Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko

Seminarska naloga pri predmetu Osnove Umetne Inteligence

Primerjava nevronske mreže naučene z genetskim algoritmom in algoritma spodbujevalnega učenja v igri Kača

Avtorja:
Andrej Hafner,
Anže Mur

Mentor:
prof. dr. Zoran Bosnić

Škofja Loka, december 2018

Kazalo vsebine

1. Uvod

Ideja naše seminarske naloge je naučiti dva algoritma strojnega učenja igrati igro kača, ter narediti primerjavo med njima. Cilj je ugotoviti kateri algoritem je bolj primeren za podano nalogo. Razvoj je potekal v programskem jeziku Python kjer smo razvili oba algoritma ter uporabniški vmesnik (igro).

2. Pospeševalno učenje

2.1 Izbira in opis algoritma

Za algoritem pospeševalnega učenja smo si izbrali algoritem Q-learning. Cilj algoritma je agenta naučiti strategijo, ki agentu v danih okoliščinah pove katero akcijo naj izvede. Algoritem ne zahteva modela okolja ampak je okolje potrebno razdeliti v različna stanja preko katerih lahko agent prehaja z točno definiranimi akcijami. Vsak prehod med stanji je nagrajen z nagrado, ki pa je lahko pozitivna ali negativna - tako agenta naučimo katere akcije so v določenem stanju zaželene in katere niso. Posledica tega ali je bila akcija nagrajen ali kaznovana, je povečava oz. zmanjšanje pogostosti pojavljanja tega vzorca v kasnejših stanjih.

Q-learning najde strategijo, ki je optimalna v smislu da maksimizira pričakovano vrednost skupne nagrade pri vseh uspešnih korakih iz začetnega stanja. Q predstavlja funkcijo, ki vrne nagrado, uporabljeno za učenje algoritma. Dobljeno nagrado si lahko razlagamo kot kvaliteto (quality) izbrane akcije v danem stanju. Q definiramo kot:

$$Q: \ S \, x \, A \rightarrow R$$

Pred začetkom učenja je ***Q*** inicializiran na poljubno vrednost (odvisno od implementacije). Potem pa v vsakem trenutku ***t*** agent izbere akcijo a_t, pridobi nagrado r_t in vstopi v novo stanje s_{t+1} in vrednost funkcije ***Q*** se posodobi.

Bistvo algoritma je preprosta posodobitev ponovitve vrednosti (Markov decision process - value iteration update), ki uporabi uteženo povprečje stare vrednosti in nove informacije:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

kjer je r_t nagrada, ki jo je agent prejel pri premikanju iz stanja s_t v stanje s_{t+1} in **α** ocena učenja.

Na voljo imamo več vedenjskih strategij algoritma:

- ***greedy*****:** vedno izbere najboljše dejanje
- ***ε-greedy*****:** večinoma izbere najboljše dejanje, občasno z verjetnostjo ***ε*** pa tudi naključno

2.2 Implementacija algoritma za igro Kača

Pri implementaciji je bilo potrebno definirati vse zahteve, ki jih podani algoritem potrebuje za delovanje:

Stanje

Stanje smo določili s pomočjo tabele z šestimi vrednosti (0 ali 1):

- clearAhead
- clearLeft
- clearRight
- appleAhead
- appleLeft
- appleRight

clearAhead - Pove ali se lahko kača varno premakne naprej v trenutni smeri.
clearLeft - Pove ali lahko kača varno zavije levo.
clearRight - Pove ali lahko kača varno zavije desno.
appleAhead - Pove ali se jabolko nahaja v smeri kače.
appleLeft - Pove ali se jabolko nahaja levo od kače.
appleRight - Pove ali se jabolko nahaja desno od kače.

Kača se varno premakne, če se ne zaleti sama vase ali v rob igralnega polja.

Nagrajevanje

Kača dobi nagrado vrednosti 1, če se v danem stanje premakne proti jabolku in nagrado (kazen) -1, če se v danem stanju zaleti sama vase ali v rob igralnega polja.

Akcije

Kača lahko v danem trenutku nadaljuje pot naravnost (**FORWARD** = 0), zavije levo (**LEFT**=1) ali pa zavije desno (**RIGHT**=2). Akcijo izberemo s pomočjo Q-tabele iz katere dobimo najbolj optimalno akcijo za trenutno stanje, če smo v tem stanju že bili. Pri izbiri akcije smo uporabili tudi strategijo ***ε-greedy***, da smo omogočili učenje agenta v prvih interacijah. Za vrednost ***ε*** smo v začetku izbrali 1 in potem v vsaki iteraciji generirali število med 0 in 1 ter preverili če je manjše od ***ε***. Če je bila generirana vrednost manjša od ***ε*** smo namesto najbolj optimalne akcije izvedeli naključno akcijo in pomnožili ***ε*** z 0.99. S tem smo dosegli da se je agent na začetku učil in raziskoval.

Q-tabela

Pri implementaciji algoritma smo uporabili Q-tabelo, v kateri smo hranili vsa pretekla stanja ter izbrano akcijo za dano stanje ter vrednost funkcije Q za dano stanje in oceno. Q-tabelo smo uporabili za izbiranje najbolj optimalnih akcij v danem stanju ter za izračun nove Q vrednosti za novo stanje in akcijo.

Potek algoritma pri igranju kače:

1. Dobimo trenutno stanje
2. Glede na Q-tabelo in naključni faktor (***ε)*** izberemo akcijo.
3. Izvedemo novo akcijo.
4. Glede na izvedeno akcijo nagradimo agenta.

2.3 Rezultati

Velik vpliv pri nadaljnji performanci agenta ima nekaj začetnih iteracij igranja, saj se v teh iteracijah agent nauči največ in je še čisto naiven. Ugotovili smo, da je najbolj pomembno pri implementaciji dobro definirati funkcijo stanja saj je to dejanska komunikacija agenta z okoljem. Več kot bo imela funkcija parametrov več informacije o okolju bo agent sprejel. Dobro je potrebno tudi definirati funkcijo nagrajevanja saj to močno vpliva na obnašanje agenta. Sčasoma postane očitno, da naš agent na prvo mesto postavlja to, da bi se čimbolj približal jabolku (nagradi) ne pa tega, da bi ostal živ. To prioriteto bi se za izboljšavo algoritma moralo spremeniti saj v končni fazi agent lahko dobi več nagrade, če dalj časa ostane živ. Do takega obnašanja pride zato ker agenta v vsakem prehodi iz starega v novo stanje nagradimo če se približa jabolku, kaznujemo ga pa samo v primeru da se agent zaleti. Agentov najboljši rezultat je bil 48 (48 pobranih jabolk) v prvih 100 iteracijah potem pa pa je se je rezultat gibal okoli 20 na iteracijo.

Prikaz delovanja algoritma si je možno ogledati v dodanem videu ali pa z ukazom py ./snake.py v mapi q_learning_algorithm.

3. Učenje nevronske mreže z genetskim algoritmom

3.1 Opis algoritma

Za učenje nevronski mrež potrebujemo velike količine podatkov, ki pa za določene vhode in izhode nevronske mreže niso na voljo. Nevronsko mrežo pa lahko (sicer slabše) naučimo z genetskim algoritmom.

Ideja je, da je uteži nevronske mreže osebek v genetskem algoritmu. Na začetku naključno ustvarimo populacijo nevronskih mrež (inicializiramo uteži), nato ocenimo kvaliteto teh nevronskih mrež. Izberemo določeno število najboljših nevronskih mrež, ki bodo starši za naslednjo populacijo. Genome staršev med seboj križamo, nato pa izvedemo še mutacije z določeno verjetnostjo, da ohranjamo raznolikost v populaciji. Dodamo še elitizem, kar pomeni da se v naslednjo generacijo vedno prenese najboljši osebek iz prejšnje generacije, brez križanja in mutacij. Dobimo novo generacijo osebkov, katerim ponovno ocenimo kvaliteto. To ponavljamo dokler nismo zadovoljni oziroma algoritem konvergira.

Opis nevronske mreže

Nevronska mreža je sestavljena iz vhodnega nivoja, dveh skritih nivojev in izhodnega nivoja. Na vhodnem nivoju imamo 7 nevronov, prvi skriti nivo je sestavljen iz 9 nevronov, drugi skriti nivo iz 15 nevronov in izhodni nivo iz 3 nivojev. Med nivoji so med seboj povezani vsi nevroni. Aktivacijska funkcija na skritih nivojih je ***tanh,*** na izhodnem nivoju pa ***softmax****.* Genom nevronske mreže je predstavljen kot stolpični vektor vseh uteži.

Vrednosti nevronov na vhodni plasti so definirane kot:

1. **Nevron**: Če je kača blokirana spredaj **1**, sicer **0**
2. **Nevron**: Če je kača blokirana levo **1**, sicer **0**
3. **Nevron**: Če je kača blokirana desno **1**, sicer **0**
4. **Nevron**: X element normaliziranega vektorja glave kače
5. **Nevron**: Y element normaliziranega vektorja glave kače
6. **Nevron**: X element normaliziranega vektorja, ki kaže od glave kače proti hrani
7. **Nevron**: Y element normaliziranega vektorja, ki kaže od glave kače proti hrani

Izhodna vrednost nevronske mreže se določi tako, da se izbere nevron, ki je najbolj aktiviran.
Nevroni na izhodni plasti so definirani kot:

1. **Nevron**: Zavij levo
2. **Nevron**: Ohrani trenutno smer
3. **Nevron**: Zavij desno

V igri kača se pred vsakim korakom izračunajo vhodne vrednosti za nevronsko mrežo, ki nato določi smer kače v naslednjem premiku.

Opis genetskega algoritma

Na začetku se inicializira populacija **100** osebkov. Nato se oceni kvaliteta teh osebkov tako, da vsaka nevronska mreža igra kače, dokler se ne zabije v steno ali sama vase ali pa se izvede končno število korakov (v primeru da se kača ujame v zanko).

Ocena kvalitete osebkov nastane na podlagi naslednjih pravil:

- Če se kača zabije v steno ali sama vase, se igra zaključi in se oceni prišteje **-50**
- Če kača uspešno poje hrano, se ji prišteje **10**
- Če se kača premakne proti jabolku (Manhattanska razdalja), se ji prišteje **1**
- Če se kača premakne stran od jabolka (Manhattanska razdalja), se ji prišteje **-1.5**. To je dodano zato, ker želimo preprečiti da bi se kača ujela v zanko. Prepreči se zato, ker se ocena kvalitete samo niža, v primeru da je kača ujeta v zanko.

Nad populacijo se nato izvede selekcija. Izmed vseh osebkov vzamemo **10** najboljših. Slednji postanejo starši za naslednjo generacijo, ki jo ustvarimo z križanjem. Izvedemo enomestno križanje med dvema naključno izbranima staršema. To pomeni da v genomu (vektor uteži) izberemo naključno točko precepa, otrok dobi prvi del genoma od prvega starša, drugi del genoma pa od drugega dela starša. Nad nato izvedemo mutacijo z iteriranjem čez njegov genom, nato z verjetnostjo **7%** vsak element vektorja spremenimo na neko naključno vrednost med -1 in 1.
Nastane nova generacija, ki je enake velikosti kot začetna.

Slednje korake ponavljamo, dokler ne pridemo do ustavitvenega pogoja, ki je lahko končno število ponovitev ali pa ocena kvalitete, ki je nad določenim pragom.

3.2 Rezultati

Učenje uteži nevronske mreže z genetskim algoritmom počasi konvergira, ampak skoraj vsakič prinese solidne rezultate. Po 400 iteracijah algoritma dobimo nevronsko mrežo, ki igra igro v kateri povprečno doseže dolžino okoli 30, v najboljših primerih pa tudi čez 50. Algoritem igra kačo na nek poseben način, saj se velikokrat giblje v *cikcaku* in pa do naslednje hrane pride ob zidu. Implementacijo bi se najbrž še dalo izboljšati z spreminjanjem nevronske mreže, dodajanjem drugih ocen kvalitete osebkov in povečanjem števila vhodov v nevronsko mrežo.

Rezultati so prikazani v priloženem posnetku, učenje in spremljanje kače pa lahko zaženete tudi sami z zagonom datoteke *main.py* v mapi *genetic_algorithm.* Potrebovali boste knjižnice *pygame* in *numpy.*

4. Analiza rezultatov

Oba algoritma sta se izkazal solidna za igranje igre kače. Spodbujevalno učenje se v začetku učenja izkaže za boljšega. Pri spodbujevalnem učenju pride raven znanja do končne ravni po okoli 150 iteracijah. Nevronska mreža počasi napreduje, ampak v povprečju po 400 iteracijah doseže dobre rezultate, če pride do konvergiranja. Iz tega, kar sva opazila, lahko rečeva da se algoritem spodbujevalnega učenja obnese bolje, saj v povprečju dosega boljše rezultate. Nevronska mreža bi lahko pri večji populaciji v večjem številu generacij dosegla še boljše rezultate.

5. Viri

- https://en.wikipedia.org/wiki/Q-learning
- https://en.wikipedia.org/wiki/Markov_decision_process#Value_iteration
- https://sl.wikipedia.org/wiki/Okrepitveno_u%C4%8Denje#Q-learning
- https://becominghuman.ai/designing-ai-solving-snake-with-evolution-f3dd6a9da867
- https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a
- Kononenko Igor, Robnik Šikonja Marko - Inteligentni sistemi (2010), Založba FE in FRI