Učenje s potporom u optimalnom oblikovanju struktura

Stojanović, Mislav

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Mislav Stojanović

UČENJE S POTPOROM U OPTIMALNOM OBLIKOVANJU STRUKTURA

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, Rujan, 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

Posljednjih 10 godina odvija se *boom* u području umjetne inteligencije predvođen dubokim učenjem. Metode dubokog učenja pogurale su unaprijed i učenje s potporom. Ako se optimizacijski problem može modelirati kao Markovljev proces odlučivanja, duboko učenje s potporom omogućava uporabu hardverskih akceleratora za rješavanje problema odlučivanja.

Ukoliko tražimo aproksimaciju neke funkcije (rješenja parcijalne diferencijalne jednadžbe) u prostoru po dijelovima polinomijalnih funkcija, sama particija domene direktno određuje uspješnost aproksimacije. Tako optimizacijski problem odabira particije formuliran kao problem odlučivanja može se rješavati metodama dubokog učenja s potporom.

Agent koji je rezultat dubokog učenja s potporom, tako iz značajki trenutne particije domene zaključuje gdje je potrebno detaljnije particionirati domenu u svrhu poboljšanja po dijelovima polinomijalne aproksimacije. Budući da su funkcije koje aproksimiramo glatke, svrha agenta je naučiti vezu između razlika u susjednim polinomima na domeni i kvalitete aproksimacije koju ti polinomi skupa izgrađuju. Shodno tome, gdje do poboljšanja detaljnim particioniranjem neće doći, može odabrati uštediti procesorske resurse.

Postojeći rezultati nadziranog dubokog učenja u rješavanju parcijalnih diferencijalnih jednadžbi [19] i dubokog učenja s potporom u rješavanju problema diskretne optimizacije [10] nagovještavaju mogućnosti dubokog učenja u rješavanju problema kombinatorne optimizacije čija je dinamika opisana parcijalnim diferencijalnim jednadžbama.

U prvom dijelu iznosimo teorijske osnove učenja s potporom. Definiramo Markovljev proces odlučivanja i postavljamo Bellmanove jednadžbe. Zatim se okrećemo nekim algoritmima učenja s potporom i tako stvaramo vezu s dubokim učenjem. Na kraju prvog dijela se osvrćemo na pretraživanje stabla Monte Carlo metodom i napredne algoritme dubokog učenja s potporom temeljene na toj metodi.

U drugom dijelu iskazujemo osnovnu teoriju koja opisuje probleme koje ćemo rješavati. Formuliramo problem adaptacije po dijelovima polinomijalnih aproksimacija rješenja parcijalnih diferencijalnih jednadžbi i prikazujemo ga kao Markovljev proces odlučivanja. Uvodimo prostor funkcija potreban za razmatranje problema na metričkom grafu. Kratko se osvrćemo na moguću uporabu geometrijskog dubokog učenja.

U posljednjem dijelu strogo oblikujemo zadatak i iznosimo informacije potrebne za im-

plementaciju na računalu. Prvo za problem na segmentu u jednoj dimenziji, a zatim detaljnije o problemu na metričkom grafu. Naposljetku, nakon opisa teorijske pozadine, prezentiramo rezultate eksperimenata upotrebe metoda dubokog učenja s potporom za rješavanje problema adaptacije po dijelu polinomijalnih funkcija koje aproksimiraju rješenja parcijalnih diferencijalnih jednadžbi. Prvo *Actor-Critic* za djelomično uočljiv Markovljev proces odlučivanja, a zatim *MuZero* na globalnoj slici stanja.

U sklopu rada izrađena su odgovarajuća programska rješenja u *Pythonu* koja koriste suvremene biblioteke za numeričku matematiku, učenje s potporom i duboko učenje.[1]

[1] Programski kod implementacija može se pronaći na `https://github.com/MislavGT/diplomski`.

Poglavlje 1

Učenje s potporom

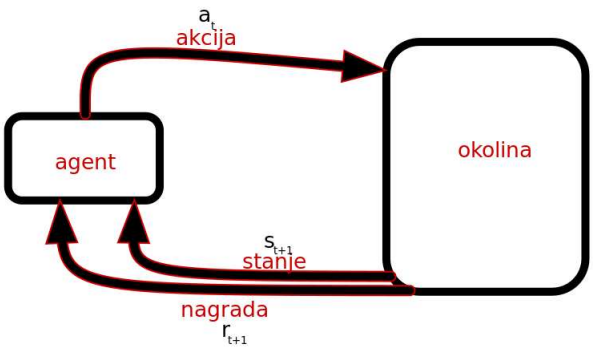
Učenje s potporom[1] jedna je od tri paradigme, uz nadzirano i nenadzirano učenje, u strojnom učenju. Cilj učenja s potporom naučiti je preslikavanje stanja u odluke koje maksimizira signal numeričke nagrade. Usko je povezano uz teoriju igara, a neka od najvećih dostignuća u području učenja s potporom upravo su u razvoju složenih algoritama umjetne inteligencije za igranje kompliciranih igara na ljudskoj i nadljudskoj razini. Osim toga, pronalazi primjene u teoriji upravljanja, operacijskim istraživanjima, financijskom inženjerstvu... Općenito, kada se učenje s potporom koristi kao optimizacijska metoda u optimizaciji temeljenoj na simulaciji, najlakše je zamisliti cijeli proces kao igranje igre. To može biti igra s jednim ili više igrača, koji surađuju ili se međusobno natječu. Ranije spomenuti signal numeričke nagrade je ishod igre, a cilj algoritma je naučiti donositi odluke koje prate promjenjivo stanje igre i tako doći do željenih rezultata. Igra je naš problem i definira stanja, a naše rješenje je agent koji ju igra i njega poistovjećujemo s odlukama ili akcijama.

1.1 Osnovni pojmovi

Učenje s potporom sastoji se od *agenta* i *okoline*. Još možemo spomenuti sljedeće pojmove koji određuju učenje s potporom:

- *Politika*[2] je preslikavanje agentovih opažanja (opažanje je njegova percepcija stanja) u akcije.
- *Nagrada* je signal koji agent prima nakon akcije i on ga mora usmjeravati željenome cilju.
- *Funkcija vrijednosti stanja* je očekivana suma nagrada i tako daje dugoročnu ocjenu.

[1] učenje s podrškom, podržano učenje, *engl.* reinforcement learning
[2] *engl.* policy



Slika 1.1: Jednostavni prikaz učenja s potporom

- *Model* o kojem ćemo više kasnije u ovom poglavlju.

Prvo ćemo definirati *Markovljev proces odlučivanja*.

Definicija 1.1.1. *Konačan Markovljev proces odlučivanja uređena je petorka* $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$*:*

- $\mathcal{S}$ *je konačan skup stanja igre;*
- $\mathcal{A}$ *je konačan skup akcija (spomenimo i da nisu svi potezi u svakom stanju legalni). U našim razmatranjima ovo je diskretan skup i u implementacijama svaka akcija predstavljena je prirodnim brojem;*
- $\mathcal{R} \subseteq \mathbb{R}$ *je konačan skup nagrada;*
- $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ *je funkcija dinamike koja za sve* $s', s \in \mathcal{S}, r \in \mathcal{R}$ *i* $a \in \mathcal{A}(s)$ *vrijedi:*

$$p(s', r|s, a) := \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a). \tag{1.1}$$

Napomenimo da p definira vjerojatnosnu distribuciju i vrijedi:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \text{za sve } s \in S \text{ i } a \in \mathcal{A}(s). \tag{1.2}$$

- $\gamma \in \langle 0, 1]$ *je diskontirajući faktor.*

Prijelazi između stanja zadovoljavaju Markovljevo svojstvo:

$$\begin{aligned}\mathbb{P}(S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1} | S_0 = s_0, A_0 = a_0, \ldots, S_t = s_t, A_t = a_t) \\ = \mathbb{P}(S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1} | S_t = s_t, A_t = a_t).\end{aligned} \tag{1.3}$$

Funkcija dinamike p u potpunosti opisuje okolinu.
Pomoću nje možemo definirati

- *Vjerojatnosnu funkciju prijelaza* $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$\mathcal{P}(s'|s, a) := \mathbb{P}(S_t = s' | S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r|s, a), \quad s', s \in S, a \in \mathcal{A}(S). \tag{1.4}$$

- *Funkciju nagrade* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s, a) := \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a), \quad s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{1.5}$$

- Možemo iz vjerojatnosne funkcije prijelaza odmah izvesti i *funkciju nagrade* koja uzima u obzir novonastalo stanje, preciznije rečeno pripadnu vjerojatnost:

$$r(s, a, s') := \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{\mathcal{P}(s'|s, a)}, \quad s, s' \in \mathcal{S}, a \in \mathcal{A}(s). \tag{1.6}$$

Politika $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ pridružuje paru akcije i stanja vjerojatnost odabira te akcije u tom stanju. *Deterministička politika*, kodomena je $\{0, 1\}$, kraće se zapisuje $\pi : \mathcal{S} \to \mathcal{A}$.
Cilj agenta je maksimirati očekivanje diskontirane kumulativne nagrade koju zovemo *povrat*:

$$G_t = R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1.7}$$

γ kontrolira preferiranje bližih nad daljim nagradama i garantira konvergenciju reda (1.7).
Sada možemo reći da je naš optimizacijski problem:

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[G_t]. \tag{1.8}$$

Definirajmo sada *funkciju vrijednosti stanja* $v_\pi : \mathcal{S} \to \mathbb{R}$, za neku politiku π:

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi[G_t | S_t = s] && (1.9)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] && (1.10)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] && (1.11)\\ &= \sum_a \pi(a, s) \sum_{s', r} p(s', r|s, a) \cdot (r + \gamma v_\pi(s')). && (1.12)\end{aligned}$$

v_π nam za početno stanje s računa očekivani povrat uz danu politiku π.
(1.12) je *Bellmanova jednadžba za* v_π. Govori nam o odnosu vrijednosti stanja i vrijednosti budućih stanja. Gleda prosjek po svim mogućnostima, uzimajući u obzir vjerojatnost pojedinog ishoda. Na njoj se temelje metode dinamičkog programiranja koje se koriste za aproksimiranje v_π.
Funkcija vrijednosti akcije za politiku π je:

$$\begin{aligned} q_\pi(s,a) &:= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] && (1.13)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] && (1.14)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] && (1.15)\\ &= \sum_{s',r} p(s', r|s, a) \cdot (r + \gamma v_\pi(s')). && (1.16) \end{aligned}$$

Slično kao ranije, q_π računa očekivani povrat uređenog para početnog stanja i akcije (s, a) uz politiku π.
Riješiti problem učenja s potporom znači pronaći politiku koja dugoročno donosi velike nagrade. Za konačne Markovljeve probleme odlučivanja, možemo precizno reći da je neka politika π bolja od neke druge politike π', u oznaci $\pi \geq \pi'$. Funkcije vrijednosti definiraju parcijalni uređaj na skupu politika.

Definicija 1.1.2. *Politika* π *je* bolja ili jednaka *od politike* π' *ako*

$$\forall s \in \mathcal{S} \qquad v_\pi(s) \geq v_{\pi'}(s). \tag{1.17}$$

Postoji (vidi [29]) optimalna politika. Može ih biti više, no sve ih označavamo π_*. Njima odgovara ista funkcija vrijednosti stanja, *optimalna funkcija vrijednosti stanja*

$$q_*(s,a) := \max_\pi q_\pi(s,a), \qquad s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{1.18}$$

Veza između njih je

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{1.19}$$

Kada bismo znali v_*, optimalna politika bi jedostavno bila ona koja u svakom koraku daje pozitivne vjerojatnosti stanjima s za koje je $v_*(s)$ maksimalan. Dakle potrebno je samo pogledati jedan korak unaprijed. Kada bismo znali q_*, ne bismo morali gledati niti korak unaprijed, nego odmah znamo vrijednosti akcija i odaberemo najveći broj.
Zamislimo da smo u stanju s i politika π nam govori da poduzmemo akciju a, i v_π nam je poznato. Pogledajmo što ako ipak odaberemo akciju a'. Usporedbom $q_\pi(s, a')$ i $v_\pi(s)$ možemo vidjeti trebamo li umjesto π pratiti politiku π' gdje je sve isto, osim što je $\pi'(s) = a'$. Tako dolazimo do sljedećeg rezultata:

Teorem 1.1.3. (Teorem poboljšanja politike). *Neka su π i π' deterministčke politike takve da za svaki $s \in \mathcal{S}$ vrijedi*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{1.20}$$

Tada je politika π' bolja ili jednaka politici π: za svaki $s \in \mathcal{S}$ vrijedi

$$v_{\pi'}(s) \geq v_\pi(s). \tag{1.21}$$

Za neko stanje, stroga nejednakost u (1.20) *povlači strogu nejednakost* (1.21).

1.2 Dinamičko programiranje

Kasnije ćemo govoriti o dubokom učenju u kontekstu učenja s potporom. U dubokom učenju, umjetne neuronske mreže iterativno se poboljšavaju gradijentnim spustom ili usponom. Stoga je važno postaviti temelje iterativnih metoda u učenju s potporom. Želimo koristiti funkcije vrijednosti kako bismo strukturirali potragu za politikama koje daju dobre rezultate.
Pogledajmo *Bellmanove jednadžbe optimalnosti* koje *optimalna funkcija vrijednosti stanja* i *optimalna funkcija vrijednosti akcije* zadovoljavaju:

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}|S_t = s, A_t = a] \tag{1.22}$$

$$= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{1.23}$$

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \tag{1.24}$$

$$= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')]. \tag{1.25}$$

Za neku politiku π, računanje v_π (1.12) zovemo *procjena politike*. Ako znamo sva pravila igre, problem je zapravo sustav $|\mathcal{S}|$ linearnih jednadžbi.
Pogledajmo iterativnu metodu. Uz početnu aproksimaciju v_0

$$v_{k+1}(s) := \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] \tag{1.26}$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')], \tag{1.27}$$

za sve $s \in \mathcal{S}$. $v_k = v_\pi$ je fiksna točka. Niz $\{v_k\}_k$ konvergira pod istim uvjetima pod kojima v_π postoji. Ovaj postupak zove se *iterativna procjena politike*. Sada koristeći teorem 1.1.3 i prije njega objašnjen pristup možemo izgraditi postupak koji nazivamo *iteracija politike*.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

E predstavlja procjenu (evaluation), a I poboljšanje (improvement).[3] Analogno možemo definirati *iteraciju vrijednosti* za v_{kk} iz (1.12).

1.3 Duboko učenje s potporom

Duboko učenje klasa je algoritama strojnog učenja koji koriste umjetne neuronske mreže za preslikavanje signala (u našem slučaju opažanje stanja) u željeni rezultat (u našem slučaju odluka). Umjetne neuronske mreže pokazale su se uspješnima u računanju kompliciranim visokodimenzionalnim podacima. U učenju s potporom koriste se kao procjenitelji ranije definiranih funkcija.
Ključne su metode konveksne optimizacije. Kod nadziranog učenja, imamo skup podataka pomoću kojega dolazimo do optimizacijskog problema. U učenju s potporom, nemamo skup podataka, nego prikupljamo opažanja i numeričke nagrade prilikom interakcija s okolinom. Zatim prikupljeno koristimo za optimizaciju našeg procjenitelja.

Metode učenja s potporom temeljene na umjetnim neuronskim mrežama postigle su značajne uspjehe u igranju igara poput šaha, goa, Starcrafta II, Dote 2... (više u [28] [27]).

1.4 Actor-Critic metode

Česte su metode temeljene na *funkciji vrijednosti akcija* koje uče vrijednosti akcija i zatim odabiru akcije na temelju svojih procjena. Te procjene zapravo definiraju politiku. Sada ćemo pogledati metode koje uče samu politiku. Politika je u ovom slučaja predstavljena nekim parametrima. Pomoću tih parametara odabire se akcija bez da se procjenjuje vrijednost. Parametrizacije mogu se nositi s beskonačnim, kontinuiranim prostorima stanja.

Neka je $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ vektor parametara politike. Vjerojatnost da će agent poduzeti akciju a u trenutku t u stanju s uz parametar $\boldsymbol{\theta}$ je $\pi(a|s, \boldsymbol{\theta}) = Pr\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$.
Uz oznaku funkcije cilja koju želimo maksimizirati $J(\boldsymbol{\theta})$, u terminima gradijentnog uspona imamo

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)} \tag{1.28}$$

gdje je $\widehat{\nabla J(\boldsymbol{\theta}_t)} \in \mathbb{R}^d$ stohastička procjena čije očekivanje aproksimira gradijent funkcije cilja s obzirom na $\boldsymbol{\theta}$. Ovakve metode zovemo *metode gradijenta politike (*engl. *policy gradient methods)*. Metode koje uče i procjenu politike i procjenu funkcije vrijednosti zovu se *actor-critic metode* jer *actor* predstavlja naučenu politiku, a *critic* naučenu funkciju vrijednosti, najčešće funkciju vrijednosti stanja.

[3] Grafički prikaz inspiriran [29].

$\pi(a|s, \boldsymbol{\theta})$ mora biti diferencijabilna po parametrima. Dakle $\nabla\pi(a|s, \boldsymbol{\theta})$, što je vektor-stupac parcijalnih derivacija po komponentama $\boldsymbol{\theta}$, postoji i konačan je za sve $s \in \mathcal{S}, a \in \mathcal{A}(s)$ i $\boldsymbol{\theta} \in \mathbb{R}^{d'}$.

Ako svakom uređenom paru stanja i akcije (s, a) uz parametre $\boldsymbol{\theta}$ dodijelimo numerički prioritet $h(s, a, \boldsymbol{\theta})$, tada primjerice možemo doći do vjerojatnosti *soft-max distribucijom*

$$\pi(a|s, \boldsymbol{\theta}) := \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}} \tag{1.29}$$

i ovakvu parametrizaciju politike zovemo *soft-max za prioritete akcija*.

Još jedan način za parametrizaciju prioriteta akcija su umjetne neuronske mreže. Tada je $\boldsymbol{\theta}$ vektor težina neuronske mreže.

Također je potrebna i diferencijabilna parametrizacija funkcije vrijednosti stanja. Za vektor $\mathbf{w}$, imamo $\hat{v}(s, \mathbf{w})$.

Uzorkovanje trajektorija način je prikupljanja iskustva u kojem politika daje akcije, okolina odgovara stanjima i nagradama. Tako se generiraju nizovi stanja i akcija, te sukladno tome ažurira se politika. Kasnije ćemo vidjeti kako trajektorije možemo koristiti za učenje neuronskih mreža. *Rollout algoritmi* algoritmi su koji iz nekog stanja generiraju trajektorije za svaku moguću akciju u tom trenutku, a nakon toga prate trenutnu politiku. Procjenjuju funkciju vrijednosti akcija politike tako što usrednjuju dobivene povrate.

Bootstrapping i n-step

Za neko najosnovnije ažuriranje

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R_{t+1} + \gamma Q(s', a') - Q(s, a)) \tag{1.30}$$

koristimo Q za ažuriranje Q. Stoga je ovo *bootstrapping* metoda. Malo smo informacija dobili iz okoline (samo R_{t+1}, stoga je ovakav pristup pristran i vrlo ovisan o inicijalizaciji.

S druge strane, gledanjem daleko u budućnost

$$Q(s, a) \leftarrow Q(s, a) + \alpha(G_t - Q(s, a)) \tag{1.31}$$

koristimo isključivo informacije iz okoline. Ova metoda dakle ima visoku varijancu i sporu konvergenciju jer moramo koristiti puno stvarnih informacija.

U praksi, najbolje rezultate daju kombinirani pristupi koji koriste činjenicu

$$G_{t:t+n} := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}). \tag{1.32}$$

gdje smo odabrali koliko koraka (*n-step*) ćemo gledati u budućnost za stvarne podatke, a ostatak je rezultat *bootstrappinga*.

Metode gradijenta politike

Metoda gradijenta politike temelji se na uzorkovanju trajektorija $\{\tau^i\}$ iz politike $\pi_\theta(a_t|s_t)$. Tako dobivamo aproksimaciju

$$\nabla_\theta J(\theta) \approx \sum_i (\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i))(\sum_{t'=t}^{T} r(s_{t'}^i a_{t'}^i)). \tag{1.33}$$

Dakle, redom

- Uzorkujemo trajektoriju.
- Sumiramo odgovarajuće nagrade.
- Za svaki korak, računamo gradijent *log-vjerojatnosti* akcija.
- Množimo gradijent s *povratom* i koristimo rezultat za ažuriranje.

Actor-Critic metoda funkcionira slično. O varijantama i detaljima više u [29].

- Uzorkujemo stanja i akcije (i nagrade) trenutnom politikom.
- Koristimo nadzirano učenje za aproksimaciju *funkcije vrijednosti stanja* korištenjem uzorka (ovo je zapravo Monte Carlo pristup jer koristimo uzorak).
- Možemo doći do *funkcije vrijednosti akcije*

  $$Q(s_i, a_i) = r(s_i, a_i) + V(s_i') - V(s_i). \tag{1.34}$$

  Ovdje i označava da se radi o članu uzorka, dok s_i' označava stanje koje slijedi nakon s_i uz odabir a_i.
- Možemo ponovno aproksimirati gradijent za korak spusta

  $$\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) Q(s_i, a_i). \tag{1.35}$$

1.5 Pretraživanje stabla Monte Carlo metodom

Monte Carlo algoritmi temelje se na ponavljanju slučajnog uzorkovanja. Nakon determinističkog vremena daju stohastički odgovor. U kontekstu pretraživanja stabla to znači da se generiraju grane (trajektorije) kojima čvorovi predstavljaju stanja, a bridovi odluke. MCTS (*engl.* Monte Carlo Tree Search) spada pod *decision-time planning* koje se tako zove jer se odabir akcije izračuna kada se dođe u novo stanje i agent treba donijeti odluku (više u [31]). Jasna je veza s *uzorkovanjem trajektorija*. Stablo je zapravo skupina trajektorija, a kada su te trajektorije rezultat uzorkovanja iz neke distribucije, radi se o Monte Carlo metodi.

Izgradnja stabla

Budući da za veliki prostor akcija $\mathcal{A}$ nije vremenski izvedivo provjeriti sve moguće ishode, želimo provjeriti samo one koji obećavaju. Ideja je da kroz izgradnju stabla dobivamo sve bolju sliku stvarnosti. Gradimo onoliko stabla koliko nam vrijeme i memorija dopuštaju. Koraci u izgradnji stabla su:[4]

- 1. *Selekcija*: Počevši od korijena, rekurzivno se primjenjuje politika odabira djeteta radi spuštanja niz stablo dok se ne dođe do čvora koji predstavlja završno stanje ili *proširiv* čvor. Čvor je *proširiv* ako nije završno stanje i ima djece koju još nismo posjetili. Ako se radi o završnom stanju, idemo na korak 4.
- 2. *Proširivanje*: Odaberemo neke akcije i sukladno tome dodamo djecu. Tako kreiramo listove.
- 3. *Simulacija*: Izvodi se *rollout* iz dodanih čvorova do završnih stanja.
- 4. *Propagiranje unatrag* (*engl.* Backup ili Backpropagation): Povrat novog čvora propagira se unatrag kroz trajektoriju iz 1. koraka. Čvorovi u sebi nose podatke o broju posjeta i nagradama. U ovom koraku ih ažuriramo.

1. i 2. korak koriste *politiku stabla* (*engl.* tree policy), a 3. *zadanu politiku* (*engl.* rollout policy). Broj iteracija gornje procedure ovisi procesorskim resursima. Zatim se odabire akcija u stanju korijena na temelju izgrađenog stabla, i postupak kreće ponovno, s time da možemo sačuvati podstablo čiji je korijen sljedeće stanje i nastaviti proceduru od njega.

1.6 MCTS i duboko učenje

Spomenuli smo *politiku stabla* i *zadanu politiku*. Upravo ovdje izvanredne rezultate polučile su metode dubokog učenja (detaljnije u [28]).

Istraživanje i iskorištavanje

Jedan problem učenja s potporom je balansiranje između detaljnijeg proučavanja obećavajućih akcija i istraživanja neistraženih akcija. Najpoznatiji primjer toga su *bandit problemi* (vidi [7]). Imamo k odabira, iza svakog se krije distribucija o kojoj ovisi nagrada. Koliko ćemo vremena provesti učeći distribucije, a koliko ćemo odabirati onu distribuciju koja nam se u

[4] Ovaj dio je prikazan kao u [21].

tom trenutku čini najpovoljnija?
Definirajmo *žaljenje* (*engl.* regret)

$$R_n := \mu^* n - \sum_{j=1}^{K} \mu_j \mathbb{E}[T_j(n)], \tag{1.36}$$

gdje je μ^* najveća nagrada, a $T_j(n)$ broj odabira j u prvih n koraka.
UCB (*(engl.)* Upper Confidence Bound) strategija nalaže da se prvo odabere svaka akcija po jednom. Zatim se odabire $\underset{j}{\text{argmax}}(\overline{X}_{j,T_j}(n) + \sqrt{\frac{2lnn}{T_j(n)}})$, gdje je $\overline{X}_{j,T_j}(n) = \frac{1}{n}\sum_{t=1}^{n} X_{j,t}$ prosječna nagrada akcije j. Taj pribrojnik sugerira odabir najveće poznate nagrade (iskorištavanje, *engl.* exploitation). Dok drugi pribrojnik potiče istraživanje (*engl.* exploration).

AlphaZero koristi formulu *PUCT* (*engl.* Predictor + Upper Confidence Bound for Trees):

$$PUCT(s,a) = Q(s,a) + U(s,a).$$

$Q(s,a)$ je usrednjena funkcija vrijednosti akcije izvedena iz simulacija MCTS-a. Drugi pribrojnik računamo:

$$U(s,a) = c_{PUCT} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}. \tag{1.37}$$

$N(s,a)$ je broj posjeta koji čuvamo i ažuriramo u čvoru tijekom MCTS-a. $P(s,a)$ je rezultat umjetne neuronske mreže koju treniramo i govori nam vjerojatnosti mogućih akcija za trenutno stanje te tako usmjerava stablo. c_{PUCT} balansira između istraživanja novih trajektorija i odabira akcija za koje se misli da su dobre. Zapravo se ne radi o hiperparametru, nego o funkciji koja ovisi o hiperparametrima i N te sporo raste. Brid a u stanju s koji se odabire je $\underset{a}{\text{argmax}}(Q(s,a) + U(s,a))$.
Spomenimo još kratko da postoje razna unaprijeđenja i generalizacije, primjerice Monte Carlo Graph Search [8]. Dok u MCTS-u je čvor zapravo uređen par stanja i akcije jer dijete može biti rezultat isključivo jednog brida, MCGS dopušta da sve akcije koje rezultiraju istim stanjem kao bridovi ulaze u isti čvor i tako se gradi aciklički graf.

Model-free

Općenito u učenju s potporom razlikujemo algoritme koji ovise o interakcijama s okolinom i algoritme koji simuliraju okolinu. Ovi prvi su *model-free*, a drugi *model-based*. Najpoznatiji primjer *model-free* algoritma koji koristi MCTS i duboko učenje je *AlphaZero* tvrtke Google DeepMind.

Model-based

U učenju s potporom, model je agentova metoda prepoznavanja kako će se stanje promijeniti nakon hipotetske akcije. Model preslikava stanje i akciju u sljedeće stanje i nagradu. Model može biti stohastičan, dakle rezultat preslikavanja su distribucije. Stohastičan model također može samo vratiti uzorak distribucije. Model *simulira* okolinu što nam omogućava prikupljanje *simuliranog iskustva* i omogućava *planiranje*. *Planiranje* bi bilo korištenje modela za poboljšanje politike.

Najpoznatiji algoritam ovog tipa, na neki način generalizacija *AlphaZeroa*, je *MuZero* [27]. *AlphaZero* gradi stablo kroz interakcije s okolinom, dok *MuZero* korak simulacije temelji u potpunosti na vlastitom modelu okoline. Naravno, model je potrebno naučiti.

MuZero možemo podijeliti na tri neuronske mreže koje surađuju s MCTS-om:

- 1. *Representation*: Neuronska mreža koja prima opažanje okoline i vraća (niskodimenzionalnu) reprezentaciju. Primijetimo sličnost s korištenjem dubokog učenja u redukciji dimenzionalnosti.
- 2. *Prediction*: Za dobivenu reprezentaciju, ova neuronska mreža je zapravo ista kao i u *AlphaZero* za usmjeravanje MCTS-a. Ona računa distribuciju vjerojatnosti mogućih akcija iz trenutnog stanja. Osim toga, vraća i funkciju vrijednosti stanja iz dane reprezentacije stanja.
- 3. *Dynamic*: Za reprezentaciju stanja i odabranu akciju, predviđa sljedeće stanje i pripadnu nagradu.

Kakav god problem *MuZero* rješava, algoritam sebi pronađe nižedimenzionalni prostor koji predstavlja prostor stanja i možemo zamisliti da na njemu igra igru. Igra je hodanje po tom prostoru. Ako zamislimo da su akcije koraci u nekim smjerovima, zapravo želi naučiti koji su to uopće smjerovi, kakav je to prostor i najbitnije od svega, kuda hodati kako bi dobio najveće nagrade.

Napomena 1.6.1. *Za treniranje mreža koristi se nagrada Markovljevog procesa odlučivanja. Moglo bi se pomisliti kako želimo da* Dynamic *vraća iste reprezentacije koje bi* Representation *vratio nakon interakcije s okolinom. Zapravo, nije nužno loše što može doći do prenaučenosti u smislu da* Dynamic *daje neku svoju sliku što su rezultati, a ne što mi, koji poznajemo problem, mislimo da su rezultati, jer upravo to vodi boljem rješenju problema učenja s potporom. Naravno, za očekivati je da* Representation *i* Dynamic *daju smisleno povezane odgovore. I to će vjerojatno biti slučaj, pogotovo kada se u pozadini problema koji* MuZero *rješava konceptualno nalazi nešto što duboko učenje stvarno može naučiti i riješiti.*

Ukoliko su interakcije s okolinom procesorski skupe, *MuZero* pruža alternativu u smislu korištenja hardverskih akceleratora za simulacije. Općenito, duboko učenje dobar je međukorak za korištenje algoritama pretraživanja u učenju s potporom na najmodernijim računalnim arhitekturama.[5]

MuZero

Recimo nešto točnije o funkciji gubitka koja se koristi u treniranju umjetnih neuronskih mreža.

Ranije smo spomenuli *n-step* i *bootstrapping* kao sastavne dijelove ovog algoritma. Metoda *N-step bootstrapping with priority* dio je *MuZero* algoritma koji funkcionira na sljedeći način:

- Procjenu funkcije vrijednosti stanja dobivamo promatranjem posljedica više akcija unaprijed.
- *Prioritetno uzorkovanje* (*engl.* priority sampling) pridaje veće težine za uzorkovanje stanju za koje ima višu grešku. Greška je u ovom slučaju razliku funkcije vrijednosti stanja i stvarnog povrata, jer tako može više naučiti.

Dobili smo (na temelju MCTS vođenog neuronskom mrežom) akciju za trenutno stanje i skupa s njom procjenu nagrade, vjerojatnost i procjenu funkcije vrijednosti stanja koje je neuronska mreža izračunala. Pogledamo zatim stvarnu nagradu nakon što poduzmemo odabranu akciju. (Postoje još i hiperparametri poput diskontirajućeg faktora γ i hiperparametra α koji je eksponent razlike stvarnog i procijenjenog povrata te se koristi u funkciji gubitka.)

Jednom kada pribavimo cijelu trajektoriju, možemo ju koristiti za gradijentni spust. Iz početnog stanja simuliramo trajektoriju. Sada možemo usporediti vrijednosti koje su nam neuronske mreže davale za trajektoriju koja se zapravo odvila, i onu koja bi nastala da smo se vodili isključivo simulacijom. Vidimo kako ovaj pristup na neki način približava naše neuronske mreže rezultatu MCTS-a. Budući da MCTS sam ovisi o mrežama, a sve ovisi o nagradi, cijeli algoritam uči postići bolje nagrade.

Ukupan gubitak se tako sastoji od gubitka nagrade, gubitka vrijednosti stanja i gubitka politike. Koristimo ga u gradijentnom spustu za sve neuronske mreže. Konkretno, za svaku od tri komponente računa se *softmax cross-entropy*. Ranije smo definirali softmax koji nam daje distribucije, a cross-entropy znači da gledamo razliku dviju distribucija sljedećom formulom

$$H(x, y) = -\sum_i x_i log(y_i) \tag{1.38}$$

[5] [30]

AlphaZero i MuZero ne moraju nužno postići poboljšanje politike. Za to je potreban posjet svoj djeci iz korijena. Primjer metode koja pokušava smanjiti broj potrebnih simulacija za poboljšanje je Gumbel AlphaZero [9].

Tijek rada algoritma MuZero

- Inicijalizirani algoritam prima opažanje i vraća (koristeći MCTS vođen neuronskim mrežama) *akciju*, *vjerojatnost* i *vrijednost*. Spremamo sve u trenutnu trajektoriju. Okolina daje sljedeće opažanje koje odgovara toj akciji.
- Kada dođemo do završnog stanja, imamo cijelu trajektoriju.
- Nakon što smo pribavili neki broj trajektorija, uzimamo k uzastopnih koraka (s nasumičnim početkom) iz svake trajektorije iz uzorka trajektorija.
- Počevši od prvog stanja, koristimo naučeni model za simulaciju trajektorije. Greška je razlika između simulirane i stvarne trajektorije (po ranije nabrojanim elementima).

Primjer Cart Pole

Ovdje ćemo ukratko prikazati jednostavno razumljiv primjer uporabe pretraživanja stabla Monte Carlo metodom. Odlični primjeri mogu se pronaći u [21] i [11].

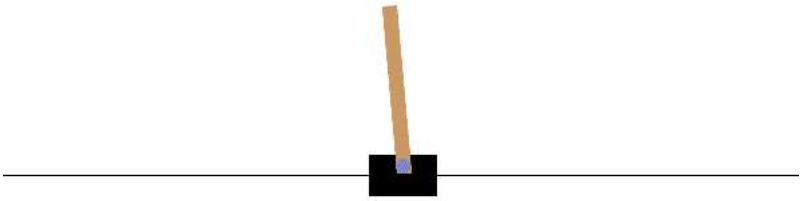
Najilustrativniji primjeri učenja s potporom su često primjeri agenata koji igraju igru. Specifično, koncentrirati ćemo se na igru za jednog igrača, jer i naš optimizacijski postupak tako funkcionira.

Ovaj problem kontrole opisan je u [4] i poznata je implementacija u [32]. Igra se sastoji od pomicanja kolica na kojima uspravno stoji štap. Cilj je pomicati kolica tako da štap ne padne niti na jednu stranu. Ne možemo se beskonačno kretati niti na jednu stranu. Radi jednostavnosti, pretpostavimo da imamo potpun model igre. Jedan trenutak igre prikazan je na slici 1.2.

Dvije su moguće akcije, primijeni silu na kolica ulijevo ili udesno. Nagrada iznosi 1 za svaku sekundu igre. Opažanje je četverodimenzionalni vektor

- Pozicija kolica.
- Brzina kolica.
- Nagib štapa.
- Kutna brzina štapa.

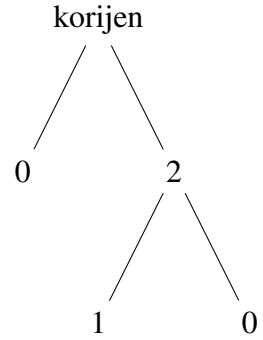
Iznesimo jedan primjer procedure odabira akcije za neko stanje.



Slika 1.2: Slika igre Cart Pole

- U korijenu smo koji predstavlja trenutno opažanje. Dodamo mu oba moguća djeteta (sila ulijevo ili udesno).
- Nasumično odaberemo desno dijete.
- Generiramo simulaciju igre do završnog stanja i pogledamo nagradu. Shodno tome, ažuriramo vrijednosti *funkcije vrijednosti akcije* po generiranoj trajektoriji (zasad imamo samo korijen i odabrano dijete).
- Sada smo ponovno u korijenu i biramo jedno dijete *politikom stabla*. Sada tom djetetu generiramo djecu i odabiremo lijevo. Ako je korijen trenutak t, sada smo na razini koja je trenutak $t + 2$. Budući da smo opet u listu, generiramo simulaciju te ažuriramo kao u prethodnoj točki.
- Sada će *politika stabla* imati više odabira i morati balansirati istraživanje i iskorištavanje prilikom sljedećih ekspanzija.

Kada ostanemo bez vremena, odaberemo bolje od originalnih dvoje djece i sljedeći potez biramo tako što su oni korijen u proceduri, s time da nešto stabla imamo već izgrađeno i možemo ponovno iskoristiti. Stablo nakon opisane procedure je na slici 1.3.



Slika 1.3: Stablo opisane procedure s brojem posjeta čvorova

Ponovimo da, ako bismo ikada došli do čvora koji smo stvorili ekspanzijom, a završno je stanje, ponašamo se isto kao kada je završno stanje rezultat simulacije i propagiramo nagradu unatrag.

Za danu igru, jasno je da će se, kada dođe do trenutka gdje se mora konačno odabrati akcija, odabrati ona akcija iz koje su prosječno simulacije najdulje trajale (jer su ti listovi u sebi imali najveću nagradu).

Radi jednostavnosti, pretpostavili smo da imamo model igre. Čar *MuZeroa* je što bi on naučio model i da ga nismo imali. Znao bi (imao bi aproksimator u obliku neuronske mreže) koje opažanje slijedi iz kombinacije opažanja i akcije. Brzo bi znao da je nagrada u svakom sljedećem djetetu 1. Simulirane trajektorije bile bi sve bolje i bolje zbog unaprijeđenja *zadane politike*.

Poglavlje 2

Učenje s potporom za adaptaciju po dijelovima polinomijalnih funkcija

Po dijelovima polinomijalna funkcija na domeni Ω definira se u ovisnosti o particiji (teselaciji) $\Omega = \cup_i \Omega_i$. Funkcija p je po dijelovima polinomijalna za teselaciju Ω_i ukoliko je restrikcija funkcije p na Ω_i, u oznaci $p\big|_{\Omega_i}$ polinom. Problem prilagodbe po dijelovima polinomijalne funkcije nekom kriteriju *kvalitete* svodi se na problem adaptivne prilagodbe teselacije Ω_i tako da se mjera kvalitete funkcije poveća. Istaknimo da ovdje možemo nametati i neka daljnja globalna svojstva na po dijelovima polinomijalnu funkciju, kao što su neprekidnost na cijelom Ω. Također, promatrat ćemo i po dijelovima polinomijalne funkcije koje ne moraju biti globalno neprekidne.

Kao oglednu primjenu principa učenja s potporom promatrat ćemo problem prilagodbe po dijelovima polinomijalne funkcije u svrhu smanjenja mjere greške aproksimacije funkcije implicitno zadane kao rješenje neke (parcijalne) diferencijalne jednadžbe.

Problem koji ćemo prikazati je (diskretna) optimizacija mreže konačnih elemenata. Problem konkretno nije primjer optimalnog oblikovanja struktura niti optimizacije topologije, no dijeli puno sličnosti i prikazane metode su primjenjive, do na implementacijske detalje.

Nakon što prikažemo osnovnu metodu adaptacije teselacije koja definira po dijelovima polinomijalnu funkciju na segmentu po članku [12], poopćit ćemo metodu na geometrijski zahtjevniju situaciju jednodimenzionalne diskretne mnogostrukosti opisane strukturom metričkog grafa [2]. Konkretno, promatrat ćemo problem aproksimacije rješenja eliptičke jednadžbe na metričkom grafu po dijelovima polinomijalnim funkcijama. Posebno će nas zanimati međuigra dijela algoritma koji se bavi razlučivanjem dijela greške koji dolazi od problema geometrije domene i onih koji dolaze zbog regularnosti funkcije rješenja koju se aproksimira.

2.1 Metoda konačnih elemenata

U uvodu poglavlja istaknuli smo da su funkcije kojima prilagođavamo teselaciju implicitno zadane kao rješenja diferencijalnih jednadžbi. Aproksimante ovih objekata konstruirat ćemo (inženjerskom) metodom konačnih elemenata. Metodu ćemo uvesti formalno algebarski, a detaljnija analiza konvergencije daleko nadilazi opseg ovog rada.

Metoda konačnih elemenata numerička je metoda za rješavanje parcijalnih diferencijalnih jednadžbi. Jedan od koraka je diskretizacija prostorne domene zadaće. U osnovi u inženjerskom pristupu uvođenja metode, *metoda konačnih elemenata* (*engl.* finite element method, FEM) svodi se na razbijanje domene na manje jednostavne podskupove, elemente, na kojima se zatim rješava lokalizirani problem. Bitno je da se unutrašnjosti elemenata ne sijeku i da je unija zatvarača svih elemenata zatvarač domene (vidi [15]).

Što su elementi manji, to je rješenje preciznije. No to ujedno znači da ih je više i da je rješavanje procesorski skuplje. Adaptacija mreže konačnih elemenata znači balansiranje između komponenti greške koje smanjujemo profinjavanjem subdivizije i računske složenosti koja raste povećanjem broja elemenata.

Kada imamo aproksimacije na elementima diskretizacije, prijelazi između elemenata nisu glatki, a u razlomljenim (diskontinuiranim) metodama čak se radi o različitim vrijednostima. Zbog navedenog prirodni prostor funkcija u kojima se rješenja promatraju je prostor kvadratno integrabilnih funkcija budući da je vrijednost integralne norme nešto što je dobro definirano za razliku od vrijednosti funkcije u nekoj danoj točki.

Međutim, ako znamo da je naše rješenje glatka funkcija onda je upravo mjera *neglatkoće* funkcije mjera koja otkriva koliko loše trenutni konstrukt aproksimira zadani objekt. Recimo da imamo elemente $E^{(l)}$ i $E^{(r)}$. $F = E^{(l)} \cap E^{(r)}$ je rub između njih. Ako su $v^{(l)}$ i $v^{(r)}$ pripadna rješenja na rubu, definiramo *skok* (kao u [17])

$$[v]_F = v^{(l)} - v^{(r)}. \tag{2.1}$$

U jednodimenzionalnom slučaju, koji ćemo mi obrađivati, rub je točka, a vrijednosti skalari. U višedimenzionalnim situacijama, radilo bi se o integralu.

Slično, za metode koje konstruiraju globalno neprekidne polinome (kontinuirane metode), gledat ćemo ono što bi intuitivno bio *lom* u rješenju, koji zapravo predstavlja nedostatak glatkoće koju takva metoda želi postići. Tada će *skok* u rješenju biti promjena gradijenta na rubovima elemenata diskretizacije.

Istaknimo da je glavna intuicija vezana uz konstrukciju po dijelovima polinomijalnih funkcija, uz ograničenje (zadanog) stupnja polinoma po elementu, da će se više elemenata pojavljivati tamo gdje je zakrivljenost (druga derivacija) velika. To slijedi jer se skok gradijenta (derivacije) se (uz pretpostavku glatkoće) mjeri drugom derivacijom. Općenito, kada govorimo o potpuno adaptivnim metodama konačnih elemenata, onda teselaciju profinjujemo tamo gdje je rješenje izgubilo glatkoću, a polinomijalni stupanj dižemo tamo gdje je rješenje glatko.

Naravno, u slučaju kada budemo prilagođavali teselaciju problemu definiranom na (diskretnoj) mnogostrukosti, gledat ćemo međuodnos zakrivljenosti ili skokovitosti funkcije i zakrivljenosti ili skokovitosti mnogostrukosti (grafa).

2.2 Formuliranje problema

Cilj je konstruirati finiju mrežu na mjestima gdje aproksimacija rješenja jako odstupa od rješenja problema kojega aproksimiramo u svrhu smanjenja greške aproksimacije, a konstruirati grublju na mjestima gdje regularnost rješenja (glatkoća) to dopušta u svrhu smanjenja (kontroliranja) računske složenosti. Odluka o profinjenju ili uklanjanju elementa (npr. spajanjem dvaju elemenata u jedan veći) mora se donositi lokalno, element po element.

U našem slučaju, profinjenje elementa znači prepolavljanje na dva elementa. Pogrubljenje znači spajanje dvaju bridova u jedan. Element se može pogrubiti samo tako što se spoji s elementom koji je nastao skupa s njim i to ako su oba jednako profinjeni.

Primjer 2.2.1. *U terminima politike agenta to formuliramo na sljedeći način: agent treba odlučiti za točno određeni element mreže želi li ga profiniti ili pogrubiti. Opažanje ovisi o trenutnom elementu. Sličnost s oblikovanjem struktura je u tome da agent može izbrisati ili ostaviti dio strukture. Za kontekst primjera pogledajte članak [5].*

Djelomično uočljiv Markovljev proces odlučivanja

Dosad nismo radili razliku između opažanja i stanja. Ovdje ćemo formulirati *djelomično uočljiv Markovljev proces odlučivanja* (*engl.* partially observable Markov decision process, POMDP). Gdje smo prije imali $\pi(a|s)$, sada je s opažanje, a ne stanje. To je zato što donosimo lokalne odluke iz lokalnih (i nešto odabranih globalnih) informacija.

Prostor akcija i prostor stanja

U ovakvoj formulaciji, zapravo imamo 3 moguće akcije u svakom trenutku. Profiniti, pogrubiti i ne učiniti ništa. Agent će stohastički skakati po mreži i tako dobivati nova opažanja. Kako bi uopće imao šanse donijeti dugoročno dobru odluku, mora imati i neku globalnu informaciju.

Globalno, stanja su zapravo sve moguće diskretizacije, s time da smo efektivno ograničeni na one koje možemo postići unutar naših računalnih resursa. Jasno je da je takav prostor stanja eksponencijalno velik i besmisleno bi bilo pokušati ga istraživati. Upravo zato ćemo za zadatak definirati POMDP.

Također, moramo nekako odabrati kada i kako ćemo doći u završno stanje. Možemo jednostavno postaviti neku gornju granicu na broj akcija.

Opažanje

Spomenuli smo ranije da će agent stohastički skakati po elementima i na njima donositi odluke. Stoga, definiramo opažanje ovisno o elementu. Ovo je vektor koji ulazi u neuronsku mrežu.

- Prvo imamo, prema (2.1), sumu svih skokova za element u kojem se nalazimo.
- Slično, za svaki susjedni element također gledamo sumu svih skokova i suma tih suma čini drugu značajku.
- Kako bismo imali ikakvu globalnu informaciju, agent će znati prosjek suma skokova po svim elementima.
- Udio resursa koji trenutno trošimo. Jednostavno $\frac{trenutni}{maksimalni}$. Imat ćemo konstantu koja predstavlja resurse koji su na raspolaganju, što je gornja granica broja elemenata diskretizacije.

Napomenimo da, u slučaju na segmentu, krajnji elementi nemaju oba susjeda.

Iz prostora stanja koji smo konstruirali, jasno je da želimo profinjenja u slučaju kada je skok rješenja (ili gradijenta) na rubu elementa natprosječan (uzmimo u obzir i susjedne skokove), s time da je šansa za profinjenje veća što je u tom trenutku manji broj elemenata.

Nagrada

Neka je K segment, a u numeričko rješenje. Tada razliku novog u^{t+1} i starog u^t rješenja računamo po elementima finije mreže $\mathcal{T}$:

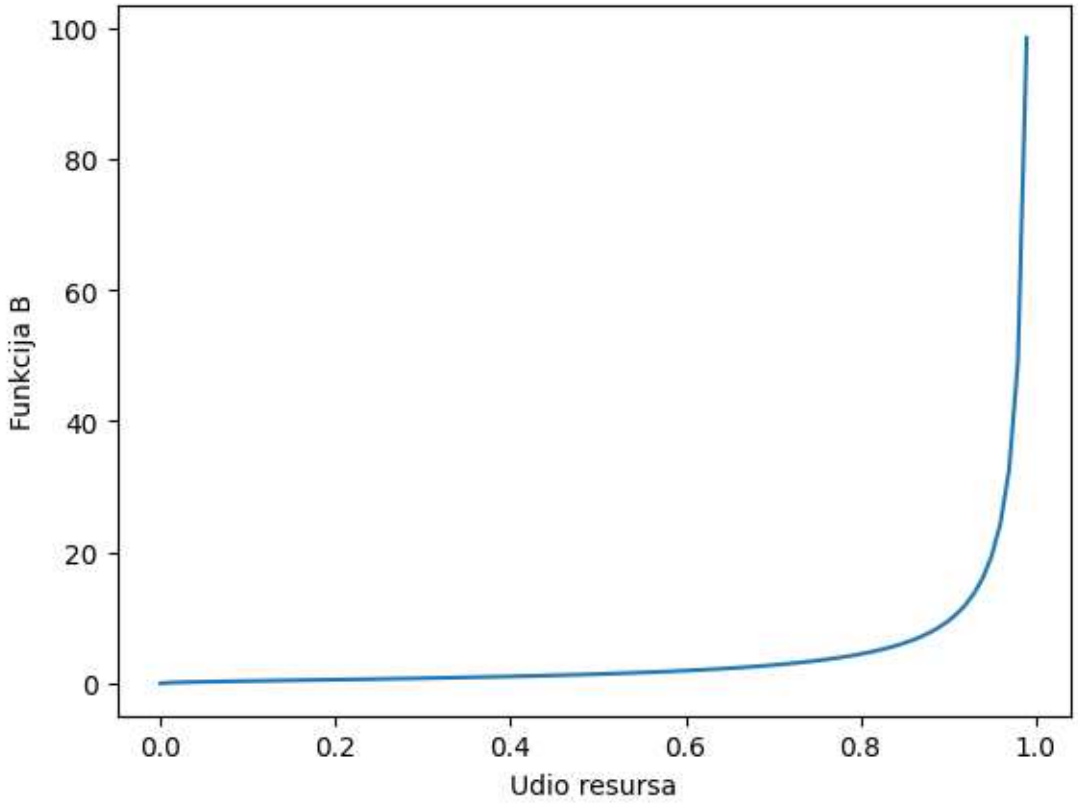
$$J(u) = \sum_{K \in \mathcal{T}} \int_K |u^{t+1} - u^t| dK. \tag{2.2}$$

Profinjenje znači da gornji rezultat uzimamo u nagradu s pozitivnim predznakom, a za pogrubljenje s negativnim.

Osim opisane sume integrala, definirajmo funkciju preko koje broj elemenata utječe na nagrade:

$$B(p) = \sqrt{(p)}/(1 - p). \tag{2.3}$$

Koeficijent p ranije je spomenut omjer trenutnog broja elemenata i neke predodređene gornje ograde. Ta ograda predstavlja količinu resursa koji su nam na raspolaganju. Vidimo na 2.1 da je odabrana tako da algoritam bude prisiljen koristiti što više dopuštenih resursa. Isto tako, vidimo da bi u slučaju korištenja više resursa no što je na raspolaganju, funkcija divergirala. To preduhitrimo i kažnjavamo algoritam velikom negativnom nagradom.



Slika 2.1: Funkcija B

γ je hiperparametar koji određuje odnos važnosti preciznosti rješenja i uloženih resursa. Tako konačno dobivamo nagradu (vidi [12]) za trenutno stanje s_t i odabranu akciju a_t

$$R(s_t, a_t) = \begin{cases} +\log J(u) - \gamma(B(p_{t+1}) - B(p_t)) & a_t \text{ je profinjenje} \\ -\log J(u) - \gamma(B(p_{t+1}) - B(p_t)) & a_t \text{ je pogrubljenje} \\ 0 & a_t \text{ ne radi ništa.} \end{cases} \tag{2.4}$$

Logaritam je tu kako bi na neki način normalizirao nagrade, što će neuronskoj mreži olakšati generalizaciju. Jasno je da ovako definirana nagrada usmjerava algoritam prema rješenju koje je bliže egzaktnom. Jedan luksuz koji koristimo je činjenica da profinjenja garantiraju bolja rješenja, te stoga možemo minimizirati grešku krećući se prema nepoznatoj vrijednosti jer uvijek znamo u kojem se smjeru krećemo.

Napomena 2.2.2. *Nekada, u FEM formulacijama, pogrubljenje mreže nije moguće. Probabilistička interpretacija toga je da postoji vjerojatnost uspjeha akcije. Općenito, problem promjenjivosti skupa dozvoljenih akcija je problematičan u području dubokog učenja s potporom. Primjer pristupa tome bio bi da agent nauči koje akcije su ilegalne time što se kroz nagradu kažnjava poduzimanje ilegalne akcije.*

MuZero, primjerice, može maskirati vjerojatnosti politike koje odgovaraju ilegalnim akcijama na jako male vrijednosti. No tijekom simuliranja, nema mu tko reći što su ilegalni potezi.

Mi ćemo se ponašati kao da je odlučio ništa ne učiniti kada poduzme ilegalnu akciju.

Problem ovakve formulacije problema za oblikovanje struktura je upravo u tome što agent ne može predvidjeti kako će akcija utjecati na neko globalno svojstvo.

Primjer 2.2.3. *Ako imamo graf i za njega binarno globalno svojstvo:* povezanost.

Za POMDP, agent ne može znati hoće li akcijom uništiti to svojstvo. Ukoliko bi to bila negativna nagrada u završnom stanju, agent nema šanse naučiti zašto. *Također, što ako naknadno izbriše jedan od dva nastala nepovezana podgrafa daljnjim akcijama i time graf ponovno postane povezan. Stoga, POMDP nije uvijek prigodan pristup za neke probleme diskretne optimizacije.*

2.3 Metrički grafovi

Osim na segmentu, promatramo i problem u kojem je struktura susjedstva među elementima složenija. Element može imati i više od dva susjeda. U tu svrhu promatramo diferencijalne jednadžbe na diskretnim mnogostrukostima (metričkim grafovima) koje nam omogućavaju kontrolirano povećanje složenosti promatranog fenomena (profinjavanje ili uklanjanje elementa teselacije na osnovu informacije iz direktnog susjedstva).

Istaknimo da diferencijalni problemi na mrežama (grafovima) mogu poslužiti za modeliranje raznih problema u kojima je pojam povezanosti ključan element fenomenologije (vidi [2]). Na metričkim grafovima na bridovima imamo definiranu metriku i tako grafovi čine jednodimenzionalnu topološku mnogostrukost.

Definicija 2.3.1. *Povezani graf* $\Gamma = (\mathcal{V}, \mathcal{E})$ *je* metrički graf *ako*

- *svaki brid e ima pozitivnu konačnu duljinu* l_e
- *točke brida imaju koordinatu* $x_e \in [0, l_e]$ *i vrijednost je monotona po bridu.*

Svaki brid grafa Γ *na ovaj način možemo identificirati s intervalom* $[0, l_e]$ *čime uvodimo prirodnu parametrizaciju brida* x_e. *Neka je* $\mathcal{V} \subset \mathbb{R}^d$. *U slučaju kada je* $e = (v_i, v_j) \in \mathcal{E}$, *tada je koordinati* x_e *pridružena točka* $v_e = x_e\, v_i + (1 - x_e)\, v_j$ *na bridu e. Alternativno, metrički graf možemo promatrati kao jednodimenzionalni simplicijalni kompleks.*

Da bismo mogli promatrati diferencijalne izraze na strukturi kao što je metrički graf, moramo prvo uvesti pogodan prostor funkcija koji je opisan grafom kao jednodimenzionalnom mnogostrukosti. U nastavku slijedimo pristup iz [2]. Istaknimo da ovdje nećemo posebno definirati Soboljevljeve prostore funkcija na segmentu. Prisjetit ćemo se da je funkcija definirana na segmentu čija je slaba prva derivacija integrabilna nužno neprekinuta te da je funkcija čija je slaba druga derivacije neprekinuta nužno klase C^1. Uz ove uvjete ćemo promatrati produktne prostore kvadratno integrabilnih funkcija koji će nam omogućiti da damo smisao diferencijalnim izrazima na grafu promatranom kao jednodimenzionalna mnogostrukost.

Definicija 2.3.2. *Prostor* $L^2(\Gamma) = \bigoplus_e L^2(e)$ *definiran grafom* $\Gamma(\mathcal{V}, \mathcal{E})$ *je skup svih kvadratno integrabilnih, izmjerivih funkcija u na bridovima grafa* Γ *za koje vrijedi*

$$\|u\|^2_{L^2(\Gamma)} := \sum_{e\in\mathcal{E}} \|u\big|_e\|^2_{L^2(e)} < \infty \, .$$

Uz ovu normu definiramo i skalarni produkt

$$\langle u, v\rangle_{L^2(\Gamma)} = \sum_{e\in\mathcal{E}} \int_e u \, v \, dx_e.$$

Među kvadratno integrabilnim funkcijama promatramo one funkcije čije su derivacija integrabilne i koje još dodatno zadovoljavaju uvjet neprekidnosti. Konkretno, funkcija $u \in L^2(\Gamma)$ nalazi se u prostoru $H^1(\Gamma)$ ako je neprekidna na Γ, u oznaci $u \in C(\Gamma)$, i ako vrijedi

$$\|u\|^2_{H^1(\Gamma)} = \sum_{e\in\mathcal{E}} \|u\big|_e\|^2_{L^2(e)} + \|(u\big|_e)'\|_{L^2(e)^2} < \infty.$$

Konačno, promatramo funkcije $u \in H^1(\Gamma)$ za koje vrijedi da je $u\big|_e \in H^2(e)$, $e \in \mathcal{E}$ i za koje dodatno u vrhovima grafa vrijede Neumann-Kirchhoffovi uvjeti

$$\sum_{e\in\mathcal{E}_v} \frac{du}{dx_e}(v) = 0 \quad \forall v \in \mathcal{V} \tag{2.5}$$

gdje su $\mathcal{E}_v$ bridovi čvora v i derivacije su po izlaznim bridovima.

Sada izraz $\mathcal{H}$ može definirati operator na metričkom grafu Γ (preuzeto iz [2]):

$$\mathcal{H}u(x) = -\frac{d^2u}{dx^2}(x) + v(x)u(x) \tag{2.6}$$

gdje je $v(x)$ potencijal, a funkcija $u \in H^1(\Gamma) \cap (\bigtimes_e H^2(e))$. Ovaj operator može se strogo definirati korištenjem varijacijskog pristupa ([18, Prvi teorem reprezentacije]) kao operator iz $L^2(\Gamma)$ u $L^2(\Gamma)$ koji reprezentira bilinearnu formu

$$\sum_{e\in\mathcal{E}} \int_e (u\big|_e)' \, (\phi\big|_e)' + (v \, u \, \psi) dx_e = \langle \mathcal{H}u, \phi\rangle_{L^2(\Gamma)}$$

za svaki $u \in H^1(\Gamma) \cap (\bigtimes_e H^2(e))$ i $v \in H^1(\Gamma)$. Istaknimo da se direktna ortogonalna suma može izometrički poistovjetiti s produktnim prostorom $\bigtimes_e H^2(e)$. Ovu identifikaciju ćemo koristiti bez da notacijski razlikujemo objekte. Za primjere vidi [14, Section: An elementary introduction to quantum graphs]. Ova bilinearna forma definira energiju ovog kompleksnog sustava. Daljnji aspekti analize ovih objekata su izvan opsega ovog rada. Zainteresirani čitatelj je pozvan proučiti članak [2] i tamo navedenu iscrpnu listu referenci.

Primijetimo da su po dijelovima polinomijalne neprekidne funkcije koje u vrhovima zadovoljavaju Neumann-Kirchhoffove uvjete u prostoru $H^1(\Gamma)$. U ovom slučaju, koristit ćemo kontinuirane metode (tj. metode koje pretpostavljaju da je po djelovima polinomijalna funkcija globalno neprekinuta) i stoga će skok biti predstavljen, kao što smo ranije spomenuli, razlikom gradijenata.

2.4 Grafovske neuronske mreže

Naš je cilj oblikovati metodu adaptacije mreže koja radi element po element. Agent dobiva na uvid dani elementa i neku njegovu okolinu. Iz ovog opisa vidimo da će odnos susjedstva između elemenata imati ključnu ulogu u hodu algoritma. U prvom primjeru ćemo promatrati teselaciju intervala gdje elemeti imaju samo kontakta sa susjednim elementima. Nakon toga promatrat ćemo problem na metričkom grafu (diskretnoj mnogostrukosti) gdje je odnos susjedstva među elementima kompleksniji. U nastavku opisujemo koncept grafovskih konvolucijskih neuronskih mreža koje bi mogle biti dovoljno *ekspresivne* da iskoriste ovu strukturu susjedstva.

Grafovske neuronske mreže vrsta su neuronskih mreža pogodne za obradu grafova, a pripadaju području dubokog učenja koje se zove *geometrijsko duboko učenje*. Prikladne su za obradu permutacijski invarijantnih podataka. Primjer problema gdje daju odlične rezultate je prepoznavanje radnji ljudi.

Mogu se promatrati kao generalizacija konvolucijskih neuronskih mreža. Matrica bi bila graf kojem su čvorovi elementi matrice, a bridovi su između onih elemenata koje istovremeno može pokriti konvolucijska jezgra. Težine bridova bili bi upravo elementi konvolucijske jezgre.

Osim konvolucijskih slojeva, mogu se promatrati slojevi *lokalnog sažimanja* (*engl.* local pooling) u svrhu smanjenja broja čvorova te *globalnog sažimanja* (*engl.* global pooling ili readout) poput sume, prosjeka ili maksimuma po svim čvorovima.

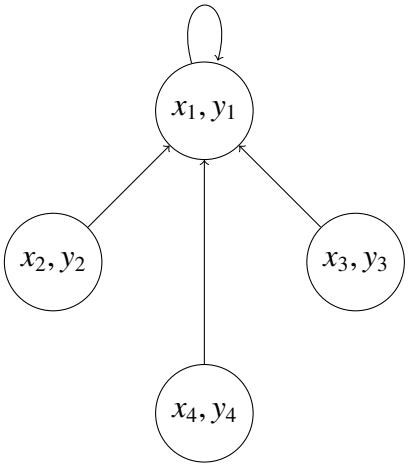
Grafovske konvolucijske mreže

Za ovu arhitekturu neuronskih mreža kaže se da *združuju susjedstva* (*engl.* neighborhood aggregation) ili *prosljeđuju poruke* (*engl.* message passing). Značajke čvorova promijene se ovisno o značajkama susjednih čvorova.

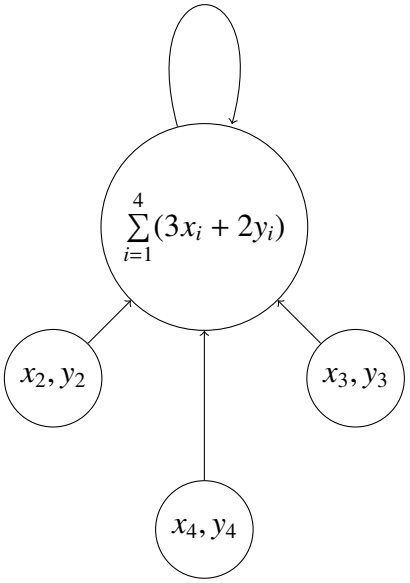
Neka je $\tilde{\mathbf{A}}$ matrica susjedstva s jedinicama na dijagonali. Neka je $\tilde{\mathbf{D}}$ pripadna dijagonalna matrica stupnjeva čvorova. Ako je $\mathbf{X}$ matrica značajki vrhova i $\mathbf{\Theta}$ matrica težina, uz aktivacijsku funkciju σ, grafovski konvolucijski sloj neuronske mreže (detaljnije u [26]) glasi:

$$\mathbf{H} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}). \tag{2.7}$$

Ukratko, dimenzija $\mathbf{\Theta}$ određuje broj izlaznih značajki po čvoru i utjecaj pojedine ulazne značajke na pojedinu izlaznu. Težine bridova određuju međuutjecaj čvorova.
Ideja je da, budući da su za rješenja parcijalnih diferencijalnih jednadžbi na elementima bitni i susjedni elementi, iskoristimo grafovske konvolucije kako bi to elementi međusobno iskomunicirali.



Primjer grafa sa značajkama, sve težine bridova su 1



Nakon primjene matrice težina $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$

Budući da je potrebno odrediti težine bridova među elementima mreže konačnih elemenata, postoji način da to neuronske mreže same računaju. *Grafovske mreže pažnje* (*engl.* graph attention network, GAT) koriste mehanizam *pažnje* (*engl.* attention) kako bi, slično kao arhitektura transformera, naučili sami računati težine prepoznajući kontekst. Više o tome u [33].

Poglavlje 3

Rezultati

3.1 Zadatak

Prvo ćemo promatrati problem konstrukcije po dijelovima polinomijalne aproksimacije funkcija definiranih na segmentu. Nakon toga rješavat ćemo parcijalne diferencijalne jednadžbe na metričkim grafovima u 2 dimenzije.

3.2 Actor-Critic algoritam za učenje s potporom

Ovdje ćemo prikazati rezultate realizacije algoritma za učenje s potporom temeljenom na djelomično uočljivom Markovljevom procesu odlučivanja (POMDP). Algoritam se sastoji od dvije komponete, generativne komponente *Actor* i diskriminatorne komponente *Critic*. Istaknimo da je upravo komponenta *Critic* ona u kojoj se novi pristup temeljen na neuronskim mrežama može zamijeniti standardnim procesom temeljenim na analizi kvalitete aproksimacije. Primjerice, mi možemo učiti heurističku mjeru koju realizira algoritam *Critic* ili možemo primijeniti neku od klasičnih a posteriori ocjena greške aproksimacije koja može biti pesimistična, ali je dokazivo pouzdana. Također, klasična ocjena greške može se koristiti kao pouzdan kriterij zaustavljanja procesa treniranja. Alternativno, algoritam možemo zaustaviti u onom trenutku u kojem statistički ustanovimo da je hod algoritma profinjenja došao do stacionarne točke, u nastavku procesa algoritam ne uvodi daljnje promjene u strukturu.

Implementacija

Korištene su *Python* biblioteke. Za metodu konačnih elemenata i rad s grafovima (rijetkim matricama) korištene su standardne biblioteke numeričke matematike i znanstvenog

računanja *numpy* i *scipy*. Za eksperiment vezan uz duboko učenje s potporom, uglavnom je potrebna implementacija okoline. Za to je najpopularnija specijalizirana biblioteka *Gymnasium* [32]. Osim toga, koristili smo implementaciju *A2C, Advantage Actor-Critic* iz biblioteke *Stable Baselines3* [16] temeljenoj na biblioteci za duboko učenje *PyTorch*. Osim toga, za algoritme vezane uz pretraživanje stabla Monte Carlo metodom, korišten je *JAX* [3] i ostale biblioteke ekosustava. Uz njih su korišteni *Haiku Geometric* [23] za geometrijsko duboko učenje te *muax* za pomoć kod *MuZero* algoritma. Istaknimo još da su *A2C* implementacije učenja s potporom pogodne su za CPU trening [22].

Počinjemo od nasumične mreže. Zaustavljamo se nakon predviđenog broja koraka ili nakon što puno puta nismo ništa napravili.

Profinjenje na segmentu

Pogledajmo jednostavnu transportnu jednadžbu

$$\begin{aligned} &\frac{\partial u}{\partial t} + 2\pi \frac{\partial u}{\partial x} = 0, \quad x \in [0, 2\pi], \\ &u(x, 0) = \sin(x), \\ &u(0, t) = -\sin(2\pi t) \end{aligned}$$

čije je egzaktno rješenje $u(x, t) = \sin(x - 2\pi t)$ na $[0, 2\pi]$.

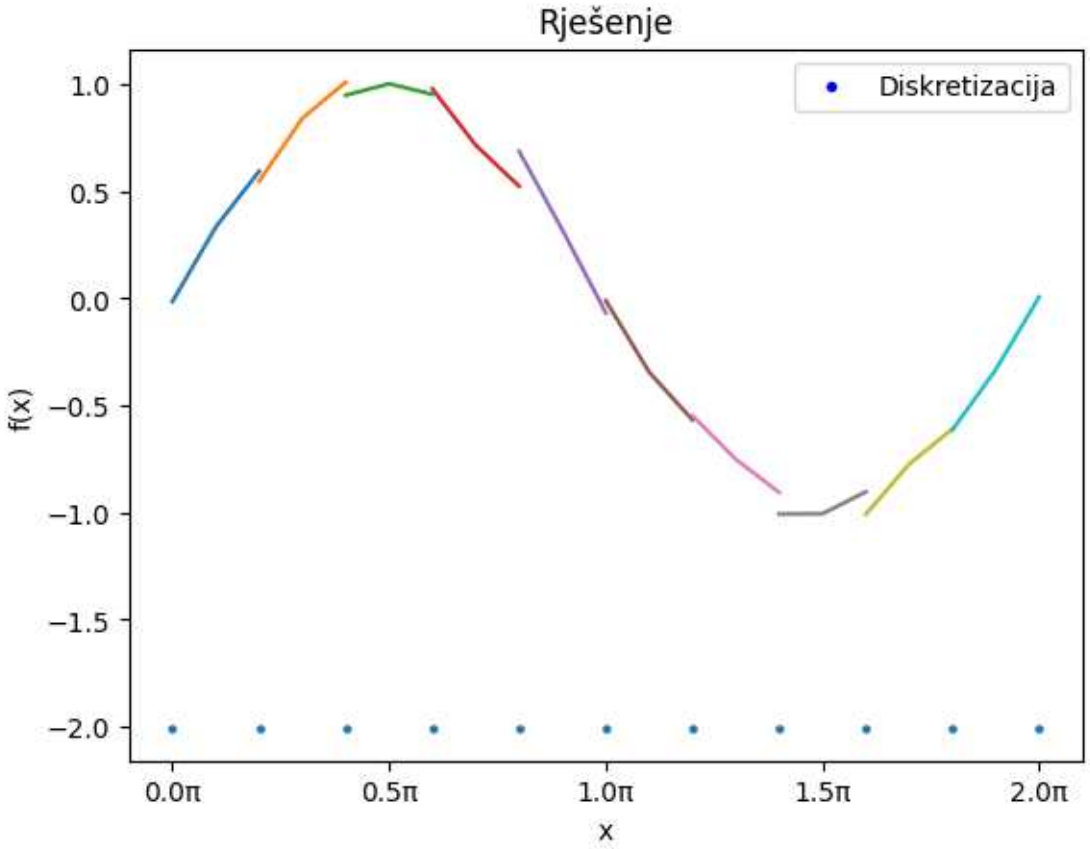
Korstimo budžet od 10 elemenata s 2 stupnja slobode. Usporedit ćemo rješenje 3.1 na uniformno particioniranoj mreži i rezultat 3.2 optimizacije algoritma učenja s potporom.

Vidimo točno kako na sredini imamo elemente koji skoro pa i nemaju prekide. S druge strane, gdje je apsolutna vrijednost druge derivacije funkcije sinus najveća, konkretno na slici 3.1 element označen sivom bojom, imamo veće odstupanje. Također, primijetimo da je algoritam korektno prepoznao dio domene gdje je funkcija jače zakrivljena i tamo stavio finiju mrežu. Istaknimo da je cilj povećati točnost bez povećavanja računske složenosti.

Algoritam *Actor-Critic* je u ovom primjeru počeo na uniformnoj mreži od 5 elemenata i radio do 50 koraka kada smo ga zaustavili. Automatsko zaustavljanje algoritma je posebna tema i zahtijevala bi konstrukciju procjenitelja greške aproksimacije. Naša funkcija nagrade je iskustveno (heuristički) oblikovana i rezultat hoda algoritma evaluiramo a posteriori. No ovdje svakako postoji mogućnost razvoja algoritma koji će heuristički konstruirati kvalitetnu mrežu, no zaustavljanje ovisi o klasičnom procjenitelju greške. Time možemo dobiti opći generativan, ali i pouzdan algoritam.

Na 3.2 točno vidimo da je diskretizacija gušća upravo tamo gdje smo očekivali, gdje su skokovi bili značajniji.

Znamo da smo formulirali funkciju cilja koja nas profinjenima približava stvarnom rješenju. A znamo i da je u ovom slučaju stvarno rješenje neprekidno (i glatko), što znači da nema skokova,kako funkcijske vrijednosti, tako i njene derivacije. Iz toga se



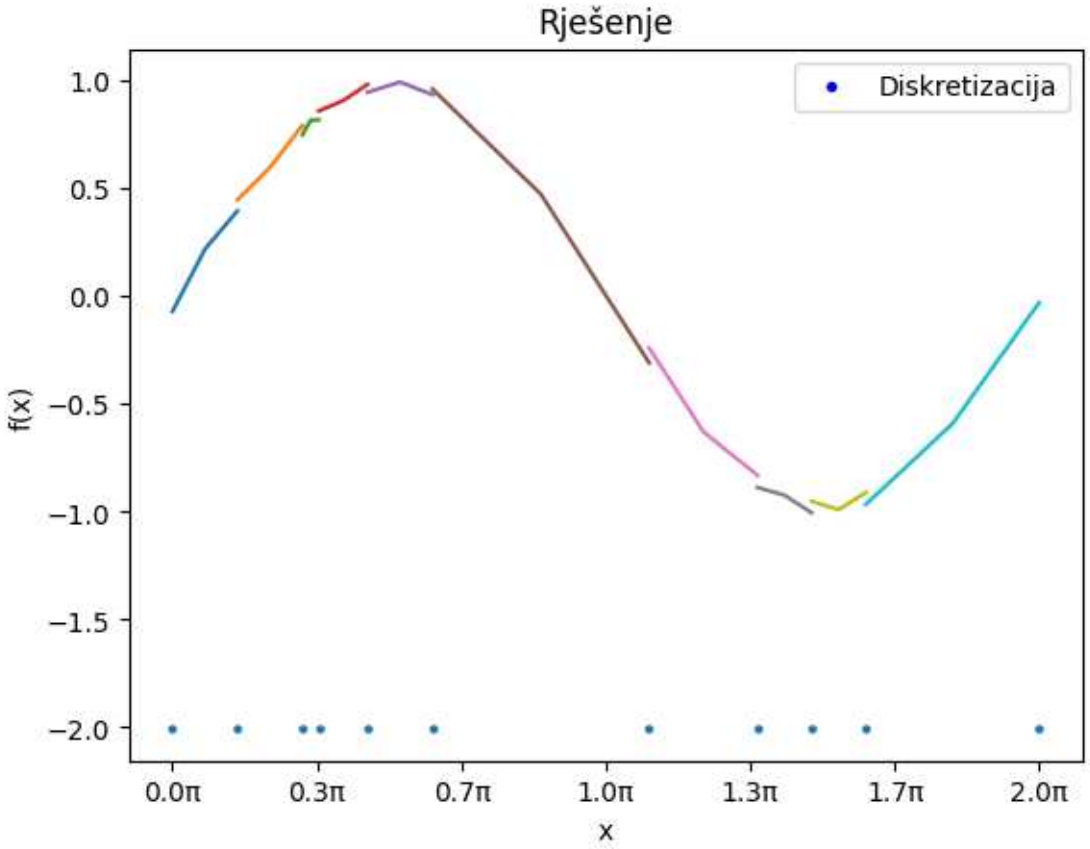
Slika 3.1: Rješenje na uniformnoj mreži

može zaključiti da su profinjenja elemenata smanjivala skokove. Tako je neuronska mreža mogla *shvatiti* korelaciju između profinjavanja elemenata sa skokovima i pozitivne nagrade. Slično, pogrubljenja elemenata bez skokova uglavnom beznačajno smanjuju kvalitetu rješenja, a donose pozitivnu nagradu zbog smanjenja udjela korištenih resursa.

Profinjenje na metričkom grafu

Neka je $\mathcal{V} = \{(-1,0),(0,0),(1,0),(0,-1),(0,1)\}$ i neka $\mathcal{E}$ sadrži sve parove elemenata s ishodištem koordinatnog sustava. Instanca problema koji promatramo je aproksimacija funkcije $u \in H^1(\Gamma) \cap (\bigtimes_e H^2(e))$ za koju vrijedi

$$\begin{cases} -\partial_{x_e x_e} u(x_e) + u(x_e) = (4\pi^2 + 1)\sin(2\pi x_e), & e \subset [-1,-1] \times 0 \\ -\partial_{x_e x_e} u(x_e) + u(x_e) = 0, & e \subset 0 \times [-1,-1] \ . \end{cases} \tag{3.1}$$



Slika 3.2: Nakon profinjenja učenja s potporom

i čije je egzaktno rješenje dano restrikcijom funkcije

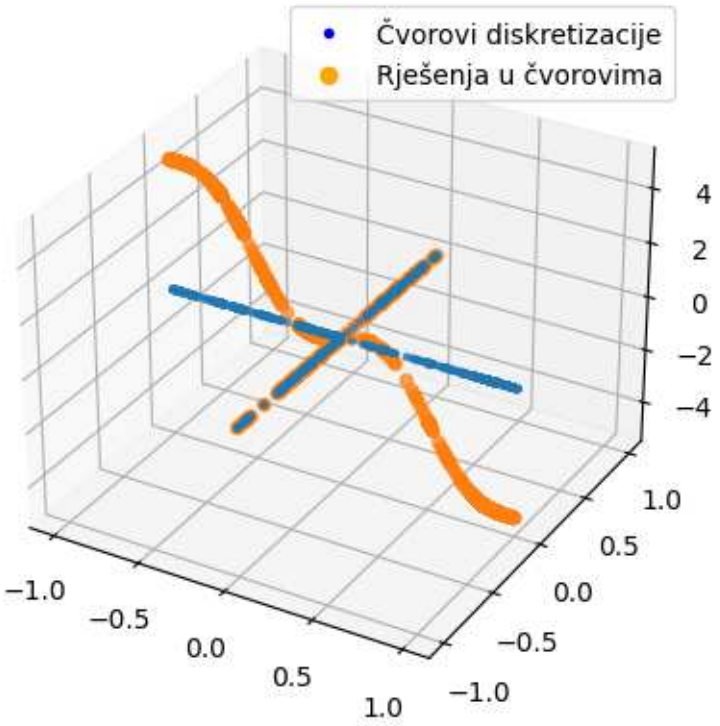
$$u(x, y) = \sin(2\pi x) \tag{3.2}$$

na Γ. Istaknimo da ovo je degenerirani primjer, budući da odnos (separabilnost) između osi nije fizikalna. No primjer će poslužiti dobro za ilustraciju ponašanja algoritma u pojedinim karakterističnim situacijama prijenosa informacija o potrebi profinjenja u okolini danog elementa (stanja algoritma).

U ovom slučaju razliku rješenja definiramo formulom

$$J(u) = \sum_{e \in \mathcal{E}_r} \int_e |u^{t+1} - u^t| dx_e, \tag{3.3}$$

gdje je $\mathcal{E}_r$ skup bridova kojega dobijemo tako da za svaki brid $(v_i, v_j) \in \mathcal{E}$ u skup $\mathcal{E}_r$ dodamo bridove $(v_i, v_{ij,1/2})$ i $(v_{ij,1/2}, v_j)$ za $v_{ij,1/2} = (1/2)v_i + (1/2)v_j$.



Slika 3.3: Rješenje (3.1)

Nadalje, prisjetimo se kako smo definirali proces opažanja ovisno o elementu i to koristili kao vektor koji ulazi u neuronsku mrežu. Njegova prilagodba za slučaj metričkog grafa i globalno neprekidnih polinoma glasi

- Umjesto skokova funkcije prema (2.1), promatramo ukupni skok prvih derivacija za svaki brid $t = (v_1, v_2) \in \mathcal{E}$ kojega definiramo kao

$$[u']_t := \sum_{e \in \mathcal{E}_{v_1}} \frac{du}{dx_e}(v) + \sum_{e \in \mathcal{E}_{v_2}} \frac{du}{dx_e}(v) \,. \tag{3.4}$$

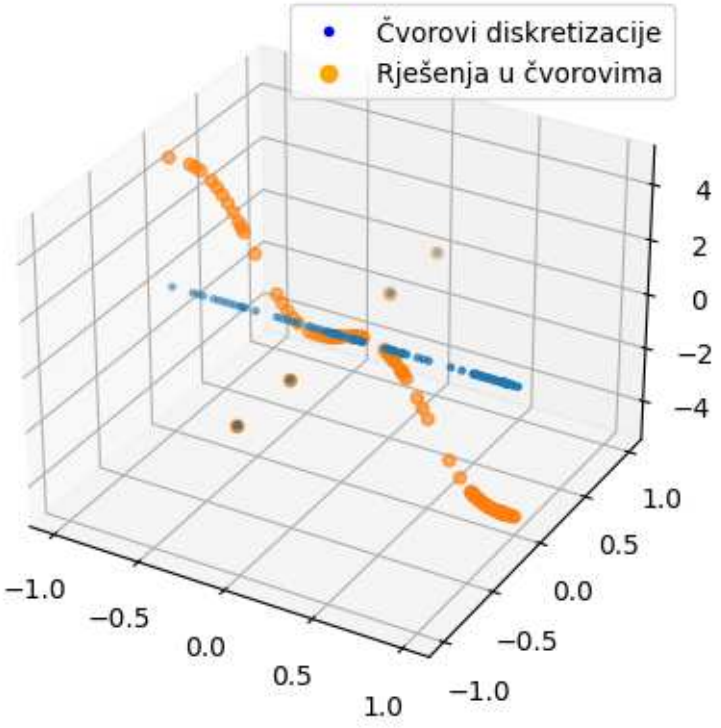
- Slično, za svaki susjedni element n brida t također gledamo njegov ukupni skok prvih derivacija i suma $[u]_n$ po svim susjedima n brida t čini drugu značajku.

- Kako bismo imali ikakvu globalnu informaciju, agent će znati prosjek suma skokova po svim elementima.

- Udio resursa koji trenutno trošimo. Jednostavno $\frac{trenutni}{maksimalni}$. Imat ćemo konstantu koja predstavlja resurse koji su na raspolaganju, što je gornja granica broja elemenata diskretizacije.

Slično kao ranije, rješenje je glatko po osima. Ovdje su nam skokovi definirani gradijentom. Jasno je da promjene gradijenta na rubovima znače neglatkoću odnosno opisuju mogući gubitak informacija. Također u točci u kojoj se dva segmenta križaju imamo elemente koji imaju više od dva susjeda pa je pitanje hoće li se procesi koji se odvijaju u dvije bitno različite osi miješati.

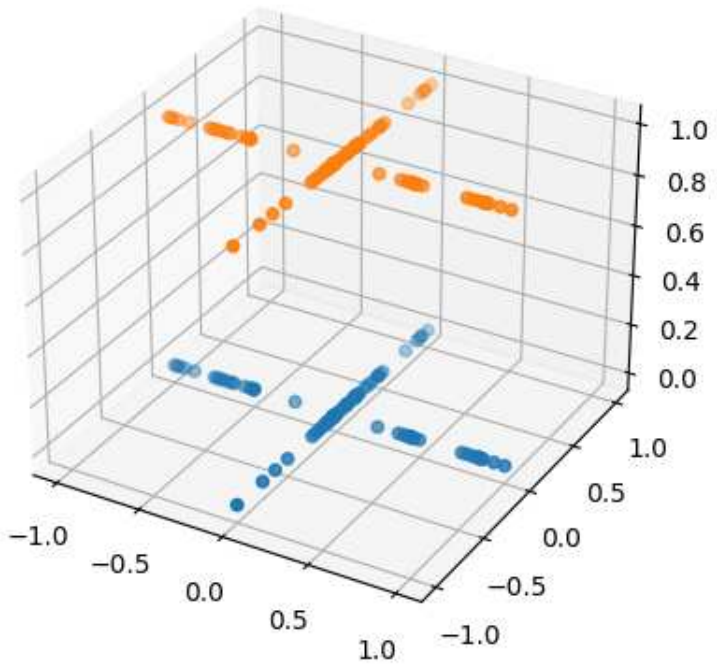
Iz slike 3.3 rješenja jasno je da profinjenja po jednoj osi nemaju nikakvog smisla. Stoga rješenje slikovito prikazuje uspješnost agenta i jasno možemo podijeliti bridove na *dobre* i *loše*. Napomenimo detalj implementacije da su dva brida po osi minimum.



Slika 3.4: Actor-Critic adaptacija mreže konačnih elemenata: samo 4 loše postavljena brida od ukupno 118. Primijetimo da se informacija o potrebi za profinjenjem s osi x nije proširila na os y.

Možemo uzeti u obzir da je ovo jako lagan primjer. S druge strane, ukoliko sada mreži damo degenerirani primjer gdje je rješenje konstantno čak i za najgrublju mrežu, svejedno

će raditi profinjenja. Ako je trenutno malo elemenata postavljeno, cijena profinjenja je niska, a lokalno se svi bridovi čine prosječno dobri jer su svi jednaki.



Slika 3.5: Usporedba konfiguracija koje je generirala primjena Actor-Critic adaptacije mreže konačnih elemenata: svi su bridovi loši.

3.3 MuZero

Ovdje ćemo iskoristiti Markovljev proces odlučivanja (MDP) i ranije spomenute grafovske konvolucije za prijenos informacija. Primjerice, u eliptičkim parcijalnim diferencijalnim jednadžbama (vidi [12]) poznato je da se greške globalnih rješenja propagiraju globalno. Odnosno, teorija regularnosti rješenja eliptičkih diferencijalnih jednadžbi s realno analitičkim koeficijentima daje da je rješenje također realno analitička funkcija.

Ovdje će prostor akcija biti jednak broju bridova i svaka akcija znači pogrubljenje.

Hod algoritma ćemo započeti na uniformno particioniranoj mreži i agent treba naučiti sam odabrati bridove koje treba pogrubiti.

Opažanje

Budući da radimo na grafovima, a zapravo su nositelji značajki bridovi, učinit ćemo sljedeće:

- Konstruiramo novi graf koji ima vrhova koliko smo imali bridova.
- U novom grafu, međusobno su povezani vrhovi ako su bridovi kojima odgovaraju na starom grafu imali zajednički vrh.
- Značajke su rješenja u vrhovima odgovarajućeg brida i duljina brida.

Osim značajki, sada je opažanje i matrica susjedstva. pogrubljenja mijenjaju matricu susjedstva, stoga *MuZero* neće moći biti potpuno *model-based*.

Neuronske mreže

- *Representation* je najjednostavnija, jednostavno preslikava značajke u nove značajke. Ne koristimo *local pooling* za smanjenje broja čvorova jer moramo imati pristup svim čvorovima kada budemo znali akciju.
- *Prediction* koristi *global pooling* za procjenu funkcije vrijednosti stanja. Za π vraća vrijednost (ulaz za *soft-max*, pa to daje vjerojatnost) za svaki čvor.
- *Dynamic* konkatenira *one-hot* vektor matrici značajki čvorova gdje 1 označava odabranu akciju. Nakon toga računa reprezentaciju novog stanja, i pripadnu nagradu.

Što bi to sve trebalo predstavljati?

Prisjetimo se 1.6.1. Najkraće rečeno, algoritam treba naučiti numerički rješavati parcijalne diferencijalne jednadžbe na način da konstruirana po djelovima polinomijalna funkcija lokalno zadovoljava diferencijalni izraz a globalno ima željena svojstva regularnosti (npr. preferira rješenja koja su *neprekidnija*). Primjer pristupa takvom problemu nadgledanim dubokim učenjem je prikazan u radu [25] i vrsta aproksimatora naziva se *fizikom inspirirana (ili ograničena) neuronska mreža* (*engl.* Physics-informed neural network, PINN). Slično, bilo bi zanimljivo pokušati povezati s Fourierovim neuralnim operatorima prikazanim u radu [19] gdje treniramo neuronsku mrežu koja preslikava Fourierove koeficijente (prikaz rješanja u Fourierovoj bazi) kao značajke u Fourierove koeficijente.

Ukoliko bi *MuZero* naučio takav model, naslijedio bi generalizacijske sposobnosti iznesene u članku. Točnije, naš algoritam uči mijenjati rješenja ovisno o promjeni domene, što bi trebalo biti vrlo slično. Ukoliko su pozivi *Dynamic* jeftiniji od *FEM* poziva, i ukoliko se model može dovoljno dobro natrenirati generalizirajući rješenja koja *FEM* daje, procedura bi trebala biti isplativa.

Jedna bizarnost primjera je što koristimo neuronske mreže koje bi trebale naučiti biti *pametnije i brže* od *FEM* poziva kako bismo optimizirali *FEM* pozive adaptacijom mreže konačnih elemenata. No to nije bitno jer je primjer samo demonstracija metode. Također, ukoliko razmišljamo o pravoj hibridnoj metodi adaptivno generirane mreže konačnih elemenata onda je prihvatljivo da generator profinjenja bude heuristički (ali brzi) proces. Validacija konstruirane mreže može se onda prepustiti standardnim adaptivnim metodama konačnih elemenata koje se temelje na rezidualnoj analizi greške. Zapravo, rezidualni indikator greške je samo jedan od konkretnih realizacija procesa *Critic*.

Istaknimo da na idejnoj razini, primjena ovakvih procesa diskretne optimizacije za optimalno oblikovanje struktura pokazuje dodatni generalizacijski potencijal te da bi dobivena metoda mogla u kontekstu efikasnosti biti kompetitivna klasičnim pristupima.

Jedan način generaliziranja učenja s potporom prikazan je u radu [24]. Tamo postoji suparnički agent koji mijenja *pravila igre* tako da našem agentu bude što teže i time nauči generalizirati na različita pravila.

Inače, problem koji rješavamo je zapravo problem kombinatorne optimizacije, čija su pravila opisana diferencijalnim računom. Primjer u [20]. Aktualna su istraživanja dubokog učenja s potporom u smjeru rješavanja *NP-teških problema* diskretne optimizacije. Česti su problemi s grafovima i time rješenja temeljena na *grafovskim neuronskim mrežama* poput [10] [1] [13] [6].

Rezultati

Prvo jedan *prenaučen* primjer gdje se uči na jednom jedinom problemu. Za testiranje naposljetku pokrenemo 10 simulacija i uzmemo najbolju. Koristili smo posljednju epohu.

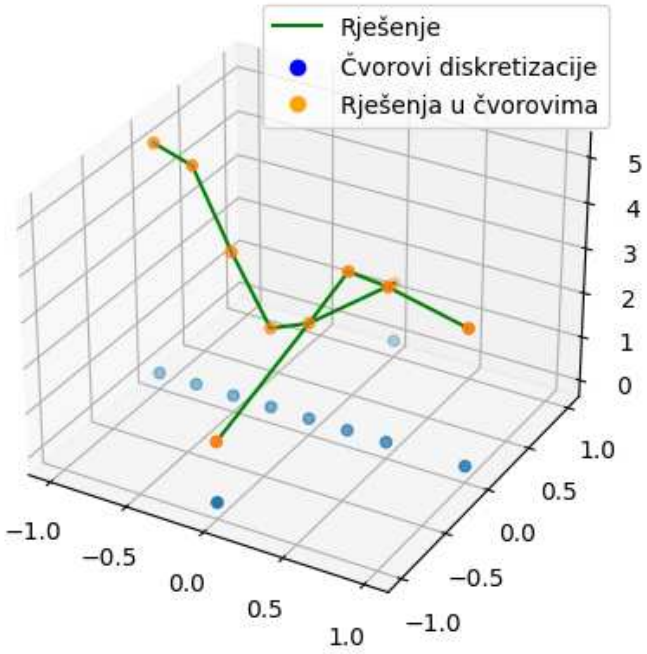
Vidimo na slici 3.6 da je algoritam ostavio sve osim jednog dobrog elementa, i uklonio sve loše elemente. Promotrimo sada vrijednost postignute nagrade kroz hod algoritma.

Iz rezultata sa slike 3.7 vidimo da posljednja epoha nije najbolja unatoč tome što ju je algoritam izabrao.

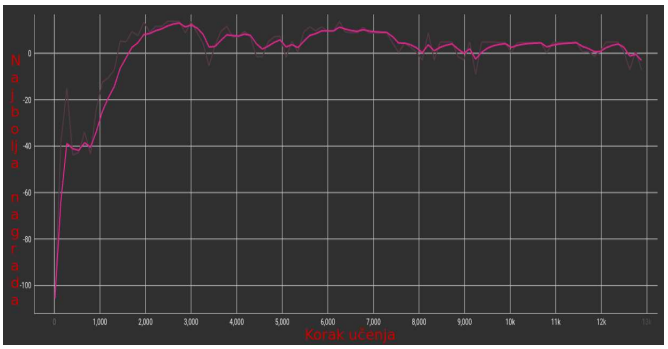
Još ćemo prikazati na 3.8 jedan pokušaj generalizacije. Cijelo vrijeme počinjemo od istog grafa, no dosad smo imali vrh $(0,0)$ i oko njega 4 vrha $(0,1)$, $(0,-1)$, $(1,0)$, $(-1,0)$. Sada ćemo tih 8 brojeva uzorkovati iz uniformne distribucije na intervalu $[-2,2]$ u svakoj iteraciji.

Prošli graf 3.7 se kasnije pogoršavao s vremenom i rješenje je dobro izgledalo, tako da možemo pretpostaviti da je, u najmanju ruku, blizu optimuma. No nije isplativo trenirati složen model koji ne generalizira.

Oscilacije na 3.9 su za očekivati zbog nasumičnosti okoline, pogotovo na početku. Kasnije je nešto naučio, no iz primjera vidimo da je generalizacija neuspjela. Promjene strukture grafa nakon profinjenja su poprilično složene jer nastaju nove veze i mora se znati s kojim elementom će se element koji pogrubljujemo spojiti.

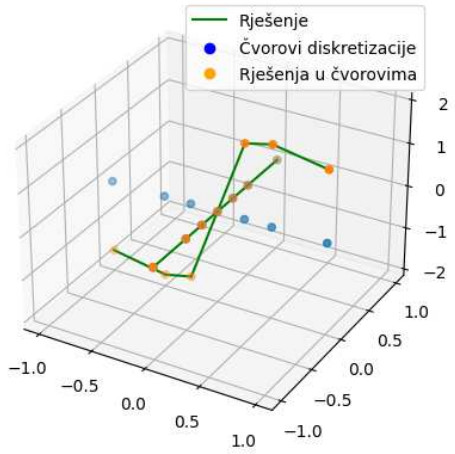


Slika 3.6: MuZero adaptacija mreže konačnih elemenata - prenaučena

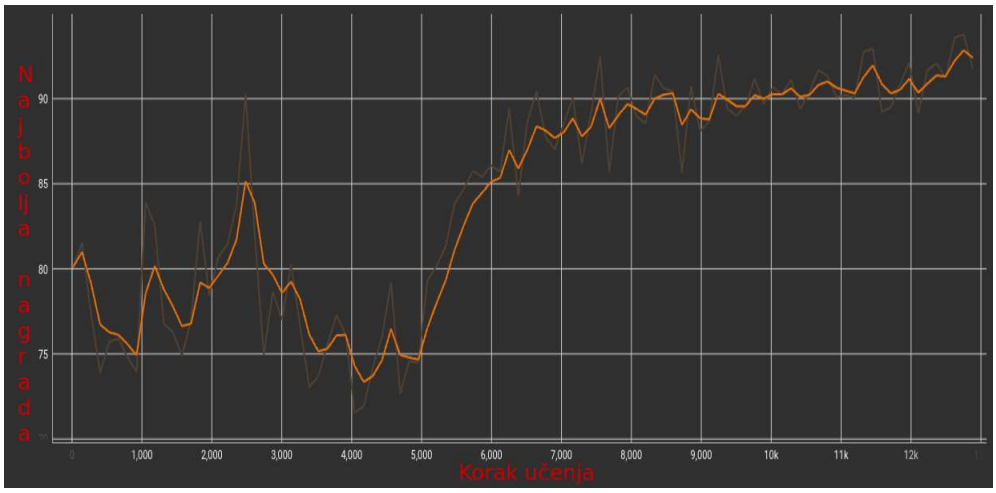


Slika 3.7: Graf najbolje nagrade u svakom vremenskom koraku učenja i pripadno vremensko usrednjenje za 3.6

Općenito u implementaciji nedostaje domenskog znanja o problemu kakvo se koristi u [25] i [19]. Osim tih radova, konceptualno, ovakav pristup bio bi pokušaj kombinacije metoda za optimizaciju struktura zasnovanih na konvolucijama poput [5] i [20] te metoda



Slika 3.8: MuZero adaptacija mreže - generalizirana, primjer loše adaptacije.



Slika 3.9: Graf najbolje nagrade u svakom vremenskom koraku učenja i pripadno vremensko usrednjenje za 3.8

učenja s potporom na grafu kao [10].

Bibliografija

[1] Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros i Albert Cabellos-Aparicio, *Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case*, Computer Communications **196** (2022), 184–194, https://doi.org/10.1016%2Fj.comcom.2022.09.029.

[2] Mario Arioli i Michele Benzi, *A finite element method for quantum graphs*, IMA Journal of Numerical Analysis **38** (2017), br. 3, 1119–1163.

[3] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou i Fabio Viola, *The DeepMind JAX Ecosystem*, 2020, http://github.com/deepmind.

[4] Andrew G. Barto, Richard S. Sutton i Charles W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems*, IEEE Transactions on Systems, Man, and Cybernetics **SMC-13** (1983), br. 5, 834–846.

[5] Nathan K. Brown, Anthony P. Garland, Georges M. Fadel i Gang Li, *"Deep reinforcement learning for engineering design through topology optimization of elementally discretized design domains"*, Materials Design **218** (2022), 110672, ISSN 0264-1275, https://www.sciencedirect.com/science/article/pii/S0264127522002933.

[6] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris i Petar Veličković, *Combinatorial optimization and reasoning with graph neural networks*, 2022.

[7] Pierre Arnaud Coquelin i Rémi Munos, *Bandit Algorithms for Tree Search*, 2007.

[8] Johannes Czech, Patrick Korus i Kristian Kersting, *Monte-Carlo Graph Search for AlphaZero*, 2020.

[9] Ivo Danihelka, Arthur Guez, Julian Schrittwieser i David Silver, *Policy improvement by planning with Gumbel*, International Conference on Learning Representations, 2022, https://openreview.net/forum?id=bERaNdoegn0.

[10] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P. Williamson i Madeleine Udell, *Learning to Solve Combinatorial Optimization Problems on Real-World Graphs in Linear Time*, 2020.

[11] Bowen Fang, *muax*, https://github.com/bwfbowen/muax, 2023.

[12] Corbin Foucart, Aaron Charous i Pierre F. J. Lermusiaux, *Deep Reinforcement Learning for Adaptive Mesh Refinement*, 2022, https://arxiv.org/abs/2209.12351.

[13] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin i Andrea Lodi, *Exact Combinatorial Optimization with Graph Convolutional Neural Networks*, 2019, https://arxiv.org/abs/1906.01629.

[14] Alexandre Girouard, Dmitry Jakobson, Michael Levitin, Nilima Nigam, Iosif Polterovich i Fré déric Rochon (ur.), *Geometric and Computational Spectral Theory*, American Mathematical Society, oct 2017, https://doi.org/10.1090%2Fconm%2F700.

[15] Jan S. Hesthaven i Tim Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, 1st., Springer Publishing Company, Incorporated, 2007, ISBN 0387720650.

[16] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor i Yuhuai Wu, *Stable Baselines*, https://github.com/hill-a/stable-baselines, 2018.

[17] Mladen Jurak, *Metoda konacnih elemenata*, Matematicki odjel PMF-a, 2004.

[18] Tosio Kato, *Perturbation theory for linear operators*, Classics in Mathematics, Springer-Verlag, Berlin, 1995, ISBN 3-540-58661-X, Reprint of the 1980 edition. MR 1335452

[19] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart i Anima Anandkumar, *Fourier Neural Operator for Parametric Partial Differential Equations*, 2021, https://arxiv.org/abs/2010.08895.

[20] Qiyin Lin, Jun Hong, Zheng Liu, Baotong Li i Jihong Wang, *Investigation into the topology optimization for conductive heat transfer based on deep learning approach*, International Communications in Heat and Mass Transfer **97** (2018), 103–109.

[21] Jelena Lončar, *AlphaZero: strojno učenje podrškom bez domenskog znanja (Diplomski rad)*, Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, 2020, https://urn.nsk.hr/urn:nbn:hr:217:006628.

[22] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver i Koray Kavukcuoglu, *Asynchronous Methods for Deep Reinforcement Learning*, 2016.

[23] Alex Oarga, *Haiku Geometric*, https://github.com/alexOarga/haiku-geometric, 2023.

[24] Lerrel Pinto, James Davidson, Rahul Sukthankar i Abhinav Gupta, *Robust Adversarial Reinforcement Learning*, 2017, https://arxiv.org/abs/1602.01783.

[25] Maziar Raissi, Paris Perdikaris i George Em Karniadakis, *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*, 2017.

[26] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner i Gabriele Monfardini, *The Graph Neural Network Model*, IEEE Transactions on Neural Networks **20** (2009), br. 1, 61–80.

[27] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap i David Silver, *Mastering Atari, Go, chess and shogi by planning with a learned model*, Nature **588** (2020), br. 7839, 604–609, https://doi.org/10.1038%2Fs41586-020-03051-4.

[28] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan i Demis Hassabis, *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, 2017.

[29] Richard S Sutton i Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[30] Rich Sutton, *The Bitter Lesson*, 2019, http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

[31] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki i Jacek Mańdziuk, *Monte Carlo Tree Search: a review of recent modifications and applications*, Artificial Intelligence Review **56** (2022), br. 3, 2497–2562, https://doi.org/10.1007%2Fs10462-022-10228-y.

[32] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen i Omar G. Younis, *Gymnasium*, ožujak 2023, https://zenodo.org/record/8127025, posjećena 2023-07-08.

[33] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò i Yoshua Bengio, *Graph Attention Networks*, 2018, https://arxiv.org/abs/1710.10903.

Sažetak

U ovom radu predstavljeni su neki osnovni pojmovi učenja s potporom i dubokog učenja s potporom. Zatim je formuliran problem adaptacije po dijelovima polinomijalnih funkcija koje aproksimiraju rješenje parcijalnih diferencijalnih jednadžbi kao Markovljev proces odlučivanja. Prikazani su rezultati uporabe nekih algoritama dubokog učenja s potporom na problem adaptacije mreže konačnih elemenata u kontekstu numeričkog rješavanja parcijalnih diferencijalnih jednadžbi metodom konačnih elemenata.

Odstupanje od glatkoće koja se želi postići u rješenju predstavljena je razlikama u rješenjima na granici između dva elementa mreže konačnih elemenata. Osim toga, značajka je iskorištenost dostupnih računalnih resursa vezana uz maksimalni dozvoljeni broj elemenata. Odluke se donose za pojedine elemente na temelju opažanja koje je djelomično i koncentrirano na lokalne aproksimacije. Uvedena je nužna teorija za postavljanje problema na metričkom grafu. Primjeri problema čija su rješenja demonstrirana su parcijalne diferencijalne jednadžbe u jednoj dimenziji, te na metričkom grafu.

Koristi se *Actor-Critic* algoritam na djelomično uočljivom Markovljevom procesu odlučivanja. Dan je osvrt i na pretraživanje stabla Monte Carlo metodom i algoritam *MuZero* te njegove moguće primjene u rješavanju navedenih i sličnih optimizacijskih problema. Također, istaknuta je sličnost između opisanih problema i problema diskretne optimizacije. Tako se pojavljuje mogućnost primjene geometrijskog dubokog učenja kojim su postignuti rezultati na zadacima opisanim strukturiranim podacima.

Summary

In this master thesis some basic concepts of reinforcement learning and deep reinforcement learning are presented. Then the problem of adapting piecewise polynomial functions which are used to approximate the solution of partial differential equations as a Markov decision process is formulated. Results of using some algorithms of deep reinforcement learning to solve the adaptive mesh refinement in the context of numerically solving partially differential equations using the finite element method are demonstrated.

The goal is to obtain smooth solutions, and lack of smoothness is measured through the difference between solutions at the boundary which the corresponding elements share. Other than that, a feature of the state is the current usage of available computation resources, meaning there is a predetermined maximum number of allowed elements. Decisions are made for individual elements based on the observation which is partial and focused on local approximations. Necessary theory required to pose the problem of metric graphs is introduced. Examples we used are partial differential equations in one dimension, and on a metric graph.

Actor-Critic algorithm is used on a partially observable Markov decision process. Also, a description of Monte Carlo tree search and the *MuZero* algorithm and their possible applications in solving mentioned optimization problems is given. Also, the similarity between described problems and discrete optimization problems is covered. That brings about the possibility of applying geometric deep learning, as it has shown promise in solving tasks represented by structured data.

Životopis

Rođen sam 1998. u Zagrebu. Završio sam Osnovnu školu Savski Gaj i V. gimnaziju u Zagrebu. Sudjelovao sam na državnim natjecanjima iz matematike.

2017. upisao sam preddiplomski studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu, a zatim diplomski studij Računarstvo i matematika na istom odsjeku.