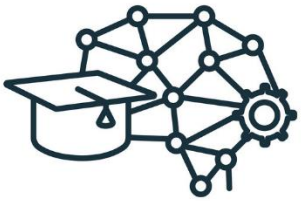
SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

Jednoduchá aplikace v Pythonu s použitím deep learningu

Lukáš Korzonek



DEEP LEARNING

Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

Třída: IT4

Školní rok: 2020/2021

Poděkování

Rád bych poděkoval panu doc. Ing. Petru Čermákovi, Ph.D. za rady a konzultace, které mi pomohly v řešení této 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

Tato práce popisuje umělé neurální sítě od základního perceptronu až po složitější vícevrstvé modely. Popsány jsou také důležité aktivační funkce, učící metody, přesněji učení s učitelem a bez něj a také deep learning, který využívám ve své práci. Vypsány jsou taktéž využité technologie, postup a způsob řešení vyskytlých problémů. Výsledkem projektu je prostředí, ve kterém je aplikována funkční neuronová síť a ve kterém může uživatel jednoduše měnit základní parametry.

KLÍČOVÁ SLOVA

Umělá inteligence; neuronová síť; deep learning; Python; programování

OBSAH

ÚVOD

Hlavním cílem bylo vytvořit aplikaci, která je ovládána umělou neuronovou sítí, v ideálním případě s použitím DQN. Jeden z mých prvních nápadů takové aplikace byla Rubikova kostka. Později sem však usoudil, že by byl tento projekt nad mé síly. Jako další mě napadla desková hra, dáma. Od tohoto nápadu jsem později také upustil a rozhodl jsem se pro něco jednoduššího. Projekt může připomínat PathFinder, kde se určený objekt má dostat z bodu A do bodu B. V cestě mu však stojí řada překážek. Výhodou by byla dostupná možnost nastavení parametrů uživatelem.

První část práce je zaměřena na problematiku AI. Popisuje základní podobu umělého neuronu (perceptron) a vyspělejší neuronové sítě (vícevrstvý perceptron). Práce rovněž zmiňuje základní přenosové (aplikační) funkce, které zastupují v neuronových sítích velmi důležitou roli. Vysvětleny jsou také metody učení (učení bez učitele, učení s učitelem) a deep learning.

Po teoretické části práce seznamuje čtenáře s využitými technologiemi a následně postupy, způsoby řešení a některými komplikacemi, které se vyskytli při tvorbě aplikace. Dále následuje malá obrazová ukázka z aplikace a přehled splněných a nesplněných cílů.

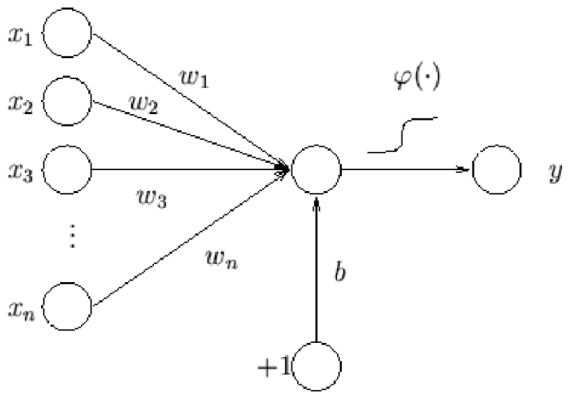
1 UMĚLÁ NEURONOVÁ SÍŤ

Umělá neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci. Jejím vzorem je chování odpovídajících biologických struktur. Skládá se z umělých neuronů, jejichž předobrazem je biologický neuron. Neurony jsou vzájemně propojeny a navzájem si předávají signály a transformují je pomocí určitých přenosových funkcí. Neuron má libovolný počet vstupů, ale pouze jeden výstup.

1.1 Perceptron

Perceptron je nejjednodušším modelem dopředné neuronové sítě. Sestává pouze z jednoho neuronu.

Perceptron se skládá ze vstupů $x = (x_1, x_2, x_3, \ldots, x_n)$, které jsou násobeny váhami (weights) $w = (w_1, w_2, w_3, \ldots, w_n)$. Tyto váhy určující důležitost jednotlivých spojů a jejich celkový podíl na ovlivnění aktivační hodnoty y. Následuje sečtení všech vážených spojů (vážený součet), přičtení prahu b a vyhodnocení aktivace pomocí přenosové (aktivační) funkce f.



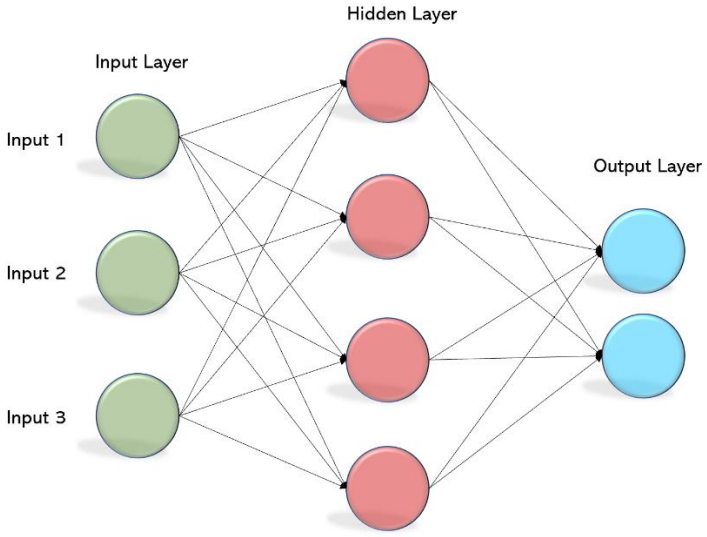
Obrázek 1.1: Schéma perceptronu

Při vstupech $x = (x_1, x_2, x_3, \ldots, x_n)$, kdy počet vstupů označíme jako n, můžeme vyjádřit výstup y jako:

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

1.2 Multilayer perceptron

Multilayer perceptron (MLP) neboli vícevrstvý perceptron je třída dopředné umělé neuronové sítě. Na rozdíl od jednovrstvého perceptronu může být vícevrstvý perceptron použit i pro řešení nelineárně separabilních problémů. Praktické je zejména jeho použití v případech, kdy máme k dispozici jen příklady vstupů a výstupů nějakého procesu.



Obrázek 1.2: Schéma MLP

Vstupní vrstva nijak nevypočítává svou aktivaci, pouze distribuuje vstupy do první skryté vrstvy, kde se až uplatňují stejné principy jako u jednoduchého perceptronu. Jednotlivé neurony si přeposílají své aktivační hodnoty jako vstupy do další vrstvy.

1.3 Přenosová funkce

Přenosová (aktivační) funkce určuje aktivaci neuronu. Zastupuje velmi důležitou roli a její výběr výrazně ovlivní chování celého modelu.

1.3.1 Funkce ReLU

Její základní podoba má výstup rovný nule, pokud je vstup menší než nula, jinak se výstup rovná vstupu. Tuto podobu lze definovat jako:

$$R(x) = \frac{x + |x|}{2}$$

1.3.2 Sigmoidní přenosová funkce

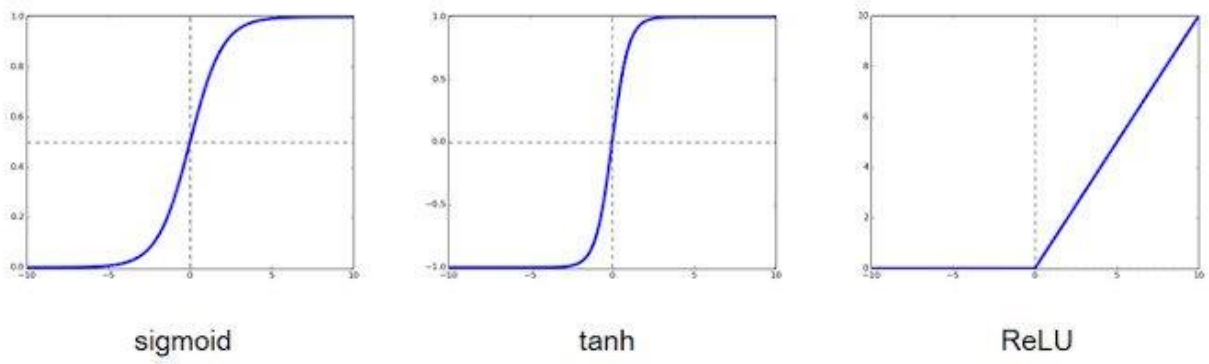
Jedna z typů sigmoidních funkcí je například logistická funkce. Logistickou funkci můžeme definovat jako:

$$S(x) = \frac{1}{1 + e^{-x}}$$

1.3.3 Přenosová funkce hyperbolické tangenty

Podobná jako sigmoidní funkce, může ovšem nabývat také záporných hodnot. Její hodnoty se blíží -1 v minus nekonečnu a jedničce v nekonečnu. Pro nulu je hodnota 0. Můžeme ji definovat jako:

$$T(x) = \frac{2}{1 + e^{-x}} - 1$$



Obrázek 1.3: Grafy přenosových funkcí

1.4 Učení neuronové sítě

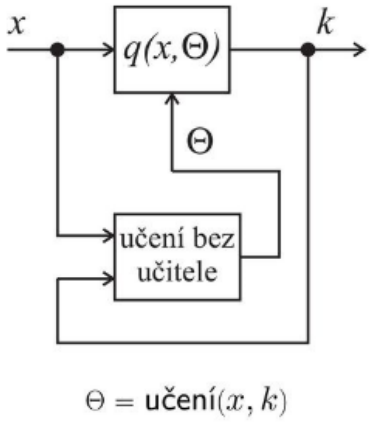
Cílem učení neuronové sítě je nastavit síť tak, aby dávala co nejpřesnější výsledky. Zatímco v biologických sítích jsou zkušenosti uloženy v dendritech, v umělých neuronových sítích jsou zkušenosti uloženy v jejich matematickém ekvivalentu – váhách. Učení neuronové sítě rozlišujeme na učení s učitelem a učení bez učitele.

1.4.1 Učení bez učitele

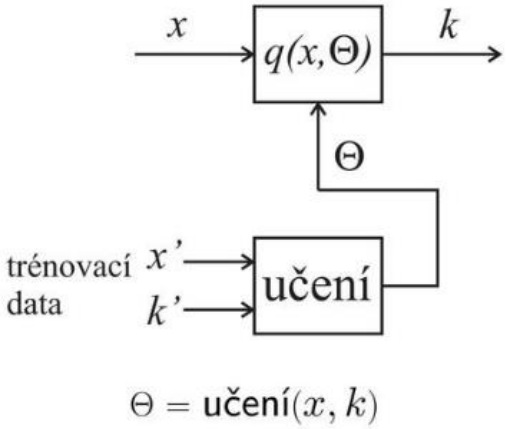
Při učení bez učitele nevyhodnocujeme výstup. Při tomto učení nám výstup není znám. Síť dostává na vstup sadu vzorů, které si sama třídí. Buď si vzory třídí do skupin a reaguje na typického zástupce, nebo si přizpůsobí topologii vlastnostem vstupu.

1.4.2 Učení s učitelem

Podobně jako v biologických sítích je zde využita zpětná vazba. Neuronové síti je předložen vzor. Na základě aktuálního nastavení je zjištěn aktuální výsledek. Ten porovnáme s vyžadovaným výsledkem a určíme chybu. Poté spočítáme nutnou korekci (dle typu neuronové sítě) a upravíme hodnoty vah či prahů, abychom snížili hodnotu chyby. Toto opakujeme až do dosažení námi stanovené minimální chyby. Poté je síť adaptována.



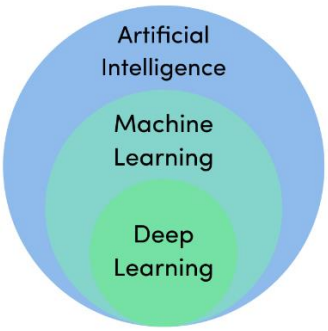
Obrázek 1.4: Schéma učení bez učitele



Obrázek 1.5: Schéma učení s učitelem

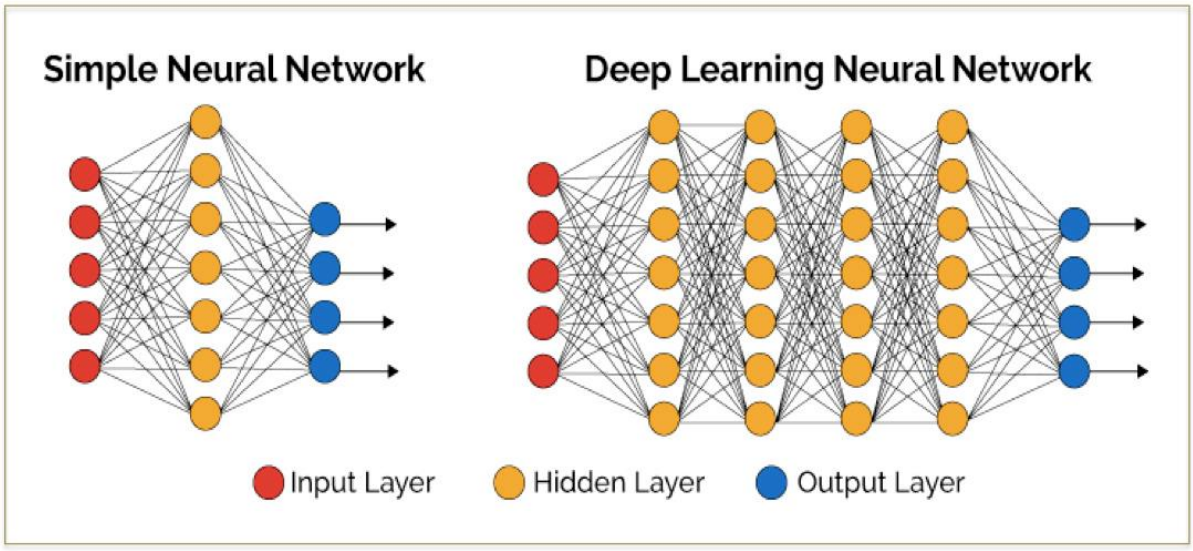
1.5 Deep learning

Deep learning, česky hluboké učení, je disciplína v rámci strojového učení, která se zabývá využitím algoritmů s velkým počtem vrstev reprezentujících data.



Obrázek 1.6: Deep learning jako podoblast

Slovo „hluboko“ v „hlubokém učení“ označuje počet vrstev, kterými se data transformují. Přesněji řečeno, systémy hlubokého učení mají podstatnou hloubku *cesty přiřazení kreditů* (*credit assignment path* – CAP). CAP je řetězec transformací od vstupu k výstupu. CAP popisují potenciálně kauzální spojení mezi vstupem a výstupem. U dopředné neuronové sítě je hloubkou CAP hloubka sítě a je počet skrytých vrstev plus jedna (protože je také parametrizována výstupní vrstva). U rekurentních neuronových sítí, ve kterých se signál může šířit vrstvou vícekrát, je hloubka CAP potenciálně neomezená. Žádný všeobecně dohodnutý práh hloubky nerozděluje mělké učení od hlubokého učení, ale většina vědců souhlasí s tím, že hluboké učení zahrnuje hloubku CAP větší než 2. CAP o hloubce 2 se ukázal jako univerzální aproximátor v tom smyslu, že může napodobovat jakoukoli funkci.



Obrázek 1.7: Rozdíl mezi jednoduchou neuronovou sítí a sítí s deep learningem

2 VYUŽITÉ TECHNOLOGIE

Celá aplikace je napsána v jazyce Python (3.7.9). Python je velice přehledný, dají se v něm jednodušeji řešit komplikovanější věcí a také nabízí přístup k mnoha užitečným frameworkům. V AI problematice je často využíván.

2.1 Kivy

Na grafickou část aplikace jsem využil knihovnu Kivy (2.0.0). Kivy je multiplatformní bezplatná grafická Python knihovna pro rychlý vývoj aplikací, které využívají inovativní uživatelská rozhraní. Výhodou knihovny Kivy je její přístup. Na rozdíl od ostatních knihoven, kde základem je opakující se vykreslovací funkce, klade Kivy spíše důraz na změny. To dělá celý kód přehlednější a rychlejší.

Obrázek 2.1: Kivy

2.2 TensorFlow, Keras

TensorFlow je bezplatná a otevřená softwarová knihovna pro strojové učení. Může být použit v celé řadě úkolů, ale má zvláštní zaměření na výcvik a odvozování hlubokých neuronových sítí. TensorFlow je symbolická matematická knihovna založená na toku dat a diferencovatelném programování. Pro samotné učení a predikci je použita knihovna Keras, což je nadstavba knihovny TensorFlow (2.1.0) specializovaná na umělé neurální sítě a umělou inteligenci. Použití takové knihovny nejen zjednodušuje celý učící proces, ale také ho zrychluje.

Obrázek 2.2: TensorFlow + Keras

3 ZPŮSOBY ŘEŠENÍ A POUŽITÉ POSTUPY

3.1 Stručný postup

Abych mohl implementovat AI, bylo nejprve potřeba vytvořit grafickou část aplikace. Součástí bylo také několik základních objektů a jejich metody. Po vyhotovení této části jsem vytvořil část druhou, jejíž součástí byla neuronovou síť. Tato část náhodně generuje vstupní data a přiřazuje k nim předpokládané výstupy. Všechny tyto data se vloží do modelu neuronové sítě, čímž se síť naučí předem. Úspěšnost naučení záleží na počtu vygenerovaných dat. Byla rovněž vytvořena metoda, která sbírá aktuální data z aplikace a posílá je do již naučené neurální sítě. Ta je vyhodnotí a vrátí zpět nejlepší možnost, podle které se aplikace dále chová.

3.2 Prostředí

Jedná se o rovinu představující moře, ve které se náhodně vygeneruje start, cíl a určitý počet překážek (skal). Následně se také vygenerují „lodě“, jejichž cílem je dorazit do cíle, aniž by narazily do skal. Lodě jsou ovládány pomocí AI, buď zatočí (doleva nebo doprava) anebo svůj směr pohybu nemění. Když narazí do skály, tak se jejich pohyb zastaví a považují se za ztroskotané. Při naražení na okraj mapy se jednoduše odrazí. Doražení do cíle rovněž zastaví pohyb lodě s tím, že se vypíše hláška do konzole, že daná loď zakotvila v místě určení.

3.3 Umělá inteligence

Základem bylo vymyslet jaké vstupy budu používat. Jako první pokus jsem si jako vstupy určil, zda se nachází objekt před lodí, napravo od ní, nalevo od ní a zda se blíží k cíli. Všechny takto vymyšlené vstupy nabývali hodnot 1 a 0 neboli True and False. Jelikož jsem se rozhodl použít učení s učitelem a počet situací které mohly nastat se rovnal $2^4 = 16$, rozhodl jsem se vytvořit si tabulku a určit, jak by se loď měla zachovat v jednotlivých situacích.

Vstupy:

- *f – před lodí*
- *r – napravo od lodi*
- *l – nalevo od lodi*
- *d – jede k cíli*

Výstupy:

- *-1 = odbočení doleva*
- *0 = nemění směr*
- *1 = odbočení doprava*

f	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
r	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
l	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
d	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
y	*±1**	*0*	*1*	*0*	*-1*	*-1*	*0*	*0*	*±1**	*±1**	*1*	*1*	*-1*	*-1*	*±1**	*±1**

*Můžeme si všimnout, že v některých situacích se nedá přesně rozhodnout, zda má loď odbočit doleva nebo doprava. Což značně komplikuje tento problém a potřebujeme nějakým způsobem určit, který směr bude loď preferovat.

Pro vyřešení tohoto problému jsem se rozhodl přidat jeden vstup (*t*) a to stranu na kterou se loď otáčela naposledy. Při vypuštění lodě ze startu ovšem žádná taková strana neexistuje, a proto ji náhodně vygeneruji při vytvoření lodě. Lodě se tak zároveň rozdělí na „praváky“ a „leváky“.

Nový vstup*:*

- *t = 0 – zatáčela doprava*
- *t = 1 – zatáčela doleva*

t	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*
f	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*1*
t	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*1*
l	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*1*
d	*0*	*0*	*1*	*0*	*1*	*0*	*0*	*1*	*0*	*1*
y	*-1*	*-1*	*1*	*-1*	*1*	*1*	*1*	*-1*	*1*	*-1*

*V tabulce jsou zobrazeny pouze situace, u kterých se nedal jednoznačně určit směr pohybu

Toto řešení se mi po dlouhém zkoušení zdálo jako nejefektivnější.

3.4 Ukázka kódu

Tato metoda zjišťuje aktuální data z aplikace a posílá je do natrénovaného modelu. Model určí předpokládaný výstup, podle kterého se loď zachová.

```
def ai_method(self,b):
    # směry na obě strany od lodě (o 30°)
    right = Vector(b.velocity_x, b.velocity_y).rotate(30)
    left = Vector(b.velocity_x, b.velocity_y).rotate(-30)
    #vytvoření proměných pro data posílané do modelu
    self.f, self.r, self.l, self.d = 0, 0, 0, 0
    # Nachází-li se nějaká skála před, napravo nebo nalevo od lodi,
    # nastaví se daná proměnná na 1 (True)
    for c in self.rocks:
        if (math.sqrt((b.center_x +b.velocity_x*10-c.center_x)**2+
           (b.center_y+b.velocity_y*10-c.center_y)**2)<(c.size[0]+5)/2):
            self.f = 1
        if (math.sqrt((b.center_x +right[0]*10-c.center_x)**2+
           (b.center_y+right[1]*10-c.center_y)**2)<(c.size[0]+5)/2): self.r = 1
        if (math.sqrt((b.center_x +left[0]*10-c.center_x)**2+
           (b.center_y+left[1]*10-c.center_y)**2)<(c.size[0]+5)/2): self.l = 1
    # Zjišťuje zda loď míří k cíli (min pod úhlem 25°)
    if abs(Vector(b.velocity).angle((self.finish.center_x - b.center_x,
            self.finish.center_y - b.center_y))) < 25: self.d = 1
    else: self.d = 0
    # vytvoření pole dat
    data=[b.t,self.f,self.r,self.l,self.d]
    # získá výstup z modelu na základě dat
    prediction = self.ai.predict(data)
    # Vyhodnotí-li model jako nejlepší výstup zatáčení doleva, nastaví se podle
    # toho promměná dir a nastaví se poslední směr ve kterém se loď otáčela
    if prediction == 0:
        dir = -1
        if self.d: b.t = 1
    # Vyhodnotí-li model jako nejlepší výstup zatáčení doprava, nastaví se
    # promměná dir a nastaví se poslední směr ve kterém se loď otáčela
    elif prediction == 2:
        dir = 1
        if self.d: b.t = 0
    # Loď pojede rovně
    else: dir = 0
    # Změna směru lodi podle výstupu z modelu
    b.velocity = Vector(b.velocity_x,b.velocity_y).rotate(10*dir)
```

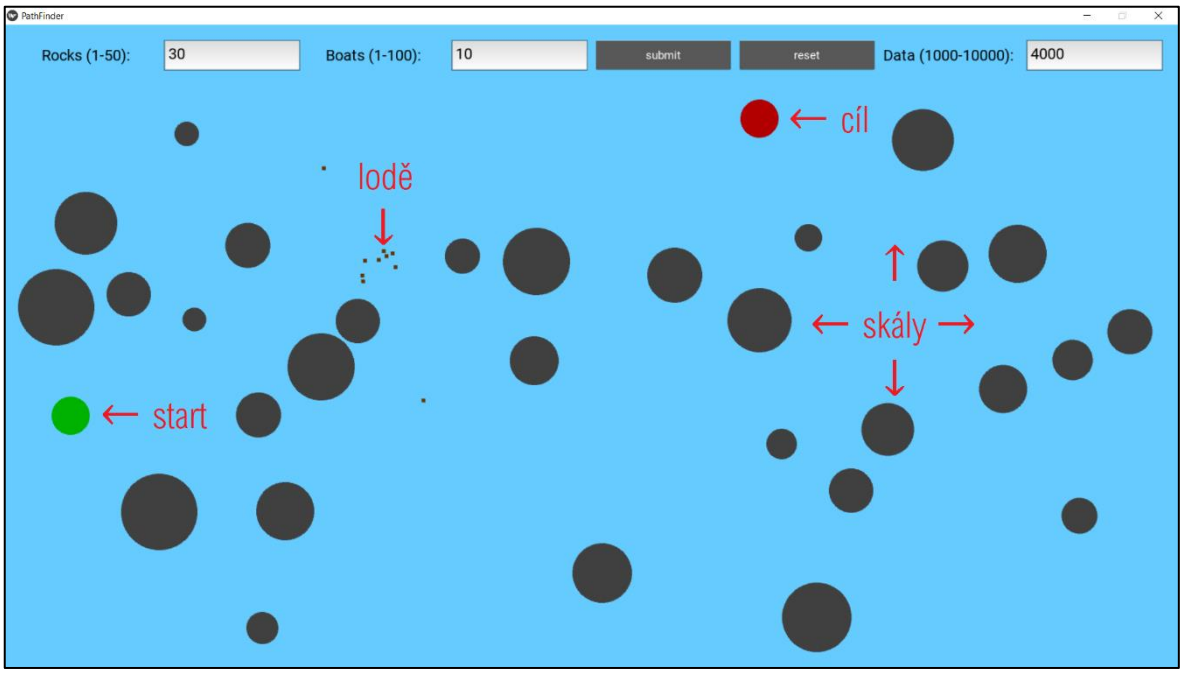
Ukázka modelu neuronové sítě:

```
def create_model(self):
    model = keras.models.Sequential()
    # Vstupní vrstva
    model.add(keras.layers.Dense(128, activation="relu"))
    # Skryté vrstvy
    model.add(keras.layers.Dense(128, activation="relu"))
    model.add(keras.layers.Dense(128, activation="relu"))
    # Vrstva, která rozhoduje nad nejlepším výstupem
    model.add(keras.layers.Dense(3, activation="softmax"))
    # Kompilace modelu
    model.compile(loss='categorical_crossentropy', optimizer='adam', me-
trics=["accuracy"])
    return model
```

4 VÝSLEDKY ŘEŠENÍ

4.1 Základní vzhled aplikace a možnosti uživatele

Uživatel může měnit počet vygenerovaných skal i lodí (v omezeném počtu). Změnu potvrdí tlačítkem „submit“ Uživatel má také možnost měnit počet dat, které budou zpracovány a podle kterých se lodě budou řídit. V aplikaci se nachází tlačítko „reset“, které vygeneruje novou mapu a přetrénuje celý model.



Obrázek 4.1: Snímek obrazovky z aplikace

4.2 Splněné cíle

- ✓ Pochopit problematiku AI (částečně)
- ✓ Funkční aplikace
- ✓ Možnost nastavení parametrů uživatelem
- ✓ Funkční neuronová síť/AI
- × DQN
- × Pestřejší grafické prvky

ZÁVĚR

Hlavním cílem bylo vytvoření aplikace s funkční neuronovou sítí a pochopení problematiky spojené s AI. Jedním z dalších cílů bylo umožnit uživateli zadávání základních parametrů, což se povedlo. S aplikací v momentálním stavu jsem spokojen. Cíl práce byl naplněn.

Naskytují se další možná vylepšení, jak grafické, tak především předělání na složitější neuronovou síť, která by byla schopna se učit sama v průběhu (DQN). Na to bych se rád zaměřil a v budoucnu se k tomuto projektu určitě ještě vrátím.

Odkaz na github:

https://github.com/Opkorny/final_project

SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ

[1] Deep learning. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-01-04]. Dostupné z: https://en.wikipedia.org/wiki/Deep_learning

[2] JULIANI, Arthur. Simple Reinforcement Learning with Tensorflow [online]. Aug 25, 2016 [cit. 2021-1-4]. Dostupné z: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-andneural-networks-d195264329d0

[3] KANTOR, Jan. Učení bez učitele [online]. Brno, 2008 [cit. 2021-1-4]. Dostupné z: https://core.ac.uk/download/pdf/30296714.pdf. Diplomová Práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. Vedoucí práce Ing. PETR HONZÍK, Ph.D.

[4] KLŮJ, Jan. Obecná umělá inteligence pro hraní her [online]. Praha, 2017 [cit. 2021-1-4]. Dostupné z: https://dspace.cuni.cz/handle/20.500.11956/90569. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta.

[5] Perceptron. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-01-04]. Dostupné z: https://cs.wikipedia.org/wiki/Perceptron

[6] Umělá inteligence. Matematická biologie [online]. Brno: Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity, - [cit. 2021-01-04]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence

[7] Umělá neuronová síť. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-01-04]. Dostupné z: https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A5