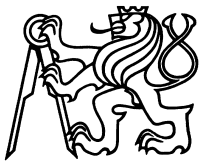
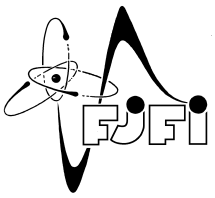


ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská



Automatické obchodování na akciové burze založené na fundamentálních datech a algoritmech zpětnovazebního učení

Automatic stock trading based on fundamental factors and reinforcement learning

Bakalářská práce

Autor: **Jakub Michna**

Vedoucí práce: **Ing. Pavel Strachota, Ph.D.**

Konzultant: **Ing. Jiří Šimonek, Ph.D.**

Akademický rok: 2022/2023

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Jakub Michna
Studijní program:	Matematické inženýrství
Studijní specializace:	Matematické modelování
Název práce (česky):	Automatické obchodování na akciové burze založené na fundamentálních datech a algoritmech zpětnovazebního učení
Název práce (anglicky):	Automatic stock trading based on fundamental factors and reinforcement learning

Pokyny pro vypracování:

1) Prostudujte zadanou literaturu a proveďte souhrnnou rešerši metod zpětnovazebního učení.
2) Seznamte se se základy fundamentální analýzy pro účely obchodování na akciových burzách.
3) Seznamte se s ekosystémem nástrojů pro manipulaci s daty a implementaci zpětnovazebního učení v jazyce Python, tj. například NumPy, Matplotlib, pandas, KerasRL, Pyqlearning, Tensorforce, RLlib apod.
4) Navrhněte konkrétní princip využití fundamentálních indikátorů pro investiční strategii agenta zpětnovazebního učení.
5) S pomocí nástrojů z bodu 3 a poskytnuté databáze historických fundamentálních dat o společnostech obchodovaných na newyorské burze se pokuste implementovat algoritmus maximalizující očekávaný zisk.

Doporučená literatura:

1) M. Sugiyama, Statistical Reinforcement Learning: Modern Machine Learning Approaches. CRC Press, Taylor & Francis Group, 2015.
2) L. Busoniu, R. Babuška, B. De Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, Taylor & Francis Group, 2010.
3) T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Data Mining, Inference, and Prediction. 2nd. ed., Springer-Verlag New York, 2009.
4) A. Plaat, Deep Reinforcement Learning, Springer, 2022.
5) C. C. Aggarwal, Neural Networks and Deep Learning, Springer, 2018.
6) McKinsey & Company, T. Koller, M. Goedhart, D. Wessels, Valuation: Measuring and Managing the Value of Companies 7th ed., Wiley, 2020.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Pavel Strachota, Ph.D.
KM FJFI ČVUT v Praze, Trojanova 13, 120 00 Praha 2

Jméno a pracoviště konzultanta:

Ing. Jiří Šimonek, Ph.D.
DYNAM Dynamic Asset Management s.r.o., Kludských 2740/4, 193 00 Praha 9

Datum zadání bakalářské práce: 31.10.2022

Datum odevzdání bakalářské práce: 2.8.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31.10.2022

.....
garant oboru

.....
vedoucí katedry

.....
děkan

Poděkování:
Chtěl bych zde poděkovat především svému školiteli Ing. Pavlu Strachotovi, Ph.D., za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé bakalářské práce. Dále děkuji svému konzultantovi Ing. Jirku Šimonovi, Ph.D. za jeho doporučení v oblasti tvorby obchodní strategie a také celé mojí rodině a kamarádům za psychickou, ale i materiální podporu během celého bakalářského studia.

Čestné prohlášení:
Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 2. srpna 2023 Jakub Michna

Název práce:

Automatické obchodování na akciové burze založené na fundamentálních datech a algoritmech zpětnovazebního učení

Autor: Jakub Michna

Obor: Matematické inženýrství

Zaměření: Matematické modelování

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Pavel Strachota, Ph.D., Katedra matematiky, Fakuta jaderná a fyzikálně inženýrská, ČVUT v Praze, Trojanova 13, 120 00 Praha 2

Konzultant: Ing. Jiří Šimonek, Ph.D., DYNAM Dynamic Asset Management s.r.o., Kludských 2740/4, 193 00 Praha 9

Abstrakt: Bakalářská práce se věnuje využití zpětnovazebního učení pro realizace investiční strategie pro obchodování na akciové burze na základě fundamentálních dat. V rámci práce jsou uceleně popsané přístupy zpětnovazebního učení v Markovských rozhodovacích procesech, konkrétně jsme se zabývali: metodou Monte Carlo, Temporální diferencí, Bootstrappingem, aproximací kvalitativní funkce pomocí neuronových sítí a metodami založenými na gradientu strategie. Popsané přístupy jsou ilustrovány na ukázkové úloze, ke které jsou přiložené i zdrojové kódy v jazyce Python. V posledních kapitolách jsme navrhli konkrétní realizaci investiční strategie, kterou jsme otestovali na syntetických datech.

Klíčová slova: gradient strategie, neuronové sítě, obchodování s akciemi, Q-učení, TD3, zpětnovazební učení

Title:

Automatic stock trading based on fundamental factors and reinforcement learning

Author: Jakub Michna

Abstract: The bachelor thesis focuses on the use of Reinforcement learning to implement an investment strategy for stock market trading based on fundamental data. The thesis will comprehensively describe the approaches of Reinforcement learning in Markov decision processes, specifically we will cover: the Monte Carlo method, Temporal Difference, Bootstrapping, approximation of the qualitative function by neural networks and gradient-based strategy methods. The described approaches are illustrated by a demonstrational problem, which is accompanied by Python source code. In the last chapters, we describe a concrete implementation of the investment strategy, which we test on synthetic data.

Key words: neural networks, policy gradient, Q-learning, Reinforcement learning, stock trading, TD3

Obsah

Úvod

V poslední době, díky zvýšení výpočetního výkonu grafických karet, došlo k rozmachu využití neuronových sítí v metodách zpětnovazebního učení (angl. RL). Algoritmy zpětnovazebního učení již porazily osmnáctinásobného mistra světa hry Go Lee Sedola v poměru 4:1 [34], naučily se hrát hry, které lze hrát na konzoli Atari 2600 [22], nebo také byly použity pro trénování dnes již tolik skloňovaného chatbota ChatGPT od společnosti OpenAi [36]. Zpětnovazební učení také proniklo do robotiky a není proto divu, že se zkouší aplikovat do světa investování [37]. Tato bakalářská práce představuje jeden z takových pokusů.

V úvodní kapitole je včleněno zpětnovazební učení mezi ostatní kategorie strojového učení. Popíšeme základní složky zpětnovazebního učení a zmíníme jeho využití v praxi. Zároveň jsou popsány matematické formulace mnohorukého bandity, plně i částečně pozorovatelného Markovova rozhodovacího procesu (MRP), kde u druhého zmíněného matematického konstruktu jsou zavedeny základní pojmy jako výnos, diskontní faktor, strategie a její kvalitativní a hodnotová funkce.

V druhé kapitole jsou prezentovány metody řešení úloh, které se dají formulovat jako MRP. Jedna taková ukázková úloha je navržena, matematicky popsána a její prostředí je implementováno v programovacím jazyce Python. Dále jsou popsány metody dynamického programování, které položily teoretické základy pro odvozování dalších metod řešení. Rešeršní část se poté zaměřuje na popis metody řešení, které jsou založeny na odhadu kvalitativní nebo hodnotové funkce strategie. Odhady v těchto metodách získáváme např. pomocí metod Monte Carlo, Temporální diference, nebo Bootstrappingu. V další části výkladu jsou popsány metody využívající neuronové sítě pro aproximace funkcí. Celý výklad je doprovázen popisy algoritmů, z nichž některé jsou implementovány v

`https://github.com/jamichy/BP_turista_RL`

a v poslední části kapitoly otestovány na ukázkové úloze.

Třetí kapitola se zabývá navržením investiční strategie využívající fundamentální indikátory. Pro realizaci navržené strategie je implementováno interaktivní prostředí. Dále je vysvětlen a následně implementován agentův algoritmus.

Čtvrtá kapitola je věnována představení syntetických dat, na kterých jsme investiční strategii testovali. V závěru kapitoly jsou diskutovány vlastnosti algoritmu, dosažené výsledky a jsou navržené kroky, které by mohly pomoci posunout navržený algoritmus blíže k praktickému využití.

Přehled použitého značení

Nebude-li řečeno jinak, budeme malá písmena používat pro hodnoty náhodných veličin a pro skalární funkce, zatímco velká písmena budou značit náhodné veličiny.

$\in$	náleží, je prvkem
$\subset$	podmnožina
$Pr(X = x)$	pravděpodobnost že náhodná veličina X nabývá hodnoty x
$\mathbb{N}$	množina přirozených čísel
$\mathbb{R}$	množina reálných čísel
$\mathbb{E}[X]$	střední hodnota náhodné veličiny X
$\arg\max_a f(a)$	hodnota a ve kterém funkce $f(a)$ nabývá maximální hodnoty
α, β	učící parametry
γ	diskontní faktor, $\gamma \in \langle 0, 1 \rangle$
ϵ	míra náhodnosti v ϵ-hladovém algoritmu, $\epsilon \in \langle 0, 1 \rangle$
$\propto$	úměrnost
$\leftarrow$	přiřazení
s, s'	stavy
a	akce
r	odměna
$\mathcal{S}$	množina všech nefinálních stavů
$\mathcal{S}^+$	množina všech stavů
$\mathcal{A}$	množina všech akcí
$\mathcal{R}$	množina všech odměn, konečná podmnožina $\mathbb{R}$
$\lvert\mathcal{S}\rvert$	počet prvků množiny $\mathcal{S}$
t	diskrétní časový krok
S_t	stav v čase t
A_t	akce v čase t
R_t	odměna v čase t
π	strategie
$\pi(s)$	akce vybraná ve stavu s deterministickou strategií
$\pi(a\lvert s)$	pravděpodobnost vybrání akce a ve stavu s pod strategií π
$p(s', r\lvert s, a)$	pravděpodobnost přesunu do stavu s' s odměnou r ze stavu s a akce a
$\tilde{p}(s'\lvert s, a)$	pravděpodobnost přesunu do stavu s' ze stavu s a akce a
$R(r\lvert s, a)$	pravděpodobnost obdržení odměny r ze stavu s a akce a

$v_\pi(s)$	hodnotová funkce ve stavu s vzhledem ke strategii π
$q_\pi(s, a)$	kvalitativní funkce ve stavu s a v akci a vzhledem ke strategii π
π_ω	strategie parametrizovaná parametrem ω
$\pi(a\|s; \omega)$	pravděpodobnost vybrání akce a ve stavu s pod strategií π_ω
$\mu(\cdot; \omega)$	diskrétní strategie parametrizovaná parametrem ω
$\mu(s; \omega)$	akce vybraná neuronovou sítí pro diskrétní strategii parametrizovanou parametrem ω
$h(s)$	pravděpodobnost začínání úlohy ve stavu s
$\lambda(s)$	distibuce stavu s pro epizodické nastavení vzhledem k strategii π_ω
$\lambda'(s)$	diskontovaná distibuce stavu s pro epizodické nastavení vzhledem k strategii π_ω
$\eta(s)$	průměrný počet stavu s v epizodě
$\eta'(s)$	diskontovaný průměrný počet stavu s v epizodě
$\rho^\pi(s)$	diskontovaná distribuce stavu s pro strategiiπ_ω
$\mathcal{D}$	zásobník zkušeností
$\mathcal{B}$	dávka dat pro trénování neuronové sítě
$\theta, \theta_1, \theta_2$	parametry neuronových sítí pro odhad kvalitativní funkce nebo hodnotové funkce
$\theta', \theta'_1, \theta'_2$	parametry cílových neuronových sítí pro odhad kvalitativní funkce nebo hodnotové funkce
ω	parametr strategie
$J(\omega)$	funkce měřící výkon strategie π_ω případně μ_ω
$b(s)$	funkce základní hodnota
ζ	prozkoumávací šum
α	učící parametr neuronové sítě, také parametr úpravy odhadu(velikost kroku)

Kapitola 1

Úvod do zpětnovazebního učení

1.1 Strojové učení

Strojové učení je soubor matematických algoritmů a statistických metod, jejichž cílem je automaticky odhalit vzory v datech a následně je použít pro předpovědi budoucích dat nebo dalších zajímavých výsledků[6]. Nejznámnější kategorie strojového učení jsou: *učení s učitelem*, *učení bez učitele* a *zpětnovazební učení*.

Strojového učení se uplatní v rozpoznávaní obrazu, detekci e-mailových spamů, cílené reklamě, doporučovacích systémech, autonomním řízení, analýze spotřebitelského chování, medicíně a mnoha dalších odvětvích[2].

1.1.1 Učení s učitelem (supervised learning)

Cílem tohoto přístupu je nalézt aproximaci funkce zobrazující jednotlivé vstupní záznamy x na požadované výstupy y na základě datového souboru dvojic $\mathcal{D}_0 = \{(x_i, y_i)\}_{i=1}^{K}$, kde K je celkový počet uspořádaných dvojic [6]. Datovou sadu $\mathcal{D}_0$ rozdělíme disjunktně do souborů s trénovacími ($\mathcal{D}$), validačními ($\mathcal{D}_\text{v}$) a testovacími ($\mathcal{D}_\text{t}$) daty. V nejjednodušším úloze je x_i j-dimenzionální vektor, kde jednotlivé složky vektoru se nazývají příznaky. Příznakem může být např. plat nebo pohlaví.

V trénovací fázi zkonstruujeme funkci f, jež bude aproximací hledané funkce, které na vstupu obdrží x_i z trénovacích dat a predikuje $f(x_i)$. Následně se vypočte chyba predikce, což je výstupní hodnota ztrátové funkce, která závisí na členech $|y_i - f(x_i)|$ pro $i \in M$, kde $M = \{k \in \mathbb{N} | k \leq O\} \wedge |M| = N$, kde O je počet dvojic v trénovacím souboru a N počet vybraných dat. V praxi se jako ztrátové funkce často používají střední kvadratická chyba (MSE) nebo střední absolutní chyba (MAE), definované jako

$$MSE = \frac{1}{N} \sum_{i \in M} (y_i - f(x_i))^2, \tag{1.1}$$

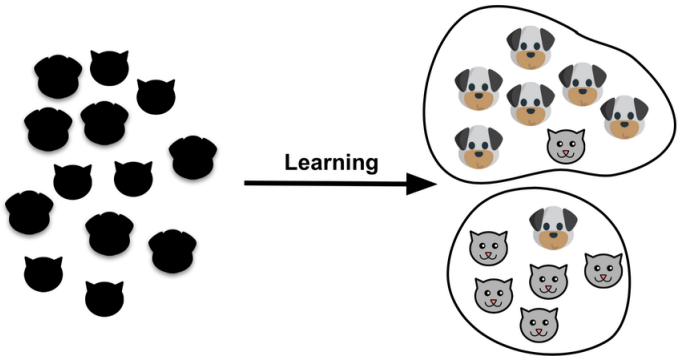
$$MAE = \frac{1}{N} \sum_{i \in M} |y_i - f(x_i)|. \tag{1.2}$$

Funkci f upravujeme tak, aby se chyba predikce na trénovacích datech minimalizovala. Tento proces se nazývá učení. Zároveň pozorujeme chybu na validačních datech. V okamžiku, kdy začne růst chyba predikce u validačních dat zastavíme trénování. Nakonec otestujeme funkci f na testovacích datech a určíme přesnost.

Podle charakteru y dělíme učení s učitelem na regresi a klasifikaci.

- V úloze na regresi je y_i vektorová veličina, kde jednotlivé příznaky nabývají reálné hodnoty. Regresní úlohou může být predikce prodejní ceny domu na základě příznaků, jakými jsou obytná plocha, počet pokojů, lokace, velikost zahrady a datum prodeje. Na začátku máme databázi prodaných nemovitostí s uvedenými příznaky a prodejní cenou. Cílem je najít funkci, která na vstupu dostane vektor příznaků a úspěšně predikuje prodejní cenu nemovitosti.

- U klasifikace veličina y_i nabývá kategorické hodnoty $\{1, \ldots, C\}$, kde C je počet tříd. Funkce f zde predikuje do jaké třídy daný datový záznam x_i patří [6]. Příkladem může být klasifikace obrázků na kočky a psy. Známými algoritmy klasifikace jsou *logistická regrese*, *rozhodovací strom*, *bayesovský naivní klasifikátor*, *metoda podpůrných vektorů* a *K-nejbližších sousedů* [7]. Na obrázku 1.1 můžeme vidět znázornění klasifikace.



Obrázek 1.1: Znázornění klasifikace obrázků na psy a kočky. Převzato z: `https://www.researchgate.net/figure/An-example-of-a-Supervised-Learning-classification-of-cats-and-dogs-and-b_fig1_328576527`

1.1.2 Učení bez učitele (unsupervised learning)

V tomto konceptu známe pouze datovou sadu $\mathcal{D} = \{x_i\}_{i=1}^{K}$, kde K je celkový počet vstupních dat a cílem je v nich nalézt zajímavé vztahy či struktury, které lze následně využít. Narozdíl od učení s učitelem zde neznáme vzory, které hledat ani na první pohled jasnou metriku, kterou použít pro výpočet chyby predikce [6]. Příklady metod učení bez učitele jsou *shlukování*, *asociaci*, *redukci dimenze*, *detekce anomálií*, *odhadování hustoty*.

Shlukování je proces slučování dat do jednotlivých skupin, kde prvky v každé skupině nabývají podobnějších hodnoty znaku. Příkladem může být rozdělení populace mezi muže a ženy. Algoritmy jsou K-means, Hiearchical clustering. Více lze nalézt v [18].

Asociace je přístup učení založený na pravidlech, při kterém objevujete vzor, který popisuje velkou část daných dat. Například v internetovém obchodě s nábytkem doporučovací stroj naznačuje, že lidé, kteří si koupí stůl A, si určitě koupí i židle B [2].

1.1.3 Zpětnovazební učení (reinforcement learning)

Myšlenka, že se učíme skrze interakci s prostředím se objevuje když přemýšlíme nad základní podstatou učení. Tato idea přispěla ke vzniku zpětnovazebního učení, jako jednoho z typů strojového učení. Na rozdíl od učení s učitelem nebo bez učitele zde nemáme zpočátku žádné vstupní ani výstupní data, zato interaktivní prostředí a agenta [1]. Data získáváme skrze interakci agenta s prostředím. Z prostředí

dostává agent zpětnou vazbu v podobě odměny či penalizace a další situace, které čelí. Cílem agenta je naučit se dělat posloupnost dobrých rozhodnutí, tj. mapovat situace na akce za účelem dosažení maximální odměny.

Zpětnovazební učení lze charakterizovat pomocí následujících vlastností.

- Optimalizace. Cílem je najít optimální způsob jak dělat rozhodnutí. Př: Najít nejkratší vzdálenost mezi dvěmi městy, když známe síť cest.
- Opožděné následky. Rozhodnutí teď může mít vliv na věci později. Typickým příkladem může být spoření na důchod. Pokud v produktivním věku investujeme nebo spoříme, nemusíme se ve stáří spoléhat na státní starobní důchod.
- Prozkoumávání. Učení se o světě děláním rozhodnutí. Když jsme se učili jezdit na kole, neměli jsme žádná manuál jak no to. Učili jsme se zkoušením a od prostředí jsme dostávali odměnu - pochvalu od rodičů případně penalizaci - modřinu na koleně.
- Cenzurované informace. Dostaneme jenom odměnu/penalizaci a následující situaci za akci, kterou uděláme. Nevíme co by se stalo, kdybychom si vzali místo modré pilulky červenou.
- Následek rozhodnutí ohledně naučené věci. Pokud si vybereme studium na MFF UK místo FJFI ČVUT, budeme mít jiné zkoušenosti a znalosti.

Dobrým způsobem jak porozumět zpětnovazebnímu učení je uvedení příkladu ze života. Když se dítě učí chodit, nemá žádné pokyny, jak by mělo postupovat. Prostě se postaví, ujde pár kroků a spadne. Tento cyklus se opakuje mnohokrát, než soustavně začne chodit. Během zkoušení samozřejmě dostává zpětnou vazbu od prostředí například v podobě dostání se k hračce nebo bolesti po pádu. Zároveň se v každém čase nachází v konkrétním stavu např. 15° ohyb v kolenou, ruce v pravém úhlu vůči tělu, trup mírně předkloněný. Dítě se snaží maximalizovat pozitivní odměnu a minimalizovat pády.

RL je využíváno v různých oblastech a jeho aplikace jsou stále rozšiřovány díky své schopnosti řešit složité úkoly za podmínek, kdy tradiční algoritmy selhávají. Uveďmě několik příkladů aplikace RL napříč obory:

1. Hra Go: Výpočetně náročná desková hra Go byla po dlouhou dobu nezdolanou výzvou pro umělou inteligenci. Až v roce 2016 byly s pomocí RL a hlubokého učení vyvinuty algoritmy (např. AlphaGo), které porazily nejlepší lidské hráče až do té doby.
2. Hra Dota 2: Multi-agentní RL bylo úspěšně nasazeno na strategickou hru Dota 2, kde modely (např. OpenAI Five) dokázaly hrát ve složitých týmových bitvách proti profesionálním hráčům.
3. Chlazení Google Data Storage: Společnost Google používá RL k optimalizaci rozložení chladicích jednotek chlazení svých datových center, čímž dosahují snížení spotřeby energie.
4. ChatGPT: RL bylo využito k trénování modelu GPT-3 pro konverzační schopnosti, což je dnes systém známý pod názvem ChatGPT. Tento model umožňuje provádět konverzace s uživateli, odpovídat na otázky a generovat text [36].
5. Zemědělství

   V zemědělství má každé rozhodnutí nebo činnost pro zemědělce následný dopad na výnos nebo produkci plodin. Typy rozhodnutí jsou výběr zasazené plodiny, míra hnojení a zalévání, doba sklizně, výběr vhodné půdy atd. Pro ulehčení práce vznikl Systém pro podporu rozhodování v oblasti transferu agrotechnologií (DSSAT) [12], což je program, který zahrnuje simulační modely

pro více než 42 plodin a také nástroje pro usnadnění efektivního používání modelů. Nástroje zahrnují programy pro správu databází půdy, počasí, řízení plodin a experimentálních dat. Modely pro simulaci plodin simulují růst, vývoj a výnos v závislosti na typu půdy, rostliny a atmosféry. V roce 2022 se podařilo skupině francouzských vědců tento model z části přenést a vytvořili prostředí gym-DSSAT, které umožňuje interakci agenta a prostředí. Následně natrénovali agenta, který v simulaci DSSAT vyprodukoval obdobné množství kukuřice s nižším použitím dusíku než expertní zemědělec [11].

Tyto příklady ukazují pár aplikací RL ze široké škály možností.

1.2 Stavební kameny

V následující části jsou ve zkratce popsány základní části zpětnovazebního učení a to prostředí, agent a odměna.

1.2.1 Prostředí

Prostředí modeluje chovaní úlohy a zprostředkovává interakci s agentem. Pokud chceme aplikovat zpětnovazební učení, musíme mít pro danou řešenou úlohu naprogramované odpovídající prostředí. V mnoha případech programátor může využít již naprogramovaného prostředí z některé knihoven. Nejznámější knihovnou prostředí je Gymnasium [32] od společnosti OpenAi obsahující pestré množství balíčků úloh jako například Atari hry nebo MuJoCo, což je fyzikální modul pro všeobecné použití v robotice, biomechanice a grafice. V případě, že pro námi řešenou úlohu neexistuje implementace prostředí, musíme ji naprogramovat. Pro úlohu, v níž se vyskytují stavy, akce a odměny, implementujeme třídu, kterou je nejprve potřeba inicializovat a posléze k ní přidat 2 metody. První metoda se označuje jako krok (step) a na vstupu obdrží akci, kterou agent vybral. Na výstupu vrátí následující stav, odměnu a zda je agent ve finálním stavu. Druhá metoda se nazývá reset. Nastaví agentův stav na počáteční a tento stav vrátí agentovi.

1.2.2 Agent

Agent je subjekt disponující strategií pro vybírání akcí. V praxi budeme chtít najít nejlepší agentovu strategii pro danou úlohu. Strategii budeme měnit v závislosti na datech, které agent dostal z prostředí pomocí různých algoritmů. Některé algoritmy jsou již implementované v knihovnách Tensorforce [33] nebo RLlib[35].

1.2.3 Odměna/penalizace

Odměna slouží agentovi jako informace, jak dobře se choval. Odměna může být pozitivní, negativní nebo žádná a agentova strategie je pak silně ovlivněna způsobem odměňování.

Nechť máme úlohu bludiště, kde se má agent dostat z počáteční pozice do finální, kde hra skončí. Zároveň chceme, aby agent našel finální pozici co nejrychleji. Pokud by agent za každý krok dostával pozitivní odměnu, neměl by žádnou motivaci dojít do finální pozice a bloudil by v bludišti do nekonečna. Řešením na popsaný problém je změnit model odměňování, kde za každý krok dostane agent negativní odměnu (penalizaci).

Odměnu může agent dostávat s různou frekvencí. Pokud agent dostává odměnu ihned po provedené akci, je trénování snadnější. V některých případech je nutné pro správné navržení způsobu odměňování v prostředí využít rad experta z praxe, který je s danou úlohou obeznámen.

1.3 Matematický popis úloh Zpětnovazebního učení

1.3.1 Mnohoruký bandita (Multi-armed bandit)

Mnohoruký bandita je klasický problém, kde se vyskytuje dilema zda má agent prozkoumávat prostředí, či vytěžovat nejlepší nalezenou strategii. Pro lepší pochopení uvedeme následující úlohu. Agent se nachází v kasínu před K herními automaty a v každém časovém kroku si zahraje na jednom automatu. Každý automat má stochastické rozdělení odměny, které je pro agenta neznámé. Cílem je během konečného počtu časových kroků dosáhnout maximální možné kumulativní odměny [9].

S využitím matematiky můžeme tento koncept přepsat následovně. Po celou dobu je na výběr K možných akcí z prostoru $\mathcal{A} = \{1, \ldots, K\}$, kde akce $a = l$ pro $l \in \{1, \ldots, K\}$ odpovídá výběru l-tého automatu. V časovém kroku t vybere agent akci $A_t = a$ a obdrží odměnu R_t což je náhodná veličina s rozdělením D_a. Pro každou akci $a \in \mathcal{A}$ definujeme hodnotu akce a jako střední hodnotu odměny za provedení akce a, tj. $q_*(a) = \mathbb{E}[R_t | A_t = a]$. Agentův cíl je během H časových kroků maximalizovat celkovou očekávanou odměnu, tj. maximalizovat $\sum_{t=1}^{H} R_t$. Pokud bychom znali pro každou akci její hodnotu, pak by bylo snadné tuto úlohu vyřešit - vždy bychom zvolili akci, která má nejvyšší hodnotu. Abychom mohli porovnávat různé akce během hry, označme jako $Q_t(a)$ odhad hodnoty akce v časovém kroku t. Chtěli bychom přirozeně, aby $Q_t(a)$ bylo blízko $q_*(a)$.

Využitím definice $q_*(a)$ odvodíme odhad $Q_t(a)$ jako

$$Q_t(a) = \frac{\text{součet odměn za provedení akce } a \text{ do času } t}{\text{počet provedení akce } a \text{ do času } t} = \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}},$$

kde $\mathbb{1}$ je indikátor jevu, který nabývá hodnoty 1 pokud jev nastal, jinak hodnoty 0. Pokud je jmenovatel nulový, nastavíme $Q_t(a) = 0$. Pokud jmenovatel půjde do $+\infty$, bude $Q_t(a)$ ze zákona velkých čísel konvergovat k $q_*(a)$.

V praxi chceme odhad hodnoty akce napočítat pokaždé, když agent udělá akci a, zároveň nechceme uchovávat celou historii odměn. Ke zjednodušení zápisu v následující rovnici se zaměříme pouze na jednu akci. Nechť R_i označuje odměnu po i-tém vybrání akce a Q_n odhad hodnoty akce. Pak $Q_n = \frac{R_1 + \cdots + R_{n-1}}{n-1}$.

Vztah pro Q_{n+1} lze přepsat jako

$$\begin{aligned} Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}(R_n + \sum_{i=1}^{n-1} R_i) = \frac{1}{n}(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i) \\ &= \frac{1}{n}(R_n + (n-1)Q_n) = \frac{1}{n}(R_n + nQ_n - Q_n) = Q_n + \frac{1}{n}(R_n - Q_n). \end{aligned} \tag{1.3}$$

Z (1.3) plyne, že při počítání odhadu nám stačí ukládat do paměti pouze poslední odhad a počet provedení akce.

Pokud uchováváme odhady pro každou akci, jedna akce bude mít v daný okamžik nejvyšší hodnotu odhadu. Tuto akci nazýváme hladovou (greedy) a pokud ji agent vybere, říkáme, že agent vytěžuje současnou znalost hodnot akcí. V opačném případě říkáme, že agent prozkoumává prostředí. Prozkoumáváním prostředí získáme lepší odhad hodnoty akcí, které nejsou hladové. Vytěžování hladové strategie je dobré pro maximalizaci očekávané odměny v případě, kdy máme dobrý odhad hodnoty všech akcí [1, kap. 2.1].

Jedním z algoritmů řešícím balancování prozkoumávaní a vytěžováním je ϵ-hladový algoritmus (ϵ-greedy). Na počátku inicializujeme pravděpodobnost výběru náhodné akce ϵ. Pro všechny akce nastavíme odhad hodnoty a počet provedení akce na 0. Pro každý časový krok vybereme hladovou akci s

pravděpodobností $1-\epsilon$ a s pravděpodobností ϵ náhodnou akci. Vybranou akci a dáme prostředí $bandit(s)$ a dostaneme zpět odměnu R. Následně zvýšíme počet provedení akce o 1, spočteme nový odhad hodnoty akce a posuneme se do dalšího časového kroku [1, kap. 2.4].

Algoritmus 1 ϵ-hladový algoritmus pro mnohorukého banditu

$t \leftarrow 1, \epsilon \in (0, 1]$
Pro $a \in \{1, \ldots, k\}$
    $Q(a) \leftarrow 0$
    $N(a) \leftarrow 0$
konec Pro
Dokud $t \leq H$
    $A \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}} Q(a), & \text{s pravděpodobností } 1-\epsilon \\ a \text{ náhodná akce}, & \text{s pravděpodobností } \epsilon \end{cases}$
    $R \leftarrow bandit(a)$
    $N(a) \leftarrow N(a) + 1$
    $Q(a) \leftarrow Q(a) + \frac{1}{N(a)}(R - Q(a))$
    $t \leftarrow t + 1$
konec Dokud

1.3.2 Markovův rozhodovací proces (Markov decision process)

Markovův rozhodovací proces lze zavést českou zkratkou MRP, je zobecnění Mnohorukého bandity, kde přidáme do rozhraní stavy a nemusíme se omezovat na konečný počet časových kroků, ve kterých bude agent s prostředím interagovat. Formálně řečeno, prostředí a agent navzájem interagují v každém diskrétním časovém kroku $t = 0, 1, \ldots$. V každém časovém kroku t agent dostane reprezentaci stavu prostředí $S_t \in \mathcal{S}$, na jejímž základě vybere akci $A_t \in \mathcal{A}$. V dalším časovém kroku agent obdrží od prostředí odměnu $R_{t+1} \in \mathcal{R}$ a další stav, ve kterém se nalézá $S_{t+1} \in \mathcal{S}$. Posloupnosti výše popsaných interakcí vzniká trajektorie $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$. Cílem je maximalizovat očekávanou kumulativní odměnu [1, kap. 3.1].

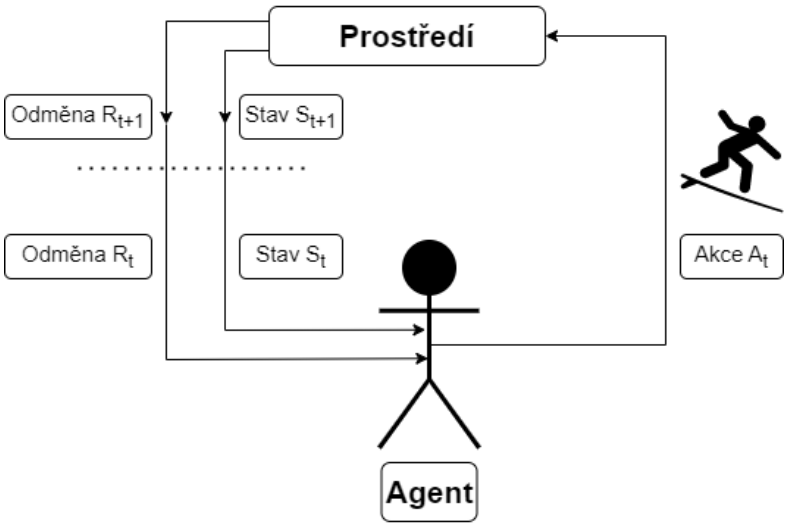
Definice 1.1. MRP je uspořádaná pětice $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, kde

- $\mathcal{S}$ je množina stavů,
- $\mathcal{A}$ je množina akcí,
- $\mathcal{R}$ je množina odměn,
- p je model prostředí MRP, tj. pro $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ a $r \in \mathcal{R}$ je

$$p(s', r|s, a) = \Pr(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a), \tag{1.4}$$

- $\gamma \in \langle 0, 1\rangle$ je diskontní faktor definovaný níže.

Poznámka: Model prostředí p má tzv. Markovovu vlastnost, která nám říká, že pravděpodobnost je podmíněna pouze hodnotami současného stavu a akce, nikoliv celou historií stavů a akcí. Díky tomu nemusíme uvádět na levé straně (1.4) závislost na t.



Obrázek 1.2: Znázornění interakce agenta a prostředí v Markovovově rozhodovacím procesu.

V následující části se zaměříme pouze na konečné Markovské rozhodovací procesy, kde množina stavů $\mathcal{S}$, akcí $\mathcal{A}$ a odměn $\mathcal{R}$ má konečný počet prvků.

Někdy se místo modelu prostředí p v definici uvádí tranzitní a odměňovací model. Odměňovací model $R : \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ definovaný $\forall r \in \mathcal{R}, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ určuje pravděpodobnost, že akce a povede ze stavu s k odměně r a platí pro něj

$$R(r|s, a) = \sum_{s' \in \mathcal{S}} p(s', r|s, a).$$

Tranzitní model $\tilde{p} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ definovaný $\forall r \in \mathcal{R}, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ určuje pravděpodobnost, že akce a povede ze stavu s do stavu s', a platí pro něj

$$\tilde{p}(s'|s, a) = \Pr(S_{t+1} = s'|S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} p(s', r|s, a).$$

V některých úlohách nemůžeme v určitých stavech provést všechny akce, proto označme $\mathcal{A}(s)$ množinu akcí, které lze provést ze stavu $s \in \mathcal{S}$.

Odměna, Výnos, diskontní faktor

V každém časovém kroku dostane agent od prostředí odměnu, která je závislá na předchozím stavu a předchozí akci. Nezajímáme se tak moc o momentální odměnu, spíše o celkový součet odměn od tohoto časového kroku, který nazýváme výnos. V nejjednodušším případě je výnos od času t součtem odměn

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T, \tag{1.5}$$

kde T je finální časový krok. Tento přístup má smysl v úlohách, kde se přirozeně vyskytne pojem finálního časového kroku, tj. když se posloupnost interakcí agenta s prostředím přirozeně rozpadne do nezávislých podposloupností. Tyto podposloupnosti nazýváme epizodami a úlohy epizodickými. Každá epizoda končí ve finálním stavu, po němž následuje návrat do výchozího počatečního stavu. Šachy jsou typickým představitelem epizodické hry, protože každá hra skončí buď výhrou, prohrou nebo remízou.

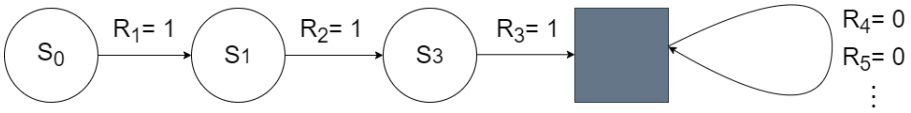
V epizodických úlohách někdy potřebujeme rozlišovat mezi množinou stavů nefinálních označených $\mathcal{S}$ a množinou všech stavů včetně finálních označenou $\mathcal{S}^+$ [1, kap. 3.3].

V opačném případě, pokud agent s prostředím vzájemně intragují nepřetržitě, tj. posloupnost interakcí se nerozpadne do nezávislých podposloupností, nazýváme úlohu nepřetržitou (continuous). Pro tyto úlohy je finální časový krok $T = +\infty$ a výše definovaný výnos (1.5) by nebylo možné obecně spočítat. Proto rozšíříme definici výnosu (1.5) pro nepřetržité úlohy o diskontní/slevový faktor $\gamma \in \langle 0, 1\rangle$ vyjadřující poměr významnosti odměny v následujícím časovém kroku vzhledem k významnosti aktuální odměny. Výnos pak přejde do tvaru

$$G_t = \sum_{k=0}^{+\infty} \gamma^k R_{t+k+1}. \tag{1.6}$$

Pro posloupnost $\{R_{t+k+1}\}_{k=0}^{+\infty}$, která je omezená a pro $\gamma \in \langle 0, 1)$ je výnos (1.6) konečný. Pro $\gamma = 0$ výnos (1.6) přejde pouze v okamžitou odměnu R_{t+1}. Čím větší bude γ, tím větší budeme klást důraz i na pozdější odměny.

Abychom sjednotili notaci výnosu pro epizodické a nepřetržité úlohy, zavádíme pro epizodické úlohy absorbční stav. Z tohoto stavu vede každá akce do absorbčního stavu a agent dostává nulovou odměnu. Díky zavedení absorbčního stavu můžeme psát i pro epizodické úlohy výnos ve tvaru (1.6).



Obrázek 1.3: Znázornění absorbčního stavu.

Strategie (Policy)

Strategie hraje ve zpětnovazebním učení klíčovou roli, neboť určuje jaké akce se mají v závislosti na stavu provést. Strategie π je rozdělení podmíněné pravděpodobnosti, která pro každý možný stav určuje pravděpodobnost každé možné akce, tj. v časovém kroku t je $\pi(a|s) = \Pr(A_t = a|S_t = s)$. Speciálním případem je deterministická strategie, která v každém stavu vybere pouze jednu akci $\pi(s) = a$ [3, str. 35-36].

Hodnotová a kvalitativní funkce (Value function, Action-Value function)

Abychom mohli porovnávat různé strategie pro Markovské rozhodovací procesy, zavedeme hodnotovou, respektive kvalitativní funkci (value function and action-value function).

Hodnotová funkce stavu $s \in \mathcal{S}$ vzhledem ke strategii π

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1}|S_t = s], \tag{1.7}$$

kde $\mathbb{E}_\pi$ je střední hodnota náhodných veličin v MRP řízeném strategií π, vyjadřuje očekávanou hodnotu výnosu, když začneme ve stavu s a akce jsou vybírány podle strategie π. Využitím vztahu (1.6) a modelu prostředí p získáme přepis

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')). \end{aligned} \tag{1.8}$$

Kvalitativní funkci ve stavu $s \in \mathcal{S}$ a v akci $a \in \mathcal{A}$ vzhledem ke strategii π označme $q_\pi(s, a)$. Vyjadřuje očekávanou hodnotu výnosu, když začneme ve stavu s, provedeme akci a a následné akce jsou vybírány podle strategie π. Platí tedy

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{k+t+1} | S_t = s, A_t = a]. \tag{1.9}$$

Mezi $v_\pi(s)$ a $q_\pi(s, a)$ platí vztahy

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) q_\pi(s, a), \\ q_\pi(s, a) &= \sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s')). \end{aligned} \tag{1.10}$$

Pokud pro libovolné dvě strategie π_1, π_2 platí: $v_{\pi_1}(s) \geq v_{\pi_2}(s)$ pro $\forall s \in \mathcal{S}$, řekneme, že strategie π_1 je lepší než strategie π_2 a značíme $\pi_1 \geq \pi_2$. Optimalní hodnotová funkce je definovaná jako

$$v_*(s) = \max_\pi v_\pi(s)$$

a analogicky definujeme optimální kvalitativní funkci

$$q_*(s, a) = \max_\pi q_\pi(s, a).$$

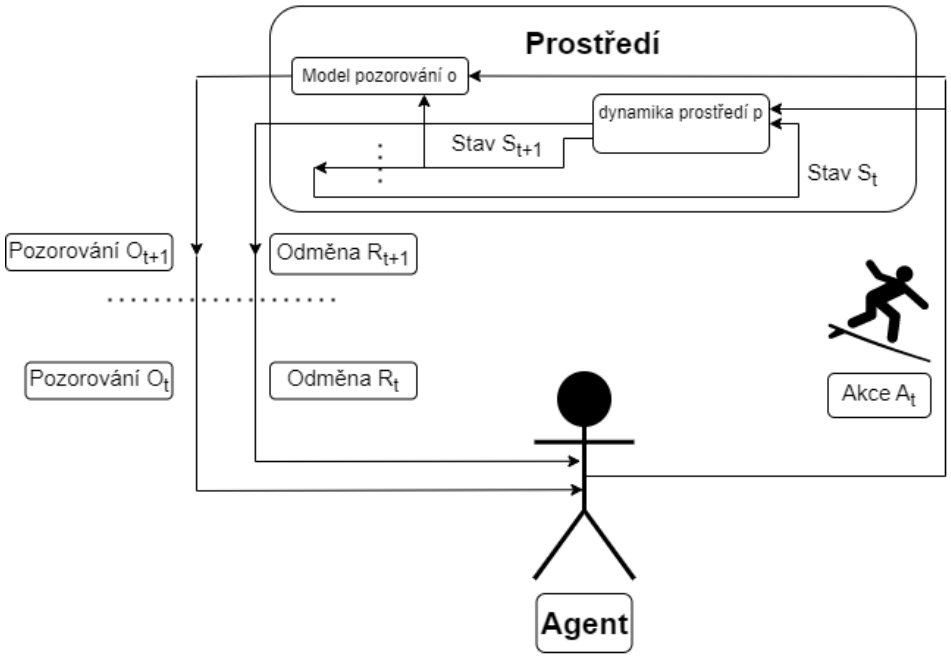
Každou strategii π_* splňující $v_\pi = v_*$ nazveme optimální strategií. Můžeme ji také definovat jako $\pi_*(s) = \operatorname{argmax}_a q_{\pi_*}(s, a)$, kde argmax nemusí být dán jednoznačně.
Pokud je stavový a akční prostor konečný, pak počet všech deterministických strategií je $|\mathcal{A}|^{|\mathcal{S}|}$.

1.3.3 Částečně pozorovatelný Markovský rozhodovací proces (Partially observable Markov decision process)

Částečně pozorovatelný Markovův rozhodovací proces je zobecnění Markovova rozhodovacího procesu, který navíc obsahuje množinu pozorování O a model pozorování $\mathbf{o}$. Prostředí a agent navzájem interagují v každém diskrétním časovém kroku $t = 0, 1, \ldots$. V každém časovém kroku t se prostředí nachází ve stavu $S_t \in \mathcal{S}$ a agent dostává od prostředí pozorování $O_t \in O$, na jehož základě vybere akci $A_t \in \mathcal{A}$. V dalším časovém kroku agent obdrží od prostředí odměnu $R_{t+1} \in \mathcal{R}$ a další pozorování $O_{t+1} \in O$, které vzniklo v prostředí z $S_{t+1} \in \mathcal{S}$ a $A_t \in \mathcal{S}$ za pomoci modelu $\mathbf{o}$. Cílem je maximalizovat očekávanou kumulativní odměnu stejně jako u Markovských pozorovatelných procesů [10].

Definice 1.2. Částečně pozorovatelný MRP je uspořádaná sedmice $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma, O, \mathbf{o})$, kde

- $\mathcal{S}$ je množina stavů,
- $\mathcal{A}$ je množina akcí,
- $\mathcal{R}$ je množina akcí,
- p je model prostředí, tj. pro $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ a $r \in \mathcal{R}$ platí (1.4) stejně jako u MRP,
- $\gamma \in [0, 1]$ je diskontní faktor,



Obrázek 1.4: Znázornění interakce agenta a prostředí v Částečně pozorovatelném Markovovově rozhodovacím procesu.

- O je množina pozorování,
- $\mathbf{o}$ je model pozorování, tj. pro $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ a $o' \in O$ platí $\mathbf{o}(o'|s', a) = \Pr(O_{t+1} = s'|S_{t+1} = s, A_t = a)$ vyjadřuje pravděpodobnost, že akce $a \in \mathcal{A}$ povede ze stavu $s' \in \mathcal{S}$ k pozorování $o' \in O$.

Aplikace zahrnují problémy navigace robotů, údržby strojů a plánování za nejistoty obecně [10].

Kapitola 2

Algoritmy Zpětnovazebního učení

V sekci 2.1 ukážeme příklad epizodické úlohy, která se dá formulovat v rámci MRP. Popíšeme model prostředí a přeneseme úlohu do zdrojového kódu. V sekci 2.2 popíšeme přístup dynamického programování, který je důležitý z teoretického hlediska a na kterém staví dále popsané metody. V sekcích 2.3, 2.4 se zaměříme pouze na metody řešení úloh popsané ve formě MRP s konečnou množinou stavů $\mathcal{S}$, akcí $\mathcal{A}$ a odměn $\mathcal{R}$ a představíme algoritmy, z nichž některé implementujeme pro řešení úlohy představené v sekci 2.1. Dále v sekci 2.5 budeme aproximovat kvalitativní funkci neuronovou sítí a v sekci 2.6 představíme gradient strategie pro parametrizované strategie.

2.1 Ukázková úloha - Turista v terénu

Pro lepší pochopení MRP s konečnou množinou stavů $\mathcal{S}^+$, akcí $\mathcal{A}$ a odměn $\mathcal{R}$ jsem navrhl epizodickou úlohu, na níž vysvětlím tvorbu prostředí a dále budu na ni ilustrovat vybrané algoritmy v kapitole 2.
Představení úlohy

Mějme úlohu, kde se agent vyskytuje v terénu a jeho cílem je objevit cestu z výchozí pozice do finální, aby spotřeboval nejméně energie. Přitom agent neví, kde se finální pozice nachází, a musí proto mapu důkladně prozkoumat, aby finální pozici našel. Mapa terénu je vyobrazena pomocí čtvercových políček s příslušnou nadmořskou výškou. Agent se může pohybovat nahoru, dolů, doprava, doleva a po diagonále vždy o jedno políčko. Pokud provede akci, při které by se dostal mimo mapu, zůstává na stejném políčku a dostává odměnu -10. Za akci, která vede do finálního stavu dostává odměnu 5, bez ohledu na to, jaká byla výška políčka, ze kterého akci udělal. Každá akce, která nevede do finální pozice stojí agenta energii - agent od prostředí dostává penalizaci. Tato penalizace je vyšší, čím větší je rozdíl výšek současného a následujícího políčka, kde se akcí dostane, a také pokud se vydal po diagonále místo pohybu do jedné ze světových stran. Zvolili jsme si, že agent bude začínat v levém dolním rohu a finální pozice bude v pravém horním rohu. K dispozici je ještě volitelný výškový faktor L, který vyjadřuje poměr spotřebované energie pro překonání výškového rozestupu o délce 1 oproti potřebné energii pro pohyb v rovině o stejné délce.

2.1.1 Matematický popis

Nechť máme mapu rozměru $m \times n$, kde m je počet sloupců a n je počet řádků. Pak

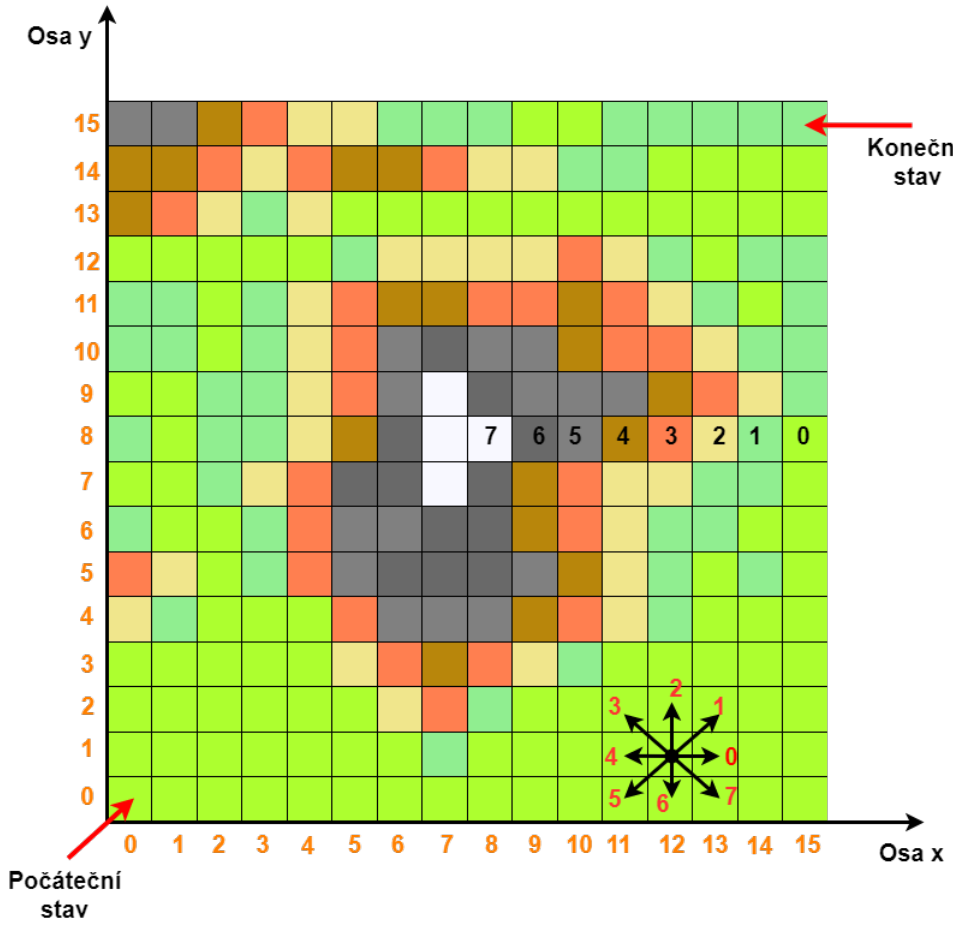
- $\mathcal{S}^+ = \{(i, j) | i \in \{0, \ldots, m-1\}, j \in \{0, \ldots, n-1\}\}$, $\mathcal{S} = \mathcal{S}^+ \setminus \{(m-1, n-1)\}$
- $\mathcal{A} = \{0, 1, \ldots, 7\}$, kde např. akce $a = 0$ má význam přesunu agenta o vektor $(1, 0)$, tj. o jedno políčko ve směru osy x a o nula políček ve směru osy y. Označení ostatních akcí je zřejmé z

obrázku 2.1. Zaveďme význam akce jako funkci f, která vezme akci $a \in \mathcal{A}$ a přiřadí ji vektor, podle výčtu: $\{0 : (1,0), 1 : (1,1), 2 : (0,1), 3 : (-1,1), 4 : (-1,0), 5 : (-1,-1), 6 : (0,-1), 7 : (1,-1)\}$

- $\tilde{p}(s'|s,a) = \begin{cases} 1, & \text{pro } s + f(a) = s' \text{ nebo } (s = s' \wedge s + f(a) \notin \mathcal{S}^+) \\ 0, & \text{jinak,} \end{cases}$
- $R(r|s,a) = \begin{cases} 1, & \text{pokud } r = -\sqrt{(h(s+f(a)) - h(s))^2 L^2 + f(a)^2} \text{ nebo } (r = -10 \wedge s + f(a) \notin \mathcal{S}^+) \\ 1, & \text{pokud } r = 5 \wedge s + f(a) = (m-1, n-1) \\ 0, & \text{jinak,} \end{cases}$

  kde $h(s)$ určuje výšku políčka, které odpovídá stavu s. Výraz $f(a)^2$ chápeme jako kvadrát normy vektoru $f(a)$ a $s + f(a)$ jako součet vektorů po složkách.
- $\mathcal{R} = \{-\sqrt{(h(s+f(a)) - h(s))^2 L^2 + f(a)^2} | s \in \mathcal{S} \wedge a \in \mathcal{A} \wedge (s + f(a)) \in \mathcal{S}^+\} \cup \{-10, 5\}$
- $\gamma = 1$

My se budeme zabývat mapou o velikosti 16×16 a výškový faktor L nastavíme na hodnotu 2. Obrázek 2.1 nám představuje konkrétní výškový profil, na kterém budeme algoritmy testovat. Barva políček vyjadřuje výšku, na stupnici od světle zelené, odpovídající výšce 0, až po bílou znamenající výšku 7.



Obrázek 2.1: Znázornění úlohy Turista v terénu

2.1.2 Tvorba kódu prostředí

Nyní převedeme matematický popis úlohy do programovacího kódu v jazyce Python a vysvětlíme jeho jednotlivé části. Zaměříme se pouze na čisté převedení úlohy a nebudeme popisovat grafické vykreslování trajektorií jednotlivých epizod.
Prvním krokem je import použitých knihoven (kód 2.1).

Kód 2.1: Import knihoven

```
1 import numpy as np
2 import pandas as pd
3 from math import sqrt
```

V kódu 2.2 zkonstruujeme třídu `GridWorld`, kterou inicializujeme funkcí

```
__init__(self, m, n, filepathTerrain, altitudeFactor),
```

kde `m` a `n` jsou rozměry mapy, `filepathTerrain` odpovídá názvu csv souboru, ve kterém jsou uloženy výšky jednotlivých políček a `altitudeFactor` odpovídá proměnné L v matematickém modelu. Nejdříve uložíme proměnné `m` a `n` a také `altitudeFactor` a načteme mapu. Poté vytvoříme seznamy obsahující $\mathcal{S}^+$ a $\mathcal{S}$. Následně vytvoříme slovník `self.actionSpace`, který odpovídá funkci f z matematického modelu, tj. pro akci

$a \in \mathcal{A}$ nám vrátí odpovídající vektor. Nakonec vytvoříme ještě seznam obsahující $\mathcal{A}$ a nastavíme agentův počáteční stav na $(0, 0)$.

Kód 2.2: Inicializace třídy

```
class GridWorld(object):
    def __init__(self, m, n, filepathTerrain, altitudeFactor):
        self.m = m
        self.n = n
        self.terrain = pd.read_csv(filepathTerrain) #nacteni souboru
        self.altitudeFactor = altitudeFactor
        self.stateSpacePlus = []

        #vytvoreni mnoziny vsech stavu vcetne finalniho stavu
        for i in range(self.m):
            for j in range(self.n):
                self.stateSpacePlus.append((i, j))
        self.stateSpace = self.stateSpacePlus.copy()
        self.stateSpace.remove((self.m-1, self.n-1))

        #vytvoreni vyznamu jednotlivych akci
        self.actionSpace = {0: (1, 0), 1: (1, 1), 2: (0, 1), 3: (-1, 1), 4: (-1, 0), 5: (-1,
            ↪ -1), 6:(0, -1), 7: (1, -1)}
        self.possibleActions = [0, 1, 2, 3, 4, 5, 6, 7] #vytvoreni vsech akci
        self.agentPosition = (0, 0) #nastaveni pocatecni pozice
```

V kódu 2.3 nalezneme definovanou funkci pro určení, zda je agent ve finálním stavu, dále zda jsme se nedostali mimo mapu a funkci pro nastavení agentova stavu.

Kód 2.3: Definování dalších funkcí

```
def isTerminalState(self, state):
    return state in self.stateSpacePlus and state not in self.stateSpace
def setState(self, state):
    self.agentPosition = state
def offGridMove(self, newState):
    return newState not in self.stateSpacePlus
```

Funkce `step` vezme na vstupu agentovou akci a provede výpočet následného stavu a odměny. Na výstupu vrací nový stav, odměnu a logickou hodnotu zda je nový stav konečný.

V kódu 2.4 se na začátku určí x-ová a y-ová složka agentova původního stavu, stejně jako x-ová a y-ová složka vektoru vybrané akce, ze které se určí délka kroku v rovině xy. Pro diagonální pohyb je délka $\sqrt{2}$ a pro pohyb do světových stran je 1. Dále se vypočte nový stav a ptáme se, zda odpovídá platné pozici.

- Pokud ano, tak spočteme standardní odměnu pro pohyb do nového stavu a uložíme do proměnné `rew`. Pokud je nový stav stavem konečným, pak přiřadíme do proměnné `reward` hodnotu 5, jinak proměnou `rew` a posuneme agenta do nového stavu. Na závěr vracíme nový stav, odměnu rovnu hodnotě `rew` a logickou hodnotu, zda je nový stav konečný.
- V opačném případě vedla akce mimo mapu a funkce vrací původní stav, odměnu rovnu -10 a logickou hodnotu, zda je stav konečný.

Při výpočtu proměnné `rew` se pro určení výšky v daném stavu používá metoda `iloc` z knihovny `Pandas.DataFrame`, která slouží k indexaci dat.

Kód 2.4: Funkce step

```
def step(self, action):
    agentX, agentY = self.agentPosition #urceni x-ove a y-ove slozky agentovy pozice
    actionX, actionY = self.actionSpace[action] #urceni x-ove a y-ove slozky akce

    #urceni delky kroku v rovine xy
    step_size_2D = sqrt(2) if (action % 2 == 1) else 1
    resultingState = (agentX + actionX,agentY + actionY) #urceni vysledneho stavu

    if not self.offGridMove(resultingState):
        rew = -sqrt(((self.terrain.iloc[self.n-agentY-1,agentX] -self.terrain.iloc[self.n-1-
            agentY-actionY,agentX + actionX])**2)*self.altitudeFactor**2 + step_size_2D**2)
            #vypocet odmeny pokud by krok nevedl do konecneho stavu
        reward = rew if not self.isTerminalState(resultingState) else 5
        self.setState(resultingState)   #nastaveni noveho stavu
        return resultingState, reward, self.isTerminalState(resultingState) #vraceni noveho
            stavu, odmeny, a hodnoty vyjadrujici, zda byl agent v konecnem stavu
    else:
        return self.agentPosition, -10, self.isTerminalState(self.agentPosition) #vraceni
            puvodniho stavu, odmeny a hodnoty vyjadrujici, zda byl agent v konecnem stavu
```

Ve funkci `reset` se nastaví agentův stav na počáteční a tento stav se vrátí agentovi. Funkci voláme v hlavním programu před začátkem každé epizody.

Kód 2.5: Funkce reset

```
def reset(self):
    self.setState((0, 0))   #nastaveni zacatecniho stavu
    return self.agentPosition   #vraceni agentova stavu
```

Samotná interakce agenta s prostředím se odehrává v hlavním programu, kde nejprve inicializujeme prostředí a zvolíme, kolik epizod se má provést. Pro každou epizodu nastavíme indikátor `done`, vyjadřující konec epizody na hodnotu nepravda, dále vynulujeme proměnnou `skore` a zavoláme funkci `reset`. Dokud nejsme na konci epizody, opakujeme následující kroky. Agent vybere akci, kterou následně zadá do prostředí a obdrží nový stav, odměnu a proměnnou `done`. Poté se agent učí ze zkušeností, přičte se odměna do `skore` a aktualizuje se současný stav.

Kód 2.6: Interakce agenta s prostředím

```
gamma = 1.0
env = GridWorld(16, 16, 'terrain_1.csv', 2) #inicializace prostredi
n_episodes = 100    #zvoleni poctu epizod
for i in range(n_episodes):
    done  = False
    score = 0
    observation = env.reset()
    while not done:
```

```
        action = agent.choose_action(observation)  #vyber akce
        observation_, reward, done = env.step(action)  #provedeni akce
        agent.learn(observation, state, action, reward, observation_, gamma) #ucici faze
            ↪ agenta
        score += reward
        observation = observation_ #aktualizace agentova stavu
```

Diskontní faktor `gamma` se v prostředí neobjevuje a používá ho až agent v učící fázi k odhadování výnosu. Celý kód prostředí včetně vypisování jednotlivých tras je k dispozici na GitHub repozitáři

`https://github.com/jamichy/BP_turista_RL`.

2.2 Dynamické programování (Dynamic programming)

Dynamické programování je přístup k řešení úlohy, kde původní úlohu rozložíme na podúlohy. Ty vyřešíme a uložíme jejich optimální řešení. Následně řešení podúloh použijeme ke konstrukci řešení původní úlohy [14]. Naším cílem je najít optimální strategii π_*, kterou dostaneme z optimální hodnotové nebo kvalitativní funkce pomocí rovnice

$$\begin{aligned}\pi_*(s) &= \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s')),\\ &= \operatorname*{argmax}_a q_*(s, a).\end{aligned}$$

V dynamickém programování potřebujeme znát model prostředí p k nalezení v_*, q_*. Využitím modelu p dostaneme Bellmanovy rovnice pro hodnotovou a kvalitativní funkci

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')), \tag{2.1}$$

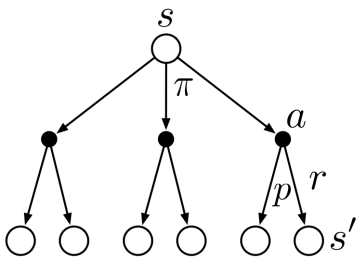
$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s', a')), \tag{2.2}$$

které platí pro každou strategii π. Rovnice (2.1) vyjadřuje vztah mezi hodnotovou funkci stavu a hodnotovou funkcí následujících stavů. Tento vztah se dá znázornit pomocí obrázku 2.2, kde bílý kruh označuje stav, černý dvojici stavu a akce. Ze stavu s agent vybere akci podle strategie π. Na obrázku 2.2 jsou ukázané tři možné akce, to odpovídá tomu, že pro konkrétní strategii π je pravděpodobnost $\pi(a|s) > 0$ jen pro tři různé akce a. Na dvojici (s, a) zapůsobí model prostředí p a prostředí agentovi vrátí některou z možností následujícího stavu s' a odměny r. Bellmanova rovnice (2.1) průměruje přes všechny možnosti, přiřazující každé čtveřici (s, a, r, s') pravděpodobnost, s jakou se mohla vyskytnout, tj. $\pi(a|s)p(s', r|s, a)$. Hodnota čtveřice (s, a, r, s') je součet diskontované hodnotové funkce ve stavu s' vzhledem ke strategii π a odměny r [1, kap. 3.5]. Pro rovnici (2.2) by diagram začínal v černém kolečku odpovídající dvojici stavu s a akce a a měl by ještě jednu úroveň listů začínající ve stavech s' a končících v dvojicích (s', a').

Pro optimalní hodnotovou a kvalitativní funkci navíc dostaneme Bellmanovy rovnice optimality

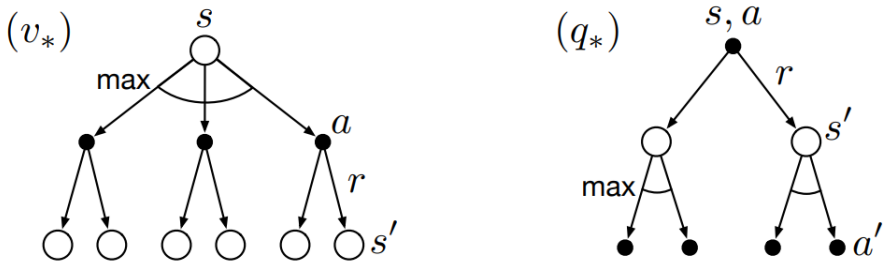
$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s')), \tag{2.3}$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} q_*(s', a')). \tag{2.4}$$



Obrázek 2.2: Bellman backup diagram pro v_π. Převzat z [1, kap. 3.5]

Vztahy (2.3) a (2.4) jdou znázornit pomocí diagramu 2.3. Pro optimální hodnotovou funkci neuvažujeme průměrování přes všechny akce jako u 2.2, ale vybereme pouze akce hladovou a_*, pro kterou nastavíme $\pi(a_*|s) = 1$.



Obrázek 2.3: Bellman backup diagram pro optimální hodnotovou a kvalitativní funkci. Převzat z [1, kap. 3.6]

2.2.1 Ohodnocení strategie (Policy evaluation)

Mějme nyní libovolnou strategii π a chceme najít její hodnotovou funkci v_π. Využijeme rovnice (2.1) k nalezení posloupnosti aproximací hodnotové funkce v_π $\{v_k\}_{k=0}^{\infty}$ zkonstruované podle

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_k(s')). \tag{2.5}$$

Hodnotu počáteční aproximace $v_0(s)$ nastavíme pro konečný stav rovnu 0, pokud je úloha epizodická. Pro ostatní stavy je hodnota libovolná. Pokud nastane $v_k(s) = v_{k+1}(s)$ pro $\forall s \in \mathcal{S}$, pak jsme nalezli hledanou hodnotovou funkci $v_\pi = v_k$. Posloupnost $\{v_k\}_{k=0}^{\infty}$ konverguje k v_π pokud je úloha epizodická, nebo $\gamma < 1$ [1, kap. 4.1]. Důkaz konvergence pro $\gamma < 1$ je k nalezení v [17] Celá procedura je shrnuta v algoritmu 2 [1, kap. 4.1].

2.2.2 Vylepšení strategie (Policy improvement)

Označme π' strategii, která se chová totožně jako strategie π až na stav s, kde vybere akci a. Pokud $q_\pi(s, a) \geq v_\pi(s)$, je strategie π' alespoň stejně dobrá jako strategie π (v případě ostré nerovnosti bude strategie π' dokonce lepší). Nyní toto vylepšení provedeme pro každý stav $s \in \mathcal{S}$, kde akce vybereme hladově a vzniklou strategii označme π' [15].

Algoritmus 2 Iterativní ohodnocení strategie pro odhad $V \approx v_\pi$

Vstup: strategie π
Parametry algoritmu: malá hodnota θ rozhodující o přesnosti odhadu
Inicializace $V(s)$ pro $\forall s \in \mathcal{S}^+$ libovolně, kromě $s' \in \mathcal{S}^+ \setminus \mathcal{S}$, pro který $V(s') = 0$
$\triangle \leftarrow \theta + 1$
Dokud $\triangle > \theta$
    $\triangle \leftarrow 0$
    Pro $s \in \mathcal{S}$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\triangle \leftarrow max(\triangle, |v - V(s)|)$
    konec Pro
konec Dokud

Tvrzení 1. O vylepšení strategie (Policy improvement theorem)
Nechť π a π' jsou libovolné deterministické strategie splňující $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ pro $\forall s \in \mathcal{S}$. Pak pro $\forall s \in \mathcal{S}$ platí $v_{\pi'}(s) \geq v_\pi(s)$.

Důkaz. V důkazu čerpáme z [1, kapitola 4.2]. Nejdříve využijeme předpokladu a následně rozepíšeme podle definice (1.9) jako $q_\pi(s, \pi'(s)) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = \pi'(s)]$. Odměna R_{t+1} je ovlivněna stavem s, výběrem akce podle strategie π' a modelem prostředí p. Výraz $\mathbb{E}_\pi[G_{t+1}]$ můžeme zapsat jako $v_\pi(S_{t+1})$, kde S_{t+1} bylo ovlivněno modelem prostředí p, předchozím stavem $S_t = s$ a akcí $\pi'(s)$. Můžeme tedy psát $\mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = \pi'(s)] = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)]$, kde střední hodnota $\mathbb{E}$ je vůči modelu prostředí p. Výraz $v_\pi(S_{t+1})$ pro pevně dané S_{t+1} je číslo. Akci A_t vybíráme podle strategie π', a proto můžeme původní výraz zapsat jako $\mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$. Výše popsaný postup používáme opakovaně, až nakonec dojdeme k $v_{\pi'}(s)$:

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] = \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] = \cdots = \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\ &\leq \cdots \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots | S_t = s] \overset{\text{dle (1.7)}}{=} v_{\pi'}(s). \end{aligned}$$

□

Za povšimnutí stojí, že zlepšení je monotónní, tj. $\forall s \in \mathcal{S}$ platí $v_{\pi'}(s) \geq v_\pi(s)$. Pokud navíc platilo pro nějaký stav $s \in S$, že $q_\pi(s, \pi'(s)) > v_\pi(s)$, pak bude i $v_{\pi'}(s) > v_\pi(s)$.

2.2.3 Iterování strategie (Policy iteration)

Získali jsme tedy lepší strategii π', která vznikla pomocí v_π. Ohodnocením strategie π' dostaneme hodnotovou funkci $v_{\pi'}$, ze které vylepšením dostáváme π''. Tento proces opakujeme a dostáváme posloupnost

$$\pi \xrightarrow{O} v_\pi \xrightarrow{V} \pi' \xrightarrow{O} v_{\pi'} \xrightarrow{V} \pi'' \xrightarrow{O} \ldots \xrightarrow{V} \pi_* \xrightarrow{O} v_{\pi_*},$$

kde $\xrightarrow{O}$ značí ohodnocení strategie a $\xrightarrow{V}$ její vylepšení. Tento proces se nazývá iterování strategie [15]. Pro konečné Markovské procesy nalezneme optimální strategii maximálně po $|\mathcal{S}|^{|\mathcal{A}|}$ krocích, protože $|\mathcal{S}|^{|\mathcal{A}|}$ je počet všech deterministických strategií. Postup shrnuje algoritmus (3) [1, kap. 4.3].

Algoritmus 3 Iterování strategie pro odhad $\pi \approx \pi_*$

Parametry algoritmu: malá hodnota θ rozhodující o přesnosti odhadu
1. Inicializace
Inicializace $V(s)$ pro $\forall s \in \mathcal{S}^+$ libovolně, kromě $s' \in \mathcal{S}^+ \setminus \mathcal{S}$, pro který $V(s') = 0$
Inicializace $\pi(s)$ pro $\forall s \in \mathcal{S}$
2. Ohodnocení strategie
$\triangle \leftarrow \theta + 1$
Dokud $\triangle > \theta$
    $\triangle \leftarrow 0$
    Pro $a \in \mathcal{S}$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\triangle \leftarrow max(\triangle, |v - V(s)|)$
    konec Pro
konec Dokud
3. Vylepšení strategie
stabilniStrategie $\leftarrow$ pravda
Pro všechna $s \in \mathcal{S}$
    staraAkce $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    Pokud staraAkce $\neq \pi(s)$, pak stabilniStrategie $\leftarrow$ nepravda
konec Pro
Pokud stabilniStrategie, pak ukonči a vrať $V \approx v_\pi$, jinak jdi na krok 2

2.2.4 Iterování hodnotové funkce (Value Iteration)

Alternativním přístupem, jak najít optimální hodnotovou funkci v_*, je iterování strategie pouze s jedním krokem ohodnocení strategie [15]. Podobně jako iterace (2.5) vychází z rovnice (2.1), modifikujeme (2.3) a konstruujeme posloupnost odhadů hodnotové funkce v_* $\{v_k\}_{k=0}^{\infty}$ pomocí

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_k(s')), \tag{2.6}$$

kde hodnotu počáteční aproximace $v_0(s)$ nastavíme pro konečný stav rovnu 0, pokud je úloha epizodická. Pro ostatní stavy je hodnota libovolná.

Tvrzení 2. Konvergence pro iterování hodnotové funkce
Pro libovolné počáteční v_0 posloupnost $\{v_k\}_{k=0}^{\infty}$ zkonstruovaná podle vztahu (2.6) konverguje pro $\gamma < 1$ stejně jako pro konečné úlohy.

Důkaz. V důkazu se inspirujeme [17] a dokážeme konvergenci odhadů $\{v_k\}_{k=0}^{\infty}$ pro $\gamma < 1$.

Pro potřeby důkazu definujeme Bellmanův operátor optimality (Bellman backup operator) $B^* : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ definovaný pro každou hodnotovou funkci $v = (v(s_1), \ldots, v(s_{|\mathcal{S}|})) \in \mathbb{R}^{|\mathcal{S}|}$ jako

$$B^* v = (Bv(s_1), \ldots, Bv(s_{|\mathcal{S}|}))^T,$$

kde pro každé $s \in \mathcal{S}$

$$B(v(s)) = \max_a \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a] \tag{2.7}$$

$$= \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v(s')), \tag{2.8}$$

kde $|\mathcal{S}|$ je počet všech stavů.
Definujme maximovou normu pro každé 2 strategie π_1, π_2 vztahem $\|v_{\pi_1} - v_{\pi_2}\|_\infty = \max_s |v_{\pi_1}(s) - v_{\pi_2}(s)|$. Nyní dokážeme, že Bellmanův operátor optimality je kontrakce pro $\gamma < 1$, tj. $\forall v_1, v_2 \in \mathbb{R}^{|\mathcal{S}|}$ platí

$$\|B^* v_1 - B^* v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty.$$

Pro zkrácení zápisu budeme v důkazu používat notaci $\max_a \mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})]$, čímž budeme mínit $\max_a \mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})|S_t = s, A_t = a]$. Důkaz provedeme sérií úprav a odhadů, které následně vysvětlíme. Platí

$$\begin{aligned}
\|B^* v_1 - B^* v_2\|_\infty &= \|\max_a \mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})] - \max_{a'} \mathbb{E}[R_{t+1} + \gamma v_2(S_{t+1})]\|_\infty \\
&= \|\max_a (\mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})] - \max_{a'} \mathbb{E}[R_{t+1} + \gamma v_2(S_{t+1})])\|_\infty \\
&\leq \|\max_a (\mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})] - \mathbb{E}[R_{t+1} + \gamma v_2(S_{t+1})])\|_\infty \\
&= \max_a (\|\mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})] - \mathbb{E}[R_{t+1} + \gamma v_2(S_{t+1})]\|_\infty) \\
&= \max_a (\|\sum_{r,s'} p(s', r|s, a)\gamma(v_1(s') - v_2(s'))\|_\infty) \\
&= \gamma \max_a (\|\sum_{s'} p(s'|s, a)(v_1(s') - v_2(s'))\|_\infty) \\
&\leq \gamma \|v_1 - v_2\|_\infty.
\end{aligned} \tag{2.9}$$

Interpretace kroků v (2.9) je následující. Nejprve rozepíšeme dle definice (2.7) a nyní chceme dva výrazy v maximové normě sloučit. Proto vytkneme první maximum a následně využijeme vztahu $|\mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})] - \max_{a'} \mathbb{E}[R_{t+1} + \gamma v_2(S_{t+1})]| \leq |\mathbb{E}[R_{t+1} + \gamma v_1(S_{t+1})] - \mathbb{E}[R_{t+1} + \gamma v_2(S_{t+1})]|$. Následně vytáhneme $\max_a$ z maximové normy, sloučíme oba výrazy pod střední hodnotu $\mathbb{E}$, výrazy R_{t+1} se odečtou a střední hodnotu vyjádříme dle modelu prostředí p. Nakonec vytáhneme γ z maximové normy a poslední nerovnost plyne z faktu, že pro každé a a s je suma $\sum_{s'} p(s'|s, a)$ rovna 1.

Z Banachovy věty o pevném bodě plyne, že existuje jediné v_{**} splňující $B^* v_{**} = v_{**}$. Tato hodnotová funkce splňuje Bellmanovu rovnici optimality (2.3) a proto je optimální hodnotová funkce. Navíc platí pro libovolnou hodnotovou funkci v v epizodické úloze, či nepřetržité úloze pro $\gamma < 1$, že iterativní aplikace Bellmanova zpětného operátoru na v konverguje k v_*, neboť

$$\|(B^*)^n v - v_*\|_\infty = \|(B^*)^n v - B^* v_*\|_\infty \leq \gamma \|(B^*)^{n-1} v - v_*\|_\infty \leq \cdots \leq \gamma^n \|v - v_*\|_\infty \xrightarrow{n \to \infty} 0. \tag{2.10}$$

□

Odhad chyby

Nechť máme nepřetržitou úlohu, která má odměny v absolutní hodnotě omezené konstantou R_{max}. Pak absolutní hodnota optimální hodnotové funkce vyčíslené v jakémkoliv stavu $s \in \mathcal{S}$ $v_*(s)$ je odhadnutelná [16] vztahem

$$|v_*(s)| \leq \sum_{t=0}^{\infty} \gamma^t R_t \leq \sum_{t=0}^{\infty} \gamma^k R_{max} = \frac{R_{max}}{1-\gamma}. \tag{2.11}$$

Pokud zvolíme $v_0(s) = 0$ pro $\forall s \in \mathcal{S}$ a využitím odhadů (2.10) a (2.11) dostaneme

$$\|(B^*)^n v_0 - v_*\|_\infty \leq \gamma^n \|v_0 - v_*\|_\infty \leq \frac{\gamma^n R_{max}}{1-\gamma} \qquad \forall n \in \mathbb{N}.$$

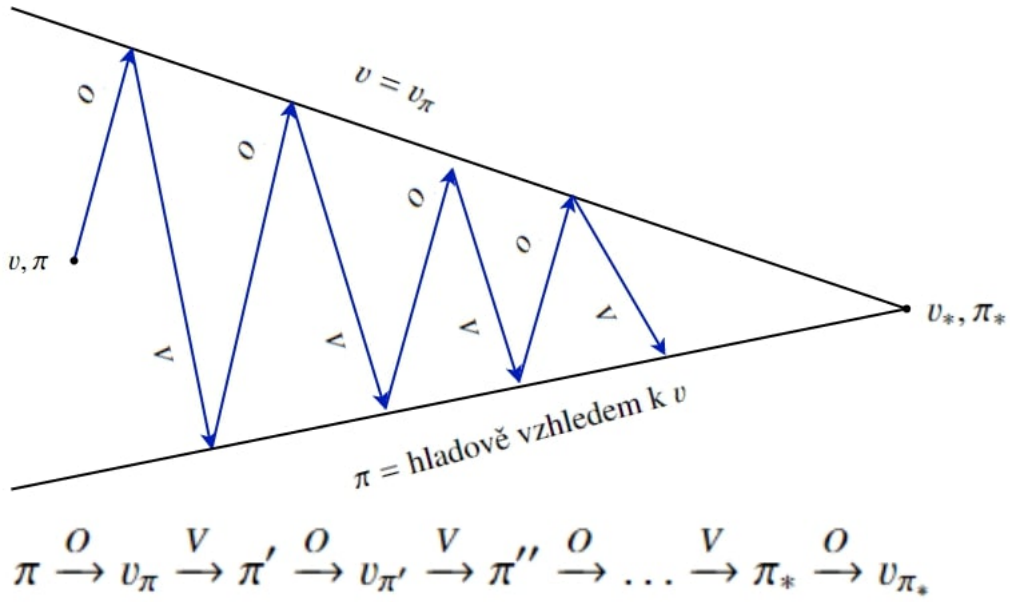
Algoritmus iterování hodnotové funkce vypadá následovně [1, kap. 4.4]

Algoritmus 4 Iterování hodnotové funkce pro odhad $\pi \approx \pi_*$

Parametry algoritmu: malá hodnota θ rozhodující o přesnosti odhadu
Inicializace $V(s)$ pro $\forall s \in \mathcal{S}^+$ libovolně, kromě $s' \in \mathcal{S}^+ \setminus \mathcal{S}$, pro který $V(s') = 0$
$\triangle \leftarrow \theta + 1$
Dokud $\triangle > \theta$
    $\triangle \leftarrow 0$
    Pro $s \in \mathcal{S}$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\triangle \leftarrow max(\triangle, |v - V(s)|)$
    konec Pro
konec Dokud
Výstup: deterministická strategie $\pi \approx \pi_*$, pro niž platí $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

2.2.5 Zobecnění iterování strategie (Generalized policy iteration)

Metoda iterování strategie se skládá ze dvou navazujících procesů. Prvním z nich vypočte hodnotovou funkci v_π, která je konzistentní se strategií π (ohodnocení strategie) a druhý vybere hladovou akci vzhledem k v_π (vylepšení strategie). V iterování strategie až po dokončení jednoho procesu začne probíhat druhý. V metodě iterování hodnotové funkce se po jedné iteraci ohodnocení strategie už vybere hladová akce a přesto metoda konverguje pro konečné úlohy i pro $\gamma < 1$. To nás přivádí k myšlence, že ohodnocení strategie nemusí být dokonalé. Pokud se procesy ohodnocení a vylepšení strategie ustálí, pak jsme nalezli optimální hodnotovou funkci v_* a optimální strategii π_*. Tento koncept, kdy necháme mezi sebou nezávisle interagovat procesy ohodnocení strategie a její vylepšení se označuje jako zobecnění iterování strategie a budeme ho používat k popisu následujících metod řešení [1, kap. 4.6].



Obrázek 2.4: Znázornění zobecnění iterování strategie. Inspirováno [1, Obr. v kap. 4.6].

2.3 Monte Carlo

Ve zpětnovazebním učení se nám často nestane, že má agent k dispozici dynamiku prostředí a může využít přímo metody dynamického programování. Jedním ze způsobů jak přesto nalézt aproximaci π_* je využití metod Monte Carlo (MC). Metody MC se spoléhají na opakovaný náhodný výběr k vytvoření velkého množství možných výsledků, které jsou následně použity k aproximaci chování nebo statistik systému, který se zkoumá. Tyto metody jsou zvláště užitečné v případech, kdy je obtížné nebo nemožné získat analytická nebo deterministická řešení. Více o metodách MC lze nalézt v [13].

2.3.1 Odhadování hodnotové funkce

Aproximaci hodnotové funkce v_π získáme průměrováním výnosů ze vzorků, vzniklých interakcemi agenta s prostředím podle strategie π. Aby byl výnos dobře definován, omezíme se pouze na epizodické úlohy. Když byl agent ve stavu s a provedl akci a, jeho výnos byl ovlivněn strategií, podle které jednal a taky dynamikou prostředí p. Teprve až po konci epizody můžeme spočítat dosažený výnos G_t pro $\forall t \in \{0, 1, \ldots, T-1\}$ jako $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$ [1, kap. 5].

Při generování epizody se výskyt stavu s označuje jako návštěva s. Samozřejmě se může ve vygenerované epizodě objevit návštěva s několikrát. Podle toho, zda budeme do průměru výnosu brát pouze výnos po první návštěvě s, nebo výnos po každé návštěvě, dělíme algoritmy na first-visit MC nebo every-visit MC [1, kap. 5.1].

Stejně jako u problému mnohorukého bandity chceme mít algoritmus využívající efektivně paměť. Proto zavedeme pro variantu první návštěvy proměnou $\#(s)$ pro každé $s \in \mathcal{S}$ vyjadřující počet epizod, ve kterých se vyskytla návštěva s a označme průměr výnosů pro stav s jako $V(s)$. Pokud se v nové epizodě vygenerované podle strategie π objeví návštěva S_t a v předchozích části se S_t neobjevilo, spočítáme výnos G_t ze stavu S_t a upravíme $\#(S_t)$ a $V(S_t)$ jako

$$\#(S_t) \leftarrow \#(S_t) + 1,$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{\#(S_t)}(G_t - V(S_t)).$$

Přístup Monte Carlo k ohodnocení hodnotové funkce ve variantě první návštěvy je shrnut v algoritmu 5 [1, kap. 5].

Algoritmus 5 První návštěva Monte Carlo pro odhad $V \approx v_\pi$

Vstup: strategie π k ohodnocení
Inicializace: $V(s) \leftarrow 0$ pro $\forall s \in \mathcal{S}$
Inicializace: $\#(s) \leftarrow 0$ pro $\forall s \in \mathcal{S}$
Opakuj do nekonečna
  Generuj epizodu podle π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Opakuj pro každý krok epizody, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Pokud se S_t neobjevil v: $S_0, S_1, \ldots, S_{t-1}$, pak
      $\#(S_t) \leftarrow \#(S_t) + 1$
      $V(S_t) \leftarrow V(S_t) + \frac{1}{\#(S_t)}(G - V(S_t))$
    konec Pokud
  konec Opakuj
konec Opakuj

V případě every-visit MC algoritmu vypustíme podmínku: "Pokud se S_t neobjevila v: $S_0, \ldots, S_{t-1}$, pak" Podle zákona velkých čísel aritmetický průměr výnosů následující po stavu s konverguje ke skutečné hodnotě $v_\pi(s)$, pokud počet epizod obsahující návštěvu s jde do nekonečna pro obě varianty návštěv [1, kap. 5.1].

Když poté budeme chtít opakovat vylepšení strategie a její ohodnocení, chtěli bychom už použít nalezený odhad hodnotové funkce pro předcházející strategii pro odhad hodnotové funkce nové strategie, abychom nemuseli začínat s prvotním odhadem $V(s) = 0$ pro $\forall s \in \mathcal{S}$. Proto budeme počítat odhad pomocí exponenciálního průměrovaní vztahem

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)),$$

kde $\alpha \in (0, 1]$. Obvykle používáme α blízko nule. Výhodou exponenciálního průměrování je, že přiřazuje větší váhu aktuálním výnosům a váha historických výnosů je zmenšována koeficientem $(1 - \alpha)$.

K vyřešení úloh se inspirujeme myšlenkou zobecnění iterování strategie, kde jsme nechali mezi sebou nezávisle interagovat ohodnocení strategie a její vylepšení. U dynamického programování jsme použili k ohodnocení strategie π hodnotovou funkci v_π. Metodou MC jsme tuto hodnotovou funkci také nalezli, ale pro výběr hladové akce potřebujeme znát dynamiku prostředí p [1, kap. 5], jelikož vylepšení strategie provádíme hladově vztahem

$$\pi'(s) \leftarrow \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')). \tag{2.12}$$

2.3.2 Odhadování kvalitativní funkce

Pokud bychom znali pro danou strategii π kvalitativní funkci q_π, pak bychom vylepšenou strategii získali snadno vztahem

$$\pi'(s) \leftarrow \underset{a}{\text{argmax}} q_\pi(s, a).$$

Proto se zaměříme na odhadování kvalitativní funkce. Kvalitativní funkce ve stavu s a akci a vzhledem ke strategii π - $q_\pi(s, a)$ je očekávaný výnos ze stavu s, kde agent provede akci a a posléze se řídí strategií π. Při generování epizody se výskyt stavu s následovaný akcí a označuje jako návštěva (s, a). V odhadování kvalitativní funkce probíhá všechno stejně jako u odhadování hodnotové funkce, až na to, že nyní používáme návštěvu (s, a) místo návštěvy s. Pro úprava odhadu kvalitativní funkce $Q \approx q_\pi$ proto platí

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t)). \tag{2.13}$$

2.3.3 ϵ-hladové Monte Carlo

V sekci věnované iterování strategie v dynamickému programování jednal agent pouze podle hladové strategie. Pokud by agent v MC jednal pouze hladově, počet některých návštěv (s, a) by nešel do nekonečna a měli bychom dobrý odhad q_π jenom pro některé návštěvy (s, a). Jedním ze způsobů jak zaručit, že počet všech možných návštěv půjde do nekonečna, je použití ϵ-hladového vybírání [1, kap. 5.4].

Definice 2.1. Nechť máme strategii π a známe její kvalitativní funkci q_π. Řekneme, že strategie π' jedná ϵ-hladově vzhledem k q_π, pokud pro každý stav $s \in \mathcal{S}$ a každou nehladovou akci vzhledem k q_π $a \in \mathcal{A}(s)$ je $\pi'(a|s) = \frac{\epsilon}{|\mathcal{A}(s)|}$ a pro hladovou akci vzhledem k q_π je $\pi(a|s) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$.

Definice 2.2. Řekneme, že strategie π je ϵ-měkká (ϵ-soft), pokud pro $\forall s \in \mathcal{S}$ a pro $\forall a \in \mathcal{A}(s)$ je $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$.

Tvrzení 3. O vylepšení ϵ-hladové strategie
Nechť π je libovolná ϵ-měkká strategie a π' jedná ϵ-hladově vzhledem k π. Pak $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ pro $\forall s \in \mathcal{S}$.

Důkaz. Důkaz přebíráme z [1, kap. 5.3]. Tvrzení dokážeme sérií úprav a odhadů, které následně vysvětlíme. Platí:

$$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s, a) = \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \epsilon) \max_a q_\pi(s, a) \tag{2.14}$$

$$= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \left(\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|} \right) \max_a q_\pi(s, a) \tag{2.15}$$

$$\geq \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \left(\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|} \right) q_\pi(s, a) \tag{2.16}$$

$$= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a|s) q_\pi(s, a) - \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) = v_\pi(s). \tag{2.17}$$

V rovnostech (2.14) využijeme faktu, že strategie π' jedná ϵ-hladově vzhledem k π. Využitím vztahů $\sum_a \pi(a|s) = 1$ a $\sum_{a \in \mathcal{A}(s)} \frac{\epsilon}{|\mathcal{A}(s)|} = \epsilon$ přepíšeme výraz $(1 - \epsilon)$ v druhém členu rovnice (2.14) a dostáváme

vztah (2.15). V nerovnosti (2.16) použijeme předpoklad ϵ-měkké strategie, tj. $\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|} \geq 0$ pro $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ a nahradíme maximum kvalitativní funkce kvalitativní funkcí ve stavu s a příslušné akci a. Roznásobením a následným sečtením členů v rovnici (2.16) dostáváme vztah (2.17). □

Následně použijeme tvrzení 1 a dostáváme $v_{\pi'}(s) \geq v_{\pi}(s)$ pro $\forall s \in \mathcal{S}$.

Víme, jakým způsobem můžeme pro každou ϵ-měkkou strategii π najít kvalitativní funkci q_π a také, jak ji ϵ-hladově vylepšit. Tyto 2 procesy spojíme ve smyslu zobecnění iterování strategie, viz 2.2.5, a můžeme psát náš první kód 2.7. V ϵ-hladové strategii ve stavu $s \in \mathcal{S}$ se agent chová nehladově s pravděpodobností $\frac{\epsilon(\mathcal{A}(s)-1)}{\mathcal{A}(s)}$. Abychom dostali optimální strategii π_*, která je hladová, budeme postupně snižovat ϵ k nule a z ϵ-hladové strategie se stane hladová.

Díky tomu, že prostory stavů $\mathcal{S}$ a akcí $\mathcal{A}$ jsou konečné, můžeme v kódu 2.7 ukládat pro každou návštěvu (s, a) hodnotu kvalitativní funkce do tabulky (v Pythonu se jedná o datový typ slovník) Q. Metody používající tuto reprezentaci se označují jako tabulkové.

Kód 2.7: Implementace ϵ-hladového MC v modifikaci každé návštěvy

```
from environment import GridWorld
import numpy as np
import random

def maxAction(Q, state):    #funkce pro urceni hladove akce
    values = np.array([Q[state,a] for a in env.possibleActions])
    action = np.argmax(values)
    return action

if __name__=='__main__':
    m = 16
    n = 16
    env = GridWorld(m, n, 'terrain_1.csv', 2)   #vytvoreni prostredi
    n_episodes = 50000
    epsilon = 1.0
    gamma = 1.0
    alpha = 0.1

    MC_history_score = []
    Q = {}
    for state in env.stateSpacePlus:
        for action in env.possibleActions:
            Q[state, action] = 0
    for i in range(n_episodes):
        done  = False
        score = 0
        states_actions = []
        rewards = []
        observation = env.reset()  #nastaveni prostredi do pocatecniho stavu
        while not done:
            rand = np.random.random()
            if rand < epsilon:
            action = random.choice(env.possibleActions)    #vyber nahodne akce
```

```
            else:
                action = maxAction(Q, observation)  #vyber hladove akce
            states_actions.append((observation, action))  #pridani stavu a akce
            observation_, reward, done = env.step(action) #zadani akce do prostredi, dostavame
                ↪  novy stav, odmenu a done
            rewards.append(reward)
            score += reward
            observation = observation_
        G = 0
        while len(rewards)>0:
            G = rewards.pop(-1) + gamma*G #spocitani vynosu
            state_action = states_actions.pop(-1)
            Q[state_action] = Q[state_action] +  alpha*(G -Q[state_action])   #uprava
                ↪ kvalitaticni funkce
        epsilon -= 1/n_episodes    #snizeni epsilon
        MC_history_score.append(score)
```

2.4 Temporální diference + Bootstrapping (Temporal difference + Bootstrapping)

Temporální diference (TD) kombinuje myšlenky z MC metod a dynamického programování. Stejně jako u metod MC agent nezná model prostředí p a učí se ze zkušeností, které získal vzájemnou interakcí s prostředím. Z dynamického programování přebírá myšlenku úpravy odhadu hodnotové funkce v konkrétním stavu pomocí odhadu hodnotové funkce v ostatních stavech, tzv. bootstrapping [1, kap. 6].

Bootstrapping je proces následného zpřesňování, při kterém se staré odhady hodnoty zpřesňují novými aktualizacemi. Bootstrapping řeší problém výpočtu konečné hodnoty, když známe pouze postupné výpočty mezihodnot [3, kap. 2.2].

2.4.1 Odhad hodnotové/kvalitativní funkce

V metodách MC čeká agent do konce epizody, aby mohl určit výnos G_t ze stavu S_t a následně použije výnos dosažený v epizodě G_t jako cílovou hodnotu pro $V(S_t)$ a nový odhad $V(S_t)$ nastaví jako

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]. \tag{2.18}$$

My se zaměříme pouze na TD(0), někdy nazývaná 1-kroková TD. Nejprve si rozepíšeme výnos G_t jako $R_{t+1} + \gamma G_{t+1}$ a hodnotu G_{t+1} nahradíme $V(S_{t+1})$, kde odhad $V(S_{t+1})$ byl už určen v čase t. Výraz $R_{t+1} + \gamma V(S_{t+1})$ nám vyjadřuje odhad $V(S_t)$ pro tuto konkrétní epizodu a my ho využijeme jako cílovou hodnotu stejně jako u MC [1, kap. 6.1].
Využitím (2.18) dostáváme vztah

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]. \tag{2.19}$$

V TD(0) nemusí agent čekat na konec epizody, aby znal odhad výnosu G_t. Stačí mu být v čase $t+1$. Následně provede změnu hodnotové funkce, případně změnu strategie. Díky okamžitému učení je TD(0) užitečný i pro nepřetržité úlohy, nebo pro epizodické úlohy s dlouhými epizodami. Existují i n-krokové varianty TD, kde až po n krocích upravujeme odhad hodnotové funkce. Algoritmus 6 popisuje způsob výpočtu odhadu hodnotové funkce TD(0), pokud můžeme začínat v libovolném stavu.

Algoritmus 6 Tabulkové TD(0) pro odhad v_π

Vstup: strategie π k ohodnocení
Parametry algoritmu: velikost kroku $\alpha \in (0, 1]$
Inicializace $V(s)$ pro $\forall s \in \mathcal{S}^+$ libovolně, kromě $s' \in \mathcal{S}^+ \setminus \mathcal{S}$, pro který $V(s') = 0$

Opakuj pro každou epizodu
  Náhodná inicializace počátečního stavu S
  Opakuj pro každý krok v epizodě
    $A \leftarrow$akce vybraná strategií π ve stavu S
    Proveď akci A, pozoruj R, S'
    $V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$
  konec Opakuj
konec Opakuj

Některé úlohy mají požadovanou vlastnost, že si můžeme určit počáteční stav. Pokud nám úloha nedovoluje začínat v libovolném stavu, přepíšeme implementaci do ϵ-hladové, abychom asymptoticky navštívili všechny stavy nekonečně mnohokrát a měli tak zajištěnou konvergenci.

Podívejme se ještě podrobněji na výraz $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$. Udává nám rozdíl (diferenci) mezi původním odhadem hodnotové funkce ve stavu S_t a odhadem hodnotové funkce ve stavu S_t pro tuto epizodu. Tento rozdíl se označuje jako TD chyba a je definovaná vztahem

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t).$$

δ_t závisí na čase a jsme ji schopni vypočítat až v čase $t + 1$, kdy známe stav S_{t+1} [1, kap. 6.1]. Úměrně δ_t se mění $V(S_t)$. TD chybu δ_t následně škálujeme parametrem $\alpha \in (0, 1]$. Pro $\alpha = 1$ nám přejde (2.19) na $V(S_t) \leftarrow R_{t+1} + \gamma V(S_{t+1})$.

Pokud se nebude měnit V v průběhu epizody, můžeme napsat chybu MC výnosu jako součet TD chyb [1, kap. 6.1], tj.

$$\begin{aligned} G_t - V_t &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) = \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &= \delta_t + \gamma\delta_t + \gamma^2(G_{t+2} - V(S_{t+2})) = \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V(S_T)) \\ &= \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) = \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k. \end{aligned}$$

Stejně jako u metod MC, zde nemáme k dispozici model prostředí p a hodnotová funkce nám při vylepšení strategie nepomůže. Proto budeme odhadovat kvalitativní funkci q_π a následně provedeme vylepšení strategie vzhledem k q_π podle zobecnění iterování strategie, viz 2.2.5. Odhad kvalitativní funkce q_π označme Q.

2.4.2 Sarsa

Jako první algoritmus využívající TD(0) k odhadování kvalitativní funkce uvedeme algoritmus Sarsa. Jeho název vznikl z komponent, které používá ke změně odhadu kvalitativní funkce - $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. Záměnou $V(S_t)$, $V(S_{t+1})$ ve vztahu (2.19) za $Q(S_t, A_t)$, $Q(S_{t+1}, A_{t+1})$ dostáváme nový odhad

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{2.20}$$

Tuto změnu odhadu děláme pokaždé, když S_t není konečný stav. Pro stav S_{t+1}, který je konečný, pokládáme $Q(S_{t+1}, A_{t+1}) = 0$. Odhady Q konvergují ke skutečné q_π pro zafixovanou strategii π, pokud počet navštívení každého stavu s a akce a jde do nekonečna. To můžeme zajistit ϵ-hladovou strategií.

Sarsa konverguje k optimální strategii π_* a optimální hodnotové funkci v_*, pokud počet návštěv všech dvojic (s, a) jde do nekonečna a ϵ-hladová strategie vzhledem ke Q se limitně stane hladovou [1, kap. 6.3,6.4].

Výše uvedený postup přepíšeme do algoritmu 7. Implementace algoritmu Sarsa pro úlohu popsanou v části 2.1 je k nalezení na adrese

`https://github.com/jamichy/BP_turista_RL/tree/main/Sarsa`

Algoritmus 7 Sarsa v ϵ-hladové implementaci

Parametry algoritmu: velikost kroku $\alpha \in (0, 1]$, malé $\epsilon > 0$
Inicializace $Q(s, a)$ pro $\forall s \in \mathcal{S}^+$ libovolně, kromě $s' \in \mathcal{S}^+ \setminus \mathcal{S}$, pro který $Q(s', \cdot) = 0$

Opakuj pro každou epizodu
  Inicializace počátečního stavu S
  $A \leftarrow$akce vybraná ve stavu S strategií, která je ϵ-hladová vzhledem k Q
  Opakuj pro každý krok v epizodě
    Proveď akci A, pozoruj R, S'
    $A' \leftarrow$akce vybraná ve stavu S' strategií, která je ϵ-hladová vzhledem k Q
    $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$
    $A \leftarrow A', S \leftarrow S'$
  konec Opakuj
  Zmenši trochu ϵ
konec Opakuj

2.4.3 Off-policy

Do této doby byly všechny popsané metody tzv. on-policy. Jedna strategie byla použita pro výběr akcí a následně ta samá strategie byla ohodnocena hodnotovou či kvalitativní funkcí. Nicméně strategie pro generování epizody byla prozkoumávající a tedy určitou část času se agent choval neoptimálně (např. ϵ-hladově). Toto chování se negativně promítlo do ohodnocení hodnotové, respektive kvalitativní funkce. Chtěli bychom mít nějaký způsob, jak ohodnotit libovolnou strategii (např. hladovou), i když data pro trénování nepochází z libovolné strategie (např. ϵ-hladové).

Předpokládejme proto dvě strategie. Strategii, podle které se budeme chovat (behaviour policy), označme π_b. Bude generovat epizodu a může být více prozkoumávající. Označme π_t cílovou strategii, kterou budeme ohodnocovat. Tyto metody, kde máme dvě strategie, z nichž jednu používáme pro generování epizody a druhou pro odhadování hodnotové, respektive kvalitativní funkce, se označují jako off-policy metody.

Předpoklad pokrytí (coverage)

Za účelem využití epizod generovaných strategií π_b pro odhad hodnotové funkce π_t, vyžadujeme, aby pro každý stav s a akci a, pro které $\pi_t(a|s) > 0 \implies \pi_b(a|s) > 0$ [1, kap. 5.5]. Slovně řečeno požadujeme, aby v každém stavu pro každou akci, která by se vybrala cílovou strategií π_t, se akce taky vybrala strategii určující chování π_b.

Některé off-policy algoritmy jsou založené na výběru důležitostí vzorků (importance sampling), což je způsob, díky kterému můžeme vypočítat střední hodnotu podle jedné distribuce, když máme data z druhé distribuce [1, kap. 5.5]. Jedním ze zástupců off-policy algoritmů využívající poměr důležitosti vybraných vzorků je n-step Expected Sarsa, kterou můžeme nalázt v [1, kap. 7.3].

Off-policy metody jsou obvykle více komplikované a pomaleji konvergují, ale jsou schopné se učit na datech vygenerovaných jinou strategií a jsou mnohem obecnější [1, kap. 5.5].

2.4.4 Q-učení (Q-learning)

Nejznámějším off-policy algoritmem je Q-učení, které představil Christopher J. C. H. Watkins v roce 1989. Shromažďuje informace z prozkoumaných tahů a upravuje odhad optimální kvalitativní funkce ve stavu S_t a akci A_t, jako kdyby ve stavu S_{t+1} byla vybraná hladová akce, i když ve skutečnosti agent provede nehladovou akci, tj. úprava odhadu probíhá podle vztahu [3, str. 53,54]

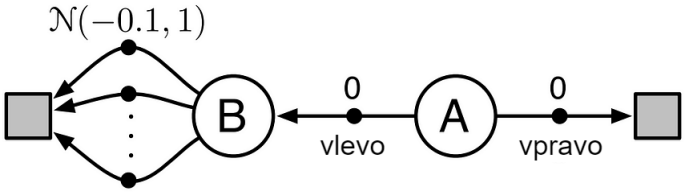
$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \\ &= Q(S_t, A_t) + \alpha(R_{t+1} + Q(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a)) - Q(S_t, A_t)). \end{aligned} \tag{2.21}$$

Jedná se proto o off-policy algoritmus, čímž se liší od algoritmu Sarsa (část 2.4.2), kde se upravuje odhad kvalitativní funkce ve směru toho, co agent udělá v dalším kroku podle aktuální strategie. Pokud u algoritmu Sarsa v ϵ-hladové implementaci zmenšíme ϵ na nulu, pak agent v Sarse jedná hladově a vztah (2.20) přejde v (2.21).

Implementace Q-učení pro úlohu popsanou v části 2.1 je k nalezení na adrese

`https://github.com/jamichy/BP_turista_RL/tree/main/Q-learning`

2.4.5 Maximalizační vychýlení (Maximization bias)



Obrázek 2.5: Znázornění úlohy pro vysvětlení maximalizačního vychýlení. Inspirováno [1, Obr. 6.5].

Představme si úlohu na obrázku 1.3, kde ve stavu A máme na výběr dvě akce - vpravo a vlevo. Když agent ve stavu A vybere akci vpravo, tak se přemístí do konečného stavu a dostane odměnu 0. Pokud zvolí ve stavu A akci vlevo, přesune se do stavu B a též obdrží odměnu 0. Ze stavu B vedou všechny akce do konečného stavu, přitom agent obdrží odměnu, která je odvozena z Normální distribuce se střední hodnotou -0.1 a rozptylem 1. Proto je očekávaný výnos při akci doleva -0.1 při hodnotě $\gamma = 1$ a optimální strategie je tedy vždy ve stavu A jít vpravo. Z důvodu vysokého rozptylu dostane agent při výběru akce ze stavu B někdy odměnu kladnou a někdy zápornou. V Q-učení bude trvat nějaký čas, než zjistíme, že akce vlevo je horší, protože když vybereme akci vlevo a ve stavu B vybereme náhodnou akci a', existuje kladná pravděpodobnost, že dostaneme pozitivní odměnu. Pak bude $Q(B, a') > 0$ a důsledkem bude, že $Q(A, vlevo) > 0$ a ve stavu A bude hladová akce rovna *vlevo*, místo skutečné hladové akci *vpravo* vzhledem k optimální strategii. Tomuto jevu, kdy náš odhad kvalitativní funkce systematicky převyšuje hodnoty optimální kvalitativní funkce, se říká maximalizační vychýlení [1, kap. 6.7].

2.4.6 Dvojité Q-učení (Double Q-learning)

V Q-učení nastává maximalizační vychýlení z toho důvodu, že používáme pro úpravu odhadu jednu sadu odhadů Q pro dvě věci. Ve členu $Q(S_{t+1}, \underset{a}{\text{argmax}} Q(S_{t+1}, a))$ v (2.21) vybíráme vzhledem k Q hladovou akci a následně ji ohodnocujeme podle Q. Pokud bychom měli pro situaci na obrázku 2.5 dvě sady nezávislých odhadů Q_1 a Q_2, pak je velká pravděpodobnost, že pokud pro nějakou akci a platilo $Q_1(B, a) > 0$, pak $Q_2(B, a) < 0$. Proto jako vylepšení Q-učení vzniklo Dvojité Q-učení [1, kap. 6.7], kde používáme dvě sady nezávislých odhadů Q_1 a Q_2. S pravděpodobností $\frac{1}{2}$ upravíme odhad Q_1 podle

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha(R_{t+1} + Q_2(S_{t+1}, \underset{a}{\text{argmax}} Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)) \tag{2.22}$$

a s pravděpodobností $\frac{1}{2}$ upravíme odhad Q_2 podle

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha(R_{t+1} + Q_1(S_{t+1}, \underset{a}{\text{argmax}} Q_2(S_{t+1}, a)) - Q_2(S_t, A_t)). \tag{2.23}$$

Ve výpisu kódu 2.8 nalezneme implementaci Dvojitého Q-učení. Vynechali jsme část obsahující naimportování knihoven a prostředí, dále počáteční inicializaci proměnných a definici funkce `maxAction`, neboť bychom jen s malými úpravami pozměnili výpis kódu 2.7. Stejně jako v kódu 2.7 reprezentujeme kvalitativní funkci pomocí tabulky.

Kód 2.8: Implementace Dvojitého Q-učení

```
for i in range(n_episodes):
    done  = False
    score = 0
    observation = env.reset()
    while not done:
        rand = np.random.random()
        if rand < epsilon:
            action = random.choice(env.possibleActions)
        else:
            action = maxAction(Q_1, observation)
            observation_, reward, done = env.step(action)
        score += reward

        #ucici faze
        rand = np.random.random()
        if rand <= 0.5:
            max_next_action = maxAction(Q_1, observation_)
            Q_1[observation, action] = Q_1[observation, action] + alpha*(reward + gamma* Q_2[
                ↪ observation_, max_next_action]- Q_1[observation, action])
        else:
            max_next_action = maxAction(Q_2, observation_)
            Q_2[observation, action] = Q_2[observation, action] + alpha*(reward + gamma* Q_1[
                ↪ observation_, max_next_action]- Q_2[observation, action])
        observation = observation_
    epsilon -= 1/n_episodes
```

Celková implementace Dvojitého Q-učení pro ukázkovou úlohu je k nalezení na

`https://github.com/jamichy/BP_turista_RL/tree/main/Double%20Q-learning`

2.5 Aproximace kvalitativní funkce pomocí neuronové sítě

V předchozí části jsme v kódech ukládali pro každý stav $s \in \mathcal{S}$ a akci $a \in \mathcal{A}$ odhad kvalitativní funkce $Q(s, a)$. To lze dělat pouze pro úlohy, které mají prostor stavů $\mathcal{S}$ a prostor akcí $\mathcal{A}$ diskrétní a dostatečně malý. My bychom chtěli také řešit úlohy, kde prostor stavů je spojitý, případně vícedimenzionální a spojitý. Pak by nebylo paměťově možné využít předchozí postup ukládání kvalitativní funkce pro každý stav a akci. Navíc trénování by trvalo dlouho, protože potřebujeme pro konvergenci zajistit, že počet návštěv (s, a) jde do nekonečna. Pokud prostor stavů $\mathcal{S}$ je spojitý, většinu stavů nikdy agent neuvidí. Přesto bychom chtěli, aby agent měl dobrý odhad kvalitativní funkce pro návštěvu (s, a), kterou nikdy nenavštívil.

Předpokládáme, že pokud je návštěva (s', a') dostatečně blízko návštěvě (s, a), pak bude $Q(s', a')$ dostatečně blízko $Q(s, a)$. Potřebujeme tedy vytvořit aproximaci kvalitativní funkce Q, kterou lze parametrizovat a je dostatečně hladká. Zároveň chceme, aby množství parametrů přislušející dané aproximaci funkce bylo menší než $|\mathcal{S} \times \mathcal{A}|$.

Libovolnou funkci můžeme aproximovat například pomocí polynomů, radiální bazické funkce, Fourierovy řady nebo neuronové sítě [1, kap. 9.5]. V sekci 2.5.1 popíšeme neuronové sítě v učení s učitelem a dále v sekci 2.5.2 vysvětlíme konstrukci Q-neuronové sítě pro aproximaci kvalitativní funkce Q a uvedeme algoritmy založené na Q-neuronové síti.

2.5.1 Neuronové sítě v učení s učitelem

Neuronové sítě jsou vzájemně propojené umělé neurony a my se zaměříme na ty, které lze uspořádat do vrstev. Síť má obvykle vstupní vrstvu, jednu nebo více skrytých vrstev a výstupní vrstvu. Každý neuron přijímá vstupní signály, provádí s nimi matematické operace a vytváří výstupní signál. Spojení mezi neurony jsou reprezentována váhami. Tyto váhy určují sílu a vliv signálů procházejících sítí. Během procesu učení neuronová síť upravuje váhy na základě trénovacích dat tak, aby minimalizovala chybu nebo maximalizovala svůj výkon při řešení konkrétní úlohy.

Neuronová síť pro jeden neuron se označuje jako perceptron a jeho znázornění můžeme vidět na obrázku 2.6. Stejně jako v sekci 1.1.1 máme datovou sadu dvojic $\mathcal{D}_0 = \{(x_i, y_i)\}_{i=1}^K$, kde $\{x_i\}_{i=1}^K$ jsou vstupní záznamy a

$$x_i = (x_i^1, \ldots, x_i^m)$$

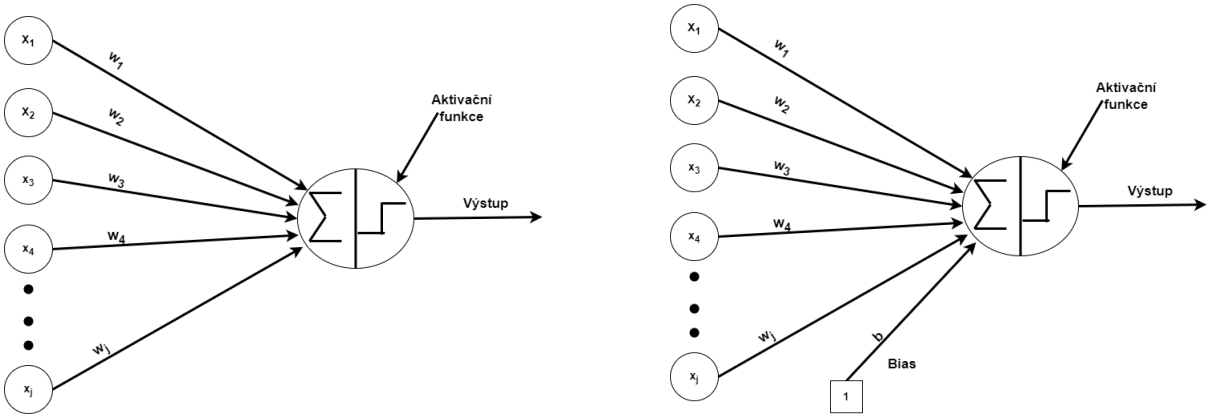
je m-složkový vektor. $\{y_i\}_{i=1}^K$ jsou požadované výstupy nabývající hodnot $\{-1, 1\}$ pro úlohu s perceptronem. Cílem je na základě trénovacích dat $\mathcal{D}$ nastavit váhy spojení tak, aby ztrátová funkce byla minimální.

Vstupní vrstva obsahuje m uzlů, které přenášejí m složek vektoru x pomocí m spojení s váhami $w = (w_1, \ldots, w_m)$ do výstupního uzlu. Ve výstupním uzlu se spočítá tzv. předaktivační hodnota pomocí vztahu $\sum_{k=1}^m w_k x^k$ a pak na tuto hodnotu zapůsobí aktivační funkce, v tomto případě funkce signum. Predikce $\hat{y}$ se dá následně zapsat jako $\hat{y} = \text{sign}(\sum_{k=1}^m w_k x^k)$. Obecně se přidává ještě neuron vychýlení (bias neuron) b, který má výstupní hodnotu 1 a váha spojení je b. Následně predikce bude $\hat{y} = \text{sign}(\sum_{k=1}^m w_k x^k + b)$ [19, str. 5,6].

Následně můžeme predikovanou hodnotu $\hat{y}$ porovnat se skutečnou hodnotou y a vypočítat chybu predikce a ztrátovou funkci. V učící fázi algoritmus prochází trénovací data a za pomocí ztrátové funkce mění váhy spojení w vztahem

$$w_k \leftarrow w_k - \alpha(y - \hat{y})x_k,$$

kde $\alpha > 0$ je vhodně nastavený parametr rychlosti učení.

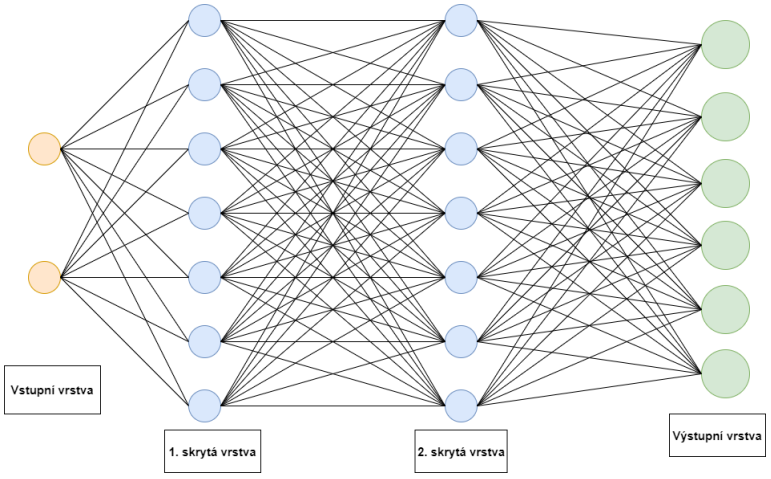


Obrázek 2.6: Perceptron bez vychýlení a s vychýlení.

Vícevrstvé Neuronové sítě

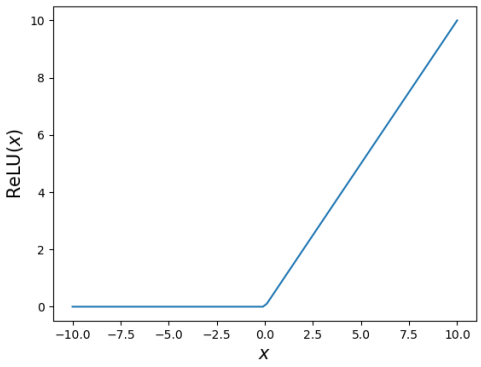
Vícevrstvé sítě mají vstupní a výstupní vrstvu jako perceptron, a navíc mají jednu či více skrytých vrstev, které se nacházejí mezi vstupní a výstupní vrstvou. Pokud mají jednu skrytou vrstvu, označují se jako mělké (shallow). V opačném případě se označují jako hluboké (deep) [3, str. 360].

Na obrázku 2.7 můžeme vidět plně propojenou dopřednou hlubokou neuronovou síť, která obsahuje vstupní vrstvu o dvou neuronech, dále dvě skryté vrstvy po sedmi neuronech a výstupní vrstvu o šesti neuronech. Plně propojená dopředná neuronová síť znamená, že z každého uzlu, který se nenachází ve výstupní vrstvě, vede spojení do kažého neuronu v další vrstvě.

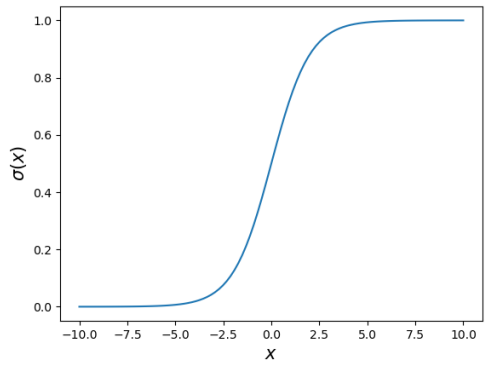


Obrázek 2.7: Znázornění plně propojení neuronové sítě s dvěma skrytými vrstvami

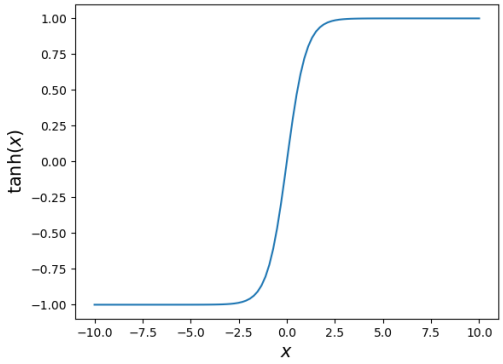
Samotná architektura neuronová sítě je charakterizováná maticí vah spojení $\mathcal{W} = [w_{ij}]$, kde w_{ij} vyjadřuje váhu spojení z uzlu i do uzlu j. Pokud w_{ij} je rovna nule, pak není v síti spojení vycházející z uzlu i do uzlu j [5, str. 9].



(a) Funkce ReLU



(b) Sigmoida



(c) Hyperbolický tangens

Obrázek 2.8: Grafy aktivačních funkcí

Výpočet hodnoty neuronu

Pokud označíme výstupní hodnotu i-tého uzlu o^i, pak předaktivační hodnota j-tého uzlu je rovna $\sum_i w_{ij} o^i + b^j$. Po aplikace aktivační funkce Φ dostáváme výstupní hodnotu uzlu j $o^j = \Phi(\sum_i w_{ij} o^i + b^j)$.

Důležitou součástí je volba aktivační funkce Φ pro každou skrytou a výstupní vrstvu. Nejběžnější jsou ReLU, sigmoida a tangens hyperbolický. Pro potřeby výpočtu gradientu ztrátové funkce potřebujeme, aby aktivační funkce byly diferencovatelné.

- ReLU (Rectified Linear Unit) je funkce, která vrací pro záporné argumenty nulu a pro nezáporné hodnotu se chová jako identita, tj.

$$\mathrm{ReLU}(x) = \max(0, x).$$

  Je diferencovatelná na $\mathbb{R} \setminus \{0\}$.

- Sigmoida je funkce jejíž obor hodnot je $(0, 1)$, dále je diferencovatelná na $\mathbb{R}$ a definovaná vztahem

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}.$$

- Hyperbolický tangens zobrazuje na $(-1, 1)$ a také je diferencovatelný na $\mathbb{R}$. Definujeme ho vztahem

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$

Grafy popsaných aktivačních funkcí jsou na obrázku 2.8.

Neuronovou síť si můžeme také představit jako funkci $f(\cdot; \theta)$, kde θ je označení vektoru všech trénovatelných parametrů. Konkrétně pro neuronovou síť na obrázku 2.7 představuje θ matici vah všech spojení $\mathcal{W}$. Predikci neuronové sítě pro vstup x označme $f(x; \theta)$. Parametr θ chceme nastavit tak, aby ztrátová funkce (např. (1.1) nebo (1.2)) na trénovacích datech byla minimální. Při trénování se jeden průchod trénovacích dat označuje jako epocha. Obvykle jsou všechna trénovací data $\mathcal{D}$ rozdělena na menší disjunktní části označené jako dávky (batch) $\mathcal{B}$. Nyní se pro každý prvek x z dávky $\mathcal{B}$ vypočte $f(x; \theta)$ a spočte se ztrátová funkce pro všechna data z dávky, kterou označme $\mathcal{L}(f(\mathcal{B}; \theta))$. Následně se provede stochastický gradientní sestup pro úpravu parametru sítě vztahem

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_\theta \mathcal{L}(f(\mathcal{B}; \theta_t)),$$

kde α je parametr rychlosti učení, θ_t je nastavení trénovatelných parametrů v iteraci t. Tato úprava se provede pro každou dávku. Síť obvykle trénujeme několik epoch [3, str. 362,363].

2.5.2 Hluboké Q-neuronové sítě

V učení s učitelem jsme minimalizovali ztrátovou funkci, která byla závislá na $(y - f(x; \theta))$, kde y byla požadovaná výstupní hodnota, která byla pro příslušné x pevná. V Q-učení jsme také při učení minimalizovali chybu mezi cílovou hodnotou $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ a momentální hodnotou $Q(S_t, A_t)$. Naše cílová hodnota v Q-učení se úpravami Q postupně mění [3, str. 76]. Zkombinujeme postupy Q-učení s hlubokými neuronovými sítěmi v učení s učitelem a dostaneme naivní hluboké Q-učení ilustrované v kódu 2.9, kde používáme knihovnu TensorFlow [38].

V kódu 2.9 dostává agent od prostředí počáteční stav. Inicializujeme tenzor `sum_sq`, který nám bude uchovávat součet druhých mocnin rozdílu cílové `target` a současné `output` hodnoty odhadu kvalitativní funkce, vyčíslené pro každý stav a akci, kterou agent v epizodě vykoná. Do tenzoru `weights` přiřadíme trénovatelné parametry sítě `Qnet()`. Dále sledujeme operace, závisející na tenzoru `weights` v bloku kódu 2.9 na řádcích 10 až 22, díky použití třídy `tf.GradientTape`. Agent interaguje s prostředím, přičemž akce vybírá ϵ-hladově vůči predikci sítě `Qnet()`. Hodnota `target` se vypočte jako součet obdržené odměny a diskontované predikce sítě `Qnet()` vyčíslené v následujícím stavu a hladové akci. Na konci epizody se spočte gradient `sum_sq` vůči parametrům neuronové sítě `weights`. Tento gradient využijeme k aktualizaci parametrů neuronové sítě `Qnet()`, tzn. `weights`, voláním metody `apply_gradients()`.

Kód 2.9: Naivní implementace Hlubokého Q-učení

```
1 import tensorflow as tf
2 ##implementace neuronove site Qnet, napriklad podle kodu \ref{lst:neural_nets_TD3} pro sit
    ↪ kritika
3 def train_qlearn(env, Qnet, alpha=0.001, gamma=1.0, epsilon=0.05
4     for i in range(n_episodes):
5         observation = env.reset() # inicializace pocatecniho stavu
6         sum_sq = tf.Variable(0)
7         done = False
8         weights = Qnet.trainable_variables;
```

```
        with tf.GradientTape(watch_accessed_variables = False) as tape:
            tape.watch(weights)
            while not done:
                observation^ = tf.convert_to_tensor(observation, dtype=tf.float32)
                action = epsilongreedy(Qnet(observation^,a)) # vyber epsilon-hladovou akci ze
                    ↪ stavu observation
                observation_, reward, done = env.step(a)
                reward^ = tf.convert_to_tensor(reward, dtype=tf.float32)
                action^ = tf.convert_to_tensor(action, dtype=tf.float32)
                observation_^ = tf.convert_to_tensor(observation_, dtype=tf.float32)

                output = Qnet.predict(observation^, action^) #predikuj hodnotu Q(observation^,
                    ↪ action^)
                target = reward^ + gamma * max(Qnet(observation_^, .))
                sum_sq = sum_sq + (target - output)**2
                observation = observation_
        grad = tape.gradient(sum_sq,weights) #spocti gradient
        Qnet.apply_gradients(zip(grad, weights)) #aplikuj gradient na Qnet
    return Qnet
```

Q-učení konvergovalo k optimální kvalitativní funkci za podmínky, že počet každé návštěvy (s, a) šel do nekonečna. Jelikož množina $\mathcal{S}$ je rozsáhlá, tak nemůžeme zaručit teoreticky konvergenci k optimální kvalitativní funkci. Navíc zkušenosti generované agentem jsou korelované, protože následující trénovací data jsou silně ovlivněná momentálními trénovacími daty. Neuronová síť, která je učena pouze na korelovaných datech konverguje do lokálního maxima. Oba členy ve ztrátové funkci ve výpisu kódu 2.9 jsou závislé na parametrech sítě `Qnet`, který je upravován a optimalizační proces se může stát nestabilní [3, str. 76-80].

2.5.3 Hluboké Q-učení

Vědci ze společnosti DeepMind přišli s řešením výše popsaných problémů a navrhli algoritmus Hlubokého Q-učení. V článku [20], [21] testovali algoritmus na hrách, které lze hrát na konzoli Atari 2600.

Pro vyřešení stability trénování se používají dvě neuronové sítě $Q(\cdot, \cdot; \theta)$ a $Q(\cdot, \cdot; \theta')$ s identickou architekturou, ale s jiným nastavením parametrů. $Q(\cdot, \cdot; \theta')$ budeme používat k ohodnocení cílové hodnoty a $Q(\cdot, \cdot; \theta)$ pro výběr ϵ-hladové akci a tuto neuronovou síť budeme trénovat. Každou pětici $(s, a, r, s', \text{done})$, kterou jsme pozorovali, uložíme do zásobníku zkušeností $\mathcal{D}$ o konečné kapacitě. done je logická proměnná vyjadřující zda je stav s' konečný. Pokud už je zásobník plný, pak uložíme současnou pětici na místo nejstarší pětice. Po každé interakci agenta s prostředím trénujeme $Q(.,.;\theta)$ na dávce dat $\mathcal{B} \subset \mathcal{D}$ o velikosti $N \in \mathbb{N}$, která je vybraná rovnoměrně a náhodně ze zásobníku. Cílem je minimalizovat chybu

$$L = \mathbb{E}_{(s,a,r,s',\text{done})\sim\mathcal{B}}[(r + \gamma \max_{a'}(\neg\text{done})Q(s', a'; \theta') - Q(s, a; \theta))^2],$$

kde zápisem $\mathbb{E}_{(s,a,r,s',\text{done})\sim\mathcal{B}}$ rozumíme výběrový průměr přes všechny záznamy v dávce $\mathcal{B}$, přičemž se jednotlivé záznamy mohou vyskytovat opakovaně. Gradient ztrátové funkce vůči θ je

$$\nabla_\theta L = -2\mathbb{E}_{(s,a,r,s',\text{done})\sim\mathcal{B}}[(r + \gamma \max_{a'}(\neg\text{done})Q(s', a'; \theta') - Q(s, a; \theta))\nabla_\theta Q(s, a; \theta)]. \tag{2.24}$$

Náhodným výběrem dávky $\mathcal{B}$ redukujeme korelaci mezi trénovacími daty. Díky použití zásobníku a trénování na dávce $\mathcal{B}$ se každá zkušenost použije několikrát a zvýšíme tak efektivitu využití pozorovaných dat. Neuronovou síť $Q(\cdot, \cdot; \theta')$ upravujeme každých C kroků, kdy nastavíme $\theta' \leftarrow \theta$ [21].

Architektura Q-neuronové sítě pro konečný počet akcí

Vstupem je vhodná reprezentace stavu $s \in S$ a výstupní vrstva obsahuje stejný počet uzlů, jako je počet akcí. Každý výstupní uzel pak odpovídá jednomu páru $Q(s, a; \theta)$, popřípadě $Q(s, a; \theta')$. Pro ohodnocení všech možných akcí pro jeden stav nám postačí pouze jeden průchod neuronové sítě. Pak již lze snadno vybrat hladovou akci ze stavu s vzhledem ke $Q(s, \cdot; \theta)$, popřípadě $Q(s, \cdot; \theta')$ [21].
Výše popsaný postup shrnuje algoritmus 8.

Algoritmus 8 Hluboké Q-učení se zásobníkem zkušeností

Parametry algoritmu: velikost kroku $\alpha \in (0, 1]$, malé $\epsilon > 0$, $C \in \mathbb{N}$
Inicializace zásobníku zkušeností $\mathcal{D}$ o konečné velikosti
Inicializace neuronové sítě pro odhad kvalitativní funkce $Q(\cdot, \cdot; \theta)$ s libovolným parametrem θ
Inicializace cílové neuronové sítě pro odhad kvalitativní funkce $Q(\cdot, \cdot; \theta'$ s parametrem $\theta' = \theta$

Opakuj pro každou epizodu
Inicializace počátečního stavu S
$A \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}} Q(S, a; \theta), & \text{s pravděpodobností } 1 - \epsilon \\ a \text{ náhodná akce}, & \text{s pravděpodobností } \epsilon \end{cases}$
Opakuj pro každý krok v epizodě
Proveď akci A, pozoruj R, S', Done
Ulož do zásobníku $\mathcal{D}$ zkušenost $(S, A, R, S'$, Done$)$
$S \leftarrow S'$
Vyber ze zásobníku $\mathcal{D}$ dávku $\mathcal{B}$ o velikosti N
Pro každé $(s_i, a_i, r_i, s'_i, \text{done}_i)$ v dávce $\mathcal{B}$
$y_i = r_i + \gamma(\neg \text{done}_i) \max_{a'} Q(s'_i, a'; \theta')$
Proveď stochastický gradientní sestup pro $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i; \theta))^2$ vzhledem k parametrům sítě θ podle rovnice (2.24)
Každých C kroků nastav $\theta' \leftarrow \theta$
konec Opakuj
konec Opakuj

2.5.4 Dvojitá hluboká Q-neuronová síť (Double deep Q-network)

Stejně jako Q-učení obsahovalo maximalizační vychýlení, viz část 2.4.5, tak i hluboké Q-učení nadhodnocuje funkční hodnoty kvalitativní funkce, které často vedou k horším strategiím. Zaměstnanci společnosti Google DeepMind přišli s řešením jak tohle vyřešit a převedli myšlenku dvojitého Q-učení pro použití v neuronových sítích [22]. Navrhli algoritmus dvojitého hlubokého Q-neuronové sítě (DDQN), kde cílovou hodnotu y_i v algoritmu 8 nastavili

$$y_i = r_i + \gamma(\neg \text{done}_i) Q(s'_i, \underset{a'}{\operatorname{argmax}} Q(s'_i, a'; \theta); \theta').$$

Pro ohodnocení cílové hodnoty y_i se použila pro výběr akce ve stavu s'_i neuronová síť $Q(\cdot, \cdot; \theta)$ a pro ohodnocení vybrané akce cílová neuronová síť $Q(\cdot, \cdot; \theta')$.

V následující části popíšeme implementaci DDQN pro úlohu turistického průvodce. Při implementaci kódu jsem čerpal z [39].Definujeme třídu `ReplayBuffer`, která obsahuje proměnné udržující v paměti historii stavů, akcí a odměn agenta. Objekt této třídy má omezenou velikost aby se zabránilo přetečení paměti. V případě, že chceme uložit záznam a zásobník zkušeností je už plný, tak příslušná funkce přepíše nejstarší záznam. Objekt dále zaznamenává, jestli daný nový stav agenta je finální.

V rámci třídy implementujeme funkce `store_transition()` a `sample_batch()`. Funkce `store_transition()` zajišťuje uložení stavu, akce a odměny agenta do příslušných proměnných zmíněné třídy. Funkce `sample_batch()` náhodně vybere dotazovaný počet záznamů a vrátí je na výstupu.

Kód 2.10: Konstrukce zásobníku zkušeností

```
class ReplayBuffer(object):
    def __init__(self, max_size, input_shape, n_actions, discrete = False):
        self.mem_size = max_size   #maximalni velikost zasobniku
        self.mem_cntr = 0  #pocitadlo zaznamu
        self.n_actions = n_actions
        self.discrete = discrete
        self.state_memory = np.zeros((self.mem_size, input_shape))
        self.new_state_memory = np.zeros((self.mem_size, input_shape))
        self.reward_memory = np.zeros(self.mem_size)
        dtype = np.int8 if self.discrete else np.float32
        self.action_memory = np.zeros((self.mem_size, n_actions), dtype = dtype)
        self.terminal_memory = np.zeros(self.mem_size)

    def store_transition(self, state, action, reward, state_, done):
        index = self.mem_cntr % self.mem_size
        self.state_memory[index] = state
        self.new_state_memory[index] = state_
        self.reward_memory[index] = reward
        self.terminal_memory[index] = 1- done
        if self.discrete:
            actions = np.zeros(self.n_actions)
            actions[action] = 1.0
            self.action_memory[index] = actions
        else:
            self.action_memory[index] = actions
        self.mem_cntr += 1

    def sample_batch(self, batch_size):
        max_mem = min(self.mem_size, self.mem_cntr)
        batch = np.random.choice(max_mem, batch_size)
        states = self.state_memory[batch]
        actions = self.action_memory[batch]
        rewards = self.reward_memory[batch]
        states_ = self.new_state_memory[batch]
        terminals = self.terminal_memory[batch]
        return states, actions, rewards, states_, terminals
```

Nyní v kódu 2.11 popíšeme vytvoření neuronové sítě za pomocí knihovny `tensorflow`, ze které naimportujeme potřebné funkce. Implementujme funkci `build_dqn()`, která bude obstarávat konstrukci neuronové sítě. Vstupními parametry funkce jsou: učící parametr `lr`, počet akcí `n_actions`, počet uzlů ve vstupní vrstvě `input_dims`, množství neuronů v první a ve druhé skryté vrstvě `fc1_dims`, respektive `fc2_dims`. Model neuronové sítě vytvoříme pomocí třídy `Sequential`, která seskupí zásobník jednotlivých vrstev. Vrstvy zhotovíme pomocí funkce `Dense`.

Kód 2.11: Konstrukce neuronové sítě v DDQN

```
from keras.layers import Dense, Activation
from keras.models import Sequential, load_model
from tensorflow.keras.optimizers import Adam

def build_dqn(lr, n_actions, input_dims, fc1_dims, fc2_dims):
    model = Sequential([Dense(fc1_dims, input_shape=(input_dims, )),
    Activation('relu'),
    Dense(fc2_dims),
    Activation('relu'),
    Dense(n_actions)])
    model.compile(optimizer = Adam(learning_rate = lr), loss = 'mse')

    return model
```

Třída `DDQNAgent` při inicializaci na vstupu obdrží proměnné: učící parametr `alpha`, diskontnní faktor `gamma`, počet možných akcí `n_actions`, míru prozkoumávání `epsilon`, velikost dávky `batch_size` a počet dimenzí stavu `input_dims`. Dále jako vstupní parametr funkci zadáváme proměnnou `epsilon_dec`, kterou v průběhu učení ve funkci `learn()` násobíme parametr míry prozkoumávání `epsilon` po každém kroku. Hodnota `epsilon` tudíž v průběhu učení klesá a její minimální hodnota je omezena hodnotou vstupní proměnné `epsilon_end`. Všechny vstupní proměnné se v inicializační funkci uloží do objektu této třídy a inicializační funkce následně zkonstruuje neuronové sítě.

Kód 2.12: Inicializace agenta v DDQN

```
class DDQNAgent(object):
    def __init__(self, alpha, gamma, n_actions, epsilon, batch_size,input_dims, epsilon_dec =
        ↪ 0.995, epsilon_end = 0.01, mem_size=100000, f_name='ddqn_model.h5',  replace_target
        ↪ =100):
        self.n_actions = n_actions
        self.action_space = [i for i in range(n_actions)]
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_dec = epsilon_dec
        self.batch_size = batch_size   #velikost davky
        self.epsilon_min = epsilon_end
        self.memory = ReplayBuffer(mem_size, input_dims, n_actions, True)  #vytvoreni
            ↪ zasobniku zkusenosti
        self.model_file = f_name
        self.replace_target = replace_target   #hodnota vyjadrujici, po kolika iteracich
            ↪ trenovani se maji zkopirovat parametry site z self.q_eval do self.q_target
```

```
13          self.q_eval = build_dqn(alpha, n_actions, input_dims, 128, 128)    #konstrukce
                ↪ neuronove site pro odhad kvalitativni funkce
14          self.q_target = build_dqn(alpha, n_actions, input_dims, 128, 128)  #konstrukce cilove
                ↪ neuronove site pro odhad kvalitativni funkce
```

V rámci třídy DDQNAgent implementujeme v kódu 2.13 funkce `choose_action()` a `learn()`. Funkce `choose_action()` slouží k výběru následující akce agenta. Funkce s pravděpodobností `epsilon` zvolí náhodou akci, tj. prozkoumávání a s pravděpodobností 1-`epsilon` vybere akci, pro nichž je predikovaná hodnota `self.q_eval()` v danám stavu největší. V rámci funkce `learn()`, pokud velikost dávky nepřesahuje počet záznamů uložených v zásobníku, predikujeme cílové hodnoty za použití neuronových sítí `self.q_eval` a `self.q_target`. Následně trénujeme síť `self.q_eval` minimalizací ztrátové funkce vypočtené díky cílovým hodnotám.

Kód 2.13: Definice funkce choose_action a funkce learn

```
1  def choose_action(self, state):
2      state = [state]
3      rand = np.random.random()
4      #vyber nahodne akce
5      if rand < self.epsilon:
6          action = np.random.choice(self.action_space)
7      #vyber hladove akce
8      else:
9          actions = self.q_eval.predict(state, verbose=0)
10         action = np.argmax(actions) #vraci pozici, ve ktere ma actions nejvetsi hodnotu
11     return action
12
13 def learn(self):
14     #pokud je v zasobniku vice dat nez je velikost davky, pak vybereme data nahodne ze
           ↪ zasobniku
15     if self.memory.mem_cntr > self.batch_size:
16         state, action, reward, new_state, done = \
17         self.memory.sample_batch(self.batch_size)
18         action_values = np.array(self.action_space, dtype=np.int8)
19         action_indieces = np.dot(action, action_values)
20         #k odhadu hodnoty
21         q_next = self.q_target.predict(new_state, verbose=0)
22         #k vyberu maximalni akce
23         q_eval = self.q_eval.predict(new_state, verbose=0)
24         q_pred = self.q_eval.predict(state, verbose=0)
25
26         max_action = np.argmax(q_eval, axis=1)
27         q_target = q_pred
28
29         #batch_index = np.arrange(self.batch_size, np.int32)
30         batch_index = np.array([j for j in range(self.batch_size)])
31         #menim pouze hodnotu odhadu q_target pro akci, kterou jsem udelal
32         q_target[batch_index, action_indieces] = reward +\
33         self.gamma*q_next[batch_index, max_action.astype(int)]*done
```

```
34
35          _  = self.q_eval.fit(state, q_target, verbose=0)   #trenovani site q_eval
36
37          self.epsilon = self.epsilon*self.epsilon_dec if self.epsilon > \
38                         self.epsilon_min else self.epsilon_min
39          if self.memory.mem_cntr%self.replace_target == 0:
40          self.update_network_parameters()   #uprava cilove site
```

Celková implementace algoritmu DDQN pro úlohu popsanou v části 2.1 je k nalezení na adrese

`https://github.com/jamichy/BP_turista_RL/tree/main/DDQN`

2.6 Gradient strategie (Policy gradient)

Do této doby byly všechny algoritmy, když agent neměl k dispozici model prostředí p, založené na odhadech kvalitativní funkce. Následně byla strategie π hladová, nebo ϵ-hladová vzhledem k odhadu kvalitativní funkce. Chtěli bychom mít na výběr více strategií, které jsou přirozeně stochastické. Strategie budeme parametrizovat pomocí parametru $\omega \in \mathbb{R}^{d'}$, kde $d' \in \mathbb{N}$ a budeme je označovat π_ω. Zápisem $\pi(a|s;\omega)$ budeme rozumět $\Pr(A_t = a|S_t = s;\omega_t = \omega)$. Pro dále popsané metody potřebujeme, aby π_ω byla diferencovatelná podle jednotlivých složek vektoru ω.

Pokud je prostor akcí $\mathcal{A}$ diskrétní a ne příliš veliký, pak můžeme zkonstruovat parametrizovanou strategii jako normalizovanou exponenciální funkci vztahem

$$\pi(a|s;\omega) = \frac{e^{u(s,a;\omega)})}{\sum_b e^{u(s,b;\omega)}}, \tag{2.25}$$

kde $u(s,a;\omega)$ je numerická preference závisející na ω, definovaná pro každý stav $s \in \mathcal{S}$ a akci $a \in \mathcal{A}$. Numerická preference u může být parametrizovaná například pomocí neuronové sítě, kde ω odpovídá nastavení sítě [1, kap. 13.1].

Nechť máme strategii π_ω a chceme vědět, jak často se určité stavy vyskytnou při agentově interakci, pokud agent jedná podle strategie π_ω. Zavedeme distribuci stavů pro epizodické úlohy $\lambda(s) \geq 0$ závisející na strategii π_ω vyjadřující jakou část epizody se vyskytuje agent ve stavu s. Samozřejmě platí $\sum_{s\in\mathcal{S}} \lambda(s) = 1$. Nechť $h(s)$ značí pravděpodobnost, že epizoda začíná ve stavu $s \in \mathcal{S}$ a $\eta(s)$ značí průměrný počet návštěv stavu s v jedné epizodě. Stav s je navštíven, pokud v něm začínáme nebo pokud se ze stavu $\bar{s}$ dostaneme do následujícího stavu s. Potom platí

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_a \pi(a|\bar{s};\omega)\tilde{p}(s|\bar{s},a) \;\forall s \in \mathcal{S}. \tag{2.26}$$

Distribuce stavů je pak rovna

$$\lambda(s) = \frac{\eta(s)}{\sum_{\bar{s}\in\mathcal{S}} \eta(\bar{s})}. \tag{2.27}$$

Když se v MDP vyskytuje diskontní faktor $\gamma < 1$ [1, kap. 9.2], pak γ začleníme do druhého členu v (2.26) a dostáváme diskontovaný průměrný počet stavů $\eta'(s)$ definovaný jako

$$\eta'(s) = h(s) + \gamma \sum_{\bar{s}} \eta'(\bar{s}) \sum_a \pi(a|\bar{s};\omega)\tilde{p}(s|\bar{s},a) = \mathbb{E}_{s'\sim h} \sum_{k=0}^{\infty} \gamma^k \Pr(s' \to s, k, \pi_\omega) \;\forall s \in \mathcal{S}, \tag{2.28}$$

kde $\Pr(s' \to s, k, \pi_\omega)$ značí pravděpodobnost, že se agent jednající podle strategie π_ω dostane ze stavu s' po k krocích do stavu s. Diskontovanou distribuci stavů $\lambda'(s)$ dostaneme úpravou rovnice (2.27) jako

$$\lambda'(s) = \frac{\eta'(s)}{\sum_{\bar{s}\in\mathcal{S}} \eta'(\bar{s})}. \tag{2.29}$$

Nyní chceme ohodnotit kvalitu strategie π_ω. Označme J funkci měřící výkon strategie a pro epizodické úlohy ji definujme vztahem

$$J(\omega) = \mathbb{E}_{s\sim h} v_{\pi_\omega}(s). \tag{2.30}$$

$J(\omega)$ je konvexní kombinace hodnotových funkcí v počátečních stavech s, kde příslušné koeficienty $h(s)$ jsou pravděpodobnosti začínání ve stavu s.

Chceme najít parametr ω, který maximalizuje hodnotu funkce J. Označme parametr strategie v čase t jako ω_t. Gradient $\nabla_\omega J(\omega_t)$ nám udává směr, v němž funkční hodnoty funkce J v bodě ω_t nejvíce rostou. Nové hodnotu parametru strategie dostaneme gradientním vzestupem podle vztahu

$$\omega_{t+1} = \omega_t + \alpha \nabla_\omega J(\omega_t). \tag{2.31}$$

V praxi budeme aplikovat stochastický gradientní vzestup, kde gradient $\nabla_\omega J(\omega_t)$ v (2.31) nahradíme jeho aproximací $\widehat{\nabla_\omega J(\omega_t)}$ a dostáváme pro úpravu parametru strategie vztah

$$\omega_{t+1} = \omega_t + \alpha \widehat{\nabla_\omega J(\omega_t)}. \tag{2.32}$$

Nyní bychom chtěli nalézt vhodné vyjádření $\nabla_\omega J(\omega)$, ze kterého bude zřejmé, jak gradient $\nabla_\omega J(\omega)$ spočítat. K tomuto vyjádření nám pomůže tvrzení 4, které následně zužitkujeme při hledání aproximace $\widehat{\nabla_\omega J(\omega_t)}$.

Tvrzení 4. Stochastický gradient strategie pro epizodické nastavení
Nechť π_ω je stochastická parametrizovaná strategie, která je diferencovatelná vzhledem ke složkám vektoru ω, $\lambda'(s)$ je diskontovaná distribuce stavů vzhledem ke strategii π_ω a $J(\omega)$ je definovaná vztahem (2.30). Pak platí

$$\nabla_\omega v_{\pi_\omega}(s) = \sum_{s'\in\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \Pr(s \to s', k, \pi_\omega) \sum_{a\in\mathcal{A}} q_{\pi_\omega}(s', a) \nabla_\omega \pi(a|s'; \omega), \forall s \in \mathcal{S} \tag{2.33}$$

a

$$\nabla_\omega J(\omega) \propto \sum_{s'\in\mathcal{S}} \lambda'(s') \sum_{a\in\mathcal{A}} q_{\pi_\omega}(s', a) \nabla_\omega \pi(a|s'; \omega), \tag{2.34}$$

kde $\propto$ je znak úměrnosti. Pro případ $\gamma < 1$ je koeficient úměrnosti roven $\frac{1-\gamma^T}{1-\gamma}$, pokud má každá epizoda délku $T \in \mathbb{N}$. Pro $\gamma = 1$ je koeficient úměrnosti roven T.

Důkaz. Nejprve dokážeme platnost (2.33) sérií úprav, které následně okomentujeme.

$$
\begin{aligned}
\nabla_\omega v_{\pi_\omega}(s) &= \nabla_\omega\Big(\sum_a \pi(a|s;\omega) q_{\pi_\omega}(s,a)\Big) \\
&= \sum_a \Big(q_{\pi_\omega}(s,a)\nabla_\omega \pi(a|s;\omega) + \pi(a|s;\omega)\nabla_\omega q_{\pi_\omega}(s,a)\Big) \\
&= \sum_a \Big(q_{\pi_\omega}(s,a)\nabla_\omega \pi(a|s;\omega) + \pi(a|s;\omega)\nabla_\omega \sum_{s',r} p(s',r|s,a)(r+\gamma v_{\pi_\omega}(s'))\Big) \\
&= \sum_a \Big(q_{\pi_\omega}(s,a)\nabla_\omega \pi(a|s;\omega) + \gamma\pi(a|s;\omega)\sum_{s'} \tilde{p}(s'|s,a)\nabla_\omega v_{\pi_\omega}(s')\Big) \\
&= \sum_a \Big(q_{\pi_\omega}(s,a)\nabla_\omega \pi(a|s;\omega) + \gamma\pi(a|s;\omega)\sum_{s'} \tilde{p}(s'|s,a)\Big(\\
&\sum_{a'}\Big(q_{\pi_\omega}(s',a')\nabla_\omega \pi(a'|s';\omega) + \pi(a'|s';\omega)\nabla_\omega q_{\pi_\omega}(s',a')\Big)\Big)\Big) \\
&= \cdots = \sum_{s'\in\mathcal{S}}\sum_{k=0}^{\infty}\gamma^k \Pr(s\to s',k,\pi_\omega)\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s',a)\nabla_\omega\pi(a|s';\omega)
\end{aligned}
\tag{2.35}
$$

Interpretace kroků v (2.35) je následující. Nejprve rozepíšeme $v_{\pi_\omega}(s)$ dle vztahu (1.10) pro hodnotovou funkci a necháme zapůsobit gradient na součin dvou funkcí. Následně vyjádříme kvalitativní funkci vyjádřenou vztahem (1.10). Odměna r nezávisí na ω a proto bude $\nabla_\omega r = 0$. Vysčítáním modelu prostředí $p(s', r|s, a)$ přes všechny odměny r dostaneme tranzitní model $\tilde{p}(s'|s, a)$, který nezávisí na ω. Nyní zapůsobí gradient pouze na hodnotovou funkci a dostaneme výraz $\nabla_\omega v_{\pi_\omega}(s')$, s kterým jsme začínali. Výše popsaný postup opakujeme a nakonec využijeme vztahu

$$
\Pr(s\to s',k,\pi_\omega) = \Pi_{i=0}^{k-1}\sum_{a_i\in\mathcal{A}}\sum_{s_{i+1}\in\mathcal{S}}\pi(a_i|s_i;\omega)\tilde{p}(s_{i+1}|s_i,a_i),\ \text{kde } s_0 = s \text{ a } s_k = s', \forall k \geq 1.
$$

$\Pr(s \to s', 0, \pi_\omega) = 1$ pro $s = s'$, jinak $\Pr(s \to s', 0, \pi_\omega) = 0$. Dále dokážeme vztah (2.34) sérií úprav, které následně vysvětlíme.

$$
\begin{aligned}
\nabla_\omega J(\omega) = \nabla_\omega \mathbb{E}_{s\sim h} v_{\pi_\omega}(s) &= \mathbb{E}_{s\sim h}\sum_{s'\in\mathcal{S}}\sum_{k=0}^{\infty}\gamma^k \Pr(s\to s',k,\pi_\omega)\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s',a)\nabla_\omega\pi(a|s';\omega) \\
&= \sum_{s'\in\mathcal{S}}\eta'(s')\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s',a)\nabla_\omega\pi(a|s';\omega) = \sum_{s\in\mathcal{S}}\eta'(s)\sum_{s'}\frac{\eta'(s')}{\sum_{s''}\eta'(s'')}\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s',a)\nabla_\omega\pi(a|s';\omega) \\
&= \sum_{s\in\mathcal{S}}\eta'(s)\sum_{s'\in\mathcal{S}}\lambda'(s')\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s',a)\nabla_\omega\pi(a|s';\omega) \propto \sum_{s'\in\mathcal{S}}\lambda(s')\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s',a)\nabla_\omega\pi(a|s';\omega)
\end{aligned}
\tag{2.36}
$$

Nejprve v (2.36) využijeme definice (2.30) a právě dokázané rovnice (2.33). Posléze využijeme definici $\eta'(s)$, viz (2.28) a vhodně rozšíříme. Následně využijeme definice diskontované stavové distribuce $\lambda'(s)$, viz (2.29). Výraz $\sum_{s\in\mathcal{S}}\eta'(s)$ označme jako koeficient úměrnosti. Čtenář snadno nahlédne, že výraz $\sum_{s\in\mathcal{S}}\eta'(s)$ je roven $\frac{1-\gamma^T}{1-\gamma} = \sum_{k=0}^{T-1}\gamma^k$, respektive T. □

Význam pravé strany (2.36), tj. výrazu $\sum_{s\in\mathcal{S}}\lambda'(s)\sum_{a\in\mathcal{A}} q_{\pi_\omega}(s,a)\nabla_\omega\pi(a|s;\omega)$, je následující. $\lambda'(s)$ nám říká, jak moc je pro nás stav s důležitý, $q_{\pi_\omega}(s,a)$ vyjadřuje, jak dobrá je akce a ve stavu s vzhledem ke strategii π_ω a $\nabla_\omega\pi(a|s;\omega)$ určuje, jakým směrem máme měnit parametr ω, aby se akce a vybírala častěji. Chceme upravit parametr ω, aby se častěji vybíraly akce a, které mají ve stavu s větší hodnotu

kvalitativní funkce $q_{\pi_\omega}(s,a)$ a stav s je pro nás důležitější. V praxi budeme používat stavovou distribuci $\lambda(s)$ namísto diskontované distribuce stavů $\lambda'(s)$, protože jsme ji schopni snáze nalézt a zajímáme se stejně o kvalitu strategie ve stavech v prvním časovém kroku jako ve stém časovém kroku.
Způsoby aproximace $\nabla_\omega J(\omega_t)$ jsou popsány v podsekcích 2.6.1 až 2.6.5.

2.6.1 REINFORCE

Potřebujeme získat data, z nichž očekávaný gradient bude úměrný aktuálnímu gradientu funkce J. Stačí nám pouze úměrnost, protože konstantu úměrnosti bude absorbovat parametr α [1, kap. 13.3]. Pokud bude agent jednat podle strategie π_ω, pak stavy se budou vyskytovat v souladu s distribucí stavů $\lambda(s)$ a můžeme přepsat (2.36) nahrazením diskontované stavové distribuce $\lambda'(s)$ stavovou distribucí $\lambda(s)$ do podoby

$$\nabla_\omega J(\omega) \propto \sum_{s\in\mathcal{S}} \lambda'(s) \sum_{a\in\mathcal{A}} q_{\pi_\omega}(s,a)\nabla_\omega \pi(a|s;\omega) = \mathbb{E}_{\pi_\omega}\left[\sum_{a\in\mathcal{A}} q_{\pi_\omega}(S_t,a)\nabla_\omega \pi(a|S_t;\omega)\right]. \tag{2.37}$$

Nyní rozšíříme výraz ve střední hodnotě (2.37) členem $\pi(a|S_t;\omega)$ a následně akci a nahradíme $A_t \sim \pi_\omega$. Člen $\frac{\nabla_\omega \pi(A_t|S_t;\omega)}{\pi(A_t|S_t;\omega)}$ přepíšeme jako $\ln \nabla_\omega \pi(A_t|S_t;\omega)$. Slovní postup shrnuje rovnice

$$\begin{aligned}\nabla_\omega J(\omega) &\propto \mathbb{E}_{\pi_\omega}\left[\sum_{a\in\mathcal{A}} q_{\pi_\omega}(S_t,a)\nabla_\omega \pi(a|S_t;\omega)\right] = \mathbb{E}_{\pi_\omega}\left[\sum_{a\in\mathcal{A}} \pi(a|S_t;\omega) q_{\pi_\omega}(S_t,a)\frac{\nabla_\omega \pi(a|S_t;\omega)}{\pi(a|S_t;\omega)}\right] \\ &= \mathbb{E}_{\pi_\omega}\left[q_{\pi_\omega}(S_t,A_t)\frac{\nabla_\omega \pi(A_t|S_t;\omega)}{\pi(A_t|S_t;\omega)}\right] = \mathbb{E}_{\pi_\omega}\left[q_{\pi_\omega}(S_t,A_t)\nabla_\omega \ln \pi(A_t|S_t;\omega)\right].\end{aligned} \tag{2.38}$$

Aproximaci gradientu funkce J získáme nahrazením členu $q_{\pi_\omega}(S_t,A_t)$ v rovnici (2.38) výnosem G_t, protože platí

$$\mathbb{E}_{\pi_\omega}[q_{\pi_\omega}(S_t,A_t)] = \mathbb{E}_{\pi_\omega}[G_t|S_t,A_t].$$

Konkrétní aplikací aproximace gradientu funkce J, kde G_t vypočteme stylem MC dostáváme algoritmus 9.

Algoritmus 9 REINFORCE - Monte Carlo gradient strategie

Vstup: diferencovatelná strategie π_ω
Parametry algoritmu: velikost kroku $\alpha > 0$
Inicializace: parametr strategie $\omega \in \mathbb{R}^{d'}$
Opakuj do nekonečna
  Generuj epizodu podle π_ω: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Opakuj pro každý krok epizody, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $\omega \leftarrow \omega + \alpha G \nabla_\omega \ln \pi(A_t|S_t;\omega)$
  konec Opakuj
konec Opakuj

2.6.2 REINFORCE with baseline

Mějme úlohu, ve které jsou všechny odměny kladné. Řekněme, že za dobrou akci dostane agent odměnu 8 a za špatnou 2. Pak je jakýkoliv výnos v této úloze vždy kladný. Pokud agent v proběhlé

epizodě vybíral špatné akce, tak algoritmus 9 upravil parametr ω tak, aby tyto špatné akce vybíral častěji. Pokud agent vybíral dobré akce, tak algoritmus 9 upravil parametr ω tak, aby tyto dobré akce vybíral častěji s větší váhou než špatné akce. Pokud bychom věděli, zda je současný výnos větší nebo menší než průměrný, pak bychom mohli upravit parametr ω aby pravděpodobnost akcí vedoucích k většímu výnosu se zvýšila a pravděpodobnost akcí vedoucích k menšímu výnosu se zmenšila. K rozlišení výnosů zavedeme libovolnou funkci b, označovanou jako základní hladina (baseline), která nezávisí na akci $a \in \mathcal{A}$ a je definovaná pro každý stav $s \in \mathcal{S}$. V praxi budeme b volit jako hodnotovou, nebo kvalitativní funkci vzhledem ke strategii π_ω. Zakomponováním funkce b zobecníme gradient strategie, viz tvrzení 4, do podoby

$$\nabla_\omega J(\omega) \propto \sum_{s\in\mathcal{S}} \lambda(s) \sum_{a\in\mathcal{A}} (q_{\pi_\omega}(s,a) - b(s)) \nabla_\omega \pi(a|s;\omega). \tag{2.39}$$

Zakomponováním b do gradientu strategie se nic nezměnilo, protože

$$\sum_{a\in\mathcal{A}} b(s)\nabla_\omega \pi(a|s;\omega) = b(s)\nabla_\omega \sum_{a\in\mathcal{A}} \pi(a|s;\omega) = b(s)\nabla_\omega 1 = 0. \tag{2.40}$$

V rovnici (2.40) jsme nejdříve využili faktu, že $b(s)$ nezávisí na akci a vytáhneme $b(s)$ před sumu. Dále jsme použili linearity gradientu, tj. součet gradientů je roven gradientu součtu a následně součet pravděpodobností je roven jedné. Finálním krokem je fakt, že gradient konstanty je roven 0.
V algoritmu 10 zvolíme funkci b jako odhad hodnotové funkce, který v algoritmu značíme $V(\cdot;\theta)$.

Algoritmus 10 REINFORCE se základní úrovní - Monte Carlo gradient strategie

Vstup: diferencovatelná strategie π_ω, diferencovatelná hodnotová funkce $V(\cdot;\theta)$
Parametry algoritmu: velikost kroku $\alpha^\omega > 0, \alpha^\theta > 0$
Inicializace: parametr strategie $\omega \in \mathbb{R}^{d'}$ a parametr hodnotové funkce $\theta \in \mathbb{R}^d$
Opakuj do nekonečna
  Generuj epizodu podle π_ω: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Opakuj pro každý krok epizody, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $\delta \leftarrow G - V(S_t;\theta)$
    $\theta \leftarrow \theta + \alpha^\theta \delta \nabla_\theta V(S_t;\theta)$
    $\omega \leftarrow \omega + \alpha^\omega \delta \nabla_\omega \ln \pi(A_t|S_t;\omega)$
  konec Opakuj
konec Opakuj

2.6.3 Aktér-kritik (Actor-Critic)

U algoritmů 9 a 10 jsme museli čekat na konec epizody a teprve potom jsme mohli upravovat parametr strategie ω. Navíc výnos G_t byl sice nestranný, ale měl vysoký rozptyl, protože byl ovlivněn akcemi a stavy $S_t, A_t, S_{t+1}, A_{t+1}, \cdots, S_{T-1}, A_{T-1}$. V této části se budeme věnovat algoritmům, ve kterých upravujeme strategii pomocí odhadu výnosu stejně jako u temporální diference, díky čemuž se sníží rozptyl. Tyto algoritmy se označují jako aktér-kritik. Aktér odpovídá části věnované strategii a kritik části odhadu hodnotové, nebo kvalitativní funkce [3, str. 111,112].

Existuje několik variant algoritmu aktér-kritik pro stochastickou strategii. Liší se například podle toho, jakým způsobem se zvolí δ v algoritmu 11 [3, str. 112-116]. Pro ilustraci tohoto typů algoritmů uvedeme algoritmus 11, který se liší od algoritmu 10 v tom, že už po vybrání akce provedeme úpravu parametrů θ a ω pomocí δ, které bude rovno $R + \gamma V(S'; \theta) - V(S; \theta)$.

Algoritmus 11 Aktér-kritik ve variantě $TD(0)$ se základní úrovní

Vstup: diferencovatelná strategie π_ω, diferencovatelná hodnotová funkce $V(\cdot; \theta)$
Parametry algoritmu: velikost kroku $\alpha^\omega > 0, \alpha^\theta > 0$
Inicializace: parametr strategie $\omega \in \mathbb{R}^{d'}$ a parametr hodnotové funkce $\theta \in \mathbb{R}^d$
Opakuj pro každou epizodu
  Inicializace počátečního stavu S
  Opakuj pro každý krok v epizodě
    $A \sim \pi(\cdot|S; \omega)$
    Proveď akci A, pozoruj R, S'
    $\delta \leftarrow R + \gamma V(S'; \theta) - V(S; \theta)$
    $\theta \leftarrow \theta + \alpha^\theta \delta \nabla_\theta V(S_t; \theta)$
    $\omega \leftarrow \omega + \alpha \delta \nabla_\omega \ln \pi(A|S; \omega)$
  konec Opakuj
konec Opakuj

2.6.4 Deterministický gradient strategie (Deterministic policy gradient)

V této části budeme vycházet z [23]. Mějme spojitý prostor akcí $\mathcal{A} = \mathbb{R}^m$ a stavů $\mathcal{S} = \mathbb{R}^n$. Pak ρ^π označuje diskontovanou distribuci stavu vzhledem ke strategii π_ω definovanou pomocí vztahu

$$\rho^\pi(s) = \int_{\mathcal{S}} \sum_{k=0}^{\infty} \gamma^k h(s') \Pr(s' \to s, k, \pi_\omega)\, ds', \tag{2.41}$$

kde $\Pr(s' \to s, k, \pi_\omega)$ značí pravděpodobnost, že se agent jednající podle strategie π_ω dostane ze stavu s' po t krocích do stavu s a $h(s')$ vyjadřuje pravděpodobnost začínání úlohy ve stavu s'.
Definujme ještě průměrnou odměnu $\tilde{r}$ ze stavu s při akci a pro diskrétní množinu odměn $\mathcal{R}$ jako $\tilde{r}(s, a) = \sum_{r \in \mathcal{R}} rR(r|s, a)$.
Pro případ spojitého prostoru stavů $\mathcal{S}$ a akcí $\mathcal{A}$ můžeme funkci J vyjádřit pomocí vztahu

$$J(\omega) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi(a|s; \omega) \tilde{r}(s, a)\, da\, ds. \tag{2.42}$$

Stochastický gradient strategie pro spojitý prostor stavů $\mathcal{S}$ a akcí $\mathcal{A}$ přejde do podoby

$$\nabla_\omega J(\omega) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} q_{\pi_\omega}(s, a) \nabla_\omega \pi(a|s; \omega)\, da\, ds = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\omega}[q_{\pi_\omega}(s, a) \nabla_\omega \ln \pi(a|s; \omega)]. \tag{2.43}$$

V této části budeme uvažovat deterministickou strategii $\mu_\omega : \mathcal{S} \to \mathcal{A}$, které je parametrizovaná vektorem ω a diferencovatelná vůči všem jeho složkám. Díky tomu, že deterministické strategie přiřadí každému stavu $s \in \mathcal{S}$ právě jednu akci $a \in \mathcal{A}$, tj. $\mu(a|s; \omega) = 1$, nemusíme pro výpočet funkce J integrovat přes množinu akcí $\mathcal{A}$ a můžeme rovnici (2.42) přepsat jako

$$J(\omega) = \int_{\mathcal{S}} \rho^\mu(s) \tilde{r}(s, \mu_\omega(s))\, ds.$$

Za předpokladu, že $\tilde{p}(s'|s,a), \nabla_a \tilde{p}(s'|s,a), \mu_\omega(s), \nabla_\omega \mu_\omega(s), \tilde{r}(s,a), \nabla_a \tilde{r}(s,a), h(s)$ jsou spojité funkce ve všech parametrech a proměnných, přejde vztah (2.43) pro deterministickou strategii na

$$\nabla_\omega J(\omega) = \int_{\mathcal{S}} \rho^\mu(s) \nabla_\omega \mu_\omega(s) \nabla_a q_{\mu_\omega}(s,a)|_{a=\mu_\omega(s)}\, ds = \mathbb{E}_{s\sim\rho^\mu}[\nabla_\omega \mu_\omega(s) \nabla_a q_{\mu_\omega}(s,a)|_{a=\mu_\omega(s)}]. \tag{2.44}$$

Důkaz vztahu (2.44) je uveden v dodatečném materiálu článku -[23]

Algoritmus Aktér-kritik pro deterministickou strategii by byl obdobný algoritmu 11, pouze bychom zaměnili stochastickou strategii π_ω za deterministickou strategii μ_ω. Kritik by nyní představoval kvalitativní funkci $Q(\cdot,\cdot,\theta) \approx q_{\mu_\omega}$ místo hodnotové funkce. Rozdíl mezi odhadem výnosu pro tuto epizodu a starým odhadem δ volíme stejně jako u algoritmu Sarsa, tj.

$$\delta \leftarrow R + \gamma Q(S', A', \theta) - Q(S, A; \theta).$$

Parametery θ a ω upravujeme vztahem

$$\theta \leftarrow \theta + \alpha^\theta \delta \nabla_\theta Q(S, A; \theta),$$

respektive

$$\omega \leftarrow \omega + \alpha^\omega \nabla_\omega \mu_\omega(S) \nabla_a Q(S, a; \theta)|_{a=\mu_\omega(S)}.$$

2.6.5 Hluboký deterministický gradient strategie (Deep deterministic policy gradient)

Spojením hlubokého Q-učení s deterministickým gradientem strategie dostaneme algoritmus Hluboký deterministický gradient strategie, který byl prvně popsán v článku [24], ze kterého budeme v této části vycházet. Pro zkracení zápisu budeme pro označení algoritmu používat zkratku DDPG.

Pro prostor akcí $\mathcal{A}$, který je spojitý, případně i vícedimenzionální nemůžeme použít Hluboké Q-učení, protože nejsme schopni jednoduše určit hladovou akci ve stavu $s \in \mathcal{S}$. Hladovou akci jsme vybírali pomocí vztahu $a \leftarrow \underset{a}{\operatorname{argmax}} Q(s, a; \theta)$.

Strategie μ_ω je reprezentovaná aktérem, který je dále implementován neuronovou sítí $\mu(\cdot;\omega)$, jejíž vstupem je stav $s \in \mathcal{S}$. Počet uzlů ve vstupní vrstvě je n. Při predikci přiřadíme do každého vstupního uzlu odpovídající složku stavu s. Výstupem je pak m-dimenzionální akce. Kritik je reprezentován neuronovou sítí $Q(\cdot,\cdot;\theta)$, jejíž vstupem je návštěva (s,a), proto má vstupní vrstva $m+n$ uzlů. Výstupní vrstva obsahuje jeden neuron, který odpovídá odhadu kvalitativní funkce pro návštěvu (s,a).

Stejně jako v hlubokém Q-učení budeme používat zásobník zkušeností $\mathcal{D}$ pro stabilnější trénování. Výsledný algoritmus bude využívat cílovou síť modifikovanou pro přístup aktér-kritik. Vytvoříme kopie kritikovy a aktérovy sítě $Q(\cdot,\cdot;\theta')$, respektive $\mu(\cdot;\omega')$, které budeme nazývat cílové neuronové sítě kritika a aktéra. Cílové neuronové sítě použijeme pro výpočet cílové hodnoty $y = r + \gamma Q(s, \mu(s;\omega');\theta')$, díky které můžeme trénovat neuronovou síť kritika $Q(\cdot,\cdot;\theta)$. Váhy cílových sítí jsou upravovány vztahem

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta',$$
$$\omega' \leftarrow \tau\omega + (1-\tau)\omega',$$

kde $\tau \ll 1$. Mírnou změnou parametrů θ', ω' se cílové hodnoty y mění pomalu, což výrazně zlepšuje stabilitu učení.

Jelikož je $\mu(s;\omega)$ reprezentace deterministické strategie, tak vybírá vždy pro daný stav s a konkrétní parametr ω stejnou akci a a nedochází proto k prozkoumávaní. Abychom zajistili prozkoumávání, agentova akce bude $\mu(s;\omega) + \zeta$, kde ζ je prozkoumávací šum. Autoři článku [24] použili prozkoumávací šum pocházející z Ornsteinova-Uhlenbeckova procesu. Výše uvedený popis shrnuje algoritmus 12.

Algoritmus 12 Hluboký deterministický gradient strategie (DDPG)

Inicializace neuronové sítě kritika $Q(s, a; \theta)$ a aktéra $\mu(s; \omega)$ s libovolnými parametry θ a ω
Inicializace cílové neuronové sítě kritika $Q(s, a; \theta')$ a aktéra $\mu(s; \omega')$ s parametry $\theta' \leftarrow \theta, \omega' \leftarrow \omega$
Inicializace zásobníku zkušeností $\mathcal{D}$ o velikosti M
Opakuj pro každou epizodu
  Inicializace počátečního stavu S
  Opakuj pro každý krok v epizodě
    Vytvoř šum ζ
    $A = \mu(S; \omega) + \zeta$
    Proveď akci A, pozoruj R, S', Done
    Ulož do zásobníku $\mathcal{D}$ zkušenost $(S, A, R, S'$, Done$)$
    Vyber ze zásobníku $\mathcal{D}$ dávku $\mathcal{B}$ o velikosti N
    Opakuj pro každé $(s_i, a_i, r_i, s'_i, \text{done}_i)$ v dávce $\mathcal{B}$
      $y_i = r_i + \gamma Q(s'_i, \mu(s'_i; \omega'); \theta')$
    konec Opakuj
    Uprav neuronovou síť kritika minimalizací chyby $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i; \theta))^2$
    Uprav neuronovou síť aktéra pomocí gradientu

$$\nabla_\omega J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a; \theta)|_{s=s_i, a=\mu(s_i;\omega)} \nabla_\omega \mu(s; \omega)|_{s=s_i}$$

    Uprav cílové sítě:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$
$$\omega' \leftarrow \tau\omega + (1 - \tau)\omega'$$

  konec Opakuj
konec Opakuj

2.7 Porovnání algoritmů na ukázkové úloze

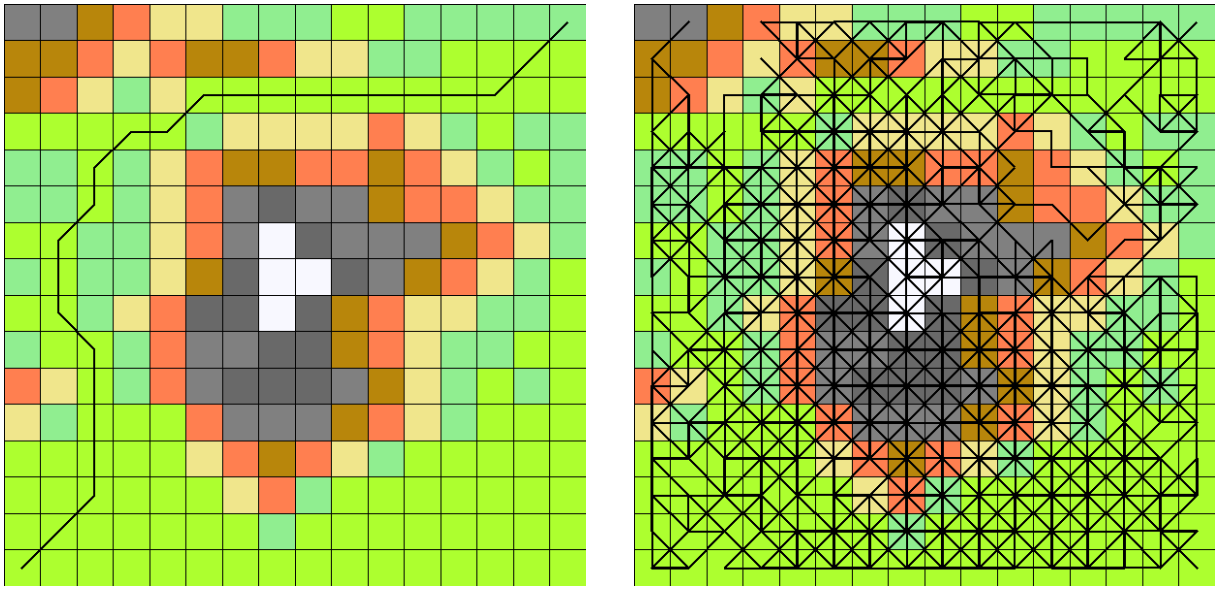
V následující části porovnáme vlastnosti algoritmů MC, Sarsa, Q-učení, Dvojitého Q-učení a DDQN na ukázkové úloze popsané v části 2.1. Nastavili jsme počáteční parametr $\gamma = 1$, protože je pro nás stejně důležitá odměna, kterou dostaneme v prvním časovém kroku jako v posledním kroku. Zvolením výškového faktoru L na hodnotu dva jsme agentovi určili množství spotřebované energie pro překonání výškového rozdílu o velikosti 1.

V levé části obrázku 2.9 můžeme vidět vykreslení nejlepší možné trasy pro právě popsané počáteční nastavení parametrů, kdy agentovo finální skóre je rovno -20.9. Napravo pak vykreslení náhodné trasy, tj. takové, kde agentův pohyb byl čistě náhodný.

Při vykreslování obrázků jsme použili knihovnu Pillow. Každý pohyb agenta je znázorněn odpovídající černou čárou. Pokud je stejný pohyb proveden několikrát, tloušťka čáry se zvětší.

Agent byl trénován několik epizod a aby se dostal do konečného stavu, nastavili jsme pro počáteční epizodu úroveň prozkoumávání ϵ na hodnotu 1. Následně pro každou další epizodu jsme snižovali lineárně ϵ, až ve finální epizodě jsme dostali hodnotu ϵ 0, tj. v poslední epizodě agent jednal po celou dobu podle nalezené strategie.

Pro algoritmy MC, Sarsa, Q-učení a Dvojité Q-učení je α parametr určující rychlost úpravy odhadu kvalitativní funkce, viz po řadě vztahy (2.13), (2.20), (2.21) a (2.22) + (2.23). Tento parametr byl nasta-



(a) Nejlepší možná trasa, skóre= -20.9

(b) Náhodná trasa ($\epsilon = 1$), skóre=-5853.4

Obrázek 2.9: Znázornění nejlepší a náhodné trasy pro nastavení parametrů $\gamma = 1$, $L = 2$

ven na hodnotu 0.1. V DDQN byly použity dvě neuronové sítě, každá s dvěma skrytými vrstvami o 128 neuronech. Při výpočtu výstupní hodnoty v skrytých vrstvách byly použity aktivační funkce ReLU. Velikost zásobníku byla nastavena na 100000 záznamů a velikost dávky na hodnotu 64. V těchto metodách rovněž vystupuje parametr α, a to jako učící parametr v algoritmech gradientního sestupu. Z hlediska velikosti je obecně mnohem menší, protože trénování probíhá po dávkách a v našem případě byl nastaven na hodnotu 0.01. Cílovou neuronovou síť jsme upravili každých sto kroků. Výsledky trénování na 101 epizodách jsou zaneseny v tabulce 2.1.

algoritmus	finální skóre	nejlepší skóre	vypočetní čas [s]
MC	X	X	X
Sarsa	X	X	X
Q-učení	-438	-101.6	2.51
Dvojité Q-učení	-506	-131	2.71
DDQN	-32.78	-27.5	2869

Tabulka 2.1: Výsledky trénování algoritmů při trenování na 101 epizodách

Z důvodu nízkého počtu epizod se setkáváme ve všech algoritmech kromě DDQN s velmi špatnými výsledky. V algoritmu MC a v Sarse v poslední epizodě, kde ϵ je nastaveno na nulu, nenašel agent cílovou pozici. Tuto skutečnost jsme označili symbolem X. Trénovací čas u DDQN byl více jak stokrát větší než u Q-učení a Dvojitého Q-učení. To je způsobeno časovou náročností trénování neuronové sítě. Řešením jak vylepšit finální skóre je buď trénovat agenta na více epizodách, nebo upravit hodnotu α. Podívejme se nejprve pro algoritmy Q-učení a Dvojitého Q-učení na závislost průměrného finálního skóre na hodnotě α. Do průměru jsme zahrnuli pět realizací experimentu.

α	průměrné finální skóre Q-učení	průměrné finální skóre Dvojitého Q-učení
0.5	-96.6	-156.45
0.7	-55.5	-125.2
0.8	-56.3	-124.1
0.9	-50.2	-148

Tabulka 2.2: Závislost průměrného skóre na hodnotě α u Q-učení a Dvojitého Q-učení při trénování na 101 epizodách

Z tabulky 2.2 je zřejmé, že pro nízký počet trénovacích epizod je lepší mít vyšší hodnotu α. Čím větší je hodnota α, tím více posuneme odhad kvalitativní funkce ve směru odhadu výnosu v této epizodě. Pro algoritmy Sarsa a MC zvýšení parametru pro trénování na 101 epizodách nepomohlo, a proto jsme v další fázi zvýšili počet epizod na 10001 a parametr α nastavíme na hodnotu 0.1.

V tabulce 2.3 vidíme, že se agent v MC i v Sarse dostal do konečného stavu i ve finální epizodě. Míru prozkoumávání ϵ jsme lineárně snižovali. Nastavení parametrů bylo $\alpha = 0.1$, $\gamma = 1$.

algoritmus	finální skóre	vypočetní čas [s]
MC	-33.06	392
Sarsa	-35.08	118
Q-učení	-20.9	178
Dvojité Q-učení	-20.9	320

Tabulka 2.3: Výsledky trénování algoritmů při trenování na 10001 epizodách

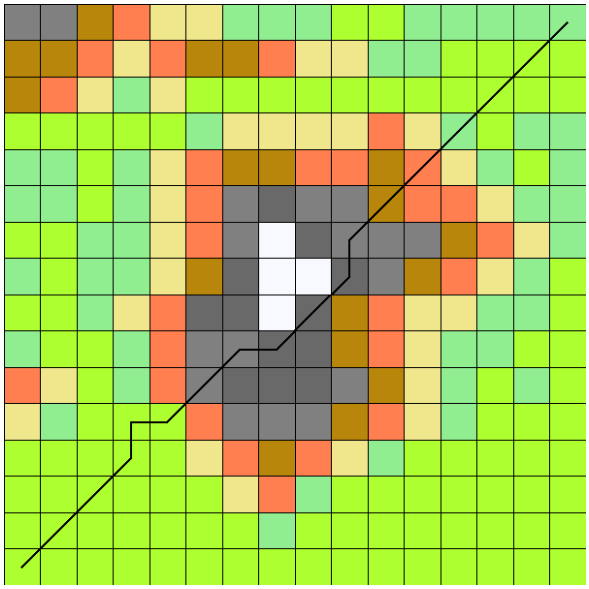
Na obrázku 2.10 jsou vykreslené finální trasy pro algoritmus MC a Sarsa.

Dále jsme snížili složitost úlohy nastavením výškového faktoru L na nulu, tj. zrušili jsme výškový význam políček, a proto se bude agent pohybovat jen po rovině. Do tabulky 2.4 jsme vynesli pro algoritmy MC, Sarsa, Q-učení a Dvojité Q-učení počet potřebných epizod a parametr α, pro které algoritmus nalezl nejlepší trasu. Ta vede z počáteční pozice přímo po diagonále do finální pozice a její hodnota je -14.8.

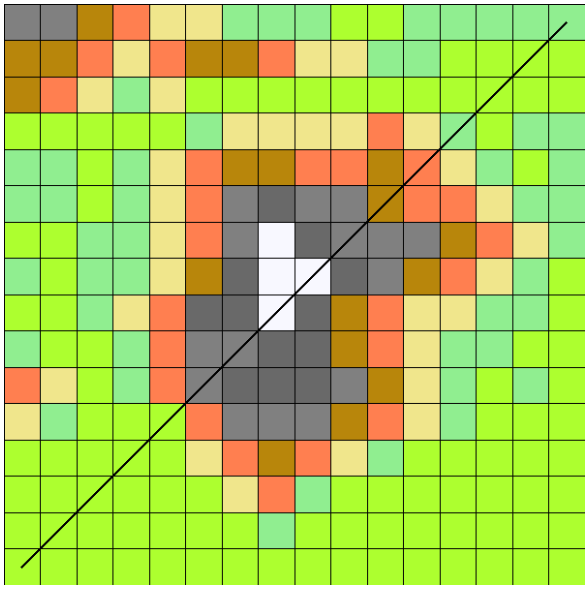
Efektivita algoritmů by se mohla ještě vylepšit nalezením lepších hodnot parametru α nebo jiným postupem snižování ϵ.

algoritmus	počet epizod	vypočetní čas [s]	α
MC	10001	173.4	0.1
Sarsa	1501	29.8	0.3
Q-učení	751	23.8	0.3
Dvojité Q-učení	1501	46.6	0.3

Tabulka 2.4: Výsledky trénování algoritmů pro $L = 0$



(a) Trasa pro desetitisícátou první epizodu pro MC agenta se skóre -33.06



(b) Trasa pro desetitisícátou první epizodu pro Sarsa agenta se skóre -35.08

Obrázek 2.10: Zobrazení poslední trasy pro metodu MC a Sarsa

Kapitola 3

Využití zpětnovazebného učení k nalezení strategie obchodování

3.1 Fundamentální analýza

Akcie (stock nebo také bývá v daném významu používáno share) spadají, do skupiny investičních nástrojů, konkrétně mezi cenné papíry. Jsou vydané společností a je s nimi spojeno právo na podíl ze zisku (dividenda), právo na likvidačním zůstatku společnosti, právo na informace nebo právo na úpis dalších akcií při zvýšení základního kapitálu. Rovněž akcie dokazuje skutečnost, že její majitel vložil konkrétní majetkový podíl (kapitál) do dané akciové společnosti [26, str. 13].

Akciový trh

Akciové trhy představují burzy, na kterých probíhá nákup, prodej a vystavování nových akcií veřejně obchodovaných společností. Jsou dynamické a mají dopad na celou tržní ekonomiku. Mezi primární funkce akciových trhů patří: poskytování přesných a okamžitých informací obchodníkům, udržování likvidity, efektivní určování ceny na základě nabídky a poptávky, zabezpečení provedení transakcí a další [27]. Obchodování na akciových trzích je zpřístupněno pouze jejím členům, bankám a makléřským společnostem které zastupují brokeři (zprostředkovávají jednotlivým investorům obchodování na akciových trzích za určitý poplatek) [28]. Standardně jsou burzy otevřený v obchodní dny od pondělí do pátku, např. New York Stock Exchange (NYSE) je otevřena od 9:30 do 16:00 místního času, pouze v těchto časech lze na burze uzavřít obchod [29].

Akcie jsou jednou z mnoha forem investic, proto je nezbytné mít neustále na paměti tzv. pravidlo investičního trojúhelníku. Ten se skládá z výnosu, rizika a likvidity. Výnos jsou příjmy z námi vložené investice, riziko je ukazatel nebezpečí nebo také „rating“ investiční příležitosti a likvidita znamená schopnost přeměnit investici na peníze, a to rychle a s minimálními náklady [30].

Rozdíl mezi fundamentální a technickou analýzou

Jedná se o 2 základní myšlenkové proudy, jak přemýšlet nad akciovými trhy. Obchodníci a investoři využívají obojí k výzkumu a predikcím cen akcií. Fundamentální analýza se pokouší zhodnotit vnitřní hodnotu akcie, je přihlíženo na spoustu faktorů jako např. celkovou ekonomiku, průmyslové podmínky či finanční sílu jednotlivých společností (zisky, výdaje, aktiva, závazky apod.). Oproti tomu v technické analýze se snaží identifikovat příležitosti sledováním statistických trendů, jako např. pohyby cen a objemu nakoupených a prodaných akcií. Základním předpokladem pro technickou analýzu je že veškeré

známe fundamenty jsou zohledněny v ceně. Není jim tedy potřeba věnovat takovou pozornost. Místo vnitřní hodnoty cenného papíru sledují akciové grafy k identifikaci vzorců a trendů, jenž by mohly naznačovat jaký vývoj můžeme čekat do budoucna [31].

3.1.1 Představení úlohy na výběr kombinace podintervalů indikátorů

Tato bakalářská práce navazuje na bakalářskou práci od Veroniky Deketové [26], kde předzpracovala data z americké burzy a uložila je do vhodného formátu. Dále zkonstruovala neuronovou síť a predikovala týdenní relativní profit v závislosti na fundamentálních datech pro daný akciový titul. Informace o akciových titulech máme od roku 2000 do současnosti. V následující části je popsaná epizodická úloha, jejíž vyřešení by měl dát odpověď na otázku, zda existuje optimální výběr kombinace podintervalů indikátorů, ve kterých se má akcie pohybovat, aby vykazovala profit v průběhu času. Agent bude během epizody obchodovat na burze s cílem maximalizovat finální kapitál.

Máme k dispozici data z akciové burzy v následujícím formátu. Každému akciovému titulu je na konci každého obchodního týdne přiřazeno N hodnot odpovídajících N fundamentálních indikátorů a přiřazena uzavírací cena za akcii. Dále máme ještě v datech uvedený týdenní relativní profit. Hodnoty každého indikátoru se pohybují v pevném rozsahu. Rozsah hodnot pro j-tý indikátor určeme jako interval $\langle p_{j,\text{Start}}, p_{j,\text{End}} \rangle$, kde $p_{j,\text{Start}}$, $p_{j,\text{End}}$ je nejmenší, respektive největší hodnota j-tého indikátoru v historii. Zkonstruujeme epizodickou úlohu, kde agent v každém časovém kroku, rozumějme obchodním týdnu, má k dispozici určité množství finančního kapitálu k zainvestování. Úloha začíná v časovém kroku initialWeek a končí v kroku endWeek, kde agent vybere akci, která se nemůže vyhodnotit, protože bychom v tomto časovém kroku na reálných datech neznali relativní profit akciových titulů splňující agentovu akci. Jinak v každém časovém kroku vybere agent kombinaci podintervalů indikátorů a určí, jaká část kapitálu se má zainvestovat. Akcie, jež splňují agentův výběr, se v prostředí nakoupí a vypočtou se nákupní poplatky. V dalším časovém kroku se nakoupené akcie prodají, vypočtou se prodejní poplatky a profit. Agent dostává od prostředí stav $s = (s_1, s_2)$ a odměnu v podobě relativní změny kapitálu oproti minulému časovému kroku, kde s_1 vyjadřuje množství dostupných peněz a s_2 v jaké části epizody se vyskytujeme. Pokud agentův výběr nesplňuje žádný akciový titul v daném čase, dostává agent od prostředí původní množství kapitálu ponížené o penalizaci a relativní změnu kapitálu. Cílem agenta je během epizody maximalizovat součet relativních změn kapitálu. Pokud jsme maximalizovali součet relativních změn kapitálu, pak jsme také maximalizovali množství vydělaných peněz. Úloha končí, pokud je agent v časovém kroku endWeek, nebo pokud je jeho kapitál menší než minimální. Rozdíl endWeek a initialWeek označme $T - 1$.

Předpoklady úlohy:

- Naše nákupy neovlivní ceny na burze.
- Vysoká likvidita trhu – není rozdíl mezi cenou nákupu a prodeje.
- Jsme schopni obchodovat nejen celé akcie, ale i jejich zlomky.
- Úloha je MRP, tj. další odměna a stav je podmíněna pouze aktuálním stavem a akcí, nikoliv celou historií.
- Na americké burze se vyplácí dividendy kvartálně, a tak neovlivní moc cenu akciového titulu. Proto zanedbáváme výplatu dividend.

Akce

Prostor akcí je definován jako

$$\mathcal{A} = \langle -1, 1\rangle^{2N+1} = \{(m_1, n_1, m_2, n_2, \cdots, m_N, n_N, o) | m_j, n_j, o \in \langle -1, 1\rangle, \text{ pro } \forall j \in \hat{N},$$

kde N je počet fundamentálních indikátorů. Agent v čase t vybere akci $A_t = (m_{1,t}, n_{1,t}, m_{2,t}, n_{2,t}, \cdots, m_{N,t}, n_{N,t}, o_t)$. V prostředí přeškálujeme A_t na $(d_{1,t}, e_{1,t}, \cdots, d_{N,t}, e_{N,t}, b_t)$, kde $d_{j,t}$ značí začátek vybraného podintervalu pro j-tý indikátor v čase t a $e_{j,t}$ značí konec vybraného podintervalu. Možné varianty přeškálování $m_{j,t}$ a $n_{j,t}$ na $d_{j,t}$ a $e_{j,t}$, kde $j \in \hat{N}$ jsou následující.

- $m_{j,t}$ bude představovat začátek podintervalu a $n_{j,t}$ jeho konec. $d_{j,t}$ a $e_{j,t}$ vypočteme pomocí vztahů

$$d_{j,t} = p_{j,\text{Start}} + \frac{(1 + m_{j,t})(p_{j,\text{End}} - p_{j,\text{Start}})}{2}, \tag{3.1}$$

$$e_{j,t} = p_{j,\text{Start}} + \frac{(1 + n_{j,t})(p_{j,\text{End}} - p_{j,\text{Start}})}{2}. \tag{3.2}$$

  Problém nastane pokud $m_{j,t} > n_{j,t}$, protože pak $d_{j,t} > e_{j,t}$ a interval $(d_{j,t}, e_{j,t})$ nemá smysl. Možné řešení by bylo dát agentovi navíc penalizaci za nevalidní akci.

- $m_{j,t}$ bude představovat střed podintervalu a $n_{j,t}$ bude udávat relativní šířku okolí podintervalu. $d_{j,t}$ a $e_{j,t}$ vypočteme pomocí rovnic

$$\begin{aligned}
\text{center} &= p_{j,\text{Start}} + \frac{(1 + m_{j,t})(p_{j,\text{End}} - p_{j,\text{Start}})}{2}, \\
\text{minDist} &= min(p_{j,\text{End}} - \text{center}, \text{center} - p_{j,\text{Start}}), \\
\text{relWidth} &= \text{minDist}\frac{1 + n_{j,t}}{2}, \\
d_{j,t} &= \text{center} - \text{relWidth}, \\
e_{j,t} &= \text{center} + \text{relWidth}.
\end{aligned}$$

- m_{jt} bude představovat začátek podintervalu a n_{jt} udává, v jaké proporcionální části od $d_{j,t}$ se bude nalézat konec podintervalu. $d_{j,t}$ vypočteme dle vztahu (3.1) a $e_{j,t}$ jako

$$e_{j,t} = d_{j,t} + \frac{(1 + n_{j,t})(p_{j,\text{End}} - d_{j,t})}{2}.$$

o_t přeškálujeme v prostředí na parametr velikosti nákupu v čase t $b_t = \frac{1+o_t}{2}$. Vyzkoušeli jsme všechny popsané varianty přeškálování akce. Nejlepších výsledků dosahovala třetí popsaná varianta.

Odměna

Agent začíná v prvním obchodním týdnu s počátečním kapitálem, který označme jako initial_budget. Obecně kapitál v čase t označme jako K_t. Nechť se v časovém kroku t vybere P_t akcií. Pro vybrané akcie načteme uzavírací ceny a relativní profit, který se vypočetl jako poměr rozdílu uzavírací ceny v časovém kroku $t + 1$ a v časovém kroku t vzhledem k uzavírací ceně v časovém kroku t. Následně vypočteme celkovou investovanou částku v čase t jako

$$\text{invested\_amount}_t = \min((K_t - \text{minimal\_budget}) * b_t, K_t * b_t * 0.95).$$

Na zaplacení nákupních poplatků si necháme částku minimal_budget, případně $K_t * b_t * 0.05$. Jelikož budeme nakupovat všechny akcie za stejný obnos peněz, použijeme pro nákup každého akciového titulu částku

$$\text{invested\_amount\_per\_stock}_t = \frac{\text{invested\_amount}_t}{P_t}.$$

Podle počtu nakupených akcií vypočteme nákupní poplatky buy_fee a do proměnné change_rate přiřadíme součet relativních profitů pro každý vybraný akciový titul. Pak změna hodnoty nakoupených akcií je rovna change_rate $*$ invested_amount_per_stock$_t$. Odměna je pak rovna

$$R_{t+1} = \frac{\text{change\_rate} * \text{invested\_amount\_per\_stock}_t - 2 * \text{buy\_fee}}{K_t}.$$

V případě, že kombinace vybraných podintervalů je validní, ale žádný akciový titul v daný čas nesplňuje vybraná kritéria, tak dostává agent odměnu $R_{t+1} = \frac{\text{penalty}_1}{K_t}$. Pokud je některý vybraný podinterval prázdný, pak dostává agent odměnu $R_{t+1} = \frac{\text{penalty}_2}{K_t}$.
Hodnota kapitálu v čase $t+1$ se vypočte jako $K_{t+1} = K_t + R_{t+1} * K_t$.

Poplatky

V úloze předpokládáme reálné poplatky akciového brokera. Do dvou set kusů akcií se platí fixně 2 dolary, za každou další akcii 4 centy. Nákupní i prodejní poplatky jsou shodné. Poplatky motivují agenta, aby vybíral jen pár akciových titulů a nenakupoval všechny akciové tituly obchodované na burze.

Stavy

Stavem v čase t - S_t budeme rozumět uspořádanou dvojici $(S_{1,t}, S_{2,t})$, kde první složka vyjadřuje podíl kapitálu v čase t a počátečního kapitálu, tj. $S_{1,t} = \frac{K_t}{\text{initial\_budget}}$. Druhá složka vyjadřuje v jaké části obchodování se agent nachází, tj. $S_{2,t} = \frac{t}{T-1}$.

Prostor všech stavů včetně finálních $\mathcal{S}^+ = \mathbb{R}^+ \times \{\frac{i}{T-1} | i \in \{0, 1, \ldots, T-1\}\}$

Prostor stavů $\mathcal{S} = \mathbb{R}^+ \times \{\frac{i}{T-1} | i \in \{0, 1, \ldots, T-2\}\}$

3.2 Prostředí

V následující části popíšeme programovacího kód prostředí. Zaměříme se pouze na funkční složky a nebudeme popisovat zapisování akcí do souboru, které se použijí pro grafické znázornění vybírání v průběhu epizody.

Úlohu budeme v sekci 4.2 testovat na uměle vytvořených datech, viz 4.1, pocházejících ze smyšlené akciové burzy, kde se po celou dobu obchodují stejné tři akciové tituly. Každý týden známe pro každý akciový titul hodnoty dvou indikátorů, uzavírací cenu a relativní profit. Pro prvotní testování jsme v prostředí načetli hodnoty akciových titulů pro prvních třicet tři týdnů a každý týden agent obchodoval na všech načtených datech. Pro tyto data jsme zkonstruovali prostředí.

Nejprve naimportujeme používané knihovny příkazem `import pandas as pd` a `import math`. Následně v kódu 3.1 zkonstruujeme třídu `StockTrading` a konstruktoru předáme potřebné parametry. Ten uloží většinu vstupních hodnot a načte soubor s daty do `self.stocksData`. Vytvoří se odpovídající seznamy pro načtení hodnot indikátorů, uzavíracích cen a relativních profitů. Dále ještě inicializujeme proměnné potřebné pro realizaci nákupu.

Kód 3.1: Inicializace třídy StockTrading

```
class StockTrading(object):
    def __init__(self, initialWeek, endWeek, initialBudget, minimalBudget, p1Start, p1End,
        ↪ p2Start, p2End, filepathStocks, penalty_1, penalty_2):
        self.initialWeek = initialWeek
        self.endWeek = endWeek
        self.initialBudget = initialBudget
        self.minimalBudget = minimalBudget
        self.p1Start = p1Start
        self.p1End = p1End
        self.p2Start = p2Start
        self.p2End = p2End
        self.penalty_1 = penalty_1
        self.penalty_2 = penalty_2
        self.stocksData = pd.read_csv(filepathStocks)
        self.currentWeek = self.initialWeek
        self.currentBudget = self.initialBudget
        self.invested_ammount_per_stock = 0
        self.change_rate = 0
        self.earned_loss_money = 0
        self.buying_fees = 0
        self.reward = 0
        self.isTerminal()
        self.invested_budget = 0
        self.closing_prices = []
        self.indicators_1 = []
        self.indicators_2 = []
        self.profits = []
        self.number_of_dones = 0
        """
        Zatim nacitame data z prvnich 33 tydnu, v dalsi fazi vyvoje pouzijeme namisto tehle
            ↪ casti zakomentovany kod ve funkci step
        """
        for i in range(99):
            self.closing_prices.append(self.stocksData.iloc[i,1])
            self.indicators_1.append(self.stocksData.iloc[i,2])
            self.indicators_2.append(self.stocksData.iloc[i,3])
            self.profits.append(self.stocksData.iloc[i,4])
```

V kódu 3.2 nejprve přeškálujeme akce. Následně se řídíme logikou popsanou v části 3.1.1.

Kód 3.2: Definice funkce step ve třídě StockTrading

```
def step(self, actions):
    m1, n1, m2, n2, ratio = actions
    #preskalovani m1, n1, m2, n2, ratio to d1, e1, d2, e2, buy_ratio
    d1 = (p1S+1.0)*(self.p1End - self.p1Start)/2 + self.p1Start
    e1 = (p1E+1.0)*(self.p1End - d1)/2 + d1
    d2 = (p2S+1.0)*(self.p2End - self.p2Start)/2 + self.p2Start
```

```
    e2 = (p2E+1.0)*(self.p2End - d2)/2 + d2
    buy_ratio = (ratio+1)/2

    self.reward = 0
    done = self.isTerminal()
    self.buying_fees = 0
    #nakup a nasledny prodej
    if not done:
        #pokud je vybrany interval prazdny, pak pridej penalizaci
        if (d1 == e1) or (d2 == e2):
            self.reward += self.penalty_2  #penalty_2
        else:
            indices = []
            self.change_rate = 0.0
            """
            #zakomentovany kod pro pozdejsi pouziti
            for i in range(3):
                closing_prices.append(self.stocksData.iloc[3*self.currentWeek+i,1])
                indicators_1.append(self.stocksData.iloc[3*self.currentWeek+i,2])
                indicators_2.append(self.stocksData.iloc[3*self.currentWeek+i,3])
                profits.append(self.stocksData.iloc[3*self.currentWeek+i,4])

            #Splnuje najaka akcie vybrana kriteria?
            for i in range(3):
                if (indicators_1[i] >= d1 and indicators_1[i] <=e1) and (indicators_2[2] >= d2
                    ↪  and indicators_2[i] <=e2):
                indices.append(i)
            """
            #pokud splnuje nejaka akcie kriteria, tak pridej jeji index do listu indices
            for i in range(99):
                if (self.indicators_1[i] >= d1 and self.indicators_1[i] <=e1) and (self.
                    ↪ indicators_2[i] >= d2 and self.indicators_2[i] <=e2):
                    indices.append(i)   #pridani indexu do zaznamu

            if len(indices) > 0:
                self.invested_budget =min((self.currentBudget-self.minimalBudget)*buy_ratio,
                    ↪ self.currentBudget*0.95*buy_ratio)
                self.invested_amount_per_stock = self.invested_budget/ len(indices)

                #vypocet poplatku
                for index in indeces:
                    if self.invested_amount_per_stock/self.closing_prices[index] > 200:
                        self.buying_fees += 2 + 0.04*(math.ceil(self.invested_amount_per_stock
                            ↪ /self.closing_prices[index])-200)
                    else:
                        self.buying_fees += 2
                    self.change_rate += self.profits[index]
```

```
                self.reward = self.reward - 2*self.buying_fees + self.
                    ↪ invested_amount_per_stock * self.change_rate

            #zadna akcie nesplnuje vybrana kriteria
            else:
                self.reward += self.penalty_1 #penalty_1

    self.currentWeek += 1
    self.reward = self.reward/self.currentBudget #vypocet odmeny
    self.currentBudget = self.currentBudget + self.reward*self.currentBudget    #vypocet nove
        ↪ hodnoty kapitalu

    nextState = [self.currentBudget/self.initialBudget, (self.currentWeek - self.initialWeek)/
        ↪ float(self.endWeek + 1 - self.initialWeek)]    #vypocet noveho stavu
    return self.reward, nextState, done
```

Definici funkce `reset`, která přesune agenta do počátečního stavu, nalezneme v kódu 3.3.

Kód 3.3: Funkce reset v třídě StockTrading

```
def reset(self):
    self.currentBudget = self.initialBudget
    self.currentWeek = self.initialWeek
    nextState = [self.currentBudget/self.initialBudget, 0.0]
    return nextState, self.isTerminal()
```

3.3 TD3 algoritmus (Twin delayed deep deterministic policy gradient)

Algoritmus DDPG rovněž trpí na maximalizační vychýlení, viz 2.4.5, tj. nadhodnocuje funkční hodnoty kvalitativní funkce. V Q-učení maximalizační vychýlení pocházelo z použití jednoho odhadu pro výběr akce i pro její ohodnocení. Při aproximaci kvalitativní funkce či strategie je nepřesnost odhadu nevyhnutelná. Tato nepřesnost je dále zvýrazněna díky povaze metod temporální diference. V článku [25] popsali vznik maximalizačního vychýlení u algoritmu Hlubokého deterministického gradientu strategie. Dále navrhli vylepšený algoritmus TD3, která není tak náchylný k maximalizačnímmu vychýlení. V této části textu budeme vycházet z článku [25].

Oproti algoritmu DDPG je přidána ještě jedna sada kritiků, takže dohromady máme dvě neuronové sítě pro kritiky $Q(\cdot,\cdot;\theta_1)$, $Q(\cdot,\cdot;\theta_2)$ a jednu pro aktéra $\mu(\cdot;\omega)$, které jsou průběžně trénované, a k tomu další sadu tří cílových neuronových sítí $Q(\cdot,\cdot;\theta'_1)$, $Q(\cdot,\cdot;\theta'_2)$ a $\mu(\cdot;\omega')$. Architektury neuronových sítí pro kritiky a cílové kritiky jsou shodné stejně jako u algoritmu DDPG. Architektury neuronových sítí aktéra a cílového aktéra jsou stejné. Na začátku trénování nastavíme $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2$ a $\omega' \leftarrow \omega$.

Každý časový krok vybere agent akci pomocí aktérovy neuronové sítě a aby se docílilo prozkoumávání, přidává se prozkoumávací šum. Agentova akce ve stavu s bude proto

$$a = \mu(s;\theta) + \zeta, \text{ kde } \zeta \sim \text{zaklipuj}(\mathcal{N}(0,\sigma), -c, c),$$

kde ζ je prozkoumávací šum a $\mathcal{N}(0, \sigma)$ je hodnota z normálního rozdělení se střední hodnotou 0 a rozptylem σ. Funkce zaklipuj(x, a, b) definovaná $\forall x, a, b \in \mathbb{R} \wedge a \leq b$, která vrací hodnoty podle logiky

$$\text{zaklipuj}(x, a, b) = \begin{cases} a & \text{pro } x \leq a, \\ x & \text{pro } x \in (a, b), \\ b, & \text{pro } x \geq b. \end{cases}$$

Budeme trénovat kritiky $Q(\cdot, \cdot, \theta_1)$ a $Q(\cdot, \cdot, \theta_2)$ po každé interakci agenta s prostředím na dávce dat $\mathcal{B}$ pocházející ze zásobníku zkušeností $\mathcal{D}$. Cílová hodnota pro trénování obou kritiků se vypočte vztahem

$$y = r + \gamma \min_{j=1,2} Q(s', \mu(s'; \omega') + \zeta; \theta_j), \zeta \sim \text{zaklipuj}(\mathcal{N}(0, \sigma), -c, c),$$

kde bereme součet obdržené odměny r a γ násobek menší hodnoty predikce cílových kritiků ve stavu s' a akci, která se vypočetla pomocí cílového kritika s příspěvkem prozkoumavacího šumu ζ. Parametr $c \in \mathbb{R}$ vymezuje velikost prozkoumávacího okolí. Trénováním dvou sad kritiků a vybíraním minima jejich predikce se dociluje snížení maximalizačního vychýlení.

Aktérovu neuronovou síť trénujeme každých d kroků na dávce dat $\mathcal{B}$ o N zkušenostech, kde gradient funkce J aproximujeme vztahem

$$\nabla_\omega J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a; \theta_1)|_{s=s_i, a=\mu(s_i;\omega)} \nabla_\omega \mu_(s; \omega)|_{s=s_i},$$

kde z dávky $\mathcal{B}$ pužíváme jen stavy $\{s_i\}_{i=1}^N$. Cílové sítě upravujeme každých d kroků vztahy

$$\begin{aligned} \omega' &\leftarrow \tau\omega + (1 - \tau)\omega' \\ \theta'_j &\leftarrow \tau\theta_j + (1 - \tau)\theta'_j, j = \{1, 2\}, \end{aligned}$$

kde $\tau \ll 1$. Kombinace opožděných úprav parametru neuronové sítě aktéra ω a opožděných úprav parametrů cílových sítí má za následek snížení rozptylu odhadů kritiků.

Algoritmus 13 Dvojitý zpožděný hluboký deterministický gradient strategie (TD3)

Parametry algoritmu: rozptyl Normálního rozdělení $\sigma \in \mathbb{R}$, parametr funkce zaklipuj $c \in \mathbb{R}$, parametr úpravy aktérovy sítě a cílových sítí $d \in \mathbb{N}$, $\tau \ll 1$
Inicializace neuronových sítí kritiků $Q(\cdot,\cdot;\theta_1)$, $Q(\cdot,\cdot;\theta_2)$ a aktéra $\mu(\cdot;\omega)$ s libovolnými parametry θ_1, θ_2 a ω
Inicializace cílových neuronových sítí kritiků $Q(\cdot,\cdot;\theta'_1)$, $Q(\cdot,\cdot;\theta'_2)$ a aktéra $\mu(\cdot;\omega')$ s parametry $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \omega' \leftarrow \omega$
Inicializace zásobníku zkušeností $\mathcal{D}$ o velikosti M
Opakuj pro každou epizodu
  Inicializace počátečního stavu S
  Opakuj pro každý krok v epizodě
    $A \leftarrow \mu(S;\omega) + \mathcal{N}(0,\sigma)$
    Proveď akci A, pozoruj R, S', Done
    Ulož do zásobníku $\mathcal{D}$ zkušenost $(S, A, R, S', \text{Done})$
    Vyber ze zásobníku $\mathcal{D}$ dávku $\mathcal{B}$ o velikosti N
    Opakuj pro každé $(s_i, a_i, r_i, s'_i, \text{done}_i)$ v dávce $\mathcal{B}$
      $\tilde{a}_i \leftarrow \mu(s'_i;\omega') + \zeta, \zeta \sim$ zaklipuj $(\mathcal{N}(0,\sigma), -c, c)$
      $y_i = r_i + \gamma(\neg\text{done}_i) \min_{j=1,2} Q(s'_i, \tilde{a}_i; \theta'_j)$
    konec Opakuj
    Uprav neuronovou síť kritiků $Q(\cdot,\cdot,\theta_1)$ a $Q(\cdot,\cdot,\theta_2)$ minimalizací chyby

$$L = \frac{1}{N}\sum_{i,j}(y_i - Q(s_i, a_i; \theta_j))^2$$

    Každých d kroků uprav neuronovou síť aktéra pomocí gradientu

$$\nabla_\omega J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a; \theta_1)|_{s=s_i, a=\mu(s_i;\omega)} \nabla_\omega \mu_(s;\omega)|_{s=s_i}$$

    Každých d kroků uprav cílové sítě:

$$\theta'_j \leftarrow \tau\theta_j + (1-\tau)\theta'_j$$
$$\omega' \leftarrow \tau\omega + (1-\tau)\omega'$$

  konec Opakuj
konec Opakuj

V následující části popíšeme implementaci algoritmu TD3. Při implementaci kódu jsem čerpal z [40]. Výsledný program je složen z jednotlivých částí, a to ze zásobníku paměti, tříd pro vytvoření neuronových sítí, učící funkce `learn()` a inicializace agenta. Zasobník paměti jsme implementovali podobným způsobem jako v kódu 2.10. Pouze jsme pozměnili ukladání proměnné `done` do `self.terminal_memory[index]`, kde jsme přímo ukládali proměnnou `done`. Proměnná `self.n_action` by nyní měla význam dimenze akce.

V kódu 3.4 inicializujeme třídy `CriticNetwork` a `ActorNetwork` pro vytvoření neuronové sítě kritika, respektive aktéra. V inicializaci nejprve uložíme počet neuronů ve skrytých vrstvách. Zkonstruujeme neuronové vrstvy pomocí funkce `Dense(,)`, kde první argument je počet neuronů ve vrstvě a druhým argumentem je aktivační funkce, která se použije při výpočtu výstupních hodnot. Neuronová síť kritika bude mít dvě skryté vrstvy, ve kterých se používají aktivační funkce ReLU, a jeden výstupní uzel, ve kterém

se pouze vypočte předaktivační hodnota. Následuje definice funkce `call`, která načte na vstupu stav a akci. Následně se stav a akce, což jsou tenzory, sloučí funkcí `tf.concat()` a provede se dopředný průchod neuronovými vrstvami. Funkce pak vrací predikovanou hodnotu kvalitativní funkce pro stav a akci. Obdobným způsobem se implementuje neuronová síť aktéra, kde se využívají také dvě skryté vrstvy s aktivační funkcí ReLU a výstupní vrstva bude mít počet uzlů roven počtu dimenzí akce. Ve výstupní vrstvě používáme aktivační funkci tanh, a tak dostáváme na výstupu n-dimenzionální vektor, kde každá složka vektoru leží v intervalu $(-1, 1)$.

Kód 3.4: Vytvoření neuronových sítí pro algoritmus TD3

```
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras.layers import Dense

class CriticNetwork(keras.Model):
    def __init__(self, fc1_dims, fc2_dims):
        super(CriticNetwork, self).__init__()
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims

        self.fc1 = Dense(self.fc1_dims, activation='relu')
        self.fc2 = Dense(self.fc2_dims, activation='relu')
        self.q = Dense(1, activation=None)

    def call(self, inputs):
        states, actions = inputs
        q1_action_value = self.fc1(tf.concat([states, actions], axis=1))
        q2_action_value = self.fc2(q1_action_value)
        q = self.q(q2_action_value)
        return q

class ActorNetwork(keras.Model):
    def __init__(self, fc1_dims, fc2_dims, action_dims):
        super(ActorNetwork, self).__init__()
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims
        self.action_dims = action_dims

        self.fc1 = Dense(self.fc1_dims, activation='relu')
        self.fc2 = Dense(self.fc2_dims, activation='relu')
        self.q = Dense(self.action_dims, activation='tanh')

    def call(self, state):
        q1_value = self.fc1(state)
        q2_value = self.fc2(q1_value)
        q = self.q(q2_value)
        return q
```

Následovala by inicializace třídy `TD3Agent`, ve které bychom uložili vstupní hodnoty, vytvořili zásobník zkušeností, incicializovali neuronové sítě pro kritiky `self.critic_1()`, `self.critic_2()`, cílové kritiky `self.target_critic_1()`, `self.target_critic_2()` a také pro aktéry `self.actor()`, `self.target_actor()`. Dále v třídě `TD3Agent` definujeme funkce `save_model()`, `load_model()` pro uložení a načtení parametrů neuronových sítí, `choose_action()` pro vybrání akce, `update_network_parameters()` pro úpravu cílových neuronových sítí, `remember()` pro uložení zkušeností do zásobníku a funkce `learn()`, kterou popíšeme v kódu 3.5. Pokud je v zásobníku méně dat než požadovaná velikost dávky, pak netrénujeme. V opačném případě vybereme dávku ze zásobníku a převedeme vybrané zkušenosti na tenzory. Pak již budeme nahrávat operace s tenzory a pomocí cílových neuronových sítí vypočteme tenzor `target`. Následně spočteme ztrátovou funkci pro oba kritiky. Pomocí gradientů ztrátové funkce vůči trénovatelným parametrům neuronové sítě kritiků upravíme kritiky. Pokud počítadlo provedených učících cyklů je dělitelné hodnotou v proměnné `self.update_actor_interval`, pak trénujeme neuronovou síť aktéra a upravíme parametry cílových neuronových sítí.

Kód 3.5: Funkce učení pro algoritmus TD3

```
def learn(self):
    if self.memory.mem_cntr < self.batch_size:
        return 0
    #ucici faze – vyber davku dat ze zasobniku zkusenosti, pak konvertuj data do tensoru
    states, actions, rewards, new_states, terminals = self.memory.sample_buffer(self.
        ↪ batch_size)
    states = tf.convert_to_tensor(states, dtype=tf.float32)
    actions = tf.convert_to_tensor(actions, dtype=tf.float32)
    rewards = tf.convert_to_tensor(rewards, dtype=tf.float32)
    new_states = tf.convert_to_tensor(new_states, dtype=tf.float32)

    with tf.GradientTape(persistent = True) as tape:
        #predikce cilovych akci z novych stavu cilovym akterem, pridani zaklipovaneho sumu,
            ↪ ktery pochazi z normalniho rozdeleni se stredni hodnotou 0 a rozptylem 0.2
        target_actions = self.target_actor(new_states)
        target_actions = target_actions + tf.clip_by_value(np.random.normal(scale = 0.2),
            ↪ -0.5, 0.5)

        #zaklipovani, aby byla cilove akce v intervalu (self.action_min, self.action_max))
        target_actions = tf.clip_by_value(target_actions, self.action_min, self.action_max)

        #predikce hodnoty kvalitativni funkce v novych stavech a novych akcich (new_states,
            ↪ target_actions)
        q1_ = self.target_critic_1((new_states, target_actions))
        q2_ = self.target_critic_2((new_states, target_actions))
        q1_ = tf.squeeze(q1_, 1)
        q2_ = tf.squeeze(q2_, 1)
        #predikce hodnoty kvalitativni funkce ve stavech a akcich (states, actions)
        q1 = tf.squeeze(self.critic_1((states, actions)), 1)
        q2 = tf.squeeze(self.critic_2((states, actions)), 1)
        critic_value = tf.math.minimum(q1_, q2_)
        #vypocet cilove hodnoty pro trenovani kritiku
```

```
29          target = rewards + self.gamma*critic_value*(1-terminals)
30          #vypocet ztratove funkce
31          critic_1_loss = keras.losses.MSE(target, q1)
32          critic_2_loss = keras.losses.MSE(target, q2)
33      params_1 = self.critic_1.trainable_variables
34      params_2 = self.critic_2.trainable_variables
35      grad_1 = tape.gradient(critic_1_loss, params_1)
36      grad_2 = tape.gradient(critic_2_loss, params_2)
37      #uprava siti kritiku pomoci gradientu
38      self.critic_1.optimizer.apply_gradients(zip(grad_1, params_1))
39      self.critic_2.optimizer.apply_gradients(zip(grad_2, params_2))
40      self.learn_cntr += 1
41      if self.learn_cntr % self.update_actor_interval != 0:
42          return
43
44      with tf.GradientTape() as tape:
45          new_actions = self.actor(states)
46          critic_1_value = self.critic_1((states, new_actions))
47          #vypocet ztatove funkce (-) za to, ze neuronove site trenujeme pomoci gradientniho
                ↪ sestupu a my chceme aplikovat gradientni vzestup
48          actor_loss = -tf.math.reduce_mean(critic_1_value)
49      params = self.actor.trainable_variables
50      grads = tape.gradient(actor_loss, params)
51      #uprava site aktera pomoci gradientu
52      self.actor.optimizer.apply_gradients(zip(grads, params))
53      #uprava cilovych siti
54      self.update_network_parameters()
```

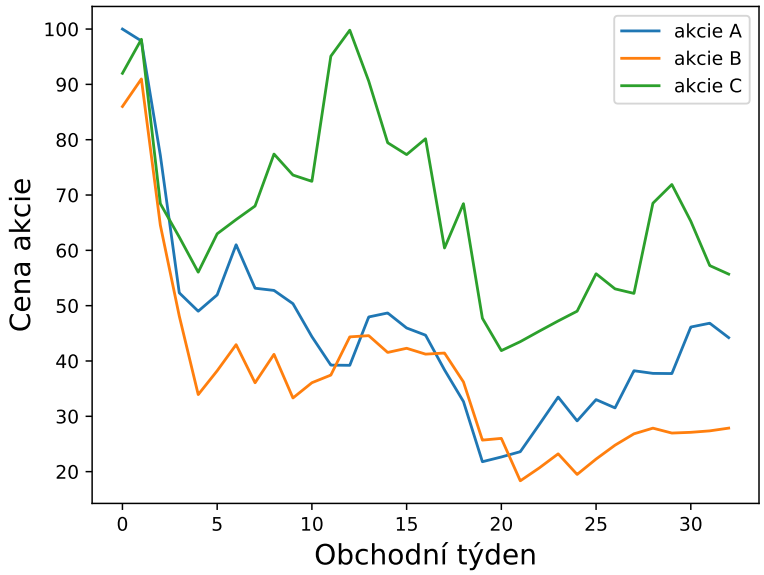
Kapitola 4

Výsledky

Abychom ověřili funkčnost a zjistili efektivitu TD3 algoritmu, zkonstruovali jsme syntetická data, u kterých víme, že optimální nastavení intervalů existuje a známe jaké je. První část kapitoly je věnována tvorbě těchto syntetických dat a ideálnímu nastavení intervalů. Dále jsou popsány výsledky testování TD3 algoritmu.

4.1 Představení syntetických dat

Máme akciové tituly A, B, C. Vývoj cen pro prvních třicet tři týdnů je vykreslen na obrázku 4.1.



Obrázek 4.1: Vývoj cen

Nyní popíšeme způsob výpočtu ceny akciového titulu v následujícím týdnu. Nechť nyní stojí akcie c a příslušné hodnoty indikátorů mají hodnotu p_1 a p_2. V závislosti na hodnotách p_1 a p_2 se určí profit_1 a profit_2 vyjadřující, jakým způsobem přispívá odpovídající hodnota indikátoru do relativního profitu. Relativní profit následně vypočteme jako profit = (1 + profit_1)(1 + profit_2) − 1. Cena akciového titulu v následujícím týdnu se vypočte jako $c\_new = c * (1 + \text{profit})$.

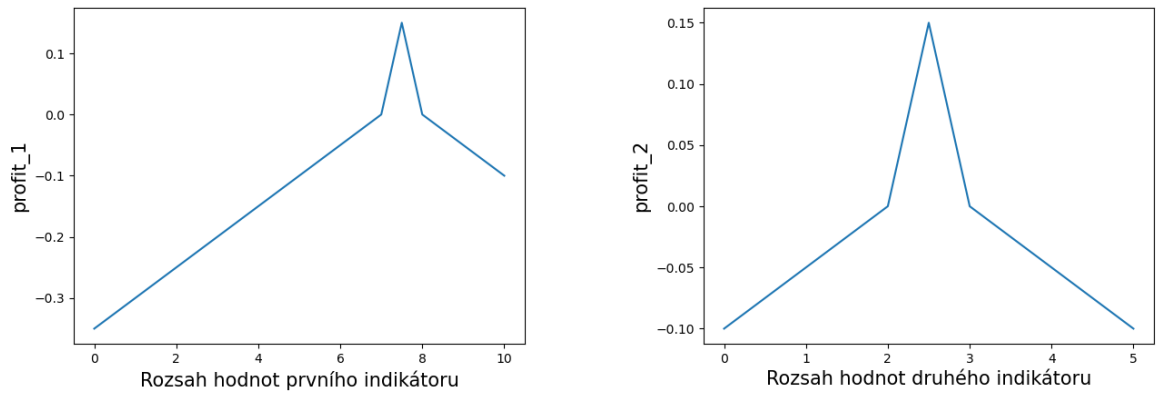
Rozsah prvního indikátoru je $\langle 0, 10\rangle$. Hodnota proměnné profit_1 lineárně roste od hodnoty -0.35 do 0 pro $p_1 \in \langle 0, 7\rangle$. Na intervalu $(7, 7.5\rangle$ roste se směrnicí 0.3 a následně na intervalu $(7.5, 8)$ klesá do nuly. Pro interval $\langle 8, 10\rangle$ proměnná profit_1 klesá se směrnicí 0.05. Slovní popis shrnuje rovnice

$$\text{profit\_1} = \begin{cases} (p_1 - 7) * 0.05, & \text{pokud } p_1 \leq 7, \\ min(p_1 - 7, 8 - p_1) * 0.3, & \text{pokud } p_1 \in (7, 8), \\ (8 - p_1) * 0.05, & \text{jinak.} \end{cases}$$

Rozsah druhého indikátoru je $\langle 0, 5\rangle$. Hodnota proměnné profit_2 lineárně roste od hodnoty -0.15 do 0 pro $p_1 \in \langle 0, 2\rangle$. Na intervalu $(2, 2.5\rangle$ roste se směrnicí 0.3 a následně na intervalu $(2.5, 3)$ klesá do nuly. Pro interval $\langle 3, 5\rangle$ proměnná profit_2 klesá se směrnicí 0.05. Slovní popis shrnuje rovnice.

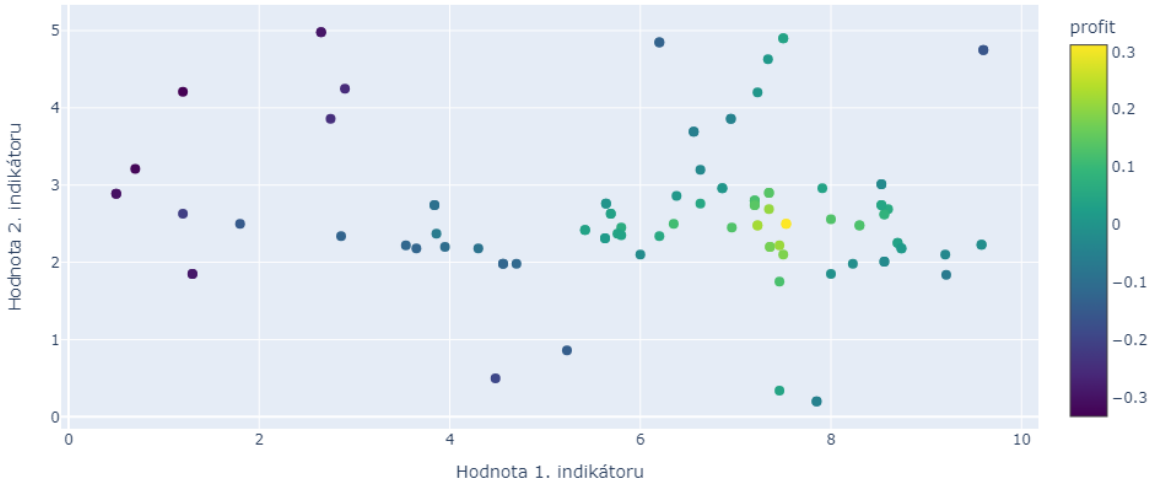
$$\text{profit\_2} = \begin{cases} (p_1 - 2) * 0.05, & \text{pokud } p_1 \leq 2, \\ min(p_1 - 2, 3 - p_1) * 0.3, & \text{pokud } p_1 \in (2, 3), \\ (3 - p_1) * 0.05, & \text{jinak.} \end{cases} \tag{4.1}$$

Grafická vizualizace závislosti profit_1 a profit_2 na p_1 a p_2 je vykreslena na obrázku 4.2.



Obrázek 4.2: Znázornění příspěvku pro první a druhý indikátor

Obrázek 4.3 zobrazuje všechny kombinace indikátorů obsažené v datovém souboru.



Obrázek 4.3: Grafické znázornění pokrytí prostoru indikátorů syntetickými daty

4.2 Trénování na syntetických datech

Při testování algoritmu jsme nastavili parametry prostředí následovně:

`initialWeek = 0, endWeek = 99, initialBudget = 40000, minimalBudget = 2000, p1Start = 0.0, p1End=10.0, p2Start = 0.0, p2End=5.0, penalty_1 = −50, penalty_2=−100`

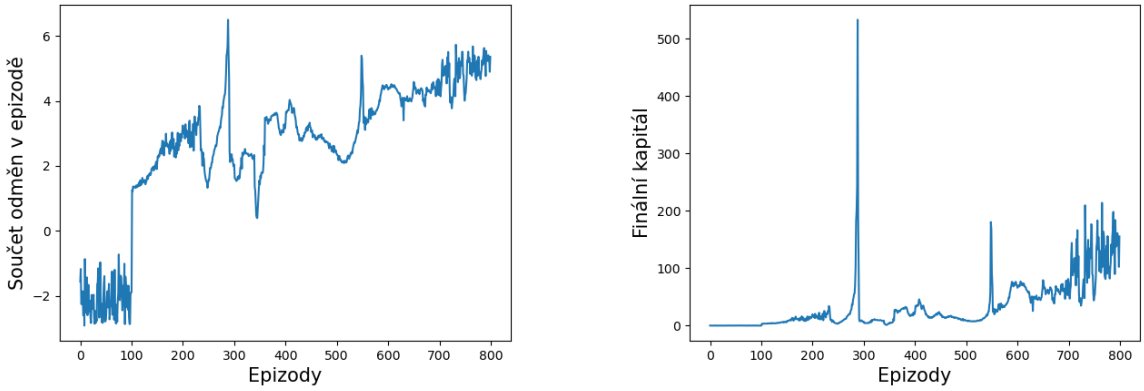
Pro prvotní testování jsme v prostředí načetli hodnoty akciových titulů pro prvních třicet tři týdnů a každý týden pak agent obchodoval na všech načtených datech. Objekt agenta jsme vytvořili příkazem

```
TD3Agent(alpha = 1e−4 , beta = 2e−3, gamma = 0, input_dims = 1, tau = 0.005, action_min =
↪  −1.0, action_dims = 5, action_max = 1.0, update_actor_interval = 2, mem_size = 20000,
↪  layer1_size_actor = 5, layer1_size_critic = 50, noise = 0.1, layer2_size_actor = 5,
↪ layer2_size_critic = 50, batch_size = 100, explore_time = 10000),
```

kde `alpha` odpovídá učícímu parametru pro neuronové sítě obou aktérů, `beta` odpovída učícímu parametru pro všechny neuronové sítě kritiků a `batch_size` pak udává velikost trénovací dávky.

Agenta jsme trénovali 800 epizod. Prvních sto tisíc kroků zadával agent do prostředí pouze náhodné akce. Posléze pro výběr akce agent používal neuronovou síť aktéra a šum pocházející z normálního rozdělení se střední hodnotou 0 a rozptylem 0.1. Součet šumu a predikce kritika byl zaklipován, aby se výsledná hodnota akce nacházela v intervalu $\langle -1, 1 \rangle$. Při výpočtu cílové akce, která se používá k odhadu cílové hodnoty pro trénování obou kritiků, jsme přidali k predikci cílového kritika v novém stavu šum, který pochází z normálního rozdělení se střední hodnotou 0 a rozptylem 0.2 a byl zaklipován na interval $\langle -0.5, 0.5 \rangle$.

Trénování trvalo 1578 sekund a vykreslení finálního součtu odměn a finálního kapitálu, vyjádřeného jako násobek počátečního kapitálu, tj. finálního stavu, pro každou epizodu můžeme vidět na obrázku 4.4. Maxima pro součet odměn bylo dosaženo v dvě stě osmdesáté deváté epizodě, kde jeho hodnota byla 6.5. Ve stejné epizodě se dosáhlo maximálního finálního kapitálu a to 533-násobku počátečního kapitálu. V poslední epizodě byl součet odměn 5.35 a finální stav 155.5.



Obrázek 4.4: Vývoj součtu odměn a finálního kapitálu s prozkoumávacím šumem

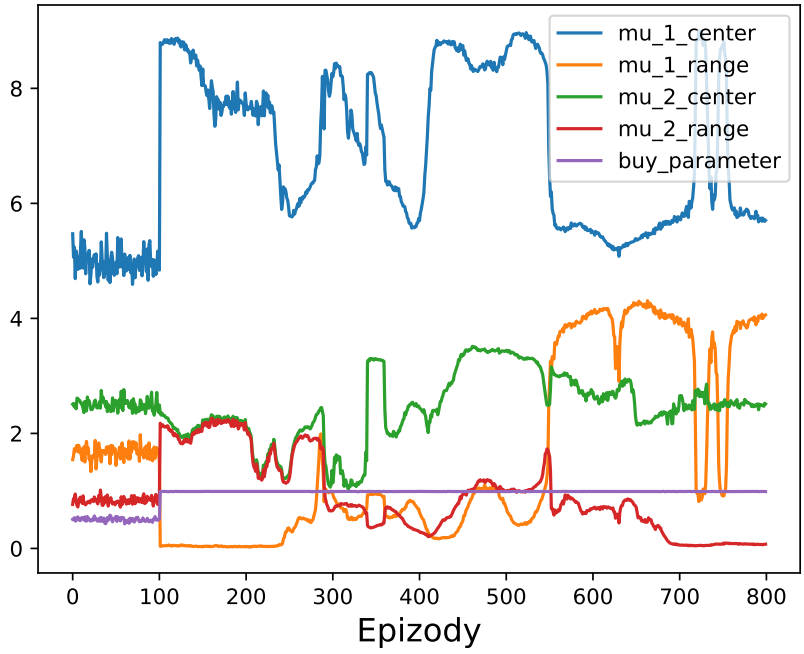
V každé epizodě jsme pro každý indikátor nalezli průměrný střed vybraných intervalu a průměrnou velikost okolí za celou epizodu a také průměrnou hodnotu parametru velikosti nákupu. Nalezené hodnoty pro každou epizodu jsou vykresleny v obrázku 4.5, kde střed výběru intervalu pro první indikátor je označen jako mu_1_center a jeho okolí mu_1_range. Analogické značení je použito pro výběr intervalu druhého indikátoru a pro průměr velikosti nákupu jsme použili značení buy_ratio. Na obrázku 4.5 jde hezky vidět prvních 100 průzkumných epizod. Agent se po sto epizodách úspěšně naučil nastavovat buy_ratio na hodnotu 1 a dále na konci trénování se blížili hodnota mu_2_center ke správné hodnotě 2.5. Výběry intervalu pro první indikátor už nebyly tak dobré. Můžeme pozorovat, že když hodnota mu_1_center roste, tak většinou hodnota mu_2_range klesá. To bylo způsobeno tím, že čtvrtá složka vektoru se blížila hodnotě 1, a tak se vybíraly akciové tituly s hodnotou prvního indikátoru až do konce rozsahu prvního indikátoru.

Následně na obrázku 4.6 jsou uvedeny výpisy čtyř epizod v průběhu trénování, kde jsme použili stejné značení jako na obrázku 4.5, pouze z významu značení se vytratil význam průměru. Vlevo nahoře vidíme časový průběh úvodní epizody bez prozkoumávacích kroků. Vpravo nahoře je uvedena epizoda, ve které se dosáhlo nejlepších výsledků. V dolní části obrázku můžeme pozorovat vývoj strategie. V první části epizody se již hodnoty středů vybraných intervalů pohybují kolem ideální hodnoty. Dále pak v pozdější fázi epizody se agent setkal se stavy, ve kterých se nikdy nenalézal, a proto jeho chování nebylo optimální. Navíc první složka stavu byla veliká a při průchodu neuronové sítě byla druhá složka akce blízko hodnoty 1. Možným řešením na tento problém by mohlo být použití větší míry prozkoumávání v pozdější fázi epizody. Dalším možným vylepšením se zdá být upravení první složky stavu, například logaritmickým škálováním. Tím by se docílilo zmenšení vstupu do neuronové sítě.

Musíme konstatovat, že agent nedosahoval maximálních výdělků. To bylo způsobeno také krátkou dobou trénování. Proto jsme dále implementovali průběžné ukládání a načtení. To se hodí i tehdy, když nemůžeme zajistit souvislý běh programu a výpočet je potřeba rozdělit na části. Při novém spuštění chceme začínat ve stejném okamžiku, proto ukládáme model neuronových sítí, data uložená v zásobníku a také počítadlo záznamů v zásobníku. Do budoucna chceme v dalším vývoji investiční strategie pokračovat, a to trénováním agenta po delší dobu a aplikováním dále popsaných možných vylepšení.

Pro další trénování na datech pocházející z americké burzy lze přidat do stavu další ukazatele, které mohou ovlivňovat vývoj cen. Těmito ukazateli by mohli být: míra inflace, vývoj HDP, stav peněžní zásob, míra nezaměstnanosti, nálada ve společnosti. Dalším možným vylepšením investiční strategie by mohlo být obchodování akcie po delší dobu.

V následující části jsme prezentovali speciální případ, kde máme stav reprezentovaný pouze jednou



Obrázek 4.5: Vývoj průměru vybraných intervalů

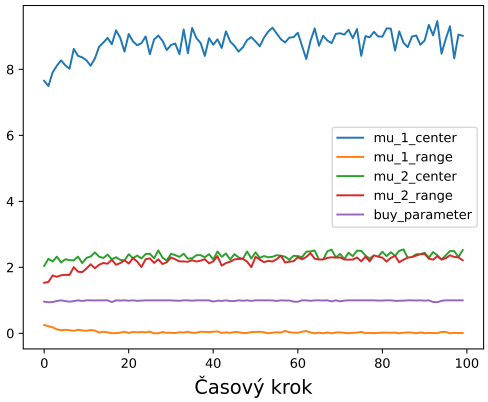
složkou, která byla v každém týdnu nastavena na hodnotu 1 a ukázalo se, že volba stavu má významný dopad na výdělečnost agenta. Díky takového nastavením jsme získali velmi dobrou aproximaci nejlepšího výsledku. Agenta jsme incializovali stejně jako v předchozí části až na to, že nyní byl počet prozkoumávajících kroků 1000.

Agent byl trénován 400 epizod se stejným průzkumným šumem jako v první části.

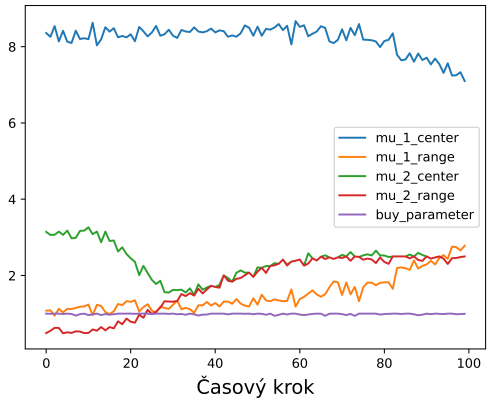
Na obrázku 4.7 vidíme v levé části vykreslený součet odměn na konci každé epizody trénování. Napravo pak výpis finálního kapitálu pro každou epizodu vyjádřený jako násobek počátečního kapitálu. V šedesáté páté epizodě se dosahuje maxima, jak finálního kapitálu o hodnotě 2×10^6-násobku počátečního kapitálu, tak i součtu odměn o hodnotě 15.76. V levé části obrázku lze také hezky vidět pro prvních deset epizod náhodné prozkoumávání. Čtyřstá epizoda měla součet odměn roven 10.08 a finální stav byl 10^4. Trénování na CPU notebooku (12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz) trvalo 620 sekund.

Nenechme se zmást na první pohled špatnými výsledky pozdější fáze trénování. Tento výsledek je způsoben tím, že se interval výběru zmenšoval a díky přidání náhodného šumu se interval výběru posunul do oblasti, kde nebyl žádný akciový titul. Průměrné hodnoty výběru intervalu jsme vynesli do obrázku pro každou epizodu jsou vykresleny v obrázku 4.8, kde jsme použili stejné význam značení jako na obrázku 4.5.

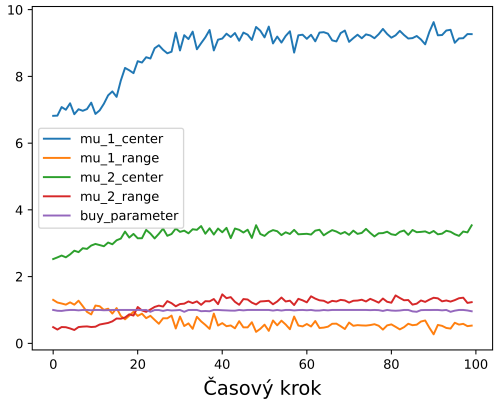
Průběh první a čtyřisté epizody je vykreslen na obrázku 4.9.



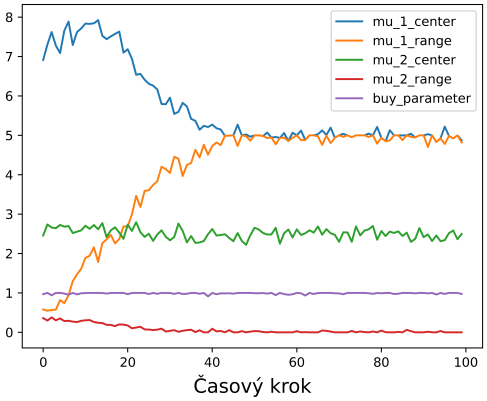
(a) Průběh epizody 101



(b) Průběh epizody 289

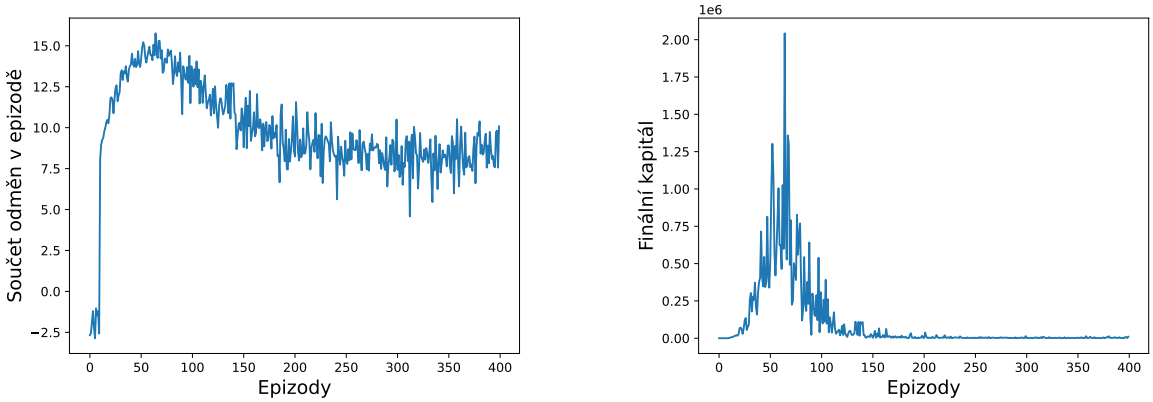


(c) Průběh epizody 537

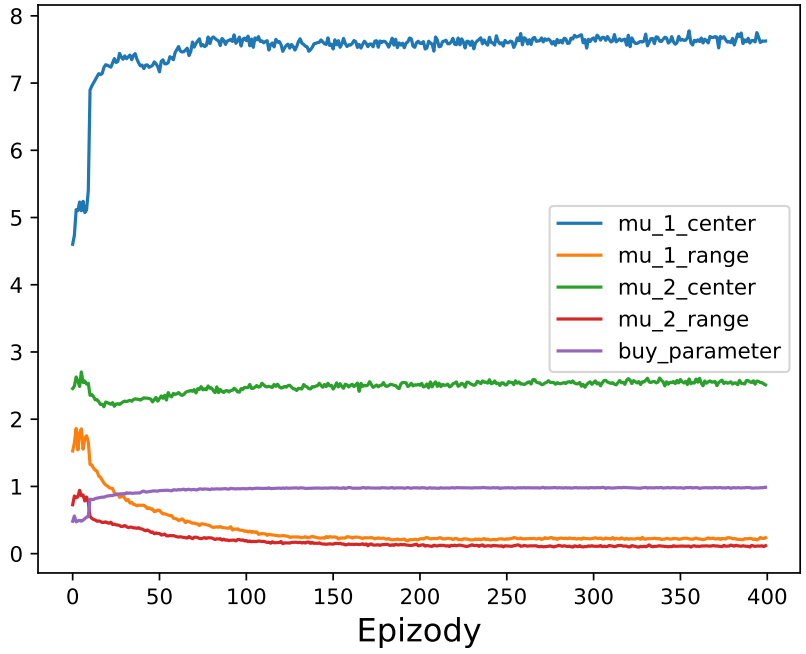


(d) Průběh epizody 800

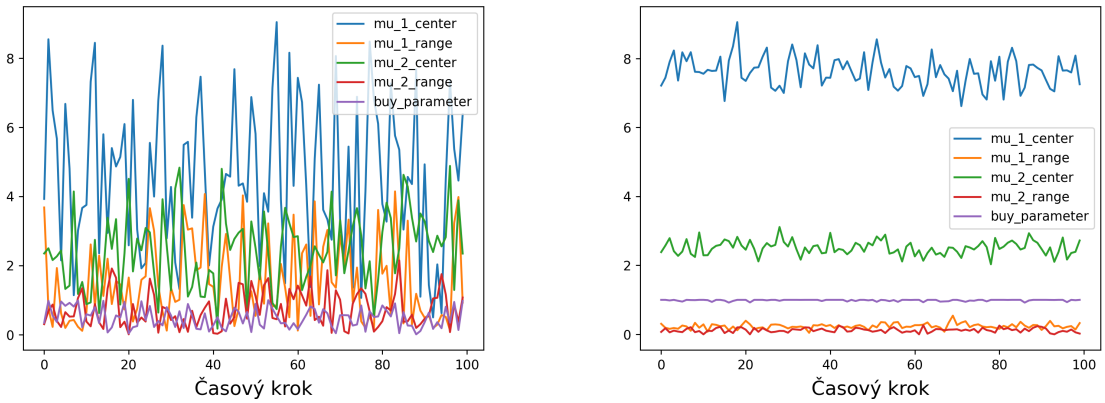
Obrázek 4.6: Vykreslení průběhu epizod



Obrázek 4.7: Vývoj součtu odměn a finálního kapitálu s prozkoumávacím šumem pro zafixovanou hodnotu stavu



Obrázek 4.8: Vývoj průměru vybraných intervalů



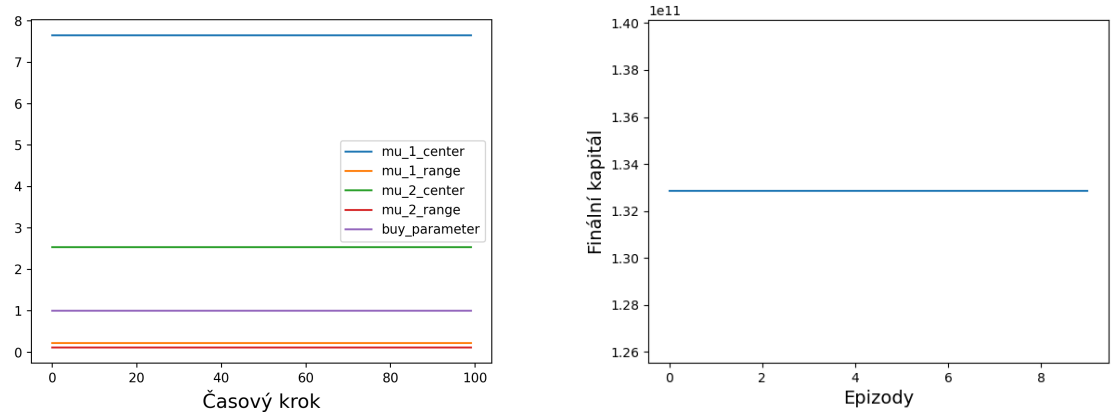
(a) Průběh epizody 1

(b) Průběh epizody 400

Obrázek 4.9: Vykreslení průběhu první a poslední epizody pro zafixovaný stav

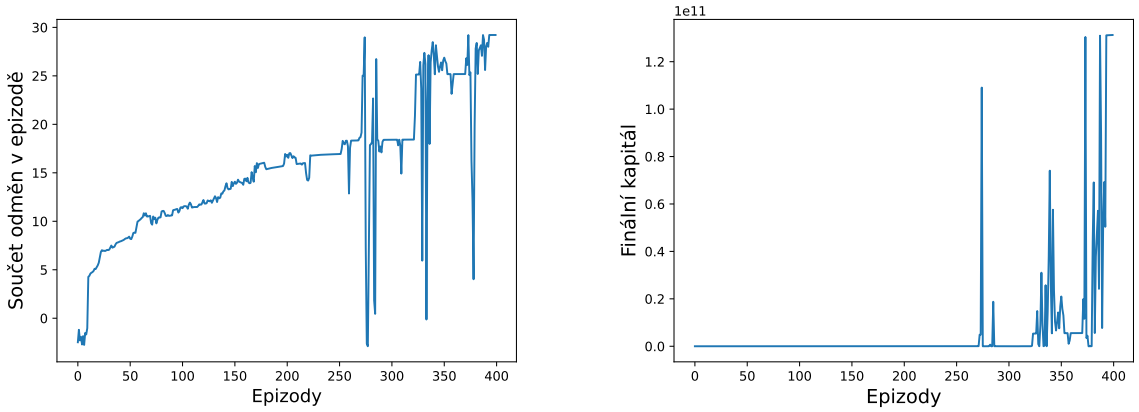
Následně jsme načetli agenta a otestovali ho na deseti epizodách. Nepoužívali jsme prozkoumávací šum a počet prozkoumávacích kroků jsme nastavili na nulu. Při testování jsme nepoužívali funkci `learn()`. Na konci každé epizody měl agent na účtu 1.328×10^{11}-násobek počátečního kapitálu a součet odměn byl 29.23. Testování trvalo 2.86 sekundy. Průběh první epizody a výpis finálního kapitálu můžeme vidět na obrázku 4.10. Stejná strategie jako v poslední epizodě, viz obrázek 4.9, nyní bez přidání šumu dosáhla o sedm řádu většího výdělku. Vidíme tak, že přidání šumu má kritický dopad na výdělek strategie.

Následně jsme ještě trénovali agenta 400 epizod pro zafixovaný stav. Pro výběr akce, která se za-

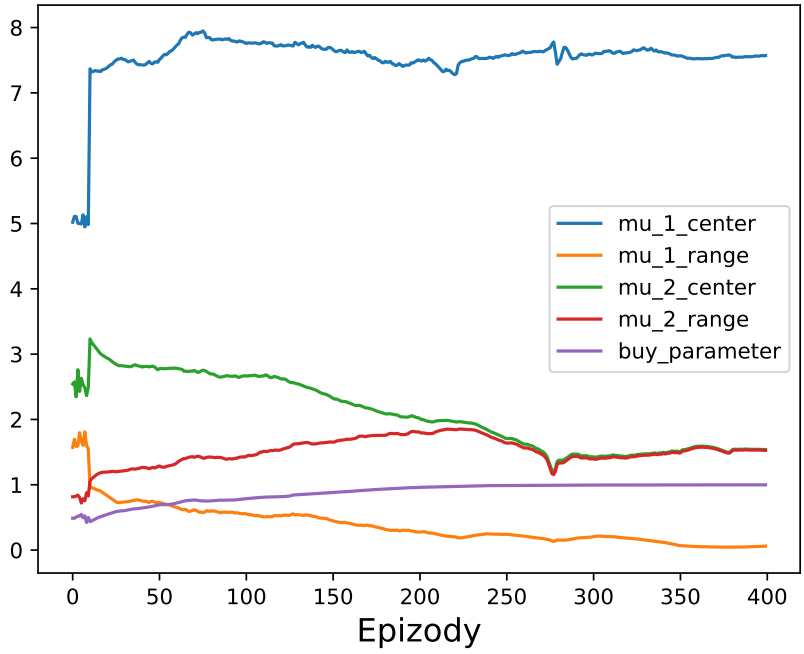


Obrázek 4.10: Vývoj první epizody a finálního kapitálu v průběhu testování bez prozkoumavacího šumu

dávala do prostředí, jsme nepoužívali prozkoumávací šumu. Prozkoumávací šum se pouze používal v predikci cílové hodnoty a pocházel z normálního rozdělení se stejnými parametry jako v předchozí části. Obrázky 4.11 a 4.12 ilustrují výsledky trénování pro právě popsané nastavení. Je překvapivé, že i když agent nepoužíval pro výběr akcí, které zadá do prostředí, prozkoumávací šum, tak nalezl nejlepší strategii.



Obrázek 4.11: Vývoj součtu odměn a finálního kapitálu v průběhu trénování pro zafixovaný stav bez použití prozkoumavacího šumu pro výběr akce, která se zadá do prostředí



Obrázek 4.12: Vývoj průměru vybraných intervalů pro zafixovaný stav bez použití prozkoumavacího šumu pro výběr akce, která se zadá do prostředí

Závěr

V rámci bakalářské práce bylo představena souhrnná rešerše na poli zpětnovazebného učení pro Markovovy rozhodovací procesy. Většina důležitých tvrzení byla matematicky popsána a řádně dokázána. Postupovali jsme od čistě teoretických metod dynamického programování přes tabulkové metody, kde jsme odhadovali různými způsoby hodnotové nebo kvalitativní funkce, až jsme se dostali k použití neuronových sítí pro aproximace funkcí. V závěru rešeršní části jsme strategie parametrizovali a propojili je s neuronovými sítěmi.

Zároveň se podařilo vymyslet, matematicky formulovat a programátorsky implementovat úlohu, na které byla řada algoritmů z rešeršní části otestována a jejich výsledky porovnány. Zdrojové kódy byly umístěny na Github

`https://github.com/jamichy/BP_turista_RL`

a jsou k dispozici pro vyzkoušení čtenáři tohoto textu.

Kromě rešeršní práce jsme představili konkrétní investiční strategii využívající fundamentální data, pro niž jsme zkonstruovali prostředí a také syntetická data pro účely ladění. Dále jsme prezentovali algoritmus agenta, který bude tuto investiční strategii realizovat. Následně jsme trénovali agenta na syntetických datech a výsledky trénování jsme popsali.

Největším přínosem práce vidíme v rešeršní části: ucelený výklad RL v češtině, s hlubokým matematickým vhledem a kombinovaný s s průběžnými praktickými demonstracemi kódu.

V praktické části byl vyvinut po technické stránce funkční algoritmus založený na pokročilé metodě TD3. Při testech na syntetických datech jsme si ověřili, že správná definice prostředí (akcí, stavů) je pro úspěšné řešení problémů klíčová. Vyvinutý algoritmus představuje solidní základ pro další vývoj a experimentování s reálnými daty z americké burzy.

Seznam kódů

Seznam algoritmů

Literatura

[1] R. S. Sutton, A. G. Barto: *Reinforcement learning: An introduction*, The MIT Press, Cambridge, MA, [2018].

[2] S.M. Farrukh Akhtar: *Practical Reinforcement Learning*, Packt Publishing, Birmingham, [2017].

[3] A. Plaat: *Deep ReinforcementLearning*, Springer, [2022].

[4] M. Sugiyama: *Statistical Reinforcement Learning: Modern Machine Learning Approaches*, CRC Press, [2015].

[5] Ke-Lin Du, M. N. S. Swamy: *Neural Networks and Statistical Learning* (second edition), Springer, [2019].

[6] K. P. Murphy: *Machine Learning: A Probabilistic Perspective*, MIT Press, Cambridge, MA, [2012].

[7] *5 Types of Classification Algorithms in Machine Learning. MonkeyLearn - Text Analytics*, [online, cit. 2023-05-09]. Dostupné z: https://monkeylearn.com/blog/classification-algorithms/

[8] D. Nikolaiev: *Dimensionality Reduction cheat sheet*, [online, cit. 2023-05-09]. Dostupné z: https://towardsdatascience.com/dimensionality-reduction-cheatsheet-15060fee3aa

[9] L. Weng: *The Multi-Armed Bandit Problem and Its Solutions*, [online, cit. 2023-06-29]. Dostupné z: https://lilianweng.github.io/posts/2018-01-23-multi-armed-bandit/

[10] M. Hahsler, H. Kamalzadeh: *POMDP: Introduction to Partially Observable Markov Decision Processes*, [online, cit. 2023-07-01]. Dostupné z: https://cran.r-project.org/web/packages/pomdp/vignettes/POMDP.html

[11] R. Gautron et al.: *gym-DSSAT: a crop model turned into a Reinforcement Learning environment*, [online, cit. 2023-07-03]. Dostupné z: https://doi.org/10.48550/arXiv.2207.03270

[12] *DSSAT Overview*, [online, cit. 2023-07-03]. Dostupné z: https://dssat.net/about/

[13] M. H. Kalos, P. A. Whitlock: *Monte Carlo Methods, Second Edition.*, WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim, [2008].

[14] *Complete Tutorial on Dynamic Programming (DP) Algorithm*, [online, cit. 2023-07-04]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-dynamic-programming-data-structures-and-algorithm-tutorials/?ref=lbp

[15] A. Choudhary: *Nuts & Bolts of Reinforcement Learning: Model Based Planning using Dynamic Programming*, [online, cit. 2023-07-04]. Dostupné z: https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/

[16] P. Meckoni: *Value Iteration Algorithm for a Discrete Markov Decision Process*, [online, cit. 2023-07-05]. Dostupné z: https://people.umass.edu/meckoni/articles/value-iteration-mdp/

[17] A. Ivora: *Dynamic Programming*, [online, cit. 2023-07-05]. Dostupné z: https://www.fi.muni.cz/~xivora/files/Chapter4_Dynamic_Programming.pdf

[18] D. Nikolaiev: *Clustering cheat sheet*, [online, cit. 2023-07-07]. Dostupné z: https://towardsdatascience.com/clustering-cheat-sheet-dcf72259abb6

[19] C. C. Aggarwal: *Neural Networks and Deep Learning: A Textbook*. Springer, [2018].

[20] V. Mnih et al.: *Playing Atari with Deep Reinforcement Learning*, arXiv:1312.5602 [2013], https://doi.org/10.48550/arXiv.1312.5602

[21] V. Mnih, K. Kavukcuoglu, D. Silver et al.: *Human-level control through deep reinforcement learning*. Nature 518(7540), [2015], p. 529-533. ISSN 0028-0836. https://doi.org/doi:10.1038/nature14236

[22] H.van Haselt et al.: *Deep Reinforcement Learning with Double Q-learning*, arXiv:1509.06461 [2015], https://doi.org/10.48550/arXiv.1509.06461

[23] D. Silver et al.: *Deterministic Policy Gradient Algorithms*, ICML, [2014]. Dostupné z: https://proceedings.mlr.press/v32/silver14.html

[24] T. P. Lillicrap et al.: *Continuous control with deep reinforcement learning*, arXiv:1509.02971v6 [2019], https://doi.org/10.48550/arXiv.1509.02971

[25] S. Fujimoto et al.: *Addressing Function Approximation Error in Actor-Critic Methods*, arXiv:1802.09477 [2018], https://doi.org/10.48550/arXiv.1802.09477

[26] Veronika Deketová: *Analýza vlivu fundamentálních dat na výkonnost akcií pomocí metod strojového učení*. Bakalářská práce, FJFI ČVUT v Praze, [2022].

[27] *Akciové trhy*, [online, cit. 2023-07-30]. Dostupné z: https://www.czechwealth.cz/slovnik-pojmu/akciove-trhy

[28] *Co je broker?*, [online, cit. 2023-07-30]. Dostupné z: https://www.moneta.cz/slovnik-pojmu/detail/broker

[29] *Holidays & Trading Hours*, [online, cit. 2023-07-30]. Dostupné z: https://www.nyse.com/markets/hours-calendars

[30] *Investiční trojúhelník*, [online, cit. 2023-07-30]. Dostupné z: https://www.mmgfg.cz/blog/investicni-trojuhelnik/

[31] Christine Majaski:*Fundamental vs. Technical Analysis: What's the Difference?*, [online, cit. 2023-07-30]. Dostupné z: https://www.investopedia.com/ask/answers/difference-between-fundamental-and-technical-analysis/

[32] *Gymnasium Documentation*, [online, cit. 2023-08-01]. Dostupné z: https://gymnasium.farama.org/index.html

[33] *Tensorforce: a TensorFlow library for applied reinforcement learning*, [online, cit. 2023-08-02]. Dostupné z: https://tensorforce.readthedocs.io/en/latest/

[34] *Alpha Go*, [online, cit. 2023-08-02]. Dostupné z: https://www.deepmind.com/research/highlighted-research/alphago

[35] *RLlib: Industry-Grade Reinforcement Learning*, [online, cit. 2023-08-02]. Dostupné z: https://docs.ray.io/en/latest/rllib/index.html

[36] *Introducing ChatGPT*, [online, cit. 2023-08-02]. Dostupné z: https://openai.com/blog/chatgpt

[37] T. Kabbani, E. Duman: *Deep Reinforcement Learning Approach for Trading Automation in The Stock Market*, arXiv:2208.07165, [2022]. https://doi.org/10.48550/arXiv.2208.07165

[38] *TensorFlow: An open-source machine learning framework for everyone*, [online, cit. 2023-08-02]. Dostupné z: https://www.tensorflow.org/

[39] Phil Tabor: *Double Deep Q Learning Is Simple with Keras*, [online, cit. 2023-08-02]. Dostupné z: https://www.youtube.com/watch?v=UCgsv6tMReY

[40] Phil Tabor: *Artificial Intelligence Learns to Walk with Actor Critic Deep Reinforcement Learning | TD3 Tutorial*, [online, cit. 2023-08-02]. Dostupné z: https://www.youtube.com/watch?v=1lZ0B2S17LU