



**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

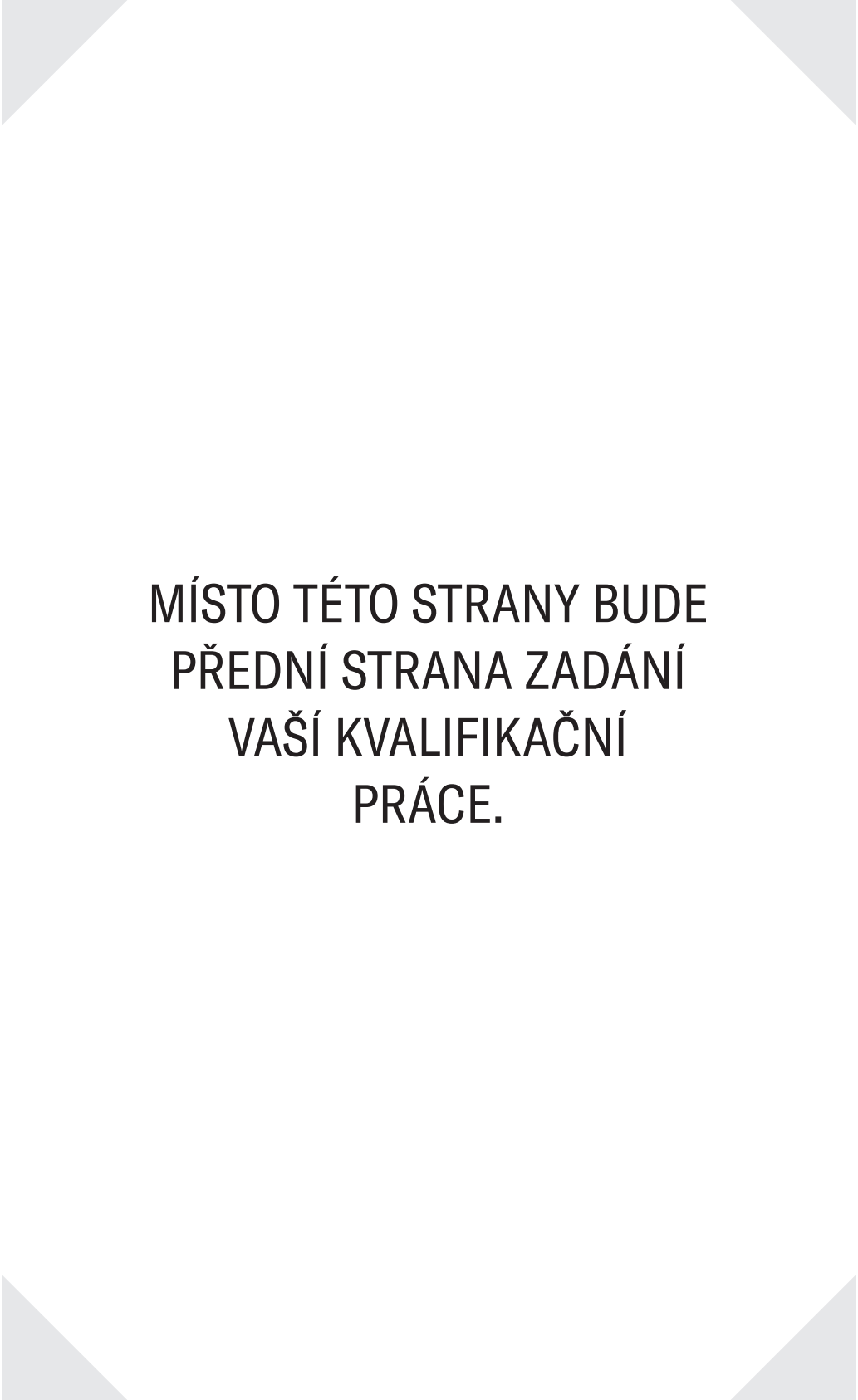
Ing. Bulín Martin, M.Sc.

© Vladimíra Kimlová, 2023.

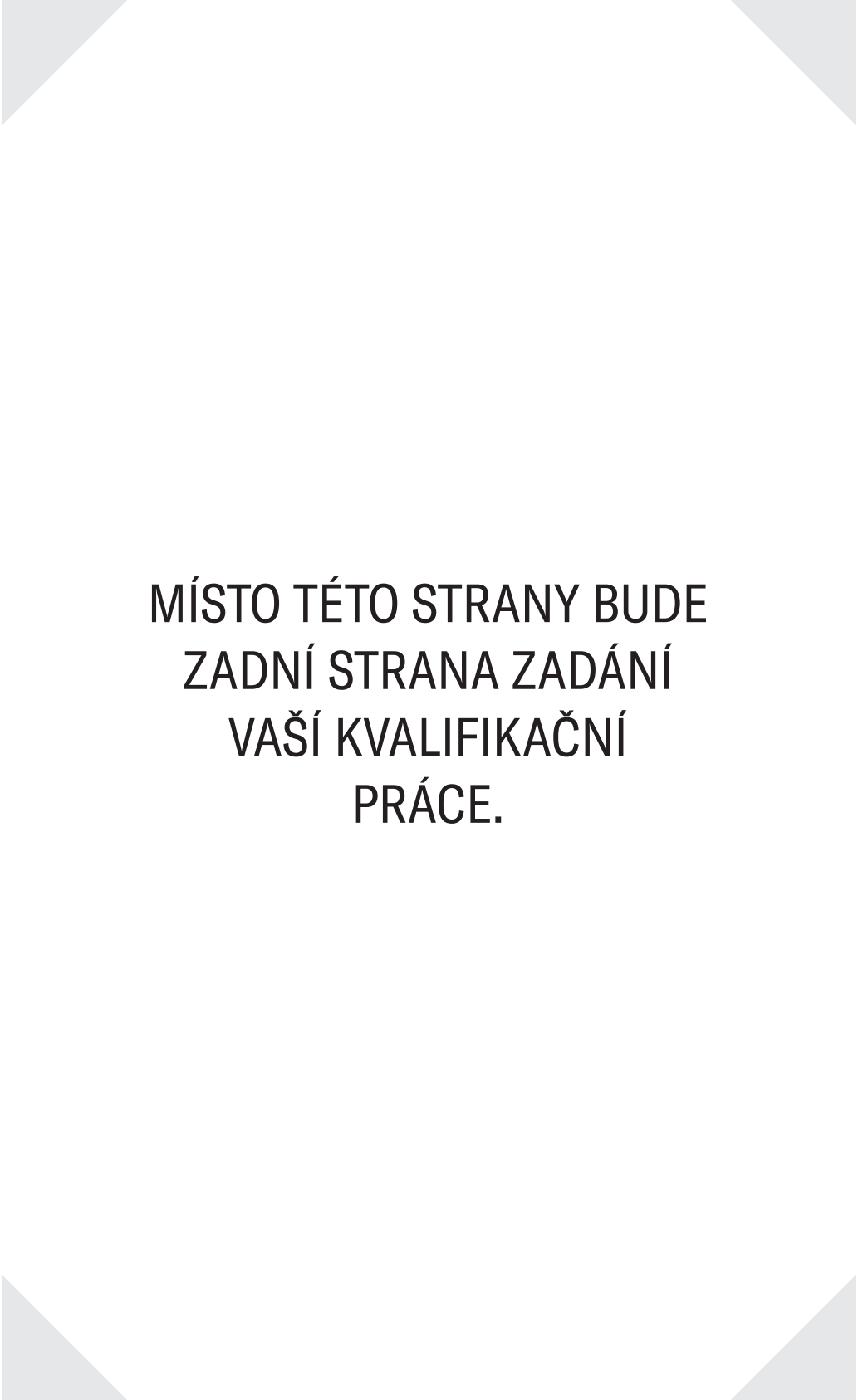
Všechna práva vyhrazena. Žádná část tohoto dokumentu nesmí být reprodukována ani rozšiřována jakoukoli formou, elektronicky či mechanicky, fotokopírováním, nahráváním nebo jiným způsobem, nebo uložena v systému pro ukládání a vyhledávání informací bez písemného souhlasu držitelů autorských práv.

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

Český abstrakt

Abstract

Anglický abstrakt

Klíčová slova

dopředná neuronová síť • Human in the loop • přetrénování (Retraining) neuronové sítě • klasifikace záměru

Poděkování

Nejprve bych chtěla poděkovat panu Ing. Martinu Bulínovi, M.Sc, mému 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	3
1.1	Cíle práce	4
1.2	Současný stav problematiky	4
1.3	Struktura práce	4
2	Použité metody a technologie	5
2.1	Dopředná neuronová síť	5
2.1.1	Hyperparametry	10
2.2	Sentence transformery	13
2.2.1	FERNET-C5	15
2.3	Fine-tuning	15
2.4	ROS (Robot Operating System)	16
2.5	Speechcloud	18
2.6	Komunikační protokoly	19
2.6.1	HTTP	19
2.6.2	MQTT	20
2.6.3	WebSockets	20
3	Navržené řešení	23
3.1	Neuronová síť pro klasifikaci záměru	23
3.2	Implementace	24
3.3	Uživatelské rozhraní	25
3.4	Robot.v1	28

3.5	Přetrénovací smyčka	28
4	Data	29
4.1	Ručně vytvořený dataset	29
4.2	Obsáhlejší dataset	29
5	Experimenty a výsledky	31
5.1	Výběr frameworku (Keras vs. PyTorch)	31
5.2	Výběr velikosti sítě v závislosti na přesnosti a rychlosti modelu . .	33
5.3	Analýza schopnosti sítě natrénovat se v závislosti na počtu dat a tříd	36
5.4	Porovnání Fine-tuningu a tréninku od nuly (Training From Scratch)	39
6	Aplickace	43
7	Diskuze	45
7.1	Rekapitulace metod	45
7.2	Shrnutí výsledků	45
8	Závěr	49
8.1	Future Work	49
A	Ručně vytvořený dataset	51
B	Testované hyperparematry pro neuronovou síť na RPi	53
C	Grafy k experimentu 5.4	55
	Bibliografie	57
	Seznam obrázků	61
	Seznam tabulek	63

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 offline trénink. 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 ztrácí přehled o tom, na čem je poslední model naučený. Tyto problémy jsou zvláště patrné v případech, kdy je potřeba se rychle adaptovat na nové podmínky a naučit se novému chování, jako například v oblasti *human-robot interaction* a online učení se novým věcem. Proto se v takových situacích často 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 přetrénování modelu. Human-in-the-loop přístup se využívá v mnoha oblastech, včetně strojového učení, robotiky, medicíny, průmyslu, autonomních vozidel a online reklamy.

Tato práce nabízí metodu pro *human-in-the-loop online přetrénování (Retraining)*. Cílem je umožnit uživateli směřovat učení neuronové sítě pomocí zpětné vazby, což umožňuje neuronové síti se učit nové věci online 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 SentenceTransformer. V rámci experimentů byly analyzovány různé frameworky pro implementaci sítě, s důrazem na výběr optimálního frameworku (v tomto případě PyTorch). Dále byla zkoumána optimální velikost sítě pro daný problém a pro nasazení na Raspberry Pi. Byl také analyzován vliv přetrénování (Retrainingu) na výsledky klasifikace, včetně použití *fine-tuningu* pro inicializaci sítě. Výsledky experimentů ukázaly, ...TODO

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

Cílem této práce je nastudovat problematiku klasifikace neuronovými sítěmi, včetně nejpoužívanějších typů sítí a knihoven Keras a PyTorch. Dále se seznámit s platformou Speechcloud a naučit se používat její služby pro rozpoznávání a syntézu řeči. Dále je cílem navrhnout neuronovou síť pro klasifikaci záměru ze vstupní promluvy a řešení pro generování odpovědí pro vybrané kategorie (záměry). Prozkoumat také možnosti automatického přetrénování klasifikátoru na nových datech a nakonec aplikovat vyvinutou metodu včetně hlasové interakce na reálnou robotickou entitu.

1.2 Současný stav problematiky

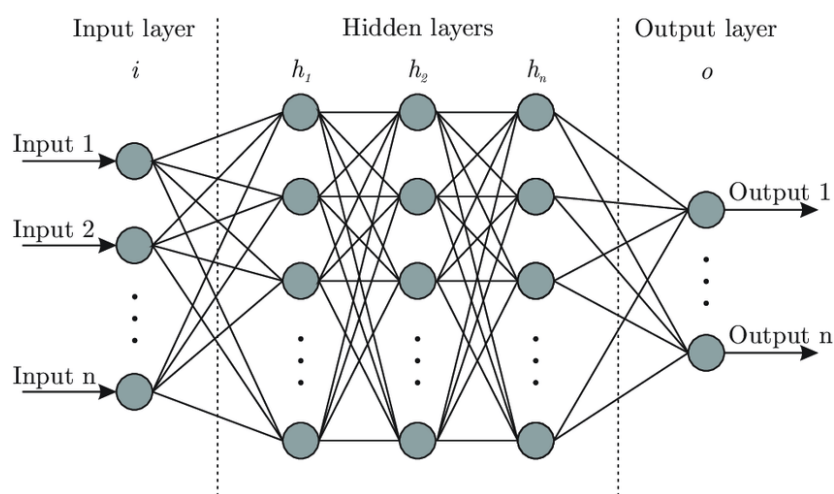
1.3 Struktura práce

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 je vektor vstupních dat x . Aktivace neuronů ve vrstvě l jsou označeny jako vektor $a^{(l)}$ [12].

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 $a^{(0)} = x$ [12].

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 [19]:

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

Sigmoidní aktivační funkce má tvar [19]:

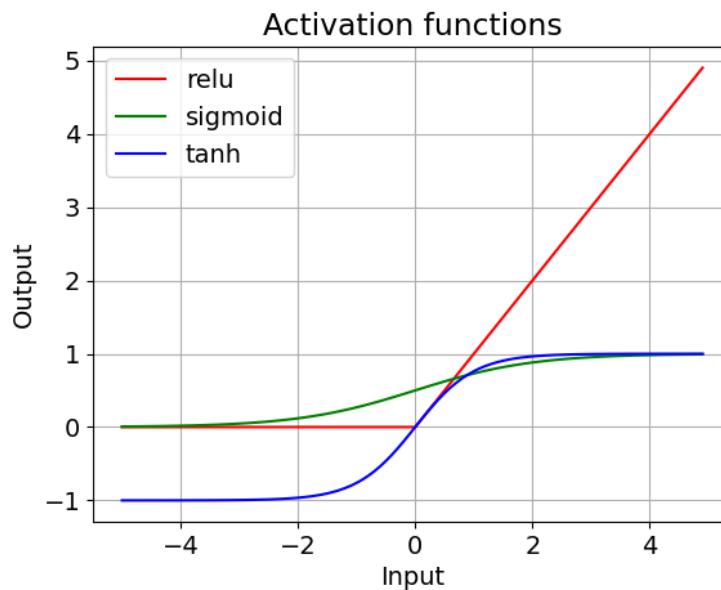
$$f_2(z) = \frac{1}{1 + e^{-z}} \quad (2.4)$$

Tanh aktivační funkce má tvar [19]:

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

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 [19]:

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



Obrázek 2.2: Aktivační funkce

Aktivační funkce ReLU, Sigmoid a Tanh jsou zobrazeny 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“ [17].

Algoritmus se skládá ze tří hlavních částí [26]:

- 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 [12]:

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

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 [27]:

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

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 [27]:

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

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 [8]:

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

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 [8]:

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

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 [12]:

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

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

kde δ_j^l je tzv. chyba neuronu j ve vrstvě l , která se vypočítá jako [12]:

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

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 [12]:

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

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 [29]:

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

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

kde η je učicí koeficient (Learning Rate), což je nastavitelný hyperparametr optimalizačního algoritmu, který určuje velikost kroku, kterým se algoritmus posouvá směrem k minimu ztrátové funkce během trénování [11]. 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 [26].

2.1.1 Hyperparametry

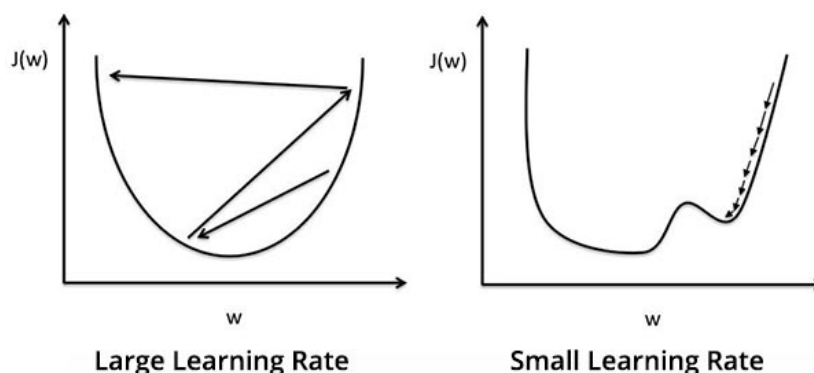
Hyperparametry neuronových sítí jsou proměnné, které určují strukturu sítě a způsob, jakým je síť trénována. Tyto hyperparametry jsou nastaveny před trénováním sítě, tedy před optimalizací vah a biasu [15].

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 [16]. 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 [15].
- *Učící koeficient (Learning Rate)*: Učící koeficient 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 [15]. Na Obrázku 2.3 je ilustrován průběh ztrátové funkce J v závislosti na hodnotě vah w vlevo pro velký, vpravo pro malý učící koeficient. 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 aktuali-

zaci nových. Rovnice 2.17 a 2.18 s využitím této metody vypadají následovně [22]:

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

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

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

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

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.

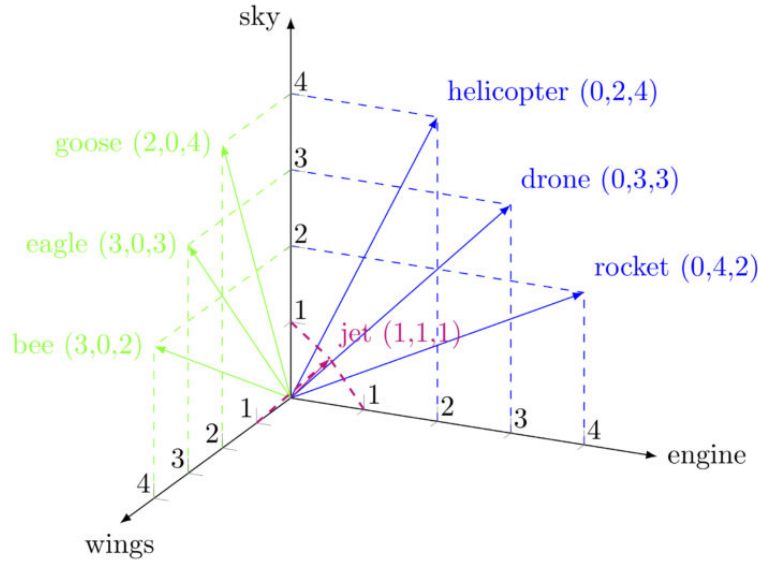
- *Velikost dávky (Batch Size)*: Velikost dávky je počet vzorků, které jsou síti předkládány současně v rámci jedné iterace aktualizace parametrů [15]. Velikost dávky ovlivňuje rychlost trénování sítě, větší velikost dávky obvykle může zvýšit rychlost trénování.
- *Počet epoch*: Počet epoch značí, kolikrát je celý trénovací dataset předložen neuronové síti během tréninku [15]. Příliš málo epoch může vést k nedotrénování sítě, zatímco příliš mnoho epoch může vést k přeučení sítě [16].
- *Optimalizační algoritmus*: Optimizér je algoritmus používaný k úpravě vlastností neuronové sítě, jako jsou váhy a učící koeficient, s cílem snížit chybovost [7]. Existuje mnoho druhů optimizérů v oblasti strojového učení, některé z nejznámějších jsou:
 - *Gradientní sestup*: 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 [7].
 - *Stochastický gradientní sestup (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 [7].

- *Adagrad*: Tento optimizér, narozdíl od všech výše zmíněných, mění učicí koeficient 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 učicí koeficient má za následek pomalé trénování [7].
- *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 učicím koeficientem. Nevýhodou je opět výpočetní náročnost [7].
- *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 [18].
- *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 [18].

2.2 Sentence transformery

Sentence Transformer je model strojového učení, který dokáže transformovat vstupní textové sekvence (např. věty) do vektorů zachovávajících 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 zobrazena na Obrázku 2.4.

Architektura zvaná transformer byla poprvé představena v roce 2017 v článku Attention Is All You Need. 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 transformeru je me-



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

chanismus pozornosti (Attention), který umožňuje modelu se zaměřit na důležité části vstupní sekvence a ignorovat méně významné informace [25].

Vzorec pro výpočet pozornosti (Scaled Dot-Product Attention) vypadá následovně [25]:

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

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 mechanismů pozornosti pracujících společně. Místo toho, aby byl použit jeden set vah pro výpočet pozornosti, je použito více setů vah, tzv. heads. Každý head slouží jako zvláštní lineární

transformace na vstupním prostoru a výsledky jsou následně kombinovány. Použitím více attention heads se umožňuje reprezentace několika souborů vztahů současně [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 nebo GPT, které jsou široce dostupné pro použití v různých aplikacích.

2.2.1 FERNET-C5

FERNET-C5 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 pozornostních hlav (Attention Heads) a skrytou velikost 768 neuronů. Na rozdíl od BERT modelů od Googlu byla použita tokenizace SentencePiece místo interní tokenizace WordPiece od Googlu [10].

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

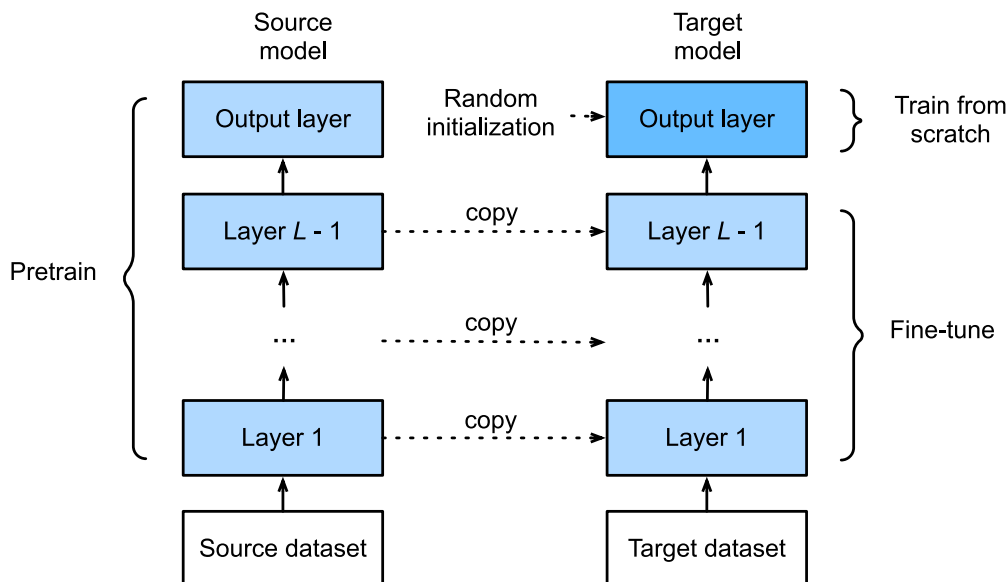
2.3 Fine-tuning

Fine-tuning neuronových sítí je proces přizpůsobení již natrénované sítě na nový úkol, který může být odlišný od původního úkolu, pro který byla síť trénována. Tento proces spočívá v použití vah již trénované sítě jako počátečních hodnot pro trénování nové sítě [24]. Při použití fine-tuningu se předtrénovaný model přizpůsobuje nové úloze tím, že se upraví váhy neuronové sítě pomocí trénovacích dat specifických pro novou úlohu.

Fine-tuning začíná kopírováním (transferem) vah z již předtrénované sítě do nové sítě, která má být trénována. Často je 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 [24]. Její inicializace je zpravidla náhodná [28]. 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 [28]. Obecně lze provádět i fine-tuning čistě T5 transformer sítě, kde se zachová kompletní architektura a jen se provede fine-tuning na nových doménových datech k naučení nové úlohy.

Nástin fungování fine-tuningu při nahrazení poslední plně propojené vrstvy je zobrazen na Obrázku 2.5.



Obrázek 2.5: Diagram fine-tuningu (převzato z [28])

Fine-tuning je velmi užitečný nástroj, protože umožňuje 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 Univerzitě v Stanfordu. Poskytuje knihovny a nástroje umožňující snadno vytvářet a sdílet software pro řízení robotů [1, 13].

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 desítkami tisíci uživateli po celém světě v oblastech od

domácích projektů v rámci koníčků, po velké průmyslové automatizované systémy [13]. 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 [9].

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 oblouky (Arcs) [13, 14].

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 [13, 14].

Uzel odesílá zprávu publikováním na dané téma. Uzel, který má zájem o určitý druh dat, se přihlásí k příslušnému tématu. 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 směrování 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 [13, 14].

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 [21].

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 [13, 14].
- *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

děny samostatnými programy, což podporuje vytváření nových, vylepšených implementací [13, 14].

- *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 [13, 14].
- *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 [13, 14].
- *Zdarma a Open-Source*: Úplný zdrojový kód ROS je veřejně dostupný [13, 14].

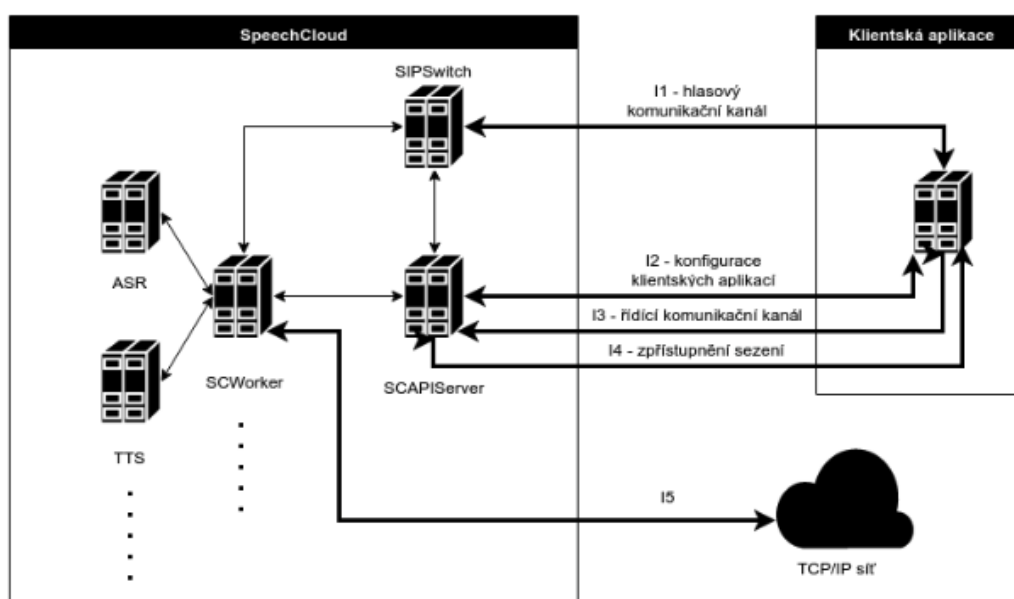
2.5 Speechcloud

SpeechCloud je platforma pro zpracování řeči a analýzu hlasu vyvinutá společností SpeechTech a Katedrou kybernetiky na ZČU. Umožňuje například automatický přepis diktátu do psané podoby, generování vysoce přirozené řeči ze zadaných textů, ověření a verifikaci osob na základě jedinečných charakteristik jejich hlasu a hlasovou komunikaci mezi člověkem a počítačem. Platforma podporuje několik jazyků včetně češtiny a slovenštiny. SpeechCloud je navržen tak, aby umožňoval snadné integrování s různými aplikacemi a systémy třetích stran. [20]. Architektura této platformy je zobrazena na Obrázku 2.6.

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 řídicí zprávy pomocí WebSocket připojení jako JSON zakódované zprávy. 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 [23].

Základní SpeechCloud technologie lze označit jako:

- Automatické rozpoznávání řeči (ASR) – SpeechCloud využívá pokročilé ASR technologie, které umožňují převádět mluvenou řeč na text. Tato technologie



Obrázek 2.6: Diagram architektury SpeechCloudu (převzato z [23])

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 odpovědi 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.

3

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.

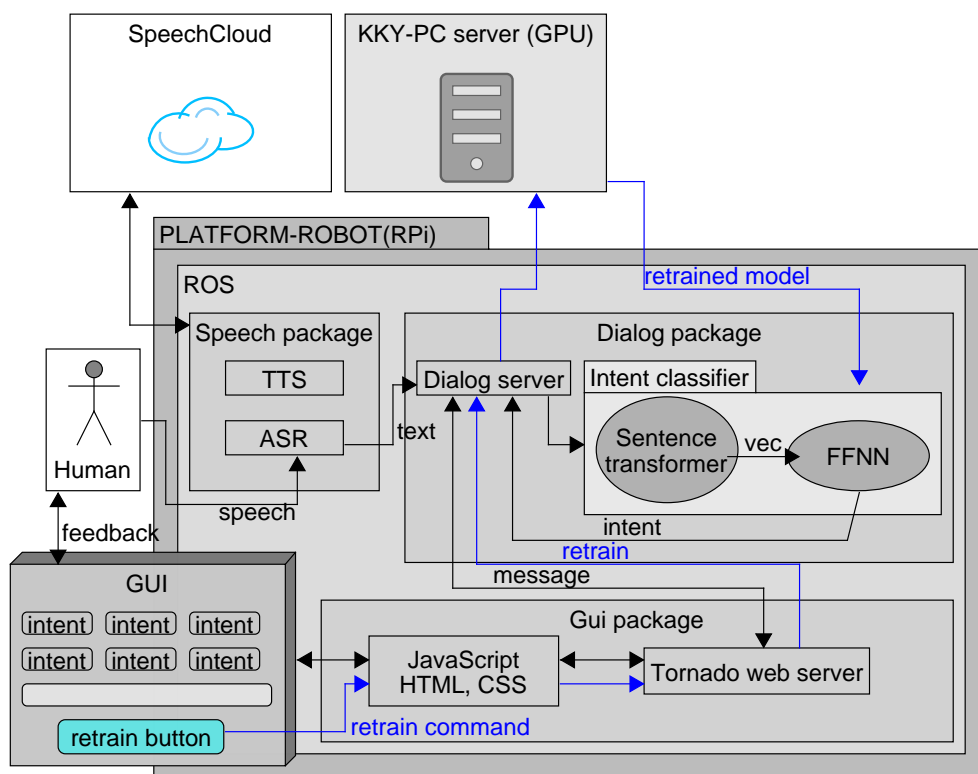
3.2 Implementace

Diagram na Obrázku 3.2 zobrazuje 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 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 websocketů.
- 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 přetrénování modelu – to je prováděno na výkonnějším stroji než RPi. Přetrénování 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 websocketů.

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 3.1), jejím výstupem je predikovaný záměr uživatele. Ten je pomocí zprávy zaslán do webového serveru a příslušně zobrazen na webové stránce. Uživatel má následně možnost na tuto predikci 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

přetrénování 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 přetrénování modelu. Tato situace je v diagramu znázorněna modrou barvou.

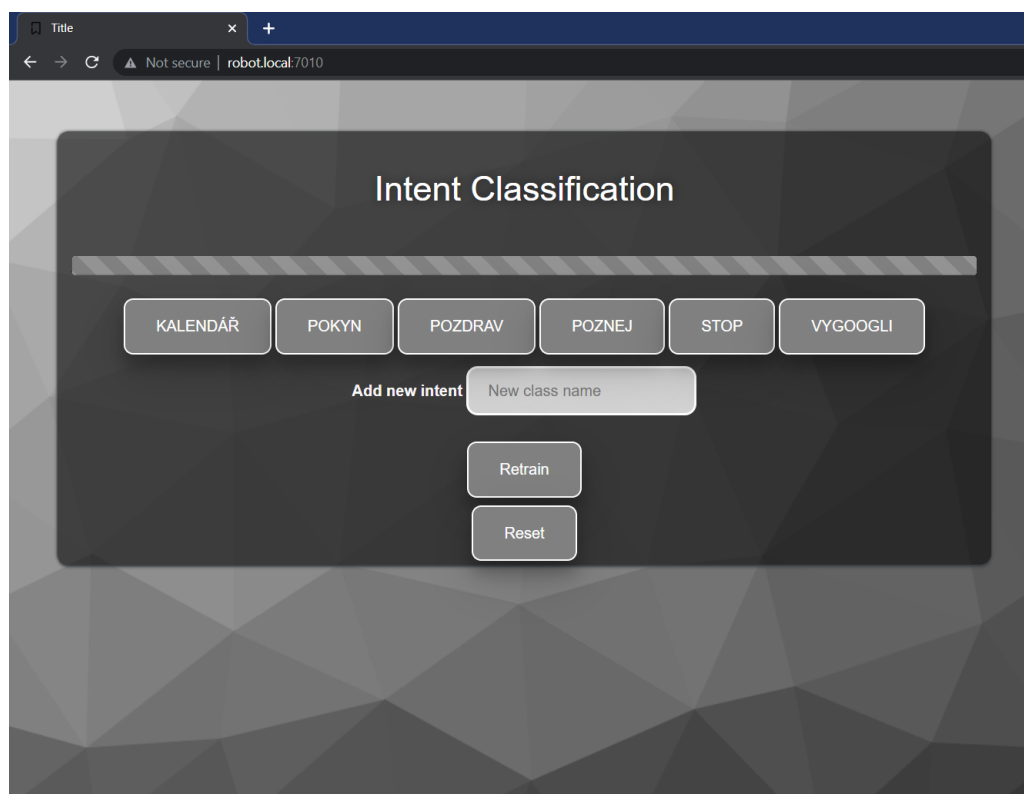


Obrázek 3.2: Nastínění propojení jednotlivých částí v rámci celé aplikace

3.3 Uživatelské rozhraní

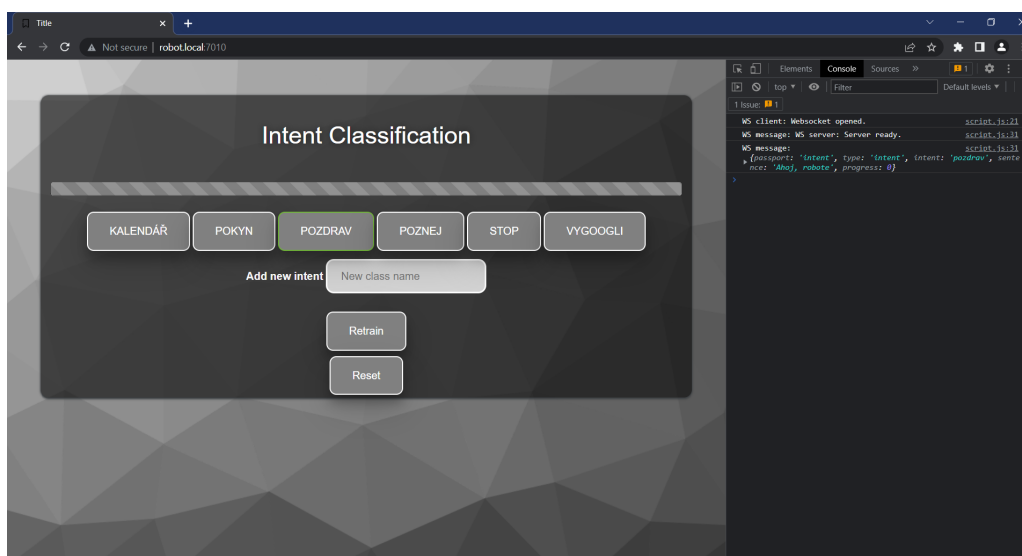
Samotné uživatelské rozhraní je zobrazeno jednak na displeji robotické entity, jednak ve webovém prohlížeči na adrese `robot.local:7010`. Nyní bude nastíněna funkčnost uživatelského rozhraní. Na Obrázku 3.3 je zobrazen základní vzhled webové stránky v prohlížeči, pro větší názornost budou některé následující obrázky webové stránky obsahovat navíc i výpis do konzole.

3. Navržené řešení



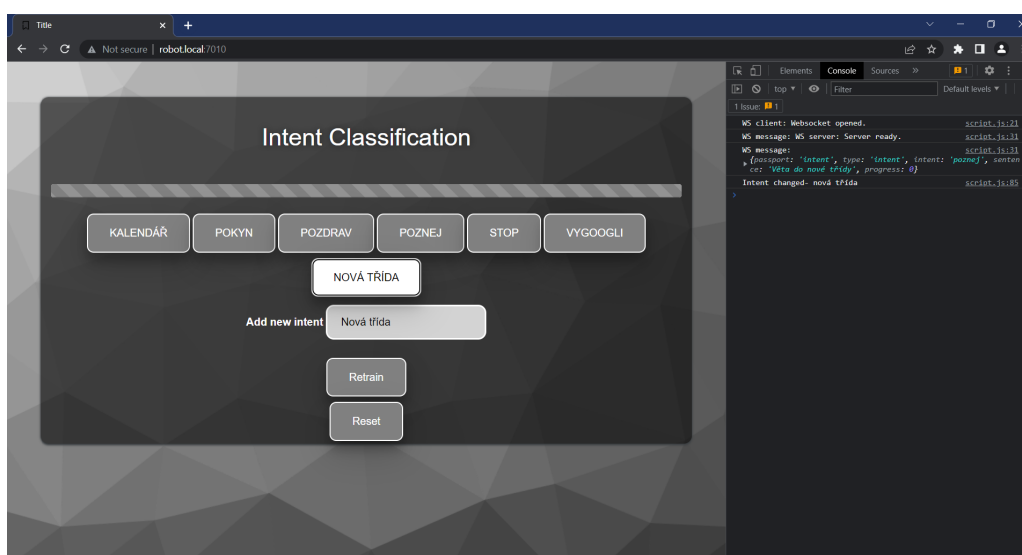
Obrázek 3.3: Celkový náhled na uživatelské rozhraní

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 orámováno zelenou barvou.



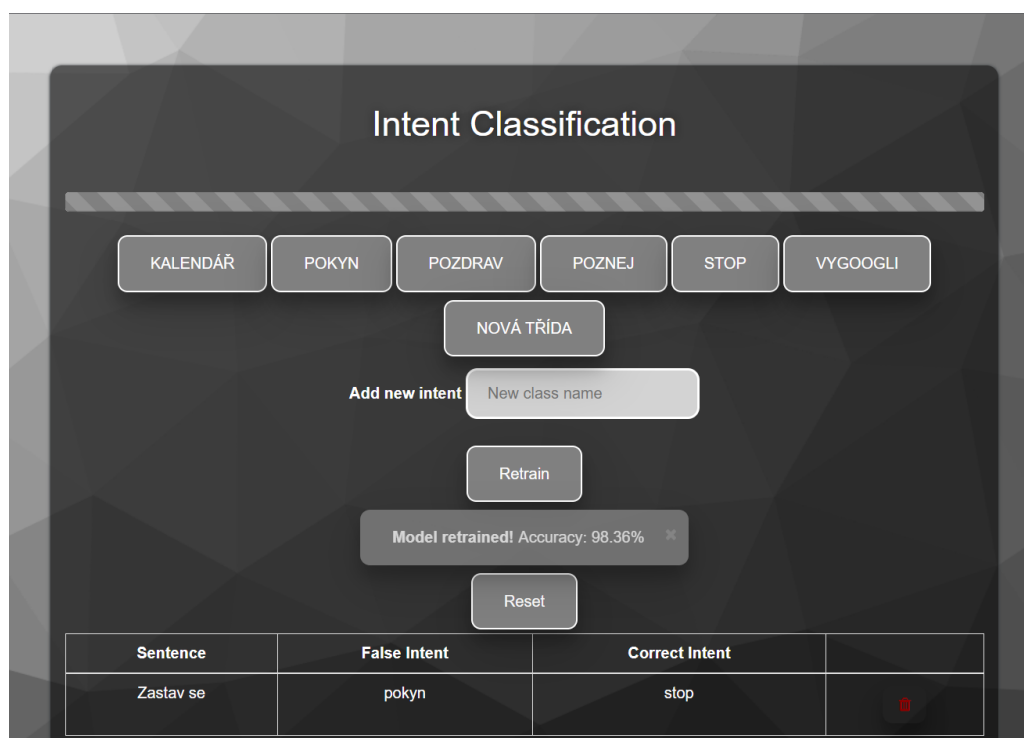
Obrázek 3.4: Zvýraznění predikovaného záměru

Po zobrazení predikovaného záměru má uživatel možnost poskytnout zpětnou vazbu na predikci. Potvrzení se provede kliknutím na zeleně orámované tlačítko. V případě, kdy 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 si jej přidat – napsáním jména nového záměru a stiskem klávesy Enter. Přejde-li teď tedy od uživatele text do nové třídy, na Obrázku 3.5 je tímto textem „Věta do nové třídy“, změnil uživatel její záměr z predikovaného „poznej“ na „nová třída“ kliknutím na příslušné tlačítko.



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 přetrénování modelu na starém datasetu obohaceném o nové vzorky vytvořené z textového vstupu uživatele a jeho korekce predikce záměru pro daný text tj. správného záměru. Po přetrénování, 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 přetrénování, možnost odstranění chybných vzorků

3.4 Robot.v1

3.5 Přetrénovací smyčka

4.1 Ručně vytvořený dataset

Tato data obsahují pouze 60 vzorků, proto nebyl tento dataset rozdělen na trénovací, testovací a validační data. Vzorků je takovéto malé množství záměrně, protože dataset má sloužit jen k základnímu předtrénování modelu, který bude později dotrénován za základě zpětné vazby od uživatele na predikce modelu (Human in the loop přístup). Celý dataset je k nahlédnutí viz Příloha 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 Obsáhlejší dataset

Tento dataset obsahuje celkem 22500 vzorků, ty byly rozděleny na trénovací, testovací a validační data. Dataset slouží k předtrénování většího modelu, který bude dotrénován pomocí Fine-tunningu. Rozdělení dat a informace o nich jsou v Tabulce 4.2.

Tabulka 4.2: Statistiky datasetu 2

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

Experimenty a výsledky

5

5.1 Výběr frameworku (Keras vs. PyTorch)

Keras a PyTorch 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 nižší úroveň abstrakce, což poskytuje větší kontrolu a flexibilitu při vytváření sítí.

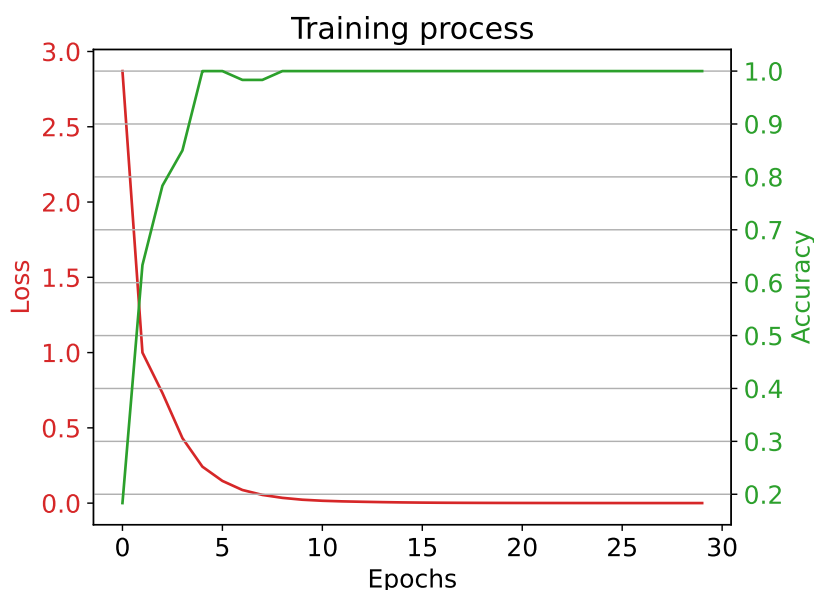
V rámci tohoto experimentu bylo provedeno porovnání rychlosti a efektivity frameworků PyTorch a Keras při použití stejných hyperparametrů neuronové sítě a stejného datasetu. Hyperparametry jsou vypsány v Tabulce 5.1, použitým datasetem byl Dataset 4.1 se šedesáti vzorky. Obě neuronové sítě dosahovaly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

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

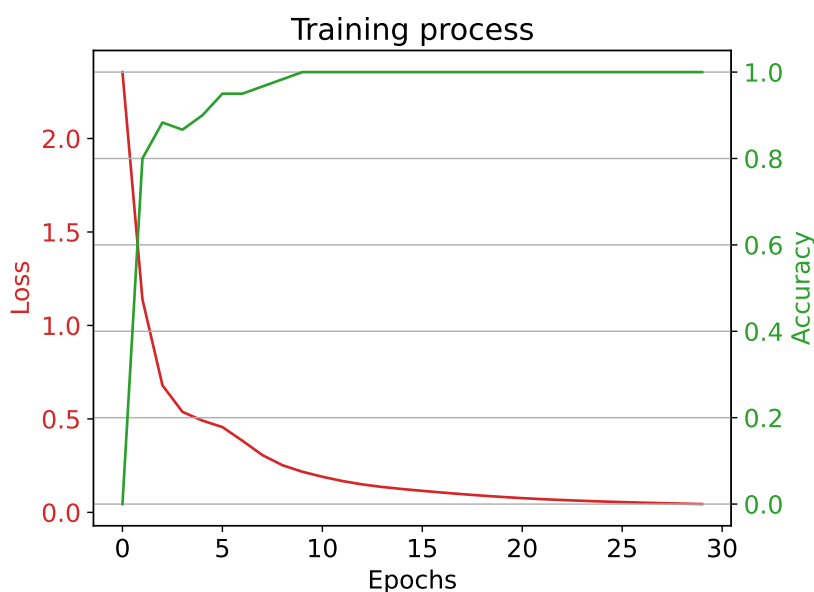
Parametr	Hodnota
Počet skrytých vrstev	1
Počet neuronů ve skryté vrstvě	35
Aktivační funkce	Relu, Softmax
Learning rate	0.01
Velikost dávky	40
Počet epoch	30
Optimizér	Adam

Obrázky 5.1 a 5.2 ilustrují vývoj přesnosti (Accuracy) a ztráty (Loss) při trénování

neuronové sítě v Pytorch a Keras pro výše zmíněné hyperparametry a dataset.



Obrázek 5.1: Průběh trénování neuronové sítě v Pytorch



Obrázek 5.2: Průběh trénování neuronové sítě v Keras

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.

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

	Pytorch	
Operace	Čas [s] (laptop)	Čas [s] (RPi)
Načtení knihoven	3.329±0.055	13.785±0.087
Načtení modelu	0.001±0.000	0.004±0.000
Predikce jednoho vzorku	0.036±0.014	0.672±0.226
	Keras	
Operace	Čas [s] (laptop)	Čas [s] (RPi)
Načtení knihoven	1.805±0.046	5.862±0.529
Načtení modelu	0.001±0.000	0.004±0.001
Predikce jednoho vzorku	0.028±0.001	0.644±0.081

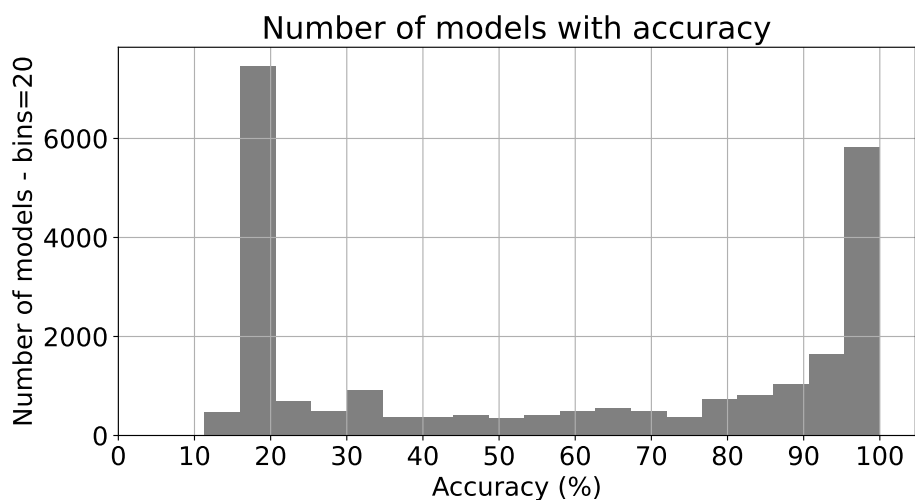
Výsledky experimentu ukázaly, že v případě importu knihoven byl PyTorch pomalejší než Keras (přibližně 2×). Pro načtení modelu byly časy téměř srovnatelné, v případě predikce jednoho vzorku byl Keras mírně rychlejší než Pytorch (o 0.008 sekund na osobním počítači, o 0.028 sekund na RPi).

5.2 Výběr velikosti sítě v závislosti na přesnosti a rychlosti modelu

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.

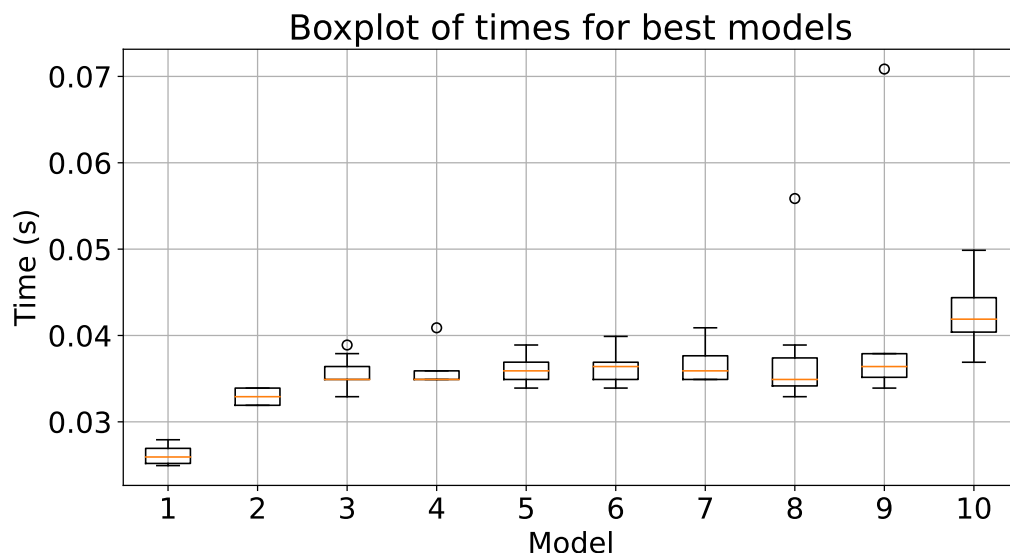
Experiment byl proveden na osobním počítači na jednoduché úloze klasifikace pro Dataset 4.1. Použit byl framework PyTorch a implementace sítě s jednou skrytou vrstvou. Nejprve bylo testováno 24000 modelů s různými hyperparametry (viz Příloha B) a s pěti různými seedy¹ z hlediska jejich přesnosti. Histogram zobrazující četnosti modelů v závislosti na dosažené přesnosti je na Obrázku 5.3.

¹ Pod pojmem seed se rozumí seed pro generování pseudonáhodných čísel, česky též semínko nebo násada.



Obrázek 5.3: 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 a evaluace sítě. Bylo získáno 514 neuronových sítí s přesností 100 % pro všechny realizace, z nich bylo vybráno 10 nejrychlejších sítí, boxplot doby jejich trénování a evaluace je na Obrázku 5.4, konkrétní hyperparametry pak v Tabulce 5.3 (indexy modelů v grafu odpovídají řádkům v tabulce).



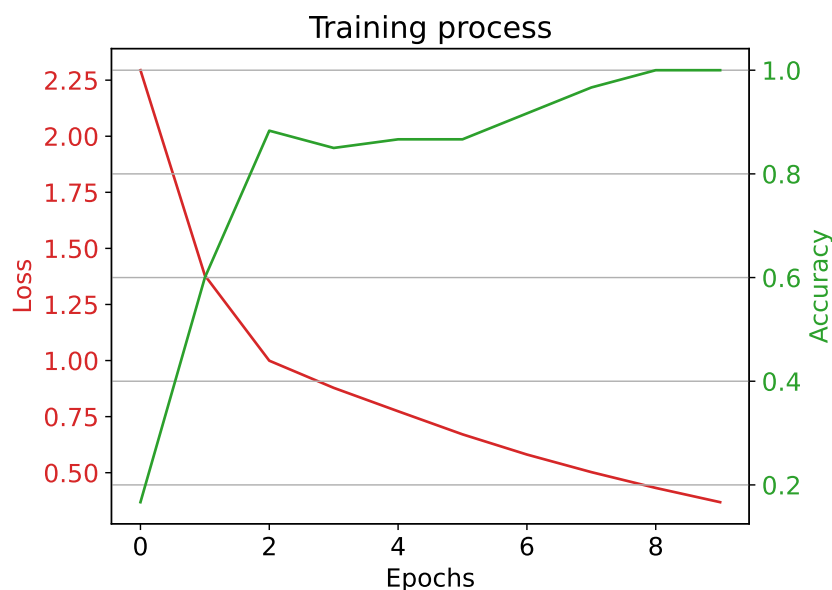
Obrázek 5.4: Boxplot doby trénování a evaluace jednotlivých modelů

Tabulka 5.3: Hyperparametry deseti nejrychlejších neuronových sítí a jejich průměrné časy trvání trénování a evaluace

Model	Počet neuronů ve skryté vrstvě	Učící koeficient	Počet epoch
1	45	0.01	10
2	20	0.01	15
3	45	0.01	15
4	50	0.01	15
5	35	0.01	15
6	35	0.01	15
7	50	0.01	15
8	30	0.01	15
9	45	0.01	15
10	40	0.01	15

Model	Velikost dávky	Optimizér	Průměrný čas [s]
1	40	Adam	0.0262
2	40	Adam	0.0329
3	40	Adam	0.0355
4	50	Adam	0.0358
5	40	Adam	0.0361
6	50	Adam	0.0363
7	40	Adam	0.0366
8	40	Adam	0.0373
9	50	Adam	0.0396
10	40	Adam	0.0424

Z Tabulky 5.3 je zřejmé, že nejlepších výsledků dosáhla neuronová síť číslo 1 se čtyřiceti pěti neurony ve skryté vrstvě, deseti epochami, učícím koeficientem 0.01, velikostí dávky 40 a optimizérem Adam. Na Obrázku 5.5 je zobrazen průběh trénovacího procesu této neuronové sítě.



Obrázek 5.5: Průběh trénování neuronové sítě s optimálními hyperparametry

5.3 Analýza schopnosti sítě natrénovat se v závislosti na počtu dat 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 dat 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 učící koeficient, 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

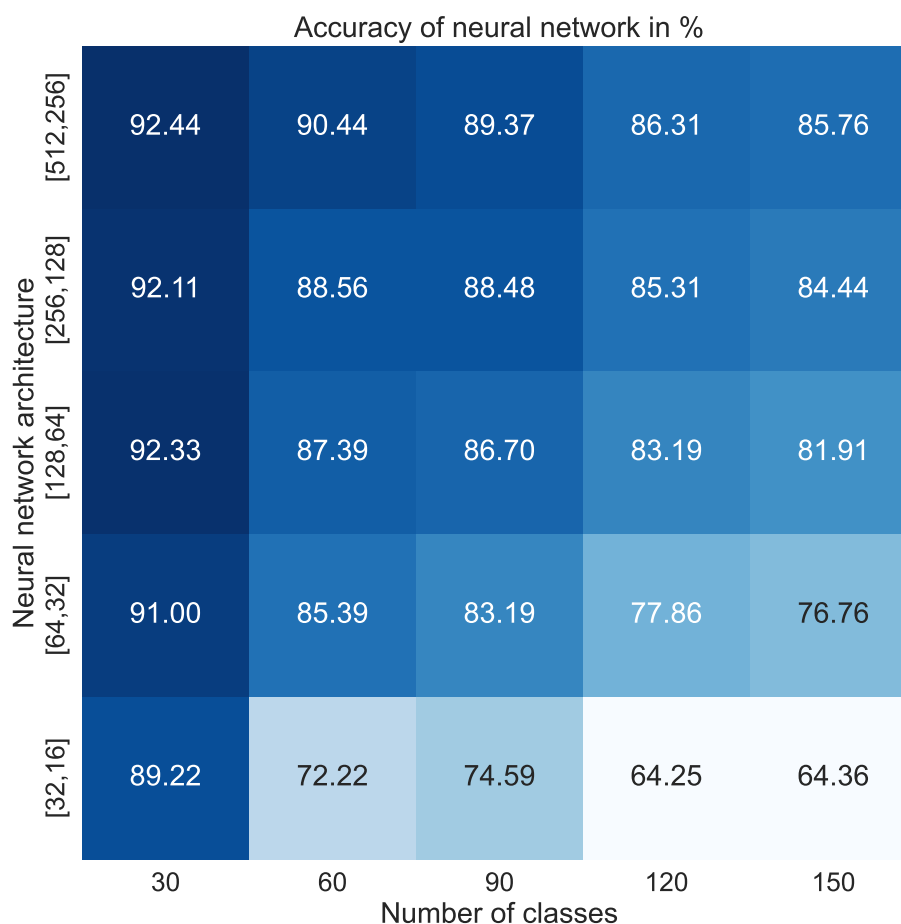
5.3. Analýza schopnosti sítě natrénovat se v závislosti na počtu dat a tříd

0.854 +/- 0.002 a ztráty 0.588 +/- 0.015 (testováno pro 15 realizací).

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

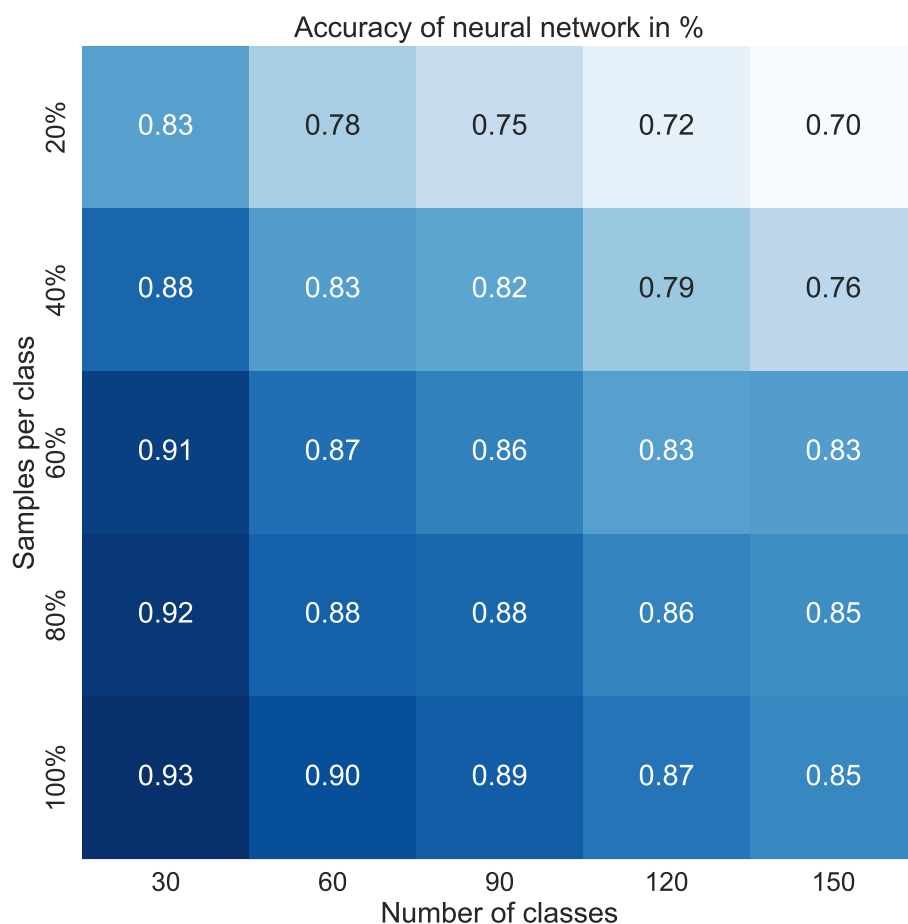
Parametr	Hodnota
Počet skrytých vrstev	2
Počet neuronů ve skrytých vrstvách	512, 256
Aktivační funkce	Sigmoid, Sigmoid, Softmax
Learning rate	0.003
Velikost dávky	2048
Počet epoch	300
Optimizér	Adam

Pro analýzu schopnosti neuronové sítě natrénovat se v závislosti na architektuře sítě a na počtu tříd bylo vytvořeno pět neuronových sítí s různým počtem 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 5.6 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 5.6: Přesnost neuronové sítě v závislosti na její architektuře (počtu neuronů ve skrytých vrstvách) a počtu tříd

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 vzorků pro jednotlivé třídy byly opět děleny do pěti kategorií po dvaceti procentních úbytcích. Například pro testovací data odpovídaly tedy: 30, 24, 18, 12, 6. Výsledky této analýzy jsou na Obrázku 5.7 představujícím heatmapu přesností neuronové sítě.



Obrázek 5.7: Přesnost neuronové sítě v závislosti na počtu tříd a počtu vzorků na třídu

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.

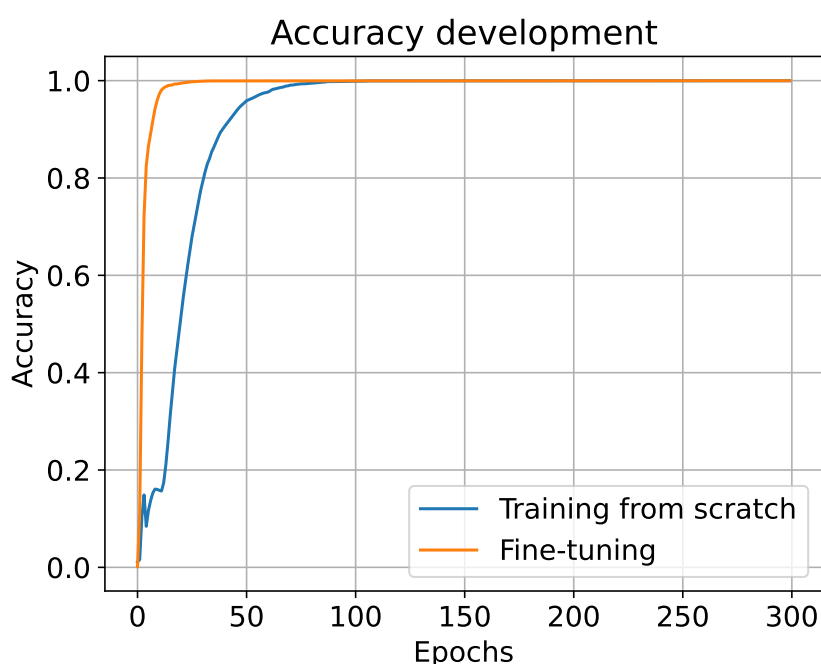
5.4 Porovnání Fine-tuningu a tréninku od nuly (Training From Scratch)

Fine-tuning a trénink od nuly jsou dva způsoby trénování neuronových sítí. Při tréninku od nuly se model učí rozpoznávat data zcela od nuly, což znamená, že se vytvoří nová síť a váhy se inicializují náhodně. Tento přístup obecně vyžaduje větší množství dat a času na trénování. Na druhé straně při fine-tuningu se používá již natrénovaný model a váhy se upravují tak, aby lépe vyhovovaly novým datům. Tento

přístup je obvykle rychlejší a vyžaduje méně nových dat než trénink od nuly.

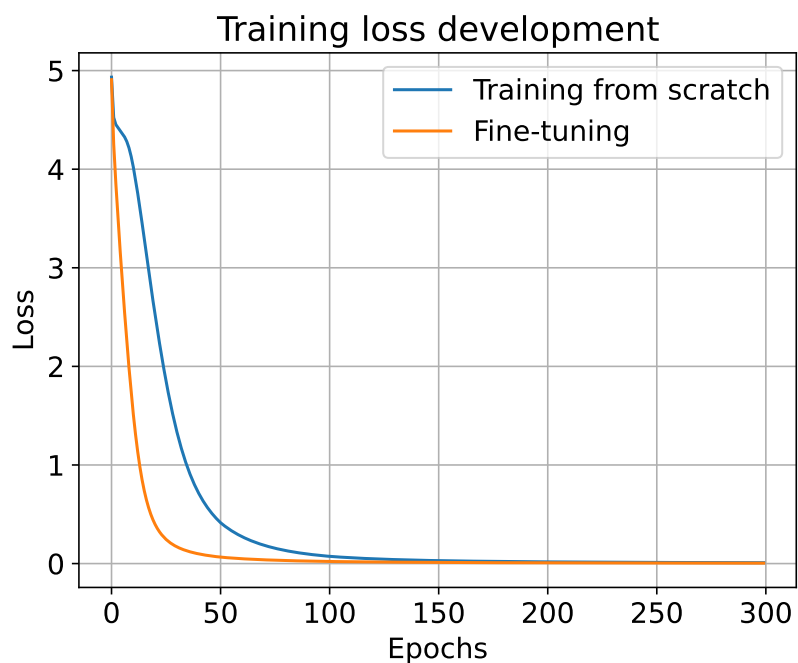
V rámci tohoto experimentu byla využita neuronová síť z experimentu 5.3 trénovaná na Datasetu 4.2. Byly porovnány dva modely neuronové sítě, první model byl vytvořen z předtrénovaného modelu pomocí fine-tuningu – poslední vrstva neuronové sítě byla nahrazena za náhodnou inicializaci, druhý model měl náhodnou inicializaci pro všechny vrstvy, jednalo se tedy o trénování od nuly. Modely byly porovnávány z hlediska jejich průběhu trénování a validace a z hlediska výsledné přesnosti.

Nejprve byly tyto modely porovnány pro modifikaci Datasetu 4.2 o šedesáti třídách a šedesáti trénovacích, osmnácti trénovacích a dvanácti validačních vzorcích. První model dosáhl přesnosti 0.89 a ztráty 0.41, druhý model dosáhl přesnosti 0.87 a ztráty 0.46. Na Obrázku 5.8 je zobrazen vývoj přesnosti při trénování těchto modelů. Je zřejmé, že model vytvořený pomocí fine-tuningu dosáhl přesnosti 1.0 přibližně o 60 epoch dříve.

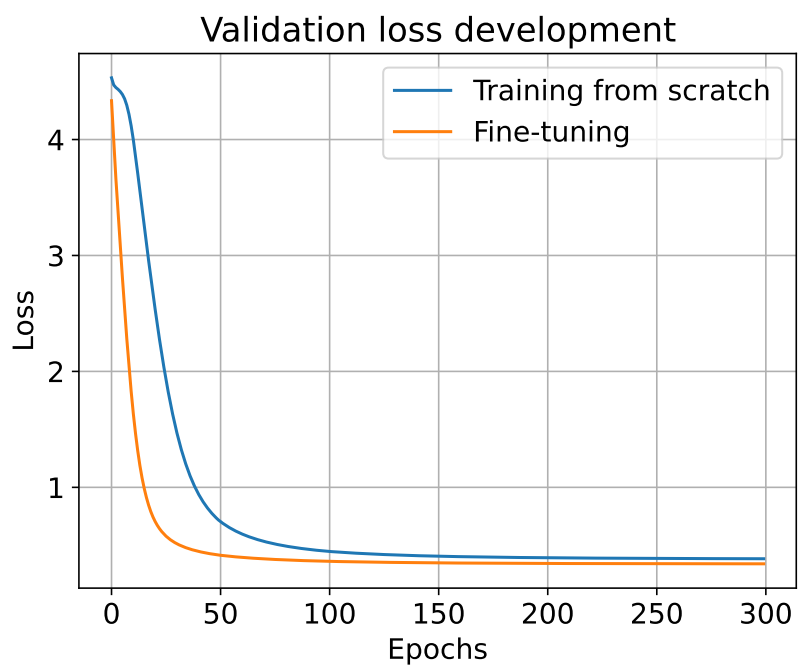


Obrázek 5.8: Vývoj přesnosti při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 60 tříd

Obrázky 5.9 a 5.10 pak zobrazují průběh ztráty při trénování a validaci těchto modelů, opět je patrné, že u modelu vytvořeného pomocí fine-tuningu dochází k rychlejšímu poklesu ztráty.

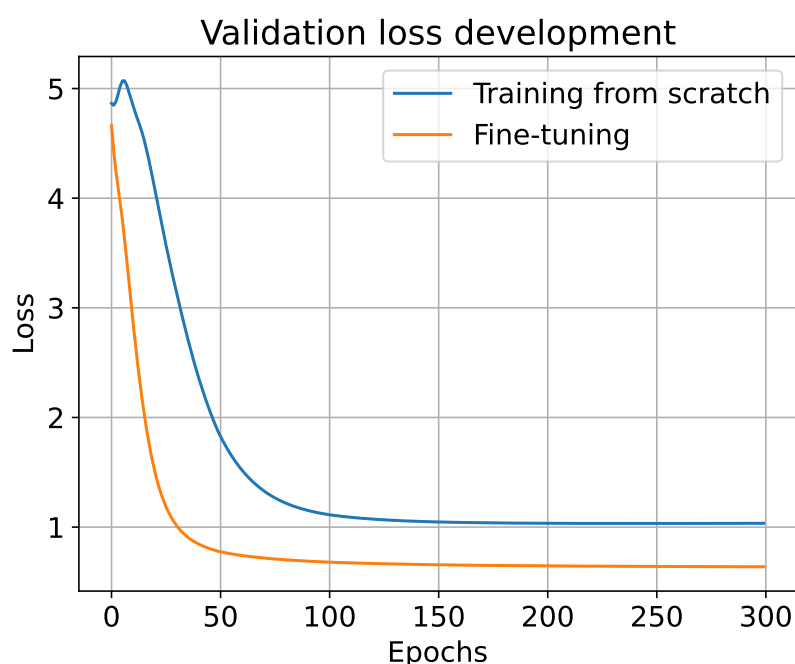


Obrázek 5.9: Vývoj ztráty při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 60 tříd



Obrázek 5.10: Vývoj ztráty při validaci modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 60 tříd

Dále byla testována situace, kde bylo z původního datasetu vybráno 130 tříd s dvaceti trénovacími, šesti testovacími a čtyřmi validačními vzorky. Jednalo se tedy o poměrně okrajový případ řídkého datasetu. V tomto případě dosáhl první model přesnosti 0.83 a ztráty 0.64 a druhý model měl přesnost 0.72 a ztrátu 1.12. Vývoj validační ztráty pro oba modely je na Obrázku 5.11, grafy vývoje přesnosti a ztráty při trénování jsou k nahlédnutí v Příloze C. Je patrné, že validační ztráta modelu trénovaného od nuly je značně vyšší (přibližně 2×) než u prvního modelu a že po přibližně posledních 100 epoch již neklesala.



Obrázek 5.11: Vývoj ztráty při validaci modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 130 tříd a málo vzorků

Výsledky experimentu naznačují, že pro pouhou modifikaci většího datasetu je fine-tuning efektivnějším přístupem než trénink od nuly, pokud je k dispozici již předtrénovaný model. Při porovnání modelů dosáhl model vytvořený pomocí fine-tuningu v obou testovaných případech vyšší přesnosti (0.89 a 0.83) a nižší ztráty (0.41 a 0.64) než model vytvořený od nuly. Navíc bylo zřejmé, že model vytvořený pomocí fine-tuningu dosáhl ve všech případech vyšší přesnosti respektive nižší ztráty rychleji – za menší počet epoch.

Aplickace

6

Discussion starter...

7.1 Rekapitulace metod

7.2 Shrnutí výsledků

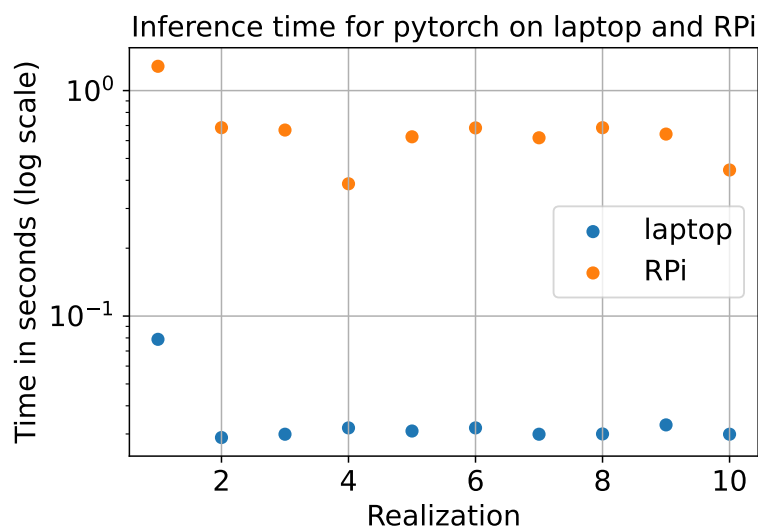
Experiment 5.1. porovnával frameworky Keras a Pytorch pro pevně danou architekturu a dataset. Keras měl kratší doby načtení knihoven (přibližně 2×), srovnatelné doby načtení modelu a mírně rychlejší dobu predikce jednoho vzorku oproti Pytorch (o 0.008 sekund na osobním počítači, o 0.028 sekund na RPi).

To, že frameworky nemají stejný průběh křivek při trénování modelů 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. Pytorch je větší než Keras, proto může trvat déle, než se celá knihovna načte. Je také více flexibilní a umožňuje uživatelům vytvářet složitější modely. Dále má Pytorch více závislostí než Keras, což může způsobit delší doby načítání.

Při predikci pro jeden vzorek pomocí Pytorch došlo k tomu, že první predikce trvala značně déle než všechny následující, viz Obrázek 7.1. Zpomalení první predikce může být způsobeno tím, že během prvního průchodu jsou nastaveny různé věci jako cache, paměť na grafické kartě, optimalizace grafu a podobně. Následující predikce jsou pak rychlejší díky tomu, že jsou již využívány předem připravené a optimalizované výpočetní zdroje.

Zanedbáním prvního měření doby predikce by framework Pytorch dosáhl hodnoty



Obrázek 7.1: Doba predikce jednoho vzorku pomocí frameworku Pytorch – 10 realizací

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.

Na základě výsledků tohoto experimentu bylo rozhodnuto, že zvoleným frameworkem pro aplikaci na robotickou entitu bude Pytorch. Delší načtení knihoven nebylo rozhodující, protože to probíhá jen jednorázově. Doba načtení modelu a doba predikce pak byly pro oba frameworky srovnatelné. 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 navíc dosahuje Pytorch obecně vyšší rychlosti než Keras, což je také výhodou, protože použitý Dataset 4.1 slouží jen k základnímu předtrénování sítě a má být rozšiřován vzorky ze zpětné vazby od uživatele.

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. Stoprocentní přesnost a nejrychlejší čas (v průměru 0.026 sekund) měla neuronová síť se čtyřiceti pěti neurony ve skryté vrstvě, deseti epochami, učícím koeficientem 0.01, velikostí dávky 40 a optimizérem Adam.

Velká četnost neuronových sítí s přesností mezi 15 a 20 % na Obrázku 5.3 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.

Z analýzy Tabulky 5.3 plyne, že rychlost neuronové sítě s jednou skrytou vrstvou a počtem neuronů v této vrstvě nižším než padesát 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 čtyřiceti pěti neurony ve skryté vrstvě dosahuje lepšího času než pátý model, který jich má třicet pět. 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 velikost dávky nabývala u všech deseti nejlepších neuronových sítí dvou nejvyšších testovaných hodnot – 40 nebo 50. Jako nejlepší hodnota pro učící koeficient se pro danou úlohu jeví 0.01. Nejvhodnějším optimizérem se zdá být Adam.

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.

V první části experimentu byly vybrány vhodné hyperparametry pro neuronovou síť, viz Tabulka 5.4. Neuronová síť s těmito hyperparametry dosáhla přesnosti 0.854 ± 0.002 a ztráty 0.588 ± 0.015 na testovacích datech.

Ve druhé části experimentu byla analyzována schopnost neuronových sítí natrénovat se v závislosti na architektuře sítě a na počtu tříd. Z heatmapy výsledků na Obrázku 5.6 lze vidět, že s rostoucím počtem tříd klesá přesnost klasifikace a že síť s větším počtem neuronů ve skrytých vrstvách dosahuje lepších výsledků. Tato zjištění potvrzují, že při klasifikaci s větším počtem tříd je potřeba mít větší a složitější neuronovou síť, aby bylo možné naučit síť rozlišovat jednotlivé třídy.

V poslední části experimentu byla analyzována schopnost neuronových sítí natrénovat se v závislosti na počtu tříd a na počtu vzorků v jednotlivých třídách. Z výsledků na Obrázku 5.7 lze vidět, že s rostoucím 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. porovnával chování modelu vytvořeného pomocí fine-tuningu s modelem vytvořeným od nuly. Tyto modely byly porovnávány pro dvě modifikace Datasetu 4.2: první modifikace obsahovala 60 tříd, 60 trénovacích dat, 18 testovacích dat a 12 validačních dat, zatímco druhá modifikace měla 130 tříd, 20 trénovacích dat, 6 testovacích dat a 4 validační data. V obou případech dosáhl model vytvořený pomocí fine-tuningu lepších výsledků. Pro první modifikaci datasetu dosáhl přesnosti 0.89 a ztráty 0.41, zatímco druhý model dosáhl přesnosti 0.87 a ztráty 0.4, pro druhou modifikaci datasetu pak přesnosti 0.83 a ztráty 0.64 a druhý model přesnosti

0.72 a ztráty 1.12. Kromě toho konvergoval model vytvořený pomocí fine-tuningu k těmto výsledkům rychleji (přibližně o 60 epoch).

Při použití druhé modifikace datasetu došlo k tomu, že při validaci v posledních přibližně 100 epochách již dále neklesala validační ztráta modelu vytvořeného od nuly a ustálila se na hodnotě 1.12. Tato hodnota je o dost vyšší než u modelu vytvořeného pomocí fine-tuningu, ten měl výslednou hodnotu 0.72. To, že se hodnota validační ztráty modelu vytvořeného od nuly již dále neměnila a zůstala vyšší může být způsobeno tím, že model se pro danou úlohu přetrénoval, protože počet vzorků na třídu byl velmi omezený.

Z výsledků experimentu tedy vyplývá, že fine-tuning existujícího modelu může být účinným způsobem, jak dosáhnout lepších výsledků než vytvoření modelu od nuly, zejména pokud je k dispozici omezený počet trénovacích dat.

Závěr

8

Conclusion text...

8.1 Future Work

Outlook...

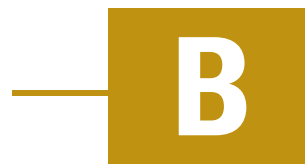
Ručně vytvořený dataset



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
KALENDÁŘ	Jaký je rok

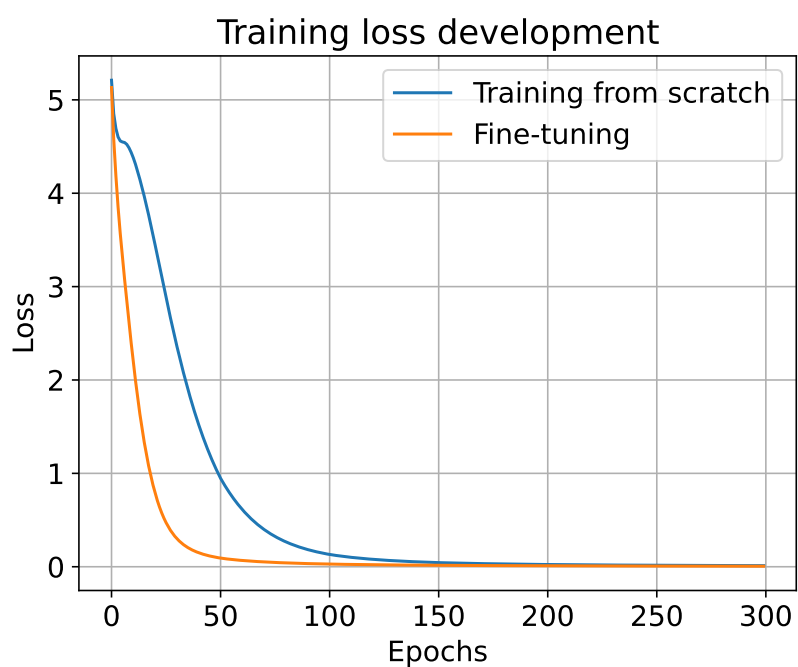
KALENDÁŘ	Co je dnes za den
KALENDÁŘ	Kdo má dnes svátek
KALENDÁŘ	Kdo má dnes narozeniny
VYGOOGLI	Kolik je států v Africe
VYGOOGLI	Najdi mi restauraci v Brně
VYGOOGLI	Jaký je počet obyvatel v Plzni
VYGOOGLI	Kde koupit bitcoin za CZK a jaký je aktuální kurz
VYGOOGLI	Kdo je prezident ve Francii
VYGOOGLI	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

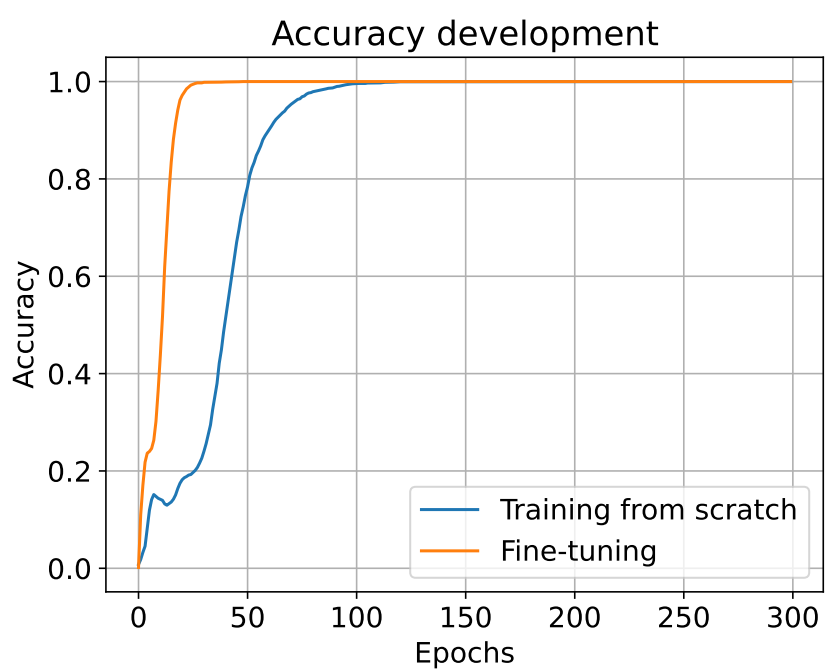


Parametr	Hodnoty
Počet neuronů ve skryté vrstvě	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Aktivační funkce	(Sigmoid, Softmax)
Learning rate	0.1, 0.01, 0.001, 0.0001
Velikost dávky	5, 10, 20, 30, 40, 50
Počet epoch	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Optimizér	SGD, Adam

Grafy k experimentu 5.4



Obrázek C.1: Vývoj ztráty při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 130 tříd a málo trénovacích dat



Obrázek C.2: Vývoj přesnosti při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 130 tříd a málo trénovacích dat

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] Sanket Doshi. *Various optimization algorithms for training neural network*. 2020. URL: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>.
- [8] Nagesh Singh Chauhan. *Loss functions in Neural Networks*. 2021. URL: <https://www.theaidream.com/post/loss-functions-in-neural-networks>.
- [9] 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>.
- [10] 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>.

- [11] 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.
- [12] Michael A Nielsen. *Neural networks and deep learning*. Sv. 25. Determination press San Francisco, CA, USA, 2015.
- [13] 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.
- [14] Morgan Quigley et al. „ROS: an open-source Robot Operating System“. In: *ICRA Workshop on Open Source Software*. 2009.
- [15] 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>.
- [16] 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/>.
- [17] David E. Rumelhart, Geoffrey E. Hinton a Ronald J. Williams. „Learning representations by back-propagating errors“. In: *Nature* 323 (1986), s. 533–536.
- [18] Nikita Sharma. *Exploring optimizers in machine learning*. 2021. URL: <https://heartbeat.comet.ml/exploring-optimizers-in-machine-learning-7f18d94cd65b>.
- [19] Sagar Sharma, Simone Sharma a Anidhya Athaiya. „Activation functions in neural networks“. In: *Towards Data Sci* 6.12 (2017), s. 310–316.
- [20] *SpeechTech* — *speechtech.cz*. <https://www.speechtech.cz/>. [Accessed 25-Feb-2023].
- [21] Stanford Artificial Intelligence Laboratory et al. *actionlib*. 2018. URL: <http://wiki.ros.org/actionlib>.
- [22] 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>.

- [23] Jan Švec, Petr Neduchal a Marek Hruz. „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>.
- [24] 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.
- [25] Ashish Vaswani et al. „Attention Is All You Need“. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: <http://arxiv.org/abs/1706.03762>.
- [26] Eva Volná. *Neuronové sítě 1*. 2008.
- [27] Vishal Yathish. *Loss functions and their use in neural networks*. 2022. URL: <https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9>.
- [28] 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>.
- [29] 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])	5
2.2	Aktivační funkce	7
2.3	Vliv učícího koeficientu při gradientním sestupu (převzato z [2])	11
2.4	Vektorový prostor sedmi slov ve třech kontextech (převzato z [5]) . . .	14
2.5	Diagram fine-tuningu (převzato z [28])	16
2.6	Diagram architektury SpeechCloudu (převzato z [23])	19
3.1	Napojení Sentence Transformeru na neuronovou síť	23
3.2	Nastínění propojení jednotlivých částí v rámci celé aplikace	25
3.3	Celkový náhled na uživatelské rozhraní	26
3.4	Zvýraznění predikovaného záměru	27
3.5	Korekce predikovaného záměru na záměr „Nová třída“	27
3.6	Dokončení přetrénování, možnost odstranění chybných vzorků	28
5.1	Průběh trénování neuronové sítě v Pytorch	32
5.2	Průběh trénování neuronové sítě v Keras	32
5.3	Histogram četností modelů v závislosti na jejich přesnosti	34
5.4	Boxplot doby trénování a evaluace jednotlivých modelů	34
5.5	Průběh trénování neuronové sítě s optimálními hyperparametry . . .	36
5.6	Přesnost neuronové sítě v závislosti na její architektuře (počtu neuronů ve skrytých vrstvách) a počtu tříd	38
5.7	Přesnost neuronové sítě v závislosti na počtu tříd a počtu vzorků na třídu	39
5.8	Vývoj přesnosti při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 60 tříd	40

5.9	Vývoj ztráty při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 60 tříd	41
5.10	Vývoj ztráty při validaci modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 60 tříd	41
5.11	Vývoj ztráty při validaci modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 130 tříd a málo vzorků	42
7.1	Doba predikce jednoho vzorku pomocí frameworku Pytorch – 10 realizací	46
C.1	Vývoj ztráty při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 130 tříd a málo trénovacích dat . . .	55
C.2	Vývoj přesnosti při trénování modelu vytvořeného pomocí fine-tuningu a modelu vytvořeného od nuly pro 130 tříd a málo trénovacích dat . .	56

Seznam tabulek

4.1	Statistiky datasetu 1	29
4.2	Statistiky datasetu 2	30
5.1	Hyperparametry neuronových sítí použitých v experimentu pro výběr frameworku	31
5.2	Doby načtení knihoven, načtení modelů a predikcí jednoho vzorku pro frameworky Pytorch a Keras (v sekundách).	33
5.3	Hyperparametry deseti nejrychlejších neuronových sítí a jejich průměrné časy trvání trénování a evaluace	35
5.4	Hyperparametry neuronové sítě pro klasifikaci záměru trénované na Datasetu 4.2	37

