VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE POSILOVANÉHO UČENÍ PŘI ŘÍZENÍ MODELU VOZIDLA

VEHICLE CONTROL VIA REINFORCEMENT LEARNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR MASLOWSKI**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN ŠŮSTEK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Maslowski Petr**
Program: Informační technologie
Název: **Aplikace posilovaného učení při řízení modelu vozidla**
Vehicle Control via Reinforcement Learning
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte metody posilovaného učení včetně těch, které využívají neuronové sítě.
2. Navrhněte využití posilovaného učení při řízení libovolně vybraného vozidla v libovolném dostupném simulátoru.
3. Návrh z předchozího bodu implementujte.
4. Proveďte experimenty. Zhodnoťte dosažené výsledky a porovnejte je s výsledky existujících řešení.

Literatura:

- A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.
- V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", 2013.
- A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator", 2017.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Šůstek Martin, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce se zabývá vytvořením autonomního agenta pro řízení modelu vozidla. Rozhodování agenta je řízeno pomocí posilovaného učení (*reinforcement learning*) s využitím neuronových sítí. Agent získává snímky z přední kamery vozidla a na základě jejich interpretace vybírá vhodné akce pro řízení vozidla. V rámci práce jsem navrhl několik funkcí odměn a s vytvořenými modely jsem experimentoval úpravou hyperparametrů. Výsledný agent pak simuluje řízení vozidla na silnici. Výsledek této práce ukazuje možný přístup k ovládání autonomního vozidla, které se učí řídit metodou strojového učení v simulátoru CARLA.

Abstract

The goal of this thesis is a creation of an autonomous agent that can control a vehicle. The agent utilizes reinforcement learning that uses neural networks. The agent interprets images from the front vehicle camera and selects appropriate actions to control the vehicle. I designed and created reward functions and then experimented with hyperparameters setup. Trained agent simulate driving on the road. The result of this thesis shows a possible approach to control an autonomous vehicle agent using machine learning method in CARLA simulator

Klíčová slova

Posilované učení, strojové učení, neuronové sítě, konvoluční neuronové sítě, DQN, autonomní řízení vozidla, simulace řízení vozidla, simulátor CARLA

Keywords

Reinforcement learning, machine learning, neural networks, convolution neural networks, DQN, autonomous vehicle control, simulated driving, CARLA simulator

Citace

MASLOWSKI, Petr. *Aplikace posilovaného učení při řízení modelu vozidla.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Šůstek

Aplikace posilovaného učení při řízení modelu vozidla

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Šůstka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Maslowski
16. května 2019

Poděkování

Děkuji svému vedoucímu Ing. Martinu Šůstkovi za cenné rady a pomoc při tvorbě této práce a Ing. Michalu Hradišovi, PhD. za uvedení do problematiky neuronových sítí.

Obsah

Kapitola 1

Úvod

V současné době rychle se vyvíjejících technologií se také velmi rychle rozvíjí odvětví umělé inteligence, a to především díky dostupnému výpočetnímu výkonu počítačů. Umělá inteligence se používá například při rozpoznávání psaného či mluveného textu, kdy se nejdříve natrénuje klasifikátor na předem známém vzorku dat, aby pak dokázal klasifikovat i data neznámá (a rozpoznat tak text, či určit mluvčího). Další možné využití umělé inteligence je v dopravě. Částečně autonomní vozidla již existují a pro jejich učení se využívá podobný princip. Tito autonomní agenti se mohou učit ze záznamů jízdy skutečných řidičů, aby pak dokázali toto chování napodobit a vozidlo řídit samostatně.

Dalším způsobem, jak tyto agenty učit řídit, je metoda posilovaného učení (*reinforcement learning*), kterou se v této práci zabývám. Tato metoda k učení nepotřebuje předem nasbíraná data, a proto může být v některých případech vhodnější. Autonomní agent se učí pouze z akcí, které sám vykonal a za které je patřičně ohodnocen. Učí se tedy převážně ze svých vlastních chyb, což přináší bohužel i nevýhody. V reálném světě by takové učení agenta pro řízení vozidla bylo příliš nákladné a nebezpečné, proto se tyto metody testují v simulátorech. Také já v této práci používám simulátor pro samotné učení a poté i k ověření schopností natrénovaného agenta. Konkrétně používám metodu posilovaného učení ve spojení s konvolučními a plně propojenými neuronovými sítěmi.

Cílem této práce je vytvořit agenta, který bude schopen řídit vozidlo na silnici na základě informací o současném stavu. Ten je získáván pouze ve formě snímků z přední kamery vozidla.

Práce je členěna do sedmi kapitol. V úvodu je stručně uvedena současná situace a motivace této práce. Druhá kapitola vysvětluje pojem *strojové učení* a jednotlivé metody a algoritmy, které do této oblasti spadají – obzvláště pak posilované učení a neuronové sítě. Třetí kapitola diskutuje konkrétní postupy při řešení problému autonomního agenta pro řízení vozidla a popisuje implementaci navrženého řešení v simulátoru CARLA a frameworku *Keras*. Čtvrtá kapitola popisuje samotné učení agenta a interpretuje pozorované výsledky natrénovaného modelu při experimentech. V závěru je zhodnocen výsledek této práce a je nastíněn další možný postup vycházející z této práce.

Kapitola 2

Posilované učení

Posilované učení (*reinforcement learning*) je jedna z kategorií strojového učení. V této kapitole je krátce popsáno základní rozdělení strojového učení a zvláště pak posilované učení, které ve své práci používám.

2.1 Strojové učení

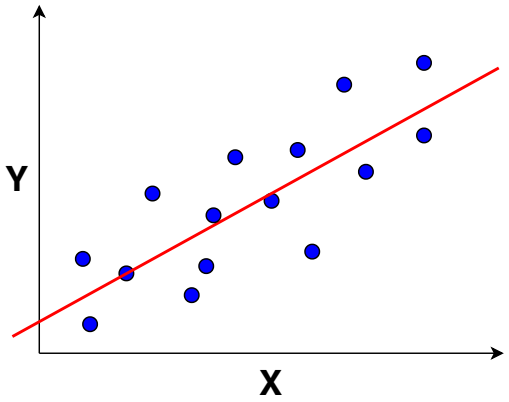
Strojové učení je nejvýznamnější oblastí umělé inteligence. Podle M. L. Minského [19] můžeme umělou inteligenci definovat takto: „Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který – kdyby ho dělal člověk – bychom považovali za projev jeho inteligence.“ Strojové učení se pak zabývá algoritmy a matematickými modely, díky kterým se stroj „učí“, tedy že se zdokonaluje na základě zkušeností. Jedno z možných rozdělení strojového učení (podle [21]) je vyobrazeno na obrázku 2.1.

Obrázek 2.1: Jedno z možných rozdělení kategorií strojového učení podle způsobu trénování na posilované učení (*reinforcement learning*), učení s učitelem (*supervised learning*), učení bez učitele (*unsupervised learning*) a kombinace předchozích dvou (*semi-supervised learning*).

Učení s učitelem

Jednou z kategorií strojového učení je učení s učitelem (*supervised learning*) [31, 21]. Algoritmy tohoto typu využívají pro své učení trénovací data (*training dataset*, což je množina dvojic, která obsahuje vstupní data a výstupní (očekávaná) data. Za pomocí algoritmu bychom chtěli produkovat hodnoty, které odpovídají očekávaným výstupům. Doufáme, že transformace vstupů na výstupy bude dostatečně obecná, tak aby se při vstupu zcela nových dat správně vyprodukoval odpovídající výstup. Do této kategorie spadají algoritmy pro klasifikaci a regresi. Při klasifikaci zařazuje vstupní data do n skupin. Příkladem může být binární klasifikátor fotografií psů a koček, kdy algoritmus určuje, jaké z těchto zvířat se nachází na fotografii. Regrese se používá, pokud predikujeme na výstupu spojité hodnoty, například odhad venkovní teploty v závislosti na denní době. Příklad regrese je zobrazen na obrázku 2.2.

Pro ověření spolehlivosti natrénovaného systému se používají data validační. Tato data by měla být disjunktní s daty trénovacími, aby se ověřila skutečná spolehlivost natrénovaného systému.



Obrázek 2.2: Příklad lineární regrese. Modré tečky znázorňují konkrétní data a červená přímka regresi (výsledek algoritmu učení). Vstupním datům X je tedy přiřazena hodnota Y podle takto natrénovaného modelu.

Při učení za pomoci algoritmu může dojít k přetrénování (*overfitting*) [9], což znamená, že se algoritmus naučí přesně rozpoznat trénovací data, na kterých dosahuje výborných výsledků. Pokud jsou ale vstupní data neznámá (dosud neviděná), tak se na výstupu objeví špatná hodnota, neboť transformace vstupů na výstupy nezobecňuje dostatečně dobře. Na obrázku 2.3 je zobrazen názorný příklad přeučení. Řešením tohoto problému může být zvětšení trénovacího datasetu, snížení komplexity systému či předčasné zastavení trénování.

Učení bez učitele

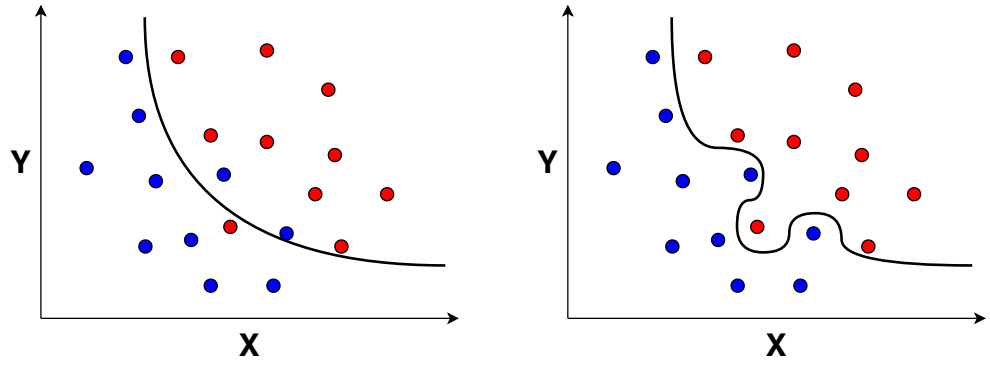
Při používání algoritmu z kategorie učení bez učitele (*unsupervised learning*) na rozdíl od učení s učitelem nemusíme znát při trénování výstupní data [21]. Místo toho hledáme s pomocí učícího algoritmu skrytou strukturu v neoznačených vstupních datech a na jejím základě je například rozdělujeme do shluků, tzv. *clusterů*.

Dalšími variantami učení bez učitele jsou například asociační analýza, která hledá pravidla spojující podobné atributy dat ve zkoumaném datasetu, nebo detekce anomálií, kdy

za pomocí algoritmu hledáme nezvyklá data v datasetu. Tento přístup se používá například při behaviorální analýze možných útočníků v počítačové síti, kdy se algoritmus snaží rozpoznat škodlivý tok od běžného síťového provozu [15].

Kombinace učení s učitelem a bez učitele

Další kategorie strojového učení vznikla kombinací dvou předchozích algoritmů (*semi-supervised learning*) [21]. Jak už název napovídá, tak kombinuje tyto dvě kategorie a učí se jak na datech anotovaných, tak i na datech neanotovaných. Toto se hodí v případě, že je náročné předpřipravit všechna trénovací data a přiřadit jim jejich očekávané výstupy (a tím je anotovat), tudíž se anotuje pouze část vstupních dat a zbytek zůstane neanotovaný. Tento algoritmus však dokáže zužitkovat i malou část ohodnocených dat a díky tomu může podávat přesnější výsledky než kdyby se použila pouze data neohodnocená.



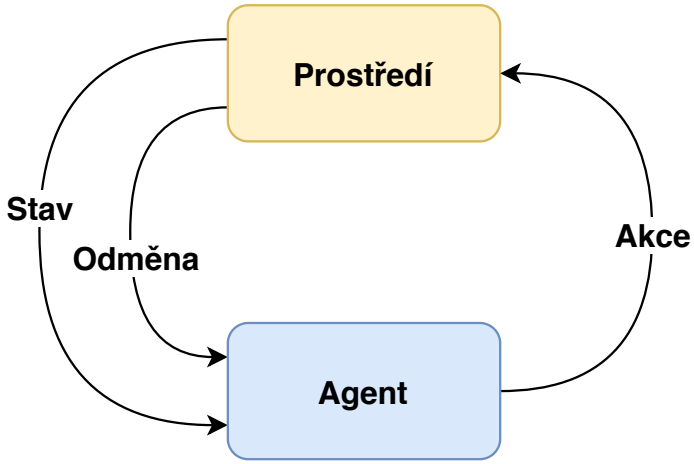
Obrázek 2.3: Modré a červené tečky znázorňují konkrétní data a černá čára, která je rozděluje, je výsledek algoritmu učení. Vlevo je očekávané rozdělení odpovídající dvěma typům dat. Vpravo je zobrazen příklad přeučení na konkrétním datasetu.

2.2 Algoritmus posilovaného učení

Další kategorií strojového učení je posilované učení. Teorie, na níž je postavený algoritmus posilovaného učení, vychází z lidského chování a ve své knize ji popisuje R. S. Sutton [29]. Jedinec se učí na základě zpětné vazby, kterou získává, když provádí různé úkony. Zpětná vazba může být jak pozitivní, tak negativní, a jedinec si zapamatuje, za které činy je odměněn a za které potrestán. V posilovaném učení se do role učícího se jedince dostává agent. Ten je v interakci s prostředím, v němž se nachází, a provádí akce, za které získává kladnou, či zápornou odměnu. Tímto způsobem se agent učí. Podrobnější popis tohoto děje je vysvětlen na obrázku 2.4.

Na rozdíl od učení s učitelem se algoritmus posilovaného učení netrénuje na datasetu vstupních a výstupních dat. Místo toho prozkoumává prostředí a hledá takové akce, které maximalizují zisk odměn. Agent navíc nevolí jen akce s ohledem na maximální okamžitou odměnou, ale i s ohledem na všechny možné odměny budoucí. Odměna však nemusí být vždy obdržena okamžitě, někdy může být získána až po mnoha krocích. Tímto procesem se agent učí strategii π (*policy*), která mapuje stav s na akci a, která se má v daném stavu s vykonat:

$$\pi(s) \rightarrow a \tag{2.1}$$



Obrázek 2.4: Zobrazení interakce mezi agentem a prostředím. Prostředí je reprezentováno stavem, jež slouží jako vstupní data pro agenta. Na základě pozorování tohoto stavu se agent rozhodne vykonat určitou akci, a tím vyvolá změnu prostředí. Tato změna je pak reprezentována následujícím stavem a okamžitou odměnou. Tento proces se provádí opakovaně a agent se tak učí optimálnímu chování v prostředí.

Markovův rozhodovací proces

Prostředí, v němž se agent nachází, je typicky formulováno jako Markovův rozhodovací proces (dále jen MDP, z anglického *Markov decision process*) [3], což můžeme definovat jako pětici (S, A, P, R, γ), kde:

- S je konečná množina všech stavů
- A je konečná množina všech akcí
- $P(s' \mid s, a)$ je funkce přechodu, značí pravděpodobnost přechodu ze stavu s do stavu s' při provedení akce a
- $R(s, a, s')$ je funkce odměny, značí okamžitou odměnu dosaženou přechodem ze stavu s akcí a do stavu s'
- γ je diskontní faktor (z anglického *discount factor*), kde $\gamma \in \langle 0, 1 \rangle$

Pří rozhodování MDP záleží pouze na aktuálním stavu, nikoli na stavech předchozích. Toto se označuje jako Markovská vlastnost. Cílem MDP je nalézt optimální strategii π^*

$$\pi^*(s) \to a \tag{2.2}$$

tak, aby se našla nejlepší možná strategie volby akcí, která by maximalizovala očekávanou odměnu. Ta se vypočítá sečtením všech budoucích odměn s ohledem na to, jak daleko v budoucnosti se nachází. K tomu slouží diskontní faktor. Pokud $\gamma = 0$, pak se bere v úvahu pouze odměna v daném kroku a budoucí odměny se neuvažují. Naopak pokud se diskontní faktor blíží 1, tak to znamená, že velkou váhu mají i odměny získané až za mnoho kroků. Kupříkladu, pokud uvažujeme konstantní odměnu o hodnotě 1 po dobu sto kroků a zvolíme $\gamma = 0.98$, pak suma všech budoucích očekávaných odměn je rovna 43.37. Pokud diskontní faktor jen nepatrně snížíme na $\gamma = 0.95$, dostaneme očekávanou odměnu 19.88, tedy více

než dvojnásobně menší. To znamená, že nás budoucí odměny zajímají mnohem méně než v případě většího diskontního faktoru.

Obecně se diskontovaná očekávaná odměna v čase t počítá jako [29]:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

Pro ohodnocení jednotlivých stavů, v nichž se může agent nacházet, se pak používá V funkce (*state-value function*), definovaná jako:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\big[G_t \mid s_t = s\big], \text{ pro všechny } s \in S \tag{2.4}$$

kde $\mathbb{E}_{\pi}[\cdot]$ značí očekávanou hodnotu odměny (z anglického *expected value*) pro všechny počáteční stavy $s \in S$, pokud se výběr akcí řídí podle strategie π. Obdobně jako V funkce hodnotí stavy, Q funkce hodnotí akce. Q funkce (*action-value function*) vyjadřuje očekávanou hodnotu odměny při volbě akce a ve stavu s podle strategie π.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\big[G_t \mid s_t = s, a_t = a\big] \tag{2.5}$$

Optimální funkce V^* je nejlépe ohodnocenou funkcí V^{π}, kterou můžeme získat, pokud se budeme řídit nejlepší možnou strategií π.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.6}$$

Podobně je pak optimální funkce Q^* nejlépe ohodnocenou funkcí Q^{π}, kterou můžeme získat, pokud se budeme řídit nejlepší možnou strategií π.

$$Q^*(s,\, a) = \max_{\pi} Q^{\pi}(s, a) \tag{2.7}$$

Tyto optimální funkce vyjadřují nejlepší možné počínání v MDP. Pokud tedy známe optimální funkci, pak můžeme považovat MDP za vyřešený. Cílem posilovaného učení je nalézt takovou funkci.

Q-učení

Jedním z algoritmů posilovaného učení je Q-učení, které je založeno na Q funkci (rovnice 2.5). V tomto algoritmu vybírá strategie π akci a podle vztahu

$$\pi(s) = \arg\max_{a \in A(s)} Q(s, a) \tag{2.8}$$

a výstupem funkce $Q(s, a)$ pro konkrétní akce a v jednotlivých stavech s je ohodnocení těchto akcí – *Q-hodnota*. Ta se obvykle ukládá do tabulky, kde jsou zaznamenány *Q-hodnoty* pro všechny akce ve všech stavech.

U Q-učení nepotřebujeme pro učení znát model prostředí v němž se agent nachází. To je jeho velkou výhodou, neboť dokáže porovnat očekávanou odměnu dostupných akcí, aniž bychom museli znát celý model prostředí.

Na začátku Q-učení se tabulka *Q-hodnot* inicializuje (nejčastěji na hodnotu nula) a poté se jednotlivé hodnoty aktualizují vztahem

$$Q^{new}(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha\Big(R(s) + \gamma \max_{a'} Q(s', a')\Big) \tag{2.9}$$

vycházejícím z Bellmanovy rovnice [32], kde výsledná hodnota $Q^{new}(s,a)$ je nová *Q-hodnota* vycházející ze staré hodnoty $Q(s,a)$ pro akci a vykonané ve stavu s a kde $\max_{a'} Q(s',a')$ označuje maximální ohodnocení ze všech akcí a' proveditelných ve stavu s'. Stav s' je nový stav, do kterého se dostaneme po provedení akce a ve stavu s. Znak $\alpha \in (0,1\rangle$ zde představuje koeficient učení (*learning rate*), který se může v průběhu učení snižovat. Tento koeficient určuje kolik z rozdílu předchozí *Q-hodnoty* a nové diskontované maximální *Q-hodnoty* bude započítán. Celý popis Q-učení je popsán v algoritmu 1.

Algoritmus 1 Q-učení (převzato z [25])

1: Inicializuj hodnoty $Q(s,a)$
2: Nastav α a γ, $(0 < \alpha \leq 1;\ 0 \leq \gamma \leq 1)$
3: **repeat** pro každou epizodu:
4: Získej počáteční stav s
5: **repeat**
6: Vyber a proveď akci a
7: $s' \leftarrow$ Nový stav, $r \leftarrow$ Odměna
8: $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a')\Big)$
9: $s \leftarrow s'$
10: **until** s je koncový stav

Explorace a exploitace

Prozkoumávání (*Exploration*) a využívání (*Exploitation*) představují dva důležité prvky v posilovaném učení [2]. Poměr mezi těmito aspekty určuje, jak se bude agent při učení chovat. Pokud se bude řídit jen využíváním předchozích znalostí a vykonávat pouze akce s nejlepším ohodnocením, může se naučit sub-optimální strategii. To znamená, že se bude chovat optimálně vzhledem k prostředí, jež zná, ale nikdy nevykoná hůře ohodnocenou akci (ani nevyzkouší akci novou), a to ani v případě, že by mohla později vést k mnohem lepší akci než všechny doposud pozorované akce. Toto se označuje jako chamtivý algoritmus (*greedy*). Naopak, pokud se agent bude řídit jen prozkoumáváním prostředí bez ohledu na ohodnocení akcí, může najít strategii optimální. Nalezení takové strategie může ovšem trvat velmi dlouho a přestože by už ji našel, tak by ji nevyužíval, neboť akce volí vždy náhodně. Je tedy nutné zvolit správný poměr mezi těmito postupy.

K tomuto může pomoci ε-chamtivý algoritmus (ε-*greedy*). V tomto případě zvolíme $\varepsilon \in \langle 0,1\rangle$ tak, že agent bude volit nejlépe ohodnocenou akci s pravděpodobností $(1-\varepsilon)$ a akci zcela náhodnou s pravděpodobností ε. Tímto se docílí dostatečného využití doposud známé optimální strategie, přičemž se stále budou prozkoumávat dosud neobjevené akce. Hodnota ε se navíc může v průběhu učení měnit. Na začátku učení, kdy je poměr prozkoumaného prostředí malý, je žádoucí, aby se vykonávaly akce náhodné a rozšiřoval se tak agentův „obzor“. Toho lze dosáhnout například nastavením hodnoty ε na hodnotu 1. Postupem učení se pak tato hodnota snižuje s tím, že agent už zná větší část prostředí a může volit akce vzhledem k naučené strategii.

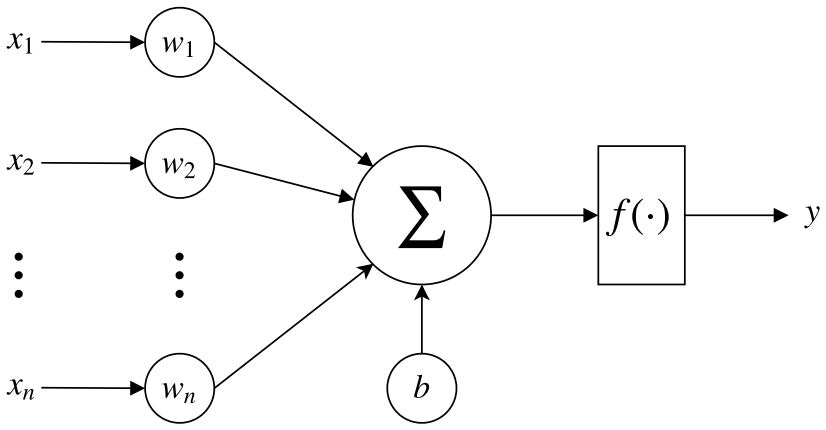
Existují i další algoritmy, které se snaží najít optimální poměr mezi prozkoumáváním a využíváním. Příklady těchto algoritmů lze nalézt v [1, 30].

2.3 Neuronové sítě

Jedním z výpočetních modelů strojového učení je umělá neuronová síť (*artificial neural network*) [28]. Vznik tohoto modelu je inspirován fungováním lidského mozku, který obsahuje neurony. Mezi těmito neurony jsou spojení – synapse, prostřednictvím nichž se může signál přenášet z jednoho neuronu na druhý. Neuron je schopný tyto signály přijímat, dále je zpracovávat a rozesílat okolním neuronům. Umělý neuron má stejně jako neuron biologický několik vstupů a jeden výstup. Na základě vstupu (vstupního vektoru) získá výstup (jedná se tedy o funkci). Model neuronu vycházející z [16] lze zapsat jako:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.10}$$

Funkce $f(\cdot)$ se nazývá aktivační (nebo také přenosová) funkce neuronu, podrobnější vysvětlení a příklady těchto funkcí jsou v sekci Aktivační funkce neuronu. Hodnota n označuje dimenzionalitu vstupního vektoru $\vec{x} = (x_1, \ldots x_n)$. Současně x_i je i-tý prvek tohoto vektoru a w_i je i-tá váha neuronu. Hodnota b značí práh (*bias*). Ilustrace fungování neuronu je na obrázku 2.5.

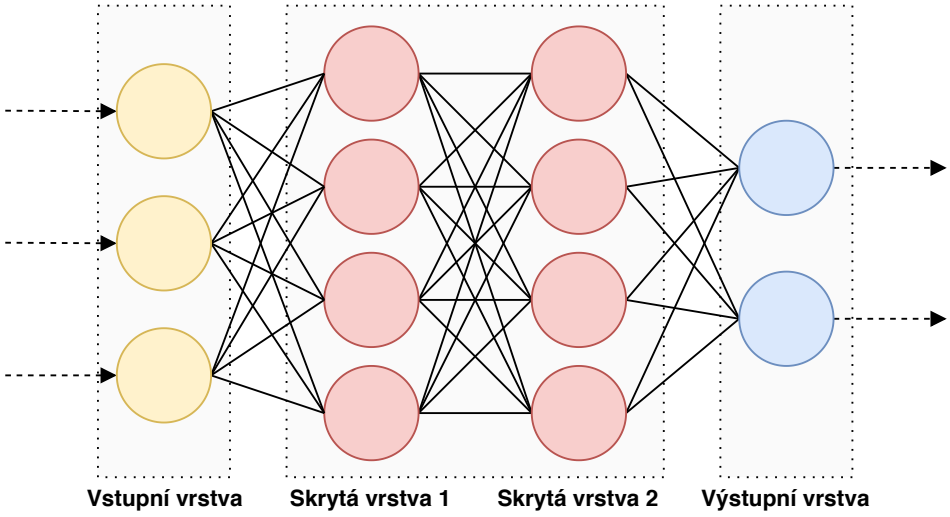


Obrázek 2.5: Zobrazení umělého neuronu. Výstup neuronu y se získá z aktivační funkce, jejíž vstup vznikne vynásobením jednotlivých vstupů neuronu x_i s jednotlivými váhami w_i a přičtením k hodnotě prahu b.

Propojením vstupů a výstupů více neuronů vzniká neuronová síť. Ta bývá rozdělena do tří vrstev [28]. Vrstvy vstupní, skryté a výstupní. Skrytých vrstev přitom může být několik za sebou (jsou ovšem i neuronové sítě bez skrytých vrstev). Příklad dopředné neuronové sítě je na obrázku 2.6. Pokud je v topologii sítě jedna či více skrytých vrstev, můžeme hovořit o hluboké neuronové síti [4].

Dopředná neuronová síť (*feedforward neural network*) je taková síť, kde výstupy jednotlivých vrstev jsou připojeny k vrstvám následujícím [17]. Signál se tedy šíří sítí směrem ze vstupu na výstup a topologie neuronové sítě je beze smyček.

Další typ neuronové sítě je síť rekurentní (*recurrent neural network*) [18]. Oproti dopředné síti se signál šíří i z výstupu na vstup a v topologii neuronové sítě tak vznikají smyčky. Zatímco dopředné sítě jsou bezstavové, rekurentní sítě mají díky těmto smyčkám „paměť“, která jim dovoluje uchovat informaci o jejich minulých výpočtech.



Obrázek 2.6: Příklad plně propojené dopředné neuronové sítě. Tato síť se skládá ze tří neuronů ve vstupní vrstvě, z osmi neuronů ve dvou skrytých vrstvách a dvou neuronů ve vrstvě výstupní.

Samotné učení neuronové sítě probíhá úpravou jednotlivých vah neuronů tak, aby síť generovala požadované výsledky. Správnost výsledků se ověřuje pomocí ztrátové funkce (*loss function*). Ta vyjadřuje odchylku skutečných výsledků od požadovaného chování (chybu). Během učení sítě by se měla tato odchylka snižovat. Jednou z nejběžněji používaných ztrátových funkcí je MSE (*mean squared error*) [11]. Ta se počítá následujícím způsobem:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.11}$$

Hodnota $(y_i - \hat{y}_i)^2$ vyjadřuje druhou mocninu rozdílu mezi skutečnou a požadovanou hodnotou pro i-tou hodnotu dat. Z těchto hodnot se pak počítá průměr pro všechna data. Váhy neuronů se upravují nejčastěji pomocí stochastického gradientního sestupu (*SGD*) aplikovaného na ztrátovou funkci. Do dalších vrstev se pak chyba propaguje pomocí zpětné propagace (*backpropagation*). Podrobnosti o gradientním sestupu a *SGD* lze najít například v [7].

Aktivační funkce neuronu

Aktivační funkce je obecně nelineární funkce (transformuje vnitřní potenciál neuronu na výstup). Zde jsou popsány běžně používané aktivační funkce a na obrázku 2.7 jsou zobrazeny jejich průběhy.

Funkce ReLU (*rectifier linear unit*) má obor hodnot v intervalu $\langle 0, \infty)$.

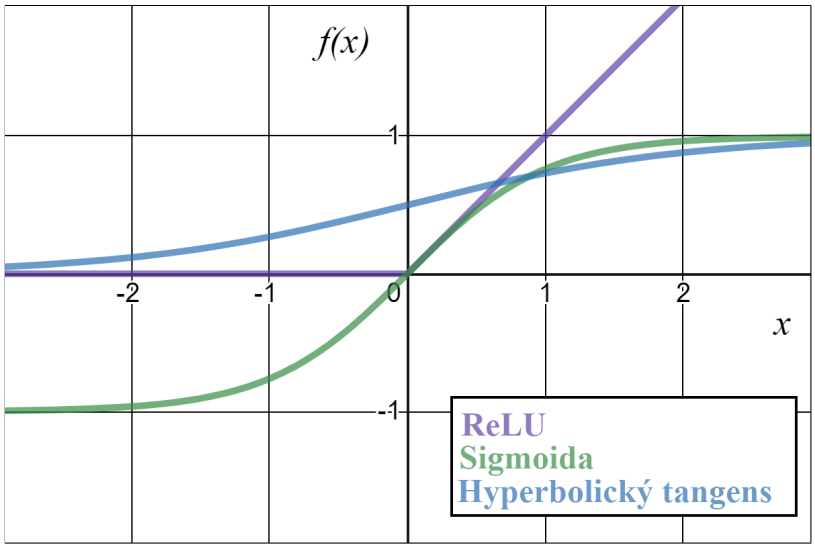
$$f(x) = \begin{cases} x & \text{pro } x > 0 \\ 0 & \text{jinak} \end{cases} \tag{2.12}$$

Sigmoida má obor hodnot v rozmezí 0 až 1.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

Hyperbolický tangens má obor hodnot v rozmezí -1 až 1.

$$f(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.14}$$



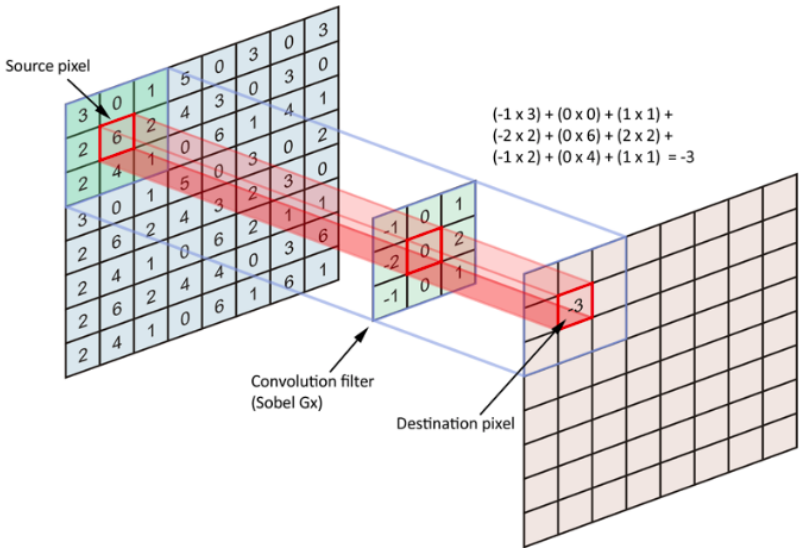
Obrázek 2.7: Zobrazení průběhu aktivačních funkcí ReLU (fialová), sigmoidy (zelená) a hyperbolického tangensu (modrá).

Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN/ConvNets) [10] jsou umělé neuronové sítě, které se často používají ke klasifikaci obrázků a rozpoznávání objektů a textů. Tyto sítě zpracovávají obrázky ve formě tenzorů (nejčastěji obrázky s kanály RGB nebo ve stupních šedi). Vstupem sítě může být tedy například obrázek o rozměrech 32×32×3 (RGB obrázek s rozlišením 32×32 pixelů). V konvolučních sítích se vyskytují tři základní typy vrstev [12]:

- Konvoluční vrstvy
- Subsamplingové vrstvy (*pooling*)
- Plně propojené vrstvy

Konvoluční vrstvy můžou být chápány jako extraktor příznaků (příznakem může být například horizontální úsečka, křivka či složitější tvary). To znamená, že z obrazu či zvukové nahrávky dokáží extrahovat důležité informace. Konvoluční vrstvy (spolu se subsamplingovými vrstvami) tímto procesem značně redukují počet vstupních parametrů. V následujícím popisu uvažujme pro zjednodušení pouze práci s obrazem. Konvoluce probíhá aplikací filtru na vstupní obraz. Jádro filtru je tvořeno maticí o rozměrech (x, y), která se přikládá na všechny pozice v obraze. Jednotlivé hodnoty pixelů se vynásobí s překrývajícími se hodnotami jádra filtru a poté se sečtou do výsledné hodnoty. Toto se provede v každém kroku, kdy se jádro filtru posouvá vstupním obrazem nad další pixely. Jádro má obvykle rozměry 3×3 a krok posunutí (*stride*) bývá roven jedné. Provedením konvoluce pak vznikají příznakové mapy (*feature maps*). Příklad konvoluce je na obrázku 2.8.

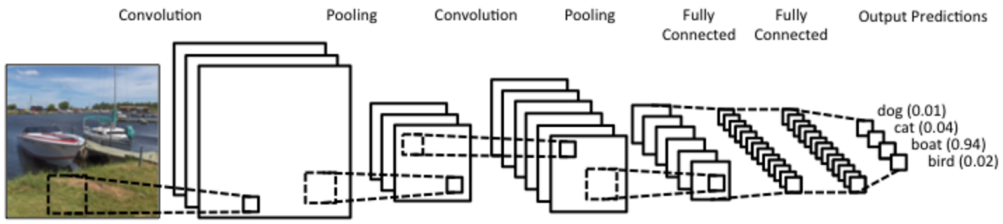


Obrázek 2.8: Příklad konvoluce. Jádro filtru je přikládáno na vstupní obraz a výsledkem je nová hodnota (převzato z [6]).

Subsamplingové vrstvy zmenšují rozměry obrazu. Obvykle vybírají maximum (*max pooling*) nebo provádí průměrování (*average pooling*) okolních pixelů, které se agregují do hodnoty jednoho pixelu. Běžně používaná maska o rozměru 2×2 s krokem 2 zmenší obraz na čtvrtinu původního rozměru (zmenšení výšky i šířky obrazu na polovinu, počet kanálů zůstává zachován). Tímto způsobem lze také částečně předcházet *overfittingu* [12].

Plně propojené vrstvy jsou standardní vrstvy jako v dopředných neuronových sítí, které generují výstupy konvoluční neuronové sítě [12].

Příklad konvoluční neuronové sítě, jež klasifikuje objekty na fotografii je znázorněn na obrázku 2.9.



Obrázek 2.9: Zobrazení konvoluční neuronové sítě, včetně tří základních vrstev – konvoluční vrstva (*convolution*), subsamplingová vrstva (*pooling*) a plně propojené vrstva (*fully connected*) (převzato z [5]).

2.4 Hluboké posilované učení

Základní algoritmus Q-učení je vhodný pouze pro řešení problému v omezeném stavovém prostoru, jelikož všechny Q-hodnoty se ukládají do tabulky. Pokud však simulujeme prostředí reálného světa, tak pozorujeme prostor s velmi rozsáhlým stavovým prostorem. Pokud bychom chtěli vypočítat Q-hodnoty pro všechny akce ve všech stavech a uložit je do tabulky tak by výsledná tabulka byla enormní. Proto vzniklo hluboké posilované učení (*deep reinforcement learning*), které Q-hodnoty aproximuje za pomocí hluboké neuronové sítě a tím pádem nevzniká tabulka Q-hodnot [29].

Hluboké Q-sítě

Hluboké Q-sítě neboli DQN (*Deep Q-Network*) je algoritmus posilovaného učení, jenž v poslední době zaznamenal velkou publicitu. To především díky výhře programu AlphaZero od společnosti DeepMind [23, 22] ve hře Go proti skutečnému hráči za pomoci hlubokých neuronových sítí a následně použití DQN pro hraní her z konzole Atari 2600, kde v některých hrách podával algoritmus lepší výsledky než člověk [20].

Algoritmus DQN aproximuje funkci $Q^*(s, a)$ pro ohodnocení akcí (rovnice 2.9) za pomoci hluboké neuronové sítě tak, aby platilo $Q(s, a \mid \theta) \approx Q^*(s, a)$, kde θ jsou váhy (neboli parametry) neuronové sítě (Q-síť), jež hodnoty aproximuje. Q-síť je následně trénována tak, aby minimalizovala ztrátovou funkci $L(\theta)$:

$$L(\theta) = \Big(R(s) + \gamma \max_{a'} Q(s', a' \mid \theta) - Q(s, a \mid \theta)\Big)^2 \tag{2.15}$$

jejíž požadovaná hodnota může být každou iteraci jiná (jedná se tedy o dynamický problém, neboť požadovaná funkce se mění v čase). Význam symbolů je stejný jako v rovnici 2.9.

Trénování sítě na základě ztrátové funkce může být prováděno například algoritmem stochastického gradientního sestupu [7].

Kapitola 3

Návrh řešení a implementace

V této kapitole je popsána implementace agenta posilovaného učení. Cílem je implementovat algoritmus učení tak, aby byl natrénovaný agent schopen řídit model vozidla v simulátoru. Agent může vozidlo řídit libovolně bez předem určené trasy. Vozidlo by však nemělo vyjet mimo silnici. Pro tento účel využívám dříve popsaný algoritmus DQN, jenž je algoritmem posilovaného učení.

3.1 Simulátor CARLA

Jako prostředí pro trénování a hodnocení agenta posilovaného učení jsem zvolil simulátor CARLA (*CAR Learn to Act*) [8]. Jedná se o open-source simulátor pro vývoj autonomního řízení. Poskytuje množství modelů a senzorů, plnou kontrolu nad simulací a několik předpřipravených map urbanistického prostředí. Funguje na bázi klient-server a podporuje také multiagentní systémy. Velkou výhodou simulátoru je aplikační rozhraní, které poskytuje všechny potřebné informace o modelu vozidla a prostředí nezbytné pro učení neuronové sítě, a současně dovoluje ovládat vozidla výstupem z neuronové sítě. Zároveň je možné spustit simulátor v synchronním módu tak, že při učení sítě se simulátor pozastaví, dokud neproběhnou všechny potřebné výpočty agenta v daném kroku. Tento simulátor je v současné době stále ve vývoji, což může způsobit i komplikace v podobě programových chyb či nestability simulátoru.

Jedny z důležitých ovládacích prvků aplikačního rozhraní pro řízení vozidla v simulátoru jsou tyto:

- Úhel natočení volantu
- Brzdový pedál
- Plynový pedál

Úhel natočení volantu je v rozmezí $\langle -1, 1 \rangle$, kde -1 je maximální natočení volantu doleva a 1 je maximální natočení doprava. Pro brzdový a plynový pedál je rozmezí hodnot v intervalu $\langle 0, 1 \rangle$, kde 0 znamená nestlačený pedál a 1 označuje maximálně stlačený pedál.

Simulátor CARLA je rozdělen na serverovou a klientskou část. Na serveru probíhá simulace a vykreslování prostředí. Klient se na server připojí a komunikuje s ním prostřednictvím zasílání požadavků (například „vytvoř model vozidla“, „aplikuj úhel natočení volantu x pro vozidlo y“). Server požadavky aplikuje a v každém kroku simulace odesílá klientské aplikaci informace o simulovaném prostředí a výstupy ze senzorů (například snímek z kamerového senzoru).

Názorné snímky ze simulace v tomto simulátoru jsou zobrazeny na obrázku 4.1.

Obrázek 3.1: Snímek ulice z pohledu třetí osoby v simulátoru CARLA za různých meteorologických podmínek (převzato z [8])

3.2 Implementační prostředí

Pro tuto práci jsem použil simulátor CARLA ve verzi 0.9.4. Implementace a testování proběhlo na operačním systému *Ubuntu*[1] verze 18.04. Programovacím jazykem byl zvolen *Python*[2] verze 2.7 (z důvodu kompatibility aplikačního rozhraní simulátoru CARLA, které je implementováno právě pro tuto verzi programovacího jazyka *Python*). Pro učení agenta a práci s neuronovou sítí jsem zvolil knihovnu *Keras*[3], která poskytuje vysoko-úrovňové aplikační rozhraní pro knihovny pracující s neuronovými sítěmi jako *TensorFlow*, *CNTK* nebo *Theano*. Z těchto knihoven jsem využil *TensorFlow*[4], jenž je stejně jako *Keras* multiplatformní a open-source.

3.3 Komunikace agenta s prostředím

Simulátor CARLA poskytuje předem připravené senzory. Kromě standardní RGB kamery je možno na vozidlo přidat kameru snímající hloubku prostoru. V simulátoru je také senzor LIDAR[5], který snímá okolí laserovým paprskem a vytváří tak 3D model prostředí, a GNSS[6], což je globální družicový polohový systém pro určování polohy.

[1] https://www.ubuntu.com/
[2] https://www.python.org/
[3] https://keras.io/
[4] https://www.tensorflow.org/
[5] LIDAR – Light Detection And Ranging
[6] GNSS – Global Navigation Satellite System

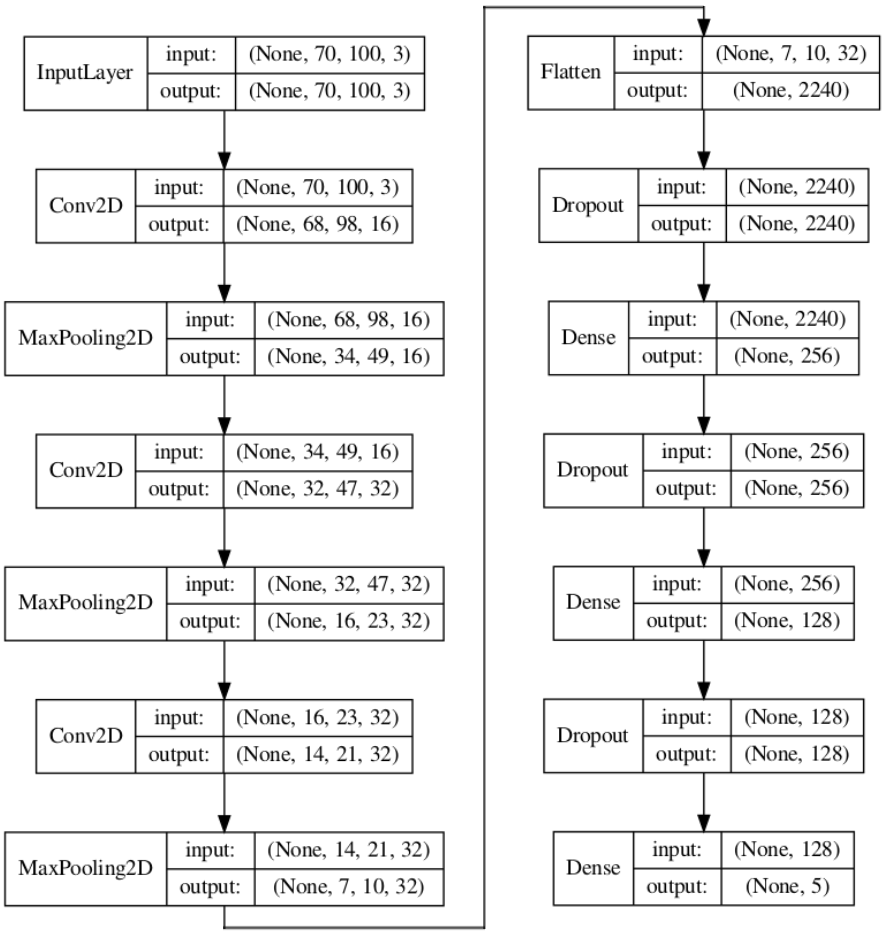
V rámci této práce jsem pro učení agenta zvolil jako vstup kameru RGB. RGB kamera je jednoduchý a běžně dostupný senzor, který je již součástí výbavy mnoha vozidel. Tuto kameru jsem v simulátoru umístil na přední stranu vozidla tak, aby snímala silnici bezprostředně před vozidlem. Pro rychlejší učení neuronové sítě je rozlišení kamery pouze 100×100 pixelů. Zachycené snímky jsou dále shora ořezány, neboť horní část obrazu není pro učení neuronové sítě důležitá. Výsledný snímek má tedy rozlišení 100×70 pixelů a tři barevné kanály (RGB). Na obrázku 3.2 je několik snímků ze simulátoru, jež slouží jako vstupy pro neuronovou síť. Vzorkovací frekvence kamery je stejná jako frekvence kroků simulace. V každém kroku simulace (a v každém kroku učení) tedy agent pozoruje právě jeden snímek kamery.

Obrázek 3.2: Snímky kamery o velikosti 100×70 pixelů nasnímané kamerou vozidla v rámci simulátoru CARLA.

Agent, který má řídit vozidlo, predikuje natočení volantu. Neuronová síť má tedy 5 výstupů, které se mapují na 5 možností natočení volantu $[-1, -0.5, 0, 0.5, 1]$, kde -1 je maximální natočení doleva a 1 je maximální natočení doprava. Plynový pedál vozidla je nastavován automaticky tak, aby udržoval předem danou rychlost.

3.4 Implementace neuronové sítě

Architekturu neuronové sítě tvoří dvě hlavní části. V první části jsou konvoluční vrstvy pro rozpoznávání obrazu a extrakci příznaků. Jádro druhé části sítě tvoří plně propojené vrstvy jejichž výstupem je předpokládané ohodnocení jednotlivých akcí. Schéma celé neuronové sítě je zobrazeno na obrázku 3.3. Tato neuronová síť vychází z implementace neuronové sítě použité týmem ze společnosti Microsoft (dostupné z repozitáře [26]).



Obrázek 3.3: Zobrazení schématu použité neuronové sítě. Jedná se o posloupnost jednotlivých vrstev u kterých je uveden typ vrstvy, očekávaný rozměr vstupu a rozměr výstupu.

Výsledná neuronová síť očekává na svém vstupu ořezaný snímek z kamery o velikosti 70×100×3 (výška×šířka×počet kanálů). Každý pixel vstupních snímků je normalizován na hodnotu $\langle 0, 1 \rangle$, z důvodu lepších výsledků extrakce příznaků. První konvoluční vrstva má 16 filtrů a konvoluční jádro o velikost 3×3. Okraje se nedoplňují nulami, takže výstup této vrstvy je zmenšený, konkrétně na 68×98. Další vrstvou je subsamplingová vrstva s maskou 2×2 a krokem 2, jež zmenšuje svůj vstup na polovinu v obou rozměrech metodou *max pooling*. Výsledkem těchto vrstev je 16 příznakových map o rozměrech 34×49. Po těchto

dvou vrstvách následují dvě dvojice konvolučních a subsamplingových vrstev (konvoluční vrstva je vždy následována vrstvou subsamplingovou). Tyto vrstvy mají totožné parametry jako vrstvy předchozí s rozdílem filtrů konvolučních vrstev, tyto konvoluční vrstvy mají obě 32 filtrů. Výstupem první z těchto dvojic je 32 příznakových map o velikosti 16×23 a výstupem druhé dvojice těchto vrstev je 32 příznakových map o velikosti 7×10.

Po těchto vrstvách se provede operace *flatten*, jež transformuje vícerozměrná data do jednoho rozměru, následovaná vrstvou zvanou *dropout* [27]. Ta funguje jako regulační metoda, která určité procento náhodně zvolených výstupů z předchozí vrstvy nastaví na hodnotu 0 a tím snižuje šanci na přeučení. V této neuronové síti jsou všechny vrstvy *dropout* nastaveny na hodnotu 0.2, což znamená, že nulují každý neuron s pravděpodobností 20%.

Další vrstvou v pořadí je plně propojená vrstva (*dense*) s 256 neurony a vrstva *dropout* následovaná další plně propojenou vrstvou se 128 neurony a poslední vrstvou *dropout*. Výstupní vrstva celé neuronové sítě je plně propojená a má 5 neuronů. Každý z těchto neuronů představuje jednu možnou akci, a to úhel natočení volantu vozidla.

Všechny vrstvy používají aktivační funkci ReLU (funkce 2.12) a jako ztrátovou funkci jsem zvolil funkci MSE (rovnice 2.11). Úpravu vah neuronové sítě provádím pomocí optimalizačního algoritmu *Adam* [13].

3.5 Funkce odměn

Pro posilované učení je důležité správně definovat funkci odměn. Měla by být definována tak, aby za akce, které jsou žádoucí (například jízda v pravém pruhu silnice) získával agent kladné odměny a za akce nežádoucí (například jízda po chodníku nebo nabourání) získal odměny záporné. Funkci odměn jsem upravoval inkrementálně podle nežádoucích akcí, které agent aktuálně prováděl.

Nejprve jsem navrhl základní funkci odměn (dále odkazovanou jako funkce odměn A), jež oceňuje agenta hodnotou 1 pokud je střed vozidla ve středu jízdního pruhu. Se zvětšující se vzdáleností vozidla od středu jízdního pruhu se pak odměna zmenšovala s druhou mocninou této vzdálenosti. Pokud vozidlo narazilo a zastavilo se, tak byl agent penalizován konstantní odměnou -25 bodů.

Do další verze funkce odměn (dále odkazovanou jako funkce odměn B) jsem přidal vliv směru vozidla vůči vozovce. Maximální odměna, kterou mohl agent získat byla nyní rovná 2 a od této hodnoty se odečítala penalizace za vzdálenost od středu vozovky a penalizace za směr natočení vozidla. Penalizace za směr natočení se vypočítala jako vzdálenost dvou bodů na jednotkové kružnici, kde souřadnice prvního bodu představovaly úhel natočení vozovky v simulátoru a souřadnice druhého bodu představovaly úhel natočení vozidla v simulátoru. Pokud tedy bylo vozidlo natočeno kolmo k silnici, tak penalizace za směr byla rovna $\sqrt{2}$ a pokud bylo vozidlo v protisměru tak se penalizace rovnala hodnotě 2. Dále byly obě penalizace podmíněné tolerancí. Pokud byla vzdálenost vozidla od středu jízdního pruhu menší než tolerance, tak nebyla uvažována žádná penalizace za špatnou pozici vozidla. Výsledná rovnice pro výpočet funkce odměn B je následující:

$$\mathrm{R} = 2 - \sqrt{(x_p - x_v)^2 + (y_p - y_v)^2} - \sqrt{(\sin r_p - \sin r_v)^2 + (\cos r_p - \cos r_v)^2} \tag{3.1}$$

Výraz $\sqrt{(x_p - x_v)^2 + (y_p - y_v)^2}$ označuje euklidovskou vzdálenost mezi středem vozidla (x_v, y_v) a nejbližším bodem ve středu jízdního pruhu (x_p, y_p). Pro výpočet rozdílů rotace slouží výraz $\sqrt{(\sin r_p - \sin r_v)^2 + (\cos r_p - \cos r_v)^2}$, kde r_p označuje úhel rotace jízdního pruhu a r_v označuje úhel rotace vozidla.

V poslední verzi funkce odměn (dále odkazovanou jako funkce odměn C) jsem se snažil bránit nežádoucímu chování agenta (změna směru vozidla o 180° před křižovatkou) zavedením „paměti“, jež si pamatovala směr vozidla v posledních x krocích (uchovávání 20 posledních kroků bylo ve většině případů dostačující). Pokud se v těchto krocích směr jízdy změnil na směr opačný, tak byl agent penalizován. Zároveň jsem přidal penalizaci za náhlé snížení rychlosti, které nastalo pokud se vozidlo dotýkalo svodidel či jiné překážky podél silnice, neboť toto chování agenta bylo taktéž nežádoucí.

3.6 Paměť zkušeností

Pro zlepšení stability systému učení jsem do algoritmu zabudoval paměť zkušeností (*experience replay*) [33] obdobně jako to udělal tým ze společnosti DeepMind ve své práci [20]. Jedná se o buffer, do kterého se ukládají jednotlivé zkušenosti zažité při simulaci. Zkušenost E je čtveřice:

$$\mathrm{E} = (s, a, r, s') \tag{3.2}$$

kde s je jeden konkrétní stav prostředí v simulaci (snímek z přední kamery), a je akce zvolená agentem ve stavu s (úhel natočení volantu), r je odměna získaná za akci a provedenou ve stavu s a s' je nový stav (následující snímek z přední kamery), do nějž se agent dostane.

Při učení agenta se v každém kroku náhodně vybírá x zkušeností na kterých se agent učí. Soubor takto vybraných zkušeností je označený jako dávka (*batch*). Během jízdy se tedy agent neučí na situacích, které v daném okamžiku zažívá, ale na různých situacích, které již zažil. Pokud by se agent učil pouze na aktuálních situacích, tak například při jízdě po rovném úseku silnice jsou stavy po sobě jdoucí velmi podobné a když by se tyto stavy agent učil ve stejném pořadí, tak by mohlo docházet k přeučení. V mé implementaci algoritmu učení má jedna dávka velikost 32 zkušeností.

3.7 Implementace algoritmu učení

Po připojení klienta na server se vytvoří neuronová síť (případně se načte již dříve vytvořená či natrénovaná síť). Celý algoritmus učení probíhá v epizodách. Jedna epizoda začíná vytvořením vozidla, které je umístěno na některou ze silnic, a končí při zastavení vozidla vlivem nabourání či dosažením počtu předem daných kroků epizody.

V každém kroku epizody agent pozoruje prostředí prostřednictvím jednoho snímku kamery vozidla a následně ε-chamtivým algoritmem vybírá akci, jež provede. Buď vybere akci zcela náhodnou, nebo vybere akci odpovídající výstupu neuronové sítě, jejíž vstupem je pozorovaný snímek. Tuto akci aplikuje a pozoruje nový stav, do nějž se dostal, a odměnu, kterou získal. Tuto čtveřici (snímek, akce, odměna, nový snímek) uloží do paměti zkušeností, a pokud je v paměti již dostatek zážitků, tak provede jednu iteraci učení neuronové sítě na náhodně vybrané dávce 32 zážitků. Tímto končí jeden krok epizody. Na začátku dalšího kroku agent pozoruje stav, do nějž se dostal v kroku předchozím.

Periodicky po určitém počtu epizod probíhá ohodnocení agenta na testovací sadě a zaznamenání tohoto výsledku (skóre).

Vzhledem k nestabilitě simulátoru CARLA může docházet při učení k přerušení komunikace mezi klientskou a serverovou aplikací a vyvoláním *timeoutu*. Toto jsem ošetřil vymazáním aktérů (vozidlo, senzory) v právě probíhající epizodě a navázáním nového spojení. Model neuronové sítě se ukládá po každé dokončené epizodě, takže je načten poslední dostupný model a učení pokračuje opětovným restartováním přerušené epizody.

Při experimentování s modely jsem agenta trénoval lokálně, takže opětovné restartování nebylo velkou překážkou. Obecně by to ale při trénování mohl být problém (například pokud by trénování agenta probíhalo na výpočetním clusteru).

Kapitola 4

Experimenty

V této kapitole popisuji experimenty na modelu navrženém v kapitole 3. Úpravou parametrů a nastavováním procesu učení se snažím dosáhnout optimálních výsledků. Ty jsou pak vyhodnoceny a prezentovány. Experimenty jsem prováděl na počítači s procesorem Intel Core i7-6500U @ 2.50 GHz×4 a dedikovanou grafickou kartou nVidia GeForce 940M. Na tomto stroji byl spuštěn zároveň server, jenž vykresluje a provádí simulaci, i klient, který provádí učení neuronové sítě a vyhodnocuje chování agenta.

4.1 Průběh experimentů

Jednotlivé modely se učí na nově inicializované neuronové síti, doba učení je tedy delší než v případě použití předučené sítě. Každý experiment sestává z 500 epizod. Epizoda končí havárií vozidla nebo po 1000 krocích simulace. Délka jednotlivých epizod se tedy může výrazně lišit. Simulace probíhá v reálném čase a průběh jednoho experimentu trval přibližně 10 hodin. Délku experimentu 500 epizod jsem zvolil proto, že jak ohodnocení agenta, tak i pozorování řízení agenta naznačuje, že se již několik desítek epizod v řízení vozidla nezlepšuje a agent v tomto ohledu dosáhl maxima svých možností a při dalším učení by mohlo dojít k přeučení. Nejdelší takto pozorované učení trvalo 875 epizod. Dalším důvodem proč jsem nezvolil více epizod je velká výpočetní a časová náročnost simulace prostředí a učení neuronové sítě.

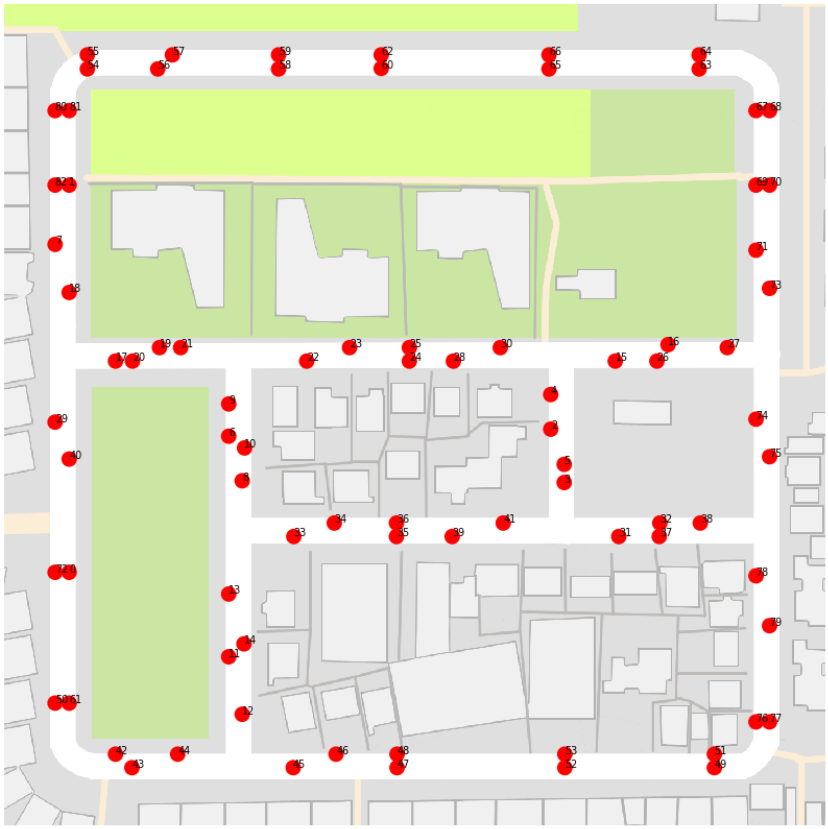
Na začátku každé epizody je vozidlo umístěno zcela náhodně na jednom z možných míst na mapě (červené body na obrázku 4.1). Díky náhodnému umístění tak v každé epizodě agent pozoruje jinou část mapy a snižuje se tím šance přeučení na konkrétním úseku silnice.

Součástí algoritmu učení je výběr akce na základě ε-chamtivého algoritmu. U všech experimentů je počáteční hodnota ε nastavena na 0.9. S každým krokem simulace se ε snižuje o hodnotu 0.0005 (nezávisle na epizodách). Po dosažení hodnoty 0.15 se ε již dále nesnižuje a učení pokračuje s touto hodnotou.

Testovací sada

Po každé páté epizodě se provádí periodicky evaluace agenta posilovaného učení. Tuto evaluaci využívám jako testovací sadu, skládající se z šesti na sobě nezávislých scénářů, které se vozidlo řízené agentem snaží projet. Na začátku každého scénáře je vozidlo umístěno na předem daných souřadnicích a před vozidlem jsou silnice různého tvaru – rovná silnice, levotočivá zatáčka, pravotočivá zatáčka a křižovatka ve tvaru „T“, ke které agent přijíždí ze tří různých směrů. Každý scénář má maximální délku 250 simulačních kroků. Může

ovšem skončit mnohem dříve vlivem havárie vozidla. V každém kroku scénáře agent zvolí akci, jež predikuje neuronová síť na základě pozorovaného snímku z kamery. Ohodnocení, jež za tuto akci agent získá, je vypočítáno stejně jako ve fázi učení a jedná se první sledovaný ukazatel schopností agenta. Výstupem testovací sady jsou dvě hodnoty. První je akumulovaná odměna, kterou agent získal v průběhu celé testovací sady vypočítána jako suma odměny v každém kroku každého scénáře. Druhá vypovídající hodnota je celkový počet kroků dosažený ve všech scénářích, což odpovídá ujeté vzdálenosti. Z těchto dvou hodnot lze vyjádřit hodnotu třetí, a to průměrnou odměnu v jednom kroku simulace. Pokud je tedy maximální ohodnocení v jednom stavu rovno 2 a testovací sada má 6 scénářů po 250 krocích, pak je maximální akumulovaná odměna, jež může agent získat za testovací sadu, rovna 3000 ($2 \times 6 \times 250$). Zároveň maximální počet kroků v testovací sadě je 1500 (6×250) a maximální dosažitelná průměrná odměna je 2. Výstupem následujících experimentů jsou průběhy hodnot v závislosti na čase simulace. Obecně se dá říci, že čím vyšší hodnoty graf zobrazuje, tím je agent lepší. Díky tomu, že jsou tyto scénáře vždy stejné, můžeme porovnat schopnosti agenta v rámci různé délky doby učení.



Obrázek 4.1: Mapa města *Town02* v simulátoru CARLA, v němž se agent posilovaného učení učí řídit model vozidla. Šedou barvou je znázorněna městská zástavba, zelenou barvou jsou znázorněny lesy a parky, bílou barvou jsou zvýrazněné silnice a červené body značí možná místa pro náhodné umístění vozidla v simulátoru (převzato z [8]).

4.2 Vliv funkce odměn

Mezi první experimenty, jež byly vykonány, patří vliv funkce odměn na proces učení algoritmu. Jednotlivé funkce odměn popsané v kapitole 3.5 byly postupně testovány a na základě výsledků pak tvořeny funkce nové.

Funkce odměn A se ukázala jako nedostačující, neboť agentem řízené vozidlo začalo jezdit střídavě zleva doprava a naopak kolem středu jízdního pruhu. Při dostatečně dlouhém učení pak začal agent zatáčet pouze doleva, a tím se vozidlo točilo v kruhu. Hlavním důvodem tohoto chování je chybějící penalizace za špatně natočené vozidlo vzhledem k vozovce. Pokud se tedy vozidlo bude neustále otáčet v kruhu uprostřed silnice, tak bude stále dostávat odměnu za to, že je na silnici. Tato situace je pro agenta z hlediska ohodnocení příznivá, neboť je malá šance, že naboura a může tedy kumulovat dosažené odměny.

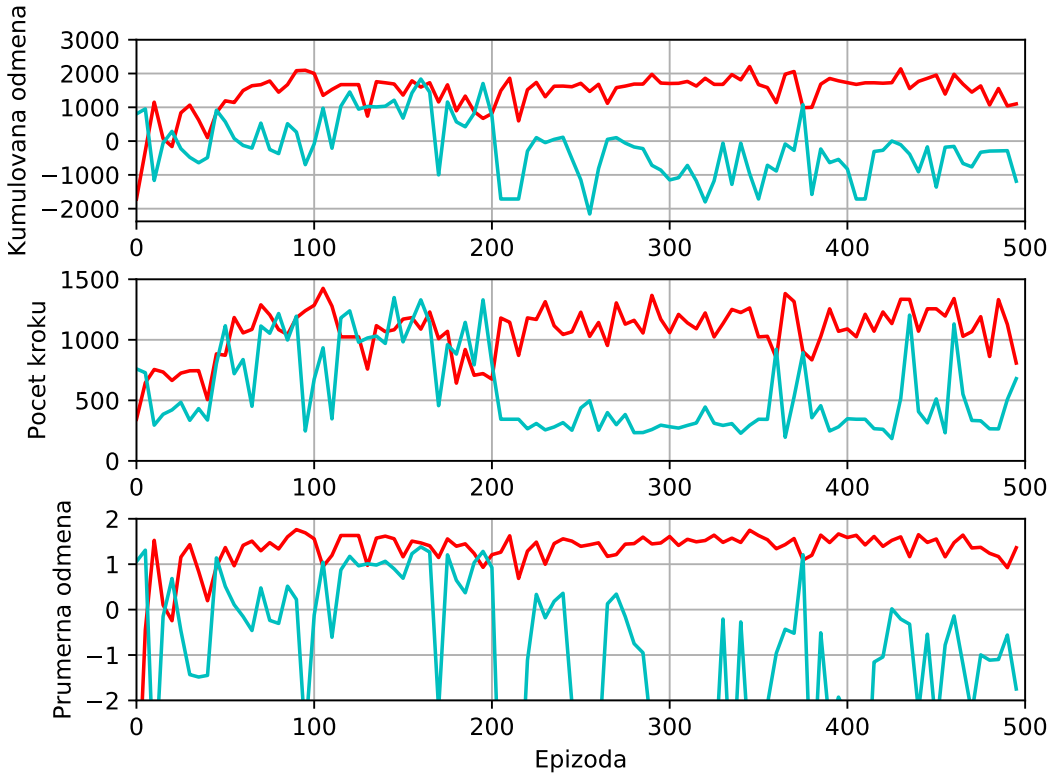
Použití funkce odměn B při učení algoritmu zabránilo agentovi otáčet vozidlem v kruhu. Při jízdě po rovné silnici stále prováděl tzv. „slalom“, ale docházelo k tomu méně často než při použití funkce odměn A. Při delší době učení pak začal agent jezdit po totožných silnicích v opačném směru tak, že při dojetí před křižovatku se otočil a odjel stejnou cestou, jakou přijel. Toto se opakovalo před každou křižovatkou a agent tak neustále jezdil mezi dvěma křižovatkami. Jedním z důvodů tohoto chování může být přeučení, v tomto případě sledování pouze středové čáry. V křižovatce tato středová čára není a agent by se tedy neměl podle čeho orientovat, proto se raději otočí.

Při použití funkce odměn C, která vycházela z nedostatků dvou předešlých funkcí odměn a přidává další dva druhy penalizace, začal agent jezdit smysluplně po většinu času řízení. Z toho důvodu používám pro další experimenty pouze funkci odměn C, pro níž se snažím nalézt optimální parametry učení. Základní verze je trénována s pamětí zkušeností o velikosti 5000 zkušeností, jednotlivé dávky mají velikost 32 zkušeností a diskontní faktor je 0.95. Na následujících obrázcích, kdy tento základní algoritmus srovnávám s jeho modifikacemi, je průběh učení tohoto algoritmu značen červenou barvou.

Porovnání funkcí odměn A, B a C mezi sebou vzhledem k evaluační sadě testů zde neuvádím. Přestože funkce odměn A a B dosahovaly v této testovací sadě obdobných a někdy i lepších výsledků než funkce odměn C, tak jejich chování je nežádoucí, a tím pádem je jakékoliv jejich další ohodnocení irelevantní.

4.3 Vliv paměti zkušeností

Velmi důležitou součástí implementovaného algoritmu je paměť zkušeností. V tomto experimentu jsem porovnával nastavení hyperparametrů algoritmu učení – konkrétně velikost paměti zkušeností. Algoritmem učení je základní algoritmus vycházející z funkce odměn C. V jednom případě jsem ponechal paměť zkušeností o velikosti 5000. Ve druhém případě jsem nastavil kapacitu paměti zkušeností na hodnotu 500. Tedy s pamětí desetkrát menší. Dávka, jež se z paměti vybírá, je shodná pro oba agenty a má kapacitu 32 zkušeností. Srovnání těchto dvou nastavení hyperparametrů je zobrazeno na obrázku 4.2, kde první zmiňované nastavení je značeno barvou červenou a druhé nastavení barvou azurovou. Zatímco agent s pamětí kapacity 5000 se začal učit poměrně rychle a proces učení začal konvergovat, agent s pamětí o kapacitě 500 byl velmi nestabilní. V určitých fázích učení podával dobré výsledky – maximální dosažené ohodnocení je 1837 bodů po 1330 krocích. Tyto výsledky byly ovšem pouze krátkodobé a agent byl následně hodnocen i záporně. Z tohoto experimentu vyplývá, že pokud není paměť zážitků dostatečně velká, tak se agent nedokáže správně učit, neboť stavy, jež jsou v paměti uloženy, jsou si podobné. Například, pokud vozidlo pojede dlouhou

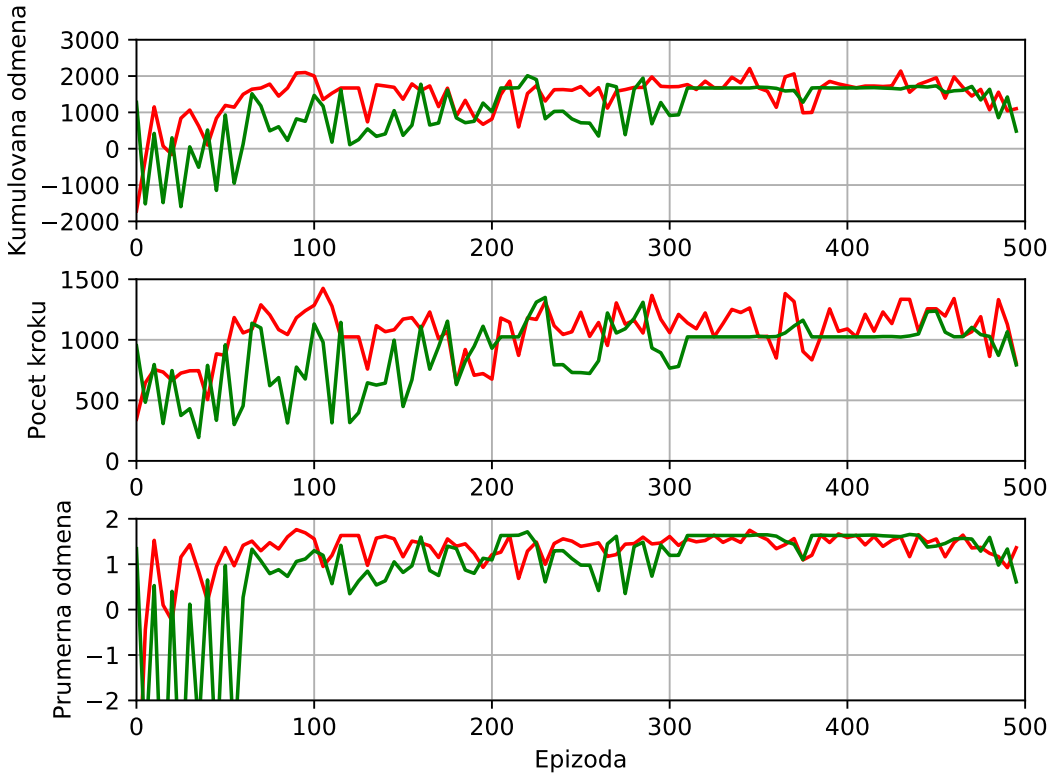


Obrázek 4.2: Ohodnocení dvou agentů během učení. Základní agent s pamětí zážitků o velikosti 5000 je značen červenou barvou. Agent s pamětí 500 je značen azurovou barvou (jeho průměrná odměna dosahovala místy i hodnoty −5).

dobu jen po rovné silnici, tak zážitky uložené v paměti budou téměř totožné. Tím pádem se agent několik iterací po sobě učí na velmi podobné dávce (učení neuronové sítě probíhá v každém kroku simulace). Toto vede k přeučení sítě a tedy neschopnosti se adaptovat na odlišné prostředí.

4.4 Vliv diskontního faktoru

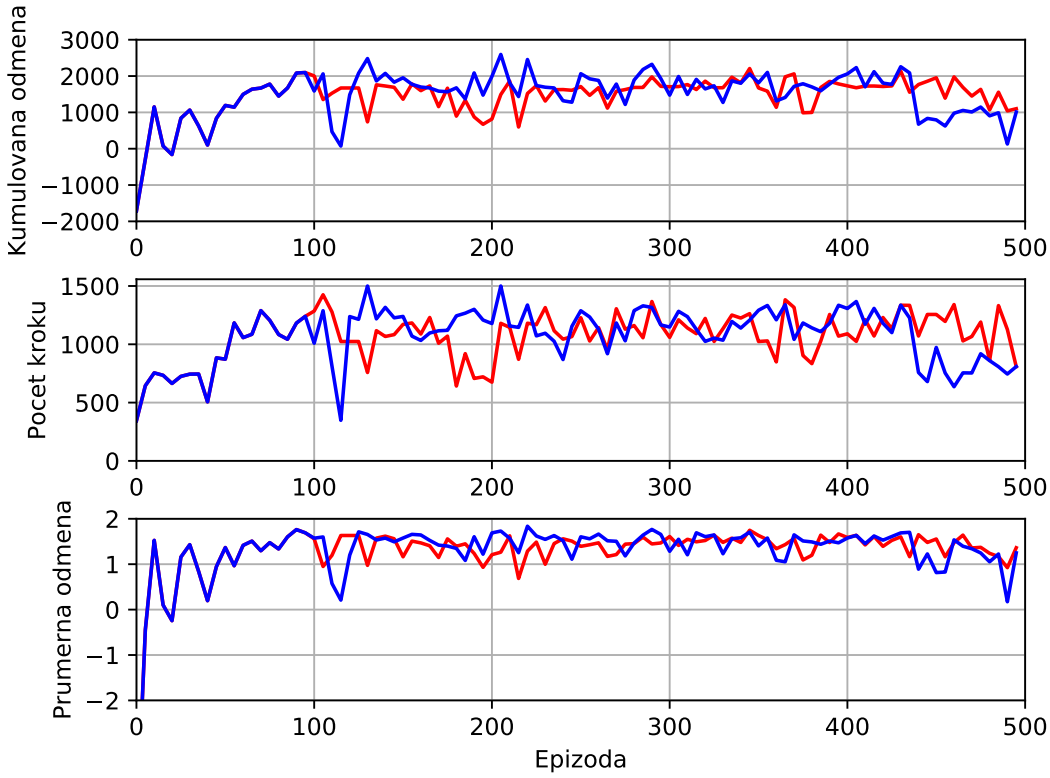
V tomto experimentu jsem porovnával opět základní algoritmus vycházející z funkce odměn C (značen červeně) s algoritmem, u nějž jsem zvýšil hodnotu diskontního faktoru na hodnotu 0.99 (značen zeleně). Výsledek porovnání ohodnocení učení je na obrázku 4.3. Algoritmus s vyšší hodnotou diskontního faktoru se zpočátku učil pomaleji než algoritmus s diskontním faktorem 0.95. Jeho ohodnocení také zpočátku více oscilovalo. Po dostatečně dlouhé době učení se však výsledky získané druhým sledovaným algoritmem stabilizovali a v průměru dosahovali podobných odměn jako výsledky prvního sledovaného algoritmu. Nicméně první algoritmus s diskontním faktorem 0.95 dokázal při evaluaci ujet více kroků před havárií než algoritmus s diskontním faktorem 0.99.



Obrázek 4.3: Porovnání ohodnocení procesu učení dvou agentů. Červeně je označen základní agent s diskontním faktorem 0.95, zeleně je označen agent s diskontním faktorem 0.99.

4.5 Výsledný algoritmus učení

Pro získání lepších výsledků a snahy o další učení algoritmu vycházejícího z funkce odměn C, jsem použil natrénovanou neuronovou síť (diskontní faktor byl roven 0.95 a paměť zkušeností měla kapacitu 5000) po 100 epizodách, kdy dosahovala nejlepšího ohodnocení. Tuto neuronovou síť jsem následně použil jako předtrénovanou síť pro nové učení. Tedy po 100 epizodách jsem znovu nastavil ε na hodnotu 0.9, a ta se opět s každým krokem snižovala, stejně jako tomu bylo u prvního učení. Na obrázku 4.4 je zobrazeno ohodnocení normálního průběhu učení algoritmu (značen červeně) a ohodnocení učení, jež vycházelo z tohoto natrénovaného modelu (označeno modře). U tohoto modelu, který se učil z předtrénované sítě je vidět velký pokles ohodnocení při začátku druhého učení. To je způsobeno velkým množstvím náhodných akcí, díky nimž se agent dostane i do nepříznivých stavů. Díky tomu ale může prozkoumat nové stavy a po dalším učení dokázal jako jediný absolvovat celou testovací sadu bez havárie. Po dalších 100 epizodách je však žádoucí ukončit proces učení, neboť pak opět klesá ohodnocení tohoto agenta, pravděpodobně z důvodu přeučení.



Obrázek 4.4: Porovnání ohodnocení základního agenta označeného červeně s agentem, jež opakuje ε-chamtivý algoritmus označeným modře.

4.6 Porovnání výsledků s existujícími modely

V porovnání s jinými algoritmy učení používaných v oblasti autonomního řízení jako například imitační učení[1], dosahuje model natrénovaný prostřednictvím posilovaného učení horších výsledků. To je dáno tím, že posilované učení musí celé prostředí prozkoumat samostatně, kdežto u imitačního učení se snaží imitovat vzor chování.

V porovnání s modelem natrénovaným týmem, vyvíjejícím simulátor CARLA za pomoci posilovaného učení[2], dosahuje můj nejlépe natrénovaný model vizuálně podobných výsledků (tedy že agentem řízené vozidla ujedou určitou vzdálenost než havarují). Model týmu CARLA však pro učení nepoužívá pouze snímky RGB kamery, ale také další měření. Jejich predikce však nepředstavují pouze úhel natočení volantu, ale také sílu stlačení plynového pedálu. Rozdílná jsou i prostředí pro trénování. Model týmu CARLA byl trénován při běžném silničním provozu, kdežto mnou trénovaný model se učil ve městě bez dalšího silničního provozu. Další podrobnosti k procesu učení tohoto modelu nejsou prozatím zveřejněny. Je proto obtížné porovnávat tyto modely mezi sebou.

Dalšími pracemi, které se zabývají tématikou autonomního řízení a ukazují jiné možné přístupy učení jsou pak například [14] a [24].

[1] https://github.com/carla-simulator/imitation-learning
[2] https://github.com/carla-simulator/reinforcement-learning

Kapitola 5

Závěr

V této práci jsem popsal principy strojového učení a algoritmus posilovaného učení ve spojení s neuronovými sítěmi. Cílem bylo aplikovat posilované učení pro ovládání modelu vozidla v simulátoru. Pro učení a hodnocení jednotlivých algoritmů jsem zvolil volně dostupný simulátor CARLA ve spojení s knihovnami Keras a TensorFlow.

Implementoval jsem algoritmus DQN, jenž predikuje akce za pomocí neuronové sítě. Tento algoritmus jsem obohatil o ε-chamtivý algoritmus výběru akcí a paměť zkušeností. Výsledný algoritmus predikoval natočení volantu na základě RGB snímku z přední kamery vozidla. V průběhu implementace jsem navrhl několik funkcí odměn, prostřednictvím kterých jsem penalizoval nežádoucí akce.

Dále byla implementována evaluační sada scénářů, na níž byly hodnoceny jednotlivé experimenty s různými modely určené pro řízení vozidla.

I přestože jednotlivá trénování modelů probíhala 500 epizod, tak většina modelů dosáhla svého limitu po 100 až 200 epizodách. Nejlépe natrénovaný model dokázal úspěšně projet všechny evaluační scénáře bez havárie a jeho nejlepší ohodnocení bylo 2595 z maximálního ohodnocení 3000. Pokud tento natrénovaný model jezdil volně v simulovaném městském prostředí, pak ujel průměrně 2400 simulačních kroků než havaroval. Maximální dosažený počet dosažených kroků této simulace byl 9361. Bohužel se nepodařilo natrénovat takový model, který by nikdy nehavaroval. Největším problémem bylo predikovat vhodné natočení volantu na křižovatce typu „T“.

Pro další vývoj této práce je možné přidat hloubkovou mapu prostředí prostřednictvím kamery nebo LIDARu, která pomůže neuronové síti lépe rozpoznat překážky a vyhnout se tak kolizím. Také by bylo možné určovat směr jízdy na křižovatkách prostřednictvím dalšího vstupu neuronové sítě a sledovat vývoj jednotlivých modelů při delší době učení.

Literatura

[1] Asadi, K.; Littman, M. L.: An Alternative Softmax Operator for Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, JMLR.org, 2017, s. 243–252.

[2] Ashraf, M.: Reinforcement Learning Demystified: Exploration vs. Exploitation in Multi-armed Bandit setting. 2018, [Online; accessed 25-April-2019].
URL https://towardsdatascience.com/reinforcement-learning-demystified-exploration-vs-exploitation-in-multi-armed-bandit-setting-be950d2ee9f6

[3] Bellman, R.: A Markovian Decision Process. *Indiana Univ. Math. J.*, ročník 6, 1957: s. 679–684, ISSN 0022-2518.

[4] Bengio, Y.: Learning Deep Architectures for AI. *Found. Trends Mach. Learn.*, ročník 2, č. 1, Leden 2009: s. 1–127, ISSN 1935-8237.

[5] Britz, D.: Understanding Convolutional Neural Networks for NLP. 2015, [Online; accessed 3-May-2019].
URL http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[6] Cornelisse, D.: An intuitive guide to Convolutional Neural Networks. 2018, [Online; accessed 3-May-2019].
URL https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

[7] Donges, N.: Gradient Descent in a Nutshell. 2018, [Online; accessed 2-May-2019].
URL https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

[8] Dosovitskiy, A.; Ros, G.; Codevilla, F.; aj.: CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, s. 1–16.

[9] EliteDataScience.com: Overfitting in Machine Learning: What It Is and How to Prevent It. 2017, [Online; accessed 20-April-2019].
URL https://elitedatascience.com/overfitting-in-machine-learning

[10] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[11] Grover, P.: 5 Regression Loss Functions All Machine Learners Should Know. 2018, [Online; accessed 2-May-2019].

URL https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

[12] Karpathy, A.; Johnson, J.: CS231n: Convolutional Neural Networks for Visual Recognition. 2019, [Online; accessed 2-May-2019].
URL http://cs231n.github.io/convolutional-networks/

[13] Kingma, D. P.; Ba, J.: Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
URL https://arxiv.org/pdf/1412.6980.pdf

[14] Lau, B.: Using Keras and Deep Deterministic Policy Gradient to play TORCS. 2016, [Online; accessed 10-May-2019].
URL https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html

[15] Leung, K.; Leckie, C.: Unsupervised anomaly detection in network intrusion detection using clusters. In *ACSC '05 Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume 38*, Australian Computer Society, Inc., 2005, ISBN 1-920-68220-1, s. 333–342.

[16] McCulloch, W. S.; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, ročník 5, č. 4, Dec 1943: s. 115–133, ISSN 1522-9602.

[17] McGonagle, J.; García, J. A.: Feedforward Neural Networks. Brilliant.org, [Online; accessed 2-May-2019].
URL https://brilliant.org/wiki/feedforward-neural-networks/

[18] McGonagle, J.; Williams, C.; Khim, J.: Recurrent Neural Network. Brilliant.org, [Online; accessed 2-May-2019].
URL https://brilliant.org/wiki/recurrent-neural-network/

[19] Minsky, M. L.: *Semantic Information Processing*. The MIT Press, 1969, ISBN 0262130440.

[20] Mnih, V.; Kavukcuoglu, K.; Silver, D.; aj.: Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*, 2013.
URL https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[21] Salian, I.: SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning? 2018, [Online; accessed 20-April-2019].
URL https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

[22] Silver, D.; Hubert, T.; Schrittwieser, J.; aj.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, ročník 362, č. 6419, 2018: s. 1140–1144.
URL http://science.sciencemag.org/content/362/6419/1140/tab-pdf

[23] Silver, D.; Schrittwieser, J.; Simonyan, K.; aj.: Mastering the game of Go without human knowledge. *Nature*, ročník 550, Říjen 2017: s. 354–359.

[24] Skácel, D.: *Navigace pomocí hlubokých konvolučních sítí*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=21455

[25] Smart, B.: Reinforcement Learning: A User's Guide. 2005, [Online; accessed 3-May-2019].
URL http://www2.econ.iastate.edu/tesfatsi/RLUsersGuide.ICAC2005.pdf

[26] Spryn, M.; Sharma, A.; Parkar, D.: Distributed Deep Reinforcement Learning for Autonomous Driving. 2018, [Online; accessed 3-May-2019].
URL https://github.com/microsoft/AutonomousDrivingCookbook/tree/master/DistributedRL

[27] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, ročník 15, č. 1, Leden 2014: s. 1929–1958, ISSN 1532-4435.
URL http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[28] Stergiou, C.; Siganos, D.: Neural Networks. [Online; accessed 27-April-2019].
URL
https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[29] Sutton, R. S.; Barto, A. G.: *Reinforcement Learning: An Introduction*. The MIT Press, 2018, ISBN 978-0262039246.
URL http://incompleteideas.net/book/RLbook2018.pdf

[30] Tokic, M.; Palm, G.: Value-difference Based Exploration: Adaptive Control Between Epsilon-greedy and Softmax. In *Proceedings of the 34th Annual German Conference on Advances in Artificial Intelligence*, KI'11, Berlin, Heidelberg: Springer-Verlag, 2011, ISBN 978-3-642-24454-4, s. 335–346.

[31] Wikipedia contributors: Supervised learning — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 20-April-2019].
URL https://en.wikipedia.org/wiki/Supervised_learning

[32] Wikipedia contributors: Bellman equation — Wikipedia, The Free Encyclopedia. 2019, [Online; accessed 23-April-2019].
URL https://en.wikipedia.org/wiki/Bellman_equation

[33] Zhang, S.; Sutton, R. S.: A Deeper Look at Experience Replay. 2017.
URL https://arxiv.org/pdf/1712.01275.pdf

Příloha A

Obsah přiloženého paměťového média

- soubory se zdrojovým kódem
- natrénované modely a jejich ohodnocení
- technická zpráva ve formě PDF s šablonou LaTeX
- soubor README.md s návodem k instalaci a spuštění