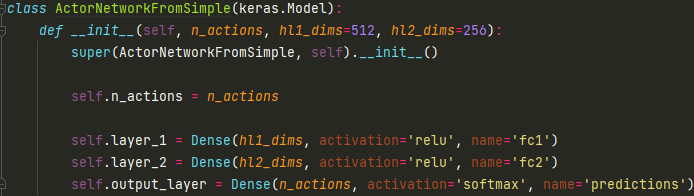
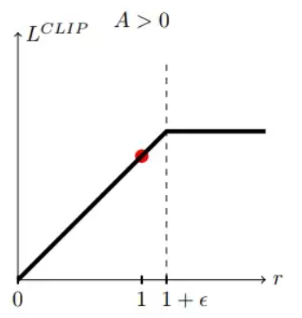
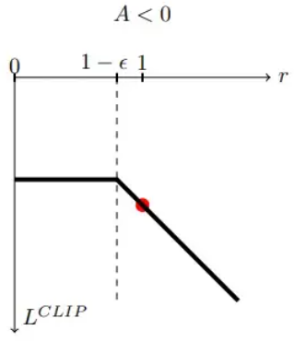
A tanításhoz, az RL algoritmusok közül az egyig legelterjedtebbet alkalmaztam, melynek neve PPO (Proximal Policy Optimization). Nagy előnye, azon felül, hogy kevesebb időt vesz igénybe a modellek tanítása a használatával, hogy más algoritmusok alkalmazásával szemben jobb, és stabilabb eredményt kaphatunk használatával. Leegyszerűsített működését az alábbi ábra szemlélteti:



. ábra A Proximal Policy Optimization algoritmus folyamatábrája

, amelyben az Actor modell feladata, hogy eldöntse, a jelenlegi szabályrendszerhez igazodva milyen cselekvést végezzünk el a környezetben. A végrehajtott akció értékelése a Critic modell feladata lesz. Ekkor megfigyelésre és elemzésre kerül az aktuális állapot, melynek függvényében visszajelzést tudunk adni az elvégzett cselekvés jóságáról. A kiértékelés legfontosabb eleme egy *loss* érték kiszámítása lesz (amellyel elvégezhető a modellek szabályrendszerének frissítése). Az érték kiszámítása során kiemelt célunk lesz, hogy minél jobb-és-jobb eredményt tudjunk elérni a tanítás során. Ennek következtében, a megoldás implementálása közben, számításba kell vennünk, hogy ne alakuljon ki visszafordíthatatlan út a tanulás folyamatában. Ez alatt értendő, hogy ha a modell egy rossz szabályrendszert kezd el követni, az rossz cselekvések végrehajtását fogja eredményezni, ahonnan már szinte lehetetlen a jó irányba visszafordítani a tanulás menetét.  
A PPO algoritmus egy iteráción alapuló megoldás, melyben a környezettel adott számú lépésben interaktálunk. Ezt a lépésszámot adja meg a *PPO steps értéke*. Egy iteráció alatt számos tapasztalati mintát gyűjtünk össze az aktuális szabályrendszer szerint, amelyek alapján aztán frissítésre kerülnek a modellek. Ezt a folyamatot egészen addig kell ismételnünk, amíg nem lesz egy olyan modellünk, amely képes az előre definiált feladat végrehajtására.

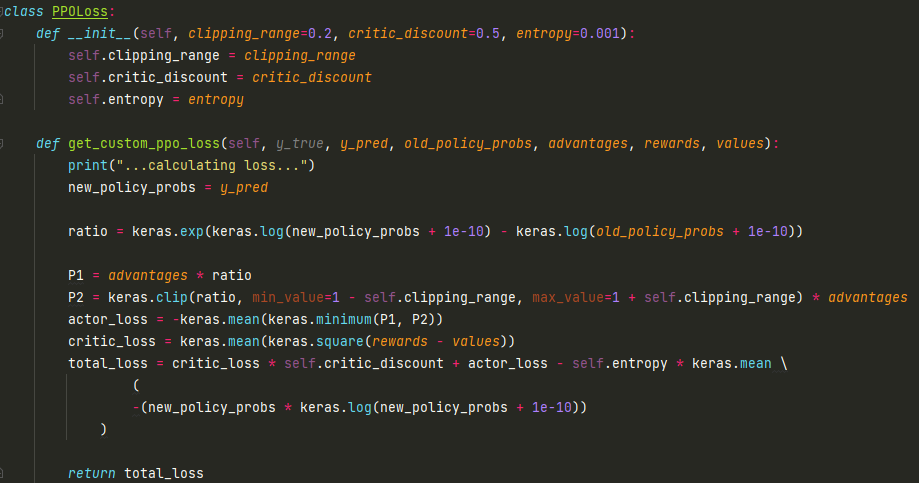
### *6.1.3.* Implementáció

Az implementáció megkezdéseként a legelső dolog, amit csináltam, (a PPO algoritmusnak megfelelően), létrehoztam a két neurális hálózatot melyek a következők lesznek: **ActorNetworkFromSimple** és **CriticNetworkFromSimple**. Fontos említést tenni róla, hogy a két hálózat bemenetét, kimenetét, aktivációs függvényét, illetve a loss funkcióját tekintve különbözőek lesznek. Az **Actor** hálózat tanításához minden olyan adatot meg kell adnunk a környezet állapotáról, amelyet a tapasztalatszerzes során összegyűjtöttünk. Ezek lesznek a hálózat input-jai. A súlyok változtatásával a hálózat egy értéket fog visszaadni, ez pedig az elvégzendő interakció lesz. A gyakorlati implementáció a következő képeken látható:   
Mint már korábban is említésre került, egy neurális hálózat tanításának elengedhetetlen része a loss funkció meghatározása. Jelen esetben, ahhoz, hogy megelőzzük a visszafordíthatatlan út kialakulását a tanulásban, egy egyedi metódus implementálása volt szükségszerű. Ebben bevezetésre került egy úgynevezett *ratio érték*, amellyel könnyen megállapítható, hogy mennyit változott a szabályrendszer a régi és az új állapot között. A megengedett legnagyobb eltérés a *clipping\_range* változóval szabályozható, melynél érdemes elkerülni a nagyobb értékeket, ezzel elkerülve a drasztikus változásokat a szabályrendszerben. Gyakorlati megvalósítása az alábbi ábrán látható:

19. ábra Actor neurális hálózat megvalósítása

20. ábra Egy jó action esetében, a loss egy bizonyos érték után kiegyenesedik, így limitálva a gradiens frissítésének hatását

21. ábra Rossz action esetében, egy bizonyos érték alatt a loss funkció görbéje kiegyenesedik, elkerülve a drasztikus romlást a szabályrendszerben

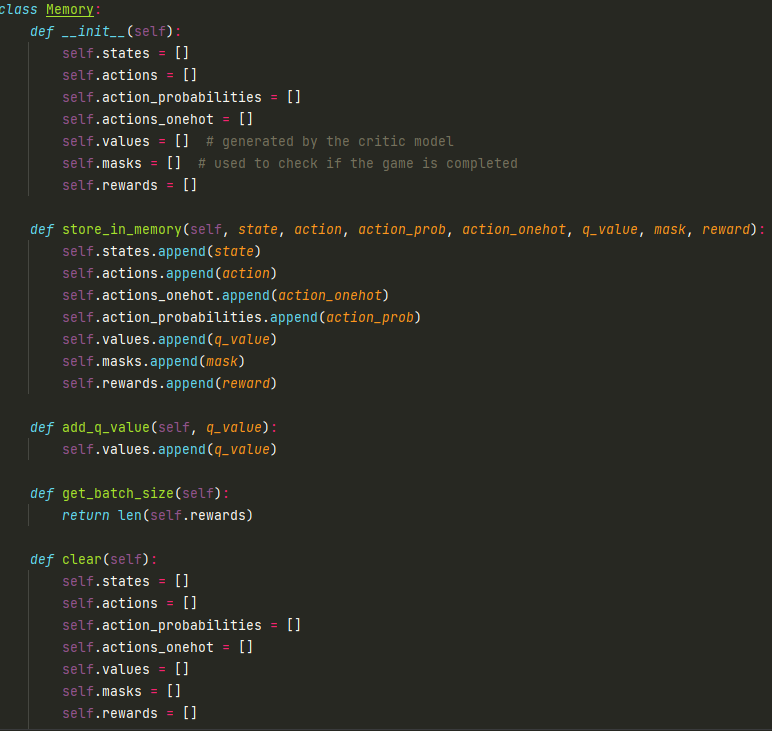
, melyben míg az actor modell loss értéke a P1, P2 function minimuma, addig a critic modellét az átlagos négyzetes hibával (angolul: Mean Squared Error) tudjuk kiszámítani. A teljes érték kiszámításához, opcionálisan bevezethető egy *entropy* változó is, amely bátorítja a modellt arra, hogy több szabályrendszert is kipróbáljon a tanulás során. Lényegében ezzel az értékkel szabályozható a kísérletezés mértéke.  
Eltérően az Actor modelltől, a Critic megvalósítása már sokkal egyszerűbb feladat: bemenő paraméterként a környezet állapotát leíró értékeket várja, amelyek alapján aztán vissza tud adni egy q értéket, ezzel meghatározva, hogy az adott állapotban mennyire volt jó az a cselekvés, amelyet elvégeztünk. Ennek a gyakorlati megvalósítása a következő képen látható:

22. ábra PPO loss gyakorlati megvalósítása



. ábra Critic modell gyakorlati megvalósítása

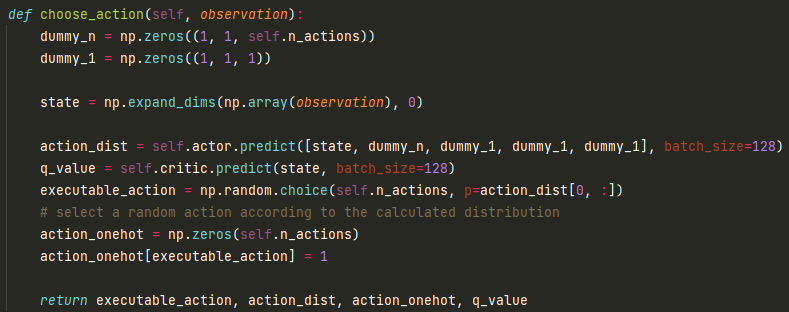
Két alapvető alkotóelem – a két neurális hálózat - tehát már rendelkezésre áll a tanuló ágens létrehozásához, azonban a PPO megvalósítás miatt, létre kell hozni egy tárolót, hogy az iterációs ciklus során folyamatosan érkező és kiszámolt állapotokat, adatokat egy helyre tudjuk gyűjteni. Ezeket aztán felhasználhatjuk az advantage érték számításához, amely megmondja, hogy az adott cselekvési sorozattal mennyivel kerültünk közelebb a végső célunk eléréséhez. A memória megvalósítása a következő:



. ábra Állapot memória megvalósítása

Ezzel lényegében az ágens három komponense, amelyet magában foglal el is készült. Ennek megfelelően el is készítettem az *Agent* osztályt az alábbi tulajdonsággokkal (a későbbiekre nézve javítási lehetőség a jelenlegi megoldáson, hogy alkalmazzuk a Dependency Injection elvet, és ne példányosítsunk a konstruktorban):

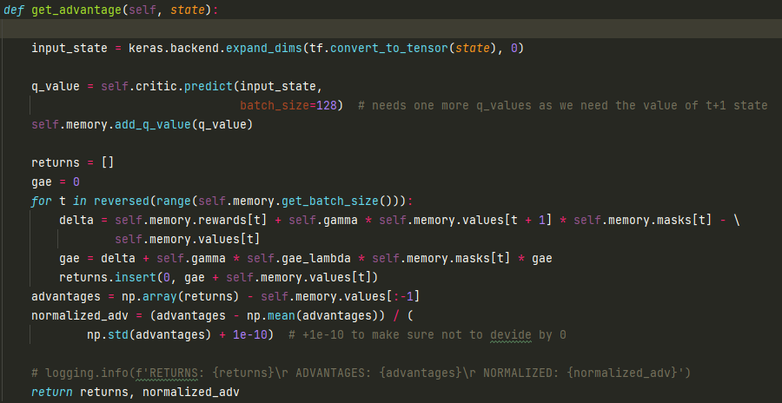
25. ábra Agent osztály konstruktora

Ezt követően meg kell határozni, hogy milyen feladatokat lásson el az osztály, milyen metódusai legyenek. Ahhoz, hogy a tapasztalatszerzés folyamatát meg lehessen valósítani, szükség van egy metódusra, amely az Actor hálózat aktuális szabályrendszere alapján visszaad egy cselekvést, amit aztán a környezetben elvégzünk. Ennek a megvalósítása a következő: 

. ábra Az Agent action kiválasztó folyamata

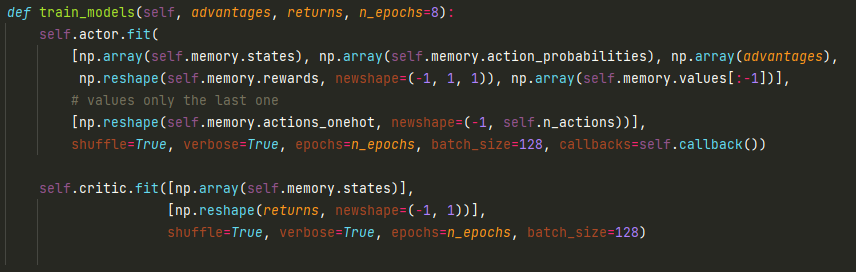
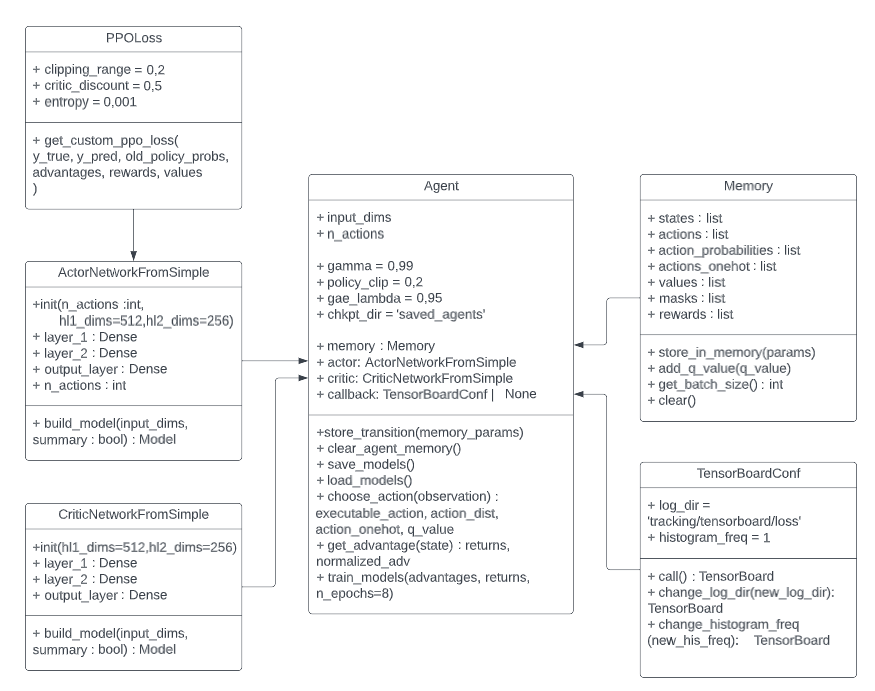
, melyben először az állapot értékeit tartalmazó tömböt a megfelelő alakúra kell alakítani, ezt követően pedig az aktuális szabályrendszer alapján kiszámoljuk a lehetséges cselekvések eloszlását. Itt megjegyzendő, hogy mivel itt nem a hálózat tanítása a cél, így bemenetként csak az aktuális állapotot kell nekünk átadnunk, a többi bemeneti értéket dummy értékkel helyettesítjük. Miután ez megvan, az eloszlási valószínűségek alapján véletlenszerűen kiválasztjuk a cselekvést. A későbbiekben szükség lesz egy q értékre is, így azt is kiszámoltatjuk a hálózattal, majd visszaadjuk.  
A tanítás során szükségünk lesz egy relatív érték kiszámítására, mely megmondja, hogy mekkora előnyhöz juthatunk az adott állapotban egy cselekvés végrehajtásával. Ehhez fel szeretnénk használni a PPO lépéseiben összegyűjtött jutalmakat, és segítségükkel kiszámolni, hogy így mekkora előnyhöz lehet jutni az általunk megtett lépésekkel. Ha például tehát kapura lövünk, vagy éppen a cselezést választjuk, megkaphatjuk, hogy ezekkel a lépésekkel mennyivel kerültünk közelebb a végső cél eléréséhez. Ezt nem csak rövidtávra nézzük meg, hanem hosszabb időre is vizsgáljuk. Ennek következtében, még ha az adott pillanatban nem is szereztünk gólt, akkor is megnézünk pár lehetséges lépést az adott akció végrehajtása után, így vizsgálva, hogy azt követően lőttünk-e gólt. Ez lesz az *advantage* számítás, amelyet az általános előny számításának módszerével (GAE – Generalized Advantage Estimation) számoltam ki, melynek lépései a következők:

1. egy *gae* értéket inicializálunk:
2. visszafelé végig iterálunk a szerzett tapasztalatokon (esetünkben t=127-től t=0-ig)
3. meghatározzuk a delta értéket:  
   , ahol **rt** az adott időpillanathoz tartozó jutalmat jelenti. **γ** az nem más, mint egy konstans, melynek neve: *discount factor.* Enneka segítségével szabadon állítható a képletben, hogy mekkora hangsúly legyen a jelen, illetve a jövő állapoton. Az ajánlott értéke az 0,99, melynek értelmében a jelenben szerzett gólon nagyobb hangsúly van, mintha ugyanezt a jövőben érnénk el. Míg **V(St)** a t, addig a **V(St+1)** a t+1 állapothoz tartozó értékeket adja meg. A képlet utolsó eleme **mt** maszk lesz. Fő funkciója, hogy ha az St+1 érték már egy újrakezdett játékból jönne, akkor azokat hagyja figyelmen kívül, ekkor a maszk értéke 0.
4. a már definiált gae értéket frissítjük a következő képlet alapján:   
   , ahol felhasználjuk az előző lépésben kiszámolt **δ**-t, illetve bevezetésre kerül egy lágyító paraméter is: **λ** , amely a tanulás során tapasztalható eltérések csökkentésére szolgál, így egy sokkal stabilabb tanulást elérve. A paraméter értéke: 0,95
5. meghatározzuk az R visszatérési értéket, majd visszafordítjuk az R értékeket tartalmazó listát

Az fent megismert algoritmus gyakorlati megvalósítása a következő ábrán látható:

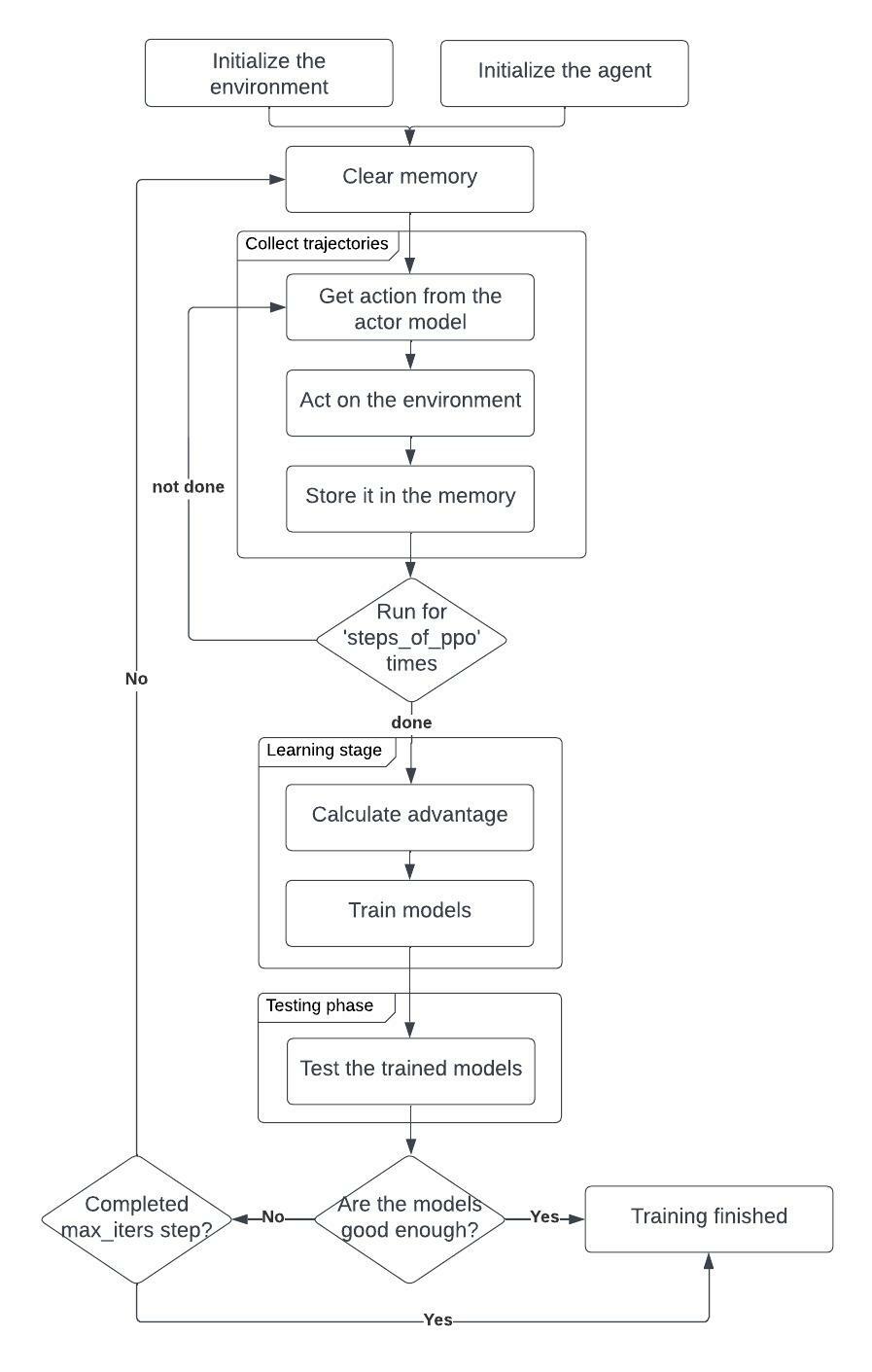
27. ábra GAE gyakorlati megvalósítása

Az ágens utolsó fontos dolga, hogy képes legyen a magában tartalmazó neurális hálózatok tanítására, aktuális szabályrendszerük frissítésére. Ennek megvalósítására a keras beépített *fit* metódusáthasználtam. Ezt meghívva a hálózat megfelelő bemeneti paramétereivel a tanítás folyamata nagyon egyszerűen implementálható:

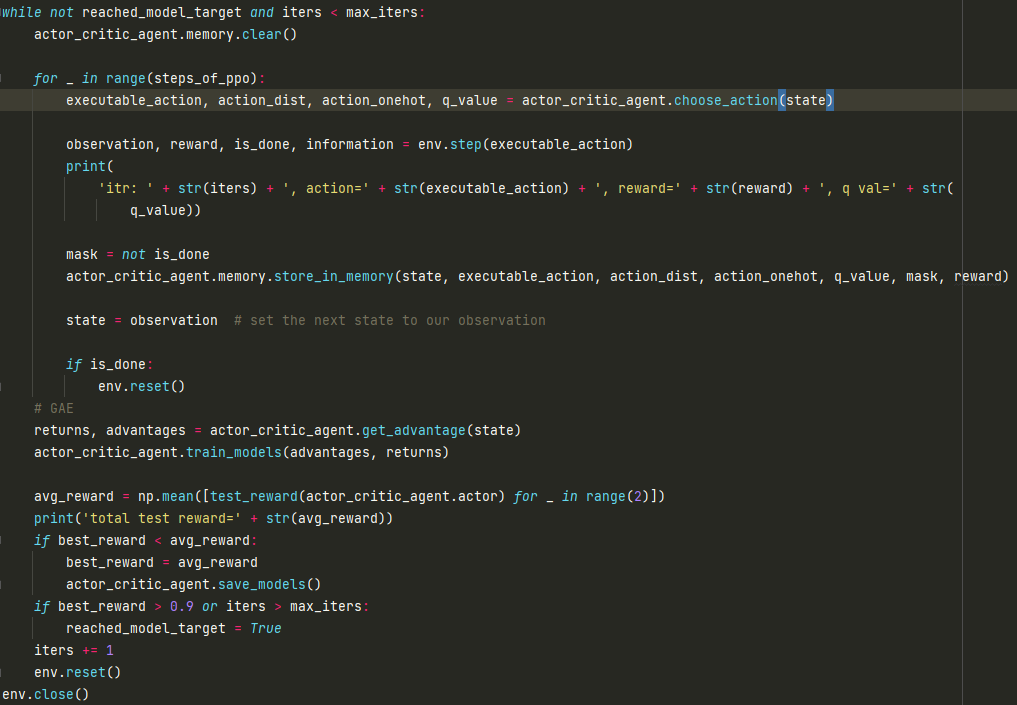
Ezzel a legfőbb osztályok el is készültek, melyek kapcsolatát az alábbi osztálydiagramm írja le:

. ábra Felhasznált komponensek osztálydiagrammja

29. ábra A modellek tanításáért felelő metódus

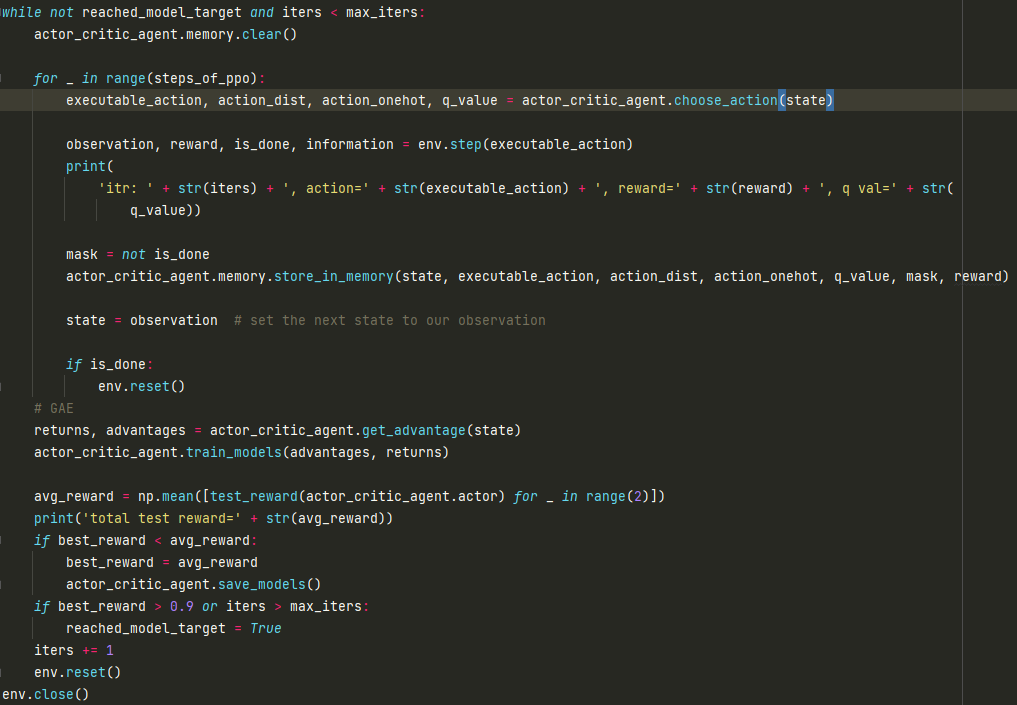
Nincs elérhető leírás.Nincs elérhető leírás.Azzal, hogy elkészült a tanuló ágensünk lényegében már csak a tanulási folyamat implementálása van hátra. Ehhez először inicializálnunk kell ezt az ágenst, illetve magát a környezetet is, amellyel interaktálni szeretnénk majd. Következő fontos lépés, hogy ürítsük ki a memóriát. Ezt követően x iteráción keresztül (steps\_of\_ppo), az éppen aktuális szabályrendszernek megfelelően, akciókat hajtunk végre a környezetben, melynek eredményeit, állapotait minden egyes lépésben a memóriába mentjük, amellyel számos tapasztalati mintát szerezhetünk. Amennyiben ebből eleget szereztünk, elkezdődhet a modellek tényleges tanítása, azaz a szabályrendszerének módosítása. Ehhez ki kell számítani egy advantage értéket, amelyet az ágens egy metódusában már definiáltunk. Ezt a lépéssorozatot addig kell ismételnünk, amíg nem lesz olyan modellünk, amely elég jó ahhoz, hogy végig vigye a játékot, vagy el nem értük a tanítás lépésszámának a határát. A modellek jóságát egy egyszerű kiértékelő módszerrel valósítottam meg, melynek a lényege, hogy a legutóbb frissített modell szabályrendszere alapján végig tudjuk-e vinni a játékot (jelen esetben góllal zárul az akció). Amennyiben ez 5-ből 5x sikeres, a tanulás befejezhető és az utolsó modell checkpoint-ja menthető, ellenkező esetben azonban egy új iterációval folytatjuk a tanulást. Ennek a folyamatnak a gyakorlati megvalósítása a következő:

30. ábra A modellek tanításának folyamata



31. ábra Modell kiértékelésének megvalósítása

. ábra Ágens tanulás gyakorlati megvalósítása



## *6.2. Összetettebb játékhelyzetre való tanítás*

# *Irodalomjegyzék*

[1] - Dick, S. (2019). Artificial Intelligence. Harvard Data Science Review, 1(1). https://doi.org/10.1162/99608f92.92fe150c

[2] - El Naqa, I., Murphy, M.J. (2015). What Is Machine Learning? In: El Naqa, I., Li, R., Murphy, M. (eds) Machine Learning in Radiation Oncology. Springer, Cham. https://doi.org/10.1007/978-3-319-18305-3\_1

[3] – A. K. Jain, Jianchang Mao and K. M. Mohiuddin, "Artificial neural networks: a tutorial," in Computer, vol. 29, no. 3, pp. 31-44, March 1996, doi: 10.1109/2.485891.

[4] - Jyh-Woei Lin [Artificial neural network related to biological neuron network: a review](http://www.m-hikari.com/asms/asms2017/asms1-2017/p/linASMS1-2017.pdf), Advanced Studies in Medical Sciences, Vol. 5, 2017, no. 1, 55-62, <https://doi.org/10.12988/asms.2017.753>

[5] - W.S. McCulloch and W. Pitts, “A Logical Calculus of Ideas Immanent in Nervous Activity,” Bull. Mathematical Biophysics, Vol.5 1943, pp. 115-133

[6] - K. Tamás, „A mesterséges neurális hálók a jövőkutatás szolgálatában,” 2002.

[7] – Kathrin Melcher - [A Friendly Introduction to [Deep] Neural Networks](https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks)

[8] -Szalóki Kristóf: [A reinforcement learning alapjai](https://dmlab.hu/blog/a-reinforcement-learning-alapjai/)

[9] - M. H. Sazlı, "A brief review of feed-forward neural networks", Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering, vol. 50, no. 01, pp. 0-0, Jan. 2006, doi:10.1501/commua1-2\_0000000026

[10] - L.R. Medsker, L.C. Jain (1999,) [Recurrent Neural Networks: Design and Applications](https://books.google.hu/books?hl=hu&lr=&id=ME1SAkN0PyMC&oi=fnd&pg=PA1&dq=recurrent+neural+networks&ots=7cwzdO-VYi&sig=c5TWiWuiDEUqsCPC8GTxGYwaDCk&redir_esc=y#v=onepage&q=recurrent%20neural%20networks&f=false)

[11] - D.T Pham, D Karaboga,Training Elman and Jordan networks for system identification using genetic algorithms, Artificial Intelligence in Engineering, Volume 13, Issue 2, 1999, Pages 107-117, ISSN 0954-1810, <https://doi.org/10.1016/S0954-1810(98)00013-2>., (https://www.sciencedirect.com/science/article/pii/S0954181098000132)

[12] - Barry J. Wythoff, Backpropagation neural networks: A tutorial, Chemometrics and Intelligent Laboratory Systems, Volume 18, Issue 2, 1993, Pages 115-155, ISSN 0169-7439, <https://doi.org/10.1016/0169-7439(93)80052-J>. (https://www.sciencedirect.com/science/article/pii/016974399380052J)

[13] – Hollan Hauule - [Understanding Error Backpropagation, Towards Data Science](https://towardsdatascience.com/error-backpropagation-5394d33ff49b)

[14] - Miikkulainen R. (2011) Topology of a Neural Network. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA.   
https://doi.org/10.1007/978-0-387-30164-8\_837

[15] - V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017,   
doi: 10.1109/JPROC.2017.2761740.

[16] - S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, https://doi: 10.1109/ICEngTechnol.2017.8308186.

[17] – Keiron O’Shea, Ryan Nash, [An Introduction to Convolutional Neural Networks](https://arxiv.org/abs/1511.08458) , <https://doi.org/10.48550/arXiv.1511.08458>

[18] - Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). doi.org/10.1023/A:1010933404324

[19] - M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998, doi: 10.1109/5254.708428.

[20] - Billard, L., Diday, E. (2002). Symbolic Regression Analysis. In: Jajuga, K., Sokołowski, A., Bock, HH. (eds) Classification, Clustering, and Data Analysis. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-56181-8\_31

[21] - Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003). KNN Model-Based Approach in Classification. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds) On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. OTM 2003. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39964-3\_62

[22] - OpenAI - https://gym.openai.com/docs/

[23] - OpenAI - https://gym.openai.com/envs/#atari

[24] - OpenAI - https://gym.openai.com/envs/#box2d

[25] - OpenAI - https://gym.openai.com/envs/#classic\_control

[26] - OpenAI - https://gym.openai.com/envs/#mujoco

[27] - OpenAI - https://gym.openai.com/envs/#robotics

[28] - TensorFlow - [Machine learning education](https://www.tensorflow.org/resources/learn-ml?gclid=Cj0KCQjwjN-SBhCkARIsACsrBz44aeochtFPQ_HkU31Xy12aL1wpbnjbWRRViSITZ6_eUchbYH814M0aAmzHEALw_wcB)

[29] - Peter Goldsborought: A Tour of TensorFlow, <https://doi.org/10.48550/arXiv.1610.01178>

[30] - Keras - https://keras.io/about/

[31] - Keras BackEnd Utilities - https://keras.io/api/utils/backend\_utils/

[32] - [TensorFlow Agents](https://www.tensorflow.org/agents) - https://www.tensorflow.org/agents

[33] - [TensorFlow.js API](https://js.tensorflow.org/api/latest/) - https://js.tensorflow.org/api/latest/

[34] - [TensorFlow Lite API - https://www.tensorflow.org/lite/api\_docs](TensorFlow%20Lite%20API%20-%20https://www.tensorflow.org/lite/api_docs)

[35] – Theano Documentation - https://theano-pymc.readthedocs.io/en/latest/

[36] – Theano Developement Team, Theano: A Python framework for fast computation of mathematical expressions, https://arxiv.org/pdf/1605.02688.pdf

[37] - [Torch](http://torch.ch/) - http://torch.ch/

[38] - R. Collobert, S. Bengio, and J. Mariéthoz, „Torch: a modular machine learning software library,” Idiap, Tech. Rep., 2002.

[39] – Torch on Windows - https://github.com/torch/torch7/wiki/Windows

[40] - [Caffe Deep Learning Framework](https://caffe.berkeleyvision.org/) - https://caffe.berkeleyvision.org/

[41] - [Deeplearning4j](https://deeplearning4j.konduit.ai/) - https://deeplearning4j.konduit.ai/

[42] - [MATLAB & Simulink](https://www.mathworks.com/products/matlab.html) - https://www.mathworks.com/products/matlab.html

[43] - [ML.NET](https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet) - https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet

[44] - [Karol Kurach](https://arxiv.org/search/cs?searchtype=author&query=Kurach%2C+K), [Anton Raichuk](https://arxiv.org/search/cs?searchtype=author&query=Raichuk%2C+A), [Piotr Stańczyk](https://arxiv.org/search/cs?searchtype=author&query=Sta%C5%84czyk%2C+P), [Michał Zając](https://arxiv.org/search/cs?searchtype=author&query=Zaj%C4%85c%2C+M), [Olivier Bachem](https://arxiv.org/search/cs?searchtype=author&query=Bachem%2C+O), [Lasse Espeholt](https://arxiv.org/search/cs?searchtype=author&query=Espeholt%2C+L), [Carlos Riquelme](https://arxiv.org/search/cs?searchtype=author&query=Riquelme%2C+C), [Damien Vincent](https://arxiv.org/search/cs?searchtype=author&query=Vincent%2C+D), [Marcin Michalski](https://arxiv.org/search/cs?searchtype=author&query=Michalski%2C+M), [Olivier Bousquet](https://arxiv.org/search/cs?searchtype=author&query=Bousquet%2C+O), [Sylvain Gelly](https://arxiv.org/search/cs?searchtype=author&query=Gelly%2C+S): Google Research Football: A Novel Reinforcement Learning Environment, <https://doi.org/10.48550/arXiv.1907.11180>

[45] – Google Research Football - https://github.com/google-research/football

[46] – RoboCup - https://www.robocup.org/

[47] - [MuJoCo Soccer Environment](https://www.deepmind.com/open-source/mujoco-soccer-environment) - https://www.deepmind.com/open-source/mujoco-soccer-environment

[48] - [FIFA 22](https://www.ea.com/games/fifa/fifa-22) - https://www.ea.com/games/fifa/fifa-22

[49] - Matheus Prado Prandini Faria, Rita Maria Silva Julia, Lídia Bononi Paiva Tomaz: A Deep Reinforcement Learn-Based FIFA Agent with Naive State Representations and Portable Connection Interfaces

[50] – [Chintan Trivedi: Using Reinforcement Learning to play FIFA](https://github.com/ChintanTrivedi/DeepGamingAI_FIFARL) - https://github.com/ChintanTrivedi/DeepGamingAI\_FIFARL

[51] – Richard Dearden, NirFriedman, Stuart Russel (1998): [Bayesian Q-Learning](https://www.aaai.org/Papers/AAAI/1998/AAAI98-108.pdf), https://www.aaai.org/Papers/AAAI/1998/AAAI98-108.pdf

[52] – Markel Sanz Austin - [Reinforcement Learning. DQN: Q-Learning with Neural Networks](https://markelsanz14.medium.com/introduction-to-reinforcement-learning-part-3-q-learning-with-neural-networks-algorithm-dqn-1e22ee928ecd) - https://markelsanz14.medium.com/introduction-to-reinforcement-learning-part-3-q-learning-with-neural-networks-algorithm-dqn-1e22ee928ecd

[53] – [Megerősítéses tanulás, mély Q-hálók (u-szeged.hu)](https://www.inf.u-szeged.hu/~tothl/ann/10.%20Megerositeses%20tanulas.pdf)

[54] - Yin-Hao Wang, Tzuu-Hseng S. Li, Chih-Jui Lin, Backward Q-learning: The combination of Sarsa algorithm and Q-learning, Engineering Applications of Artificial Intelligence, Volume 26, Issue 9, 2013, Pages 2184-2193, ISSN 0952-1976, <https://doi.org/10.1016/j.engappai.2013.06.016>. (https://www.sciencedirect.com/science/article/pii/S0952197613001176)

[55] – Tompa Tamás, Kovács Szilveszter: Heurisztikusan gyorsított megerősítéses tanulás módszerek, doi: https://doi.org/10.35925/j.multi.2020.3.48

[56] - L. Busoniu, R. Babuska and B. De Schutter, "Multi-Agent Reinforcement Learning: A Survey," 2006 9th International Conference on Control, Automation, Robotics and Vision, 2006, pp. 1-6, doi: 10.1109/ICARCV.2006.345353.

[57] - C. Boutilier, “Planning, learning and coordination in multiagent decision processes,” in Proceedings Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96), De Zeeuwse Stromen, The Netherlands, 17–20 March 1996, pp. 195–210.

[58] - M. T. J. Spaan, N. Vlassis, and F. C. A. Groen, “High level coordination of agents based on multiagent Markov decision processes with roles,” in Workshop on Cooperative Robotics, 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02), Lausanne, Switzerland, 1 October 2002, pp. 66–73.

[59] - M. J. Mataric,´ “Learning in multi-robot systems,” in Adaptation and Learning in Multi–Agent Systems, G. Weiß and S. Sen, Eds. Springer Verlag, 1996, pp. 152–163.

[60] - R. H. Crites and A. G. Barto, “Improving elevator performance using reinforcement learning,” in Advances in Neural Information Processing Systems, vol. 8, 1996, pp. 1017–1023.

[61] - [Chintan Trivedi](https://github.com/ChintanTrivedi/rl-bot-football): RL Agent for the Google Football environment, https://github.com/ChintanTrivedi/rl-bot-football

[62] - Microsoft: [A gépi tanulási algoritmus módosítása](https://docs.microsoft.com/hu-hu/bonsai/inkling/advanced/algorithm), https://docs.microsoft.com/hu-hu/bonsai/inkling/advanced/algorithm

[63] - OpenAI: [Proximal Policy Optimization,](https://openai.com/blog/openai-baselines-ppo/) https://openai.com/blog/openai-baselines-ppo/

[64] - Chintan Trivedi: [Proximal Policy Optimization Tutorial](https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6) - <https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6>

[65] - Vishal Bidawatka: Google football environment – installation and Training RL agent using A3C - https://towardsdatascience.com/google-football-environment-installation-and-training-rl-agent-using-a3c-d058a44f0fad