VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technológií

Dice Wars

(Umelá inteligencia)

Autori: Michal Kabáč (xkabac00)
Maroš Kabáč (xkabac01)
Bettina Pinkeová (xpinke00)
Daniel Kavuliak (xkavul01)

1. Úvod

Cieľom projektu je vytvoriť umelú inteligenciu, ktorá hrá ťahovú strategickú hru Dice Wars. Jedná sa o nedeterministickú hru fungujúcu na princípe zero sum game. Umelou inteligenciou by sme následne mali prehľadávať stavový priestor s využitím prvkov strojového učenia. Bežne sa používajú pre riešenia týchto typov úloh rôzne typy heuristík, napríklad Expectiminimax. My sme sa pre riešenie tejto úlohy rozhodli použiť zaujímavejšiu alternatívu, ktorou je učenie posilovaním (ang. reinforcement learning). Toto učenie je vhodné pre tento typ problému, nakoľko nemáme k dispozícií trénovacie dáta, prípadne správne odpovede ako mal daný ťah vyzerať. Vytvorili sme agenta, ktorý sa učí na základe akcií ktoré vykonáva. Tieto akcie su ohodnotené nejakou odmenou, ktorú dosiahne po ich vykonaní. Agent sa snaží odmenu maximalizovať. V našej práci sme nadviazali na implementáciu Dice Wars z github repozitára[1].

Cieľom tejto hry je poraziť ostatných hráčov a ovládnuť celú mapu. Mapa sa skladá z políčok, na ktorých sú umiestnené kocky hráčov a na jednom políčku sa nachádza 1 až 8 kociek. Každý hráč na začiatku začína s 3 hracími kockami. Hráč môže napádať územia nepriateľských hráčov zo svojich území, útok môže vykonať vtedy, ak je počet kociek na políčku viac ako 1. Políčka z ktorých sa útočí musia byť susedné. Pri útoku sa sčítajú kocky útočiaceho hráča a kocky brániaceho hráča a vyhráva ten hráč, ktorý má väčší súčet hodnôt kociek. Obranca vyhráva aj vtedy, ak sa rovnajú súčty kociek oboch hráčov. Hráč môže kedykoľvek ukončiť ťah. Po ukončení ťahu sa hráčovi pridajú kocky na každé jeho územie podľa počtu území v najväčšej súvislej oblasti.

Na základe týchto pravidiel sme vytvorili reprezentáciu stavového priestoru, v ktorom reprezentujeme prislúchajúce hodnoty každého políčka (viac v kapitole x.y). Taktiež počítame aj možnú odmenu, ktorú sme schopný získať po dobytí súperovho políčka. Pre agenta sme definovali neurónovú sieť (viac kapitola x.y), ktorú sme na základe vykonaných ťahov trénovali. Výsledky tejto siete je možné vidieť v kapitole (x.y). Po dosiahnutí uspokojivých výsledkov sme tieto váhy uložili a následne sme ukončili ďalšie trénovanie siete.

[1] https://github.com/ibenes/dicewars

2. Návrh riešenia

Aby sme predstavili návrh inteligentného agenta tejto hry, je potrebné, aby sme si vysvetlili princíp a pojmy posilovaného učenia. Posilované učenie je jednou zo základných metód spolu so supervízovaným učením a učením bez učiteľa. Zakladá sa na odmene a treste modelu, za každú akciu dostane odozvu, či vykonal správnu akciu alebo nie. Pod pojmom akcia si môžeme predstaviť činnosť, pomocou ktorej agent mení prostredie, do ktorého je agent zasadený. Tieto akcie vykonáva v prostredí v ktorom sa nachádza a každou vykonanou akciou vytvára nasledujúci stav hry. Z tohto vyplýva, že stav je prostredie po vykonaní akcie agenta. Odmena je vypočítaná pomocou vzorca definujúceho stratégiu, ktorú sa má model naučiť. Táto odmena môže mať záporný alebo kladný charakter, čo znamená, že buď model bude vykonávať túto akciu v budúcom ťahu alebo nebude ju naďalej vykonávať.

Náš návrh spočíva v definovaní stavového priestoru, návrhu stratégie odmeňovania a implementačného návrhu.

2.1 Návrh stavového priestoru

Náš stavový priestor sme si reprezentovali na základe jednotlivých políčok hracej plochy. Tento stav bol reprezentovaný v 29 políčkach. Každé políčko má hodnoty: identifikačné číslo políčka, identifikačné číslo vlastníka políčka, boolovská hodnota možnosti útoku na políčko (agent nemôže vykonať útok ak políčko nesusedí s jeho územím alebo nemôže vykonať útok na svoje vlastné územie), počet kociek, boolovská hodnota príslušnosti do najväčšej oblasti. Ku každému políčku sú uvedené susedné políčka a k nim pravdepodobnosti úspešnosti útoku. Všetky tieto hodnoty reprezentujú náš stavový priestor, ktorý slúži agentovi pre výber jednotlivých akcií, ktoré vykoná.

2.2 Návrh stratégie odmeňovania

Stratégiu odmeňovania sme zvolili nasledujúcim spôsobom. Dostali sme príslušnú mapu a meno hráča. Príslušná mapa bola reprezentovaná v aktuálnom stave. Pre tento aktuálny stav vypočítame najväčší región daného hráča v aktuálnom a predchádzajúcom kole. Vytvoríme si zoznam políčok predchádzajúceho a aktuálneho kola. Zo zoznamov si vypočítame ich dĺžku.

Odmeny sú udeľované v prípade, že aktuálny počet území ja väčší ako predchádzajúci. V tom prípade sa ku celkovej odmene priráta príslušná hodnota zo vzorca. Ako vzorec je použitý: | aktuálny počet území - predchádzajúci počet území | * 0,0001. V prípade, že je počet území menší, výsledná hodnota je odpočítaná od celkovej odmeny.

Nasledujúca odmena počíta so zväčšením regiónu, na základe ktorého hráč dostane viac kociek. Táto odmena je počítaná ako | veľkosť najväčšieho regiónu v aktuálnom ťahu - veľkosť najväčšieho regiónu v predchádzajúcom ťahu | * 0,0005. Na základe toho, či je nový región väčší, alebo menší ako predchádzajúci sa výsledná hodnota zo vzorca buď pričíta, alebo odpočíta od celkovej odmeny.

Posledná odmena je udeľovaná v prípade, že pravdepodobnosť útoku je väčšia ako 0,5 (polovica). V prípade, že táto podmienka bola splnená, ku celkovej odmene sa pričíta 0,0002. Ak podmienka splnená nie je, táto hodnota sa odčíta.

2.3 Návrh implementácie

Na základe vyššie spomenutých spôsobov reprezentácie stavového priestoru a spôsobu návrhu odmeňovanie, plánujeme implementovať učenie posilovaním. Toto učenie bude obsahovať neurónové siete, ktorých inputom bude stavové prostredie a budú trénované na základe odmien. Ich výstupom je príslušná akcia ktorá sa vykoná. Detailná implementácia je popísaná v kapitole 3.

3. Implementácia riešenia

Naše riešenie bolo implementované v programovacom jazyku Python. Pre trénovania neurónových sietí bola použitá knižnica Keras nad backendom Tensorflow verzie 2.2.0. Zadaný problém sme sa rozhodli riešiť implementáciou Deep Reinforcement learning modelu. Vstupom do tohto modelu je reprezentácia aktuálneho stavu hry vo forme dvojrozmernej matice. Pred vstupom do neurónovej siete je táto mapa transformovaná z 2D na 1D použitím flatten vrstvy. Výstupom neurónovej siete sú aproximované Q-hodnoty pre každý dostupný útok. Pri dueloch a hre 4 hráčov sa na mape nachádza celkovo 29 políčok, o ktoré sa bojuje. Teda výstupom neurónovej siete je vektor o dĺžke 29*29 prvkov, kde každá hodnota predstavuje osobitnú akciu - útok z políčka A na políčko B. Vzhľadom na fakt, že nie všetky z týchto ťahov sú reálne v každom kole dostupné, pred ďalším spracovaním sme vyfiltrovali iba dostupné ťahy. Po vyfiltrovaní týchto ťahov sme si vybrali ten, ktorý má najväčšiu Q-hodnotu. Tento ťah je vrátený funkciou ai_turn() ako ťah, ktorý má byť vykonaný. Prehľad našich sietí a použitých vrstiev:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 551)	0
d1 (Dense)	(None, 1200)	662400
leaky_re_lu_3 (LeakyReLU)	(None, 1200)	0
d2 (Dense)	(None, 1000)	1201000
leaky_re_lu_4 (LeakyReLU)	(None, 1000)	0

dout (Dense) (None, 841) 841841

leaky_re_lu_5 (LeakyReLU) (None, 841) 0

Total params: 2,705,241

Trainable params: 2,705,241

Non-trainable params: 0

3.1 Proces trénovania siete

Používame 2 neurónové siete. Prvá z nich - Target_model - je semi-statická. To v našom prípade znamená, že nie je priebežne trénovaná priamo, ale načíta váhy pre rozhodovanie zo súboru "xkabac00_weights.h5". Pre všetky finálne výbery ťahov je použitá táto sieť.

Druhá sieť - Q_model - je použitá pre trénovanie a úpravu váh, ktoré sa priebežne počas trénovania ukladajú, a vždy po 30 ťahoch našej siete sa prepíšu do Target-modelu.

Proces trénovania funguje tak, že pri každom ťahu si do inštančnej premennej self.memory ukladáme objekt, ktorého obsahom je stav hry pred vykonaním akcie, a taktiež samotná vykonaná akcia(akcia je uložená ako index ťahu z poľa všetkých ťahov). Po vykonaní ťahu sa spätne do tohto objektu doplní nový stav hry, a taktiež vypočítaná odmena, ktorú daná akcia vyprodukovala.

Trénovanie začína prebiehať vtedy, keď je veľkosť tejto pamäte aspoň 40 údajov (teda naša sieť musí vykonať aspoň 40 ťahov počas danej hry). Z pamäte sa náhodne vyberie tzv. minibatch, ktorý obsahuje 40 objektov. Každý z týchto objektov teda obsahuje:

< Stav hry pred vykonaním akcie | Stav hry po vykonaní akcie | Vykonaná akcia | Odmena za vykonanú akciu >

Následne pokračuje trénovanie krokmi:

1. Q_model vykoná aproximáciu Q-hodnôt pre všetky akcie (841 hodnôt) nad pôvodným stavom hry. 841 Q-hodnôt je uložených pod názvom q_pred.
2. Target_model vykoná aproximáciu Q-hodnôt pre všetky akcie nad novým stavom hry (teda po vykonaní akcie), výsledok je uložený v premennej t_pred
3. Upraví sa Q-hodnota pre vykonanú akciu v q_pred podľa vzorca: q_pred[index vykonanej akcie] = Odmena + γ*max(t_pred) , kde γ predstavuje konštantu pre tzv. Discount factor, ktorý pomáha ku konvergencii váh neurónovej siete. Jedná sa o Bellmanov vzorec, ktorý sa bežne používa pri Q-learningu a Reinforcement learningu.
4. Následne je Q_model trénovaný - vstupom(Xs) je pôvodný stav hry, a cieľom(Ys) sú Q-hodnoty všetkých ťahov s upravenou Q-hodnotou pre vykonaný ťah. Použitá je funkcia model.fit().

Tieto kroky sa následne opakujú pre každý zvolený "experience memory" objekt. Ako už bolo spomenuté vyššie, po 30 ťahoch sú váhy z Q_modelu prepísané do Target_modelu.

3.2 Ladenie modelu

Počas testovania nášho modelu sme dosiahli rôzne vylepšenia, ale aj narazili na rôzne problémy. Sú nimi:

- Lepší výpočet odmeny - pri pridaní negatívnych odmien sa model dokázal lepšie naučiť, ktoré ťahy nie sú najvhodnejšie, čo celkovo zlepšilo výsledky.
- Lepšie nastavené hyperparametre modelu - náš prvý model mal veľmi zlé výsledky a v dueloch nedokázal nikoho poraziť. Bolo to preto, že bol moc jednoduchý a teda nebol vhodný na riešenie tejto úlohy. Opakom bola druhá verzia, kedy bol model naopak moc zložitý, a veľmi rýchlo dochádzalo k pretrénovaniu a následnému zaseknutiu v lokálnom minime. Pri upravení modelu na niečo medzi spomínanými modelmi sa nám podarilo v dueloch poraziť všetky referenčné implementácie.
- Dlhší tréning - Čím viac hier na trénovanie má model k dispozícii, tým lepšie výsledky dosahuje. Chybou, ktorú sme urobili pri trénovaní, bolo trénovanie pre duely, a nie trénovanie pre turnaj 4 hráčov, ktorý je podstatnejší v rámci hodnotenia. Tento fakt sme si uvedomili až pomerne

neskoro pred odovzdaním projektu, a teda sme bohužiaľ nestihli sieť nanovo natrénovať.

- Trénovanie siete proti samej sebe - Z počiatku sme trénovali sieť v súbojoch proti poskytnutým modelom, čo síce fungovalo, ale takéto trénovanie nebolo úplne efektívne. Po trénovaní siete proti sebe samej sa tréning výrazne zrýchlil.

4. Záver

Implementovali sme funkčný model pre reinforcement learning známy ako Double Deep Q-learning. Implementácia síce nevyšla úplne podľa našich predstáv, ale model je funkčný a pri dueloch porazil aj referenčné modely spomínané v zadaní projektu. Pri turnajoch avšak nedosahuje moc dobré výsledky. Toto je spôsobené viacerými faktormi, najmä celkovým nedostatkom času pre trénovanie siete. Vzhľadom na fakt, že sme s týmto modelom pracovali prvý krát, implementácia nám zabrala omnoho viac času než sme očakávali, a teda sme nestihli natrénovať model tak, ako by sme chceli.