|  |  |  |
| --- | --- | --- |
|  | | |
| **Závěrečná studijní práce**  **dokumentace** | | |
| **Deep learning FlappyBird** | | |
| Petr Krautwurst | | |
|  | | |
|  | |  |
| **Obor:** | 18-20-M/01 INFORMAČNÍ TECHNOLOGIE  se zaměřením na počítačové sítě a programování | |
| **Třída:**  **Školní rok:** | IT4  2020/2021 | |

#### Poděkování

*Rád bych poděkoval doc. Ing. Petru Čermákovi, Ph.D. za objasnění nejasností a složité problematiky*

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité   
informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové   
a umělecké škole v Opavě, Praskova 399/8.

V Opavě 31. 12. 2020

*podpis autora práce*

**ANOTACE**

Práce popisuje umělé neuronové sítě od perceptronu až po vícevrstvé modely. Dále jsou popsány důležité aktivační funkce, metody učení – učení s učitelem, učení bez učitele, posilované učení a hluboké učení. Výsledkem práce je prostředí vytvořené v Kivy na základě hry FlappyBird, ve kterém je pomocí knihovny Keras (TensorFlow API) aplikovaná neuronová síť a učení s učitelem. Uživatel může nastavovat základní parametry učení.

**KLÍČOVÁ SLOVA**

Umělá inteligence, umělá neuronová síť, hluboké učení, programování

OBSAH

[Úvod 5](#_Toc60740676)

[Umělé neuronové sítě 6](#_Toc60740677)

[1.1 Perceptron 6](#_Toc60740678)

[1.2 Přenosová (aktivační) funkce 7](#_Toc60740679)

[1.2.1 Skoková funkce 7](#_Toc60740680)

[1.2.1 ReLU 7](#_Toc60740681)

[1.2.2 Sigmoidální funkce 7](#_Toc60740682)

[1.2.1 Hyperbolický tangens (tanh) 8](#_Toc60740683)

[1.3 Multi-Layer perceptron (MLP) 8](#_Toc60740684)

[1.4 Trénování neuronové sítě 9](#_Toc60740685)

[1.4.1 Učení s učitelem 9](#_Toc60740686)

[1.4.2 Učení bez učitele 11](#_Toc60740687)

[1.4.3 Posilované učení 11](#_Toc60740688)

[1.5 Hluboké učení 12](#_Toc60740689)

[2 Využité technologie 13](#_Toc60740690)

[2.1 Kivy 13](#_Toc60740691)

[2.2 TensorFlow, Keras 13](#_Toc60740692)

[3 Způsoby řešení a použité postupy 14](#_Toc60740693)

[3.1 Prostředí 14](#_Toc60740694)

[3.2 Umělá inteligence 14](#_Toc60740695)

[3.2.1 Struktura modelu 15](#_Toc60740696)

[4 Výsledky řešení 16](#_Toc60740697)

[4.1 Základní funkce 16](#_Toc60740698)

[4.2 Splněné a nesplněné cíle 16](#_Toc60740699)

[Závěr 18](#_Toc60740700)

[Seznam použitýCH INFORMAČNÍCH ZDROJů 19](#_Toc60740701)

Úvod

Cílem práce bylo vytvořit aplikaci, ve které bych využil umělou inteligenci. Původně jsem chtěl využít DQN, ale později jsem tento cíl změnil za hluboké učení s učitelem, protože DQN bylo nad mé síly.

Hlavní motivací bylo pochopit základy neuronových sítí a posilovaného učení. Je to velmi zajímavé, ale komplikované téma, které je čím dál více populární. Rozhodl jsem se proto pro studium základů umělé inteligence.

Aplikace je založená na konceptu hry FlappyBird, kde se objekt snaží proskakovat mezi překážkami.

Tato práce nejprve popisuje problematiku umělé inteligence – základní podobu umělého neuronu, jeho parametry, využití, výhody a nevýhody. Dále popisuje jeho řetězení do vrstev a vytváření umělých neuronových sítí. Rovněž se práce zmiňuje o základních aktivačních funkcích, metodách učení a o hlubokém učení.

Po teoretické části práce seznamuje čtenáře s využitými technologiemi a postupy. Následuje vysvětlení funkčnosti, způsoby řešení, některé komplikace při tvorbě aplikace a přehled splněných a nesplněných cílů.

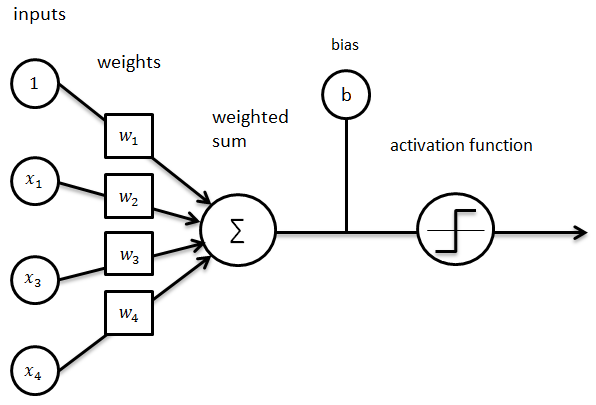
# Umělé neuronové sítě

Umělá neuronová síť (*artifical neural network*) je matematický model používaný v umělé inteligenci. Jejím vzorem je chování odpovídajících biologických struktur. Umělá neuronová síť je struktura určená pro distribuované paralelní zpracování dat.

Skládá se z umělých neuronů, které jsou inspirovány biologickými neurony. Neurony jsou navzájem propojeny a předávají si signály a transformují je pomocí přenosových funkcí.

## Perceptron

Perceptron je základ neuronových sítí. Jedná se o nejjednodušší model dopředné neuronové sítě. Sestává pouze z jednoho neuronu, který je možno učit změnami jeho parametrů.

Perceptron se skládá ze vstupů, které jsou násobeny váhami (*weights*). Tyto váhy určují důležitost jednotlivých vstupů a jejich podíl na ovlivnění aktivační hodnoty . Následuje sečtení (vážený součet), přičtení prahu (*bias*) a vyhodnocení pomocí aktivační (přenosové) funkce .

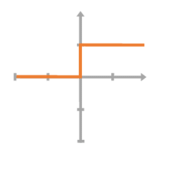
Obrázek 1.1: Schéma perceptronu

Při vstupech , kdy počet vstupů označíme jako n, můžeme vyjádřit výstup *y* jako:

## Přenosová (aktivační) funkce

Přenosová (aktivační) funkce určuje aktivaci neuronu. Zastupuje velmi důležitou roli a její výběr výrazně ovlivní chování celého modelu. Nedoporučuje se vybírat lineární funkci, ale raději některou z nelineárních – např. skokovou, ReLU nebo sigmoidální.

### Skoková funkce

Skoková funkce je nejjednodušší aktivační funkce. Její výstup se rovná nule, dokud vstup nepřekročí určitou hranici (např. nula); po překročení hranice se výstup rovná jedné. Této funkci se také přezdívá binární skok (*binary step*).

Obrázek 1.2: Skoková funkce

### ReLU

ReLU funkce má více různých podob. Základní podoba má výstup rovný nule, pokud je vstup menší než nula. Lze definovat jako:

Obrázek 1.3: ReLU

### Sigmoidální funkce

Sigmoidální funkce má podobný tvar jako písmeno „S“. Jedna z nich je například logistická funkce, kterou můžeme definovat jako:

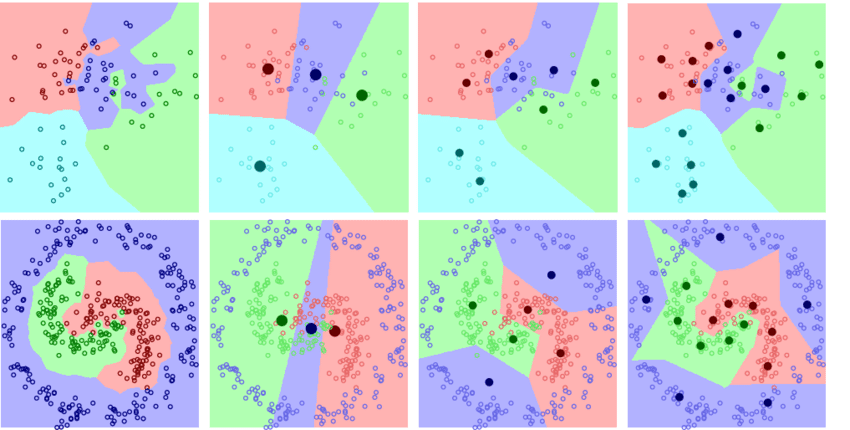
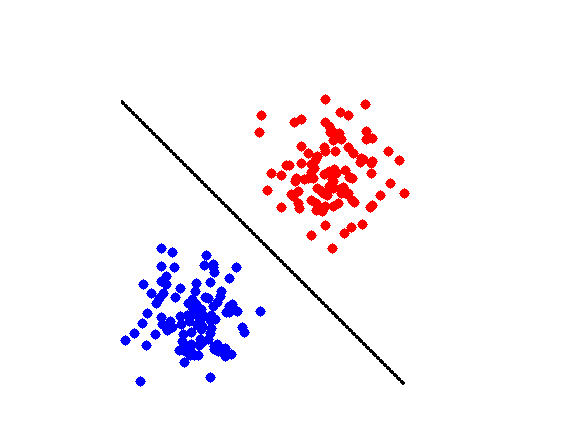
Obrázek 1.4: Sigmoidální funkce

### Hyperbolický tangens (tanh)

Tanh je funkce, která transformuje reálné číslo do intervalu (-1, 1). Má všechny dobré vlastnosti sigmoidy a přidává k tomu skutečnost, že průměrný výstup má hodnotu 0, což je ještě výhodnější z hlediska normalizace.

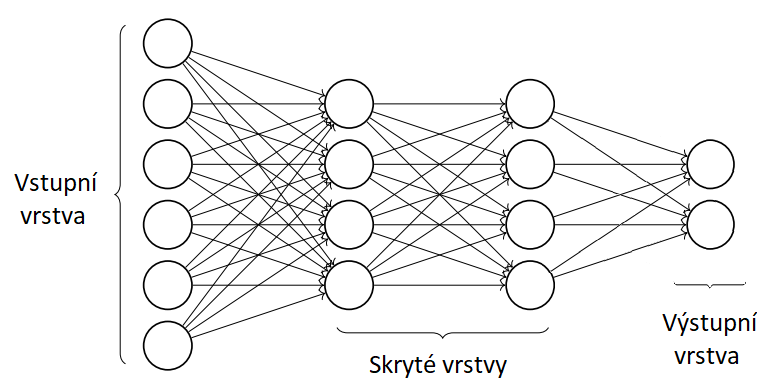
Obrázek 1.5: Hyperbolický tangens

## Multi-Layer perceptron (MLP)

Samotný perceptron umí kategorizovat pouze lineárně rozdělená data a není schopen řešit komplexnější úlohy. Je však možné spojit více perceptronů do více vrstev a vytvořit tím výpočetně výkonnější model.

Obrázek 1.6: Rozdělení bodů podle perceptronu

Obrázek 1.7: Příklady rozdělení bodů podle vícevrstvé sítě

Neuronové sítě se skládají z vrstev, které můžeme rozdělit na tři základní skupiny:

Obrázek 1.8: Schéma vícevrstvé perceptronové sítě

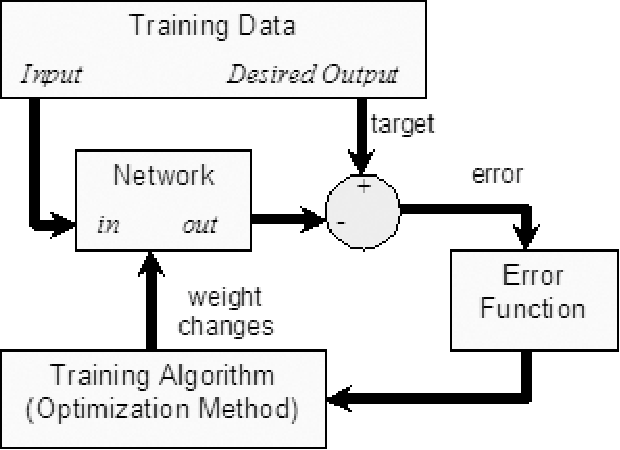
1. Vstupní vrstva – nevypočítává aktivaci, pouze dál posílá vstup,
2. Skrytá vrstva – nelze k ní přistupovat přímo, uplatňují se zde principy jako u jednoho perceptronu, neurony posílají své aktivační hodnoty do další vrstvy,
3. Výstupní vrstva – její výstup je výstupem celé sítě.

## Trénování neuronové sítě

Abychom dostali požadované výsledky, musíme nejprve síť učit. To znamená měnit parametry (váhy, prahy) sítě tak, aby se výsledek co nejvíce blížil požadovanému výsledku. Způsoby trénování se dají rozdělit do tří skupin. Učení s učitelem, bez učitele a posilované učení.

### Učení s učitelem

Učení s učitelem předpokládá, že máme pro dané vstupy i očekávané výstupy. Jedním z nejznámějších způsobů učení s učitelem pro učení MLP sítě je algoritmus zpětného šíření chyby – *backpropagation*.

Potřebujeme si připravit trénovací data (vstupy a očekávané výstupy). Nejdříve se nastaví parametry sítě náhodně. Neuronové síti jsou předloženy vstupy z trénovací množiny, dostaneme výstup, ten porovnáme s požadovaným výstupem. Následně je nutné určit chybu a upravit nastavení parametrů sítě tak, aby se co nejvíce snížila hodnota chyby.

Obrázek 1.9: Učení s učitelem

Tato chyba může mít různé definice, například průměrná kvadratická chyba (*mean squared error).* Pro počet trénovacích párů , dimenzí výstupu sítě , výstup sítě a očekávaného výstupu , lze chybu vyjádřit jako:

Tuto chybu chceme minimalizovat a najít její minimum – ideálně globální. Abychom tuto funkci mohli minimalizovat, musíme znát, do jaké míry jednotlivé parametry ovlivňují tuto funkci. Od každého parametru sítě musíme odečíst jeho vliv na chybovou funkci. Vliv nám říká, jakým směrem se bude funkce měnit, proto musíme vypočítat gradient (pomocí derivace funkce vzhledem k vahám). Nechť α je míra učení – obecně pak pro váhy v čase *t* platí:

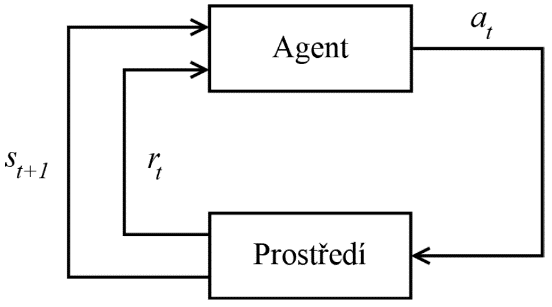
Odečítáme tedy gradient od současných parametrů. Předpokládá se, že aktivační funkce bude spojitá a bude mít spojitou derivaci.

Základní verze počítá gradient každou iteraci přes celou trénovací množinu. Kvůli náročnosti na výpočetní výkon a rychlost se využívá aproximace gradientu na několika vzorech, které tvoří dávku (*batch*). Dávka se vybírá náhodně, proto se tomu říká stochastická gradientní metoda – SGD (*stochastic gradient descent*). Jedna z nejnovějších metod je algoritmus Adam.

### Učení bez učitele

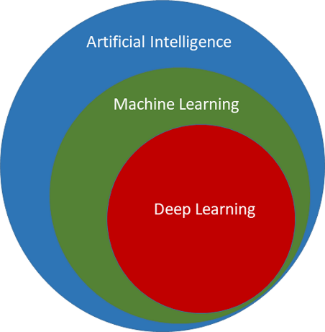
Učení bez učitele (*unsupervized learning*) nepotřebuje znát správné výstupy. Je to způsob hledání předem neznámých vzorů v daných datech, hledají se v nich podobnosti či rozdíly. Tomuto přístupu se také říká samoorganizující, jelikož učící techniky nemají učitele a musí si samy organizovat učící proces. Jednou z možností, jak tato data třídit, je shlukování (*clustering*).

### Posilované učení

Posilované učení (*reinforcement learning*) také nezná správné výstupy, ale na rozdíl od učení bez učitele dostává zpětnou vazbu od prostředí. Jedná se například o odměnu (*reward*), která může být kladná či záporná (chtěná akce – kladná odměna, nechtěná akce – záporná odměna). Učený model se označuje jako agent. Tento agent má určitý stav, na který reaguje akcí. Akce poté ovlivní prostředí, které vrátí agentovi nový stav a odměnu za předchozí akci. Takto se proces opakuje, dokud agent nedosáhne cíle.

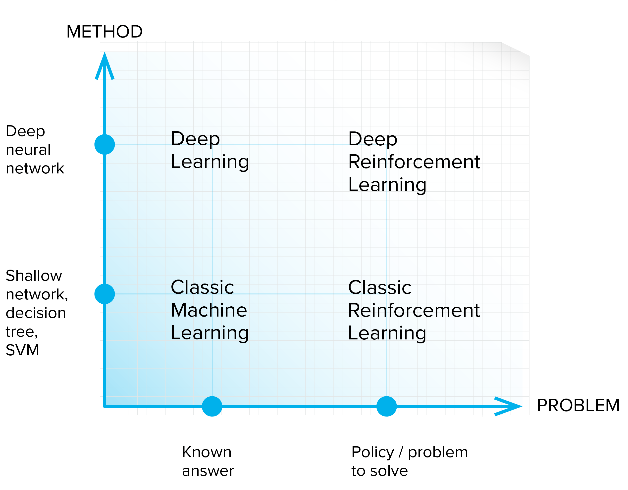
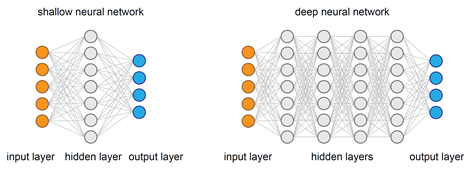
Obrázek 1.10: Posilované učení

## Hluboké učení

Hluboké učení (*deep learning*) je disciplína strojového učení, která se zabývá využitím algoritmů s větším počtem vrstev.

Obrázek 1.11: Hluboké učení jako podobor strojového učení

Přídavné jméno „hluboké“ v hlubokém učení označuje počet vrstev, kterými se data transformují. Systémy hlubokého učení mají podstatnou hloubku cesty přiřazení kreditů (*credit assignment path* – CAP). CAP označuje řetězec transformací od vstupu k výstupu. CAP popisují potenciálně kauzální spojení mezi vstupem a výstupem. Prakticky to znamená počet neuronů, kterými signál projde, než dojde k výstupu. U dopředných neuronových sítí je CAP roven počtu skrytých vrstev v síti plus jedna (neurony v poslední vrstvě mají také vliv na výstup). U rekurentních neuronových sítí je hloubka CAP potenciálně neomezená (cykly).

Není žádný oficiální práh hloubky, který by rozděloval hluboké učení a obyčejné „mělké“ učení, nicméně většina vědců se shoduje na tom, že deep learning zahrnuje hloubku CAP větší než dva. CAP o hloubce dva se ukázal jako univerzální aproximátor ve smyslu, že umí napodobovat jakoukoli funkci.

Obrázek 1.13: MLP vs deep MLP

Obrázek 1.12: Hluboké vs posilované učení

# Využité technologie

Celá aplikace je napsána v programovacím jazyce Python. Python je přehledný, jednoduchý na čtení, rychle a jednoduše se v něm realizují složité věci, bývá často používán ve světě umělých inteligencí. Konkrétně je využit Python 3.7.9, jelikož TensorFlow 2.1.0 nepodporuje novější verzi jazyka Python.

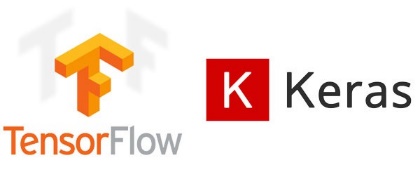
## Kivy

Grafická část aplikace je realizována pomocí knihovny Kivy. Kivy je multiplatformní grafická knihovna, akcelerovaná pomocí grafického rozhraní OpenGL. Další výhodou knihovny Kivy je její přístup. Na rozdíl od ostatních knihoven, kde základem je opakující se vykreslovací funkce, Kivy klade spíše důraz na změny. Díky tomu je celý kód přehlednější, jednodušší a rychlejší.

Obrázek 2.1: Kivy

## TensorFlow, Keras

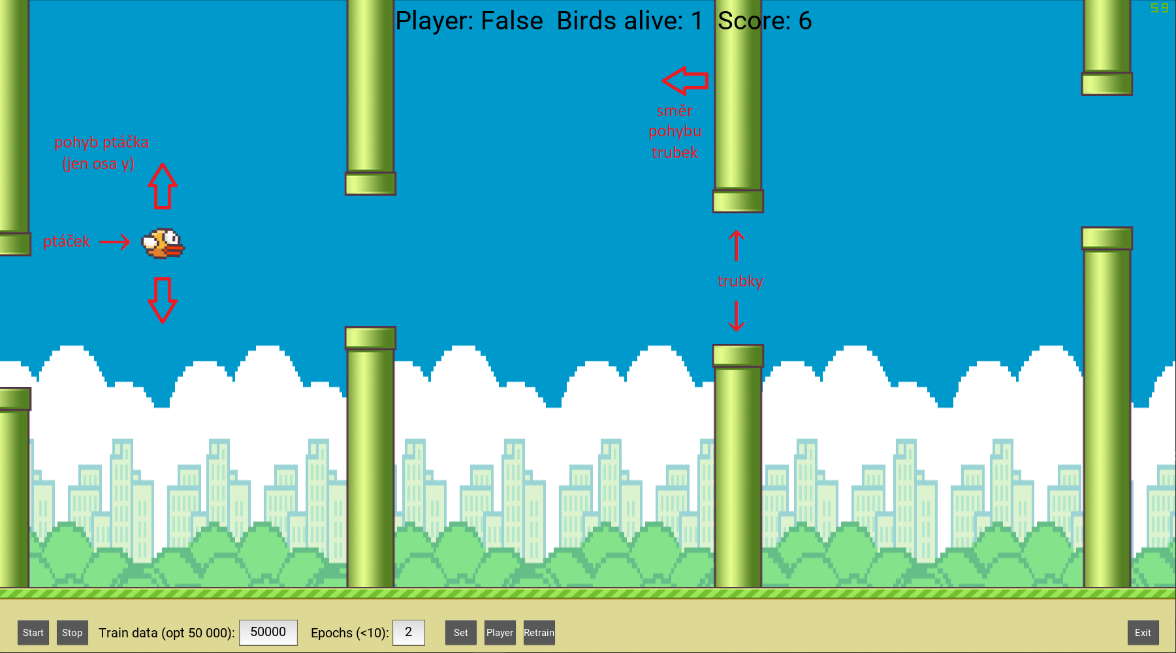
TensorFlow je bezplatná a otevřená softwarová knihovna pro strojové učení. Může být použit v celé řadě úkolů, ale má zvláštní zaměření na trénink a odvozování hlubokých neuronových sítí. TensorFlow je symbolická matematická knihovna založená na toku dat a diferencovatelném programování. Konkrétně jsem využil TensorFlow 2.1.0.

Pro samotné učení a predikci je použita knihovna Keras, což je nadstavba knihovny TensorFlow specializovaná na umělé neuronové sítě a umělou inteligenci. Použití této knihovny značně zrychluje i zjednodušuje celý proces.

Obrázek 2.2: TensorFlow a Keras

# Způsoby řešení a použité postupy

## Prostředí

Prostředí je 2D rovina, která je založena na hře FlappyBird. Jedná se o side-scrolling hru, ve které hráč ovládá ptáčka a má za úkol prolétávat mezerami mezi trubkami. Ptáček se pohybuje pouze po ose *y*, při stisknutí klávesy ptáček skočí a následně padá (podle kvadratické funkce). Trubkám se náhodně vygeneruje *y* pozice mezery, poté se jen posouvají doleva. Když ptáček proskočí trubkou, přičte se jeden bod. Pokud dojde ke kolizi ptáčka s trubkou, ptáček je označen jako mrtvý – přestane se hýbat, zčervená a je odsunut za okraj spolu s trubkou.

Obrázek 3.1: Prostředí

## Umělá inteligence

Umělá inteligence je realizována pomocí vícevrstvého perceptronu a deep learningu – metoda učení s učitelem. Model získává informace o pozici ptáčka na ose *y* a o pozici mezery nejbližší trubky na ose y.

Model se při vytváření rovnou naučí. Nejdříve se vygenerují náhodná data situací, které mohou ve hře nastat. K těmto datům se vygenerují očekávané výstupy modelu – pokud je pták níž, než je spodní trubka, musí skočit, jindy zůstává v klidu. Poté se tyto čísla převedou mezi 0 a 1 a model se na nich učí.

Při každém kroku model dostává data, opět je převede do intervalu 0 až 1 a vrátí svůj výstup. Výstup modelu určuje, zda ptáček skočí, či nikoli.

Ukázka kódu – zadávání dat modelu a vyhodnocení jeho výstupu:

data = [bird.center\_y, *# y ptáčka*

*# y horní trubky* self.current\_pipe.pipe\_center - self.current\_pipe.GAP\_SIZE / 2]

*# Pokud model vrátí hodnotu větší než 0.5, ptáček skočí***if** self.bird\_ai.predict(data) > 0.5:bird.jump()

### Struktura modelu

Vstupní vrstva obsahuje dva neurony. Dále jsou v modelu dvě skryté vrstvy o šestnácti neuronech, aktivační funkce skrytých vrstev je ReLU. Výstupní vrstva sestává z jednoho neuronu, jehož aktivační funkce je hyperbolický tangens – tanh.

Model využívá optimizer Adam a chybovou funkci MSE.

Ukázka kódu – metoda vytváření modelu:

**def** create\_model(structure=**None**):  
 **if** structure **is None**:  
 structure = [[**"dense"**, {**"units"**: 16, **"activation"**: **"relu"**}],  
 [**"dense"**, {**"units"**: 16, **"activation"**: **"relu"**}]]  
  
 model = tf.keras.models.Sequential()  
 tf.keras.backend.set\_floatx(**'float64'**)  
 **for** i, layer **in** enumerate(structure):  
 **if** i == 0: *# Vstupní vrstva* model.add(tf.keras.layers.Dense(  
 layer[1][**"units"**], activation=layer[1][**"activation"**],

input\_dim=2))  
 **else**: *# Skryté vrstvy* model.add(tf.keras.layers.Dense(  
 layer[1][**"units"**], activation=layer[1][**"activation"**]))  
 model.add(tf.keras.layers.Dense(1, activation=**"tanh"**)) *# Výstupní vrstva* model.compile(loss=**'mean\_squared\_error'**, optimizer=**'adam'**,

metrics=[**"accuracy"**])  
  
 **return** model

# Výsledky řešení

## Funkce aplikace

Dole se nachází hlavní menu, skrze které je ovládána celá aplikace.

Tlačítko „Start“ slouží ke spuštění hry. Pokud ještě není vytvořená umělá inteligence, vytvoří se a rovnou se spustí proces učení podle nastavených hodnot. Po dokončení učícího procesu se spustí samotná hra. Ptáček se snaží prolétat trubkami. Povede-li se mu proletět, dostává bod. Když se mu to nepovede, hra se restartuje a zkouší to znovu, dokud ho uživatel nezastaví tlačítkem „Stop“. Hra se průběhem času zrychluje, čímž se průlet mezi trubkami stává složitější. Aplikaci lze vypnout pomocí tlačítka „Exit“.

Uživatel si může aplikaci nastavit. Má na výběr ze dvou základních módů. Buď bude hrát pouze umělá inteligence a uživatel bude jen přihlížet, nebo může zkusit změřit síly s umělou inteligencí. Pokud zmáčkne tlačítko „Player“, při příštím startu do hry přibyde další ptáček, který je ovládaný člověkem a skočí při stisku klávesy w, mezerník nebo šipka nahoru. Ptáček ovládaný umělou inteligencí je v tomto případě kvůli přehlednosti částečně průhledný.

Dále lze nastavit učení modelu pomocí vstupních polí „Train data“, určující počet trénovacích dat, a „Epochs“, které nastavuje počet epoch trénování – kolikrát při trénování dostane stejná data. Zde může uživatel experimentovat a zjišťovat, při kolika datech a epochách je model schopen se spolehlivě naučit požadovaným výsledkům. Tlačítkem „Retrain“ tyto změny aplikuje a znovu vygeneruje model s aktuálním nastavením.

Obrázek 4.1: Nastavení aplikace

## Splněné a nesplněné cíle

Hlavní cílem bylo pochopit základní principy neuronových sítí a částečně s posilovaným učením, konkrétně s DQN (deep Q-Learning).

Tento cíl se mi z větší části podařilo splnit – pochopil jsem základy neuronových sítí, hlubokého učení i DQN.

Dalším cílem bylo následně s těmito poznatky vytvořit grafickou aplikaci s jednoduchým prostředím. Původně jsem měl plánu využít DQN i genetický algoritmus.

Aplikaci se mi sice povedlo vytvořit, zdárně jsem implementoval i učení s učitelem, avšak při pokusu o DQN nastaly vážnější komplikace – nedokázal jsem agenta naučit správnému chování ani po značném počtu různých změn. Po opakovaných nezdarech a s blížícím se termínem odevzdání jsem se rozhodl, že se raději vrátím k předchozí verzi a o DQN se pokusím později.

Seznam cílů:

* Pochopení základů umělé inteligence
* Pochopení základů DQN (jen částečně)
* Funkční aplikace
* Hluboké učení, učení s učitelem
* Funkční implementace DQN

# Závěr

Cílem projektu bylo pochopit a uplatnit teorii ohledně hlubokého učení neuronových sítí, ideálně s využitím DQN a vytvořit prostředí, kde uživatel může ovlivnit proces učení.

Aplikace umožňuje uživateli upravovat parametry učení a pozorovat jejich vliv na chování modelu při změnách počtu trénovacích dat či trénovacích epoch. Cíl práce byl z velké části naplněn.

Jsou zde vylepšení, které bych rád v budoucnu přidal. Mezi ně patří hlavně implementace DQN, ale také například větší moc uživatele nad učením modelu a nad modelem samotným a grafické informace o procesu učení.

Praktické řešení: <https://github.com/it1712/Zaverecny_Projekt>

Seznam použitýCH INFORMAČNÍCH ZDROJů

[1] 3Blue1Brown. *Neural networks* [online], YouTube. Aug 1, 2018 [cit. 2020-12-28]. Dostupné z: <https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi>

[2] BROWNLEE, Jason. *Gentle Introduction to the Adam Optimatization Algorithm for Deep Learning* [online]. Aug 20, 2020 [cit. 2020-12-28]. Dostupné z: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

[3] JULIANI, Arthur. *Simple Reinforcement Learning with Tensorflow* [online]. Aug 25, 2016 [cit. 2020-12-28]. Dostupné z: <https://medium.com/emergent-future/sim-ple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0>

[4] KANTOR, Jan. *Učení bez učitele* [online]. Brno, 2008 [cit. 2020-12-28]. Dostupné z: <https://core.ac.uk/download/pdf/30296714.pdf>. Diplomová Práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. Vedoucí práce Ing. PETR HONZÍK, Ph.D.

[5] KLŮJ, Jan. Obecná umělá inteligence pro hraní her [online]. Praha, 2017 [cit. 2020-12-28]. Dostupné z: <https://dspace.cuni.cz/handle/20.500.11956/90569>. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta.

[6] MAINI, Vishal. Machine Learning for Humans [online]. Aug 19, 2017 [cit. 2020-12-28]. Dostupné z: <https://medium.com/machine-learning-for-humans/unsupervi-sed-learning-f45587588294>

[7] Přispěvatelé Wikipedie, *Deep learning* [online], Wikipedia, The Free Encyclopedia. Dec 27, 2020 [cit. 2020-12-28]. Dostupné z: <https://en.wikipedia.org/wiki/Deep_learning>

[8] The Kivy Authors. *Welcome to Kivy – Kivy 2.0.0 documentation* [online]. [cit. 2020-11-28]. Dostupné z: <https://kivy.org/doc/stable/>