SNAKE TANÍTÁSA GENETIKUS ALGORITMUSSAL

Neurális hálózatok házi feladat

Cseppentő Viktor
F2MJMI

Specifikáció

Feladat leírása

Neurális hálózat implementálása Java környezetben, mely genetikus algoritmus során megtanul Snake-et játszani, illetve a különbözően paraméterezett algoritmusok és a klasszikus módszerek összehasonlítása.

Motiváció

Egyre növekvő számítási erőforrásainkkal szinte bármilyen problémát meg lehet oldani neurális hálózatokkal és a videójáték-iparban is előszeretettel alkalmazzák mesterséges intelligencia megvalósítására, azonban kevés betanított Snake játékkal találkoztam eddig. Optimalizáláshoz azért genetikus algoritmust választottam, mert ugyanazokkal a paraméterekkel sokszor különböző, meglepő eredményekre juthatunk.

Játék leírása

A játék során egy kígyót irányítunk egy négyzethálós pályán és célunk, hogy a véletlenszerűen felbukkanó gyümölcsöket felszedjük. Ahogy szedjük a gyümölcsöket nő a méretünk és el kell kerülnünk, hogy ne ütközzünk falba vagy önmagunkba. Én egy kicsit még módosítottam a játékszabályt annyival, hogyha sokáig nem talál magának gyümölcsöt (éhezik) akkor is vége a játéknak.

Módszerek a mesterséges intelligencia megvalósítására

Kapzsi módszer

Ez a módszer arról szól, hogy ha látunk gyümölcsöt akkor megkeressük hozzá a legrövidebb utat és felszedjük, miközben kerülgetjük magunkat és az akadályokat. A játék elején nagyon jól működik, azonban amikor hosszabbak leszünk, akkor nem feltétlenül van út a gyümölcshöz. Ezt valósítják meg az útkereső algoritmusok, mint például a Dijkstra algoritmus vagy az A* algoritmus.

Megerősítéses tanulás

Megerősítéses tanulás során az ágensnek eleinte nincsen semmilyen tudomása a környezetről, és véletlenszerűen cselekszik. A cselekedeteket a környezet vagy jutalmazza vagy bünteti. A kígyó a jutalmak függvényében fogja eldönteni, hogy folytassa-e ezt a stratégiát vagy változtasson rajta. Ilyen módszer a Q-tanulás.

Genetikus algoritmus

Genetikus algoritmus alkalmazása során a kígyót megfeleltetjük egy DNS mintának, amibe bele van kódolva, hogy mikor mit cselekedjen. Eleinte véletlenszerűen generálunk sok DNS mintát és kiértékeljük milyen jól teljesítenek, meghatározzuk a fitneszüket. Ezek után a legjobbak mintáit összekeresztezzük így létrehozva a kígyók

következő generációját. Keresztezés után egy mutációt is végzünk az utódon, hogy fenntartsuk a genetikai sokszínűséget. Majd ezeket is kiértékeljük és létrehozunk belőlük egy újabb, jobb generációt. Ezt addig folytatjuk amíg nem vagyunk megelégedve az eredménnyel, vagy amíg nem tudnak tovább fejlődni.

Az alkalmazásomban az A* és a genetikus algoritmust implementáltam, ahol a DNS minták mindegyike egy-egy neurális hálózat.

Neurális hálózat architektúrája

Feladatom során saját neurális hálózat könyvtárat fejlesztettem. Teljesen csatlakoztatott visszacsatolás nélküli hálózatokat használtam.

Bemeneti neuronok

A használt hálóknak 14 bemeneti neuronja van. A kígyó öt irányba lát (balra, átlósan balra, egyenesen, átlósan jobbra, jobbra) és mindegyik irányban megnézi hogy van-e arra akadály vagy gyümölcs. Ez így eddig 10 neuron. A neuron aktivációja egyenlő a legközelebbi akadály vagy gyümölcs és a kígyó fejének a távolságának a reciprokával. Ha nincs abban az irányban gyümölcs akkor a neuron értéke 0. Egy bemenetként a háló a kígyó hosszát kapja meg, így ösztönözve az ágenst, hogy talán más stratégiát kell találnia ha nagyon megnyúlik. A maradék 3 bemeneti neuron azért felelős, hogy az ágens jobban kiismerje a környezetét. Ezek aktivációja attól függ, hogy a kígyó a feje melletti és előtte lévő helyeken járt-e már.

Kimeneti neuronok

A hálózatnak 3 kimeneti neuronja van. Mindegyik egy irányt mutat, hogy kanyarodjon balra, jobbra, vagy ne változtassa az irányát.

Algoritmus paraméterei

A fejlesztett alkalmazásban két tanítás között változtathatjuk az algoritmus paramétereit.

Rejtett rétegek

Megadhatjuk a rejtett rétegek számát és melyik rétegben mennyi neuron legyen. Így ki lehet próbálni bármilyen mély hálót, vagy akár rejtett réteg nélkülit is.

Aktivációs függvény

Különböző aktivációs függvényeket is használhatunk a neurális hálózathoz. Sigmoid, hiperbolikus tangens és ReLu függvényeket implementáltam.

Populáció nagysága

Ez adja meg, hogy genetikus algoritmus során egy generációba hány kígyó tartozzon.

Mutációs ráta

A mutációs ráta módosításával tudjuk a keresztezések utáni mutáció erősségét befolyásolni. Ez egy százalékszám, ami megmondja, hogy szaporításnál az utód DNS-ében a gének (hálózat súlyai és biasai) mekkora eséllyel legyenek különbözőek a szülők génjeitől.

Elitek száma

Miután kiértékelünk egy generációt és találunk néhány nagyon jó példányt, nem feltétlen akarjuk elveszteni a génállományaikat, hanem egy az egybe áttehetjük őket a következő generációba is. Ez a módszer képes meggyorsítani az algoritmust és megadhatjuk, hogy generációnként mennyi kígyót emeljünk át a következőbe.

Játéktér

3 különböző 25x25-ös pályát készítettem az algoritmusok kiértékeléséhez:

- könnyű, nincs akadály rajta csak oldalfalak.
- közepes, kevés akadállyal.
- nehéz , nagyon sok akadállyal, illetve néhány helyet csak egyféleképpen lehet megközelíteni.

Segédfüggvények

Az algoritmus megfelelő működéséhez megfelelően kell implementálni néhány függvényt.

Fitnesz függvény

A fitnesz függvény mondja meg, hogy egy ágens mennyire teljesített jól. Mivel a játék célja, hogy minél hosszabbra nyúljunk meg és támogatni kell a környezet minél jobb kiismerését ezért a kígyó hossza és a felfedezett területek száma határozza meg a az értéket. Kódrészlet:

```
double fitness = log(10 * explored.size()); //Felfedezett blokkok száma
if (getLen < 8) //Kígyó hossza
        fitness *= pow(2, getLen() – 3);
else
        fitness *= 256 * (getLen() – 7);
return fitness;
```

Kiválasztás

Minden utódnak kettő szülője van, amiket az utolsó generációból választunk ki. A kiválasztás valószínűsége a fitnesz értékkel arányos.

Szaporítás

Miután kiválasztottuk a szülőket keresztezzük a DNS-üket. Egyenletes keresztezést használtam, tehát minden egyes génnél ugyanannyi az esélye, hogy az melyik

szülőtől származik. Mutáció során pedig egyenletes eloszlással adunk a géneknek véletlenszerű értéket. Például ha 2% a mutációs ráta, akkor 49%, hogy egy gén az egyik szülőtől származik, 49%, hogy a másik szülőtől és 2%, hogy véletlenszerű értéket kap a gén.

Genetikus algoritmus kiértékelése

Tesztelés során 30 darab különbözően paraméterezett algoritmust próbáltam ki, melyek mindannyian máshogy teljesítettek.

Általános észrevételek

A könnyebb pályákon többnyire azt a stratégiát tanulták meg, hogy körbe-körbe mennek és végig csak egy irányba kanyarodnak. Ez eleinte jól működik, azonban amikor megnyúlnak hamar bekerítik önmagukat és utána nem igazán képesek újabb stratégiákat kipróbálni, egy evolúciós gödörben maradnak és megáll a fejlődés. A tanh aktivációs függvényű hálók tanulnak a leggyorsabban és teljesítenek a legjobban, azonban ők is csak egy pályán tanulnak meg jól játszani, másik pályán hamar veszítenek. A ReLu-s hálók nagyon nehezen tanulnak, a sigmoid aktivációs függvényűek pedig szinte mindig ugyanazt a stratégiát tanulják meg függetlenül a háló struktúrájától. A mélyebb (legalább 3 rejtett rétegű) hálók ritkán teljesítenek jól. A legjobbak az egy rejtett rétegű 20-50 rejtett neuront tartalmazó struktúrák. Minél nehezebb a pálya, annál nagyobb számú populációt érdemes generálni, hogy sokszínűbbek legyenek és konvergáljon a tanítás.

Könnyű pálya

Az akadály nélküli pályán a legjobb eredmény 72 pont lett. Ezt az eredményt egy tanh aktivációs függvényű hálózat érte el 1 rejtett réteggel és 20 rejtett neuronnal mindössze 4 generáció alatt. Ezek néha kockázatos és érdekes mozdulatokat mutatnak. A sigmoid-os hálók nagyon gyorsan megtanulták a körbe-körbe stratégiát, de 65 pontnál többet nem nagyon tudtak szerezni, nem képesek jobb stratégiára. A ReLu-s hálók szintén a körbe-körbe stratégiát tanulták meg csak sokkal lassabban.

Közepes pálya

Itt is többnyire lehet alkalmazni az egy irányba kanyarodós módszert, azonban 40 pontnál nem lehet sokkal többet szerezni vele. Itt is tanh-os háló volt a legjobb 58 ponttal, amelyik érdekes, nem kiszámítható stratégiát alkalmazott. A többi hálókra itt is jellemzőek a könnyű pályás megjegyzések.

Nehéz pálya

Ez a pálya tele van akadállyal, nagyon sok generáció kell mire megtanul egy struktúra játszani. Sigmoid-os hálóknak egyáltalán nem megy a tanulás. A tanh-os hálók főleg azt a stratégiát találták ki, hogy a falat követik végig, ha látják a gyümölcsöt akkor gyorsan felszedik és gyorsan mennek vissza a falhoz. Ezzel a

módszerrel nagyon jól teljesítenek és 30 pontot is könnyen szereznek. A ReLu-s hálók szintén képesek ugyanerre, csak nekik sokkal több időbe telik.

Összehasonlítás A* algoritmussal

Az A* algoritmus garantálja azt, hogy ha eléri a gyümölcsöt akkor fel is szedi. Ez nem a kígyó látóterét és hosszát kapja meg, hanem az összes akadály és a gyümölcs helyzetét és ez így egy kicsit csalás a genetikus algoritmussal szemben, mivel több információja van. Könnyű pályákon ez jobban teljesít, mint a genetikus algoritmus, csak akkor keríti be önmagát ha nagyon hosszúra megnyúlik. Nehezebb pályákon azonban a kapzsisága miatt hamar eltorlasztja önmagával az utat a gyümölcshöz és kénytelen a testébe menni. Emiatt a falat követő módszerek felülmúlják ezt, így mondható, hogy a neurális hálózatok a nehéz pályákon jobban teljesítenek mint a klasszikus A* algoritmus.

Alkalmazás további funkciói

Az alkalmazásban az A* és a genetikus algoritmus lefuttatásán kívül, módosíthatjuk a hiperparamétereket, újranézhetjük a legutolsó tanítás összes egyénét, kimenthetjük a tanítást egy excel fájlba, ahonnan további statisztikákat szűrhetünk le és a legjobban teljesítő hálózatokat kimenthetjük fájlokba, amiket bármikor betölthetünk és újranézhetjük a játékmenetüket. Ezen kívül írtam egy segédalkalmazást, amikbe betöltve a kimentett neurális hálózatokat megnézhetjük a struktúrájukat és paramétereiket. Az összes kód, a legjobb hálózatok és az eredmények megtalálhatók a github oldalamon.

Továbbfejlesztési lehetőségek

A későbbiekben szeretném a tanítást Q-tanulással is megoldani, úgy hiszem azzal még jobb eredményre jutnék. Emellett a mostani neurális hálózatot kicserélném konvolúciós hálóra és úgy ugyanazt az információt kapja meg a hálózat a játéktérről, mint az A* algoritmus.