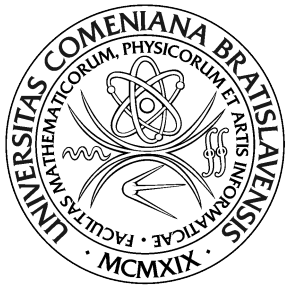
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

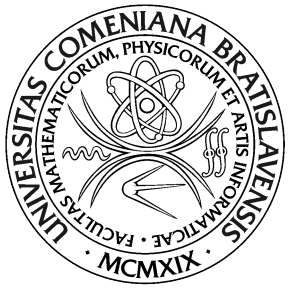


UČENIE POSILŇOVANÍM S INTERNOU MOTIVÁCIOU ZALOŽENOU NA PREDIKTÍVNOM MODELI

Diplomová práca

2022 Bc. Erik Bíly

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



UČENIE POSILŇOVANÍM S INTERNOU MOTIVÁCIOU ZALOŽENOU NA PREDIKTÍVNOM MODELI

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	prof. Ing. Igor Farkaš, Dr.
Konzultant:	Mgr. Matej Pecháč

Bratislava, 2022

Bc. Erik Bíly

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Erik Bíly
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Učenie posilňovaním s internou motiváciou založenou na prediktívnom modeli
Reinforcement Learning with Intrinsic Motivation Based on Predictive Modeling

Anotácia: V súčasnosti predstavuje modelovanie internej motivácie jednu z aktívne sa rozvíjajúcich oblastí učenia posilňovaním, ktorá výrazne skracuje proces učenia a zvyšuje tak jeho efektivitu. Významnú skupinu tvoria motivácie založené na predikčných modeloch, no tie majú svoje slabé miesta ako je obtiažnejšie modelovanie nelineárneho prostredia, či problém bieleho šumu.

Cieľ: Nájdite vylepšenie pre predikčný model, použitý na generovanie motivačného signálu. Dobrým základom je dopredný model, pričom inšpiráciou môže byť učenie posilňovaním založené na modeli prostredia. Vylepšený model otestujte na vybratých Atari hrách a porovnajte ho so základným modelom z hľadiska rýchlosti konvergencie a priemernej získanej externej odmeny.

Literatúra: Aubret A., Matignon L., Hassas S.: A survey on intrinsic motivation in reinforcement learning. arXiv:1908.06976 [cs], 2019.
Pathak D., Agrawal P., Efros A., Darrell T.: Curiosity-Driven Exploration by Self-supervised Prediction. In ICML, 2017.

Vedúci: prof. Ing. Igor Farkaš, Dr.
Konzultant: Mgr. Matej Pecháč
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 19.11.2020

Dátum schválenia: 07.12.2020 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a s pomocou môjho konzultanta a školiteľa.

. . . . . .

Bratislava, 2022 Bc. Erik Bíly

Abstrakt

V učení posilňovaním sa častokrát objavuje problém riedkej odmeny, pri ktorom sa klasickým algoritmom nedarí nájsť vôbec žiadnu pozitívnu odmenu a tým pádom sa agent nedokáže naučiť požadované správanie. Interná motivácia je jedným z riešení problému riedkej odmeny, ktorej cieľom je vytvoriť dodatočnú hustú odmeňovaciu funkciu. Tento prístup je aplikovateľný na ľubovoľné prostredie a navyše sa interne motivovaní agenti spravidla učia rýchlejšie. Oboznámime sa s viacerými spôsobmi implementácie internej motivácie, prejdeme si zopár súčasných riešení a vylepšíme ich pomocou zlomkových vzdialenostných metrík, ktoré sľubujú lepšiu rozlišovaciu schopnosť medzi stavmi vo vysokorozmernom priestore a taktiež pomocou učenia reprezentácie stavu. Tieto vylepšené spôsoby generovania internej motivácie testujeme pomocou troch Atari hier a porovnávame ich z hľadiska dosiahnutej externej odmeny. Dva spôsoby dosahujú veľmi dobré výsledky, ale nemôžeme povedať, že by niektorý z nich bol všeobecne lepší, lebo ich úspešnosť závisí aj od konkrétneho prostredia. Zdrojové kódy a výsledky sú dostupné na https://github.com/Iskandor/MotivationModels/tree/diplomka_fm.

Kľúčové slová: učenie posilňovaním, interná motivácia

Abstract

In reinforcement learning a problem with sparse reward often arises, which causes classic algorithms to not find a single positive reward and therefore the agent is unable to learn the intended behaviour. Intrinsic motivation is one of the solutions to this problem, which generates additional dense reward function for the agent. This approach is universal for all environments and it also causes the agent to learn faster. We outline multiple ways of generating an intrinsic motivation, describe few state-of-the-art methods and enhance them by using fractional distance metrics that promise better resolution capabilities in high-dimensional states or by state representation learning. We evaluate these enhanced approaches on three Atari games and compare them based on achieved external reward. Two of the proposed approaches achieve very good results, however we cannot say that any of them is universally better, because their performance depends also on the environment. The source code with all results can be found at https://github.com/Iskandor/MotivationModels/tree/diplomka_fm.

Key words: reinforcement learning, intrinsic motivation

Zoznam obrázkov

Obsah

Úvod

Učenie posilňovaním (reinforcement learning) je jedna z troch oblastí strojového učenia, popri učení s učiteľom a bez učiteľa. Cieľom učenia posilňovaním je vytvoriť stratégiu, podľa ktorej sa agent rozhoduje a následne vykonáva akcie, za ktoré je odmenený. Cieľom agenta je dosiahnuť čo najväčšiu odmenu z dlhodobého hľadiska. Avšak pri využití učenia posilňovaním v reálnom svete častokrát nastáva situácia, že agent odmenu nedostáva po každej vykonanej akcii, ale až po dlhšej dobe, prípadne až úplne na konci, keď splní svoju úlohu. Pravdepodobnosť, že agent náhodne vykoná správnu sekvenciu akcií, ktorá vedie k takejto riedkej odmene je veľmi nízka hlavne v komplikovanejších prostrediach. V praxi sa problém s riedkou odmenou väčšinou rieši manuálne, kedy ľudský expert z domény daného použitia nadizajnuje dodatočnú hustú odmeňovaciu funkciu, ktorá pomáha agentovi učiť sa požadované správanie. Interná motivácia je univerzálnejšie riešenie problému s riedkou odmenou, pri ktorom tiež tvoríme dodatočnú odmeňovaciu funkciu, ale nie je potrebná žiadna expertná znalosť daného prostredia. Takýto agent je motivovaný explorovať svoje prostredie, čo dodatočne rieši aj dilemu medzi exploráciou a exploatáciou a taktiež urýchľuje proces učenia.

V kapitole 1 popíšeme teóriu učenia posilňovaním, vysvetlíme fungovanie niektorých algoritmov a okrem spomenutého problému riedkej odmeny a dilemy medzi exploráciou a exploatáciou, popíšeme aj prekliatie dimenzionality ako ďalší problém, ktorý sa častokrát vyskytuje pri učení posilňovaním. Podrobnejšie si rozoberieme dve skupiny riešení prekliatia dimenzionality. Následne sa zameriavame na internú motiváciu a jednu skupinu metód internej motivácie, ktoré agenta motivujú na základe chyby predikcie.

V kapitole 2 sa venujeme dvom súčasným riešeniam internej motivácie a jednému enkóderu, ktorý využíva kontrastívne učenie a rieši prekliatie dimenzionality. Podrobne si popíšeme ich fungovanie, pretože súčasťou našej práce sú experimenty, v ktorých tieto metódy využívame.

V kapitole 3 prestavujeme tri Atari hry použité ako prostredia na trénovanie, učiaci algoritmus PPO Proximal Policy Optimization) a jeho implementáciu pomocou neurónových sietí a spôsoby generovania internej motivácie, v ktorých vylepšujeme niektoré metódy z predchádzajúcej kapitoly. Výsledky experimentov prezentujeme v kapitole 4 pomocou grafov a tabuliek dosiahnutých odmien počas učenia.

Kapitola 1

Teória učenia posilňovaním

V tejto kapitole popíšeme učenie posilňovaním, zadefinujeme základné pojmy potrebné na formuláciu a riešenie problému učenia posilňovaním a spomenieme aj rôzne kategórie do ktorých sa rozdeľujú algoritmy riešiace tento problém. Spôsob ako fungujú algoritmy z kategórie gradientových metód učenia stratégie objasníme podrobnejšie a ukážeme ako fungujú niektoré vybrané algoritmy. Nakoniec spomenieme tri problémy, ktoré sa často vyskytujú pri aplikovaní algoritmov učenia posilňovaním a spomenieme aj prístupy, ktorými sa tieto problémy dajú riešiť.

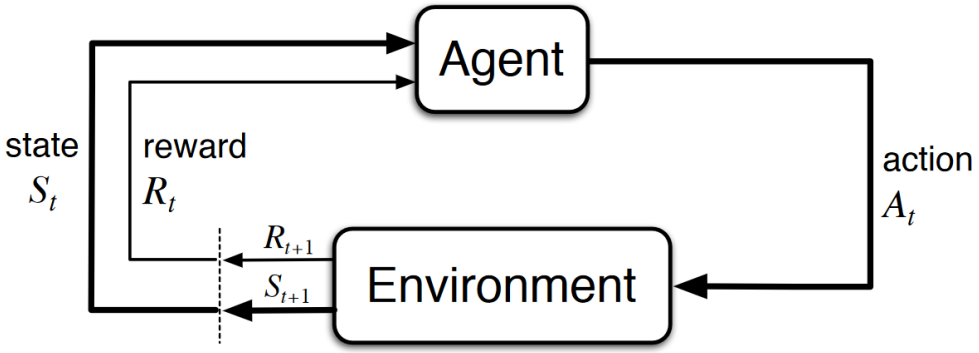
1.1 Učenie posilňovaním

Učenie posilňovaním (reinforcement learning - RL) je jednou z troch oblastí strojového učenia. RL je založené na jednoduchej myšlienke, ktorú môžeme pozorovať napríklad aj v prírode na zvieratách, alebo na ľudoch. Tou hlavnou myšlienkou je, že ak po nejakom správaní nasleduje v organizme pocit satisfakcie, neskôr má organizmus tendenciu takéto správanie častejšie opakovať a hovoríme tomu, že takéto správanie je posilnené (reinforced) (Barto, 2013). Cieľom učenia posilňovaním je naučiť nejakého agenta interagovať s jeho prostredím metódou pokus-omyl. Agent vykonáva akcie, o ktorých si myslí, že mu prinesú maximálnu odmenu z dlhodobého hľadiska. Odmenu dostáva v každom časovom kroku z prostredia, ale niekedy je odmena rovná nule a keďže nulová odmena nijako ne-

prispieva do celkovej odmeny, tak častokrát hovoríme, že pri nulovej odmene agent odmenu vôbec nedostane. Čas väčšinou vnímame ako diskrétne hodnoty $t = 1, 2, 3, ..$, ale tie nemusia reprezentovať plynutie reálneho času. Zo skúsenosti si potom agent upravuje svoje vedomosti o prostredí tak, aby si zapamätal, ktoré akcie priniesli akú odmenu. Agent svojimi akciami mení prostredie a tým pádom akcia, ktorú vykoná niekedy na začiatku, môže ovplyvniť jeho výber akcií neskôr v čase, takže aby správne odhadoval celkovú odmenu, ktorú dostane v budúcnosti, agent potrebuje mať aj nejaké vedomosti o tom, ako jeho akcie vplývajú na prostredie. Okrem toho agent nemá žiadnu vonkajšiu informáciu o tom, ktorú akciu má zvoliť, ale musí sám objaviť ktoré akcie vedú k najväčšej odmene tým, že ich vykoná (Sutton and Barto, 2018).

1.2 Markovov rozhodovací proces

Markov rozhodovací proces (Markov Decission Process - MDP) je matematický framework z oblasti optimálneho rozhodovania, pomocou ktorého vieme problematiku učenia posilňovaním formálne definovať. Ľubovoľný problém bez ohľadu na jeho senzorické, pamäťové a ovládacie špecifikácie a taktiež s akýmkoľvek cieľom, ktorý sa snaží dosiahnuť, sa dá zredukovať na 3 komunikujúce signály medzi agentom a jeho prostredím. Tieto signály sú znázornené na obrázku 1.1.



Obr. 1.1: Jeden signál reprezentuje agentove rozhodnutia (actions), druhý signál reprezentuje stav podľa ktorého sa agent rozhoduje (state) a tretí signál reprezentuje odmenu (reward). Obrázok je prebratý od Sutton and Barto (2018)

MDP sa skladá z nasledujúcej pätice $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\gamma)$ kde:

- $\mathcal{S}$ je množina stavov
- $\mathcal{A}$ je množina akcií
- $\mathcal{T}$ je prechodová funkcia $\mathcal{T}(s'|s,a) = p(S_{t+1} = s'|S_t = s, A_t = a)$. V literatúre sa niekedy označuje aj ako $\mathcal{P}(s'|s,a)$. $s, s' \in \mathcal{S}, a \in \mathcal{A}$
- $\mathcal{R}$ je odmeňovacia funkcia $R(s)$ pre stav $s \in \mathcal{S}$
- γ je diskontný faktor z intervalu $(0, 1]$

Prechodová funkcia $\mathcal{T}$ reprezentuje podmienenú pravdepodobnosť, že sa agent ocitne v stave s' v čase $t+1$, ak sa v čase t nachádzal v stave s a vykonal akciu a. V MDP prechodová funkcia $\mathcal{T}$ kompletne definuje dynamiku prostredia. Inak povedané, pravdepodobnosť každého stavu S_{t+1} a odmeny R_{t+1} závisí iba od predchádzajúceho stavu S_t a akcie A_t. Každý stav teda musí v sebe obsahovať informáciu o predošlých interakciách medzi agentom a prostredím, ktoré by mohli mať efekt aj v budúcnosti. Ak tieto informácie stav obsahuje, hovoríme, že stav má Markovovu vlastnosť. Ďalej môžeme definovať stratégiu (policy) π ako funkciu, ktorá pre stav s vypočíta pravdepodobnosť zvolenia všetkých akcií $a \in \mathcal{A}$.

$$\pi_t(s,a) = p(A_t = a|S_t = s) \quad \text{a platí, že:} \quad \sum_{a \in \mathcal{A}} \pi(s,a) = 1$$

Cieľom agenta je naučiť sa takú stratégiu, aby maximalizovala kumulovanú odmenu R_t, ktorú pomocou diskontného faktora vyjadríme nasledovne:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Čím je diskontný faktor bližší k číslu 1, tým viac agent berie do úvahy odmeny z budúcnosti.

1.2.1 Hodnotové funkcie

Pomocou hodnotových funkcií vieme zkvantifikovať ako výhodné sú jednotlivé stavy pre agenta (alebo dvojice stav a akcia, ktorú v stave agent vykoná). *Výhodnosť* jednotlivých stavov je definovaná pomocou očakávaných odmien v budúcnosti vždy pre konkrétnu stratégiu π podľa ktorej by sa agent rozhodoval. Hodnotová funkcia pre stav s pri dodržovaní stratégie π $V^\pi(s)$ vyzerá nasledovne:

$$V^\pi(s) = \mathbb{E}_\pi[R_t|S_t = s] = \mathbb{E}_\pi\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s\big]$$

Hodnotová funkcia $V^\pi(s)$ sá dá upraviť s vyjadrením nasledujúceho stavu s' z čoho dostaneme Bellmanovu rovnicu:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\big[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|S_t = s\big] \\ &= \mathbb{E}_\pi\big[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + ...)|S_t = s\big] \\ &= \mathbb{E}_\pi\big[r_{t+1} + \gamma(R_{t+1})|S_t = s\big] \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{T}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

Podobne vieme vyjadriť Bellmanovu rovnicu aj z hodnotovej funkcie $Q^\pi(s,a)$ pre dvojicu (stav, akcia):

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t|S_t = s, A_t = a] = \mathbb{E}_\pi\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s, A_t = a\big] \\ &= \mathbb{E}_\pi\big[r_{t+1} + \gamma(R_{t+1})|S_t = s, A_t = a\big] \\ &= \sum_{s'} \mathcal{T}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

Hodnotové funkcie V^π a Q^π sú aproximované z agentových skúseností napríklad tak, že by si agent pre každý stav (príp. pre každú dvojicu (stav,akcia)) ukladal priemernú odmenu, ktorú následne dostal. Ak by sa počet navštívení jednotlivých stavov blížil k nekonečnu, uložený priemer by konvergoval k hodnotám V^π a Q^π. Takéto aproximačné metódy závislé od náhodných vzoriek odmien sa

nazývajú *Monte Carlo metódy.*

1.2.2 Bellmanov princíp optimálnosti

Riešenie problému učenia posilňovaním spočíva v hľadaní stratégie, ktorá z dlhodobého hľadiska dosahuje vysokú odmenu. Každú stratégiu π vieme porovnať s inou stratégiou π' pomocou hodnotových funkcií. Ak $V^{\pi}(s) > V^{\pi'}(s)$ tak hovoríme, že stratégie π je lepšia ako stratégia π'. Vždy existuje aspoň jedna stratégia, ktorá je lepšia, alebo rovnako dobrá ako všetky ostatné stratégie a túto stratégiu voláme *optimálna stratégia.* V MDP môže existovať viacero optimálnych stratégií súčasne, ale všetky zdieľajú rovnaké hodnotové funkcie V a Q. Optimálnu stratégiu značíme π_*. Bellmanove rovnice pre hodnotové funkcie s optimálnou stratégiou vyzerajú nasledovne:

$$\begin{aligned} V^*(s) &= \max_a \mathbb{E}\big[r_{t+1} + \gamma V^*(s_{t+1}) | S_t = s, A_t = a\big] \\ &= \max_a \sum_{s'} \mathcal{T}(s,a,s')\big[\mathcal{R}(s,a,s') + \gamma V^*(s')\big] \end{aligned}$$

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}\big[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | S_t = s, A_t = a\big] \\ &= \sum_{s'} \mathcal{T}(s,a,s')\big[\mathcal{R}(s,a,s') + \gamma \max_{a'} Q^*(s', a')\big] \end{aligned}$$

Z takto vypočítanej hodnotovej funkcie $Q^*(s,a)$ vieme priamo získať optimálnu stratégiu. V praxi sa ale často vyskytuje problém, že nemáme kompletné vedomosti o prostredí, v ktorom sa agent nachádza (niekedy nepoznáme prechodovú funkciu $\mathcal{T}(s,a,s')$, alebo odmeňovaciu funkciu $\mathcal{R}$(s)) a teda na hľadanie optimálnej stratégie nemôžeme priamo použiť Bellmanove rovnice. Aj napriek tomu sú ale Bellmanove rovnice teoretickým základom mnohých RL algoritmov. (Sutton and Barto, 2018)

1.3 Algoritmy učenia posilňovaním

Algoritmy, ktoré počítajú optimálnu stratégiu π^* delíme podľa znalosti o dynamike prostredia a ich spôsobu učenia sa zo skúseností do štyroch hlavných ka-

tegórií: dynamické programovanie, Monte-Carlo metódy, Temporal-Difference (TD) learning a gradientové metódy učenia stratégie. Dynamické programovanie môžeme využiť iba za predpokladu, že poznáme aj prechodovú aj odmeňovaciu funkciu daného prostredia. Monte-Carlo metódy nevyžadujú znalosť prechodovej funkcie $\mathcal{T}$, ale učia sa z histórie interakcií medzi agentom a prostredím. Temporal-Difference je kombinácia medzi dynamickým programovaním a Monte-Carlo metódami. V TD tiež nevyžadujeme znalosť prechodovej funkcie, ktorú nahradíme skúsenosťou a odhadujú sa aj hodnoty odmeňovacej funkcie. Všetky tri doteraz spomenuté kategórie sú takzvané *action-value metódy*, ktoré potrebujú najskôr poznať odhady hodnôt jednotlivých akcií, podľa ktorých sa medzi akciami agent rozhoduje. Gradientové metódy učenia stratégie metódy sú iné v tom, že priamo upravujú stratégiu π pomocou gradientného výstupu. V našej práci sa budeme venovať poslednej skupine algoritmov, ktorej základy budú v ďalšej časti podrobnejšie vysvetlené a tiež popíšeme aj niektoré algoritmy.

1.3.1 Gradientové metódy učenia stratégie

Gradientové metódy sa buď priamo učia *parametrizovanú stratégiu*, pomocou ktorej si agent vyberá akciu bez toho, aby potreboval hodnoty k jednotlivým akciám, alebo druhý spôsob akým môžu fungovať je, že sa namiesto stratégie učia aproximovať hodnotové funkcie, z ktorých sa neskôr dá získať stratégia. Gradientové metódy, ktoré sa učia aproximovať hodnotové funkcie voláme *actor-critic* metódy, kde *actor* reprezentuje naučenú stratégiu a *critic* označuje naučenú hodnotovú funkciu. Vektor parametrov pre stratégiu π označujeme ako $\theta \in \mathbb{R}^{d'}$ a teda pravdepodobnosť, že agent zvolí akciu a s tým, že prostredie je v stave s v čase t a parametrami θ označujeme nasledovne:[1]

$$\pi(a|s, \theta) = p(A_t = a|S_t = s, \theta_t = \theta)$$

Aj v tomto prípade je cieľom maximalizovať odmeňovaciu funkciu, ktorá je

[1]Niekedy sa kvôli jednoduchosti značí aj iba ako $\pi(a|s)$, ak je z kontextu jasné, že ide o parametrizovanú stratégiu.

teraz definovaná ako:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) V^{\pi_\theta}(s) = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s, \theta) Q^{\pi_\theta}(s, a)$$

kde $d^{\pi_\theta}(s)$ je *nemenná distribúcia Markovovho reťazca*, ktorá je definovaná ako pravdepodobnosť stavu $S_t = s$ ak počiatočný stav bol s_0 a nasledujeme stratégiu π_θ v čase $\lim t \to \infty$:

$$d^{\pi_\theta}(s) = \lim_{t \to \infty} p(S_t = s|s_0, \pi_\theta)$$

Gradientové metódy učenia stratégie sú založené na tom, že pomocou gradientného výstupu vieme upraviť parameter θ tak, aby sme maximalizovali odmeňovaciu funkciu $J(\theta)$. Podmienkou samozrejme je, že odmeňovacia funkcia musí mať parciálnu deriváciu vzhľadom na θ, teda existuje $\nabla_\theta J(\theta)$.

Metódy, ktoré sa učia pomocou parametrizovanej stratégie majú mnoho výhod. Môžu sa presne naučiť pravdepodobnosti výberu akcií a tie sa počas učenia menia plynulo, zatiaľ čo pri algoritmoch založených na hodnotových funkciách sa môže pri malej zmene hodnotovej funkcie prudko zmeniť stratégia. Táto vlastnosť zaručuje lepšiu konvergenciu učenia. Taktiež im nerobí problém situácia, kedy výber akcií nie je diskrétny ale spojitý a možno najtriviálnejšou výhodou je, že v niektorých situáciách je jednoduchšie aproximovať stratégiu pre dané prostredie ako hodnotové funkcie.

1.3.2 Teoréma gradientu stratégie

Gradient $\nabla_\theta J(\theta)$ závisí od vybranej akcie a aj od distribúcie nasledovných stavov, ktorá vyplýva zo stratégie π_θ:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s, \theta) Q^{\pi_\theta}(s, a)$$

Nie vždy máme úplne informácie o prostredí a teda nevieme presne, aký efekt by mala zmena stratégie na distribúciu stavov. Počítanie takéhoto gradientu by bolo veľmi komplikované, ale vďaka teorému gradientu stratégie vieme rovnicu

pre $\nabla_\theta J(\theta)$ analyticky upraviť tak, aby neobsahoval deriváciu d^π ako:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s,\theta) Q^{\pi_\theta}(s,a) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s,\theta)\end{aligned}$$

Vychádzame z učebnice (Sutton and Barto, 2018), kde je aj dôkaz. Gradient môžeme ďalej upraviť:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s,\theta) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s,\theta) Q^{\pi_\theta}(s,a) \frac{\nabla_\theta \pi_\theta(a|s,\theta)}{\pi_\theta(a|s,\theta)} \\ &= \mathbb{E}_\pi\big[Q^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s,\theta)\big]\end{aligned} \tag{1.1}$$

Z čoho dostaneme aktualizačné pravidlo pre parametre θ s krokom učenia α:

$$\theta_{t+1} = \theta_t + \alpha \mathbb{E}_\pi\big[Q^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s,\theta)\big]$$

1.3.3 REINFORCE

REINFORCE je jeden z najjednoduchších algoritmov spomedzi gradientových metód učenia stratégie. Tento algoritmus funguje vďaka tomu, že výraz $\nabla_\theta J(\theta)$, ktorý sme v predchádzajúcej kapitole odvodili vieme ešte upraviť nasledovne :

$$\begin{aligned}\nabla_\theta J(\theta) &= E_\pi\big[Q^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s,\theta)\big] \\ &= E_\pi\big[R_t \nabla_\theta \ln \pi_\theta(a|s,\theta)\big]\end{aligned}$$

Algoritmus počíta R_t pomocou kumulovanej odmeny, ktorá zahŕňa odmenu vo všetkych časoch t až po koniec epizódy, kvôli čomu je tento algoritmus niekedy nazývaný aj *Monte-Carlo policy gradient.* Algoritmus môžeme vidieť tu 1.1:

Algorithm 1.1 REINFORCE

1: Náhodne inicializuj parameter stratégie θ
2: Generuj jednu trajektóriu pomocou $\pi_\theta : S_1, A_1, R_2, S_2, A_2, R_3, ..., S_t$
3: **for** $t = 1$ to T **do**
4: Vypočítaj kumulovanú odmenu R_t
5: Aktualizuj parameter stratégie $\theta \leftarrow \theta + \alpha\gamma^t R_t \nabla_\theta \ln \pi_\theta(a|s, \theta)$
6: **end for**

REINFORCE sa častokrát implementuje s drobnou modifikáciou, pri ktorej sa v algoritme 1.1 v kroku 4. odčíta od R_t baseline hodnota pomocou čoho sa zredukuje rozptyl funkcie. Ako baseline sa zväčša používa nasledovná funkcia: $A(s, a) = Q(s, a) - V(s)$, ktorú voláme funkcia výhody (advantage function).

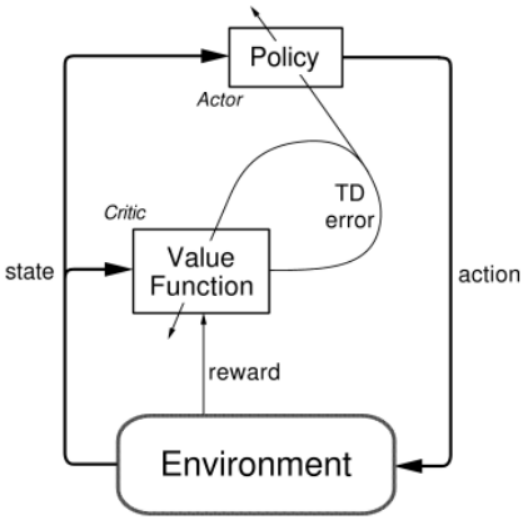
1.3.4 Aktér-kritik

Aktér-kritik (Actor-critic) algoritmy sa skladajú z dvoch hlavných komponentov, ktoré si výpočet gradientu z rovnice 1.1 medzi sebou rozdeľujú. Prvý komponent *kritik* má na starosti parametrizovanú hodnotovú funkciu $Q_w(a|s)$, ktorú počas behu algoritmu vylepšuje a druhý *aktér* reprezentuje stratégiu $\pi_\theta(a|s, \theta)$, ktorá je aktualizovaná pomocou prvého komponentu kritika. Schéma takejto architektúry je zobrazená na obrázku 1.2 a algoritmus je popísaný nižšie (algoritmus 1.2).

Algorithm 1.2 Aktér-kritik

1: Náhodne inicializuj parametre: s, θ, w
2: **for** $t = 1$ to T **do**
3: Získaj vzorku $r_t \sim R(s, a)$ a ďalší stav $s' \sim \mathcal{T}(s'|s, a)$
4: Vyber nasledujúcu akciu $a' \sim \pi_\theta(a'|s', \theta)$
5: Aktualizuj parametre stratégie $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s, \theta)$
6: Vypočítaj *korekciu* hodnotovej funkcie : $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$ a pomocou korekcie uprav parametre w: $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
7: Aktualizuj $a \leftarrow a'$ a $s \leftarrow s'$
8: **end for**

Členy α_θ a α_w v algoritme 1.2 reprezentujú dve rôzne rýchlosti učenia, a_θ je určená pre actora a α_w pre kritika.



Obr. 1.2: Aktér-kritik architektúra prebratá z učebnice Sutton and Barto (2018)

1.3.5 Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) (algoritmus 1.3) je už jedným zo súčasných gradientových metód zameraný na paralelný tréning, kde súčasne beží niekoľko inštancií aktéra a kritika. Každá z nich má vlastné lokálne parametre θ' a w', samostatne interaguje s prostredím a počíta si potrebnú zmenu parametrov na optimalizáciu riešenia. Po prekročení vymedzeného času, alebo dosiahnutí konečného stavu sa globálne parametre θ a w upravia pomocou doteraz nazbieraných gradientov. Takýto prístup lineárne znižuje čas potrebný na učenie v závislosti od počtu vlákien a je výhodný aj tým, že preskúma viac rôznych stavov, keďže agenti konajú nezávisle od seba.

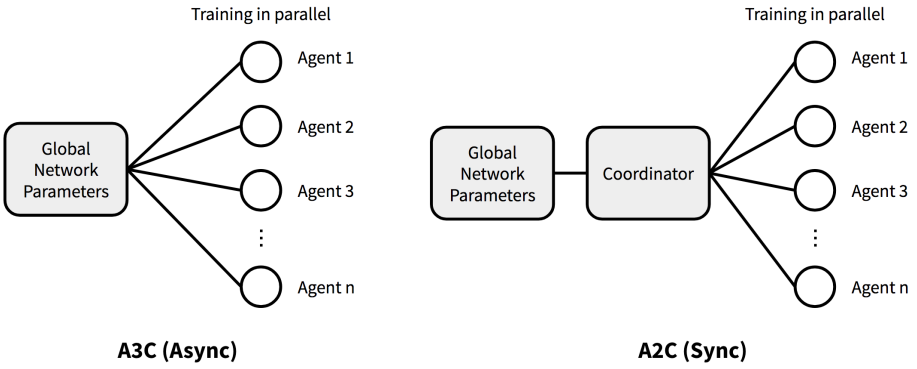
Algorithm 1.3 A3C

Máme globálne parametre θ, w a im podobné parametre θ', w',
ktoré si interne pamätá každé vlákno
Nastav časový krok $t = 1$
while $T \leq T_{\max}$ **do**
  Vynuluj gradient $\nabla_\theta = 0$ a $\nabla_w = 0$
  Zosynchronizuj parametre všetkých vlákien: $\theta' = \theta$ a $w' = w$
  $t_{\text{start}} = t$ a získaj počiatočný stav s_t
  while $(s_t! = TERMINAL)$ and $(t - t_start \leq t_{max})$ **do**
    Vyber akciu $A_t \sim \pi_{\theta'}(A_t|S_t, \theta')$ a získaj odmenu R_t a nasledujúci stav s_{t+1}
    $t = t + 1$ a $T = T + 1$
  end while
  Odhad odmeny inicializuj nasledovne:

$$R = \begin{cases} 0 & \text{if } s_{t_{\text{end}}} \\ V_{w'}(s_t), & \text{inak} \end{cases}$$

  for $i = t - 1$ to t_{start} **do**
    $R \leftarrow \gamma R + R_i$
    Vypočítaj celkový gradient $d_\theta \leftarrow d_\theta + \nabla_{\theta'} \log \pi_{\theta'}(a_i|s_i, \theta')(R - V_{w'}(s_i))$
    Vypočítaj celkový gradient $d_w \leftarrow d_w + 2(R - V_{w'}(s_i))\nabla_{w'}(R - V_{w'}(s_i))$
  end for
  Asynchrónne aktualizuj θ pomocou d_θ a w pomocou d_w
end while

A2C je synchrónna verzia algoritmu A3C, pri ktorom agenti neaktualizujú globálne parametre θ, w svojvoľne, ale svoje výsledky posielajú koordinátorovi, ktorý najskôr počká na všetkých agentov a až potom upraví globálne parametre a začína ďalší cyklus. Na obrázku1.3 je graficky znázornený tento rozdiel. Týmto sa zamedzí možnosti, že niektorý agent ešte pracuje s už neaktuálnou stratégiou, čím môže zhoršiť výpočet globálnych parametrov. Aj napriek tejto synchrónnosti je algoritmus stále časovo veľmi efektívny a väčšinou dosahuje lepšie výsledky ako A3C (Wu et al., 2017).



Obr. 1.3: Architektúra A3C v porovnaní s A2C

1.3.6 Proximal Policy Optimization

Algoritmus Proximal Policy Optimization (PPO) tiež patrí medzi aktér-kritik algoritmy, ktorý funguje podobne ako jeho predchodca TRPO. PPO sa snažia stratégiu vylepšiť o čo najviac v každom kroku, ale zároveň dáva pozor, aby túto zmenu neprehnal, keďže to bývalo častokrát kontraproduktívne. PPO je oproti TRPO z hľadiska výpočtovej zložitosti oveľa jednoduchší, taktiež sa ľahšie implementuje ich výkony sú porovnateľné (Schulman et al., 2017). Existujú dve primárne varianty tohto algoritmu: *PPO-Penalty* a *PPO-Clip* a druhú variantu si ukážeme na pseudokóde 1.4.

V pseudokóde 1.4 sa chyba L v šiestom kroku počíta nasledovne:

$$L(s, a, \theta_k, \theta) = \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a)))$$

kde

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A \leq 0 \end{cases}$$

V experimentoch budeme na trénovanie stratégie využívať najmä algoritmus PPO, ktorý v porovnaní s ostatnými algoritmami dosahuje veľmi dobré výsledky v prostredí Gravitar.

Algorithm 1.4 PPO-Clip

1: Náhodne inicializuj nasledujúce parametre: θ_0, ϕ
2: **for** $k = 0, 1.2, ...$ **do**
3: Zozbieraj množinu trajektórií $\mathcal{D}_k = \{\tau_i\}$ pomocou stratégie $\pi_k = \pi(\theta_k)$
4: Vypočítaj odmeny $\hat{R}_t$
5: Vypočítaj funkciu výhody $\hat{A}_t$ pomocou hodnotovej funkcie V_{ϕ_k}
6: Uprav stratégiu:

$$\theta_{k+1} = \arg \underset{\theta}{max} \underset{s,a \sim \pi_{\theta_k}}{\mathbb{E}} [L(s, a, \theta_k, \theta)]$$

Napríklad pomocou optimalizátora Adam
7: Natrénuj hodnotovú funkciu regresiou:

$$\phi_{k+1} = arg\, min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

Niektorým z algoritmov gradientného zostupu.
8: **end for**

1.4 Výzvy pri učení posilňovaním

Pri aplikovaní algoritmov učenia posilňovaním sa v praxi častokrát stretávame s viacerými komplikáciami. V tejto kapitole si popíšeme tri z nich: problém riedkej odmeny, dilema medzi exploráciou a exploatáciou a nakoniec prekliatie dimenzionality (curse of dimensionality), pretože v našej práci sa venujeme internej motivácií, metóde, ktorá dokáže riešiť prvé dva problémy a do určitej miery riešime aj posledný zo spomenutých problémov pomocou učenia reprezentácie stavu.

1.4.1 Dilema medzi exploráciou a exploatáciou

Jedným z najcharakteristickejších problémov učenia posilňovaním je hľadanie rovnováhy medzi exploráciou a exploatáciou. V časti 1.2 sme písali, že cieľom agenta je správať sa tak, aby nazbieral čo najväčšiu kumulovanú odmenu. Na jednej strane to znamená to, že by sa agent mal rozhodovať na základe svojich doterajších vedomostí o prostredí a vyberať si akcie, ktoré prinesú najväčšiu odmenu (takzvané greedy/chamtivé akcie), ale na druhej strane, na to aby jeho vedomosti

o prostredí boli čo najpresnejšie, agent by mal občas vyskúšať aj iné ako chamtivé akcie, napríklad tie, ktoré doteraz neskúsil. Keď sa agent rozhodne vykonať chamtivú akciu hovoríme, že exploatuje svoje aktuálne vedomosti o prostredí. Ak sa ale rozhodne pre nejakú inú ako chamtivú akciu , hovoríme, že agent exploruje. Explorácia umožnuje agentovi objavovať nové stavy prostredia, do ktorých by sa inak vôbec nedostal a vďaka tomu dokáže lepšie odhadnúť hodnotové funkcie.

Z pohľadu jedného časového kroku je vždy výhodnejšie exploatovať aktuálne vedomosti, ale z dlhodobého hľadiska explorácia môže priniesť vyššiu celkovú odmenu, aj keď občas vyberáme akcie, ktoré sú horšie. Keďže sa nedá naraz aj explorovať aj exploatovať, agent si musí medzi týmito dvoma možnosťami nájsť nejaký kompromis (exploration vs exploitation trade-off). Jednou základnou exploračnou metódou je ϵ-greedy výber akcií, pri ktorej sa agent rozhoduje podľa hodnotových funkcií. S pravdepodobnosťou $1 - \epsilon$ si agent vyberie akciu s najväčšou hodnotou, ale s pravdepodobnosťou ϵ si agent náhodne vyberie niektorú z ostatných akcií (Sutton and Barto, 2018). V časti 1.5 ukážeme sofistikovanejšie riešenie explorácie pomocou internej motivácie agenta.

1.4.2 Riedka odmena

Veľkou výhodou učenia posilňovaním oproti učení s učiteľom je, že pri učení posilňovaním nie je potrebná detailná spätná väzba. Inak povedané, nie je potrebné vedieť, ktorú z akcií by si mal agent po správnosti v danom stave zvoliť, lebo odmenu je možné vygenerovať pre hociktorú z možných akcií. Ale úspech celkového učenia vysoko závisí od toho, ako dobre odmeňovacia funkcia reprezentuje cieľ návrhára aplikácie a aj progres pri dosahovaní daného cieľa. Navrhovanie správnej odmeňovacej funkcie je preto kľúčovou častou každej aplikácie učenia posilňovaním.

Algoritmy, ktoré sme uviedli v časti 1.3 fungujú správne v prostrediach, kde sú odmeny husté, t. j. keď agent dostáva nenulovú odmenu takmer po každej vykonanej akcii. V takýchto prostrediach nám postačujú aj jednoduché exploračné metódy ako ϵ-greedy výber akcií, ktorú sme uviedli v predošlej kapitole, alebo pridávanie Gaussovho šumu do výberu akcií (Lillicrap et al., 2016). V učení

posilňovaním sa ale stretávame aj so situáciou, kedy agent dostáva nenulovú odmenu iba zriedkavo, väčšinou až po vykonaní dlhej a špecifickej postupnosti akcií. Predstavme si príklad, kedy učiaci sa agent ovláda robotickú ruku a jeho cieľom je otvoriť krabicu a vložiť do nej hraciu kocku. Definovanie odmeňovacej funkcie je v takomto prípade veľmi jednoduché a priamočiare (napríklad vložíme do krabice nejaký senzor, ktorý keď deteguje objekt, tak pošle agentovi odmenu), ale naučiť túto aktivitu agenta už také jednoduché nie je. Agent potrebuje získať odmenu na to, aby sa niečo naučil, čiže musí najskôr vykonať postupnosť takých akcií, ktorými otvorí krabicu a potom ďalšiu dlhú postupnosť akcií vďaka ktorej chytí kocku a vloží ju do krabice (za jednu akciu považujeme otočenie jednotlivých motorov v kĺboch ruky o nejaký uhol). V takýchto prostrediach s riedkou odmenou je skoro až nemožné naučiť agenta riešit danú úlohu aj s použitím doteraz spomenutých exploračných stratégií, pretože sa agentovi vôbec nemusí podariť nájsť odmenu a teda netuší ako by sa mal zlepšovať (Mnih et al., 2015). Na obrázku 1.4 je ilustrovaný tento problém riedkej odmeny na triviálnom prostredí.

Obr. 1.4: Ilustrácia problému riedkej odmeny v prostredí, kde sa agent (guličkа) snaží dostať do cieľa (hviezdička). Agent dostane odmenu až keď príde do cieľa. Na ľavom obrázku agent exploruje prostredie pomocou štandardných exploračných stratégií ako je napríklad ϵ-greedy a vidíme, že bezhlavo blúdi viac-menej na mieste . Chceli by sme ale docieliť, aby agent spreskúmaval stavový priestor rozumenjšie, tak ako vidíme na pravom obrázku. Obrázok je prebraný od Aubret et al. (2019).

Namiesto snahy o vytvorenie sofistikovanejších exploračných stratégií, ktoré by efektívne prebádali prostredie s riedkou odmenou, sa častokrát v praxi používa *tvarovanie odmeny (reward shaping)* (Su et al., 2015), kedy obohacujeme

pôvodnú odmeňovaciu funkciu dodatočnými odmenami. Týmto spôsobom dokážeme "vyplniť medzery v riedkej odmene", teda agentovi vieme vygenerovať nejakú odmenu, ktorá by bola pôvodne nulová a tým pádom agenta dokážeme potrestať alebo odmeniť za jeho správanie. Myšlienka tvarovania odmeny nie je nová (Mataric, 1994), ale stále sa využíva aj v moderných aplikáciach, ako bol napríklad úspešný agent od OpenAI hrajúci videohru Dota 2 (Berner et al., 2019).

Tvarovanie odmeny má ale aj svoje nevýhody. Dodatočné odmeňovacie funckie, ktoré pridávame k primárnej odmeňovacej funkcii sú väčšinou manuálne zkonštruované ľudskými expertmi z dómeny danej aplikácie. Ďalším problémom pri tvorení dodatočnej odmeňovacej funkcie človekom býva to, že agent pri učení inklinuje k stratégiam, ktoré sú už luďom známe, namiesto toho aby prišiel s niečim novým. Taktiež sa môže stať, že agent síce bude dostávať pozitívne odmeny, ale svoj hlavný cieľ nikdy nedosiahne, lebo tieto odmeny budú generované iba dodatočnými odmeňovacími funkciami.

My si v práci ukážeme, že aj problém riedkej odmeny sa dá riešiť internou motiváciou, metódou, ktorá generuje agentovi dodatočnú odmenu počas trénovania, ale ponecháva agentovi možnosť hľadať nové, prípadne vhodnejšie stratégie. Interná motivácia taktiež nevyžaduje žiadne znalosti z danej domény, takže nie je potrebný zásah ľudského experta.

1.4.3 Prekliatie dimenzionality

Prekliatie dimenzionality je spoločným problémom viacerých oblastí, ktoré pracujú s vysoko rozmernými dátami, ako sú okrem učenia posilňovaním napríklad učenie s učiteľom, alebo učenie bez učiteľa. (Bellman, 1957) za prekliatie dimenzionality označuje viaceré komplikácie, ktoré sa začínajú objavovať s prudkým nárastom priestorových dimenzií dát, ako je napríklad znížená schopnosť vizualizácie, neefektívne ukladanie dát, ale hlavne exponenciálny nárast výpočtovej zložitosti algoritmov, ktoré pracujú s takýmito dátami. Niektoré algoritmy umelej inteligencie dokonca strácaju schopnosť spŕavne fungovať (Aggarwal et al., 2001). Na prvý pohľad by nám síce mohlo prísť logické, že čím máme vyšší počet dimenzií, tým viac informácií o dátach dokážeme uložiť a teda tým lepšie výsledky by sme mali od algoritmov dostať, ale v praxi to od určitého bodu pre-

stáva platiť, lebo s narástajúcim počtom dimenzií prichádza aj väčšia pravdepodobnosť výskytu šumu v dátach a taktiež niektoré dimenzie bývajú častokrát úplne redundantné. Ak sa pozrieme znova na náš príklad s robotickou rukou, ktorá mala za úlohu otvoriť krabicu a vložiť do nej kocku, a chceli by sme takýto takýto tréning uskutočniť v reálnom svete, jedným spôsobom, ako získať informáciu o stave prostredia by bolo robotickú ruku aj s objektmi snímať kamerami. Takýto stav by mal veľmi vysokú dimenziu (aby sme boli schopný lokalizovať kocku v priestore, potrebujeme obraz aspoň z dvoch kamier. Aj keby sme použili kamery s nižším rozlíšením napríklad 1280x1024, tak stále dostaneme 2 621 440 dimenzionálny stav prostredia, s ktorým agent pracuje v každom časovom kroku.), ale nám by na splnenie úlohy úplne stačili iba informácie o pozíciach v priestore kocky, krabice a jednotlivých kĺbov robotickej ruky (pozíciu v priestore každého objektu vieme zapísať pomocou troch čísiel: hodnoty na súradnicových osiach x,y,z a teda ak by naša robotická ruka mala päť kĺbov, dostali by sme iba 21-rozmerný stav prostredia). Dnes sa už vďaka hlbokému učeniu posilňovaním dajú riešiť podobné problémy ako je tento náš ilustračný aj priamo, teda že algoritmy pracujú so surovým vstupom z kamier (Mnih et al., 2015), ale ak by sme vedeli z obrazu kamier nejako získať spomenuté 3D pozície, veľmi by sme zefektívnili trénovanie agenta (Munk et al., 2016).

1.4.3.1 Učenie reprezentácie stavu

Takáto redukcia dimenzie dát je vo všeobecnosti nazývaná *učenie význačných vlastností* (feature learning) a jej cieľom je z dát vytiahnúť význačné vlastnosti, ktoré budú nižšej dimenzie, ale zároveň budú plnohodnotne charakterizovať pôvodné dáta a irelevantné aspekty dát budú ignorované. Tento proces v strojovom učení a robotike bol donedávna vykonávaný z velkej časti manuálne odborníkmi z oblasti využitia (Lesort et al., 2018). *Učenie reprezentácie stavu* (state representation learning - SRL) je špecifický prípad učenia význačných vlastností, kedy sa reprezentácia dát hľadá pomocou hlbokého učenia so snahou minimalizovať ľudský zásah do procesu. Pri učení posilňovaním sa na redukovanie dimenzie stavu prostredia vužívajú aj ďalšie informácie ako je napríklad akcia, alebo odmena. V literatúre venujúcej sa SRL sa stav prostredia nazýva *pozorovanie* (observation o)

a naučená reprezentácia pozorovania sa nazýva *stav* (state s), čo je trochu nekonzistentné s terminológiou MDP (Burda et al., 2019; Lesort et al., 2018). Formálne zapísané, SRL sa učí mapovaciu funkciu ϕ medzi pozorvaniami a stavmi:

$$s_t = \phi(o_t)$$

a prípadne medzi parametre funkcie ϕ môže byť ešte pridaná akcia a_t, alebo odmena r_t.

Böhmer et al. (2015) hovoria, že naučená reprezentácia je dobrá ak má nasledujúce vlastnosti:

- Je Markovovská, teda ak aktuálny stav obsahuje všetky informácie potrebné na to, aby sa agent mohol rozhodnúť pre správnu akciu.
- Stav reprezentuje prostredie dostatočne dobre a nekomplikuje učiacim algoritmom vylepšovať stratégiu.
- Ak sú nejaké dve pozorovania podobné, tak aj z nich vypočítané stavy budú podobné.
- Dimenzia stavu je dostatočne nízka na zvýšenie efektívnosti výpočtov.

Dopredu sa ale nedá zaručiť, že učenie reprezentácie stavu nadobudne všetky tieto vlastnosti, no neskôr sa to dá preveriť.

Medzi najčastejšie používané stratégie implementácie učenia reprezentácie stavu patrí autoenkóder (auto-encoder), dopredný model (forward model) a inverzný model (inverse model). SRL môže byť vo všetkých týchto prípadoch implementované pomocou neurónových sietí, kedy sa menia parametre siete θ v závislosti od minimalizovania nejakej chyby.

Autoenkóder

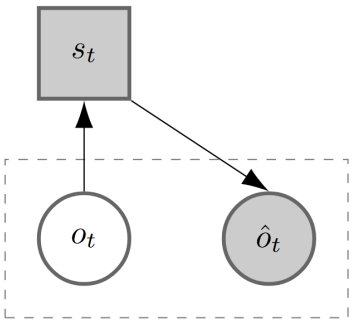
Autoenkóder (obr. 1.5) sa skladá z dvoch častí. Prvá časť nazývaná *enkóder* ϕ slúži na zakódovanie pozorovania o_t do stavu s_t:

$$s_t = \phi(o_t; \theta_\phi)$$

a druhá, *dekóder* ϕ^{-1}, sa snaží stav s_t naspäť dekódovať na pôvodné pozorovanie $\hat{o}_t$:

$$\hat{o}_t = \phi^{-1}(s_t; \theta_{\phi^{-1}})$$

Enkóder sa potom učí na základe chyby medzi pozorovaním a zrekonštruovaným pozorovaním z dekódera.



Obr. 1.5: Schéma autoenkódera. Chyba sa ráta vo vyznačenom obdĺžniku medzi pozorovaniami o_t a $\hat{o}_t$. Sivou farbou sú vyznačené časti ktoré sú vypočítané autoenkóderom a bielou sú pozorované dáta priamo z prostredia. Prevzaté z Lesort et al. (2018).

Dopredný model

Dopredný model f predikuje stav nasledujúceho časového kroku $\hat{s}_{t+1}$ na základe stavu s_t a akcie a_t:

$$\hat{s}_{t+1} = f(s_t, a_t; \theta_f)$$

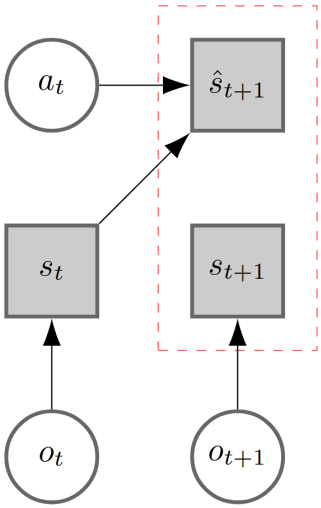
Stav s_t je najskôr vypočítaný z pozorovania o_t funkciou ϕ

$$s_t = \phi(o_t; \theta_\phi)$$

ale chybu počítame v až v ďalšom časovom kroku a to medzi predikovaným stavom $\hat{s}_{t+1}$ a reálnym stavom v s_{t+1} ako je znázornené na obrázku 1.6.

Inverzný model

Inverzný model g zobrazený na obrázku 1.7 podobne ako dopredný model najskôr zakóduje pozorovanie o_t do stavu s_t funkcou ϕ. Druhý krok spočíva v tom,



Obr. 1.6: Dopredný model. Sivá farba znova označuje vypočítané časti a biela dáta sledované v prostredí a medzi komponentami označenými obdĺžnikom sa počíta chyba (Lesort et al., 2018).

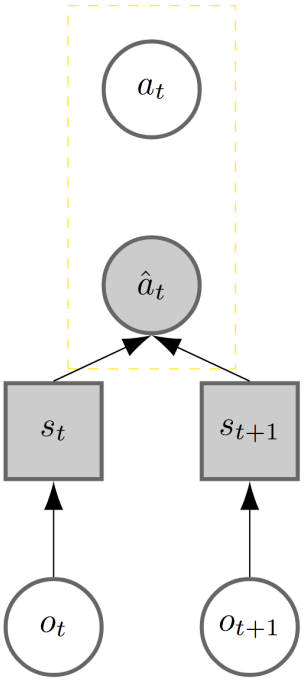
že keď už je k dispozícií aj pozorovanie o_{t+1} z ďalšieho časového kroku a teda aj stav s_{t+1}, tak predikuje akciu $\hat{a}_t$ ktorá viedla k prechodu zo stavu s_t do s_{t+1}.

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_g)$$

Na základe vypočítanej chyby medzi predikovanou akciou $\hat{a}_t$ a skutočnou akciou a_t sa funkcia ϕ učí.

InfoMax

Pri doteraz spomenutých stratégiach sa enkóder ϕ vždy učil pomocou chyby medzi výpočtom a správnym výsledkom (napríklad chyba medzi nasledujúcim stavom s_{t+1} a predikovaným stavom $\hat{s}_{t+1}$). Iný spôsob implementácie SRL, kde sa enkóder neučí z chyby, ale maximalizuje *vzájomnú informáciu* (mutual information - MI) $\mathcal{I}$ medzi pozorovaním o_t a stavom s_t sa nazýva InfoMax. Vzájomná informácia medzi dvoma náhodnými premennými X a Y je definovaná pomo-



Obr. 1.7: Inverzný model. Farby a obdĺžnik majú rovnaký význam ako pri predchádzajúcich obrázkoch (Lesort et al., 2018).

cou Kullback-Leiblerovej divergencie D_{KL} ako

$$\mathcal{I} = D_{\mathrm{KL}}(p(x, y)||p(x)p(y))$$

kde $p(x, y)$ označuje spoločné rozdelenie pravdepodobnosti a členy $p(x)$ a $p(y)$ sú marginálne rozdelenia pravdepodobnosti.

V praxi tieto distribúcie väčšinou nie sú známe a počíta sa iba dolný odhad vzájomnej informácie napríklad pomocou Jensen-Shannonovej divergencie, alebo pomocou Information Noise-Contrastive Estimation (InfoNCE). Takéto odhady sú postačujúce, pretože ide o maximalizovanie vzájomnej informácie a nie o presnú hodnotu (Hjelm et al., 2019).

1.4.3.2 Vzdialenostné metriky

Vzdialenostné metriky sa v umelej inteligencii používajú na vyjadrenie podobnosti alebo vzdialenosti medzi dvoma bodmi v priestore a vďaka tejto vzdialenosti sa niektoré algoritmy učia riešiť daný problém. Aggarwal et al. (2001) tvrdia, že oveľa závažnejší problém ako znižnovanie efektívnosti algoritmov je skutočnosť, že pri mnohých distribúciách dát a s použitím klasických vzdialenostných metrík je pomer vzdialenosti najbližšieho a najvzdialenejšieho suseda ľubovolného bodu vo vysokorozmernom priestore skoro vždy rovný jednej. Inými slovami povedané, všetky body vo vysokorozmernom priestore sú od seba približne rovnako vzdialené a tým pádom koncept vzdialenosti prestáva dávať zmysel. Existuje viacero vzdialenostných metrík, ale nie je vždy jednoznačné ako z nich správne vybrať tú najzmysluplnejšiu. Nasledujúce tri metriky používame a porovnávame aj v našich experimentoch.

Euklidovská vzdialenosť - L2 norma

Medzi najpoužívanejšie vzdialenostné metriky patrí Euklidovská vzdialenosť (niekedy nazývaná aj L2 norma) asi aj z toho dôvodu, že pomocou tejto metriky počítame vzdielenosti bodov v mnohých aplikáciach v reálnom živote, napríklad Pytagorova počíta Euklidovskú vzdialenosť v dvojrozmernom priestore. Euklidovskú vzdialenosť $dist^{L2}$ vieme pre ľubovolné vektory x, y dimenzie d zapísať nasledovne: [2]

$$dist^{L2}(x, y) = \sqrt{\sum_{i=1}^{d}(x^i - y^i)^2}$$

Manhattanská vzdialenosť - L1 norma

Manhattanská vzdialenosť sa počíta ako súčet absolútnych hodnôt rozdielov medzi jednotlivými komponetami dvoch vektorov:

$$dist^{L1}(x, y) = \sum_{i=1}^{d} |x^i - y^i|$$

[2] x^i v tomto prípade znamená i-ty prvok vektora x

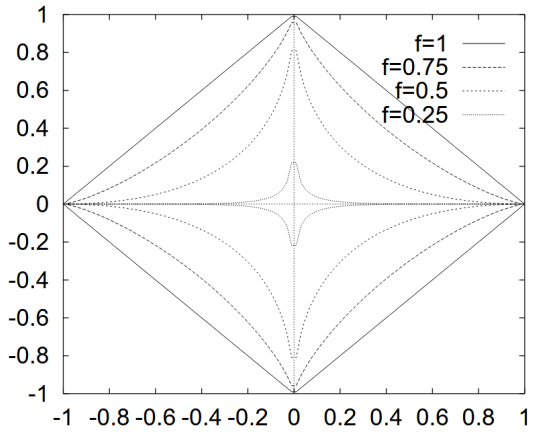
Pri dátach s vysokou dimenziou Aggarwal et al. (2001) preferujú Manhattanskú vzdialenosť pred Euklidovskou a vo všeobecnosti tvrdia, že Lk norma s narastajúcim k stráca na rozlišovacej schopnosti a toto tvrdenie aj matematicky dokazujú.

Zlomkové vzdialenosti

Autori ďalej ukazujú, že predošlé tvrdenie platí aj pre $k < 1$ a definujú zlomkové vzdialenosti (fractional distance metrics) $dist^f$ pre $f \in (0, 1)$ ako:

$$dist^f(x, y) = \sum_{i=1}^{d} [(x^i - y^i)^f]^{1/f}$$

Lepšia rozlišovacia schopnosť ale automaticky ešte nezaručuje vyšší úspech natrénovaného algoritmu. Obrázok 1.8 graficky porovnáva správanie niektorých zlomkových vzdialeností v dvojrozmernom priestore.



Obr. 1.8: Jednotkové kružnice pre zlomkové vzdialenosti s rôznym parametrom f. Prevzaté z Aggarwal et al. (2001).

1.5 Interná motivácia

Vo vývinovej psychológií ľudí, ale aj pri niektorých zvieratách sa rozlišuje medzi tým, či je organizmus motivovaný k svojim činom interne alebo externe. Ryan and Deci (2000) hovoria o *internej motivácii* (intrinsic motivation, IM) vtedy, keď

je nejaká aktivita vykonávaná z dôvodu vnútorného uspokojenia, zabávania sa, alebo splnenia nejakej vlastnej výzvy. Takéto správanie môžeme častokrát pozorovať napríklad pri malých deťoch, ktoré skúmajú svoje prostredie tým, že chytajú, ochutnávajú a búchajú do všetkého, čo im príde pod ruku, ale aj pri dospelých ľuďoch, ktorý z dôvodu internej motivácie napríklad čítajú knihy, alebo lúštia krížovky. Na druhú stranu, ak k činnosti viedla externá motivácia, vo výsledku očakávame nejakú explicitnú odmenu. Samozrejme, nie vždy sme motivovaný iba externe alebo iba interne a teda k nejakej aktivite môže viesť kombinácia týchto dvoch motivácií. Toto sledovanie viedlo k tomu, že sa už dávnejšie aj v učení posilňovaním začala okrem externej motivácie (za ktorú v RL považujeme nazbieranú odmenu z prostredia) pridávať agentovi aj interná motivácia. V literatúre sa niekedy ešte ďalej rozlišuje medzi anglickými výrazmi *intrinsic motivation* a *internal motivation* a taktiež *extrinsic motivation* a *external motivation* (Oudeyer and Kaplan, 2009), ale my filozofiu motivácie až do takej hĺbky riešiť nebudeme, a tieto pojmy budeme považovať za synonymá.

V učení posilňovaním sa teda pomocou pridania internej motivácie snažíme agenta doviesť k tomu, aby prehľadával zaujímavé alebo ešte nepoznané oblasti stavového priestoru bez ohľadu na externú odmenu (odmeňovaciu funkciu prostredia). Správnou implementáciou internej motivácie dokážeme riešiť vyššie spomenutú dilemu medzi exploráciou a exploatáciou, keďže sa agent už nebude rozhodovať iba na základe externej odmeny a prípadne nejakej náhodnosti, ale aj na základe toho, ako zaujímavo na neho daný stav pôsobí. Ďalej to dokáže riešiť aj problém riedkej odmeny, keďže agentovi vieme vygenerovať nejakú odmenu na základe jeho internej motivácie aj v časových krokoch, kedy je odmena z prostredia nulová. Oudeyer and Kaplan (2009) rozdeľujú spôsoby implementácie internej motivácie do troch nasledovných kategórií:

- **metódy založené na kompetencii** (competence-based models) spočívajú v tom, že si agent stanoví nejaký vlastný cieľ, ktorý chce dosiahnuť (tento cieľ nemusí nijako súvisieť s celkovou úlohou, ktorú rieši) a na základe toho, ako sa mu daný cieľ darí plniť, agent dostáva internú odmenu
- **morfologické metódy** (morphological models), ktoré počítajú internú odmenu napríklad na základe korelácie nasledujúcich stavov prostredia

- **metódy založené na znalostiach** (knowledge-based models) a tie popíšeme podrobnejšie v nasledujúcej podkapitole.

1.5.1 Metódy založené na znalostiach

Táto skupina generovania internej motivácie je zameraná na agentovú znalosť o jeho prostredí. Odmena sa počíta na základe podobnosti medzi stavom prostredia, ktorý agent práve zažíva a medzi jeho očakávaním, teda ako by mal stav vyzerať podľa agentových doterajších skúsenosti. Vedľajším, ale pozitívnym efektom takéhoto prístupu častokrát býva, že si agent vytvorí presný model sveta a teda vie ako jeho akcie pôsobia na prostredie. Ďalej sa táto skupina podľa spôsobu reprezentácie týchto znalostí o svete rozdeľuje na dve podskupiny: distribučná informácia (information distributional) a prediktívne metódy.

Distribučná informácia

Tento prípad je špecifický tým, že agent sa učí rozdelenie pravdepodobnosti výskytu nejakej *udalosti* v danom čase. Za takúto udalosť môžeme považovať napríklad pravdepodobnosť, že ak sme v časte t zaznamenali stav prostredia s_t, tak v čase $t+1$ sa dostaneme do stavu $s_{t+1} : P(s_t, s_{t+1})$. Z vypočítaného rozdelenia pravdepodobnosti vieme už ľahko docieliť, aby bol agent priťahovaný k novým stavom napríklad tak, že okrem odmeny z prostredia bude agent dostávať aj ďalšiu odmenu r_t^{int} , a tá bude nepriamo úmerná k pravdepodobnosti $P(s_t, s_{t+1})$

$$r_t^{int} = C \cdot (1 - P(s_t, s_{t+1}))$$

kde C je nami zvolená konštanta a teda celková odmena, ktorú agent dostáva v každom časovom kroku bude vyzerať nasledovne:

$$r_t = r_t^{int} + r_t^{ext} \tag{1.2}$$

V rovnici 1.2 člen r_t^{int} reprezentuje odmenu založenú na internej motivácii a r_t^{ext} odmenu založenú na externej motivácii. Za odmenu z externej motivácie sa považuje odmena generovaná prostredím, ktorú sme v časti 1.2 označovali $\mathcal{R}$.

Prediktívne metódy

V skutočnosti sú agentove znalosti o svete málokedy reprezentované pomocou kompletného rozdelenia pravdepodobností a skôr sa využívajú neurónové siete alebo podporné vektorové stroje na predikciu nasledujúceho stavu, alebo nejakej vlastnosti nasledujúceho stavu (Oudeyer and Kaplan, 2009). Vstupom do predikčných modelov býva väčšinou aktuálny stav prostredia a niekedy aj *kontext*, čo je určitá informácia z predchádzajúcich stavov. Všeobecne môžeme fungovanie nejakého predikčného systému Π zapisať nasledovne:

$$\Pi(s_t, s_{\to t}) = \hat{s}_{t+1}$$

kde s_t je aktuálny stav, $s_{\to t}$ je kontext z minulých stavov a $\hat{s}_{t+1}$ označuje predpovedaný stav ktorý by mal nasledovať. Chyba predikcie e_t sa počíta ako vzdialenosť medzi predpovedaným stavom $\hat{s}_{t+1}$ a skutočným stavom prostredia v nasledujúcom časovom kroku s_{t+1} pomocou niektorej zo vzdialenostných metrík.

$$e_t = dist(\hat{s}_{t+1}, s_{t+1})$$

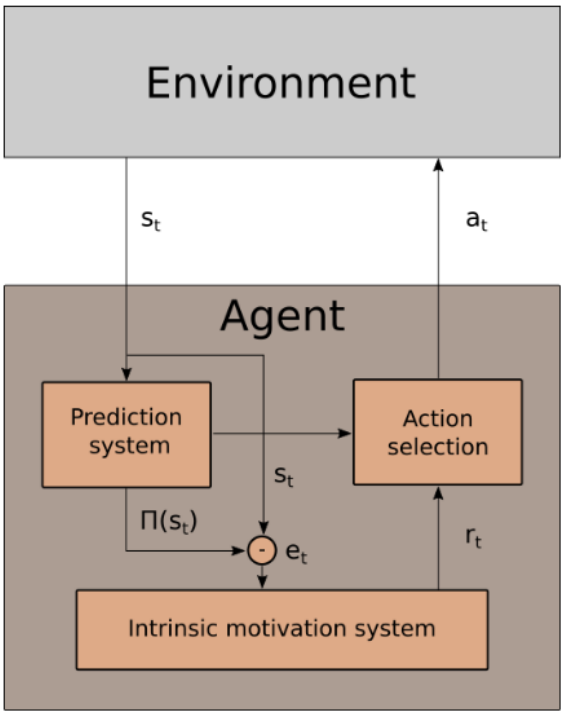
Pomocou tejto predikčnej chyby vieme agentovi modelovať internú motiváciu napríklad priamou úmerou, teda čím väčšia chyba pre dvojicu stavov $\hat{s}_{t+1}$ a s_{t+1}, tým viac je agent interne motivovaný dostať sa do stavu s_{t+1}. Celková odmena, ktorú agent dostáva sa počíta pomocou tej istej rovnice 1.2 ako aj v predošlej kategórií, ale člen r_t^{int} sa počíta nasledovne:

$$r_t^{int} = C \cdot e_t$$

kde C je konštanta a e_t predikčná chyba. Vizuálne znázornenie štruktúry prediktívnych modelov sa nachádza na obrázku 1.9.

Predikčná chyba môže vo všeobecnosti vznikať z nasledujúcich štyroch dôvodov (Burda et al., 2019):

1. Nedostatok trénovacích dát. Predikčná chyba je vysoká, lebo prediktor ešte nemal dostatočný počet podobných vstupov na to, aby sa naučil správne



Obr. 1.9: Architektúra prediktívnych metód generovania internej motivácie. Obrázok a notácia prevzaté z Pecháč (2019).

generalizovať.

2. Stochastickosť udalosti. Predikčná chyba vzniká vtedy, ak sa udalosť, ktorú chceme predikovať správa náhodne.

3. Ak je architektúra predikčného modelu nedostatočne komplexná a tým pádom nie je schopná naučiť sa udalosť správne predikovať, alebo model nemá k dispozícii všetky potrebné informácie na úspešné predikovanie udalosti.

4. Predikčná chyba je vysoká v prípade, ak sa optimalizačnému procesu nepodarí nájsť taký prediktor z vybranej triedy, ktorý by dostatočne dobre predpovedal udalosť.

Pri prediktívnych metódach je žiadúci ale iba prvý zdroj predikčnej chyby, lebo ostatné spôsobujú problém *zašumenej TV* (noisy-TV problem). Tento problém

hovorí o situácií, kedy sa agent zasekne kvôli vlastnej zvedavosti. Agent pozoruje nejakú náhodnu udalosť, ktorú nedokáže predpovedať a preto je vysoko interne motivovaný v danom stave ostať. Problém zašumenej TV býva častokrát ilustrovaný na prostredí, v ktorom agent skúma bludisko, ale akonáhle natrafí na televíziu prehrávajúcu náhodný šum, už sa ďalej nepohne.

Kapitola 2

Existujúce prístupy

2.0.1 Modul internej zvedavosti

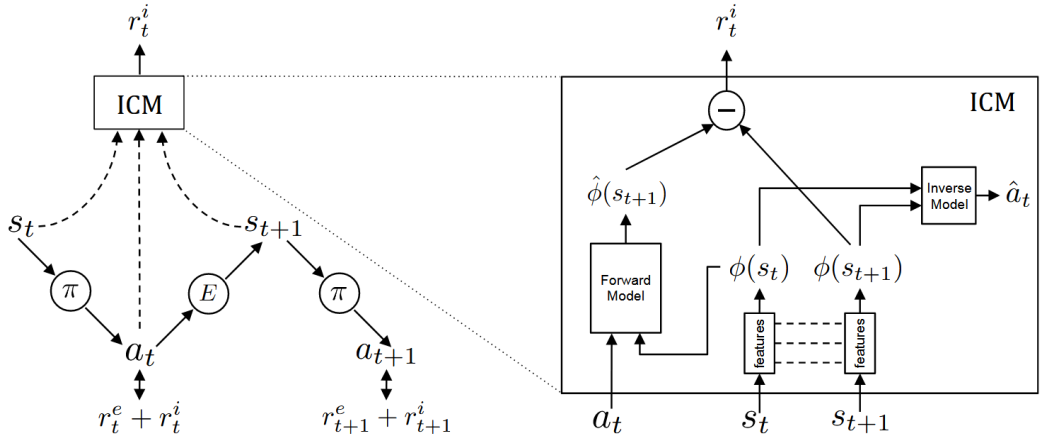
Pathak et al. (2017) predstavili prediktívnu metódu generovania internej motivácie s názvom *modul internej zvedavosti* (Intrinsic Curiosity Module, ICM), ktorý kombinuje dopredný a inverzný model z časti 1.4.3.1 (učenie reprezentácie stavu). Cieľom ICM je naučiť sa reprezentáciu stavu, ktorá bude obsahovať informácie iba o takých faktoroch prostredia, ktoré buď agent dokáže ovplyvňovať, alebo ktoré ovplyvňovať nedokáže, ale naopak tieto faktory majú vplyv na agenta a ignoruje zvyšné faktory, ktoré na agenta vplyv nemajú a ani agent na ne. Na základe takejto reprezentácie je potom generovaná interná motivácia.

Inverzný model g implementovaný jednoduchou dvojvrstvovou neurónovou sieťou s parametrami θ_g predikuje akciu $\hat{a}_t$ ktorá viedla zo stavu s_t do stavu s_{t+1} a pomocou chyby medzi predikovanou akciou $\hat{a}_t$ a skutočnou akciou a_t sa učí aj enkóder ϕ, ktorý z pozorovaní o_t a o_{t+1} generuje stavy s_t a s_{t+1} (ϕ je v tomto prípade konvolúčna neurónová sieť).

$$\hat{a_t} = g(s_t, s_{t+1}; \theta_g)$$

Interná motivácia r_t^{int} sa počíta z chyby medzi predikovaným stavom $\hat{s}_{t+1}$ z dopredného modelu f a stavom s_{t+1} vypočítaným pomocou enkódera ϕ:

$$\hat{s}_{t+1} = f(a_t, s_t; \theta_f)$$



Obr. 2.1: Schéma ICM modulu (Pathak et al., 2017). Autori využívajú inú nomenklatúru, kde písmenom s označujú pozorovanie prostredia a stavy vypočítané enkóderom označujú ako $\phi(s)$

$$r_t^{int} = \frac{\eta}{2}||\hat{s}_{t+1} - s_{t+1}||_2^2$$

kde θ_f sú parametre doprednej neurónovej siete f a η je škálovací faktor. Obrázok 2.1 graficky znázorňuje schému ICM.

Autori článku tvrdia, že agent s ICM nebude interne motivovaný vyhľadávať nepredvídateľné pozorovania a teda jeho exploračná stratégia bude odolná voči "rozptyľujúcim objektom", lebo funkcia ϕ sa učí reprezentáciu stavov, ktorá zahŕňa iba informácie ovplyvniteľné akciou agenta.

2.0.2 Destilácia náhodnej siete

Burda et al. (2019) prichádzajú s trochu iným, menej intuitívnym spôsobom generovania internej motivácie s názvom *Destilácia náhodnej siete* (Random network distillation, RND). RND patrí tiež medzi prediktívne metódy, ale v tomto prípade sa nepredpovedá niečo zmysluplné ako napríklad nasledujúci stav, alebo predchádzajúca akcia, ale predikovaný problém je generovaný náhodne. Tento prístup zahŕňa dve neurónové siete:

- *cieľová* (target) sieť f, ktorá je na začiatku náhodne inicializovaná a ktorá

sa počas tréningu vôbec nemení. Táto sieť generuje predikčný problém

- *prediktívna* (prediction) sieť $\hat{f}$, ktorá sa učí napodobňovať výstupy cieľovej siete f.

Cieľová sieť kóduje stav prostredia na vektor vlastností $f : s \to \mathbb{R}^k$ a prediktívna sieť $\hat{f} : s \to \mathbb{R}^k$ sa pomocou metódy gradientového zostupu učí (teda mení svoje parametre θ) tak, aby bola nasledovná chyba medzi sieťami minimálna $||\hat{f}(x;\theta) - f(x)||^2$. Takýto proces, kedy sa jedna sieť učí vedomosti z inej siete je nazývaný destilácia.

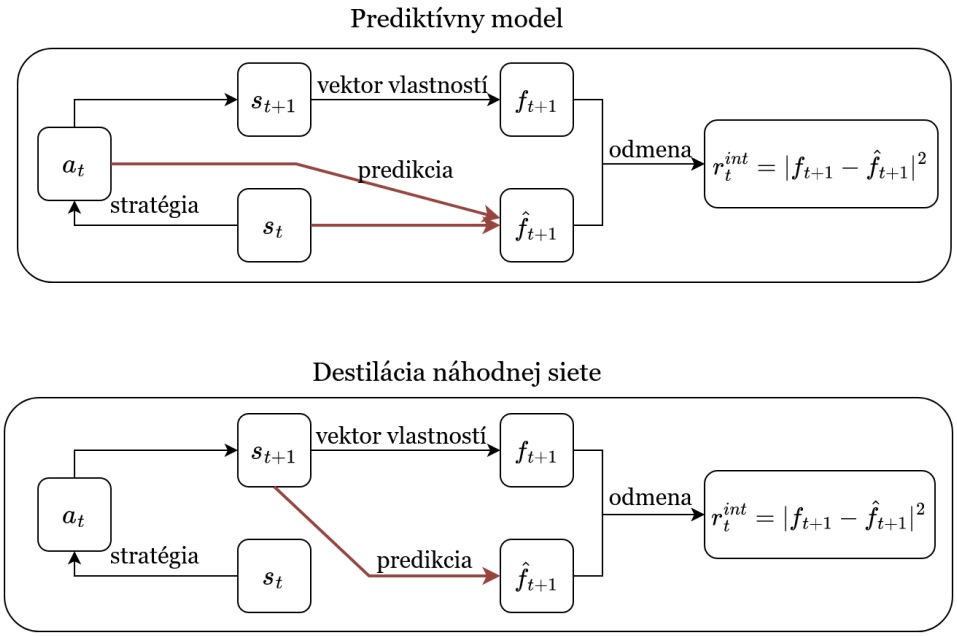
Rovnako ako pri ICM, aj teraz agent dostáva menšiu internú odmenu v stavoch, ktoré už pozná a vyššiu odmenu v neznámych stavoch, pretože v nich je predikčná chyba väčšia. Pri ICM sa predickie robili na základe dvoch po sebe nasledujúcich nasledujúcich stavov, ale pri RND sa predikuje iba na základe jedného stavu, tým pádom agent nemá tendenciu sústrediť sa na nepredvídateľnosť prechodov zo stavu s_t do s_{t+1}. Ďalšou výhodou je skutočnosť, že sa cieľová neurónová sieť nikdy nemení a teda rovnaké vstupy budú vždy produkovať rovnaké výstupy. Vďaka týmto vlastnostiam je RND odolná voči problému zašumenej TV, ale za najväčšiu výhodu sa považuje jej jednoduchá architektúra, ktorá sa ľahko implementuje a aj trénuje. Na obrázku 2.2 je porovnanie medzi architektúrou doprednej metódy založenej na predikovaní ďalšieho stavu a RND.[1]

Autori článku vypočítanú internú odmenu pred každou aktualizáciou učiaceho algoritmu normalizujú, aby bol tento prístup vhodný pre ľubovolné hyperparametre a prostredie a aby interná odmena časom úplne nevymizla. Taktiež normalizujú vstupy do oboch neurónových sietí (stavy prostredia), aby dosiahli väčší rozptyl hodnôt vektora vlastností.

2.0.3 SpatioTemporal DeepInfomax

SpatioTemporal DeepInfomax (ST-DIM) je metóda učenia reprezentácie stavu založená na maximalizovaní viacerých vzájomných informácií pomocou samoučiaceho kontrastného učenia (self-supervised contrastive learning) (Anand et al.,

[1] inšpirované blogom https://openai.com/blog/reinforcement-learning-with-prediction-based-rewards



Obr. 2.2: Porovnanie prediktívnej metódy, ktorá predpovedá stav v ďalšom časovom kroku na základe aktuálneho stavu a vybranej akcie a RND metódou.

2019). Cieľom metódy ST-DIM je, aby naučená reprezentácia obsahovala informáciu o komplexných konceptoch (napríklad nepriateľov, agenta a iné objekty) a ignorovala nepodstatné detaily z pozorovaní (napríklad farbu pozadia). Enkóder ϕ je v tomto prípade rozdelený na dve časti $\phi = G \circ L$, kde z časti L dostávame lokálne príznaky a až aplikovaním G dospejeme ku globálnym príznakom, teda k finálnej reprezentácii.

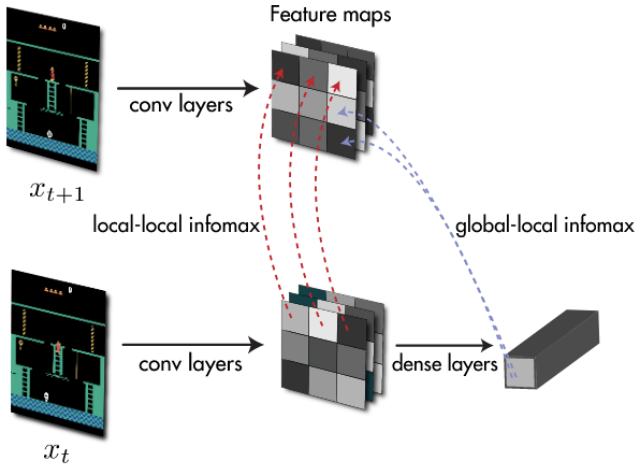
Učenie enkódera ϕ spočíva jednak v maximalizovaní *globálnej* vzájomnej informácie, ktorá sa počíta na základe podobnosti medzi globálnymi príznakmi v čase t a úsekmi lokálnych príznakov v čase $t+1$ a súčasne na základe rozdielov týchto príznakov v časoch t a $t+2, t+3, ..., N$. Ďalej sa maximalizuje *lokálna* MI podobne ako globálna, ale teraz sa sleduje podobnosť medzi korešpondujúcimi úsekmi lokálnych príznakov v čase t a $t+1$ a rozdielnosť medzi inými časovými hodnotami.

Učenie je založené na minimalizovaní nasledujúcich dvoch chýb:

$$\mathcal{L}_G = -\sum_{m=1}^{M}\sum_{n=1}^{N}\log\frac{\exp(g_{m,n}(x_t, x_{t+1}))}{\sum_{x_{t^*}\in X_{next}}\exp(g_{m,n}(x_t, x_{t^*}))}$$

$$\mathcal{L}_L = -\sum_{m=1}^{M}\sum_{n=1}^{N}\log\frac{\exp(f_{m,n}(x_t, x_{t+1}))}{\sum_{x_{t^*}\in X_{next}}\exp(g_{m,n}(x_t, x_{t^*}))}$$

,kde $\mathcal{L}_G$ označuje globálnu a $\mathcal{L}_G$ lokálnu vzájomnú informáciu, f a g sú *skórovacie* lineárne funkcie, ktoré reprezentujú spomenuté podobnosti. Funkcie f a g sa učia pomocou chyby InfoNCE a sú implementované jednovrstvovými neurónovými sieťami. Výpočet je ilustrovaný na obrázku 2.3.



Obr. 2.3: Znázornenie, výpočtu globálnej (global-local infomax) a lokálnej (local-local infomax) vzájomnej informácie v metóde ST-DIM. Prevzaté z Anand et al. (2019).

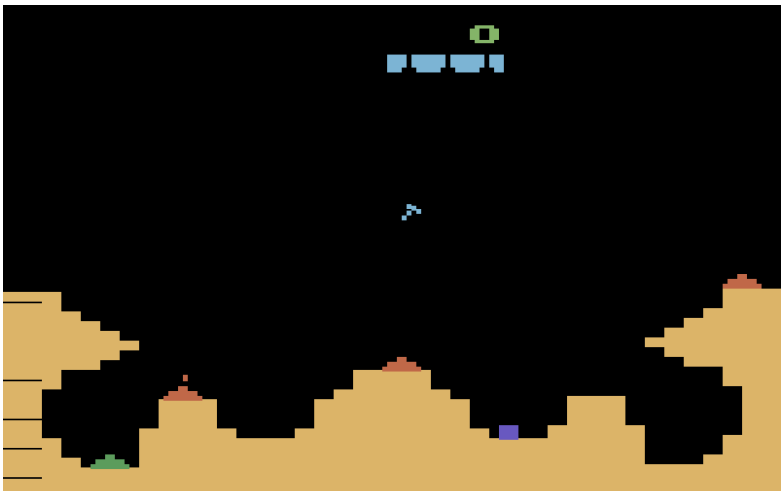
Kapitola 3

Experimenty

Na základe teoretických vedomostí spísaných v predchádzajúcich kapitolách teraz nadizajnujeme experimenty, v ktorých učíme agenta pomocou učenia posilňovaním a internej motivácie hrať rôzne Atari hry. Podrobnejšie popíšeme tieto prostredia, implementáciu učiaceho algoritmu PPO a taktiež tri spôsoby, akými generujeme internú motiváciu. V ďalšej kapitole tieto prístupy generovania internej motivácie medzi sebou porovnáme.

3.1 Prostredie

Atari hry (Atari games) je skupina asi šesťdesiatich videohier, ktoré boli pôvodne vytvorené pre konzolu s názvom Atari 2600 z roku 1977. V súčasnosti sa viaceré z týchto hier používajú na meranie úspešnosti algoritmov učenia posilňovaním, pretože majú relatívne nízke rozlíšenie, takže dimenzia stavového priestoru nie je príliš vysoká a zároveň obsahujú veľmi riedku odmenu. Klasické algoritmy RL sú v nich častokrát neúspešné, lebo agentovi nenájde ani jednu pozitívnu odmenu (Burda et al., 2019). Medzi tie náročnejšie patria hry ako Freeway, Gravitar, Montezuma's Revenge, Pitfall! a Venture. Keďže učíme agenta pomocou internej motivácie, čo ma za dôsledok aj lepšiu exploráciu, rozhodli sme sa použiť tieto ťažšie hry. Najskôr sme experimenty testovali na hre Gravitar a neskôr, keď sa ukázalo, že niektoré z nich majú potenciál, spustili sme tréning aj na hrách Montezuma's Revenge a Venture. Všetky tieto prostredia sú dostupné v Python-ovskej knižnici



Obr. 3.1: Konkrétny level hry Gravitar. Modrý objekt v strede je loď ovládaná agentom, dole je terén planéty, na ktorom stoja nepriateľské bunkre (farebné polkruhy) a fialová kocka je palivo, ktoré si agent môže vyzdvihnúť.

gym[1] a hry sa dajú hrať aj online na stránke https://www.retrogames.cz.

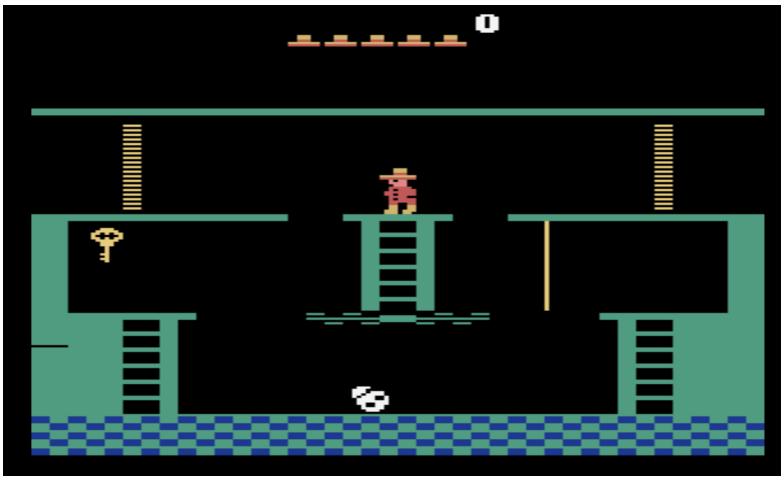
Gravitar

V tejto hre agent ovláda malú ozbrojenú kozmickú loď v ohraničenom 2D vesmíre, kde sa stretáva s rôznymi výzvami. Napríklad v jednom leveli sa priblíži k planéte, kde má za úlohu zničiť všetky nepriateľské bunkre, ktoré paľbu opätujú. Okrem vyhýbaní nepriateľským strelám si agent musí dávať pozor aj na terén planéty, ku ktorej je priťahovaný gravitáciou. Ak ho nepriateľ zasiahne, alebo agent do niečoho narazí, stráca jeden život (na začiatku ich má päť) a po strate posledného života hra končí. Hra tiež končí, ak sa agentovi minie palivo. Agent má na výber z piatich akcií: otočiť loď vľavo, otočiť loď vpravo, aktivovať motor a tým rozbehnúť loď vpred, ďalej môže vystreliť a poslednou akciou ovláda štít alebo magnet v závislosti od levelu. Za každého zničeného nepriateľa získava odmenu. Toto prostredie môžeme vidieť na obrázku 3.1.

Montezuma's Revenge

Montezuma's Revenge (obr 3.2) pozostáva z deviatich úrovní pričom každá obsahuje niekoľko miestností. Agent ovláda panáčika menom Panama Joe, ktorý

[1]https://github.com/openai/gym

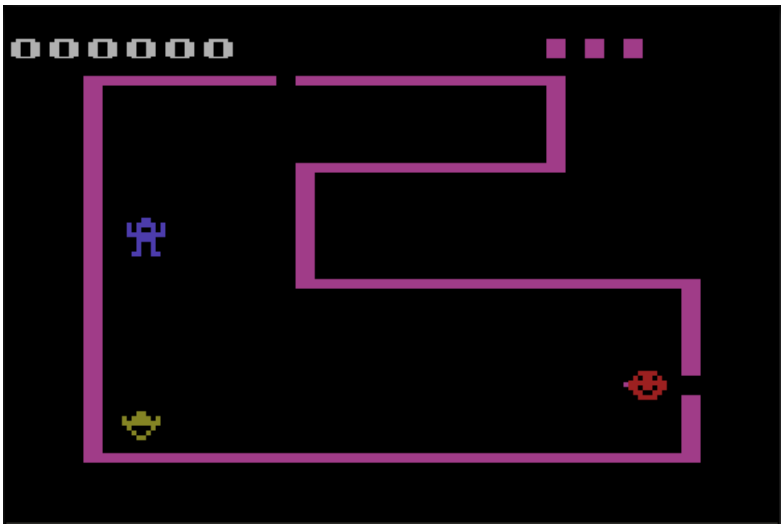


Obr. 3.2: Prvá miestnosť prvej úrovne v Atari hre s názvom Montezuma's Revenge. Postavička v strede je ovládaná agentom, vľavo je vidno kľúč potrebný na to, aby sa agent z miestnosti dostal a biela lebka dole je nepriateľ, ktorému sa treba vyhnúť.

sa objavuje v prvej miestnosti bludiska a jeho úlohou je dostať sa von, pričom dostáva odmenu za zbieranie pokladov a elimináciu nepriateľov. Miestnosti sú plné rôznych pascí, ktoré môžu pripraviť agenta o život. Okrem toho, na otvorenie ďalších miestností je niekedy potrebný kľúč, inokedy je potrebná fakľa, aby agent vôbec videl, v iných prípadoch potrebuje amulet atď. Agent ma k dispozícií znova päť rôznych akcií: pohyb vpravo a vľavo, hore a dole ale iba v prípade, že je blízko rebríka a skok. Táto hra je asi najznámejšia spomedzi všetkých Atari hier v učení posilňovaním, lebo aj v prípade optimálnej stratégie trvá niekoľko stoviek časových krokov, kým agent získa prvú odmenu.

Venture

Posledné prostredie, ktoré sme v experimentoch použili je hra Venture, kde agent ovláda panáčika v bludisku plnom príšer s cieľom nazbierať čo najviac pokladov bez toho, aby ho príšery chytili. Pohľad na bludisko je zhora a tým pádom sa agent môže hýbať vo všetkých štyroch smeroch (hore, dole, vľavo a vpravo) a ako poslednú akciu, z ktorých má na výber je streľba z luku. V bludisku sú dva typy príšer, jeden typ dokáže agent zneškodniť, ale nie je za to nijako ohodnotený. Čím skôr stihne agent pozbierať všetky poklady, tým väčšiu bonusovú odmenu



Obr. 3.3: Atari hra Venture. Červený panáčik vpravo dole je ovládaný agentom, žltý objekt vlavo je poklad, ktorý ma agent za cieľ zobrať a modrá je príšera, ktorá naháňa agenta.

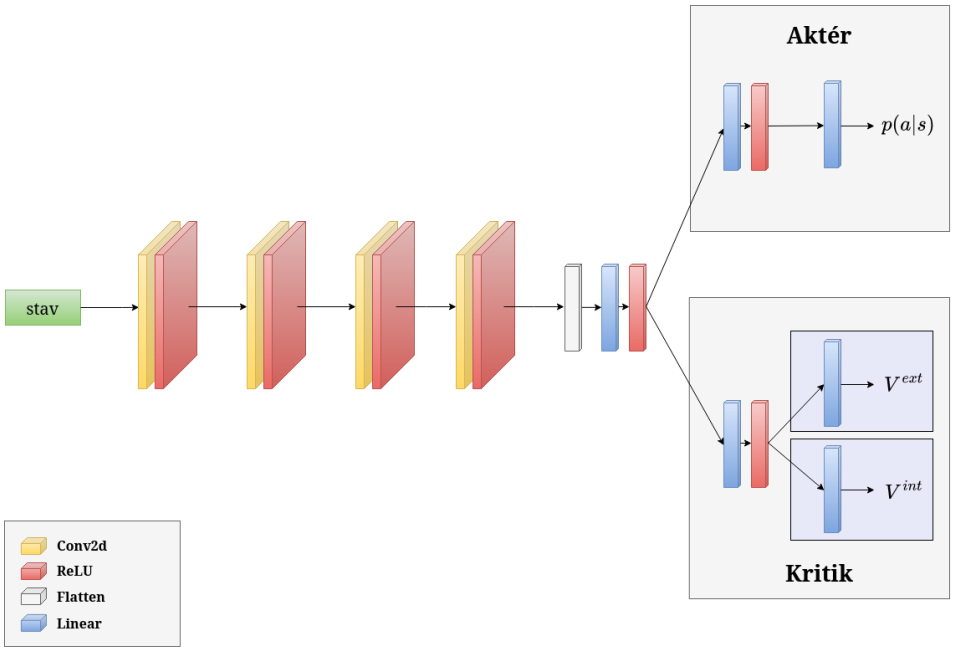
dostane na konci levelu. Na obrázku 3.3 vidíme ukážku tohto prostredia.

3.2 Algoritmus učenia

Agenti sa v experimentoch učia svoje stratégie pomocou algoritmu PPO, ktorého pseudokód je spomenutý v kapitole 1.3.6, ale teraz si podrobnejšie popíšeme našu/Matejovu implementáciu tohto algoritmu, ktorá je dostupná na githube[2].

Ako sme spomenuli, PPO patrí medzi aktér-kritik algoritmy. My aktéra a kritika implementujeme pomocou jednej neurónovej siete, ktorú spolu zdieľajú (backbone) a z nej potom ďalej pokračujú do vlastných sietí, ktoré sa v takomto prípade nazývajú hlavy (heads). Túto architektúru je vidno na obrázku 3.4. Aktér pokračuje do neurónovej siete, ktorej výstupom sú pravdepodobnosti výberu jednotlivých akcií, ale kritik sa znova delí na ďalšie dve hlavy, z ktorých jedna reprezentuje hodnotovú funkciu V^{ext} externej odmeny (z prostredia) a druhou hlavou sa predpovedá hodnotová funkcia internej odmeny V^{int} (generovanou internou motiváciou). Výstupom z oboch hláv kritika je teda jedno číslo.

[2]https://github.com/Iskandor/MotivationModels

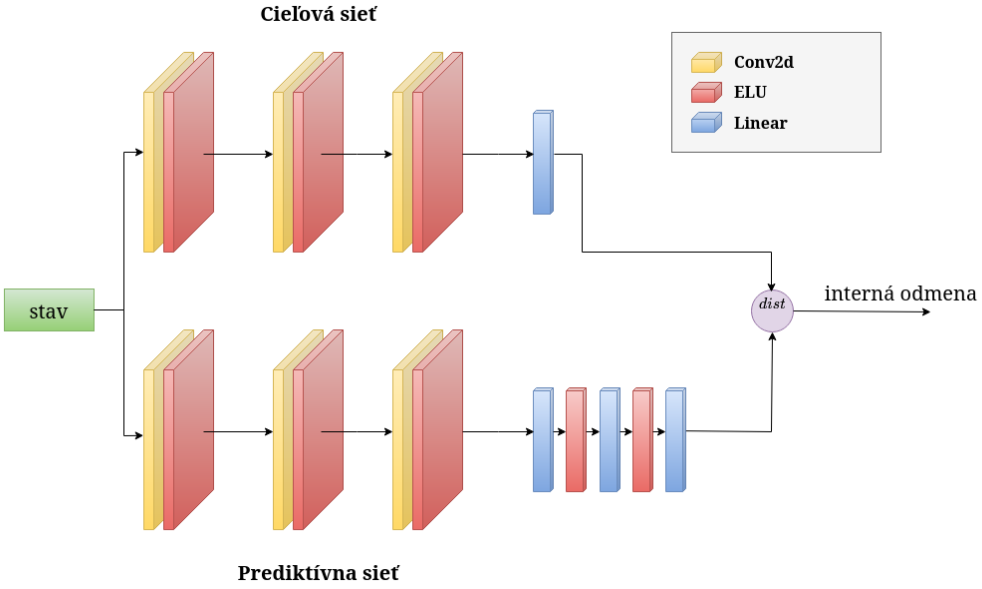


Obr. 3.4: Implementácia PPO pomocou neurónovej siete. Aktér a kritik spolu zdieľajú jednu neurónovú sieť, z ktorej sa na konci rozdeľujú a pokračujú do vlastných hláv (sivé obdĺžniky). Hlava kritika sa ďalej delí na dve hlavy (znázornené modrými obdĺžnikmi) z ktorých sa počíta hodnotová funkcia internej odmeny a externej odmeny.

Neurónové siete sa učia optimalizátorom Adam s rýchlosťou učenia $lr = 0.0001$ (zvyšné parametre optimalizátora sú štandardné), tak aby minimalizovali súčet váhovaných chýb jednotlivých komponentov. Vo všetkých experimentoch je váha chyby aktéra nastavená na hodnotu $w^A = 1$ a chybu kritika násobíme váhou $w^K = 0.5$.

3.3 Generovanie internej motivácie

Naše experimenty delíme do troch skupín a to podľa spôsobu generovania internej motivácie. Vo všetkých prípadoch vypočítanú internú motiváciu ohraničujeme zhora číslom 1 $r^{int} = \max(1, r^{int})$ ešte pred tým ako ju agent dostane. Na nasledujúcich obrázkoch sa dá všimnúť, že architektúry použitých neurónových



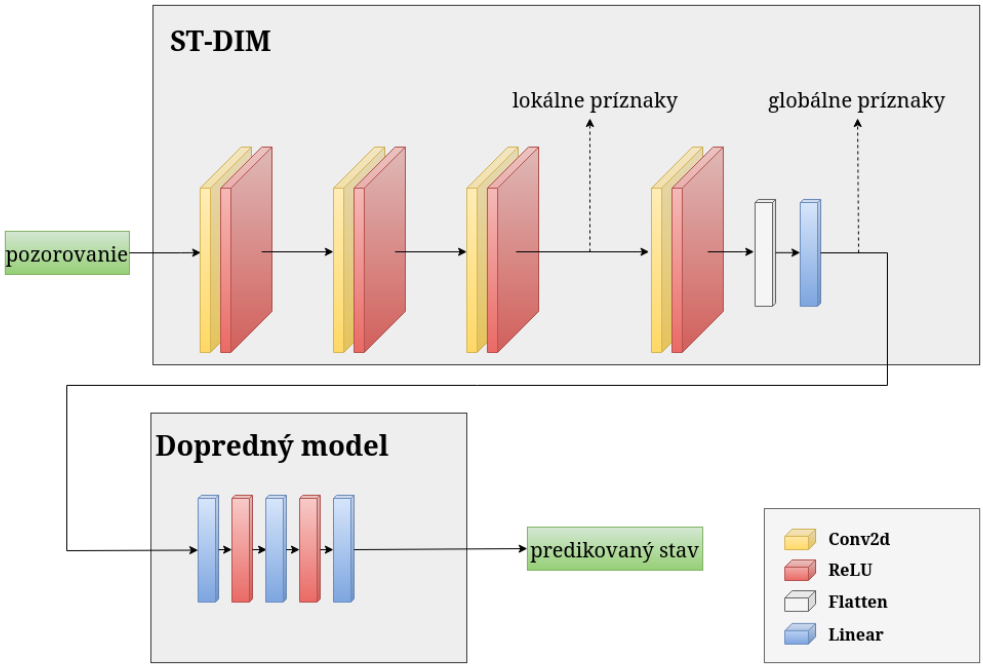
Obr. 3.5: Architektúra cieľovej a prediktívnej siete v naších RND experimentoch. Prediktívna sieť je zložená z viacerých vrstiev kvôli tomu, aby sa ľahšie naučila imitovať cieľovú sieť.

sietí sú podobné, ale jednotlivé prístupy sa veľmi líšia v optimalizačnej úlohe.

3.3.1 Destilácia náhodnej siete

Prvou skupinou sú experimenty, kde internú motiváciu generujeme pomocou destilácie náhodnej siete. Architektúru cieľovej a prediktívnej siete môžme vidieť na obrázku 3.5. Táto architektúra je vždy rovnaká, ale experimenty tejto skupiny sú rozdielne v spôsobe výpočtu chyby $dist$ medzi výstupmi sietí.

V Burda et al. (2019) počítajú autori chybu pomocou vzdialenostnej metriky L2, ale my na základe časti 1.4.3.2, kde tvrdíme že vo vysokorozmernom priestore sa metrika L1 a zlomková metrika správa lepšie, skúšame aj tieto metriky. Výstupy z cieľovej a prediktívnej siete sú 512-dimenzionálne a to už považujeme za vysokú dimenziu. Zlomkovú metriku skúšame s rôznymi hodnotami f a okrem toho, že inak počítame internú motiváciu, skúšame pomocou týchto metrík aj trénovať prediktívnu sieť (v pôvodnom článku bola sieť trénovaná znova po-



Obr. 3.6: Kombinácia ST-DIM a dopredného modelu. V ST-DIM enkóderi tiež vidno, ako sa počítajú lokálne a globálne príznaky.

mocou L2 metriky). Na trénovanie prediktívnej siete využívame optimalizátor Adam s rýchlosťou učenia $lr = 0.0001$. Ešte pred tým ako agent dostane internú odmenu, prenásobíme ju váhou w^{int}, ktorá sa líši medzi experimentami.

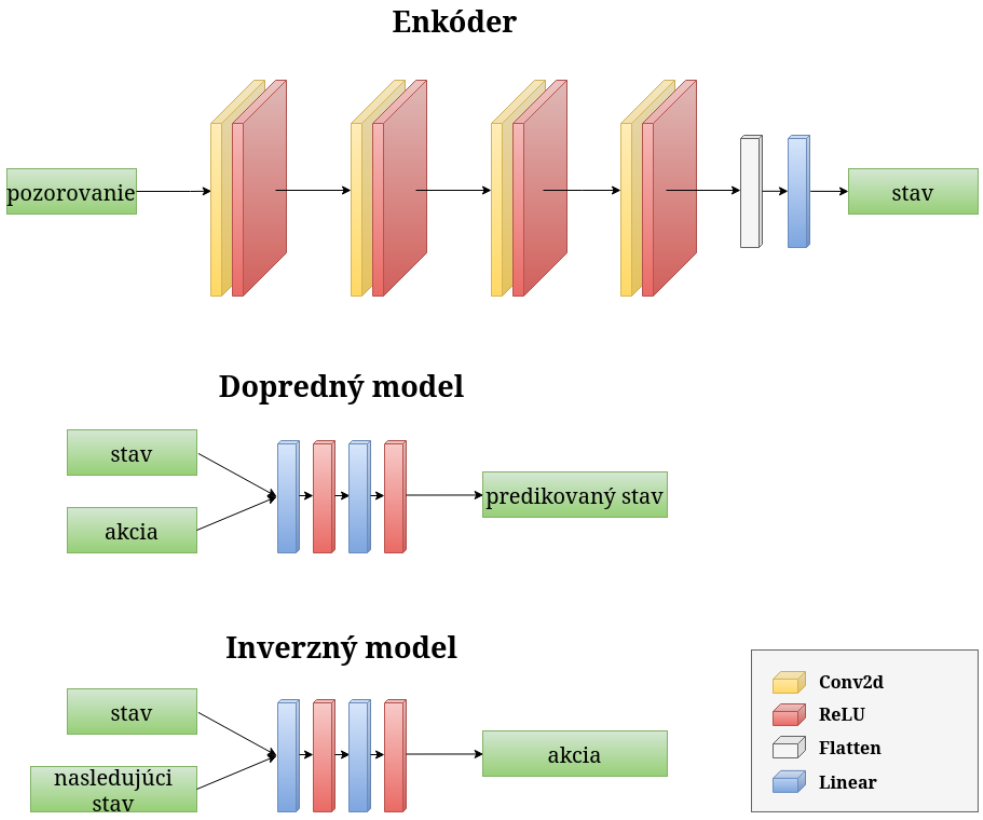
3.3.2 ST-DIM a dopredný model

Druhý spôsob, ako generujeme internú motiváciu je kombináciou metódy SpatioTemporal DeepInfomax a dopredného modelu. Pomocou ST-DIM enkódera generujeme stavovú reprezentáciu, ktorá potom spolu so zvolenou akciou vstupuje do dopredného modelu a ten predikuje ako bude vyzerať stavová reprezentácia v ďalšom časovom kroku. Na základe strednej kvadratickej chyby (mean squared error - MSE) medzi predikovanou a reálnou reprezentáciou generujeme internú odmenu. Obidva komponenty sú implementované pomocou neurónových sietí a ich architektúry sú zobrazené na obrázku 3.6. Tréning ST-DIM enkódera realizujeme dvoma spôsobmi. Jedným z nich je pomocou kontrastívneho učenia tak,

ako bolo spomenuté v článku Anand et al. (2019) a v druhom prípade sa enkóder ešte navyše učí aj pomocou chyby z dopredného modela. Dopredný model je trénovaný štandardne prostredníctvom MSE. Znova využívame optimalizátor Adam s rýchlosťou učenia $lr = 0.0001$

3.3.3 Modul internej zvedavosti

Taktiež sme vyskúšali internú motiváciu generovať pomocou modulu ICM, v ktorom sme enkóder, dopredný model a inverzný model implementovali neurónovými sietami zobrazenými na obrázku 3.7. Na výpočet všetkých chýb používame MSE. Pri optimalizovaní používame Adam s rovnakou rýchlosťou učenia ako v predošlích experimentoch, ale chybu z dopredného modela najskôr násobime škálovacím faktorom $\beta = 0.2$ a chybu z inverzného modela násobime hodnotou $1 - \beta$.



Obr. 3.7: Naša architektúra enkódera, dopredného modelu a inverzného modelu, spojením ktorých implementujeme modul internej zvedavosti. Schéma zobrazujúca zapojenie jednotlivých komponentov je zobrazená na obrázku 2.1.

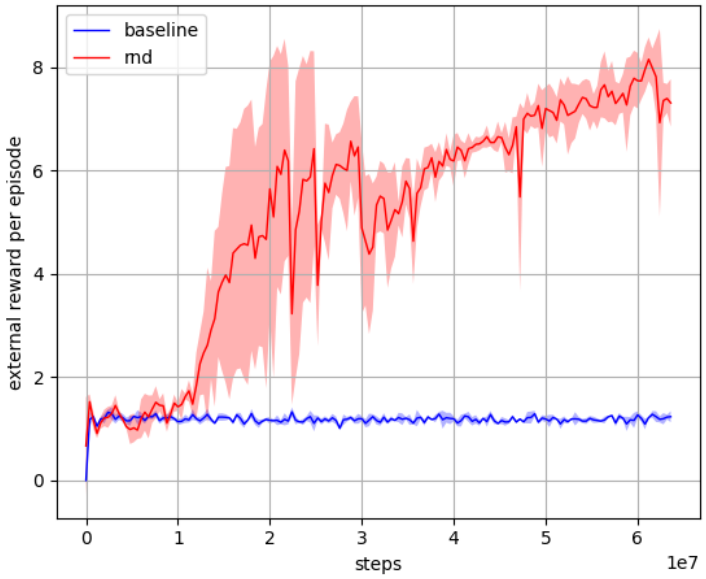
Kapitola 4

Výsledky

V tejto kapitole porovnáme dosiahnuté výsledky spomenutých stratégií generovania internej motivácie z hľadiska priemernej externej odmeny, ktorú agent dosiahol v zmienených Atari hrách. Agenti sa v našich experimentoch v prostrediach Montezuma's Revenge a Venture učia na základe skúseností zo 128 paralelných simulácií po dobu 128 miliónov časových krokov. V prostredí Gravitar, ktoré používame vo väčšine experimentov, sme kvôli výpočtovým a pamäťovým limitáciam znížili dobu učenia na 64 miliónov krokov a paralelne beží 32 simulácií. Všetky experimenty sme realizovali na univerzitnom serveri, ktorý využívali aj iní študenti a už takýto skrátený tréning v prostredí Gravitar trval približne 42 hodín. Aby boli výsledky štatisticky validnejšie, výsledne hodnoty počítame ako aritmetický priemer troch behov jednotlivých experimentov.

4.1 Prostredie Gravitar

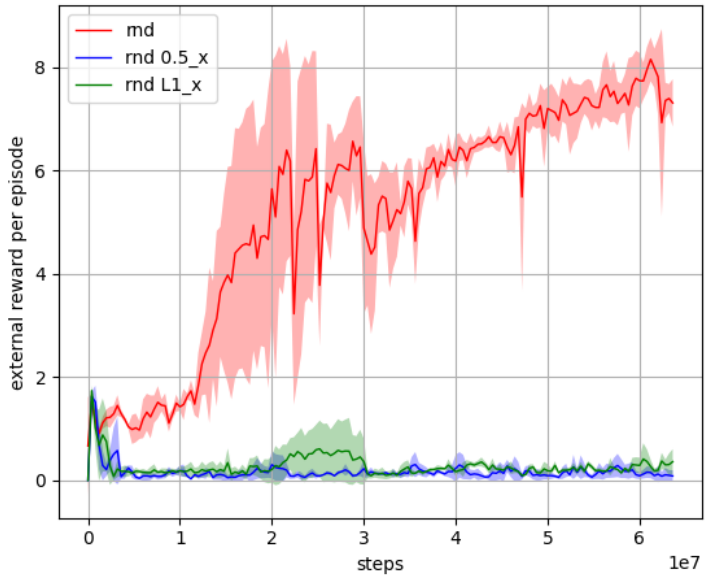
Našim prvým cieľom bolo zreplikovať dosiahnuté výsledky v Burda et al. (2019) v prostredí Gravitar, kedy sa agent učí pomocou internej odmeny generovanej destiláciou náhodnej siete a tento prístup porovnať s agentom, ktorý sa učí svoju stratégiu iba na základe externej odmeny. Na obrázku 4.1 je vidieť, že agent bez internej odmeny (model s názvom *baseline*) sa síce na začiatku naučí niečo zmysluplné, ale vyzerá to tak, že až po zvyšok tréningu tieto vedomosti exploatuje a preto ďalej dosahuje iba rovnakú externú odmenu. Druhý agent s internou od-



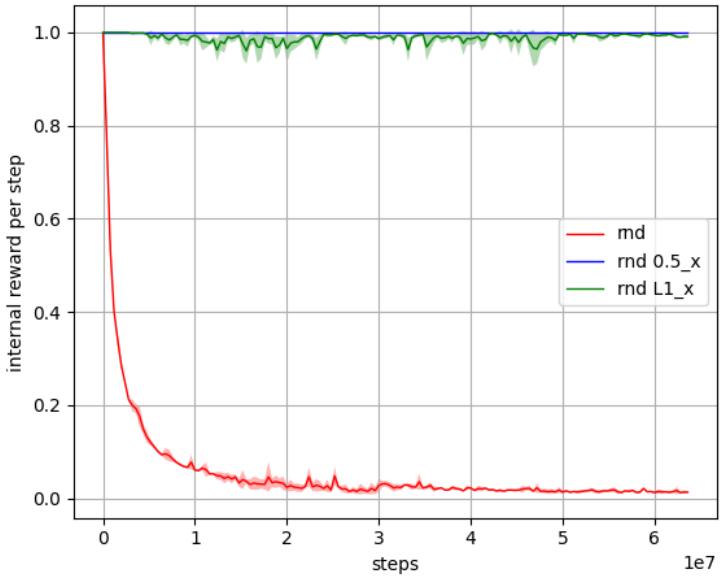
Obr. 4.1: Porovnanie modelu RND s modelom bez internej motivácie z hľadiska externej odmeny. Svetlá farba znázornuje rozptyl a tmavšia priemernú externú odmenu. Dĺžka jednej epizódy je 4096 časových krokov.

menou z destilácie náhodnej siete (*rnd*) začne po prvom milióne krokov baseline agenta dominovať a ak by tréning trval dlhšie, pravdepodobne by dosiahol ešte vyššie hodnoty externej odmeny.

Následne sme chybu medzi cieľovou a prediktívnou sieťou v RND počítali pomocou Manhattanskej vzdialenosti (experiment *rnd L1_x*) a pomocou zlomkovej vzdialenosti s parametrom $f = 0.5$ (experiment *rnd 0.5_x*). Dosiahnuté externé odmeny týchto dvoch experimentov aj s pôvodným *rnd*, ktorý využíva Euklidovskú vzdialenosť, sú porovnané na obrázku 4.2. Vidno, že v experimentoch *rnd L1_x* a *rnd 0.5_x* niečo nie je v poriadku, keďže agenti dosahujú skoro celú dobu externú odmenu blízku nule. Zistili sme, že agenti boli veľmi vysoko interne motivovaní v každom stave, ktorý navštívili (zobrazené na obrázku 4.3), čo malo za dôsledok, že agenti svoje vedomosti o externej odmene vôbec neexploatovali a správali sa viac-menej náhodne. Tento problem sme vyriešili prenásobením internej odmeny konštantou $w^{int} = 10^{-3}$ v prípade L1 normy (*rnd L1*)

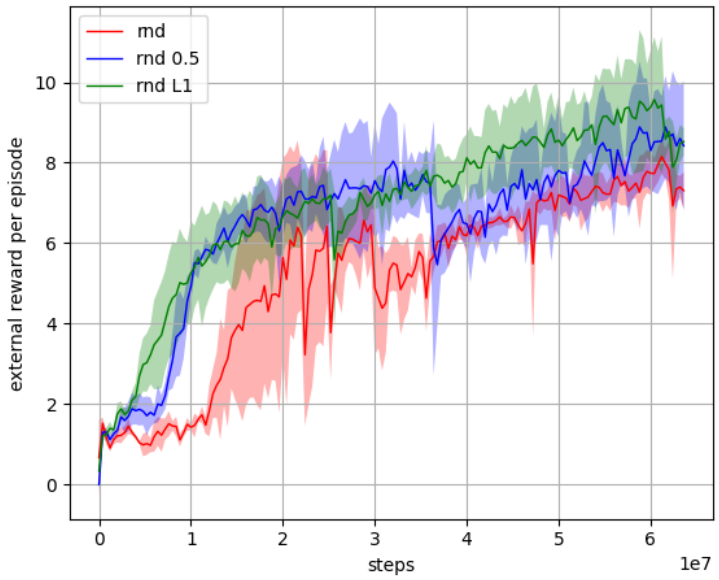


Obr. 4.2: Porovnanie modelov rnd, rnd L1_x a rnd 0.5_x z hľadiska externej odmeny.



Obr. 4.3: Interná odmena v spomenutých troch experimentoch. Ak by sme odmenu z internej motivácie neohraničovali číslom 1, interná odmena by pravdepodobne dosahovala ešte vyššie hodnoty.

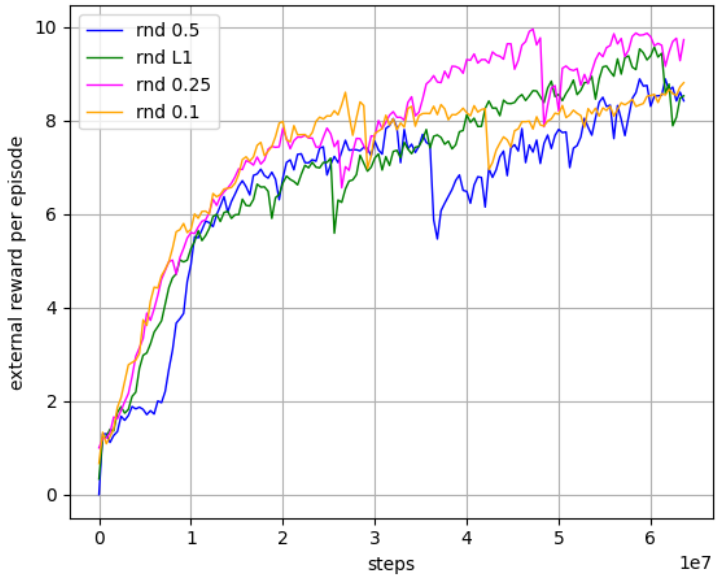
a konštantou $w^{int} = 10^{-5}$ v prípade zlomkovej vzdialenosti (*rnd 0.5*). Výsledky nových experimentov sú zobrazené na obrázku 4.4. Vidno, že oba tieto experimenty dosahujú skoro celý čas vyššiu priemernú externú odmenu ako pôvodný *rnd* a taktiež sa učia rýchlejšie.



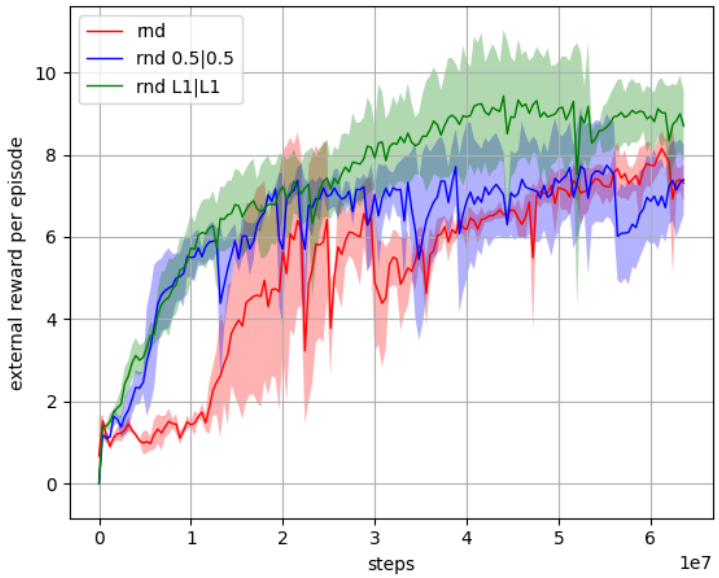
Obr. 4.4: Externá odmena RND experimentov s výpočtom chyby pomocou Euklidovskej, Manhattanskej a zlomkovej vzdialenosti.

Následne sme vyskúšali zlomkovú vzdialenosť s nižšími hodnotami $f = 0.25$ (experiment s názvom *rnd 0.25* a konštantou $w^{int} = 10^{-11}$) a $f = 0.1$ (*rnd 0.1* a $w^{int} = 10^{-27}$) a všetky prístupy, v ktorých sme menili spôsob výpočtu chyby, sme porovnali na obrázku 4.5. Tento obrázok potvrdzuje tvrdenie z Aggarwal et al. (2001), že vyššia rozlišovacia schopnosť vzdialenostných metrík automaticky nezaručuje lepšie výsledky algoritmov strojového učenia.

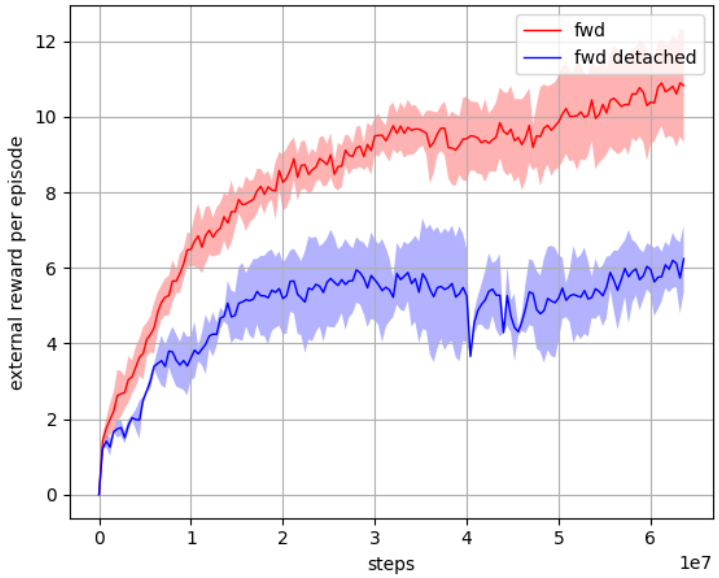
Naše posledné porovnávanie vzdialenostných metrík je na obrázku 4.6, kde v experimente s názvom *rnd L1|L1* využívame Manhattanskú vzdialenosť jednak na výpočet internej motivácie, ale aj na trénovanie prediktívnej neurónovej siete a v experimente *rnd 0.5|0.5* využívame zlomkovú vzdialenosť s parametrom $f = 0.5$ na oba účely. Na základe predchádzajúcich grafov a tabuľky 4.1, ktorá obsahuje



Obr. 4.5: Porovnanie modelov s L1 a zlomkovou vzdialenosťou. V tomto grafe vykreslujeme iba priemernú dosiahnutú externú odmenu z dôvodu lepšej viditelnosti.



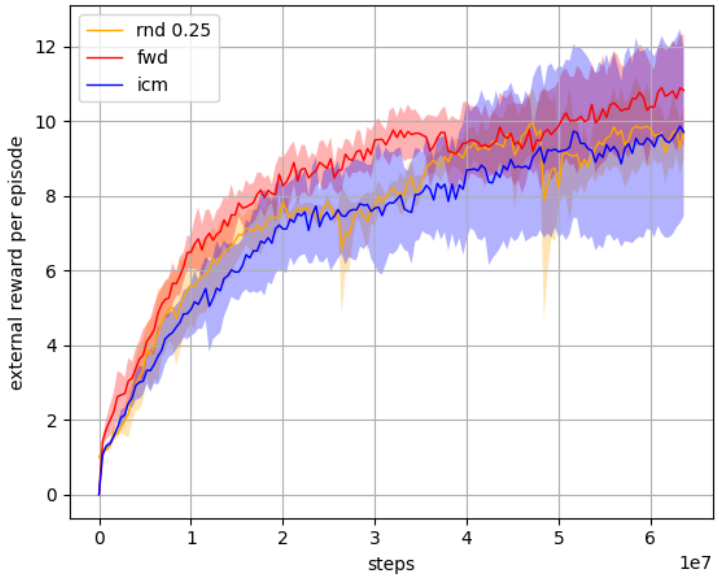
Obr. 4.6: Porovnanie modelov, ktoré učia prediktívnu neurónovu sieť pomocou rôznych vzdialenostných metrík.



Obr. 4.7: Dopredný model s ST-DIM enkóderom. V experimente *fwd* sa enkóder učí pomocou dvoch chýb.

vypočítaný priemer, smerodajnú odchýlku a 95. percentil dosiahnutej externej a internej odmeny, vidíme, že najlepšie výsledky dosiahol experiment, ktorý počítal internú motiváciu pomocou zlomkovej vzdialenosti s parametrom $f = 0.25$ a prediktívnu sieť trénoval L2 vzdialenosťou. Od originálnej implementácie *rnd* z článku Burda et al. (2019) dosahuje v priemere o 2.48 vyššiu externú odmenu, čo je zlepšenie skoro o 50%.

V prostredí Gravitar sme dalej testovali úspešnosť dopredného modelu s ST-DIM enkóderom, ktorý sa učí aj pomocou chyby z dopredného modelu (experiment *fwd*) a modelu, kedy sa ST-DIM enkóder učí iba kontrastívnym učením (*fwd detached*). Z obrázku 4.7 je zrejmé, že chyba, ktorá pretečie z dopredného modelu až do enkódera, výrazne pomáha agentovi dosahovať vyššiu externú odmenu. Nakoniec sme *fwd* porovnali s najúspešnejším experimentom využívajúcim RND architektúru a taktiež s ICM modulom na obrázku 4.8. Je zrejmé, že najlepšie výsledky v prostredí Gravitar jednoznačne dosiahol dopredný model,



Obr. 4.8: Porovnanie najlepších modelov z jednotlivých skupín generovania internej odmeny.

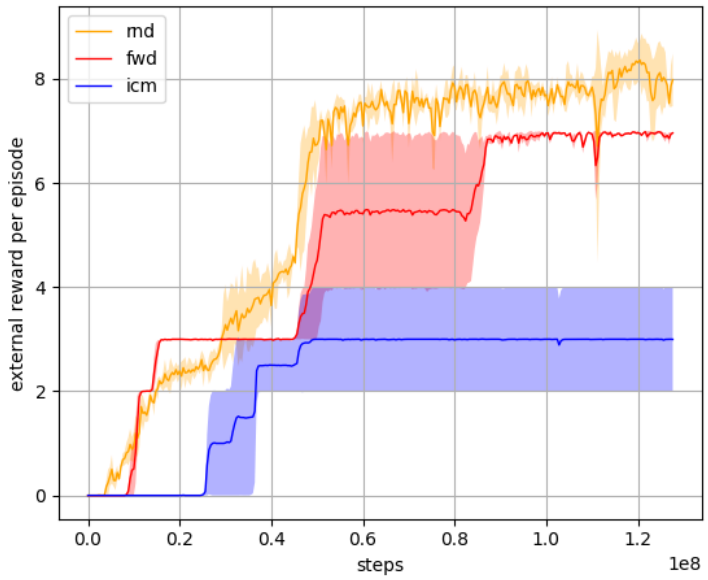
kde sa enkóder učil podľa oboch chýb.

4.2 Prostredie Montezuma's Revenge

V prostredí Montezuma's Revenge sme už neskúmali správanie vzdialenostných metrík, ale merali sme úspešnosť ICM modulu, lepšieho variantu dopredného modelu a RND s originálnymi nastaveniami. Dosiahnutá externá odmena v týchto troch experimentoch je zobrazená na grafe (Obr. 4.9) a všetky vypočítané hodnoty z externej a internej odmeny sa nachádzajú v tabuľke 4.2. Na obrázku externej odmeny je vidno, že ICM modul a dopredný model sa častokrát zaseknú na dlhšiu dobu v nejakom lokálnom minime. Toto správanie je s najväčšou pravdepodobnosťou spôsobené extrémne riedkou odmenou prostredia a nedostatočne vysokou internou motiváciou, ktorá by dostala agenta do nových stavov. Experiment *rnd* je teda lepší nie len z dôvodu najvyššej dosiahnutej externej odmeny, ale taktiež vytvára vhodnejšiu exploračnú stratégiu v tomto prostredí.

Tabuľka 4.1: Prehľad experimentov v prostredí Gravitar z hľadiska získanej externej a internej odmeny, vyjadrením pomocou štatistických ukazovateľov.

	externá odmena			interná odmena		
názov	mean	std	95%	mean	std	95%
baseline	1.188	0.002	3.0	-	-	-
rnd	5.118	0.271	9.333	0.053	0.001	0.220
rnd 0.5_x	0.163	0.036	1.0	1.0	0.0	1.0
rnd L1_x	0.269	0.111	1.333	0.988	0.001	1.0
rnd 0.5	6.518	0.781	10.667	0.0189	0.002	0.048
rnd L1	6.843	0.672	11.0	0.005	0.0	0.001
rnd 0.25	7.595	0.507	11.667	0.004	0.0	0.011
rnd 0.1	7.091	0.794	10.667	0.007	0.001	0.020
rnd 0.5|0.5	6.196	0.374	10.0	0.0160	0.0	0.042
rnd L1|L1	7.396	0.805	11.0	0.004	0.0	0.010
fwd	8.346	0.592	13.0	0.004	0.0	0.006
fwd detached	4.889	0.800	8.0	0.049	0.002	0.083
icm	7.130	1.416	11.333	0.002	0.001	0.004



Obr. 4.9: Porovnanie RND, ICM modulu a dopredného modelu z hľadiska dosiahnutej externej odmeny v prostredí Montezuma's Revenge.

Tabuľka 4.2: Prehľad experimentov v prostredí Montezuma's Revenge z hľadiska získanej externej a internej odmeny, vyjadrením pomocou štatistických ukazovateľov.

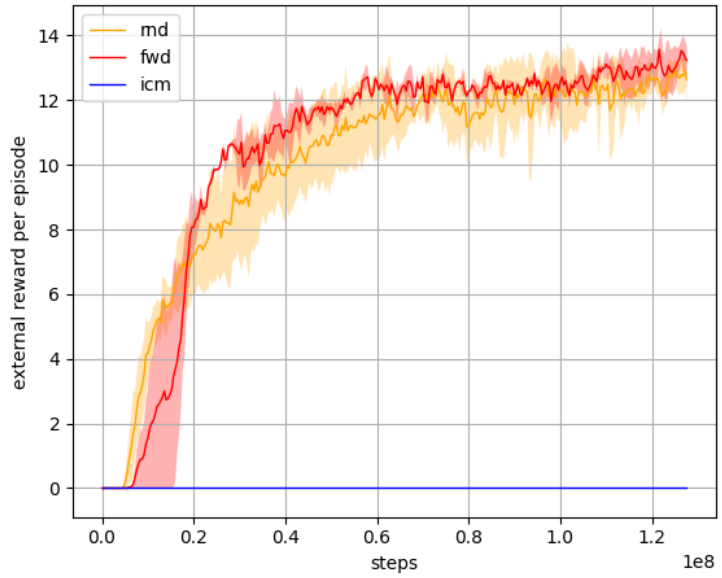
	externá odmena			interná odmena		
názov	mean	std	95%	mean	std	95%
rnd	5.071	0.060	8.667	0.050	0.003	0.100
fwd	4.178	0.240	7.0	0.003	0.0	0.007
icm	1.591	0.350	3.0	0.001	0.0	0.001

4.3 Prostredie Venture

Prostredie Venture sme využili na porovnanie tých istých troch prístupov generovania internej motivácie, ako v prostredí Montezuma's Revenge. Priebeh týchto experimentov je zobrazený na obrázku 4.10 a vypočítané hodnoty sú v tabuľke 4.3. Zaujímavé v tomto prípade je, že agent s ICM modulom síce občas dosahoval pozitívnu externú odmenu, ale takýchto prípadov bolo veľmi málo a nepodarilo sa mu z nich nič užitočné naučiť. V tabuľke 4.3 je priemerná externá hodnota experimentu s ICM modulom rovná nule, lebo výpočty robíme s presnosťou na 3 desatinné miesta. RND a dopredný model sa učia viac-menej rovnakou rýchlosťou, ale dopredný model dosahuje o trochu lepšie výsledky.

Tabuľka 4.3: Prehľad experimentov v prostredí Venture z hľadiska získanej externej a internej odmeny, vyjadrením pomocou štatistických ukazovateľov.

	externá odmena			interná odmena		
názov	mean	std	95%	mean	std	95%
rnd	10.247	0.884	15.0	0.015	0.0	0.055
fwd	10.930	0.217	15.0	0.003	0.0	0.004
icm	0.0	0.0	0.0	0.0	0.0	0.0



Obr. 4.10: Porovnanie modelov RND, ICM a dopredného modelu z hľadiska dosiahnutej externej odmeny v prostredí Venture.

Záver

V diplomovej práci sme sa venovali učeniu posilňovaním s internou motiváciou. Popísali sme viaceré problémy, ktoré sa častokrát vyskytujú pri učení posilňovaním aplikovanom na praktické úlohy a ako sa tieto problémy dajú riešiť pomocou stavovej reprezentácie a internej motivácie. Na problém prekliatia dimenzionality sme sa pozreli aj z pohľadu vzdialenostných metrík. Implementovali sme tri prediktívne metódy generovania internej motivácie založené na súčasných riešeniach a s rôznymi nastaveniami sme testovali ich úspešnosť na vybratých Atari hrách. Z našich experimentov vidíme, že modul internej zvedavosti dosahuje jednoznačne najhoršie výsledky, ale destilácia náhodnej siete a dopredný model s ST-DIM enkóderom fungujú lepšie a dosahujú porovnateľné výsledky. Úspešnosť jednotlivých modelov veľmi závisí od konkrétneho prostredia a preto nemôžeme povedať, že by niektorý z týchto dvoch prístupov bol najlepší.

Príloha

V elektronickej prílohe sa nachádzajú všetky zdrojové kódy využité v našej práci. Stručný návod na inštaláciu a použitie sa nachádza v súbore README.md a v priečinku results sa nachádzaju výsledky naších experimentov uložené ako .npy súbory. Tieto súbory spolu aj s konkrétnymi architektúrami neurónových sietí sú dostupné na https://github.com/Iskandor/MotivationModels/tree/diplomka_fm.

Literatúra

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In Van den Bussche, J. and Vianu, V., editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg. Springer Berlin Heidelberg.

Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M.-A., and Hjelm, R. D. (2019). Unsupervised state representation learning in atari. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Aubret, A., Matignon, L., and Hassas, S. (2019). A survey on intrinsic motivation in reinforcement learning. *ArXiv*, abs/1908.06976.

Barto, A. G. (2013). *Intrinsic Motivation and Reinforcement Learning*, pages 17–47. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bellman, R. (1957). *Dynamic Programming*. Dover Publications.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H., Raiman, J., Salimans, T., Schlatter, J., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning.

Böhmer, W., Springenberg, J. T., Boedecker, J., Riedmiller, M. A., and Obermayer, K. (2015). Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *KI - Künstliche Intelligenz*, 29:353–362.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019). Exploration by random network distillation. In *International Conference on Learning Representations*.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. (2019). Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*.

Lesort, T., Rodríguez, N. D., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural networks : the official journal of the International Neural Network Society*, 108:379–392.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Mataric, M. J. (1994). Reward functions for accelerated learning. In Cohen, W. W. and Hirsh, H., editors, *Machine Learning Proceedings 1994*, pages 181–189. Morgan Kaufmann, San Francisco (CA).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Munk, J., Kober, J., and Babuška, R. (2016). Learning state representation for deep actor-critic control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4667–4673.

Oudeyer, P.-Y. and Kaplan, F. (2009). What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 2778–2787. JMLR.org.

Pecháč, M. (2019). *Reinforcement learning for intrinsically motivated robot behavior.* PhD thesis, FMFI, Comenius University in Bratislava.

Ryan, R. M. and Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25(1):54–67.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Su, P.-H., Vandyke, D., Gašić, M., Mrkšić, N., Wen, T.-H., and Young, S. (2015). Reward shaping with recurrent neural networks for speeding up on-line policy learning in spoken dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 417–421, Prague, Czech Republic. Association for Computational Linguistics.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* The MIT Press, second edition.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5285–5294, Red Hook, NY, USA. Curran Associates Inc.