UNIVERZITA SV. CYRILA A METODA V TRNAVE
FAKULTA PRÍRODNÝCH VIED

RL-TOOLKIT: NÁVRH A IMPLEMENTÁCIA SADY NÁSTROJOV PRE UČENIE S POSILŇOVANÍM V ROBOTIKE

Diplomová práca

2023 **Bc. Martin Kubovčík**

UNIVERZITA SV. CYRILA A METODA V TRNAVE
FAKULTA PRÍRODNÝCH VIED

RL-TOOLKIT: NÁVRH A IMPLEMENTÁCIA SADY NÁSTROJOV PRE UČENIE S POSILŇOVANÍM V ROBOTIKE

Diplomová práca

Študijný program: aplikovaná informatika
Študijný odbor: 2511 informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: prof. RNDr. Jiří Pospíchal, DrSc.

2023 **Bc. Martin Kubovčík**

UCM Trnava
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Martin Kubovčík
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	18. - informatika
Typ záverečnej práce:	Diplomová práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: RL-Toolkit: Návrh a implementácia sady nástrojov pre učenie s posilňovaním v robotike

Anotácia:
1. Oboznámte sa problematikou učenia s posilňovaním pre simulovaných robotov v spojitých stavových a akčných priestoroch.
2. Analyzujte možnosti vybraných algoritmov učenia s posilňovaním, vybrané algoritmy implementujte či preberte a skombinujte do nadväzujúceho celku.
3. Navrhnite a na simulovanom robotovi overte vlastnosti vybraných algoritmov.
4. Navrhnite prípadne modifikácie týchto algoritmov, preštudujte citlivosť nástrojov na nastavenie parametrov.
5. Porovnajte s existujúcimi riešeniami, zhodnoťte výsledky a diskutujte možnosť úpravy platformy pre učenie reálnych robotov.

Vedúci:	prof. RNDr. Jiří Pospíchal, DrSc.
Katedra:	KAI - Katedra aplikovanej informatiky
Vedúci katedry:	PaedDr. Mgr. Miroslav Ölvecký, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 28.09.2021

Dátum schválenia: 01.12.2021

PaedDr. Mgr. Miroslav Ölvecký, PhD.
vedúci katedry

Čestné vyhlásenie

Čestne vyhlasujem, že svoju diplomovú prácu s názvom RL-Toolkit: Návrh a implementácia sady nástrojov pre učenie s posilňovaním v robotike som vypracoval samostatne a všetku použitú literatúru uvádzam v zozname bibliografických odkazov.

Trnava dňa 9.5.2023 .....

podpis

Poďakovanie

Ďakujem prof. RNDr. Jiřímu Pospíchalovi, DrSc. za jeho cenné rady, odborné vedenie a pripomienky pri spracovaní témy a vypracovaní diplomovej práce.

Abstrakt

Kubovčík, Martin Bc.: *RL-Toolkit: Návrh a implementácia sady nástrojov pre učenie s posilňovaním v robotike.* [Diplomová práca]. Univerzita sv. Cyrila a Metoda v Trnave, Fakulta prírodných vied, Katedra aplikovanej informatiky. Školiteľ: prof. RNDr. Jiří Pospíchal, DrSc. Trnava, 2023. Počet strán práce 76 s.

Diplomová práca je zameraná na samoučenie sa robotov v simulovaných prostrediach, kde musia plniť zadané úlohy pričom získavajú skóre, ktoré sa snažia maximalizovať. Cieľom je získať čo najlepšie skóre, čo predstavuje najideálnejšie splnenie danej úlohy. K dosiahnutiu cieľa je použitá umelá neurónová sieť, ktorá transformuje stavový vektor predstavujúci merania senzorov na akčný vektor, teda pohyby motorov robota. Robot využíva učenie posilňovaním, ktorého princípom je maximalizácia Q-hodnoty predstavujúca kvalitu daného prechodu z aktuálneho stavu do nasledovného stavu vplyvom vykonanej akcie. Pre presnejšie vyjadrenie Q-hodnôt je použitá distribúcia pravdepodobností týchto hodnôt učená kvantilovou regresiou. Pre ukladanie si interakcií získaných počas hrania v simulácii je použitý databázový server Reverb. Výsledky sú porovnané s pôvodnými prácami autorov použitých algoritmov a dokazujú zlepšenie oproti pôvodným výsledkom. Keďže ide o sadu nástrojov, je k dispozícii niekoľko nástrojov, ktoré spolu tvoria RL-Toolkit. Konkrétne ide o databázový server, monitorovací nástroj Weights & Biases a 3 rôzne populárne prostredia simulácií, Gymnasium, DeepMind Control Suite a PyBullet. Na záver je vytvorený aj návrh architektúry pre reálneho robota, ktorý aplikuje RL-Toolkit do reálneho sveta.

Kľúčové slová: soft actor-critic model, hlboká neurónová sieť, Gymnasium, robotika, Q-hodnota, kvantilová regresia, Docker

Abstract

Kubovčík, Martin Bc.: *RL-Toolkit: Design and implementation of a set of tools for reinforcement learning in robotics*. [Diploma's Thesis]. University of Ss. Cyril and Methodius in Trnava, Faculty of Natural Sciences, Department of Applied Informatics. Supervisor: prof. RNDr. Jiří Pospíchal, DrSc. Trnava, 2023. Number of pages 76 p.

The thesis focuses on self-learning of robots in simulated environments where they have to perform given tasks while obtaining scores that they try to maximize. The goal is to obtain the best score, which represents the most ideal performance of a given task. To achieve the goal, an artificial neural network is used to transform the state vector representing the sensor measurements into an action vector, i.e., the movements of the robot's motors. The robot uses reinforcement learning, whose principle is to maximize a Q-value representing the quality of a given transition from the current state to the next state as a result of the performed action. For a more accurate representation of the Q-values, the probability distribution of these values learned by quantile regression is used. The Reverb database server is used to store the interactions obtained during the simulations. The results are compared with the original work of the authors of the algorithms used and show an improvement over the original results. Several tools are provided which together form the RL-Toolkit. Specifically, these are a database server, a Weights & Biases monitoring tool, and 3 different popular simulation environments such as Gymnasium, DeepMind Control Suite a PyBullet. Finally, an architecture design for a real robot that applies RL-Toolkit to the real world is also created.

Keywords: soft actor-critical model, deep neural network, Gymnasium, robotics, Q-value, quantile regression, Docker

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam skratiek

Adam – adaptive moment estimation/adaptívny odhad momentu

gSDE – generalized state-dependent exploration/algoritmus generalized state-dependent exploration

MDP – Markov decision process/Markov rozhodovací proces

QR-DQN – quantile regression deep Q network/algoritmus quantile regression deep Q network

ReLU – rectified linear unit/usmernená lineárna jednotka

RL – reinforcement learning/učenie posilňovaním

SAC – soft actor-critic/algoritmus soft actor-critic

SPI – samples per insert/pomer vzorkovania ku vkladaniu

TanH – hyperbolic tangent/hyperbolický tangens

TD – temporal difference/časová rozdielnosť

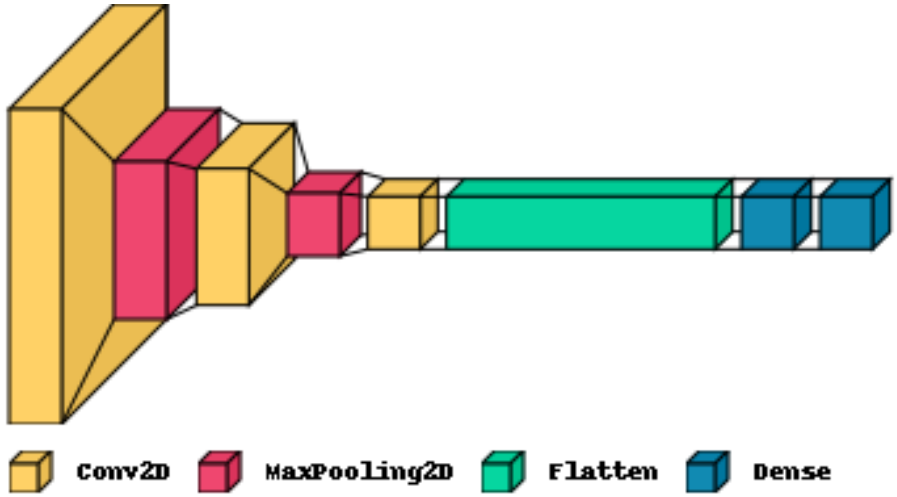
TD3 – twin delayed deep deterministic policy gradient/algoritmus twin delayed deep deterministic policy gradient

Úvod

Roboty sa stali neodmysliteľnou súčasťou nášho života a využívajú sa v rôznych oblastiach, ako je výroba, zdravotníctvo, vzdelávanie a zábava. Ovládanie robotov však nebolo vždy jednoduché, najmä v začiatkoch, keď boli roboty prvýkrát predstavené. Na začiatku boli roboty ovládané prostredníctvom vopred naprogramovaných inštrukcií, ktoré boli pevne zakódované v ich systémoch. Tieto inštrukcie hovorili robotovi, čo má v danej situácii robiť, a robot ich bez odchýlky dodržiaval. Tento prístup fungoval dobre pri jednoduchých úlohách, ale nebol vhodný na zložitejšie úlohy, ktoré vyžadovali, aby sa robot prispôsoboval meniacim sa podmienkam. S narastajúcou zložitosťou úloh boli obmedzenia vopred naprogramovaných pokynov stále zjavnejšie. Napríklad robot naprogramovaný na montáž auta v továrni mohol vykonávať len túto konkrétnu úlohu a nemohol vykonávať žiadnu inú úlohu. To robota robilo neprispôsobivým a obmedzovalo jeho užitočnosť. Na prekonanie obmedzení vopred naprogramovaných pokynov začali výskumníci skúmať nové metódy ovládania robotov. Jednou z takýchto metód je učenie posilňovaním. Učenie posilňovaním často vyžaduje veľké množstvo údajov, ktorých zber môže byť nákladný a časovo náročný. Preto je na zefektívnenie procesu učenia posilňovaním nevyhnutné mať nástroje na spracovanie a správu údajov. Algoritmy učenia posilňovaním sú často výpočtovo náročné a vyžadujú veľké množstvo výpočtových zdrojov. Preto je výhodné proces učenia rozdeliť do niekoľkých nezávislých inštancií, čím sa zvýši celkový výpočtový výkon. Algoritmy učenia posilňovaním sú veľmi závislé od hyperparametrov, ktoré môžu výrazne ovplyvniť ich výkonnosť. Výhodné je využívať automatizované nástroje pre vyladenie hyperparametrov. Učenie posilňovaním si vyžaduje dobré znalosti teórie pravdepodobnosti a optimalizačných techník. Súbor nástrojov na učenie posilňovaním s rôznymi nástrojmi strojového učenia by mal výskumníkom poskytnúť prístup k nástrojom na štatistickú analýzu a optimalizáciu.

1 Viacvrstvové neurónové siete

Viacvrstvové neurónové siete patria do rodiny neurónových sietí a skladajú sa z viacerých po sebe nasledujúcich vrstiev perceptrónov. Perceptrón je chápaný ako základný prvok, matematický model neurónu, ktorý je aktivovaný vstupným vektorom a na jeho základe predikuje jeden výstup. Tento prístup je inšpirovaný biologickými neurónovými sieťami, avšak dôraz sa nekladie na napodobnenie živého mozgu, ale na riešenie komplexných matematických modelov. Pre viacvrstvové neurónové siete existuje učiaci algoritmus *backpropagation*, ktorý ich učí mapovať vstupný vektor z tréningovej množiny na výstupný vektor, teda z matematického hľadiska ide o univerzálny aproximátor. Základný neurón (perceptrón) pozostáva zo vstupov, váh, prahu, aktivačnej funkcie a výstupu. Váhy kontrolujú veľkosť signálu prevedeného na výstup neurónu. Prah slúži na kontrolu otvárania výstupu neurónu podobne ako pri lineárnej regresii, kde predstavuje posun priamky. Pre aproximáciu nelineárnych problémov je súčasťou neurónov aktivačná funkcia, ktorá je zväčša iná ako lineárna funkcia. Viacero neurónov tvoriacich skupinu s rovnakým počtom hrán od vstupnej vrstvy predstavujú vrstvu. Prvá vrstva je vždy chápaná ako vstupná vrstva a sú na ňu priamo privedené vstupy z okolitého sveta. Posledná vrstva neurónov zase predstavuje výstupnú vrstvu a jej výstupy predstavujú odpoveď na riešený problém. Pri úlohách regresie sa často používa lineárna aktivačná funkcia v poslednej vrstve pre docielenie pracovného rozsahu (-∞, ∞). Všetky vrstvy obsiahnuté medzi vstupnou a výstupnou vrstvou sú označené ako skryté vrstvy. Tu sa práve používajú nelineárne aktivačné funkcie ako napríklad usmernená lineárna jednotka (ReLU) (Agarap, 2018) alebo hyperbolický tangens (TanH) (Brownlee, 2016).



Obrázok 1 Architektúra viacvrstvovej neurónovej siete pre klasifikáciu

1.1 Algoritmus backpropagation

Algoritmus spätného šírenia chyby je najrozšírenejším prístupom pri aktualizácii váh a prahu neurónových sietí. Vychádza z princípu spádu gradientu teda vypočíta sa gradient chybovej funkcie vzhľadom na aktualizované parametre modelu. Ide o generalizáciu delta pravidla pre jednoduchý neurón do podoby viacvrstvovej neurónovej siete. Delta pravidlo je pravidlo gradientného učenia na aktualizáciu váh vstupov umelých neurónov využívajúce rozdiel medzi očakávanou a súčasnou hodnotou. Prvý sa vypočíta gradient pre výstupnú vrstvu, ktorý sa ďalej šíri až k prvej vstupnej vrstve. Cieľom *backpropagation* je učiť skryté vrstvy modelu, kde nie je jednoduché určiť presnú chybu predikcie, preto vychádza z chyby predikcie nasledujúcich vrstiev. Keďže sa jedná o výpočtovo náročný problém, najčastejšie je realizovaný pomocou grafického procesoru (McGonagle, 2022).

Gradient g_w a g_b je určený parciálnymi deriváciami:

$$g_w = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} y'(z_i) x_j \tag{1.1}$$

$$g_b = \frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial b_i} = \frac{\partial E}{\partial x_j} y'(z_i) \tag{1.2}$$

$$z_i = (\sum_{j \in N} w_{ij} x_j) + b_i \tag{1.3}$$

- E – funkcia chyby siete,
- x_j – vstupný vektor neurónu,
- $y'(z_i)$ – 1. derivácia aktivačnej funkcie neurónu.

Výpočet parciálnej derivácie $\partial E/\partial x_j$ je zvlášť vyjadrený pre neuróny výstupnej vrstvy, skrytej vrstvy a vstupnej vrstvy:

$$\frac{\partial E}{\partial x_j} = \begin{cases} \left(y_i^{\text{očakávaný}} - y_i\right) & pre\ výstupnú\ vrstvu \\ \sum_k \frac{\partial E}{\partial x_k}\frac{\partial x_k}{\partial x_j} = \sum_k \frac{\partial E}{\partial x_k^{l+1}} y'\left(z_k^{l+1}\right) w_{kj}^{l+1} & pre\ skryté\ vrstvy\ a\ vstupnú\ vrstvu \end{cases} \quad (1.4)$$

Algoritmus 1 Algoritmus backpropagation (Guo, 2019)

procedure TRAIN

1. X ← Training Data Set of size mxn
2. y ← Labels for records in X
3. w ← The weights for respective layers
4. l ← The number of layers in the neural network, 1...L
5. $D_{ij}^{(l)}$ ← The error for all l,i,j
6. $t_{ij}^{(l)}$ ← 0. For all l,i,j
7. For i = 1 to m
8. a^{l} ← I feedforward($x^{(i)}$, w)
9. d^{l} ← a(L) - y(i)
10. $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_{j}^{(l)} * t_{i}^{l+1}$
11. if j ≠ 0 then
12. $D_{ij}^{(l)} \leftarrow 1/m * t_{ij}^{(l)} + \lambda w_{ij}^{(l)}$
13. else
14. $D_{ij}^{(l)} \leftarrow 1/m * t_{ij}^{(l)}$
15. where $dJ(w)/dw_{ij}^{(l)} = D_{ij}^{(l)}$

1.2 Inicializátor

Základom pre správne natrénovanie modelu je výber vhodného inicializátora počiatočných váh a prahov. Výber vhodnej metódy vedie aj k stabilnejšiemu priebehu učenia. Pre použitie vo vybraných úlohách s robotmi je aplikovaná metóda výberu náhodných hodnôt z rovnomernej distribúcie s rozsahom [-*limit*, *limit*] daným vzťahom, čo predstavuje škálu hodnôt. Škála je chápaná ako miera rozloženosti hodnôt v rozdelení pravdepodobnosti. Rozptyl hodnôt predstavujúcich inicializačné trénované parametre modelu určuje škála (Hoffman, 2020):

$$limit = \frac{\sqrt{3 * scale}}{fan\_in} \tag{1.5}$$

- *scale* – škála pre náhodnú premennú s hodnotou 1.0,
- *fan_in* – počet vstupov do vrstvy.

1.3 Optimalizátor Adam

Algoritmus adaptívny odhad momentu (Adam) využíva pri výpočte moment prvého a druhého rádu. Moment je jednoduchá technika, ktorá často zvyšuje rýchlosť aj presnosť tréningu, pri aktualizácii aktuálnych váh berie do úvahy predchádzajúce zmeny váh. Počas tréningu sa zapamätáva exponenciálne klesajúci priemer minulých štvorcových gradientov v_t ako aj exponenciálne klesajúci priemer minulých gradientov m_t. Výhodou tohto algoritmu je rýchla konvergencia neurónovej siete, kedy postačuje menší počet krokov tréningu pre dosiahnutie globálneho minima chybovej funkcie (Kingma, Ba, 2015).

$$m_t \leftarrow \beta_1 * m_{t-1} + (1 - \beta_1) * \frac{\partial E}{\partial w_{ij}^t} \tag{1.6}$$

$$v_t \leftarrow \beta_2 * v_{t-1} + (1 - \beta_2) * (\frac{\partial E}{\partial w_{ij}^t})^2 \tag{1.7}$$

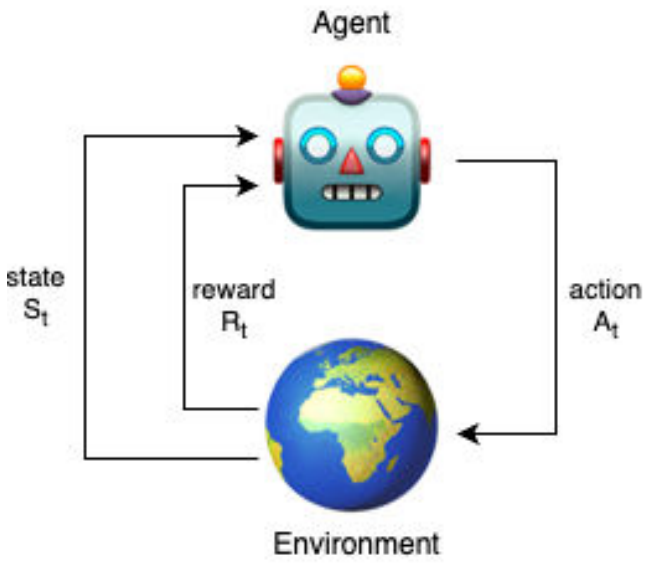
$$w_{ij}^t = w_{ij}^{t-1} - \eta \frac{\frac{m_t}{(1-\beta_1^t)}}{\sqrt{\frac{v_t}{(1-\beta_2^t)}} + \varepsilon} \tag{1.8}$$

- β_1 – parameter miery klesania momentu 1. rádu, použitá hodnota je 0.9,
- β_2 – parameter miery klesania momentu 2. rádu, použitá hodnota je 0.999,
- ϵ – malá konštanta pre numerickú stabilizáciu, použitá hodnota je 10^{-7},
- η – parameter rýchlosti učenia.

Rovnakým postupom sú odvodené aj rovnice pre parametre prahov neurónov (Kingma, Ba, 2015). Pre limitovanie maximálnej úrovne gradientov je v optimalizátore použitý aj globálny *clipnorm*, čo predstavuje spôsob obmedzenia maximálnej úrovne jednotlivých hodnôt gradientov na základe ich globálnej L2 normalizácie (Pascanu, 2012).

2 Učenie posilňovaním

Učenie posilňovaním (RL) je typ strojového učenia, ktoré umožňuje robotom učiť sa na základe vlastných skúseností. Učenie posilňovaním funguje tak, že robotovi poskytuje odmenu alebo trest na základe jeho činnosti. Napríklad, ak má robot za úlohu prejsť bludiskom, dostane odmenu za úspešnú navigáciu bludiskom a trest za náraz do steny alebo míňanie energie pri prehľadávaní, čo ho núti nájsť vždy najkratšiu cestu v bludisku. Postupom času sa robot naučí, ktoré činnosti vedú k odmene a ktoré k trestu a podľa toho upraví svoje správanie. Učenie posilňovaním umožňuje robotom učiť sa z vlastných skúseností a prispôsobovať sa meniacim sa podmienkam, čím sa stávajú flexibilnejšími a prispôsobivejšími. Tento prístup viedol k vývoju robotov, ktoré dokážu vykonávať širokú škálu úloh, a otvoril nové možnosti využitia robotov v rôznych oblastiach. Úspešne sa okrem robotiky uplatňuje v rôznych oblastiach, ako sú hry či spracovanie prirodzeného jazyka. Učenie posilňovaním je náročná oblasť výskumu, ktorá si vyžaduje širokú škálu nástrojov a techník strojového učenia. Cieľom je maximalizovať kumulatívnu odmenu agenta v prostredí (Bhatt, 2018).



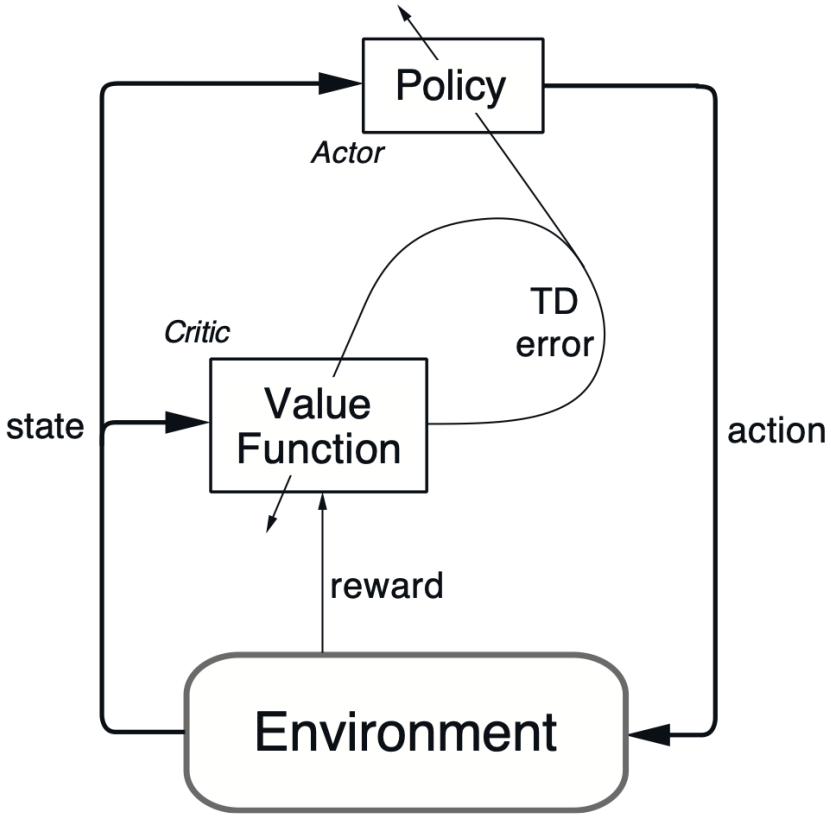
Obrázok 2 Učenie posilňovaním

Prostredie je chápané ako svet, v ktorom agent vykonáva operácie. Stav je chápaný ako aktuálna situácia, v ktorej sa agent nachádza a je vyjadrená meraním zo senzorov robota. Odmena je spätná väzba z prostredia za vykonanú akciu. Politika je metóda akou agent mapuje stav na vykonanú akciu. Problémom učenia posilňovaním je, akým spôsobom agent

bude prehľadávať prostredie. Tento problém sa označuje ako aj prieskum vs. využívanie. Vyváženie medzi prieskumom a využívaním naučených znalostí z prostredia vedie k najoptimálnejšej politike agenta. Teoretické základy pre učenie posilňovaním vychádzajú z Markovovho rozhodovacieho procesu. Markovov rozhodovací proces (MDP) pozostáva z konečnej množiny stavov, množiny možných akcií v danom stave, odmeny za vykonanú akciu a model prechodu daného pravdepodobnosťou $P(s_{t+1}, s_t \mid a_t)$. Nasledujúci stav s_{t+1} vyjadruje stav, ktorý agent zmeria po vykonaní akcie a_t v stave s_t. Avšak u reálnych prostredí nie je znalosť tejto pravdepodobnosti a model sa musí naučiť sám dynamiku prostredia, *model-free* metóda. Model-free nezahŕňa znalosť dynamiky procesu, ale snaží sa ju aproximovať učením sa v prostredí. Q-hodnota je najčastejšie používanou veličinou pre *model-free* odhad kvality vykonanej akcie. Q-hodnota predstavuje súčet odmien za istý časový horizont (Bhatt, 2018).

3 Metóda Actor-Critic

Actor-critic metódy patria do skupiny temporal-difference (TD) metód vychádzajúcich z myšlienky kombinácie Monte Carlo prístupu a dynamického programovania. TD metódy sú učené priamo zo zmeraných skúseností senzormi agenta bez poznania modelu dynamiky prostredia, kde agent vykonáva akcie. Model pre politiku je známy ako *actor*, pretože predikuje akcie agenta. TD metódy využívajú oddelenú štruktúru modelov, ktoré reprezentujú zvlášť politiku π (akcie generované agentom), podľa ktorej sa agent riadi v prostredí a ohodnocovaciu funkciu Q, ktorá vyjadruje kvalitu danej akcie. Ohodnocovacia funkcia vyjadrená modelom je známa ako *critic*, pretože kritizuje akcie vykonané modelom *actor*. Kritika je učená v zmysle TD chyby čo predstavuje skalárny signál použitý pre tréning oboch modelov, čo je naznačené na obrázku 3. Ak je TD chyba pozitívna, vyplýva z toho, že zvolená akcia a_t by sa v budúcnosti mala posilniť, na druhú stranu ak je negatívna, akcia by sa mala potlačiť (Sutton, Barto, 2015).



Obrázok 3 Architektúra *actor-critic* (Sutton, Barto, 2015)

Prvky architektúry *actor-critic*:

- *actor* (politika) – model predikujúci akcie a_t na základe stavu s_t,
- *critic* (ohodnocovacia funkcia) – model predikujúci kvalitu vykonanej akcie,
- prostredie – priestor, v ktorom sa agent pohybuje a interaguje s okolím:
  - stav – reprezentuje pozorovanie agenta pomocou svojich senzorov,
  - akcia – aktivita vykonaná agentom v prostredí,
  - odmena – skalárna spätná väzba na akciu a_t vykonanú agentom v stave s_t,
- TD chyba – pravidlo, podľa ktorého sa učí kvalita vykonaných akcií.

TD chyba je použitá vo vzťahu pre aktualizáciu *Q* hodnoty:

$$Q_{(s_t,a_t)} = Q_{(s_t,a_t)} + \alpha\left[r_{t+1} + \gamma Q_{(s_{t+1},a_{t+1})} - Q_{(s_t,a_t)}\right] \tag{3.1}$$

- $Q(s_t, a_t)$ – Q-hodnota v stave s_t pre akciu a_t (kvalita vykonanej akcie),
- α – pozitívna hodnota určujúca veľkosť kroku aktualizácie,
- r_{t+1} – hodnotiaca funkcia (odmena/trest) za vykonanú akciu a_t,
- γ – faktor utlmujúci podiel nasledovnej kvality za akciu pri aktualizácii súčasnej Q-hodnoty,
- $Q(s_{t+1}, a_{t+1})$ – vyjadruje kvalitu akcie a_{t+1} v budúcom stave s_{t+1}, ktorý nastal po akcii a_t (Sutton, Barto, 2015).

3.1 Algoritmus Soft Actor-Critic

Algoritmus *Soft Actor-Critic* (SAC) vychádza z princípu učenia sa z interakcií získaných od počiatku učenia agenta v hernom prostredí s prihliadnutím na maximalizáciu entropie. Pri maximalizácii entropie sa politika agenta upravuje tak, aby sa maximalizovala entropia rozdelenia distribúcie akcií agenta. Do účelovej funkcie agenta sa pridá člen entropie, ktorý agenta povzbudzuje k tomu, aby vykonával menej sa opakujúce akcie, čím narastá neurčitosť teda entropia. Člen entropie je definovaný ako záporná hodnota súčtu pravdepodobností jednotlivých akcií vynásobená logaritmom pravdepodobnosti. Tento člen penalizuje akcie s vysokou pravdepodobnosťou výberu a podporuje akcie s nižšou pravdepodobnosťou výberu. Týmto prístupom *actor* vykonáva čo možno najnáhodnejšie akcie. *Critic* sa popritom učí predikovať čo možno najpresnejšie Q-hodnotu udávajúcu kvalitu akcie v danom stave. Na začiatku učenia nie je známy model prostredia a SAC je teda náročné na množstvo príkladov interakcií, ktoré potrebuje zbierať rádovo v miliónoch pri vysoko rozmerných

stavových/akčných priestoroch. *Actor* sa počas učenia snaží pomocou spádu gradientu maximalizovať Q-hodnotu predikovanú častou *Critic*. Výhodou tohto prístupu je aj nízka citlivosť na nastavenie hyperparametrov v zmysle použitia agenta v rôznych herných prostrediach (Haarnoja, 2018). Podobne ako pri algoritme *twin delayed deep deterministic policy gradient* (TD3) sú použité viaceré *critic* modely, z ktorých je brané minimum Q-hodnoty čo zabraňuje nadhodnocovaniu Q-hodnoty (Fujimoto, 2018).

Chybová funkcia minimalizovaná počas učenia *actor* modelu:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D, \epsilon_t \sim \mathcal{N}}\left[\alpha \log \pi_\phi(f_\phi(\epsilon_t, s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t, s_t))\right] \tag{3.2}$$

- $J_\pi(\phi)$ – chybová funkcia *actor* modelu,
- α – váha pre podiel entropie na chybe *actor* modelu,
- $log[\pi_\phi(f_\phi(\varepsilon_t, s_t)|s_t)]$ – logaritmus pravdepodobnosti akcie vygenerovanej *actor* modelom v stave s_t (vyjadruje entropiu),
- $Q_\theta(s_t, f_\phi(\varepsilon_t, s_t))$ - Q-hodnota v stave s_t pre akciu vygenerovanú *actor* modelom (kvalita vykonanej akcie).

Chybová funkcia minimalizovaná počas učenia *critic* modelu:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim D}\left[\frac{1}{2}(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma(Q_\theta(s_{t+1}, a_{t+1}) - \alpha \log \pi_\phi(a_t|s_t))))^2\right] \tag{3.3}$$

- $J_Q(\theta)$ – chybová funkcia *critic* modelu,
- $Q_\theta(s_t, a_t)$ – Q-hodnota v stave s_t pre akciu a_t (kvalita vykonanej akcie),
- $r(s_t, a_t)$ – odmena za akciu a_t vykonanú v stave s_t,
- γ – parameter regulácie podielu nasledujúcej kvality akcie na súčasnej kvalite akcie,
- $Q_\theta(s_{t+1}, a_{t+1})$ – Q-hodnota v stave s_{t+1} pre akciu a_{t+1} (kvalita budúcej akcie),
- α – váha pre podiel entropie na chybe *actor* modelu,
- $log[\pi_\phi(a_t|s_t)]$ – logaritmus pravdepodobnosti akcie a_t v stave s_t (vyjadruje entropiu).

3.2 Zovšeobecnené prehľadávanie závislé od stavu

Základný prístup pri generovaní akcií z Gaussovej distribúcie, ktorej parametre sú predikované pomocou neurónovej siete spôsobuje roztrasené správanie sa robota či jeho vibrácie. Jeden z prvých robotov využívajúcich umelú inteligenciu sa veľmi triasol, čo sa pretavilo aj do súčasnosti, kedy roztrasené akcie vyjadrujú akcie doplnené o šum, ktorý napomáha robotovi prehľadávať prostredie. Toto správanie môže v praxi viesť ku skoršiemu opotrebovaniu mechaniky robota. Algoritmus *generalized state-dependent exploration* (gSDE) rieši tento problém pridaním Gaussovho šumu k deterministickej akcii, ktorý je generovaný na základe stavového priestoru. Výsledná akcia je generovaná podľa vzťahu (Raffin, 2021a):

$$a_t = \mu(s_t, \theta_\mu) + \varepsilon(z_t, \theta_\varepsilon) \qquad \theta_\varepsilon \sim \mathcal{N}(0, \sigma^2) \qquad (3.4)$$

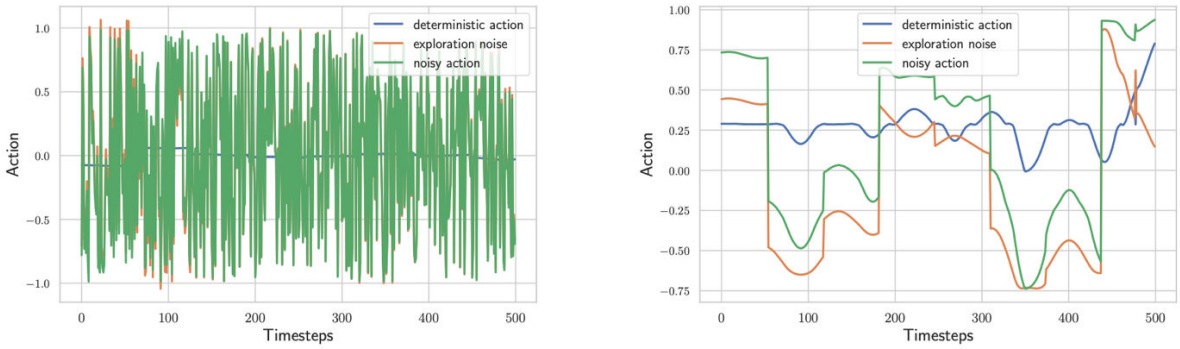
- $\mu(s_t, \theta_\mu)$ – deterministická akcia generovaná *actor* modelom s parametrami θ_μ,
- $\varepsilon(z_t, \theta_\varepsilon)$ – parametre generované Gaussovou distribúciou na základe latentného priestoru z_t.

Parameter Gaussovej distribúcie σ je adaptovaný počas procesu učenia sa agenta v zmysle:

$$\frac{\partial log\ \pi(a|s)}{\partial \sigma_{ij}} = \frac{(a_j - \mu_j)^2 - \hat{\sigma}_j^2}{\hat{\sigma}_j^3} \frac{z_i^2 \sigma_{ij}}{\hat{\sigma}_j} \qquad (3.5)$$

$$\hat{\sigma}_j = \sqrt{\sum\nolimits_i (\sigma_{ij} z_i)^2} \qquad (3.6)$$

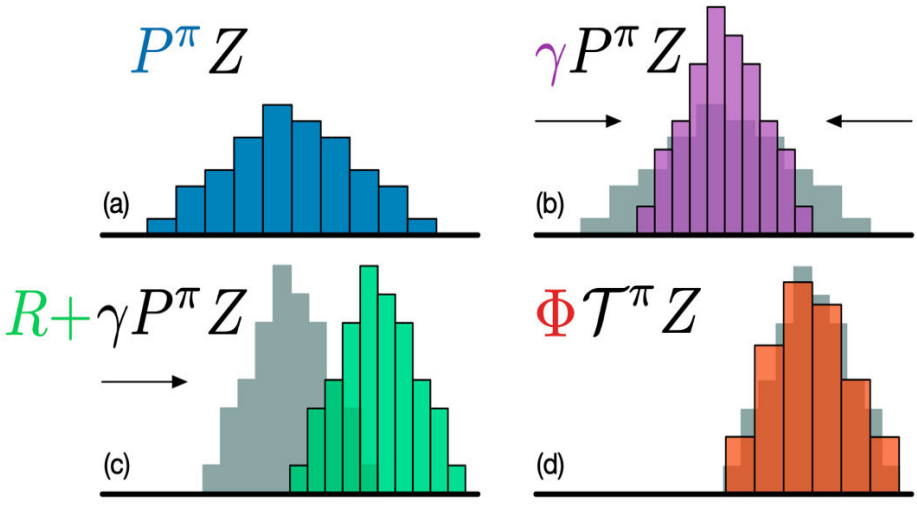
- z_i – latentný priestor v *actor* modeli,
- σ_{ij} – trénované parametre smerodajnej odchýlky obsiahnuté medzi parametrami *actor* modelu,
- $\hat{\sigma}_j$ – parametre smerodajnej odchýlky aktuálnej distribúcie závislé od stavu s_t.



Obrázok 4 Porovnanie akcií generovaných algoritmom gSDE (Raffin, 2021a)

3.3 Algoritmus C51

Typicky *critic* model predikuje jednu priemernú hodnotu veličiny, ale ak má model opísať presnejšie reálny svet, je potrebné predikovať distribúciu tejto veličiny. Prvým z prístupov ako predikovať distribúciu Q-hodnoty, bolo použitie kategorickej distribúcie atómov generovaných pomocou aktivačnej funkcie *softmax* s označením C51. Atóm predstavuje jednu z hodnôt diskrétnej distribúcie. V tomto prípade atómy vyjadrujú príklady Q-hodnôt z distribúcie, pričom predikovaná je pravdepodobnosť tejto hodnoty, čo zabezpečuje *critic* model. Príklady Q-hodnôt predstavovali rovnomerne rozdelené hodnoty z predom definovaného intervalu, podľa definovaného počtu atómov. Označenie C51 vyjadruje počet atómov použitých pre opis Q-hodnoty, ktorých bolo 51 (Bellemare, 2017).

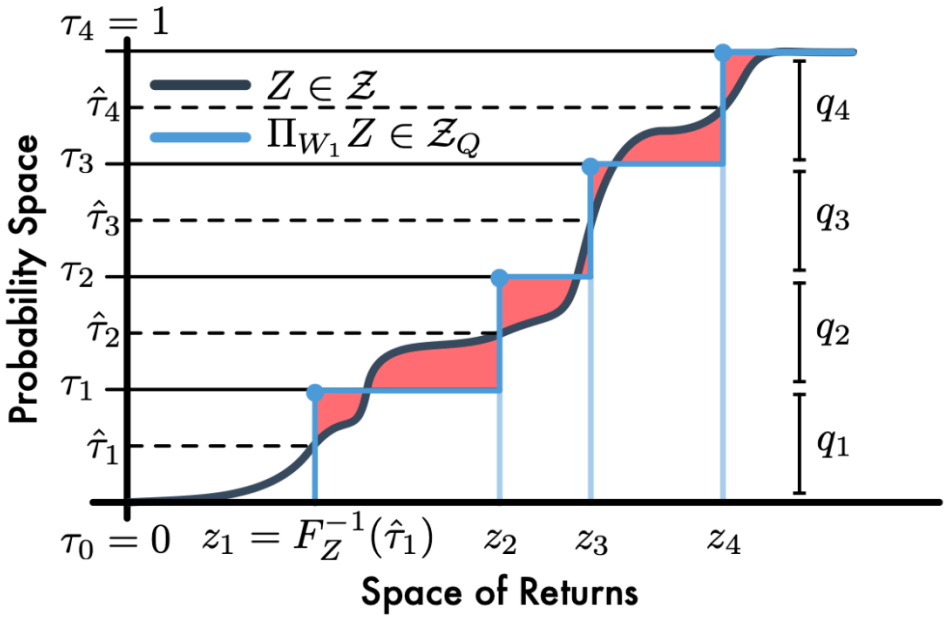


Obrázok 5 Distribúcia Q-hodnoty (Bellemare, 2017)

Na obrázku 5 je možné vidieť ukážku distribúcie Q-hodnoty (a), zmenšenie distribúcie smerom k 0 vplyvom parametra γ (b), funkcia odmeny spôsobila posun distribúcie na osi x (c), projekcia distribúcie pomocou 51 atómov (d).

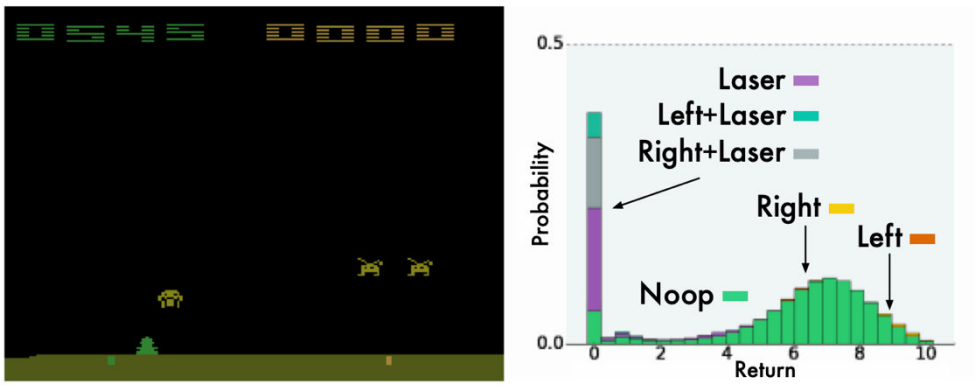
3.4 Algoritmus QR-DQN

Pri popísaní Q-hodnoty formou distribúcie je jednou z ďalších možností využitie kvantilov. Pri učení hlbokých neurónových sietí (pojem "hlboká" sa vzťahuje na skutočnosť, že sieť má viac ako jednu skrytú vrstvu) je možné využiť algoritmus kvantilovej regresie (viď ďalej v sekcii 3.5.1) a učiť model predikovať opis distribúcie na rozdiel od jej priemernej hodnoty. Pretože nie je možné predom presne stanoviť interval Q-hodnôt, presnejšou metódou sa ukázala predikcia kvantilov na rozdiel od pevne stanovených atómov. Pre výpočet chyby medzi predikovanou a očakávanou distribúciou je možné použiť 1-Wasserstein metriku chyby. Táto metrika je založená na výpočte veľkosti plochy tvoriacej rozdiel medzi dvomi distribúciami definovanými pomocou kvantilov (Stéphanovitch, 2022). Ako je možné vidieť na obrázku 6 pri algoritme *quantile regression deep Q network* (QR-DQN) dochádza k aproximácii distribúcie pomocou diskrétneho vyjadrenia stanoveným počtom kvantilov. Preto je aj aplikovaná chybová funkcia aproximáciou veľkosti tejto plochy počas procesu jej minimalizácie. QR-DQN model získal lepšie výsledky ako C51 model v typickom prostredí pre porovnávanie algoritmov Atari 2600 (Dabney, 2017).



Obrázok 6 Ukážka 1-Wasserstein chyby aplikovanej na distribúciu (Dabney, 2017)

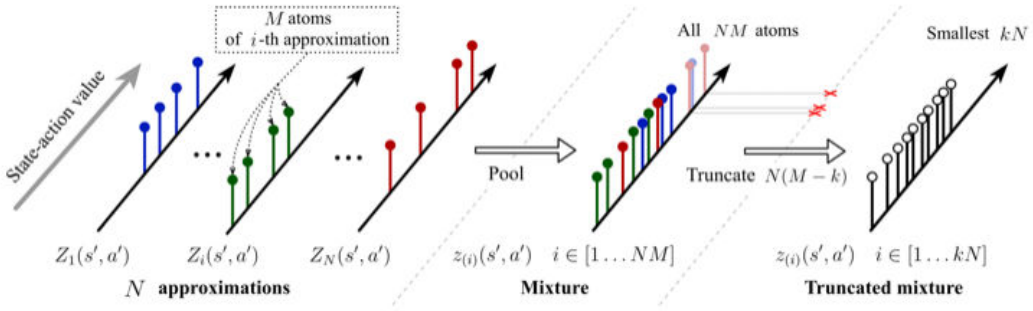
Na obrázku 7 je rôznymi farbami označená jedna z možných diskrétnych akcií agenta hrajúceho jednu z hier v prostredí Atari 2600, *SpaceInvanders*. Graf distribúcie je tvorený kombináciou viacerých histogramov umiestnených na sebe. Každý z histogramov vyjadruje distribúciu jednej z akcií. Možnými akciami sú streľba, vľavo + streľba, vpravo + streľba, vľavo, vpravo a nerobí nič.



Obrázok 7 Ukážka distribúcie Q-hodnoty po natrénovaní QR-DQN (Dabney, 2017)

3.5 Obmedzenie nadhodnocovania kvality akcií s kvantilovým modelom

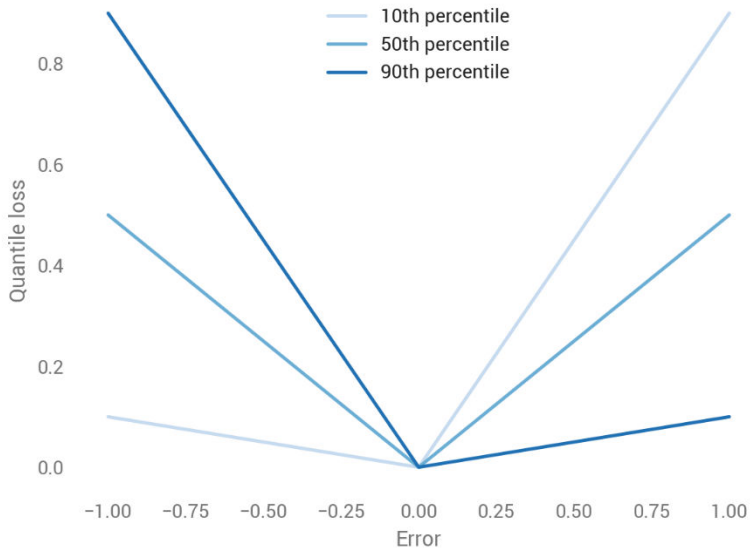
Soft Actor-Critic vychádza z princípu ohodnocovania pomocou Q-hodnoty, ktorá môže byť podobne ako pri QR-DQN vyjadrená pomocou kvantilov. Týmto spôsobom dôjde k zlepšeniu presnosti predikovaných kvalít, čím sa priamo úmerne zlepší aj učenie *actor* modelu pomocou gradientov. Q-hodnota všeobecne je pri aproximácii pomocou hlbokej neurónovej siete nadhodnocovaná. Túto nevýhodu je potrebné eliminovať pomocou odstránenia niekoľkých horných kvantilov, čím sa umelo zníži priemerná Q-hodnota vyplývajúca z distribúcie. Taktiež je výhodnejšie používať viacero *critic* modelov v kombinácii, pričom je možné hodnotu ich kvantilov triediacim algoritmom usporiadať od najnižšej Q-hodnoty po najvyššiu Q-hodnotu. Takýmto spôsobom je distribúcia Q-hodnoty vyjadrená kvantilmi rekonštruovateľná. Tým je zabezpečená ešte lepšia presnosť generovaných akcií po učení sa *actor* modelu (Kuznetsov, 2020).



Obrázok 8 Princíp výberu kvantilov k odstráneniu (Kuznetsov, 2020)

3.5.1 Kvantilová regresia

Kvantilová regresia využíva opis veličiny na základe kvantilov, ktoré definujú rozsah možných hodnôt náhodnej veličiny. Výhodné je to napríklad pri predikcii cien nehnuteľností, ktoré majú určitý možný interval ohodnotenia ceny. Pri kvantilovej regresii nie je možné použiť bežné regresné chyby ako priemerná štvorcová chyba ale je vyvinutá špeciálna chybová funkcia. Hodnota chyby závisí od kvantilu, ktorý je penalizovaný, teda negatívne chyby sú penalizované viacej, pokiaľ ide o vyššie kvantily a pozitívne chyby sú viac penalizované pokiaľ ide o nižšie kvantily (Ghenis, 2018).



Obrázok 9 Ukážka kvantilovej chyby pre rôzne kvantily a chyby (Ghenis, 2018)

Ako je možné vidieť z obrázku 9, 50ty kvantil (medián) je symetrický okolo nuly a váha pre nadhodnotené alebo podhodnotené predikcie je rovnaká. 10ty kvantil prikladá nižšiu váhu k negatívnym chybám a vyššiu váhu k pozitívnym chybám, nakoľko vyjadruje spodnú

hranicu intervalu predikovanej veličiny, a teda jeho podhodnotenie nie je také kritické ako jeho nadhodnotenie. Pre 90ty kvantil to platí inverzne (Ghenis, 2018).

4 RL-Toolkit

Učenie agentov vykonávať správne zadané úlohy v rôznych prostrediach si vyžaduje komplexnú tréningovú sadu nástrojov. Ako prvé je výhodné mať k dispozícii viacero simulačných prostredí, kde je možné trénovať agentov. Každé z prostredí od iného autora môže mať rozdielnu definíciu API, čo komplikuje prenositeľnosť procesu učenia sa agenta. Ďalším problémom je ukladanie a distribúcia získaných interakcií (skúseností) z prostredia. Pre umožnenie distribuovania učenia medzi viacerých agentov zbierajúcich skúsenosti v prostredí a viacerých *learner* učiacich sa z uložených skúseností je výhodné použiť inštanciu databázového servera. Databázový server by mal za úlohu poskytovať komplexné API, ktoré umožní vytvoriť úložisko pre získané skúsenosti, umožní zápis do tohto úložiska a taktiež čítanie z úložiska, ktoré sa jednoducho aplikuje do existujúcich riešení *Actor-Critic* algoritmov. Posledným faktorom je monitorovanie procesu učenia a preto musí byť užívateľom dostupný monitoring rôznych veličín. Tieto veličiny sú závislé od typu použitého algoritmu pre učenie sa agenta. Výhodné je mať uložené a dostupné grafické vyjadrenie meraní kdekoľvek na svete, čo je docielené využitím cloudových služieb.

V porovnaní s konkurenčnými riešeniami MATLAB Reinforcement Learning Toolbox ponúka na výber viacero rôznych typov RL algoritmov. Avšak neposkytuje databázový server ako RL-Toolkit. Jeho ďalšou výhodou je kombinovanie so Simulink (MATLAB, 2023).

Principiálne sa RL-Toolkit podobá na nástroje s otvoreným zdrojom ako ClearRL. ClearRL podobne poskytuje monitorovacie nástroje Weights & Biases, okrem čoho aj monitorovací nástroj Tensorboard. ClearRL naviac obsahuje až 7 rôznych RL algoritmov. Podobne ako RL-Toolkit má kontajnerovú integráciu v prostredí Docker. ClearRL využíva ako knižnicu pre neurónové siete PyTorch (Huang, 2022).

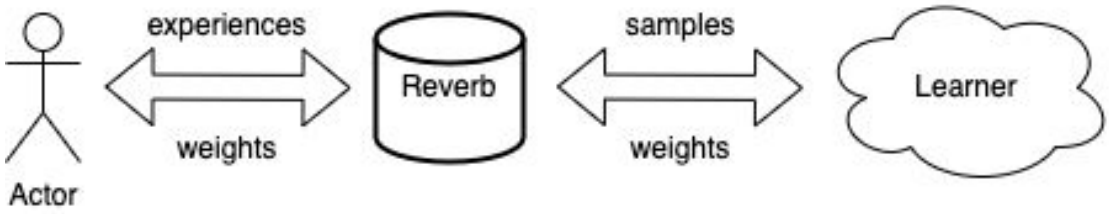
Ďalším známym nástrojom je Acme od DeepMind. Acme ponúka rovnako možnosť využitia databázového servera DeepMind Reveb pre ukladanie a distribuovanie interakcií. Avšak neposkytuje monitorovacie nástroje Weights & Biases a nástroje pre automatizované ladenie hyperparametrov. Taktiež Acme nie je určené pre kombinovanie rôznych prístupov ale využitie existujúcich algoritmov a ich porovnanie pri riešení zadanej úlohy. Acme sa nezaoberá ani priamou aplikáciou na reálneho robota. Acme využíva ako knižnicu pre neurónové siete Jax a TensorFlow (Hoffman, 2020).

Nástroj Stable-baselines3 využíva podobne ako RL-Toolkit algoritmus gSDE a TQC, avšak neposkytuje databázový server. Podobne ako CleanRL využíva knižnicu PyTorch. Stable-baselines3 má aj oficiálnu integráciu monitorovacieho nástroja zo strany Weights & Biases. Poskytuje niekoľko typov rôznych RL algoritmov. Stable-baselines3 bol použitý aj pri tréningu reálnych robotov (Raffin, 2021b). V tabuľke 1 je uvedené porovnanie viacerých nástrojov pre učenie agentov pomocou RL algoritmov.

Tabuľka 1 Porovnanie konkurenčných nástrojov

Názov nástroja	**CleanRL**	**Acme**	**Stable-baselines3**	**RL-Toolkit**	**MATLAB**
Databázový server	❌	✅	❌	✅	❌
Monitorovací a ladiaci nástroj Weights & Biases	✅	❌	✅	✅	❌
Podpora algoritmu gSDE	❌	❌	✅	✅	❌
Podpora algoritmu TQC	❌	❌	✅	✅	❌
Podpora služby Docker	✅	❌	✅	✅	❌
Viaceré typy RL algoritmov	✅	✅	✅	❌	✅

RL-Toolkit poskytuje všetky tieto vyššie uvedené druhy nástrojov. Umožňuje využívať pri učení agenta prostredia DeepMind Control Suite, PyBullet, Gymnasium a v prípade doinštalovania prostredia z tretej strany pre Gymnasium bude možné využívať aj toto prostredie. Ako databázový server používa DeepMind Reverb, ktorý je špecializovanou databázou pre výskum v oblasti učenia s posilňovaním. Pre monitorovanie procesu učenia agenta aplikuje cloudovú službu Weights & Biases (WanDB, 2023a).



Obrázok 10 Architektúra RL-Toolkit

Súpravy nástrojov s otvoreným zdrojovým kódom a súpravy nástrojov veľkých spoločností sa líšia v niekoľkých ohľadoch. Jedným z najvýznamnejších rozdielov je prítomnosť alebo neprítomnosť grafického používateľského rozhrania (GUI). Zatiaľ čo veľké spoločnosti zvyčajne ponúkajú súpravy nástrojov s užívateľsky prívetivým grafickým rozhraním, súpravy nástrojov s otvoreným zdrojovým kódom často grafické rozhranie nemajú a nechávajú viac práce na používateľovi. Pre projekty s otvoreným zdrojovým kódom to však nemusí byť nevyhnutne zlé. Súpravy nástrojov s otvoreným zdrojovým kódom zvyčajne vyvíja komunita dobrovoľníkov, ktorých spája spoločný záujem o technológiu alebo oblasť použitia. Títo dobrovoľníci prispievajú svojím časom a odbornými znalosťami k vývoju a údržbe súboru nástrojov, často bez akejkoľvek finančnej odmeny. Výsledkom je, že súpravy nástrojov s otvoreným zdrojovým kódom majú tendenciu uprednostňovať funkčnosť pred používateľskou prívetivosťou. To znamená, že im môže chýbať grafické používateľské rozhranie a na efektívne používanie si vyžadujú viac technických znalostí. Na druhej strane, veľké spoločnosti majú špecializované tímy vývojárov, ktorí sú platení za vytváranie a údržbu súprav nástrojov pre svojich zákazníkov. Tieto sady nástrojov sú často navrhnuté tak, aby boli čo najprívetivejšie pre používateľov, s intuitívnym grafickým rozhraním a rozsiahlou dokumentáciou. Tento prístup má pre veľké spoločnosti zmysel, pretože sa často zameriavajú na netechnických používateľov, ktorí musia byť schopní používať sadu nástrojov bez rozsiahleho školenia. Absencia grafického používateľského rozhrania v súboroch nástrojov s otvoreným zdrojovým kódom však môže byť dobrá z niekoľkých dôvodov. Po prvé, umožňuje komunite sústrediť sa na vývoj základných funkcií súpravy nástrojov namiesto vynakladania zdrojov na vývoj grafického rozhrania. Po druhé, podporuje používateľov, aby sa stali technicky zdatnejšími, čo môže viesť k hlbšiemu pochopeniu základnej technológie a inovatívnejšiemu použitiu súpravy nástrojov. Nakoniec umožňuje väčšiu flexibilitu a prispôsobenie, keďže používatelia môžu sadu nástrojov prispôsobiť svojim špecifickým potrebám bez toho, aby boli obmedzovaní vopred definovaným grafickým rozhraním. Hoci absencia grafického používateľského rozhrania v

tejto práci navrhnutom súbore nástrojov s otvoreným zdrojovým kódom môže spôsobiť, že ich používanie je pre netechnických používateľov náročnejšie, ide o zámernú voľbu, ktorá uprednostňuje funkčnosť, flexibilitu a spoluprácu v rámci komunity pred používateľskou prívetivosťou.

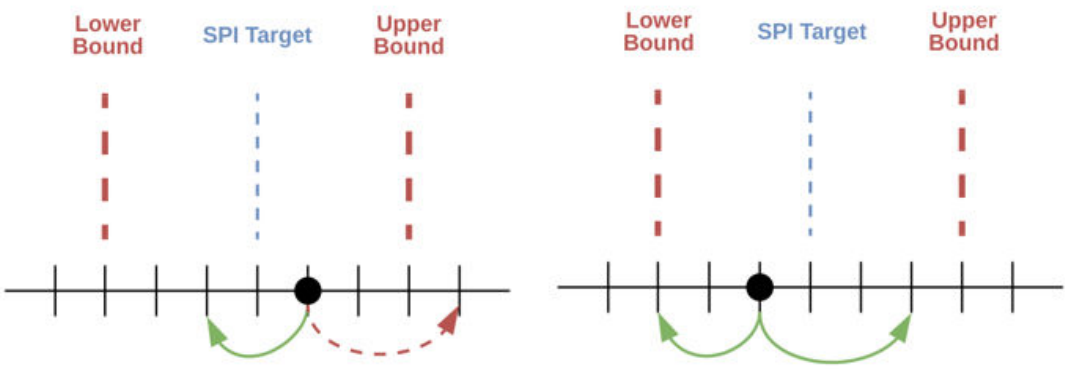
4.1 Databázový server

Ukladanie interakcií (skúseností) v typickom riešení s *replay buffer* (Fedus, 2020) je zabezpečené pomocou súboru matíc uložených v operačnej pamäti počítača, kde prebieha proces učenia. Pri veľkom počte agentov zbierajúcich interakcie súčasne a využití niekoľkých počítačov pre proces učenia je výhodné mať oddelenú inštanciu databázového servera určenú pre ukladanie a distribúciu. Takáto databáza by mohla popri ukladaní ešte regulovať ich vkladanie a vzorkovanie.

Toto riešenie ponúka databáza DeepMind Reverb. S doplnením o inštanciu databázového servera je možné celý proces učenia algoritmom Soft *Actor-Critic* rozdeliť na 3 samostatné role. Každá z rolí by tvorila samostatnú inštanciu, pričom by išlo o tých čo zbierajú interakcie (skúsenosti) v prostredí – *actors*, tých čo sa na základe týchto interakcií učia – *learners* a tých čo ukladajú a distribuujú interakcie – *servers*. Tento prístup umožňuje paralelné spustenie na viacerých inštanciách, pričom škálovanie je jednoduché.

Pre reguláciu pomeru vkladaní a samplovania je určený hyperparameter – pomer *samples per insert* (SPI). Reverb poskytuje vytvorenie úložiska na základe algoritmu fronty, zásobníka alebo haldy. Uloženým interakciám môžu byť pridelené priority, podľa ktorých budú vyberané do procesu učenia (Cassirer, 2021). V tejto práci sa osvedčil pomer SPI nastavený na 32, čo pri veľkosti tréningového balíčku 256 príkladov zodpovedalo približne osem násobku počtu nazbieraných interakcií k počtu krokov učenia sa agenta.

$$SPI = \frac{num\_sampled\_items}{num\_inserted\_items} \tag{4.1}$$



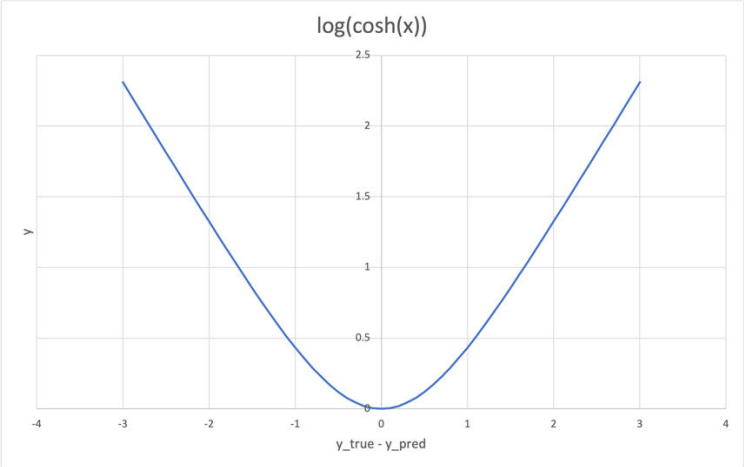
Obrázok 11 Ukážka princípu SPI pomeru (Cassirer, 2021)

Na obrázku 11 sú zobrazené dve modelové situácie, ktoré môžu nastať počas regulácie vkladania a vzorkovania z úložiska na základe SPI pomeru 3/2 s toleranciou ± 3. Možnosť vyobrazená na ľavej strane obrázka 11 je vkladanie príkladov blokované lebo prevyšuje hornú hranicu tolerancie. Možnosť vyobrazená na pravej strane obrázka 11 je blokované vkladanie príkladov lebo sú v tolerancii (Cassirer, 2021). V tejto práci je tolerancia počítaná ako 10 percent z SPI pomeru a minimálna hranica počtu príkladov uložených v databáze pred začiatkom procesu učenia (samplovania) je 10 000.

4.2 Zmeny v použitej chybovej funkcii

Pri kvantilovej regresii u QR-DQN modelu bola použitá *Huber loss*. Táto chybová funkcia je menej náchylná k ovplyvneniu presnosti vyjadrenej chyby spôsobenej odľahlými hodnotami (Gokcesu, 2021). Avšak v tejto práci je použitá LogCosh chybová funkcia, ktorá na rozdiel od *Huber loss* má hladkú prvú aj druhú deriváciu. LogCosh sa osvedčila kvôli svojím vlastnostiam pri kvantilovej regresii, nakoľko sa radí do kategórie robustných odhadov. Jej predikcia sa zameriava na riešenia v blízkosti mediánu a nie priemeru. Svojou vlastnosťou sa pri hodnotách vzdialených od x = 0 správa podobne ako L1 (absolútna hodnota). Ak x je blízko 0 potom sa správa podobne ako L2 (kvadratická hodnota). Nakoľko priame použitie funkcie LogCosh by mohlo spôsobiť numerickú nestabilitu, rôzne prostredia implementujú jej numericky stabilnú aproximáciu (Saleh, 2022). Rovnica 3.2 popisuje aproximáciu LogCosh funkcie v prostredí TensorFlow (Tensorflow, 2023). Na obrázku 12 je ukážka funkcie LogCosh, pričom na osi x je rozdiel medzi predikovanou a očakávanou hodnotou a na osi y je chyba siete.

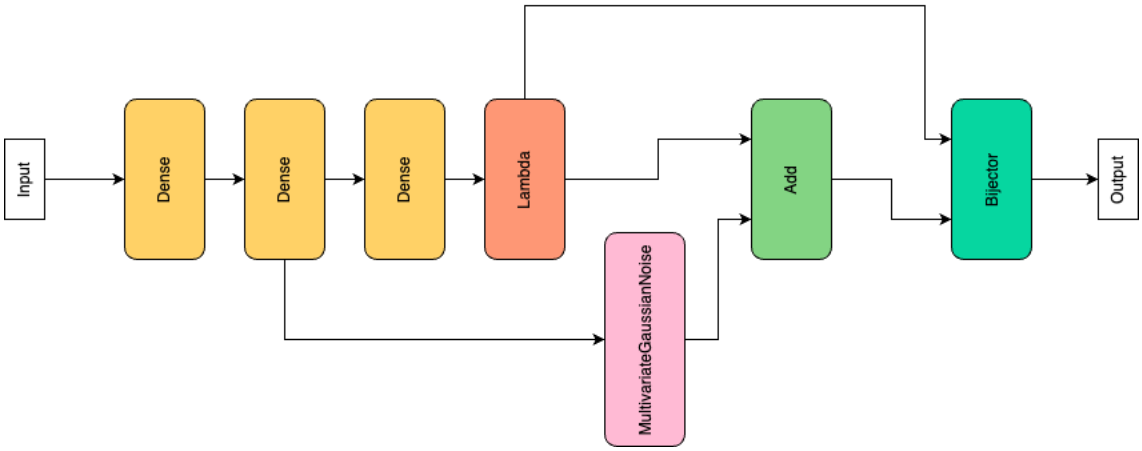
$$\log(\cosh(x)) \approx x + softplus(-2.0 * x) - log(2.0) \tag{4.2}$$



Obrázok 12 Ukážka chybovej funkcie LogCosh

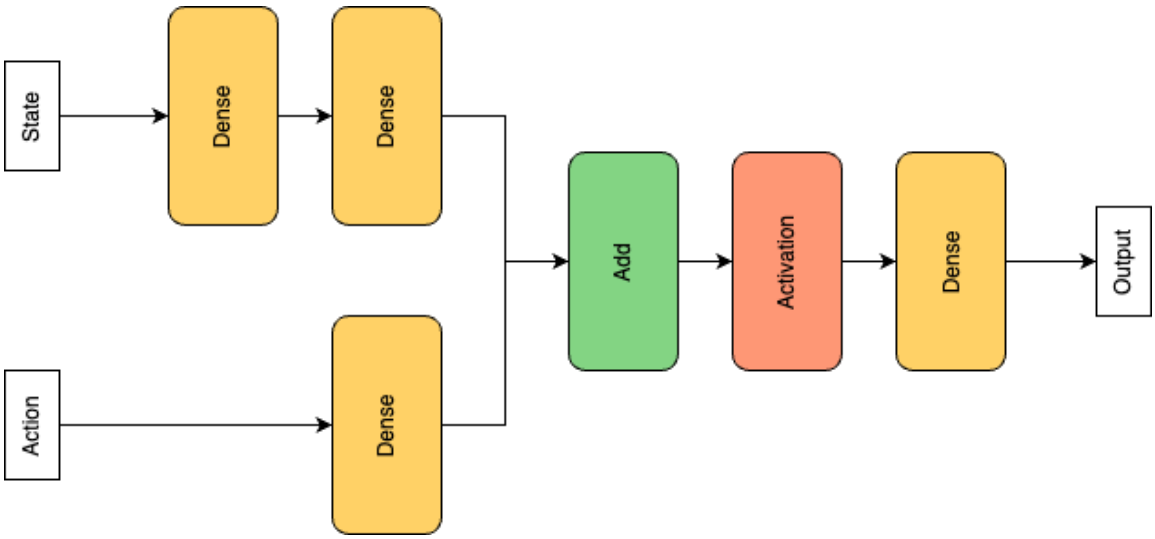
4.3 Architektúra modelov

Použité modely obsahujú ako základný prvok vrstvu perceptrónov, ktorá transformuje svoj vstup na výstup podľa trénovaných parametrov. Okrem perceptrónov sú použité aj špeciálne vrstvy ako vrstva súčtu, lambda vrstva, *bijector* vrstva aktivačná vrstva a vrstva predikujúca šum. Model *actor* aj model *critic* nebolo možné realizovať ako klasický sekvenčný model, kvôli špeciálnym vnútorným prepojeniam medzi vrstvami, a teda jedná sa o definíciu modelov pomocou orientovaného grafu. Vrstva predikcie šumu sa nenachádza v ponuke preddefinovaných vrstiev knižnice TensorFlow a bolo teda potrebné ju vytvoriť ako vlastnú vrstvu dedením z triedy všeobecnej vrstvy. Výstup modelu *actor* je privedený na jeden zo vstupov modelu *critic*, čím je zabezpečený tok gradientov od siete kritik až po sieť *actor*. V takomto prípade sú trénovateľné parametre modelu *critic* zamrazené, aby nenastávala ich modifikácia ale ich hodnota je využitá pri výpočte zmeny trénovateľných parametrov modelu *actor*.



Obrázok 13 Architektúra modelu *actor*

Na obrázku 13 je vyobrazená architektúra modelu *actor*. Stavový priestor vstupuje do prvej vrstvy perceptrónov, ktorá ho ďalej po transformovaní posiela na ďalšie dve vrstvy perceptrónov. Po tretej vrstve perceptrónov nasleduje lambda vrstva, ktorá obmedzí rozsah výstupov predchádzajúcej vrstvy podľa stanoveného intervalu ako hyperparametra. Ak má *actor* generovať stochastické akcie je potrebné predikovať hodnoty šumu pomocou vrstvy *MultivariateGaussianNoise*, ktorá ako vstup používa latentný priestor z druhej vrstvy. Predikovaný šum je potom vo vrstve súčtu sčítaný s predikovaným priemerom akcie, ktorý je výstupom lambda vrstvy. Získaný súčet je ešte potrebné obmedziť na rozsah [-1, 1], aký majú všetky akcie v použitých prostrediach. Toto obmedzenie zabezpečuje *bijector* vrstva, ktorá na vstup aplikuje TanH funkciu. V prípade predikcie deterministických akcií je výstup lambda vrstvy predstavujúci priemernú akciu privedený priamo na vstup *bijector* vrstvy. Každá z vrstiev perceptrónov obsahuje zároveň nelineárnu aktivačnú funkciu ReLU.

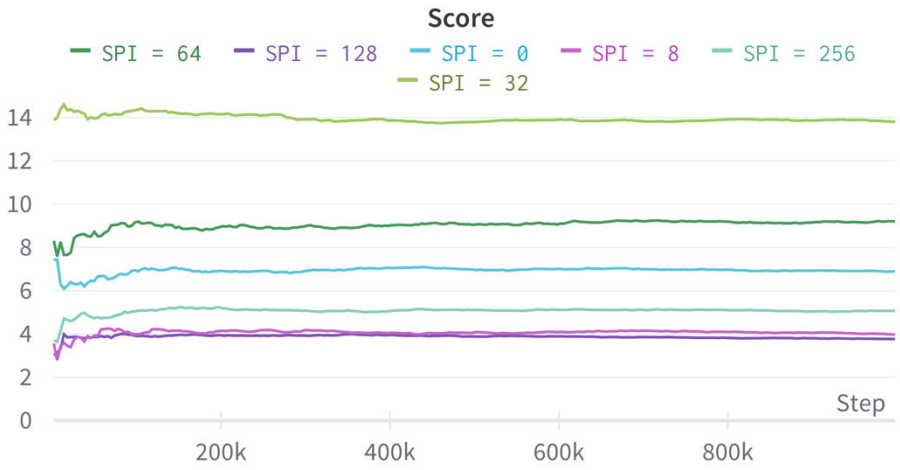


Obrázok 14 Architektúra modelu *critic*

Na obrázku 14 je vyobrazená architektúra modelu *critic*. Stavový priestor vstupuje do prvej vrstvy perceptrónov, ktorá ho ďalej po transformovaní posiela druhú vrstvu perceptrónov. Prvá vrstva obsahuje aj aktivačnú funkciu ReLU. Druhá vrstva perceptrónov neobsahuje aktivačnú funkciu a teda slúži iba ako projekčná vrstva. Akčný priestor vstupuje do vrstvy perceptrónov, ktorá tiež neobsahuje aktivačnú funkciu a slúži iba ako projekčná vrstva. Ďalej nasleduje vrstva súčtu, kde sa sčíta vetva stavov a vetva akcii. Po súčte je výsledok transformovaný aktivačnou vrstvou pomocou funkcie ReLU. Posledná výstupná vrstva perceptrónov predikuje hodnoty kvantilov, pričom ale obsahuje aj aktivačnú funkciu ReLU.

4.4 Optimalizácia hyperparametrov agenta

Optimalizácia hyperparametrov agenta bola vykonaná v zmysle úspešnosti vyriešenia všetkých zadaných úloh v rôznych prostrediach. Pre nájdenie vhodných parametrov bol použitý nástroj WanDB Sweep (WanDB, 2023b). Testovanie agentov na získané priemerné skóre pri riešení úloh prebiehalo na 10 nezávislých experimentoch v každom z prostredí. Pri každom experimente boli náhodne vygenerované hlboké neurónové siete ako aj herné prostredie, kde sa agent pohyboval.



Obrázok 15 Porovnanie skóre natrénovaného agenta pri použití rôzneho pomeru SPI

Na obrázku 15 je možné vidieť výsledky porovnania nastavení SPI pomeru z intervalu [0, 256]. Výsledky boli získané pomocou RL-Toolkit v prostredí MinitaurBulletEnv s deterministickým agentom, ktorý sa predtým učil vyriešiť zadanú úlohu. Z výsledkov vyplýva, že ideálnym nastavením SPI pomeru je **32**.

Tabuľka 2 Nájdené optimálne nastavenie hyperparametrov

Názov	**Hodnota**
Počet krokov učenia	1 000 000
γ	0,98
τ	0,005
Maximálna veľkosť úložiska interakcií	1 000 000
SPI pomer	32
Veľkosť tréningového balíčka	256
Počet neurónov v skrytých vrstvách modelu *actor*	400, 300
Inicializačná hodnota pre smerodajnú odchýlku akcií	-3,0
Hranica pre priemernú hodnotu akcie	± 2,0
Počet neurónov v skrytých vrstvách modelu *critic*	400, 300
Počet modelov *critic*	5
Počet kvantilov	35
Počet vynechaných horných kvantilov	3
Inicializačný α parameter pre reguláciu entropie akcií	1,0
Parameter rýchlosti učenia	$7{,}3 * 10^{-4}$
Gradient *clipnorm*	40,0
Nelineárna aktivačná funkcia	ReLU

Z tabuľky 2 vyplývajú všeobecne optimálne parametre, s ktorými bolo možné vyriešiť zadanú úlohu v každom z testovaných prostredí. Pre prípadné ďalšie zlepšenie skóre agenta je potrebné precízne nastaviť hyperparametre pre konkrétnu riešenú úlohu v prostredí.

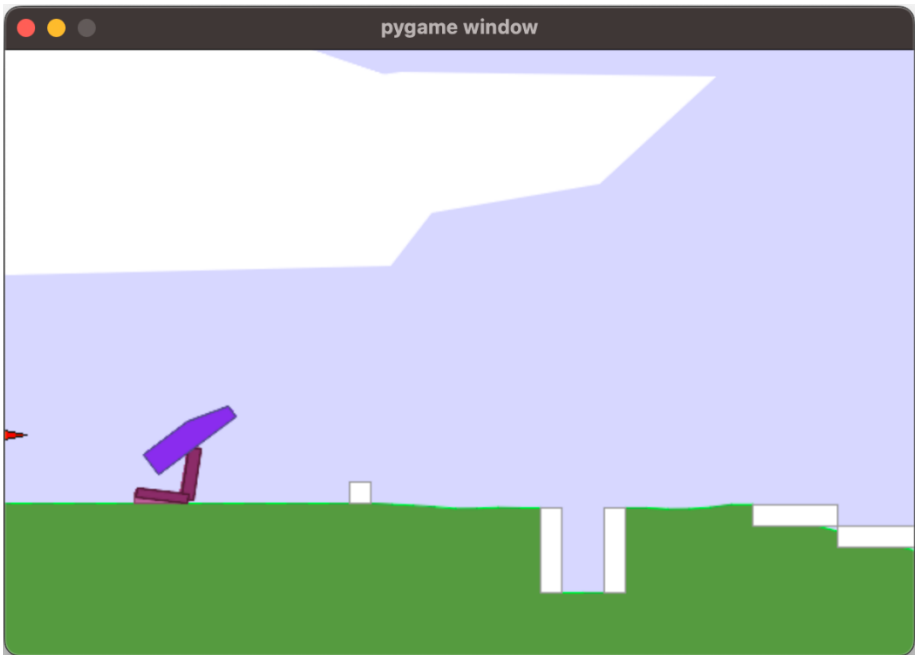
4.5 Testovacie prostredia

RL-Toolkit bol testovaný v štandardných prostrediach, ktoré používajú viaceré práce. Ich cieľom je vždy vytvoriť prostredie, agenta a zadanú úlohu tak, aby ju nezvládol vyriešiť náhodný agent. Náhodný agent je algoritmus, kedy sa akcie generujú bez znalosti stavového

priestoru a ich pravdepodobnosť výberu je daná rovnomernou distribúciou. Týmto spôsobom je možné dokázať, že agent využívajúci znalosť zo stavového priestoru a vhodnú rozhodovaciu logiku vie správne vyriešiť zadanú úlohu. V niektorých prípadoch sa porovnáva aj skóre s ľudským hráčom, čo ešte navyše môže dokázať, že použitý algoritmus je lepší ako priemerný ľudský hráč. Nakoľko je chôdza subjektívnym problémom, nie je možné presne povedať, ktorá chôdza je estetickejšia a teda porovnanie s ľudským hráčom je irelevantné.

4.5.1 BipedalWalkerHardcore-v3

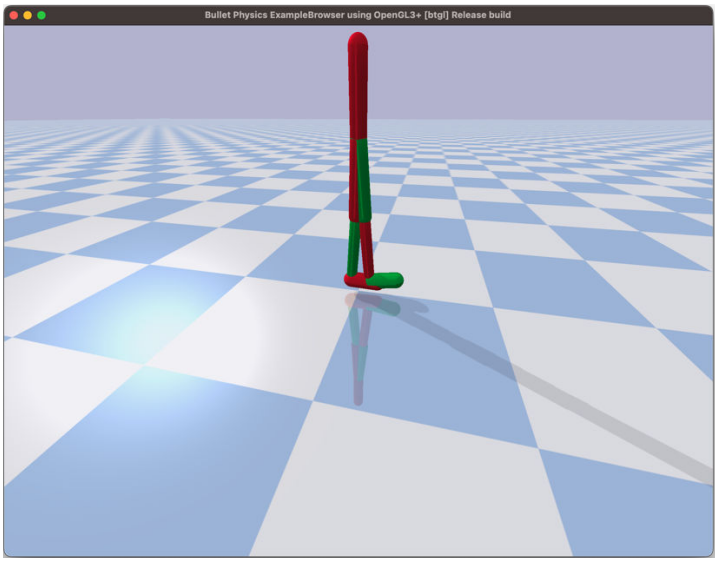
Prostredie BipedalWalkerHardcore-v3 simuluje kráčajúceho dvojnohého robota so štvoricou kĺbov. Jeho cieľom je kráčať smerom vpred pričom musí prekonávať prekážky ako priepasť, krabice a schody. Priepasti môžu mať rôznu náhodnú šírku, krabice majú rôznu náhodnú veľkosť a schody môžu ísť náhodne smerom hore ako aj smerom dole. Terén nie je celkom rovný a robot sa musí naučiť kráčať aj do kopca alebo z kopca. Stavový priestor tvorí uhlová rýchlosť trupu, horizontálna rýchlosť, vertikálna rýchlosť, polohy kĺbov a uhlové rýchlosti kĺbov, či bol kontakt nôh so zemou a desať meraní diaľkomerom *lidar*. Akčný priestor tvoria hodnoty rýchlosti rotorov v rozsahu [-1, 1] pre každý zo štyroch kĺbov na bedrách aj kolenách. Odmena sa udeľuje za postup dopredu, pričom na koniec je možné nazbierať viac ako 300 bodov. Ak robot spadne, dostane odmenu -100 bodov. Podmienky konca simulácie sú ak sa trup dotkne zeme alebo robot príde na pravý koniec dĺžky terénu, čo je signalizované špeciálnym signálom, ktorý ukončí trajektóriu na úrovni MDP. Krútiaci moment motora predstavuje malý záporný počet bodov, čo núti robota optimalizovať svoje správanie aj v zmysle šetrenia energie (Gymnasium, 2023a).



Obrázok 16 Ukážka z prostredia BipedalWalkerHardcore-v3

4.5.2 Walker2DBulletEnv-v0

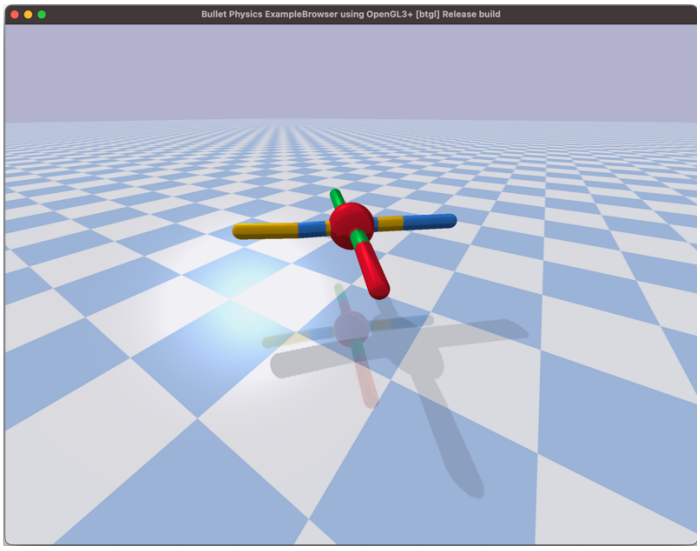
Prostredie Walker2DBulletEnv-v0 simuluje dvojnohého kráčajúceho robota so šiestimi kĺbmi. Jeho cieľom je kráčať po rovnom povrchu smerom vpred. Stavový priestor tvorí uhlová rýchlosť trupu, horizontálna rýchlosť, vertikálna rýchlosť, polohy kĺbov, uhlové rýchlosti kĺbov a výška v akej sa nachádza trup. Akčný priestor tvoria hodnoty krútiaceho momentu aplikovaného na šesť rotorov v kĺboch z rozsahu [-1, 1]. Kladná odmena sa udeľuje za každý časový krok, kedy je agent živý a agent kráča vpred čo najvyššou rýchlosťou. Agent je živý ak simulácia neskončí predčasne, čo je signalizované špeciálnym signálom, ktorý ukončí trajektóriu na úrovni MDP. Podmienky konca simulácie sú ak stavový priestor obsahuje hodnoty mimo hraníc prostredia alebo výška v akej sa nachádza trup je mimo preddefinovaného intervalu [0.2, 1.0], čo znamená, že je trup blízko pri zemi. V prípade dosiahnutia 1000 krokov simulácia končí ale na úrovni MDP sa trajektória nepreruší ale naopak blíži sa k nekonečnému MDP. Záporná odmena je udeľovaná za veľký krútiaci moment aplikovaný na rotory, čo vedie robota optimalizovať spotrebu energie (Gymnasium, 2023b).



Obrázok 17 Ukážka z prostredia Walker2DBulletEnv-v0

4.5.3 AntBulletEnv-v0

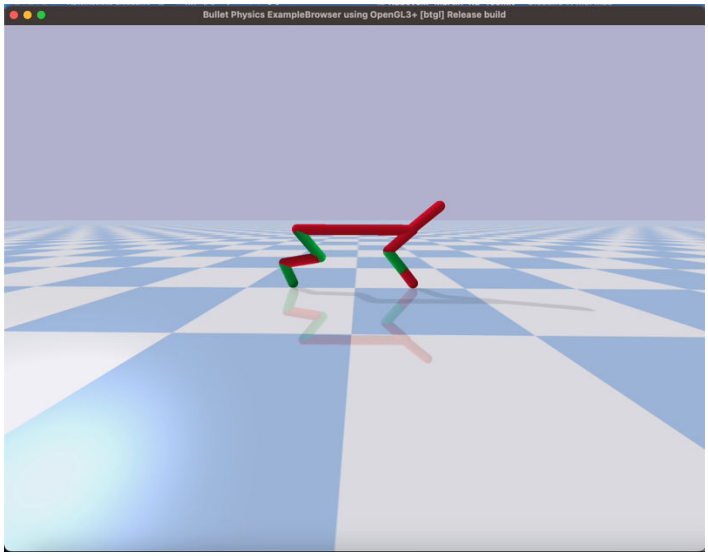
Prostredie AntBulletEnv-v0 simuluje štvornohého kráčajúceho robota s ôsmimi kĺbmi. Jeho cieľom je kráčať po rovnom povrchu smerom vpred. Stavový priestor tvorí uhlová rýchlosť trupu, orientácia trupu v priestore, horizontálna rýchlosť, vertikálna rýchlosť, polohy kĺbov, uhlové rýchlosti kĺbov a výška v akej sa nachádza trup. Akčný priestor tvoria hodnoty krútiaceho momentu aplikovaného na osem rotorov v kĺboch z rozsahu [-1, 1]. Kladná odmena sa udeľuje za každý časový krok, kedy je agent živý a agent kráča vpred čo najvyššou rýchlosťou. Agent je živý ak simulácia neskončí predčasne, čo je signalizované špeciálnym signálom, ktorý ukončí trajektóriu na úrovni MDP. Podmienky konca simulácie sú ak stavový priestor obsahuje hodnoty mimo hraníc prostredia alebo výška v akej sa nachádza trup je mimo preddefinovaného intervalu [0.2, 1.0], čo znamená, že je trup blízko pri zemi. V prípade dosiahnutia 1000 krokov simulácia končí ale na úrovni MDP sa trajektória nepreruší ale naopak blíži sa k nekonečnému MDP. Záporná odmena je udeľovaná za veľký krútiaci moment aplikovaný na rotory, čo vedie robota optimalizovať spotrebu energie (Gymnasium, 2023c).



Obrázok 18 Ukážka z prostredia AntBulletEnv-v0

4.5.4 HalfCheetahBulletEnv-v0

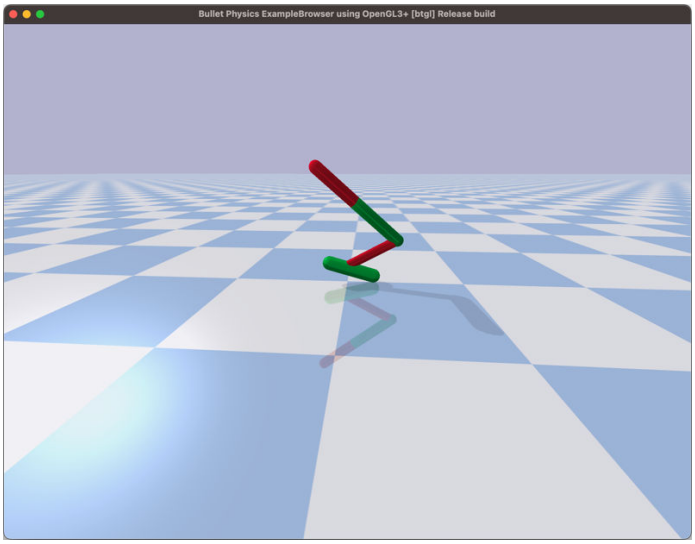
Prostredie HalfCheetahBulletEnv-v0 simuluje dvojnohého robota podobného mačke so šiestimi kĺbmi. Jeho cieľom je kráčať po rovnom povrchu smerom vpred. Stavový priestor tvorí horizontálna rýchlosť, vertikálna rýchlosť, polohy kĺbov, uhlové rýchlosti kĺbov a výška v akej sa nachádza robot. Akčný priestor tvoria hodnoty krútiaceho momentu aplikovaného na šesť rotorov v kĺboch z rozsahu [-1, 1]. Kladná odmena sa udeľuje za kráčanie agenta vpred čo najvyššou rýchlosťou. V prípade dosiahnutia 1000 krokov simulácia končí ale na úrovni MDP sa trajektória nepreruší ale naopak blíži sa k nekonečnému MDP. Záporná odmena je udeľovaná za veľký krútiaci moment aplikovaný na rotory, čo vedie robota optimalizovať spotrebu energie (Gymnasium, 2023d).



Obrázok 19 Ukážka z prostredia HalfCheetahBulletEnv-v0

4.5.5 HopperBulletEnv-v0

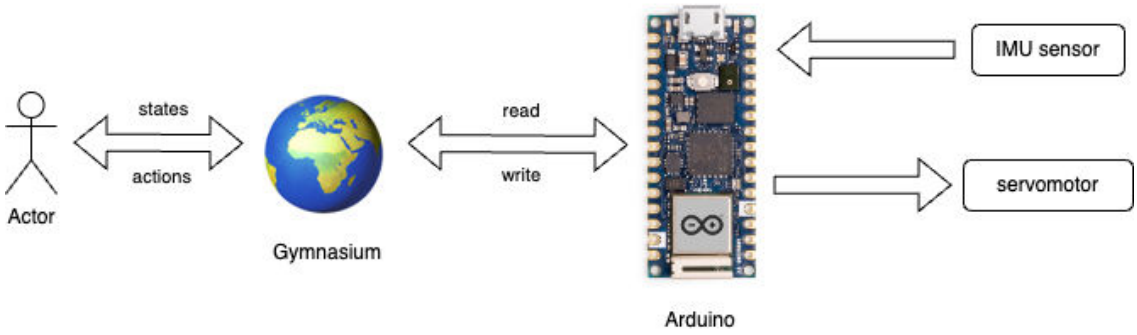
Prostredie HopperBulletEnv-v0 simuluje jednonohého skákajúceho robota s tromi kĺbmi. Jeho cieľom je skákať po rovnom povrchu smerom vpred. Stavový priestor tvorí uhlová rýchlosť trupu, horizontálna rýchlosť, vertikálna rýchlosť, polohy kĺbov, uhlové rýchlosti kĺbov a výška v akej sa nachádza trup. Akčný priestor tvoria hodnoty krútiaceho momentu aplikovaného na trojicu rotorov v kĺboch z rozsahu [-1, 1]. Kladná odmena sa udeľuje za každý časový krok, kedy je agent živý a agent kráča vpred čo najvyššou rýchlosťou. Agent je živý ak simulácia neskončí predčasne, čo je signalizované špeciálnym signálom, ktorý ukončí trajektóriu na úrovni MDP. Podmienky konca simulácie sú ak stavový priestor obsahuje hodnoty mimo stanoveného intervalu, výška v akej sa nachádza trup je mimo preddefinovaného intervalu a uhol náklonu trupu je mimo daného intervalu. V prípade dosiahnutia 1000 krokov simulácia končí ale na úrovni MDP sa trajektória nepreruší ale naopak blíži sa k nekonečnému MDP. Záporná odmena je udeľovaná za veľký krútiaci moment aplikovaný na rotory, čo vedie robota optimalizovať spotrebu energie (Gymnasium, 2023e).



Obrázok 20 Ukážka z prostredia HopperBulletEnv-v0

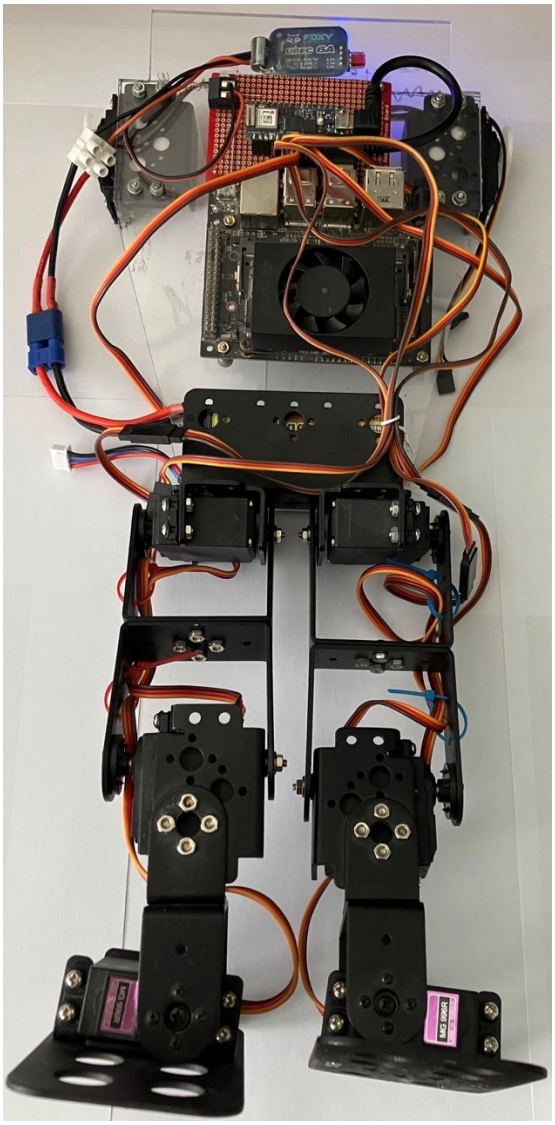
4.6 Návrh aplikácie na reálneho robota

Každé z použitých simulovaných prostredí má jednotný štandard pre komunikáciu agenta s prostredím. Stavový priestor, ktorý predstavujú senzory robota je možné posielať agentovi tým istým spôsobom, pričom však sa nebude jednať o simulované merané veličiny. Reálne senzory umiestnené na konštrukcii robota budú posielať svoje aktuálne namerané hodnoty do vytvoreného falošného simulovaného prostredia. Falošné znamená, že sa navonok správa a komunikuje s agentom ako typické simulované prostredie, avšak v skutočnosti ide o reálneho robota s reálnymi senzormi a motormi. V takomto falošnom simulovanom prostredí neprebiehajú výpočty fyziky ale jedná sa len o komunikáciu medzi agentom a reálnym svetom. Akcie, ktoré generuje agent na základe stavového priestoru sú naopak falošným simulovaným prostredím odosielané na reálne pohyblivé časti robota. RL-Toolkit teda reálnemu robotovi rozumie tak, že sa jedná o bežné simulované prostredie. Na obrázku 21 je ukážka realizácie falošného simulovaného prostredia.



Obrázok 21 Architektúra falošného simulovaného prostredia

Na obrázku 22 je ukážka reálneho kráčajúceho dvojnohého robota, ktorý využíva RL-Toolkit ako nástroj pre učenie sa úspešne vykonávať zadané úlohy v reálnom svete. Ako hlavnú riadiacu jednotku robot využíva NVIDIA Jetson Xavier NX, ktorá obsahuje aj grafický procesor od NVIDIA. Je teda možné priamo na robotovi spúšťať veľké modely neurónových sietí, avšak nie je určená na ich učenie. Ako senzory používa IMU, teda kombináciu akcelerometra a gyroskopu, aby vedel určiť svoju polohu v priestore. Pohyb kĺbov zabezpečuje šestica servomotorov. Komunikáciu so senzormi a ovládanie servomotorov zabezpečuje riadiaca jednotka Arduino Nano RP2040 Connect. Tá potom priamo cez USB rozhranie komunikuje s NVIDIA Jetson Xavier NX. RL-Toolkit je na robotovi spustený v móde *actor* a teda robot iba zbiera interakcie (skúsenosti). Robot je pomocou WiFi pripojený k databázovému serveru Reverb, kam ukladá získané dáta. Inštancia *learner* je spustená na výkonnom serveri s grafickou kartou, ktorý sa taktiež pripojí k databáze. Ako zdroj energie robot využíva lítium-polymérový akumulátor s kapacitou 1800mAh s nominálnym napätím 11,1V.



Obrázok 22 Ukážka reálneho dvojnohého robota

4.7 Výsledky

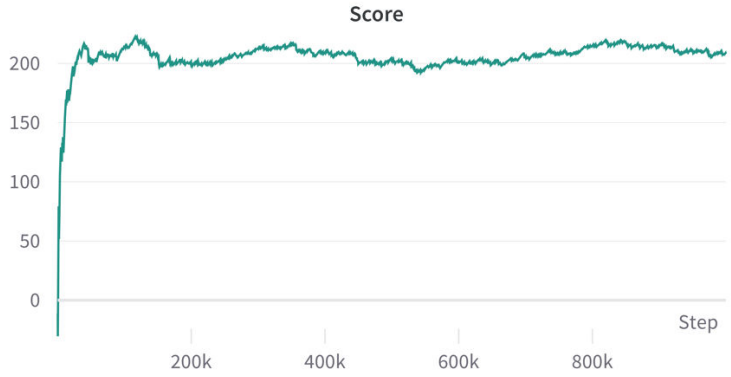
Učenie aplikovaných hlbokých neurónových sietí agenta prebiehalo s využitím grafickej karty NVIDIA GeForce RTX 3090 ako akcelerátora výpočtov. Simulované robotické prostredia nevyužívali akceleráciu výpočtov pomocou grafickej karty a teda výpočty fyziky boli realizované na CPU Intel Core i9-12900K.

Tabuľka 3 Porovnanie skóre agentov s použitím rôznych algoritmov učenia

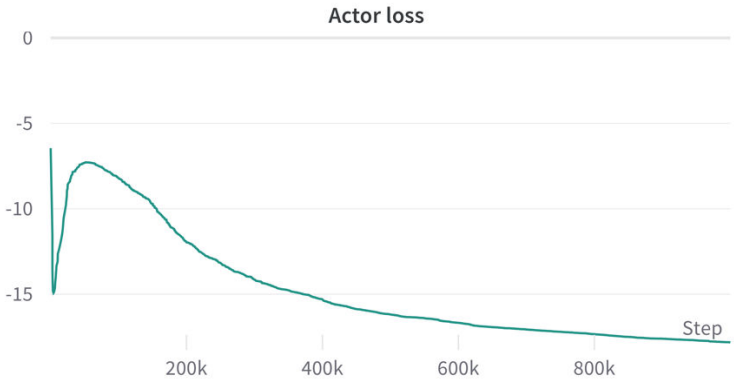
Názov prostredia	**SAC + gSDE (Raffin, 2021a)**	**SAC + gSDE + Huber loss**	**SAC + TQC + gSDE (Stable Baseline Contrib, 2020)**	**RL-Toolkit**
BipedalWalkerHardcore-v3	13 ± 18	**239 ± 118**	228 ± 18	205 ± 134
Walker2DBulletEnv-v0	2270 ± 28	2732 ± 96	2535 ± 94	**3123 ± 594**
AntBulletEnv-v0	3106 ± 61	3460 ± 119	3700 ± 37	**3993 ± 214**
HalfCheetahBulletEnv-v0	2945 ± 95	3003 ± 226	**3041 ± 157**	2762 ± 153
HopperBulletEnv-v0	2515 ± 50	**2555 ± 405**	2401 ± 62	2151 ± 664

Podľa výsledkov skóre v jednotlivých prostrediach z tabuľky 3 je dokázané, že *Huber loss* podobne ako RL-Toolkit dosahovali lepšie výsledky oproti konkurenčným prácam. Z tabuľky vyplýva, že použitie chybovej funkcie menej citlivej na odľahlé hodnoty či kvantilovej regresie v kombinácii s databázovým serverom, viedlo k lepším výsledkom a presnejšiemu učeniu kritiky vykonaných akcií. V priemere bolo možné dosiahnuť zlepšenia skóre o **9,38 %**. Bez použitia databázového servera Reverb bol priemerný čas učenia agenta **1 hodina 35 minút**, avšak s použitím databázového servera sa učenie spomalilo a priemerný čas bol **5 hodín 19 minút**. Analýzou sa zistilo, že s použitím databázového servera a konkrétnym nastavením SPI pomeru sa nazbieralo 8 násobne viacej skúseností z prostredia, čo však malo za následok výrazné spomalenie, nakoľko sú výpočty simulácie prostredia realizované iba na CPU. Z experimentov vyplýva, že použitie kombinácie algoritmov *Soft Actor-Critic, generalized state-dependent exploration* a chybovej funkcie *Huber loss* viedlo k zlepšeniu priemerného skóre agenta v dvoch simuláciách. Avšak až kombinácia s databázovým serverom a optimalizované hyperparametre, čo ponúkol RL-Toolkit viedli k zlepšeniu v ďalších dvoch typoch simulácií. Ako sa pri porovnaní RL-Toolkit s konkurenčnými nástrojmi ukázalo je výhodné kombinovať algoritmy a využívať špecializované nástroje ako monitorovacie, optimalizačné, databázové a simulačné. RL-Toolkit sa svojou špecializáciou na robotiku taktiež osvedčil pri použití na reálnom robotovi, kde úspešne vytvoril falošné simulované prostredie. Agent si myslel, že ide o typické simulované prostredie pričom ako stavy mu boli posielané reálne merania

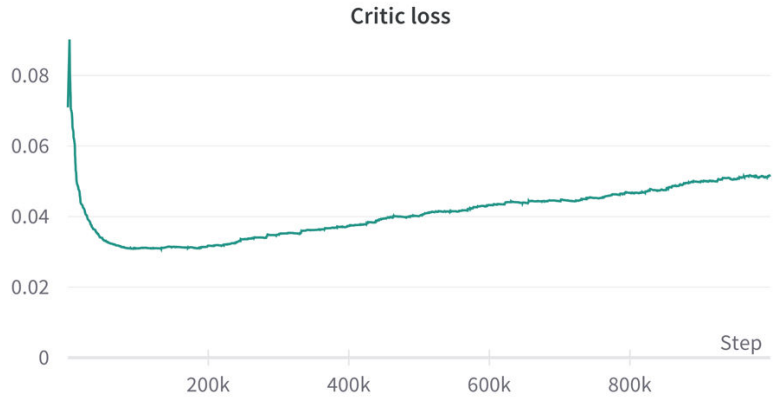
z akcelerometra, gyroskopu a polohy kĺbov v stupňoch. Predikované akcie hlbokou neurónovou sieťou boli posielané na servomotory a ovládali nohy robota. Odmenu predstavovalo dosiahnutie správnej polohy trupu v priestore definovanej polohou podľa Eulera. Cieľom nebolo robota naučiť ale overiť si, či je možné aplikovať RL-Toolkit v reálnom prostredí, čo sa potvrdilo.



Obrázok 23 Vývoj skóre agenta v prostredí BipedalWalkerHardcore-v3

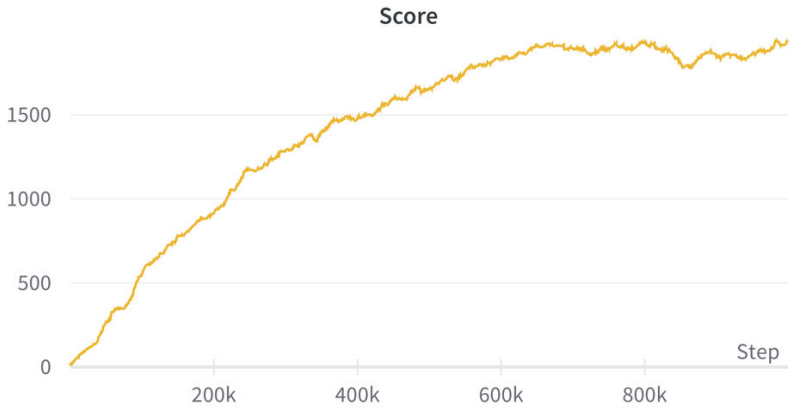


Obrázok 24 Vývoj chyby *actor* modelu v prostredí BipedalWalkerHardcore-v3

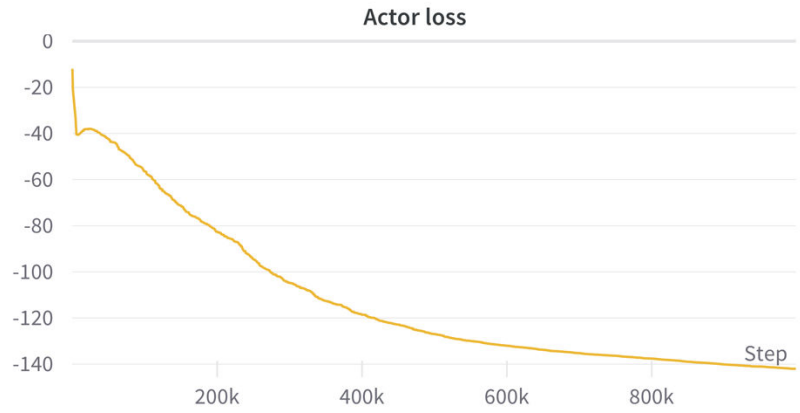


Obrázok 25 Vývoj chyby *critic* modelu v prostredí BipedalWalkerHardcore-v3

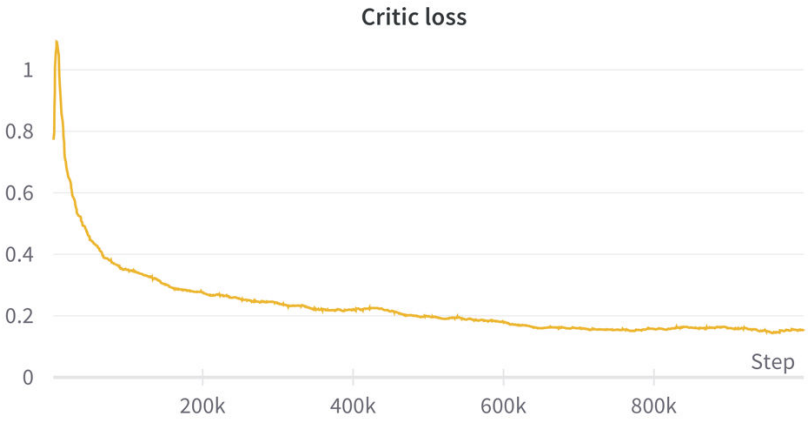
Pri učení sa v prostredí BipedalWalkerHardcore-v3 je možné pozorovať logaritmický vývoj skóre v čase, kedy agent už po približne **100 000** iteráciách bol schopný vyriešiť požadovanú úlohu. Chyba siete *actor* v čase klesala, čo dokazuje, že nastala maximalizácia Q-hodnoty. Chyba siete *critic* v prvých iteráciách klesala, potom však začala narastať z čoho vyplýva meniaca sa cieľová Q-hodnota., teda Q-hodnota, ktorá má byť predikovaná *critic* modelom.



Obrázok 26 Vývoj skóre agenta v prostredí Walker2DBulletEnv-v0

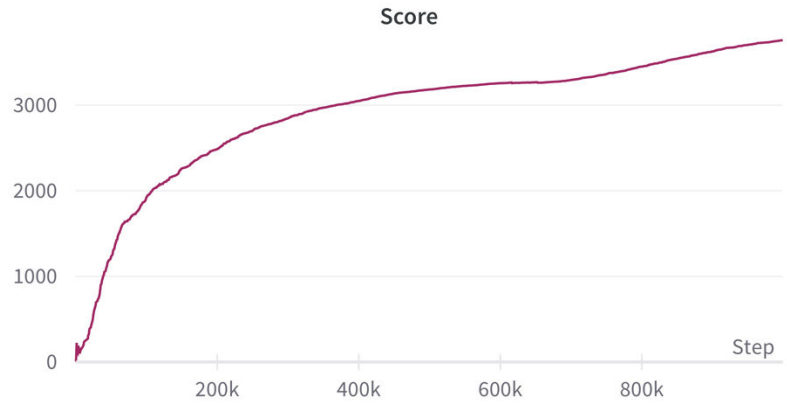


Obrázok 27 Vývoj chyby *actor* modelu v prostredí Walker2DBulletEnv-v0

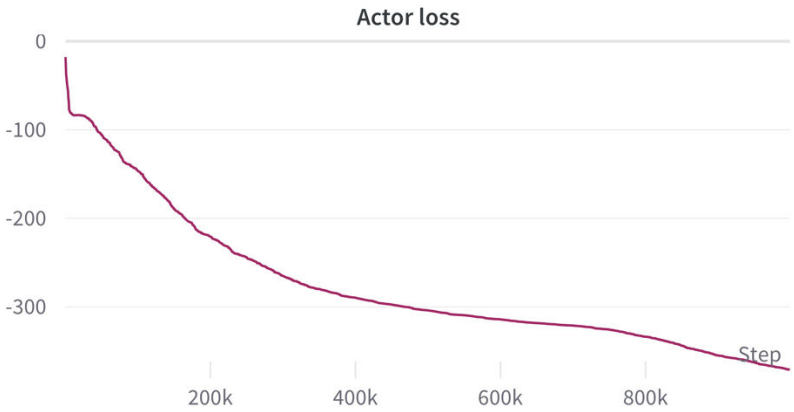


Obrázok 28 Vývoj chyby *critic* modelu v prostredí Walker2DBulletEnv-v0

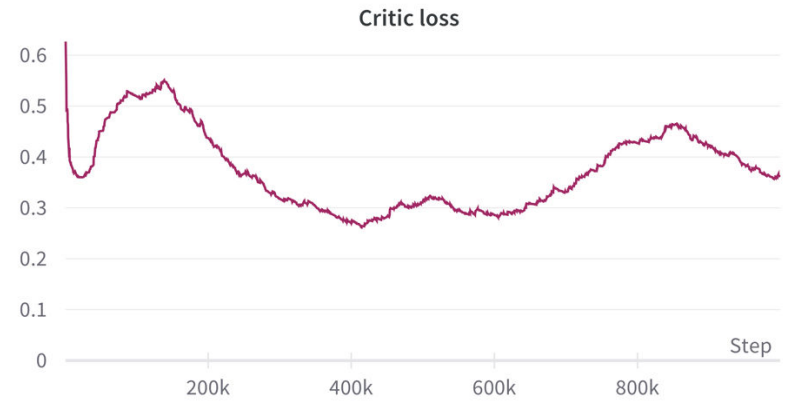
Pri učení sa v prostredí Walker2DBulletEnv-v0 je možné pozorovať logaritmický vývoj skóre v čase, kedy agent po približne až **700 000** iteráciách bol schopný vyriešiť požadovanú úlohu. Chyba siete *actor* v čase klesala, čo dokazuje, že nastala maximalizácia Q-hodnoty. Chyba siete *critic* v čase postupne klesala, teda sieť sa zlepšovala pri predikcii cieľovej Q-hodnoty. Po porovnaní niekoľkých meraní bolo zistené, že špička chyby pri *critic* modely v prvých pár iteráciách učenia je pravdepodobne spôsobená inicializačnými hodnotami modelu.



Obrázok 29 Vývoj skóre agenta v prostredí AntBulletEnv-v0



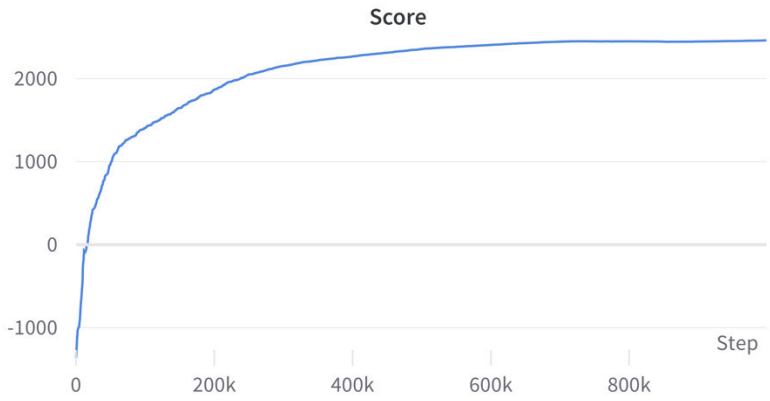
Obrázok 30 Vývoj chyby *actor* modelu v prostredí AntBulletEnv-v0



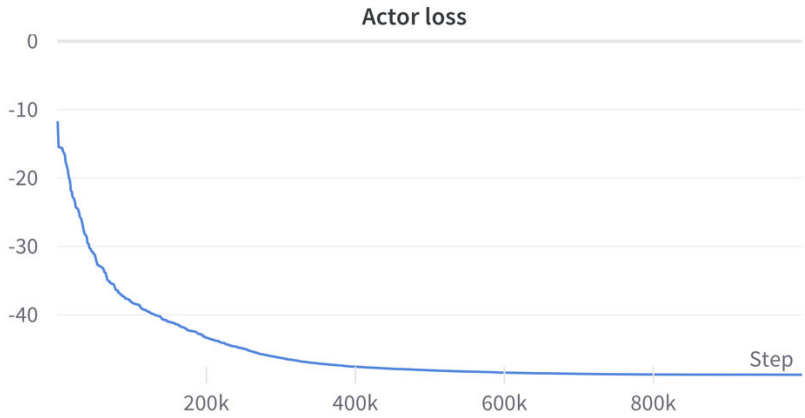
Obrázok 31 Vývoj chyby *critic* modelu v prostredí AntBulletEnv-v0

Pri učení sa v prostredí AntBulletEnv-v0 je možné pozorovať logaritmický vývoj skóre v čase, kedy agent po celú dobu učenia vylepšuje svoje skóre. Chyba siete *actor* v čase klesala, čo dokazuje, že nastala maximalizácia Q-hodnoty. Chyba siete *critic* v čase oscilovala čo

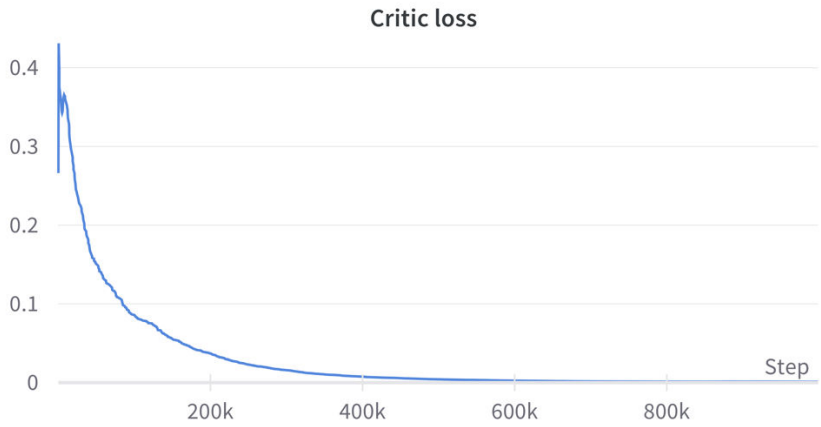
znamená, že sa cieľová Q-hodnota čiastočne v čase menila, teda cieľ, na ktorý bola sieť učená, sa v čase menil.



Obrázok 32 Vývoj skóre agenta v prostredí HalfCheetahBulletEnv-v0

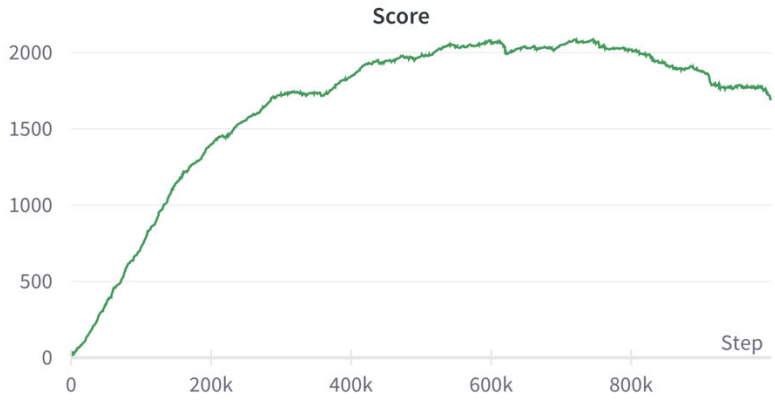


Obrázok 33 Vývoj chyby *actor* modelu v prostredí HalfCheetahBulletEnv-v0

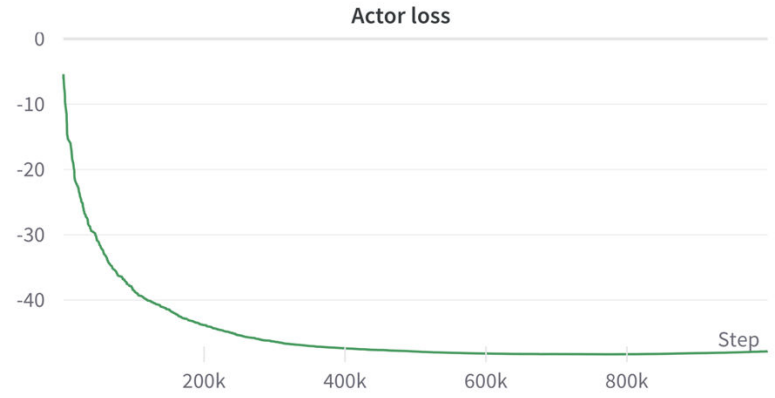


Obrázok 34 Vývoj chyby *critic* modelu v prostredí HalfCheetahBulletEnv-v0

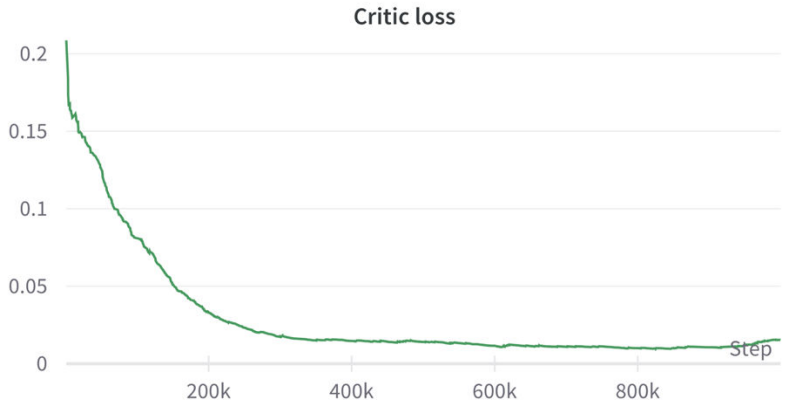
Pri učení sa v prostredí HalfCheetahBulletEnv-v0 je možné pozorovať logaritmický vývoj skóre v čase, kedy agent po približne **500 000** iteráciách bol schopný vyriešiť požadovanú úlohu. Chyba siete *actor* v čase klesala, čo dokazuje, že nastala maximalizácia Q-hodnoty. Chyba siete *critic* v čase postupne klesala, teda sieť sa zlepšovala pri predikcii cieľovej Q-hodnoty.



Obrázok 35 Vývoj skóre agenta v prostredí HopperBulletEnv-v0



Obrázok 36 Vývoj chyby *actor* modelu v prostredí HopperBulletEnv-v0



Obrázok 37 Vývoj chyby *critic* modelu v prostredí HopperBulletEnv-v0

Pri učení sa v prostredí HopperBulletEnv-v0 je možné pozorovať logaritmický vývoj skóre v čase, kedy agent po približne **500 000** iteráciách bol schopný vyriešiť požadovanú úlohu. Avšak po približne 700 000 iteráciách toto skóre degradovalo. Po porovnaní niekoľkých meraní sa ukázalo, že nie vždy dochádzalo ku degradácie skóre, čo by mohlo byť spôsobené inicializačnými parametrami modelu. Išlo o jediné parametre, čo sa menili medzi jednotlivými meraniami. Vzhľadom na to, že sa zbieralo 8 000 000 interakcií je štatisticky pravdepodobné, že sa množiny interakcií medzi jednotlivými meraniami podobali. Tu by pomohlo špecializovanejšie nastavenie hyperparametrov iba pre túto konkrétnu úlohu, ďalšou možnosťou by bolo zastavenie procesu učenia pred iteráciou, kedy skóre začína klesať. Chyba siete *actor* v čase klesala, čo dokazuje, že nastala maximalizácia Q-hodnoty. Chyba siete *critic* v čase postupne klesala, teda sieť sa zlepšovala pri predikcii cieľovej Q-hodnoty, až na posledné iterácie, kedy chyba mierne narástla. Z tohto nárastu vyplýva meniaca sa cieľová Q-hodnota.

Záver

Cieľom práce bolo vytvorenie súboru nástrojov RL-Toolkit pre samoučenie sa robotov v simulovaných prostrediach. Úspešné splnenie cieľu bolo ilustrované na optimalizácii učenia v niekoľkých štandardných prostrediach vymyslených firmou OpenAI (rovnakou, čo neskôr vytvorila ChatGPT). Výslední roboti boli vo väčšine prípadov lepšie naučení ako roboti učení konkurenčnými prístupmi. Podstatnou pridanou hodnotou oproti ostatným riešeniam je aj integrovanie databázového servera do RL-Toolkitu.

Simulovaní roboti museli získať čo najlepšie skóre, čo predstavuje najideálnejšie splnenie danej úlohy. K dosiahnutiu cieľa je v práci použitá umelá neurónová sieť, ktorá transformuje stavový vektor predstavujúci merania senzorov na akčný vektor, teda pohyby motorov robota. Robot využíva učenie posilňovaním, ktorého princípom je maximalizácia Q-hodnoty predstavujúca kvalitu daného prechodu z aktuálneho stavu do nasledovného stavu vplyvom vykonanej akcie. RL-Toolkit rozdelil inštancie do nezávislých skupín, čo dovolilo spustenie *actor* inštancie na menej výkonnom hardvéri reálneho robota, pričom databázový server podobne ako inštancia *learner* sú spustené na oddelenom výkonnom počítači. Inštancia databázového servera si vyžadovala počítač s veľkou kapacitou operačnej pamäti. Na rozdiel od toho inštancia *learner* si vyžadovala vysoký výpočtový výkon poskytnutý grafickou kartou obsiahnutou v počítači.

RL-Toolkit dosiahol v priemere zlepšenie skóre o **9,38 %**, čo sa podarilo dosiahnuť zlepšením presnosti predikcie Q-hodnoty, od ktorej záviseli generované akcie. Toto vylepšenie znamená, že je ho možné používať pre budúci výskum v oblasti RL. Ukázalo sa, že je v zásade výhodnejšie používať chybové funkcie menej citlivé na odľahlé hodnoty s hladkým tvarom tejto funkcie. Taktiež sa potvrdila výhoda v použití databázového servera DeepMind Reverb ako špecializovanejšieho nástroja pre správu získaných interakcií agentom v prostrediach.

RL-Toolkit poskytuje taktiež monitorovací nástroj Weights & Biases a 3 rôzne populárne prostredia simulácií. Úspešný sa ukázal aj návrh architektúry pre reálneho robota, ktorý aplikuje RL-Toolkit do reálneho sveta.

Pri budúcom vývoji by bol RL-Toolkit doplnený o ďalšie typy algoritmov učenia posilňovaním, čo by umožnilo napríklad aj tréning v diskrétnych akčných priestoroch. Tieto algoritmy by umožnili riešiť úlohy ako sú hry pre Atari alebo hľadanie východu z

bludiska. Taktiež pri súčasnom riešení sa agent spolieha na časté odmeny z prostredia, čo nie je vždy možné zabezpečiť ako napríklad pri úlohách, kedy sa kladný či záporný výsledok nášho správania potvrdí až pri konci simulácie. Potenciálne môže byť RL-Toolkit rozšírený o zabudovanie vnútornej odmeny, ktorá zvládne aj problém so zriedkavými vonkajšími odmenami (Kubovčík, 2023).

Zoznam použitej literatúry

1. AGARAP, Abien Fred. Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375, 2018.
2. BELLEMARE, Marc G.; DABNEY, Will; MUNOS, Rémi. A distributional perspective on reinforcement learning. In: International conference on machine learning. PMLR, 2017. p. 449-458.
3. BHATT, S. 2018. Reinforcement Learning 101. In: *Towards Data Science*. [online]. Towards Data Science [cit. 2022-08-12]. Dostupné z: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
4. BROWNLEE, J. 2016. Crash Course on Multi-Layer Perceptron Neural Networks. In: *Machine Learning Mastery*. [online]. Machinelearningmastery [cit. 2022-08-12]. Dostupné z: https://machinelearningmastery.com/neural-networks-crash-course/
5. CASSIRER, Albin, et al. Reverb: A framework for experience replay. arXiv preprint arXiv:2102.04736, 2021.
6. DABNEY, Will, et al. Distributional reinforcement learning with quantile regression. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2018.
7. FEDUS, William, et al. Revisiting fundamentals of experience replay. In: International Conference on Machine Learning. PMLR, 2020. p. 3061-3071.
8. FUJIMOTO, Scott; HOOF, Herke; MEGER, David. Addressing function approximation error in actor-critic methods. In: International conference on machine learning. PMLR, 2018. p. 1587-1596.
9. GHENIS, M. 2018. Quantile regression, from linear models to trees to deep learning. In: *Machine Learning Mastery*. [online]. Towards Data Science [cit. 2022-08-12]. Dostupné z: https://towardsdatascience.com/quantile-regression-from-linear-models-to-trees-to-deep-learning-af3738b527c3
10. GOKCESU, Kaan; GOKCESU, Hakan. Generalized huber loss for robust learning and its efficient minimization for a robust statistics. arXiv preprint arXiv:2108.12627, 2021.

11. GUO, Hongquan, et al. Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach. Resources Policy, 2021, 74: 101474.

12. GYMNASIUM. 2023a. Bipedal Walker. In: *Gymnasium.* [online]. [cit. 2023-03-09]. Dostupné z: https://gymnasium.farama.org/environments/box2d/bipedal_walker/

13. GYMNASIUM. 2023b. Walker2D. In: *Gymnasium.* [online]. [cit. 2023-03-09]. Dostupné z: https://gymnasium.farama.org/environments/mujoco/walker2d/

14. GYMNASIUM. 2023c. Ant. In: *Gymnasium.* [online]. [cit. 2023-03-09]. Dostupné z: https://gymnasium.farama.org/environments/mujoco/ant/

15. GYMNASIUM. 2023d. Half Cheetah. In: *Gymnasium.* [online]. [cit. 2023-03-09]. Dostupné z: https://gymnasium.farama.org/environments/mujoco/half_cheetah/

16. GYMNASIUM. 2023e. Hopper. In: *Gymnasium.* [online]. [cit. 2023-03-09]. Dostupné z: https://gymnasium.farama.org/environments/mujoco/hopper/

17. HAARNOJA, Tuomas, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. PMLR, 2018. p. 1861-1870.

18. HOFFMAN, Matthew W., et al. Acme: A research framework for distributed reinforcement learning. arXiv preprint arXiv:2006.00979, 2020.

19. HUANG, Shengyi, et al. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. The Journal of Machine Learning Research, 2022, 23.1: 12585-12602.

20. MATLAB 2023. Reinforcement Learning Toolbox. In: *MATLAB*. [online]. MATLAB [cit. 2022-08-12]. Dostupné z: https://www.mathworks.com/products/reinforcement-learning.html

21. KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

22. KUBOVČÍK, Martin; DIRGOVÁ LUPTÁKOVÁ, Iveta; POSPÍCHAL, Jiří. Signal Novelty Detection as an Intrinsic Reward for Robotics. Sensors, 2023, 23.8: 3985. https://doi.org/10.3390/s23083985

23. KUZNETSOV, Arsenii, et al. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In: International Conference on Machine Learning. PMLR, 2020. p. 5556-5566.

24. MCGONAGLE, J. et al. 2022. Backpropagation. In: *Brilliant*. [online]. Brilliant [cit. 2022-08-12]. Dostupné z: https://brilliant.org/wiki/backpropagation/

25. PASCANU, Razvan; MIKOLOV, Tomas; BENGIO, Yoshua. On the difficulty of training recurrent neural networks. In: International conference on machine learning. Pmlr, 2013. p. 1310-1318.

26. RAFFIN, Antonin; KOBER, Jens; STULP, Freek. Smooth exploration for robotic reinforcement learning. In: Conference on Robot Learning. PMLR, 2022a. p. 1634-1644.

27. RAFFIN, Antonin, et al. Stable-baselines3: Reliable reinforcement learning implementations. The Journal of Machine Learning Research, 2021b, 22.1: 12348-12355.

28. SALEH, Resve A., et al. Statistical properties of the log-cosh loss function used in machine learning. arXiv preprint arXiv:2208.04564, 2022.

29. STABLE BASELINE CONTRIB. 2020. TQC. In: *Stable Baseline Contrib 2020.* [online]. Stable baseline [cit. 2023-03-09]. Dostupné z: https://sb3-contrib.readthedocs.io/en/master/modules/tqc.html

30. STÉPHANOVITCH, Arthur, et al. Optimal 1-Wasserstein Distance for WGANs. arXiv preprint arXiv:2201.02824, 2022.

31. SUTTON, Richard S.; BARTO, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.

32. TENSORFLOW. 2023. Losses. In: *TensorFlow.* [online]. [cit. 2023-03-09]. Dostupné z: https://github.com/keras-team/keras/blob/v2.12.0/keras/losses.py#L1876-L1919

33. WEIGHTS & BIASES. 2023a. The developer-first MLOps platform. In: *Weights & Biases.* [online]. WanDB [cit. 2023-04-11] Dostupné z: https://wandb.ai/site

34. WEIGHTS & BIASES. 2023b. Tune Hyperparameters. In: *Weights & Biases Docs 2023*. [online]. [cit. 2023-03-09]. Dostupné z: https://docs.wandb.ai/guides/sweeps

Prílohy

Príloha A: Používateľská príručka

Je dôležité poznamenať, že tu prezentovaný prístup je založený na princípoch open-source a neobsahuje grafické používateľské rozhranie (GUI). Táto používateľská príručka je určená pre odborníkov s technickým vzdelaním, ktorí sú oboznámení s koncepciami programovania a rozhraniami príkazového riadka.

V tejto príručke nájdete stručné informácie o používaní a navigácii v kóde vrátane pokynov na inštaláciu a príkladov použitia. Uvedený je aj zoznam dostupných prostredí pre riešenie rôznych problémov.

RL-Toolkit je možné spustiť v kontajnerovej službe Docker, ktorý je potrebné mať nainštalovaný pred spustením alebo priamo v systéme s predinštalovaným Python interpreterom a potrebnými knižnicami. Odporúčaný operačný systém Ubuntu 20.04 LTS, interpreter Python 3.9 s knižnicami Gym v0.25.1, Box2D v2.4.1, PyBullet v3.2.4, TensorFlow v2.9.1, TensorFlow Probability v0.17.0, WanDB v0.13.0, dm-reverb 0.8.0.

Docker: https://www.docker.com/products/docker-desktop

Zoznam simulácií:

- BipedalWalkerHardcore-v3
- Walker2DBulletEnv-v0
- AntBulletEnv-v0
- HalfCheetahBulletEnv-v0
- HopperBulletEnv-v0
- HumanoidBulletEnv-v0
- MinitaurBulletEnv-v0

Pre zavedenie kontajnera treba v príkazovom riadku systému (cmd, bash, zsh) spustiť:
docker build -t markub/rl-toolkit:latest -f docker/Dockerfile .

Pre potvrdenie príkazu treba stlačiť ENTER.

Súbor Dockerfile sa nachádza na CD a pred zavedením kontajnera treba zmeniť v príkazovom riadku aktuálny adresár na hlavný priečinok projektu. Na konci príkazu je potrebné nezabudnúť na bodku označujúcu, že sa jedná o aktuálny adresár.

Kontajner sa spustí pomocou príkazu:

docker run -it --rm markub/rl-toolkit:latest

Pre potvrdenie príkazu treba stlačiť ENTER.

V adresári **/root/rl-toolkit** sa nachádza hlavný priečinok projektu.

Inštalácia RL-Toolkit:

python3 -m pip install --no-cache-dir -e .[all]

Pre potvrdenie príkazu treba stlačiť ENTER.

V prípade operačného systému Ubuntu alebo Debian treba mať nainštalované knižnice pomocou príkazu:

apt update -y && apt install swig -y

Pre potvrdenie každého príkazu treba stlačiť ENTER.

RL-Toolkit Training mode:

Pre spustenie ***server***, ktorý bude ukladať nazbierané interakcie je potrebné spustiť v príkazovom riadku:

python3 -m rl_toolkit -c ./rl_toolkit/config.yaml -e GAME_NAME server

GAME_NAME predstavuje názov simulácie zo zoznamu dostupných simulácií.

Pre potvrdenie príkazu treba stlačiť ENTER.

Pre spustenie ***agent***, ktorý bude zbierať interakcie na základe vykonaných akcií v prostredí simulácie je potrebné spustiť v príkazovom riadku:

python3 -m rl_toolkit -c ./rl_toolkit/config.yaml -e GAME_NAME agent --db_server SERVER_NAME

GAME_NAME predstavuje názov simulácie zo zoznamu dostupných simulácií.

SERVER_NAME predstavuje názov servera alebo jeho IP adresu.

Pre potvrdenie príkazu treba stlačiť ENTER.

Pre spustenie ***learner***, ktorý sa bude z nazbieraných interakcií učiť je potrebné spustiť v príkazovom riadku:

python3 -m rl_toolkit -c ./rl_toolkit/config.yaml -e GAME_NAME learner --db_server SERVER_NAME

GAME_NAME predstavuje názov simulácie zo zoznamu dostupných simulácií.

SERVER_NAME predstavuje názov servera alebo jeho IP adresu.

Pre potvrdenie príkazu treba stlačiť ENTER.

RL-Toolkit Testing mode:

Po natrénovaní a uložení naučenej vedomosti je možné túto vedomosť využiť prakticky, čo sa dá overiť v testovacom móde. Pre spustenie **testovacieho agenta** je potrebné spustiť v príkazovom riadku:

python3 -m rl_toolkit -c ./rl_toolkit/config.yaml -e GAME_NAME tester -f FILE_NAME

GAME_NAME predstavuje názov simulácie zo zoznamu dostupných simulácií.

FILE_NAME predstavuje cestu k súboru s uloženými naučenými vedomosťami.

Pre potvrdenie príkazu treba stlačiť ENTER.

Konfigurácia hyperparametrov:

Pri každej simulácii je vhodné upraviť hyperparametre podľa potreby simulácie, aby agent dosahoval čo možno najlepšie skóre. Hyperparametre sa nachádzajú vo zvláštnom súbore formátu YAML, ktorý je umiestnený v **rl_toolkit/config.yaml**. Okrem hyperparametrov obsahuje aj port databázového servera, ktorý je predvolený ako 8000.

```yaml
# Database
port: 8000

# Agent
env_steps: 8
warmup_steps: 10000

# Learner
train_steps: 1000000
gamma: 0.99
tau: 0.01
min_replay_size: 10000
max_replay_size: 1000000
samples_per_insert: 32
batch_size: 256
log_interval: 1000

# Paths
save_path: "./save/model"
db_path: "./save/db"

# Actor model
Actor:
  units: [400, 300]
  init_noise: -3.0
  learning_rate: !!float 7.3e-4
  clip_mean_min: -2.0
  clip_mean_max: 2.0

# Critic model
Critic:
  count: 5
  units: [400, 300]
  n_quantiles: 35
  top_quantiles_to_drop: 3
  learning_rate: !!float 7.3e-4

# Alpha parameter
Alpha:
  init: 1.0
  learning_rate: !!float 7.3e-4
```

Obrázok 38 Ukážka základnej konfigurácie

Príloha B: Zdrojový kód s popisom

Obrázok 39 popisuje súbor rl-toolkit/networks/layers/noise.py definujúci vlastnú vrstvu, ktorá generuje náhodné premenné prostredníctvom Gaussovho šumu ovplyvnené aktuálnym stavovým vektorom. Tie sa následne pripočítajú k deterministickým akciám predikovanými *actor* častou modelu.

```
36      def build(self, input_shape):
37          super(MultivariateGaussianNoise, self).build(input_shape)
38
39          self.kernel = self.add_weight(
40              name="kernel",
41              shape=(input_shape[-1], self.units),
42              initializer=self.kernel_initializer,
43              regularizer=self.kernel_regularizer,
44              constraint=self.kernel_constraint,
45              trainable=True,
46          )
47          self.epsilon = self.add_weight(
48              name="epsilon",
49              shape=(input_shape[-1], self.units),
50              initializer=initializers.Zeros(),
51              trainable=False,
52          )
53
54          # Re-new noise matrix
55          self.sample_weights()
56
57      def call(self, inputs):
58          return tf.matmul(inputs, self.epsilon)
59
60      def get_config(self):
61          config = super(MultivariateGaussianNoise, self).get_config()
62          config.update(
63              {
64                  "units": self.units,
65                  "kernel_initializer": initializers.serialize(self.kernel_initializer),
66                  "kernel_regularizer": regularizers.serialize(self.kernel_regularizer),
67                  "kernel_constraint": constraints.serialize(self.kernel_constraint),
68              }
69          )
70
71          return config
72
73      @property
74      def scale(self):
75          return tf.math.softplus(self.kernel)
76
77      def sample_weights(self):
78          w_dist = tfp.distributions.MultivariateNormalDiag(
79              loc=tf.zeros_like(self.kernel), scale_diag=(self.scale + 1e-6)
80          )
81          self.epsilon.assign(w_dist.sample())
```

Obrázok 39 Definícia *Multivariate Gaussian Noise for exploration* vrstvy

Riadok 39-46 definícia parametrov učených počas tréningu modelu, ktoré reprezentujú základ pre výpočet smerodajnej odchýlky Gaussovho šumu.

Riadok 47-52 definícia parametrov vygenerovaného šumu, ktoré sa použijú v kombinácii so vstupom do vrstvy pre vyjadrenie náhodnej akcie robota.

Riadok 55 vyvolanie generovania náhodných premenných pred prvotným použitím vrstvy.

Riadok 57-58 hlavná metóda aktivácie vrstvy, ktorá maticovým násobením vráti náhodné premenné závislé na vstupe do vrstvy.

Riadok 74-75 funkcia, ktorá škáluje trénované parametre do rozsahu $(0, \infty)$.

Riadok 77-81 generátor náhodnej premennej s nulovým priemerom a smerodajnou odchýlkou vyjadrenou trénovanými premennými. Po vygenerovaní sú parametre priradené do netrénovanej premennej.

Obrázok 40 popisuje súbor rl-toolkit/networks/callbacks/agent.py definujúci volania na začiatku a konci celého procesu učenia a konci každej epizódy učenia. Cieľom je komunikácia s databázovým serverom a nahrávanie aktualizovaných parametrov modelu ako aj inkrementovanie počítadla krokov učenia. Taktiež má za úlohu informovať agentov o konci učiaceho procesu a zastaviť ich činnosť v simulácii.

```python
class AgentCallback(Callback):
    def __init__(self, db_server: str):
        super(AgentCallback, self).__init__()

        self._db_server = db_server

    def on_train_begin(self, logs=None):
        # Variables
        self._train_step = tf.Variable(
            0,
            trainable=False,
            dtype=tf.uint64,
            aggregation=tf.VariableAggregation.ONLY_FIRST_REPLICA,
            shape=(),
        )
        self._stop_agents = tf.Variable(
            False,
            trainable=False,
            dtype=tf.bool,
            aggregation=tf.VariableAggregation.ONLY_FIRST_REPLICA,
            shape=(),
        )

        # Table for storing variables
        self._variable_container = VariableContainer(
            db_server=self._db_server,
            table="variable",
            variables={
                "train_step": self._train_step,
                "stop_agents": self._stop_agents,
                "policy_variables": self.model.actor.variables,
            },
        )

        # Init variable container from DB server
        self._variable_container.update_variables()

    def on_epoch_end(self, epoch, logs=None):
        # increase the training step
        self._train_step.assign_add(1)

        # Store new actor's params
        self._variable_container.push_variables()

    def on_train_end(self, logs=None):
        # Stop the agents
        self._stop_agents.assign(True)
        self._variable_container.push_variables()
```

Obrázok 40 Definícia volaní pre agentov počas učenia *learner*

Riadok 13 metóda volaná raz na začiatku celého procesu učenia *learner*.

Riadok 15-21 definícia premennej počítajúcej učiace epizódy, ktorá informuje všetkých agentov o aktuálnom stave učenia.

Riadok 22-28 definícia premennej informujúcej agentov o konci procesu učenia, ak nadobudne hodnoty *True*, agenti automaticky zastavia svoju činnosť v simulácii a ukončia program.

Riadok 31-39 definícia kontajneru s premennými, ku ktorým majú prístup všetci agenti a načítavajú z neho aktuálne parametre *actor* časti modelu ako aj informácie o stave učenia.

Riadok 42 vyvolanie aktualizácie parametrov v *learner* na začiatku celého procesu učenia pre prvotné synchronizovanie *learner* s databázovým serverom.

Riadok 44 metóda volaná po každej epizóde učenia modelu.

Riadok 46 inkrementácia kroku učenia o 1 krok.

Riadok 49 vyvolanie zápisu aktuálnych premenných zo *learner* do databázového servera.

Riadok 51 metóda volaná raz na konci celého procesu učenia *learner*.

Riadok 53 nastavenie premennej zastavujúcej činnosť agentov na *True*, čím agenti zastavia svoju činnosť.

Riadok 54 vyvolanie zápisu aktuálnych premenných zo *learner* do databázového servera.

Obrázky 41 a 42 popisujú súbor rl-toolkit/networks/models/actor.py definujúci vrstvy použité modelom *actor* pri transformácii stavového priestoru na akčný priestor a ich aktivácie.

```
28      def __init__(
29          self,
30          units: list,
31          n_outputs: int,
32          clip_mean_min: float,
33          clip_mean_max: float,
34          init_noise: float,
35          **kwargs
36      ):
37          super(Actor, self).__init__(**kwargs)
38
39          # 1. layer
40          self.fc_0 = Dense(
41              units=units[0],
42              activation="relu",
43              kernel_initializer=uniform_initializer,
44          )
45
46          # 2. layer     TODO(markub3327): Transformer
47          self.fc_1 = Dense(
48              units=units[1],
49              activation="relu",
50              kernel_initializer=uniform_initializer,
51          )
52
53          # Deterministicke akcie
54          self.mean = Dense(
55              n_outputs,
56              activation="linear",
57              kernel_initializer=uniform_initializer,
58              name="mean",
59          )
60          self.clip_mean = Lambda(
61              lambda x: tf.clip_by_value(x, clip_mean_min, clip_mean_max),
62              name="clip_mean",
63          )
64
65          # Stochasticke akcie
66          self.noise = MultivariateGaussianNoise(
67              n_outputs,
68              kernel_initializer=Constant(value=init_noise),
69              name="noise",
70          )
71
72          # Vystupna prenosova funkcia
73          self.bijector = tfp.bijectors.Tanh()
```

Obrázok 41 Definícia vrstiev *actor* časti modelu

Riadok 40-44 definícia prvej vrstvy modelu s aktivačnou funkciou ReLU, na ktorú je priamo privedený stavový priestor.

Riadok 47-51 definícia druhej vrstvy modelu s aktivačnou funkciou ReLU, ktorá predstavuje skrytú vrstvu modelu.

Riadok 54-59 definícia vrstvy vyjadrujúcej priemery pre náhodné premenné generované z Gaussovho šumu. Počet neurónov v tejto vrstve je rovný počtu kĺbov robota a jej aktivačná funkcia je lineárna pre výstupný rozsah (-∞, ∞).

Riadok 60-63 definícia špeciálnej vrstvy, ktorá škáluje priemer do zvoleného rozsahu hodnôt pre stabilizáciu procesu učenia.

Riadok 66-70 definícia špeciálnej vrstvy, ktorá generuje náhodné premenné na základe stavového priestoru, ktoré sú pripočítavané k deterministickým akciám agenta.

Riadok 73 definícia funkcia, ktorá usmerní náhodné premenné do rozsahu (-1, 1), čo predstavuje pracovný rozsah kĺbov robota.

```
75      def reset_noise(self):
76          self.noise.sample_weights()
77
78      def call(self, inputs, with_log_prob=True, deterministic=None):
79          # 1. layer
80          x = self.fc_0(inputs)
81
82          # 2. layer
83          latent_sde = self.fc_1(x)
84
85          # Output layer
86          mean = self.mean(latent_sde)
87          mean = self.clip_mean(mean)
88
89          if deterministic:
90              action = self.bijector.forward(mean)
91              log_prob = None
92          else:
93              noise = self.noise(latent_sde)
94              action = self.bijector.forward(mean + noise)
95
96              if with_log_prob:
97                  variance = tf.matmul(tf.square(latent_sde), tf.square(self.noise.scale))
98                  pi_distribution = tfp.distributions.TransformedDistribution(
99                      distribution=tfp.distributions.MultivariateNormalDiag(
100                         loc=mean, scale_diag=tf.sqrt(variance + 1e-6)
101                     ),
102                     bijector=self.bijector,
103                 )
104                 log_prob = pi_distribution.log_prob(action)[..., tf.newaxis]
105             else:
106                 log_prob = None
107
108         return [action, log_prob]
```

Obrázok 42 Volanie akcie agenta na základe stavového priestoru

Riadok 75-76 definícia metódy pre vygenerovanie nových náhodných premenných.

Riadok 78 hlavná metóda aktivácie modelu podľa daného vstupu.

Riadok 80 aktivácia prvej vrstvy modelu podľa stavového priestoru.

Riadok 83 aktivácia druhej vrstvy modelu podľa výstupu z predchádzajúcej vrstvy.

Riadok 86 aktivácia výstupnej vrstvy modelu podľa latentného priestoru modelu.

Riadok 87 usmernenie priemeru podľa zadaných hraničných hodnôt.

Riadok 89-91 predikcia deterministickej akcie.

Riadok 93 predikcia náhodnej premennej na základe latentného priestoru modelu.

Riadok 94 predikcia akcie na základe deterministickej a stochastickej zložky.

Riadok 97-104 výpočet logaritmu pravdepodobnosti vygenerovanej akcie, ktorý je neskôr použitý pri procese učenia. Vychádza z aktuálneho predikovaného priemeru a smerodajnej odchýlky vyjadrenej kombináciou aktuálneho latentného priestoru a parametru vlastnej vrstvy šumu, ktorý je tiež učený spolu s modelom.

Riadok 106 zápis prázdneho objektu *None* v prípade, že nie je potrebné počítať logaritmus pravdepodobnosti, čo je využité v prípade agentov.

Riadok 108 návrat dvojice akcia a logaritmus pravdepodobnosti z modelu.

Obrázok 43 popisuje súbor rl-toolkit/networks/models/critic.py definujúci vrstvy *critic* časti modelu a ich aktivácie.

```
22      def __init__(self, units: list, n_quantiles: int, **kwargs):
23          super(Critic, self).__init__(**kwargs)
24
25          # 1. layer
26          self.fc_0 = Dense(
27              units=units[0],
28              activation="relu",
29              kernel_initializer=uniform_initializer,
30          )
31
32          # 2. layer     TODO(markub3327): Transformer
33          self.fc_1 = Dense(
34              units=units[1],
35              kernel_initializer=uniform_initializer,
36          )
37          self.fc_2 = Dense(
38              units=units[1],
39              kernel_initializer=uniform_initializer,
40          )
41          self.add_0 = Add()
42          self.activ_0 = Activation("relu")
43
44          # Output layer
45          self.quantiles = Dense(
46              n_quantiles,
47              activation="linear",
48              kernel_initializer=uniform_initializer,
49              name="quantiles",
50          )
51
52      def call(self, inputs):
53          # 1. layer
54          state = self.fc_0(inputs[0])
55
56          # 2. layer
57          state = self.fc_1(state)
58          action = self.fc_2(inputs[1])
59          x = self.add_0([state, action])
60          x = self.activ_0(x)
61
62          # Output layer
63          quantiles = self.quantiles(x)
64          return quantiles
```

Obrázok 43 Definícia *critic* časti modelu

Riadok 26-30 definícia prvej vrstvy modelu s aktivačnou funkciou ReLU, na ktorú je priamo privedený stavový priestor.

Riadok 33-36 definícia druhej vrstvy modelu bez aktivačnej funkcie, ktorá predstavuje skrytú vrstvu modelu.

Riadok 37-40 definícia prvej vrstvy modelu bez aktivačnej funkcie, na ktorú je priamo privedený akčný priestor.

Riadok 41 definícia vrstvy zlučovania stavových príznakov s akčnými príznakmi pomocou operácie sčítavania.

Riadok 42 definícia aktivačnej vrstvy ReLU aplikovanej po zlučovacej vrstve.

Riadok 45-50 definícia výstupnej vrstvy, ktorej počet neurónov je zhodný s počtom predikovaných kvantilov a jej aktivačná funkcia je lineárna pre výstupný rozsah (-∞, ∞).

Riadok 52 hlavná metóda aktivácie modelu podľa daného vstupu.

Riadok 54 aktivácia prvej vrstvy modelu podľa stavového priestoru.

Riadok 57 aktivácia druhej vrstvy modelu podľa výstupu z predchádzajúcej vrstvy.

Riadok 58 aktivácia prvej vrstvy modelu podľa akčného priestoru.

Riadok 59 aktivácia zlučovacej vrstvy na základe dvojice stav-akcia.

Riadok 60 aktivačná vrstva ReLU aplikovaná po zlúčení dvojice stav-akcia.

Riadok 63 aktivácia výstupnej vrstvy predikujúcej kvantily.

Riadok 64 návrat kvantilov z modelu.

Príloha C: Obsah CD

Priložené CD obsahuje teoretickú časť pozostávajúcu z docx a pdf súboru, ktorá sa nachádza v adresári teoreticka_cast a praktickú časť v adresári prakticka_cast. V adresári prakticka_cast sa nachádza aj používateľská príručka v súbore pouzivatelska_prirucka.pdf potrebná k dosiahnutiu výsledkov popisujúcich v praktickej časti práce. Súčasťou praktickej časti je aj video ukážka jedného zo simulovaných robotov.

Štruktúra umiestnenia súborov na CD je nasledovná:

- Obsah_CD.docx
- Obsah_CD.pdf
- [prakticka_cast]
  - pouzivatelska_prirucka.docx
  - pouzivatelska_prirucka.pdf
  - rl-toolkit.zip
  - ukazka.mp4
- [teoreticka_cast]
  - Kubovcik_Martin_RL_Toolkit.docx
  - Kubovcik_Martin_RL_Toolkit.pdf