# Učení posilováním v umělých neuronových sítích Q-algoritmus

Bc. Michal Brückner

Diplomová práce
2021



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Michal Brückner** |
| Osobní číslo: | **A18546** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Forma studia: | **Kombinovaná** |
| Téma práce: | **Učení posilováním v umělých neuronových sítích – Q-algoritmus** |
| Téma práce anglicky: | **Reinforcement Learning in Artificial Neural Nets – The Q-algorithm** |

## Zásady pro vypracování

1. Proveďte rešerši v oblasti použití metod umělé inteligence v počítačových hrách.
2. Seznamte se s problematikou učení posilováním v umělých neuronových sítích.
3. Zaměřte se především na Q-algoritmus a implementujte jej do zvoleného typu počítačové hry.
4. Edukativním způsobem objasněte rozdíl mezi učením bez Q-algoritmu a s Q-algoritmem.
5. Připravte názornou prezentaci k dané tématice.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:


1. COMI, Mauro, 2019. How to teach an AI to play Games: Deep Reinforcement Learning. Medium [online] [vid. 2019-11-28]. Dostupné z: https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a
2. ANON., nedatováno. [1802.06225] A Deep Q-Learning Agent for the L-Game with Variable Batch Training [online] [vid. 2019-11-28]. Dostupné z: https://arxiv.org/abs/1802.06225
3. MNIH, Volodymyr, Koray KAVUKCUOGLU, David SILVER, Alex GRAVES, Ioannis ANTONOGLOU, Daan WIERSTRA a Martin RIED-MILLER, 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs] [online]. [vid. 2019-11-28]. Dostupné z: http://arxiv.org/abs/1312.5602
4. SURMA, Greg, 2019. Cartpole – Introduction to Reinforcement Learning (DQN – Deep Q-Learning). Medium [online] [vid. 2019-11-28]. Dostupné z: https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288
5. ANON., nedatováno. Atari – Solving Games with AI (Part 1: Reinforcement Learning) [online] [vid. 2019-11-28]. Dostupné z: https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f
6. HABIB, Nazia, 2019. Hands-On Q-Learning with Python: Practical Q-learning with OpenAI Gym, Keras, and TensorFlow. B. m.:Packt Publishing. ISBN 978-1-78934-580-3.
7. LAPAN, Maxim, 2018. Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more. Birmingham Mumbai: Packt Publishing. ISBN 978-1-78883-424-7.


Vedoucí diplomové práce: **doc. Ing. Zuzana Komínková Oplatková, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

**prof. Mgr. Roman Jašek, Ph.D.** v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Michal Brückner, v.r.
podpis studenta

## ABSTRAKT

V této práci je srozumitelnou formou popsána problematika zpětnovazebního učení, především v poslední době velmi populární Q-algoritmus. Úvodní část je věnována historii použití umělé inteligence v počítačových hrách. Dále je popsán princip fungování posilovaného učení a neuronových sítí. Praktická část práce se věnuje vysvětlení a implementaci konkrétního algoritmu.



Klíčová slova: Umělá inteligence, Zpětnovazební učení, Q-učení, Hluboké Q-učení, Neuronové sítě, Konvoluční neuronové sítě, Gomoku



## ABSTRACT

In this diploma thesis, the matter of reinforcement learning is described in an understandable manner. Especially the Q-algorithm, which is nowadays very popular. Introductory part is related to the history of the use of artificial intelligence in the computer games. Furthermore, the principle of functioning of reinforcement learning and related neural networks is described. Practical part of the thesis explains and shows the implementation of a specific algorithm.



Keywords: Artificial intelligence, Reinforcement learning, Q-learning, DQN, Neural networks, CNN, Gomoku

Chtěl bych poděkovat vedoucí mé diplomové práce doc. Ing. Zuzaně Komínkové Oplatkové, Ph.D. za odborné vedení, konzultace, rady a za pomoc při zpracování této práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

# OBSAH

# ÚVOD

Cílem této práce je seznámit všechny její čtenáře jednoduchou a srozumitelnou formou s tématikou posilovaného učení.

Práce se skládá z 5 na sebe tematicky a obsahově navazujících kapitol a má zejména edukativní charakter.

V praktické části je implementován Q-algoritmus do deskové hry Gomoku, kdy uživatel může sledovat a ovlivňovat průběh učení. Vytrénované agenty lze otestovat nasazením proti jiným hráčům. K snazšímu pochopení fungování Q-algoritmu slouží pak přehledné statistiky a grafy.

# I. TEORETICKÁ ČÁST

# 1 HISTORIE A POUŽITÍ METOD UMĚLÉ INTELIGENCE V POČÍTAČOVÝCH HRÁCH

Počítačové hry a umělá inteligence mají dlouhou společnou historii. Velká část badatelské činnosti týkající se umělé inteligence použité ve hrách se zabývala tvorbou činitele (agenta), který dokázal zlepšovat své schopnosti na základě učení z předchozího průběhu hry.

Historicky to byl první a po dlouhou dobu jediný způsob využívání umělé inteligence. Ještě předtím, než byla umělá inteligence uznána jako samostatný vědní obor, začali první průkopníci informatiky vytvářet herní programy, jelikož chtěli zjistit, zda by počítače byly schopné vyřešit úkoly, u nichž se zdálo, že vyžadují jistou dávku inteligence.

Alan Turing, kterého lze prokazatelně považovat za zakladatele počítačové vědy, vymyslel algoritmus zvaný Minimax a použil ho při hraní šachů.

První herní program byl naprogramován A. S. Douglasem v roce 1952. Digitální verze hry zvané „Tic – Tac – Toe“ byla použita jako součást jeho disertační práce na univerzitě v Cambridge.

O pár let později vymyslel Arthur Samuel jako první způsob strojového učení, který se nyní nazývá posilované/zpětnovazební učení. Použil při tom program, který se naučil hrát dámu hraním proti sobě.

Většina badatelské činnosti zaměřující se na výzkum herní umělé inteligence byla soustředěna na klasické deskové hry jako jsou dáma nebo šachy. Vycházelo se přitom z představy, že tyto hry vyžadující po staletí a tisíciletí to nejlepší z lidské mysli, u nichž s použitím jednoduchých pravidel může vzniknout složitý a obtížně řešitelný úkon, nějakým způsobem zachytily skutečnou podstatu myšlení.

Po více než třech desetiletích výzkumu v oblasti stromového vyhledávání se programu Chinook podařilo jako prvnímu počítačovému programu porazit světového šampiona ve hře dáma matematika Mariona Tinsleye.

Hra v šachy byla po dekády považována za nepřekonatelnou výzvu umělé inteligence ve smyslu modelového organismu, na kterém se testovaly nové, nenumerické modely umělé inteligence.

Nakonec byl vyvinut software, jež byl schopen hrát lépe než člověk. V určitém bodě dokázala umělá inteligence najít efektivnější tahy. Program, který poprvé ukázal své nadlidské

schopnosti ve hře v šachy, byl Deep Blue vytvořený firmou IBM. Deep Blue obsahoval již dříve zmiňovaný algoritmus Minimax s mnoha šachovými modifikacemi a také velmi dobře vyladěnou vyhodnocovací základnu. Vše fungovalo za pomoci na míru sestaveného superpočítače. V roce 1997 Deep Blue slavně zvítězil proti šachovému velmistrovi Garry Kasparovovi. Tato výhra superpočítače byla ve své době velmi sledována a oběhla doslova celý svět.

Dnes, o více jak 20 let později, lze na internetu získat veřejně dostupný software schopný porazit v šachu kohokoliv a stačí mu k tomu běžný výpočetní výkon.

Neméně důležitým se ve výzkumu umělé inteligence stal software pro hru Vrchcáby nazvaný TD-Gammon. Ten byl vytvořen v roce 1992 - jen pár let před úspěchy Deep Blue a programu Chinook - Garaldem Tesaurem. TD-Gammon pracoval s umělou neuronovou sítí, která se zdokonalovala pomocí tzv. dočasného rozdílového učení. Proces fungoval tak, že tento program hrál několik miliónů her sám proti sobě. Díky tomu byl schopen hrát Vrchcáby na úrovni nejlepších světových hráčů v této hře.

Poté, co se Deep Blue podařilo porazit G. Kasparova, přišla firma IBM s novým softwarem, který nesl název Watson. Tento software byl schopen odpovídat na otázky kladené v lidské řeči. V roce 2011 vyhrál Watson 1 milion amerických dolarů, když v televizní hře známé v České republice jako Riskuj porazil předešlého lidského šampiona tohoto pořadu.

Dalším zlomovým rokem pro umělou inteligenci se stal rok 2016. Tehdy prohrál špičkový profesionální hráč ve hře GO Lee Sedol pět her proti softwaru Alphago od společnosti Google (DeepMind´s).

Hra GO byla v té době brána za nepřekonatelnou laťku pro software s umělou inteligencí, neboť větvící faktor se u této hry blíží 250 a nabízí tak daleko více variant pro vyhledávání než hra v šachy. Předpokládalo se, že program schopný hrát na lidské úrovni bude vytvořen v daleké budoucnosti, ale podařilo se to už v roce 2016.

O rok později, mezi dny 23. 5. - 27.5. 2017 vyhrál zdokonalený AlphaGo třikrát nad světovým šampionem Ke Jie. Opět k tomu byl použit běžný stolní počítač. Go se tak stala poslední klasickou deskovou hrou, kde se ukázaly nadlidské schopnosti umělé inteligence a možnosti zpětnovazebního učení. Pro počítače je samozřejmě možné vytvořit ještě náročnější deskovou hru než je Go, avšak taková hra by byla až příliš náročná a komplikovaná pro lidské hráče.

Klasické stolní hry s předepsaným systémem tahů, kde oba hráči mohou pozorovat aktuální stav hry, nejsou jedinými hrami, při nichž lze využít možnosti umělé inteligence.

Po roce 2000 se vědecká komunita zaměřila na možnost využití umělé inteligence v jiných než deskových hrách, a to ve videohrách.

Značná část výzkumu této vědecké komunity se zabývá vývojem umělé inteligence pro hraní počítačových her buď ve smyslu co nejvyšší a nejefektivnější úrovně, nebo ve schopnost reagovat a řešit herní problém tak, jak by jej řešil lidský hráč.

Milníkem v použití umělé inteligence ve hrách se stal rok 2014, kdy se algoritmus vyvinutý společností Google DeepMind naučil hrát několik her z klasické videoherní konzole Atari 2600 a to na takové úrovni, kterou by žádný člověk nezvládl, přičemž vycházel pouze z bodů obsažených na obrazovce.

Jedna z her Atari, jež se zdála být oříškem pro systémy umělé inteligence, byla hra Ms Pac-Man vydaná v roce 1982. Avšak i toto bylo vyřešeno v roce 2017 díky hybridní odměňovací architektuře zpětnovazebního učení od týmu Maluuba (Microsoft).

Další využití umělé inteligence v počítačových hrách na sebe nenechalo dlouho čekat. Jedním z dalších přístupů bylo procedurální generování obsahu. Počátkem 80 let minulého století začaly některé počítačové hry vytvářet část svého obsahu algoritmicky v průběhu jejich samostatného běhu místo toho, aniž by byly dopředu navrhnuty člověkem. Dřívější postup byl totiž takový, že obsah hry byl neměnný.

Dvě hry, které s tímto průlomovým řešením přišly, byly hry Rogue (Zloděj, 1980) a Elite (1984). Ve hře Rogue se prostředí hry, umísťování herních nestvůr a herních předmětů generovalo znovu vždy pro každou novou hru.
Ve hře Elite se vždy s novou hrou vytvářel unikátní vesmír a hvězdný systém, ve kterém se děj hry odehrával.

Velkou výhodou her, které byly vždy schopny znovu generovat část svého obsahu, bylo nekonečné množství herních variant bez nutnosti ručního nastavování. Zároveň tato schopnost snížila nároky na harddisky počítačů. Úspěch automaticky generovaných her se naplno ukázal u herních titulů jako např. Diablo III (2012) a No Mans Sky (2016).

Relativně nedávno se systémy umělé inteligence začaly používat při analýze počítačových her a vytváření modelů samotných hráčů her.

Toto se ukázalo jako velmi důležité při snaze herních studií zaujmout co nejširší spektrum hráčů. Velmi k tomu přispěl i prudký rozvoj internetového připojení, které předává množství dat a nových poznatků přímo na servery tvůrců počítačových her.

Velmi populární začaly být Facebookové hry, jako například hra FarmVille (2009), které těží z neustálého sběru dat a na základě těchto dat částečně vytváří herní obsah. Herní prostředí se rozvíjí pomocí analýzy ohromného množství dat, kterou zpracovává umělá inteligence.

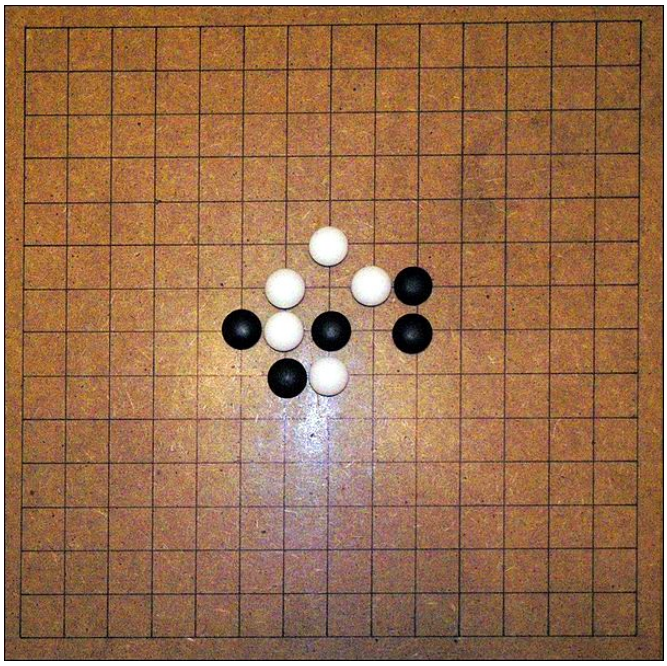
Dnešní moderní hry (Nevermind, 2016) umí zjistit emoční rozpoložení hráče a na základě toho upravit obsah hry.

Poslední vývoj výzkumu umělé inteligence se zaměřuje na schopnost napodobování lidského chování. Testování napodobování lidského chování probíhá díky programu zvanému Turingův test. Turingův test hodnotí, zda pozorované chování hráče odpovídá spíše chování živého člověka, nebo je naopak činností herního robota ovládaného umělou inteligencí.

V roce 2012 - sto let od narození A. Turinga – dva hráči ovládaní umělou inteligencí uspěli v tomto testu ve hře Unreal Tournament. Jejich chování bylo nerozeznatelné od chování živých hráčů. [2]

## 1.1 Desková hra Gomoku

Tuto prastarou hru, u nás známou spíš pod názvem Piškvorky, určitě každý z nás někdy v životě hrál. Málokdo ale asi ví, že vychází ze hry Go a že její historie sahá do období dva tisíce let před naším letopočtem, do oblasti kolem delty řeky Hwang Ho v Číně. Odtamtud se časem rozšířila do Japonska, kde byla známá pod názvem *gomokunarabe*, což by se zhruba dalo přeložit jako pět v řadě. Hra se původně hrála na desce o velikosti 19x19 (Obr. 1), nyní se ale nejčastěji používá deska o rozměrech 15x15. [3]



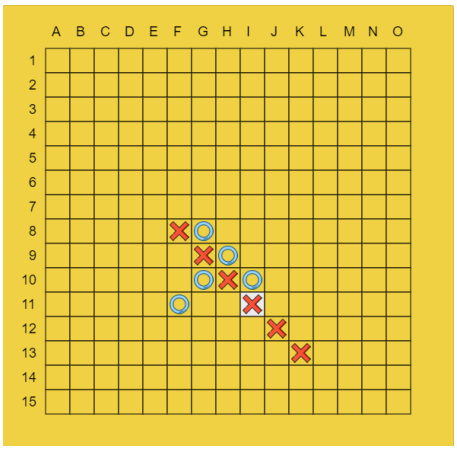
Obrázek 1. Hrací deska s bílými a černými kameny [4]

### 1.1.1 Pravidla

Pravidla jsou velmi jednoduchá. Dva hráči se střídají v provádění tahů a při každém položí hráč jeden kámen svojí barvy – a to buď bílé, nebo černé – na volné místo na hrací desce. Vyhrává ten, který jako první spojí pět svých kamenů, a to ortogonálně, nebo diagonálně. Hru vždy začíná hráč s černými kameny. To mu ale dává velkou počáteční výhodu. Tak značnou, že pokud bude hráč s černými kameny hrát přesně, vždy vyhraje. Ostatně tento fakt již matematicky potvrdil holandský počítačový odborník Victor Allis v roce 1994 ve své práci Searching for Solutions in Games and Artificial Intelligence. [5]. Proto také vzniklo mnoho modifikací, které se snaží hru pro hráče hrajícího s bílými kameny vyrovnat.

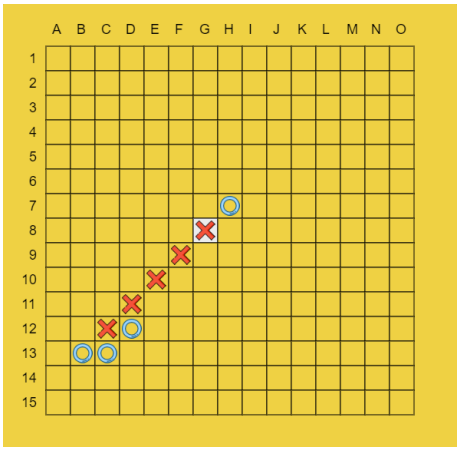
### 1.1.2 Varianta Caro

Varianta Caro je oblíbená hlavně v asijských zemích a někdy je také nazývána *gomoku+* [6]. Základním pravidlem v této variantě hry gomoku je, že vyhrává ten hráč, který spojí přesně 5 svých kamenů. Nesmí jich být ani o jeden více (Obr. 2).




Obrázek 2. Šest kamenů v řadě nevyhrává

Dalším důležitým pravidlem je, že vítězná pětice kamenů nesmí být blokovaná z obou stran soupeřovými kameny (Obr. 3).




Obrázek 3. Pět blokovaných kamenů v řadě

Těmito pravidly se hra stává hráče s bílými vyrovnanější a dává mu mnohem víc prostoru pro obranu.

# 2 NEURONOVÉ SÍTĚ

Patří do oblasti strojového učení a bývají označovány jako distribuované paralelní výpočetní systémy.
Inspiraci našly v našem mozku a v biologických procesech, které v něm probíhají. Jejich největší předností je schopnost učit se a zobecňovat.
V této práci bude nastíněn jen lehký úvod do této obsáhlé problematiky.

## 2.1 Použití

Neuronové sítě se používají jako univerzální aproximátor. Můžeme tedy díky nim aproximovat jakoukoliv funkci. Typickými úlohami neuronových sítí jsou klasifikace a regrese.

### 2.1.1 Klasifikace

Na základě vytvořeného modelu, dokáže síť zařazovat vstupní data do předem připravených kategorií. Na výstupu tedy bude pravděpodobnost té či oné kategorie.

Používá se například u detekce spamu, diagnostiky nemocí či rozpoznávání objektů atd.

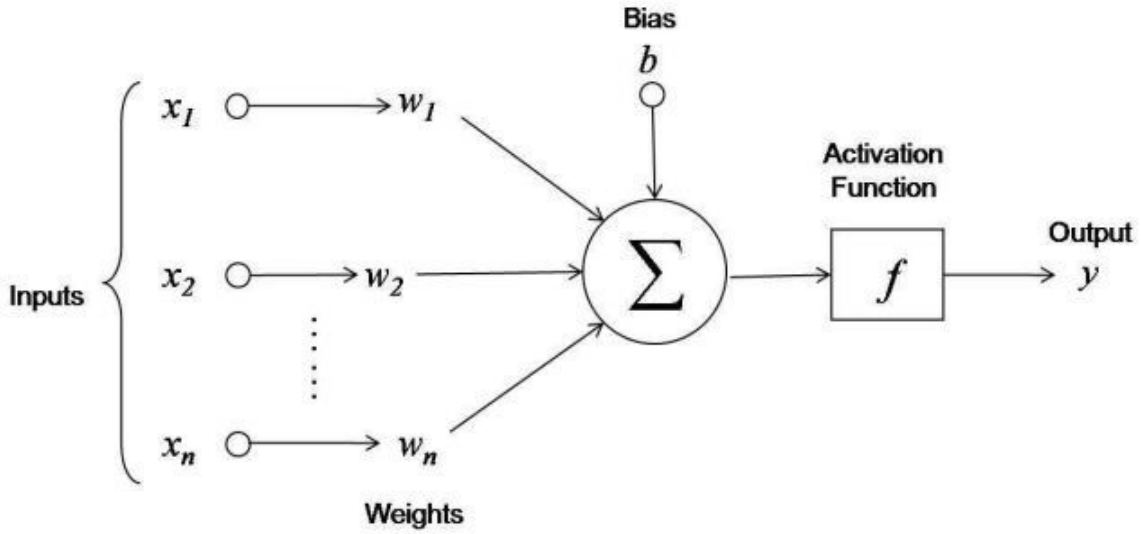
### 2.1.2 Regrese

Model sítě dokáže předpovídat vztah mezi dvěma proměnnými, kdy jedna je závislá a druhá nezávislá proměnná. Na výstupu takovéto sítě bude číslo.

Regresi je možné použít například v predikci cen nemovitostí, na základě toho, jak se vyvíjely ceny v minulosti.

## 2.2 Neuron

Základním stavebním prvkem neuronové sítě jsou neurony, které na vstupu přivedený signál transformují pomocí přenosové funkce a posílají takto upravený signál dál. Neuron může mít více vstupů, ale vždy jen jeden výstup. Nejjednodušší neuronovou sítí je Perceptron (Obr. 4), který obsahuje právě jeden neuron.




Obrázek 4. Model umělého neuronu [7]

Vstupní vektor $\vec{x}= (x_1 \dots x_n)$ je přiveden na vstup sítě a je vynásoben vektorem vah $\vec{w}= (w_1 \dots w_n)$. Váhy přikládají důležitost jednotlivým vstupům. Čím je hodnota vyšší, tím bude mít daný vstup větší vliv na celkový výstup sítě. Většinou síť obsahuje ještě jeden další vstup ve formě prahu (bias), který ovlivňuje to, jestli bude nebo nebude neuron aktivován. Součet takto upravených vstupů je přiveden do aktivační (přenosové) funkce *f* a odtamtud putuje transformovaný signál na výstup neuronu, kde většinou pokračuje na vstup dalšího neuronu (Vzorec 1).

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

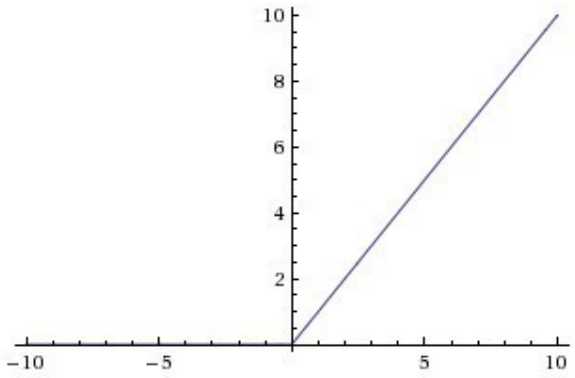
Vzorec 1. Výpočet výstupu neuronu

### 2.2.1 Přenosové funkce

Přenosové funkce přidávají do neuronových sítí nelinearitu. Nejčastější používané funkce jsou ReLU, sigmoida, nebo hyperbolický tangens.

**ReLU** (Rectified linear unit, Obr. 5) transformuje všechny negativní hodnoty na nulu. Je výpočetně nenáročná a snadno ji lze zderivovat.

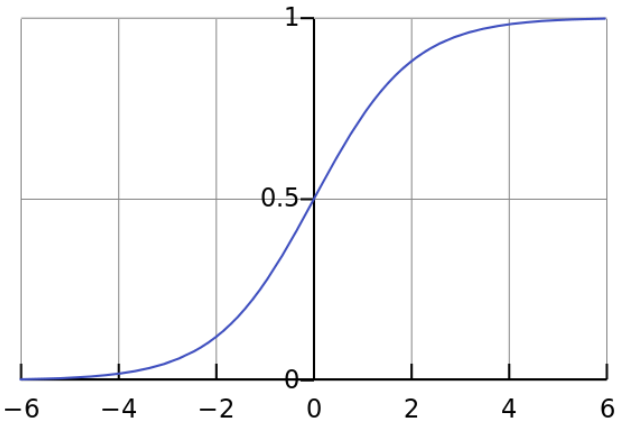
$$\sigma(v) = \max(0, v)$$




Obrázek 5. ReLU [8]

**Sigmoida** (Obr. 6) transformuje hodnoty do intervalu $(0,1)$ .

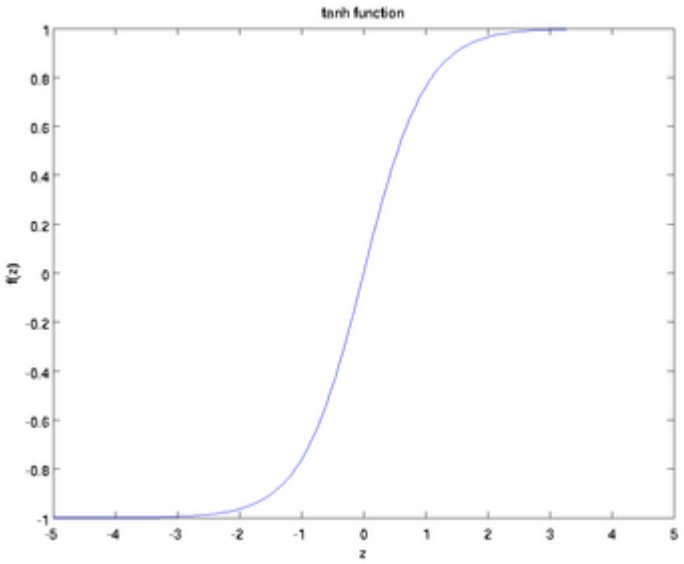
$$\sigma(v) = \frac{1}{1 + e^{-av}}$$




Obrázek 6. Sigmoida [8]

**Hyperbolický tangens** (Obr. 7) transformuje hodnoty do intervalu $(-1,1)$ .

$$\sigma(v) = \tanh(v)$$




Obrázek 7. Hyperbolický tangens [8]

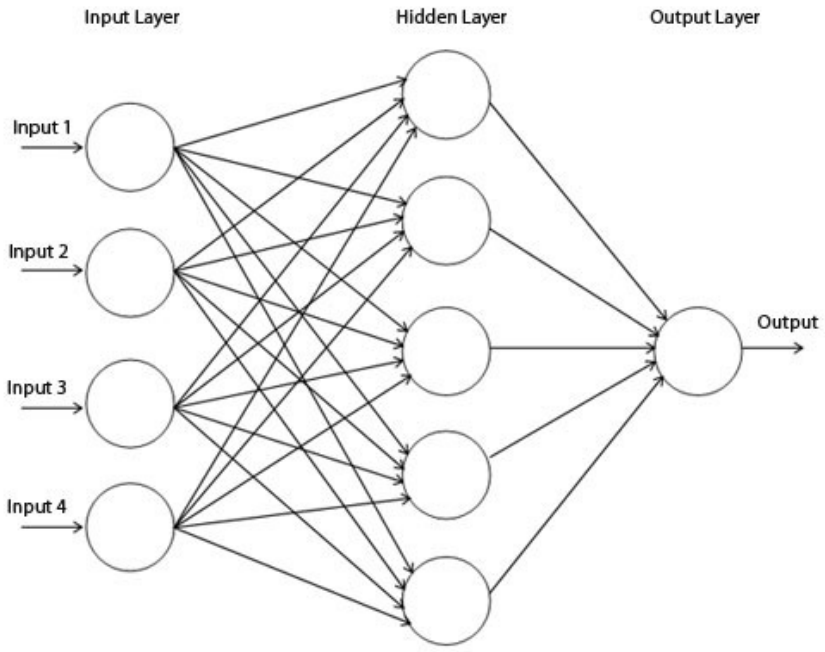
**Softmax** (Vzorec 2) každou komponentu vstupního vektoru transformuje do intervalu $(0,1)$, kdy celkový součet těchto hodnot je jedna. Používá se hlavně u klasifikačních úloh.

$$f(x)_i = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}} pro\ i = 1, \dots, N$$

Vzorec 2. Softmax funkce

## 2.3 Topologie

Vzájemné propojení více neuronů nám charakterizuje topologii sítě. Nejčastěji se neurony propojují do vrstev, přičemž v každé vrstvě je určitý počet neuronů.




Obrázek 8. Propojení vrstev neuronové sítě [9]

V takovéto propojené sítí existuje vždy jedna vstupní vrstva (input layer), libovolný počet skrytých vrstev (hidden layers) a jedna výstupní vrstva (output layer), Obr. 8. Pokud informace proplouvá sítí pouze v jednom směru a to od vstupní vstvy po výstupní, mluvíme o dopředné síti (feedforward network). Pokud síť naopak obsahuje nějakou zpětnou vazbu, jedná se o síť rekurentní (recurrent neural network - RNN).

## 2.4 Učení

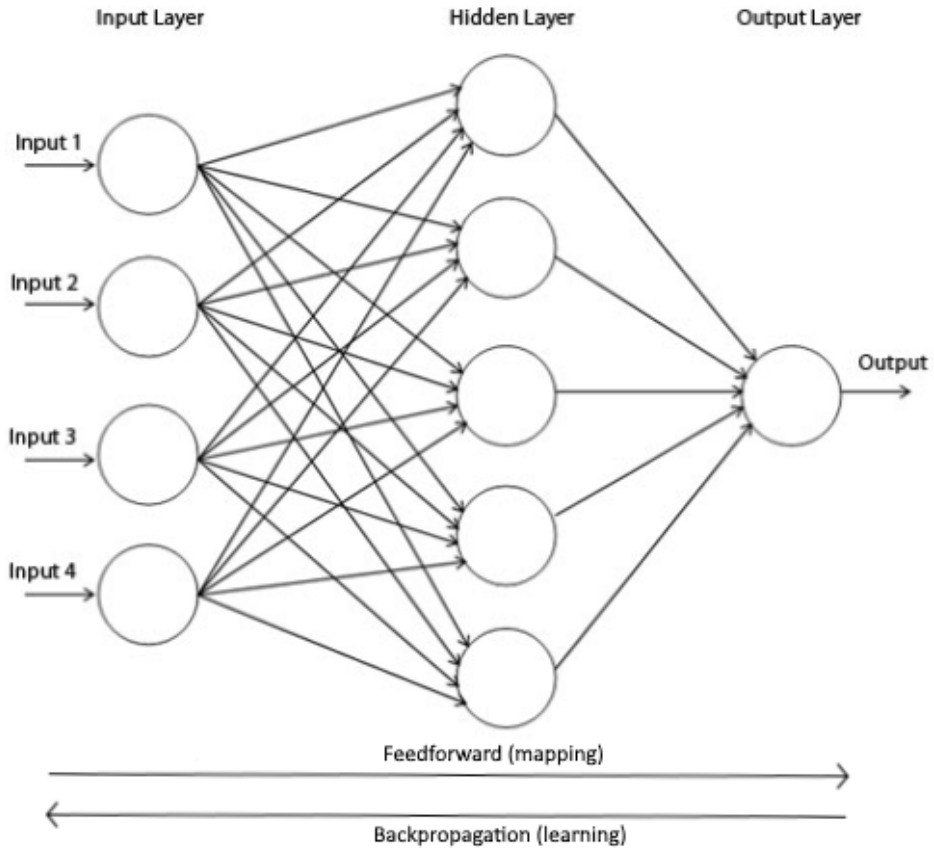
Je proces, kdy se postupně upravují synaptické váhy a prahy jednotlivých neuronů, dokud síť nevrací požadované výstupy. Mezi hlavní způsoby učení patří učení bez učitele, učení s učitelem a zpětnovazební učení, které je někde na pomezí těchto dvou metod.

### 2.4.1 Učení bez učitele

Jsou známy pouze vstupní hodnoty, ale chybí jim odpovídající výstupy. Síť si proto vytváří skupiny (clustery) na základě vzájemné podobnosti. Patří sem zejména shluková analýza, nebo samoorganizující se mapy.

### 2.4.2 Učení s učitelem

V učení s učitelem se nejčastěji používá algoritmus zpětného šíření chyby (backpropagation). Na začátku trénování sítě exitují jak vstupní hodnoty, tak i jejich požadované výstupy. Takováto množina uspořádaných dvojic $\{(x_1 d_1), \ldots, (x_N d_N)\}$ se nazývá trénovací data, která bývají normalizovaná na hodnoty v intervalu $\langle 0,1 \rangle$. Tuto množinu většinou nastavuje expert na danou oblast a je klíčová pro úspěšné natrénování sítě. Kromě trénovací množiny se používá ještě validační množina, která zabraňuje přeučení (overfitting) a množina testovací sloužící k otestování sítě. Přeučení sítě nastává v momentě, kdy je síť sice přizpůsobená trénovacím datům, ale selhává na validační a testovací množině.




Obrázek 9. Vybavovací a adaptační fáze učení [9]

Samotné trénování probíhá tak, že na vstup sítě (input layer), která je inicializovaná náhodnými vahami, přijde vstupní vektor hodnot z trénovací množiny. Síť vstup zpracuje a na výstupu (output layer) se nám vrátí nějaká hodnota. Této fázi se říká vybavovací (feedforward), nebo také aktivační (Obr. 9).

Dalším krokem, který následuje je výpočet chyby sítě. Ta je definována jako rozdíl mezi požadovanou hodnotou a výstupní hodnotou sítě. K tomuto účelu se nejčastěji používá funkce Mean Squared Error (MSE), jež je rovna výpočtu sumy rozdílu středních hodnot čtverců (Vzorec 3).

$$E = \frac{1}{2}\sum_{i}^{N}\sum_{j}^{n}\left(y_{ij} - d_{ij}\right)^2$$

Vzorec 3. Mean Squared Error

Jestliže je známa velikost chyby, je potřeba ji propagovat sítí zpět a upravit podle ní váhy jednotlivých neuronů. Zde nastupují optimalizační metody, které se snaží správným nastavením vah minimalizovat chybovou funkci E. Typickým zástupcem je metoda gradientního sestupu (gradient descent) počítající nové hodnoty vah pomocí parciálních derivací (Vzorec 4).

$$\theta_i(t+1) = \theta_i(t) - \alpha\frac{\delta E}{\delta \theta_i}$$

Vzorec 4. Výpočet nových hodnot vah sítě

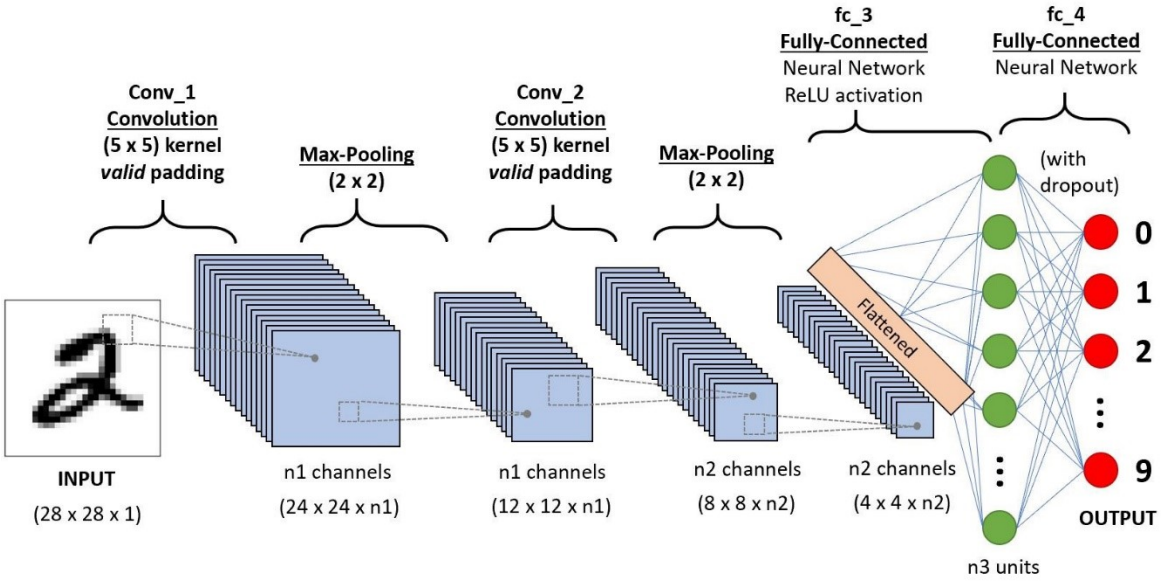
Kde $\theta_i(t)$ je i-tá váha v čase $t$ a α je parametr učení (learning rate), který určuje velikost kroku. V praxi se potom častěji používá vylepšená stochastická gradientní metoda (SGD), nebo metoda, která počítá adaptivní parametr učení pro každou váhu zvlášť Adam (Adaptive Moment Estimation) [10]. Této části, kdy se chyba zpětně propaguje sítí a přenastavují se váhy jednotlivých neuronů se říká adaptační (učící) fáze.

Výše popsaný proces se cyklicky opakuje, dokud neklesne chyba sítě pod stanovenou mez.

## 2.5 Konvoluční neuronové sítě

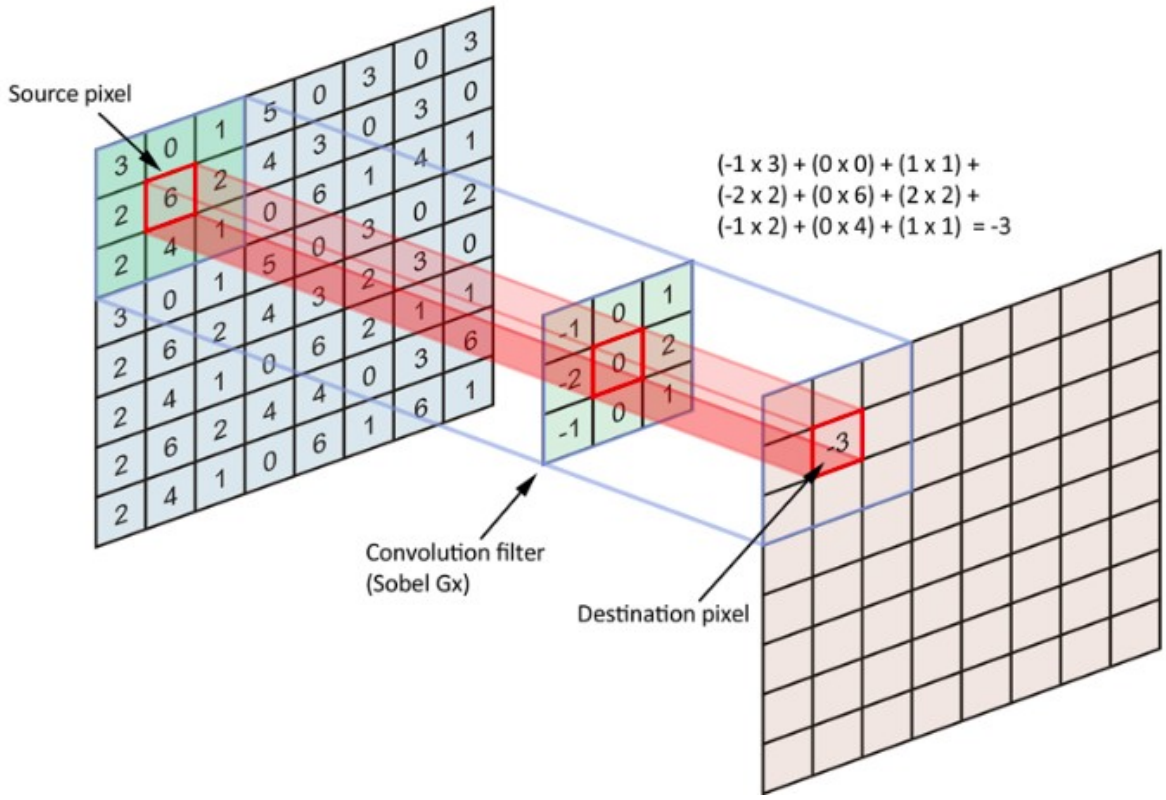
Konvoluční neuronové sítě (ConvNet, CNN) jsou nejčastěji využívány v aplikacích počítačového vidění. Od hustě propojených sítí, kdy jsou všechny neurony jedné vrstvy propojeny se všemi neurony následující vrstvy se liší tím, že obsahují takzvané konvoluční a poolingové vrstvy (Obr. 10). Díky těmto vrstvám, je síť schopna zpracovávat velká vstupní data s pomocí mnohem menšího množství parametrů. Důležitou charakteristikou konvoluční sítě je schopnost naučit se prostorové hierarchii, kdy každá konvoluční vrstva se postupně

učí složitější a složitější vzory a také to, že vzory, které se síť naučí jsou prostorově invariantní, což znamená, že naučené vzory síť nalezne kdekoliv v obrázku [11]. Jelikož síť zpracovává hlavně obrazová data, očekává na svém vstupu vždy vektor ve tvaru (výška, šířka, hloubka). V praxi často obsahuje ještě jeden vektor a tím je velikost dávky. Pokud budeme mít na vstupu například černobílý obrázek s ručně napsanou číslicí dva definovaný svou výškou a šířkou v pixelech, potom hloubka bude jedna (u barevného obrázku by byla tři) a velikost dávky bude také jedna. Jestliže bychom na vstup přivedli těchto obrázku třeba deset, pak by byla velikost dávky deset.




Obrázek 10. Konvoluční neuronová síť [12]

Takovýto obrázek vstupuje do konvoluční vrstvy, kde jsou na něho postupně aplikovány konvoluční filtry (Obr. 11). Filtr si lze představit jako malé okno (třeba o velikosti 3x3). Filtr má za cíl z obrázku vyextrahovat nějaké zajímavé vlastnosti. Postupným posouváním okna po obrázku, kdy velikost posunu určuje krok (stride) a aplikací stejného filtru na každou takovou pozici, získáme na výstupu mapu příznaků (feature map). Výsledná mapa příznaků bude mít stejné rozměry jako vstupní obrázek, avšak hloubka (počet kanálů) se bude rovnat počtu aplikovaných filtrů.




Obrázek 11. Aplikace konvolučního filtru [13]

Pro snížení velikosti obrázku lze aplikovat sub-sampling (pooling) vrstvu. Ta funguje podobně jako operace konvoluce, kdy se aplikuje na všechny části obrázku. Na rozdíl od konvoluce by se ale okna při posunu neměla překrývat. Nejčastěji se používá max-pooling vrstva, kdy z každého takového okna je vybrána vždy ta nejvyšší hodnota.

Střídáním konvolučních a max-poolingových vrstev docílíme zmenšení vstupního obrázku, který potom můžeme napojit na plně propojené vrstvy.

# 3 ZPĚTNOVAZEBNÍ UČENÍ

*Reinforcement learning*, v překladu posilované učení, vychází z behaviorální psychologie, kdy se zvířata učí formou odměn a trestů [14]. Požadované chování je tedy odměňováno, kdežto chování, které chceme potlačit, trestáme. Pokud například po psovi chceme, aby si sedl, tak v momentě, kdy příkaz provede, dostane pamlsek. Tím si pes spojí provedený příkaz s následnou odměnou. Pokud nechceme, aby po nás skákal, tak ho můžeme okřiknout. Pes si takto zapamatuje, že takovéto chování není žádoucí. Samozřejmě je potřeba toto opakovat, dokud se to pes nenaučí. Obdobné je to u zpětnovazebního učení, kdy stylem pokus-omyl a získáváním zpětně vazby z prostředí, dochází k procesu učení.

## 3.1 Základní pojmy

Pro snadnější orientaci v problematice zpětnovazebního učení jsou v podkapitolách 3.1.1–3.1.5 popsány a vysvětleny základní pojmy

### 3.1.1 Agent

Agent je vše, co vnímá pomocí svých senzorů okolní prostředí a zároveň ho ovlivňuje díky svým akčním členům. Může to být například člověk, který má oči, nos, uši a další orgány sloužící jako senzory a ruce, nohy, ústa a jiné části těla, které představují akční členy, nebo robot s kamerou a infračervenými čidly, které plní funkci senzorů a akčním členem může být třeba nějaký druh motoru a tak dále. V prostředí se může nacházet více agentů, kteří se mohou navzájem ovlivňovat. Mluvíme potom o kooperativních agentech, jež spolu spolupracují, nebo o kompetitivních agentech, kteří mezi sebou soutěží.

### 3.1.2 Prostředí

Prostředí je cokoliv, s čím agent přichází během své činnosti do styku. Může to být třeba bludiště, počítačová hra, akciový trh a tak dále. V každém časovém okamžiku se prostředí nachází v určitém stavu.

Prostředí rozlišujeme na:

- Diskrétní, nebo spojitá.
- Deterministická, nebo stochastická.
- Plně pozorovatelná, nebo částečné pozorovatelná.
- V prostředí se nachází pouze jeden agent, nebo jich tam je více (multiagentní).

- Epizodická, nebo sekvenční.
- Statické, nebo dynamické.

### 3.1.3 Model

Model prostředí napodobuje chování prostředí. Pro daný stav a akci může model odhadovat následující stav a získanou odměnu.

### 3.1.4 Akce

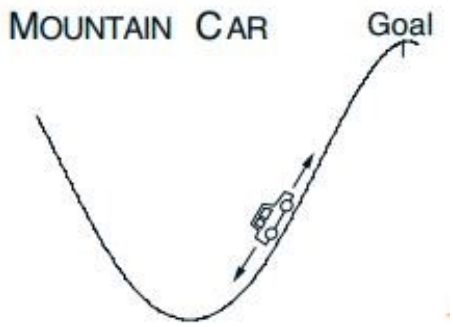
Jsou úkony, které agent vykonává a které způsobují změny v prostředí.

### 3.1.5 Odměna

Je číselné ohodnocení, které agenta informuje o tom jak dobrá, nebo špatná byla ním provedená akce. Odměna může být kladná, což znamená chtěné chování, nebo záporná, které indikuje nežádoucí chovaní.

## 3.2 Příklad

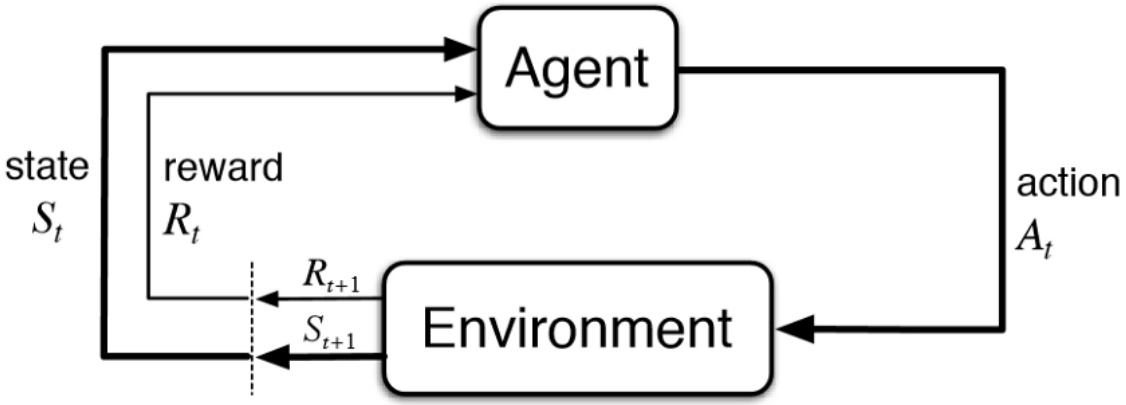
Typickým úkolem zpětnovazebního učení je Mountain Car (Obr. 12). Auto, které se nachází v údolí se z něj snaží dostat ven. Jenže proto, aby vyjelo, nemá dostatečně silný motor. Proto je potřeba auto postupnými pohyby dopředu a dozadu rozhýbat a tím získat rychlost potřebnou pro úspěšné vyjetí z údolí. Stavem jsou potom poloha a rychlost auta. V každém stavu auto může provádět tři akce. Pohyb dopředu, pohyb dozadu, nebo neprovádět nic. Za každý krok v údolí získává agent odměnu -1 a pokud se mu podaří vyjet z údolí tak 0.




Obrázek 12. Mountain car [15]

## 3.3 Popis

V agentních systémech se díky zpětnovazebnímu učení snažíme naučit agenta takovému chování, které maximalizuje jeho celkovou odměnu, kterou dostává z prostředí za akce, které v něm provádí.




Obrázek 13. Princip zpětnovazebního učení [16]

Na obrázku (Obr. 13) se nalézá agent ve stavu *s* a provádí akci *a*, tím získává od prostředí odměnu *r* a zároveň se dostává do nového stavu $s_{+1.}$ Tento proces se opakuje, dokud agent nedorazí do nějakého cílového stavu.

Takovéto prostředí, ve kterém se agent pohybuje, můžeme formálně popsat díky ruskému matematikovi Andreji Markovovi a jeho rozhodovacích procesech MDP.

### 3.3.1 Markovův rozhodovací proces

Je matematický rámec pro modelování chování agenta v stochastickém prostředí. Tam, kde výsledek závisí z části na náhodě a z části na tom, jak se agent rozhodne, lze aplikovat Markovův rozhodovací proces (MDP). Důležitou vlastností tohoto rozhodovacího procesu je nezávislost aktuálního stavu a akce na stavech a akcích předchozích. Pomocí MDP lze simulovat mnoho důležitých náhodných systémů, včetně biologických, epidemiologických, finančních a fyzických. V současnosti nacházejí rozhodovací procesy uplatnění v robotice, automatizaci, ekonomice, výrobě a v dalších disciplínách. [17]

MDP tvoří čtveřice (S, A, P, R), kde:

- S je konečná množina stavů prostředí.
- A je konečná množina akcí proveditelných v daném prostředí. Množina A může být omezená daným stavem, to znamená, že ne ve všech stavech musí být všechny akce stejné.
- $P_a(s, s')$ je přechodová funkce, která udává pravděpodobnost, že akcí *a* ve stavu *s* přejde prostředí do stavu *s´*.
- $R_a(s, s')$ je funkce odměn, která na základě provedení akce *a* ve stavu *s* vrátí okamžitou odměnu.

Agenti většinou nečiní jen jedno rozhodnutí, ale provádí jich více. Jak se bude agent v prostředí chovat nám určuje strategie.

### 3.3.2 Strategie π

Strategie (policy) π nám udává pravděpodobnost provedení akce *a* ve stavu *s*.

$$\pi : S\ x\ A\ \rightarrow [0, 1]$$

Cílem zpětnovazebního učení je maximalizovat celkovou odměnu, kterou agent od prostředí získá. Celkovou odměnu, kterou takto agent získá spočítáme jako prostý součet všech odměn:

$$R = r_t + r_{t+1} + \cdots + r_n$$

Je důležité si uvědomit, že stejná sekvence akcí, nemusí vždy přinést stejnou celkovou odměnu, protože se nacházíme ve stochastickém prostředí. Proto je potřeba do rovnice přidat tzv. diskontní faktor (discount factor) $\gamma$, který nám rozmělňuje odměny budoucích stavů.

$$R = r_t + \gamma^1 r_{t+1} + \cdots + \gamma^{n-t} r_n$$

Jak už jsme si řekli, cílem zpětnovazebního učení je maximalizovat kumulativní odměnu. K tomu je potřeba nalézt optimální strategii $\pi^*$. Abychom takovouto strategii nelezli, musíme napřed definovat tzv. hodnotící funkce a těmi jsou hodnota stavu a hodnota akce.

**Hodnota stavu** *V* (state-value function) určuje jak moc je pro agenta z dlohodobého hlediska výhodné navštívit určitý stav. Jinými slovy, jaká bude jeho kumulativní odměna, pokud začne ve stavu *s* a bude se držet strategie $\pi$ (Vzorec 5).

$$V^{\pi}(s) = E_{\pi}[R_t|S_t = s] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^t R_{t+k+1}|S_t = s\right]$$

Vzorec 5. Výpočet hodnoty stavu

**Hodnota akce** *Q* (action-value function) definuje jak moc je výhodné ve stavu *s* provést akci *a*, pokud se agent bude držet strategie $\pi$ (Vzorec 6). V tomto případě není potřeba znát model prostředí, protože se vliv akcí na prostředí učí agent za pochodu.

$$Q^{\pi}(s,a) = E_{\pi}[R_t|S_t = s, A_t = a] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]$$

Vzorec 6. Výpočet hodnoty akce stavu

**Optimální strategie**, je taková strategie, která ze všech možných strategií ve stavu *s* vrací nejvyšší kumulativní odměnu.

Pro funkci hodnoty stavu:

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

Pro funkci hodnoty akce:

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

Díky Bellmanovu principu optimality, kdy při hledání optimálního řešení můžeme problém rozdělit na dílčí podproblémy a hledat optimální řešení v nich [18], můžeme rovnice přepsat do tvaru:

Pro funkci hodnoty:

$$V_*(s) = max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_*(s')]$$

Pro funkci akce:

$$Q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma max_a Q_*(s',a')]$$

Optimální řešení potom získáme iterací nad jednou z těchto rovnic.

### 3.3.3 Strategie prohledávání

Hodnotu stavu $V$ i hodnotu akce $Q$ může agent použít k tomu, aby vylepšil svoji strategii. Jenže na začátku nezná hodnoty jednotlivých stavů. Pokud by tedy vybíral jenom akce, které budou přinášet nejvyšší užitek (exploatace), nebude tím pádem prohledávat stavový prostor a nemusí tudíž najít nějaké lepší řešení. Pokud by naopak vybíral jen náhodné akce (explorace), nebude jeho prohledávání příliš efektivní. Proto se využívá kombinace těchto dvou přístupů.

**$\varepsilon -$ greedy** metoda prohledávání vybere v daném stavu s pravděpodobností $(1 - \varepsilon)$ tu nejlepší akci a s pravděpodobností $\varepsilon$ vybere náhodnou akci. Z počátku, kdy není stavový prostor znám, je vhodné vybírat jen náhodné akce a postupně s nabitými znalostmi daného prostředí lze volit již naučené akce. Použitím tohoto postupu je možné dosáhnout velmi dobrých výsledků.

## 3.4 Rozdělení zpětnovazebního učení

### 3.4.1 Aktivní vs pasivní učení

V pasivním učení je předem dána určitá strategie, kterou se agent řídí, ale chybí přechodová funkce a ohodnocení stavů. V pasivním učení se tedy agent snaží vyhodnotit kvalitu používané strategie na základě očekávaného užitku.

V aktivním učení naopak není strategie známa, proto se agent snaží prozkoumávat prostředí a strategii se naučit na základě odměny, kterou v procházených stavech získá.

### 3.4.2 Podle znalosti modelu

Učení založené na znalosti modelu (**model-based**) využívá přechodovou funkci ke svému rozhodování, přičemž jsou známy odměny v jednotlivých stavech. Naopak bez znalosti modelu (**model-free**) je třeba prostor v každém kroku prozkoumávat a rozhodovat se na základě zkušeností.

### 3.4.3 Podle druhu strategie

**On-policy** využívá a vylepšuje pouze jednu strategii, zatímco **off-policy** metoda může používat současně více strategií např. Ɛ-greedy.

## 3.5 Q-učení

> *Q-learning is a simple way for agents to learn how optimally in controlled Markovian domains.*
>
> [19]

Z anglického Q-learning, kde ´Q´ znamená kvalita (quality). Patří mezi nejpopulárnější metody zpětnovazebního učení. Metodu popsal Christhoper Watkins v roce 1989.

Q-learning je kroková metoda aktivního učení bez pevně dané strategie (off-policy), která nepotřebuje ke své činnosti model prostředí. Metoda se snaží najít optimální strategii na základě funkce hodnoty akce *Q*, určující kvalitu konkrétní akce v daném stavu. Je dokázáno, že za určitých podmínek algoritmus konverguje k optimálnímu řešení. [14]

Metoda je založena na vyhledávací tabulce (lookup-table), ve které existují k jednotlivým akcím a stavům uložené Q hodnoty, jenž reprezentují již zmiňovanou kvalitu akcí v daných stavech.

Tabulka 1. Vyhledávací tabulka pro Q-učení

| | Akce | | | |
|---|---|---|---|---|
| Stavy | a0 | a1 | a2 | … |
| s0 | $Q(s_0,a_0)$ | $Q(s_0,a_1)$ | $Q(s_0,a_2)$ | … |
| s1 | $Q(s_1,a_0)$ | $Q(s_1,a_1)$ | $Q(s_1,a_2)$ | … |
| s2 | $Q(s_2,a_0)$ | $Q(s_2,a_1)$ | $Q(s_2,a_2)$ | … |
| … | … | … | … | … |

Formálně lze Q-learning popsat:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)],$$

kde $0 < \alpha < 1$ je učící konstanta, která udává, jak moc se bude původní Q hodnota měnit vzhledem k aktuální Q hodnotě a $\gamma$ je diskontní faktor.

Princip je jednoduchý. Aktuální Q hodnota se aktualizuje na základě aktuální odměny $R_t$ a rozmělněné očekávané odměny $\gamma max_a Q(S_{t+1}, a)$, kterou bychom získali, pokud bychom v následujícím stavu vybrali tu nejlépe hodnocenou akci.

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$;
  until $S$ is terminal

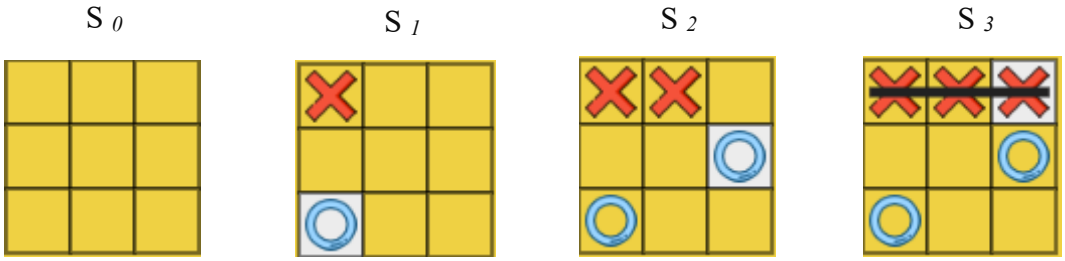
Obrázek 14. Pseudokód Q-learning algoritmu [14]

Algoritmus (Obr. 14) funguje tak, že na začátku inicializujeme tabulku libovolnými hodnotami. Obvykle to bývají nuly. Agent se následně pohybuje v prostředí na základě své strategie např. $\varepsilon - greedy$ a v každém kroku aktualizuje tabulku nově spočítanou $Q$ hodnotou danou stavem *s* a akcí *a*.

### 3.5.1 Příklad výpočtu na hře Piškvorky

Q-learning algoritmus nejlépe pochopíme na názorném příkladu. Proto si ukážeme postup učení pomocí této metody na hře piškvorky, která je vlastně variantou hry Gomoku, kdy se hraje na hrací desce o velikosti 3x3.

Agent (X) hraje piškvorky (Obr. 15) proti hráči (O), který provádí jen náhodné tahy. Tento agent se řídí svou strategií a tahy vybírá podle ní (např. $\varepsilon - greedy$).




Obrázek 15. Posloupnost odehraných tahů z pohledu hráče X

Dokud nenastane konec hry, není známa ani odměna za provedené akce v procházených stavech. Agent odměnu získá až na konci.

Tabulka 2. Odměny

| Výhra | Prohra | Remíza |
|---|---|---|
| +1 | -1 | 0 |

V naší odehrané partii agent, protože spojil tři stejné symboly, získá ve stavu *s2* za akci *a3* odměnu +1. Pro snazší pochopení jsou stavy očíslované 0 až 3 (Obr. 15). Samozřejmě jsou mezi nimi i další stavy. Určující jsou takové stavy, které obsahují i odpověď protihráče.

Nejdříve je nutné nastavit některé parametry učení. Tyto parametry si nastavíme například takto:

$$\alpha = 0.2; \gamma = 0{,}9$$

Poté začne samotný proces učení, tzn. aktualizace Q hodnot v tabulce na základě vzorce:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Na začátku je tabulka (Tabulka 3) vyplněna samými nulami.

Tabulka 3. Tabulka počátečních hodnot

| | Akce | | | | |
|---|---|---|---|---|---|
| Stavy | $a_0$ | $a_1$ | $a_2$ | $a_3$ | … |
| $s_0$ | 0 | 0 | 0 | 0 | 0 |
| $s_1$ | 0 | 0 | 0 | 0 | 0 |
| $s_2$ | 0 | 0 | 0 | 0 | 0 |
| $s_3$ | 0 | 0 | 0 | 0 | 0 |
| … | 0 | 0 | 0 | 0 | 0 |

Použijeme přitom poslední stav $s_2$, který vedl k výhře a vypočteme jeho novou Q hodnotu pro akci $a_2$, kterou agent v tomto stavu provedl a aktualizujeme tabulku (Tabulka 4).

$$Q_{(s2,a2)} = 0 + 0{,}2(1 + 0 - 0) = 0{,}2$$

Tabulka 4. Tabulka po prvním kroku výpočtu

| | Akce | | | | |
|---|---|---|---|---|---|
| Stavy | $a_0$ | $a_1$ | $a_2$ | $a_3$ | … |
| $s_0$ | 0 | 0 | 0 | 0 | 0 |
| $s_1$ | 0 | 0 | 0 | 0 | 0 |
| $s_2$ | 0 | 0 | 0,2 | 0 | 0 |
| $s_3$ | 0 | 0 | 0 | 0 | 0 |
| … | 0 | 0 | 0 | 0 | 0 |

Nyní přejdeme do předcházejícího stavu $s_1$ a pro akci $a_1$ vypočteme novou Q hodnotu.

Opět aktualizujeme tabulku (Tabulka 5).

$$Q_{(s1,a1)} = 0 + 0{,}2(0 + 0{,}9.0{,}2 - 0) = 0{,}036$$

Tabulka 5. Tabulka po druhém kroku výpočtu

| | Akce | | | | |
|---|---|---|---|---|---|
| Stavy | $a_0$ | $a_1$ | $a_2$ | $a_3$ | … |
| $s_0$ | 0 | 0 | 0 | 0 | 0 |
| $s_1$ | 0 | 0,036 | 0 | 0 | 0 |
| $s_2$ | 0 | 0 | 0,2 | 0 | 0 |
| $s_3$ | 0 | 0 | 0 | 0 | 0 |
| … | 0 | 0 | 0 | 0 | 0 |

Takto se dostaneme až do stavu $s_0$, který je počáteční. Postup znovu zopakujeme a aktualizujeme vypočtenou hodnotu pro akci $a_0$ (Tabulka 6).

$$Q_{(s0,a0)} = 0 + 0{,}2(0 + 0{,}9.0{,}036 - 0) = 0{,}00648$$

Tabulka 6. Tabulka v poledním kroku výpočtu

| | Akce | | | | |
|---|---|---|---|---|---|
| Stavy | $a_0$ | $a_1$ | $a_2$ | $a_3$ | … |
| $s_0$ | 0,00648 | 0 | 0 | 0 | 0 |
| $s_1$ | 0 | 0,036 | 0 | 0 | 0 |
| $s_2$ | 0 | 0 | 0,2 | 0 | 0 |
| $s_3$ | 0 | 0 | 0 | 0 | 0 |
| … | 0 | 0 | 0 | 0 | 0 |

Abychom tabulku naplnili celou, musíme tento tréninkový proces mnohokrát opakovat.

## 3.6 Hluboké Q-učení

Nevýhodnou předchozího přístupu, ve kterém používáme k uložení $Q$ hodnot tabulku, je neefektivní, nebo dokonce nemožné spravovat takovou tabulku v případě velkého počtu stavů, nebo akcí.

V roce 2013 firma DeepMind vyřešila tuto situaci konceptem nahrazení tabulky neuronovými sítěmi a pojmenovala jej hluboké Q-učení (DQN).

Neuronové sítě ale také přinášejí do procesu učení určitou nestabilitu a snadno se může stát, že se přeučí. Aby se tomu předešlo, používají se různá vylepšení jako jsou vzpomínková paměť, cílová síť, nebo dvojité DQN.

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
**for** episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  **for** $t = 1, T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  **end for**
**end for**

Obrázek 16. Vylepšený DQN algoritmus využívající vzpomínkovou paměť [20]

Algoritmus (Obr. 15) na začátku náhodně inicializuje váhy sítě. Agent začne procházet prostředí a sbírat zkušenosti, které se ukládají do vzpomínkové paměti. Jakmile je paměť naplněná, vezme z paměti malé náhodné množství vzpomínek, které se používají pro trénování neuronové sítě. Jedná se o učení s učitelem, protože na vstupu máme vektor stavů a na výstupu vektor odhadovaných Q hodnot.

### 3.6.1 Vzpomínková paměť

Vzpomínková paměť (experience replay) je vlastně takový buffer, do kterého se ukládá tato čtveřice (*s,a,r,s´*), která reprezentuje zkušenost agenta.

*s* … Stav, ve kterém se agent nacházel.

*a* … Akce, kterou agent ve stavu *s* provedl.

*r* … Odměna, kterou agent obdržel.

*s´*… Nový stav, do kterého se agent přesunul.

Hlavním účelem paměti je zabránit případným závislostem mezi daty, protože některé stavy si mohou být velmi podobné a nepříznivě by ovlivňovaly nastavení vah neuronové sítě [21]. Paměť má určitou předem nastavenou kapacitu a plní se postupně tak, jak agent prochází prostředí. Než se naplní, je proces učení pozastavený. Jakmile je paměť plná, náhodně se z paměti vybere malá dávka (batch) vzpomínek, které jsou pak použity pro trénování. Dávkové zpracování má za následek zrychlení procesu učení. V každém agentově kroku jsou pak staré vzpomínky postupně nahrazovány těmi novými.

### 3.6.2 Cílová síť

Cílová síť (target network) má za úkol stabilizovat proces učení. Jedná se vlastně o identickou kopii primární sítě (online network), která má ale uzamčené váhy a k její aktualizaci dochází jen jednou za čas. Aktualizace může probíhat buď tak, že se po čase překopírují všechny váhy z první sítě do cílové, nebo se váhy můžou v cílové síti upravovat v každém kroku po malých kouscích. Cílová síť je potom použita pro odhadování kvality jednotlivých akcí během trénování. [22]

Aktualizace hodnot potom probíhá podle:

$$Q(s, a; \theta) \leftarrow R + \gamma maxQ(s', a'; \theta^{-}),$$

kde $\theta$ jsou váhy aktuální a $\theta^{-}$ váhy cílové sítě.

### 3.6.3 DDQN

Dvojitá DQN jde ještě dál a využívá cílovou síť k predikci samotných hodnot Q funkce, zatímco primární síť má za úkol vybrat akci s nejlepším ohodnocením. Je to proto, že primární síť má nejaktuálnější odhady a cílová síť zase nejstabilnější hodnoty. [23]

$$Q(s, a; \theta) \leftarrow R + \gamma maxQ(s', argmaxQ(s', a'; \theta); \theta^{-})$$

# II. PRAKTICKÁ ČÁST

# 4 IMPLEMENTACE Q-ALGORITMU

V této kapitole je popsána praktická část diplomové práce, ve které bylo cílem implementovat Q-algoritmus do zvolené počítačové hry. Tou je hra Gomoku. U ní se soustředíme hlavně na zvolený algoritmus a zpětnovazební učení. Gomoku je desková hra o mnoha variantách. V této práci bude praktická část věnována její rozšířené verzi, tak aby hra byla pro oba hráče vyrovnanější. V rámci hry je například možné měnit velikost herní desky, hrát s různými enginy, trénovat agenta či sledovat případně ovlivňovat průběh jeho učení.

## 4.1 Použité knihovny

K usnadnění vývoje bylo použito několik knihoven. Mezi nejzajímavější patří zejména React, Createjs a Tensorflow.js., jež jsou popsány v následujících podkapitolách.

### 4.1.1 React

Javascriptová knihovna určená k tvorbě uživatelského rozhraní za kterou stojí firma Facebook. Aplikace napsané v Reactu používají tzv. komponenty, které mají své vlastnosti, stav a co je důležité, jsou znovupoužitelné na různých místech v aplikaci,

### 4.1.2 Createjs

Modulární knihovna a nástroje, kterými můžeme vytvářet interaktivní webové stránky. Oficiálně je sponzorovaná firmami Adobe, Microsoft a Mozzila. Tato knihovna byla použita pro vytvoření hrací desky presentované v praktická části této práce.

### 4.1.3 Tensorflow.js

Je velmi populární knihovna pro strojové učení, za kterou stojí tým Brain z firmy Google. Základní jednotkou je Tensor, který si můžeme představit jako vícenásobné pole nad kterým můžeme provádět různorodé matematické operace, které je framework schopen vykonávat na grafické kartě díky rozhraní WebGL.

## 4.2 Proč Javascript?

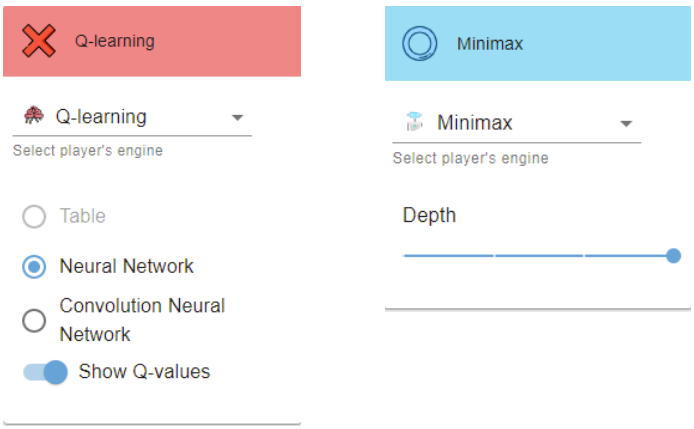
I když v poslední době roste popularita jazyků Python a Java, patří Javascript mezi nejčastěji používané programovací jazyky, kteří lidé využívají ve své práci [26]. Aby nebyla zapotřebí instalace dalšího softwaru a nastavení prostředí je hra Gomoku dostupná prostřednictvím webové aplikace.

## 4.3 Q-Gomoku

Za tímto účelem byla vytvořena komplexní webovou aplikace dostupná na http://qgomoku.rostiapp.cz/, ve které si uživatel může nejenom zahrát hru Gomoku na různých velikostech hrací desky a v různých módech, ale může také vytrénovat svého vlastního agenta v připravené tělocvičně a postavit ho následně proti jinému agentovi se kterým poměří svoje síly. Velkou výhodou je vícevláknové zpracování, kdy dlouhotrvající operace, jako například učení, se provádějí na jiném, než hlavním vlákně. Tím je používání aplikace pro uživatele mnohem přívětivější. Další předností aplikace je možnost sledovat průběh učení a do jisté míry jej ovlivňovat. Vše je doplněno přehlednými statistikami a grafy.

### 4.3.1 Hrací místnost

Hrací místnost je vizuálně rozdělena na tři části. Po stranách jsou formuláře pro výběr herních agentů (Obr. 17). Těch si uživatel může vybrat hned několik. Někteří agenti mají svá specifická nastavení, která si uživatel může měnit.




Obrázek 17. Formuláře pro výběr agentů

#### *4.3.1.1 Výběr agenta*

**Člověk** (human) je lidský agent, který reprezentuje skutečného lidského hráče u kterého lze měnit jeho jméno.

**Náhodný** (random) agent hraje pseudo-náhodné tahy.

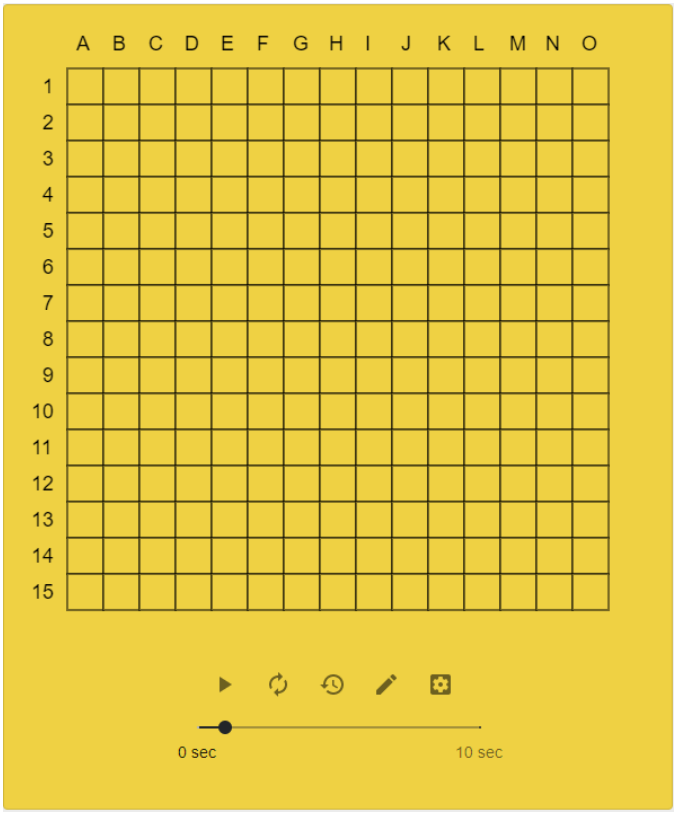
**Minimax** agent implementuje Minimax algoritmus s alfa-beta ořezáváním. Má nastavitelnou hloubku propočtu, která je závislá na velikosti herní desky.

**Q –** agent (Q-learning) obsahuje námi vytrénované agenty pomocí Q-algoritmu. Nejprve je tedy vždy potřeba agenta vytrénovat. Jednotlivé typy Q-agenta jsou odvislé z velikosti hrací desky a zvoleného herního módu. Lze také zapnout zobrazování Q-hodnot na hrací desce.

#### *4.3.1.2 Hrací deska*

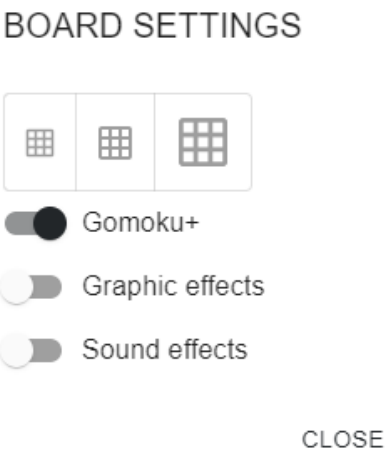
Herní deska (Obr. 18) se skládá ze dvou částí. A to z herní desky samotné, která je v horizontálním směru popsána písmeny anglické abecedy a ve vertikálním směru arabskými číslicemi a ovládacím panelem, který se nachází přimo pod ní. Pomocí ovládacího panelu můžeme:

- Hru pozastavovat a opětovně spouštět.
- Restartovat rozehrátou hru.
- Vracet tahy až do počáteční pozice.
- Editovat aktuální pozici, nebo si vytvářet svoje vlastní.
- Nastavovat časovou délku mezi jednotlivými tahy (to se hodí hlavně v případech, kdy chceme například nalyzovat průběh hry, nebo sledovat měnící se Q-hodnoty).
- Otevřít nastavení herní desky.




Obrázek 18. Hrací deska

V nastavení herní desky (Obr. 19) můžeme měnit její velikost, kdy máme na výběr tři různé velikosti (3x3, 9x9 a 15x15). Také tu lze vypnout, nebo zapnout grafické a zvukové efekty či ativovat Gomoku+ mód, který je dostupný jen pro větší herní desky (9x9 a 15x15).




Obrázek 19. Nastavení herní desky

### 4.3.2 Tělocvična

Účelem tělocvičny je kromě natrénování agentů, abychom si proti nim mohli potom zahrát, také vyzkoušet vliv jednotlivých nastavení na samotné učení. Protože trénování probíhá asynchronně, můžeme některá natavení měnit v průběhu trénování. Proces trénování spustíme tlačítkem Start, které se nachází v levém horním oknu obrazovky. Stejným tlačítkem, můžeme trénování předčasně ukončit. Tím ale dosavadní průběh učení ztratíme.

Obrazovka je rozdělena na tři části.

#### *4.3.2.1 Nastavení trénování*

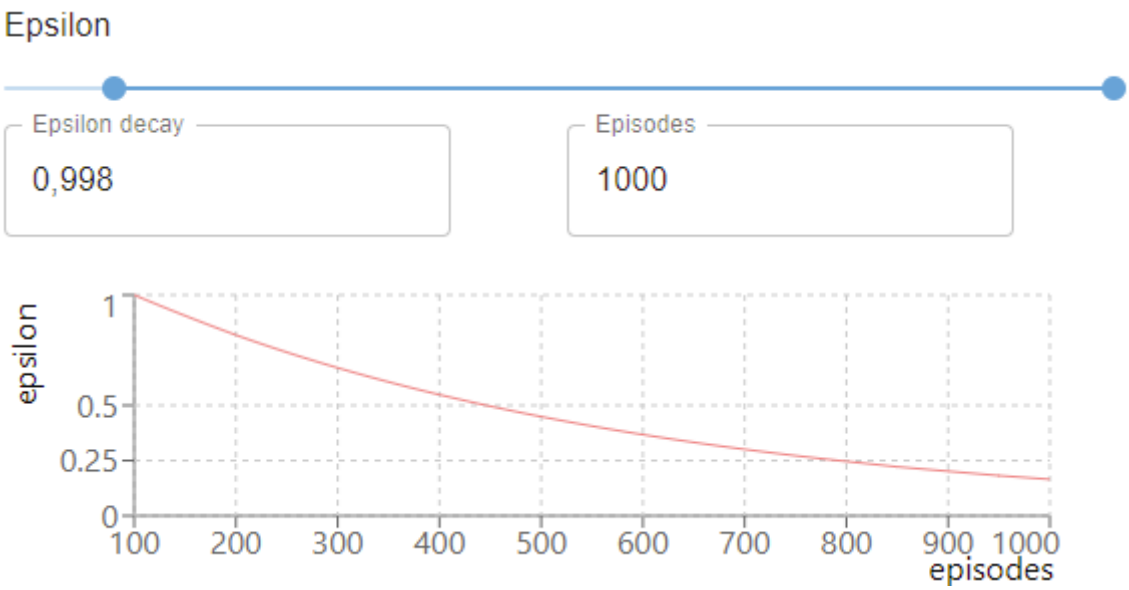
V poli **Trainer** lze vybrat trenéra (agenta), který bude hrát proti našemu agentovi. Máme na výběr tři možnosti.

- Náhodný (random) provádějící náhodné tahy.
- Minimax s nastavitelnou hloubkou propočtu.
- Sám sebe (self-play), kdy trénovaný agent je zároveň i protihráč.

Výběrem z comboboxu **Learn from** ovlivníme to, z jakých tahů se bude agent učit. Pokud zvolíme X, bude se učit jen z tahů provedených a stavů navštívených hráčem X. Analogicky je to stejně i s volbou O. Pokud zvolíme Both, bude se agent učit tahy od obou hráčů. Samozřejmě musíme počítat s tím, že učení bude o to pomalejší.

Zajímavou částí je modelování průběhu hodnoty epsilon (Obr. 20), která definuje chování agenta při výběru tahů tzv. ε-greedy startegii. Ta závisí na čtyřech faktorech.

- Počáteční hodnota epsilon, která se v průběhu času mění.
- Minimální hodnota epsilon, pod kterou už algoritmus nejde.
- Koeficient rozkladu (decay) epsilon, který definuje rychlost, jak se hodnota epsilon bude v průběhu trénování měnit.
- Počet epizod, které bude muset agent během trénovaní projít.




Obrázek 20. Modelování průběhu epsilon

Křivka má exponenciální průběh a určuje s pravděpodobností $\varepsilon$, že agent vybere náhodný tah a s pravděpodobností $1 - \varepsilon$, že bude následovat svou naučenou strategii.

Za zmínku stojí také výběr typu sítě, kde máme tři možnosti.

- Tabulka (Table)
- Neuronová síť (Neural Network)
- Konvoluční neuronová síť (Convolution Neural Network)

Tabulka není vlastně žádná síť, ale jen vyhledávací tabulka, kde řádky v tabulce reprezentují stavy a sloupce zase jednotlivé akce. Tato volba je dostupná jen pro hrací desku o velikosti 3x3, z důvodu množství všech možných kombinací.

Neuronové a konvoluční sítě obsahují další nastavení. Především počet a velikost skrytých vrstev (Hidden layers sizes), které definují kolik skrytých vrstev síť bude obsahovat a jaké budou velikosti. Například (128;64;32) bude reprezentovat tři skryté vstvy, kde první skrytá bude mít 128 neuronů, druhá skrytá 64 a poslední skrytá 32 neuronů. Všechny skryté vstvy používají aktivační funkci ReLU. Aktivační funkci lze měnit pouze na výstupní vrstvě.

Konvoluční síť má navíc možnost definovat počet konvolučních vrstev a počet filtrů. Analogicky je to stejné jako při volbě počtu a velikosti skrytých vrstev. Velikost konvolučního okna u hrací desky 3x3 je stejná a u větších desek je potom 5x5.

Konvoluční krok (stride) je nastaven na jedna. Vycpávka (padding) je nastavena na hodnotu ‘same’, aby výstup měl stejnou šířku a výšku jako vstup.

Parametrem Target network update rate ovlivňujeme to, jak často se mají přepisovat váhy cílové sítě. Například hodnota 100 znamená, že se bude síť kopírovat vždy po 100 epizodách.

Trénováním se nastavení těchto sítí uloží a pokud je chceme měnit, musíme nejdříve síť smazat a potom znovu vytvořit.

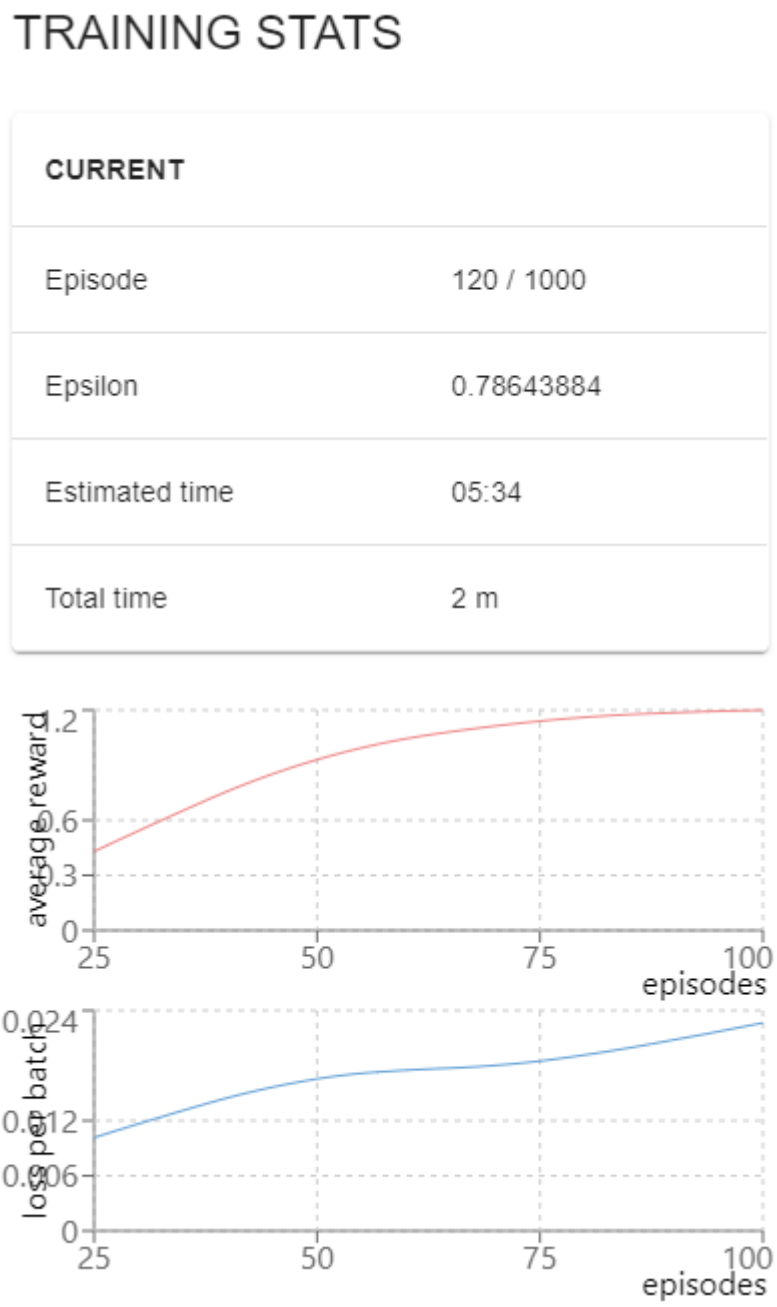
Zajímavou funkcí je možnost sledovat průběh trénování na herní desce, kdy můžeme sledovat jednotlivé tahy, nebo jen konečné pozice. Je také dostupné zobrazení Q-hodnot během procesu učení.

#### *4.3.2.2 Hrací deska*

Je popsána už v kapitole o hrací desce. Rozdíl je akorát v tom, že některé ovládací prvky zde nejsou dostupné, protože by byly zbytečné.

#### *4.3.2.3 Statistiky a grafy*

Průběh trénování můžeme sledovat pomocí detailních statistik, které se za běhu aktualizují.




Obrázek 21. Průběh trénování

Aktuální statistiky (current, Obr. 21) zobrazují aktuální informace o průběhu trénování, jako je aktuální epizoda, nebo hodnota epsilon. Zajímavou hodnotou je odhadovaný čas ukončení trénování (estimated time), která se aktualizuje průběžně na základě délky jedné epizody a počtu epizod zbývajícím do konce.

Graf „Average Reward“ (Obr. 21) znázorňuje průběh průběrné odměny během procesu trénování.

Graf „Loss per batch“ (Obr. 21) znázorňuje chybovost učení.

## 4.4 Propojení teorie a praxe

Na základě předchozích teoretických kapitol si můžeme propojit některé pojmy s konkrétními prvky, které se nachází ve hře.

**Agent** v naší hře je herní program, chovající se jako hráč, mezi nímž a herní deskou dochází k interakci.

**Prostředí**, ve kterém se agent pohybuje je:

- Diskrétní, protože máme konečnou množinu možných tahů.
- Stochastické, protože nevíme, co soupeř zahraje.
- Plně pozorovatelné, protože všechny informace o stavu herního prostředí jsou dostupné.
- Multiagentní kompetitivní, protože hrají dva agenti proti sobě.
- Sekvenční, protože se tahy navzájem ovlivňují.
- Statické, protože se hrací deska, během přemýšlení agenta nemění.

**Akce** jsou tahy, které provádí agent a které mění prostředí.

**Odměna**, získaná agentem, je předem dána a je za výhru +1, prohru -1 a za remízu je nulová.

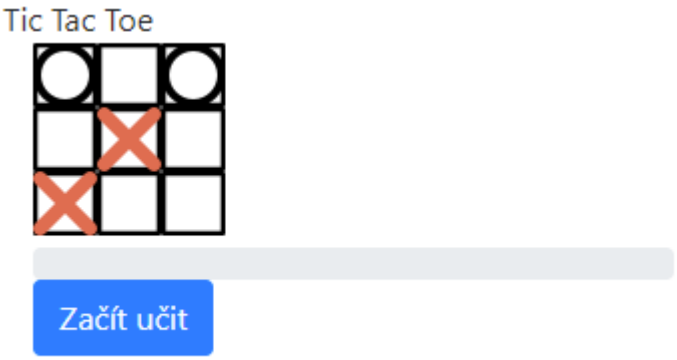
**Model** prostředí v této hře není znám.

## 4.5 Technická specifikace

### 4.5.1 Prototyp aplikace

K vývoji aplikace byl použit prototypový přístup, kdy byl nejprve vytvořen prototyp (Obr. 22) a to za účelem vyzkoušení různých knihoven, frameworků i samotného algoritmu učení. Pro práci s neuronovými sítěmi byla použita knihovna Synaptic. Ve finální verzi aplikace je použita knihovna Tensorflow.js.

.




Obrázek 22. Prototyp hry

### 4.5.2 Komponenty

Díky použité knihovně React, jsou veškeré části uživatelského rozhraní implementovány jako komponenty tzn. mají své vstupní parametry, kterými můžeme upravovat jejich chování, či vzhled a jsou znovupoužitelné ve zbytku aplikace.

Příklad komponenty reprezentující herní desku.

```
<BoardComponent color="#00000"
                background="#F3D250"
                showNotation={true}
                width={200}
                height={200}
                onClick={this.handleBoardClick.bind(this)}
                board={this.state.board} />
```

### 4.5.3 Asynchronní zpracování

Proces učení, nebo náročné propočty Minimax algoritmu běží asynchronně mimo hlavní vlákno. Je to díky konceptu pracovníků (Web Workers), kteří se spouští v novém vlákně a komunikují s hlavním vláknem pomocí zpráv (AJAX). Tím je zaručeno, že uživatelské rozhraní bude plynulé i během trénování agenta.

### 4.5.4 Rozhraní algoritmů

Všechny herní algoritmy ve hře dědí ze společné třídy *Engine*, která obsahuje základní metody pro hraní hry. Nejdůležitější metodou, kterou je potřeba implementovat, je *getBestMove(board)*, jenž má na vstupu herní desku a vrací tah.

Ve hře jsou dostupné tyto třídy:

- *HumanEngine* (zastupuje lidského hráče)
- *RandomEngine* (hraje náhodné tahy)
- *MinimaxEngine* (Minimax algoritmus s alfa-beta ořezáváním)
- *TabularQEngine* (tabulkový Q-learning)
- *DQNEngine* (Q-learning pomocí neuronových sítí)

Díky společnému rozhraní, není problém do hry přidat vlastní herní algoritmus.

*TabularQEngine* a *DQNEngine* dědí ze třídy *ReinforcementEngine*, která obsahuje navíc metodu pro trénování *train(replay, mark)*. Tato metoda má na vstupu záznam jedné hry a značku hráče (X, nebo O), která určuje ze kterých tahů se agent bude učit.

# ZÁVĚR

Cílem práce bylo povést rešerši v oblasti metod umělé inteligence v počítačových hrách a seznámit se s posilovaným učením v neuronových sítích. Práce je zaměřena především na algoritmus Q-learning a jeho implementaci s různými vylepšeními do webové aplikace hrající hru Gomoku.

Byla implementována základní tabulková metoda tohoto algoritmu, jež byla dále vylepšována pomocí hlubokých neuronových sítí a technik jako je například vzpomínková paměť, cílová síť nebo dvojité hluboké Q-učení.

Na hrací desce o velikosti 3x3 se podařilo vytrénovat všechny agenty do takové míry, že hráli obstojně proti algoritmu Minimax.

Na větších hracích deskách bylo zpozorováno, že agent je schopný se učit určité vzory, avšak k tomu, aby dosáhl lepší úrovně je zapotřebí delšího času potřebného pro trénování a také správné nastavení parametrů učení.

Vytvořená aplikace má vzdělávací charakter a jejím prostřednictvím si každý může vyzkoušet trénování agenta a seznámit se zpětnovazebním učením.

# SEZNAM POUŽITÉ LITERATURY

[1] ANDREJ KARPATHY. ConvNetJS Deep Q Learning Demo [online]. [cit. 17.5.2021]. Dostupný na WWW: https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

[2] GEORGIOS N. YANNAKAKIS AND JULIAN TOGELIUS. Artificial Intelligence and Games. neuvedeno: Springer, 2018, ISBN 978-3-319-63519-4.

[3] Historie gomoku a renju | Česká federace piškvorek a renju. Česká federace piškvorek a renju [online]. Copyright © 2021 [cit. 07.05.2021]. Dostupné z: http://www.piskvorky.cz/clanky/zajimavosti-ze-sveta-piskvorek-a-renju/historie-gomoku-a-renju/

[4] WIKIPEDIA CONTRIBUTORS. Gomoku [online]. [cit. 17.5.2021]. Dostupný na WWW: https://en.wikipedia.org/wiki/Gomoku

[5] LOUIS VICTOR ALLIS. Searching for Solutions in Games and Artificial Intelligence. neuvedeno: Ponsen & Looijen, 1994, ISBN 9090074880.

[6] Caro (aka Gomoku) - LearnPlayWin. LearnPlayWin.net - Skill Games Rules and Strategy [online]. Dostupné z: https://learnplaywin.net/caro/

[7] ARTHUR ARNX. First neural network for beginners explained (with code) [online]. [cit. 17.5.2021]. Dostupný na WWW: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

[8] AVINASH SHARMA V. Understanding Activation Functions in Neural Networks [online]. [cit. 17.5.2021]. Dostupný na WWW: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[9] NICOLA MANZINI. Single hidden layer neural network [online]. [cit. 17.5.2021]. Dostupný na WWW: https://www.nicolamanzini.com/single-hidden-layer-neural-network/

[10] JASON BROWNLEE. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning [online]. [cit. 17.5.2021]. Dostupný na WWW: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[11] CHOLLET, François. Deep learning v jazyku Python: knihovny Keras, Tensorflow. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

[12] SUMIT SAHA. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [online]. [cit. 17.5.2021]. Dostupný na WWW: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[13] VINICIUS C. COSTA. Understanding the Structure of a CNN [online]. [cit. 17.5.2021]. Dostupný na WWW: https://medium.com/@viniciuscantocosta/understanding-the-structure-of-a-cnn-b220148e2ac4

[14] SUTTON, Richard S. a Andrew G. BARTO. Reinforcement learning: an introduction. Second edition. Cambridge, Massachusetts: The MIT Press, [2018]. ISBN 978-0262039246

[15] NEUVEDEN. Mountain–Car Task [online]. [cit. 17.5.2021]. Dostupný na WWW: https://www.programmersought.com/article/83806541470/

[16] MOHAMMAD ASHRAF. Reinforcement Learning Demystified: A Gentle Introduction [online]. [cit. 17.5.2021]. Dostupný na WWW: https://m-elsersy96.medium.com/reinforcement-learning-demystified-36c39c11ec14

[17] Markov decision process/cs - Simulace.info. [online]. Dostupné z: http://www.simulace.info/index.php/Markov_decision_process/cs

[18] POLANSKÝ, Dušan. Bellmanův princip optimality [online]. [cit. 17.5.2021]. Dostupný na WWW: http://www.dusanpolansky.cz/clanky/bellman.html

[19] WATKINS, Chris a kol. Q Learning: Technical Note. Boston: Kluwer Academic Publishers, 1992, MACHINE LEARNING, 8, 279-292.

[20] MNIH, Volodymyr; KAVUKCUOGLU, Koray; SILVER, David; GRAVES, Alex; ANTONOGLOU, Loannis; WIERSTRA, Daan; RIEDMILLER, Martin. Playing Atari with Deep Reinforcement Learning [online]. [cit. 17.5.2021]. Dostupný na WWW: https://arxiv.org/abs/1312.5602

[21] HUI, Jonathan. RL — DQN Deep Q-network [online]. [cit. 17.5.2021]. Dostupný na WWW: https://jonathan-hui.medium.com/rl-dqn-deep-q-network-e207751f7ae4

[22] DOSHI, Ketan. Reinforcement Learning Explained Visually (Part 5): Deep Q Networks, step-by-step [online]. [cit. 17.5.2021]. Dostupný na WWW: https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

[23] NEUVEDEN. Learn Reinforcement Learning (3) - DQN improvement and Deep SARSA [online]. [cit. 17.5.2021]. Dostupný na WWW: https://greentec.github.io/reinforcement-learning-third-en/

[24] EPPES, Marissa. Game Theory — The Minimax Algorithm Explained [online]. [cit. 17.5.2021]. Dostupný na WWW: https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1

[25] SONI, Devin. Introduction to Evolutionary Algorithms [online]. [cit. 17.5.2021]. Dostupný na WWW: https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac

[26] STACK OVERFLOW. Developer Survey Results 2019 [online]. [cit. 17.5.2021]. Dostupný na WWW: https://insights.stackoverflow.com/survey/2019#technology-_-programming-scripting-and-markup-languages

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CNN Convolution Neural Network.

RNN Recurrent Neural Network.

MDP Markovův rozhodovací proces.

DQN Hluboké Q učení.

DDQN Dvojité hluboké Q učení.

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: NÁZEV PŘÍLOHY