Sinteza funkcije nagrade iz opisa u prirodnom jeziku transformerom

Corazza, Jan

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Jan Corazza

Sinteza funkcije nagrade iz opisa u prirodnom jeziku transformerom

Diplomski rad

Voditelj rada:
Luka Grubišić

Zagreb, 2023

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Novi smjerovi istraživanja u učenju s potporom usmjereni su na algoritme koji eliminiraju pretpostavku da funkcija nagrade zadovoljava Markovljevo svojstvo. Potreba za takvim funkcijama nagrade prirodno se javlja u situacijama u kojima agent mora ostvariti kompleksne nizove koraka, odnosno u kojima nagrada ne ovisi samo o trenutnom stanju i potezu agenta, nego o proizvoljno dugom (ali konačnom) nizu stanja i poteza. Funkcije nagrade koje ne zadovoljavaju Markovljevo svojstvo mogu obuhvatiti probleme u kojima je nagrada rijetka i ovisi o nizu događaja u vremenu, no trebaju ih zadati inženjeri u obliku konačnih automata, LTL formula, ili regularnih izraza.

U području obrade prirodnog jezika najveće pomake donijela je nova arhitektura transformera koja omogućuje da se znanje iz općenite domene modeliranja jezika prenese na uže domene specifičnih zadataka. Konkretno, budući modeli dubokog učenja mogu imati koristi od polaznih parametara koji su pronađeni u *offline* fazi računanja s veoma visokim uloženim procesorskim vremenom na bogatom skupu podataka za učenje.

Diplomski rad će razviti autoregresivni model koristeći polazne parametre modela *Generative Pre-trained Transformer 2* (GPT-2) [10], s ciljem olakšavanja uporabe formalizama u učenju s potporom. Razvijeni jezični model će taj cilj ostvariti eliminiranjem potrebe da inženjeri ručno zadaju formalizme koji obuhvaćaju funkciju nagrade. Umjesto toga, uz pomoć jezičnog modela i razvijenog grafičkog sučelja, omogućit će se sljedeći niz koraka.

1. Korisnik na Engleskom jeziku opisuje zadatak za agenta u okruženju, na primjer *"patrol the forest and the factory, but avoid traps"*.

2. Poziva se jezični model koji prevodi korisnikov opis u formalizam koji obuhvaća funkciju nagrade. Taj formalizam ima semantiku koja odgovara korisnikovom unosu, te je posebno namijenjen obuhvaćanju funkcija nagrade koje ne zadovoljavaju Markovljevo svojstvo.

3. Trenira se agent za obavljanje zadatka kojeg je korisnik opisao u koraku 1. Za treniranje se koristi poseban algoritam za učenje s potporom, namijenjen vrsti formalizma koji je pronađen u koraku 2.

4. Rezultati se korisniku demonstriraju vizualno na malom broju epizoda.

Novi parametri jezičnog modela bit će pronađeni putem dotreniranja modela GPT-2 na skupu podataka koji sadrži uređene parove opisa semantike funkcije nagrade (to jest zadatka za agenta), i zapisa formalizma koji obuhvaća tu semantiku.

Također, prezentirat će se grafičko sučelje koje korisniku prikazuje rezultate njegovog upita u prirodnom jeziku kroz djelovanje agenta koji obavlja zadatak. Sav kod koji je potreban za reprodukciju rezultata u ovom radu i pokretanje grafičkog korisničkog sučelja nalazi se na povezanom Git repozitoriju.[1]

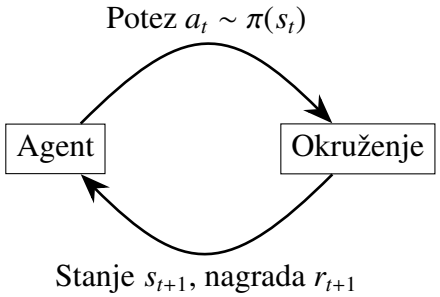
[1] `https://github.com/corazza/text2task`

Poglavlje 1

Učenje s potporom

Učenje s potporom (UP, engl. *reinforcement learning, RL*) je grana strojnog učenja koja se bavi problemom učenja optimalnog ponašanja[1] agenta u nekom okruženju na temelju povratne informacije–nagrade. Malo preciznije, cilj učenja s potporom je razvoj algoritama koji iterativno, kroz interakciju agenta i okruženja, pronalaze ponašanja koja maksimiziraju očekivanu buduću korisnost agenta uvjetovanu funkcijom nagrade.

Na slici 1.1 nalazi se ilustracija procesa učenja s potporom, odnosno interakcije agenta i okruženja u diskretnim vremenskim koracima.



Slika 1.1: U koraku t, agent bira potez (radnju, akciju)[2] iz zadanog skupa dozvoljenih poteza uzorkovanjem ponašanja π, a okruženje odgovara prijelazom u novo stanje i povratnom informacijom (nagradom).

U kontekstu učenja s potporom pojmovi agenta i okruženja mogu se odnositi na apstraktne stvari, jer je UP primjenjiv, na primjer, i na probleme diskretne optimizacije u kojima je teže jasno odrediti granicu između agenta i njegovog okruženja [7]. Naravno, kod mnogih drugih problema kojima se može pristupiti putem učenja s potporom, poput

[1] U literaturi na Engleskom ponašanje se najčešće naziva *policy* (politika).

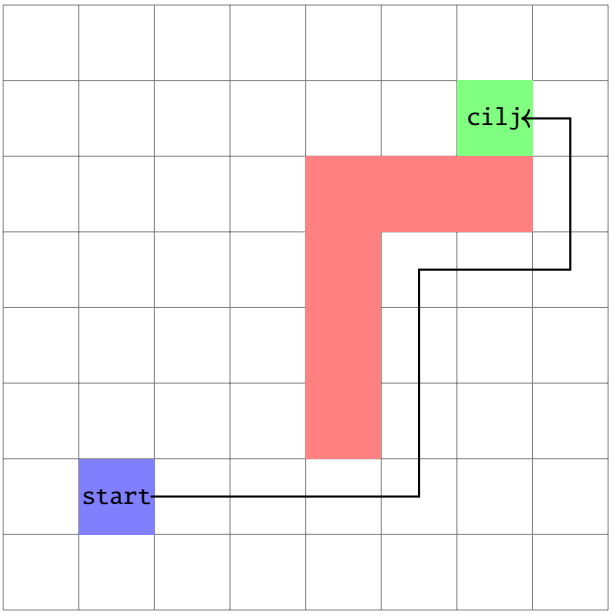
[2] Oznaka *a* dolazi iz engl. *action*.

igara s jednim igračem ili upravljanja robotskih manipulatora [1], ta distinkcija je očita. Tema diplomskog rada nije praktična primjena algoritama za učenje s potporom na konkretan problem, već proširivanje klase problema na koje se UP može primijeniti (to jest olakšavanje tog proširenja korištenjem drugih tehnika strojnog učenja). Shodno tome koristit će se jednostavne realizacije pojmova agenta, okruženja, i ponašanja.

U narednoj sekciji 1.1 slijedi detaljniji opis relevantnih pojmova, popraćen primjerom formalnog problema koji će poslužiti za ilustraciju primjene UP, i definicijama koje čine zajednički jezik za primjene i razvoj algoritama za UP.

1.1 Markovljev proces odlučivanja

Kao osnovnu ilustraciju problema u UP može se uzeti primjer zadatka koji se odvija u *Gridworldu*, to jest u omeđenom, rešetkastom okruženju u kojemu je stanje agenta zadano uređenim parom x i y koordinata.



Slika 1.2: 8x8 *Gridworld* okruženje. Agent počinje na plavom kvadratu označenom sa `start`, a zadatak mu je doći do zelenog kvadrata označenog s `cilj`. Dozvoljeni potezi su pokreti u četiri smjera: gore, dolje, lijevo, desno; te čekanje (preskakanje koraka). Crveni kvadrati predstavljaju zidove, preko kojih se agent ne može kretati. Crnom linijom koja završava strelicom označena je jedna putanja koja zadovoljava zadatak.

Primjer 1.1.1 (*Gridworld*: dolaženje do odredišta)**.** *Zadatak agenta: krećući od polja gridworlda označenog sa* `start`*, pronađi polje koje je označeno sa* `cilj`*. Detaljnije na slici 1.2.*

Na početku učenja agent nema pristup informacijama o Gridworldu u kojemu se nalazi (izuzev stanja (x, y)*), o efektima svojih poteza, ili drugim specifičnostima zadatka. Mora pronaći dobro ponašanje samo na temelju pokušaja i pogreške, te rezultantne povratne informacije (nagrade). Funkcija nagrade preslikava uređene trojke* (s, a, s') *(gdje je* s *početno stanje, a potez,* i s' *rezultantno stanje) u realne brojeve (nagrade), a ima vrijednost* 1 *ako se potezom* a *iz stanja* s *prelazi u stanje* s' *takvo da je ono označeno s* `cilj`*. U svim drugim slučajevima nagrada je* 0*.*

U učenju s potporom, agent uči ponašanja kroz iterativnu interakciju s okruženjem. Interakcija je često epizodna, odnosno učenje se odvija u nekom konačnom broju koraka– do ispunjenja zadatka, ili do unaprijed određenog maksimalnog broja koraka. Na početku interakcije, odnosno epizode, agent se nalazi u istaknutom početnom stanju (nekada je početno stanje uzorkovano iz zadane distribucije). Interakcija se odvija tako da agent donese odluku o potezu uzorkovanjem distribucije koja je funkcija stanja. Okruženje odgovara prijelazom u sljedeće stanje, te agentu daje povratni signal to jest nagradu. Nagrada je realni broj koji odgovara mjeri uspješnosti obavljanja zadatka. Agentu je u interesu maksimizirati očekivanu sumu nagrade tijekom interakcije. Točnije, suma uključuje eksponencijalno prigušenje kako bi se osigurala konvergencija u slučaju neograničenog broja koraka, i prilagodilo ekonomskoj logici da su nagrade u budućnosti manje vrijedne nego neposredne nagrade.

Definicija 1.1.2 formalizira okruženje kroz pojam Markovljevog procesa odlučivanja. Standardno se MPO definira uz pripadnu funkciju nagrade. No u ovom radu je ta definicija izdvojena, jer se u sekciji 2.2 uvodi pojam koji zamijenjuje standardnu funkciju nagrade.

Definicija 1.1.2 (Markovljev proces odlučivanja (MPO))**.** *Markovljev proces odlučivanja je uređena četvorka* $\mathcal{M} = (S, s_I, A, p)$ *koja se sastoji od skupa stanja* S*, istaknutog početnog stanja* s_I*, skupa poteza* A*, te prijelazne distribucije*[3] $p(s' \mid s, a)$ *koja određuje vjerojatnost prijelaza MPO iz stanja* $s \in S$ *u stanje* $s' \in S$ *pri potezu* $a \in A$*. Kardinalnost MPO definira se kao kardinalnost njegovog skupa stanja, to jest* $|\mathcal{M}| = |S|$*.*

U ovom radu relevantni MPO su konačni (pritom se misli i na konačnost skupa poteza A). Također, u primjerima je tranzicijska funkcija p zapravo deterministička, to jest par početnog stanja i poteza (s, a) jedinstveno određuje sljedeće stanje s' u koje okruženje prelazi. Definicija 1.1.3 formalizira pojam ponašanja agenta u okruženju.

[3] Funkcija prijelaza p se u Engleskoj literaturi često naziva *environment dynamics* (dinamika okruženja). Ovaj rad podrazumijeva konačnost skupova S i A.

Definicija 1.1.3 (Ponašanje $\pi(a \mid s)$)**.** *Ponašanje (engl.* policy*) je funkcija* $\pi : S \times A \to [0, 1]$ *koja preslikava stanja MPO* $s \in S$ *u vjerojatnosne distribucije nad potezima* $a \in A$*. Nerijetko se promatraju i ponašanja u obliku* $\pi : S \to A$*, ona se zovu deterministička ponašanja.*

Zadatak za agenta standardno se zadaje preko funkcije nagrade, koju formalizira definicija 1.1.4.

Definicija 1.1.4 (Funkcija nagrade $R(s, a, s')$)**.** *Funkcija nagrade (engl.* reward function*) je funkcija* $R : S \times A \times S \to \mathbb{R}$ *koja preslikava uređene trojke stanja MPO* $s \in S$*, poteza* $a \in A$*, i rezultantnog stanja* $s' \in S$ *u realne brojeve koji predstavljaju korisnost, odnosno povratnu informaciju o odabiru akcije* a *ako se agent nalazi u stanju* s *te je po* p *prešao u stanje* s'*. Ako je funkcija prijelaza* p *deterministička, onda se na nagradu može gledati kao na funkciju* $R(s, a)$*. Nekada se definira konačan skup mogućih nagrada* $\mathcal{R} \subset \mathbb{R}$*, i funkcija nagrade kao preslikavanje* s (s, a, s') *u distribucije nad skupom* $\mathcal{R}$*.*

Bitno je istaknuti da definicija funkcije nagrade kao u 1.1.4 zadovoljava *Markovljevo svojstvo*. Tokom učenja, nagrada ovisi samo o uređenoj trojci (s, a, s'), to jest o trenutnom stanju, izboru poteza, i rezultatnom stanju–a ne o cjelokupnoj trajektoriji. Markovljevo svojstvo je bitna pretpostavka u dokazima točnosti i konvergencije nekih algoritama UP. Na primjeru 1.1.1 to znači da R ne smije uzeti u obzir cijeli put agenta od ćelije `start` do ćelije `cilj`, već samo neposredno nagraditi dolazak u `cilj`.

Učenje u MPO odvija se u diskretiziranim vremenskim koracima $t = 0, 1, 2, \ldots$. Agent će u svakom koraku t u stanju s_t konzultirati (trenutno) ponašanje $\pi_t(\cdot \mid s_t)$, te donijeti odluku $a_t \sim \pi_t(s_t)$. Shodno tome MPO prelazi u stanje $s_{t+1} \sim p(s_t, a_t)$, te se taj proces nastavlja (uz mogućnost promjene ponašanja radi učenja, koje je iz tog razloga također indeksirano koracima). Početno ponašanje odabire se po dogovoru (na primjer, uniformno).

MPO, ponašanje, i funkcija nagrade zadaju niz slučajnih varijabli $s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots$ koji se zove *trajektorija*, i za kojeg vrijedi $s_0 = s_\mathcal{I}$, $s_{i+1} \sim p(s_i, a_i)$, $r_{i+1} = R(s_i, a_i, s_{i+1})$, te $a_i \sim \pi(s_i)$.

Sada možemo precizno odrediti cilj UP algoritama kao maksimizaciju očekivanja u 1.1 po ponašanjima π. Broj $\gamma \in (0, 1)$ zove se diskontni faktor[4] i čini hiperparametar algoritama. On je mjera opadanja vrijednosti budućih nagrada (broj blizu 0 označava brzo opadanje vrijednosti nagrada u budućnosti, a broj blizu 1 sporo).

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\middle|\; s_{t+1} \sim p(s_t, a_t), a_t \sim \pi(s_t), s_0 = s_\mathcal{I}\right] \tag{1.1}$$

[4] engl. *discount factor*

Definicije 1.1.2– 1.1.4 daju osnovne pojmove koji su dovoljni za postavljanje problema u UP. No za proučavanje algoritama za učenje s potporom, korisni su također i pojmovi Q-funkcija i optimalnih ponašanja, koje formaliziraju definicije 1.1.5 i 1.1.6. Izraz $\mathbb{E}_\pi$ koristi se kao skraćeni zapis informacije o distribucijama poteza a_t, dok se distribucija stanja podvrgava još i dinamici p.

Definicija 1.1.5 (Q-funkcija i Bellmanova jednadžba za Q-funkcije)**.** *Q-funkcija za ponašanje π preslikava parove stanja i poteza (s, a) u očekivanu korisnost ako agent u stanju s odabere potez a, a nakon toga nastavi odabirati poteze prema π.*

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a\right] \tag{1.2}$$

Po formuli potpune vjerojatnosti, iz ove definicije dolazimo do Bellmanove jednadžbe za Q-funkcije.

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \;\middle|\; s_0 = s, a_0 = a\right] \tag{1.3}$$

$$= \mathbb{E}_\pi\left[R(s_0, a_0, s') + \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \;\middle|\; s_0 = s, a_0 = a, s' = s_0 \sim p(s_1, a_0)\right] \tag{1.4}$$

$$= \sum_{s' \in S} p(s' \mid s_0, a_0)\left(R(s_0, a_0, s') + \gamma \sum_{a' \in A} \pi(a' \mid s') \cdot Q^\pi(s', a')\right) \tag{1.5}$$

U Bellmanovoj jednadžbi krije se uvjet optimalnosti Q-funkcija, kao i princip algoritama UP da se novi procjenitelji ($Q^\pi(s,a)$) računaju iz postojećih ($Q^\pi(s',a')$).

Definicija 1.1.6 (Optimalno ponašanje $\pi^*(s,a)$ i optimalna Q-funkcija Q^*)**.** *Ponašanje π^* i Q-funkcija Q^* su* optimalni *ako zadovoljavaju naredne jednadžbe (Bellmanova jednadžba optimalnosti).*

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') \;\middle|\; s' \sim p(s,a), r = R(s,a,s')\right] \tag{1.6}$$

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a) \tag{1.7}$$

Iz formule 1.7 može se vidjeti da je optimalno ponašanje određeno optimalnom Q-funkcijom. Spomenuto je da algoritmi učenja s potporom računaju niz ponašanja. Zapravo, radi se o isprepletenom procesu s dva glavna momenta.

1. Agent prikuplja iskustvo (informaciju o vrijednosti) ponašajući se po π. To iskustvo se koristi da bi se izračunao procjenitelj funkcije Q^π.

2. Bolji procjenitelj starog ponašanja koristi se da bi se izvelo bolje ponašanje π'.

Neka je π neko početno ponašanje. Ako kroz interakciju, odnosno uzorkovanje MPO, izračunamo procjenitelj Q^π, intuitivno je kako izračunati bolje (u smislu veće očekivane koristi) ponašanje π'.

$$\pi'(a \mid s) = \begin{cases} 1 & \text{ako } a = \arg\max_{a' \in A} Q^\pi(s, a') \\ 0 & \text{inače} \end{cases}$$

Ponašanje π' zove se *pohlepno ponašanje* u odnosu na Q^π. Odnos „biti bolje" može se formalizirati relacijom. $\pi' \geq \pi$ ako za svako stanje $s \in S$ ponašanje π' ostvaruje veću ili jednaku korisnost u očekivanju od π.

Sada teorem 1.1.7 daje opravdanje za dvojni proces poboljšavanja ponašanja i procjenitelja. Radi jednostavnosti neka su π i π' deterministička ponašanja, to jest $\pi(s) \in A$.

Teorem 1.1.7 (Teorem o poboljšavanju ponašanja)**.** *Neka su π i π' ponašanja takva da za svaki $s \in S$ vrijedi $Q^\pi(s, \pi'(s)) \geq \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \;\middle|\; s_0 = s, a_t = \pi(s_t)\right]$, odnosno da je u stanju s bolje odabrati potez po π' i nakon toga pratiti ponašanje π, nego odabrati potez po ponšanju π i također nastaviti pratiti π.*

Tada vrijedi $\pi' \geq \pi$. Drugim riječima, ako je u jednom koraku bolje odabrati potez po π' u svakom stanju, onda je bolje u svakom koraku odabrati potez po π', to jest smijemo „zaboraviti" staro ponašanje π u potpunosti.

Dokaz. Definirajmo pomoćni niz slučajnih varijabli za nagrade i funkciju vrijednosti stanja.

$$R_t = R(s_t, a_t, s_{t+1})$$
$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t \;\middle|\; s_0 = s, a_t = \pi(s_t)\right]$$

U literaturi se najčešće navodi sljedeći niz nejednakosti kao dokaz ovog teorema.

$$
\begin{aligned}
V^{\pi}(s) &\leq Q^{\pi}(s, \pi'(s)) && (1.8)\\
&= \mathbb{E}_{\pi}\left[R_0 + \sum_{t=1}^{\infty} \gamma^t R_t \;\middle|\; s_0 = s, a_0 = \pi'(s_0), a_{t\geq 1} = \pi(s_t)\right] && (1.9)\\
&= \mathbb{E}_{\pi'}\left[R_0 + \gamma V^{\pi}(s_1) \;\middle|\; s_0 = s, a_0 = \pi'(s_0)\right] && (1.10)\\
&\leq \mathbb{E}_{\pi'}\left[R_0 + \gamma Q^{\pi}(s_1, \pi'(s_1)) \;\middle|\; s_0 = s, a_0 = \pi'(s_t), a_1 = \pi'(s_t)\right] && (1.11)\\
&= \cdots && (1.12)\\
&\leq \mathbb{E}_{\pi'}\left[R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots \;\middle|\; s_0 = s, a_t = \pi'(s_t)\right] && (1.13)\\
&= \mathbb{E}_{\pi'}\left[\sum_{t=0}^{\infty} \gamma^t R_t \;\middle|\; s_0 = s, a_t = \pi'(s_t)\right] && (1.14)\\
&= V^{\pi'}(s) && (1.15)
\end{aligned}
$$

U nejednakostima 1.8 i 1.11 koristi se pretpostavka teorema. U jednakosti 1.9 koristi se definicija Q-funkcija, s time da je naznačeno da se prvi potez bira po ponašanju π', dok su ostali potezi odabrani po π (prema oznaci $\mathbb{E}_{\pi}$). Za jednakost 1.10 koristi se definicija funkcije $V^{\pi}(s)$ s početka iskaza teorema, s time da je iz reda u 1.9 izlučen faktor γ.

U 1.12–1.13 poziva se na beskonačnu iteraciju spomenutih nejednakosti. Iako takav postupak daje slikovit uvid u intuiciju iza teorema, nije formalan dokaz. Niz nejednakosti trebalo bi zamijeniti limesom i primjenom Bellmanovog operatora, a više o tome (puni formalni dokaz) može se vidjeti u [5]. □

Teorem 1.1.7 odgovara intuitivnoj činjenici da su ponašanja koja su bolja lokalno zapravo bolja i globalno. Kada pronađemo ponašanje koje bolje odabire poteze u ograničenom vremenskom horizontu (dovoljan je 1 korak, ali iz svih stanja), onda je bolje u potpunosti prijeći na njega (u svim naknadnim koracima). To je relevantno za algoritme koji poboljšavaju ponašanja prelaskom na pohlepna ponašanja za nove procjenitelje Q-funkcija, to jest za one koji se podvrgavaju dvojnom procesu opisanom nakon definicije 1.1.6.

1.2 Q-učenje

Najpoznatiji algoritam za učenje s potporom zove se Q-učenje (engl. *Q-learning*). Q-učenje je u osnovnoj formulaciji primjenjivo samo na konačne MPO (to jest one za koje vrijedi da je $|\mathcal{M}| = |S| = n \in \mathbb{N}$), no postoje varijante poput algoritma DQN (*Deep Q-Network*) koje su inspirirane Q-učenjem no namijenjene MPO s neprebrojivo mnogo stanja.

Konačnost, to jest u praksi malen broj stanja MPO, omogućuje zapisivanje podataka o korisnosti (to jest Q-vrijednosti) stanja i poteza direktno u tablicu u memoriji, odnosno bez aproksimiranja Q-vrijednosti kao u DQN-u (ili aproksimiranja poteza u *policy gradijent* metodama). Kako fokus diplomskog rada nije na razvijanju ili analizi algoritama za UP samih po sebi, te su svi tu relevantni primjeri MPO konačni i maleni, dovoljno će biti promatrati Q-učenje u tabličnom obliku.

Q-učenje osim diskontnog faktora γ ima hiperparametar $\epsilon \in [0, 1]$, realan broj koji parametrizira omjer istraživanja naspram iskorištavanja. Preciznije, ϵ je vjerojatnost da će u pojedinom koraku agent odabrati *nasumičan* (uniformno) potez $a \in A$. Za ϵ blizu 1, Q-učenje u potpunosti prioritizira istraživanje (odnosno svaki odabir poteza blizu uniformne distribucije nad A), dok je za ϵ blizu 0 svaki odabir vođen trenutnom procjenom Q-funkcije. Kod MPO koji modeliraju *Gridworld*, tipična vrijednost hiperparametra ϵ je 0.1, a γ od 0.9 do 0.99.

Definicija 1.2.1 (ϵ-pohlepno ponašanje π)**.** *Neka je $Q(s, a)$ Q-funkcija i $\epsilon \in [0, 1]$. Tada ϵ-pohlepno ponašanje π za Q definiramo kao:*

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{ako je } a = \arg\max_{a' \in A} Q(s, a'), \\ \frac{\epsilon}{|A|} & \text{inače.} \end{cases}$$

Drugim riječima, π odabire najbolji potez (prema Q-tablici) s vjerojatnosti $1 - \epsilon + \frac{\epsilon}{|A|}$, a sve ostale poteze s jednakom vjerojatnosti $\frac{\epsilon}{|A|}$. Radi dobre definiranosti $\arg\max_{a' \in A}$ pretpostavlja se proizvoljni fiksni uređaj na A.

Zadnji hiperparametar je stopa učenja (*learning rate*) $\alpha > 0$, najčešće malen pozitivan realan broj, koji regulira brzinu promjene zapisa Q-vrijednosti u tablici uzrokovanu novim informacijama o vrijednosti.

Uvjet kraja epizode se može ostvariti preko istaknutih završnih stanja ili ograničavanja broja koraka u jednoj epizodi. Uvjet zaustavljanja ovisi o dogovoru. U praksi je često odabran maksimalan broj koraka.

Rezultat Q-učenja je alternirajući niz Q-tablica (to jest procjenitelja Q-funkcija) i ponašanja. Agent se u koraku t ponaša po π_t, i na bazi dobivene nagrade računa se sljedeći procjenitelj Q-funkcije. Onda se iz te osvježene procjene konstruira sljedeće ponašanje π_{t+1}. Teorem 1.1.7 osigurava $\pi_{t+1} \geq \pi_t$. Može se pokazati da uz uvjet *Markovljevosti R* i ϵ-pohlepnog ponašanja za istraživanje, niz $Q_0, \pi_0, Q_1, \pi_1, \ldots$ kovergira prema Q^* odnosno π^*.

Glavni dio algoritma Q-učenja dan je pravilom 1.16, kao što se vidi na linij 7 algoritma 1.

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{1.16}$$

Algorithm 1 Q-učenje

Inicijaliziraj $Q(s, a)$ proizvoljno za sve $s \in S, a \in A$
repeat
    Inicijaliziraj s na s_I
    repeat
        Odaberi potez a u s koristeći ponašanje izvedeno iz Q (ϵ-pohlepno)
        Izvrši a, dobij nagradu r i novo stanje s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$
    until kraj epizode
until ispunjen uvjet zaustavljanja

Bitna značajka Q-učenja je razlika u *učenom* ponašanju i *aktualnom*, to jest onom koje se koristi za istraživanje. Naime, Q-učenje uči optimalno ponašanje (zadano preko Q-tablice), no poteze koje agent obavlja uvijek uzorkuje iz distribucije koju daje ϵ-pohlepno ponašanje. Nužni uvjet konvergencije Q-učenja ka optimalnoj Q-funkciji Q^* je da je aktualno ponašanje ϵ-pohlepno, i da funkcija nagrade R zadovoljava Markovljevo svojstvo.

Markovljevo svojstvo R predstavlja veliku mana Q-učenja. Ono onemogućuje formuliranje zadataka koji ovise o cijeloj povijesti stanja i poteza agenta u MPO. U praksi se prirodno pojavljuju zadaci koji ovise o cijelom trajektoriju u MPO, odnosno od agenta zahtijevaju da postigne određene redoslijede događaja. Oblikovanje algoritama koji mogu pronaći optimalno ponašanje i za takvu širu familiju funkcija nagrade je aktivno područje istraživanja u UP. Poglavlje 2 bavi se obećavajućim smjerom u suočavanju s problemom funkcija nagrade koje nemaju Markovljevo svojstvo.

Poglavlje 2

Strojevi nagrade

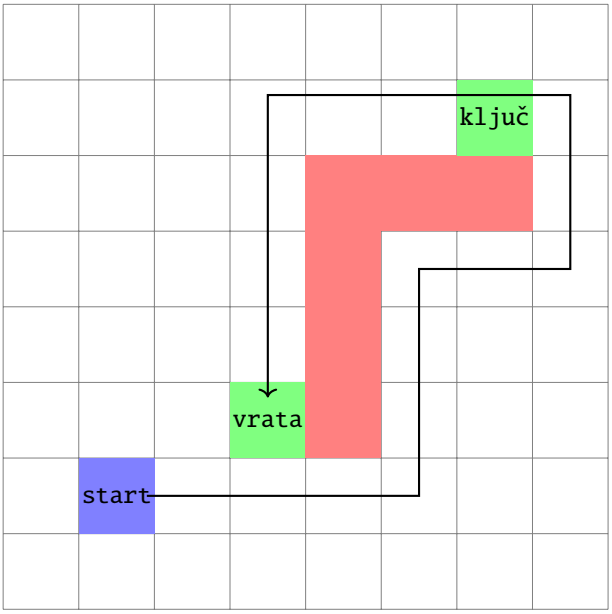
U primjeru 1.1.1, zadatak za agenta je bio jednostavan: posjetiti određeno stanje. Markovljevu funkciju nagrade za takav problem je jednostavno isprogramirati. Neka je P konačan skup propozicionalnih varijabli, u ovom slučaju $P = \{\texttt{start}, \texttt{cilj}\}$ (standardno se propozicionalne varijable označavaju s $P_{i\in\mathbb{N}}$, no u skupu P ih ima konačno mnogo, i svakoj je pridružena jedinstvena oznaka). Pretpostavimo da nam je na raspolaganju funkcija $L : S \times A \times S \to 2^P$ koja za dano rezultantno stanje (početno stanje i potez su u ovom primjeru irelevantni) može odgovoriti na pitanje nalazi li se agent u tom stanju na cilju. Odnosno takva da vrijedi $L(s, a, s') = \{\texttt{cilj}\}$ ako i samo ako je s' stanje koje odgovara cilju, a $L(s, a, s') = \emptyset$ inače. U tom slučaju, možemo definirati funkciju nagrade kao u 2.1.

$$R(s, a, s') = \begin{cases} 1 & \text{ako } \texttt{cilj} \in L(s, a, s'), \\ 0 & \text{inače.} \end{cases} \tag{2.1}$$

Funkcija $L(s, a, s')$ zove se funkcija označavanja (engl. *labeling function*). L se može shvatiti kao valuacija nad prijelazima MPO jer određuje koje su od konačno mnogo propozicionalnih varijabli iz skupa P istinite ako agent krene u stanju s, odabere potez a, i završi u stanju s'.

Q-učenje s ovako zadanom funkcijom nagrade konvergira u optimalnu Q-funkciju (jer funkcija nagrade 2.1 zadovoljava Markovljevo svojstvo). No što ako, uz veći P i kompliciraniju L, želimo definirati i kompliciranije funkcije nagrade–koje ovise o *nizovima* oznaka (događaja) u MPO, a ne oznakama samo neposrednih stanja? Primjer 2.0.1 pokazuje kako je to sasvim prirodan zahtjev.

Primjer 2.0.1 (*Gridworld*: otvaranje vrata)**.** *Zadatak agenta: krećući od polja gridworlda označenog sa* `start`*, posjeti polje koje je označeno s* `vrata`*, ali prije toga posjetiti polje označeno s* `ključ`*. Detaljnije na slici 2.1.*



Slika 2.1: Novo 8x8 *Gridworld* okruženje s oznakama. Za razliku od primjera 1.1.1, gdje je zadatak za agenta doći na ćeliju `start`, u ovom okruženju zadatak je prvo skupiti ključ (dolaskom u ćeliju `ključ`), pa onda doći do ćelije označene s `vrata`. Kao i ranije, agent se ne može kretati preko crvenih ćelija (zidova).

Problem iz primjera 2.0.1 je bitno drugačiji od onog iz primjera 1.1.1. Ne postoji funkcija nagrade koja može obuhvatiti taj zadatak, odnosno nagraditi samo agenta koji je posjetio vrata nakon što je posjetio ključ, a da se ujedno podvrgava familiji funkcija koje su definirane u 1.1.4. Razlog je to što događaj posjećivanja vrata može biti proizvoljno daleko (u broju koraka) od događaja posjećivanja ključa, pa svaka funkcija nagrade koja vjerno obuhvaća taj problem nužno ne zadovoljava Markovljevo svojstvo. To vrijedi čak i ako uzmemo u obzir širu klasu funkcija nagrade koje se odnose na zadani konačni broj prethodnih koraka. Posljedično, ne možemo koristiti Q-učenje kako bi riješili ovaj problem, iako se na prvi pogled ne čini puno težim od onog sa slike 1.2.

Tema narednih sekcija je proširenje pojmova iz poglavlja 1. To uključuje formalizaciju pojma oznake (događaja), formalizaciju šire familije funkcija nagrade, i novi algoritam UP koji se može nositi s njima.

2.1 Označeni MPO i rijetke nagrade

U narednim primjerima MPO i problemima koji se navode koristit će se funkcija označavanja L. Radi potpunosti slijedi definicija 2.1.1, koja obuhvaća MPO i pripadnu funkciju označavanja.

Definicija 2.1.1 (Označeni MPO (O-MPO))**.** *Označeni MPO je uređena petorka* $\mathcal{M} = (S, s_I, A, p, L)$*, gdje su* S*,* s_I*,* A*, i* p *kao u definiciji 1.1.2,* $L : S \times A \times S \to 2^P$ *je funkcija označavanja, i* P *je konačan skup propozicionalnih varijabli koje zovemo oznake (ili događaji).*

Uz ranije definirani niz $s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_n, a_n, r_{n+1}$ javlja se i pripadni niz oznaka $\ell_0, \ell_1, \ldots, \ell_n$ i vrijedi $\ell_i = L(s_i, a_i, s_{i+1})$. Treba napomenuti da je svaki događaj ℓ_i podskup skupa propozicionalnih varijabli P. Razlog tome je da u pojedinom koraku i može vrijediti više propozicionalnih varijabli istovremeno (ili nijedna). Na primjer, kako L kao argument prima i potez (a ne samo stanja), funkcija L može se zadati tako da $a_i \in \ell_i$ ($\forall i \in \mathbb{N}$). Time se postiže mogućnost referiranja na odabrani potez kroz oznake.

Kao što je rečeno ranije, za zadatke od više koraka prirodno je promatrati širu klasu funkcija nagrade. Za nagrađivanje ostvarivanja određenih nizova događaja prirodno se nameće spomenuti niz oznaka. Tu ideju formalizira definicija 2.1.2.

Definicija 2.1.2 (Funkcija nagrade iz oznaka $R(\ell_0, \ell_1, \ldots, \ell_n)$)**.** *Funkcija nagrade iz oznaka je funkcija* $R : (2^P)^* \to \mathbb{R}$ *koja preslikava konačne nizove oznaka* $\ell_0, \ell_1, \ldots, \ell_n$ *u realne brojeve (nagrade). U ovom radu se podrazumijeva da je svaki O-MPO uparen s funkcijom nagrade iz oznaka.*

U praksi su funkcije nagrade iz oznaka često i *rijetke*, odnosno osim kršenja Markovljevog svojstva, posao algoritama za učenje je također otežan kašnjenjem između relevantnih događaja i nagrada. Drugim riječima, osim same vremenske međuzavisnosti poteza i nagrada, ta zavisnost je često odgođena i može joj se svjedočiti tek pri kraju niza događaja, što otežava učenje.

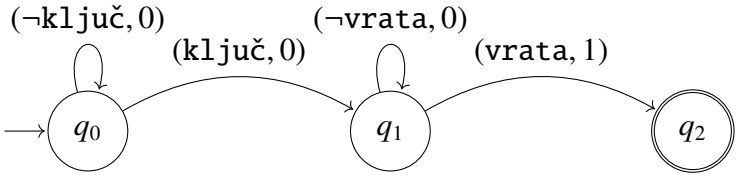
Za ovaj rad zapravo je relevantan samo podskup funkcija iz definicije 2.1.2: one koje se mogu zapisati u obliku Mealyjevog konačnog automata nad ulaznim alfabetom 2^P. Funkcijama nagrade s takvim zapisom se bavi sljedeća sekcija 2.2.

2.2 Stroj nagrade

Strojevi nagrade (SN, engl. *reward machine*), su vrsta konačnih automata (Mealyjevih strojeva) koji se koriste u učenju s potporom za zadavanje funkcija nagrade koje ovise o kompleksnom nizu stanja i poteza. Glavna ideja strojeva nagrade je da funkcioniraju kao oblik konačne memorije u MPO: potezi agenta i oznake okruženja mogu se pamtiti u stanjima stroja, a značenje zadatka može se zapisati putem njegove funkcije prijelaza.

Slika 2.2 ilustrira stroj nagrade za zadatak opisan u primjeru 2.0.1.

Definicija 2.2.1 formalizira ideju stroja nagrade.



Slika 2.2: Stroj nagrade za problem iz primjera 2.0.1. Stroj se sastoji od tri stanja q_0, q_1, i q_2, s time da je q_0 označeno kao početno, a q_2 kao završno (prihvaćajuće) stanje. Prijelazi su označeni formulama logike sudova nad skupom P i pripadnim iznosima nagrada.

Definicija 2.2.1 (Stroj nagrade (SN))**.** *Stroj nagrade je uređena šestorka* $\mathcal{A} = (Q, q_I, 2^P, O, \delta, \sigma)$*, gdje je* $Q = \{q_0, q_1, \ldots, q_n\}$ *konačan skup stanja, zajedno s jednim istaknutim stanjem* $q_I \in Q$ *koje je početno,* 2^P *ulazni alfabet (isti skup oznaka kao u O-MPO),* $O \subset \mathbb{R}$ *konačni izlazni alfabet (skup mogućih nagrada),* $\delta : Q \times 2^P \to Q$ *tranzicijska funkcija, i* $\sigma : Q \times 2^P \to O$ *funkcija izlaza. Skup prihvaćajućih stanja u literaturi najčešće nije dio definicije SN, jer u ovom slučaju služi kao pomoćni signal simulatoru da završi epizodu. Kardinalnost SN definira se kao kardinalnost njegovog skupa stanja, to jest* $|\mathcal{A}| = |Q|$.

Iako je tranzicijska funkcija zadana s obzirom na skup 2^P, odnosno preko parcijalnih interpretacija u P, na slici 2.2 prijelazi se pišu pomoću formula logike sudova. Naravno, oba zapisa su ekvivalentna ako poistovjetimo formulu i skup parcijalnih formulacija koje je ispunjuju.

Za niz oznaka $\ell_0, \ell_1, \ldots, \ell_n$ stroj nagrade $\mathcal{A}$ računa pripadni niz nagrada $r_0, r_1, \ldots, r_n$ (u zapisu $\mathcal{A}(\ell_0, \ell_1, \ldots, \ell_n) = (r_0, r_1, \ldots, r_n)$), ako postoji niz $(q_i)_i \subset Q$ takav da za svaki $i \in \{0, \ldots, n\}$ vrijedi $\sigma(q_i, \ell_i) = r_i$ i $q_{i+1} = \delta(q_i, \ell_i)$, uz $q_0 = q_I$. Na primjer, za SN sa slike 2.2 (u oznaci $\mathcal{A}$) vrijedi $\mathcal{A}(\{\}, \{\texttt{ključ}\}, \{\}, \{\texttt{vrata}\}) = (0, 0, 0, 1)$, ali $\mathcal{A}(\{\}, \{\}, \{\}, \{\texttt{vrata}\}) = (0, 0, 0, 0)$. Ako je zanimljiva samo zadnja nagrada u nizu, može se pisati $\mathcal{A}(\ell_0, \ell_1, \ldots, \ell_n) = r_n$.

2.3 Q-učenje za strojeve nagrade

Formalizam stroja nagrade dovoljan je da znatno proširimo klasu problema koje možemo specificirati. Nažalost, ako samo zamijenimo klasičnu funkciju nagrade u MPO sa strojem nagrade, Q-učenje više neće morati konvergirati k optimalnoj Q-funkciji (ili uopće). Potreban je novi algoritam, odnosno prilagodba Q-učenja: QRM[1], to jest Q-učenje za strojeve nagrade, da bismo ostvarili željene garancije točnosti i svojstva konvergencije algoritama za UP.

[1] Iz engleskog naziva *Q-learning for Reward Machines*.

Krenimo s početnim MPO $\mathcal{M}$ opremljenim strojem nagrade $\mathcal{A}$. Da bismo došli do algoritma QRM, dovoljno je zapitati se postoje li MPO $\mathcal{N}$ i *klasična* (Markovljeva) funkcija nagrade R, koji su u uskoj vezi s $\mathcal{M}$ i $\mathcal{A}$, ali na kojima možemo primijeniti Q-učenje? Vrsta veze koju tražimo je da postoji bijektivno preslikavanje ponašanja između $(\mathcal{M}, \mathcal{A})$ i $(\mathcal{N}, R)$ koje čuva očekivane vrijednosti i optimalnost Q-funkcija (parcijalni uređaj na ponašanjima). Ako imamo takvo preslikavanje onda svako ponašanje $\pi^{\mathcal{N}}$ u $\mathcal{N}$ možemo preslikati u ponašanje $\pi^{\mathcal{M}}$ na $\mathcal{M}$: ako je $\pi^{\mathcal{N}}$ bilo optimalno ponašanje u $\mathcal{N}$, onda će ponašanje $\pi^{\mathcal{M}}$ koje nam da preslikavanje biti optimalno u $\mathcal{M}$. U tom slučaju egzistencija $(\mathcal{N}, R)$ bi riješila problem: u njemu možemo koristiti Q-učenje, imamo garanciju konvergencije i točnosti, i znamo ponašanje vratiti nazad u $(\mathcal{M}, \mathcal{A})$. Pokazat će se da samo preslikavanje trivijalno postoji i ima tražena svojstva. Definicije 2.3.1 i 2.3.2 daju konstrukciju $(\mathcal{N}, R)$.

Definicija 2.3.1 (Produktni MPO)**.** *Neka je $\mathcal{M} = (S^{\mathcal{M}}, s_I^{\mathcal{M}}, A^{\mathcal{M}}, p^{\mathcal{M}}, L)$, označeni MPO i $\mathcal{A} = (Q, q_I, 2^P, O, \delta, \sigma)$ SN koji zadaje funkciju nagrade na $\mathcal{M}$.*

Produktni MPO za $(\mathcal{M}, \mathcal{A})$ je MPO $\mathcal{N} = (S^{\mathcal{N}}, s_I^{\mathcal{N}}, A^{\mathcal{N}}, p^{\mathcal{N}})$ takav da vrijedi sljedeće.

1. *$S^{\mathcal{N}} = S^{\mathcal{M}} \times Q$. Stanja MPO $\mathcal{N}$ su Kartezijev produkt stanja originalnog, označenog MPO $\mathcal{M}$, i stanja stroja nagrade $\mathcal{A}$. Svako stanje u $\mathcal{N}$ je oblika (s, q): sadrži informaciju i o stanju u $\mathcal{M}$ i o stanju u $\mathcal{A}$.*

2. *$s_I^{\mathcal{N}} = (s_I^{\mathcal{M}}, q_I)$. Produktni MPO počinje u stanju koje odgovara početnim stajnima i $\mathcal{M}$ i $\mathcal{A}$.*

3. *$A^{\mathcal{N}} = A^{\mathcal{M}}$. Agentu su u $\mathcal{N}$ dostupni isti potezi kao i u $\mathcal{M}$.*

4. *$p^{\mathcal{N}}((s, q), a, (s', q')) = p^{\mathcal{M}}(s, a, s') \cdot \mathbb{1}_{\delta(q,\ell)=q'}$, gdje je $\ell = L(s, a, s')$. Dinamika $\mathcal{N}$ u potpunosti prati dinamiku $\mathcal{M}$: stanja $\mathcal{A}$ su "prišivena" i prate niz $\ell_0\ell_1\ell_2\cdots$ po tranzicijskoj funkciji δ.*

Definicija 2.3.2 (Pripadna (produktna) Markovljeva funkcija nagrade)**.** *Neka su $\mathcal{M}$, $\mathcal{N}$, i $\mathcal{A}$ kao u definiciji 2.3.1. Pripadna klasična funkcija nagrade je $R : S^{\mathcal{N}} \times A^{\mathcal{N}} \times S^{\mathcal{N}} \to \mathbb{R}$, s pravilom $R((s, q), a, (s', q')) = \sigma(q, \ell)$, gdje je $\ell = L(s, a, s')$.*

Iz definicije 2.3.2 i četvrte točke definicije 2.3.1 slijedi da „ista" ponašanja na $\mathcal{M}$ i $\mathcal{N}$ ostvaruju istu očekivanu korisnost. Naravno, jasno je da ponašanje $\pi^{\mathcal{M}}$ u $\mathcal{M}$ mora imati pristup istim informacijama koje su relevantne za korisnost kao i ponašanje $\pi^{\mathcal{N}}$ u $\mathcal{N}$, odnosno da agent u $\mathcal{M}$ mora poznavati tranzicijsku funkciju δ stroja nagrade $\mathcal{A}$, kao i trenutno stanje u računanju $\mathcal{A}$. U tom pogledu, ako ponašanje u $\mathcal{M}$ zapišemo kao funkciju i stanja u $\mathcal{M}$ i stanja u $\mathcal{A}$, možemo koristiti ponašanje iz $\mathcal{N}$ direktno. Lema 2.3.3 tvrdi da su ta dva ponašanja (zapravo, isto ponašanje u dva različita konteksta) ekvivalentna u smislu ostvarivanja korisnosti.

Lema 2.3.3. *Neka su $\mathcal{M}$, $\mathcal{N}$, i $\mathcal{A}$ kao u definiciji 2.3.1, a $R : S^{\mathcal{N}} \times A^{\mathcal{N}} \times S^{\mathcal{N}} \to \mathbb{R}$ kao u definiciji 2.3.2. Neka je $\pi = \pi^{\mathcal{M}} = \pi^{\mathcal{N}}$ ponašanje. Tada za svako stanje $s \in S^{\mathcal{M}}$, $q \in Q$, i potez $a \in A$ vrijedi $Q^{\pi}_{\mathcal{M}}(((s,q),a)) = Q^{\pi}_{\mathcal{N}}(((s,q),a))$, odnosno Q-funkcije za π su jednake u oba MPO.*

Dokaz. Slijedi neposredno iz definicije, ako prihvatimo formulu 2.2 kao definiciju Q-funkcije za $(\mathcal{M}, \mathcal{A})$.

$$Q^{\pi}_{\mathcal{M}}(((s,q),a)) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \sigma(q_t, \ell_t)\right] \tag{2.2}$$

Samo treba funkciju R iz produktnog MPO-a zamijeniti njenom definicijom.

$$\begin{aligned} Q^{\pi}_{\mathcal{N}}(((s,q),a)) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R((s_t,q_t),a_t,(s_{t+1},q_{t+1})) \;\middle|\; \right. \\ &\qquad \left. (s_0,q_0) = (s,q), a_0 = a, (s_{t+1},q_{t+1}) \sim p((s_t,q_t),a_t)\right] \qquad (2.3) \\ &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \sigma(q_t,\ell_t) \;\middle|\; \right. \\ &\qquad \left. s_{t+1} \sim p(s_t,a_t), q_0 = q, q_{t+1} = \delta(q_t,\ell_t), \ell_t = L(s_t,a_t,s_{t+1})\right] \qquad (2.4) \\ &= Q^{\pi}_{\mathcal{M}}(((s,q),a)) \qquad (2.5) \end{aligned}$$

U jednakosti 2.4–2.5 koristi se formulacija Q-funkcije za označeni MPO u kojem je funkcija nagrade dana strojem nagrade s funkcijom izlaza σ i funkcijom prijelaza δ, što je operativna definicija koju koristi algoritam QRM. □

Iako lema 2.3.3 pokazuje da je u principu moguće naučiti optimalno ponašanje za dani MPO i SN, to je plaćeno drastičnim povećanjem broja stanja MPO: $|\mathcal{N}| = |\mathcal{A}| \times |\mathcal{M}|$. Zadnji trik u rukavu algoritma QRM, i ono što ga izdvaja od naivnog preslikavanja Q-učenja u kontekst strojeva nagrade, je iskorištavanje poznavanja strukture SN za ubrzavanje učenja putem rezanja prostora stanja $|\mathcal{A}|$ puta! Naime, za jedan prijelaz (s,a,r,s') u MPO $\mathcal{M}$, osim pripadnog prijelaza $((s,q),a,r,(s',q'))$ u produktnom MPO, zapravo možemo konstruirati $|\mathcal{A}| - 1$ dodatnih „virtualnih" prijelaza, po jedno za svaki $v \in Q \setminus \{q\}$. Formula 2.6 pokazuje kako, i u njoj je $v' = \delta(v,\ell)$ stanje $\mathcal{A}$ koje slijedi v po ulazu ℓ. Drugim riječima $v, v' \in Q$ su „virtualna" stanja, koja nisu zapravo bila ostvarena u danom koraku dosadašnjom trajektorijom, no i dalje imamo dovoljno informacija da osvježimo procjenu $Q((s,v),a)$ (za svaki $v \neq q$). Stanja s i s', te potez a, dolaze iz iskustva odnosno dijelovi su stvarnog trajektorija.

$$Q((s,v),a) \leftarrow Q((s,v),a) + \alpha\left[\sigma(q,\ell) + \gamma \max_{a'} Q((s',v'),a') - Q((s,v),a)\right] \tag{2.6}$$

U praksi, algoritam QRM funkcionira tako da čuva odvojenu Q-tablicu za svako stanje $q \in Q$, i generira „virtualna iskustva"[2] opisana ranije za svaki prijelaz.

Algorithm 2 QRM

Inicijaliziraj $Q^q(s,a)$ proizvoljno za sve $s \in S$, $a \in A$, $q \in Q$
repeat
  Inicijaliziraj $s = s_I$, $q = q_I$
  repeat
    Odaberi a iz s koristeći politiku izvedenu iz Q^q (npr., ε-pohlepno)
    Poduzmi radnju a, dobij novo stanje s'
    $\ell \leftarrow L(s,a,s')$
    $q' \leftarrow \delta(q,\ell)$
    for $u \in Q$ **do** ▷ Uključuje i stvarni prijelaz $q \to q'$.
      $u' \leftarrow \delta(u,\ell)$
      $r \leftarrow \sigma(u,\ell)$ ▷ Podudara se s nagradom okruženja za $u = q$.
      $Q^u(s,a) \leftarrow Q^u(s,a) + \alpha\left[r + \gamma \max_{a'} Q^{u'}(s',a') - Q^u(s,a)\right]$
    $s \leftarrow s'$
    $q \leftarrow q'$
  until kraj epizode
until ispunjen uvjet zaustavljanja

Strojevi nagrade i QRM su dobra metoda za pristupanje problemima u potpornom učenju s rijetkim i vremenski međuzavisnim nagradama. U radu [4] mogu se naći eksperimenti koji uspoređuju QRM s alternativama poput hijerarhijskog potpornog učenja (HRL), te pokazuju da je QRM (na promatranoj klasi problema) najbolji izbor jer konvergira najbrže i jedini pronalazi optimalno ponašanje. Strojevi nagrade također mogu poslužiti u inverznom slučaju, kao alat za otkrivanje i objašnjavanje strukture nagrade nekog problema. Time se bave algoritmi JIRP [13] (u standardnom slučaju gdje je izlazna abeceda SN konačni podskup skupa $\mathbb{R}$, a izlazna funkcija deterministička) i SRMI [2] (koji koristi stohastičke strojeve nagrade i radi sa signalima nagrade koji mogu sadržavati i šum). Ta dva algoritma ne zahtijevaju pretpostavku da je znanje o strukturi nagrade u obliku konačnog automata unaprijed poznato agentu (kao što je to u slučaju algoritma QRM), već tu strukturu otkrivaju samo iz interakcije agenta i okruženja. Naravno, pretpostavlja se da postoji neki SN (ili stohastički analogon) koji može obuhvatiti nagrade do kojih agent dolazi. U grubim crtama, takve metode koje istovremeno uče i ponašanje i SN, problem učenja automata svode na niz problema zadovoljavanja ograničenja, koji se mogu riješiti dostupnim rješavačima SAT i SMT problema.

[2] U literaturi se ona nekad nazivaju *counterfactual experiences*.

Poglavlje 3

Sinteza funkcija nagrade

Spomenuti algoritam JIRP pokazuje da strojevi nagrade pridonose objašnjivosti modela u strojnom učenju. Inženjerima mogu predočiti strukturu nagrade bez da je unaprijed poznaju. U drugu ruku, JIRP ne može pomoći pri definiranju novog problema (nove funkcije nagrade) iz nule. Postojeće metode koje su izgrađene na temelju strojeva nagrade pate od poznatog problema: korisnici izbjegavaju pisanje formalizama.

Svrha ovog rada je olakšati taj posao u domeni strojeva nagrade, s ciljem da se inženjerima i drugim korisnicima olakša uporaba formalnih metoda u strojnom učenju. To će se postići uz pomoć modela za razumijevanje prirodnog jezika na bazi transformera, kako bi se omogućilo stvaranje SN samo iz opisa semantike (odnosno zadatka) na Engleskom jeziku.

U grubim crtama, na jednostavnom okruženju koje služi za vizualizaciju ponašanja treba omogućiti sljedeći proces, koji je u manje detalja već najavljen u uvodu.

1. Korisnik opisuje zadatak na Engleskom jeziku, na primjer *"patrol the forest and the factory, but avoid traps"*.
2. Poziva se jezični model koji prevodi korisnikov opis u formalizam stroja nagrade.
3. Koristi se QRM i SN iz koraka 2 da bi agent naučio obavljati zadatak kojeg je korisnik opisao u koraku 1.
4. Rezultati se korisniku demonstriraju vizualno na malom broju epizoda.

Osnovno korisničko iskustvo treba odavati dojam da je agent neposredno naučio obavljati zadatak kojeg je korisnik opisao prirodnim jezikom. Drugim riječima, koraci 2 i 3 trebaju biti skriveni od korisnika (to jest ne zahtijevati intervenciju korisnika).

U ovom radu od posebnog je interesa klasa SN čiji je izlazni alfabet O ograničen na dvočlani skup $\{0, 1\}$. Takvi SN mogu poslužiti kao pogodan alat kod zadavanja funkcija nagrade za zadatke u kojima je bitan samo uspjeh ili neuspjeh agenta, a ne, na primjer,

gradacija uspjeha u nekom većem (ili čak gustom) skupu. Sekcija 3.1 detaljnije opisuje vezu između opisa zadataka i semantike strojeva nagrade, te stvaranje prikladnog skupa podataka za treniranje i testiranje jezičnog modela.

3.1 Skup podataka

Kako bi se mogao ostvariti korak 2, u kojemu se korisnikov opis zadatka iz prirodnog jezika prevodi u stroj nagrade? Ako se u tu svrhu koristi autoregresivni jezični model (kao što je ovdje slučaj), najlakši je pristup koristiti međujezik. Popularni predtrenirani jezični modeli poput GPT-2 imaju istovrsni ulaz i izlaz: tekst (odnosno nizove *tokena*), pa je korištenje međujezika kao izlaza modela prirodan odabir. Taj međujezik mora određivati (ne jednoznačno) strojeve nagrade, kako bi izlaz iz jezičnog modela mogli interpretirati u ostatku programa, odnosno kompajlirati u SN.

Glavni praktični zadatak u ovom radu je predtrenirani jezični model poput GPT-2 dotrenirati na potproblemu prevođenja opisa u odabrani međujezik. U zaključku, skup podataka za treniranje se mora sastojati od parova stringova oblika (A, B), gdje A odgovara opisu na Engleskom jeziku, a B odgovara zapisu pripadnog SN u međujeziku.

Jedna od glavnih prednosti velikih jezičnih modela poput GPT-2 je da mogu poslužiti kao odskočna daska: umjesto da se model iznova trenira na velikom skupu podataka koji točno odgovara danom problemu, možemo koristiti objavljene parametre koji su pronađeni tako što se puno računalnih resursa uložilo u rješavanje općenitijih problema. Pošto je specifični problem prevođenja u međujezik uvelike vezan uz razumijevanje jezika, ti ranije pronađeni parametri su dobra polazna točka i za novi problem, i potreban je puno manji skup podataka za dotreniranje (*fine-tuning*).

Međujezik

Kao što je najavljeno na početku poglavlja 3, zanimaju nas zadaci koji se tiču ispunjavanja niza događaja u okruženju, i to s binarnim uvjetom uspjeha i neuspjeha, poput onog u primjeru 2.0.1. Odnosno, zadaci za koje se funkcija nagrade može predstaviti kao SN s izlaznom abecedom $\{0, 1\}$. Specifičnost takvih SN je da odgovaraju klasičnim konačnim automatima, u kojima prijelaz u stanja prihvaćanja odgovara izlazu 1, a ostali prijelazi izlazu 0. Preciznije, vrijedi propozicija 3.1.1.

Propozicija 3.1.1. *Za svaki stroj nagrade oblika* $\mathcal{A} = (Q, q_I, 2^P, set0, 1, \delta, \sigma)$ *postoji konačni automat* $\mathcal{B}$ *takav da za svaki niz događaja* $\ell_0, \ell_1, \ldots, \ell_n \subset 2^P$ *vrijedi* $\sum \mathcal{A}(\ell_0, \ell_1, \ldots, \ell_n) = m$*, gdje je* m *broj pojavljivanja prihvaćajućih stanja u računanju* $\mathcal{B}(\ell_0, \ell_1, \ldots, \ell_n)$*. Drugim riječima, za svaki SN postoji konačni automat nad istim alfabetom koji će za isti ulaz posjetiti prihvaćajuće stanje onoliko puta koliko SN na izlazu daje* 1.

Nadalje, ako $|\mathcal{A}| = n$, tada $|\mathcal{B}| \leq 2n$, to jest pripadni konačni automat ima najviše dva puta više stanja od početnog stroja nagrade.

Dokaz. Neka je $\mathcal{A} = (Q^{\mathcal{A}}, q_I^{\mathcal{A}}, 2^P, \{0,1\}, \delta^{\mathcal{A}}, \sigma^{\mathcal{A}})$ dani stroj nagrade. Konstruiramo pomoćni konačni automat, $\mathcal{B} = (Q^{\mathcal{B}}, q_I^{\mathcal{B}}, F^{\mathcal{B}}, 2^P, \delta^{\mathcal{B}})$ tako da vrijedi sljedeće.

1. Skup stanja je $Q^{\mathcal{B}} = \{0,1\} \times Q^{\mathcal{A}}$. U početku skup stanja $\mathcal{B}$ sadrži dvije kopije skupa stanja $\mathcal{A}$, no kasnije ćemo mnoga od njih moći izbaciti kao višak. Stanja $\mathcal{B}$ oblika $(0,q)$ nisu prihvaćajuća, a ona oblika $(1,q)$ (druga kopija stanja $\mathcal{A}$) jesu.

2. Početno stanje je $q_I^{\mathcal{B}} = (0, q_I^{\mathcal{A}})$. $\mathcal{B}$ nikada ne počinje u prihvaćajućem stanju jer je broj stanja u računanju $\mathcal{A}$ uvijek za jedan veći od broja izlaza $\mathcal{A}$ (odnosno prazan ulaz za $\mathcal{A}$ uvijek vodi do praznog izlaza koji ima sumu 0).

3. Skup prihvaćajućih stanja je $F^{\mathcal{B}} = \{(1,q) \mid q \in Q^{\mathcal{A}}\} \subseteq Q^{\mathcal{B}}$. Kao što je navedeno svako stanje u $\mathcal{A}$ dobija prihvaćajuću kopiju u $\mathcal{B}$.

4. Funkcija prijelaza $\delta^{\mathcal{B}}$ zadaje se na sljedeći način. Neka je $q \in Q^{\mathcal{A}}$ proizvoljno stanje u SN, a $\ell \in 2^P$ proizvoljna oznaka u O-MPO. Nadalje neka je $\beta \in \{0,1\}$. Tada formula 3.1 definira prijelaze iz stanja (β, q) u $\mathcal{B}$ pri ulazu ℓ.

$$\delta^{\mathcal{B}}((\beta,q),\ell) = \begin{cases} (0, \delta^{\mathcal{A}}(q,\ell)) & \text{ako } \sigma(q,\ell) = 0 \\ (1, \delta^{\mathcal{A}}(q,\ell)) & \text{inače} \end{cases} \tag{3.1}$$

   Drugim riječima, ako prijelaz u $\mathcal{A}$ po ℓ iz stanja q vodi do izlaza 0 (odnosno ako $\sigma(q,\ell) = 0$), stroj $\mathcal{B}$ prati prijelaz $\mathcal{A}$, odnosno prelazi u stanje $(0,q')$ gdje je $q' = \delta(q,\ell)$. Inače, ako $\sigma(q,\ell) = 1$, stroj $\mathcal{B}$ također prati prijelaz $\mathcal{A}$, no prelazi u kopiju stanja $(1,q') \in F$, odnosno prihvaća. U nastavku, to jest u naknadnim ulazima, funkcija prijelaza stroja $\mathcal{B}$ osigurava da njegova stanja prate stanje stroja $\mathcal{A}$.

Neka je $k \geq 1$ i $\omega = \ell_0, \ell_1, \ldots, \ell_{k-1}$ niz događaja, to jest ulaz. Neka je $q_0, q_1, \ldots, q_k$ izračunavanje stroja $\mathcal{A}$, a $(\beta_0, u_0), (\beta_1, u_1), \ldots, (\beta_k, u_k)$ izračunavanje stroja $\mathcal{B}$, koja inducira ω. Iz definicije funkcije prijelaza $\mathcal{B}$ u 4 je jasno da za svaki $i \in \{0,1,\ldots,k\}$ vrijedi $q_i = u_i$. Tvrdnja propozicije 3.1.1 se sada svodi na jednakost $\sum_{i\in\{0,1,\ldots,k\}} \beta_i = \sum_{i\in\{0,1,\ldots,k-1\}} \sigma(q_i,\ell_i)$. Neka za proizvoljni $i \in \{0,1,\ldots,k-1\}$ vrijedi $\sigma(q_i,\ell_i) = 1$. Tada po definiciji također vrijedi $\delta^{\mathcal{B}}((\beta_i,q_i),\ell_i) = (1, \delta^{\mathcal{A}}(q_i,\ell_i))$, što povlači $\beta_{i+1} = 1$.

Egzistencija pomoćnog automata $\mathcal{B}$ već dokazuje glavnu tvrdnju propozicije 3.1.1, no automat se zove pomoćni jer sam po sebi ima više stanja nego što je potrebno. Sporedna tvrdnja propozicije, $|\mathcal{B}| \leq 2n$, ne vrijedi samo u trivijalnom smislu, jer iz automata $\mathcal{B}$ možemo izbaciti sva nedostiživa stanja oblika $(1,q)$. Nedostiživa stanja tog oblika odgovaraju onim stanjima $q \in Q^{\mathcal{A}}$ za koje vrijedi da niti jedan njihov prethodnik (po $\delta^{\mathcal{A}}$) ne vodi u q s izlazom 1. □

Propozicija 3.1.1 daje korolar 3.1.2 koji je od stvarne koristi u ovom slučaju.

Korolar 3.1.2. *Svaki stroj nagrade s izlaznom abecedom* $\{0, 1\}$ *i ulaznom abecedom* 2^P *može se zadati regularnim izrazom nad ulaznom abecedom* 2^P*. Preciznije, za svaki SN* $\mathcal{A}$ *postoji regularni izraz B takav da za svaki niz događaja* $\omega = \ell_0, \ell_1, \ldots, \ell_n$ *vrijedi* $\mathcal{A}(\omega) = 1$ *ako i samo ako* $\omega \in L(B)$*.*

Ta činjenica nas poziva da kao međujezik odaberemo regularne izraze. Još jedan dobar izbor su LTL formule (*linear temporal logic*). U ovom slučaju detalji razlike u semantici Büchijevih i konačnih automata nisu zanimljivi, regularni izrazi su izabrani zbog šire rasprostranjenosti i savjeta da je s njima lakše raditi u ovom kontekstu.

Regularni izrazi koji se koriste u skupu podataka za treniranje zadani su nad alfabetom koji sadrži pojmove (objekte, područja, radnje) koji bi mogli biti relevantni agentu u vrsti demonstracija koje će biti prikazane korisniku. Primjer 3.1.3 prikazuje mali dio tih pojmova, u formatu `<pojam> <popis odrednica>`. Odrednice (poput `AREA`) bit će detaljnije objašnjene u narednim sekcijama.

Primjer 3.1.3.

```
forest AREA PLACE AVOID
coffee OBJECT NO_THE OBJECT_SINGULAR
right ACTION DIRECTION EXECUTABLE
red COLOR
```

Primjer 3.1.4 prikazuje stvarne elemente skupa podataka. Svaki primjer je na zasebnoj liniji. Format primjera je `A => B`, gdje je riječ => separator koji odvaja opis (označen s `A`) od regularnog izraza (označenog s `B`).

Primjer 3.1.4.

```
visit the garage. don't get near keys => (!key)*&((.)* > garage)

first find 5 fish in the hill. second pivot, pivot, and pivot =>
((.)* > fish&hill){5} > pivot > pivot > pivot

you have to find tools => (.)* > tools

action: pivot, do it once => pivot
```

Sintaksa međujezika

U diplomskom radu je reimplementiran sustav regularnih izraza zbog želje za većom kontrolom nad cijelim skupom podataka, olakšavanjem metoda proširivanja, i drugih specifičnosti problema. Podržani su standardni operatori konkatenacije >, unije |, presijeka &,

i Kleenijeve zvijezde `*`. Također je podržan operator +, koji odgovara ponavljanju jednom ili više puta, operator `{#n}` koji odgovara ponavljanju $n \in \mathbb{N}_+$ puta, te još neke konstrukcije koje olakšavaju stvaranje primjera. Slijede detaljnija sintaksna pravila.

1. Pojmovi (poput `key` ili `right`) su regularni izrazi.
2. Negirani pojmovi (poput `!key` ili `!right`) su regularni izrazi.
3. Posebni znakovi `.` (bilo koji događaj) i `_` (neprazan događaj) su regularni izrazi.

Neka su `X` i `Y` regularni izrazi. Tada su i sljedeće riječi regularni izrazi.

4. Konkatenacija: `X > Y`.
5. Disjunkcija: `X | Y`.
6. Konjunkcija: `X & Y`.
7. Komplement: `(X)~`.
8. Kleenijeva zvijezda: `(X)*`, i plus: `(X)+`.
9. Ponavljanje n puta: `(X){#n}`, gdje je `n` broj, i ponavljanje za unaprijed zadan, konačan broj puta: `(X){#some}` .

Operacija konjunkcije, to jest presijeka, često nije uključena u definicije regularnih izraza. Ali kako je klasa regularnih jezika zatvorena na konačne presijeke, uključiti je u definiciju ne predstavlja fundamentalni problem. Konjunkcija je uključena zato što je prirodna operacija za relevantnu semantiku: nizove događaja, odnosno zadatke. Više o tome rečeno je u narednoj sekciji koja se bavi semantikom međujezika.

Semantika međujezika

Cilj ovog pododjeljka nije dati potpunu definiciju semantike regularnih izraza, koja je velikim dijelom preuzeta iz standardnih izvora, već razjasniti potencijalno nejasne dijelove u ovoj konkretnoj primjeni.

Korolar 3.1.2 govori da je za uporabu definiranog međujezika regularnih izraza u svrhu definiranja funkcija nagrade potrebno poznavati njihovu semantiku, odnosno koje nizove prihvaćaju a koje ne. Radi se o nizovima događaja (oznaka), odnosno nizovima $\ell_0, \ell_1, \ldots, \ell_n \subset 2^P$, a ne o nizovima pojedinačnih pojmova. Naime, definiranje regularnih izraza nad nizovima pojmova (na primjer (`key`, `right`, `right`, `green`)) dovelo bi do raznih problema, poput činjenice da tada ne bi bilo moguće izraziti prazan događaj $\emptyset \in 2^P$, niti istovremenu istinitost više od jednog događaja, na primjer { `right`, `key` } $\in 2^P$.

Primjer niza događaja je ({}, {}, {right, key}, {right}, {}, {up, door}). Taj niz počinje s dva prazna događaja {}, i primjer regularnog izraza koji ga prihvaća je `(.)* > key > (.)* > door`. To je regularni izraz koji dobro obuhvaća zadatak iz primjera 2.0.1.

Poseban znak `!` koji nastupa u negiranim pojmovima danim sintaksnim pravilom 2 odnosi se samo na jedan događaj, to jest jedan element niza oznaka. Ako je A pojam (na primjer A = `key`), tada regularni izraz `!`A prihvaća sve nizove od točno jednog događaja u kojima pojam A nije istinit, to jest sve nizove $\omega = \ell_0$ takve da $A \notin \ell_0$. Na primjer, vrijedi {`door`} $\in L($`!key`$)$, ali ne vrijedi ({`door`}, {`door`}) $\in L($`!key`$)$ (naravno, ne vrijede ni {`key`} $\in L($`!key`$)$, ni $\epsilon \in L($`!key`$)$, gdje ϵ označava prazan niz događaja, iako su samo neprazni nizovi zanimljivi u ovom slučaju). U drugu ruku, operacija komplementa $\sim$ ima standardnu (skupovno-jezičnu) semantiku: *vrijedi* ({`key`}, {`key`}) $\in L($`(key)`$\sim)$, jer je jezik $L($`key`$)$ jednočlan.

Jezik $L($`(key)`$\{3\})$ sadrži sve nizove duljine 3 u kojima je `key` istinit u sva tri elementa. Na primjer, sadrži niz ({`key`}, {`door, key`}, {`key`}). U drugu ruku, jezik $L($`(key){#some}`$)$ ovisi o definiciji posebnog simbola `#some`, koji je zadan unaprijed (formalno, u definiciji semantike sustava). Simbol `#some` služi tome da se izlaz jezičnog modela može referirati na ponovljenu konkatenaciju, no bez da se referira na konkretan broj ponavljanja, to jest tome da se broj ponavljanja može izostaviti ukoliko to odgovara semantici opisanog zadatka. Primjer opisa zadatka u kojemu je to korisno je *"patrol the house and the field"*: jasno je da se od agenta traži da nekoliko puta prijeđe put između kuće i polja, ali nije određeno koliko mnogo puta. Zato skup podataka takve opise zadataka vezuje uz regularne izraze koji koriste konstantu `#some`.

Operator konjunkcije `&` posebno je koristan kada se agentu želi zabraniti ostvarivanje određenih nizova događaja. Neka je A regularni izraz. Ako želimo da agent nauči odrađivati zadatak vezan uz A, ali da ujedno, na primjer, nikada ne pronađe ključ, onda je korisno promatrati regularni izraz $B \equiv A$ `& ((.)* > key > (.)*)` $\sim$. Regularni izraz B prihvaća sve nizove događaja koji se nalaze u prijesijeku $L(A)$ (dakle ispunjavaju zadatak vezan uz A), te komplementa $L($`((.)* > key > (.)*)`$)$ koji sadrži sve nizove u kojima je `key` istinit u barem jednom elementu. Drugim riječima, ako agent ostvari događaj `key` i u jednom koraku, ne može ispuniti zadatak vezan uz B.

Naredna sekcija bavi se proširivanjem skupa podataka.

3.2 Proširenje podataka

U prethodne dvije sekcije opisan je sustav regularnih izraza koji omogućuje fleksibilno pisanje primjera zadataka koji se odnose na Markovljeve procese odluke s funkcijama nagrade koje ne zadovoljavaju Markovljevo svojstvo. Nažalost, doći do skupa podataka koji podliježe opisanom sustavu nije lagano. Za mnoge probleme u strojnom učenju postoje pri-

premljeni skupovi podataka na kojima se bez puno izmjena mogu testirati razvijene nove metode, no to ovdje nije slučaj. Postoje dvije mogućnosti za rješavanje tog problema.

1. Pronaći dostupan skup podataka koji je dovoljno sličan opisanom, te ga prenamijeniti.

2. Koristiti metode proširivanja skupa podataka (*data augmentation*).

Kao što je najavljeno, u ovom radu odabran je drugi pristup koji se temelji na proširivanju postojećeg manjeg skupa podataka (ali koji ipak točno odgovara potrebama). To je tema sekcije 3.2. Iako postoje skupovi podataka koji sadrže regularne izraze i njihove opise na Engleskom jeziku, prvi pristup je odbačen iz sljedećih razloga.

1. Opisi regularnih izraza koji se mogu pronaći u drugim izvorima najčešće se referiraju na sintaksna svojstva jezika stringova, i bilo bi ih teško transformirati u opise zadataka kakvi su ovdje potrebni.

2. Regularni izrazi u standardnim skupovima podataka pisani su u puno širem sustavu, koji je specijalno prilagođen radu s nizovima znakova (stringovima). To znatno povećava kompleksnost sustava značajkama koje nisu relevantne u ovom slučaju.

3. Slično kao u 2, povećana kompleksnost isključuje mnoge opcije za proširivanje, koje bi same po sebi bile korisne.

4. Razlike u semantici mogle bi dovesti do problema u distribuciji podataka: standardni regularni izrazi formuliraju se nad znakovima, no elementi nizova su u ovom slučaju skupovi. To bi moglo dovesti do neprirodne distribucije regularnih izraza, odnosno distribucije koja nije prilagođena problemima u ovom radu, iz koje bi jezični model teže učio.

I odabrana metoda (automatsko proširivanje manjeg, probranog skupa podataka) ima bitne mane.

1. Potreba za ručnim pisanjem primjera se nije eliminirala, samo je smanjena.

2. Treba se implementirati velika količina programske podrške za proširivanje, koja uzima u obzir i iskorištava specifične značajke danog sustava regularnih izraza i samog problema učenja.

3. Metode proširivanja također mogu dovesti do problema u distribuciji podataka za treniranje. O tome ima više riječi u sekciji 4.2, koja se bavi analizom rezultata treniranja jezičnog modela.

Odabir prikladne veličine skupa podataka za treniranje najčešće je empirijsko pitanje, ali u velikoj mjeri ovisi o vrsti modela i prirodi samih primjera. U praksi je gornja granica sama dostupnost (kvalitetnih) primjera. Također treba uzeti u obzir da se u ovom radu ne govori o treniranju velikog jezičnog modela ispočetka, već do dotreniranju već pronađenih parametara, što dodatno smanjuje dovoljan broj primjera za treniranje. Više riječi o ovome će biti u sekciji 3.3 o treniranju. Za sada je dovoljno reći da je za eksperimente najčešće korišteno 50–100k parova opisa i regularnih izraza sličnih onima iz primjera 3.1.4.

Ti primjeri nisu pisani ručno, već su izvedeni nizom metoda iz manjeg originalnog skupa. Primjer 3.2.1 prikazuje jedan element tog originalnog skupa.

Primjer 3.2.1.

```
0 @ {}, {}, {}
0 @ {}, {}, {'$C'}
0 @ {}, {'$A'}, {'$C'}
0 @ {}, {'$B'}, {'$C'}
0 @ {}, {'$A', '$B'}, {'$C'}
1 @ {}, {}, {'$C', '$A'}
1 @ {}, {}, {'$C', '$B'}
1 @ {}, {}, {'$C', '$A', '$B'}
=
$A % COLOR, $B % COLOR, $C % OBJECT_SINGULAR
=
Find $A or $B $C
Get $A or $B $C
Find a $C (but it needs to be $A or $B)
Find $Cs (it should be either $A or $B)
Get a $C (but it needs to be $A or $B)
Get a $C (it should be either $A or $B)
=
(.)* > (($A | $B)&$C)
=
Find $A or $B $Cs
```

Shema po kojoj su pisani primjeri unutar originalnog skupa je sljedeća.

```
<popis validacijskih izračunavanja>
=
<popis deklaracija varijabli>
=
```

```
<popis opisa u prirodnom jeziku>
=
<popis regularnih izraza>
=
<identifikator>
```

Prva stavka sheme je popis validacijskih izračunavanja. Svaki element tog popisa ima dva dijela, odvojena separatorom `@`. Prvi dio je prirodan broj koji određuje ukupnu sumu nagrade koju SN za dani zadatak treba izdati kada mu se na ulazu nalazi niz oznaka kojeg zadaje drugi dio. Validacijska izračunavanja su testovi originalnih primjera, a mogu poslužiti (gdje je to moguće) i kao testovi izvedenih primjera. Najveća korist tih testova je osiguravanje usklađenosti regularnih izraza u primjeru s namijenjenom semantikom zadatka, odnosno osiguravanje da strojevi nagrade koji su izvedeni iz svakog regularnog izraza na popisu točno klasificiraju pozitivne (uspješne) i negativne (neuspješne) nizove događaja u validacijskim izračunavanjima.

Drugi dio sheme su deklaracije varijabli, to jest upute za supstituciju fiksnih pojmova. U primjeru 3.2.1 ne nastupaju pojmovi poput onih iz primjera 3.1.3, nego varijable kao što su `$A` i `$B`. Nastupi tih varijabli u regularnim izrazima ili opisima će se u procesu proširenja originalnog skupa podataka zamijeniti konkretnim pojmovima iz unaprijed zadanog popisa. Pravila te zamjene, to jest dopuštene pojmove za određene varijable, definira dio sheme `<popis deklaracija varijabli>`. U primjeru 3.2.1 navodi se jedan skup deklaracija–`$A % COLOR, $B % COLOR, $C % OBJECT_SINGULAR`–no u jednom originalnom primjeru može nastupiti više skupova deklaracija varijabli. U tom slučaju stvorit će se nove kopije originalnog primjera, i svakoj će se pridružiti po točno jedan skup deklaracija iz originalnog primjera. Varijable omogućuju osnovni način proširenja originalnog skupa podataka: za različite supstitucije (npr. `$A % red, $B % blue, $C % coffee`) dobivamo nove (različite) izvedene primjere.

Treći i **četvrti** dio sheme navode opise na Engleskom jeziku i regularne izraze. Svaki od tih opisa i regularnih izraza odnose se na isti zadatak. Drugim riječima, bilo koji opis može se upariti s bilo kojim regularnim izrazom unutar istog primjera. Time je dobiven novi par spomenutog oblika (A, B). Iako se odnose na isti zadatak, svi takvi parovi koje poluči jedan originalni element su različiti, i kontribuiraju nove informacije pri treniranju jezičnog modela. Shodno tome Kartezijev produkt opisa i regularnih izraza unutar istog originalnog elementa je druga metoda proširivanja skupa podataka.

Peti i zadnji dio sheme je identifikator. Tokom stvaranja izvedenog skupa primjera, bitno je spriječiti preveliko iskrivljavanje distribucije podataka, na primjer, asimetričnim brojem ponavljanja nekih opisa koji se razlikuju samo u supstituiranim pojmovima. Taj problem može nastupiti jer neki originalni primjeri dijele regularne izraze i neke od opisa (dok su neki opisi pak različiti, pa se primjeri ne mogu skupiti u isti element). Takvi primjeri imaju isti identifikator, koji služi da se ograniči broj pojavljivanja opisa. Preciznije,

ograničen je broj parova `(opis, identifikator)`. Kad identifikator nije naveden, koristi se prvi navedeni opis u primjeru.

Slijedi popis svih metoda proširenja.

1. Kartezijev produkt opisa i regularnih izraza.
2. Dopuna primjera ekvivalentnim regularnim izrazima.
3. Zamjena varijabli u primjerima s fiksnim pojmovima.
4. Stvaranje novih primjera iz obrazaca.

Izvedeni skup podataka je tri reda veličine veći od polaznog. Metoda 4 najviše kontribuira faktoru povećanja originalnog skupa.

Metode **Kartezijev produkt opisa i regularnih izraza** 1 i **zamjena varijabli u primjerima s fiksnim pojmovima** 3 već su objašnjene u na početku sekcije, u diskusiji sheme zapisa originalnih primjera.

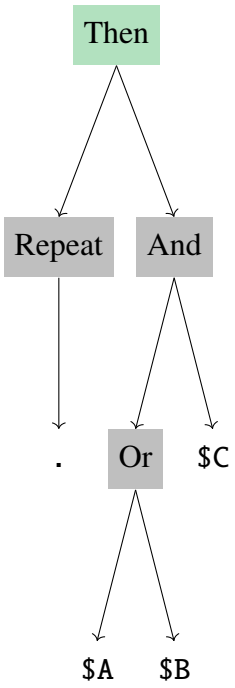
Dopuna primjera ekvivalentnim regularnim izrazima 2 odnosi se na povećavanje broja različitih regularnih izraza koji su pridruženi jednom originalnom primjeru. Neka je $\mathcal{L}$ regularan jezik i R regularan izraz tako da vrijedi $\mathcal{L} = L(R)$. Ako za neki regularni izraz R' vrijedi $\mathcal{L} = L(R) = L(R')$ kažemo da su R i R' ekvivalentni izrazi i pišemo $R \sim R'$. Svaki regularan izraz ima prebrojivo mnogo ekvivalentnih izraza (to je trivijalna činjenica: na primjer, presijek jezika L sa samim sobom nema efekta na prepoznati jezik, ali ima na sintaksu odnosno sam izraz).

U ovom slučaju, cilj jezičnog modela nije naučiti distribuciju regularnih izraza kakvi se pojavljuju u digitaliziranim pisanim materijalima poput udžbenika ili javno dostupnog softverskog koda, nego naučiti mapiranje koje odgovara semantici zadataka. Ta činjenica dopušta da skup primjera proširimo primjenom pravila za pronalaženje ekvivalentnih regularnih izraza. Jasno je da korištenje trivijalnih pravila poput presijeka jezika L sa samim sobom neće puno kontribuirati informacijskom sadržaju skupa podataka. Takve transformacije u početne regularne izraze samo dodaju redundanciju, te se na njih može gledati kao na mali pomak unutar niskodimenzionalnog potprostora u prostoru svih primjera. S druge strane, transformacije koje koriste druga pravila u sustavu regularnih izraza, poput De Morganovih zakona ili pravila distributivnosti, predstavljaju veće pomake koji vežu udaljenije izraze istom semantikom.

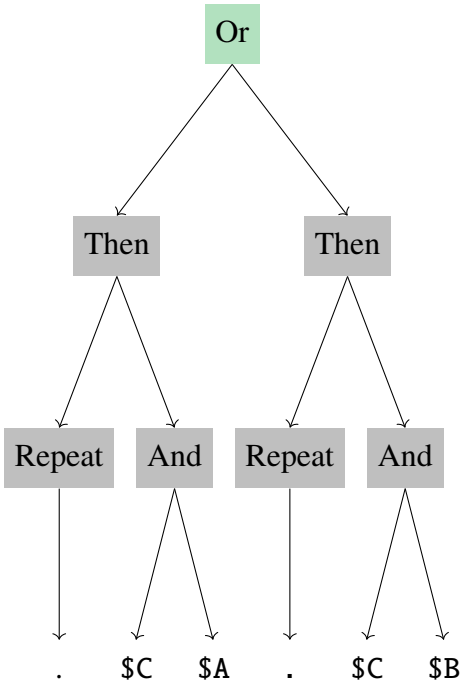
Odabrana pravila za pronalaženje ekvivalentnih regularnih izraza ostvarena su transformacijama sintaksnog stabla koje čuvaju semantiku. Slika 3.3 prikazuje efekt transformacija na primjeru jednostavnog izraza.

Slijedi popis implementiranih pravila. Također su implementirani inverzi ovih pravila.

1. Distributivnost presijeka prema uniji. Primjer: `$A & ($B | $C)` u `$A & $B | $A & $C`.



Slika 3.1: `(.)* > (($A | $B) & $C)`



Slika 3.2: `(.)* > $C&$A | (.)* > $C&$B`

Slika 3.3: Sintaksna stabla dvaju ekvivalentnih regularnih izraza. Lijevi izraz je početni: nalazi se u originalnom skupu podataka (naveden je u primjeru 3.2.1). Desni izraz dobiven je primjenom više pravila transformacije sintaksnih stabala, te bi se nalazio u izvedenom skupu podataka (zajedno s desnim).

2. Distributivnost konkatenacije prema uniji. Primjer: `$A > ($B | $C) > $D` u `($A > $B | $A > $C) > $D`.
3. Zamjena redoslijeda u presijeku i uniji. Primjer: `$A & $B` u `$B & $A`.
4. Raspisivanje Kleenijevog plusa preko konkatenacije i Kleenijeve zvijezde. Primjer: `(A)+` u `A > (A)*`.
5. De Morganovi zakoni. Primjer: `$A | $B` u `(($A)~ & ($B)~)~`.
6. Micanje duple negacije i sličnih artefakata koji mogu nastati primjenom ostalih pravila. Primjer: `(($A)~)~` u `$A`.

Jedan potencijalni smjer daljnjeg rada je pronalaženje novih (kompliciranijih) pravila. Algoritam koji za konkretno sintaksno stablo pronalazi ekvivalentne regularne izraze je

jednostavna rekurzija po čvorovima. Da bi se pronašli ekvivalentni izrazi za dano sintaksno stablo, algoritam rekurzivno pronalazi ekvivalentne izraze za djecu korijena stabla. Nakon toga se navedena pravila primjenjuju nad pronađenim ekvivalentnim stablima i početnim korijenom. Čvorovi koji odgovaraju pojedinoj varijabli ili posebnom znaku nemaju ekvivalentnih stabala, te rekurzija staje na njima (vraćaju sami sebe). Broj rezultata je ograničen parametrom algoritma, jer je eksponencijalan u broju čvorova.

Stvaranje novih primjera iz obrazaca 4 je zadnja metoda proširenja skupa podataka, te ujedno i najmoćnija, jer koristi operatore u sustavu regularnih izraza kako bi stvorila nove primjere, a ne samo izmijenila ili nadopunila postojeće. Regularni izrazi (odnosno konačni automati) su popularan formalizam u mnogim primjenama jer su jednostavni za analizu i implementaciju no i dalje dovoljno kompliciran model računanja da se u njima mogu izraziti specifikacije protokola, zadaci u učenju s potporom, upiti na grafovima, itd. No imaju još jedno dobro svojstvo: početni konačni automati se vrlo lako mogu komponirati u nove. Primjerice, ako postoje dva elementa koji opisuju neke različite zadatke, može se dobiti novi, treći primjer koristeći konkatenaciju. Sve što treba napraviti je koristiti spojnicu („ljepilo") koje u prirodnom jeziku odgovara konkatenaciji, a pripadne regularne izraze početnih primjera spojiti operatorom >. Spojnica je niz znakova poput "$\cdot$, *and then* $\cdot$". Takve konstrukcije mogu se napraviti i za ostale operatore u sustavu. Nema razloga stati samo na tome: korisno je stvarati i kompliciranije konstrukte, to jest obrasce za spajanje početnih primjera, koji ne moraju odgovarati primjeni samo jednog operatora. Tih obrazaca je definirano puno i nema ih potrebe navoditi sve (dostupni su na otvorenom repozitoriju koda), nego samo naglasiti da je princip pisanja novog obrasca gotovo identičan pisanju novog originalnog primjera. Razlika je u tome da varijable, osim iz fiksnih pojmova, mogu preuzimati vrijednosti u opisima ili izrazima iz originalnih primjera koji se supstituiraju.

Primjer 3.2.2 prikazuje jedan takav obrazac.

Primjer 3.2.2.

```
0 @ {}, {}
=
$Z % AVOID
=
Avoid the $Z. DESC0, then DESC1
DESC0. Then, DESC1. Avoid the $Z
Don't get near the $Z: DESC0, then DESC1
DESC0. Then DESC1. Avoid the $Z
DESC0. Then, DESC1. Don't approach the $Z
=
((SRC0) > (SRC1)) & (!$Z)*
```

```
((SRC0) > (SRC1)) & ((.)* > $Z > (.)*)~
=
concat_avoid
```

Iz dva polazna primjera s opisima DESC0 i DESC1, te izrazima SRC0 i SRC1, bit će stvoren novi primjer, koji odgovara konkatenaciji zadataka iz dva početna, uz dodatan uvjet da se događaj (označen varijablom $Z) ne smije dogoditi. Shema je ista kao za obične primjere.

U cjelosti proces proširenja skupa podataka je dan sljedećim nizom koraka.

Algorithm 3 Proširenje skupa podataka

```
primjeri ← učitajSveOriginalnePrimjere()
validiraj(primjeri)
primjeri ← proširiEkvivalentnimIzrazima(primjeri)
primjeri ← KartezijevProduktOpisaIIzvora(primjeri)
obrasci ← učitajObrasce()
noviPrimjeri ← stvoriNovePrimjere(obrasci, primjeri)
primjeri ← primjeri + noviPrimjeri
fiksniPojmovi ← učitajFiksnePojmove()
primjeri ← supstituirajVarijable(fiksniPojmovi, primjeri)
return primjeri
```

Treba naglasiti da se u algoritmu 3 ne navode koraci koji pomažu pri povećavanju sličnost u disribuciji originalnog skupa podataka i onog koji rezultira metodama proširenja. Oni se ostvaraju uz pomoć identifikatora primjera, koji je objašnjen u sklopu sheme nakon primjera 3.2.1, a neki od tih koraka nalaze se i u realizaciji funkcije supstituirajVarijable(.)

3.3 Jezični model

Za ostvarivanje koraka 2 korišten je OpenAI model GPT-2 koji je dostupan na Hugging Face repozitoriju[1]. U ovom slučaju, koristi se *small* varijanta GPT-2 modela koja ima 124 milijuna parametara. Konkretan jezični model lagano je zamijeniti putem konfiguracijskih datoteka. Model je treniran na izvedenom skupu podataka koji je ostvaren metodama koje su opisane u sekciji 3.2. Kao što je već rečeno, preuzeti model je predtreniran na jako širokom skupu podataka koji se sastoji od 8 milijuna mrežnih stranica, a u sklopu ovog rada on je samo dotreniran na jednom specifičnom jezičnom zadatku.

[1] https://huggingface.co/gpt2

Treniranje jezičnog modela

Tijekom prvotnog treniranja model je optimiran na temelju cilja kauzalnog jezičnog modeliranja, to jest cilja CLM (*causal language modelling*). CLM spada pod metode nenadziranog strojnog učenja. Početni skup podataka u tekstualnom obliku prvo se tokenizira, odnosno pretvori u niz jedinica koje su na višoj razini od samih znakova u tekstu, ali na nižoj razini od riječi. Proces tokenizacije se određuje prije samog treniranja statističkim metodama koje analiziraju frekvencije pojavljivanja kombinacija znakova u skupu podataka za treniranje. U CLM-u, model predviđa sljedeći token za dani početni komad u nizu tokena. Predikcija modela je vjerojatnosna distribucija nad potencijalnim sljedećim tokenima. Točna predikcija, koja se očita iz primjera to jest iz skupa podataka za treniranje, je vektor koji sadrži 1 na indeksu tokena iz primjera (kad fiksiramo uređaj nad svim tokenima), a 0 na svim ostalim indeksima (*one-hot vector*). Model se trenira tako da minimizira razliku predviđene i točne vjerojatnosne distribucije.

Kao što je već navedeno jezični zadatak u ovom radu je preslikavanje opisa semantike strojeva nagrade u regularne izraze. U standardnoj formulaciji CLM je preopćenit okvir za taj problem, jer nije osjetljiv na postojeću strukturu skupa podataka, koja je u ovom slučaju jako izražena. Svaki primjer ima spomenuti oblik (A, B), pa u model možemo unijeti korisnu pristranost (*bias*) tako da iskoristimo poznate razlike u distribuciji tokena u nizovima A i B. U ovom radu to je postignuto tako da se modificira izračun funkcije gubitka na način da se gubitak računa samo u B dijelu, odnosno na tokenima koji pripadaju regularnom izrazu.

Koristi se optimizator AdaFactor [12] umjesto varijante Adama zbog boljeg profila iskorištenja memorije. Hiperparametri za AdaFactor su sljedeći:

- Stopa učenja (*learning rate*): 5×10^{-4}. Ovo je samo početna vrijednost. Algoritam AdaFactor dinamički postavlja stopu učenja.
- Stopa opadanja parametara (*weight decay*): 1×10^{-4}. Opadanje parametara je metoda regularizacije (prevencije prenaučenosti) koja je prisutna u mnogim optimizatorima. U funkciju gubitka dodaje penalizaciju koja je proporcionalna L^2 normi parametara modela to jest težina.
- Ostali parametri za regularizaciju koji su specifični za AdaFactor nisu mijenjani.
- Opcije `scale_parameter`, `relative_step`, i `warmup_init` u Hugging Face implementaciji su isključene.

Koristi se jedna epoha treniranja, iz dva razloga.

1. Veličina izvedenog skupa direktno određuje broj iteracija optimizacije. Umjesto ponavljanja optimizacijskih koraka na istim podacima za treniranje u novoj epohi, slutnja je da je bolje te korake napraviti na podacima koji sadrže nove informacije, ako je to moguće.

2. *One Epoch Is All You Need* [6] sugerira korištenje samo jedne epohe pri treniranju jezičnih modela temeljenih na arhitekturi transformera.

Ukupna veličina grupa primjera (*batch size*) je 64, ali to se postiže tako da se osnovna veličina grupa postavi na 8, te se koristi 8 koraka akumulacije gradijenta. To je metoda koja omogućuje računanje s većim grupama u situacijama kada je memorija sustava ograničena.

Skup podataka se dijeli 9:1 na skup za treniranje i skup za validaciju.

Ispravljanje semantičkih grešaka

Korištenje jezičnih modela temeljenih na arhitekturi transformera za rad s formalizmima je novi smjer istraživanja u području formalnih metoda. Bitan problem u takvom pristupu je da jezični modeli često rade greške. Sustavi poput PICARD-a [11] i Syncromesha [9] koriste kompleksne metode ograničavanja izlaza jezičnih modela u svrhu eliminiranja značajnih grešaka. U drugu ruku, LtlTalk [3] uključuje korisnika u interaktivni proces popravljanja izlaza (iako LtlTalk ne koristi jezični model, te ideje su primjenjive i šire). Metode predložene u navedenim radovima su moćne i općenite. U ovom radu implementirane su manje napredne metode, ali one odgovaraju specifičnostima danog problema.

Greške u izlazu jezičnog modela mogu se razvrstati na sintaksne i semantičke. Sintaksne greške odnose se na slučajeve kada izlaz modela ne odgovara opisanoj sintaksi regularnih izraza (na primjer, izlaz sadrži neuravnotežen broj zagrada). Semantičke greške je teže lagano definirati jer se odnose na razliku između onoga što je korisnik naumio, i semantike izraza kojeg je jezični model vratio. Pristup rješavanju obje klase grešaka u ovom radu je relativno jednostavan, te počinje od toga da od modela umjesto jednog odgovora tražimo njih N (gdje je N parametar, uz tipičnu vrijednost oko 50). Kako su izlazi jezičnih modela poput GPT-2 stohastički, odnosno rezultiraju uzorkovanjem distribucije nizova tokena, i sami rezultati će se razlikovati (ovisi o načinu uzorkovanja i drugim parametrima). Zatim dobivenih N odgovora filtriramo kompajliranjem, te odbacimo odgovore koji su sintaksno neispravni.

Semantičke greške je puno teže riješiti bez naprednih metoda koje mijenjaju uzorkovanje samog modela, ili u proces uključuju korisnika. Ipak, iskorištavanjem činjenice da su sintaktički validni izlazi modela regularni izrazi, može se nešto postići.

Ocjenjivanje izlaza modela na temelju sličnosti s ulazom je prva metoda koja se koristi za smanjivanje vjerojatnosti semantičkih grešaka. Metoda je motivirana hipotezom da je vjerojatnost da u točnom odgovoru i upitu (to jest ulazu) modela nastupaju različiti

pojmovi malena. Neka je L skup svih riječi u ulazu (opisu), a R skup svih riječi u izlazu modela. Skup R sadrži fiksne pojmove, odnosno riječi koje se pojavljuju na popisu mogućih vrijednosti varijabli u skupu podataka (nakon što izbacimo `#some`). Ocjena odgovora modela je tada $-|L \triangle R|$, odnosno kardinalnost simetrične razlike ta dva skupa (veća razlika je lošija). Inicijalni sintaksno ispravni odgovori se silazno sortiraju preko izračunatih ocjena, te se odbacuje najgore ocijenjena polovina odgovora.

Semantičko klasteriranje je druga metoda za smanjivanje vjerojatnosti grešaka koja koristi mjeru sličnosti izlaza modela. Ova metoda se ne koristi u krajnjoj implementaciji, jer se kosi s načinom uzorkovanja modela koji sam po sebi daje bolje rezultate. Osnovna hipoteza iza semantičkog klasteriranja je da su semantički netočni odgovori također međusobno semantički različiti, te će pojedinačno imati manju vjerojatnost od semantički točnih odgovora sveukupno. To znači da će za neku mjeru semantičke sličnosti odgovora, oni točni tvoriti međusobno blizak klaster (idealno u jednoj točki), dok će netočni odgovori biti raspršeni nasumično. Koristi se stohastička mjera sličnosti. Prvo se generira $R \cdot M$ nasumičnih nizova događaja (u alfabetu pojmova koji nastupaju primjerima), gdje je R faktor redundancije (koristi se $R = 10$). Zatim se nasumični nizovi ocijene po tome koliko dobro „cijepaju" skup odgovora na dvije polovine: traže se nizovi koji daju uvid u semantičke razlike među odgovorima. Zadrži se najboljih M nizova. Za računanje sličnosti strojeva nagrade $\mathcal{A}$ i $\mathcal{A}'$, oni se evaluiraju nad svih M nasumičnih nizova, te se rezultati klasificiraju kao pozitivni (došlo je do nagrade) ili negativni (nije došlo do nagrade). Neka je $M' \leq M$ broj jednako klasificiranih nizova. Sličnost se računa kao omjer $a(\mathcal{A}, \mathcal{A}') = \frac{M'}{M}$. Konačno, međusobne sličnosti svih odgovora modela zapisuju se u simetričnu matricu, te se odgovori klasteriraju.

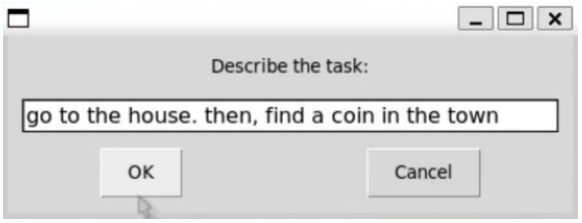
Da bi metoda semantičkog klasteriranja imala smisla potrebno je nezavisno uzorkovati N odgovora modela. No u praksi se pokazalo da je bolji pristup koristiti pretragu po snopovima (*beam search*), što je općenita metoda uzorkovanja jezičnih modela bez direktne veze s pojmom semantike odgovora koji je ovdje relevantan. Pretraga po snopovima će također vratiti N odgovora, ali će prostor nizova tokena pretraživati u obliku stabla, te pamtiti N najvjerojatnijih puteva kroz stablo. Posljedično, odgovore možemo poredati po vjerojatnosti, te ih nakon filtriranja, ocjenjivanja, i sortiranja nema smisla naknadno klasterirati.

Poglavlje 4

Rezultati i analiza

4.1 Korisničko sučelje

Svi koraci 1– 4 implementirani su kroz grafičko korisničko sučelje.

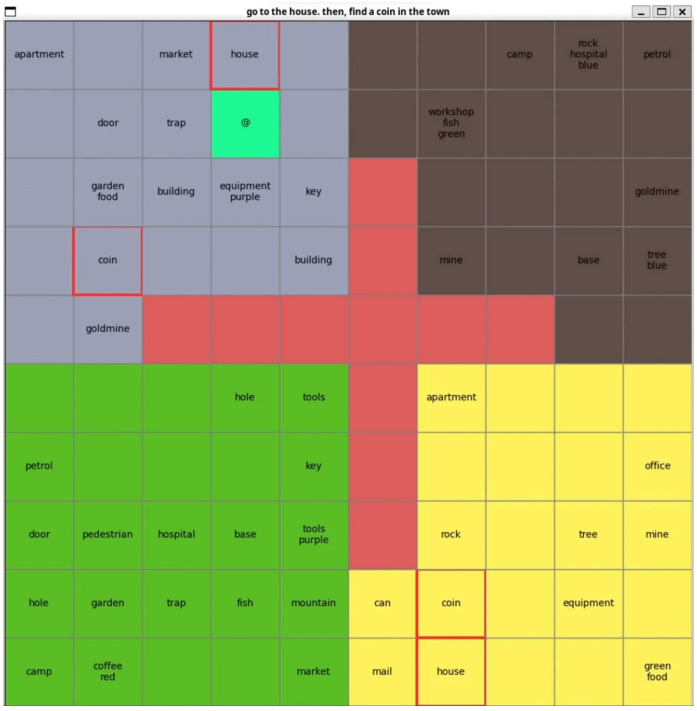


Slika 4.1: Prozor za upis opisa zadatka

Korisnik na početku vidi prikaz okruženja bez agenta, zajedno s prozorom za upis opisa zadatka koji je prikazan na slici 4.1. Nakon klika na gumb OK, poziva se model s korisnikovim opisom kao ulazom, te se rezultati poziva obrade na način koji je opisan u prethodnim sekcijama. Pronađeni stroj nagrade koristi se pri treniranju uz QRM algoritam, a korisniku se prikazuju statusne poruke koje uključuju informaciju o pronađenom regularnom izrazu, prosječnoj nagradi agenta u danom trenutku treniranja, i slično. Slika 4.2 prikazuje demonstraciju naučenog ponašanja nakon treniranja.

4.2 Rezultati treniranja jezičnog modela

Generirani skup podataka prvo je rastavljen na podskup za treniranje (90% primjera) i podskup za validaciju (10% primjera). Rezultati dotreniranja jezičnog modela praćeni su pomoću mjere BLEU [8] (engl. *bilingual evaluation understudy*). BLEU je mjera koja je namijenjena za ocjenjivanje kvalitete prijevoda teksta, i služi kao standardni izbor mjerila

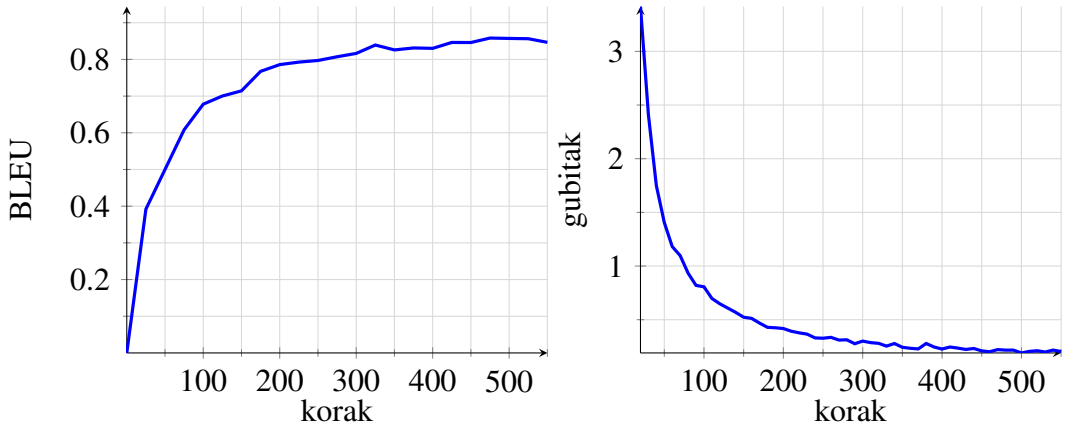


Slika 4.2: Demonstracija epizode nakon učenja. Neke oznake stanja su naznačene su tekstom u poljima (na primjer, *apartment*). Neke oznake su naznačene bojom. Crvena odgovara zidu (oznaka *wall*), siva gradu (oznaka *town*), zelena šumi (oznaka *forest*), smeđa tvornici (oznaka *factory*), a žuta polju (oznaka *field*). Stanje agenta je označeno svijetlom zelenom bojom i simbolom `@`. Tijekom demonstracije sučelje crvenim obrubom ističe stanja okruženja čije su oznake relevantne za pronađeni stroj nagrade što olakšava praćenje demonstracija.

kvalitete jezičnih modela u širokom nizu zadataka. Idealno mjerilo za jezični zadatak kakvog se promatra zapravo bi u obzir trebalo uzimati i semantiku. Na primjer, jedan kandidat je metoda za određivanje sličnosti strojeva nagrade kakva se koristi u ranije opisanom semantičkom klasteriranju. U drugu ruku, mjerilo kvalitete također mora biti efikasno (jer se model evaluira tokom treniranja na velikom broju primjera). BLEU je odabran zbog široke prihvaćenosti, efikasnosti, i zato što čak i uz svoje mane (ignoriranje semantike) pruža uvid u kvalitetu izlaza modela. BLEU rezultat je računat na validacijskom skupu.

Slika 4.3 (lijevo) prikazuje BLEU rezultat modela u ovisnosti o optimizacijskom koraku. Također je prikazan (desno) iznos funkcije gubitka (isto u ovisnosti o optimizacijskom koraku).

Model postiže BLEU rezultat od 86%, što se u literaturi velikih jezičnih modela smatra vrlo uspješnim rezultatom. Nažalost, zbog navedenih mana primjene tog mjerila kvalitete



Slika 4.3: Rezultati dotreniranja GPT-2 modela na izvedenom skupu podataka. Lijevi graf prikazuje rast BLEU ocjene na validacijskom skupu podataka u ovisnosti o optimizacijskom koraku. Desni dijagram prikazuje pad funkcije gubitka tokom treniranja. Na desnom dijagramu izostavljen je prvi izračun gubitka koji se odvija prije dotreniranja modela jer odskače nekoliko redova veličine od drugih vrijednosti.

na ovom problemu, model se ne može smatrati dovoljno dobrim samo na temelju visokog BLEU rezultata. Glavni razlog za posumnjati u stvarnu kvalitetu modela je u ovom slučaju kvaliteta izvedenog (sintetičkog) skupa podataka. U strojnom učenju je korisna pretpostavka da ulazni podaci tvore nižedimenzionalnu mnogostrukost ulegnutu u visokodimenzionalnom prostoru značajki, a da se skup podataka za treniranje sastoji od točaka koje su (u idealnom slučaju) uniformno uzorkovane na toj mnogostrukosti. U ovom se slučaju može postaviti neformalna analogija s takvim razmatranjem: ako su originalni (malobrojni) primjeri u skupu podataka uniformno distribuirani na mnogostrukosti koja odgovara dijelu Engleskog jezika koji sadrži opise semantike strojeva nagrade (to jest, zadatke za agente), onda preslikavanje tog originalnog skupa u veliki izvedeni skup podataka (kakvo je opisano u sekciji 3.1) sigurno ne čuva mnoga poželjna statistička svojstva podataka za treniranje. Novi opisi koji rezultiraju obradom obrazaca sigurno neće biti ni blizu uniformno distribuirani, jer opisi koji se umeću u obrasce nisu nezavisno uzorkovani iz jezika. Nadalje, transformacije koje se primjenjuju u obrascima smanjuju varijabilnost rečenica, odnosno čine ih neprirodnima: u analogiji, „pogađa” se manjedimenzionalna mnogostrukost nego što bi bilo idealno.

Navedene mane umanjuju mogućnost generalizacije modela, odnosno kvalitetu odgovora za ulaze koji su različiti nego oni u skupu podataka za treniranje. Štoviše, zato što validacijski skup podataka–na kojemu se računa BLEU–pati od istih mana, posredno je umanjena vjerodostojnost samih BLEU ocjena (zato što se predikcije računaju iz podskupa jezika koji nije uniformno i nezavisno uzorkovan, pa se njihov BLEU rezultat ne

može nužno interpretirati kao dobro mjerilo izvan tog skupa).

Jedan način na koji bi se mogao pokušati umanjiti efekt navedenih mana je korištenje alata za automatsko parafraziranje jezika. Drugi jezični modeli (na primjer, Pegasus [14]) mogli bi se iskoristiti za parafraziranje generiranih opisa, s ciljem da se rečenice učine prirodnijima, to jest da se popravi distribucija skupa. Nažalost, to je netrivijalan zadatak zbog jako uske veze opisa i semantike izlaza u ovom jezičnom problemu, i implementacija te ideje zahtijeva još dorade.

No ipak, eksperimentiranjem se može vidjeti da model nije u potpunosti pretreniran, odnosno da ipak može generalizirati, nekad čak i u slučajevima gdje ulazni podatak odudara od viđenih u velikoj mjeri. Primjer 4.2.1 prikazuje odgovore modela na tri upita koji ne odgovaraju primjerima u skupu podataka za treniranje. Upiti počinju simbolom `:`, a odgovor modela je u sljedećoj liniji.

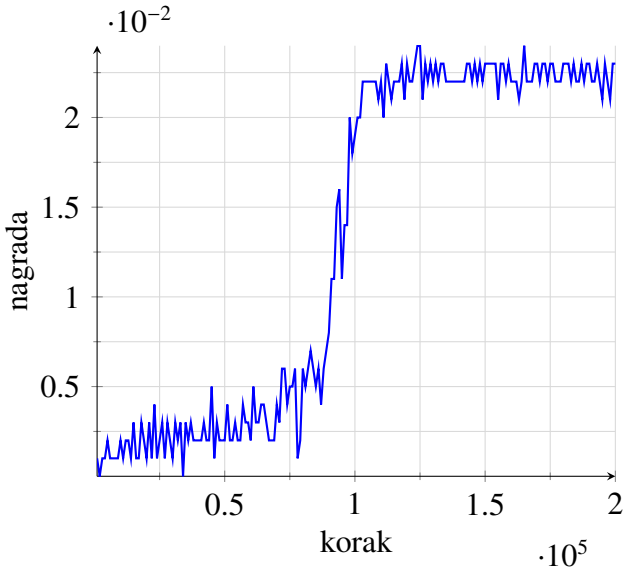
Primjer 4.2.1.

```
: don't find cans
((.)* > can > (.)*)~&((.)* > can > (.)*)~

: find a positive number of cans
((.)* > can){#some}

: find several cans
((.)* > can){#some}
```

Zadatak koji od agenta traži da ne pronađe neki predmet ne postoji u podacima za treniranje. Riječi *positive* i *several* se uopće ne pojavljuju u skupu generiranih primjera. Odgovori modela u sva tri slučaja se podudaraju s upitima u intendiranoj semantici. Jedna hipoteza zašto je model generirao zadovoljavajuće odgovore je da se znanje o jeziku uspješno prenijelo iz starih vrijednosti parametara (originalnog GPT-2), te da je dotreniranje kontribuiralo mogućnosti generalizacije u ovom jezičnom zadatku.

4.3 Rezultati algoritma QRM sa sintetiziranim funkcijama nagrade

Na slici 4.4 prikazan je graf nagrade agenta tokom treniranja algoritmom QRM, s ukupnim trajanjem od $2 \cdot 10^5$ koraka. Slika 4.5 također prikazuje nagradu agenta, ali s puno jednostavnijim upitom i trajanjem od samo 16000 koraka. U oba slučaja radi se o treniranju uz SN koji je pronađen iz opisa zadatka korisnika, te se može vidjeti da algoritam konvergira u optimalno ponašanje.

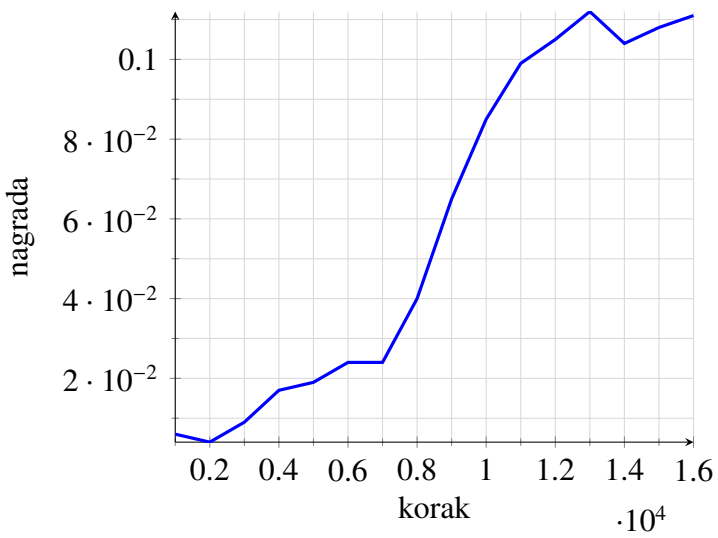


Slika 4.4: Rezultati algoritma QRM za upit `patrol the field and the town`. Nagrada je izaražena kao prosječna nagrada po koraku simulacije u zadnjih 1000 koraka. Izlaz modela bio je `((.)* > field > (.)* > town > (.)* > field){#some}`.

Na slici 4.4 je zanimljiv nagli skok u nagradi po koraku: agent u prvoj polovici treniranja ne postiže dobre rezultate, onda u kratkom periodu nagrada skoči do maksimuma za taj zadatak, te se stabilizira. Jedan mogući uzrok takvog ponašanja učenja je broj stanja SN koji je pronađen za pripadni upit, kao i semantika regularnog izraza kojeg je vratio model: `((.)* > field > (.)* > town > (.)* > field){#some}`. U ovom slučaju, `#some` označava ponavljanje 2 puta, što znači da agent za pozitivnu nagradu mora ostvariti dugačak niz događaja.

Najzanimljivija usporedba dvaju grafova je da složeniji upit usporava konvergenciju algoritma za gotovo cijeli red veličine. U jednu ruku to je nužno: kompliciraniji zadatak je teže naučiti. Ali također treba naglasiti da pronađeni stroj nagrade za upit na slici 4.4 (`patrol the field and the town`) ima 45 stanja, dok onaj za upit na slici 4.5 (`find a key`) ima samo 5 stanja. Taj manji SN je prikazan na slici 4.6.

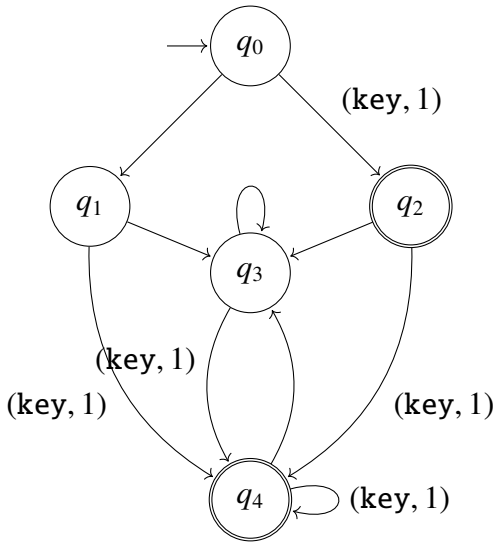
QRM algoritam informaciju o vrijednosti propagira unatrag kroz stanja stroja nagrade (to je glavni praktični razlog zašto se izlaz modela, to jest regularni izraz, ne koristi direktno, nego ga se pretvara u stroj nagrade). Iz toga slijedi da između dva stroja nagrade s istom semantikom treba preferirati manji (onaj koji ima manji broj stanja), jer bi se time ubrzala konvergencija algoritma (i smanjila potrebna količina radne memorije). Shodno tome, jedan način na koji bi se učenje agenta moglo ubrzati je uvođenje dodatnog koraka: minimizacije pronađenog stroja nagrade (u broju stanja). Minimizacija stroja nagrade može se svesti na problem mješovitog cjelobrojnog linearnog programiranja, te je



Slika 4.5: Rezultati algoritma QRM za upit `find a key`. Izlaz modela bio je `(.)* > key`.

implementirana u sklopu rada na SRMI algoritmu i dostupna je na GitHub repozitoriju.[1] Uporaba ove metode bila bi prioritetnija u kompliciranijim okruženjima gdje bi vrijeme utrošeno na rješavanje problema cjelobrojnog programiranja bilo isplativije.

[1] `https://github.com/corazza/stochastic-reward-machines/blob/master/src/rl_agents/jirp/mip_approx.py`



Slika 4.6: Stroj nagrade za upit `find a key`. Radi bolje čitljivosti, detalji funkcije prijelaza i izlaza prikazani su samo na prijelazima koji donose pozitivnu nagradu (svi ostali prijelazi donose nagradu 0). Rezultati pokretanja algoritma QRM s ovim SN prikazani su na slici 4.5. Sasvim je jasno da ni ovaj SN nije optimalan, jer postoji SN koji ima istu semantiku ali samo 2 stanja.

Zaključak

U diplomskom radu predstavljeni su problemi u kojima se od agenta zahtijeva da ostvari kompleksan niz koraka, odnosno problemi u kojima nagrada na složen način ovisi o cijeloj povijesti interakcije u MPO. Klasični algoritmi za učenje s potporom ne mogu riješiti takve probleme. Na primjer, Q-učenje pretpostavlja da se informacija o vrijednosti može zapisati kao funkcija parova stanja i poteza, pa se time gubi podatak o povijesti.

Kao alternativa predloženi su strojevi nagrade i algoritam QRM. U svojoj srži, strojevi nagrade opremaju MPO oblikom konačne memorije, odnosno omogućuju računanje nagrade za proizvoljno duge interakcije. Algoritam QRM iskorištava strukturu konačnih automata kako bi ubrzao proces učenja. Iako su time riješeni fundamentalni problemi u učenju s potporom u slučaju gdje funkcija nagrade ne poštuje Markovljevo svojstvo, zadavati strojeve nagrade je i dalje težak posao. Zbog toga se javlja pitanje kako strojeve nagrade otkriti pasivno (naučiti zajedno s ponašanjem), ili kako ih zadati lakšim putem.

Glavni doprinos diplomskog rada je razvoj jezičnog modela koji omogućuje zadavanje strojeva nagrade putem opisivanja zadataka za agenta u prirodnom jeziku. Konkretnije, doprinosi uključuju razvoj metoda za proširivanje skupa podataka za treniranje, razvoj međujezika temeljenog na regularnim izrazima, izmjenu metode treniranja kako bi se unijelo domensko znanje, te razvoj metoda za ispravljanje grešaka jezičnog modela. Model na validacijskom skupu podataka ostvaruje BLEU rezultat od 86%, što se smatra vrlo dobrim rezultatom.[2] Model je demonstriran kroz aplikaciju u kojoj korisnik u prirodnom jeziku može opisati željeni zadatak (to jest ono što agent mora postići), te može vidjeti agenta koji opisani zadatak obavlja u okruženju. Time je ostvaren glavni cilj diplomskog rada: olakšan je rad s funkcijama nagrade koje ne zadovoljavaju Markovljevo svojstvo. No također je napravljen korak k spajanju naizgled raznorodnih tehnologija: velikih jezičnih modela i formalnih metoda, što je zanimljivo i novo područje istraživanja.

[2] No u ovom slučaju ta mjera može ima mane, relevantna analiza je u poglavlju 4.

Bibliografija

[1] Luka Baković, *Potporno učenje za robotske manipulatore*, 2019, https://urn.nsk.hr/urn:nbn:hr:168:021228.

[2] Jan Corazza, Ivan Gavran i Daniel Neider, *Reinforcement Learning with Stochastic Reward Machines*, Proceedings of the AAAI Conference on Artificial Intelligence **36** (2022), br. 6, 6429–6436, https://ojs.aaai.org/index.php/AAAI/article/view/20594.

[3] Ivan Gavran, Eva Darulova i Rupak Majumdar, *Interactive Synthesis of Temporal Specifications from Examples and Natural Language*, Proc. ACM Program. Lang. **4** (2020), br. OOPSLA, https://doi.org/10.1145/3428269.

[4] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano i Sheila McIlraith, *Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning*, Proceedings of the 35th International Conference on Machine Learning (Jennifer Dy i Andreas Krause, ur.), Proceedings of Machine Learning Research, sv. 80, PMLR, 10–15 Jul 2018, str. 2107–2116, https://proceedings.mlr.press/v80/icarte18a.html.

[5] Shivaram Kalyanakrishnan, *Theoretical Analysis of Policy Iteration*, IJCAI 2017 Tutorial, 2017, https://www.cse.iitb.ac.in/~shivaram/resources/ijcai-2017-tutorial-policyiteration/tapi.pdf.

[6] Aran Komatsuzaki, *One Epoch Is All You Need*, 2019.

[7] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov i Evgeny Burnaev, *Reinforcement Learning for Combinatorial Optimization: A Survey*, 2020.

[8] Kishore Papineni, Salim Roukos, Todd Ward i Wei Jing Zhu, *Bleu: a Method for Automatic Evaluation of Machine Translation*, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (Philadelphia, Pennsylvania, USA), Association for Computational Linguistics, srpanj 2002, str. 311–318, https://aclanthology.org/P02-1040.

[9] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek i Sumit Gulwani, *Synchromesh: Reliable code generation from pre-trained language models*, 2022.

[10] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei i Ilya Sutskever, *Language Models are Unsupervised Multitask Learners*, (2019).

[11] Torsten Scholak, Nathan Schucher i Dzmitry Bahdanau, *PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models*, 2021.

[12] Noam Shazeer i Mitchell Stern, *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost*, 2018.

[13] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu i Bo Wu, *Joint Inference of Reward Machines and Policies for Reinforcement Learning*, CoRR **abs/1909.05912** (2019), `http://arxiv.org/abs/1909.05912`.

[14] Jingqing Zhang, Yao Zhao, Mohammad Saleh i Peter J. Liu, *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*, CoRR **abs/1912.08777** (2019), `http://arxiv.org/abs/1912.08777`.

Sažetak

Zadaci koji od agenta zahtijevaju ostvarivanje kompleksnog niza događaja u okruženju često se javljaju u praksi, no funkcije nagrade koje obuhvaćaju takve zadatke ne zadovoljavaju Markovljevo svojstvo, koje je nužno za osiguravanje točnosti i konvergencije klasičnih algoritama za učenje s potporom.

U radu je uveden pojam strojeva nagrade te posebnog algoritma za potporno učenje: *Q-learning for Reward Machines* (QRM), koji je namijenjen radu sa strojevima nagrade. Strojevi nagrade su vrsta konačnih automata kojima je moguće zapisati funkcije nagrade koje ne zadovoljavaju Markovljevo svojstvo, te se mogu shvatiti kao oblik konačne memorije u Markovljevom procesu odlučivanja. Nažalost, pisanje takvih formalizama je mukotrpan posao, te se javila potreba da se taj posao olakša. Neke metode tome pristupaju putem učenja automata paralelno uz učenje ponašanja agenta, no moraju pretpostaviti da je signal nagrade unaprijed definiran i da se podvrgava strukturi konačnog automata.

U diplomskom radu istražen je drugi pristup tom problemu: korištenje novih tehnologija iz područja razumijevanja prirodnog jezika, s ciljem da inženjerima i istraživačima olakša sam postupak definiranja signala nagrade. U tu svrhu dotreniran je jezični model temeljen na poznatom modelu GPT-2, te je detaljno opisan proces proširenja skupa podataka za treniranje. Krajnji rezultat je omogućio sljedeće korake.

1. Korisnik opisuje zadatak na Engleskom jeziku, na primjer *"patrol the forest and the factory, but avoid traps"*.
2. Poziva se jezični model koji prevodi korisnikov opis u formalizam stroja nagrade.
3. Koristi se QRM i stroj nagrade iz koraka 2 da bi agent naučio obavljati zadatak kojeg je korisnik opisao u koraku 1.
4. Rezultati se korisniku demonstriraju vizualno na malom broju epizoda.

Rad predstavlja i analizira rezultate treniranja jezičnog modela, kao i rezultate QRM algoritma koji je uparen sa strojevima nagrade pronađenim pomoću modela. Također je razvijeno grafičko korisničko sučelje, pomoću kojega se demonstrira uporaba jezičnog modela i rezultati treniranja agenta.

Summary

Tasks that require an agent to accomplish a complex sequence of events in the environment are common in practice. However, reward functions that capture such tasks do not satisfy the Markov property, which is necessary to ensure the correctness and convergence properties of classical reinforcement learning algorithms.

This thesis introduces the concept of reward machines and a related reinforcement learning algorithm: *Q-learning for Reward Machines* (QRM). Reward machines are a type of finite automaton that can capture reward functions that do not satisfy the Markov property. They can be understood as a form of finite memory in the Markov decision process. Unfortunately, writing such formalisms is a difficult manual task, and there is a need to simplify this process. Some methods approach this by learning automata jointly with the policy, but they assume a predefined reward signal that conforms to a finite-state structure.

This thesis explores a different approach: using new technologies from the field of natural language processing, aiming to facilitate the process of defining reward signals. For this purpose, a language model based on the well-known GPT-2 model was fine-tuned, and the process of data augmentation that was employed to create the training dataset is described in detail. The final result enables the following steps.

1. The user describes the task in English, for example, *"patrol the forest and the factory, but avoid traps"*.

2. The language model is invoked to translate the user's description into a reward machine formalism.

3. QRM and the reward machine from Step 2 are used to train the agent to perform the task described by the user in Step 1.

4. The results are visually demonstrated to the user in a small number of episodes.

The thesis presents and analyzes the results of training the language model, as well as the results of the QRM algorithm paired with the reward machines discovered using the model. Additionally, a graphical user interface has been developed to demonstrate the usage of the language model and the training results of the agent.

Životopis

Rođen sam 16. travnja 1996. godine u Zagrebu. Pohađao sam osnovnu školu Gustava Krkleca u Travnom, te sudjelovao na državnim natjecanjima iz robotike. 2015. godine završio sam srednjoškolsko obrazovanje u V. gimnaziji u Zagrebu. Sudjelovao sam na natjecanjima iz informatike, biologije, i astronomije, a u trećem razredu srednje škole predstavljao sam svoju gimnaziju na međunarodnom natjecanju American Computer Science League u Denveru. 2015. godine upisao sam Fakultet Elektrotehnike i Računarstva, no 2017. godine prebacio sam se na preddiplomski studij matematike na Prirodoslovno Matematičkom Fakultetu Sveučilišta u Zagrebu, gdje sam 2021. godine upisao diplomski studij Računarstvo i Matematika. Tijekom diplomskog studija surađivao sam sa znanstvenicima na Max Planck Institutu u Kaiserslauternu, što je rezultiralo objavljivanjem znanstvenog rada u području strojnog učenja na AAAI konferenciji, 2022. godine.