Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v ČB
Pedagogická fakulta
Katedra informatiky

Využití umělé inteligence pro analýzu dat z Bobříka informatiky

Use of artificial intelligence for data analysis from Bebras challenge

Bakalářská práce

Vypracoval: Luboš Zeman

Vedoucí práce: Mgr. Václav Dobiáš Ph. D.

České Budějovice 2022

Zadání bakalářské práce

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Luboš ZEMAN**
Osobní číslo:	**P19455**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Téma práce:	**Využití umělé inteligence pro analýzu dat z Bobříka informatiky**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování

Bobřík informatiky generuje velké množství anonymizovaných dat. Úkolem studenta bude vytvořit databázový systém pro zpracování dat z výstupu interaktivních úloh Bobříka informatiky. Na základě těchto dat student vytvoří a vytrénuje neuronovou síť. Po prvním sestavení programu v interaktivní úloze Bobříka informatiky bude neuronová síť predikovat pravděpodobnost úspěšného vyřešení celé této úlohy.
V teoretické části práce student popíše základy neuronových sítí, jejich rozdělení, principy a zásady návrhu neuronových sítí. V praktické části pak student popíše, proces tvorby jím tvořené neuronové sítě a výslednou neuronovou síť.

Rozsah pracovní zprávy:	**40**
Rozsah grafických prací:	**CD ROM**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. RUSSELL, Stuart J. a Peter NORVIG. Artificial intelligence: a modern approach. 3rd ed. Harlow: Pearson Education, c2014. ISBN 978-1-29202-420-2.
2. CHOLLET, François. Deep learning v jazyku Python: knihovny Keras, Tensorflow. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.
3. MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ a Jiří LAŽANSKÝ. Umělá inteligence (6): knihovny Keras, Tensorflow. Praha: Academia, 2013. Knihovna programátora (Grada). ISBN 978-80-200-2276-9.
4. VOLNÁ, Eva, Olga ŠTĚPÁNKOVÁ a Jiří LAŽANSKÝ. Umělá inteligence: rozpoznávání vzorů v dynamických datech. Praha: BEN – technická literatura, 2014. Knihovna programátora (Grada). ISBN 978-80-7300-497-2.
5. POINTER, Ian. Programming PyTorch for Deep Learning. Sebastopol, California: O'Reilly UK Limited, 2019. ISBN 9781492045359.

Vedoucí bakalářské práce: **Mgr. Václav Dobiáš, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **13. dubna 2021**
Termín odevzdání bakalářské práce: **30. dubna 2022**

L.S.

doc. RNDr. Helena Koldová, Ph.D.
děkanka

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 13. dubna 2021

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval(a) pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 30. června 2022

Luboš Zeman

Vlastnoruční podpis

Abstrakt

Bobřík informatiky generuje velké množství anonymizovaných dat. Úkolem studenta bude vytvořit a databázový systém pro zpracování dat z výstupu interaktivních úloh Bobříka informatiky. Na základě těchto dat student vytvoří a vytrénuje neuronovou síť. Po prvním sestavení programu v interaktivní úloze Bobříka informatiky bude neuronová síť predikovat pravděpodobnost úspěšného vyřešení celé této úlohy.

V teoretické části práce student popíše základy neuronových síti, jejich rozdělení, principy a zásady návrhu neuronových sítí. V praktické části pak student popíše, proces tvorby jím tvořené neuronové sítě a výslednou neuronovou síť.

Klíčová slova

Analýza dat, strojové učení, umělá inteligence

Abstract

Bebras challenge generates a large amount of anonymized data. Student's task will be to creat databases system for processing data from output of Bebras challange interactive exercise. Student will create and train neural network based on these data. Neural network will predict probability of accomplishment solution after first build of Bebras challenge exercise program.

Student will describe in theoretical part bases of neural networks, their division and design principles of neural networks. The practical part will describe process of creation student created neural network and the resulting neural network.

Keywords

Data analysis, machine learning, artificial intelligence

Obsah

1 Úvod

1.1 Úvod

Tato bakalářské práce se bude zabývat zpracováním dat z webové aplikace bobřík informatiky do přehledné podoby a následné sestrojení neuronové sítě a její vytrénování.

Součástí soutěže bobřík informatiky je otázka, kde se soutěžící snaží sestrojit algoritmus pro splnění úlohy podle zadání. Uživatel může každé své řešení vyzkoušet. Každý tento pokus se ukládá v kódu, který má podobu JSON a nese potřebné informace, jako je, číslo otázky, čas odeslání pokusu, dokončení nebo kolizi algoritmu, který reprezentuje algoritmus uživatele, ale je zaznamenán strukturou XML. Účelem mojí první aplikace bude, aby tato data převedla do přehledné podoby formátu CSV a očistila kód od přebytečných znaků, které jsou potřeba pro formáty JSON a XML. Také dopočítá počet pokusů uživatele a vypočítá časový rozdíl mezi nimi. Zbaví algoritmus XML a převede jej do přehledné podoby.

Další aplikace bude načítat data ze souboru CSV a získá potřebné informace a uspořádá je do podoby, která bude ještě přehlednější. Rozdělí se podle čísla úlohy, protože každá úloha má jiné řešení, zároveň rozdělení podle úlohy poskytne i rozdělení podle věku, protože každá věková kategorie bude mít jinou úlohu. Dál budou rozděleny podle každého pokusu, ale každý pokus bude obsahovat informaci posledního pokusu uživatele, tedy jestli se nakonec dostal ke správnému řešení a další informace, které by mohli mít spojitost s dokončením úlohy.

Poslední aplikace bude trénovat neuronovou síť. Pro její vytrénování je třeba převést data do vhodné podoby, ale už ne jejich manipulací, ale jejích převedení do podoby, které neuronová síť porozumí. Jsou dvě možnosti. Buď budou data reprezentována binárním kódem, nebo jako sada desetinných čísel v rozmezí od nuly do jedné, popřípadě od mínus jedné do plus jedné. Pak

se data rozdělí do tří množin. První množina bude trénovací, druhá validační a třetí testovací. To bude užitečné pro vývoj, protože se při běžných postupech sleduje úspěšnost odpovědí neuronové sítě na datech, které se používají pro trénování a na datech, na kterých se síť netrénuje a nakonec se výsledná úspěšnost vyhodnotí na množině dat, kterou je pouze testovací sada dat.

1.2 Cíle

V první řadě je nutné získat potřebné množství dat. To zajistí soutěž bobřík informatiky. Tato data bude třeba zpracovat a uložit do databáze. To obstará první program. Pro první kroky, jako je manipulace s daty a jejich převádění stačí malý vzorek. Ale pro hlavní účel, tedy vytrénování neuronové sítě, bude třeba už znatelně větší množina uživatelů. Čím více vzorků, tím bude neuronová síť přesnější. Přesnosti by měla dosáhnout alespoň 90 %. Celý proces bych rozdělil na 3 fáze.

1. Získání dat

   (a) Převod z JSON a XML

   (b) Překlad algoritmu

   (c) Zápis do CSV

   Tyto kroky jsou nezbytné pro možné zapsání dat do databáze, a zvýšení přehlednosti a možnou budoucí manipulaci.

2. Převedení dat

   (a) Převod na JSON

   (b) Překlad do binárního kódu

   (c) Rozdělení na skupiny

   Po dokončení této fáze budou data plně připravena pro trénování neurální sítě.

3. Vývoj neuronové sítě

   (a) Sestrojení modelu

   (b) Testování a nastavování parametrů

   Po několika iteracích všech kroků této fáze se vybere nejlepší varianta, která bude dosahovat největší úspěšnosti na předpovídání výsledků.

2 Teoretická část

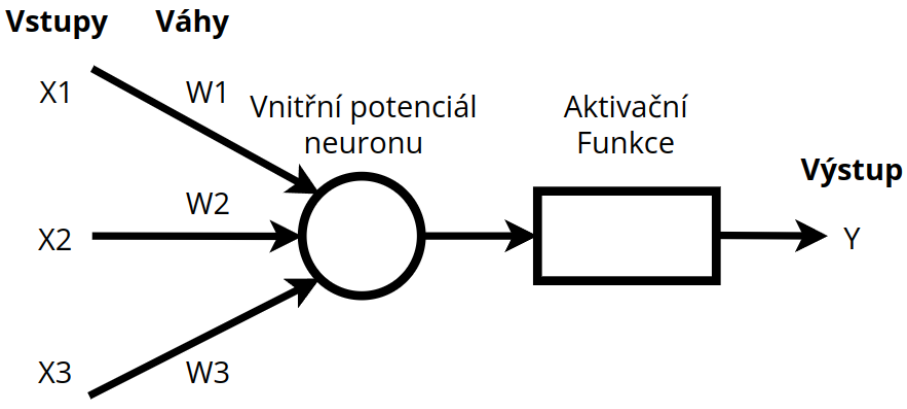
V této kapitole jsou popsány veškeré informace, které jsou nezbytné pro porozumění praktické části.

2.1 Neuronové sítě

Neuronová síť je rozložení jednotlivých neuronů do vrstev a jejich propojení. Tvar neurální sítě záleží na počtu neuronů v jednotlivých vrstvách a způsobu propojení jednotlivých neuronů. Účelem neuronových sítí může být určení třídy, do které spadá informace na vstupu, nebo neúplná data ze vstupu dokončit na úplná, nebo redukce vstupních dat, aniž by došlo ke ztrátě důležitých informací. [1]

2.1.1 Neuron

Základem neuronové sítě je formální neuron. Jeho struktura je složena z několika obecně reálných vstupů. Vstupy jsou ohodnoceny odpovídajícími obecně reálnými synaptickými váhami, které určují jejich propustnost. Vnitřní potenciál neuronu je závěrečná suma vstupních hodnot.[2]



Obrázek 1: Model neuronu

2.1.2 Vrstvy neuronových sítí

Vrstvy jsou specifické pro acyklickou architekturu, kde si lze neurony představit, že jsou řazené za sebou a vedou z jedné strany na druhou. Typicky síť obsahuje první vstupní vrstvu, která je tvořena ze vstupních neuronů a poslední vrstvu, která je výstupní a je tvořena výstupními neurony. Pro hluboké učení je specifické, že obsahuje více než dvě vrstvy neuronů. Tyto vrstvy se označují jako skryté vrstvy a jsou složeny ze skrytých neuronů.[3]

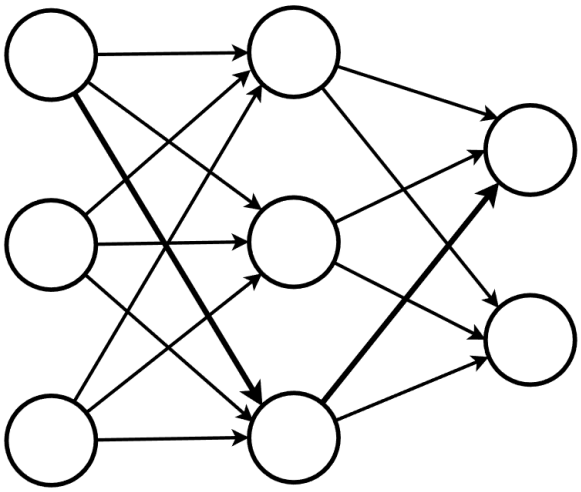
2.1.3 Architektura neuronových sítí

Architektura neuronové sítě se skládá z vrstev neuronů a jejich vzájemného propojení. Architektura vrstvy se dělí na dva typy: *cyklická (rekurentní)* a *acyklická (dopředná)*.[4]

Cyklická topologie obsahuje neurony, které jsou propojeny tak, že v jejich vazbách existuje smyčka (cyklus). Například tedy výstup prvního neuronu vede do vstupu druhého, výstup druhého do třetího a třetí zpátky do prvního. Další zajímavý příklad je, že výstup neuronu je vstup pro něj samotný. Naopak v acyklických topologiích neexistují smyčky a všechny cesty mají jen jeden směr.[4]

2.1.4 Hustě propojené neuronové sítě (DNN)

Hustě propojená neuronová síť je často acyklická a vrstvy v ní jsou hustě propojené. To znamená, že výstup jednoho neuronu z první vrstvy je vstup pro všechny neurony, které obsahuje následující vrstva. Vrstvy hustě propojených sítí jsou z pravidla ploché, tedy v reálném použití jsou jako vstup použity jenom jednoduché vektory. Nejčastěji se používají jako zakončení neuronové sítě, které mohou v prvních vrstvách pracovat z více rozměrnými vektory (tenzory). Pro jejich bezchybnou spolupráci je potřeba tenzory zploštit na vektory. [5]



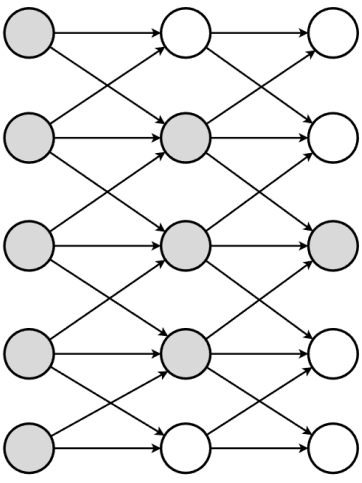
Obrázek 2: Hustě propojená neurální síť (DNN)

2.1.5 Konvoluční neuronové sítě (CNN)

Rozpoznávání objektů v obrázku na lidské úrovní je v poslední době nejzajímavější téma ve vývoji umělé inteligence například u samořídících aut. Dnes se používají různé kombinace způsobu propojení neuronů, ale základem tomu jsou konvoluční neuronové sítě. [6]

Zásadní rozdíl mezi hustě propojenou sítí a konvoluční sítí je v použití. Hustě propojená síť vyhledává vzory v celém vstupu, který dostane, zatímco konvoluční síť je hledá jen lokálně. Při učení rozpoznávání objektů v obrázku by to znamenalo, že pokud se konvoluční síť naučí, jak objekt vypadá, dokáže ho rozpoznat kdekoliv na obrázku, zatímco hustá síť to rozpozná jen na místech obrázku, kde se to naučila. [5]

Konvoluční sítě proti hustým sítím mají řidší propojení mezi neurony (řídké váhy). Toho je dosaženo tím, že je jádro menší, než vstup. Z toho vyplývá, že síť bude ukládat méně parametrů a bude mít nižší paměťové nároky a zvýší se statistická efektivita. Další zefektivnění má na svědomí vlastnost sdílení parametrů. V husté síti je každý prvek váhy použit jen jednou pro výpočet výstupu vrstvy. V konvoluční síti je hodnota váhy svázána s hodnotou váhy aplikovanou jinde. To znamená, že namísto učení samostatné sady parametrů pro každé umístění se učíme pouze jednu sadu. [7]

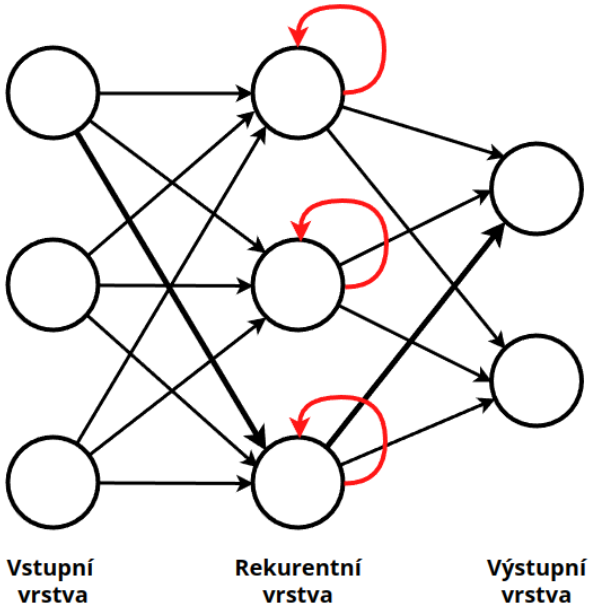


Obrázek 3: Konvoluční síť (CNN)

2.1.6 Rekurentní neuronové sítě (RNN, LSTM, GRU)

Rekurentní neuronová síť je název pro síť, která pomocí zpětnovazebního algoritmu umožňuje zpracovávat posloupnosti proměnlivé délky. Dá se říct, že je to dopředná síť s pamětí výstupů předchozího kroku. Je potřeba vybrat vhodnou aktivační funkci, jinak dojde k zahlcení pamětí. Speciální architektury, jako jsou LSTM sítě, obsahují síť pro výpočet výstupu a oddělené sítě, které rozhodují o tom, co je důležité uchovávat v paměti do dalšího kroku, a co ne. [8]

Stejně jako konvoluční sítě mají specializaci na rozpoznávání objektů z obrázků, tak rekurentní sítě mají specializaci na zpracování sekvenčních dat, jako je například vyvození informace z textu, ať je posloupnost slov v jakékoliv podobě (v souladu s pravidly jazyka). Konvoluční sítě i rekurentní sítě mají společnou vlastnost a tou je sdílení parametrů, což umožňuje aplikovat model na příklady různých forem (různých délek sekvence). [7]



Obrázek 4: Rekurentní neurální síť (RNN)

2.1.7 Vrstva Dropout (výpadek)

Vrstva Dropout náhodně nastavuje vstupní jednotky na 0 s frekvencí hodnocení v každém kroku během tréninku, což pomáhá předcházet přetrénování. Vstupy, které nejsou nastaveny na 0, se zvětší o 1/(1 - hodnocení), takže součet všech vstupů se nezmění.[9]

2.1.8 Vrstva Embedding (vnoření)

Vrstvy vnoření mají za cíl přetvářet lidská slova na vektory. Slova, která mají spolu vetší souvislost budou mít v geometrickém prostoru podobný směr a budou u sebe blíže, než slova, mezi kterými je souvislost nižší, nebo žádná.[5]

2.1.9 Vrstva Pooling (sdružování)

Sdružovací vrstvy se používají ke zmenšení rozměrů map prvků. Snižuje tak počet parametrů, které se mají učit, a množství výpočtů prováděných v síti. Sdružovací vrstva také shrnuje prvky přítomné v oblasti mapy prvků generované konvoluční vrstvou. Další operace se tedy provádějí se souhrnnými prvky namísto přesně umístěných prvků generovaných konvoluční vrstvou. Díky tomu je model odolnější vůči změnám polohy prvků ve vstupním obrázku. [10]

2.1.10 Vrstva Flatten (zploštění)

Vrstvy pro zploštění se používají pro zakončení v sítích, které mají více rozměrů na vrstvy, které mají jen jeden rozměr. Typicky se vkládají mezi konvoluční vrstvy, které mají 3 rozměry a hustou vrstvu, která má jen jeden rozměr. [5]

2.2 Umělá inteligence

2.2.1 Symbolická umělá inteligence

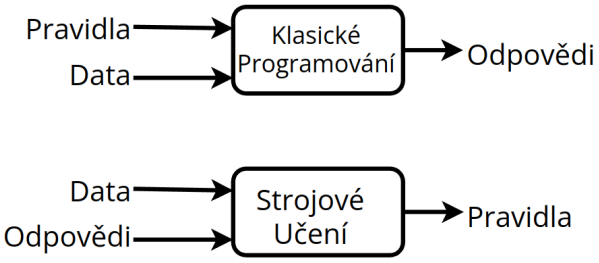
Hlavním úkolem umělé inteligence (artificial intelligence) je automatizovat intelektuální úlohy, které normálně provádí lidé. Původní principy spočívaly v tom, že se jasně definovaly podmínky a pravidla pro postup. To fungovalo skvěle pro umělou inteligenci, která uměla hrát šachy. Odborníci se dlouho domnívali, že umělá inteligence na úrovni člověka potřebuje mít dostatečně rozsáhlý soubor jasných pravidel pro chování v různých situacích. Tomuto přístupu se říká symbolická AI (artificial intelligence). [5]

Svého vrcholu symbolická AI dosáhla kolem 80. let. Pak se ukázalo, že přístup jasně definovaných pravidel je vhodný pro řešení logických problémů, ale už nestačí na řešení nejasných problémů, jako je klasifikace obrazů, rozpoznávání řeči atd...[5]

2.2.2 Strojové učení

Strojové učení vychází z otázky: Můžeme obejít programátory ručně zpracovaná pravidla pro zpracování dat, může se počítač automaticky naučit tato pravidla při pohledu na data? [5]

Předpoklad symboliké AI je, že dáme programu data a pravidla a dostane odpovědi. Strojové učení se liší tím, že dáme programu data, odpovědi a dostaneme pravidla.[5]

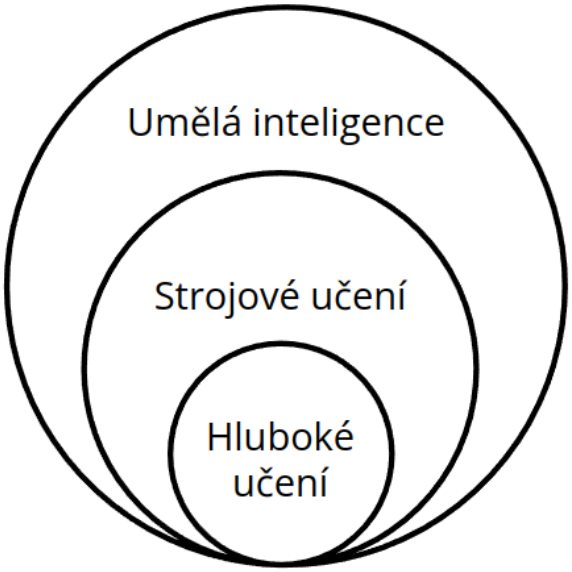


Obrázek 5: Paradigma strojového učení

Program, který je založený na strojovém učení je tedy spíše vytrénovaný na množině vhodných dat, která jsou relevantní pro daný problém. Program mezi daty objeví stálou schodu pravidel, která nám pak umožní onu automatizaci.[5]

2.2.3 Hluboké učení

Hluboké učení je podskupinou strojového učení, a to je podskupinou umělé inteligence.[5]



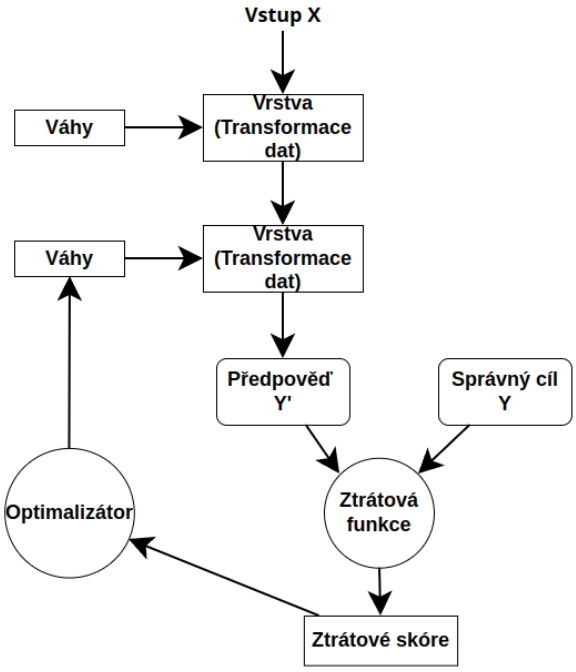
Obrázek 6: Umělá inteligence, strojové učení, hluboké učení

Hloubka v hlubokém učení (deep learning) nám označuje počet vrstev v neurální síti. To, co dělá vrstva neurální sítě se svými vstupními daty, je uloženo ve vahách vrstvy. Teď tedy učení znamená, že se zjišťují množiny hodnot pro váhy všech vrstev v síti tak, aby síť správně mapovala příklady vstupů na příbuzné cíle.[5]

Výstup neuronové sítě se pak porovná s původním výsledkem. To zařídí ztrátová funkce, která přebírá tyto dva výsledky a vypočítá ztrátové skóre. To nám označuje vzdálenost odhadované hodnoty od správné hodnoty.[5]

Protože jsou ze začátku váhy vrstev nastavený spíše náhodně, tak se vyu-

žívá mechanismus zpětné vazby. Optimalizátor podle zpětnovazebního signálu nastaví novou váhu vrstvám tak, aby se co nejvíce snížilo ztrátové skóre. To se provádí během trénovacího cyklu, který se opakuje tak dlouho, dokud nejsou váhy sítě správně nastaveny. Tento stav se nazývá natrénovaná síť.[5]



Obrázek 7: Postup hlubokého učení

2.2.4 Učení s učitelem

Učení s učitelem je založené na tom, že známe výsledky pro příklady, na kterých učíme síť. Tedy adaptační algoritmus má dostupnou množinu dvojic, které představují vstupy a správné výstupy pro řešení dané úlohy. Tato množina dat, která představuje učitele slouží k naučení sítě a ověření toho, zda správně funguje a jakých dosahuje výsledků. O správnosti výsledku tedy nikdo nerozhoduje a celé učení je založeno jen na vstupních datech. Tyto sítě mají často aktivní dynamickou strukturu, kdy se mohou po každém novém vzorku změnit i parametry sítě a učení může nabrat i nekonečný charakter. [11]

2.2.5 Učení bez učitele

V případě učení bez učitele adaptační algoritmus nemá žádnou množinu dat, která by ověřovala správnost výsledku. Princip učení je založen na shlukování podobných elementů do skupin na základě předpokládaných vzorků vstupních dat. Přitom počet hledaných skupin nemusí být předem definován. [12]

2.3 Konfigurace učebního procesu

2.3.1 Aktivační funkce

Aktivační funkce je nelineární funkce používaná k trénování neuronové sítě definováním výstupu neuronu zadaného sadou vstupů. Tato funkce se používá v každém neuronu sítě, aby pomohla síti naučit se rozpoznat složité vzory v datech, což jí umožňuje provádět předpovědi. Kromě toho je účelem aktivačních funkcí zavedení nelinearity do neuronových sítí. [13]

2.3.2 Ztrátová funkce

Ztrátová funkce ve strojovém učení počítá rozdíl mezi očekávaným výsledkem a výsledkem, který vypočítá model neuronové sítě. Výsledný rozdíl se nazývá ztráta. Zvyšuje se, jak se předpověď odchyluje od skutečného výsledku. Pokud je skutečný výsledek 1, model by měl vytvořit odhad pravděpodobnosti, který se co nejvíce blíží 1, aby se ztráta co nejvíce snížila. Pokud je skutečný výsledek 0, model by měl vytvořit odhad pravděpodobnosti, který se co nejvíce blíží 0. [14]

2.3.3 Optimalizátor

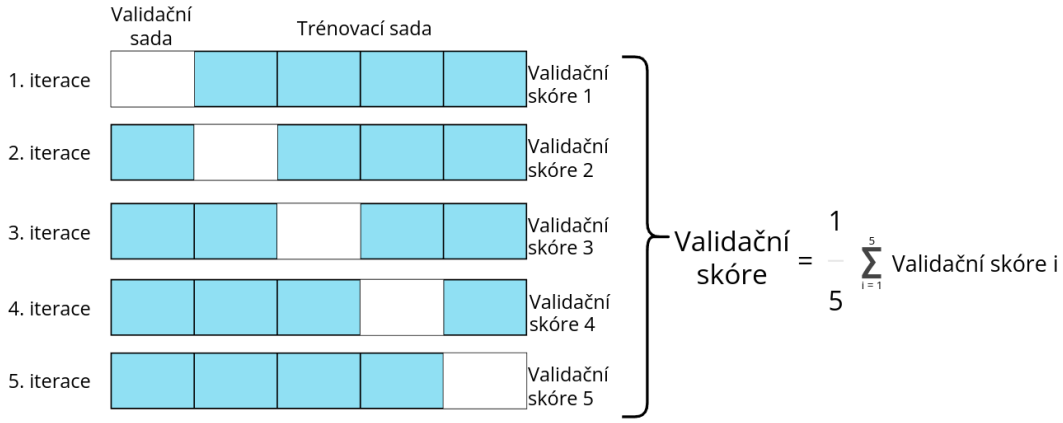
Optimalizátory jsou algoritmy, nebo metody používané ke změně atributů neuronové sítě, jako jsou váhy a rychlost učení pro poskytování co nejpřesnějších výsledků. [15]

2.4 Příprava dat

Zpracovaná data je potřeba rozdělit podle principu učení s učitelem na dvě, nebo na tři skupiny. Pro dvě skupiny se používá trénovací množina, která představuje poměr 70 %-80 %, z celkového počtu dat za předpokladu, že máme dostatečný vzorek dat, a množina testovací, která bude obsahovat zbylá data. V případě tří skupin se přidá skupina validační, která nám v průběhu učení po každém cyklu měří úspěšnost sítě. V případě, že se počet úspěšných výsledků nezvyšuje, tak máme natrénovanou síť. Hlavní přínos validační skupiny dat je vyhnutí se přeučení, nebo podučení sítě. [11]

2.4.1 K-násobná křížová validace

Pro vyhodnocení správnosti sítě na validační skupině při malém vzorku dat se využívá k-násobná křížová validace. Principem je rozdělení dat do k skupin, kdy jedna bude validační skupina a zbytek trénovací skupina. Při rozdělení na $k = 5$, se model učí na 80 % dat a na 20 % se validuje. To se provádí v cyklu s počtem opakování hodnoty k a rozsah validační a trénovací skupiny se liší v závislosti na čísle cyklu a poměru počtu dat a čísla k. Výsledek validační správnosti se ukládá do polí, ze kterých se vypočítá průměr validační správnosti pro jednu epochu. [5]



Obrázek 8: K-násobná validace

2.5 TensorFlow

TensorFlow je systém strojového učení, který funguje ve velkém měřítku v heterogenních prostředích. TensorFlow používá grafy toku dat k reprezentaci výpočtu, sdílený stav a operace. Pracuje na více výpočetních zařízeních, včetně vícejádrových CPU, univerzálních GPU a na zakázku navržených ASIC známých jako Tensor Processing Units (TPU). Tato architektura dává flexibilitu pro vývojáře aplikací: zatímco v předchozích *parametrových serverech* bylo navrženo řízení sdílených stavů zabudovaných do systému. TensorFlow umožňuje vývojářům experimentovat s novými optimalizacemi a trénovacími algoritmy. TensorFlow podporuje různé aplikace se zaměřením na trénink a vyvozování hlubokých neuronových sítí. [16]

2.5.1 Keras

Keras je knihovna pro hluboké učení, která je nyní součástí TensorFlow. Keras je pohodlný pro definování a trénování mnoho druhů modelů hlubokého učení. Díky přívětivému API Keras využívá spoustů uživatelů, ale i spoustu velkých firem, jakou jsou Google, Netflix, CERN. Kromě metod pro budování modelů, Keras také obsahuje mnoho databank, ze kterých lze modely učit.[5]

2.6 Chollet - hodnocení filmových recenzí

Tato kapitola ukazuje proces vývoje umělé inteligence na problému, který je podobný tomu, kterým se zabývá tato bakalářská práce.

François Chollet ve své knize, Deep learning v jazyku python, popisuje řešení pro hodnocení filmových recenzí. Chce vytvořit umělou inteligenci, která na základě obsahu textu recenzí filmů z datové množiny IMDB předpoví, jestli je recenze pozitivní, nebo negativní. [5]

Množina dat hodnocení IMDB je již součástí knihovny Keras, takže data jsou již zpracovaná do posloupnosti čísel, kdy každé číslo reprezentuje jedno slovo. Dále Chollet převádí data na vektory, které budou posloupností jedniček a nul.[5]

2.6.1 Využití DNN

Poté Chollet vytváří strukturu neuronové sítě, přičemž vysvětluje jenom základy. Podrobně návrh neuronové sítě vysvětluje až v dalších kapitolách. Zatím tedy použije vstupní vrstvu Dense se vstupním tvarem 10000. To je počet neuronů a každý vektor, který bude procházet neuronovou sítí musí mít stejný tvar. Tato vrstva má 16 výstupů, které jsou vstupem pro první skrytou vrstvu. I ta má 16 výstupů. Obě tyto vrstvy mají aktivační funkci *relu*, která slouží k vynulování záporných hodnot. Poslední vrstva Dense má jen jeden výstup a má aktivační funkci sigmoid, která dává hodnoty mezi nulou a jedničkou. Tuto hodnotu lze vnímat jako pravděpodobnost. [5]

Nakonec Chollet vybírá ztrátovou funkci a optimalizátor. Pro binární klasifikaci je nejlepší využít ztrátovou funkci *binary_ crossentropy*. Jako optimalizátor zvolil *rmsprop*. Konfiguraci kompilace modelu dokončil nastavení metriky na *accuracy*.[5]

Po připravení modelu už zbývalo ho natrénovat. Nejdřív Chollet rozdělil data na tři části. Trénovací skupinu, ze kterých se bude síť učit, validační skupinu, na které se sleduje úspěšnost sítě v průběhu učení, a testovací skupinu,

která nám ukáže úspěšnost sítě po jejím natrénování. Model neuronové sítě byl natrénován na dvacet epoch v minidávkách 512 příkladů. Po zobrazení průběhu učení na grafu bylo vidět, že validační ztráta stoupá po čtvrté epoše. Proto byla síť znovu naučena, ale už jenom na čtyři epochy. Takhle natrénovaný model se dostal na testovací množině dat na 88 %. Chollet dodává, že s nejmodernějšími přístupy by mělo být možné dosáhnout hodnoty úspěšnosti až 95 %. [5]

2.6.2 Využití RNN

Chollet se ještě vrátil k problému klasifikace IMDB recenzí filmu, kdy chtěl na řešení využít rekurentní neuronové sítě. Nejprve využil vrstvy *SimpleRNN*, se kterými se dostal na úspěšnost 85 %, která nepřekonává předchozí úspěšnost 88 %. To z důvodu, že tato malá síť kouká jen na první 500 slov a ne na celou sekvenci slov, kterou obsahovala recenze. Další problém je v tom, že *SimpleRNN* nefunguje nejlépe při práci s dlouhými sekvencemi, narozdíl od jiných složitějších rekurentních vrstev. [5]

Dále se tedy rozhodl zkusit složitější rekurentní vrstvu, a to *LSTM*. Model tedy vypadal tak, že začínal vrstvou *Embedding*, pak následovala rekurentní vrstva a nakonec hostá vrstvá s jedním výstupem.[5]

S tímto modelem, který byl naučený na 10 epoch se dostal k validační správnosti na 89 %. To je lehce lepší než výsledek u hustě propojených sítí, i když se i u tohoto pokusu použilo jen prvních 500 slov recenze a u hustých sítí plná délka recenze. Chollet tvrdí, že by výsledek mohl být i lepší, kdyby se věnovalo více času na optimalizaci hyperparametrů. Ale nejhlavnější důvod je to, že *LSTM* jsou dobré na globální, dlouhodobé struktury, ale to není úplně vhodné pro tento konkrétní problém. [5]

2.6.3 Využití CNN

Další možností pro zpracování sekvencí textu jsou jednorozměrné konvoluční sítě. Ty umožní v textu hledat lokální vzory. [5]

Síť začíná vrstvou *Embedding* (vnoření), Pokračuje vrstvou *Conv1D* o velikosti okna 7 vektorů příznaků a aktivační funkcí *relu*. Pak následuje vrstva *MaxPooling1D* (sdružení). Za ní je stejná vrstva *Conv1D* a další vrstva *GlobalMaxPooling1D* a hustá vrstva s jedním výstupem. [5]

Tento model natrénovaný na 10 epoch dosáhl nejlepšího výsledku ve čtvrté epoše, kdy dosahoval necelých 85 %. I když je tato správnost nižší než u rekurentních sítí, tak se tento model natrénoval rychleji. Tudíž konvoluční sítě mohou být rychlou alternativou pro rekurentní sítě.[5]

3 Praktická část

3.1 Úvod do problematiky praktické části

Tato část popisuje fungování jednotlivých komponent, které byly součástí procesu pro vytvoření umělé inteligence.

3.1.1 Blockly aplikace Bobříka informatiky

Bobřík informatiky je soutěž zaměřená na informatické myšlení. Tento rok se přidala blockly aplikace, která je inspirovaná robotem Karlem. Karel je programovací jazyk, ve kterém lze řídit pohyb robota po čtvercové mapě pomocí základních úkonů (krok, otoč se vlevo/vpravo, zvedni, polož), které se skládají do sekvence povelů.[17]

Soutěžící skládá kód z bloků. Pro správné dokončení je potřeba algoritmus spustit. Spuštění je možno provádět opakovaně. Pokaždé, když uživatel spustí algoritmus, uloží se záznam o jeho pokusu. Tyto data je třeba zpracovat.

3.1.2 DataConverter

DataConverter je aplikace, která slouží pro lepší interpretaci dat uložených z otázky soutěže bobříka informatiky. Tato uložená data jsou v podobě smíšeného kódu JSON a XML. JSON zobrazuje data, jako je čas, datum, číslo otázky, číslo uživatele. Obsahuje i položku pro kód, který uživatel použil pro splnění otázky, ale ten kód samotný je interpretovaný v XML. Tento zápis je velmi nepřehledný. Úkolem DataConveteru je data z JSON formátu přeložit do CSV a položku kód, která je v XML podobě zobrazil v podobě kódu, který se dobře čte, navíc přidat ještě data pro číslo pokusu uživatele a dopočítat čase mezi jednotlivými pokusy.

3.1.3 Umělá inteligence

Hlavním úkolem umělé inteligence je zadat kód uživatele a odhadnout, jak moc se blíží ke správnému výsledku na základě podobnosti kódu s kódy soutěžících, kteří se dostali ke správnému výsledku, nebo úlohu nesplnili.

Pro naučení umělé inteligence je potřeba zpracovat data z DataConverteru a rozdělit je na úspěšné a neúspěšné a vybrat jenom ta data, která jsou důležitá. Protože jeden záznam obsahuje až zbytečné informace, jako je datum, čas, druhá varianta kódu. Naopak důležitá data jsou číslo otázky a kód. Dále je potřeba znovu zpracovat tato data a převést je do podoby, kterou je umělá inteligence schopna zpracovat a učit se z nich.

Dalším důležitou částí pro uskutečnění hlubokého učení je sestrojení neuronové sítě, což zahrnuje správnou volbu vstupních, výstupních a skrytých vrstev, jejich počet. Dále je třeba nastavit správnou ztrátovou funkci, optimalizátor a aktivační funkci.

Po naučení neuronové sítě je třeba zhodnotit výsledek a zvážit jiný postup a nakonec vybrat ten, který bude dosahovat nejlepších výsledků.

3.2 Softwarová architektura DataConverteru

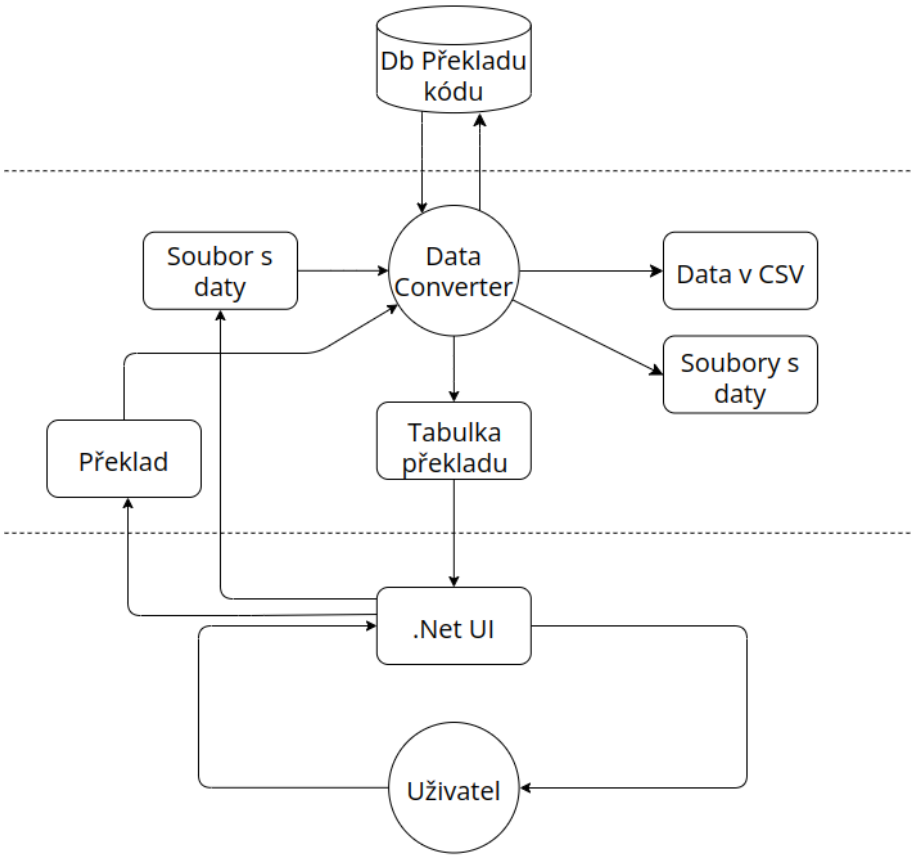
DataConverter je aplikace, která slouží ke zpracování dat, které produkuje soutěž Bobřík informatiky. Její detailní fungování a architektura je popsána v této kapitole.

3.2.1 Rozvržení aplikace

Aplikaci je rozvržena do tří částí. V první části je databáze překladů, která slouží pro překlad z kód XML do uživatelem zvolené podoby. Je to řešeno tak, že základní nastavení je zapsáno přímo v kódu. Další nastavení je v podobě CSV souboru, který obsahuje aktuální nastavení překladů. Další CSV soubor slouží pro vytvoření zálohy nastavení. Uživatel tedy může měnit překlad. Může přidávat, mazat i upravovat překlad. Když bude chtít provádět změny, u kterých si není jistý, může zálohovat aktuální stav databáze překladů. Když se něco pokazí, může načíst poslední zálohu, nebo základní nastavení, které je zapsáno přímo v kódu.

Další vrstva zpracovává uživatelem zadané LOG soubory, s daty ze soutěže Bobřík informatiky a překládá je do CSV souborů, nebo je rozděluje na 16 menších souborů.

Další vrstva slouží pro zpracování instrukcí uživatele. Uživatel může vybrat LOG soubor, který chce přeložit, nebo ho rozdělit na menší soubory. Nebo může vybrat místo pro uložení cílového CSV souboru s daty. Dále má také možnost pracovat s tabulkou, která slouží pro práci s databází překladů. Popřípadě vypnout/zapnout řazení dat.

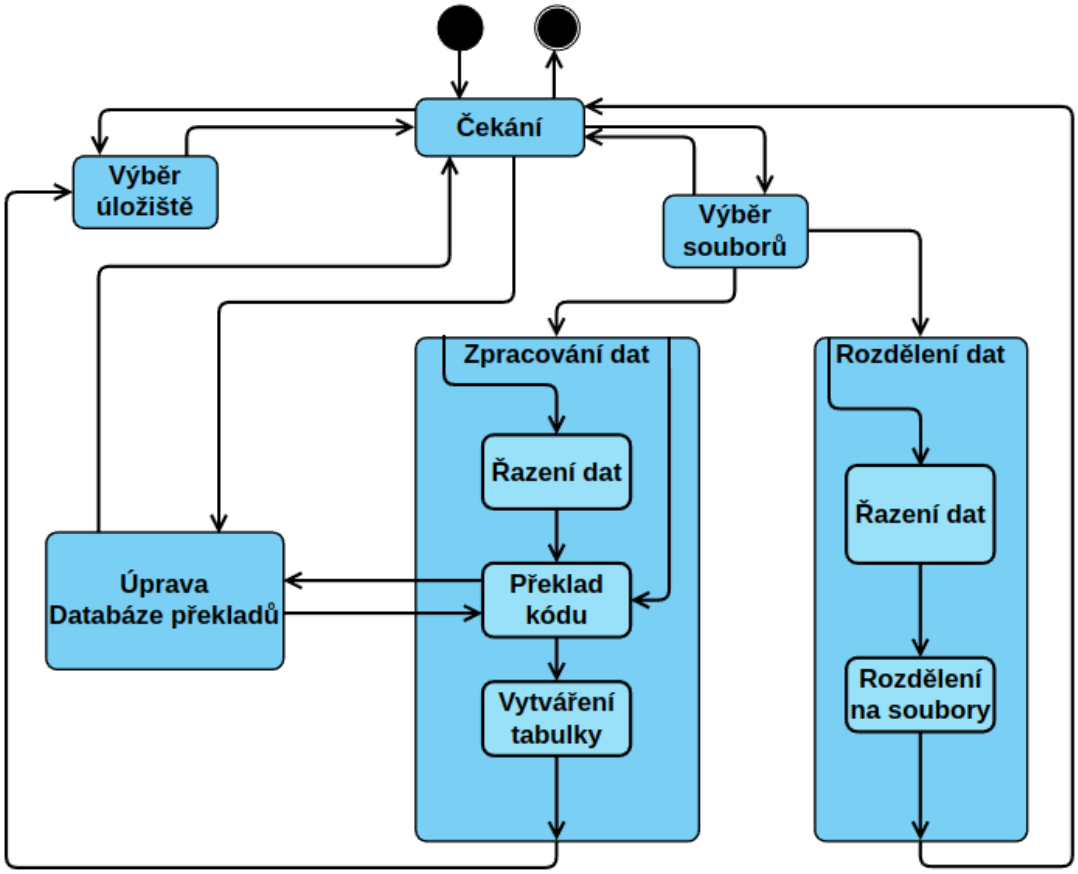


Obrázek 9: Architektura aplikace

3.2.2 Stavy aplikace

Aplikace se může dostat do různých stavů. Celá aplikace běží na jediném vlákně, takže v momentě, kdy se dostane do jiného stavu, než je čekání, nelze efektivně ovládat grafické rozhraní. Ideální scénář použití je po spuštění a stavu čekání vybrat soubor pro zpracování dat. Zpracování obrovských souborů může trvat i několik desítek minut až hodin. Jestliže uživatel nevybral předem místo uložení cílového souboru, aplikace přejde do stavu pro uložení výsledných dat a vrací se znovu do stavu čekání.

Pro zrychlení aplikace je přidáno řazení, které samotné trvá několik minut, ale pro budoucí běh programu ušetří několik desítek minut. Další věc, která šetří čas pro zpracování dat je rozdělení souboru na menší soubory. Po-



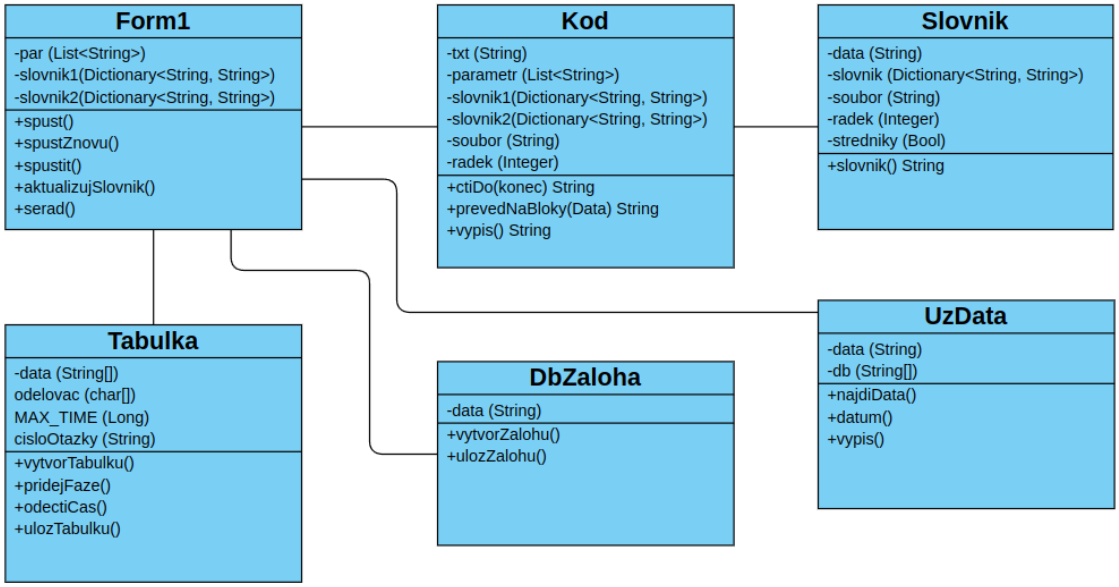
Obrázek 10: Stavový diagram aplikace

užité algoritmy pro zpracování dat jsou efektivnější při menším vzorku dat. Se zvyšujícím se počtem dat se doba běhu algoritmu prodlužuje. Protože se dopočítává počet pokusů a čas mezi pokusy a odpovědi jednotlivých uživatelů jsou na přeskáčku a nelze určit, který pokus je poslední, proto je potřeba data nejdříve seřadit. Dále běh programu zpomaluje práce s objekty, které si nárokují hodně paměti.

V průběhu překladu kódu může aplikace najít slovo, pro který není výraz, v ten moment vyzve uživatele, aby zadal překlad pro toto klíčové slovo. Dostává se tedy do stavu pro úpravu databáze překladů. Po potvrzení překladu se aplikace dostává zpátky ke stavu pro překládání kódu.

3.2.3 Diagram tříd

V této kapitole popíšu strukturu tříd jejich hlavní funkce.



Obrázek 11: Diagram tříd

Form1

Form1 je třída, která je jádro uživatelského rozhraní i funkčnosti aplikace. Je to komponenta, která představuje dialogové okno a spouští ostatní třídy, které vykonávají potřebné akce.

Obsahuje tlačítko button1. To spouští metodu, která uživateli umožní si vybrat soubory pro překlad (OpenFileDialog). Po výběru spouští další metodu která postupně pro všechny vybrané soubory a pro každý řádek souboru spouští třídu Kod s metodou vypis() a jejich výstup ukládá do Listu. Po dokončení práce na jednom souboru se spustí třída Tabulka s její metodou ulozTabulku() a obsah Listu převede na tabulku a uloží ji. Pak se obsah Listu vyprázdní a cyklus se opakuje pro další soubor.

Kod

Smyslem třídy Kod je převést obsah souboru na srozumitelná data. Jeho konstruktor přijímá parametry string kod (obsah souboru), List<string> para-

metry (název příkazů, které mají parametr), Dictionary<string,string> slovnik1 (první parametr obsahuje hledaný příkaz a druhý jeho překlad), Dictionary<string,string> slovnik2 (od prvního se liší překladem), string soubor (název souboru) a int radek (aktuální řádek v souboru).

Třída začíná u metody vypis(). Ta rozdělí kód na dvě části. První částí jsou uživatelská data. Pro ni je volána třída UzData s metodou vypis(). Ta v této části kódu najde potřebná uživatelská data, která pak vrátí v patřičné podobě. Druhou část rozděleného kódu zpracuje metoda prevedNaBloky(), která v kódu najde bloky, které se budou překládat. Dále tyto bloky předá třídě Slovnik s metodou slovnik() dvakrát. Pokaždé s jiným slovníkem (Dictionary slovnik1, slovnik2). slovníky tedy postupně probublávají skrz jednotlivé třídy.

Slovnik

Jak už název třídy napovídá, třída bude fungovat jako slovník v této aplikaci. Konstruktor přijímá argumenty string data(zdrojová data pro překlad), dictionary<string,string> slovnik, string soubor, int radek, bool stredniky. Aplikace překládá dva různé kódy, první středníky neobsahuje, ale druhý už jo. Středníky pro nás značí odřádkování. Existuje jeden univerzální parametr, který nejde dát do překladače, a to je číslo. Tudíž je tato informace pro nás důležitá, aby se středník vložil za číslo, které má mít také odřádkování.

Třída startuje metodou slovnik(). Která jednoduše projíždí zdrojová data a hledá jim překlad ve slovníku (Dictionary<string, string> slovnik). Jestliže se příkaz nenajde ve slovníku, tak informujeme uživatele. Uživatel buď přidá do databáze příkaz bez překladu, nebo zastaví překlad přidá ho sám i s překladem. Jestliže se přidá bez překladu, tak se běh zastaví, vyhodí se výjimka, která nese informace o tom, že je třeba přidat daný příkaz do databáze a proces se opakuje od čtení aktuálního souboru.

UzData

Tato třída je volána třídou Kod. Její konstruktor přebírá jediný parametr, a to string data. Což je část z kódu, ze které je třeba získat uživatelská data. Třída jednoduše rozluští formát json a vrátí hledaná data v náležitém formátu.

Tabulka

Tato třída je volána jednou s metodou ulozTabulku. Její konstruktor přijímá dva parametry. Jeden je string[] data, což je pole dat, které je třeba vložit do tabulky, a druhý, je název souboru. Při vytváření tabulky se jako první údaj zapíše id otázky, které se získá z názvu souboru, pak se postupně zapíší data. Po prvním vytvoření tabulky se do ní zapíší fáze a čas mezi fázemi. Pak vytvořená tabulka zapíše do souboru Table.csv. Jestliže soubor neexistuje vytvoří se, jestliže existuje, data se připíší na její konec.

DbZaloha

Tato třída bude dělat a spouštět zálohu databáze na příkazy a překlady vytvořený uživatelem. Bude obsahovat metodu vytvorZalohu, která bude přijímat proměnou var, která bude obsahovat výsledek dotazů na databázi. Ty se pak uloží jako soubor csv (Zaloha.csv). Další metoda obnovDb vrací data ze souboru Zaloha.csv a naplní s nimi databázi pro překlad.

3.3 Softwarová implementace

Níže je popsáno použití aplikace Dataconverter, detailně vysvětlená práce s daty a jejich příprava pro možné zpracování neuronovou sítí. Hlavně je zde popsána architektura neuronové sítě, její parametry a celý proces jejího učení, validace a testování.

3.3.1 Data csv souboru

První cíl je dostat data z LOG souboru do CSV souboru, ze kterého budeme později čerpat data. CSV soubor obsahuje tyto sloupce:

1. Datum
2. Čas
3. Id
4. Soutěžící
5. GameStopped
6. GameSucceeded
7. usedBlocks
8. codeLines
9. heroStepsCount
10. CollisionPointsCount
11. Fáze
12. Čas trvání fáze
13. kód (1. varianta)
14. kód (2. varianta)

To odpovídá jednomu řádku v LOG souboru, který je ve formátu JSON. Až na poslední 4 sloupce, které je třeba dopočítat, nebo jinak získat z jiného datového údaje.


```
[2021-02-09 15-08-51] {"id":11,"soutezici":11,"gameStopped"
:false,"gameSucceeded":false,"usedBlocks":5,"codeLines"7,
"heroStepsCount":8,"collisionPointsCount":0,
"heroSteps":[{"x":0,"y":8,"d":"up"},{"x":0,"y":7,"d":"up"},
{"x":0,"y":6,"d":"up"},{"x":0,"y":5,"d":"up"},
{"x":0,"y":4,"d":"up"},{"x":0,"y":3,"d":"up"},
{"x":0,"y":2,"d":"up"},{"x":1,"y":2,"d":"right"}],
"collisionPoints":[],,"recoveryData":"
<block type=\"start\" id=\":-gJgTIx{n7:fN:|eM3#\"
deletable=\"false\" movable=\"false\" x=\"0\" y=\"0\"><next>
<block type=\"rotate-left\" id=\"[qZ88^‘~whC/(.!wyOAf\"><next>
<block type=\"controls_repeat_ext\" id=\"q3lV,9nP[J?/v$1#6rs_\">
<value name=\"TIMES\">
<block type=\"math_number\" id=\"uxHkdoOJ{QWYnknzj^Qp\">
<field name=\"NUM\">7</field></block></value>
<statement name=\"DO\">
<block type=\"move\" id=\"c;el])]T(U%x-p1‘5K(Q\"/>
</statement><next>
<block type=\"rotate-right\" id=\"de|$44,.h}PyJixT~$R+\"><next>
<block type=\"move\" id=\"FKIqfo+2%@}JQ!D6[0-Y\"/>
</next></block></next></block></next></block></next></block>"}
@https://ibobr.pf.jcu.cz/test/blockly/save/1216
```


Listing 1: Příklad záznamu dat

```
"date":2021-02-09,
"time":15-08-51,
"id":11,
"soutezici"11,
"gameStopped":false,
"gameSucceeded":false,
"usedBlocks":5,
"codeLines":7,
"heroStepsCount":8,
"collisionPointsCount":0,
"collisionPoints":[],
"kod_1": SLO7(K)RK,
"kod_2":
start
vlevo_vbok
opakuj 7
(
  krok
)
vpravo_vbok
krok
```

Listing 2: Příklad výsledku

3.3.2 Sada dat

Pro učení umělé inteligence jsou vybrána pouze data z kolonky kód_1, která má kratší zápis. Data jsou zapsána v souborech result1.csv, result2.csv, result3.csv result4.csv. Python skript je přečte a každý řádek uloží jako jednu položku do pole rows.

```
import csv

filenames = ["data/result1.csv",
             "data/result2.csv",
             "data/result3.csv",
             "data/result4.csv"]

rows = []
for filename in filenames:
        with open(filename, 'r') as csvfile:

        csv reader = csv.reader(csvfile)

        for row in csvreader:
                rows.append(row)
```

Listing 3: Načtení dat

Dále se vyberou uživatelé, kteří došli ke správnému výsledku. Jejich použité kódy se uloží do pole *codes*, jestliže uživatel nedošel ke správnému výsledku, jeho kódy se uloží do pole *failcodes*, podle toho se do polí *codecount* a *failcount* zapíše číslo jedna. Jestliže se už kód v poli nachází, tak se jenom zvýší číslo počtu použitého kódu. Kódy jsou ze čtyř různých otázek, proto je důležité i rozlišovat to, ke které otázce kód patří. Protože to, co je správná odpověď pro jednu otázku není správně pro jinou. Proto se před kód zapisuje číslo otázky.

Protože soutěžící, kteří došli ke správnému řešení občas použili stejný kód jako soutěžící, kteří správného řešení nedosáhli, tak se kódy roztřídí podle toho, která skupina soutěžících použíla daný kód vícekrát. Pro účely roztřídění jsem vytvořil nová dvě pole s názvy *correctcodes* a *incorrectcodes*. Rozhodování záleželo na tom, jestli stejný kód víckrát použili úspešní, nebo neúspěšní soutěžící. Toto roztřídění je důležité proto, že pro učení neurální sítě je potřeba jedno-

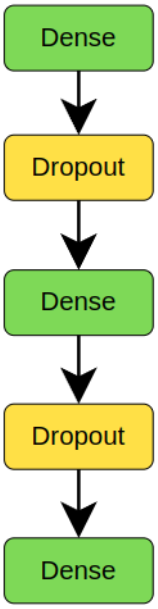
značnost trénovacích dat. Z toho důvodu z původních 582291 odpovědí zůstane pouze 11593 jedinečných kódů.

Data jsou pak uložena do souboru *LearningData.JSON*, který je poslední zdrojový soubor dat.

3.3.3 Model

Nejstěžejnější částí umělé inteligence je model neuronové sítě. Ten je definován vrstvami z knihovny Keras, která je již součástí frameworku Tensorflow. Nejdřív je definován model metodou *Sequential()*, která definuje, že vrstvy jsou zapojeny sekvenčně za sebou, a ne funkcionálně, nebo podtřídně. První vstupní vrstva je vrstva Dense, která má vstupní velikost podle délky vstupních dat (všechna data mají stejnou délku). Výstup je tvaru 1024, to znamená, že obsahuje 1024 neuronů, které mají 1024 spojů s dalšími neurony v následující vrstvě. Při nastavení nižšího, nebo vyššího počtu neuronů nedosahovala síť takové úspěšnosti při odhadování správného výsledku. Další vrstvou v síti je vrstva Dropout, která nám zajistí výpadek u patnácti procent neuronů, a tím zefektivnit učení. Bez vrstev Dropout dosahovala síť horších výsledků. Při nastavení výpadku na více než patnáct procent síť opět dosahovala horších výsledků. Při nastavení hodnoty výpadku nižší než patnáct procent síť dosahovala lepších výsledků než bez vrstev Dropout, ale horších než při nastavení patnácti procent, které jsou optimální pro tento konkrétní typ úlohy. Dále následuje další vrstva Dense a Dropout, které mají už jen 512 neuronů. Obě tyto vrstvy Dense mají aktivační funkci relu. Jen poslední Dense vrstva má aktivační funkci sigmoid, která se používá typicky při binární klasifikaci.[5]

Model je zkompilován ztrátovou funkcí *binary_ crossentropy*, optimalizační funkcí *rmsprop* a jako metriku správnosti *accuracy*. Toto bývá nejlepší varianta pro učení modelů zaměřených na binární klasifikaci.[5] A i v tomto případě byla nejefektivnější.



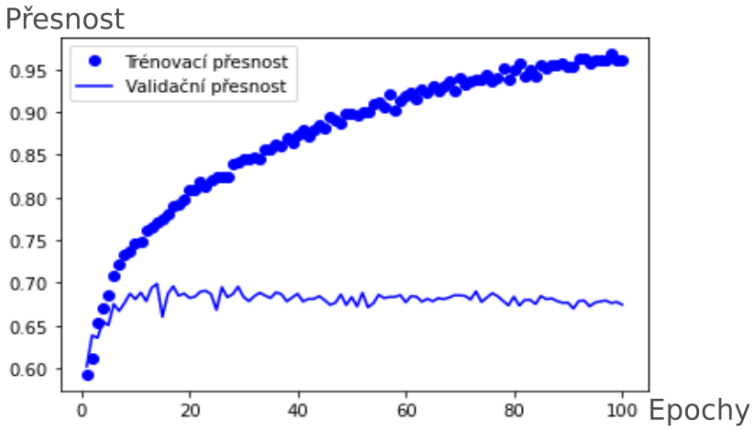
Obrázek 12: Model neurální sítě

3.3.4 Učení

Pro zahájení učení je důležité připravit data do podoby, která je schopna procházet neurální sítí. Rozhodl jsem se pro podobu v binární soustavě. Neurony budou jednoduše rozlišovat, jestli na jejich vstupu bude hodnota jedna, nebo nula. Nuly tak budou ignorovány. Kódy jsou načteny, každý nový znak kódu je uložen do pole *bagOfWords*. Do pole *binBagOfWords* jsou uložena data z bagOfWords, kdy jejich binární podoba se rovná jejich indexu zvýšenému o jedna, aby se nestalo, že první budou samé nuly. Aby měla všechna data stejnou délku, tak se doplní nulami do délky nejdelšího kódu. Sada dat pro určení správnosti bude pole o délce nejdelšího pole kódu a bude naplněna buď samými jedničkami, nebo nulami. Jedničky představují odpověď pro kód, který použili úspěšní řešitelé úlohy. Nuly pak představují odpověď pro kód, který použili neúspěšní řešitelé.

Účastníci při řešení úloh často používali shodné kódy. Z toho důvodu je z původních 582291 odpovědí jen 11593 unikátních kódů. To je méně, než jsem původně čekal. Z toho důvodu jsem využil speciální postup pro natrénování

sítě. Neuronová síť byla natrénována na všech vzorcích dat. Pro zkoumání validační správnosti jsem využil k-násobné validace. Data byla rozdělena na 10 částí, takže síť byla vždy natrénována z 90 % dat a při tom jsem sledoval validační správnost na zbylých 10 %. Tyto kroky se opakovaly desetkrát, ale pokaždé byla validační množina jiná část dat. Jejich validační výsledky se uložily a vypočítala se průměrná hodnota správnosti jedné epochy. Pak jsem ověřil celkovou správnost na testovací množině, která je úplná sada dat.



Obrázek 13: Průběh učení

Na grafu je vidět, že validační množina se blíží 70 %. To znamená, že na datech, na kterých se neuronová síť neučila a tudíž nezná jejich výsledek, dokáže určit správnost přibližně na 68 %. I když to není mnoho, tak taková situace nastane zřídka, protože z přes půl milionu odpovědí bylo jen přes jedenáct tisíc unikátních kódů. Proto je důležité natrénovat síť na co nejvyšší úspěšnost na kódech, které jsou k dispozici, a to i za cenu lehkého přetrénování, které lze na grafu pozorovat tak, že validační správnost lehce klesá, zatímco trénovací správnost stoupá.

Výsledek správnosti na testovacích datech je 95,99 %.

```
1 results= model.evaluate(np.array(training),np.array(result))
363/363 [==============================] - 0s 648us/step - loss: 0.1939 - accuracy: 0.9599
```

Obrázek 14: Testovací správnost

3.4 Experimenty

Tato kapitola popisuje další různé neuronové sítě a způsoby zpracování dat, které jsem zkoušel využít pro řešení úlohy. Další typy neurálních sítí by mohly odhalit skrytou souvislost, kterou jednoduché hustě propojené sítě neodhalily.

3.4.1 2D konvoluční sítě

Účelem tohoto experimentu je zjistit, jestli existuje souvislost mezi kódy, dalšími daty a úspěšným řešením. 2D konvoluční sítě se nejvíce využívají na zpracování obrázku, z toho důvodu umí pracovat s vícerozměrnými poli, kdy se rozměry předpokládá výška, šířka, hloubka obrázku. [5] Já jsem tyto vlastnosti využil pro zpracování různých informací z dat.

Příprava dat

Rozdíl v přípravě dat pro toto řešení je odlišný v tom, že jsem zkoumal všechny data, než jenom kód a číslo otázky. Konkrétně se jedná o data ze sloupců *Datum, Čas, Id, Soutěžící, GameStopped, GameSucceeded, usedBlocks, codeLines, heroStepsCount, CollisionPointsCount, Fáze, Čas trvání fáze, kód (1. varianta)*. Kód druhé varianty není zahrnut, protože by to byla redundantní informace.

Každá tato informace je zakódována do binární soustavy a každý tento binární kód má stejnou délku. Informace, které jsou jen čísla, jako je *Id, fáze, čísla přímo v kódu* a další, jsou zakódována standardním způsobem, jakým se kódují čísla z dekadické soustavy do binární. Informace, které jsou *true, false* jsou zakódována jako jedničky(true) a nuly(false). Časy jsou kódovány obdobně, jako čísla. Formát zapisování času je: *hodina:minuta:sekunda*. Odděleně jsou kódovány hodiny, minuty a sekundy do binární soustavy. A pak jsou jejich binární hodnoty zapsány za sebe. Kód je pak zakódovaný tak, že každý znak, který není číslice, má pevně stanovenou binární podobu, do které je zakódován.

Všechny zakódované informace by měly mít stejný tvar. U 55 zakódovaných

informací nastal problém. Některé odeslané kódy obsahovaly moc velká čísla, která měla exponenciální zápis. Tudíž tímto způsobem kódování, který jsem použil, se nezapsaly. Protože se jedná jen o experimentální pokus, rozhodl jsem se tyto data vyřadit.

Po vyřazení vadných pokusů měly všechny data stejný tvar. To znamená, že 582235 polí obsahovalo 65 polí, a každé z nich obsahovalo devítimístné pole, kdy každá jeho hodnota byla 1, nebo 0. Tato data pak byla rozdělena na tři téměř stejně velké skupiny (trénovací, validační a testovací).

Model

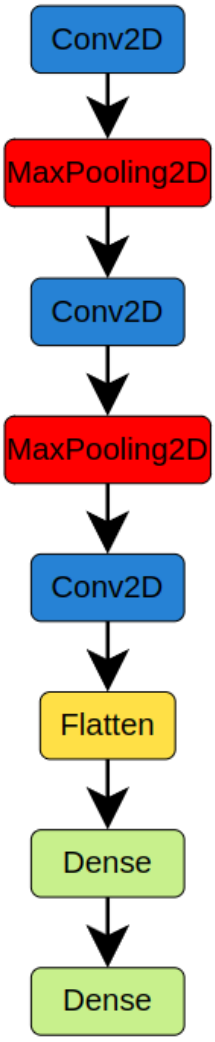
Vrstvy modelu jsou opět zapojeny sekvenčně. Model začíná 2D konvoluční vstvou (Conv2D). Běžně jsou konvoluční sítě používány na obrázky, proto parametry pro vstupy jsou výška, šířka, hloubka.[5] V tomto případě má vstup tvar (65,9,1). Protože výška a šířka nemají stejnou velikost, musel jsem nastavit parametr padding na hodnotu "same". To data rovnoměrně doplní o nuly, aby měly stejnou velikost na vstupu i na výstupu. [21] Další parametr *kernel_size* je n-tice 2 čísel, reprezentující vášku a šířku okna 2D konvoluce. [21] To je nastaveno na hodnotu (3,3). To znamená, že každá neurální buňka bude propojená s tři krát tři buňkami kolem sebe.

Další vrstvou je vrstva MaxPooling2D s nastaveným argumentem *pool_size* na (2, 2). To sníží vstup podél jeho prostorových rozměrů (výška a šířka) tak, že vezme maximální hodnotu z okna dva krát dva. Okno so posouvá dál o dva kroky. [22] Tím se získají nejdůležitější zakódované informace.

Tyto dvě vrstvy se ještě jednou opakují s tím rozdílem, že konvoluční vrstva má výstup 32 neuronů.

Pak následuje vrstva Flatten. Ta je nezbytná pro zakončení neurální sítě. Hustě propojené sítě totiž neumí pracovat s více rozměrnými poli. Tato vrstva přijímá více rozměrné vstupy a mění je na jednorozměrné. Další vrstvou je hustě propojená vrstva Dense, která má výstupní tvar 32 neuronů. Za ní už je poslední vrstva pro zakončení pro binární klasifikaci. Hlavní rozdíl mezi poslední vrstvou a všemi ostatními je, že má aktivační funkci *sigmoid*. Zbylé

vrstvy mají aktivační funkci *relu*.

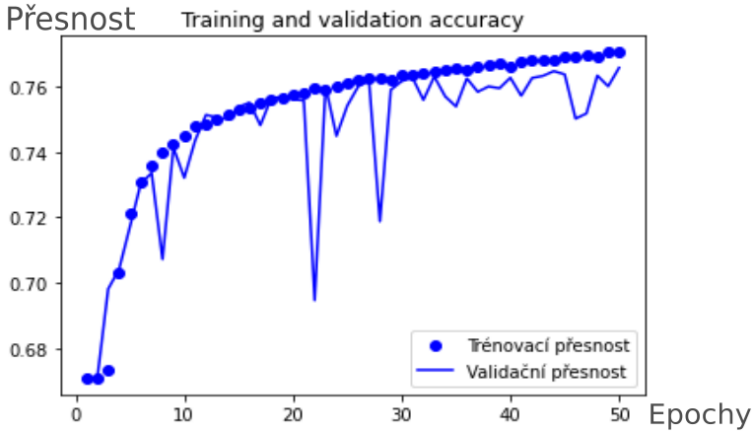


Obrázek 15: Model 2D konvoluční neurální sítě

Učení

Učení proběhlo na trénovacích datech na padesát epoch o velikosti dávky dat 256, což je počet vzorků, které se při učení budou šířit sítí. Po každé epoše byla data vyhodnocena na validační skupině, tedy na datech, na kterých se síť neučila.

Na grafu je vidět, že správnost předpovídání výsledků na validační skupině dat roste společně se správností předpokladů pro trénovací skupinu dat. Největší nárůst správnosti je po prvních patnácti epochách, a to o 7 %. Po dal-



Obrázek 16: Průběh učení konvoluční sítě

ších 35 epochách se správnost zvýšila už jen lehce přes jedno procento. Proto jsem netrénoval síť na více epoch. I kdyby se dala síť trénovat do doby, než by nastalo přeučení.

Po celém průběhu učení se správnost na testovacích datech dostala až na 76.49 %. Je to zhruba o 7 % přesnější výsledek než na průměru validační skupiny při použití jen hustě propojených sítí. Ale budu-li brát v potaz to, že chci předpovídat správnost na základě jen kódu a čísla otázky, tak je řešení na hustě propojených sítí efektivnější v tom, že by bylo obtížné vymyslet nový kód, který sítí ještě neprošel a neměl by společné charakteristiky s kódy, které použili úspěšní, nebo neúspěšní řešitelé úloh.

3.4.2 1D konvoluční sítě

Obecně se konvoluční sítě využívají pro zpracování obrázků. 1D konvoluční sítě se nejčastěji využívají pro zpracování sekvencí textu. [5] Moje úvaha byla taková, že když konvoluční sítě dokáží v obrázku rozpoznat objekt a ostatní data zanedbat, dalo by se to využít pro zpracování kódu, kdy síť naučím jenom na úspěšných kódech a kódech, které použili neúspěšní řešitelé úlohy. Síť by pak mohla rozpoznat jistý vzor, nebo klíčovou vlastnost v úspěšných řešeních. Konvoluční síť by pak dokázala tento vzor odhalit na jakékoliv pozici v ostatních kódech.

Příprava dat

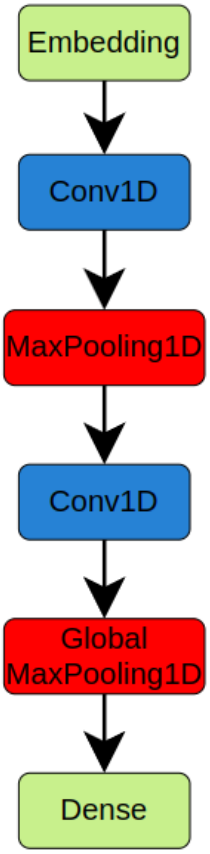
Data jsem hned ze začátku rozdělil jinak, než v předchozích případech. Rozdělil jsem je na skupiny úspěšné kódy, neúspěšné kódy a použité kódy. Kdy úspěšné kódy jsou jen správná řešení. Neúspěšné kódy jsou všechny kódy, které použili neúspěšní řešitelé. Použité kódy jsou kódy, které použili úspěšní řešitelé, ale nesplnili jimi úlohu. Kvůli jednoznačnosti jsem z použitých kódu odebral kódy, které použili neúspěšní řešitelé.

Kódy jsem zakódoval tak, že jsem cyklem projel všechny kódy, v každém kódu jsem projel každý znak. Jestliže se znak nenacházel v poli *bagOfWords*, tak jsem ho tam přidal. Pole *bagOfWords* obsahuje znaky, které se vyskytují v kódech. Výsledný jeden kód je reprezentovaný polem indexů daných slov v poli *bagOfWords*, který je zvýšený o hodnotu 1. Je to z důvodu, protože nulami jsem doplňoval zakódované kódy, aby všechny měli stejnou délku jako nejdelší kód. Kdyby jeden znak měl hodnotu nula, tak by ho síť mohla začít ignorovat.

Kódy jsou rozděleny do skupin úspěšné, neúspěšné a použité. Potřeboval jsem je rozdělit do skupin pro učení, testování a validaci. Ve skupině pro učení jsou kódy úspěšné a kódy neúspěšné. Ve skupině pro validaci jsou kódy úspěšné a neúspěšné. Ve skupině pro testování jsou kódy použité. Kódy ve skupinách pro učení a validaci jsou 50/50.

Model

Model je opět definován metodou *Sequiential()*. První vrstvou je vrstva Embedding, díky které se ze zpracované formy dat stanou vektory. Protože je její první parametr číslo 27, znamená to, že má výsledné vektory budou mít 27 rozměrů. Další vrstvou je 1D konvoluční vrstva (Conv1D). Za ní následuje vrstva MaxPooling1D. Co tyto vrstvy dělají bylo popsáno dříve. rozdíl je jen to, že nefungují ve dvou rozměrech, ale jen v jednom. Pak následuje znovu konvoluční vrstva Conv1D a s ní i vrstva GlobalMaxPooling1D. Protože nejsou ve více rozměrech jako 2D konvoluční sítě, není třeba vrstva Flatten pro sploštění a rovnou jsem navázal poslední vrstvou hustou vrstvou Dense, která měla aktivační funkci sigmoid. Konvoluční vrstvy měly aktivační funkci relu.

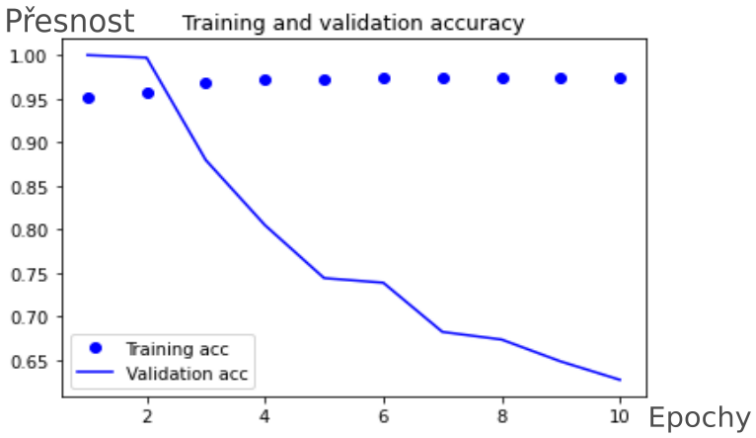


Obrázek 17: Model 1D konvoluční neurální sítě

Učení

Proces učení byl zvláštní. Správnost na validační množině prudce klesá a na trénovací množině se vyskytuje jen drobný nárust hodnot. V podstatě jsem chtěl síti akorát ukázat, jak vypadají správné kódy. Síť pak předpoví číslo mezi jedničkou a nulou, kdy když se číslo blíží víc k nule, znamená to, že je to neúspěšný kód. Čím více se blíží k jedné, tím více má společného s úspěšným kódem, který síť viděla. Validační skupina vlastně neobsahuje žádný úspěšný kód. Proto v tomto případě její správnost není úplně směrodatná.

Pro vyzkoušení testovacích dat, tedy dat, která jsou tvořena z kódů, který použili úspěšní řešitelé úlohy, ale nejsou správným řešením úlohy, a kódů, který použili úspěšní řešitelé, jsem vytvořil dvě metody. První metoda skládá kód do původní podoby. Druhá jako jediný parametr přebírá kód v binární po-



Obrázek 18: Průběh učení 1D konvoluční sítě

době, pak neurální síť určí předpověď výsledku a jaká je pravděpodobnost správnosti výsledku. Tedy čím více se pravděpodobnost blíží nule, tím je pravděpodobnější, že výsledek bude neúspěšný. Tato metoda pak vrací jen toto pravděpodobnostní číslo.

Pro testování jsem náhodně vybral 7 kódů, které jsou podobné správným odpovědím. Pro porovnání jsem vytvořil tabulku, kde je porovnání správného a použitého kódu vidět i s předpovědí neurální sítě. Ale na správnost tohoto experimentu se nedá spolehnout. Spolehlivější řešení by bylo, kdyby kódy byly označeny, jak moc se podobají správným řešením na škále od nuly do sta. A neuronová síť by byla zakončena regresivně. Tedy by předpovídala toto číslo v tomto rozmezí.

Porovnání výsledků

Otázka	**Správný kód**	**Použitý kód**	**Předpověď**
1379	SLLKRKKRKKLKKLKKKLKRK	SLLKKRKKLK	43,91 %
1379	SKKKRKKRKKKKKKRKKKKK	SKKKRKKKLKKKL	64,95 %
1391	SO5(K)RO5(K)RO5(K)	SO5(K)O5(RK)	89,08 %
1391	SO5(K)RO5(K)RO5(K)	SO5(K)RO5(K)RO5(R)	99,58 %
1379	SKKKRKKRKKKKKKRKKKKK	SKKKRKRKKRKK	61,44 %

Tabulka 1: Kódy správných řešení

3.4.3 Rekurentní neuronové sítě

Hlavní charakteristikou neurálních sítí je, že díky zpětnovazebním smyčkám se dá říct, že obsahují paměť, tudíž se dají využít na zpracování sekvencí dat, než jediné datové jednotky. [5] Mojí myšlenkou bylo, že by síť mohla předpovídat výsledek na základě celé sekvence kódů, který uživatel použil, než na základě jen jednoho použitého kódu.

Příprava dat

Data jsem roztřídil tak, že jsem nejdřív zjistil, kteří uživatelé se dostali ke správnému řešení. Pak jsem do jednoho pole sbíral kódy jednoho uživatele, a podle toho, jestli byl uživatel úspěšný řešitel jsem pole kódů zařadil do pole s úspěšnými řešeními. V opačném případě jsem jej zařadil do pole s neúspěšnými řešeními.

Jednotlivé kódy jsem opět zakódoval podle pole *bagOfWords*, kdy jsem projížděl každý kód a jestli byl kód v poli *bagOdWords*, tak ho reprezentoval jeho index v tomto poli navýšený o jedničku. Pak se takto zakódovaný kód přidal do pole *codesArray*, které obsahuje celou sekvenci použitých kódů uživatele. Pak se pole *codesArray* přidá do pole *training*, které obsahuje sekvence kódů všech uživatelů. Informaci o tom, jestli sekvence kódů je zakončena úspěšně je uložena do pole *result*. Protože hodnoty kódů obsahují velké hodnoty, tak jsem je vydělil hodnotou 255 a změnil datový typ na float32. Je to z důvodů, protože se při učení nastavuje váha na učicích datech a při velkých číslech by se změna váhy neprojevila.

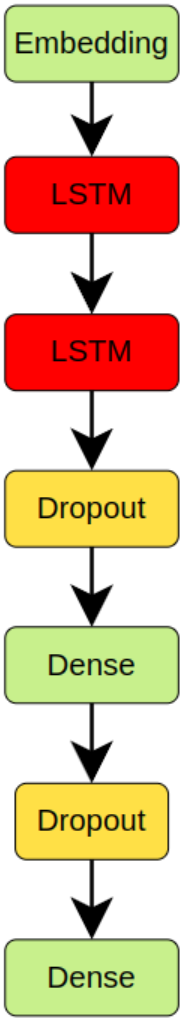
Pořadí dat jsem pak náhodně promíchal a rovnoměrně rozdělil do skupin pro trénování, validaci a testování.

Model

Model začíná vrstvou Embedding, která změní vstupní data na mnohorozměrné vektory. Za ní následují dvě LSTM (Long short-term memory) vrstvy. Ty jsou v tomto experimentu nejdůležitějšími vrstvami, protože obsahují zpětnovazební propojení narozdíl od všech ostatních vrstev, které jsem doteď po-

užil. Za ní následuje vrstva Dropout, která díky výpadku zlepší úspěšnost při učení. Za ní už je hustě propojená vrstva Dense, znovu vrstva Dropout a poslední vrstva Dense, která ukončuje celou síť do binární klasifikace.

Další změna v tomto modelu je, že aktivační funkci relu obsahuje jenom předposlední vrstva Dense, všechny ostatní vrstvy mají aktivační funkci sigmoid.



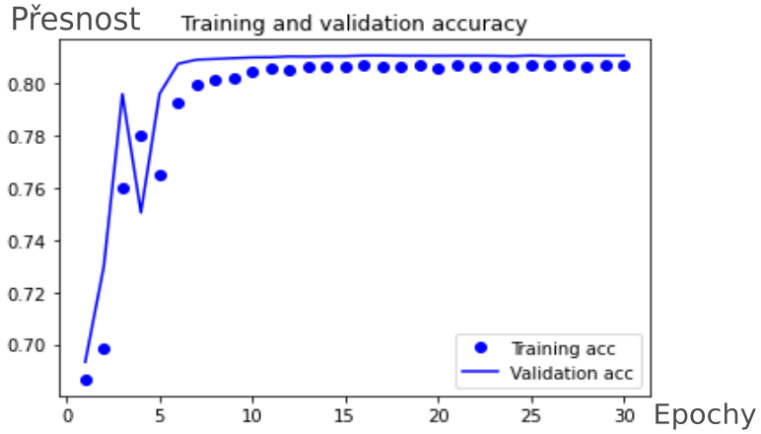
Obrázek 19: Model Rekurentní sítě

Průběh učení

Na grafu průběhu učení učení je vidět, že správnost validační skupiny přesně kopíruje trénovací skupinu. Zajímavostí je, že validační správnost lehce převy-

šuje trénovací správnost.

Na grafu je také vidět, že správnost po patnácté epoše už neroste. Proto je zbytečné trénovat síť dál, i když nedošlo k přeučení. Celkově se správnost pohybuje kolem 81 %. Na testovací množině dat vyšla správnost na 80,098 %. To není velká úspěšnost. Ale shodly se na ní všechny tři skupiny dat.



Obrázek 20: Průběh učení rekurentní sítě

Tato úvaha na zpracování kódů jako sekvencí mi nakonec přišla zbytečně složitá. Protože se v podstatě stačí koukat na poslední kód celé sekvence, podle kterého poznáme jestli je řešení správné nebo ne. Všechny předchozí kódy jsou v podstatě nadbytečná data.

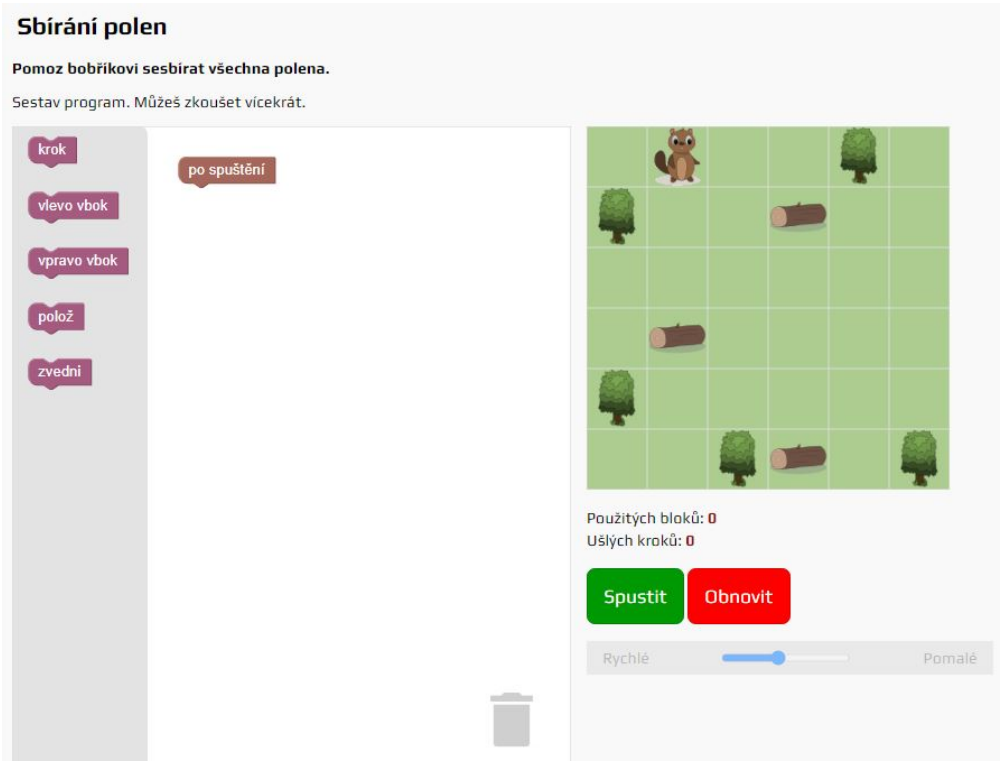
3.5 Nejpoužívanější kódy

Hlavním účelem této kapitoly je představení konkrétní podobu čtyř zkoumaných úloh a jejich možná řešení, ke kterým soutěžící nejčastěji došli.

3.5.1 Úloha 1376

Na první úloze je vidět, že má uživatel k dispozici jen pět variant bloků. Proto jsou všechna použitá řešení téměř stejná. To může být problém pro neuronovou síť, kdy v kódu bude nedostatek vzorů, které odlišují.

Jeden ze základních předpokladů pro splnění úlohy je, že když má bobřík zvednout tři polena, tak se v kódu tři krát objeví písmeno Z (zvedni) a také tímto písmenem bude končit. Za zmínku stojí poslední dva kódy v tabulce, které mají kratší délku, než všechny předchozí a mezi sebou se liší pouze tím, jakým směrem se bobřík otáčí.



Obrázek 21: Úloha 1376

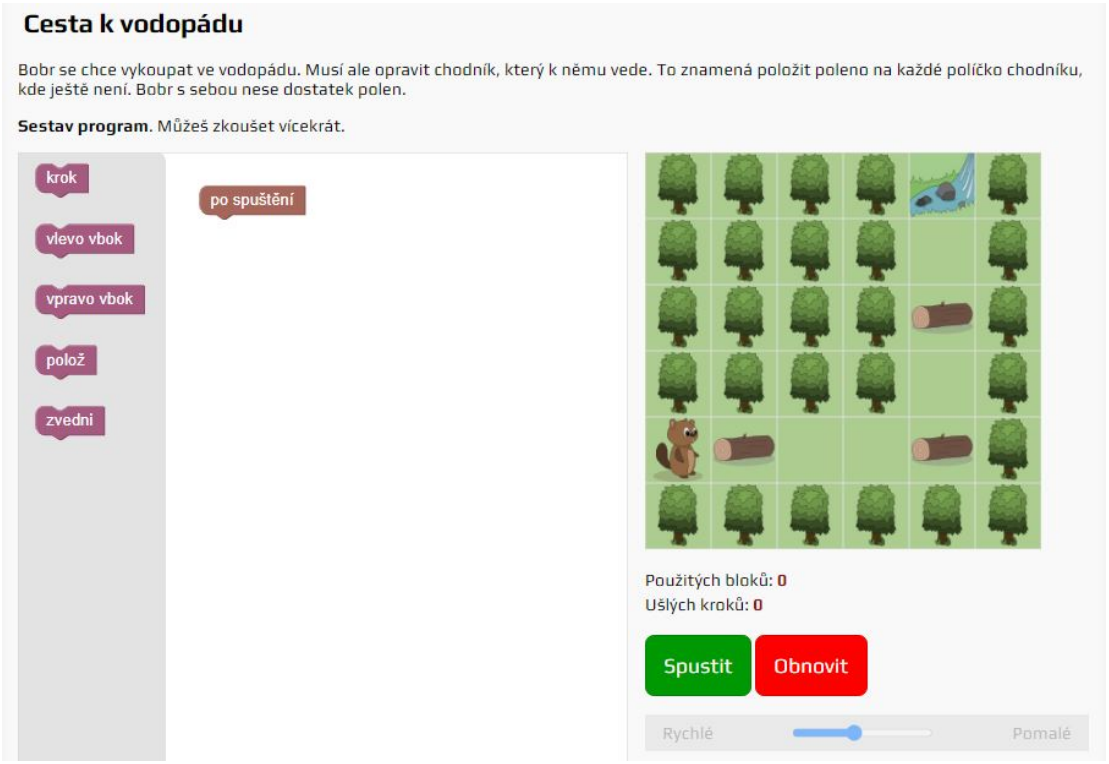
Kód	**Počet použití**	**Předpověď**
SKLKKZRKKRKKZLKLKKRKZ	2417	0.998158
SLKKRKZKKRKKZLKLKKRKZ	1459	0.83214223
SLKKRKZKRKKLKZKLKKRKZ	501	0.93810916
SLKKRKZRKKLKKZLKKRKKZ	401	0.96430165
SLKKRKZKRKKLKZLKKRKKZ	334	0.9766025
SKLKKZRKKRKKZLLKKRKKZ	267	0.99931407
SKLKKZRKRKKLKZKLKKRKZ	265	0.654086
SKLKKZRKRKKLKZLKKRKKZ	231	2.7029148e-05
SKKKZLKKLKKZRRKKKKZ	207	0.5870492
SKKKZLKKRKKZLLKKKKZ	197	0.93812704

Tabulka 2: Nejpoužívanější úspěšná řešení otázky 1376

3.5.2 Úloha 1378

Druhá úloha opět obsahuje pouze jen pět bloků, které lze použít, z čehož dává smysl použít pouze tři bloky, a to bloky krok, vlevo vbok a polož. Základním předpokladem pro splnění úlohy je, že kód začne blokem polož a bude končit blokem krok. Dále je vidět, že je potřeba poleno položit na pět políček, takže kód musí obsahovat písmeno P (polož) pět krát.

Z tabulky nejčastěji použitých správných řešení pro tuto otázku je vidět, že nejvíce uživatelů šlo nejkratší možnou variantou. Zajímavé je také to, že na druhém místě je téměř stejný kód, který se liší pouze tím, že bobřík již položená polena zvedne a znovu položí. Což má za následek, že je kód o šest znaků delší a používá se v něm čtyři různé bloky. Dále jsou různé varianty, kdy bobřík poleno zvedne a znovu položí a někdy ho nechá již položené.



Obrázek 22: Úloha 1378

Kód	**Počet použití**	**Předpověď**
SPKKPKPKLKPKKPK	14009	0.96924067
SPKZPKPKPKZPLKPKZPKPK	368	1.0
SPKZPKPKPKLKPKKPK	61	0.999993
SKZLLKPRRKPKPKPKLKPKKPK	49	0.015467614
SPKZPKPKPKZLPKPKZPKPK	46	1.0
SPKKPKPKRRRKPKKPK	23	0.97500396
SKZLLKPLLKPKPKPKLKPKKPK	23	0.2761082
SPKZPKPKPKLKPKZPKPK	20	0.95899016
SPKZPKPKPKLZPKPKZPKPK	18	0.9003872
SPKZPKPKPKZPLKPKKPK	16	1.0

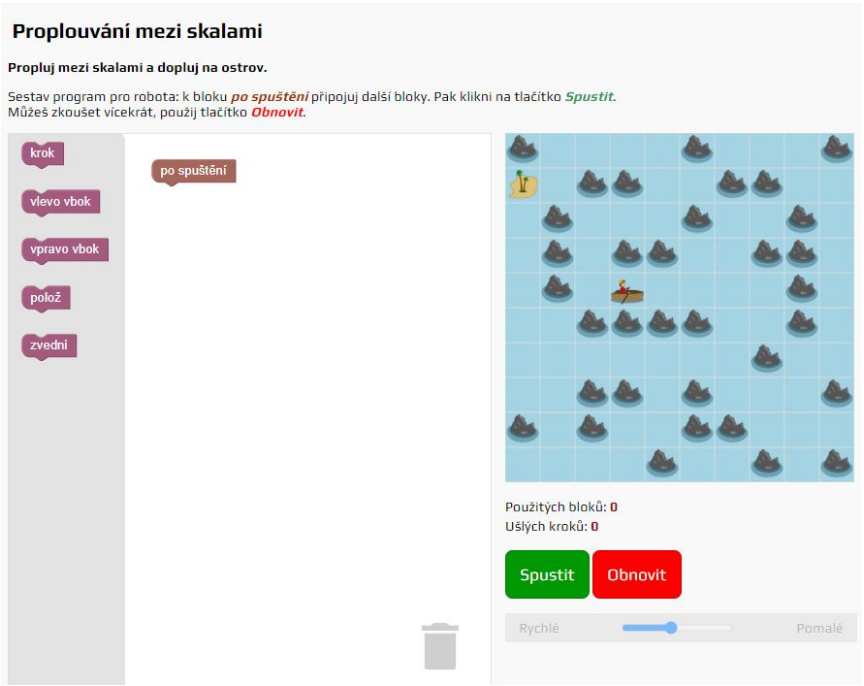
Tabulka 3: Nejpoužívanější úspěšná řešení otázky 1378

3.5.3 Úloha 1379

I v této úloze lze použít pouze pět různých bloků, přičemž dává smysl použít bloky krok, vlevo vbok a vpravo vbok. V této úloze není nic ke zvedání, či položení, tak není důvod tyto bloky používat. Dále je vidět výrazný rozdíl oproti předchozím úlohám v tom, že má tato úloha výrazně větší mapu, tudíž může nabízet více možných řešení.

Z tabulky nejvíce použitých kódů je vidět, že nejvíce uživatelů se vydalo s loďkou rovně a k ostrovu přijeli zespoda. Druhé nejpoužívanější řešení začíná tím, že se loď hned ze začátku otočí a přijede k ostrovu ze shora. Zajímavé je, že třetí nejpoužívanější řešení je to samé, jako to druhé, ale s rozdílem hned ze začátku, kdy se loď otáčí. Mnohem více soutěžících začalo s otáčením použitím dvou bloků vlevo vbok, kterých bylo přes sedm tisíc, a pouze přes jeden tisíc uživatelů použilo dva bloky vpravo vbok.

Další použité kódy jsou podobné prvním třem jen s rozdílem, že končí blokem zvedni, nebo polož, nebo oběma.



Obrázek 23: Úloha 1379

Kód	**Počet použití**	**Předpověď**
SKKKRKKRKKKKKKRKKKKK	11129	0.9735694
SLLKRKKRKKLKKLKKKLKRK	7285	0.985055
SRRKRKKRKKLKKLKKKLKRK	1158	0.9999968
SKKKRKKRKKKKKKRKKKKKP	725	0.9997414
SLLKRKKRKKLKKLKKKLKRKP	314	0.96390444
SKKKKRKRKLKRKKKKKKRKKKKK	210	1.0
SKKKRKKRKKKKKRKLKRKKKK	183	0.9999869
SKKKRKKRKKKKKKRKKKKKPZ	106	0.43838876
SRRKRKKRKKLKKLKKKLKRKP	62	0.9993739
SLLKRKKRKKLKKLKKKLKRKPZ	30	0.99825954

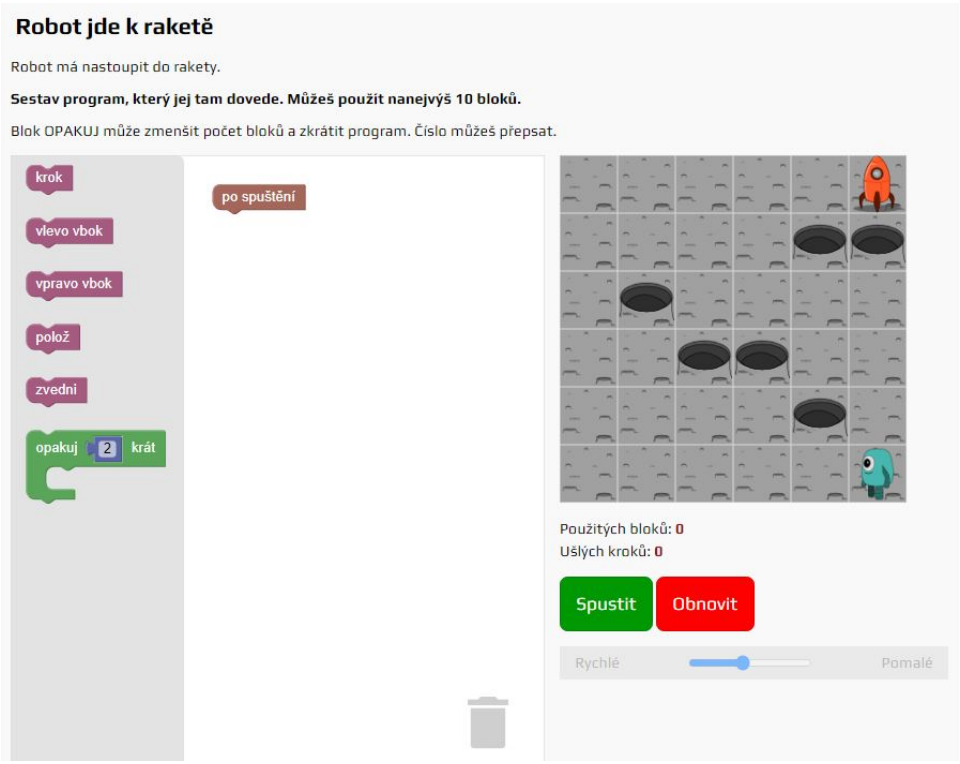
Tabulka 4: Nejpoužívanější úspěšná řešení otázky 1379

3.5.4 Úloha 1391

Poslední zkoumaná úloha již obsahuje o jeden blok více, s názvem opakuj. Další bloky uvnitř tohoto bloku se budou opakovat takovým počtem, který je v bloku zadaný. Tento blok do kódu přidá nejeden, ale čtyři další znaky. Jsou to O (opakuj), číslo, které označuje počet opakování a závorky, které ohraničují bloky, které budou uvnitř bloku iterovat. Tyto znaky mohou zefektivnit trénování neuronové sítě, protože je v kódu více parametrů, které mohou rozlišit úspěšný a neúspěšný kód.

V této úloze opět nedává smysl používat bloky zvedni a polož, protože pro splnění úlohy není třeba nic zvedat, nebo pokládat. Ale co je navíc odlišné od předchozích úloh je to, že má soutěžící jen omezený počet bloků, které může použít.

Z nejpoužívanějších kódu se zdá nejlepší třetí nejčastěji používaný kód. Ten obsahuje pouze čtyři bloky a neobsahuje žádnou část kódu, která by se opakovala. Z hlediska optimalizace algoritmů odpovídá nejvíc tento kód.



Obrázek 24: Úloha 1391

Kód	**Počet použití**	**Předpověď**
SO5(K)RO5(K)RO5(K)	8412	1.0
SO3(KKKKKR)	1033	0.9975202
SO3(O5(K)R)	595	0.9999998
SRO3(K)LO2(K)O2(RKK)	266	0.9348822
SRO3(K)LO3(KKR)	254	0.99993527
SRO3(K)LKKO2(RKK)	238	0.99994856
SRO3(K)LO2(KKR)KK	182	0.9536617
SRO3(K)LO2(KKR)O2(K)	122	0.55468744
SO5(K)RO5(K)RO5(K)P	114	0.9999987
SRO3(K)LO2(K)O2(RO2(K))	93	0.99999684

Tabulka 5: Nejpoužívanější úspěšná řešení otázky 1391

3.5.5 Předpovědi nejpoužívanějších kódů

U převážné vetšiny nejpoužívanějších úspěšných kódů je vidět vysoká pravděpodobnost na správné dokončení úlohy. Což je správně. Tato pravděpodobnost by měla být ideálně kolem 99 %. Problém je v tom, že jsou kódy, kterým neuronová síť předpovídá pravděpodobnost kolem padesáti procent, nebo dokonce jsou výjimečné případy, kdy se předpověď blíží jednomu procentu. Může to být tím, že jsou kódy moc podobné nesprávným řešením, které použilo více neúspěšných řešitelů a síť nerozpozná rozdíl mezi nimi.

Doopravdy lze jen těžko říct, čím to může být způsobeno. O neuronových sítích se říká, že jsou to „černé skříňky“, protože je velmi obtížné zjistit, co se v nich odehrává.[5] Kdyby se proces trénování neuronové sítě opakoval znovu, síť by předpovídala podobné výsledky, ale nebyly by úplně shodné. Zároveň by mohla předpovídat nízkou pravděpodobnost pro kód, kterému teď předpovídá vysokou a obráceně. Síť byla natrénována na 95 %, což se může zdát jako vysoká úspěšnost, ale pořád bude 5 % kódů, pro které bude odhad sítě nesprávný.

3.6 Ověření výsledků

Pro ověření výsledků jsem náhodně vybral několik kódů, které soutěžící sestrojili jako první. Ty jsem předložil neuronové síti a ta odhadla výsledek, který reprezentuje hodnota mezi nulou a jedničkou. Čím více se předpověď blíží jedné, tím si je síť jistější, že má uživatel šanci se s tímto kódem dostat ke správnému výsledku. Čím více se blíží k nule, tím spíše může být kód cestou k nesprávnému řešení. Pro ověření jsem přidal kolonku *Konečné řešení*, která znamená to, jestli posledním použitým kódem uživatel dospěl k úspěšnému řešení, nebo ne.

Otázka	**Kód**	**Předpověď**	**Konečné řešení**
1376	SLKRPKLKK	0.007179469	neúspěšné
1376	SLKKP	0.12332356	neúspěšné
1376	SKLKK	0.8595502	úspěšné
1376	SKLKKRKRKKLKKLKKRK	0.9991236	úspěšné
1378	SKZKKZ	3.7407128e-05	neúspěšné
1378	SKZK	1.2489797e-06	neúspěšné
1378	SKKKKLKKKK	0.49813163	úspěšné
1378	SKZLLKPRRKKPKPKLKPKK	0.9205437	úspěšné
1379	SLLKRKKRKKLKKKKL	0.99961525	úspěšné
1379	SKKKRKKRKKKKKR	0.9996289	úspěšné
1379	SKKKKZ	0.99931276	neúspěšné
1379	SKKKRKKRKKKKLKKK	0.9966332	úspěšné
1391	SO5(KR)	0.9999993	úspěšné
1391	SKL	9.9798395e-05	neúspěšné
1391	SRKKKLKK	0.012745351	neúspěšné
1391	SO5(K)RO5(K)RO5(K)	1.0	úspěšné

Tabulka 6: Tabulku s výsledky

Na posledním řádku je vidět nejpoužívanější kód pro otázku 1391. V tomto případě si je síť jistá na 100 %, že uživatel bude úspěšný, když jako první použil nejpopulárnější správné řešení.

Dále jsem do tabulky zahrnul i kód, kterému síť předpověděla 99.93 % na úspěšné splnění otázky 1379, přesto se uživatel nedostal ke správnému řešení. To je nejspíš způsobeno tím, že více uživatelů, kteří použili tento kód, bylo úspěšných při konečném řešení úlohy.

V tabulce je spousta kódů, u kterých si je síť jistá výsledkem, tudíž jsou hodnoty vysoké, a to kolem 99 %, nebo naopak nízké hodnoty, které jsou kolem jednoho procenta. Ale je zde zahrnutý i kód, u kterého si síť není jistá a předpovídá hodnotu kolem padesáti procent.

4 Závěr

Na mé práci je vidět, že vytvořit umělou inteligenci a zkoumat různé druhy neuronových sítí není v dnešní době nic náročného. A to hlavně díky současným knihovnám, na kterých lze v krátké době vyzkoušet mnoho variant a tím dosahovat stále lepších a lepších výsledků.

V teoretické části jsem popsal základní principy, které souvisejí s fungováním umělé inteligence, postupem jejího vývoje a využití již existujících knihoven. Nejvíce popsány jsou různé druhy vrstev neuronů, ze kterých se skládá neuronová síť.

V praktické části nejvíce popisuji proces tvorby umělé inteligence na konkrétním příkladu, kdy se síť naučí odhadnout výsledek z označených dat uživatelů, kteří se pokouší vyřešit algoritmickou úlohu. Po naučení byla síť schopna předpovědět, jestli kód může vést ke správnému řešení čtyř různých algoritmických úloh ze soutěže Bobřík informatiky se správností nad 95 % bude-li zadán kód, ze kterých síť byla naučená, nebo se správností necelých 70 % bude-li zadán kód, který se síť neučila.

V kapitole ověření výsledků jsou vidět konkrétní předpovědi neuronové sítě na první sestrojený kód soutěžících a informace o tom, jestli síť předpověděla výsledek správně.

V praktické části jsou také popsány další experimenty vývoje umělé inteligence z různých neuronových vrstev a jejich výsledky jsou srovnávány s hlavním řešením. Nejhorší experiment byl s 1D Konvolučními sítěmi. Na dosažených výsledcích nelze ani říct, jestli úvaha pro toto řešení byla správná. Řešení 2D konvolučních sítí a rekurentních sítí byla úspěšnější. Ale pořád tyto experimenty měly své chyby, kvůli kterým nedokázaly překonat řešení s hustě propojenými sítěmi.

Výsledkem práce je aplikace, která zpracuje surová data, které produkuje soutěž Bobřík informatiky, do přehledné podoby, která je snadno zpracovatelná databázovými systémy. Další produkt praktické části je umělá inteli-

gence, která je v podobě souboru ve formátu H5, který lze implementovat do dalších aplikací pomocí metody *keras.models.load_ model('cesta/k/modelu')*. A jeho kód, díky kterému je v budoucnu možné naučit síť na dalších datech, které mohou být z výsledků nových úloh bobříka informatiky.

Reference

[1] JAKLIN, Petr., *Neuronové sítě*[online], 2004, Dostupné z: https://cgg.mff.cuni.cz/ pepca/prg022/mucha/

[2] ŠÍMA, J., Neruda, R.: Teoretické otázky neuronových sítí, MATFY-ZPRESS, 1996, ISBN 80-85863- 18-9

[3] Volná, E. *Neuronové sítě 1.* Ostrava 2008, Dostupné z: https://web.osu.cz/Ṽolna/Neuronove_site_skripta.pdf

[4] FAUSETT, L. V. *Fundamentals of Neural Networks.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1994, ISBN 978-0133341867

[5] 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.

[6] MUELLER, John Paul a Luca MASSARON. *Data science programming all-in-one for dummies* Indianapolis: John Wiley, 2019. ISBN 978-1-119-62611-4.

[7] GOODFELLOWpůvodní, I., Bengio, Y., Courville, A., Deep Learning. MIT Press, 2006. Dostupné z: <https://www.deeplearningbook.org/

[8] KLUSÁČEK, David. *Umělá inteligence – od pravidel k neuronovým sítím*[online], 2019, Dostupné z: https://www.youtube.com/watch?v=2BBz7QQspyI

[9] CHOLLET, François. *Keras – Dropout layer*[online], 2015, Dostupné z: https://keras.io/api/layers/regularization_layers/dropout/

[10] Geeks for geeks,*CNN | Introduction to Pooling Layer*, 2021, Dostupné z: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer

[11] Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity, *Učení s učitelem*, Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat–umela-inteligence–neuronove-site-jednotlivy-neuron–adaptacni-dynamika-neuronu–uceni-s-ucitelem

[12] Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity, *Učení bez učitelem*, Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat–umela-inteligence–neuronove-site-jednotlivy-neuron–adaptacni-dynamika-neuronu–uceni-bez-ucitele

[13] Gustineli, Murilo. *A survey on recently proposed activation functions for Deep Learning*[online], 2022, Dostupné z: https://arxiv.org/abs/2204.02921

[14] Programmathically *An Introduction to Neural Network Loss Functions*[online], 2021, Dostupné z: https://programmathically.com/an-introduction-to-neural-network-loss-functions/

[15] Abdolrasol, Maher G. M., *Artificial Neural Networks Based Optimization Techniques*, 2021, Dostupné z: https://www.mdpi.com/2079-9292/10/21/2689

[16] ABADI, Martín. *TensorFlow: A System for Large-Scale Machine Learning*. USA: O'Reilly Media, 2016. ISBN 978-1-931971-33-1

[17] Frýbert, David. 7.5. 2008 *Robot Karel* Dostupné z: https://spomocnik.rvp.cz/clanek/11795/ROBOT-KAREL.html

[18] MARRS, Tom. *JSON at Work*. 1.vyd. USA: O'Reilly Media, 2017. 376 s. ISBN 978-1-449-35832-7.

[19] DYKES, Lucinda a Ed Tittel. *XML for dummies*. 4.vyd. Canada: Wiley Publishing, 2005. 376 s. ISBN 978-0-7645-8845-7.

[20] Edoceo, Inc., *CSV Standard File Format*[online], 2004, Dostupné z: http://www.edoceo.com/utilis/csv-file-format.php

[21] CHOLLET, François. *Keras – Conv2D layer*[online], 2015, Dostupné z: https://keras.io/api/layers/convolution_layers/convolution2d/

[22] CHOLLET, François. *Keras – MaxPooling2D layer*[online], 2015, Dostupné z: https://keras.io/api/layers/pooling_layers/max_pooling2d/

Seznam obrázků

Seznam tabulek

Seznam kódů

A Přílohy

CD s použitými kódy, aplikací Dataconverter, soubor s neuronovou sítí formátu h5 a textem práce ve formátu PDF.