VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

SOUTĚŽNÍ HŘIŠTĚ PRO UMĚLOU INTELIGENCI
COMPETITION PLAYGROUND FOR ARTIFICIAL INTELLIGENCE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Matouš Benda**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Radomil Matoušek, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Matouš Benda**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2017/18

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Soutěžní hřiště pro umělou inteligenci

Stručná charakteristika problematiky úkolu:

Cílem práce bude navrhnout hru s rozhraním pro soutěž algoritmů umělé inteligence a vytvořit vlastního hráče (umělou inteligenci).

Cíle diplomové práce:

Popis problematiky implementované umělé inteligence.
Vlastní implementace zavedené 2D hry.
Tvorba programové implementace platformy – "hřiště" pro soutěž dvou umělých inteligencí (předpokladem je socketové rozhraní).
Návrh vlastního hráče – umělé inteligence.
Tvorba příslušné e–dokumentace.

Seznam doporučené literatury:

MURPHY, Kevin P. (2012): Machine Learning: A Probabilistic Perspective, MIT Press. ISBN 978--262-01802-9.

MARK, D. (2009): Behavioral Mathematics for Game AI (Applied Mathematics), Cengage Learning PTR. ISBN-13: 978-1584506843.

JOSHI, P. (2017): Artificial Intelligence with Python, Packt Publishing - ebooks Account. ISBN-13: 978-1786464392.

GOODFELLOW, I et al. (2016): Deep Learning (Adaptive Computation and Machine Learning series), The MIT Press. ISBN-13: 978-0262035613.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2017/18

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Diplomová práce pojednává o možnostech hraní her pomocí metod umělé inteligence a představuje návrh řešení konkrétní hry využívající tento přístup. Umělá inteligence je v této práci reprezentována umělou neuronovou sítí. První teoretická část je zaměřena na popis celosvětově úspěšných a známých umělých inteligencí. Následuje stručný popis neuronových sítí a rozbor některých knihoven pro strojové učení. Praktická část je zaměřena na implementaci vytvořené hry pomocí programovacího jazyka Python. V závěru je proveden rozbor předloženého řešení.

ABSTRACT

Master thesis deals with possibilities of playing games using artificial intelligence. Also, there is presented artificial intelligence with ability of playing selected game. Artificial intelligence presented in this work is considered as artificial neural networks. First theoretical part is focused on description worldwide successful and known artificial intelligence strategies. After that there is brief description of neural networks and analysis of some libraries for machine learning. Practical part is focused on implementation of created game with Python programming language and in the end, there is analysis of designed solution.

KLÍČOVÁ SLOVA

Umělá inteligence, umělá neuronová síť, strojové učení, hraní počítačových her

KEYWORDS

Artificial intelligence, artificial neural network, machine learning, playing computer games

BIBLIOGRAFICKÁ CITACE

BENDA, Matouš. *Soutěžní hřiště pro umělou inteligenci*. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Ústav automatizace a informatiky. Brno, 2018, 63 s. Vedoucí práce doc. Ing. Radomil Matoušek Ph.D.

PODĚKOVÁNÍ

V první řadě děkuji panu doc. Ing. Radomilovi Matouškovi, Ph.D. za cenné rady a zkušenosti při vedené této práce, dále panu Ing. Petrovi Šoustkovi za nenahraditelnou pomoc a v neposlední řadě svým rodičům, kteří mi umožnili dostat se tam, kde jsem.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Radomila Matouška Ph.D. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 20. 5. 2018 ……

Bc. Matouš Benda

OBSAH

1 ÚVOD 15

2 UMĚLÁ INTELIGENCE A HRY 17
2.1 Umělá inteligence 17
2.2 Herní průmysl 17
2.3 AlphaGo 18
2.3.1 Go 18
2.3.2 Učení hry 18
2.3.3 Hraní hry 20
2.4 DeepMind Atari 2600 21
2.4.1 Atari 2600 21
2.4.2 Základní myšlenka 21
2.4.3 Vstupní data 22
2.4.4 Architektura sítě 22
2.4.5 Výsledky 22

3 UMĚLÉ NEURONOVÉ SÍTĚ 25
3.1 Učení umělých neuronových sítí 26
3.1.1 Učení s učitelem 26
3.1.2 Posilované učení 26
3.1.3 Učení bez učitele 26
3.2 Backpropagation metoda 26
3.3 Aktivační funkce 27
3.3.1 Jednotkový skok 27
3.3.2 Lineární funkce 27
3.3.3 Sigmoida 28
3.3.4 Hyperbolický tangens 28
3.3.5 Rektifikovaná lineární jednotka 29
3.3.6 Softmax 30
3.4 Typy umělých neuronových sítí 30
3.4.1 Dopředná neuronová síť 30
3.4.2 Rekurentní neuronová síť 31
3.4.3 Konvoluční neuronová síť 32

4 STROJOVÉ UČENÍ 33
4.1 Knihovny pro strojové učení 33
4.1.1 Keras 33
4.1.2 TensorFlow 34
4.1.3 Theano 36
4.1.4 Microsoft Cognitive Toolkit 36
4.1.5 Caffe2 36
4.1.6 MXNet 36
4.1.7 PyTorch 36
4.2 Q learning 37
4.3 Deep Q learning 38

5 POPIS IMPLEMENTACE 41
5.1 Snake 41
5.2 SnakeAI 41

5.2.1 Pygame 42
5.2.2 Herní objekty 43
5.2.3 Soketové rozhraní 44
5.2.4 Herní algoritmus 45
5.3 Hodnotící heuristiky 47
5.3.1 Manhattan vzdálenost (Hammingova metrika) 47
5.3.2 Euklidovská vzdálenost 48
5.3.3 Vzdálenost pomocí algoritmu A* 48
5.4 Navržené umělé inteligence 51
5.4.1 Umělá inteligence Alfa 51
5.4.2 Umělá inteligence Beta 51
5.4.3 Umělá inteligence Beta+ 52
5.4.4 Umělá inteligence Gama 54
6 ZÁVĚR 55
7 SEZNAM POUŽITÉ LITERATURY 57
8 SEZNAM OBRÁZKŮ 61
9 SEZNAM PŘÍLOH 63

1 ÚVOD

Umělá inteligence je v našem století velmi komentované a živé téma a z toho důvodu jsem vybral právě tuto problematiku. Tématem diplomové práce je soutěžní hřiště pro umělou inteligenci. Každý člověk se rád baví, aneb slovní spojení škola hrou je stále platné. Řešení práce je zaměřeno na využití umělé inteligence v kontextu hraní počítačových her. Na umělou inteligenci je v textu převážně pohlíženo jako na umělé neuronové sítě.

Mým cílem bylo vytvořit práci schopnou čtenáři přiblížit praxí ověřené algoritmy a ukázat, že na základě neustálého vývoje výpočetní techniky je v dnešní době možné některé tyto algoritmy svépomocí implementovat a vytvořit vlastní umělou inteligenci.

Cílem práce je navrhnout počítačovou hru, na kterou lze pohlížet jako na hrací hřiště, kde je možné výkonnostně porovnat vytvořené umělé inteligence. V první části práce jsou rozebrány profesionální přístupy, díky kterým se vytvořená umělá inteligence naučila hrát některé hry a svým výkonem v mnoha případech zastínila lidské protihráče. Na tuto část navazuje popis umělých neuronových sítí, jejich rozdělení a popis některých základních komponent.

Pro vytváření vlastní neuronové sítě je použita open source knihovna pro strojové učení, proto jsou v další části popsány nejpoužívanější knihovny. Kapitola je zakončena dvěma na sebe navazujícími algoritmy (*Q learning* a *Deep Q learning*). Pro pochopení problematiky strojového učení je vhodné tyto algoritmy znát.

Poslední částí je vlastní implementace. Tato část popisuje vytvořenou hru a způsob použití některých heuristik, které jsou z hlediska strojového učení a umělých inteligencí velmi důležité. Celou práce je zakončena vlastním návrhem umělé inteligence pro řešení zvolené arkádové hry – Snake. Navrženy jsou čtyři umělé inteligence, přičemž v prvních třech případech tyto inteligence dosahují úctyhodných výsledku i při hraní na běžných osobních počítačích.

Předložení práce byla pro autora prvotní studií v daném kontextu. Je nesporné, že metody umělé inteligence budou do oblasti počítačových her zasahovat v budoucnu ještě intenzivněji a schopnosti lidských hráčů budou méně a méně konkurenční. Pravděpodobným trendem pak bude umělá inteligence pomáhající lidským hráčům, otázkou pak zůstane, pro koho to bude hra.

2 UMĚLÁ INTELIGENCE A HRY

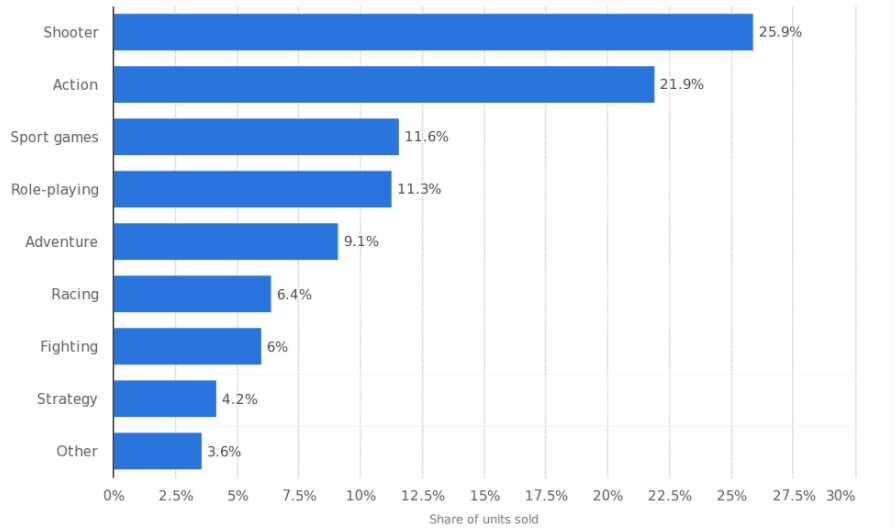
2.1 Umělá inteligence

Umělou inteligencí je lidstvo fascinováno již téměř 100 let. První zmínku o umělé inteligenci, i když nepřímou, lze nalézt ve vědeckofantastickém dramatu spisovatele Karla Čapka R.U.R.. Toto dílo pochází z roku 1920 a je zde poprvé použit pojem „robot“. Tato entita představuje inteligentní, lidmi vyvinutý stroj, schopný vykonávat běžné činnosti, jinými slovy umělou inteligenci uvnitř robotického těla.

Umělou inteligenci lze definovat více způsoby, ale pro účely této práce se omezíme na definici následující. Umělá inteligence je termín pro simulovanou inteligenci ve strojích. Tyto stroje jsou programované tak, aby myslely jako člověk a napodobovaly lidské, nebo zvířecí chování. Ideální charakteristika umělé inteligence je schopnost logicky vyhodnotit prostředí a na základě tohoto vyhodnocení vybrat akci, která vede k dosažení určitého cíle, nebo k přiblížení k tomuto cíli. [1]

2.2 Herní průmysl

V posledních několika desetiletích došlo k velkému vývoji a rozmachu herního průmyslu. Obliba videoher mezi mladými, ale i staršími lidmi, je zapříčiněna jejich snadnou dostupností a stále rostoucí kvalitou zpracování. Videohry dávají člověku možnost relaxace a vzdělávání z pohodlí domova. Nejčastější rozdělení videoher je podle žánru. Příkladem můžou být akční hry, adventury, sportovní hry, deskové hry, strategické hry a další. Na obr. 1. je graf procentuálního zastoupení prodeje her dle žánru, vytvořený roku 2017 v USA. [2]



Obr. 1) Podíl prodeje her dle žánrů [3]

2.3 AlphaGo

AlphaGo je počítačový program společnosti *DeepMind*, jehož úkolem je porazit člověka v deskové hře Go. Největším úspěchem této umělé inteligence byla porážka světového šampiona a pravděpodobně nejlepšího hráče na světě Ke Jie. V průběhu her AlphaGo přišla s takovými vítěznými tahy, které následně elitní hráči studovali a na jejich základě změnili celé herní strategie. [4]

2.3.1 Go

Go je desková hra pro dva hráče pocházející z Číny (obr. 2). Hru tvoří deska s vyznačenými čtverci. Mezi těmito čtverci jsou průsečíky, na které hráči střídavě pokládají černé a bílé kameny. Obklíčením skupiny protivníkových kamenů dojde k zajmutí těchto kamenů a hráč je bodově ohodnocen. Koncept této hry je velmi jednoduchý, ale její komplexita astronomická. Počet všech možných herních stavů je okolo 10^{170}. [4][5]

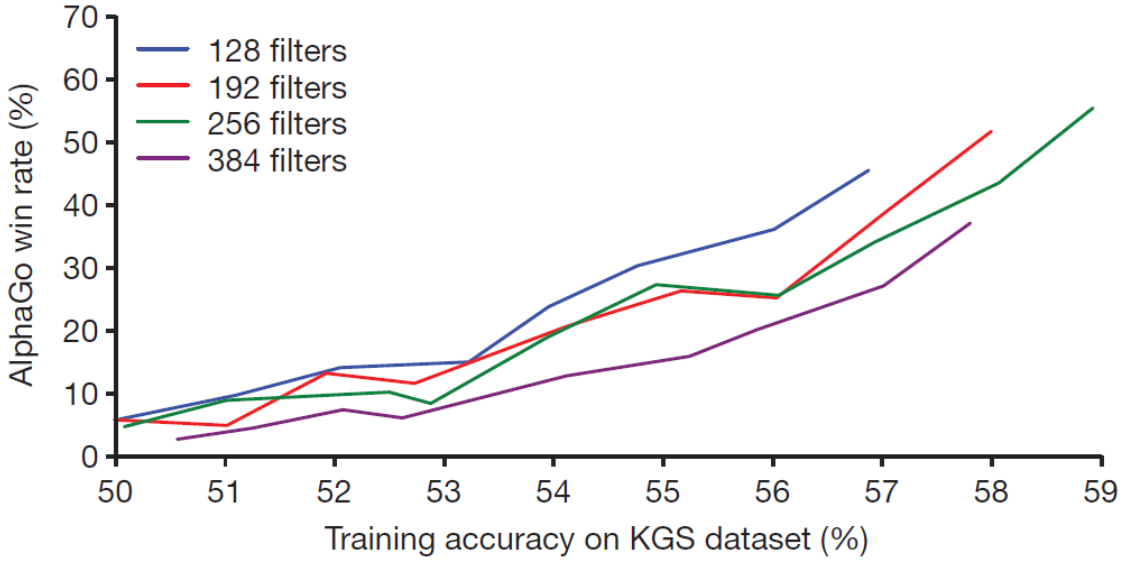
Obr. 2) Desková hra Go [6]

2.3.2 Učení hry

Klasické metody umělých inteligencí, jako prohledávání stromového grafu, se ukázaly za zcela nedostatečné. Hlavním problémem bylo velké množství možností, ale i způsob ohodnocování jednotlivých tahů. Z tohoto důvodu, byly jako hlavní komponenty umělé inteligence, zvoleny hluboké neuronové sítě v kombinaci s algoritmem *Monte Carlo tree search*. Umělé neuronové sítě, jejich typy, typy učení a aktivační funkce jsou vysvětleny ve 3. kapitole této práce. Základním prvkem jsou 2 konvoluční neuronové sítě nazývané *rozhodovací síť* (*policy network)* a *ohodnocovací síť* (*value network)*. S pomocí těchto sítí dojde k značné redukci prohledávaného prostoru. Jako vstup těchto sítí byla zvolena herní plocha o velikosti 19 x 19 bodů. Učení těchto sítí probíhalo v následujících etapách. [7]

Učení s učitelem rozhodovací sítě

Rozhodovací síť je tvořena konvolučními vrstvami s *ReLU* aktivační funkcí. Poslední vrstvou se *softmax* aktivační funkcí je vrstva klasifikační, která udává pravděpodobnost na základě všech možných tahů. Trénování probíhalo na více než 30 milionech trénovacích dat. Data byla získána pomocí profesionálních hráčů, byla vybírána náhodně a každý trénovací set se skládal z herního stavu *s* a zvolené akce *a*. Přesnost predikce těchto tahů dosáhla 57%. Následující graf znázorňuje závislost mezi přesností predikce, procentuálním vítězství a počtu použitých konvolučních filtrů. [7]



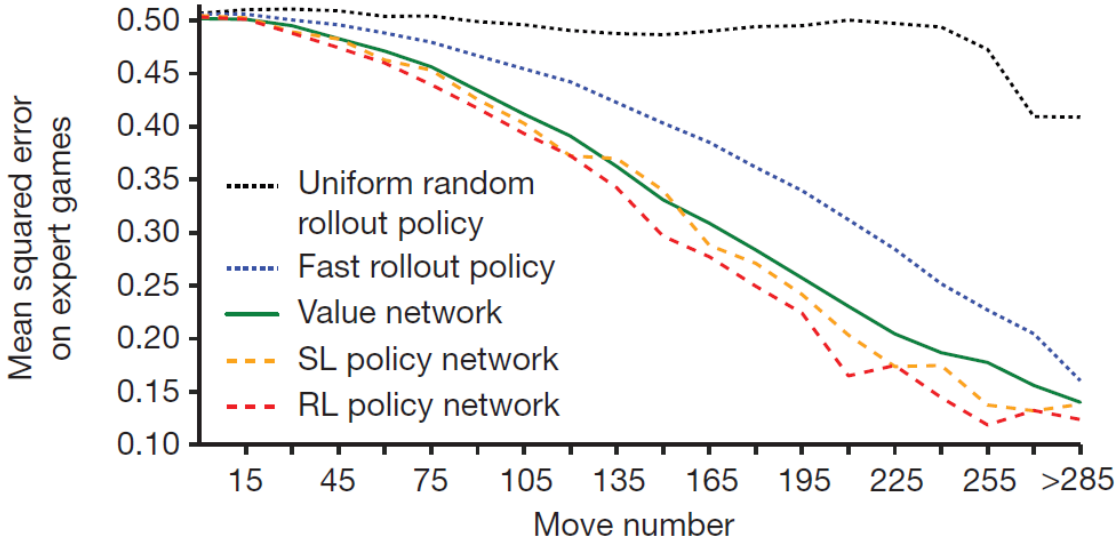
Obr. 3) AlphaGo výkonnostní graf, [7]

Posilované učení rozhodovací sítě

Posilované učení probíhalo hraním aktuální rozhodovací sítě proti náhodně zvolené síti z předchozích iterací. Tato strategie byla zvolena jako prevence *přeučení* (*overfitting*) aktuální sítě. Díky posilovanému učení tato síť vyhrála více jak 80% her proti svému předchůdci. Síť byla následně testována v souboji s nejsilnějším open source programem *Pachi*. AlphaGo vyhrála více jak 85% her, přičemž síť z předchozí fáze vyhrála pouze 11% her. [7]

Posilované učení ohodnocovací sítě

Poslední fáze učení byla zaměřena na ohodnocení jednotlivých herních pozic. Architektura této sítě je stejná jako architektura sítě předchozí, s rozdílem pouze jednoho výstupu. Trénování probíhalo na více než 30 milionech různých pozicích, kde každá pozice byla získána právě z jedné hry hrané mezi dvěma rozhodovacími sítěmi. Trénovací data tvoří herní pozice a odpovídající ohodnocení. Trénovací ohodnocení bylo získáno na základě odhadu rozhodovací sítě. Následující graf odpovídá závislosti mezi chybou predikce, hernímu tahu a porovnává výsledky různých hodnotících přístupů. [7]



Obr. 4) Graf přesnosti ohodnocení [7]

2.3.3 Hraní hry

Vlastní hraní hry funguje na principu prohledávání stromového grafu. Každá hrana (s, a) uchovává ohodnocení $Q(s, a)$, počet navštívení $N(s, a)$ a předchozí pravděpodobnost $P(s, a)$. Stromový graf je procházen a v každém kroku simulace t je ze stavu s_t zvolena akce a_t na základě vztahu (1).

$$a_t = argmax\big(Q(s_t, a) + u(s_t, a)\big)$$
$$u(s_t, a) \propto \frac{P(s, a)}{1 + N(s, a)} \tag{1}$$

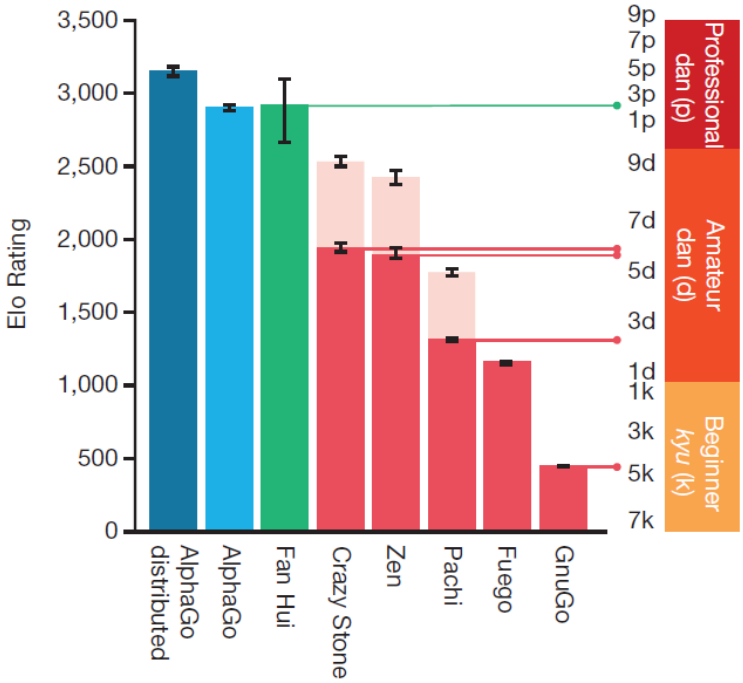
Jakmile simulace dospěje k listovému uzlu s_L, dojde k ohodnocení tohoto uzlu $V(s_L)$ a následně jsou všechny minulé hrany aktualizovány podle vztahů (2) a (3).

$$N(s, a) = \sum_{i=1}^{n} 1(s, a, i) \tag{2}$$

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{n} 1(s, a, i) V(s_L^i) \tag{3}$$

Na konci algoritmu je vybrána hrana s největším počtem navštívení. [7]

AlphaGo využívá asynchronní vícevláknové prohledávání. Konečná verze AlphaGo využívá 40 prohledávacích vláken běžících na 48 CPU a rozhodovací síť běžící společně s ohodnocovací sítí na 8 GPU. Graf na obr. 5 porovnává jednotlivé Go programy v závislosti na oficiálním *Elo* hodnocení výkonu. [7]



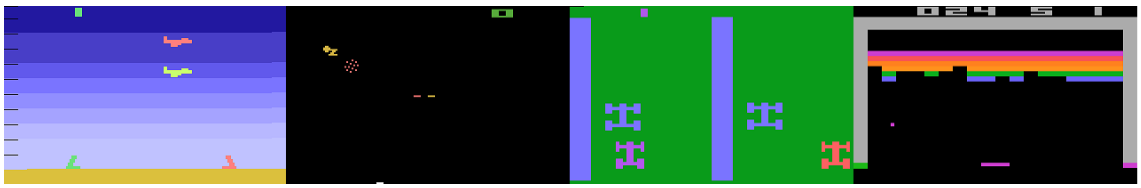
Obr. 5) porovnání Go programů [7]

2.4 DeepMind Atari 2600

Společnost DeepMind vytvořila počítačový program se schopností naučit se hrát *Atari 2600* hry. Tento algoritmus kombinuje konvoluční neuronovou síť a specificky upravený algoritmus posilovaného učení nazvaný *Deep Q Learning*. Tento algoritmus je popsán v kapitole 4.3. [8]

2.4.1 Atari 2600

Atari 2600 je herní konzole představená roku 1977 a vyvinutá společností *Atari, Inc.* Na tuto konzoli se historicky pohlíží jako na velkého popularizátora mikroprocesorového hardwaru. Typické bylo použití ROM kazet, obsahujících jednotlivé hry. Na obr. 6 jsou některé hry vybrány. [9]



Obr. 6a) Air-Sea Battle [10], 6b) Star Ship [11], 6c) Street Racer [12], 6d) Breakout [13]

2.4.2 Základní myšlenka

Uvažujme herní prostředí E, v tomto případě Atari 2600 emulátor a posloupnost akcí a, pozorování x a odměn r. V každém herním časovém okamžiku vybírá umělá inteligence akci a_t z povolených akcí $A=\{1,\ldots,K\}$. Akce je následně vykonána a na jejím základě

dojde ke změně herního stavu a herního skóre. Agent sám o sobě vnitřní stav hry nezná, zná pouze herní obraz x_t, který je reprezentován pomocí pixelů. Pozorování samostatného stavu hry se ukázalo jako nedostatečné, protože vyhodnotit situaci nelze z jednoho obrazu (dynamika hry). Z tohoto důvodu se zavádí posloupnost akcí a pozorování $s_t=x_1,a_1,x_2,\ldots a_{t-1},x_t$. Všechny tyto posloupnosti jsou konečné, pro řešení lze aplikovat *Markovův rozhodovací proces* (*Markov decision process*). Základní myšlenkou mnoha algoritmů posilovaného učení je získat ohodnocení *Q(s,a)* na základě stavu *s* a akce *a* pomocí *Bellmanovi rovnice*. K získání tohoto ohodnocení se většinou používá lineární aproximace, která je ale v mnoha případech nedostatečná, proto je vhodné lineární aproximátor nahradit umělou neuronovou sítí. [8]

2.4.3 Vstupní data

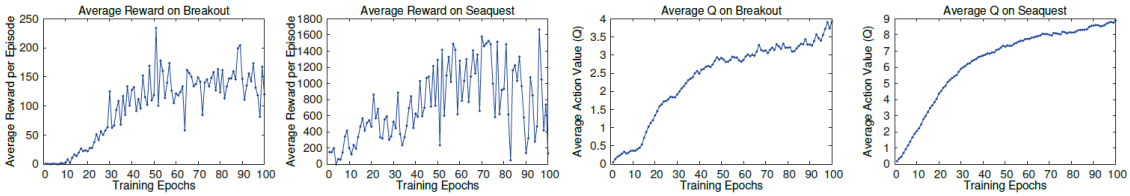
Každý výstupní obraz Atari 2600 emulátoru má 210 x 160 pixelů a je tvořený pomocí 128 barev. Přímé zpracování takového obrazu je výpočetně náročné. Z toho důvodu byl zvolen takový preprocessing dat, který vede ke snížení barevné hloubky na odstíny šedi a následného zmenšení obrazu. Obraz byl zmenšen na velikost 110 x 84 pixelů a jako vstup byla vybrána oblast 84 x 84 pixelů, která odpovídá samotnému hernímu prostoru. Finální výřez byl aplikován z důvodu použití 2D konvolucí, běžících na GPU, které očekávají čtvercový vstup. Použita byla taková architektura, kde vstupem je stavová reprezentace (4 poslední herní obrazy) a výstupem jsou separované neurony odpovídající každé možné akci v daném stavu. Jinými slovy, výstupem je ohodnocení *Q(s,a)* každé možné akce *a* ve stavu *s*. [8]

2.4.4 Architektura sítě

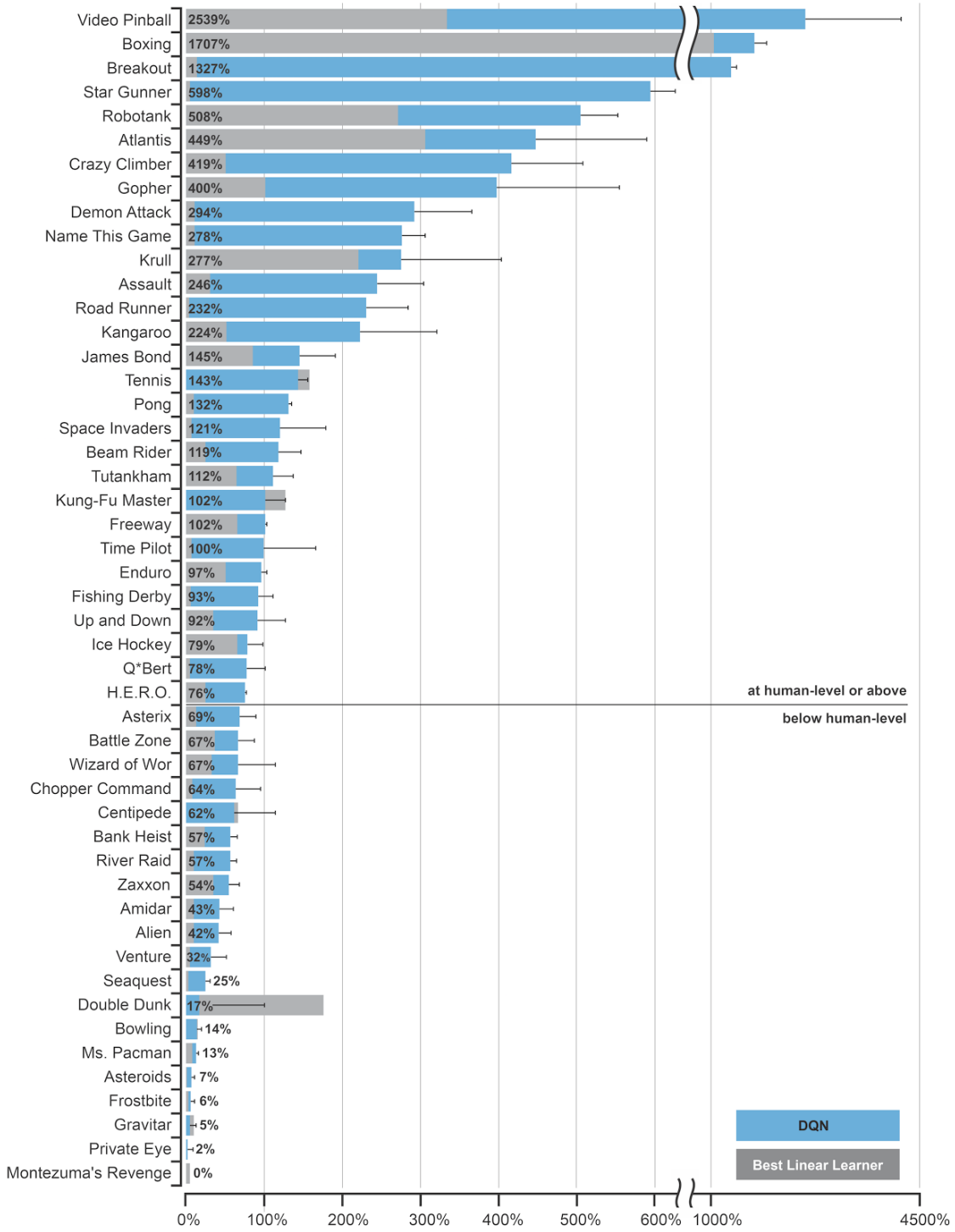
Vstupem neuronové sítě je konvoluční vrstva skládající se z 16 konvolučních filtrů o velikosti 8 x 8 a posunem 4 pixely. Následuje druhá konvoluční vrstva, kterou tvoří 32 filtrů. Tyto filtry mají velikosti 4 x 4 a posunují se o 2 pixely. Obě tyto vrstvy jsou aktivovány pomocí ReLU aktivační funkce. Poslední plně propojená vrstva obsahuje 256 neuronů a aktivační funkce je stejná, jako u vrstev předešlých. Výstupní vrstva obsahuje 4 až 18 neuronů, které odpovídají možným akcím podle konkrétní hry. [8]

2.4.5 Výsledky

Jako ohodnocení bylo použito průměrné herní skóre za epochu. Ukázalo se, že tento přístup není přímo ideální, protože i malá změna ve vahách neuronové sítě, vede k velkým změnám na výstupu. Z toho důvodu se grafy na obr. 7a jeví jako velmi zašumělé. Opakem je průměrné ohodnocení budoucích akcí na obr. 7b, kde je progres v obou případech znatelný. Grafy byly vytvořeny při trénování agenta na hře *Breakout* a *Seaquest*. Nakonec na obr. 8 je graf vyjadřující porovnání výkonu Deep Q Learning agenta s běžnou lineárně pracující umělou inteligencí. [8]



Obr. 7a) Průměrné skóre za epochu [8], 7b) průměrné ohodnocení akcí za epochu [8]



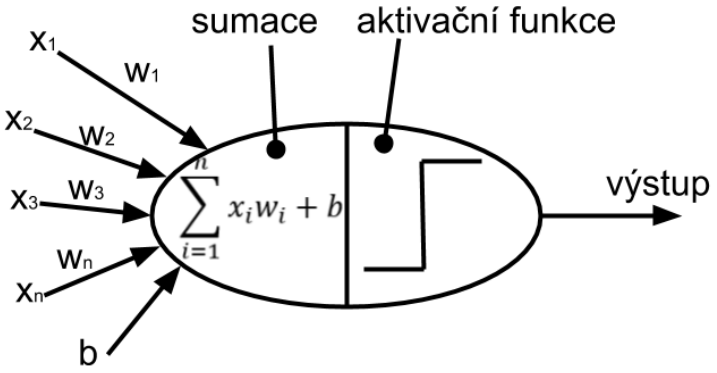
Obr. 8) Výkon DQN agenta [14]

3 UMĚLÉ NEURONOVÉ SÍTĚ

Lidský mozek lze chápat jako velmi komplexní biologickou neuronovou síť, skládající se ze vzájemně propojených neuronů. Tyto neurony přenáší elektrické signály, které tvoří složité vzory. Mezi hlavní části biologického neuronu patří dendrity, které tyto elektrické signály do neuronů přenáší. Dostanou-li se tyto elektrické signály na dostatečnou úroveň, dojde k aktivaci neuronu a neuron vyšle nervový vzruch, který se šíří prostřednictvím axonu až na synaptické zakončení neuronu. Pomocí těchto zakončení se signál šíří do dalších neuronů. Následující text čerpá ze zdroje [15].

První koncepční model umělé neuronové sítě představili roku 1943 Warren S. McCulloch a Walter Pitts [16]. Tento koncept pohlížel na umělý neuron jako na jednu buňku, která je součástí umělé neuronové sítě. Na základě vstupních signálů dojde k určitému vnitřnímu zpracování a výsledkem je výstupní signál. Obecně lze říct, že neuronové sítě jsou nejčastěji používány pro takové aplikace, které jsou jednoduché pro člověka, ale složité pro stroj. Tyto aplikace se obecně nazývají rozpoznávání vzorů a zahrnují takové případy, jako rozpoznávání objektů, rozpoznávání přirozených jazyků, detekce anomálií, nebo hraní počítačových her.

Klíčová schopnost umělých neuronových sítí je schopnost učit se. Stejné neuronové sítě lze pomocí učících algoritmů adaptovat na různé problémy. Učení probíhá pomocí tzv. upravování vah. Mezi neurony v jednotlivých vrstvách existuje spojení, kterým se šíří signál a tento signál je upraven pomocí odpovídající váhy tohoto spojení. Na obr. 17 je vyobrazen umělý neuron, kde w značí váhu odpovídajícího spojení, x vstupní signál a b bias. Učení neuronových sítí nejčastěji probíhá jednou z následujících možností. Učení s učitelem, učení bez učitele a posilované učení. V následující kapitole jsou tyto způsoby stručně vysvětleny.



Obr. 9: Umělý neuron, upraveno dle [17]

3.1 Učení umělých neuronových sítí

3.1.1 Učení s učitelem

Učení s učitelem probíhá pomocí trénovacích setů, kde se každý set skládá ze vstupního vzoru a správného výstupu aktivovaných výstupních neuronů. Pro každý trénovací set, kterým je umělá neuronová síť aktivována, je výstupní hodnota (výstupní vzor) porovnána se správným řešením. Na základě tohoto rozdílu jsou váhy mezi neurony upraveny. Nejčastější metoda úpravy vah je metoda Backpropagation, která bude vysvětlena v další části této kapitoly. Cílem učení není pouze nastavit váhy tak, aby síť byla schopna sdružit naučené vstupní a výstupní vzory, ale aby síť měla uspokojivé výsledky i v souvislosti s neznámými vstupními vzory stejné kategorie. Této vlastnosti se říká zobecnění. [18]

3.1.2 Posilované učení

Posilované učení je typ učení s učitelem. V průběhu posilovaného učení dojde po ukončení určité sekvence k ohodnocení neuronové sítě. Toto ohodnocení může být jak binární, tak i reálná hodnota. Pomocí binárního ohodnocení lze určit, zda byly výsledky dobré, nebo špatné. Kdežto ohodnocení číslem reálným vypovídá o tom, jak moc dobré, nebo špatné výsledky byly. Trénovací set je obdobný trénovacímu setu v předchozím případě, kdy je k dispozici pouze vstupní vzor, ale klíčové je v tomto případě ohodnocení výsledků získaných v průběhu řešení určitého problému. [18]

Příkladem takového problému může být hraní počítačových her, kdy na základě dosaženého herního skóre lze výsledek neuronové sítě ohodnotit.

3.1.3 Učení bez učitele

Učení bez učitele je z biologického hlediska nejrozšířenější způsob trénování neuronových sítí. Tento způsob není ale vhodný pro řešení všech problémů. Učení probíhá obdobně jako v předchozím případě, ale trénovací set se skládá pouze ze vstupních vzorů. Úkolem neuronové sítě je identifikovat stejné vzory a tyto vzory přiřadit do stejné kategorie neboli je klasifikovat. Mezi nejznámější příklady samoučících neuronových sítí patří *samo-organizační mapy* a *teorie adaptivní rezonance*. [18]

3.2 Backpropagation metoda

Backpropagation metoda patří k nejrozšířenějším metodám upravování vah. Tato metoda je využívána v optimalizačních algoritmech založených na optimalizaci pomocí gradientního sestupu (*gradient descent*) a lze ji aplikovat na jednoduché, ale i hluboké neuronové sítě. Základem této metody je chybová funkce výstupu sítě, kterou se snažíme pomocí úpravy jednotlivých vah minimalizovat. Za předpokladu, že požadovaný výstup neuronové sítě i-tého vstupního vektoru $\mathbf{x}_i$ značíme y_i a existuje dopředným průchodem získaný aktuální výstup $\hat{y}_i$, lze spočítat chybovou funkci E (*loss function*). Pomocí zpětné

propagace dojde k upravení jednotlivých vah podle vztahu (4). Přičemž a_j^k je výstupní hodnota odpovídajícího neuronu a w_{ij}^k je váha mezi dvěma odpovídajícími neurony. [19]

$$E = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$
$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k} \tag{4}$$

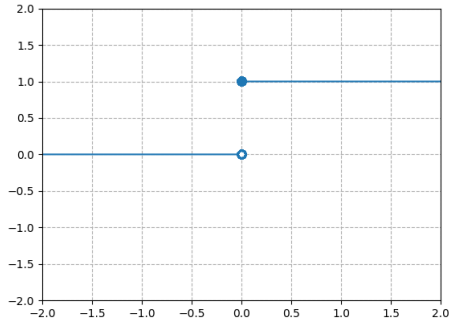
3.3 Aktivační funkce

Aktivační funkce je velmi důležitá část umělých neuronových sítí, pomocí které se rozhodne, zda se daný neuron aktivuje, nebo ne. Aktivační funkce většinou představuje nelineární transformace signálu vstupujícího do neuronu, jejímž výsledkem je signál, který může vstupovat do dalších umělých neuronů, nebo také být signálem výstupním [20]. Následuje popis vybraných aktivačních funkcí.

3.3.1 Jednotkový skok

Jednotkový skok (binary step) je velmi jednoduchá aktivační funkce. Vhodné použití této funkce je jako aktivace binárního klasifikátoru, kde jsou pouze dva možné stavy, *ANO*, nebo *NE*. Derivace této funkce je nula, tudíž zde metoda Backpropagation nepřichází v úvahu. Binární skok je definován pomocí vztahu (5) a znázorněn na obr. 10. [20]

$$f(x) = \begin{cases} 1, x \geq 0 \\ 0 \end{cases} \tag{5}$$

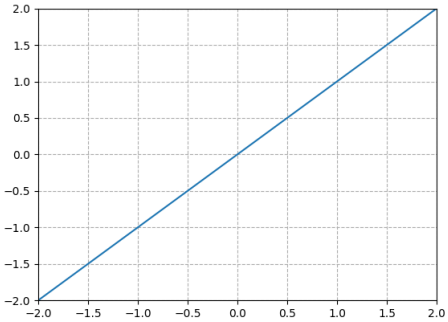


Obr. 10) Jednotkový skok

3.3.2 Lineární funkce

Lineární (linear function) je definovaná pomocí vztahu (6) a znázorněna na obr. 11. Aktivační hodnota neuronu je přímo úměrná vstupnímu signálu. Derivace lineární funkce je konstanta, z toho důvodu tato aktivační funkce také není vhodná pro Backpropagation metodu. [20]

$$f(x) = a\,x \tag{6}$$

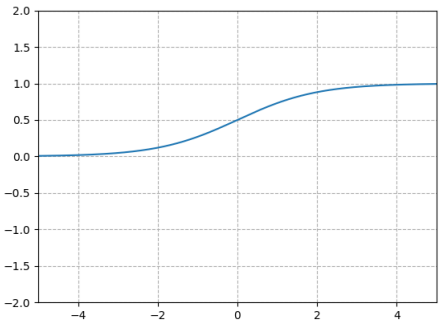


Obr. 11) Lineární funkce

3.3.3 Sigmoida

Sigmoida je široce užívaná aktivační funkce. Tato funkce je typická svojí nelinearitou. Funkce je definovaná vztahem (7) a znázorněna na obr. 12. Nevýhodou této funkce je rozsah výstupních hodnot od 0 do 1, takže výstupní signál je stále kladný. Tuto nevýhodu lze odstranit pomocí posunutí funkce ve svislém směru, čímž získáme hyperbolický tangens. [20]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

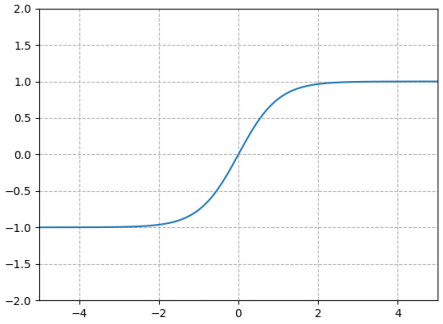


Obr. 12) Sigmoida

3.3.4 Hyperbolický tangens

Hyperbolický tangens (tanh) má stejné vlastnosti jako Sigmoida, ale dosahované výstupní hodnoty jsou symetrické okolo počátku souřadného systému. Výstupem mohou tedy být hodnoty mezi -1 a 1. Hyperbolický tangens je definován vztahem (8) a vyobrazen na obr. 13. [20]

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{8}$$

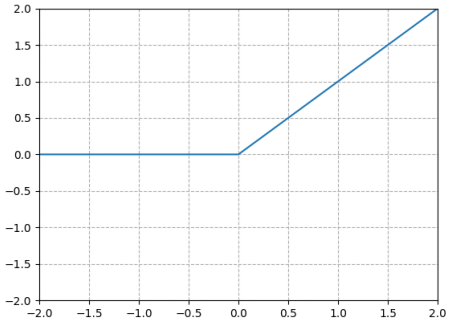


Obr. 13) Hyperbolický tangens

3.3.5 Rektifikovaná lineární jednotka

Rektifikovaná lineární jednotka, zkráceně nazývaná ReLU, je v dnešní době nejrozšířenější aktivační funkce. Největší výhodou této funkce je, že nedochází k aktivování všech neuronů ve stejnou chvíli, což má za následek větší efektivitu neuronové sítě a menší nároky na výpočetní výkon. ReLU je definovaná pomocí vztahu (9) a graficky znázorněna na obr. 14. [20]

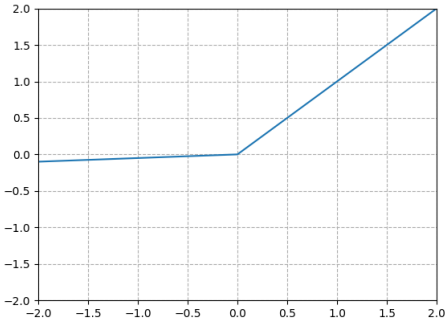
$$f(x) = \max(0, x) \tag{9}$$



Obr. 14) ReLU

Z důvodu nulového výstupu při záporných hodnotách se mohou v síti objevit tzv. mrtvé neurony, které v některých případech nemusí být přínosem. Z tohoto důvodu lze ReLU zaměnit za Leaky ReLU, která díky své definici (10) zamezí vzniku mrtvých neuronů. Leaky ReLU je vyobrazena na obr. 15. [20]

$$f(x) = ax, x < 0 \tag{10}$$
$$f(x) = x, x \geq 0$$



Obr. 15) Leaky ReLU

3.3.6 Softmax

Aktivační funkce Softmax se nejčastěji využívá jako aktivace výstupních neuronů při řešení klasifikačních úkonů. Na rozdíl od Sigmoidy, která je vhodná pouze pro binární klasifikaci, dokáže funkce Softmax klasifikovat více tříd. Přičemž výstupem je pravděpodobnost, že vstupní vzor spadá do určité třídy. Softmax funkce je definovaná pomocí vztahu (11), kde i je index právě počítané třídy a k je počet tříd. [21]

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}, i = 0,1,2, \dots k \tag{11}$$

3.4 Typy umělých neuronových sítí

Rozeznáváme několik typů neuronových sítí, které se liší architekturou a funkčností. Tato podkapitola je věnována popisu nejrozšířenějších typů.

3.4.1 Dopředná neuronová síť

Dopředná neuronová síť (feedforward neural network, FFNN), jakožto jeden z prvních modelů umělé neuronové sítě, je na rozdíl od ostatních typů sítí poměrně jednoduchá. Spojení mezi jednotlivými neurony netvoří žádné cykly a typickou vlastností této sítě je, že informace cestuje pouze jedním – dopředným směrem. Topologicky jde o acyklickou strukturu grafu, kde jsou všechny neurony z *n*-té vrstvy propojeny se všemi neurony z *n*+1 vrstvy. [22]

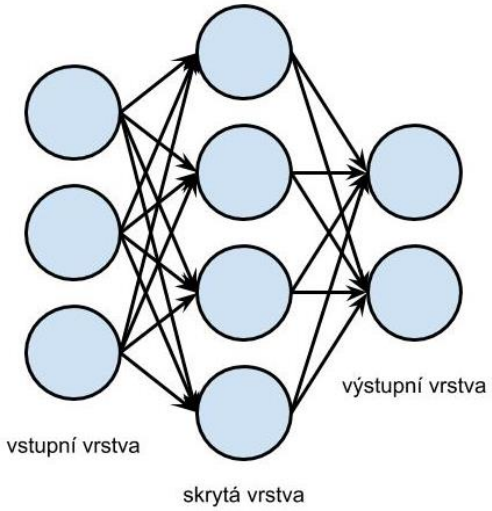
Single-layer perceptron

Nejjednodušším typem dopředné neuronové sítě je single-layer perceptron, který se skládá pouze z jedné vrstvy neuronů, přičemž výstupní vrstva obsahuje pouze 1 neuron. Single-layer perceptron je také znám jako lineární klasifikátor, protože je vhodný pouze na lineárně separovatelné vzory, což jsou takové, jejichž datová množina (dataset) lze rozdělit pomocí lineární hranice. [22]

Multi-layer perceptron

Dalším navazujícím typem dopředné neuronové sítě je multi-layer perceptron. Tento typ neuronové sítě se skládá z více single-layer perceptronů a typickou vlastností je schopnost

spočítat nelineárně separovatelné vzory. Multi-layer perceptron se skládá mimo vstupní a výstupní vrstvy minimálně z jedné vrstvy skryté. Na obr. 16 je znázorněn multi-layer perceptron tvořen třemi vstupními neurony, jednou skrytou vrstvou se čtyřmi neurony a výstupní vrstvou se dvěma neurony. [22]

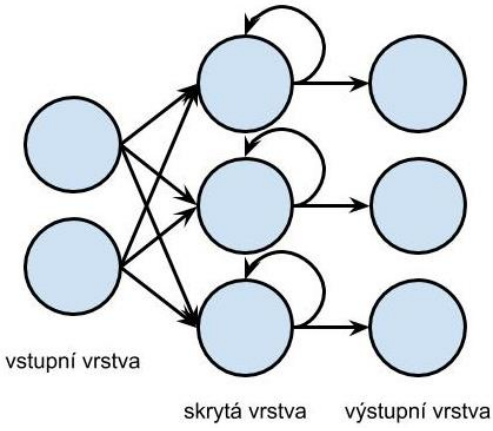


Obr. 16) Příklad dopředné neuronové sítě (Multi-layer perceptron), upraveno podle [23]

3.4.2 Rekurentní neuronová síť

Pro rekurentní neuronové sítě jsou charakteristické například uzavřené cykly mezi neurony. Topologicky jde o cyklickou strukturu grafu. Stejně, jako u předchozích sítí, mají rekurentní sítě skryté vrstvy, ale v tomto případě výstup skrytého neuronu ústí zpět sám do sebe, znázorněno na obr. 17. Lze říct, že skryté neurony mají paměť, ve které se uchová vnitřní stav daného časového okamžiku a tento stav je použit jako vstup do skryté vrstvy v jiný časový okamžik. Jinými slovy, aktuální výstup této neuronové sítě je závislý na předchozích stavech. Častým typem těchto sítí jsou *Long Short-Term Memory*. [24]

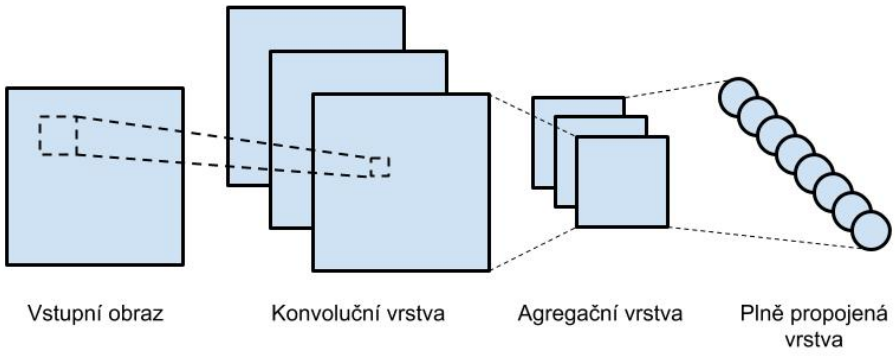
Rekurentní sítě se využívají v takových aplikacích, jako je rozpoznávání řeči, hudební kompozice, rozpoznávání gramatiky, nebo predikce lidského chování.



Obr. 17) Příklad rekurentní neuronové sítě, upraveno podle [23]

3.4.3 Konvoluční neuronová síť

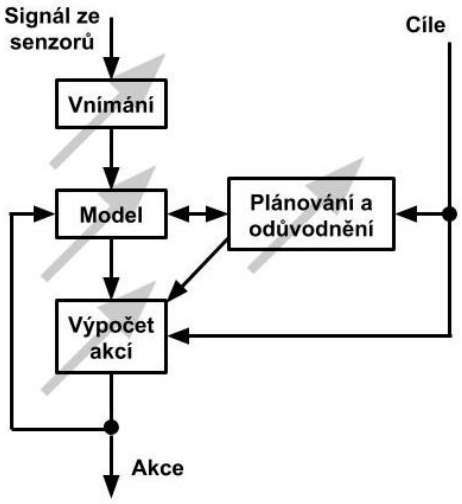
Konvoluční neuronové sítě se primárně zaměřují na zpracování obrazu. Na rozdíl od dopředných neuronových sítí nedochází ke zpracování jednotlivých pixelů zvlášť, ale pixely jsou zpracovávány ve skupinách, protože vzájemná pozice pixelů je důležitá. Pro konvoluční síť jsou charakteristické následující vrstvy. Konvoluční vrstva, agregační vrstva a poslední plně propojená klasifikační vrstva. Architektura této sítě je znázorněna na obr. 18. Konvoluční neuronové sítě jsou v dnešní době velmi populární a jejich nejčastější aplikací je rozpoznávání obrazu a objektů. [25]



Obr. 18) Koncept konvoluční neuronové sítě, upraveno podle [26]

4 STROJOVÉ UČENÍ

Učení strojů (*machine learning*) může probíhat tak, že pokaždé, když dojde k naučení stroje, tak se pozmění jeho struktura, program, nebo data. Tyto změny proběhnou za účelem zlepšení budoucích výsledků, které má za úkol stroj vykonávat. Změny mohou nastat v již vytvořených a nějakým způsobem fungujících systémech, nebo může dojít k vytvoření nového systému. Strojové učení je úzce spjaté s umělou inteligencí. Na následujícím obrázku je znázorněn typický model umělé inteligence a autor zde poukazuje na to, že na jednotlivý subsystém je třeba aplikovat jinou formu strojového učení. [19]



Obr. 19) Obecný model umělé inteligence, upraveno dle [19]

4.1 Knihovny pro strojové učení

V současnosti vzniká velký zájem o strojové učení a s tímto zájmem vznikají také nové, více, či méně kvalitní knihovny, určené pro podporu tohoto typu učení.

4.1.1 Keras

Keras se řadí mezi populární open source knihovny pro tvorbu a práci s neuronovými sítěmi. Tato kapitola převážně čerpá z oficiální webové dokumentace [27]. Při vývoji knihovny Keras byl kladen důraz na možnost rychlého experimentování, kde čas mezi prvotní myšlenkou a následnou realizací je minimální. První verze byla vydána 27. března 2015 a za stabilní verzi se považuje verze 2.1.6 z 23. dubna 2018. Celý vývoj probíhá v programovacím jazyce Python. Keras sám o sobě neuronové sítě nevytváří a pro svoji funkčnost potřebuje další knihovnu v pozadí, na jejímž pomyslném vrcholu pracuje. Těmto knihovnám se říká *backend* a jako backend je možno použít *TensorFlow*, *Theano*,

nebo *CNTK*. Další z možností je také neoficiálně podporovaný *MXNet*. Každé z těchto knihoven je věnován prostor v této podkapitole.

Mezi hlavní výhody Kerasu patří možnost rychlého a jednoduchého prototypování. Této vlastnosti je dosaženo uživatelsky příjemným přístupem, modularitou a možností použít fungující kód na více zařízeních. Další výhodou je, že Keras mimo klasické dopředné neuronové sítě podporuje sítě konvoluční, dále sítě rekurentní a jejich následné kombinace. Poslední vlastností je podpora práce na *CPU* a *GPU*, což je z důvodu neustále se zvyšujících nároků na výpočetní výkon velmi důležitý atribut.

Model neuronové sítě je v Kerasu považován za sekvenci, nebo graf, skládající se z jednotlivých modulů. Za tyto moduly jsou považovány jednotlivé vrstvy neuronové sítě, aktivační funkce, hodnotící funkce, optimizátory atd. Moduly je možno kombinovat s co nejmenším omezením.

Základní datová struktura Kerasu se nazývá *model* a nejjednodušší způsob tvorby neuronových sítí je použít *Sequential model*. Sequential model funguje jako lineární zásobník jednotlivých vrstev, zadáním jednoduchých příkazů lze vrstvy přidávat. Algoritmus 1 obsahuje ukázku zdrojového kódu, kde pomocí Sequential modelu lze vytvořit jednoduchou neuronovou síť s jednou vstupní, jednou skrytou a jednou výstupní vrstvou.

Algoritmus 1:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(units=10, input_dim=4))
model.add(Activation("relu"))
model.add(Dense(2, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer="adam",
metrics=['accuracy'])
```

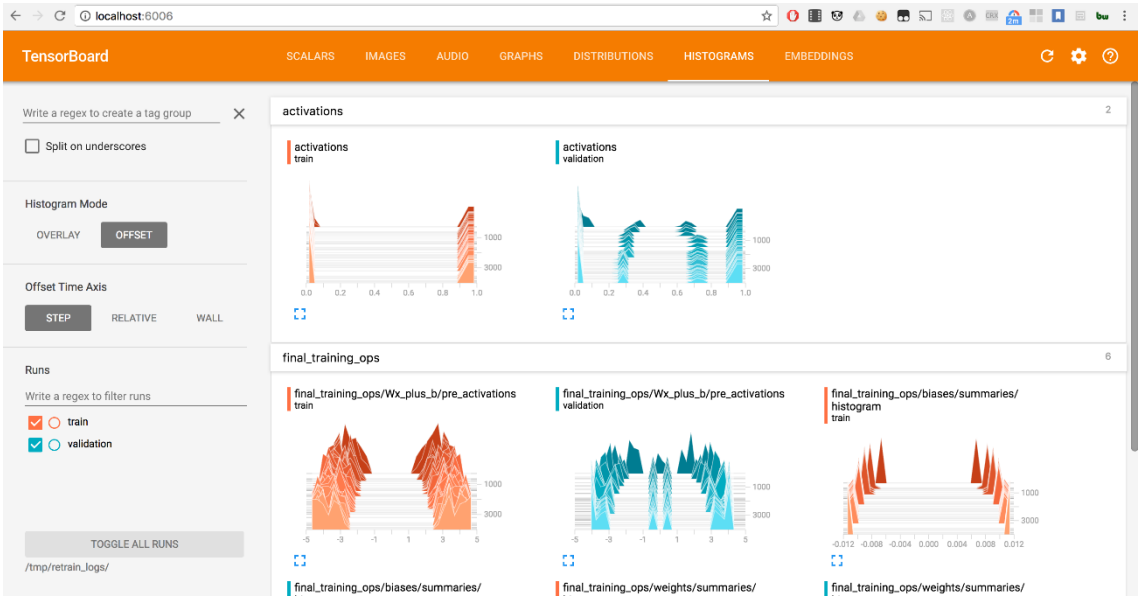
Dalším způsobem tvorby neuronových sítí je použit *Functional API*. Tyto API dovolují tvorbu složitějších modelů, které mohou obsahovat více vstupové a výstupové modely, modely se sdílenými vrstvami neuronů a další.

4.1.2 TensorFlow

V současné době je Tensorflow nejpopulárnější open source knihovna podporující vysoce výkonné výpočty. Tensorflow využívají společnosti jako Google, Uber, Airbnb, Nvidia, nebo Dropbox a hlavní doménou této knihovny jsou hluboké neuronové sítě. [28]

Tensorflow podporuje výpočty na jednom, nebo více CPU, GPU a také na *TPU*. Tyto výpočty je možné provádět na stolním PC, serveru, nebo třeba mobilním zařízení. Všechny výpočty jsou vyjádřeny jako datový tok grafovou strukturou. Grafové struktury je možno vizualizovat pomocí zabudovaného nástroje *TensorBoard* (obr. 20), který mimo vizualizaci dovoluje debugování a optimizaci vytvořených sítí. [29]

Grafová struktura se skládá z uzlů reprezentujících matematické operace a z hran, kde každá hrana je nositelkou datových struktur, kterým se říká *tenzory*. Výstup jedné operace se stává vstupem operace jiné.

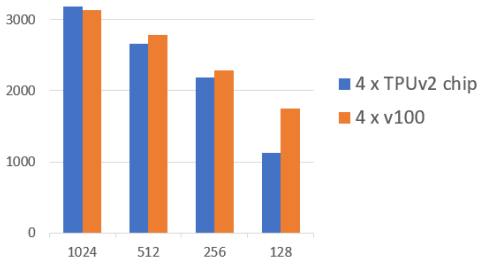


Obr. 20) TensorBoard [30]

TPU

Tensor processing unit (obr. 21) je speciálně vyvinutý chip společností Google pro efektivní a velmi výkonnou práci s Tensorflow. TPU byla navržena jako přímá konkurence grafických karet za účelem snížení ceny výpočtů. Společnost Google nabízí TPU v rámci jejich cloudové nabídky. [31]

Obr. 21) TPU [31]



Obr. 22) Porovnání TPU a GPU [32]

Na obr. 22 je příklad porovnání *TPUv2* vůči *GPU Nvidia V100* při úloze zpracování obrazu. Testování probíhalo použitím neuronové sítě *ResNet (Residual Networks)*. Uvedené výsledky jsou založené pouze na tomto typu testů. V jiných testech se výkon TPUv2 a GPU Nvidia V100 bude pravděpodobně lišit. [32]

4.1.3 Theano

Theano je open source knihovna pro programovací jazyk *Python* vyvíjena od roku 2007. Theano ve své podstatě funguje jako kompilátor matematických výrazů, které kompiluje do více efektivní podoby. Zefektivnění je dosaženo využitím knihoven jako je *NumPy*, *BLAS*, nebo programovací jazyk *C++*. Výhodou této knihovny jsou optimalizátory, které urychlují a optimalizují výpočty pro aktuálně používaný hardware. [33], [34]

4.1.4 Microsoft Cognitive Toolkit

Microsoft Cognitive Toolkit, zkráceně *CNTK*, je open source knihovna určená k vývoji a učení hlubokých neuronových sítí, vyvíjená od roku 2016 společností Microsoft. Neuronová síť je popsána jako sekvence výpočetních kroků v orientovaném grafu. Listové uzly grafu odpovídají vstupním hodnotám a ostatní uzly vyjadřují maticové operace aplikované na tyto vstupy. Pomocí této knihovny lze realizovat feedforward, konvoluční a rekurentní neuronové sítě. [35] [36]

Výhodou této knihovny je podpora formátu zvaného *Open Neural Network Exchange*, zkráceně *ONNX*, díky kterému je možné sdílet vytvořené modely mezi více knihovnami pro hluboké učení. [35], [36]

4.1.5 Caffe2

Caffe2 je poměrně jednoduchá knihovna zaměřená na hluboké učení. Na rozdíl od předchozích knihoven se tato knihovna nezaměřuje na vytváření exotických neuronových sítí, nýbrž na konvenční typy. Výhodou je velká podpora mobilních zařízení, možnost psaní kódu v programovacím jazyku *Python* a *C++* a také podpora ONNX formátu. [37]

4.1.6 MXNet

Apache MXNet je framework pro hluboké učení, který podporuje programovací jazyky jako *Python*, *R*, *Scala*, *Julia*, nebo *C++*. Mezi hlavní výhody patří efektivní práce s pamětí, podpora chytrých zařízení a dynamický plánovač, starající se o paralelizmus za běhu programu. MXNet je široce podporován cloudovou službou *Amazon Web Services*. [38]

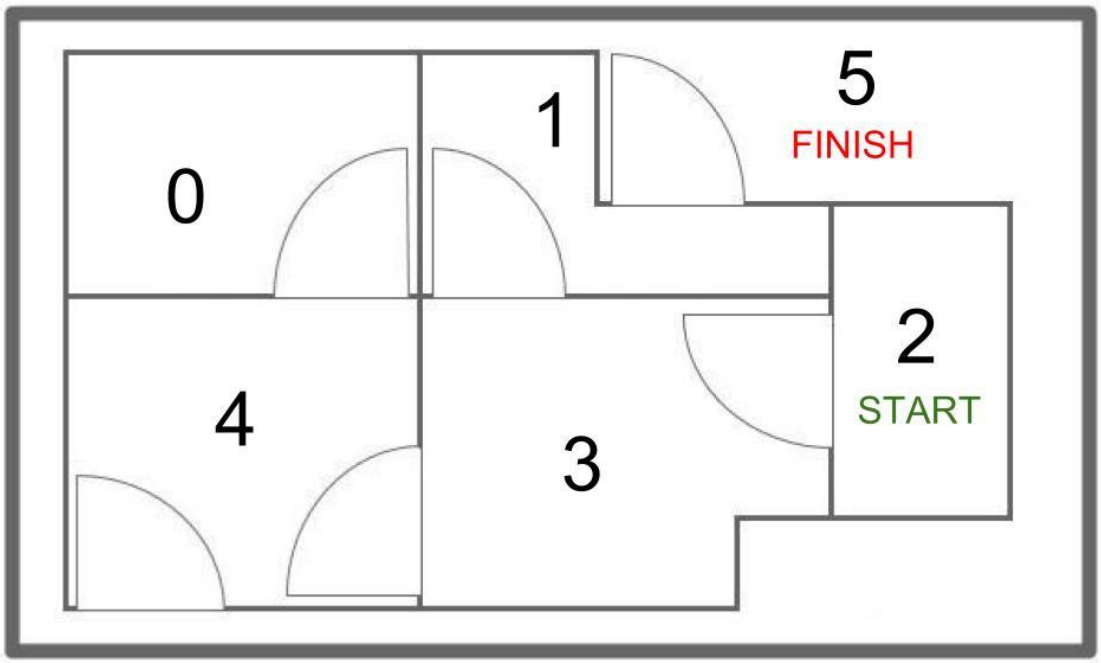
4.1.7 PyTorch

PyTorch je knihovna vytvářená v programovacím jazyce *Python* pro hluboké učení. Vývoj probíhá od roku 2016 a podílí se na něm společnosti jako je *Facebook*, *Nvidia*, nebo *Uber*. Pro PyTorch je typická metodika *define-by-run*. Každý řádek kódu zodpovědný za vytváření grafu definuje určité komponenty, na které je možné provádět výpočty ještě před dokončením celého grafu. Toto má za následek poměrně široké možnosti ladění a pracování s neuronovou sítí za běhu. Dalšími typickými vlastnostmi PyTorch je práce na CPU, nebo GPU a podpora formátu ONNX. [39], [40]

4.2 Q learning

Q learning (Q učení) je typ posilovaného učení pocházející z roku 1989 [41]. Tento typ učení je inspirován výcvikem psa, kdy na základě správného chovaní je pes oceněn a na základě špatného chování potrestán. *Agent* (umělá inteligence) pozorováním prostředí získá *stav* (*state*). V tomto stavu agent zvolí *akci*, díky které přejde do stavu jiného. Každá akce má potenciální hodnotu mající za následek ohodnocení. [42]

Na následujícím obrázku je dům s označenými místnostmi (0 až 5) a robotem, nacházejícím se ve startovní místnosti (2). Cílem robota je dostat se do cílové oblasti (5). Pro každou místnost, která není cílová zvolíme ohodnocení 0 a pro oblast cílovou ohodnocení 100. Ohodnocení -1 je typické pro nedosažitelnou akci z určitého stavu. Tento způsob hodnocení může být znázorněn pomocí ohodnocovací tabulky *R* na obr. 24. [43]



Obr. 23) Příklad pro Q learning, upraveno podle [43]

		akce						
		stav	0	1	2	3	4	5
		0	-1	-1	-1	-1	0	-1
R	=	1	-1	-1	-1	0	-1	100
		2	-1	-1	-1	0	-1	-1
		3	-1	0	0	-1	0	-1
		4	0	-1	-1	0	-1	100
		5	-1	0	-1	-1	0	100

Obr. 24) Ohodnocovací tabulka R, upraveno podle [43]

Q učení spočívá v tom, že agent nezná tabulku R, ale tabulku Q, která je na začátku algoritmu inicializována s nulami na všech pozicích. Prostřednictvím zkoumání a zkoušení se tabulka upravuje a hodnotami se přibližuje k tabulce R. Hodnoty této tabulky se upravují pomocí *Bellmanovi* rovnice (12). V tomto vztahu s reprezentuje aktuální stav, a akci v tomto stavu, s´ nový stav na základě akce a, r ohodnocení a γ slevový faktor. [43]

$$Q(s,a) = r + \gamma(max\, Q(s´,a´)) \tag{12}$$

Tento algoritmus je popsán pomocí následujícího pseudokódu. [44]

1. Inicializuj Q tabulku s nulami
2. Vyber náhodný počáteční stav
3. FOR EACH ITERATION (posloupnost akcí, vedoucí k cílovému stavu):
   DO WHILE stav není cílový:
   a. V aktuálním stavu (s) vyber náhodnou akci (a)
   b. Přejdi do nového stavu ($s´$) na základě vykonané akce (a)
   c. Pro všechny možné akce ze stavu ($s´$) vyber jednu s nejvyšší Q hodnotou
   d. Aktualizuj Q tabulku použitím rovnice (12)
   e. Nastav stav ($s´$) jako (s)
   f. Zkontroluj, jestli stav (s) není cílový

4.3 Deep Q learning

V roce 2013 společnost *DeepMind* uveřejnila algoritmus, pomocí kterého se umělá inteligence sama naučila hrát Atari 2600 videohry. Jednalo se o první umělou inteligenci schopnou adaptovat se na různá prostředí (úkoly). DeepMind následně koupila společnost Google a vyvinula umělou inteligenci schopnou hrát dalších 49 her a ve více jak polovině, dosahovat lepších výsledků než člověk (obr. 8.). [14], [45]

Základním stavebním kamenem Deep Q learning je Q hodnota vyjádřená pomocí konvoluční neuronové sítě. Vstupem této sítě bývá celá herní obrazovka a navrhovaná akce. Neuronová sít dopředným průchodem toto vyhodnotí a výstupem je Q hodnota. Druhou možností je jako vstup použít pouze herní obrazovku a získat Q hodnotu pro každou z možných akcí. Výhodou je potřeba pouze jednoho průchodu celou sítí. Tyto dva způsoby jsou schematicky znázorněny na obr. 25. Pro zachování herní dynamiky se obvykle jako vstup dávají 4 po sobě jdoucí snímky obrazovky. [45]

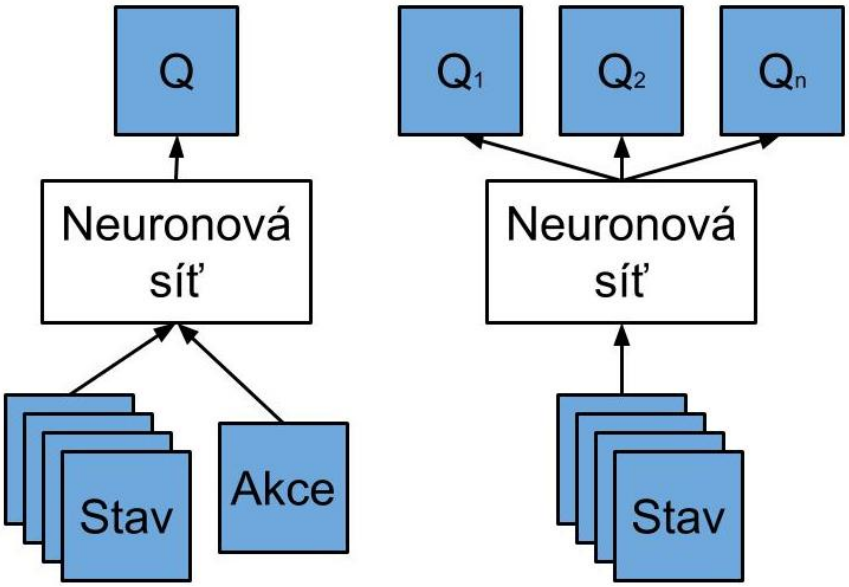
K optimalizaci neuronové sítě je použita hodnotící funkce pro výpočet chyby (13).

$$L = \frac{1}{2}(r + \gamma maxQ(s´,a´) - Q(s,a))^2 \tag{13}$$

Úprava vah neuronové sítě probíhá pomocí následujícího algoritmu. [45]

1. Pomocí dopředného průchodu získej Q hodnoty pro všechny akce (a) v aktuálním stavu (s)

2. V dalším stavu (s´) vykonej dopředný průchod a vyber maximální Q hodnotu pomocí $maxQ(s´, a´)$
3. Na základě této hodnoty urči $r + \gamma maxQ(s´, a´)$ a pomocí vztahu (13) vyjádři chybu
4. Zpětným průchodem (Backpropagation) aktualizuj váhy



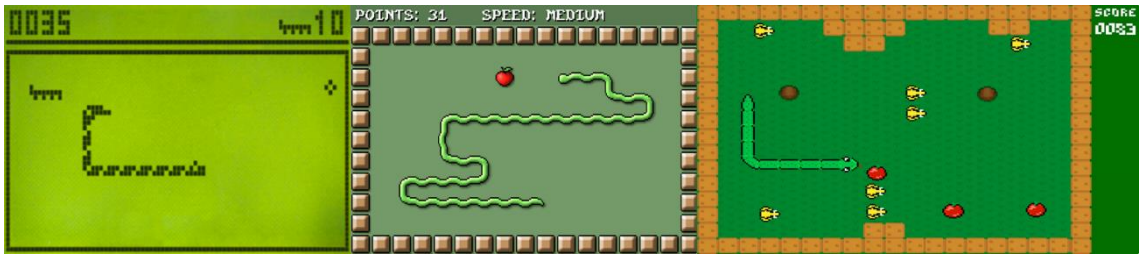
Obr. 25) Architektura pro Deep Q Learning, upraveno podle [45]

5 POPIS IMPLEMENTACE

Prvním cílem mé diplomové práce bylo navrhnout soutěžní hřiště pro umělou inteligenci. Hřiště mělo splňovat určité požadavky jako je poměrná jednoduchost, grafická nenáročnost a možnost hraní jednoho, nebo dvou hráčů. Nejvhodnějším kandidátem, který splňuje tyto předpoklady je hra *Snake*.

5.1 Snake

Snake je populární 2D hra, kde hráč ovládá hada, který má za úkol sníst co nejvíce náhodně se objevující potravy. Konec hry nastává ve chvíli, kdy hráč narazí do svého vlastního těla nebo do překážky. Tato hra existuje v mnoha verzích, které se liší jak pravidly, tak i grafickým zpracováním, ale základní koncept zůstává stejný. Některé příklady zpracování této hry jsou vyobrazeny na následujícím obrázku.

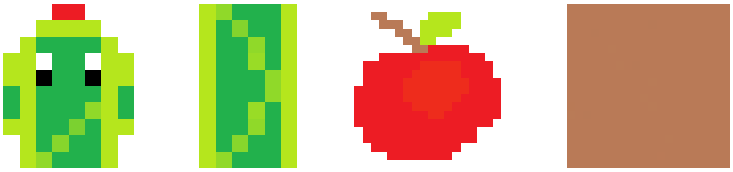


Obr. 26a) Snake II [46], 26b) Snake by Ilija Mihajlov [47], 26c) Snake Game by Banditmoviegames [48]

5.2 SnakeAI

Pro vytvoření vlastní verze této hry jsem použil programovací jazyk *Python* a jako základní framework *Pygame*. Samotnou hru tvoří herní pole o velikosti 32 x 24, které se skládá ze základních čtverců. K pohybu hada dochází právě po těchto čtvercích. Každý čtverec může obsahovat následující objekty, jejichž zpracování je možno vidět níže:

- hlava
- tělo
- jablko
- zeď



Obr. 27a) hlava, 27b tělo, 27c) jablko, 27d) zeď

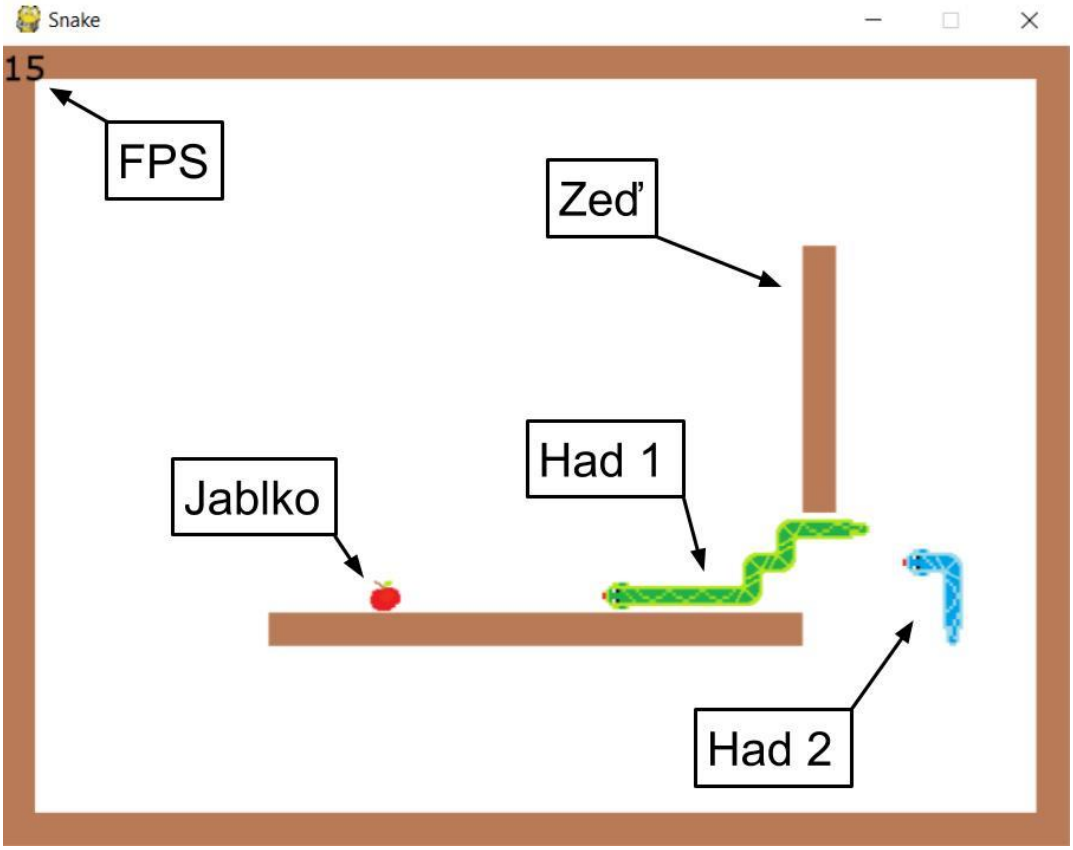
Pravidla hry jsou následující:

- hráč získá bod, pokud dojde na pole s jablkem
- jablko je následně vygenerováno na náhodné prázdné pozici
- pokud hráč narazí do zdi, tak prohrál
- pokud hráč narazí do svého těla, tak prohrál
- pokud hráč narazí do nepřátelského hada, tak prohrál
- pokud se hráči srazí hlavami, nastává remíza

Možnosti hry:

- hra pro jednoho, nebo dva hráče
- výběr barvy hada mezi zeleným a modrým
- pseudonáhodně generované řady zdí
- fullscreen

Následuje obrázek herní situace s popisky uživatelského rozhraní a herních objektů.

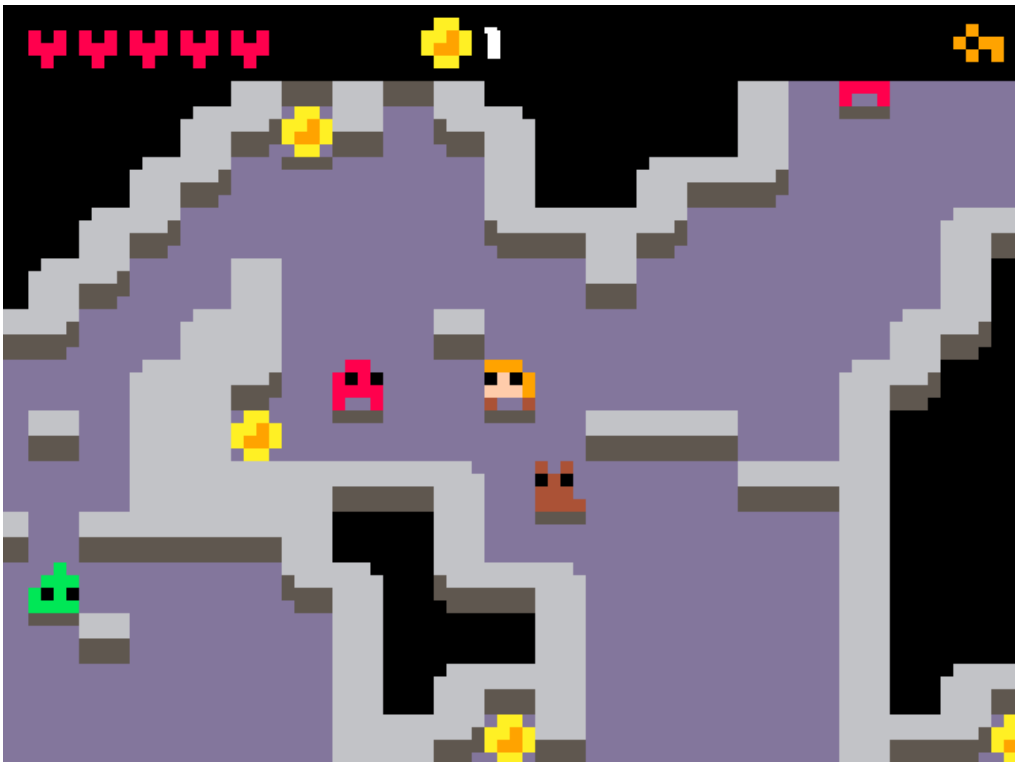


Obr. 28) SnakeAI s popisem

5.2.1 Pygame

Pygame je open source knihovna programovacího jazyka *Python* určená pro tvorbu multimediálních aplikací a her. Pomocí této knihovny lze pracovat s grafikou, zvukem, vstupními zařízeními a je kompatibilní s většinou platforem. Pygame je postavena

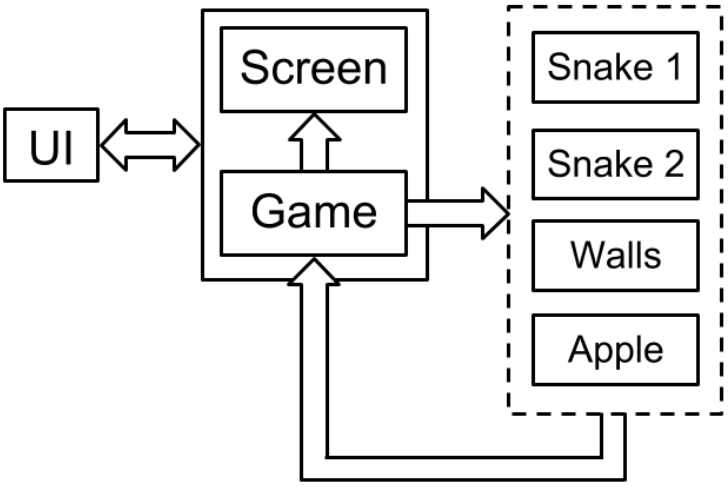
na knihovně *Simple DirectMedia Layer*, zkráceně *SDL*. Následující obrázek je příkladem aplikace, vytvořené pomocí této knihovny. [49]



Obr. 29) Roguelike-Pygame [50]

5.2.2 Herní objekty

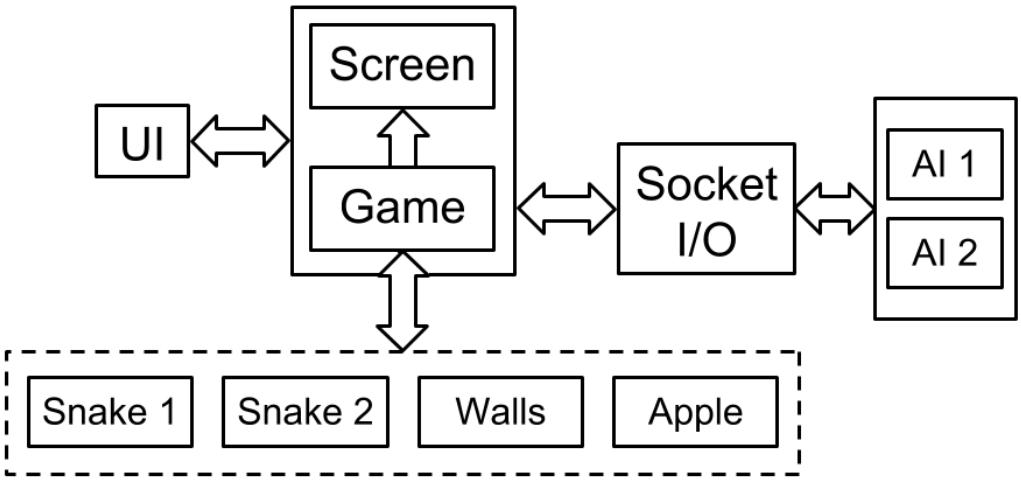
Hra je vytvořena s použitím objektově orientovaného programování. Základní třída se nazývá *Game* a úkolem této třídy je správa herních objektů a logika celé hry. Další třída odpovědná za herní model a vykreslování je třída s názvem *Screen*. Následují třídy *Snake*, *Apple* a *Walls*. Hierarchie těchto tříd je znázorněna na obrázku pod odstavcem.



Obr. 30) Hierarchie tříd

5.2.3 Soketové rozhraní

Možná implementace soketového rozhraní v této práci má dva důvody. Hra může běžet na jednom počítači a veškeré výpočty, související s umělými inteligencemi mohou běžet na počítači druhém. Druhou možností je připojení cizí umělé inteligence. Základní prvek obsahující statický herní popis je herní matice. Každý prvek matice odpovídá právě jednomu hracímu čtverci a je reprezentován pomocí sedmi bitů. Dynamický popis hry obsahuje směr, jakým se jeden, nebo oba hadi pohybují. Schéma je možno vidět na následujícím obrázku.



Obr. 31) Soketové rozhraní

Následuje bitová reprezentace možných stavů:

- prázdné pole [0, 0, 0, 0, 0, 0, 1]
- hlava hada [1, 0, 0, 0, 0, 0, 0]
- tělo hada [0, 1, 0, 0, 0, 0, 0]
- hlava nepřátelského hada [0, 0, 1, 0, 0, 0, 0]
- tělo nepřátelského hada [0, 0, 0, 1, 0, 0, 0]
- jablko [0, 0, 0, 0, 1, 0, 0]
- zeď [0, 0, 0, 0, 0, 1, 0]

Implementace soketového rozhraní se pomocí programovacího jazyka Python provádí pomocí knihovny *Socket*. Algoritmus 2 popisuje jednoduchý kód pro připojení k webové stránce www.google.com.

Algoritmus 2:

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
port = 80
host_ip = socket.gethostbyname('www.google.com')
s.connect((host_ip, port))
```

5.2.4 Herní algoritmus

Následuje ukázka herního scénáře, kdy hrají dvě umělé inteligence a jsou generovány náhodné pásy zdí.

Algoritmus 3:

```
import pygame
import sys
from items import Snake, Apple, Walls, Matrix
from manager import Game, Screen
import neuralNetwork as nn


# Game for two AI_Beta
def twoAI_Beta():
    # Game utilities
    screen = Screen(fullscreen_=False)

    # Game items
    snake1 = Snake(11, 20, "up", "green")
    snake2 = Snake(11, 10, "down", "blue")
    apple = Apple()
    walls = Walls()
    walls.makeRandomWalls()
    matrix = None

    # Game manager
    game = Game(snake1, snake2, apple, walls, matrix)

    clock = pygame.time.Clock()

    # Neural network
    model = nn.loadNeuralNetwork("NeuralNetwork_Beta.h5")
    suggestedDirection1 = 0
    suggestedDirection2 = 0

    while game.isGameRunning:
        for event in pygame.event.get():
            if event.type == pygame.QUIT or event.type == pygame.KEYDOWN \
and event.key == pygame.K_ESCAPE:
                game.isGameRunning = False
                sys.exit()

        # Control
        snake1.turnSnake(suggestedDirection1)
        snake2.turnSnake(suggestedDirection2)

        # Logic
        game.moveBothSnakes()
        game.checkSnakeCollision()
        game.collisionInference()
        game.checkResults()
        if snake1.isDead or snake2.isDead:
            break
```

```
        # Graphic
        screen.drawBackground()
        screen.drawAppleOnScreen(apple)
        screen.drawWallsOnScreen(walls)
        screen.drawSnakeOnScreen(snake1)
        screen.drawSnakeOnScreen(snake2)
        screen.showFPS()
        pygame.display.flip()

        # Predict move
        suggestedDirection1 = nn.predictMove_Beta(snake1, snake2, walls, \
apple, model)
        suggestedDirection2 = nn.predictMove_Beta(snake2, snake1, walls, \
apple, model)

        # FPS limit
        clock.tick(16)
```

5.3 Hodnotící heuristiky

Pro zhodnocení, zda na základě vykonaného pohybu došlo ke zlepšení stavu, či nedošlo, jsem zvolil takovou heuristiku, která měří vzdálenost mezi hlavou a jablkem. Pro herní scénář, kde nejsou generovány náhodné zdi a hru hraje pouze jedna umělá inteligence, jsou možné heuristiky dvě, a to *Manhattan* a *Euklidovská vzdálenost*. Pro druhý scénář, kde hru hrají současně dvě umělé inteligence a kde každá hra je typická náhodným rozmístěním řad zdí, už je tato heuristika nedostatečná. Z toho důvodu jsem zvolil heuristiku pro výpočet vzdálenosti pomocí algoritmu *A**.

5.3.1 Manhattan vzdálenost (Hammingova metrika)

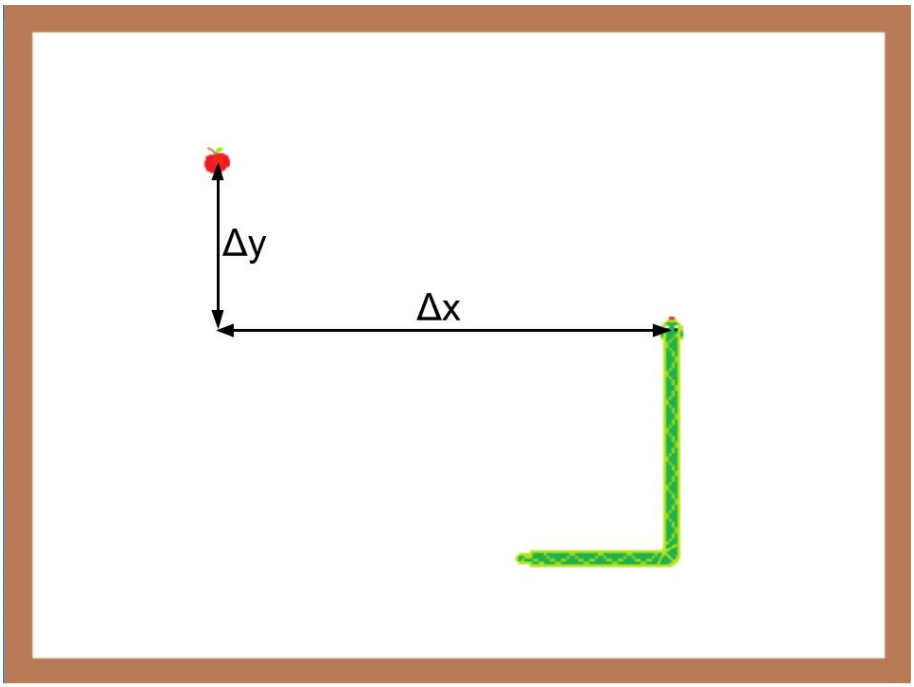
Manhattan vzdálenost je vzdálenost založená, v případě 2D prostoru, na rozdílu souřadnic dvou bodů ve svislém a vodorovném směru. Tato vzdálenost je pro 2D prostor definovaná podle vztahu (14), v Pythonu vyjádřena pomocí algoritmu 4 a na obr. 32 znázorněna v určitém herním stavu.

$$d = \Delta x + \Delta y = |x_2 - x_1| + |y_2 - y_1| \tag{14}$$

Algoritmus 4:

```
def distanceManhattan(snake, apple):
    deltaX = abs(snake.x - apple.x)
    deltaY = abs(snake.y - snake.y)
    dist = deltaX + deltaY

    return dist
```



Obr. 32) Znázorněná Manhattan vzdálenost

5.3.2 Euklidovská vzdálenost

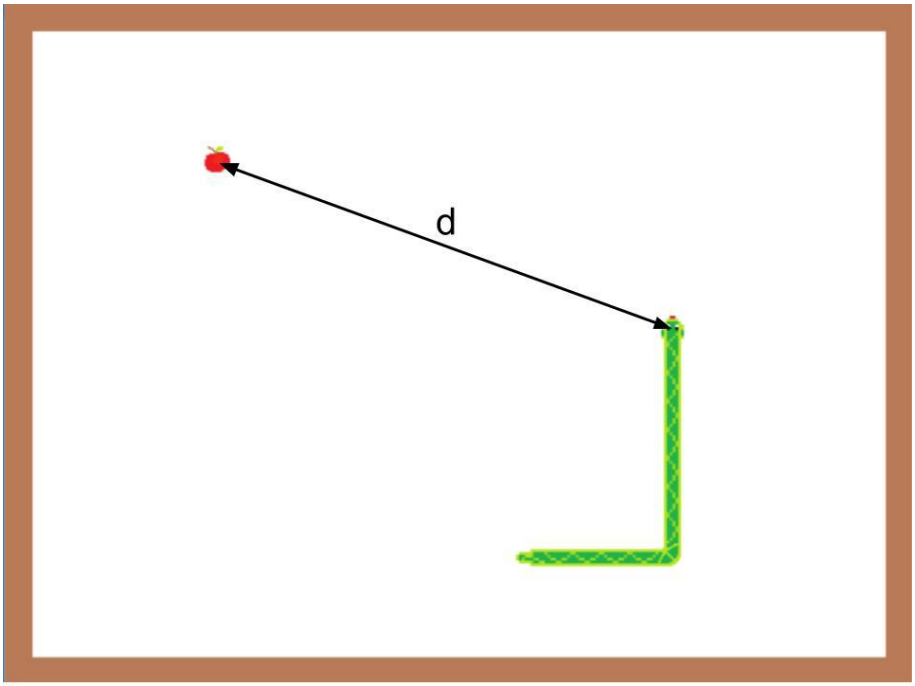
Euklidovská vzdálenost je definovaná v n rozměrném prostoru pomocí vztahu (15). Pod tímto vztahem je ukázka algoritmu 5, kde je definovaná funkce s názvem *distance* pro výpočet této vzdálenosti ve 2D prostoru. Na obr. 33 je tato vzdálenost znázorněna v určitém herním stavu.

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{15}$$

Algoritmus 5:

```
def distance(snake, apple):
    deltaX = snake.x - apple.x
    deltaY = snake.y - apple.y
    dist = (deltaX**2 + deltaY**2)**0.5

    return dist
```



Obr. 33) Znázorněná Euklidovská vzdálenost

5.3.3 Vzdálenost pomocí algoritmu A*

Algoritmus A* se používá pro hledání nejkratší cesty v mřížce mezi dvěma uzly. Tento algoritmus jsem zvolil z důvodu poměrně jednoduché implementace pomocí knihovny *NetworkX*.

Algoritmus A*

Algoritmus A* je modifikací *Dijsktrova algoritmu* s tím rozdílem, že Dijkstrův algoritmus vyhledává nejkratší trasu ze startovního uzlu do všech ostatních uzlů, zatímco algoritmus A* hledá trasu pouze do žádané lokace. K tomuto hledání algoritmus využívá

ohodnocovací funkci *f(n)*, vyjádřenou pomocí vztahu (16). Funkce *g(n)* odpovídá vzdálenosti mezi počátečním a daným uzlem, zatímco funkce *h(n)* představuje heuristické ohodnocení vzdálenosti daného uzlu a uzlu koncového. Heuristika bývá nejčastěji Manhattanská, nebo Euklidovská. V případě mé implementace je použita heuristika Euklidovská. [51]

$$f(n) = g(n) + h(n) \tag{16}$$

NetworkX

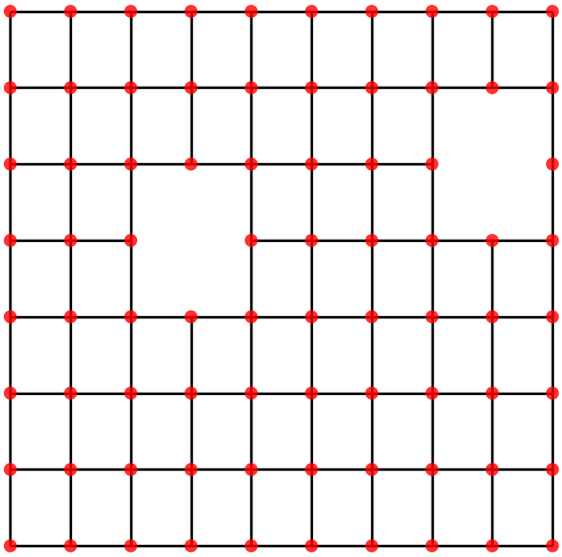
NetworkX je knihovna vytvořená v roce 2002 určená pro tvorbu, manipulaci a zkoumání vlastností grafových struktur. [52]

Pomocí jednoduchých příkazů lze vytvořit graf požadovaných rozměrů a s grafem dále pracovat. Pomocí algoritmu 6 je vytvořen 2D graf o rozměrech 10x8 a následně jsou dva uzly na souřadnicích (3, 4) a (8, 5) odmazány. Výsledný graf můžeme vidět na obr. 34.

Algoritmus 6:

```
import networkx as nx

graph = nx.grid_2d_graph(10, 8)
graph.remove_node((3, 4))
graph.remove_node((8, 5))
```



Obr. 34) Ukázkový graf vytvořený pomocí knihovny NetworkX

Implementace NetworkX

NetworkX je v mé práci implementován tak, že v každém herním kroku se vytváří graf o velikosti herního pole. Následně dojde k odstranění uzlů na takových pozicích, které ve hře odpovídají pozicím zdí, vlastnímu tělu (mimo hlavy) a nepřátelskému hadu. Pomocí

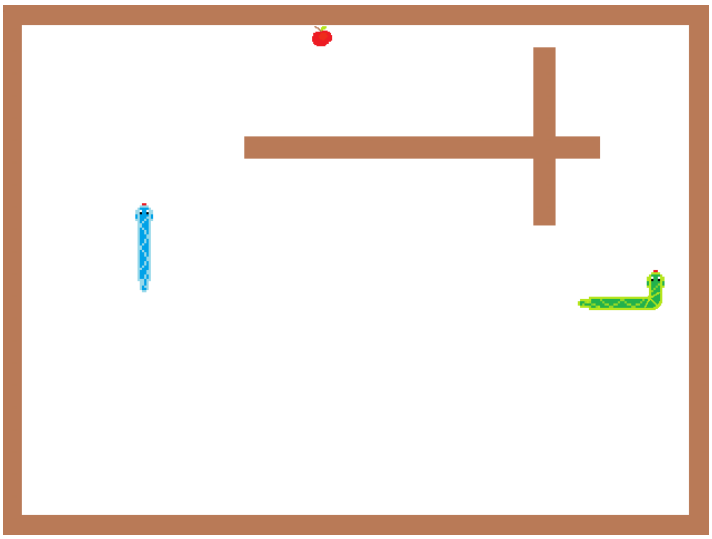
následujícího algoritmu 7, který využívá funkci této knihovny, je určená vzdálenost mezi hlavou hada a jablkem, dle algoritmu A*.

Na obr. 35 je vyobrazena herní situace, která je na dalším obr. 36 znázorněna pomocí NetworkX grafu. V tomto grafu je mimo jiné tato cesta vyznačena zeleně.

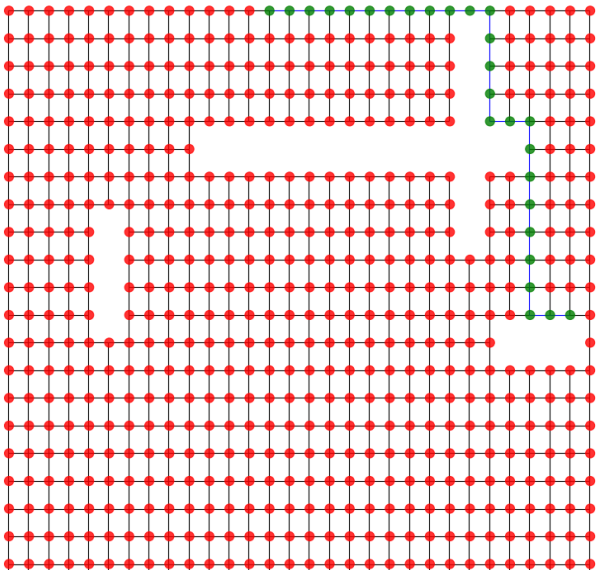
Algoritmus 7:

```
import networkx as nx
def heuristicForAStar(a, b):
    (x1, y1) = a
    (x2, y2) = b
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

pathLength = nx.astar_path_length(graph, (snake.x, snake.y), (apple.x,
apple.y), heuristicForAStar)
```



Obr. 35) Herní předloha pro NetworkX



Obr. 36) Graf dle herní předlohy s vyznačenou cestou

5.4 Navržené umělé inteligence

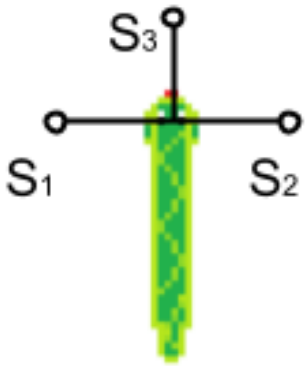
Předposledním cílem mé práce bylo navrhnout vlastní umělou inteligenci. Rozhodl jsem se navrhnout 4 hráče, které jsem pro jednoduchost a přehlednost pojmenoval dle písmen řecké abecedy Alfa, Beta, Beta+ a Gama. Následující část je věnována jejich popisu.

5.4.1 Umělá inteligence Alfa

První navržená umělá inteligence je poměrně jednoduchá. Tvoří ji dopředná neuronová síť, jejíž architektura je znázorněna na obr. 37. Při vytváření této a následující umělé inteligence jsem se inspiroval podle zdroje [53]. Základem jsou 3 senzory (S_n) rozmístěné okolo hlavy, které vrací hodnoty 0 a 1 podle toho, zda se v bezprostřední blízkosti nachází nějaký objekt (obr. 38). Čtvrtý vstup této sítě odpovídá navrhovanému směru, do jakého by se měl had dát. Tento neuron nabývá hodnot -1, 0 a 1. Výstupem sítě jsou 2 neurony, kde první potvrzuje směr a druhý tento směr zamítá.

Tréning neuronové sítě probíhal na datasetu obsahujícím 18 vytvořených situací. Pro hodnocení této sítě jsem zvolil graf závislosti počtu trénovacích epoch na průměrných herních krocích (obr. 41) během 100 her. Protože tato umělá inteligence nemá žádný senzor pro jablko, tak je hodnocení dle počtu nasbíraných jablek zbytečné.

Obr. 37) Schéma neuronové sítě Alfa



Obr. 38) Znázorněné senzory

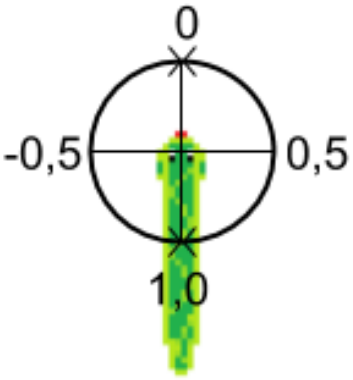
5.4.2 Umělá inteligence Beta

Umělá inteligence Beta je vylepšenou verzí Alfy. Neuronová síť je složitější a její architektura je znázorněna na obr. 39. Hlavní komponenty jsou opět 3 senzory rozmístěné okolo hlavy a vylepšení spočívá v použití dalšího senzoru, který jako vstup dostává normovaný úhel mezi hlavou (v závislosti na směru pohybu) a jablkem. Posledním vstupem je opět navrhovaný směr. Tato síť má 3 výstupy, kde aktivace prvního odpovídá

situaci, kdy se had přiblíží k jablku, aktivace druhého je situace, kdy se had k jablku nepřiblíží a aktivace třetího oznamuje srážku hada.

Tréning probíhal na datech, která byla získána hraním této hry předchozí umělou inteligencí. Typ hry byl pro jednoho hráče bez náhodně generovaných zdí. Na základě pohybů hada došlo k vyhodnocení jeho akcí a k vstupnímu vzoru se přidal odpovídající vzor výstupní. Ohodnocení probíhalo pomocí Euklidovské heuristiky a určovalo se, jestli se had svým pohybem k jablku přiblížil, nebo ne. Celý dataset obsahoval cca 3000 situací a průběh tréningu je možno vidět na obr. 41 a obr. 42. Druhý graf odpovídá závislosti trénovacích epoch na průměrném množství nasbíraných jablek během 100 her. Typ testované hry je hra pro jednoho hráče s náhodně generovanými zdmi.

Obr. 39) Schéma neuronové sítě Beta



Obr. 40) Znázorněný normovaný úhel

5.4.3 Umělá inteligence Beta+

Jedná se o stejný typ neuronové sítě, jako v předchozím případě. Rozdílem jsou trénovací data. Trénovací data byla získána hraním předešlé umělé inteligence ve hře pro jednoho hráče s náhodně generovanými zdmi. Jako ohodnocovací heuristika byla použita vzdálenost pomocí algoritmu A*. Trénovací set se skládal asi z 3000 trénovacích vzorů. A průběh tréningu je vidět na obr. 41 a obr. 42.

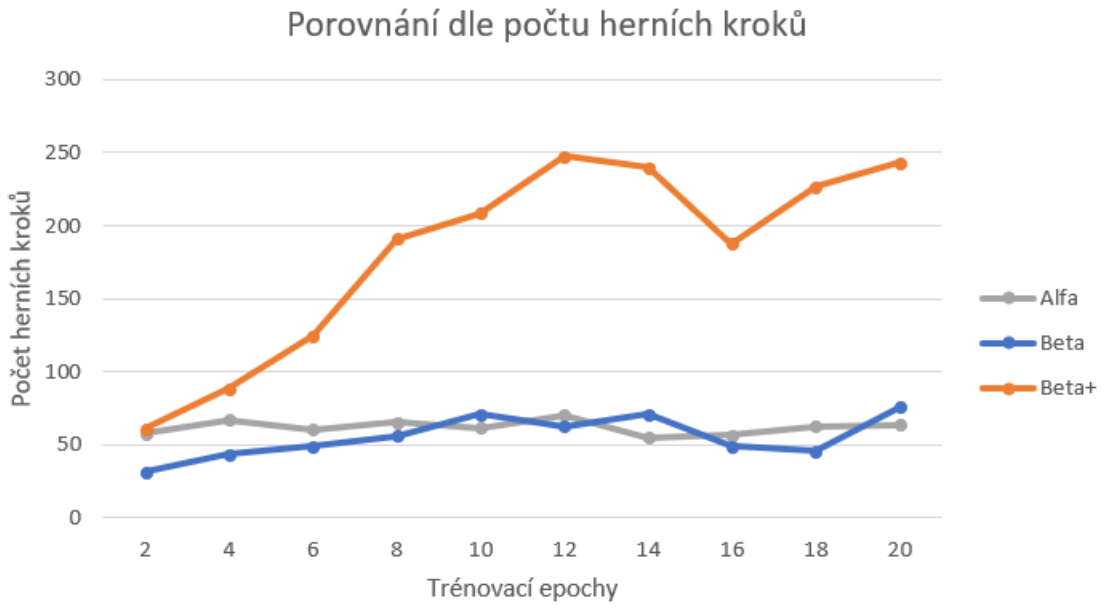
Volba akce

Volba akce probíhá tak, že v každém herním kroku se náhodným způsobem zvolí požadovaný směr. Na základě herního stavu a tohoto směru následuje dopředný průchod odpovídající neuronovou sítí a výsledkem je potvrzení, nebo zamítnutí. Výběrem je první potvrzený směr. Pokud dojde k zamítnutí, je ze zbývajících možností náhodně vybrána možnost další a proces se opět opakuje. Funkce, která odpovídá náhodnému výběru směrů z aktuálně možných je popsána pomocí následujícího algoritmu.

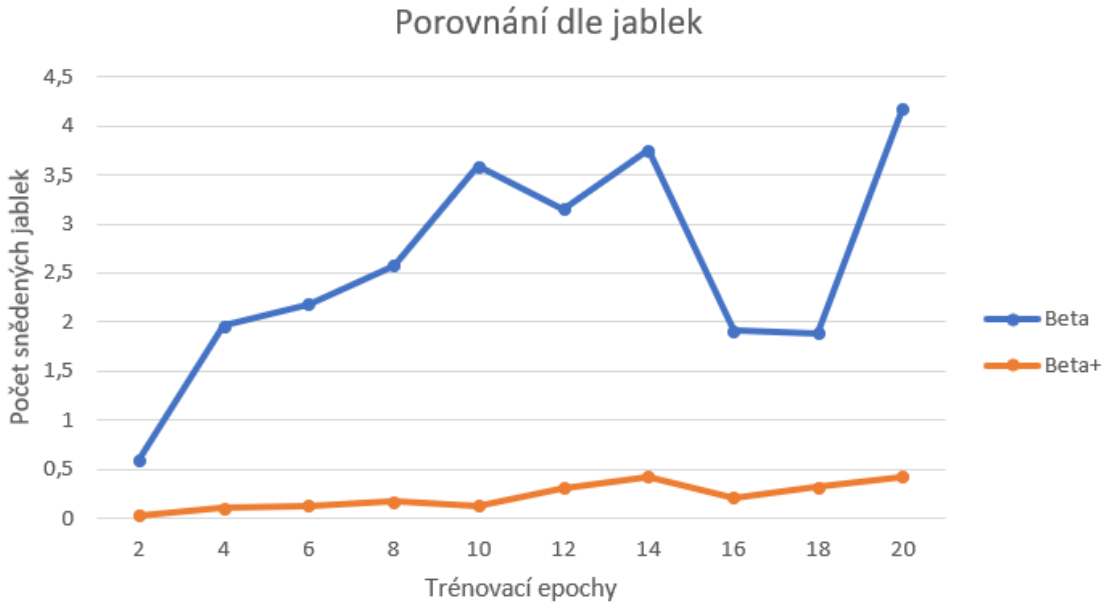
Algoritmus 8:

```
directionVec = [-1, 0, 1]
def choseRandomDirection(directionVec):
    idx = randint(0, len(directionVec)-1)
    suggestedDirection = directionVec.pop(idx)

    return suggestedDirection, directionVec
```



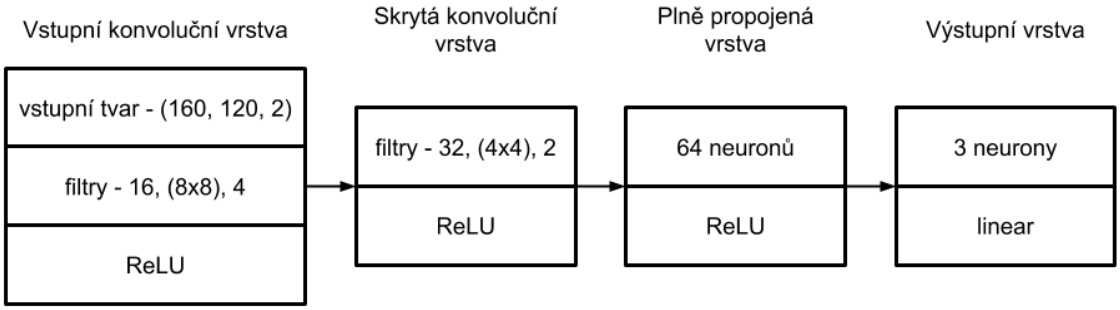
Obr. 41) Porovnání umělých inteligencí v počtu herních kroků



Obr. 42) Porovnání umělých inteligencí v počtu snědených jablek

5.4.4 Umělá inteligence Gama

Poslední umělá inteligence je založena na konvoluční neuronové síti. Vstupem jsou 2 po sobě jdoucí herní snímky ve stupni šedi, zmenšené na 160 x 120 pixelů. Vstupní vrstva je tvořena 16ti konvolučními filtry, každý o rozměru 8 x 8, s posunem 4 pixely. Následuje druhá konvoluční vrstva obsahující 32 konvolučních filtrů o velikosti 4 x 4 a posunem 2 pixely. Další vrstvou je plně propojená vrstva s 64 neurony. Aktivační funkcí těchto vrstev je ReLU. Výstupní vrstvu tvoří 3 neurony s lineární aktivací, tudíž se jedná o mnohavýstupní regresní model. Každý jeden výstup odpovídá ohodnocení právě jedné akce. Architektura této sítě je znázorněna na obr. 43.



Obr. 43) Schéma neuronové sítě Gama

Tréning sítě probíhá po každé odehrané hře, kde konečný stav je smrt hada, smrt nepřátelského hada, nebo dosažení limitu herních kroků. V každém herním kroku se ukládá herní stav (aktuální a minulý snímek) a dojde k ohodnocení vykonané akce. Ohodnocení dle situací je následující:

- had vyhraje, ohodnocení: 1,0
- had prohraje, ohodnocení: -1,0
- přiblížení k jablku, ohodnocení: 1,0
- had se dostane do pasti, ohodnocení: -0,5
- žádná situace, ohodnocení: 0

Tato trénovací data lze získat pomocí předchozích neuronových sítí, ale i použitím aktuální sítě.

Volba akce

V každém herním kroku do neuronové sítě vstupují 2 po sobě jdoucí herní snímky. Výstupem jsou 3 hodnoty, které odpovídají ohodnocení každé možné akce. Umělá inteligence volí akci s největším možným ohodnocením z těchto 3 hodnot.

6 ZÁVĚR

Cílem diplomové práce bylo navrhnout soutěžní hřiště pro umělé inteligence, včetně vlastních řešení herních strategií implementovaných prostřednictvím neuronových sítí. Práce byla rozdělena do několika kapitol, kde první kapitola představila pojem umělá inteligence a dala obecný náhled na videohry. Druhá kapitola byla směřována na popis a zhodnocení dosažených výsledků společnosti DeepMind (Google), která přišla s výborným a v praxi fungujícím algoritmem posilovaného učení Deep Q Learning. Vhodnou kombinací algoritmů vytvořili takové umělé inteligence, které v některých hrách bezkonkurenčně porazili nejen elitní lidské protihráče, ale i doposud nejsilnější umělé inteligence.

V této práci bylo na umělé inteligence pohlíženo především v kontextu umělých neuronových sítí, proto třetí kapitola obsahuje základní popis zmíněných sítí, jejich rozdělení (zaměřené na použité sítě) a také popis jednotlivých aktivačních funkcí. Další důležitá část je kapitola s názvem Strojové učení. V této kapitole se čtenář přesvědčil, že strojové učení je v dnešní době velmi živé téma a je široce podporované množstvím dostupných knihoven.

Poslední nejdůležitější částí je vlastní implementace zavedené hry. Byla vytvořena počítačová hra Snake, vhodná pro aplikaci různých typů umělých inteligencí. Toto hřiště podporuje hru 2 hráčů a umožňuje nechat soutěžit umělé inteligence mezi sebou, nebo také vyzkoušet souboj člověka proti stroji. Na tento cíl navazoval návrh vlastní umělé inteligence. V závěru této kapitoly je popis 4 vytvořených umělých inteligencí. Za zmínku stojí porovnávací grafy, které měří průměrné množství nasbírané potravy a průměrné množství herních kroků v závislosti na počtu trénovacích epoch (obr. 41 a obr. 42). Pro Betu je charakteristický pokles v počtu nasbíraných jablek při 16. a 18. epoše. Tento pokles se také projevuje u Bety+, ale v kategorii průměrně dosažených herních kroků. Tento problém pravděpodobně vznikl na základě poměrně malého množství trénovacích dat. Tuto statistiku tvoří první 3 neuronové sítě, které nemají velké požadavky na výpočetní výkon. Čtvrtá neuronová síť pro úspěšné natrénování potřebuje obrovské množství dat a herního času, proto její zpracování zůstává ve formě návrhu. Popis programu, tříd a metod se nachází v dokumentaci, která je přílohou (Příloha A).

7 SEZNAM POUŽITÉ LITERATURY

[1] Artificial Intelligence - AI. INVESTOPEDIA [online]. , 1 [cit. 2018-05-20]. Dostupné z: https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp

[2] A Look at the Video Game Industry. Big Fish [online]. 2018 [cit. 2018-05-27]. Dostupné z: https://www.bigfishgames.com/blog/a-look-at-the-video-game-industry/

[3] AUTOR NEUVEDEN. www.statista.com [online]. [cit. 20.5.2018]. Dostupný na WWW: https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre

[4] A Look at the Video Game Industry. Big Fish [online]. 2018 [cit. 2018-05-27]. Dostupné z: https://www.bigfishgames.com/blog/a-look-at-the-video-game-industry/

[5] AlphaGO Rules of Go - introductory. Sensei's Library [online]. 2018 [cit. 2018-05-27]. Dostupné z: https://senseis.xmp.net/?BasicRulesOfGo

[6] AUTOR NEUVEDEN. http://www.ultraboardgames.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: http://www.ultraboardgames.com/go/gfx/go-table.jpg

[7] SILVER, David, Aja HUANG, Chris J. MADDISON, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016, 529(7587), 484-489. DOI: 10.1038/nature16961. ISSN 0028-0836. Dostupné také z: http://www.nature.com/articles/nature16961

[8] MNIH, Volodymyr, Koray KAVUKCUOGLU, David SILVER, Alex GRAVES, Ioannis ANTONOGLOU, Daan WIERSTRA a Martin RIEDMILLER. Playing Atari with Deep Reinforcement Learning. , 9.

[9] Atari 2600. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Atari_2600

[10] AUTOR NEUVEDEN. https://atariage.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://atariage.com/2600/screenshots/s_AirSeaBattle_1.png

[11] AUTOR NEUVEDEN. https://atariage.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://atariage.com/2600/screenshots/s_StarShip_2.png

[12] AUTOR NEUVEDEN. https://atariage.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://atariage.com/2600/screenshots/s_StreetRacer_1.png

[13] AUTOR NEUVEDEN. https://atariage.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://atariage.com/2600/screenshots/s_Breakout_2.png

[14] AUTOR NEUVEDEN. https://deepmind.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://storage.googleapis.com/deepmind-live-cms/images/Mnih_Fig3_R3%2520SM.width-1500_ozgffCw.png

[15] SHIFFMAN, Daniel. The nature of code: simulating natural systems with processing. Version 1.0, generated December 6,2016. New York: Free Software Foundation, 2012. ISBN 978-0985930806.

[16] MCCULLOCH, Warren S. a Walter PITTS. A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics. 1943, 5(4), 115-133. DOI: 10.1007/BF02478259. ISSN 0007-4985. Dostupné také z: http://link.springer.com/10.1007/BF02478259

[17] HUANG. Independent Seminar Blog [online]. [cit. 20.5.2018]. Dostupný na WWW: https://independentseminarblog.files.wordpress.com/2017/10/c8057-0ohlzxsodsebxw0ii.jpg?w=625&zoom=2

[18] KRIESEL, David. A Brief Introduction to Neural Networks. 2012.

[19] NILS, J. Nilsson. INTRODUCTION TO MACHINE LEARNING. 1998.

[20] GUPTA, Dishashree. Fundamentals of Deep Learning – Activation Functions and When to Use Them?. Analytics Vidhya [online]. [cit. 2018-05-20]. Dostupné z: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

[21] POLAMURI, Saimadhu. DIFFERENCE BETWEEN SOFTMAX FUNCTION AND SIGMOID FUNCTION. Dataaspirant [online]. 2017 [cit. 2018-05-20]. Dostupné z: http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

[22] MCGONAGLE, John. Feedforward Neural Networks. Brilliant [online]. [cit. 2018-05-20]. Dostupné z: https://brilliant.org/wiki/feedforward-neural-networks/

[23] DONGES, Niklas. https://towardsdatascience.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://cdn-images-1.medium.com/max/1000/0*mRHhGAbsKaJPbT21.png

[24] MCGONAGLE, John a Christopher WILLIAMS. Recurrent Neural Network. Brilliant [online]. [cit. 2018-05-20]. Dostupné z: https://brilliant.org/wiki/recurrent-neural-network/

[25] O'SHEA, Keiron a Ryan NASH. An Introduction to Convolutional Neural Networks. , 11.

[26] PAVLOVSKY, Vojtech. http://www.vaetas.cz/ [online]. [cit. 20.5.2018]. Dostupný na WWW: http://www.vaetas.cz/img/machine-learning/convolutional-neural-network.png

[27] Keras: The Python Deep Learning library [online]. [cit. 2018-05-20]. Dostupné z: https://keras.io/

[28] TensorFlow [online]. [cit. 2018-05-20]. Dostupné z: https://www.tensorflow.org/

[29] SHEKHAR, Amit. How Does The Machine Learning Library TensorFlow Work?. Let's Learn AI [online]. 2018 [cit. 2018-05-20]. Dostupné z: https://www.letslearnai.com/2018/02/02/how-does-the-machine-learning-library-tensorflow-work.html

[30] AUTOR NEUVEDEN. https://franciskim.co/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://d37q6i2m2fcvyp.cloudfront.net/wp-content/uploads/2017/03/img_58c5145cbe7e5.png?x38627

[31] CLOUD TPU BETA. Google Cloud [online]. [cit. 2018-05-20]. Dostupné z: https://cloud.google.com/tpu/

[32] HAUßMANN, Elmar. Comparing Google's TPUv2 against Nvidia's V100 on ResNet-50. RiseML Blog [online]. [cit. 2018-05-20]. Dostupné z: https://blog.riseml.com/comparing-google-tpuv2-against-nvidia-v100-on-resnet-50-c2bbb6a51e5e

[33] Theano [online]. [cit. 2018-05-20]. Dostupné z: http://deeplearning.net/software/theano/

[34] BROWNLEE, Jason. Introduction to the Python Deep Learning Library Theano. Machine Learning Mastery [online]. 2016 [cit. 2018-05-20]. Dostupné z: https://machinelearningmastery.com/introduction-python-deep-learning-library-theano/

[35] The Microsoft Cognitive Toolkit. Microsoft Docs [online]. [cit. 2018-05-20]. Dostupné z: https://docs.microsoft.com/en-us/cognitive-toolkit/

[36] CNTK. GitHub [online]. [cit. 2018-05-20]. Dostupné z: https://github.com/microsoft/cntk

[37] Caffe2 [online]. [cit. 2018-05-20]. Dostupné z: https://caffe2.ai/

[38] Apache MXNet (incubating) for Deep Learning. GitHub [online]. [cit. 2018-05-20]. Dostupné z: https://github.com/apache/incubator-mxnet

[39] PyTorch [online]. [cit. 2018-05-20]. Dostupné z: https://pytorch.org/

[40] SHAIKH, FAIZAN. An Introduction to PyTorch – A Simple yet Powerful Deep Learning Library. Analytics Vidhya - Learn everything about Analytics [online]. [cit. 2018-05-20]. Dostupné z: https://www.analyticsvidhya.com/blog/2018/02/pytorch-tutorial/

[41] WATKINS, Christopher. Learning From Delayed Rewards. Royal Holloway, 1989. Thesis. University of London.

[42] PATTERSON, Josh. Introduction To Deep Q-Learning. Medium [online]. [cit. 2018-05-20]. Dostupné z: https://medium.com/@joshpatterson_5192/introduction-to-deep-q-learning-1bded90a6193

[43] ARAUJO, Leonardo. Artificial Inteligence [online]. [cit. 2018-05-28]. Dostupné z: https://leonardoaraujosantos.gitbooks.io/

[44] DAS, Aneek. Introduction to Q-Learning. Towards Data Science [online]. [cit. 2018-05-20]. Dostupné z: https://towardsdatascience.com/introduction-to-q-learning-88d1c4f2b49c

[45] MATIISEN, Tambet. Demystifying Deep Reinforcement Learning. Intel AI [online]. [cit. 2018-05-20]. Dostupné z: https://ai.intel.com/demystifying-deep-reinforcement-learning/

[46] HILEY, Catherine. https://www.uswitch.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://uswitch-mobiles1-contentful.imgix.net/qhi9fkhtpbo3/2V9jWt5q0wywQK2W2Eua6Y/03616ea1a87ef1c1c9fed196b4247c2f/snake2.png?w=770

[47] MIHAJLOV, Ilija. https://www.dosgamesarchive.com [online]. [cit. 20.5.2018]. Dostupný na WWW: https://image.dosgamesarchive.com/screenshots/thumbnails/snake2.gif

[48] AUTOR NEUVEDEN. https://www.chupamobile.com/ [online]. [cit. 20.5.2018]. Dostupný na WWW: https://chupacdn.s3.amazonaws.com/catalog/product/cache/1/thumbnail/108x90/17f82f742ffe127f42dca9de82fb58b1/s/n/snake-game-6488_imgs_6488_2.jpg

[49] Pygame About. Pygame [online]. [cit. 2018-05-20]. Dostupné z: https://www.pygame.org/wiki/about

[50] AUTOR NEUVEDEN. https://github.com/ [online]. [cit. 27.5.2018]. Dostupný na WWW: https://raw.githubusercontent.com/Hevaesi/Roguelike-pygame/master/etc/screenshot.png

[51] Introduction to A*. Red Blob Games [online]. [cit. 2018-05-20]. Dostupné z: https://www.redblobgames.com/pathfinding/a-star/introduction.html

[52] NetworkX [online]. [cit. 2018-05-20]. Dostupné z: https://networkx.github.io/documentation/stable/index.html

[53] KOROLEV, Slava. NEURAL NETWORK TO PLAY A SNAKE GAME. Towards Data Science [online]. [cit. 2018-05-20]. Dostupné z: https://towardsdatascience.com/today-im-going-to-talk-about-a-small-practical-example-of-using-neural-networks-training-one-to-6b2cbd6efdb3

8 SEZNAM OBRÁZKŮ

Obr. 1. Podíl prodeje her dle žánrů [3] 17
Obr. 7. Desková hra Go [6] 18
Obr. 3. AlphaGo výkonnostní graf, [7] 19
Obr. 4. Graf přesnosti ohodnocení [7] 20
Obr. 5. porovnání Go programů [7] 21
Obr. 6a Air-Sea Battle [10] 21
Obr. 6b Star Ship [11] 21
Obr. 6c Street Racer [12] 21
Obr. 6d Breakout [13] 21
Obr. 7a Průměrné skóre za epochu [8] 23
Obr. 7b Průměrné ohodnocení akcí za epochu [8] 23
Obr. 8. Výkon DQN agenta [14] 23
Obr. 9. Umělý neuron, upraveno podle [17] 25
Obr. 10. Jednotkový skok 27
Obr. 11. Lineární funkce 28
Obr. 12. Sigmoida 28
Obr. 13. Hyperbolický tangens 29
Obr. 14. ReLU 29
Obr. 15. Leaky ReLU 30
Obr. 16. Příklad dopředné neuronové sítě (Multi-layer perceptron), upraveno podle [23] 31
Obr. 17. Příklad rekurentní neuronové sítě, upraveno podle [23] 32
Obr. 18. Koncept konvoluční neuronové sítě, upraveno podle [26] 32
Obr. 19. Obecný model umělé inteligence, upraveno podle [19] 33
Obr. 20. TensorBoard [30] 35
Obr. 21. TPU [31] 35
Obr. 22. Porovnání TPU a GPU [32] 35
Obr. 23. Příklad pro Q learning, upraveno podle [43] 37
Obr. 24. Ohodnocovací tabulka R, upraveno podle [43] 37
Obr. 25. Architektura pro Deep Q Learning, upraveno podle [45] 39
Obr. 26a Snake II [46] 41
Obr. 26b Snake by Ilija Mihajlov [47] 41
Obr. 26c Snake Game by Banditmoviegames [48] 41
Obr. 27a Hlava 41
Obr. 27b Tělo 41
Obr. 27c Jablko 41
Obr. 27d Zeď 41
Obr. 28. SnakeAI s popisem 42

Obr. 29. Roguelike-Pygame [50] 43
Obr. 30. Hierarchie tříd 43
Obr. 31. Soketové rozhraní 44
Obr. 32. Znázorněná Manhattan vzdálenost 47
Obr. 33. Znázorněná Euklidovská vzdálenost 48
Obr. 34. Ukázkový graf vytvořený pomocí knihovny NetworkX 49
Obr. 35. Herní předloha pro NetworkX 50
Obr. 36. Graf dle herní předlohy s vyznačenou cestou 50
Obr. 37. Schéma neuronové sítě Alfa 51
Obr. 38. Znázorněné senzory 51
Obr. 39. Schéma neuronové sítě Beta 52
Obr. 40. Znázorněný normovaný úhel 52
Obr. 41. Porovnání umělých inteligencí v počtu herních kroků 53
Obr. 42. Porovnání umělých inteligencí v počtu snědených jablek 53
Obr. 43. Schéma neuronové sítě Gama 54

9 SEZNAM PŘÍLOH

Příloha A Dokumentace