Primena dubokog Q učenja na automatsko igranje video igara

Matematički fakultet
Univerzitet u Beogradu

Student:
Nikola Milev

Mentor:
Mladen Nikolić

Članovi komisije:
dr Filip Marić, vanredni profesor
dr Aleksandar Kartelj, docent

Beograd, 2018.

Sadržaj

1 Uvod **1**

2 Mašinsko učenje **3**

2.1 Vrste mašinskog učenja 4

2.1.1 Nadgledano mašinsko učenje 4

2.1.2 Nenadgledano mašinsko učenje 5

2.2 Dizajn sistema za mašinsko učenje 6

2.2.1 Podaci 7

2.2.2 Evaluacija modela 8

2.3 Problemi pri mašinskom učenju 8

3 Neuronske mreže **10**

3.1 Neuronske mreže sa propagacijom unapred 11

3.1.1 Aktivacione funkcije 12

3.1.2 Optimizacija 14

3.1.3 Prednosti i mane 18

3.2 Konvolutivne neuronske mreže 19

3.2.1 Svojstva konvolucije 20

3.2.2 Slojevi konvolutivne neuronske mreže 21

3.2.3 Mane 21

4 Učenje potkrepljivanjem **22**

4.1 Osnovni pojmovi 22
4.2 Markovljevi procesi odlučivanja 23
4.2.1 Osnovni pojmovi 23
4.3 Rešavanje Markovljevih procesa odlučivanja 27
4.3.1 Učenje potkrepljivanjem u nepoznatom okruženju 30

5 Duboko Q učenje (DQN) 33
5.1 Struktura agenta 33
5.1.1 Q mreža 34
5.1.2 Memorija 35
5.2 Algoritam DQN 35

6 Detalji implementacije 37
6.1 Detalji implementacije 37
6.1.1 Prelazi i čuvanje u memoriji 39

7 Detalji treniranja i eksperimentalne evaluacije 41
7.1 Pretprocesiranje 41
7.2 Detalji treniranja 42
7.3 Detalji testiranja 43
7.4 Eksperimenti 45

8 Zaključak 48

Literatura 50

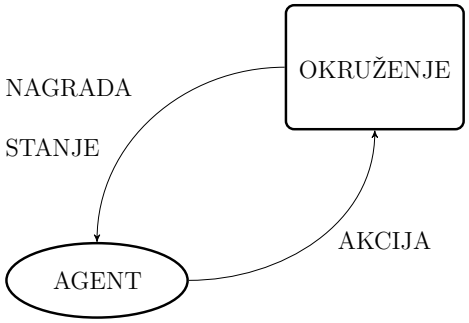
Glava 1

Uvod

U maju 1997. godine, Gari Kasparov, tadašnji svetski šampion u šahu, izgubio je meč protiv računarskog sistema pod nazivom Deep Blue. Skoro dvadeset godina kasnije, program pod nazivom AlphaGo pobedio je profesionalnog ljudskog igrača u igri go. Iako su obe igre strateške i igraju se na tabli, između šaha i igre go postoji ogromna razlika. Pravila igre go dosta su jednostavna u odnosu na šah ali je prostor koji opisuje poteze igre go više od 10^{100} puta veći od prostora koji opisuje poteze šaha. Programi koji igraju šah često se zasnivaju na korišćenju stabala pretrage i ovaj pristup jednostavno nije primenljiv na igru go.

Na čemu je onda zasnovan AlphaGo? U pitanju je učenje potkrepljivanjem (eng. reinforcement learning). Ovo je vrsta mašinskog učenja koja počiva na sistemu kazne i nagrade. Podrazumeva se da se sistem sastoji od agenta i okruženja u kom agent dela (vrši akcije) i dobija o numeričku nagradu i informaciju o promeni stanja okruženja. Osnovni dijagram ove komunikacije može se videti na slici 1.1. Poput dresiranja psa, nagradama se ohrabruje poželjno ponašanje dok se nepoželjno kažnjava. Cilj jeste ostvariti što veću dugoročnu nagradu. Međutim, agent mora sam kroz istraživanje da shvati kako da dostigne najveću nagradu tako što isprobava različite akcije. Takođe, preduzete akcije mogu da utiču i na nagradu koja se pojavljuje dugo nakon što je sama akcija preduzeta. Ovo zahteva da se uvede pojam dugoročne nagrade. Pojmovi istraživanja i dugoročne nagrade su ključni pri učenju potkrepljivanjem.

Pri učenju potkrepljivanjem, najčešće se pretpostavlja da je skup svih mogućih stanja



Slika 1.1: Dijagram komunikacije agenta sa okruženjem pri učenju potkrepljivanjem

okruženja diskretan. Ovo dozvoljava primenu Markovljevih procesa odlučivanja i omogućuje jednostavan formalan opis problema koji se rešava i pristupa njegovog rešavanja. Formalno predstavljanje problema i rešenja dato je u poglavlju 4.

Učenje potkrepljivanjem jedna je od tri vrste mašinskog učenja, pored nadgledanog i nenadgledanog učenja. Pri nadgledanom učenju, sistem dobija skup ulaznih i izlaznih podataka s ciljem da izvrši generalizaciju nad tim podacima i uspešno generiše izlazne podatke na osnovu do sada neviđenih ulaznih podataka. Pri učenju potkrepljivanjem, ne postoje unapred poznate akcije koje treba preduzeti već sistem na osnovu nagrade mora zaključi koji je optimalni sled akcija. Iako široko korišćeno, nadgledano učenje nije prikladno za učenje iz novih iskustava, kada izlazni podaci nisu dostupni. Kod nenadgledanog učenja, često je neophodno pronaći neku strukturu u podacima nad kojima se uči bez ikakvog pređašnjeg znanja o njima. Iako učenje potkrepljivanjem liči i na nadgledano i na nenadgledano učenje, agent ne traži strukturu niti mu je unapred data informacija o optimalnom ponašanju već teži maksimizaciji nagrade koju dobija od okruženja.

Učenje potkrepljivanjem ima mnogobrojne primene kao što su samostalna vožnja automobila i letelica, automatsko konfigurisanje algoritama, trgovina na berzi, igranje igara, itd. Ovaj vid mašinskog učenja pokazao se kao dobar i za igranje video igara. U radu objavljenom 2015. godine u časopisu Nature, DeepMind predstavlja sistem koji uči da igra video igre sa konzole Atari 2600, neke čak i daleko bolje od ljudi [16]. U avgustu 2017. godine, OpenAI predstavlja agenta koji isključivo kroz igranje igre i bez pređašnjeg znanja o igri stiče nivo umeća dovoljan da pobedi i neke od najboljih ljudskih takmičara u video igri Dota 2.[1]

U naučnom radu koji je objavila kompanija DeepMind u časopisu Nature predložen je novi algoritam, DQN (eng. *deep Q - network*), koji koristi spoj učenja uslovljavanjem i duboke neuronske mreže i uspešno savladava razne igre za Atari 2600 konzolu. Sve informacije dostupne agentu jesu pikseli sa ekrana, trenutni rezultat u igri i signal za kraj igre. U sklopu ovog rada, ispitana je struktura algoritma DQN i data je implementacija čije su performanse testirane na manjoj skali od one date u radu, zbog ograničenih resursa. Takođe je eksperimentisano sa elementima samog algoritma i opisano je kako oni utiču na njegovo ponašanje. Dva elementa, ciljna mreža i memorija, ključni su za uspešan proces učenja. Dobijeni rezultati pokazuju njihovu važnost.

U sklopu rada opisani su osnovni pojmovi mašinskog učenja (glava 2), zadržavajući se na neuronskim mrežama uopšte (glava 3) i na konvolutivnim neuronskim mrežama (deo 3.2). Glava 4 posvećena je učenju potkrepljivanjem dok je algoritam DQN u celosti opisan u glavi 5. U glavi 6 data je implementacija, njena evaluacija, kao i eksperimenti i njihovi rezultati.

[1] https://blog.openai.com/dota-2/

Glava 2

Mašinsko učenje

Mašinsko učenje počelo je da stiče veliku popularnost devedesetih godina prošlog veka zahvaljujući potrebi i mogućnosti da se uči iz velike količine dostupnih podataka i uspešnosti ovog pristupa u tome. Za popularizaciju mašinskog učenja početkom 21. veka najzaslužnije su neuronske mreže, u toj meri da je pojam mašinskog učenja među laicima često poistovećen sa pojmom neuronskih mreža. Ovo, naravno, nije tačno; sem neuronskih mreža, postoje razne druge tehnike, kao što su metod potpornih vektora, linearni modeli, probabilistički grafovski modeli, itd.

Mašinsko učenje nastalo je iz čovekove želje da oponaša prirodne mehanizme učenja kod čoveka i životinja kao jedne od osnovnih svojstava inteligencije i korišćenja dobijenih rezultata u praktične svrhe. Termin mašinsko učenje prvi je upotrebio pionir veštačke inteligencije, Artur Semjuel [18], koji je doprineo razvoju veštačke inteligencije istražujući igru dame (eng. checkers) i tražeći način da stvori računarski program koji na osnovu iskusva može da savlada ovu igru [1].

Mašinsko učenje može se definisati kao disciplina koja se bavi izgradnjom prilagodljivih računarskih sistema koji su sposobni da poboljšaju svoje performanse koristeći informacije iz iskustva [12]. No, u biti mašinskog učenja leži generalizacija, tj. indukcija. Dve vrste zaključivanja, indukcija[1] i dedukcija[2] imaju svoje odgovarajuće discipline u sklopu veštačke inteligencije: mašinsko učenje i automatsko rezonovanje. Kao što se indukcija i dedukcija razlikuju, i mašinsko učenje i automatsko rezonovanje imaju različite oblasti primene. Automatsko rezonovanje zasnovano je na matematičkoj logici i koristi se kada je problem relativno lako formulisati ali ga, često zbog velikog prostora mogućih rešenja, nije jednostavno rešiti. U ovoj oblasti, neophodno je dobiti apsolutno tačna rešenja, ne dopuštajući nikakav nivo greške. S druge strane, mašinsko učenje pogodnije je kada problem nije moguće precizno formulisati i kada se očekuje neki novo greške. Čovek neke od ovih problema lako rešava a neke ne. Ukoliko je neophodno napraviti sistem koji prepoznaje životinje na slikama, kako definisati problem? Koji su tačno elementi oblika životinje? Kako ih prepoznati? Metodama automatskog rezonovanja bilo bi nemoguće definisati ovaj problem i rešiti ga. Mašinsko učenje, s druge strane, pokazalo se kao dobar pristup. Ono što je još karakteristično za mašinsko učenje jeste da rešenje ne mora biti

[1] Indukcija – zaključivanje od pojedinačnog ka opštem
[2] Dedukcija – zaključivanje od opšteg ka konkretnom

savršeno tačno, iako se tome teži, i nivo prihvatljivog odstupanja zavisi od problema i konteksta primene.

Ova oblast je kroz manje od 20 godina od popularizacije postala deo svakodnevice. U sklopu društvene mreže Fejsbuk (eng. Facebook) implementiran je sistem za prepoznavanje lica koji preporučuje profile osoba koje se nalaze na slikama. Razni veb servisi koriste metode mašinskog učenja radi stvaranja sistema za preporuke (artikala u prodavnicama, video sadržaja na platformama za njihovo gledanje, itd.) i sistema za detekciju prevara. Mnoge firme koje se bave trgovinom na berzi imaju sisteme koji automatski trguju deonicama. U medicini, jedna od primena mašinskog učenja jeste za uspostavljanje dijagnoze. Još neke primene su u marketingu, za procesiranje prirodnih jezika, bezbednost, itd.

2.1 Vrste mašinskog učenja

Kada se govori o određenoj vrsti mašinskog učenja, podrazumevaju se vrste problema, kao i načini za njihovo rešavanje. Prema problemima koji se rešavaju, mašinsko učenje deli se na tri vrste: nadgledano učenje (eng. supervised learning), nenadgledano učenje (eng. unsupervised learning) i učenje potkrepljivanjem (eng. reinforcement learning). Iako se podela mnogih autora sastoji samo iz nadgledanog i nenadgledanog učenja, postoji razlika između učenja potkrepljivanjem i preostale dve vrste. U nastavku su dati opisi pristupa nadgledanog i nenadgledanog učenja. Učenju uslovljavanjem, kao centralnoj temi ovog rada, posvećeno je više pažnje u poglavlju 4.

2.1.1 Nadgledano mašinsko učenje

Pri nadgledanom mašinskom učenju, date su vrednosti ulaza i izlaza koje im odgovaraju za određeni broj slučajeva. Sistem treba na osnovu već datih veza za pojedinačne parove da ustanovi kakva veza postoji između tih parova i izvrši generalizaciju, odnosno, ukoliko ulazne podatke označimo sa x a izlazne sa y, sistem treba da odredi funkciju f takvu da važi

$$y \approx f(x)$$

Pri uspešno rešenom problemu nadgledanog učenja, funkcija f davaće tačna ili približno tačna rešenja i za podatke koji do sada nisu viđeni. Ulazne vrednosti nazivaju se atributima (eng. features) a izlazne ciljnim promenljivima (eng. target variables). Ovim opisom nije određema dimenzionalnost ni za ulazne ni za izlazne promenljive, iako je dimenzija izlazne promenljive uglavnom 1. Funkcija f naziva se modelom.

Skup svih mogućih funkcija odgovarajuće dimenzionalnosti bio bi previše veliki za pretragu i zbog toga se uvode pretpostavke o samom modelu. Pretpostavlja da je definisan skup svih dopustivih modela i da je potrebno naći najpogodniji element tog skupa. Najčešće je taj skup određen parametrima, tj. uzima se da funkcija zavisi od nekog parametra w koji je u opštem slušaju višedimenzioni i tada se funkcija označava sa $f_w(x)$.

Neophodno je uvesti funkciju greške modela (eng. loss function), odnosno funkciju koja opisuje koliko dati model dobro određuje izlaz za dati ulaz. Ova funkcija se najčešće označava sa L i $L(y, f_w(x))$ predstavlja razliku između željene i dobijene vrednosti za pojedinačni par promenljivih. No, nijedan par vrednosti promenljivih nije dovoljan za opis kvaliteta modela već treba naći funkciju koja globalno ocenjuje odstupanje modela od stvarnih vrednosti. U praksi, podrazumeva se postojanje uzorka:

$$D = \{(x_i, y_i)|i = 1, ..., N\}$$

i uvodi sledeća funkcija:

$$E(w, D) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_w(x_i))$$

koja se naziva prosečnom greškom. Uobičajeno, algoritmi nadgledanog mašinskog učenja zasnivaju se na minimizaciji prosečne greške.

Postoje dva osnovna tipa zadataka nadgledanog mašinskog učenja:

- Klasifikacija
- Regresija

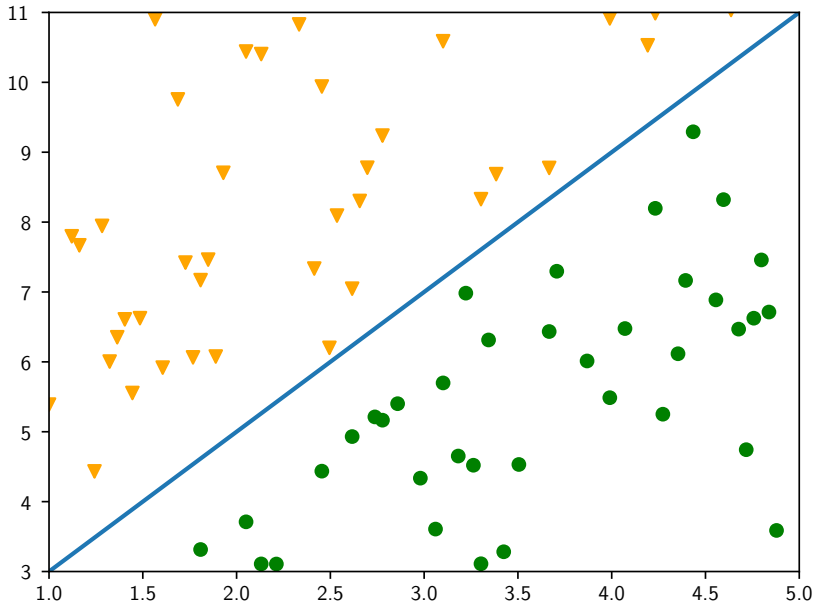
Klasifikacija (eng. *classification*) predstavlja zadatak mašinskog učenja gde je cilj predvideti klasu u kojoj se ciljna promenljiva nalazi. Neki od primera klasifikacije su svrstavanje slika na one koje sadrže ili ne sadrže lice, označavanje nepoželjne (eng. *spam*) elektronske pošte i prepoznavanje objekata na slikama. Jednostavan primer klasifikacije može se videti na slici 2.1, gde su trouglovima označeni podaci iznad prave $y = 2x + 1$ a krugovima podaci ispod date prave. Dakle, ta prava je klasifikacioni model sa parametrima $(w_0, w_1) = (1, 2)$.

Regresija je zadatak predviđanja neprekidne ciljne promenljive. Na primer, cene nekretnina mogu se predvideti na osnovu površine, lokacije, populacije koja živi u komšiluku, itd. Često korišćena vrsta regresije jeste linearna regresija. U slučaju linearne regresije, podrazumeva se da je funkcija $f_w(x)$ linearna u odnosu na parametar w. Iako se ovo na prvi pogled čini kao prilično jako ograničenje, to nije nužno slučaj; kako za atribute ne postoji zahtev za linearnosti, oni pre pravljenja linearne kombinacije mogu biti proizvoljno transformisani. Primer linearne regresije jeste aproksimacija polinomom:

$$f_w(x) = w_0 + \sum_{i=1}^{N} w_i x^i$$

2.1.2 Nenadgledano mašinsko učenje

Nenadgledano učenje obuhvata skup problema (i njihovih rešenja) u kojima sistem prihvata ulazne podatke bez izlaznih. Ovo znači da sistem sam mora da zaključi kakve



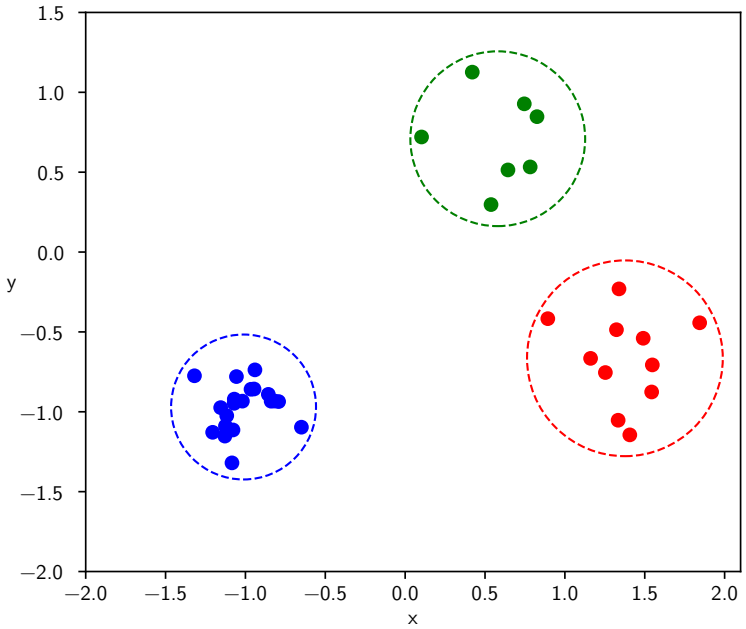
Slika 2.1: Binarna klasifikacija tačaka u skladu sa položajem u odnosu na pravu $2x + 1$

zakonitosti važe u podacima. Kako nije moguće odrediti preciznost sistema, cilj je naći najbolji model u odnosu na neki kriterijum koji je unapred zadat. Jedan primer nenadgledanog mašinskog učenja je klasterovanje: sistem grupiše neoznačene podatke u odnosu na neki kriterijum. Svaka grupa (klaster) sastoji se iz podataka koji su međusobno slični i različiti od elemenata preostalih grupa u odnosu na taj kriterijum. Cilj algoritma je određivanja tog kriterijuma grupisanja. Jednostavan primer klasterovanja po numeričkim atributima x i y može se videti na slici 2.2.

2.2 Dizajn sistema za mašinsko učenje

Okvirno, koraci u rešavanju problema su sledeći [12]:

- Prepoznavanje problema mašinskog učenja (nadgledano učenje, nenadgledano učenje, učenje potkrepljivanjem);
- Prikupljanje i obrada podataka, zajedno sa odabirom atributa;
- Odabir skupa dopustivih modela;
- Odabir algoritma učenja; moguće je odabrati postojeći algoritam ili razviti neki novi koji bolje odgovara problemu
- Izbor mere kvaliteta učenja;
- Obuka, evaluacija i, ukoliko je neophodno, ponavljanje nekog od prethodnih koraka radi unapređenja naučenog modela



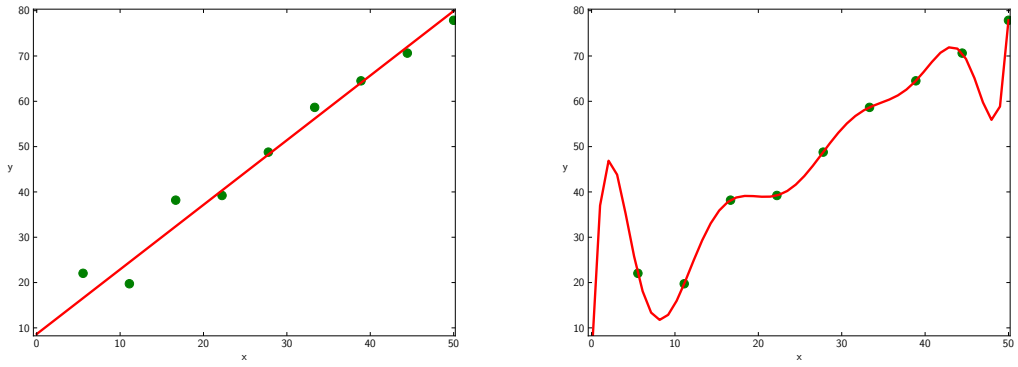
Slika 2.2: Klasterovanje

Prilikom odabira modela treba imati na umu vrstu problema koja se rešava, količinu podataka, zakonitosti koje važe u podacima, itd.

2.2.1 Podaci

Mašinsko učenje bavi se generalizacijom nad nepoznatim objektima na osnovu već viđenih objekata. Pod pojmom objekta misli se na pojedinačni podatak koji sistem vidi. Koriste se još i izrazi primerak i instanca. Vrednosti podataka pripadaju nekom unapred zadatom skupu. Podaci mogu biti različitog tipa: numerički ili kategorički. Skupovi koji određuju vrednosti kojima se instance određuju nisu unapred zadati i neophodno ih je odrediti na način pogodan za rešavanje konkretnog problema. Na primer, ukoliko je neophodno razvrstati slike životinja i biljaka na te dve kategorije, informacija o količini zelene boje na slici može biti prilično korisna, dok pri razvrstavanju vrste biljaka u zavisnosti od lista ovaj podatak skoro nije upotrebljiv (ali podatak o nijansi zelene boje može biti). Dakle, dobar izbor atributa imaće veliki uticaj na kasnije korake učenja. Podaci se sistemu daju kao vektori atributa. Nije uvek neophodno vršiti ekstrakciju atributa. Na primer, neuronske mreže u stanju su da uče nad sirovim podacima.

Podaci se neretko pre slanja sistemu obrađuju na neki način; ovaj postupak zove se pretprocesiranje. Postoje mnogi razlozi za pretprocesiranje a glavni cilj jeste da se dobiju objekti nad kojima učenje može da se vrši, imajući u vidu zahteve algoritama učenja. Međutim, i to zavisi od problema. Nekada će nepotpuni objekti, podaci koji ne sadrže sve informacije neophodne za učenje, biti izbačeni iz skupa podataka koji se razmatra, a u nekom drugom slučaju, i oni će biti korišćeni. Primeri pretprocesiranja su pretvaranje slike koja je u boji u crno beli zapis, normalizacija, umetanje nedostajućih vrednosti, itd.



Slika 2.3: Primer odabira modela pri linearnoj regresiji polinomom

2.2.2 Evaluacija modela

Nakon obučavanja (treniranja), neophodno je izvršiti evaluaciju dobijenog modela. Na koji god način se ovo izvršava, podaci korišćeni za obučavanje ne smeju se koristiti za evaluaciju modela. Često se pribegava podeli podataka na skupove za obučavanje i za testiranje. Skup za obučavanje obično iznosi dve trećine skupa ukupnih podataka. No, kako različite podele skupa mogu izazvati dobijanje različitih modela, slučajno deljenje nije najbolji izbor. Često korišćena tehnika jeste unakrsna validacija. Ovaj pristup podrazumeva podelu skupa podataka D na K podskupova približno jednake veličine, S_i za $i = 1, ..., K$. Tada se za svako i model trenira na skupu $D \setminus S_i$ a evaluacija se vrši pomoću podataka iz S_i. Posle izvedenog postupka za sve i, kao konačna ocena uzima se prosečna ocena svakog od K treniranja i evaluacija modela. Za vrednosti K uobičajeno se uzimaju vrednosti 5 ili 10. Ovaj metod vodi pouzdanijoj oceni kvaliteta modela.

2.3 Problemi pri mašinskom učenju

Kao što je podrazumevano pri pomenu pojma generalizacije, nije dovoljno odrediti funkciju koja dobro određuje izlazne vrednosti na osnovu promenljivih nad kojima se uči već je poželjno i novim ulaznim podacima dodeliti tačnu izlaznu vrednost. Odavde se može videti da je primer lošeg sistema za mašinsko učenje onaj sistem koji će izuzetno dobro naučiti da preslikava ulazne vrednosti iz skupa za učenje u odgovarajuće izlazne vrednosti ali u situaciji kada se iz tog skupa izađe neće davati zadovoljavajuće rezultate. Ovaj problem ima svoje ime: preprilagođavanje. Postoji i problem potprilagođavanja, koji podrazumeva da se sistem nije dovoljno prilagodio podacima. I preprilagođavanje i potprilagođavanje predstavljaju veliki problem ukoliko do njih dođe. Primer preprilagođavanja može se videti na slici 2.3, koja prikazuje razliku između dva modela iz skupa dopustivih modela za linearnu regresiju polinomom nad 10 različitih tačaka. Na levom delu slike prikazan je polinom reda 1 (prava) a na desnom delu prikazan je polinom reda 10. Iako će polinom reda 10 savršeno opisivati 10 tačaka sa slike, prava će verovatno bolje generalizovati nad novim podacima.

Na još jedan od mogućih problema nailazi se u slučaju neprikladnih podataka. Nekada

ulazni atributi ne daju dovoljno informacija o izlaznim. Takođe, moguće je da podataka jednostavno nema dovoljno. U ovom slučaju, sistem ne dobija dovoljno bogat skup informacija kako bi uspešno izvršio generalizaciju. S druge strane, moguće je da postoji prevelika količina podataka. Tada se pribegava pažljivom odabiru podataka koji se koriste za učenje ali ovo u opštem slučaju treba izbegavati jer su podaci izuzetno vredan element procesa mašinskog učenja. Još jedan problem vezan za podatke može biti njihova nepotpunost. Na primer, moguće je da u nekim instancama postoje nedostajuće vrednosti atributa.

Kako je najčešče potrebno pretprocesirati podatke u sklopu procesa mašinskog učenja, moguće je da u ovom postupku dođe do greške. Primera radi, prilikom rada sa konvolutivnim neuronskim mrežama, o kojima će biti reči u jednoj od narednih glava, nekada se slike u boji pretvaraju u crno-bele. Ako se primeni transformacija koja onemogućuje razlikovanje objekata koji su različiti u početnoj slici a razlikovanje je neophodno za ispravno učenje, tada proces treniranja neće teći kako je planirano.

Problem može da nastane i ukoliko nije odabran pravi algoritam za učenje, ukoliko se loše pristupilo procesu optimizacije, prilikom lošeg procesa evaluacije i, naravno, prilikom loše implementacije algoritma. Sve ove prepreke često je moguće prevazići ali je jasno da je neophodno biti izuzetno pažljiv prilikom celog procesa mašinskog učenja.

Glava 3

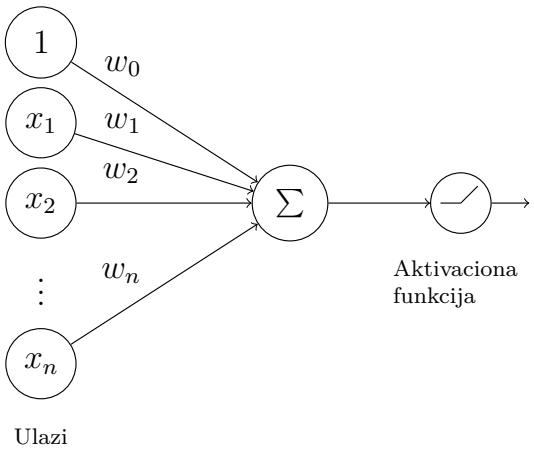
Neuronske mreže

Neuronske mreže (eng. *neural networks*) predstavljaju danas izuzetno popularan vid mašinskog učenja. Ovi modeli izuzetno su fleksibilni i imaju široku primenu. Koriste se za prepoznavanje govora, prevođenje, prepoznavanje oblika na slikama, upravljanje vozilima, uspostavljanje dijagnoza u medicini, igranje igara itd. Pun naziv je veštačka neuronska mreža (eng. *artificial neural network*, skr. *ANN*) jer se ovakvi modeli idejno zasnivaju na načinu na koji mozak funkcioniše. Ipak, neuronske mreže ne predstavljaju vernu kopiju mozga. Osnovne gradivne jedinice, neuroni, zasnovani su na neuronima u mozgu, dok veze između njih predstavljaju sinapse.[1] Te veze opisuju odnose između neurona i obično im se dodeljuje numerička težina.

Postoji nekoliko različitih vrsta neuronskih mreža. Tipičan primer jesu neuronske mreže sa propagacijom unapred. Ime proističe iz činjenice da se podaci obrađuju krećući se od ulaza mreže do izlaza, bez postojanja ikakve povratne sprege. Neuronske mreže sa propagacijom unapred sastoje se iz slojeva neurona. Ukoliko se u ovaj model uvede neki tip povratne sprege, tada se govori o rekurentnim neuronskim mrežama. Pri radu sa slikama i raznim drugim vrstama signala, najčešće se koriste konvolutivne neuronske mreže, o kojima će biti reči kasnije. Ono što je zajedničko je da su neuronske mreže sposobne za izdvajanje određenih karakteristika u podacima koji se obrađuju. To znači da se vrši kreiranje novih atributa na osnovu već postojećih atributa ili direktno iz ulaznih podataka. Taj proces naziva se ekstrakcijom atributa i smatra se da je to jedan od najbitnijih razloga za delotvornost neuronskih mreža.

Za uspeh neuronskih mreža zaslužna je njihova fleksibilnost, ali se rezultati najčešće dobijaju eksperimentisanjem. Naime, veliki deo zaključaka o ponašanju neuronskih mreža u raznim situacijama nije teorijski potkrepljen. Stoga, istraživački rad vezan za neuronske mreže zahteva dosta pokušaja da bi se došlo do uspeha.

[1] Sinapsa je biološka struktura koja omogućuje komunikaciju između neurona.



Slika 3.1: Neuron

3.1 Neuronske mreže sa propagacijom unapred

Neuronske mreže sa propagacijom unapred jedna su od najkorišćenijih vrsta neuronskih mreža. Gradivni elementi ovakvog modela, neuroni (koji se još nazivaju i jedinicama), organizuju se u slojeve koji se nadovezuju i time čine neuronsku mrežu. Organizacija neurona i slojeva, uključujući i veze između neurona, predstavlja arhitekturu mreže. Prvi sloj mreže naziva se ulaznim slojem dok se poslednji sloj naziva izlaznim slojem. Neuroni prvog sloja kao argumente primaju ulaze mreže dok neuroni svakog od preostalih slojeva kao svoje ulaze prihvataju izlaze prethodnog sloja. Svi slojevi koji svoje izlaze prosleđuju narednom sloju nazivaju se skrivenim slojevima. Mreže sa više od jednog skrivenog sloja nazivaju se dubokim neuronskim mrežama. Broj slojeva mreže određuje njenu dubinu. Termin duboko učenje nastao je baš iz ove terminologije.

Svaki neuron opisuje se pomoću vektora $w = (w_0, ..., w_n)$ koji se naziva vektorom težina. Ulazni vektor $x = (x_1, ..., x_n)$ linearno se transformiše na sledeći način:

$$w_0 + \sum_{i=1}^{n} x_n w_n \tag{3.1}$$

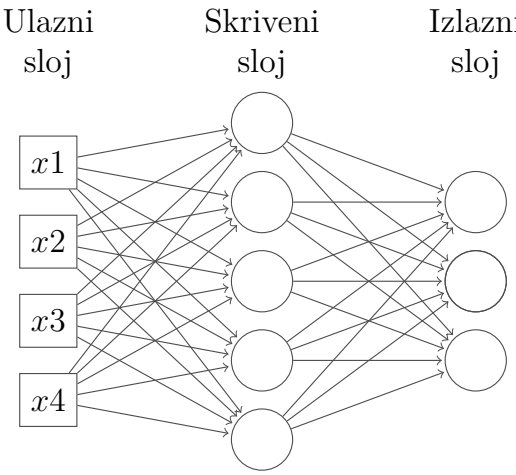
a zatim se primenjuje takozvana aktivaciona funkcija, g. Izlaz iz neurona je

$$g(w_0 + \sum_{i=1}^{n} x_n w_n)$$

i, uprkos linearnosti prve transformacije, izlaz ne mora biti linearna transformacija ulaza, tj. g najčešće nije linearna funkcija. Za g se bira nelinearna funkcija jer se u suprotnom kao celokupna transformacija koju neuron vrši dobija linearna funkcija; na ovaj način, mreža bi predstavljala linearnu funkciju i ne bi bilo moguće njom aproksimirati nelinearne funkcije dovoljno dobro. Vrednost w_0 naziva se slobodnim članom. Nekada se vektor x proširuje tako da bude oblika $x = (1, x_1, ..., x_n)$ kako bi izraz (3.1) imao kraći zapis $f_w(x) = w \cdot x$, gde $\cdot$ označava skalarni proizvod.

Model, tj. neuronska mreža, formalno se definiše na sledeći način:

$$\begin{gathered} h_0 = x \\ h_i = g_i(W_i h_{i-1} + w_{i0}), \text{ za } i = 1, ..., L \end{gathered} \tag{3.2}$$



Slika 3.2: Neuronska mreža koja sadrži jedan skriveni sloj

gde je x vektor ulaza u mrežu predstavljen kao kolona, W_i je matrica čija j-ta vrsta predstavlja vektor težina j-tog neurona u sloju i a w_{i0} je kolona slobodnih članova svih jedinica u sloju i. Funkcije g_i su nelinearne aktivacione funkcije i za vektor $t = (t_1, ..., t_n)$, $g_i(t)$ predstavlja kolonu $(g_i(t_1), ..., g_i(t_n))^T$. Na ovaj način dobija se funkcija čiji su parametri W_i i w_{i0} za $i = 1, ..., L$. Ako se parametri označe sa w, tada se model zapisati kao f_w. Parametri w mogu se pronaći matematičkom optimizacijom nekog kriterijuma kvaliteta modela. Taj proces opisan je u delu 3.1.2.

3.1.1 Aktivacione funkcije

Preteča neuronskih mreža, perceptron, je model koji se sastoji samo iz jednog neurona čija je aktivaciona funkcija data sledećim izrazom:

$$g(x) = \begin{cases} 1, & \text{ako } x \geq 0 \\ 0, & \text{inače} \end{cases}$$

Definicija aktivacione funkcije perceptrona znači da njegova primena ima relativno jako ograničenje. Izvod ove funkcije je 0 u svim tačkama sem u $x = 0$, gde izvod ne postoji. To znači da ovakva funkcija nije pogodna za upotrebu uz optimizaciju metodom gradijentnog spusta, koji se oslanja na korišćenje izvoda funkcije.

Dakle, neophodno je naći druge funkcije koje služe kao aktivacione funkcije. Poželjna svojstva aktivacione funkcije su:

- Nelinearnost: Kao što je objašnjeno ranije, kompozicija linearnih funkcija daje linearnu funkciju, što onemogućuje dovoljno preciznu aproksimaciju nelinearnih funkcija;
- Diferencijabilnost: Optimizacija se najčešće vrši metodima koji koriste gradijent funkcije;

- Monotonost: Ako aktivaciona funkcija nije monotona, povećavanje nekog od težinskih parametara neurona, umesto da poveća izlaz i time proizvede jači signal, može imati suprotan efekat;
- Ograničenost: Ukoliko vrednosti unutar neuronske mreže nisu ograničene, moguće je da dođe do pojavljivanja ogromnih vrednosti koje potencijalno dovode do prekoračenja. Ograničene aktivacione funkcije znatno ublažuju ovaj problem.

Dozvoljeno je da aktivaciona funkcija ne poseduje neko od navedenih svojstava.

Najčešće korišćene aktivacione funkcije su:

- Sigmoidna funkcija: $\sigma(x) = \dfrac{1}{1+e^{-x}}$
- Tangens hiperbolički: $tanh(x) = \dfrac{e^{2x}-1}{e^{2x}+1}$
- Ispravljena linearna jedinica: $ReLU(x) = max(0, x)$

Sigmoidna funkcija bila je najkorišćenija aktivaciona funkcija pri radu sa neuronskim mrežama. Ograničena je (sve slike nalaze se u intervalu $(-1, 1)$), monotona i diferencijabilna u svakoj tački skupa $\mathbb{R}$. Međutim, što se argument više udaljava od nule, to nagib funkcije postaje manji. To znači da će gradijent funkcije biti mali i da će učenje teći jako sporo.

Tangens hiperbolički srodan je sigmoidnoj funkciji ($tanh(x) = 2\sigma(x) - 1$), ali je imao veći uspeh od sigmoidne funkcije. U okolini nule, ova funkcija slična je identičkoj, što olakšava učenje. Međutim, i pri korišćenju ove funkcije može se naići na problem sa malim gradijentima ukoliko se argument dovoljno udalji od nule.

Uprkos tome što za razliku od prethodne dve funkcije nije ni ograničena ni diferencijabilna u svim tačkama domena, danas je ispravljena linearna jedinica najpopularniji izbor za aktivacionu funkciju. Funkcija je jednaka identitetu desno od nule i stoga se gradijent ne menja. Takođe, verovatnoća da se traži gradijent u tački u kojoj funkcija nije diferencijabilna je mala. Ipak, ni ova funkcija nije bez mana; problem često pravi deo levo od nule, gde je funkcija konstantna. To znači da je gradijent nula i da se prilikom optimizacije težine neurona neće izmeniti. Zbog nedostatka promene, može se desiti da neki neuroni u mreži postanu pasivni, tj. da im izlaz postane 0. Za ovaj problem postoje rešenja; jedno jeste da izlaz funkcije desno od nule ne bude konstanta 0 već αx, za neko malo α. Ta modifikovana ReLU funkcija naziva se nakošena ispravljena linearna jedinica (eng. *leaky rectified linear unit*).

Iako sve ove funkcije imaju prednosti i mane u odnosu na preostale, ne postoji jedinstveni izbor nego je na osnovu problema neophodno zaključiti koju je aktivacionu funkciju najbolje koristiti.

Izlazni sloj

Neuronske mreže koriste se pri regresiji, određivanju funkcije koja opisuje vezu izmedju ulaza i izlaza, i klasifikaciji, svrstavanje ulaznih vektora u jednu od konačnog broja kategorija. Pri regresiji, u poslednjem sloju ne primenjuje se aktivaciona funkcija. Proces optimizacije svodi se na minimizaciju funkcije greške. Kod rešavanja problema klasifikacije (u N kategorija), koristi se funkcija mekog maksimuma (eng. *softmax*):

$$softmax(x) = \left(\frac{e^{x_1}}{\sum_{i=1}^{N} e^{x_i}}, \ \dots \ , \frac{e^{x_N}}{\sum_{i=1}^{N} e^{x_i}} \right)$$

Suma ovako dobijenog vektora je 1 i stoga može predstavljati diskretnu raspodelu verovatnoća. Za vrednost aproksimacije uzima se kategorija kojoj odgovara najviša vrednost izlaznog vektora. Za optimizaciju pri radu sa probabilističkim problemima, kao što je problem klasifikacije, primenjuje se metod maksimalne verodostojnosti (eng. *maximum likelihood estimate*), odnosno traži se maksimum sledećeg izraza po parametru w:

$$P_w(y_1, ..., y_N | x_1, ..., x_N)$$

3.1.2 Optimizacija

Ukoliko je neuronska mreža predstavljena kao funkcija f_w, gde su w parametri mreže, neophodno je izvršiti minimizaciju[2] funkcije koja predstavlja kriterijum kvaliteta aproksimacije. Problem optimizacije u slučaju neuronskih mreža težak je zbog nekonveksnosti. Ona čini neke metode teško primenljivim ili izuzetno sporim. Moguće je i završiti u lokalnom optimumu funkcije. Uobičajeno se koriste metodi zasnovani na gradijentu funkcije. Postoje metodi drugog reda, zasnovani na hesijanu,[3] ali je njegovo računanje u slučaju većeg broja parametara preskupo.

Učenje funkcioniše na sledeći način za fiksirane ulaze x posmatra se njima uparen izlaz y i $f_w(x)$ a zatim i $L(y, f_w(x))$, odnosno funkcija greške između stvarne i očekivane vrednosti. Koristeći algoritam propagacije unazad (3.1.2) uz neki od algoritama za optimizaciju, vrši se minimzacija funkcije L u odnosu na parametre mreže, w.

Metod gradijentnog spusta i stohastičkog gradijentnog spusta

Gradijent funkcije $f : \mathbb{R}^n \to \mathbb{R}$ u tački $x = (x_1, ..., x_n)$ označava se sa ∇f i predstavlja vektor parcijalnih izvoda u toj tački:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), ..., \frac{\partial f}{\partial x_n}(x) \right)$$

[2]Naravno, optimizacioni metodi primenjuju se i na maksimizaciju, ali je u slučaju mašinskog učenja najčešće neophodno minizovati funkciju greške.

[3]Hesijan je matrica parcijalnih izvoda drugog reda.

Gradijent funkcije u tački x predstavlja pravac i smer najbržeg rasta funkcije pa $-\nabla f(x)$ predstavlja pravac smer najbržeg opadanja funkcije. Kako se najčešće minimizuje funkcija greške, u oznaci L, na dalje je korišćeno to ime za funkciju umesto f.

Metod gradijentnog spusta jedan je od najstarijih metoda optimizacije. Iterativnim pristupom minimizuje se konveksna diferencijabilna funkcija. Polazeći od nasumično odabrane tačke i prateći pravac i smer gradijenta u svakom koraku, dolazi se do minimuma funkcije. Iterativni korak definisan je na sledeći način:

$$w_{k+1} = w_k - \alpha_k \nabla L(w_k),\ k = 0, 1, 2, ..., \tag{3.3}$$

gde je w_0 ta nasumično odabrana početna tačka a α_k je pozitivan realan broj koji se naziva veličinom koraka ili stopom učenja (eng. *learning rate*). Za funkciju greške u ovom slučaju uzima se srednjekvadratno odstupanje:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - f_w(x_i))^2$$

Bitno je pažljivo odabrati veličinu koraka jer ova vrednost može uticati na konvergenciju. Jedan primer odabira veličine koraka jeste niz za koji važe Robins Monroovi uslovi [10]:

$$\sum_{k=0}^{\infty}\alpha_k = \infty \qquad\qquad \sum_{k=0}^{\infty}\alpha_k^2 < \infty$$

Jednostavniji pristup bio bi da se odabere mali pozitivan parametar α i da za svako k važi $\alpha_k = \alpha$.

Postavlja se i pitanje koliko koraka načiniti pre zaustavljanja. U praksi se koristi nekoliko kriterijuma kao što su zaustavljanje kada su dve uzastopne vrednosti w_k i w_{k+1} dovoljno bliske, kada su vrednosti funkcije za dve uzastopne vrednosti dovoljno bliske, ali se može zaustaviti i nakon unapred određenog broja koraka. Postoji još kriterijuma i moguće ih je kombinovati.

Iako jednostavan i široko primenljiv metod optimizacije, gradijentni spust nije najbolji izbor. Naime, pravac najbržeg uspona funkcije nije uvek i pravac koji osigurava najbrže približavanje optimumu funkcije. U praksi, gradijentni spust ume da proizvodi cik-cak kretanje koje dovodi do spore konvergencije. Takođe, za jedan iterativni korak neophodno je proći kroz sve parove ulaza i izlaza, što u slučaju velikog skupa podataka za obučavanje može biti jako velika količina podataka.

Za obučavanje neuronskih mreža češće se koristi metod stohastičkog gradijentnog spusta. Pretpostavka je da je funkcija koja se optimizuje oblika:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(w)$$

odnosno da se može predstaviti kao prosek nekih N funkcija. Kako je neuronska mreža jedan od metoda mašinskog učenja, na raspolaganju je skup za obučavanje pa se funkcija

greške na celom skupu može predstaviti kao prosek grešaka na pojedinačnim instancama skupa. Novi oblik jednakosti (3.3) je:

$$w_{k+1} = w_k - \alpha_k \left(\frac{1}{N} \sum_{i=1}^{N} \nabla L_i(w_k) \right), \; k = 0, 1, 2, ...$$

Pri korišćenju stohastičkog gradijentnog spusta za minimizaciju funkcije greške, iterativni korak izgleda ovako:

$$w_{k+1} = w_k - \alpha \nabla L_i(w_k)$$

Postoje razni načini za odabir vrednosti i u nekom koraku, kao što je $i = k(mod\ N) + 1$, gde je N veličina skupa za obučavanje. Još jedan primer je nasumični odabir instance u svakom koraku. Kakav god način izbora bio, neophodno je iskoristiti sve greške. Moguće je proći greške iz skupa za obučavanje i nekoliko puta dok se ne postigne željeni nivo aproksimacije.

Kako ova aproksimacija može biti prilično neprecizna, pribegava se kompromisu: prilikom iterativnog koraka ne koriste se pojedinačne instance već neki podskup skupa za obučavanje (eng. *minibatch*) i umesto greške na pojedinačnoj instanci koristi se prosek grešaka na odabranom podskupu. Pri treniranju neuronskih mreža, ovo je uobičajeni pristup.

Metod stohastičkog gradijentnog spusta manje je računski zahtevan od gradijentnog spusta, ali je manje precizan i neophodan je veći broj iteracija kako bi se dostigao minimum.

Postoje razni metodi optimizacije koji se koriste pri mašinskom učenju. Neki menjaju veličinu koraka u zavisnosti od prethodnih izračunatih koraka i gradijenata. Takvi metodi nazivaju se adaptivnim metodima optimizacije. Primer adaptivnih metoda optimizacije su Adam i RMSProp.

RMSProp

Algoritam RMSProp (eng. *root mean square propagation*) predložio je Džefri Hinton (eng. *Geoffrey Hinton*) na jednom od svojih predavanja na sajtu Kursera [7]. Ovo je algoritam optimizacije korišćen prilikom razvijanja DQN algoritma. Glavna ideja je čuvanje dosadašnjeg otežanog proseka kvadrata gradijenta funkcije koji će biti obeležen sa g_k. Simbol $\odot$ obeležava pokoordinatno množenje dva vektora. Kako algoritam nije objavljen u radu, može se naći veliki broj različitih implementacija. U nastavku je predstavljen algoritam u skladu sa implementacijom iz biblioteke Keras, koja je korišćena za implementaciju DQN algoritma u ovom radu.

$$g_0 = 0$$
$$\alpha_0 = \alpha$$
$$g_{k+1} = \gamma g_k + (1 - \gamma) \nabla L(w_k) \odot \nabla L(w_k)$$
$$\alpha_{k+1} = \frac{\alpha_k}{1 + d(k+1)}$$

Tada se iterativni korak definiše:

$$w_{k+1} = w_k - \frac{\alpha_{k+1}}{\sqrt{g_{k+1}} + \varepsilon} \nabla L(w_k)$$

Sve operacije vrše se pokoordinatno. Parametar γ pripada poluotvorenom intervalu $[0, 1)$. U svom predavanju, Hinton predlaže da njegova vrednost bude 0.9. Preporučena vrednost za veličinu koraka odnosno stopu učenja, u oznaci α, je 0.001 dok d označava faktor opadanja za parametar α. Parametar ε služi da bi se izbeglo deljenje nulom i obično je reda veličine 10^{-8}.

Adam

Adam (eng. *adaptive moment estimation*) jedan je od najčešćih algoritama za optimizaciju korišćen pri obučavanju neuronskih mreža. Algoritam Adam zasnovan je na korišćenju ocena prvog i drugog momenta gradijenta, datim sledećim formulama:

$$m_0 = 0$$
$$v_0 = 0$$
$$m_{k+1} = \beta_1 m_k + (1 - \beta_1)\nabla L(w_k)$$
$$v_{k+1} = \beta_2 v_k + (1 - \beta_2)\nabla L(w_k) \odot \nabla L(w_k)$$

Ocena prvog momenta, m_0, predstavlja otežani prosek pravca kretanja dok ocena drugog momenta, v_0, predstavlja otežani prosek kvadrata norme gradijenata. Međutim, ove dve ocene su pristrasne ka početnoj vrednosti, u ovom slučaju 0.[4] Da bi se to ispravilo, vrši se sledeća korekcija:

$$\hat{m}_{k+1} = \frac{m_{k+1}}{1 - \beta_1^{k+1}}$$
$$\hat{v}_{k+1} = \frac{v_{k+1}}{1 - \beta_2^{k+1}}$$

Na kraju, iterativni korak dat je ispod. Dodavanje skalara ε na vektor $\hat{v}_{k+1}$ predstavlja dodavanje tog skalara svakom članu datog vektora. Korenovanje, deljenje i oduzimanje vrše se pokoordinatno.

$$w_{k+1} = w_k - \alpha \frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_{k+1}} + \varepsilon}$$

Parametar α naziva se veličina koraka ili stopa učenja. Vrednosti parametara β_1 i β_2 ograničene su na skup $[0, 1)$ i preporučene vrednosti su 0.9 i 0.999, redom, dok se za ε preporučuje vrednost 10^{-8}. Kao i u algoritmu RMSProp, svrha parametra ε je izbegavanje deljenja nulom. Takođe nalik algoritmu RMSProp opisanom iznad, moguće je uvesti stopu opadanja parametra α.

Intuicija kojom se vodi algoritam Adam jeste da dužina svakog koraka zavisi od osobina funkcije u regionu u kom se trenutno vrši optimizacija. Ovaj algoritam u mnogim primenama pokazao se kao superioran u odnosu na ostale algoritme za optimizaciju.

[4] Ovde se misli na 0 vektor istih dimenzija kao x_k u slučaju prvog momenta i skalar 0 u slučaju drugog momenta

Metod propagacije unazad

Do sada je objašnjeno kako iskoristiti gradijent funkcije radi nalaženja odgovarajućih parametara funkcije. Međutim, računanje gradijenta u slučaju neuronskih mreža izuzetno je zahtevan proces. U ovu svrhu koristi se metod propagacije unazad (eng. *back propagation*). Prilikom obučavanja mreže, cilj je minimizovati funkciju greške, $L(w)$, gde w predstavlja parametre mreže. Ulazi i izlazi mreže ne nalaze se u ovom zapisu jer su u konkretnom slučaju fiksirani i neophodno je izvršiti izmenu parametara mreže tako da funkcija greške bude što manja.

Iz jednakosti (3.2) jasno je da neuronska mreža predstavlja složenu funkciju a, kako je funkcija L mera odstupanja vrednosti koje mreža predviđa i željenih vrednosti, tada je i L složena funkcija. Izvod složene funkcije $g \circ h$ računa se na sledeći način:

$$(g \circ h)' = (g' \circ h)h'$$

Ovo pravilo može se primeniti i kada kompoziciju čine više od dve funkcije.

Kako neuronske mreže skoro uvek sadrže više od jednog parametra, koristi se pravilo za računanje parcijalnog izvoda složene funkcije više promenljivih. Neka su date funkcije $h : \mathbb{R}^m \to \mathbb{R}^k$ i $g : \mathbb{R}^k \to \mathbb{R}$. Tada se parcijalni izvod funkcije $g \circ h$ po i-toj promenljivoj računa koristeći sledeće pravilo:

$$\partial_i(g \circ h) = \sum_{j=1}^{k} (\partial_j g \circ h)\partial_i h_j$$

gde h_j označava j-tu komponentu funkcije h a $\partial_j g$ parcijalni izvod funkcije g po j-tom argumentu. Metod propagacije unazad koristi ovo pravilo za izračunavanje parcijalnih izvoda po svim parametrima mreze. Počinje se od izlaznog sloja i kreće se ka ulaznom, odakle potiče ime metoda. Čuva se do sada akumulirani izvod funkcije. Za svaki sloj računa se izvod po parametrima tekućeg sloja i dosadašnji akumulirani izvod proširuje se izvodom tekućeg sloja. Kako jedan sloj predstavlja linearnu kombinaciju izlaza prethodnog sloja i parametara mreže na koju je primenjena aktivaciona funkcija, neophodno je računati izvode i aktivacione funkcije i te linearne kombinacije.

Sada su dati svi alati za optimizaciju neuronske mreže sa propagacijom unapred. Treba imati u vidu da proces obučavanja velikih neuronskih mreža može biti izuzetno skup. Takođe, na sam proces učenja mogu uticati razni faktori kao što su arhitektura mreže, podela podataka na skupove za obučavanje i testiranje ili parametri algoritma za optimizaciju. Ovi faktori nazivaju se metaparametrima (eng. *hyperparameter*) i neretko je neophodno isprobati razne njihove kombinacije dok ponašanje mreže ne dostigne željeni nivo. Često se umesto traženja metaparametara pribegava korišćenju unapred ispitanih vrednosti za koje je već pokazano da daju željene rezultate pri rešavanju nekog problema.

3.1.3 Prednosti i mane

Neuronske mreže pokazale su se kao jako korisne za rešavanje praktičnih problema zbog svoje izuzetne fleksibilnosti. Međutim, za proces obučavanja neuronske mreže neophodno je imati veliku količinu podataka. Proces učenja može biti izuzetno spor, posebno

ukoliko se uvede isprobavanje raznih vrednosti metaparametara. Velika fleksibilnost može izazvati i preprilagođavanje podacima i time učiniti performanse mreže nad novim podacima lošim. Postoje i problemi pri optimizaciji kao što su takozvani problemi nestajućih i eksplodirajućih gradijenata. Iako su u stanju da konstruišu nove atribute na osnovu starih, struktura obučene mreže nije čitljiva za ljude. U nekim situacijama, ovo može izazvati probleme. Na primer, ukoliko klijent podnese zahtev za kredit i neuronska mreža odluči da nije podoban, nije moguće objasniti razlog odbijanja. Neuronske mreže takođe su dosta računski i razvojno zahtevne. Nekada će neki već poznat algoritam dati zadovoljavajuće rešenje dok razvoj neuronske mreže može biti skup i po pitanju vremena razvijanja sistema i po pitanju kasnijeg rada sistema.

Pri radu sa neuronskim mrežama pojavljuje se još jedan problem, takozvano katastrofalno zaboravljanje, pojava koja podrazumeva da neuronska mreža nekada zaboravlja već naučeno. Kada se uči nad novim podacima, parametri mreže se menjaju kako bi odgovarali traženom izlazu i u ovom proces može se dogoditi da se se parametri dovoljno izmene da ne daju prihvatljive izlaze čak i za već viđene ulaze.

Kako ne postoje teorijske smernice za rad sa neuronskim mrežama, odluke vezane za razvoj sistema neophodno je donositi empirijski.

3.2 Konvolutivne neuronske mreže

Konvolutivne neuronske mreže (eng. *convolutional neural network*), kraće nazivane konvolutivne mreže, su oblik neuronskih mreža specijalizovan za učenje nad signalima, kao što su zvuk ili slike. Kao što je nagovešteno nazivom ove vrste modela, u srži konvolutivnih neuronskih mreža leži operacija pod nazivom konvolucija.

Konvolucija funkcija I i K obeležava se sa $I * K$ i data je narednim izrazom:

$$(I * K)(x) = \sum_t I(t)K(x-t) = \sum_t I(x-t)K(t)$$

Ovim izrazom definisana je jednodimenziona diskretna konvolucija. Neretko se sreće i dvodimenziona konvolucija koja se definiše na sledeći način:

$$(I * K)(x,y) = \sum_m \sum_n I(m,n)K(x-m,y-n) = \sum_m \sum_n I(x-m,y-n)K(m,n)$$

U prethodnim jednakostima nisu zapisana ograničenja za argumente jer bi to zakomplikovalo zapis, ali naravno, treba paziti na to da su I i K definisani za sve argumente. Takođe, iz datih jednakosti vidi se da je operacija komutativna. Konvolucija se može definisati i u više dimenzija.

U konvolutivnim mrežama, ulaz predstavlja sirovi ili minimalno obrađeni signal. U tom slučaju, $I(x)$ označava vrednost signala na poziciji x i analogno za dvodimenzioni signal. Nad tim signalom i nekim filterima primenjuje se konvolucija. Svrha te primene je određivanje prisustva nekih šablona tj. karakteristika u signalu. Primer tih šablona su uspravne ili vodoravne linije na slici. Na ovaj način konstruišu se novi atributi na osnovu ulaza. Konvolutivnom primenom filtera na prethodno konstruisane atribute moguće je

konstruisati i složenije atribute. Vrednosti ovih filtera biće naučene u toku obučavanja mreže. Filteri su obično dosta manjih dimenzija od slike i, ukoliko je ulazni signal obeležen sa I a filter K, tada su drugi zapisi u jednakostima datim iznad intuitivniji za shvatanje.

Rezultat primene konvolucije za sve validne x i y manjih je dimenzija od početnog signala, I. Ukoliko je signal I dimenzija $k \times l$ a filter K dimenzija $p \times q$, tada je $I * K$ dimenzija najviše dimenzija $k-p+1 \times l-q+1$. Moguće je i preskakati neke x i y prilikom računanja konvolucije, što dodatno smanjuje dimenziju rezultata. Na primer, moguće je računati konvoluciju za svako drugo x i svako drugo y. Razlika između uzastopnih pozicija naziva se pomerajem (eng. *stride*). Takodje je moguće proširiti I nekim vrednostima kako vrednosti na rubu ne bi izgubile na značaju. Često se proširivanje (eng. *padding*) vrši nulama ili vrednostima koje se nalaze na rubu.

Još jedan bitan pojam pri radu sa konvolutivnim mrežama jeste agregacija (eng. *pooling*). Primena agregacije na signal podrazumeva da se za svaku oblast određenih dimenzija izračunati neki izraz, poput proseka ili maksimuma. Svrha agregacije jeste stvaranje otpornosti na male translacije signala. Agregacijom se takođe postiže umanjenje dimenzija izlaza u odnosu na ulaz, što smanjuje memorijske zahteve za kasnije slojeve. I pri agregaciji se nekada koristi pomeraj različit od 1. Ipak, u slučaju da je neophodno sačuvati informaciju o tome gde je neki atribut pronađen, agregacija nije pogodna.

3.2.1 Svojstva konvolucije

Konvolucija ima nekoliko svojstava koji konvolutivne neuronske mreže čine pogodnijim za učenje nad signalom. Prvo takvo svojstvo su proređene interakcije (eng. em sparse interactions). Jedna jedinica sloja predstavlja jedan filter i primena tog filtera podrazumeva njegovu konvolutivnu primenu na sve moguće pozicije ulaza. Ova primena može se posmatrati kao prevlačenje filtera preko ulaza. To znači da su parametri jedne jedinice u stvari vrednosti filtera. Kako je filter obično dimenzija dosta manjih od dimenzija ulaza, ovo znači da je smanjen broj parametara jedne jedinice u odnosu na mreže sa propagacijom unapred, kod kojih je svaka jedinica povezana sa svakom jedinicom prethodnog sloja.

Kod mreža sa propagacijom unapred, jedan težinski parametar sloja koristi se pri samo jednom izračunavnaju izlaza. Kod konvolutivnih mreža, kako je već rečeno, svi parametri filtera koriste se na svim lokacijama koje je neophodno proći prilikom njegove konvolutivne primene. Dakle, može se reći da u konvolutivnim mrežama postoje deljene težine, odnosno deljeni parametri.

Još jedno bitno svojstvo konvolucije je neosetljivost na translacije. Ukoliko se ulazni signal translira pa se primeni konvolucija nekim filterom, rezultat je isti kao da je primenjena translacija na konvoluciju tim filterom.

Konvolucija je izuzetno pogodna operacija prilikom rada sa signalima. Iako je neuronska mreža sa propagacijom unapred u stanju da uči nad signalom, taj tip neuronskih mreža nije u stanju da iskoristi susednost podataka i konstruiše tip atributa kakav konvolutivna mreža konstruiše. Takođe, višedimenzioni ulazi morali bi da budu transformisani

do jednodimenzionog.

3.2.2 Slojevi konvolutivne neuronske mreže

Jedna konvolutivna jedinica primenjuje jedan filter na svoj ulaz. Konvolutivni slojevi sastoje se od više takvih jedinica tj. jedan konvolutivni sloj obično paralelno primenjuje više filtera na svoj ulaz. Na izlaz konvolutivnog sloja najčešće se primenjuje neka aktivaciona funkcija. Primera radi, slika zapisana u RGB formatu ima tri dimenzije: širina, visina i dubina, koja je određena brojem kanala. Filteri se većinom primenjuju na sve kanale slike. Najčešće se naizmenično primenjuju konvolutivni i agregacioni slojevi i obično se nakon konvolutivnih slojeva nalazi jedan ili više gusto povezanih slojeva. Kada neuronska mreža sadrži barem jedan konvolutivni sloj, tada se govori o konvolutivnoj neuronskoj mreži.

3.2.3 Mane

Navedena svojstva konvolucije opisuju njihovu prednost pri radu sa signalima, što ih čini primenljivijim na takve podatke od neuronskih mreža sa propagacijom unapred. Ipak, postoje i razne poteškoće. Iako je neosetljiva na translaciju, konvolucija nije neosetljiva na neke druge tipove transformacija kao što su rotacija i homotetija (skaliranje). Obučavanje konvolutivnih neuronskih mreža takođe zahteva veliku količinu podataka i bez određenog hardvera može biti izuzetno dugotrajno. Uz to, pri radu sa konvolutivnim mrežama pojavljuju se i problemi na koje se nailazi pri radu sa mrežama sa propagacijom unapred.

Glava 4

Učenje potkrepljivanjem

Učenje potkrepljivanjem je vid mašinskog učenja koji podseća na učenje koje se može naći u prirodi: ljudi i životinje uče na osnovu interakcije sa svetom oko sebe. Razlikovanje povoljnog i nepovoljnog ponašanja nije unapred učinjeno već je neophodno da onaj koji uči donese taj zaključak. U učenju potkrepljivanjem kao grani mašinskog učenja, komunikacija sa okruženjem svodi se na preduzimanje akcija u nekoj situaciji i dobijanja odgovora u vidu numeričke nagrade i informacije o tome kako se situacija promenila. Entitet koji komunicira sa svetom naziva se (softverskim) agentom dok se svet naziva okruženjem. Kao i kod ostalih oblasti mašinskog učenja, učenje potkrepljivanjem podrazumeva skup problema i njihovih rešenja. Cilj je baš učenje na osnovu komunikacije sa okruženjem, bez potrebe za ljudskom intervencijom.

4.1 Osnovni pojmovi

Učenje potkrepljivanjem počiva na četiri komponente: politici ponašanja (ili samo politici), nagradi, funkciji vrednosti i modelu okruženja. Politika opisuje način na koji se agent ponaša. Nagrada predstavlja numerički signal koji agent dobija od okruženja. U toku učenja, cilj agenta je da maksimizuje ukupnu nagradu dobijenu od okruženja. Dakle, nagradom je implicitno objašnjeno šta je dobro a šta loše ponašanje. Prilikom rešavanja problema učenja potkrepljivanjem, cilj je nalaženje optimalne politike, tj. politike čijim se praćenjem dobija najveća dugoročna nagrada. Funkcija vrednosti govori koliko je dobro naći se u nekom stanju okruženja. Za razliku od nagrade, funkcija vrednosti opisuje kvalitet nekog stanja na duže staze. Ova komponenta je neophodna jer je moguće da dolaskom u neko stanje agent dobije malu nagradu ali da dato stanje ima veliku vrednost, što znači da je dolaskom u to stanje moguće ostvariti veliku dugoročnu nagradu. Neki algoritmi učenja potkrepljivanjem koriste model okruženja kako bi planirali unapred. Moguće je koristiti pristupe koji koriste model, pristupe koji ne koriste model već se uči iz iskustva, kao i pristupe koji koriste učenje iz iskustva radi učenja modela.

U učenju potkrepljivanjem javlja se potreba za uspostavljanjem balansa između istraživanja i iskorišćavanja već stečenog znanja (eng. *exloration vs. exploitation*). Naime, na početku učenja, agent istražuje okruženje i time uči kako bi trebalo da se ponaša. Čak i

kada se nauči neko ponašanje, često je neophodno nastaviti sa istraživanjem u nekoj meri. Kako već naučeno ponašanje možda nije najbolje, bez istraživanja je moguće završiti u nekom od lokalnih optimuma. Međutim, kako učenje teče i agent unapređuje svoje ponašanje, poželjno je pratiti politiku koja dovodi do velikih dugoročnih nagrada. Najčešće je stopa istraživanja velika na početku učenja i opada u toku ovog procesa.

4.2 Markovljevi procesi odlučivanja

Markovljevi procesi odlučivanja (eng. *Markov decision process*, skr. *MDP*) daju teorijski okvir u kome je moguće relativno jednostavno postaviti i rešiti problem učenja potkrepljivanjem. Markovljevi procesi odlučivanja opisuju okruženje s kojim je moguće komunicirati. Ta komunikacija sastoji se iz toga da se od okruženja može dobiti informacija o stanju i da se okruženju može poslati informacija o akciji koja se preduzima, na šta okruženje odgovara informacijama o novom stanju i nagradi. Izuzetno je važno da se od okruženja ne može dobiti informacija o tome da li je preduzeta akcija prava ili ne već samo informacija o nagradi, koju na duže staze treba maksimizovati. Kako se pri učenju potkrepljivanjem pretpostavlja postojanje agenta, kao i postojanje okruženja, nadalje se podrazumeva da postoji neki agent koji komunicira sa okruženjem. Treba imati u vidu da Markovljevi procesi odlučivanja opisuju idealno okruženje. U praksi okruženja često nisu idealna i tada se pribegava metodima koji ne koriste MDP direktno. Iako je veliki deo teorije u učenju potkrepljivanjem ograničen pretpostavkom korišćenja MDP, iste ideje imaju širu primenu.

4.2.1 Osnovni pojmovi

Pod pretpostavkom da se interakcija između agenta i okruženja izvršava u diskretnim trenucima, stanje okruženja u trenutku t označava se sa S_t dok se skup svih stanja označava sa $\mathcal{S}$. Agent u stanju S_t preduzima akciju $A_t \in \mathcal{A}(S_t)$ i prelazi u novo stanje, S_{t+1} dobijajući nagradu R_{t+1}. $\mathcal{A}(s)$ označava skup dozvoljenih akcija u stanju s. Ukoliko se sa $\mathbb{A}$ označi skup svih akcija, $\mathcal{A}$ se može posmatrati kao funkcija $\mathcal{A} : \mathcal{S} \to \mathbb{P}(\mathbb{A})$.[1] Skup $\mathcal{R}$ je skup mogućih realnih nagrada. Sekvenca $S_0, A_0, R_1, S_1, A_1, R_2, S_2, ...$ naziva se putanjom i dobija se interakcijom agenta sa okruženjem. Putanja može biti ili konačna i beskonačna. Neophodno je definisati i funkciju prelaska, p:

$$p(s', r \mid s, a) \stackrel{\text{def}}{=} P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

koja predstavlja verovatnoću prelaska u stanje s' i dobijanje nagrade r pod uslovom da je u stanju s preduzeta akcija a. Još jedna bitna pretpostavka je da je p raspodela verovatnoće, iz čega sledi da:

$$\sum_{s',r} p(s', r \mid s, a) = 1$$

za sve s i $a \in \mathcal{A}(s)$. Ova skraćena oznaka za dvostruku sumu biće korišćena nadalje i označava sumiranje po svim $r \in \mathcal{R}$ i svim $s' \in \mathcal{S}$.

[1] $\mathbb{P}(\mathbb{A})$ označava partitivni skup skupa $\mathbb{A}$.

Markovljevi procesi odlučivanja imaju takozvano Markovljevo svojstvo tj. osobinu da trenutno stanje i nagrada zavise isključivo od prethodnog stanja i u njemu preduzete akcije a ne od cele putanje koja je dovela do datog stanja. Formalno, ovo svojstvo zapisuje se sledećom jednakošću:

$$P(S_t, R_t \mid S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}) = P(S_t, R_t \mid S_{t-1}, A_{t-1})$$

U odnosu na putanju od trenutka t, može se govoriti o dugoročnoj nagradi, koja se još naziva i dobitkom:

$$G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \tag{4.1}$$

Metaparametar γ naziva se umanjenjem i ukazuje na to koliko se značaja pridaje kasnije dobijenim nagradama u odnosu na neposrednu nagradu. Za $\gamma = 0$[2], buduće nagrade nisu bitne dok postavljanje vrednosti γ na 1 ukazuje na to da se smatra da su sve nagrade putanje podjednako bitne. Iz ove sume uočava se i odnos sa dugoročnom nagradom od trenutka $t + 1$:

$$\begin{aligned} G_t &= \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \\ &= R_{t+1} + \gamma \sum_{i=0}^{\infty} \gamma^i R_{(t+1)+i+1} \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Sada je moguće dati formalnu definiciju: Markovljev proces odlučivanja je uređena petorka $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$. Ova definicija je prilično jednostavna a ipak je dovoljno fleksibilna za formalne opise raznih modela. Ako su skupovi $\mathcal{S}$, $\mathcal{R}$, $\mathcal{A}(s)$, za svako s, konačni, tada se za Markovljev proces odlučivanja kaže da je konačan. Umesto zahteva da $\mathcal{A}(s)$ bude konačan za svako s, može se zahtevati da je skup $\mathbb{A}$ konačan.

Markovljevi procesi odlučivanja koriste se za modeliranje interakcije sa okruženjem i donošenje odluka. Ova primena pokazala se izuzetno pogodno za probleme učenja potrepljivanjem. Cilj agenta biće maksimizacija dugoročne nagrade prilikom interakcije sa okruženjem.

Epizode

U jednakosti (4.1) pretpostavlja se da niz interakcija sa okruženjem, tj. putanja, traje beskonačno, što se vidi iz gornje granice u sumi. Međutim, često je prirodnije pretpostaviti da su putanje konačne i da se završavaju u nekom posebnom stanju iz kog nije moguće dalje preduzimati akcije. Ovakva stanja nazivaju se završnim stanjima. Takvih stanja može biti više ali, zbog načina na koji je definisana funkcija prelaska[3], za time nema potrebe i bez gubitka opštosti se može pretpostaviti da je ovakvo stanje, ukoliko postoji, jedinstveno. Jedan niz interakcija agenta sa okruženjem koji se završava završnim stanjem naziva se epizodom. Epizode su međusobno nezavisne u smislu da ishod jedne epizode ni

[2] 0^0 definiše se kao 1.

[3] Agent dobija nagradu za preduzimanje akcije u stanju a ne za dolazak u stanje.

na koji način ne utiče na neku od narednih epizoda, što se tiče samog okruženja. Ukoliko agent treba da igra, na primer, šah, partije se mogu smatrati epizodama.

Neki problemi ne mogu se razbiti na epizode. Ovi problemi predstavljaju dugoročne zadatke kao što su beskonačno balansiranje uspravnog štapa ili odbrana od nadolazećih talasa neprijatelja u slučaju nekih video igara. Kod ovakvih problema, jako je važno postaviti umanjenje na vrednost manju od 1. Naime, ukoliko važi $\gamma < 1$ i niz nagrada R_t je ograničen, tada će suma (4.1) konvergirati. Ukoliko ta suma divergira, odnosno ako je njena vrednost ∞ ili neodređena, tada će maksimizacija postati trivijalna, odnosno nemoguća.

Sa stanovišta završnih stanja i dugoročne nagrade, ova dva slučaja mogu se objediniti bez izmene (4.1). Kod problema koji se ne mogu podeliti na epizode, suma ostaje ista uz zahtev da je umanjenje strogo manje od 1. Kod problema koji se mogu podeliti na epizode, može se uvesti pretpostavka da će se iz završnog stanja sa verovatnoćom 1 prelaziti u to isto stanje uz vrednost nagrade 0. Ovo je neizvodljivo za implementaciju pa se u praksi koristi konačna suma oblika:

$$G_t = \sum_{i=0}^{T} \gamma^i R_{t+i+1}$$

gde je $t + T + 1$ trenutak kraja epizode.

Politika; vrednosti stanja i akcije

Kako je neophodno opisati pravila ponašanja agenta, uvodi se funkcija π, koja predstavlja verovatnoću da agent u stanju s preduzme akciju a. Ova funkcija naziva se politikom i, ukoliko neki agent preduzima akciju a u stanju s sa verovatnoćom $\pi(a \mid s)$, za sva stanja s i sve akcije a, kaže se da agent prati politiku π. Vrednosti $\pi(a \mid s)$, kad $a \notin \mathcal{A}(s)$, se ne razmatraju.

Ukoliko za neku politiku π važi da za svako s postoji stanje a_s takvo da $\pi(a_s \mid s) = 1$ i $\pi(a \mid s) = 0$ za sve ostale akcije a, tada se kaže da je politika deterministička. Radi jednostavnosti, u tom slučaju se može napisati $\pi(s) = a_s$.

Ako je poznata politika π, može se definisati funkcija vrednosti stanja pri praćenju politike π:

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s]$$

Na sličan način uvodi se i funkcija vrednosti akcije u stanju pri praćenju politike π:

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

Simbol $\mathbb{E}_\pi$ označava matematičko očekivanje ako se podrazumeva da se pri preduzimanju akcija prati politika π. Radi jednostavnostu će u nastavku funkcija v_π, odnosno q_π, biti nazivana samo funkcija vrednosti stanja, odnosno funkcija vrednosti akcije u stanju, dok će oznaka politike biti zapisana u indeksu.

Mnogi algoritmi učenja potkrepljivanjem zasnivaju se na nalaženju optimalne politike, odnosno politike čijim se praćenjem dolazi do maksimalne dugoročne nagrade. Moguće je

uvesti parcijalno uređenje politika definisano na sledeći način:

$$\pi_1 \leq \pi_2 \overset{\text{def}}{\iff} \Big(\forall s \in \mathcal{S}\Big)\Big(v_{\pi_1}(s) \leq v_{\pi_2}(s)\Big) \tag{4.2}$$

i tada se kaže da politika π_2 nije lošija od politike π_1. Ukoliko postoji neka politika π_* koja nije lošija ni od jedne politike za dati Markovljev proces odlučivanja, tada se ona naziva optimalnom politikom.

Belmanove jednakosti; optimalna politika

Funkcije v_π i q_π zadovoljavaju rekurentne relacije koje se zovu Belmanovim jednakostima. U daljim izvođenjima za funkciju v_π podrazumeva se da jednakosti važe za sva stanja s odnosno za sva stanja s i dozvoljene akcije u tim stanjima, a, za q_π. Važi:

$$\begin{aligned} v_\pi(s) &\overset{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_t = s] \end{aligned}$$

Sada je neophodno izračunati oba sabirka. Prvi sabirak označava očekivanu neposrednu nagradu polazeći iz stanja s i prateći politiku π:

$$\mathbb{E}_\pi[R_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s',r} r p(s', r \mid s, a)$$

Iz stanja s preduzima se akcija a sa verovatnoćom $\pi(a \mid s)$, dok se za preduzetu akciju a u stanju s sa verovatnoćom $p(s', r \mid s, a)$ prelazi u stanje s' uz dobijanje nagrade r.

Drugi sabirak, bez množioca γ, proširuje se na sledeći način:

$$\begin{aligned} \mathbb{E}_\pi[G_{t+1} \mid S_t = s] &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) v_\pi(s') \end{aligned}$$

Sumiranje se vrši i po r jer nije nužno da stanju s' odgovara jedinstvena nagrada r.

Spajanjem dve jednakosti dobija se:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} r p(s', r \mid s, a) + \gamma \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) v_\pi(s') \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s')\Big] \end{aligned} \tag{4.3}$$

Analogno se izvodi rekurentna veza za funkciju vrednosti akcije u stanju, q_π:

$$q_\pi(s, a) = \sum_a \pi(a \mid s) \sum_{s',r} \Big[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a')\Big] \tag{4.4}$$

Jednakosti (4.3) i (4.4) nazivaju se Belmanovim jednakostima i ključne su za mnoge algoritme učenja potkrepljivanjem.

Iz definicije optimalne politike, π_*, slede definicije optimalne funkcije vrednosti stanja i optimalne funkcije vrednosti akcije u stanju koje odgovaraju optimalnoj politici, u oznaci v_* i q_*:

$$v_*(s) \stackrel{\text{def}}{=} \max_\pi v_\pi(s)$$

$$q_*(s,a) \stackrel{\text{def}}{=} \max_\pi q_\pi(s,a)$$

Ukoliko neki agent prati optimalnu politiku, ona će ga dovesti do maksimalne dugoročne nagrade. Ovo sledi iz činjenica da $v_\pi(s)$ predstavlja dugoročnu nagradu polazeći iz stanja s, i da v_* za svako stanje s predstavlja najveću vrednost stanja među svim politikama.

Moguće je uspostaviti vezu između v_* i q_*:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_*(s,a)$$

$$q_*(s,a) = \mathbb{E}_\pi\Big[R_{t+1} + \gamma v_*(s_{t+1}) \mid S_t = s, A_t = a\Big]$$

Relativno jednostavnim izvođenjem dolazi se do još jednog para jednakosti koje se nazivaju Belmanovim jednakostima optimalnosti.

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big] \tag{4.5}$$

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big] \tag{4.6}$$

U slučaju konačnih Markovljevih procesa odlučivanja, rekurentne relacije (4.5) čine sistem od $n = |\mathcal{S}|$ jednačina sa n nepoznatih.[4] Isto važi i za relacije (4.6). Ovi sistemi sadrže funkciju max pa stoga nisu linearni. Ukoliko je funkcija prelaska za dati MDP poznata, ovi sistemi se mogu rešiti; ta rešenja dobijena su bez prethodnog znanja o optimalnoj politici.

Ukoliko je v_* poznata, moguće je odrediti optimalnu politiku. Iz jednakosti (4.5) jasno je da postoji jedna ili više akcija koje u stanju s dovode do maksimalne sume. Bilo koja politika koja nekim od ovih akcija dodeljuje nenula vrednosti a svim ostalim dodeljuje vrednost 0 je optimalna. Ukoliko je poznata funkcija q_*, nalaženje optimalne politike još je jednostavnije: u svakom stanju s preduzima se akcija a takva da se maksimizuje $q_*(s,a)$. Poznavanje q_*, dakle, omogućuje nalaženje optimalne politike bez ikakvog uvida u funkciju prelaska. Međutim, ovaj direktni pristup nalaženju v_* ili q_* obično se ne koristi jer funkcija prelaska u praksi uglavnom nije poznata.

4.3 Rešavanje Markovljevih procesa odlučivanja

Uz pretpostavku da se radi o konačnom MDP, osnovni pristup njegovom rešavanju jeste koristeći principe dinamičkog programiranja. Kako je cilj učenja traženje optimalne

[4] Kao što je pomenuto na početku dela o Belmanovim jednakostima, one važe za svako stanje s.

politike, dve politike biće poređene u skladu sa definicijom (4.2), odnosno koristeći funkciju vrednosti politike. Stoga, prvo je neophodno naći metod kojim se računa funkcija vrednosti stanja koja odgovara nekoj politici π. Način na koji se ovo radi je korišćenjem Belmanovih jednakosti, (4.3). Počinje se od vektora v_0 koji sadrži proizvoljne vrednosti i dužine je koja odgovara broju stanja. Dalje se primenjuje iterativno ažuriranje po sledećem pravilu:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big]$$

za sve $s \in \mathcal{S}$. Pod uslovom da $\gamma < 1$, niz vektora v_k konvergira ka v_π. Proces se zaustavlja kada razlika dve uzastopne funkcije, v_k i v_{k+1}, postane dovoljno mala, u skladu sa normom $\|\cdot\|_\infty$ definisanom na sledeći način:

$$\|v\|_\infty = \max_x |v(x)|$$

Sada je i u praksi moguće uporediti dve politike.

Kada je data deterministička politika π, postavlja se pitanje da li je moguće unaprediti je, tj. da li u stanju s odabrati akciju $\pi(s)$ ili neku drugu akciju? Tada je za sve akcije a dozvoljene u s neophodno ispitati vrednost $q_\pi(s, a)$ u odnosu na vrednost $v_\pi(s)$. Ako neko s postoji akcija a takva da je $q_\pi(s, a) \geq v_\pi(s)$, tada je moguće konstruisati politiku π' takvu da važi $\pi' \geq \pi$. Ovo važi na osnovu teoreme o unapređenju politike (eng. *policy improvement theorem*):

Teorema. *Neka su π i π' dve deterministične politike takve da za sve $s \in \mathcal{S}$ važi $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. Tada politika π' nije lošija od politike π, odnosno $\pi' \geq \pi$.*

Dokaz teoreme relativno je jednostavan i može se naći u [8]. Preostalo je da se za proizvoljnu politiku π nađe politika π' takva da za svako s važi $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. Ukoliko se u svakom stanju s odabere akcija a koja maksimizuje izraz $q_\pi(s, a)$, tada tako dobijena politika zadovoljava uslove teoreme. Dakle, od politike π dobija se politika π' tako što se za svako $s \in \mathcal{S}$ računa:

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s, a)$$

Proces dobijanja π' od π naziva se unapređenjem politike (eng. *policy improvement*) i omogućuje iterativni pristup traženju optimalne politike. Politika π' naziva se pohlepnom politikom jer se njenim praćenjem uvek bira najbolja akcija u neposrednom smislu. Ova politika zadovoljava uslove teoreme jer

$$\begin{aligned}\max_a q_\pi(s, a) &\geq q_\pi(s, \pi(s)) \\ &= v_\pi(s)\end{aligned}$$

Druga jednakost sledi iz definicije funkcija q_π i v_π.

Polazeći od nasumično kreirane determinističke politike i prateći ovaj postupak, naizmenično sa evaluacijom novodobijene politike, dobija se niz politika $\pi_0, \pi_1, ...$ koji u slučaju konačnih MDP konvergira optimalnoj politici, π_*. Ime ovog procesa je iterativno unapređenje politike (eng. *policy iteration*). Nasumična deterministička politika π kreira

se tako što se za svaku vrednost $\pi(s)$ proizvoljno bira element iz $\mathcal{A}(s)$. Proces se zaustavlja kad $\pi'(s) = \pi(s)$, za sve $s \in \mathcal{S}$.

Iterativno unapređenje politike, iako relativno brzo konvergira u odnosu na broj stanja, zahteva evaluaciju politike u svakoj iteraciji. Postoji i algoritam iterativnog unapređenja funkcije vrednosti direktno, bez potrebe za naizmeničnim napređenjem i evaluacijom politike. Naime, umesto traženja optimalne politike, iterativno se traži optimalna funkcija vrednosti stanja, v_*. Na osnovu Belmanove jednakosti optimalnosti za funkciju vrednosti, (4.5), kreira se iterativno pravilo ažuriranja:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \tag{4.7}$$

gde se, kao i kod evaluacije politike, polazi od vektora v_0 čije su vrednosti nasumično odabrane. I ovaj proces konvergira u odnosu na normu $\|\cdot\|_\infty$.

Pravilo ažuriranja (4.7) može se predstaviti kao operator $B : R^{|S|} \to R^{|S|}$ koji se naziva Belmanovim operatorom:

$$Bv(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v(s')\Big]$$

Ukoliko se pokaže da je operator B kontrakcija, tada po teoremi o fiksnoj tački važi da postoji vektor v_* takav da $Bv_* = v_*$. Opis i dokaz ove teoreme može se naći u [17]. Uslov kontrakcije zapisuje se sledećim izrazom:

$$\|Bv_1 - Bv_2\|_\infty \leq \alpha \|v_1 - v_2\|_\infty$$

za neko $\alpha \in [0, 1)$. Dokaz da je Belmanov operator kontrakcija dat je u nastavku:

$$\begin{aligned}
\|Bv_1 - Bv_2\|_\infty &= \max_s |Bv_1(s) - Bv_2(s)| \\
&= \left| \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_1(s')] - \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_2(s')] \right| \\
&\leq \max_a \left| \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_1(s')] - \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_2(s')] \right| \\
&= \max_a \left| \sum_{s',r} p(s', r \mid s, a)\Big[(r + \gamma v_1(s')) - (r + \gamma v_2(s'))\Big] \right| \\
&= \gamma \max_a \left| \sum_{s',r} p(s', r \mid s, a)\Big[v_1(s') - v_2(s')\Big] \right| \\
&\leq \gamma \max_s |v_1(s) - v_2(s)| \\
&= \gamma \|v1 - v2\|_\infty
\end{aligned}$$

Ukoliko je $\gamma \in [0, 1)$, Belmanov operator je kontrakcija. Treći red sledi iz činjenice da

$$\left| \max_x f(x) - \max_x g(x) \right| \leq \max_x |f(x) - g(x)|$$

dok pretposlednji red sledi iz činjenice da je p raspodela verovatnoće za fiksirane parametre s i a. Pošto je dokazano da postoji jedinstvena fiksna tačka Belmanovog operatora i kako

po njegovoj konstrukciji važi $Bv \geq v$, vidi se da je niz dobijen njegovom uzastopnom primenom rastući. Stoga je fiksna tačka zaista i optimalna funkcija vrednosti.

Ranije je objašnjeno kako se, ukoliko je poznata funkcija v_*, može konstruisati optimalna politika. Iz postojanja v_* sledi postojanje π_* ali ovakvih politika može biti više zbog mogućnosti da iz jednog stanja više akcija vode ka stanjima sa istom, maksimalnom, vrednošću.

Iako sporiji postupak, traženje q_* funkcije pogodnije je od traženja v_* u slučajevima kada funkcija prelaska nije poznata. Pristupi traženju ovih funkcija dinamičkim programiranjem podjednako su zastupljeni.

Funkciju vrednosti stanja i funkciju vrednosti akcije u stanju moguće je odrediti metodima linearnog programiranja. Jedan od primera je traženje v_* na osnovu jednakosti (4.5), postavljanjem skupa uslova:

$$v(s) \geq \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v(s')\Big], \text{ za sve } s \in \mathcal{S} \text{ i sve } a \in \mathcal{A}(s)$$

Ovih uslova ima $\sum_{s\in\mathcal{S}} |\mathcal{A}(s)| \leq |\mathcal{S}||\mathbb{A}|$. Pri tim uslovima, treba minimizovati izraz $\sum_{s\in\mathcal{S}} v(s)$. Rešavanje ovog problema daje v_*. Ovaj pristup rešavanju MDP-a pri određenim uslovima daje teorijski bržu konvergenciju od pristupa dinamičkim programiranjem. Uprkos tome, pri porastu broja stanja, pristup linearnim programiranjem brže postaje neizvodljiv od pristupa dinamičkim programiranjem.

4.3.1 Učenje potkrepljivanjem u nepoznatom okruženju

Do sada opisani pristupi rešavanju problema učenja potkrepljivanjem opisuju postupak u slučaju da su informacije o okruženju dostupne, odnosno da je funkcija prelaska, p, poznata. Međutim, u realnim situacijama ovo često nije slučaj. Učenje kada funkcija prelaska nije poznata naziva se učenjem u nepoznatom okruženju. U ovoj situaciji, na neki način neophodno je prikupljati podatke o okruženju. Pri učenju u nepoznatom okruženju, moguća su dva pristupa za učenje i skupljanje podataka o okruženju. Prvi podrazumeva praćenje neke politike i njeno konstantno unapređenje na osnovu signala dobijenog od okruženja. Ovo je pristup u skladu sa politikom (eng. *on-policy*). Drugi pristup podrazumeva aproksimaciju optimalne politike dok se dela na osnovu neke druge politike. Naziv je pristup mimo politike (eng. *off-policy*). Koji god pristup da se odabere, zbog toga što podataci o okruženju nisu dostupni, neophodno je održavati balans između iskorišćavanja već naučenog i istraživanja. Pitanje koji se javlja prilikom traženja ovog odnosa naziva se dilema između istraživanja i iskorišćavanja (eng. *exporation vs. exploitaion dilemma*) i prisutna je i u svakodnevnom životu. Čest pristup rešavanju ove dileme jeste ε-pohlepna politika, koja podrazumeva da se, za neko $\varepsilon \in (0, 1]$, sve akcije sem najbolje u nekom stanju s biraju sa verovatnoćom $\frac{\varepsilon}{|\mathcal{A}(s)|}$ dok se najbolja bira sa verovatnoćom $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$. Drugim rečima, sa verovatnoćom $1-\varepsilon$ bira se najbolja akcija u skladu sa nekom politikom a sa verovatnoćom ε biće odabrana nasumična vrednost iz $\mathcal{A}(s)$. Metaparametar ε često se menja u toku učenja.

Osnovni primeri metoda u skladu sa politikom i metoda mimo politike u nepoznatom okruženju su, redom, Sarsa i Q učenje. Metod Sarsa podrazumeva učenje politike

implicitno, tako što se iterativno unapređuje funkcija vrednosti akcije u stanju koja joj odgovara. Za izvršavanje ovog algoritma neophodni su sledeći podaci u jednom vremenskom trenutku, t: stanje u kome se na početku agent nalazi, s_t, akcija koja je preduzeta u tom stanju, a_t, nagrada i stanje koji zauzvrat dobijeni, r_{t+1} i s_{t+1} i akcija koja se u novom stanju preduzima, a_{t+1}. Ovi podaci čine petorku, $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, odakle potiče ime metoda. Pravilo ažuriranja tekuće aproksimacije q funkcije je:

$$\begin{aligned} q(s_t, a_t) &\leftarrow q(s_t, a_t) + \alpha[R_{t+1} + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t)] \\ &= (1-\alpha)q(s_t, a_t) + \alpha[R_{t+1} + \gamma q(s_{t+1}, a_{t+1})] \end{aligned}$$

Simbol $\leftarrow$ označava da je nova vrednost za $q(s_t, a_t)$ dobijena od tekuće datim izrazom,[5] dok je α stopa učenja, metaparametar čija je uloga da odredi koliku težinu imaju nove vrednosti u odnosu na stare, kao što se vidi iz druge jednakosti. Stopa učenja ne mora biti fiksna. Petorke $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ prikupljaju se u skladu sa politikom određenom trenutnom aproksimacijom q. Uz određene uslove vezane za ε i stopu učenja, ovaj postupak dovešće do konvergencije ka optimalnoj funkciji vrednosti akcije u stanju.

Predstavnik učenja mimo politike je takozvano Q učenje (eng. *Q-learning*). Ovaj algoritam podrazumeva poboljšavanje tekuće aproksimacije uz pretpostavku da će se u narednom stanju odabrati najbolja akcija, nezavisno od toga koja će akcija zaista biti preduzeta. Pravilo ažuriranja je:

$$q(s_t, a_t) \leftarrow \ q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a q(s_{t+1}, a) - q(s_t, a_t)]$$

Algoritam Q učenje garantuje konvergenciju tekuće aproksimacije, q, ka optimalnoj funkciji vrednosti akcije u stanju, q_*, ako se vrednosti za sve parove s, a stalno ažuriraju. Na ovom algoritmu zasnovan je DQN, opisan kasnije.

U oba algoritma se vrednosti za $q(s, a)$ inicijalizuju na nasumično odabrane vrednosti, sem za završna stanja, gde se, nezavisno od akcije, vrednost q funkcije postavlja na 0. Takođe, politika se ne konstruiše tako da se u svakom stanju s bira akcija koja maksimizuje $q(s, a)$, odnosno na pohlepan način, već se uvek uključuje element istraživanja. Često se od trenutne aproksimacije q konstruiše ε-pohlepna politika.

Na prvi pogled, ova dva algoritma deluju jako slično. U slučaju da ne postoji element istraživanja, odnosno ako se od trenutne aproksimacije q konstruise politika na pohlepan način, tada su i isti. Međutim, element istraživanja uvek postoji jer, u suprotnom, do učenja ne bi došlo. Glavna razlika između Sarsa algoritma i Q učenja jeste u tome što u se prvom algoritmu ažuriranje vrši na osnovu akcije koja će se zaista preduzeti praćenjem politike, dok će se u drugom algoritmu ažuriranje uvek vršiti optimistično, kao da će se praćenjem politike uvek odabrati akcija koja maksimizuje $q(s, a)$ po a, što nije nužno tačno.

Svi navedeni metodi prestaju da budu izvodljivi kada MDP dostigne određenu veličinu. Ovo se, na primer, može desiti ukoliko je prostor stanja neprekidan ali je urađena diskretizacija ili ako je sam prostor diskretan ali izuzetno veliki. U velikim prostorima može se desiti da su dva ili više stanja suštinski ista ali do sada opisani algoritmi nisu u stanju to

[5] Nisu korišćeni izrazi oblika $q_k(s_t, a_t)$, koji označavaju aproksimaciju u k-toj iteraciji jer jedno ažuriranje utiče na vrednost samo za jedan par s_t, a_t.

da zaključe. Uz to, može se desiti da postoje stanja u koja je nemoguće doći. Prethodno opisani algoritmi oslanjaju se na to da će se stanja iznova posećivati ali i ovo postaje problem u velikim prostorima. U cilju rešavanja ovih problema, pribegava se korišćenjem nekih od modela nadgledanog učenja radi aproksimacije funkcije vrednosti stanja ili akcije u stanju. Nadgledano učenje zahteva skup unapred uparenih ulaznih i izlaznih vrednosti ali, kao što je već pomenuto, u učenju potkrepljivanjem ove informacije nisu dostupne. Stoga je neophodno izvršiti određene izmene algoritama učenja poktrepljivanjem. Jedan primer ove kombinacije pristupa je DQN, opisan u nastavku.

Glava 5

Duboko Q učenje (DQN)

Razvojem neuronskih mreža i rastom njihove popularnosti, nameće se pitanje da li je moguće izvršiti funkcionalnu aproksimaciju funkcije vrednosti stanja ili akcije u stanju koristeći duboke neuronske mreže. Međutim, funkcionalna aproksimacija optimalne funkcije vrednosti akcije u stanju mimo politike nelinearnom funkcijom, kao što je duboka neuronska mreža, nema teorijske ni empirijske garancije, kao što je pokazano u [19]. Termin duboko Q učenje (eng. *deep Q learning*) odnosi se baš na ovakvo uopštenje algoritma Q učenja. Prvi uspešan pristup rešavanju problema učenja potkrepljivanjem na ovaj način prikazan je 2013. godine u radu pod naslovom "Playing Atari with Deep Reinforcement Learning" [15]. U radu je predstavljen algoritam kojim, korišćenjem konvolutivne neuronske mreže i još nekih elemenata, agent uspešno uči da igra video igre sa Atari 2600 konzole. Za razliku od nekih ranijih radova, u kojima su bili konstruisani razni atributi na osnovu kojih se uči, u tom radu opisuje se agent koji uči samo na osnovu slike koja je dostupna na ekranu i rezultata u igri, što znači da će se neophodni atributi konstruisati treniranjem neuronske mreže. Autori su predstavljeni algoritam imenovali duboko Q učenje a neuronsku mrežu koja služi za aproksimaciju duboka Q mreža (eng. *Deep Q network*, skr. *DQN*). Baš zbog date mreže koja je deo modela, algoritam se često naziva *DQN* pa će to biti slučaj i u ovom radu.

Ovaj rad popraćen je raznim sličnim rezultatima od kojih je verovatno najpoznatiji rad koji je objavio DeepMind, pod naslovom "Human-level Control through Deep Reinforcement Learning" [16]. U nastavku je opisan algoritam u celosti, kao i unapređenja predstavljena u [16].

5.1 Struktura agenta

Osnovna struktura agenta vodi se idejom Q učenja s tim što se za aproksimaciju q_* funkcije koristi neuronska mreža. Uz to, primenjivane su određene tehnika koje osiguravaju stabilnu konvergenciju.

Za učenje će biti korišćene četvorke (s, a, r, s') koje predstavljaju deo putanje i koje će biti nazivane prelazima. Glavni elementi sistema za učenje su neuronska mreža koja

aproksimira q_* funkciju i memorija koja čuva već viđene prelaze radi učenja nad njima. U nastavku su opisani ovi elementi i prikaz samog algoritma.

5.1.1 Q mreža

U prethodno razmatranim algoritmima učenja potkrepljivanjem, q funkcija mogla se predstaviti tablicom. Redovi bi označavali stanja a kolone akcije (ili obrnuto) i u njihovom preseku nalazila bi se vrednost akcije u stanju. Dakle, funkcija vrednosti akcije u stanju je funkcija dva argumenta. Umesto tablice, radi treniranja agenta da igra video igru, biće korišćena duboka konvolutivna neuronska mreža. Oznaka ove funkcije biće $q_w(s, a)$, gde su w parametri mreže. Prateći terminologiju korišćenu u radovima, mreža će biti nazivana dubokom Q mrežom ili samo Q mrežom.

Kao i osnovni algoritam Q učenja, DQN će se zasnivati na Belmanovoj jednakosti optimalnosti za q funkciju:

$$q_*(s, a) = \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]$$

Pretpostavlja se da je okruženje deterministično, tj. da ukoliko se u stanju s preduzme akcija a, tada se sa verovatnoćom 1 prelazi u neko drugo stanje s' i dobija nagrada r dok je verovatnoća da se pređe u neko drugo stanje ili da se dobije neka druga nagrada 0. Prvo pitanje koje treba postaviti jeste kako definisati funkciju greške. Cilj treniranja je približno odrediti optimalnu funkciju vrednosti akcije u stanju, q_*. Kako q_* zadovoljava Belmanove jednakosti, teži se tome da njena aproksimacija, q_w približno zadovoljava ove jednakosti. Vodeći se ovom idejom, biće konstruisana funkcija greške, koju treba minimizovati.

Za jednu četvorku s, a, r, s' treba minimizovati izraz:

$$\left(q_w(s, a) - \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_w(s', a')\Big]\right)^2$$

Međutim, kako funkcija p nije poznata, biće korišćena informacija o tome kolika je nagrada za preduzimanje akcije a u stanju s i koje je novo stanje s'. Stoga, izraz

$$\sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_w(s', a')\Big]$$

aproksimira se sledećom vrednosti

$$r + \gamma \max_{a'} q_w(s', a')$$

gde se pri računanju $q_w(s', a')$ koristi informacija o tome da li je stanje s' završno. Ukoliko jeste, vrednosti $q_w(s', a')$ za sve a' biće 0. Dakle, funkcija greške može se predstaviti kao sledeće očekivanje:

$$L(w) = \mathbb{E}_{s,a,r,s'}\left[\Big(r + \gamma \max_{a'} q_w^-(s', a') - q_w(s, a)\Big)^2\right]$$

Oznaka q_w^- ukazuje na to da se za traženje maksimalne vrednosti akcije u stanju s koriste težine mreže fiksirane pre ažuriranja. Za optimizaciju se koristi stohastički gradijentni spust, opisan u 3.1.2. Oznaka $\mathbb{E}_{s,a,r,s'}$ označava očekivanje u skladu sa raspodelom iz koje se dobijaju četvorke (s, a, r, s') ali će u praksi ove četvorke biti prikupljane empirijski jer o samoj raspodeli iz koje potiču unapred ne postoje informacije.

Još jedan izbor koji treba načiniti vezan je za strukturu izlaza i ulaza u mrežu. Ukoliko se mreža konstruiše tako da kao ulaz prima stanje i preduzetu akciju i kao izlaz daje vrednost dosadašnje aproksimacije, tada bi za odabir najbolje akcije u stanju bilo neophodno propustiti podatke kroz mrežu jednom za svaku akciju. Ovaj pristup mogao bi znatno da uspori obučavanje. S druge strane, moguće je koristiti arhitekturu takvu da mreža kao ulaz primi stanje a kao izlaz da vektor vrednosti za svaku od mogućih akcija. Ova arhitektura biće korišćena u implementaciji.

U radu koji je objavio DeepMind 2015. godine, predloženo je da se koriste dve mreže, Q mreža i ciljna mreža. Težine Q mreže biće ažurirane, dok će se izraz $\max_{a'} q_w^-(s', a')$ računati u skladu sa ciljnom mrežom. Na svakih C koraka, težine Q mreže kopiraće se u ciljnu mrežu. Ova tehnika osigurava stabilnije učenje i bolje rezultate, kao što je pokazano u [16].

5.1.2 Memorija

Prilikom prikupljanja i korišćenja prelaza za obučavanje treba biti oprezan: ukoliko su uzastopno prikupljene četvorke visoko korelisane, može doći do neefikasnog učenja. Potreba za uklanjanjem visoke korelisanosti uzastopnih prelazaka motiviše korišćenje memorije (eng. *experience replay* ili *replay memory*) u kojoj će se čuvati ti podaci. Nasumični uzorak iz ove memorije koristiće se za obučavanje Q mreže, čime se smanjuje verovatnoća da je skup podataka nad kojima se vrši ažuriranje parametara korelisan. Memorija ima još jednu ulogu prilikom obučavanja agenta. Naime, uzorkovanjem se prelazi potencijalno više puta koriste za treniranje mreže, što znači da je stopa iskorišćenosti podataka veća.

Kako samo treniranje može da bude vrlo dugo, nije izvodljivo čuvati sve prelaze viđene do nekog trenutka već samo poslednjih N. Veličina memorije je jedan od metaparametara modela.

5.2 Algoritam DQN

Sada su dati svi elementi neophodni za konstrukciju *DQN* algoritma. Algoritam 1 okvirno opisuje postupak treniranja agenta. Radi lakšeg razumevanja, dosta detalja je izostavljeno ili izmenjeno. Ti detalji, vezani za konkretnu implementaciju, dati su u glavi 6.

Oznake su sledeće:

- D je memorija u kojoj se čuva poslednjih N viđenih prelaza.

- q_w je Q mreža a $\hat{q}_{\hat{w}}$ je ciljna mreža.
- M je broj epizoda koje će se odigrati u toku učenja, a e je broj trenutne epizode. Način na koji se vrši podela na epizode takođe se može naći u detaljima implementacije.
- t je brojač koraka unutar jedne epizode.
- x_t je stanje ekrana u koraku t.
- s_t je stanje u kom se agent nalazi u koraku t.
- ϕ predstavlja funkciju kojom se vrši pretprocesiranje. ϕ_t označava pretprocesirano stanje s_t.
- C je frekvencija ažuriranja parametara ciljne mreže.

Algoritam 1 DQN

Inicijalizovati memoriju D i postaviti njenu maksimalnu dužinu na N
Inicijalizovati q_w i $\hat{q}_{\hat{w}}$ mreže istim nasumičnim težinama
for $e = 1, ..., M$ **do**
  $s_1 \leftarrow \{x_1\}$
  $\phi_1 \leftarrow \phi(s1)$
  $t \leftarrow 1$
  repeat
    Odabrati $a_t \leftarrow \begin{cases} \text{nasumično}, & \text{sa verovatnoćom } \varepsilon \\ \text{argmax}_a\, q_w(\phi(s_t), a), & \text{inače} \end{cases}$
    Izrši akciju a_t, dobijajući nagradu r_t i sliku ekrana x_{t+1}
    $s_{t+1} \leftarrow s_t, a_t, x_{t+1}$
    $\phi_{t+1} \leftarrow \phi(s_{t+1})$
    Smestiti prelaz $(\phi_t, a_t, r_t, \phi_{t+1})$ u memoriju D
    Dohvatiti nasumičan skup uzoraka $(\phi_j, a_j, r_j, \phi_{j+1})$ iz memorije D
    Izračunati $y_j \leftarrow \begin{cases} r_j, & \text{ako je } s_{j+1} \text{ završno stanje} \\ r_j + \gamma \max a' \hat{q}_{\hat{w}}(\phi_{j+1}, a'), & \text{inače} \end{cases}$
    Izvršiti korak stohastičkog gradijentnog spusta u odnosu na izraz $(y_j - q_w(\phi_j, a_j))^2$
    Svakih C koraka izvršiti dodelu $\hat{w} \leftarrow w$
    $t \leftarrow t + 1$
  until stanje t je terminirajuće

U slučaju da se ne koristi ciljna mreža, za računanje vrednosti y_j koriste se vrednosti $q_w^-(\phi_{j+1}, a')$, izračunate korišćenjem Q mreže pre ažuriranja parametara. Tada se takođe ne vrši ažuriranje parametara $\hat{w}$ jer u tom slučaju oni ne postoje.

Glava 6

Detalji implementacije

U ovoj glavi opisana je struktura koda. Implementacija je zasnovana na radovima [15] i [16].

6.1 Detalji implementacije

Prilikom implementacije, korišćen je programski jezik Python [5], verzija 3.5.2 kao i sledeće biblioteke:

- `NumPy` [3] – Biblioteka otvorenog koda za numeričko izračunavanje. Korišćena je verzija 1.14.1.
- `Keras` [2] – Biblioteka otvorenog koda koja pruža visoko apstraktni API, što je čini jednostavnim za korišćenje. Ova biblioteka zahteva pozadinsku biblioteku za izračunavanje; u ovu svrhu je korišćena biblioteka `TensorFlow`[6]. Verzija biblioteke `Keras` koja je upotrebljavana je 2.2.0, dok je verzija biblioteke `TensorFlow` 1.5.0.
- `OpenAI Gym` [4] – Biblioteka otvorenog koda koja pruža mogućnost korišćenja već gotovih okruženja radi istraživanja raznih pristupa učenju potkrepljivanjem. Korišćena je verzija 0.10.5 ove biblioteke.

Implicitno su korišćene i biblioteke od kojih navedene biblioteke zavise. U sklopu implementacije korišćeni su i neki moduli koji su deo standardne biblioteke jezika python.

Implementacija *DQN* sastoji se iz dve klase, *DQNAgent* i *ExperienceReplay*, i nekoliko pomoćnih funkcija. Struktura koda opisana je u nastavku, na apstraktnom nivou. Ovo znači da su opisi nekih funkcija, koje nisu od važnosti za shvatanje koda, izostavljeni. Izvorni kod može se naći na sledećoj adresi: https://github.com/NikolaMilev/DQN.

Klasa `DQNgent` opisuje funkcionisanje agenta koji se obučava prateći *DQN* algoritam. Glavne komponente klase su memorija i dve neuronske mreže: Q mreža i ciljna mreža.

Glavne komponente klase su memorija i Q i ciljna mreža. Metodi bitni za učenje su:

- `chooseAction(self, state)`[1] – Vrši odabir akcije na osnovu stanja koje je prosleđeno referencom `state`. Odabir se vrši u skladu sa ε-pohlepnom politikom.
- `trainOnBatch(self, batchSize)` – Trenira Q mrežu nasumičnim uzorkovanjem memorije, koristeći formule opisane u algoritmu 1.
- `test(self, network, numSteps)`[2] – Vrši testiranje mreže date argumentom `network` u skladu sa pravilima opisanim u delu 7.3. Testiranje traje `numSteps` koraka.
- `learn(self)` – Vrši obučavanje agenta.

Klasa `ExperienceReplay` opisuje omotač oko klase `deque`, koja opisuje objedinjenje pojmova steka i reda.[3] Bitni metodi klase su:

- `addTuple(self, state, action, reward, nextState, terminal)` – Dodaje petorku `(state, action, reward, nextState, terminal)` u memoriju. Prva četiri elementa odgovaraju četvorki (s, a, r, s'), odnosno jednom prelazu koji je agent video, dok peti element predstavlja istinitosnu vrednost koja označava da li je stanje s' završno. Ovu informaciju neophodno je čuvati jer se ona ne nalazi u samim stanjima.
- `getMiniBatch(self, sampleSize)` – Dohvata uzorak veličine `sampleSize` iz memorije.

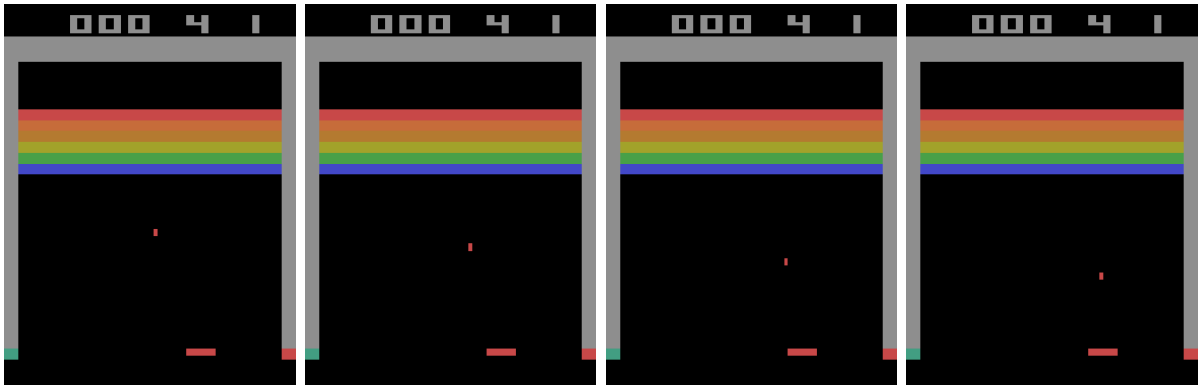
Pored opisanih klasa, kod sadrži i funkcije koje pomažu boljoj strukturi koda:

- `buildNetwork(height, width, depth, numActions)` – Koristeći pozive iz biblioteke Keras, gradi neuronsku mrežu i vraća referencu na dati objekat.
- `copyModelWeights(srcModel, dstModel)` – Kopira težine mreže `srcModel` u mrežu `dstModel`.
- `saveModelWeights(model, name)` – Čuva neuronsku mrežu čija referenca se prosleđuje argumentom `model` na putanju određenu argumentom `name`.
- `preprocessSingleFrame(img)` – Vrši pretprocesiranje jedne slike sa ekrana, opisano u 7.1.
- `transformReward(reward)` – Transformiše nagradu.

[1] Pri pisanju instancnih metoda u jeziku python, prvi argument je uvek referenca na objekat nad kojim se taj metod poziva. Ta referenca označava se ključnom reči `self`

[2] Parametar `network` stoji jer je isti kod korišćen za obučavanje raznih varijacija algoritma pri ispitivanju njegovih komponenti.

[3] Dokumentacija ove klase može se naći na adresi https://docs.python.org/3/library/collections.html#collections.deque.



Slika 6.1: Prikaz 4 uzastopna stanja ekrana u igri Breakout. Pločica miruje dok se loptica kreće ka donjem desnom uglu ekrana.

U više navrata je rečeno da *DQN* algoritam uči nad slikama ekrana i nagradom dobijenom od okruženja. Međutim, jedna slika ne daje mnogo informacija. Na primeru igre Breakout, ukoliko je dostupna samo jedna slika ekrana, nemoguće je odrediti u kom se pravcu kreće loptica, ni kojom brzinom. Isto važi i za pločicu koja služi za odbijanje loptice. Međutim, iz nekoliko uzastopnih slika može se odrediti smer i brzina kretanja loptice i pločice. Na slici 6.1 prikazane su četiri uzastopne slike sa ekrana.[4]

Kako igre sa Atari 2600 konzole prikazuju 60 slika u sekundi (eng. *frame per second*, skr. *fps*) i kako nije realno delati tom frekvencijom, vrši se preskakanje slika (eng. *frame skipping*). Biblioteka `OpenAI Gym` pruža mogućnost korišćenja raznih vrednosti za broj preskočenih slika. U ovom radu, koristi se okruženje pod imenom `BreakoutDeterministic-v4`,[5] što znači da će okruženje za jedan poziv kojim se vrši akcija izvršiti tu akciju na 4 uzastopne slike koja se prikazuje. Deo imena `Deterministic` označava da će se uvek pozivom prikazivati svako četvrto stanje ekrana; neka okruženja dopuštaju da se preskakanje vrši nasumično, u nekom intervalu dozvoljenih vrednosti. Na primer, okruženje `Breakout-v0` za svaki poziv vrši preskakanje nasumično odabranog broja slika iz intervala $[1, 4]$. U ovoj implementaciji, po uzoru na rad [16], agent će videti svako četvrto stanje ekrana.

Na svake 4 viđene slike, vršiće se jedan korak stohastičkog gradijentnog spusta, nasumičnim uzorkovanjem 32 elementa memorije.

6.1.1 Prelazi i čuvanje u memoriji

U cilju čuvanja informacija o smeru kretanja objekata na ekranu, svako stanje biće sačinjeno od određenog broja uzastopnih slika koje agent vidi. Svaki prelaz sačuvan u memoriju u stvari je petorka oblika (s, a, r, s', t). Ono na šta pri implementaciji treba obratiti pažnju su memorijski zahtevi. Nakon transformacije, dobijene slike su dimenzija 105×80 i njihove koordinate implicitno su zapisane u pokretnom zarezu. Ukoliko je jednostruka tačnost u pitanju, jedno stanje zauzima 32.8 kB. Dakle, čuvanje milion stanja, kako je predloženo u radu [16], zahteva između 32 i 33 GB memorije. Ovo je

[4] Radi ilustracije, nisu prikazane uzastopne slike već je razlika između njih 7 slika. Prikazivanje uzastopnih slika sa ekrana dalo bi kao rezultat 4 skoro identične slike.

[5] Ovo okruženje ne može se naći u zvaničnoj dokumentaciji.

moguće popraviti. Naime, kako su vrednosti dobijene nakon transformacije u intervalu $[0, 255]$ i kako će kasnije biti transformisane tako da staju u interval $[0, 1]$, to je bez gubitka značajne količine podataka moguće zaokružiti ih i čuvati ih u jednobajtnom podatku. Biblioteka `NumPy` definiše tip `uint8` koji ovo omogućuje. Takođe treba paziti na to da dva uzastopna stanja dele sve sem jedne slike. Ukoliko se stanja definišu tako da sadrže reference na pojedinačne slike umesto pravljenja novog niza, deljenje referenci osigurava da se neće praviti bespotrebne kopije podataka. Zajedno sa ostalim memorijskim zahtevima, prilikom treniranja agenta bilo je zauzeto oko *10 GB* memorije.

Glava 7

Detalji treniranja i eksperimentalne evaluacije

U ovoj glavi opisano je testiranje agenta, kao i izvršeni eksperimenti, koji su zasnovani na radovima [15] i [16].

7.1 Pretprocesiranje

Okruženja koja služe kao omotači oko igara sa konzole Atari 2600 u `OpenAI Gym` biblioteci kao stanje ekrana vraćaju RGB slike dimenzija 210×160 zapisane u `NumPy` nizu. Na te slike primenjuje se pretprocesiranje u cilju smanjenja računskih zahteva modela. To pretprocesiranje podrazumeva tranformaciju slike tako da se svaki piksel, koji je trojka koja određuje količinu crvene (R), plave (B) i zelene (G) boje zameni njegovom osvetljenošću (Y), određenom na sledeći način:

$$Y = 0.299R + 0.587G + 0.114B$$

Kako su sve vrednosti R, G i B u intervalu $[0, 255]$ i kako se koeficijenti sabiraju na 1, novodobijena vrednost je u intervalu $[0, 255]$. Vrednosti svakog piksela se u ulaznom sloju mreže dele sa 255 i time dovode na interval $[0, 1]$. Još jedna transformacija koja se primenjuje na slike je njihovo smanjivanje. U radu [16] predloženo je da se slike skaliraju na dimenzije 84×84 ali, radi jednostavnosti implementacije i brzine izvršavanja, odlučeno je da će se u ovoj implementaciji visina i širina slike prepolove u odnosu na početnu, dobijajući niz dimenzija 105×80.

Kako je *DQN* algoritam prvobitno ispitivan na mnoštvu igara sa Atari 2600 konzole, nagrade koje su dobijene od okruženja mogu znatno da variraju. Stoga, sve nagrade su odsecane tako da pripadaju intervalu $[-1, 1]$. Nažalost, ovo onemogućava razlikovanje većih i manjih nagrada, što može dovesti do manje efikasnog učenja. Ovaj postupak je zadržan radi ispitivanja njegovog efekta na proces učenja.

7.2 Detalji treniranja

U ovom delu opisana je arhitektura mreže, kao i ostali bitni metaparametri. Zbog hardverskih ograničenja, eksperimentisano je samo sa igrom Breakout sa Atari 2600 konzole. Zbog nedovoljno objašnjene implementacije u radu [15], kao i jer je implementacija vezana za rad [16] u programskom jeziku Lua, bilo je neophodno načiniti određene izmene u metaparametrima. Ipak, glavna logika algoritama je sačuvana. Potpun spisak metaparametara treniranja i testiranja agenta može se naći u tabeli 7.1.

Metaparametar ε predstavlja stopu istraživanja. To je broj koji predstavlja verovatnoću da agent u nekom koraku načini nasumično odabranu akciju. U suprotnom, akcija se bira na pohlepan način u skladu sa politikom koja je izvedena iz trenutne aproksimacije funkcije q_*. Radi što potpunijeg istraživanja prostora stanja igre, koji je ogroman, stopa istraživanja, počinje od 1 i onda se linearno, svakim korakom, smanjuje dok ne postane 0.1. Na ovaj način, agent na početku u potpunosti istražuje dok se kasnije sve više oslanja na izgrađenu aproksimaciju. Stopa istraživanja zadržava se na krajnjoj pozitivnoj vrednosti radi forsiranja agenta da i u odmaklom stadijumu treniranja nastavi da istražuje.

Radovi [15] i [16] predlažu dve slične arhitekture koje se razlikuju u veličini mreže. Obe arhitekture su predstavljene ispod. Zbog ograničenih resursa, u ovom radu treniranje je vršeno koristeći manju mrežu. Samo jedan eksperiment izvršen je koristeći mrežu predloženu u radu [16].

Arhitektura mreže u radu [15] je sledeća. Ulazni sloj je dimenzija $84 \times 84 \times 4$. Nakon ulaznog, sledi konvolutivni sloj koji primenjuje 16 filtera dimenzija 8×8 sa pomerajem 4. Izlazi ovog sloja prosleđuju se u drugi konvolutivni sloj koji primenjuje 32 filtera dimenzija 4×4 sa pomerajem 2. Nakon konvolutivnih slojeva sledi jedan gusto povezani sloj od 256 neurona. Izlazni sloj je takođe gusto povezan i broj neurona koji sadrži odgovara broju mogućih akcija u igri za koju se obučava. Svim slojevima sem ulaznog i izlaznog pridružena je ReLU aktivaciona funkcija. Ova arhitektura biće nazivana arhitekturom I.

U radu [16], arhitektura podrazumeva tri konvolutivna sloja. Ulaz u mrežu takođe je dimenzija $84 \times 84 \times 4$. Prvi konvolutivni sloj sastoji se od 32 filtera dimenzija 8×8 sa pomerajem 4. Drugi konvolutivni sloj primenjuje 64 filtera dimenzija 4×4 sa pomerajem 2. Poslednji konvolutivni sloj primenjuje isto 64 filtera, dimenzija 3×3 i sa pomerajem 1. Nakon konvolutivnih slojeva sledi gusto povezani sloj od 512 neurona. Izlazni sloj, kao i u prethodnoj arhitekturi, sadrži broj neurona koji odgovara broju mogućih akcija u igri za koju se mreža obučava. Svim slojevima sem ulaznog i izlaznog pridružena je ReLU aktivaciona funkcija. Ova arhitektura biće nazivana arhitekturom II.

Za sve eksperimente, u konvolutivnim slojevima nije primenjivano proširivanje. Agregacija takođe nije primenjivana. Ova odluka dolazi iz potrebe da se sačuva informacija o poziciji elemenata na ekranu.

Za optimizaciju se koristi algoritam RMSProp (3.1.2). Nakon računanja, vrši se pojedinačno odsecanje koordinata gradijenta na interval $[-1, 1]$. Ova tehnika korišćena je u [16] zbog potencijalnog velikog rasta samog gradijenta koji može da izazove eksplozije u vrednostima parametara mreže.

Igre sa Atari 2600 konzole često su koncipirane tako da igrač ima određeni broj života koji se na neki način gube i igra se završava kada se svi životi izgube. Ovo dovodi do pitanja kada je kraj epizode. Moguće je odlučiti se da je kraj epizode gubitak jednog života u igri ili gubitak svih života. Radi motivacije agentu da manje gubi živote, u toku procesa treniranja će gubitak jednog života označavati kraj epizode.

Treniranje svih implementacija trajalo je 10000000 koraka, što je na kasnije opisanom računaru trajalo oko 5 dana. Treniranje implementacije koja koristi arhitekturu II zahtevalo je znatno više vremena za obučavanje i, zbog tehničkog problema, izvršilo se malo manje koraka treniranja. Treniranje implementacije koja koristi arhitekturu II trajalo je znatno duže i, zbog tehničkog problema, trajalo je nešto kraće. Uprkos tome, rezultati su, očekivano, bolji od ostalih implementacija.

Tesitranje razlicitih varijacija *DQN* algoritma izvršeno je na računaru sa četiri *AMD Opteron™ Processor 6168* procesora i *96 GB* radne memorije. Iako su za rad sa konvolutivnim mrežama, zbog velikih mogućnosti paralelizacije, često dosta pogodnije grafičke karte, ovaj računar pružio je najpogodniji dostupan hardver.

7.3 Detalji testiranja

Proces testiranja izvršen je po uzoru na [15]. Treniranje i testiranje izvršeni su naizmenično kako bi se pratio napredak u ponašanju agenta. Na svakih 200000 koraka (ovaj period nazvan je epohom), treniranje se pauzira (odnosno, za vreme testiranja, agent ne uči), i vrši se testiranje, uz korišćenje okruženja nezavisnog od okruženja nad kojim se uči. Pri testiranju vrše se evaluacija ponašanja agenta i evaluacija trenutne aproksimacije funkcije q_*.

U toku eksperimentisanja sa elementima *DQN* algoritma korišćeni su isti postupci evaluacije agenta. Kako je cilj kreirati agenta sa najboljim rezultatom u toku epizode, na disk je čuvana mreža sa najboljim postignutim rezultatom u toku evaluacije. Rezultati testiranja različitih eksperimenata mogu se naći u slikama 7.1 do 7.5. Levi grafici predstavljaju rezultate evaluacije trenutne aproksimacije funkcije q_* (dalje samo "evaluacija funkcije q_*") dok desni grafici predstavljaju rezultate evaluacije ponašanja, merene prosečnim rezultatom po epizodi evaluacije.

Evaluacija ponašanja podrazumeva da agent komunicira sa novim okruženjem uz praćenje ε-pohlepne politike, gde je $\varepsilon = 0.05$, i pamćenje prosečnog postignutog rezultata po epizodi. Epizoda evaluacije ponašanja traje dok agent ne izgubi sve živote. Sama evaluacija ponašanja traje bar 10000 koraka; ovo znači da se ona neće završiti čim se odigra taj broj koraka već završetkom epizode čiji je kraj na bar 10000-tom koraku. Na ovaj način poslednja epizoda evaluacije ponašanja neće biti završena pre vremena, što znači da se prosek odnosi na cele epizode. Na disk je sačuvana neuronska mreža koja je dala najbolje rezultate prilikom evaluacija ponašanja.

Razlog korišćenja $\varepsilon > 0$ pri testiranju je to što mnoge igre sa Atari 2600 konzole zahtevaju od korisnika da eksplicitno, nekom akcijom, pokrene igru nakon gubitka života. Ukoliko agent još uvek nije naučio da to treba da se uradi, testiranje bi bilo zaglavljeno

Ime promenljive	Vrednost	Značenje
NETWORK_UPDATE_FREQUENCY	10000	Na koliko izvršenih iteracija treniranja se težine ciljne mreže zamenjuju težinama Q mreže
INITIAL_REPLAY_MEMORY_SIZE	50000	Treniranje počinje kada memorija dostigne ovaj broj elemenata
MAX_REPLAY_MEMORY_SIZE	1000000	Maksimalna veličina memorije
OBSERVE_MAX	30	Najveći broj slika koje agent vidi na početku igre pre nego što počne da dela
TIMESTEP_LIMIT	10000000	Dužina treniranja
MINIBATCH_SIZE	32	Veličina skupa nad kojim se vrši jedan stohastički gradijentni spust
INITIAL_EPSILON	1	Početna vrednost stope istraživanja ε
FINAL_EPSILON	0.1	Krajnja vrednost stope istraživanja ε
EPSILON_DECAY_STEPS	1000000	Broj koraka koji se izvrši pre nego što se vrednost ε spusti sa početne vrednosti na krajnju vrednost
GAMMA	0.99	Stopa umanjenja
NET_H, NET_W	105, 80	Visina i širina ulaza u mrežu
NET_D	4	Broj uzastopnih slika koje čine jedno stanje; dubina ulaza u mrežu
LEARNING_RATE	0.95	Stopa učenja u RMSProp algoritmu (parametar α iz dela 3.1.2)
MOMENTUM	0.95	Momenat; Parametar γ iz dela 3.1.2
MIN_GRAD	0.01	Parametar ε iz dela 3.1.2
TRAIN_FREQUENCY	4	Koliko slika agent vidi između dve optimizacije mreže
SAVE_FREQUENCY	25000	Koliko često se mreža čuva na disku, mereno u broju optimizacija mreže
TEST_STEPS	10000	Koliko koraka se agent testira
TEST_FREQ	200000	Koliko često se agent testira, mereno u broju koraka
TEST_SET_SIZE	2000	Veličina testnog skupa
TEST_EPSILON	0.05	Vrednost parametra ε u toku treniranja

Tabela 7.1: Metaparametri u toku treniranja i testiranja

u beskonačnoj petlji.

Nakon završenog treniranja, učinjena je još jedna evaluacija ponašanja, na osnovu sačuvane mreže. Agent komunicira sa novim okruženjem uz praćenje ε-pohlepne politike, gde je $\varepsilon = 0$.[1] Ovo znači da sve odluke donosi agent. U većini slučajeva, rezultat nije lošiji od najboljeg rezultata dobijenog prilikom evaluacije ponašanja. Ako agent ne nauči da preduzima akciju za početak igre, testiranje može biti zaglavljeno i stoga je ovo testiranje sprovedeno je uz eksplicitno preduzimanje te akcije nakon gubitka života i na samom početku testiranja.

Pri evaluaciji funkcije q_* treba biti pažljiv jer različita stanja mogu imati različite vrednosti. Zbog toga se na početku treniranja bira fiksni skup stanja koji se koristi u kasnijoj evaluaciji. Elementi ovog skupa biraju se nasumično. Evaluacija funkcije q_* vrši se tako se za svako od stanja pomenutog skupa izračuna maksimalna vrednost akcije i izračuna se prosek ovih maksimuma.

Za istu implementaciju, grafik koji predstavlja evaluaciju funkcije q_* sadrži manje šuma nego grafik koji prikazuje evaluaciju ponašanja; ovo je posledica činjenice da male promene težina mreže mogu da izazovu veliku promenu u ponašanju agenta. U [15] je primećeno da grafici koji predstavljaju evaluaciju funkcije q_* prikazuju stabilan napredak, odnosno učenje. Dve implementacije od kojih se očekivalo da daju najbolje rezultate (čiji su grafici prikazani na slikama 7.1 i 7.5) zaista imaju relativno glatke grafike koji predstavljaju evaluaciju funkcije q_*.

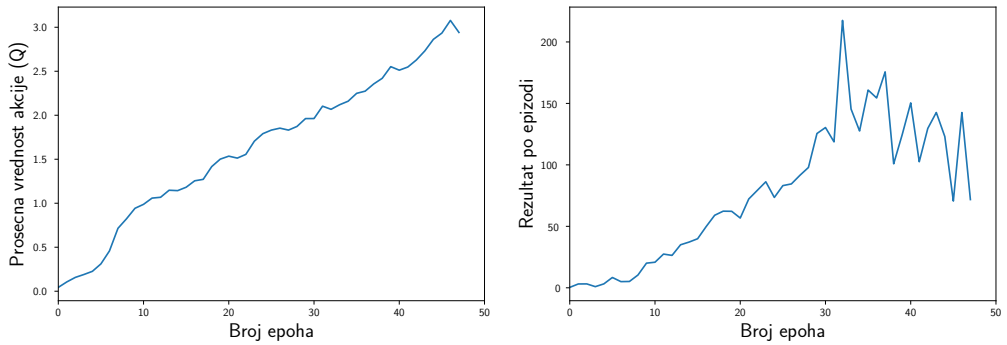
7.4 Eksperimenti

Kao što je diskutovano, dva najvažnija elementa algoritma *DQN*, koja sprečavaju divergenciju aproksimirane funkcije q_*, jesu memorija i ciljna mreža. Slike 7.1 do 7.4 prikazuju rezultate testiranja, koje je vršeno u toku treniranja, u zavisnosti od postojanja datog elementa. Te slike odnose se na mreže sa arhitekturom I, dok se slika 7.5 odnosi na mrežu sa arhitekturom II. Važnost ciljne mreže i memorije jasno se vidi sa ovih slika: agent koji je obučavan koristeći oba elementa (slike 7.1) jasno postiže bolji rezultat nego agenti koji ne koriste bar jedan od tih elemenata.

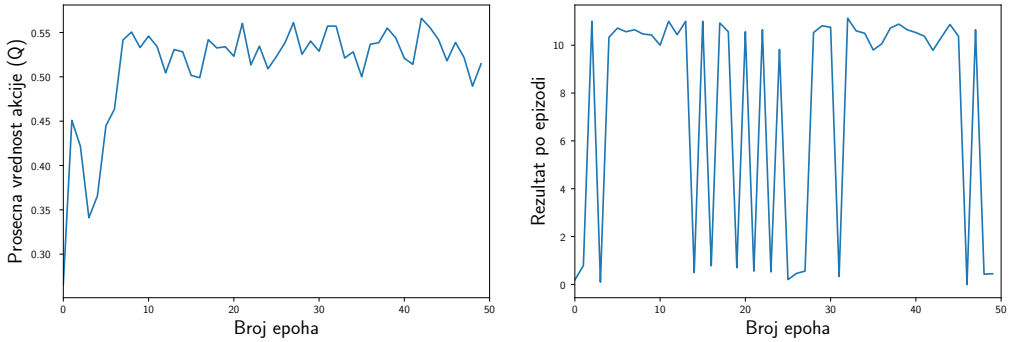
U [16] može se videti da uklanjanjem ciljne mreže rezultat opada ali ne u meri u kojoj se to desilo u eksperimentima koji su izvršeni u sklopu ovog rada. Međutim, optimizacija je empirijski proces koji zahteva mnogo isprobavanja različitih kombinacija metaparametara. Čak i sa proverenom arhitekturom neuronske mreže i algoritmom optimizacije, ovo može biti dug i težak proces. U ovom slučaju, korišćene su druge, šire rasprostranjene biblioteke, i izmenjeni su neki metaparametri; ovo je verovatno dovoljno da dovede do razlike u postignutom rezultatu.

Poredeći slike 7.1 i 7.5 takođe se jasno vidi da mreža sa više slojeva koji su veći čak i sa manje koraka treniranja dostiže bolje rezultate. Ipak, vreme treniranja ovakve mreže bilo je skoro dvostruko u odnosu na arhitekturu I. Zbog ograničenih resursa, arhitektura

[1] Ovakva politika naziva se pohlepnom politikom jer uvek bira najbolju akciju u nekom stanju, u skladu sa trenutnom aproksimacijom funkcije q_*.



Slika 7.1: Rezultati treniranja uz korišćenje ciljne mreže i memorije



Slika 7.2: Rezultati treniranja bez korišćenja ciljne mreže i sa korišćenjem memorije

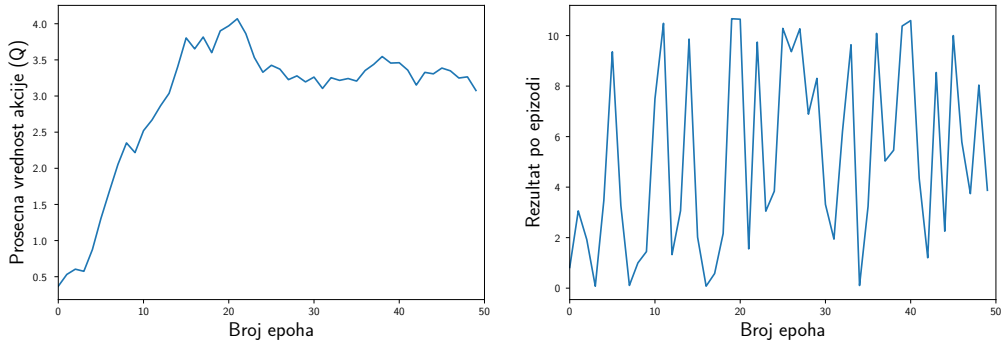
II ispitana je samo uz prisustvo ciljne mreže i memorije, a treniranje je trajalo malo manje od 40 epoha tj. 8 miliona koraka.

Radi lakšeg pregleda, najbolji postignuti rezultati u toku testiranja prikazani su u prvom redu tabele 7.2. U drugom redu iste tabele nalaze rezultati za prethodno opisano testiranje sa $\varepsilon = 0$. Tabela 7.3 prikazuje rezultate dobijene na isti način ali za dublju mrežu sa većim slojevima, odnosno za mrežu sa arhitekturom II.

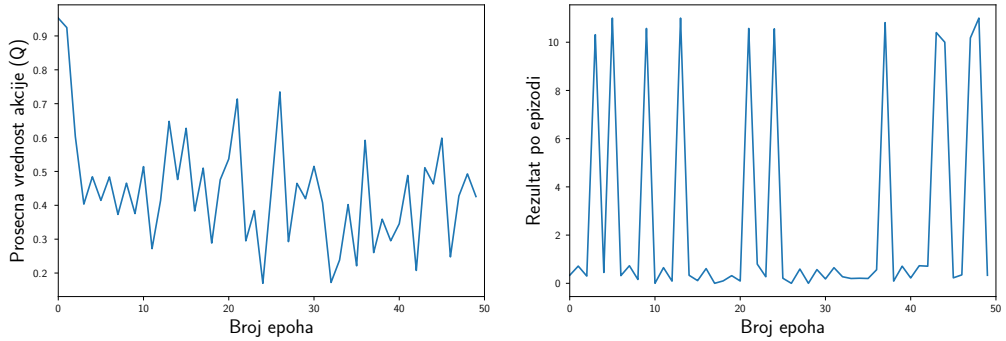
Iako neuronske mreže teorijski nisu dobar izbor za aproksimaciju funkcije q_*, jasno je da je uvođenje memorije i ciljne mreže znatno pomoglo učenju. Ciljna mreža povećava stabilnost učenja dok memorija osigurava smanjenu korelaciju između uzastopnih obučavanja i potencijalno povećava iskorišćenost podataka koje sistem vidi. Iako je algoritam *DQN* danas prevaziđen po pitanju performansi, njegovo uvođenje predstavlja veliki pomak u učenju potkrepljivanjem.

Memorija	✓	✓	✗	✗
Ciljna mreža	✓	✗	✓	✗
$\varepsilon = 0.05$	217.500	11.125	10.667	11.000
$\varepsilon = 0$	269.000	2.000	11.000	11.000

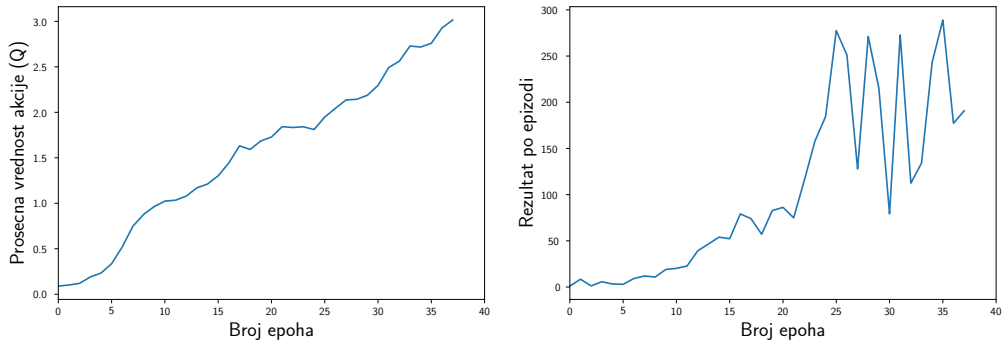
Tabela 7.2: Rezultati testiranja agenta na igri Breakout uz korišćenje arhitekture I



Slika 7.3: Rezultati treniranja uz korišćenje ciljne mreže i bez korišćenja memorije



Slika 7.4: Rezultati treniranja bez korišćenja ciljne mreže i bez korišćenja memorije



Slika 7.5: Rezultati treniranja koristeći ciljnu mrežu i memoriju, uz arhitekturu opisanu u [15]

$\varepsilon = 0.05$	289.00
$\varepsilon = 0$	325.00

Tabela 7.3: Rezultati testiranja agenta na igri Breakout korišćenje arhitekture II

Glava 8

Zaključak

Cilj mašinskog učenja je generalizacija, učenje na osnovu dostupnih podataka i primena tog znanja u novim situacijama. Idealan sistem treba da bude u stanju da uči sa minimalnom količinom ljudske intervencije. Ipak, neretko je neophodno izvršiti razna pretprocesiranja podataka nad kojima se uči. Algoritam *DQN* pokazuje da je moguće kreirati sistem koji uči na osnovu neobrađenih podataka. Koristeći metode učenja potkrepljivanjem zajedno sa neuronskom mrežom i memorijom, čija je svrha da imitira pamćenje, postignuti su zadovoljavajući rezultati pri igranju video igre Breakout sa konzole Atari 2600. Sprovedeni eksperimenti jasno, možda i ekstremno, prikazuju važnost korišćenja ciljne mreže i memorije za proces učenja.

Igranje video igara zahteva donošenje odluka u hodu (eng. *real time*). Uspesi u stvaranju sistema koji nezavisno igraju video igre nameću težnju da se slični sistemi koriste za kontrolu raznih drugih procesa.

Ipak, dobijene rezultate moguće je unaprediti. Idealna taktika igranja igre Breakout podrazumeva pravljenje prolaza kroz cigle i navođenje loptice kroz njega, što daje kao rezultat da loptica udara u cigle sa gornje strane, dajući bolji rezultat uz manje angažovanje agenta. Video zapisi dati u GitHub repoztorijumu[1] prikazuju da su agenti došli prilično blizu ovog ponašanja, u slučaju podešavanja čiji su rezultati prikazani grafovima 7.1 i 7.5. Očekivano je da bi korišćenje različitih metaparametara, uključujući i duže vreme treniranja, dalo različite, odnosno bolje, rezultate. Unapređenje bi bilo znatno olakšano pristupom odgovarajućem hardveru kao što su grafičke karte sa CUDA jezgrima. Na ovaj način, promene vrednosti metaparametara bile bi lakše ispitane, što bi dovelo do mogućnosti finih podešavanja i, očekuje se, boljih rezultata.

Sem *DQN*, postoji još algoritama dubokog učenja potkrepljivanjem, od kojih su neki dali bolje rezultate na Atari 2600 emulatoru. Neki od tih algoritama su *DDQN* [20] (eng. *dueling DQN*), varijacija *DQN* algoritma i, možda najuspešniji u ovom zadatku, algoritam *A3C* [14] (eng. *Asynchronous Advantage Actor-Critic*). Koristeći algoritam *A3C*, kreiran je agent koji je u stanju da igra video igru Doom, koja je dosta kompleksnija od igara sa Atari 2600 konzole.

[1] https://github.com/NikolaMilev/DQN

Dakle, pored podešavanja metaparametara, u budućnosti treba istražiti (možda čak i kreirati) nove metode dubokog učenja potkrepljivanjem i ispitati njihovu uspešnost.

Literatura

[1] Arthur Samuel: Pioneer in machine learning. http://infolab.stanford.edu/pub/voy/museum/samuel.html. Datum poslednjeg pristupa: 10.8.2018.

[2] Keras programska biblioteka. https://keras.io/.

[3] NumPy programska biblioteka. http://www.numpy.org/.

[4] OpenAI Gym programska biblioteka. https://gym.openai.com/.

[5] Python programski jezik. https://www.python.org/.

[6] TensorFlow programska biblioteka. https://www.tensorflow.org/.

[7] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Overview of mini--batch gradient descent.

[8] Richard S. Sutton i Andrew G. Barto. *Reinforcement Learning: An Introduction*, pages 63–64. MIT Press, 2017.

[9] Richard S. Sutton i Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2017.

[10] Mladen Nikolić i Anđelka Zečević. *Mašinsko učenje*, page 118. Matematički fakultet, Univerzitet u Beogradu, 2018. http://ml.matf.bg.ac.rs/readings/ml.pdf.

[11] Mladen Nikolić i Anđelka Zečević. *Mašinsko učenje*. Matematički fakultet, Univerzitet u Beogradu, 2018. http://ml.matf.bg.ac.rs/readings/ml.pdf.

[12] Predrag Janičić i Mladen Nikolić. *Veštačka inteligencija*. Matematički fakultet, Univerzitet u Beogradu, 2018. http://poincare.matf.bg.ac.rs/~janicic//courses/vi.pdf.

[13] Yoshua Bengio i Aaron Courville Ian Goodfellow. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[14] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv.org*, februar 2016. https://arxiv.org/abs/1602.01783.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv.org*, decembar 2013. https://arxiv.org/abs/1312.5602.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, februar 2015. https://www.nature.com/articles/nature14236.

[17] Desanka Radunović. *Numeričke metode*, pages 160–162. Akademska misao, 2004.

[18] Arthur L. Samuel. Some studies in machine learning using the game of checkers. 1959. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.2254&rep=rep1&type=pdf.

[19] John N. Tsitsiklis, Member, IEEE, and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 42(5), maj 1997. http://www.mit.edu/~jnt/Papers/J063-97-bvr-td.pdf.

[20] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv.org*, novembar 2015. https://arxiv.org/abs/1511.06581.