



**FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI**

**KATEDRA
KYBERNETIKY**

Bakalářská práce

Neuronové sítě pro porozumění řeči

Vladimíra Kimlová



FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

Bakalářská práce

Neuronové sítě pro porozumění řeči

Vladimíra Kimlová

Vedoucí práce

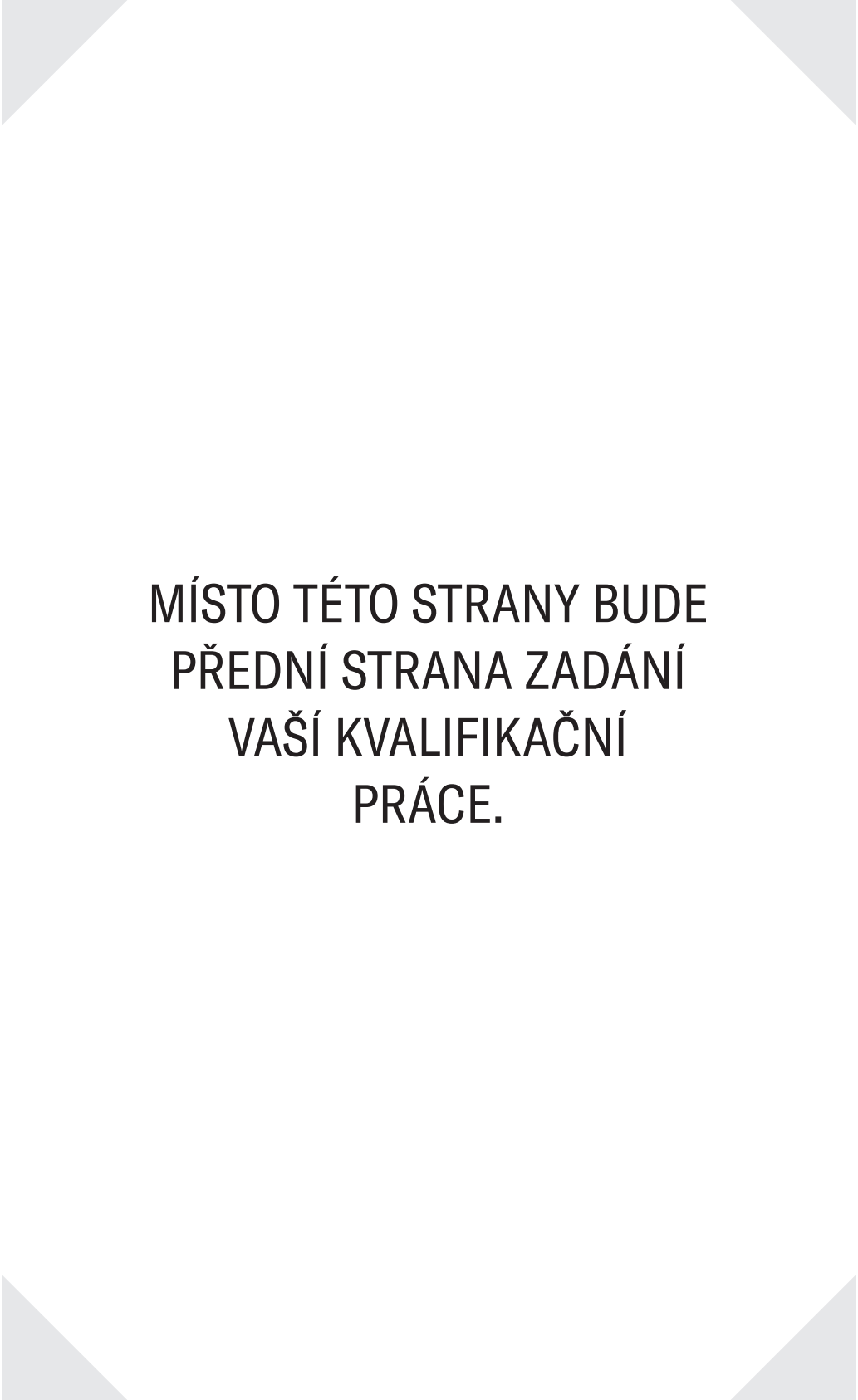
Ing. Bulín Martin, M.Sc.

© Vladimíra Kimlová, 2023.

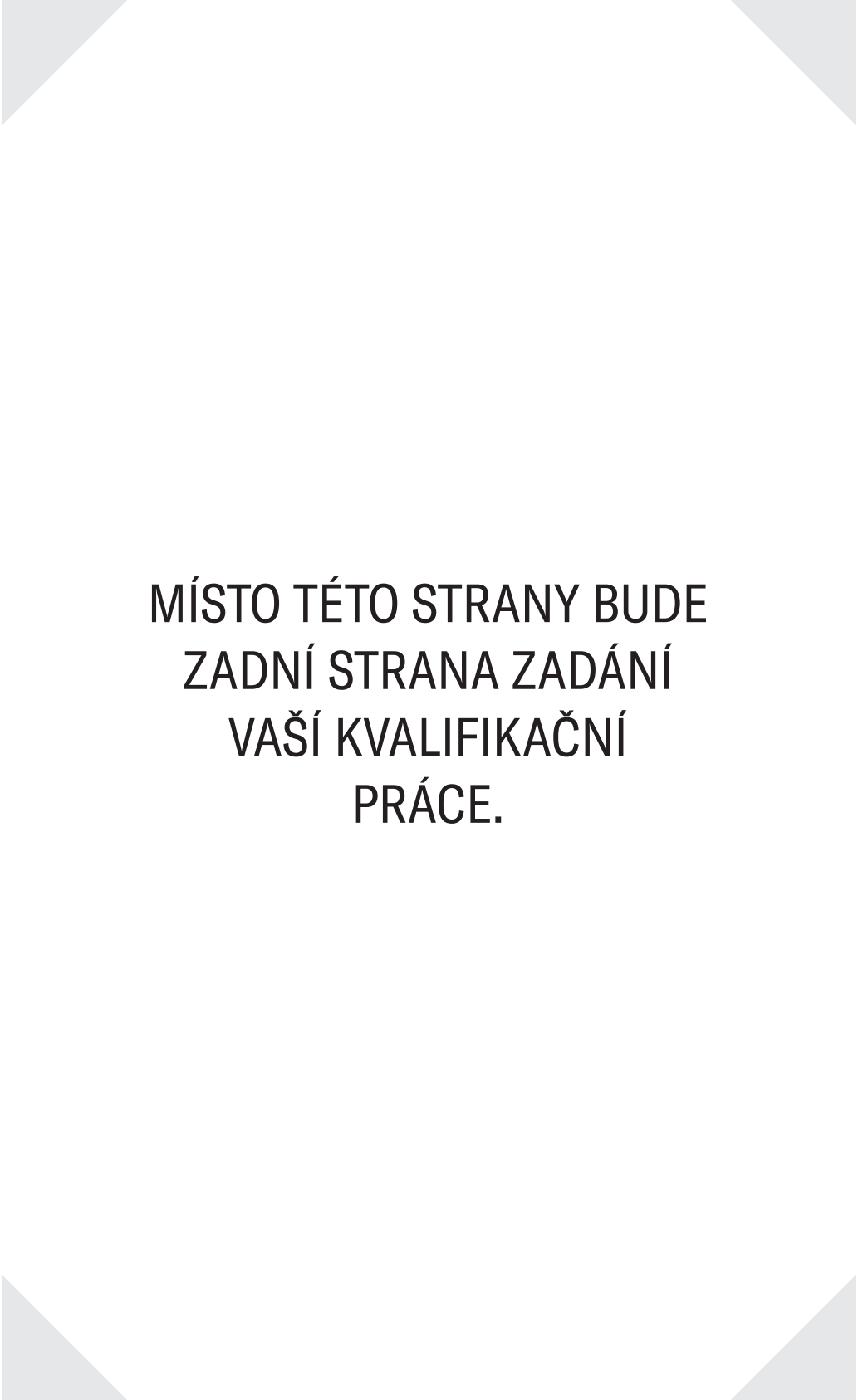
Všechna práva vyhrazena. Žádná část tohoto dokumentu nesmí být reprodukována ani rozšiřována jakoukoli formou, elektronicky či mechanicky, fotokopírováním, nahráváním nebo jiným způsobem, nebo uložena v systému pro ukládání a vyhledávání informací bez písemného souhlasu držitelů autorských práv.

Citace v seznamu literatury:

KIMLOVÁ, Vladimíra. *Neuronové sítě pro porozumění řeči*. Plzeň, 2023. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra kybernetiky. Vedoucí práce Ing. Bulín Martin, M.Sc.



MÍSTO TÉTO STRANY BUDE
PŘEDNÍ STRANA ZADÁNÍ
VAŠÍ KVALIFIKAČNÍ
PRÁCE.



MÍSTO TÉTO STRANY BUDE
ZADNÍ STRANA ZADÁNÍ
VAŠÍ KVALIFIKAČNÍ
PRÁCE.

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Plzni dne 22. května 2023

.....

Vladimíra Kimlová

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Neuronové sítě jsou dnes dominantním nástrojem pro mnoho úloh v oblasti strojového učení a umělé inteligence. Avšak pro dosažení vysoké přesnosti řešení dané úlohy je potřeba velké množství anotovaných dat pro trénování sítě, což může být časově náročné a drahé. Tato bakalářská práce se zaměřuje na využití lidské interakce při trénování neuronových sítí (tzv. *human-in-the-loop* přístup) s cílem umožnit neuronové síti se učit nové věci v reálném čase a rychle a flexibilně reagovat na nové situace. **Výsledná metoda umožňuje dotrénovat libovolné vzorky a třídy v reálném čase pomocí hlasového dialogu a Sentence Transformeru na neuronové síti pro klasifikaci záměru.** Síť pro klasifikaci záměru byla vytvořena pomocí transfer learningu pro efektivnější trénování a zlepšení schopnosti generalizace. Celý systém je navržen tak, aby byl snadno použitelný a mohl být nasazen na zařízení s omezenými výpočetními zdroji, jako je Raspberry Pi.

Abstract

Neural networks have become a dominant tool for many tasks in the field of machine learning and artificial intelligence. However, achieving high accuracy in solving a given task requires a large amount of annotated data for network training, which can be time-consuming and expensive. This bachelor's thesis focuses on utilizing human interaction in the training of neural networks (the human-in-the-loop approach) **with the aim of enabling** the network to learn new things in real-time and to quickly and flexibly respond to new situations. The resulting method allows for real-time fine-tuning of ~~any~~ samples and classes using a voice dialogue and Sentence Transformer on a neural network for intent classification. The intent classification network was created using transfer learning for more efficient training and improved generalization ability. The ~~entire~~ system is designed to be easily deployable and can be deployed on devices with limited computing resources, such as Raspberry Pi.

Klíčová slova

dopředná neuronová síť • Human-in-the-loop • Sentence-Transformer • Transfer learning • Fine-tuning • klasifikace záměru • Raspberry Pi

Poděkování

Nejprve bych chtěla poděkovat Ing. Martinu Bulínovi, M.Sc, svému vedoucímu práce, za jeho odborné rady, cenné nápady a ochotu věnovat mi svůj čas a energii. Bez jeho podpory by tato práce nebyla možná.

Ráda bych také poděkovala své rodině a přátelům za jejich nekonečnou podporu, povzbuzení a laskavá slova během mého studia.

Obsah

1	Úvod	1
1.1	Cíle práce	2
1.2	Současný stav problematiky	2
1.2.1	Human-in-the-loop	3
1.2.2	Klasifikace záměru	4
1.3	Struktura práce	4
2	Použité metody a technologie	7
2.1	Dopředná neuronová síť	7
2.1.1	Hyperparametry	12
2.2	Sentence Transformer	15
2.2.1	FERNET-C5	17
2.3	Fine-tuning a Transfer learning	17
2.4	ROS (Robot Operating System)	19
2.5	Speechcloud	20
2.6	Komunikační protokoly	22
2.6.1	HTTP	22
2.6.2	MQTT	22
2.6.3	WebSockets	23
2.7	Základní webové technologie	23
3	Učení neuronové sítě hlasovým dialogem	25
3.1	Neuronová síť pro klasifikaci záměru	25
3.2	Zpětnovazební hlasové učení	26
3.3	Uživatelské rozhraní	27
4	Data	31
4.1	Ručně vytvořený dataset	31
4.2	Robustní dataset pro klasifikaci záměrů	31

5	Experimenty a výsledky	33
5.1	Porovnání frameworků	33
5.2	Volba struktury sítě	35
5.3	Trénování v závislosti na počtu vzorků a tříd	38
5.4	Analýza použití předtrénované neuronové sítě	41
5.5	Vyhodnocení metody učení dialogem	44
6	Aplikace	47
6.1	Cílová robotická platforma	47
6.2	Učení konkrétních záměrů	50
7	Diskuze	51
7.1	Rekapitulace metod	51
7.2	Shrnutí výsledků	52
8	Závěr	55
8.1	Vylepšení do budoucna	55
A	Ručně vytvořený dataset	57
B	Testované hyperparametry pro neuronovou síť na RPi	59
C	Grafy k experimentu 5.4	61
D	Scénáře pro Experiment 5.5	63
D.1	Scénář 1	63
D.2	Scénář 2	64
	Bibliografie	65
	Seznam obrázků	69
	Seznam tabulek	71

V posledních letech se neuronové sítě staly dominantním nástrojem pro mnoho úloh v oblasti strojového učení a umělé inteligence. Nicméně k dosažení vysoké přesnosti při řešení dané úlohy je obvykle zapotřebí velkého množství anotovaných dat pro trénování neuronové sítě. Sběr a anotace dat mohou být časově náročné, drahé a mnohdy jsou data nedostupná, což může být pro některé aplikace překážkou. Trénink na velkém množství dat může být také drahý a zabere hodně času, což způsobuje problémy při iterativním vylepšování modelů, protože s rostoucím množstvím dat se stává stále obtížnější udržet přehled o tom, co se poslednímu modelu podařilo naučit. Tyto problémy jsou zvláště patrné v případech, kdy je potřeba rychle se adaptovat na nové podmínky a naučit se novému chování, jako například v oblasti *human-robot interaction* a v metodách učení přímo za běhu programu. Proto se v takových situacích dnes stále častěji používá přístup zvaný „*human in the loop*“, který umožňuje zapojení lidí do tréninkového procesu. Tento přístup umožňuje využívat odborné znalosti a schopnosti člověka pro rychlou a flexibilní reakci na nové situace a zvyšuje kvalitu výsledných modelů tím, že lidé poskytují cennou zpětnou vazbu a pomoc při anotaci dat pro přeučení modelu. Human-in-the-loop přístup se využívá v mnoha oblastech, včetně strojového učení, robotiky, medicíny, průmyslu, autonomních vozidel nebo i online reklamy a personalizace.

Tato práce nabízí metodu trénování, která umožňuje lidskou interakci s trénovacím algoritmem přímo za běhu programu (tzv. *human-in-the-loop* přístup [16]). Cílem je umožnit uživateli směřovat učení neuronové sítě pomocí zpětné vazby, což umožňuje neuronové síti učit se nové věci v reálném čase a rychleji se přizpůsobovat měnícím se podmínkám. Metoda je demonstrována na jednoduché dopředné neuronové síti používané pro klasifikaci záměrů. Dialog učení je realizován pomocí hlasového rozhraní a pro převod vstupních dat na vektory je používán Sentence Transformer [25]. V rámci experimentů byly analyzovány různé frameworky pro implementaci sítě, s cílem vybrat ten nejvhodnější pro zadanou úlohu. Dále byla zkoumána optimální velikost sítě pro daný problém a pro nasazení

na Raspberry Pi. Byl také analyzován vliv přeučení (Retrainingu) na výsledky klasifikace, včetně použití *transfer learningu* pro inicializaci sítě. Výsledná metoda využívá Sentence Transformer k modelování sémantiky vět a *transfer learningu* pro efektivnější trénování modelů a zlepšení schopnosti jejich generalizace. Dále je možné v reálném čase dotrénovat libovolné vzorky a třídy pomocí hlasového dialogu, což zlepšuje schopnost systému přizpůsobit se konkrétním potřebám uživatele. Celý systém je navržen tak, aby běžel na platformě s Raspberry Pi 4B, což umožňuje jeho efektivní nasazení v různých aplikacích.

V rámci práce bylo dále navrženo řešení jednoho ze záměrů pro vyhledání informace pomocí vyhledávače Google, respektive pomocí tzv. *Featured Snippet*, což je zvláštní zobrazení výsledků vyhledávání na Google, které prezentuje nejrelevantnější odpověď na konkrétní dotaz. Navíc bylo vytvořeno grafické uživatelské rozhraní (GUI) pro aplikaci na robotickou platformu.

1.1 Cíle práce

Cíle práce jsou následující:

1. Nastudovat problematiku klasifikace neuronovými sítěmi a seznámit se s knihovnami Keras a PyTorch.
2. Seznámit se s platformou Speechcloud a naučit se používat její služby pro rozpoznávání a syntézu řeči.
3. Navrhnout neuronovou síť pro klasifikaci záměru ze vstupní promluvy.
4. Vyvinout řešení pro generování odpovědí pro vybrané kategorie (záměry).
5. Prozkoumat možnosti automatického přeučení klasifikátoru na nových datech.
6. Aplikovat vyvinutou metodu včetně hlasové interakce na reálnou robotickou entitu.

1.2 Současný stav problematiky

Tato část se soustředí na dvě klíčová témata v oblasti strojového učení a umělé inteligence, kterými se tato práce primárně zabývá. První oblastí je *human-in-the-loop* přístup, který představuje klíčové téma této práce. Druhou oblastí je klasifikace záměru, což je ilustrativní úloha používaná k demonstraci *human-in-the-loop* přístupu.

1.2.1 Human-in-the-loop

V dnešní době existuje velká poptávka po řešeních založených na strojovém učení, protože pokroky, které se v posledních letech v této oblasti udály, ho popularizovaly a přiblížily širšímu publiku. Vytváření systémů strojového učení je však složitý proces, který vyžaduje hluboké znalosti technik strojového učení. V tradičním konceptu jsou učící algoritmy navrženy, vyvinuty a testovány, a pak nabízeny veřejnosti bez dalších změn. Tento monolitický přístup s sebou však nese důsledky, jako jsou obtížná rozšiřitelnost, staticnost, nemožnost správného vyhodnocení a snížení výkonu v důsledku změn kontextu. Byly proto definovány nové typy interakcí mezi lidmi a algoritmy strojového učení, které lze zařadit pod označení *human-in-the-loop*. Cílem není pouze zlepšit přesnost strojového učení nebo dosáhnout požadované přesnosti rychleji, ale také zvýšit účinnost a efektivitu zapojení lidí do tohoto procesu.

Vztah člověka a procesu strojového učení umožňuje rozdělit tuto problematiku do následujících kategorií [16]:

- *Aktivní učení (Active Learning)*: Tento proces využívá lidskou expertizu k anotaci dat, strojový model si sám vybírá, která data by měla být anotována člověkem, aby se zlepšila jeho schopnost učit se a získat přesnější výsledky.
- *Interaktivní strojové učení (Interactive machine learning)*: Interaktivní strojové učení zahrnuje bližší interakci mezi člověkem a učícím se systémem. Lidé aktivně přispívají k procesu učení poskytováním informací a zpětné vazby. Tento proces je častý, postupný a zaměřený na specifické úkoly, což vede k dosažení lepších výsledků než u tradičních metod strojového učení.
- *Machine teaching*: Při machine teachingu má lidský odborník kontrolu nad procesem učení a rozhoduje, jaké informace budou do strojového modelu přeneseny.
- *Curriculum learning*: Curriculum learning je metoda strojového učení, při které se postupně a systematicky předkládají příklady strojovému modelu tak, aby byl schopen lépe porozumět datovým vzorcům a zlepšit svou výkonost. Tento postup zahrnuje strukturované a logické načítání informací a příkladů, které jsou určeny odborníky v dané oblasti.

Obor věnující se analýze schopnosti modelů vysvětlit lidským uživatelům, jak bylo dosaženo určitého rozhodnutí nebo výsledku, se nazývá *Explainable Active Learning*. Cílem tohoto výzkumného oboru je zlepšit srozumitelnost výstupů, které jsou generovány AI systémy.

1.2.2 Klasifikace záměru

Klasifikace záměru (Intent classification) je oblast strojového učení, která se zaměřuje na určování úmyslů lidí z textových nebo hlasových dat. Tato technologie se používá v mnoha aplikacích, jako jsou chatboti, asistenti a automatizované telefonické systémy. V současné době se v této oblasti používají pokročilé techniky strojového učení, jako jsou například rekurentní neuronové sítě (RNN) nebo transformery, které dokážou efektivně zpracovávat dlouhé sekvence textových nebo hlasových dat.

Například v práci [24] byly mezi sebou porovnávány dopředné, rekurentní, GRU a LSTM neuronové sítě pro dataset ATIS [9]. Nejlepších výsledků (nejnižší chybovosti) při úkolu klasifikace textu dosáhly gated rekurentní sítě GRU a LSTM (1.30 %), které měly téměř srovnatelný výkon. Poté následovaly běžné rekurentní sítě (2.45 %) a nakonec dopředné sítě (3.60 %).

Porovnání řady různých modelů (např. LSTM, BLSTM, GRU, BGRU) bylo provedeno v práci [37]. Porovnání bylo provedeno pro datasety SMP¹ a RWC². Nejlepších výsledků dosáhla hybridní architektura konvoluční neuronové sítě a obousměrné gated rekurentní neuronové sítě (CNN-BGRU), 93 % přesnosti na datsetu SMP a 97 % na datsetu RWC.

Práce [15] navrhla hluboké hierarchické LSTM modely pro klasifikaci záměrů dialogu v oblasti elektronického obchodu (na datsetu obsahujícím 24760 vět), tyto dva modely zahrnují HLSTM a memory-augmented HLSTM. Výsledky experimentů ukazují, že tyto navržené modely mají lepší výkon (přesnost 81.6 % a 83.9 %) než základní přístupy použité jako srovnání (např. LSTM model měl přesnost 79.7 %).

Přesnost modelů na rámcových (*in-scope*) datech³ a datech mimo rámec (*out-of-scope*) byla testována v článku [13]. Na rámcových datech dosáhly největší přesnosti BERT (96.2 %) a MLP (93.4 %). U dat mimo rámec dosáhly největší přesnosti MLP (47.4 %) a Rasa (45.3 %). Nejhorší si v obou případech vedl FastText (89.0 % a 9.7 %).

1.3 Struktura práce

V úvodní kapitole je představena metoda trénování neuronové sítě, která umožňuje lidskou interakci s trénovacím algoritmem přímo za běhu programu. Dále jsou zde

¹<https://smp-challenge.com/dataset.html>

²<https://paperswithcode.com/dataset/rwc>

³Stejný dataset byl použit i v této práci, viz sekce 4.2.

popsány cíle práce a shrnut současný stav problematiky týkající se přístupu *human-in-the-loop* a klasifikace záměru.

Ve druhé kapitole jsou popsány použité metody a technologie. Nejprve je popsána dopředná neuronová síť a proces jejího učení, včetně nastavitelných hyperparametrů. Dále následuje popis Sentence transformerů, fine-tuningu a transfer learningu, Robot operating systému a Speechcloudu. Nakonec jsou zmíněny komunikační protokoly a základní webové technologie.

Třetí kapitola je věnována popisu navrženého řešení, tj. učení neuronové sítě hlasovým dialogem. Je představena neuronová síť pro klasifikaci záměru a její napojení na Sentence Transformer, dále algoritmus dialogu a přeučení neuronové sítě. Nakonec je ukázáno uživatelské rozhraní.

Čtvrtá kapitola se zabývá popisem použitých datasetů, tj. kolik mají vzorků a jak byly tyto vzorky rozděleny.

Pátá kapitola je o experimentech a jejich výsledcích. Experimenty se zabývaly porovnáním frameworků pro tvorbu neuronové sítě, dále volbou struktury sítě a také trénováním v závislosti na počtu vzorků a tříd. Dále byla provedena analýza použití předtrénované neuronové sítě a vyhodnocení metody učení dialogem.

Šestá kapitola je věnována konkrétním aplikacím této práce, je zde popsána cílová robotická platforma a konkrétní učené záměry.

Sedmá kapitola se zabývá diskuzí výsledků dosažených v experimentech.

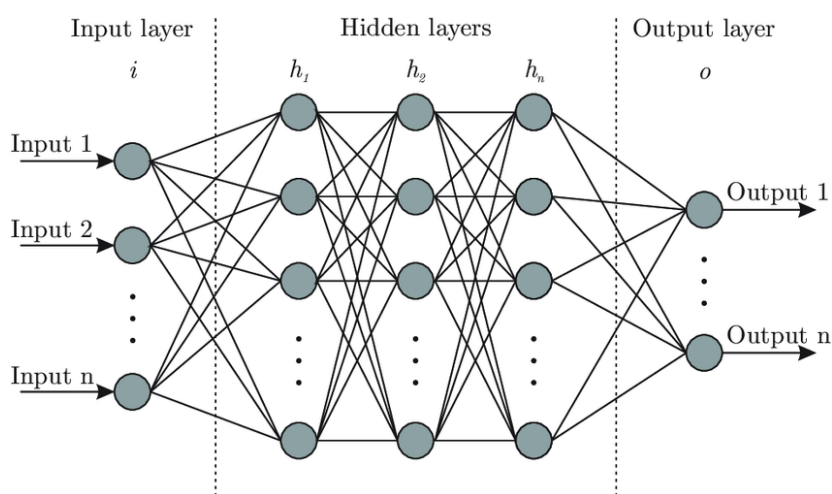
V závěrečné kapitole jsou shrnuty dosažené výsledky v rámci této práce. A jsou navržena další vylepšení do budoucna.

Použité metody a technologie

2

2.1 Dopředná neuronová síť

Dopředná neuronová síť (Feedforward Neural Network) je typ umělé neuronové sítě, která se skládá z několika vrstev neuronů. První vrstva se nazývá vstupní, poslední výstupní a vrstvy mezi nimi jsou označovány jako skryté. Ve většině případů je každý neuron spojen se všemi neurony v následující vrstvě. Architektura je nastíněna na Obrázku 2.1.



Obrázek 2.1: Diagram architektury neuronové sítě (převzato z [6])

V dopředné neuronové síti je signál šířen pouze ve směru od vstupní vrstvy přes skryté vrstvy až k výstupní vrstvě. Matematicky lze dopřednou neuronovou síť popsat jako zobrazení vstupního signálu na výstupní signál. Obecně platí, že velikost vstupní vrstvy je dána dimenzí vstupních dat a velikost výstupní vrstvy je za předpokladu klasifikačního problému určena počtem tříd.

Nechť má neuronová síť L vrstev, kde $l = 1, 2, \dots, L$ označuje vrstvu a $n^{(l)}$ označuje

počet neuronů v l -té vrstvě. Na vstup sítě je přiveden vektor vstupních dat x . Aktivace neuronů ve vrstvě l jsou označeny jako vektor $a^{(l)}$ [18].

Každý neuron ve vrstvě l zpracovává vstup $a^{(l-1)}$ z předchozí vrstvy sítě pomocí lineární transformace:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \quad (2.1)$$

kde $W^{(l)}$ je matice vah neuronů mezi $l - 1$ a l vrstvou, $b^{(l)}$ je bias vektor pro vrstvu l a $a^{(0)} = x$ [18].

Následně se na lineární transformaci aplikuje nelineární aktivační funkce f (obecně může být různá v každé vrstvě sítě). Výsledná rovnice aktivace neuronů v dané vrstvě vypadá následovně:

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) \quad (2.2)$$

Tento proces se opakuje pro každou vrstvu a výstup poslední vrstvy neuronů je výstupem celé sítě.

Nejčastěji používanými aktivačními funkcemi jsou ReLU (Rectified Linear Unit), sigmoid, tanh a softmax.

ReLU je definována jako [29]:

$$f_1(z) = \max(0, z) \quad (2.3)$$

Sigmoidní aktivační funkce má tvar [29]:

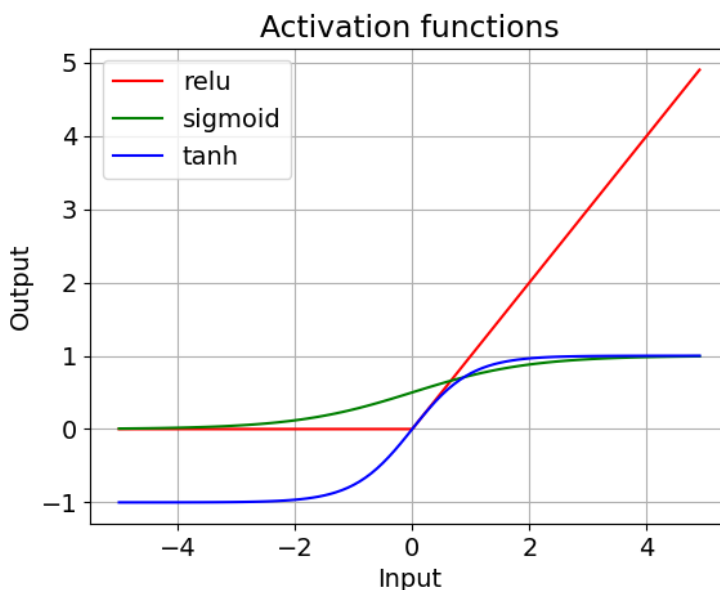
$$f_2(z) = \frac{1}{1 + e^{-z}} \quad (2.4)$$

Tanh aktivační funkce má tvar [29]:

$$f_3(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.5)$$

Funkce Softmax může být na rozdíl od sigmoidních funkcí, které se používají pro binární klasifikaci, použita pro problémy s klasifikací více tříd. Pokud má výstupní vrstva K neuronů, potom má Softmax funkce tvar [29]:

$$\mathbf{a}^{(L)}_j = \frac{e^{\mathbf{z}^{(L)}_j}}{\sum_{k=1}^K e^{\mathbf{z}^{(L)}_k}} \quad (2.6)$$



Obrázek 2.2: Aktivační funkce

Aktivační funkce ReLU, Sigmoid a Tanh jsou k vidění na Obrázku 2.2.

Cílem trénování neuronové sítě je nalézt takové hodnoty parametrů (váh synapsí \mathbf{W} a biasů neuronů \mathbf{b}), aby síť byla schopna klasifikace nebo regrese vstupních dat s co největší správností. K porovnání cílové a předpovídané výstupní hodnoty neuronové sítě slouží tzv. ztrátová funkce (Loss Function), v rámci učení neuronové sítě je úkolem hodnotu této funkce minimalizovat.

Backpropagation je algoritmus pro učení s učitelem umělých neuronových sítí pomocí gradientního sestupu. Je využíván k výpočtu gradientu chybové funkce vzhledem k vahám umělé neuronové sítě, což umožňuje adaptovat váhy sítě tak, aby byla minimalizována chyba výstupu sítě. Tento postup umožňuje sítím adaptovat se na nová data a provádět přesné predikce. Algoritmus byl představen již v 60. letech, ale až téměř 30 let poté (v roce 1989) se stal populárním díky práci Rumelharta, Hintona a Williamse nazvané „Learning representations by back-propagating errors“ [27].

Algoritmus se skládá ze tří hlavních částí [36]:

- Dopředného šíření vstupního signálu
- Zpětného šíření chyby
- Aktualizace váhových hodnot neuronů

Tyto kroky jsou cyklicky opakovány, dokud není dosaženo dostatečně malé chyby sítě, mezního počtu iterací nebo jiného kritéria pro ukončení učení. Následuje podrobnější popis jednotlivých kroků.

Dopředné šíření (Forward Propagation):

Rovnice dopředného šíření odpovídá kombinaci Rovnic 2.1 a 2.2:

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \quad (2.7)$$

Zpětné šíření (Backpropagation):

C představuje tzv. ztrátovou funkci (Loss Function), označovanou také jako Cost function. Ztrátová funkce C může mít například tvar kvadratické funkce [18]:

$$C = \frac{1}{2} \sum_j (y_j - a_j^L)^2 \quad (2.8)$$

kde y_j je požadovaný výstup neuronu j a a_j^L je vypočtený výstup tohoto neuronu, L představuje počet vrstev v síti.

Obecně lze rozdělení ztrátových funkcí provést podle toho, zda se používají pro klasifikační nebo regresní problémy.

Klasifikační problémy mají za cíl predikovat diskrétní třídy nebo kategorie. Mezi ztrátové funkce vhodné pro klasifikační problémy patří například:

- *Categorical Cross-Entropy*: Tato funkce se používá pro klasifikační úlohy s více třídami. Její vzorec vypadá následovně [38]:

$$C_1 = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \log(p_{ij}) \quad (2.9)$$

kde N je počet datových vzorků, M je počet tříd, y_{ij} je indikátor, zda je i -tý vzorek ve třídě j , a p_{ij} je pravděpodobnost, že i -tý vzorek patří do třídy j .

- *Binary Cross-Entropy*: Jedná se o speciální případ Categorical Cross-Entropy pro úlohy, kde je počet tříd omezen na dvě. Lze ji popsat takto [38]:

$$C_2 = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (2.10)$$

kde N je počet datových vzorků, y_i označuje skutečnou třídu vzorku i , která může nabývat hodnoty 0 nebo 1, p_i je pravděpodobnost, že i -tý vzorek patří do třídy 1 (pravděpodobnost, že vzorek patří do třídy 0, je rovna $1 - p_i$).

Regresní problémy mají za cíl predikovat spojité hodnoty. Mezi ztrátové funkce vhodné pro regresní problémy patří například:

- *Mean Squared Error (MSE)*: Funkce je citlivá na outliery. Používá se, pokud jsou cílová data normálně rozdělena kolem střední hodnoty, a když je důležité více penalizovat odlehlé hodnoty. Vypadá následovně [10]:

$$C_3 = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.11)$$

kde N je počet datových vzorků, y_i je skutečná hodnota a \hat{y}_i je predikovaná hodnota.

- *Mean Absolute Error (MAE)*: Tato funkce není citlivá na outliery. Nevýhodou MAE je, že velikost gradientu není závislá na velikosti chyby, pouze na znaménku $(y_i - \hat{y}_i)$, což může vést k problémům s konvergencí, protože velikost gradientu bude velká i při malé chybě. MAE vzorec je [10]:

$$C_4 = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (2.12)$$

kde N je počet datových vzorků, y_i je skutečná hodnota a \hat{y}_i je predikovaná hodnota.

Zpětné šíření se zaměřuje na to, jak změna vah a biasů v síti ovlivňuje hodnotu ztrátové funkce. Dále jsou tedy uvedeny parciální derivace C vzhledem k váhám a biasům sítě. Pro neuron j ve vrstvě l jsou tyto derivace následující [18]:

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \quad (2.13)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.14)$$

kde δ_j^l je tzv. chyba neuronu j ve vrstvě l , která se vypočítá jako [18]:

$$\delta_j^l = f'(z_j^l) \sum_k w_{jk}^{l+1} \delta_k^{l+1} \quad (2.15)$$

kde $f'(\cdot)$ je derivace aktivační funkce neuronu j ve vrstvě l , w_{jk}^{l+1} je váha mezi neuronem j ve vrstvě l a neuronem k ve vrstvě $l + 1$ a z_j^l je vážená suma vstupů neuronu j ve vrstvě l [18]:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \quad (2.16)$$

Aktualizace vah a biasů:

Po výpočtu gradientu chyby se váhy a biasy sítě aktualizují pomocí gradientního sestupu. Nová hodnota váhy w_{jk}^l a biasu b_j^l je vypočtena jako [40]:

$$w_{jk}^l \leftarrow w_{jk}^l - \eta \frac{\partial C}{\partial w_{jk}^l} \quad (2.17)$$

$$b_j^l \leftarrow b_j^l - \eta \frac{\partial C}{\partial b_j^l} \quad (2.18)$$

kde η je koeficient učení (Learning Rate), což je nastavitelný hyperparametr optimalizačního algoritmu, který určuje velikost kroku, kterým se algoritmus posouvá směrem k minimu ztrátové funkce během trénování [17]. Obecně platí, že příliš nízká hodnota η může vést k pomalé konvergenci a příliš vysoká hodnota může vést k oscilaci a neefektivnímu trénování modelu.

Tyto kroky jsou cyklicky opakovány, dokud není dosaženo dostatečně malé chyby sítě, mezního počtu iterací nebo jiného kritéria pro ukončení učení [36].

2.1.1 Hyperparametry

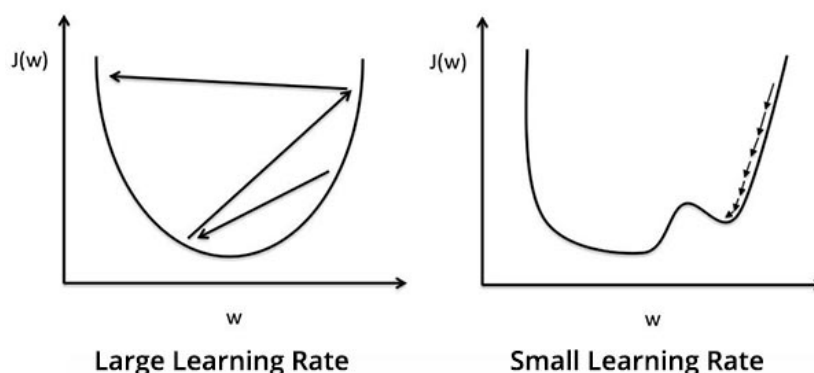
Hyperparametry neuronových sítí jsou proměnné, které určují strukturu sítě a způsob, jakým je síť trénována. Tyto hyperparametry jsou nastaveny před trénováním sítě, tedy před optimalizací vah a biasu [23].

Hyperparametrické ladění je proces hledání optimálních hyperparametrů pro danou úlohu (např. maximalizace přesnosti modelu na validačním datasetu). Tento proces

je často prováděn pomocí opakovaného trénování modelu s různými hodnotami hyperparametrů a vyhodnocováním výsledků pomocí validační sady dat.

Následuje výběr nastavitelných hyperparametrů:

- *Počet skrytých vrstev*: Počet skrytých vrstev představuje počet vrstev mezi vstupní a výstupní vrstvou. Vyšší počet vrstev může způsobit delší čas trénování.
- *Počet neuronů ve skrytých vrstvách*: Jedná se o počet neuronů pro každou skrytou vrstvu, může být obecně různý pro jednotlivé vrstvy. Počet neuronů by měl být přizpůsoben komplexitě řešené úlohy [26]. Zvýšení počtu neuronů může zlepšit výkon sítě, ale zároveň zvyšuje náročnost trénování sítě.
- *Aktivační funkce*: Aktivační funkce je funkce, která je aplikována na výstup každého neuronu v neuronové síti. Slouží k zavedení nelinearity do modelů [23].
- *Koeficient učení (Learning Rate)*: Koeficient učení definuje rychlost aktualizace parametrů neuronové sítě během trénování. Nižší hodnota zpomaluje proces učení, ale zpravidla vede k hladší konvergenci. Naopak vyšší hodnota zrychluje učení, ale může mít za následek, že síť se nebude schopna dostat k optimálnímu řešení a bude oscilovat kolem něj [23]. Na Obrázku 2.3 je ilustrován průběh ztrátové funkce J v závislosti na hodnotě vah w vlevo pro velký, vpravo pro malý koeficient učení. Navíc je vlevo zobrazena i možnost uvíznutí v lokálním minimu při malém učícím koeficientu.



Obrázek 2.3: Vliv učícího koeficientu při gradientním sestupu (převzato z [2])

- *Momentum*: Tato metoda pomáhá řešit výše zmíněný problém uvíznutí v lokálním minimu. Zahrnuje informace o minulých změnách parametrů při

aktualizaci nových. Rovnice 2.17 a 2.18 s využitím této metody vypadají následovně [32]:

$$v_{jk}^l \leftarrow \mu v_{jk}^l - \eta \frac{\partial C}{\partial w_{jk}^l} \quad (2.19)$$

$$w_{jk}^l \leftarrow w_{jk}^l + v_{jk}^l \quad (2.20)$$

$$v_j^l \leftarrow \mu v_j^l - \eta \frac{\partial C}{\partial b_j^l} \quad (2.21)$$

$$b_j^l \leftarrow b_j^l + v_j^l \quad (2.22)$$

kde v_{jk}^l a v_j^l jsou momenty pro váhy a biasy, μ je momentový koeficient (typicky se používají hodnoty 0.9 nebo 0.99 [4]). Výhodou této metody je rychlejší konvergence k optimálnímu řešení a menší pravděpodobnost uvíznutí v lokálních minimech.

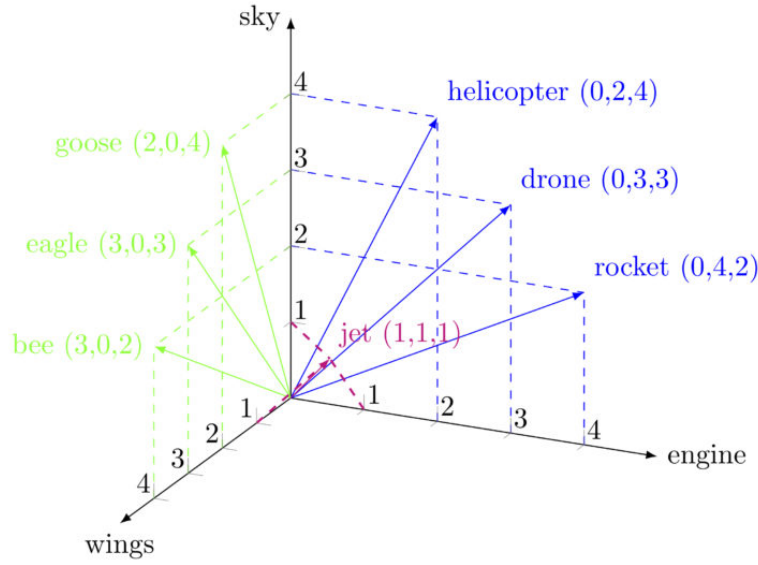
- *Batch Size*: Batch size je počet vzorků, které jsou síti předkládány současně v rámci jedné iterace aktualizace parametrů [23]. Batch size ovlivňuje rychlost trénování sítě, větší batch size obvykle může zvýšit rychlost trénování.
- *Počet epoch*: Počet epoch značí, kolikrát je celý trénovací dataset předložen neuronové síti během tréninku [23]. Příliš málo epoch může vést k nedotrénování sítě, zatímco příliš mnoho epoch může vést k přeučení sítě [26].
- *Optimalizační algoritmus*: Optimizér je algoritmus používaný k úpravě vlastností neuronové sítě, jako jsou váhy a koeficient učení, s cílem snížit chybovost [8]. Existuje mnoho druhů optimizérů v oblasti strojového učení, některé z nejznámějších jsou:
 - *Gradient descent*: Jedná se o nejprimitivnější typ optimizéru. Jeho výhodou je jednoduchost, rychlost a snadná implementace. Na druhou stranu ale může uváznout v lokálním minimu, může být pomalý, pokud je dataset velký a má mnoho příznaků, a má vysoké paměťové nároky [8].
 - *Stochastic gradient descent (SGD)*: Jedná se o variantu gradientního sestupu, která se snaží o častější aktualizaci parametrů. Mezi jeho výhody patří právě častá aktualizace parametrů a tím rychlejší konvergence, dále má nižší paměťové nároky. Nevýhodou je vysoká variabilita parametrů modelu a nutnost snižování učícího koeficientu pro dosažení stejné konvergence jako u gradientního sestupu. Algoritmus může být navíc nestabilní [8].

- *Adagrad*: Tento optimizér, narozdíl od všech výše zmíněných, mění koeficient učení pro každý parametr a v každém časovém kroku, odpadá tedy nutnost ručního ladění. Tento algoritmus je vhodný pro datasety s chybějícími vzorky. Je však výpočetně náročný a snižující se koeficient učení má za následek pomalé trénování [8].
- *Adadelat*: Jendá se o rozšíření metody Adagrad. Adadelat omezuje počet předchozích gradientů, které jsou započítávány, na pevnou velikost za pomoci klouzavého exponenciálního průměru a tím odstraňuje problém spojený se snižujícím se učícím koeficientem. Nevýhodou je opět výpočetní náročnost [8].
- *Root Mean Square Propagation (RMS-Prop)*: RMS-Prop je podobný Adagrad, rozdíl spočívá v použití exponenciálně klesajícího průměru místo součtu gradientů. V podstatě tedy kombinuje Momentum s AdaGradem. Kromě toho místo použití všech gradientů pro výpočet momenta, zahrnuje pouze nejnovější gradienty. To modelu umožňuje adaptaci koeficientu učení vzhledem k aktuální situaci. Nevýhodou je, že učení je pomalé [28].
- *Adaptive Moment Estimation (Adam)*: Adam optimizer lze považovat za kombinaci AdaGrad a RMS-Prop. Na rozdíl od RMSProp používá Adam průměr druhých momentů gradientů. Jedná se o nejvíce používaný optimalizační algoritmus pro řešení široké škály problémů, je vhodný pro velké datasety a je výpočetně efektivní. Výkon algoritmu Adam závisí na typu poskytnutých dat a jedná se o kompromis mezi rychlostí a generalizací [28].

2.2 Sentence Transformer

Sentence Transformer je model strojového učení, který dokáže transformovat vstupní textové sekvence (např. věty) do vektorů reprezentujících sémantickou informaci. Tyto vektory pak mohou být využity například pro kategorizaci, porovnávání podobnosti nebo další úlohy zpracování přirozeného jazyka. Ilustrační vektorový prostor sedmi slov je na Obrázku 2.4.

Architektura zvaná transformer byla poprvé představena v roce 2017 v článku *Attention Is All You Need* [35]. Tato architektura od svého nástupu překonávala do té doby tradiční rekurentní a konvoluční neuronové sítě postupně prakticky ve všech oblastech strojového učení. Modely dosahují lepší kvality, jsou více paralelizovatelné a vyžadují podstatně méně času na trénování. Klíčovou součástí



Obrázek 2.4: Vektorový prostor sedmi slov ve třech kontextech (převzato z [5])

transformeru je *attention* mechanismus, který umožňuje modelu se zaměřit na důležité části vstupní sekvence a ignorovat méně významné informace.

Vzorec pro výpočet *Scaled Dot-Product Attention* vypadá následovně [35]:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V}$$

kde:

- $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$ je matice dotazů (Query Matrix)
- $\mathbf{K} \in \mathbb{R}^{m \times d_k}$ je matice klíčů (Key Matrix)
- $\mathbf{V} \in \mathbb{R}^{m \times d_v}$ je matice hodnot (Value Matrix)
- d_k je počet dimenzí vektoru klíče (Key Vector)
- n a m jsou počty dotazů a klíčů/hodnot
- $\sqrt{d_k}$ slouží k normalizaci skalárního součinu a zlepšuje stabilitu gradientů v tré-nování

Multi-head attention lze chápat jako několik paralelních attention mechanismů pracujících společně. Místo toho, aby byl použit jeden set vah pro výpočet

attention, je použito více setů vah, tzv. *heads*. Každý head slouží jako zvláštní lineární transformace na vstupním prostoru a výsledky jsou následně kombinovány [3].


Hlavním cílem Sentence Transformeru je naučit se reprezentovat věty tak, aby výsledné vektory zachovávaly sémantickou podobnost mezi větami. To umožňuje využití těchto vektorů pro různé úlohy, jako například hledání podobností mezi větami, kategorizaci textů, vyhledávání odpovědí atd. Pro výpočet vektorové reprezentace vět se často používají předtrénované modely, jako jsou BERT [7] nebo GPT [22], které jsou široce dostupné pro použití v různých aplikacích.

2.2.1 FERNET-C5

FERNET-C5 [14] je jednojazyčný BERT model, který byl od úplného počátku trénován na datech českého korpusu Colossal Clean Crawled Corpus (C5) – obdoba anglického datasetu C4 pro češtinu. Trénovací data obsahují téměř 13 miliard slov. Model má stejnou architekturu jako původní BERT model, tedy 12 transformačních bloků, 12 *Attention Heads* a skrytou velikost 768 neuronů. Na rozdíl od BERT modelů od Googlu byla použita tokenizace SentencePiece místo interní tokenizace WordPiece od Googlu.

Model je veřejně k dispozici online¹ a umožňuje integraci s různými aplikacemi, jako jsou chatboty, analýza sentimentu, nebo klasifikace textu.

2.3 Fine-tuning a Transfer learning

Fine-tuning a *transfer learning* neuronových sítí jsou  metody strojového učení, které umožňují přizpůsobení již natrénované neuronové sítě na nový úkol, který může být odlišný od původního úkolu, pro který byla síť trénována.

Transfer learning je metoda, která využívá model naučený na jednom datasetu (zdrojový dataset) k řešení úloh na jiném datasetu (cílový dataset). Nejprve je model naučen na zdrojovém datasetu, a poté jsou váhy modelu použity jako výchozí nastavení pro trénování na cílovém datasetu. Tímto způsobem se model naučí rozpoznávat obecné vzory v datech a může být použit na řešení úloh, které mají podobné charakteristiky jako zdrojový dataset.

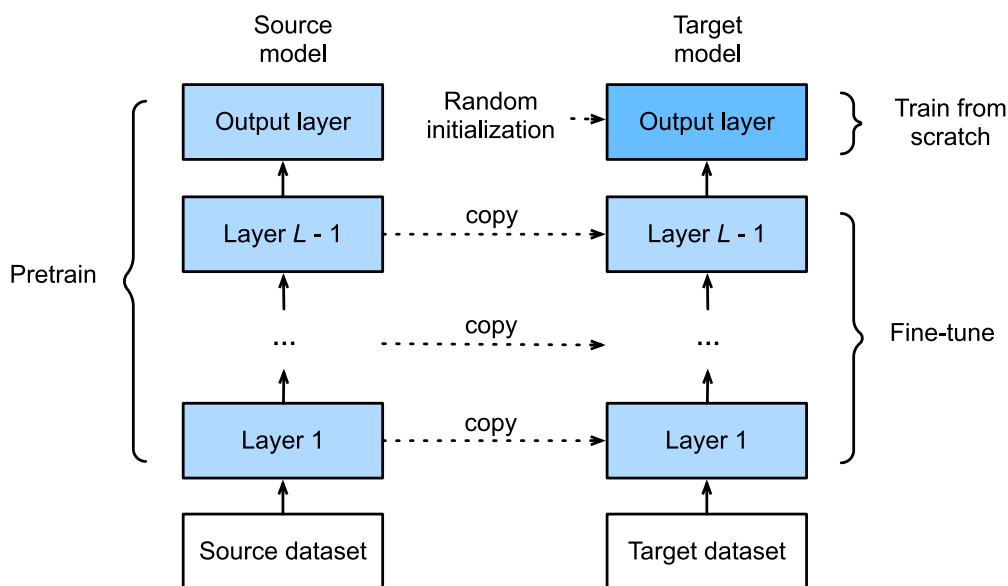
Fine-tuning je metoda, která využívá předtrénovaný model jako výchozí bod pro další trénování na novém datasetu. V případě fine-tuningu je obvykle použit model, který byl trénován na velkém datasetu, a ten je následně použit k řešení úloh na

¹<https://huggingface.co/fav-kky/FERNET-C5>

menším datasetu s podobnou problematikou. Cílem fine-tuningu je optimalizovat váhy modelu tak, aby se co nejlépe přizpůsobily specifickým charakteristikám cílového datasetu a zlepšily výsledky modelu na této úloze.

Celý proces začíná kopírováním (transferem) vah z již předtrénované sítě do nové sítě, která má být trénována. V případě klasifikační úlohy je často výjimkou poslední plně propojená vrstva, jejíž počet neuronů závisí na počtu tříd v původním datasetu. Běžnou praxí je nahradit poslední plně propojenou vrstvu předtrénovaného neuronového modelu novou plně propojenou vrstvou, která má stejný počet neuronů jako počet tříd v nové cílové aplikaci [34]. Její inicializace je zpravidla náhodná [39]. Nakonec je nový model trénován na cílovém datasetu. Výstupní vrstva je trénována od začátku, zatímco parametry všech ostatních vrstev jsou vyladěny na základě parametrů zdrojového modelu [39]. Obecně lze provádět i fine-tuning čistě T5 transformer sítě, kde se zachovává kompletní architektura a jen se provede fine-tuning na nových doménových datech k naučení nové úlohy. Alternativně mohou být některé přenesené váhy zmrazeny. Zmrazení vah znamená, že se během tréninku nové sítě váhy přenesené z původní sítě neaktualizují.

Nástin fungování těchto metod při nahrazení poslední plně propojené vrstvy je znázorněn na Obrázku 2.5.



Obrázek 2.5: Diagram transfer learningu a fine-tuningu (převzato z [39])

Fine-tuning a transfer learning jsou velmi užitečné nástroje, protože umožňují využít existující znalosti a zkušenosti z předtrénovaných modelů, které mohou být

velmi složité a nákladné na trénování, a aplikovat je na nové úlohy s menším počtem trénovacích dat. To může být velmi užitečné, například pokud není k dispozici dostatečné množství dat pro natrénování nového modelu od začátku.

2.4 ROS (Robot Operating System)

ROS (Robot Operating System) je open-source softwarový framework určený pro vývoj robotických aplikací. Poprvé byl uveden v roce 2007 na Stanfordově univerzitě. Poskytuje knihovny a nástroje umožňující snadno vytvářet a sdílet software pro řízení robotů [1, 20].

ROS byl vyvíjen veřejně s použitím permissivní licence BSD a postupně se stal široce používaným v komunitě výzkumníků robotiky. Na rozdíl od klasického přístupu, kdy všichni přispěvatelé umístí svůj kód na stejné servery, byl ROS vyvíjen v několika institucích a pro různé druhy robotů. Toto se stalo jednou z největších výhod ROS ekosystému. Současně je používán desetitisíci uživateli po celém světě v oblastech od domácích projektů v rámci koníčků, po velké průmyslové automatizované systémy [20]. V současné době je ROS stále aktivně vyvíjen a jeho verze 2.0, známá jako ROS 2, je určena pro použití v průmyslových aplikacích s většími nároky na spolehlivost a bezpečnost [12].

Základními koncepty implementace ROS jsou uzly (Nodes), zprávy (Messages), témata (Topics) a služby (Services). Uzly jsou procesy provádějící výpočty, přičemž systém obvykle sestává z mnoha uzlů. Termín uzel je v tomto kontextu zaměnitelný se softwarovým modulem. Použití termínu uzel vzniká z vizualizací systémů pomocí grafu, kde jsou procesy zobrazeny jako uzly grafu a peer-to-peer spojení jako orientované hrany (Arcs) [20, 21].

Uzly mezi sebou komunikují předáváním zpráv. Zpráva je striktně typovaná datová struktura. Podporovány jsou jak standardní primitivní datové typy (celé číslo, desetinné číslo, boolean, atd.), tak pole primitivních datových typů. Zprávy mohou být složeny z jiných zpráv a polí jiných zpráv, v libovolné hloubce vnoření [20, 21].

Uzel odesílá zprávu publikováním na dané téma. Uzel, který má zájem o určitý druh dat, se přihlásí k odběru příslušného téma. Pro jedno téma může existovat několik souběžných vydavatelů a odběratelů a jeden uzel může publikovat a/nebo odebírat více témat. Vydavatelé a odběratelé většinou nevědí o existenci druhého. Idea publikování a odběru témat v ROS umožňuje komunikaci mezi moduly na velmi flexibilní úrovni. Broadcast směřování však není vhodné pro synchronní transakce. Proto ROS poskytuje tzv. *service* (službu), která je definována názvem a dvěma přísně typovanými zprávami: jednou pro požadavek a druhou pro odpověď [20, 21].

Servisy ale nejsou vhodné pro úlohy, které mohou trvat dlouho, nebo pro situace, kdy je potřeba sledovat stav celého procesu. K tomuto účelu slouží balíček *actionlib*, který umožňuje uživateli během vykonávání požadavku jeho zrušení nebo získání zpětné vazby o postupu vykonávání požadavku. Actionlib poskytuje nástroje pro vytváření serverů vykonávajících dlouhodobé úkoly, které mohou být přerušeny. Kromě toho poskytuje klientské rozhraní pro odesílání požadavků na server [31].

Filozofické cíle ROS lze shrnout následovně:

- *Peer-to-peer*: ROS by měl být navržen tak, aby jednotlivé uzly mohly komunikovat přímo mezi sebou, aniž by bylo nutné centralizované řízení, tj. aby byl decentralizovaný [20, 21].
- *Založení na nástrojích (Tools-based)*: ROS poskytuje mnoho nástrojů, jako jsou nástroje pro získávání a nastavování konfiguračních parametrů, vizualizaci topologie připojení peer-to-peer, měření využití šířky pásma, grafické vykreslování data zpráv, automatické generování dokumentace atd. Avšak nemá kanonické integrované vývojové a runtime prostředí, všechny úkoly jsou prováděny samostatnými programy, což podporuje vytváření nových, vylepšených implementací [20, 21].
- *Mnohojazyčnost (Multi-lingual)*: Softwarové moduly ROS lze psát v jakémkoli programovacím jazyce, pro který byla napsána klientská knihovna. Tyto moduly mezi sebou komunikují díky jazykově neutrálnímu a jednoduchému jazyku pro definici rozhraní (IDL), které slouží k popisu zpráv odesílaných mezi moduly [20, 21].
- *Thin*: Konvence ROS vybízí přispěvatele k vytváření samostatných knihoven. Tyto knihovny jsou následně zabaleny a umožňují komunikaci pomocí zpráv s jinými ROS moduly. Tato další vrstva umožňuje opětovné použití softwaru mimo ROS pro jiné aplikace [20, 21].
- *Zdarma a Open-Source*: Úplný zdrojový kód ROS je veřejně dostupný [20, 21].

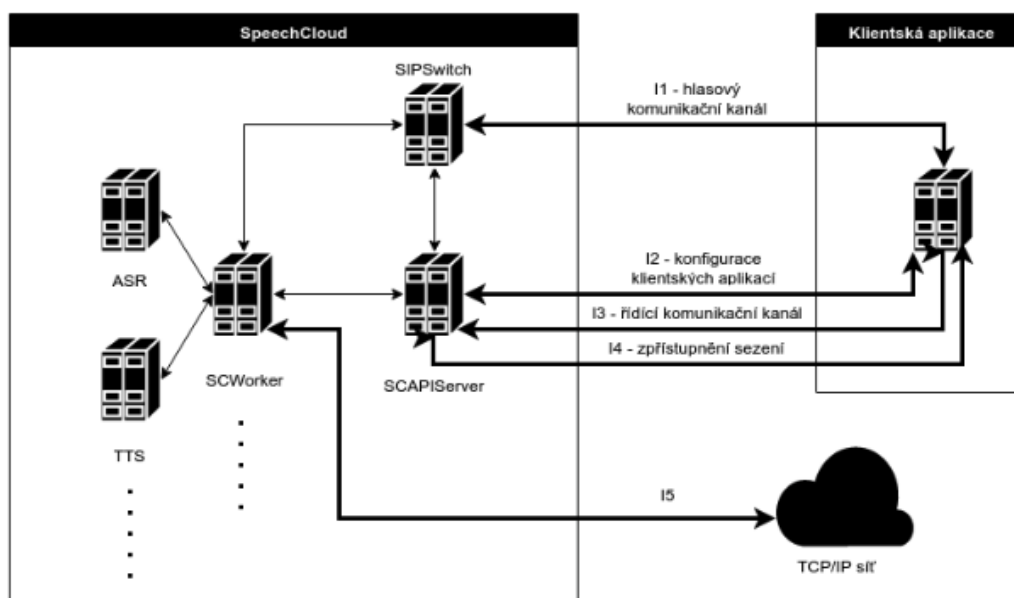
2.5 Speechcloud

SpeechCloud je platforma pro zpracování řeči a analýzu hlasu vyvinutá společností SpeechTech² a Katedrou kybernetiky na ZČU³. Umožňuje například automatický přepis diktátu do psané podoby, generování přirozené řeči ze zadaných textů,

²<https://www.speechtech.cz>

³<https://www.kky.zcu.cz>

verifikaci osob na základě jedinečných charakteristik jejich hlasu a hlasovou komunikaci mezi člověkem a počítačem. Platforma podporuje několik jazyků; češtinu, slovenštinu, angličtinu a němčinu. SpeechCloud je navržen tak, aby umožňoval snadné integrování s různými aplikacemi a systémy třetích stran. [30]. Architektura této platformy je naznačena na Obrázku 2.6.



Obrázek 2.6: Diagram architektury SpeechCloudu (převzato z [33])

Architektura SpeechCloudu je založena na Docker images a umožňuje využít výpočetní výkon velkého počítačového clusteru. Klient SpeechCloudu komunikuje s API serverem a spouští sezení pomocí specifické URL s požadovanými technologiemi. API server poté přidělí konkrétního pracovníka, který zpracovává zvuk a přenáší ho pomocí FreeSwitch mezi klientem a pracovníkem. Kromě zvuku jsou přenášeny i řídící zprávy pomocí WebSocket připojení jako zprávy ve formátu JSON. Pracovníci mají také sadu nástrojů pro zpracování výstupů rozpoznávání řeči, včetně algoritmu pro detekci sémantických entit výstupu rozpoznávání. Kromě toho existuje nástroj pro údržbu modelů a automatickou interpolaci specifických jazykových modelů pro konkrétní dialogy [33].

Základní SpeechCloud technologie lze označit jako:

- Automatické rozpoznávání řeči (ASR) – SpeechCloud využívá pokročilé ASR technologie, které umožňují převádět mluvenou řeč na text. Tato technologie používá strojové učení a hluboké neuronové sítě, aby dosáhla vysoké přesnosti a robustnosti při rozpoznávání řeči.

- Text-to-speech (TTS) – SpeechCloud také umožňuje generovat řeč z textu pomocí TTS technologie. Podporováno je několik hlasových stylů.

2.6 Komunikační protokoly

Komunikační protokoly jsou pravidla a postupy, které slouží k výměně informací mezi dvěma nebo více zařízeními v počítačové síti. Tyto protokoly definují, jakým způsobem jsou data přenášena, jak jsou zabezpečena a jakým způsobem jsou přijímána a zpracovávána. Existuje mnoho různých komunikačních protokolů, které se liší podle účelu a specifikace použití. V této práci jsou využity následující; HTTP, MQTT a WebSockets.

2.6.1 HTTP

HTTP (Hypertext Transfer Protocol) je protokol určený pro přenos hypertextových dokumentů na internetu. Jedná se o bezstavový protokol, tj. každý požadavek je zpracováván odděleně a nezávisle na předchozích požadavcích. HTTP funguje na aplikační vrstvě v TCP/IP síťové architektuře a využívá standardní port 80 pro komunikaci. HTTP využívá model klient-server, kde klient pošle požadavek na server, který odpovídá příslušnou odpovědí.

HTTP požadavek se skládá z několika částí, včetně řádku požadavku, hlaviček a těla. Řádek požadavku obsahuje metodu požadavku (GET, POST, PUT, DELETE apod.), adresu URL, která má být požadována, a verzi protokolu HTTP, která má být použita. Hlavičky obsahují další informace o požadavku, jako jsou např. informace o klientovi a typy dat, které server může poskytnout. Tělo obsahuje samotná data, pokud jsou součástí požadavku.

HTTP odpověď se skládá také z několika částí, včetně řádku s kódem odpovědi, hlaviček a těla. Kódy odpovědí jsou standardizované a určují výsledek požadavku, zda byl úspěšně zpracován, nebo zda došlo k nějaké chybě apod. Hlavičky obsahují další informace o odpovědi, jako jsou např. informace o serveru a typy dat, které jsou součástí odpovědi. Tělo obsahuje samotná data, která jsou součástí odpovědi.

2.6.2 MQTT

MQTT (Message Queuing Telemetry Transport) je protokol pro komunikaci mezi zařízeními přes síť, který je navržen pro efektivní a spolehlivou výměnu zpráv v síťových prostředích s omezenými prostředky.

Protokol MQTT je založen na publish/subscribe architektuře, kde zařízení publikují zprávy na témata (Topics) a jiná zařízení mohou tyto zprávy odebírat ze stejných témat. Témata jsou řetězce textu, které umožňují organizovat zprávy do hierarchické struktury.

MQTT je navržen tak, aby byl velmi jednoduchý a lehký na použití, což znamená, že může být využíván i na zařízeních s omezenými výpočetními a paměťovými kapacitami. Díky této vlastnosti je MQTT ideální pro komunikaci mezi IoT zařízeními.

Další vlastností MQTT je zajištění kvality doručení zpráv. Zařízení mohou publikovat zprávy s různými úrovněmi kvality doručení (QoS), které ovlivňují spolehlivost přenosu.

2.6.3 WebSockets

WebSockets je komunikační protokol umožňující dvěma stranám – klientovi a serveru – navázat interaktivní a duplexní komunikaci v reálném čase. Jedním z hlavních rozdílů oproti klasickému HTTP protokolu je, že WebSockets nevyžadují, aby klient neustále dotazoval server na nová data. Místo toho umožňují otevřít trvalé spojení mezi klientem a serverem, které umožňuje okamžité posílání zpráv mezi oběma stranami.

WebSockets využívá standardní TCP protokol pro vytvoření spojení mezi klientem a serverem. Po navázání spojení mohou obě strany odesílat zprávy, přičemž každá zpráva je tvořena hlavičkou a tělem. Hlavička obsahuje informace o zprávě, jako je například typ zprávy a délka těla. Tělo zprávy obsahuje samotná data, která se mají přenést.

WebSockets je používán pro různé typy aplikací, například pro online chatování, online hry, real-time sledování dat a mnoho dalších. Díky rychlosti a nízké latenci jsou WebSockets obvykle preferovanou volbou pro aplikace, které potřebují rychlou a spolehlivou duplexní komunikaci.

2.7 Základní webové technologie

Základní webové technologie jsou klíčovými stavebními kameny pro vytváření webových stránek, umožňují tvůrcům stránek vytvářet komplexní a interaktivní obsah, který je přizpůsobený různým zařízením a uživatelům. V této práci byly pro tvorbu uživatelského rozhraní využity HTML, CSS a JavaScript.

HTML (*HyperText Markup Language*) je standardní značkovací jazyk používaný

k tvorbě struktury a obsahu webových stránek. Pojem *HyperText* vyjadřuje možnost vzájemně propojovat texty pomocí odkazů, *Markup* označuje schopnost jazyka HTML dávat významy jednotlivým blokům textu s pomocí speciálních značek tzv. elementů či tagů. Mezi základní tagy patří např. <html>, <head>, <body>, <div>, <p>, a <a>.

CSS (*Cascading Style Sheets*) je deklarativní jazyk používaný pro popis vizuální prezentace a formátování webových stránek psaných ve značkovacích jazycích. Umožňuje vytvářet vzhled jako druhou, na obsahu nezávislou vrstvu a různě ho měnit podle aktuálního kontextu.

JavaScript je objektově orientovaný, událostmi řízený programovací jazyk používaný k tvorbě interaktivního a dynamického obsahu na webových stránkách. Umožňuje tvůrcům stránek reagovat na uživatelské akce, jako je například kliknutí na tlačítko nebo odeslání formuláře.

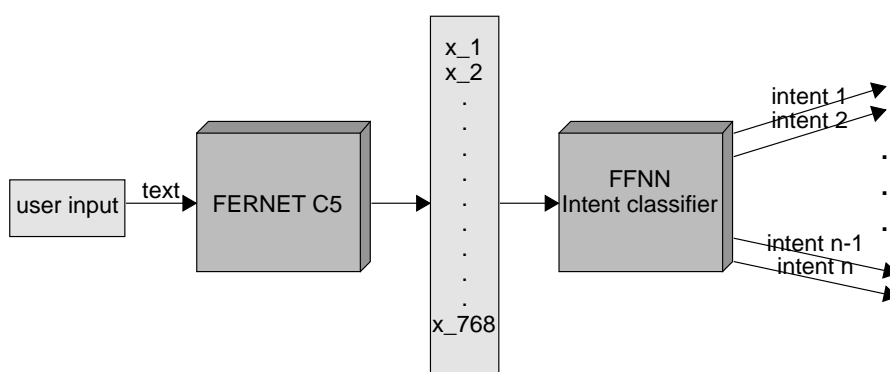
Učení neuronové sítě hlasovým dialogem

3

3.1 Neuronová síť pro klasifikaci záměru

Neuronová síť pro klasifikaci záměru (Intent Classifier) je dopředná neuronová síť, viz Sekce 2.1, která se používá pro automatickou klasifikaci krátkých textů do předem definovaných kategorií (záměrů). Pro trénování takovéto sítě se používá dataset, který obsahuje příklady textů a jim odpovídající záměry. Vstupem do této sítě ale není samotný text, nýbrž jeho vektorová reprezentace. Po natrénování neuronové sítě ji lze použít pro klasifikaci záměru pro nové texty, které nebyly v trénovacím datasetu.

Pro vytvoření vektorových reprezentací textů se používá technologie zvaná sentence embedding, konkrétně Sentence Transformer, viz Sekce 2.2. Tato technologie umožňuje převádět texty na vektorové reprezentace zachycující sémantické vlastnosti.



Obrázek 3.1: Napojení Sentence Transformeru na neuronovou síť

Ilustrativní napojení Sentence Transformeru FERNET-C5 (viz Sekce 2.2.1) na dopřednou neuronovou síť pro klasifikaci záměru je na Obrázku 3.1. Textový

uživatelský vstup je pomocí transformeru FERNET-C5 převeden na vektor délky 768. Tento vektor pak slouží jako vstup do neuronové sítě, jejíž výstupem je záměr textu uživatele.

Přidání nového záměru do této architektury zahrnuje několik kroků. Nejprve musí být vytvořen nový dataset obsahující texty pro nový záměr. Poté musí být použit Sentence Transformer k vytvoření vektorových reprezentací pro tyto texty. Nakonec musí být natrénován nový model neuronové sítě pro klasifikaci záměru, který bude zahrnovat nový záměr.

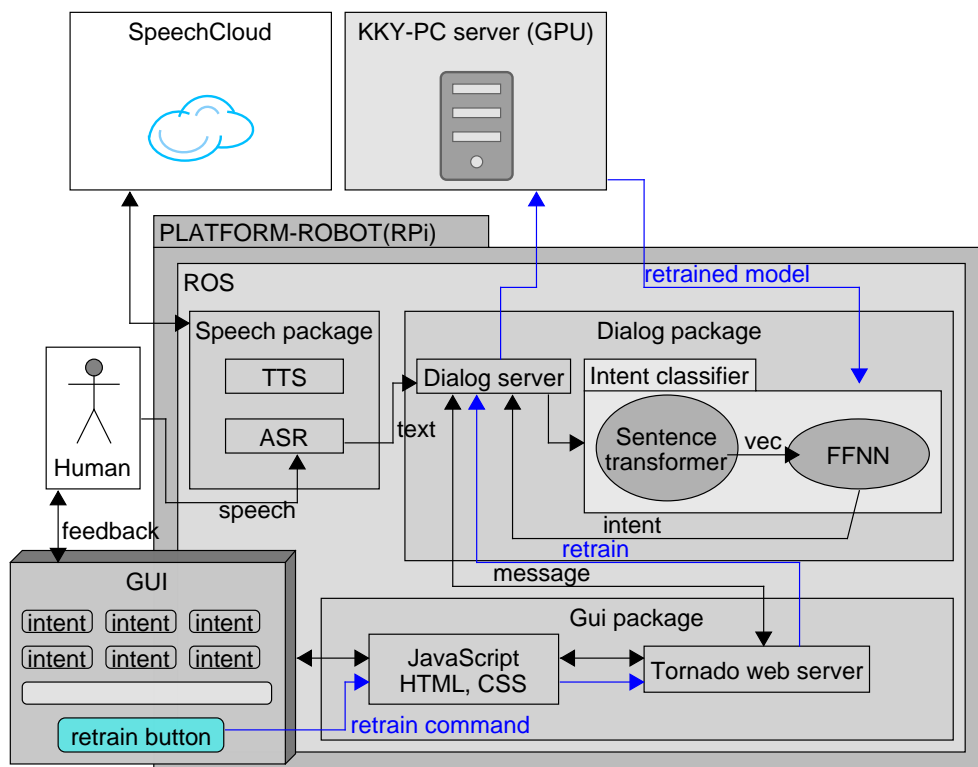
3.2 Zpětnovazební hlasové učení

Diagram na Obrázku 3.2 naznačuje propojení jednotlivých stavebních bloků celé aplikace a tok informací mezi uživatelem a jednotlivými bloky. Běh všech skriptů v rámci robotické entity, konkrétně Raspberry Pi 4B, je zajištěn pomocí softwarového frameworku ROS (více viz Sekce 2.4). Pro každou funkční část aplikace byl vytvořen ROS balík, který umožňuje interakci mezi jednotlivými částmi prostřednictvím zasílání a přijímání zpráv v rámci ROS komunikačního protokolu. Mezi vytvořené ROS balíky patří:

- Speech package – Balík zprostředkovává rozpoznávání řeči a její syntézu. Tento balík komunikuje se Speechcloudem pomocí Websockets.
- Dialog package – Tento balík zajišťuje hlavní funkcionalitu celé dialogové smyčky. Běží zde dialogový server, v rámci nějž jsou registrováni odběratelé pro hlasový vstup od uživatele a pro zprávy z webové stránky (tj. zpětné vazby od uživatele). Dále dialogový server umožňuje publikování zpráv pro webovou stránku. Dialogový balík také zastřešuje funkcionalitu klasifikátoru záměru – kromě trénování modelu – to je prováděno na výkonnějším stroji než RPi. Přeučení probíhá na katedrálním stolním počítači, dialogový balíček komunikuje s tímto počítačem pomocí HTTP požadavků.
- GUI package – Zde běží Tornado webový server, který komunikuje s dialogovým balíčkem pomocí zpráv a s webovou stránkou (JavaScriptem) pomocí Websockets.

Hlasový vstup uživatele je převeden pomocí automatického rozpoznávání řeči do textové podoby. Tento text je předán jako vstup pro predikci neuronové sítě pro klasifikaci záměru (podrobněji o její struktuře viz Sekce 3.1), jejím výstupem je predikovaný záměr uživatele. Ten je pomocí zprávy zaslán do webového serveru a zobrazen na webové stránce. Uživatel má následně možnost na tuto predikci přes grafické uživatelské rozhraní adekvátně reagovat – potvrdit správnost nebo

provést korekci. Tato zpětná vazba je zaslána webovému serveru a pomocí zprávy následně dialogovému serveru, který si ji ukládá. Uživatel má kromě zpětné vazby na určený záměr možnost dát pokyn pro přeučení celé neuronové sítě na základě předchozích zpětných vazeb. V tomto případě se analogicky jako u zpětné vazby dostane tato informace k dialogovému serveru. Ten v tomto případě však navíc zasílá HTTP požadavek stolnímu počítači pro přeučení modelu. Tato situace je v diagramu znázorněna modrou barvou.

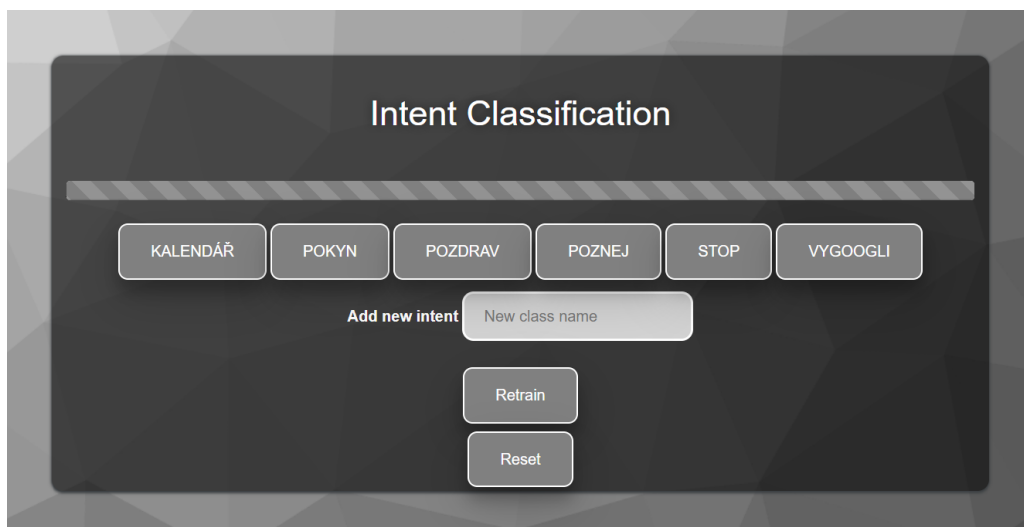


Obrázek 3.2: Propojení jednotlivých částí v rámci celé aplikace

3.3 Uživatelské rozhraní

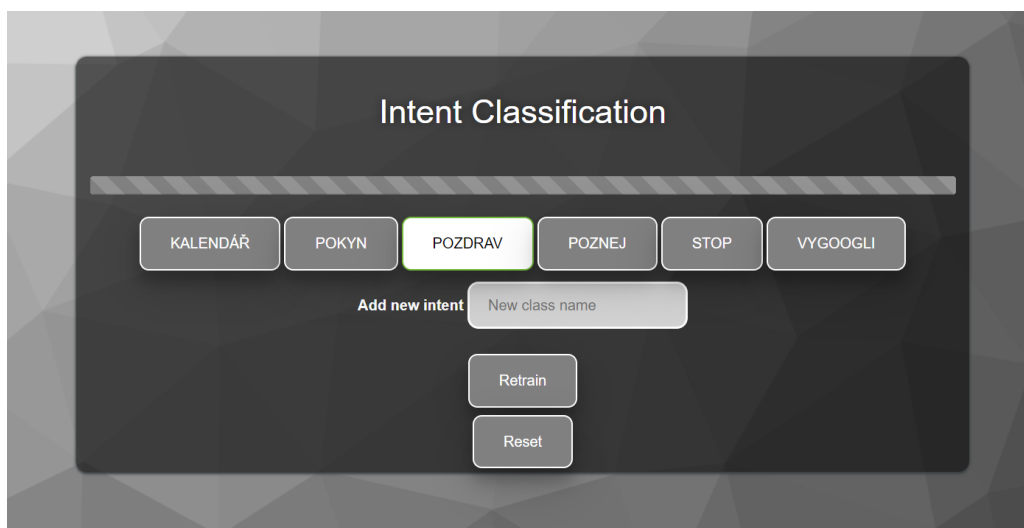
Samotné uživatelské rozhraní je zobrazeno na displeji robotické entity a (v případě propojení s RPi přes LAN kabel a díky běžícímu webovému serveru na RPi) také ve webovém prohlížeči na adrese `robot.local:7010`. Nyní bude nastíněna funkčnost uživatelského rozhraní. Na Obrázku 3.3 je základní vzhled webové stránky v prohlížeči. V horní části stránky je indikátor stavu přeučování sítě (v tomto případě již model dokončil trénování). Pod tímto prvkem jsou umístěna tlačítka reprezentující jednotlivé záměry a pod nimi se nachází textové pole, které umožňuje přidání nového záměru. V dolní části stránky se nacházejí další dvě

tlačítka, přičemž první slouží k přeučení sítě a druhé ke resetování modelu do původního stavu (tj. stavu před zahájením učení dialogem).



Obrázek 3.3: Celkový náhled na uživatelské rozhraní

Obrázek 3.4 ilustruje situaci, kdy byl rozpoznán hlasový vstup uživatele a byl predikován jeho záměr. V této ilustrativní situaci byl rozpoznáný text „Ahoj, robote“, klasifikátor záměru predikoval, že záměr tohoto textu je „pozdrav“, v uživatelském rozhraní je proto tlačítko s touto třídou zvýrazněno.

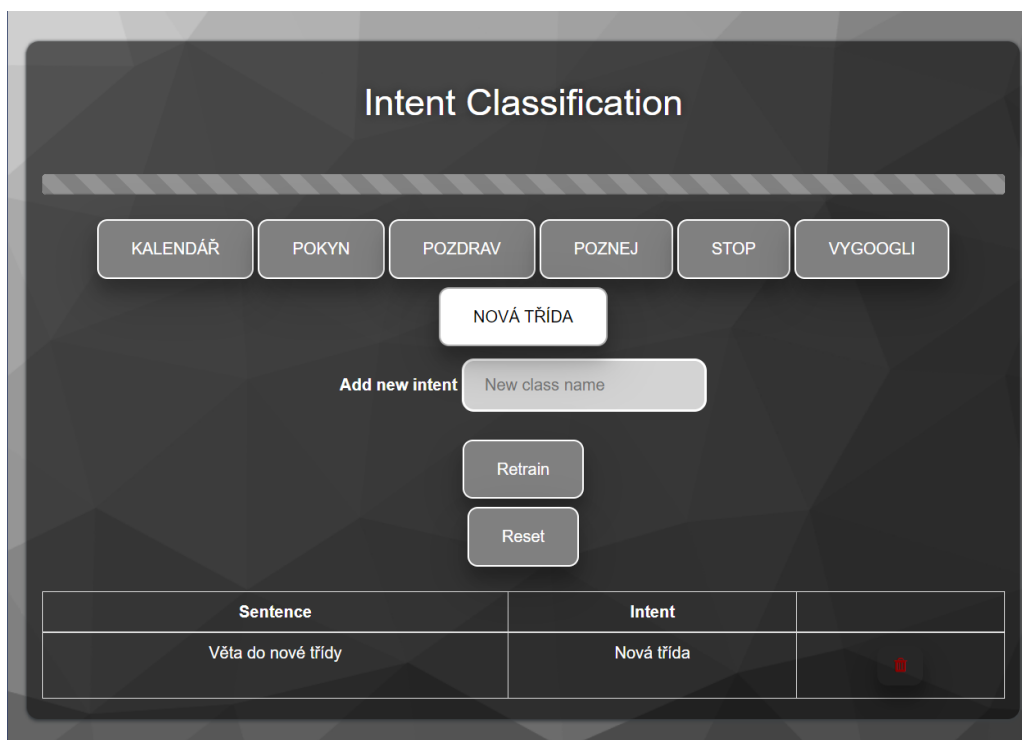


Obrázek 3.4: Zvýraznění predikovaného záměru

Po zobrazení predikovaného záměru má uživatel možnost poskytnout zpětnou vazbu na predikci. Potvrzení lze provést kliknutím na zvýrazněné tlačítko.

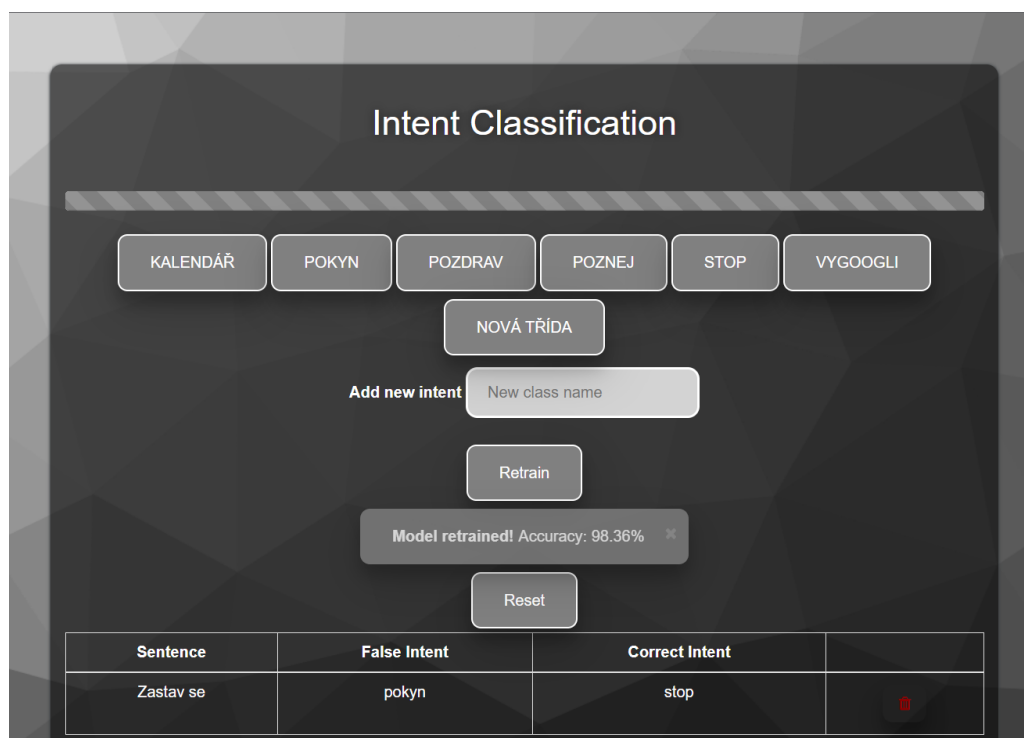
V případě, kdy uživatel shledá predikci jako nesprávnou může provést její korekci kliknutím na tlačítko se správným záměrem. Pokud záměr, který uživatel považuje za korektní, není na žádném z tlačítek, může jej přidat – napsáním jména nového záměru a stiskem klávesy *Enter*. Aby byla zajištěna plynulost učícího dialogu, má uživatel k dispozici 2 sekundy na poskytnutí zpětné vazby. Pokud v této době uživatel nezasáhne, predikovaný záměr je považován za správný. Po poskytnutí zpětné vazby se v dolní části stránky objeví tabulka obsahující posledních 5 anotovaných vzorků, kliknutím na ikonku koše může uživatel libovolný vzorek odstranit (například v případě, kde omylem klikl na nesprávné tlačítko nebo nestihl zareagovat do dvou sekund).

Je-li nyní vstupem od uživatele text do nové třídy, na Obrázku 3.5 je tímto textem „Věta do nové třídy“, změnil uživatel jeho záměr na „nová třída“ vytvořením tohoto tlačítka a kliknutím na něj.



Obrázek 3.5: Korekce predikovaného záměru na záměr „Nová třída“

Tyto zpětné vazby od uživatele jsou průběžně ukládány a uživatel má možnost je propagovat do modelu klasifikátoru záměru pomocí tlačítka *Retrain*. Po stisknutí tohoto tlačítka dojde k přeučení modelu na starém datasetu obohateném o nové vzorky (duplicity jsou odstraněny). Po přeučení, Obrázek 3.6, je zobrazena přesnost nového modelu a vzorky z celého datasetu, u nichž model při validaci chyboval. Uživatel má možnost tyto vzorky smazat kliknutím na ikonu koše.



Obrázek 3.6: Dokončení přeučení, možnost odstranění chybných vzorků

4.1 Ručně vytvořený dataset

Tento dataset obsahuje pouze 60 vzorků a nebyl rozdělen na trénovací, validační a testovací data. Vzorků je malé množství záměrně, protože dataset sloužil především k odladění vyvíjených metod (*proof of concept*). Je možné tato data pojmout jako startovací bod a neuronovou síť dotrénovat navrženým algoritmem, tj. za základě zpětné vazby od uživatele (*human in the loop* přístup). Celý dataset je k nahlédnutí v Příloze A. V Tabulce 4.1 jsou podrobnější informace o tomto datasetu.

Tabulka 4.1: Statistiky datasetu 1

Informace o datasetu	
Počet vzorků	60
Počet unikátních záměrů/tříd	6
Počet unikátních slov	114
Počet vzorků v jedné třídě	10
Průměrný počet slov na vzorek	2.78



4.2 Robustní dataset pro klasifikaci záměrů

Tento dataset byl převzat z publikace [13]. Obsahuje více než 23000 vzorků, z toho 22500 *in-scope* vzorků pokrývá 150 různých záměrů, které lze rozdělit do deseti různých oblastí, zbylé vzorky jsou tzv. *out-of-scope*. Dataset je veřejně dostupný na platformě GitHub¹.

V rámci této práce bylo využito 22500 *in-scope* vzorků, které byly přeloženy do češtiny a byly rozděleny na trénovací, validační a testovací data. Dataset slouží

¹<https://github.com/clinc/oos-eval>

k předtrénování většího modelu. Rozdělení dat a informace o nich jsou uvedeny v Tabulce 4.2.

Tabulka 4.2: Statistiky datasetu 2

Informace o datasetu			
	Trénovací	Validační	Testovací
Počet vzorků	15000	3000	4500
Počet unikátních záměrů	150	150	150
Počet unikátních slov	10633	4100	5114
Počet vzorků na jeden záměr	100	20	30
Průměrný počet slov na vzorek	6.78	6.78	6.70

Experimenty a výsledky

5

5.1 Porovnání frameworků

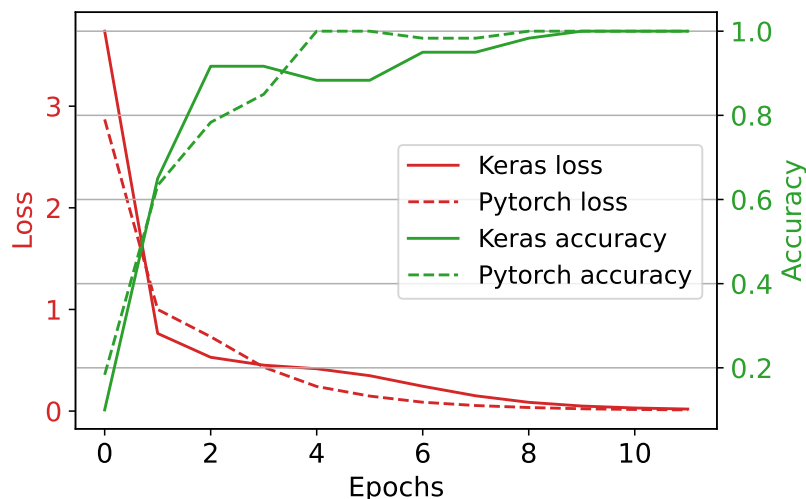
Keras [11] a PyTorch [19] jsou dva populární frameworky pro vytváření a trénování neuronových sítí. Keras poskytuje vyšší úroveň abstrakce, což umožňuje snazší vytváření a trénování sítí bez nutnosti velkého množství kódu. Na druhé straně PyTorch poskytuje větší kontrolu a flexibilitu při vytváření sítí a při zásahu do detailnějších částí algoritmů.

V rámci tohoto experimentu bylo provedeno porovnání rychlosti a efektivity frameworků PyTorch a Keras při použití stejných hyperparametrů neuronové sítě a stejného datasetu. Hyperparametry jsou vypsány v Tabulce 5.1, použitým datasetem byl Dataset 4.1 se šedesáti vzorky, zvolená ztrátová funkce byla *Categorical Cross-Entropy*. Obě neuronové sítě dosahovaly s tímto nastavením přesnosti 100 %.

Tabulka 5.1: Hyperparametry neuronových sítí použitých v experimentu pro výběr frameworku

Parametr	Hodnota
Počet skrytých vrstev	1
Počet neuronů ve skryté vrstvě	35
Aktivační funkce	Relu, Softmax
Learning rate	0.01
Batch size	40
Počet epoch	30
Optimizér	Adam

Obrázek 5.1 ilustruje vývoj přesnosti (Accuracy) a ztráty (Loss) při trénování neuronové sítě v nástrojích Pytorch a Keras pro výše zmíněné hyperparametry, ztrátovou funkci a dataset.



Obrázek 5.1: Průběh trénování neuronové sítě v nástrojích Pytorch a Keras pro prvních 12 epoch

Experiment byl proveden na osobním počítači Lenovo IdeaPad Gaming 3 s procesorem Intel Core i5 a grafickou kartou NVIDIA GeForce GTX 1650 a na Raspberry Pi 4B. Během experimentu byly měřeny tři časy: doba importu knihoven, doba načtení modelu neuronové sítě a doba predikce třídy pro jeden vzorek. Bylo provedeno 10 realizací tohoto experimentu, výsledky jsou v Tabulce 5.2.

Poslední sloupec této tabulky pak navíc obsahuje doby načtení modelu a doby predikce jednoho vzorku pro neuronovou síť trénovanou na Datasetu 4.2, hyperparametry této sítě jsou popsány v Tabulce 5.4 (doby načtení knihoven nebyly měřeny, protože byly použity stejné knihovny jako v případě neuronové sítě trénované na Datasetu 4.1). Porovnání bylo provedeno pouze na osobním počítači pro ověření konceptu rychlosti frameworků pro větší neuronovou síť, používanou v následujících experimentech.

Tabulka 5.2: Doby načtení knihoven, načtení modelů a predikcí jednoho vzorku pro frameworky Pytorch a Keras (v sekundách).

	Pytorch		
	Dataset 4.1		Dataset 4.2
Operace	Čas [s] (laptop)	Čas [s] (RPi)	Čas [s] (laptop)
Načtení knihoven	3.329±0.055	13.785±0.087	-
Načtení modelu	0.001±0.000	0.004±0.000	0.011±0.013
Predikce jednoho vzorku	0.036±0.014	0.672±0.226	0.036±0.003
	Keras		
	Dataset 4.1		Dataset 4.2
Operace	Čas [s] (laptop)	Čas [s] (RPi)	Čas [s] (laptop)
Načtení knihoven	1.805±0.046	5.862±0.529	-
Načtení modelu	0.001±0.000	0.004±0.001	0.184±0.010
Predikce jednoho vzorku	0.028±0.001	0.644±0.081	0.136±0.011

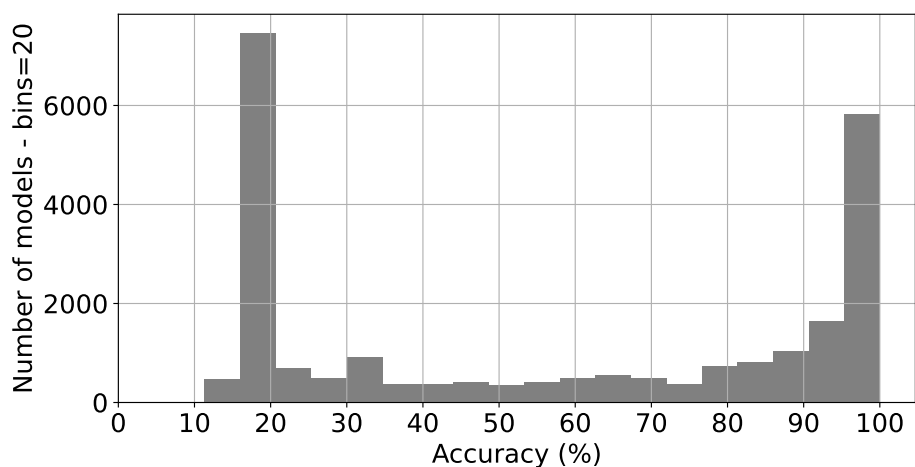
Výsledky experimentu ukázaly, že v případě importu knihoven byl PyTorch pomalejší než Keras (přibližně 2×). Pro načtení modelu byly časy téměř srovnatelné, v případě predikce jednoho vzorku byl Keras mírně rychlejší než Pytorch (o 0.008 sekund na osobním počítači, o 0.028 sekund na RPi).

5.2 Volba struktury sítě

Výběr velikosti sítě je jedním z klíčových rozhodnutí při návrhu neuronových sítí a ovlivňuje jak přesnost, tak rychlost učení modelu. V rámci tohoto experimentu bylo cílem nalézt hyperparametry pro neuronovou síť, která bude aplikována na Raspberry Pi 4B a to tak, aby tato síť dosahovala co největší přesnosti a zároveň byla dostatečně rychlá.

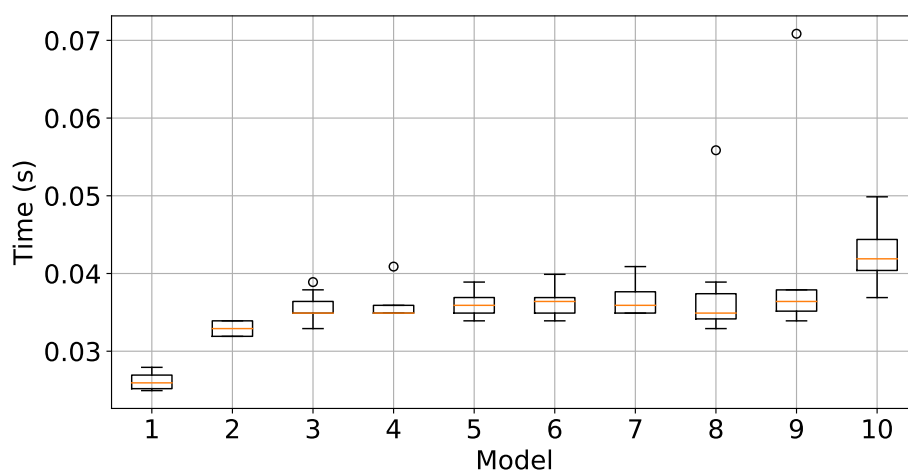
Experiment byl proveden na osobním počítači na jednoduché úloze klasifikace pro Dataset 4.1. Použit byl framework PyTorch a implementace sítě s jednou skrytou vrstvou. Nejprve bylo metodou *Grid search* testováno 24000 modelů s různými kombinacemi hyperparametrů (viz Příloha B) a s pěti různými seedy¹ z hlediska jejich přesnosti. Histogram zobrazující četnosti modelů v závislosti na dosažené přesnosti je na Obrázku 5.2.

¹ Pod pojmem seed se rozumí seed pro generování pseudonáhodných čísel.



Obrázek 5.2: Histogram četností modelů v závislosti na jejich přesnosti

Dále byly vybrány neuronové sítě, které dosáhly přesnosti 100 % pro všech pět testovaných seedů. Tyto vybrané neuronové sítě byly znovu testovány pro dalších 10 různých seedů, aby byla důkladněji ověřena jejich kvalita, tj. jestli pro všechny tyto realizace dosahují přesnosti 100 %. Kromě toho byla pro jednotlivé realizace měřena doba trénování sítě. V kontextu cílového úkolu a potřeb sítě umožňující online přeučení byl kladen důraz na dosažení dostatečné rychlosti trénování. Bylo získáno 514 neuronových sítí s přesností 100 % pro všechny realizace, z nich bylo vybráno 10 nejrychlejších sítí. Boxplot doby trénování vybraných sítí je na Obrázku 5.3, konkrétní hyperparametry pak v Tabulce 5.3 (indexy modelů v grafu odpovídají řádkům v tabulce).

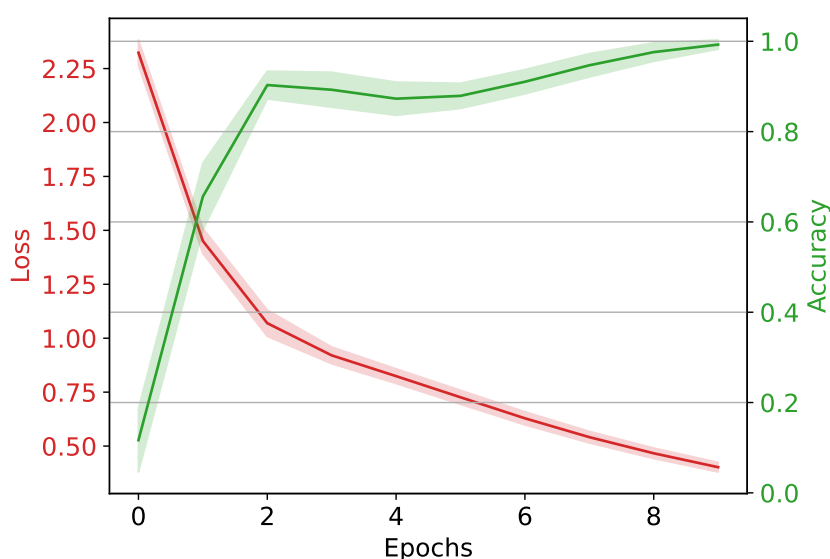


Obrázek 5.3: Boxplot doby trénování jednotlivých modelů

Tabulka 5.3: Hyperparametry deseti nejrychlejších neuronových sítí a jejich doby trvání trénování, koeficient učení byl ve všech případech 0.01 a optimizér Adam

Model	Počet neuronů ve skryté vrstvě	Počet epoch	Batch size	Doba trénování [s]
1	45	10	40	0.0262±0.0011
2	20	15	40	0.0329±0.0009
3	45	15	40	0.0355±0.0017
4	50	15	50	0.0358±0.0018
5	35	15	40	0.0361±0.0017
6	35	15	50	0.0363±0.0019
7	50	15	40	0.0366±0.0019
8	30	15	40	0.0373±0.0065
9	45	15	50	0.0396±0.0105
10	40	15	40	0.0424±0.0036

Z Tabulky 5.3 je zřejmé, že nejlepší výsledků dosáhla neuronová síť číslo 1 se 45 neurony ve skryté vrstvě, 10 epochami, učícím koeficientem 0.01, velikostí dávky 40 a optimizérem Adam. Na Obrázku 5.4 je zobrazen průběh trénovacího procesu této neuronové sítě, ztrátová funkce byla *Categorical Cross-Entropy*.



Obrázek 5.4: Průběh trénování neuronové sítě s optimálními hyperparametry pro 10 realizací

5.3 Trénování v závislosti na počtu vzorků a tříd

U klasifikačních úloh je nezbytné mít dostatečné množství trénovacích dat, aby bylo možné naučit neuronovou síť rozlišovat jednotlivé třídy. V rámci tohoto experimentu byly analyzovány schopnosti neuronových sítí natrénovat se s různým počtem vzorků a tříd a vliv tohoto počtu na přesnost klasifikace.

V průběhu experimentu byly použity různé datové sady s různým počtem tříd a vzorků na třídu, vždy se jednalo o modifikace Datasetu 4.2. Tento experiment byl rozdělen na tři části; výběr hyperparametrů pro neuronovou síť pro klasifikaci záměru na Datasetu 4.2, analýza schopnosti neuronové sítě natrénovat se v závislosti na architektuře sítě a na počtu tříd, analýza schopnosti neuronové sítě natrénovat se v závislosti na počtu tříd a na počtu vzorků v jednotlivých třídách.

Výběr hyperparametrů pro neuronovou síť byl proveden na celém Datasetu 4.2 pro framework Pytorch. Validační data byla využita k výpočtu validační ztráty po každé epoše, přičemž pokud byla tato ztráta rostoucí po více než pětinu počtu epoch, bylo trénování neuronové sítě předčasně ukončeno. V průběhu trénování neuronové sítě byl také dynamicky měněn koeficient učení, každých padesát epoch byl přenásoben koeficientem 0.9 a došlo tak k jeho poklesu. Experimentálně vybrané hyperparametry jsou vypsány v Tabulce 5.4. Tato neuronová síť dosahovala přesnosti 0.854 ± 0.002 a ztráty² 0.588 ± 0.015 (testováno pro 15 realizací).

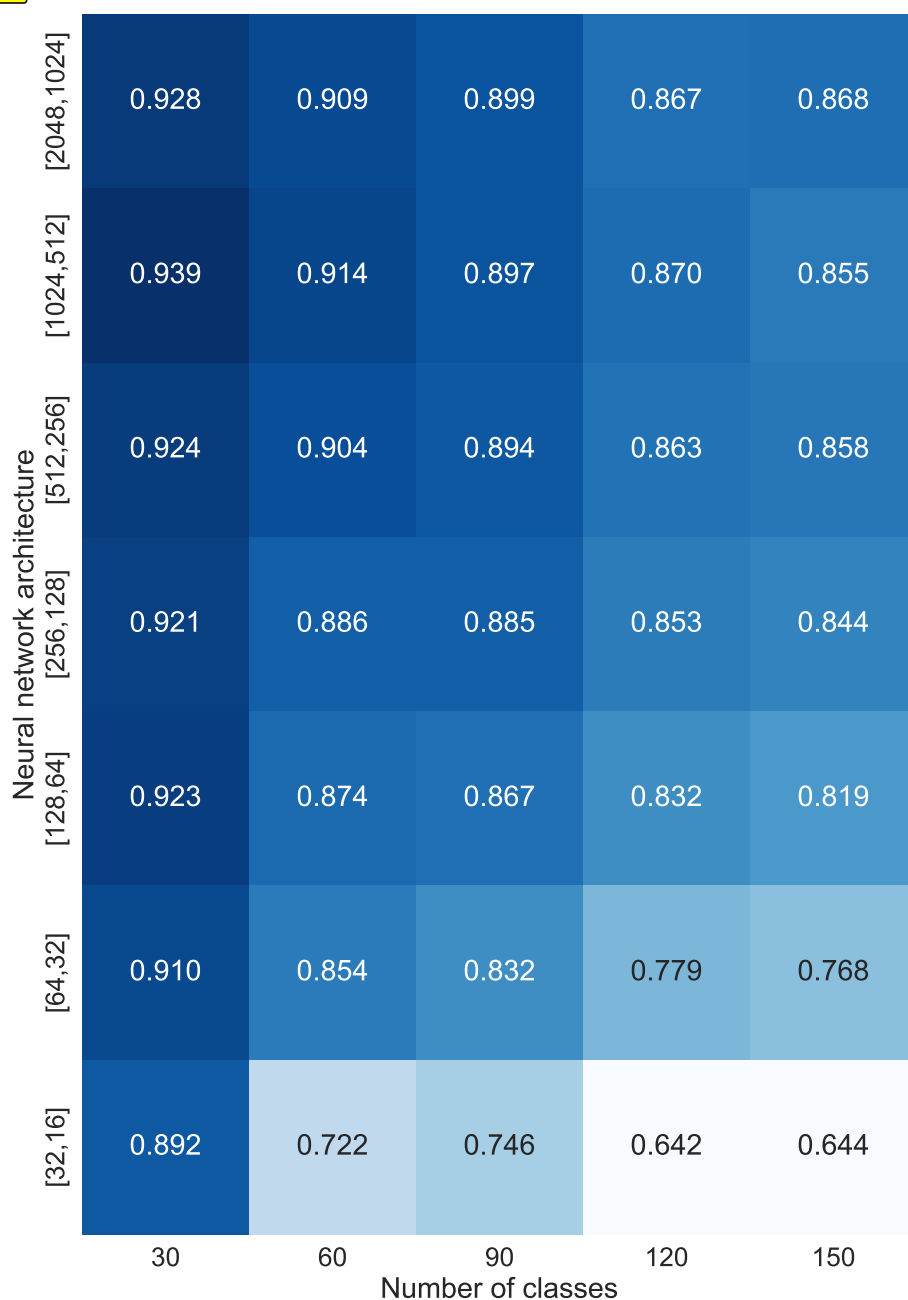
Tabulka 5.4: Hyperparametry neuronové sítě pro klasifikaci záměru trénované na Datasetu 4.2

Parametr	Hodnota
Počet skrytých vrstev	2
Počet neuronů ve skrytých vrstvách	512, 256
Aktivační funkce	Sigmoid, Sigmoid, Softmax
Počáteční hodnota koeficientu učení	0.003
Batch size	2048
Počet epoch	300
Optimizér	Adam

Pro analýzu schopnosti neuronové sítě natrénovat se v závislosti na architektuře sítě a na počtu tříd bylo vytvořeno sedm neuronových sítí s různým počtem

²Ztrátová funkce byla *Categorical Cross-Entropy*.

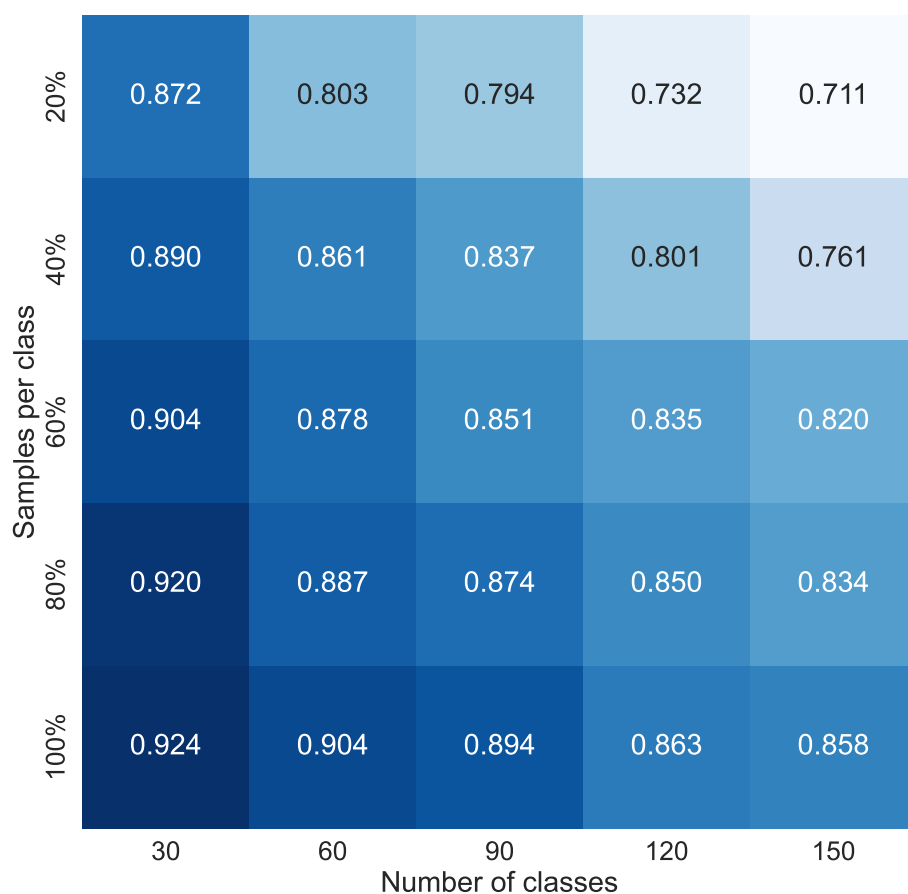
neuronů ve skrytých vrstvách, ostatní hyperparametry odpovídaly hyperparametrům z Tabulky 5.4. Pro každou z těchto sítí bylo vytvořeno pět modifikací Datasetu 4.2 obsahující různé počty tříd. Vzhledem k tomu, že v původním datasetu bylo 150 tříd, byly počty tříd rozděleny po násobcích třiceti – 30, 60, 90, 120, 150. Na Obrázku 5.5 je heatmapa zobrazující přesnosti neuronové sítě v závislosti na její architektuře a počtu tříd. Na svislé ose jsou vypsány počty neuronů v jednotlivých skrytých vrstvách.



Obrázek 5.5: Přesnost neuronové sítě v závislosti na její architektuře (počtu neuronů ve skrytých vrstvách) a počtu tříd, počty vzorků ve třídách byly 100 %

Analýza schopnosti neuronové sítě natrénovat se v závislosti na počtu tříd a na počtu vzorků v jednotlivých třídách byla provedena pro neuronovou síť s hyperparametry z Tabulky 5.4. Počty tříd byly stejné jako u předchozí analýzy. Počty trénovacích vzorků pro jednotlivé třídy byly opět děleny do pěti kategorií po

dvaceti procentních úbytcích. Výsledky této analýzy jsou na Obrázku 5.6 představujícím heatmapu přesností neuronové sítě.



Obrázek 5.6: Přesnost neuronové sítě v závislosti na počtu tříd a počtu vzorků na třídu, pro zafixovanou architekturu [512,256]

Výsledky experimentu ukázaly, že přesnost sítě se zvyšovala s rostoucím počtem neuronů ve skrytých vrstvách, zároveň tato přesnost klesala s rostoucím počtem tříd. Vyšší počet vzorků v jednotlivých třídách pak opět zvyšoval přesnost neuronové sítě.

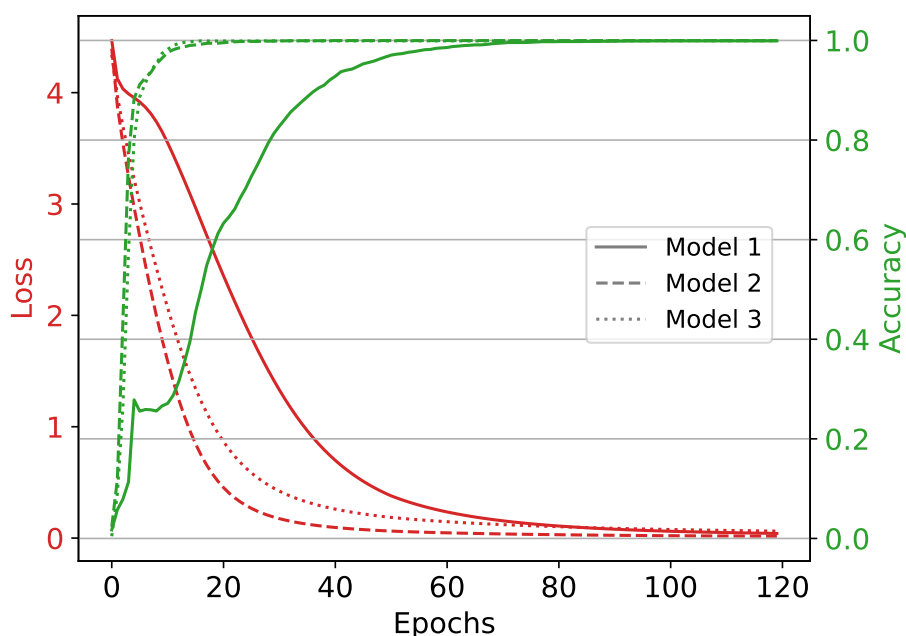
5.4 Analýza použití předtrénované neuronové sítě

Předtrénované neuronové sítě jsou síťové architektury, které se naučily efektivně reprezentovat vysokoúrovňové funkce na základě velkého množství trénovacích dat a mohou být využity pro různé úlohy v oblasti umělé inteligence. Tento experiment se zabývá zhodnocením využití takovéto sítě.

V rámci experimentu byla využita neuronová síť definovaná v Tabulce 5.4 trénovaná na Datasetu 4.2. Byly vytvořeny tři modely neuronové sítě, které byly porovnávány z hlediska jejich průběhu trénování a validace a z hlediska výsledné přesnosti a ztráty³:

1. *Model 1*: Model vytvořený náhodnou inicializací všech vrstev.
2. *Model 2*: Model vytvořený z přetrénovaného modelu transferem parametrů – poslední vrstva neuronové sítě byla nahrazena za náhodnou inicializaci.
3. *Model 3*: Model vytvořený z přetrénovaného modelu transferem parametrů – poslední vrstva neuronové sítě byla nahrazena za náhodnou inicializaci, ostatní vrstvy byly „zmrazeny“.

Nejprve byly tyto modely porovnány pro modifikaci Datasetu 4.2 o 60 třídách a 60 trénovacích vzorcích. Model 1 dosáhl přesnosti 0.89 a ztráty 0.39, model 2 dosáhl přesnosti 0.91 a ztráty 0.35, model 3 dosáhl přesnosti 0.90 a ztráty 0.39. Na Obrázku 5.7 je zobrazen vývoj přesnosti a ztráty při trénování těchto modelů. Je zřejmé, že model 2 a model 3 mají téměř identický průběh přesnosti trénování a že dosahují přesnosti 1.0 přibližně o 60 epoch dříve než model 1. V případě poklesu ztráty si vedl nejlépe model 2, mírně pomalejší byl model 3 a nejpomalejší byl model 1.

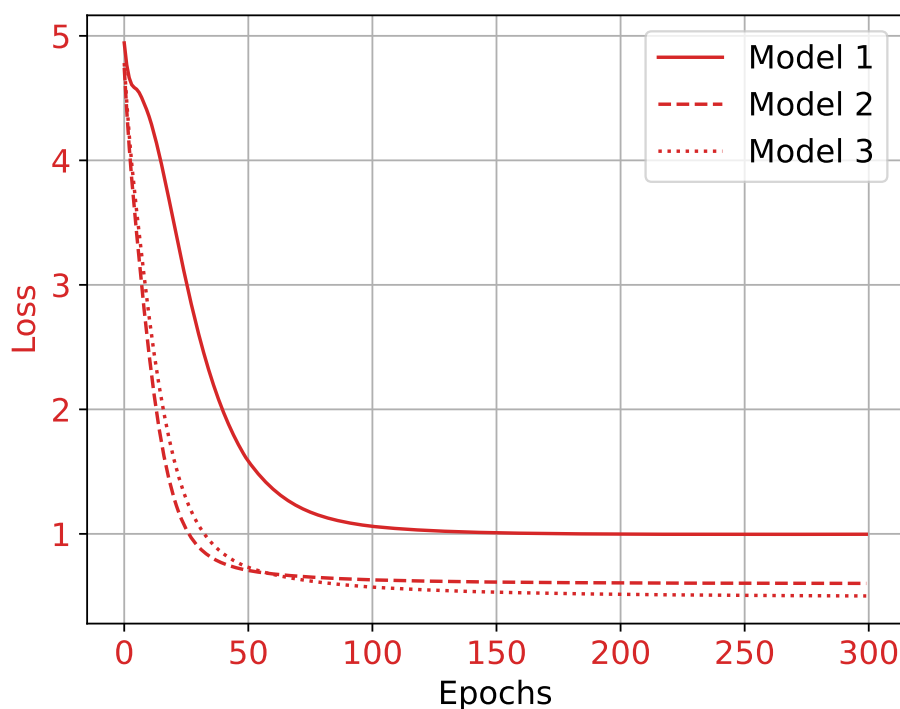


Obrázek 5.7: Vývoj přesnosti a ztráty při trénování modelů pro 60 tříd pro prvních 120 epoch

³Ztrátová funkce byla *Categorical Cross-Entropy*.

V příloze C na Obrázku C.1 je pak znázorněn průběh ztráty při validaci těchto modelů, trend poklesu validační ztráty je analogický s trendem poklesu trénovací ztráty pro všechny tři modely.

Dále byla testována situace, kde bylo z původního datasetu vybráno 130 tříd **s 20 trénovacími vzorky**. Jednalo se tedy o poměrně okrajový případ řídkého datasetu. V tomto případě dosáhl model 1 přesnosti 0.73 a ztráty 1.04, model 2 přesnosti 0.83 a ztráty 0.65 a model 3 měl přesnost 0.85 a ztrátu 0.57. Vývoj validační ztráty pro všechny modely je na Obrázku 5.8, graf vývoje přesnosti a ztráty při trénování je k nahlédnutí v Příloze C na Obrázku C.2. Je patrné, že validační ztráta modelu 1 je značně vyšší (přibližně 2×) než u ostatních modelů a že po přibližně posledních 150 epoch již neklesala. Validační ztráta modelu 2 klesala v prvních 60 epochách rychleji než v případě modelu 3, po té se ale ustálila, zatímco validační ztráta modelu 3 dále klesala a ve výsledku dosáhla hodnoty o 0.08 nižší.



Obrázek 5.8: Vývoj ztráty při validaci modelů pro 130 tříd

Tabulka 5.5 pak obsahuje **dobu** trénování modelů pro obě testované situace. Je patrné, že model 3 měl **nejrychlejší** dobu trénování pro obě situace, a to přibližně o 10 sekund.

Výsledky experimentu naznačují, že pro pouhou modifikaci většího datasetu je fine-tuning a transfer learning efektivnějším přístupem než trénink „od nuly“, pokud je k dispozici již předtrénovaný model. Při porovnávání modelů pro dataset

Tabulka 5.5: Doby trénování modelů pro různý počet tříd

Model	60 tříd	130 tříd
1	33.57 [s]	32.48 [s]
2	34.01 [s]	32.02 [s]
3	23.41 [s]	25.79 [s]

se 60 třídami dosahovaly modely 2 a 3 srovnatelných výsledků s přesností 0.9. Průběh poklesu trénovací i validační ztráty byl u modelu 2. Modelu 1 dosáhl přesnosti pouze 0.89 a měl nejpomalejší (z hlediska počtu epoch) trend při trénování i validaci. V případě datasetu obsahujícího 130 tříd dosáhl nejlepších výsledků model 3 (přesnosti 0.85 a ztráty 0.57).

5.5 Vyhodnocení metody učení dialogem

Učení dialogem umožňuje interaktivně trénovat a zlepšovat schopnosti modelu prostřednictvím komunikace s uživateli v reálném čase. V tomto experimentu byla simulována situace opakovaného přeučování modelu na základě zpětné vazby od uživatele.

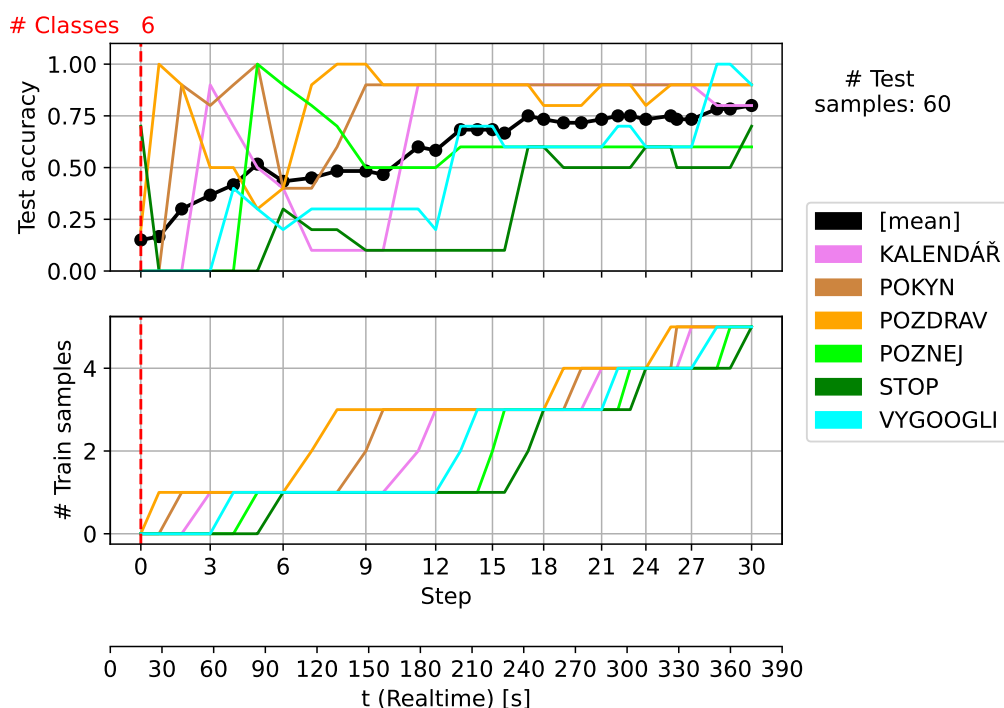
Tento experiment používal předtrénovaný model popsaný v Tabulce 5.4 trénovaný na Datasetu 4.2, přeučována byla vždy jen poslední plně propojená vrstva. Testovacími a validačními vzorky byly všechny vzorky z Datasetu 4.1. V rámci experimentu byly do trénovací množiny iterativně přidávány nové vzorky, pokud se daný vzorek ještě nevyskytoval v testovací a validační množině, byl přidán i tam. Na konci každého kroku byl model přeučen a byla určena jeho přesnost na testovacích datech. Simulovány byly dva scénáře:

1. *Scénář 1:* Počáteční trénovací množina byla prázdná, v každém kroku byl přidán jeden vzorek z Datasetu 4.1, celkem byla přidána polovina tohoto datasetu, pořadí přidávaných vzorků je k dispozici v příloze D.
2. *Scénář 2:* Počáteční trénovací množina obsahovala všech 10 vzorků pro 4 vybrané třídy (POZDRAV, KALENDÁŘ, VYGOOGLI, STOP), zbylé 2 třídy (POZNEJ, POKYN) z Datasetu 4.1 tedy nebyly zastoupeny. V prvních čtyřech krocích bylo přidáno po jednom vzorku do již naučených tříd (trénovací, validační ani testovací množina se tedy neměnila). V dalších krocích bylo přidáno po 5 vzorcích do tříd POZNEJ a POKYN. Nakonec byla zavedena ještě úplně nová třída CHAT a bylo do ní přidáno 6 vzorků, tyto vzorky byly

úplně nové – nejsou obsaženy v Datasetu 4.1. Konkrétní vzorky a jejich pořadí jsou opět v příloze D.



Obrázek 5.9 znázorňuje iterativní přidávání vzorků v rámci scénáře 1. Červenou barvou je znázorněn počet tříd v testovacích datech. První podgraf znázorňuje vývoj přesnosti na testovacích datech pro jednotlivé třídy, druhý podgraf zobrazuje počty trénovacích vzorků pro jednotlivé třídy. Vodorovná osa obou podgrafů zobrazuje počet aktuálních kroků – jejich rozložení je závislé na reálném čase na vodorovné ose v dolní části.

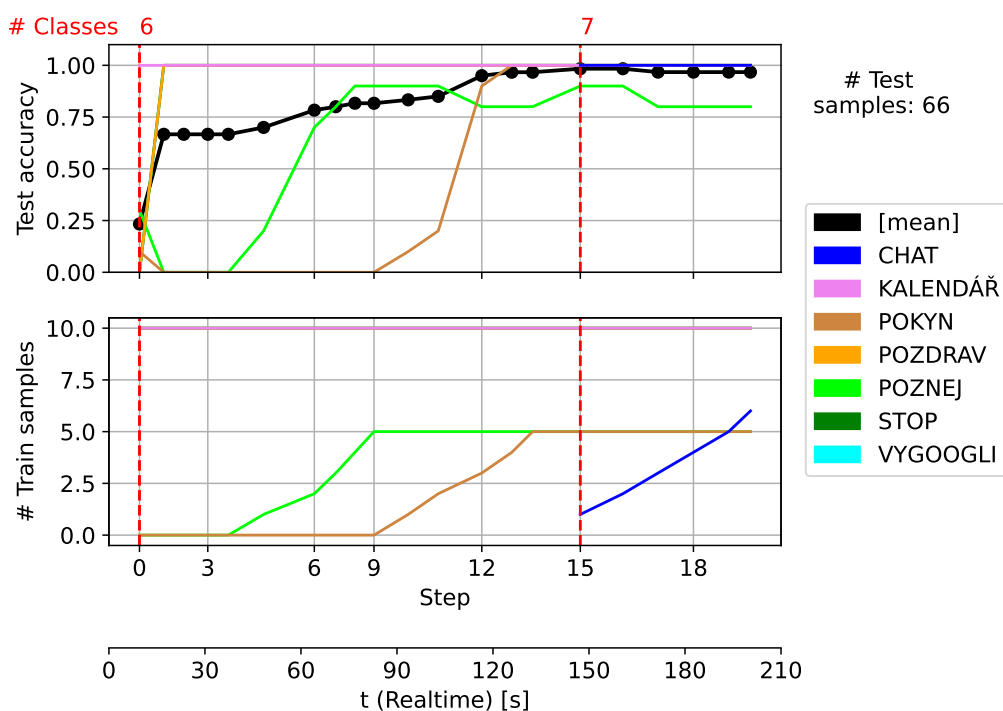


Obrázek 5.9: Vývoj přesnosti pro jednotlivé třídy při iterativním přidávání vzorků v rámci scénáře 1



Na Obrázku 5.10 jsou pak analogické grafy pro scénář 2, kde byla v kroku 15 zavedena nová třída CHAT a bylo do ní přidáno 6 nových vzorků. Celkový počet testovacích vzorků na konci tohoto scénáře byl tedy 66.

Vzorky byly v jednotlivých krocích zadávány ručně, čas uplynulý mezi jednotlivými kroky tedy představuje dobu zadávání daného textu, dobu predikce záměru tohoto textu, dobu strávenou poskytnutím zpětné vazby (korekcí predikovaného záměru nebo jeho potvrzením) a dobu přeučení modelu.



Obrázek 5.10: Vývoj přesnosti pro jednotlivé třídy při iterativním přidávání vzorků v rámci scénáře 2

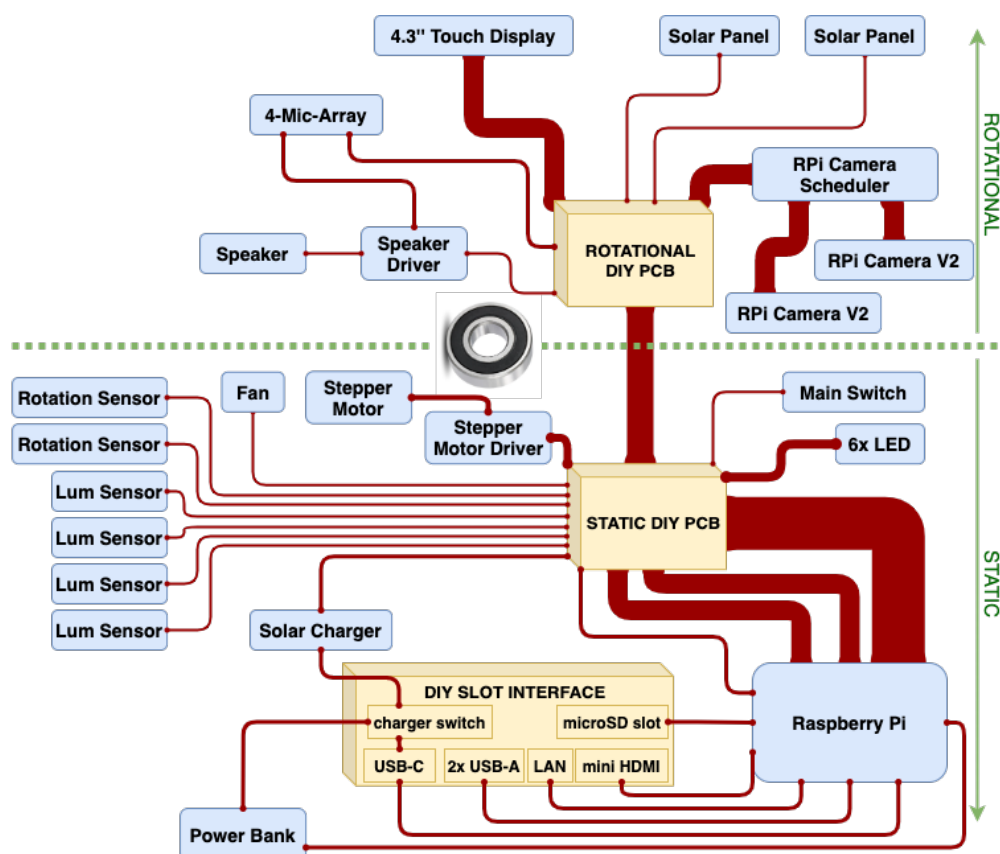
Výsledky experimentu naznačují, že přidání malého množství trénovacích vzorků do třídy neobsahující vzorky žádné se projeví výrazným zvýšením přesnosti klasifikace. Zvláště patrný je tento trend v případě scénáře 2 u přidání zcela nové třídy, je zřejmé, že celková přesnost klasifikace tímto nebyla téměř ovlivněna a nová třída se velice rychle ujal

6.1 Cílová robotická platforma

Cílová robotická platforma s pracovním názvem „Robot“¹ byla navržena jako fyzická entita, která je založena na jednodeskovém počítači ~~typu~~ Raspberry Pi. Tato platforma je navržena s cílem poskytnout nízkonákladové řešení pro testování a aplikaci vědeckých úloh v oblasti přirozené komunikace člověka a stroje.

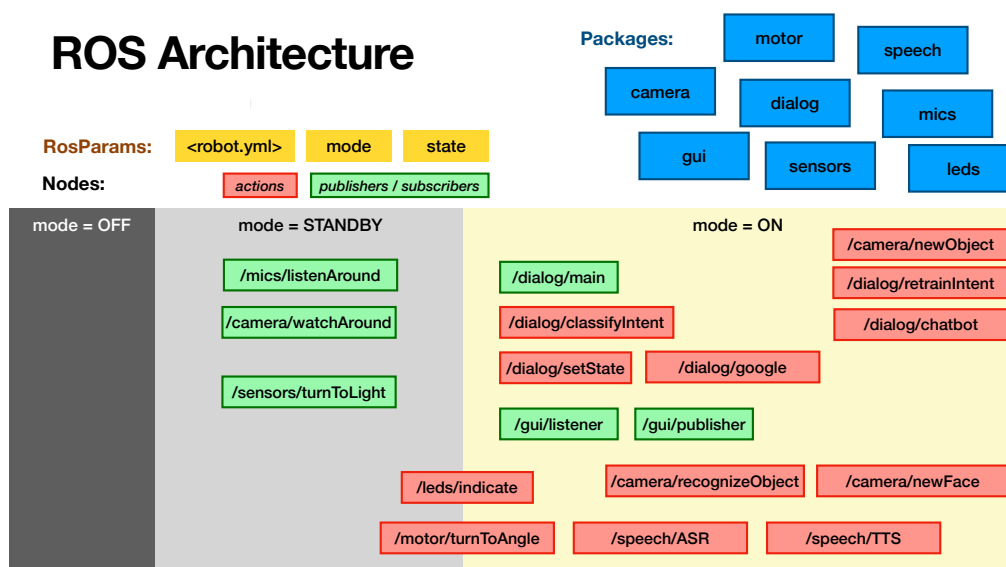
Platforma je vybavena hlasovým a vizuálním rozhraním, které zprostředkovává komunikaci s uživatelem pomocí pole mikrofونů, reproduktoru, dotykového displeje a kamery. Kromě toho platforma obsahuje krokový motor, který umožňuje rotaci určité části robota, senzory světla a solární panel pro zajištění energetického zdroje, indikační LED diody, vlastní zdroj napájení a displej pro zobrazení aktuálního stavu baterie a jednobolohový hlavní spínač. Jednotlivé komponenty využité v robotické entitě a jejich umístění v rámci statické a rotační části je nastíněno na Obrázku 6.1, je zde také náčrt elektrického propojení těchto součástek. Významnou součástí celého projektu je také ROS (Robot Operating System), který zajišťuje hlavní funkčnost platformy. Obrázek 6.2 ukazuje náčrt ROS architektury a seznam použitých balíčků a uzlů.

¹<https://github.com/kitt10/robot>



Obrázek 6.1: Hardwarové propojení jednotlivých součástek použitých na robotické entitě

ROS Architecture



Obrázek 6.2: ROS architektura robotické entity



Výsledná robotická platforma využívá SpeechCloud k syntéze a rozpoznávání řeči pro komunikaci s člověkem. Navíc dokáže rozpoznat různé objekty a známé tváře a umí provádět rotaci horní části tak, aby se natočila směrem k zdroji světla nebo zdroji zvuku. Řízení dialogu je zajištěno klasifikátorem záměru, jako jeden ze záměrů byl navržen chatbot založený na T5. Na Obrázku 6.3 je pak fotografie výsledného vzhledu vytvořené robotické entity.



Obrázek 6.3: Fotografie výsledné robotické entity

6.2 Učení konkrétních záměrů

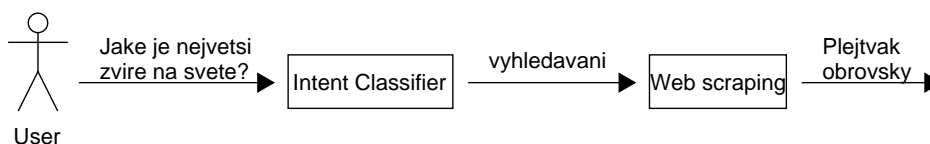
Tato práce se na projektu *Robot.v1* podílela účastí na řešení hlavního dialogového manažeru. Pro potřeby projektu byla vytvořena neuronová síť, která dokáže klasifikovat záměr ze vstupní promluvy, a k ní byl navržen algoritmus přeučení. Dále bylo navrženo řešení generování odpovědi u vybraných záměrů.

Vzhledem k povaze navrženého algoritmu lze dle potřeby robota libovolně přeučovat a přidávat, respektive ubírat třídy příslušné jednotlivým záměrům.

Níže následuje výčet navržených záměrů a jejich uplatnění v rámci robotické entity:

- „otoč se“ – ovládání natočení rotační části robota
- „LED“ – kontrola pole LED světel, například zablikání příslušného světla
- „detekce světla“ – detekce zdroje světla
- „detekce objektu“ – rozpoznávání naučených objektů
- „seznam se“ – seznámení s novou osobou pro detekci známé tváře
- „překlad“ – přeložení libovolné promluvy do vybraného jazyka
- „Chat GPT“ – komunikace s chatovací platformou Chat GPT
- „vyhledávání“ – vyhledání požadované informace na internetu

Pro vyhledávání informací na internetu byla využita funkce *Featured Snippet* vyhledávače Google. Tato funkce umožňuje uživatelům získat přesnou odpověď na jejich dotaz přímo na výsledkové stránce vyhledávání, bez nutnosti klikat na odkazy a procházet další stránky. Pro získání těchto odpovědí byla použita technika *web scraping*, která umožňuje automatické stahování a extrakci dat z webových stránek. Pro tento účel byly v jazyce Python použity knihovny *requests*² a *BeautifulSoup*³. Knihovna *requests* umožňuje odesílat HTTP požadavky na server a získávat tak obsah stránek, pomocí knihovny *BeautifulSoup* lze z tohoto obsahu extrahovat požadované informace. Obrázek 6.4 znázorňuje vzorovou situaci klasifikace záměru uživatele a generování odpovědi za základě *Featured Snippet*.



Obrázek 6.4: Příklad užití – záměr „vyhledávání“

²<https://pypi.org/project/requests/>

³<https://pypi.org/project/beautifulsoup4/>

Přeučování neuronových sítí je významným aspektem vývoje umělé inteligence, který poskytuje možnost dosahovat výsledků přesahujících úroveň původního modelu a umožňuje neuronovým sítím přizpůsobit se novým podmínkám a novým datům. Klíčovými faktory úspěšného přeučení neuronové sítě jsou nalezení vhodné struktury sítě, výběr vhodného datasetu a také zvolení vhodného způsobu přeučení.

7.1 Rekapitulace metod

V této práci byla vyvinuta metoda trénování neuronové sítě, která umožňuje plynulé přeučení sítě v reálném čase na základě interakce s uživatelem, metoda byla demonstrována na jednoduché dopředné síti pro klasifikaci záměru. Komunikace s uživatelem byla zprostředkována použitím katedrální platformy *Speechcloud* pro rozpoznávání a syntézu řeči na RPi. Dále byla využita moderní technologie *Sentence Transformer*, která umožňuje převedení rozpoznávaného vstupního textu na vektory reprezentující sémantickou informaci. Díky tomu dokáže neuronová síť správně klasifikovat různé fráze, i když nejsou v tréninkovém datasetu přesně definovány, to například znamená, že když se ve tréninkovém datasetu nachází fráze "Ahoj", bude síť správně klasifikovat i fráze jako „Dobrý den“, „Čau“ nebo „Nazdar“.

Výše zmíněná jednoduchá dopředná neuronová síť pro klasifikaci záměru byla optimalizována pro danou úlohu a pro použití na RPi. Přeučení této sítě se uskutečňuje velice rychle díky transferu parametrů z již předtrénované neuronové sítě na robustním Datasetu 4.2 a *fine tuningem* pouze poslední plně propojené vrstvy.

Zpětnou vazbu poskytuje uživatel pomocí implementovaného webového uživatelského rozhraní – uživatelé mohou vymazat chybné vzorky a zadat pokyn k přetrénování. Díky této zpětné vazbě se síť postupně učí a zlepšuje své

klasifikační schopnosti.

Výsledná aplikace klade důraz na efektivitu a snadnou použitelnost, což zahrnuje také možnosti začít s omezeným počtem tříd nebo vzorků a postupně přidávat další, nebo naopak využít již připravenou síť natrénovanou na Datasetu 4.1 a tuto síť upravit navrženou metodou, aby byla co nejefektivnější pro danou úlohu. Tato flexibilita umožňuje uživatelům přizpůsobit si síť dle svých potřeb a zároveň minimalizuje nároky na tréninková data.

7.2 Shrnutí výsledků

Experiment 5.1. porovnával frameworky Keras a Pytorch pro pevně danou architekturu a dataset za účelem vybrání optimálního frameworku pro zadanou úlohu. Výsledné doby načtení knihoven, načtení modelů a doby predikcí jednoho vzorku pro jsou v Tabulce 5.2.

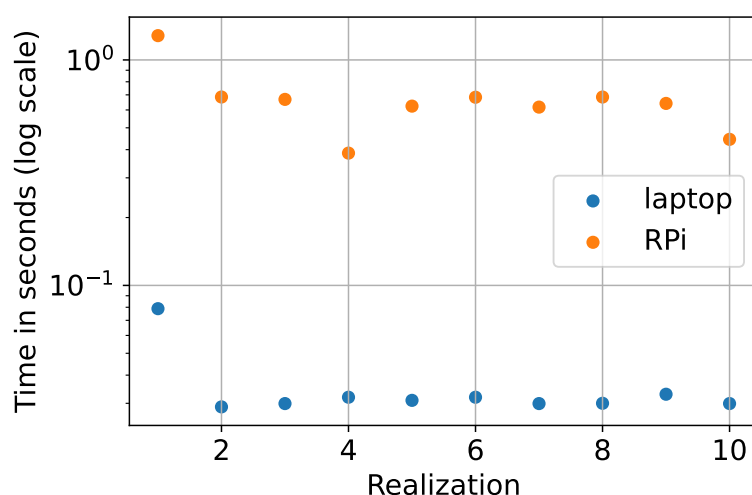
To, že frameworky nemají stejný průběh křivek při trénování modelů na Datasetu 4.1 navzdory tomu, že mají stejné hyperparametry, může být způsobeno tím, že Keras používá *glorot/xavier_uniform* pro váhy a nuly pro bias, zatímco PyTorch používá *kaiming_uniform* pro váhy a jinou uniformní inicializaci pro bias. Dalším důvodem může být různé výchozí nastavení pro optimalizační parametry, to může ovlivnit konvergenci sítě.

Delší doby načtení knihoven frameworku Pytorch mohou být způsobeny několika faktory; velikostí knihoven, jejich komplexností a počtem jejich závislostí.

V případě první predikce jednoho vzorku pomocí PyTorch bylo pozorováno výrazné zpomalení, viz Obrázek 7.1, které může být způsobeno inicializací některých výpočetních zdrojů, jako je cache nebo paměť na grafické kartě, během prvního průchodu. Následující predikce jsou pak rychlejší, protože využívají již optimalizované výpočetní zdroje.

Zanedbáním prvního měření doby predikce by framework Pytorch pro model trénovaný na Datasetu 4.1 dosáhl hodnoty 0.031 ± 0.001 sekund na osobním počítači a hodnoty 0.604 ± 0.105 sekund na RPi, čímž by se ještě více přiblížil rychlosti Kerasu, v případě RPi dokonce dosahuje vyšší rychlosti o 0.04 sekundy. Pro model trénovaný na Datasetu 4.2 byla první měření rovnou zanedbána.

Na základě výsledků tohoto experimentu bylo rozhodnuto, že zvoleným frameworkem pro aplikaci na robotickou entitu bude Pytorch. Delší načtení knihoven nebylo rozhodující, protože probíhá jen jednorázově. Pytorch byl zvolen z toho důvodu, že nabízí větší kontrolu a flexibilitu při vytváření neuronových sítí. Pro situace s většími datovými sadami a výkonnostně náročnějšími úlohami pak



Obrázek 7.1: Doba predikce jednoho vzorku pomocí frameworku Pytorch pro model trénovaný na Datasetu 4.1 – 10 realizací

navíc dosahuje Pytorch vyšší rychlosti než Keras.

Experiment 5.2. se zabýval výběrem optimálních hyperparametrů pro neuronovou síť implementovanou pomocí Pytorch. Byl kladen důraz na přesnost této sítě a zároveň na její rychlost. Jednalo se o ověření konceptu, že danou úlohu dokáže splnit i velice malá neuronová síť.

Velká četnost neuronových sítí s přesností mezi 15 % a 20 % na Obrázku 5.2 je zapříčiněna tím, že se tyto neuronové sítě nedokázaly natrénovat pro daný klasifikační problém (daná kombinace hyperparametrů a jejich hodnoty nebyly vhodné) a pro všechny testovací vzorky predikovaly jednu stejnou třídu (protože v Datasetu 4.1 je 6 tříd).

Z analýzy Tabulky 5.3 plyne, že rychlost trénování neuronové sítě s jednou skrytou vrstvou a počtem neuronů v této vrstvě nižším než 50 není ovlivněna tímto počtem neuronů ve skryté vrstvě. Například třetí a pátý model v tabulce mají až na počet neuronů ve skryté vrstvě stejné hyperparametry a třetí model se 45 neurony ve skryté vrstvě dosahuje lepšího času než pátý model, který jich má 35. Dále lze vyvodit, že pro daný klasifikační problém stačilo neuronové síti menší množství epoch (nejvýše 15) pro to, aby se natrénovala na 100 % a dosahovala dostatečné rychlosti. Naopak batch size nabýval u všech deseti nejlepších neuronových sítí dvou nejvyšších testovaných hodnot – 40 nebo 50.

Celkově lze říci, že všechny modely se natrénovány velmi rychle (v řádu desítek milisekund) díky využití výkonných algoritmů strojového učení a moderních výpočetních schopností počítače. To umožňuje využití lidské interakce

s trénovacím algoritmem, díky krátkým trénovacím časům je možné rychle iterovat a přeučovat modely, což zvyšuje efektivitu procesu a kvalitu modelů.

Experiment 5.3. se zabýval výběrem hyperparametrů neuronové sítě trénované na Datasetu 4.2 a analýzou vlivu architektury sítě (počtu neuronů ve skrytých vrstvách), počtu tříd a počtu vzorků na výslednou přesnost neuronové sítě.


Analýza ukázala, že sítě s větším počtem neuronů ve skrytých vrstvách dosahují lepších výsledků, ale při příliš velké síti hrozí přetrénování (tzv. overfitting). Také bylo zjištěno, že s rostoucím počtem tříd klesá přesnost klasifikace a že se zvyšujícím se počtem vzorků v jednotlivých třídách se zvyšuje přesnost klasifikace, což potvrzuje, že pro trénování neuronových sítí je důležité mít dostatečné množství trénovacích dat.

Experiment 5.4. se zabýval analýzou využití předtrénované sítě.

Při použití druhé modifikace datasetu došlo k tomu, že při validaci v posledních přibližně 100 epochách již dále neklesala validační ztráta modelu 1, to může být způsobeno tím, že model se pro danou úlohu přetrénoval, protože počet vzorků na třídu byl velmi omezený.

Z výsledků experimentu vyplývá, že využití předtrénovaného modelu může být účinným způsobem, jak dosáhnout lepších výsledků než při trénování „od nuly“. Lze také říci, že v případě řídkého datasetu může být zmrazení části modelu lepší volbou než fine-tuning všech vrstev.

Experiment 5.5. se zabýval vyhodnocením metody iterativního přeučování modelu na základě zpětné vazby od uživatele. Byla testována přesnost modelu v závislosti na počtu trénovacích vzorků pro dva různé scénáře.

Výsledky provedeného experimentu naznačují, že přidání malého počtu trénovacích vzorků do třídy, která doposud neobsahovala žádné trénovací vzorky, může výrazně zlepšit kvalitu klasifikace. Tento výsledek byl dosažen díky použití pokročilých metod strojového učení, jako jsou Sentence Transformer a předtrénované neuronové sítě, které umožňují modelům lépe generalizovat na nová data. Výsledky tak potvrzují účinnost těchto metod pro zlepšování výkonnosti klasifikačních modelů. 

V rámci této práce byla vyvinuta metoda trénování neuronové sítě, která umožňuje lidskou interakci s trénovacím algoritmem v reálném čase. Tato metoda byla demonstrována na jednoduché dopředné síti pro klasifikaci záměru, která byla optimalizována pro použití na Raspberry Pi, pro tyto účely byl jako framework pro implementaci sítě experimentálně vybrán *Pytorch*. Pro zprostředkování komunikace s uživatelem byla využita katedrální platforma *Speechcloud*, která umožňuje rozpoznávání a syntézu řeči. Kromě toho byla využita moderní technologie *Sentence Transformer* sloužící k převodu rozpoznaného vstupního textu na vektory reprezentující sémantickou informaci.

Dále bylo implementována webové uživatelské rozhraní, které umožňuje poskytovat zpětnou vazbu na predikce, mazat chybné vzorky a zadat pokyn k přeučení sítě. Tento proces postupného učení umožňuje síti zlepšovat své klasifikační schopnosti a přizpůsobit se potřebám uživatele. Přeučení sítě se uskutečňuje velice rychle díky transferu parametrů z již předtrénované neuronové sítě na robustní datasetu a *fine-tuningem* pouze poslední plně propojené vrstvy.

Tato aplikace je navržena s důrazem na efektivitu a snadnou použitelnost, umožňuje začít s omezeným počtem tříd nebo vzorků a postupně přidávat další, nebo použít již připravenou síť a upravit ji navrženou metodou. Tato flexibilita umožňuje uživatelům přizpůsobit si síť dle svých potřeb a minimalizuje nároky na tréninková data.

8.1 Vylepšení do budoucna

V rámci budoucího rozšíření této práce lze zaměřit pozornost na několik oblastí. Jednou z nich může být rozšíření o parametry záměrů, což by umožnilo sítím rozpoznávat nejen záměr, ale i další potřebné informace pro úspěšné splnění zadaného úkolu. Příkladem může být rozpoznávání cílového jazyka při překladu textu.

Další možností je využití *end2end transformerů* namísto použití SentenceTransformer v kombinaci s dopřednou neuronovou sítí. End2end transformers, jako jsou T5 nebo BERT, by mohly poskytnout lepší výsledky a větší flexibilitu při trénování sítí. Pro porovnání by bylo zajímavé testovat rychlost a kvalitu výsledků při použití různých typů modelů. Kromě toho by mohl být využit ONNX formát pro Pytorch modely. Tento formát by umožnil optimalizaci modelu pro rychlejší predikci.

~~Jinou oblastí, kterou lze při budoucím rozšíření práce zvážit, je testování složitějších architektur neuronové sítě, např. RNN nebo LSTM. Tyto architektury by mohly poskytnout lepší výsledky v úlohách, kde je důležitá posloupnost vstupních dat.~~

Kromě toho by bylo vhodné vytvořit uživatelské rozhraní s rozmanitější funkcionalitou, které by dovolilo uživatelům upravovat hyperparametry sítě. To by umožnilo lepší optimalizaci sítí pro specifické úkoly a zlepšení výkonu sítě při různých typech dat.



Ručně vytvořený dataset



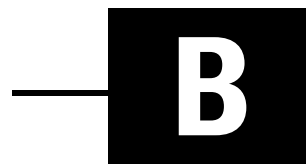
Záměr	Text
POZDRAV	Ahoj
POZDRAV	Dobrý den
POZDRAV	Čau
POZDRAV	Nazdar
POZDRAV	Pozdrav pánbůh
POZDRAV	Zdar
POZDRAV	Dobré ráno
POZDRAV	Dobrý večer
POZDRAV	Dobrej
POZDRAV	Zdravíčko
POKYN	Rozsviť LED
POKYN	Zamávej
POKYN	Otoč se
POKYN	Zapískej
POKYN	Zvedni levou ruku
POKYN	Zvedni pravou ruku
POKYN	Zablikej
POKYN	Jdi vpřed
POKYN	Rozjeď se
POKYN	Zatoč
KALENDÁŘ	Kolik je hodin
KALENDÁŘ	Co mám v plánu zítra odpoledne
KALENDÁŘ	Naplánuj schůzku na pátek ve 3
KALENDÁŘ	Kolikátého je dnes
KALENDÁŘ	Jaký je datum
KALENDÁŘ	Co je za měsíc

(tabulka pokračuje na další stránce)

(pokračování z předchozí stránky)

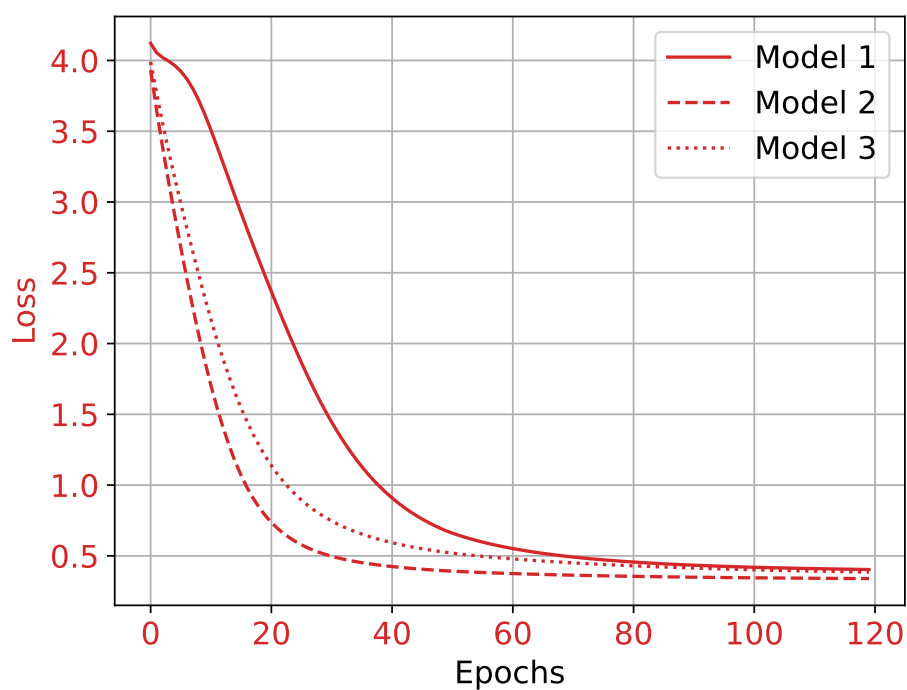
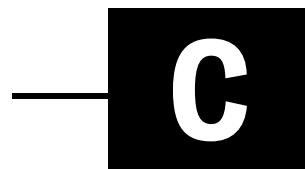
Záměr	Text
KALENDÁŘ	Jaký je rok
KALENDÁŘ	Co je dnes za den
KALENDÁŘ	Kdo má dnes svátek
KALENDÁŘ	Kdo má dnes narozeniny
VYGOOGLI	Kolik je států v Africe
VYGOOGLI	Najdi mi restauraci v Brně
VYGOOGLI	Jaký je počet obyvatel v Plzni
VYGOOGLI	Kde koupit bitcoin za CZK a jaký je aktuální kurz
VYGOOGLI	Kdo je prezident ve Francii
VYGOOGLI	Kde budou další olympijské hry
VYGOOGLI	Jak vybrat běžky
VYGOOGLI	Jak napsat životopis
VYGOOGLI	Jak rychle zhubnout
VYGOOGLI	Jak vydělat peníze
POZNEJ	Co to je?
POZNEJ	Jaký je to předmět
POZNEJ	Poznej předmět
POZNEJ	Poznej, co to je
POZNEJ	Urči, o co se jedná
POZNEJ	Jaká je to věc
POZNEJ	Co to mám
POZNEJ	Co to držím
POZNEJ	Co vidíš
POZNEJ	Co je tohle
STOP	Vypnout
STOP	Ukončit
STOP	Zastav
STOP	Přestaň
STOP	Konec
STOP	Stop
STOP	Dost
STOP	Vypni se
STOP	Ukonči se
STOP	Zastav se

Testované hyperparametry pro neuronovou síť na RPi

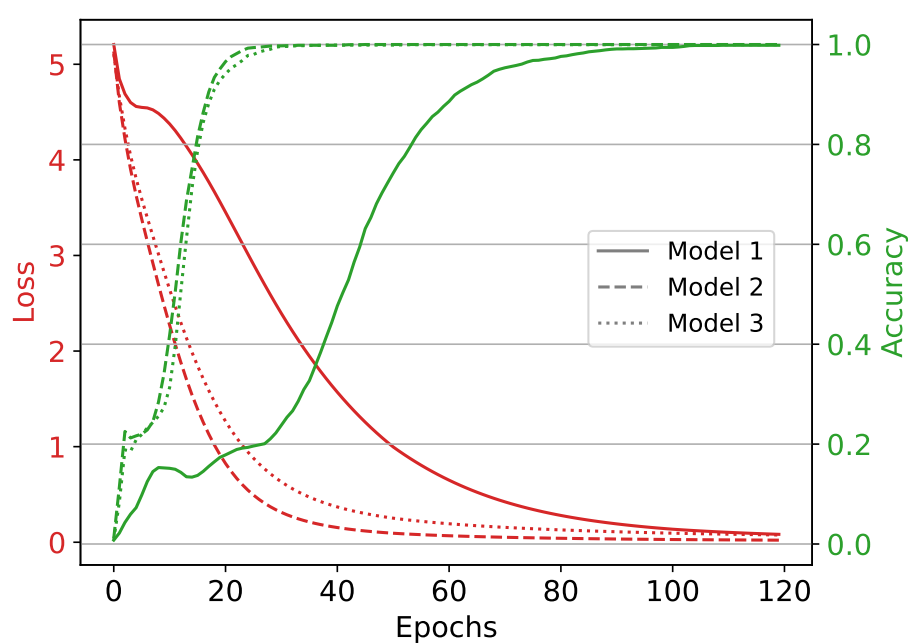


Parametr	Hodnoty
Počet neuronů ve skryté vrstvě	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Aktivační funkce	(Sigmoid, Softmax)
Learning rate	0.1, 0.01, 0.001, 0.0001
Velikost dávky	5, 10, 20, 30, 40, 50
Počet epoch	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Optimizér	SGD, Adam

Grafy k experimentu 5.4



Obrázek C.1: Vývoj ztráty při validaci modelů pro 60 tříd pro prvních 120 epoch



Obrázek C.2: Vývoj přesnosti a ztráty při trénování modelů pro 130 tříd pro prvních 120 epoch

Scénáře pro Experiment 5.5



D.1 Scénář 1

Záměr	Text
POZDRAV	Ahoj
POKYN	Rozsviť LED
KALENDÁŘ	Kolikátého je dnes
VYGOOGLI	Kolik je států v Africe
POZNEJ	Co to je?
STOP	Ukončit
POZDRAV	Dobrý den
POZDRAV	Čau
POKYN	Otoč se
POKYN	Zapískej
KALENDÁŘ	Jaký je datum
KALENDÁŘ	Co je za měsíc
VYGOOGLI	Najdi mi restauraci v Brně
VYGOOGLI	Jaký je počet obyvatel v Plzni
POZNEJ	Poznej, co to je
POZNEJ	Urči, o co se jedná
STOP	Přestaň
STOP	Konec
POZDRAV	Nazdar
POKYN	Zablikej
KALENDÁŘ	Jaký je rok
VYGOOGLI	Kdo je prezident ve Francii
POZNEJ	Co to mám
STOP	Stop

(tabulka pokračuje na další stránce)

(pokračování z předchozí stránky)

Záměr	Text
POZDRAV	Zdravíčko
POKYN	Zatoč
KALENDÁŘ	Co je dnes za den
VYGOOGLI	Kde budou další olympijské hry
POZNEJ	Co je tohle
STOP	Ukonči se

D.2 Scénář 2

Záměr	Text
POZDRAV	Nazdar
KALENDÁŘ	Jaký je rok
VYGOOGLI	Kdo je prezident ve Francii
STOP	Stop
POZNEJ	Co to je?
POZNEJ	Jaký je to předmět
POZNEJ	Poznej, co to je
POZNEJ	Urči, o co se jedná
POZNEJ	Jaká je to věc
POKYN	Rozsviť LED
POKYN	Zamávej
POKYN	Otoč se
POKYN	Zapískej
POKYN	Zvedni levou ruku
CHAT	Jak se máš?
CHAT	Jak se jmenuješ?
CHAT	Kdo jsi?
CHAT	Máš rád čokoládu?
CHAT	jaký je účel existence
CHAT	co je smyslem života
xc	

Bibliografie

- [1] Evan Ackerman a Erico Guizzo. *Wizards of ROS: Willow Garage and the making of the Robot Operating System*. 2022. URL: <https://spectrum.ieee.org/wizards-of-ros-willow-garage-and-the-making-of-the-robot-operating-system>.
- [2] Saugat Bhattarai. *What is gradient descent in machine learning?* 2018. URL: <https://saugatbhattarai.com np/what-is-gradient-descent-in-machine-learning/>.
- [3] James Briggs. *Sentence transformers and embeddings*. 2023. URL: <https://www.pinecone.io/learn/sentence-embeddings/>.
- [4] Jason Brownlee. *How to configure the learning rate when training deep learning neural networks*. 2019. URL: <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/>.
- [5] Guillaume Desagulier. *Word embeddings: The (very) basics*. 2018. URL: <https://corpling.hypotheses.org/495>.
- [6] *Designing your neural networks*. URL: <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee a Kristina Toutanova. „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: *arXiv preprint arXiv:1810.04805* (2018).
- [8] Sanket Doshi. *Various optimization algorithms for training neural network*. 2020. URL: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>.
- [9] Charles T. Hemphill, John J. Godfrey a George R. Doddington. „The ATIS Spoken Language Systems Pilot Corpus“. In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*. 1990. URL: <https://aclanthology.org/H90-1021>.
- [10] Nagesh Singh Chauhan. *Loss functions in Neural Networks*. 2021. URL: <https://www.theaidream.com/post/loss-functions-in-neural-networks>.

- [11] Francois Chollet et al. *Keras*. 2015. URL: <https://github.com/fchollet/keras>.
- [12] Jongkil Kim, Jonathon M. Smereka, Calvin Cheung, Surya Nepal a Marthie Grobler. „Security and Performance Considerations in ROS 2: A Balancing Act“. In: *CoRR* abs/1809.09566 (2018). arXiv: 1809.09566. URL: <http://arxiv.org/abs/1809.09566>.
- [13] Stefan Larson et al. „An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction“. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, lis. 2019, s. 1311–1316. DOI: 10.18653/v1/D19-1131. URL: <https://aclanthology.org/D19-1131>.
- [14] Jan Lehecka a Jan Svec. „Comparison of Czech Transformers on Text Classification Tasks“. In: *CoRR* abs/2107.10042 (2021). arXiv: 2107.10042. URL: <https://arxiv.org/abs/2107.10042>.
- [15] Lian Meng a Minlie Huang. „Dialogue Intent Classification with Long Short-Term Memory Networks“. In: led. 2018, s. 42–50. ISBN: 978-3-319-73617-4. DOI: 10.1007/978-3-319-73618-1_4.
- [16] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán a Ángel Fernández-Leal. „Human-in-the-loop machine learning: a state of the art“. In: *Artificial Intelligence Review* 56 (srp. 2022). DOI: 10.1007/s10462-022-10246-w.
- [17] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.
- [18] Michael A Nielsen. *Neural networks and deep learning*. Sv. 25. Determination press San Francisco, CA, USA, 2015.
- [19] Adam Paszke et al. „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems* 32. Curran Associates, Inc., 2019, s. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [20] Morgan Quigley, Brian Gerkey a William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. 1st. O'Reilly Media, Inc., 2015. ISBN: 1449323898.

- [21] Morgan Quigley et al. „ROS: an open-source Robot Operating System“. In: *ICRA Workshop on Open Source Software*. 2009.
- [22] Alec Radford, Karthik Narasimhan, Tim Salimans a Ilya Sutskever. „Improving language understanding by generative pre-training“. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, s. 3568–3577.
- [23] Pranoy Radhakrishnan. *What are hyperparameters ? and how to tune the hyperparameters in a deep neural network?* 2017. URL: <https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a>.
- [24] Suman Ravuri a Andreas Stoicke. „A comparative study of neural network models for lexical intent classification“. In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 2015, s. 368–374. DOI: 10.1109/ASRU.2015.7404818.
- [25] Nils Reimers a Iryna Gurevych. „Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks“. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, lis. 2019, s. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: <https://aclanthology.org/D19-1410>.
- [26] Kurnia Rendy. *Tuning the hyperparameters and layers of neural network deep learning*. 2022. URL: <https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/>.
- [27] David E. Rumelhart, Geoffrey E. Hinton a Ronald J. Williams. „Learning representations by back-propagating errors“. In: *Nature* 323 (1986), s. 533–536.
- [28] Nikita Sharma. *Exploring optimizers in machine learning*. 2021. URL: <https://heartbeat.comet.ml/exploring-optimizers-in-machine-learning-7f18d94cd65b>.
- [29] Sagar Sharma, Simone Sharma a Anidhya Athaiya. „Activation functions in neural networks“. In: *Towards Data Sci* 6.12 (2017), s. 310–316.
- [30] *SpeechTech* — [speechtech.cz](https://www.speechtech.cz/). <https://www.speechtech.cz/>. [Accessed 25-Feb-2023].
- [31] Stanford Artificial Intelligence Laboratory et al. *actionlib*. 2018. URL: <http://wiki.ros.org/actionlib>.

- [32] Ilya Sutskever, James Martens, George Dahl a Geoffrey Hinton. „On the importance of initialization and momentum in deep learning“. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. Sanjoy Dasgupta a David McAllester. Sv. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, s. 1139–1147. URL: <https://proceedings.mlr.press/v28/sutskever13.html>.
- [33] Jan Švec, Petr Neduchal a Marek Hruž. „Multi-modal communication system for mobile robot“. In: *IFAC-PapersOnLine* 55.4 (2022). 17th IFAC Conference on Programmable Devices and Embedded Systems PDES 2022 — Sarajevo, Bosnia and Herzegovina, 17-19 May 2022, s. 133–138. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2022.06.022>. URL: <https://www.sciencedirect.com/science/article/pii/S2405896322003378>.
- [34] Nima Tajbakhsh et al. „Convolutional neural networks for medical image analysis: Full training or fine tuning?“ In: *IEEE transactions on medical imaging* 35.5 (2016), s. 1299–1312.
- [35] Ashish Vaswani et al. „Attention is all you need“. In: *Advances in Neural Information Processing Systems*. 2017, s. 5998–6008.
- [36] Eva Volná. *Neuronové sítě* 1. 2008.
- [37] Yufan Wang, Jiawei Huang, Tingting He a Xinhui Tu. „Dialogue intent classification with character-CNN-BGRU networks“. In: *Multimedia Tools and Applications* 79 (ún. 2020). DOI: 10.1007/s11042-019-7678-1.
- [38] Vishal Yathish. *Loss functions and their use in neural networks*. 2022. URL: <https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9>.
- [39] Aston Zhang, Zachary C. Lipton, Mu Li a Alexander J. Smola. „Dive into Deep Learning“. In: *CoRR* abs/2106.11342 (2021). arXiv: 2106.11342. URL: <https://arxiv.org/abs/2106.11342>.
- [40] XueFei Zhou. „Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation“. In: *Journal of Physics: Conference Series* 1004.1 (2018), s. 012028. DOI: 10.1088/1742-6596/1004/1/012028. URL: <https://dx.doi.org/10.1088/1742-6596/1004/1/012028>.

Seznam obrázků

2.1	Diagram architektury neuronové sítě (převzato z [6])	7
2.2	Aktivační funkce	9
2.3	Vliv učícího koeficientu při gradientním sestupu (převzato z [2])	13
2.4	Vektorový prostor sedmi slov ve třech kontextech (převzato z [5]) . . .	16
2.5	Diagram transfer learningu a fine-tuningu (převzato z [39])	18
2.6	Diagram architektury SpeechCloudu (převzato z [33])	21
3.1	Napojení Sentence Transformeru na neuronovou síť	25
3.2	Propojení jednotlivých částí v rámci celé aplikace	27
3.3	Celkový náhled na uživatelské rozhraní	28
3.4	Zvýraznění predikovaného záměru	28
3.5	Korekce predikovaného záměru na záměr „Nová třída“	29
3.6	Dokončení přeučení, možnost odstranění chybných vzorků	30
5.1	Průběh trénování neuronové sítě v nástrojích Pytorch a Keras pro prvních 12 epoch	34
5.2	Histogram četností modelů v závislosti na jejich přesnosti	36
5.3	Boxplot doby trénování jednotlivých modelů	36
5.4	Průběh trénování neuronové sítě s optimálními hyperparametry pro 10 realizací	37
5.5	Přesnost neuronové sítě v závislosti na její architektuře (počtu neuronů ve skrytých vrstvách) a počtu tříd, počty vzorků ve třídách byly 100 %	40
5.6	Přesnost neuronové sítě v závislosti na počtu tříd a počtu vzorků na třídu, pro zafixovanou architekturu [512,256]	41
5.7	Vývoj přesnosti a ztráty při trénování modelů pro 60 tříd pro prvních 120 epoch	42
5.8	Vývoj ztráty při validaci modelů pro 130 tříd	43
5.9	Vývoj přesnosti pro jednotlivé třídy při iterativním přidávání vzorků v rámci scénáře 1	45

5.10	Vývoj přesnosti pro jednotlivé třídy při iterativním přidávání vzorků v rámci scénáře 2	46
6.1	Hardwarové propojení jednotlivých součástí použitých na robotické entitě	48
6.2	ROS architektura robotické entity	48
6.3	Fotografie výsledné robotické entity	49
6.4	Případ užití – záměr „vyhledávání“	50
7.1	Doba predikce jednoho vzorku pomocí frameworku Pytorch pro model trénovaný na Datasetu 4.1 – 10 realizací	53
C.1	Vývoj ztráty při validaci modelů pro 60 tříd pro prvních 120 epoch . .	61
C.2	Vývoj přesnosti a ztráty při trénování modelů pro 130 tříd pro prvních 120 epoch	62

Seznam tabulek

4.1	Statistiky datasetu 1	31
4.2	Statistiky datasetu 2	32
5.1	Hyperparametry neuronových sítí použitých v experimentu pro výběr frameworku	33
5.2	Doby načtení knihoven, načtení modelů a predikcí jednoho vzorku pro frameworky Pytorch a Keras (v sekundách).	35
5.3	Hyperparametry deseti nejrychlejších neuronových sítí a jejich doby trvání trénování, koeficient učení byl ve všech případech 0.01 a optimizér Adam	37
5.4	Hyperparametry neuronové sítě pro klasifikaci záměru trénované na Datasetu 4.2	38
5.5	Doby trénování modelů pro různý počet tříd	44