

BAKALÁŘSKÁ PRÁCE



Neuronové sítě pro porozumění řeči

Autor:
Vladimíra KIMLOVÁ

Vedoucí práce:
Ing. Bulín Martin, M.Sc.

Závěrečná práce pro získání titulu Bakalář (Bc.)

Katedra kybernetiky

18. března 2023

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval(a) samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

Vlastnoruční podpis:

V Plzni dne:

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Abstrakt

Fakulta aplikovaných věd

Katedra kybernetiky

Bakalář (Bc.)

Neuronové sítě pro porozumění řeči

Vladimíra KIMLOVÁ

Na další stranu své DP/BP uvede student v ČJ a AJ anotaci své práce a klíčová slova (max. 10 + 3 řádky)

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

Abstrakt	ii
1 Introduction	1
1.1 State of the Art	1
1.2 Thesis Objectives	1
1.3 Thesis Outline	1
2 Metody	2
2.1 Dopředná neuronová síť	2
2.1.1 Hyperparametry	6
2.2 Sentence transformery	8
2.2.1 FERNET-C5	9
2.3 Neuronová síť pro klasifikaci záměru	10
2.4 Jemné ladění (Fine-tuning)	10
2.5 ROS (Robot Operating System)	11
2.6 Speechcloud	13
2.7 Komunikační protokoly	13
2.7.1 HTTP	13
2.7.2 MQTT	14
2.7.3 WebSockets	14
2.8 Implementace	14
2.9 Uživatelské rozhraní	16
2.10 Robot.v1	20
2.11 Přetrénovací smyčka	20
3 Data	21
3.1 Malý dataset	21
4 Examples	22
4.1 XOR Function	22
5 Discussion	23
5.1 Recapitulation of Methods	23
5.2 Summary of Results	23
6 Conclusion	24
6.1 Future Work	24
Literatura	25
A1 Malý dataset	27
A2 Structure of the Workspace	30

Seznam obrázků

2.1	Diagram architektury neuronové sítě (převzato z [24]) . . .	2
2.2	Aktivační funkce (převzato z [15])	4
2.3	Vliv učícího koeficientu při gradientním sestupu (převzato z [12])	7
2.4	Napojení Sentence Transformeru na neuronovou síť	10
2.5	Diagram jemného ladění (převzato z [20])	11
2.6	Nastínění propojení jednotlivých částí v rámci celé aplikace	16
2.7	Celkový náhled na uživatelské rozhraní	17
2.8	Zvýraznění predikovaného záměru	17
2.9	Potvrzení predikce záměru „pozdrav“ uživatelem	18
2.10	Přidání nové třídy (nového záměru) s názvem „Nová třída“	18
2.11	Korekce predikovaného záměru na záměr „Nová třída“ . . .	19
2.12	Proces přetrénování modelu	19
2.13	Dokončení přetrénování, možnost odstranění chybných vzorků	20

Seznam tabulek

Seznam zkratek

AI	A rtificial I ntelligence
ANN	A rtificial N eural N etwork
Your abbreviations...	

Kapitola 1

Introduction

Your intro...

1.1 State of the Art

1.2 Thesis Objectives



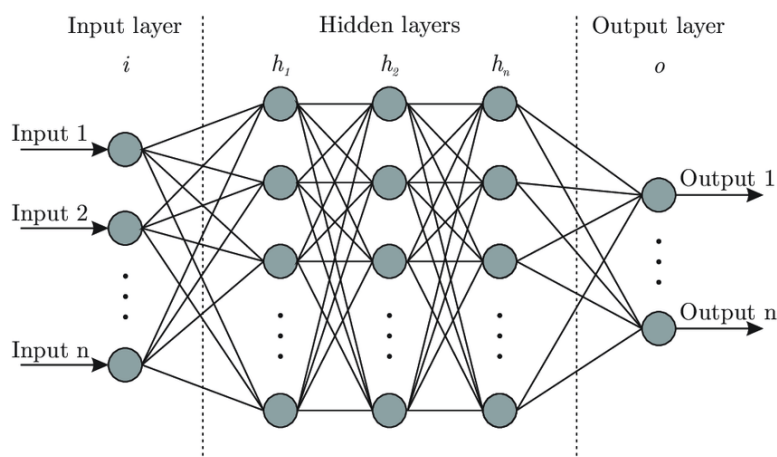
1.3 Thesis Outline

Kapitola 2

Metody

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. Každý neuron je spojen se všemi neurony v následující vrstvě. Architektura je nastíněna na obrázku 2.1.



OBRÁZEK 2.1: Diagram architektury neuronové sítě (převzato z [24])

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 a výstupní signál. Obecně platí, že velikost vstupní vrstvy je dána rozměrem vstupních dat a velikost výstupní vrstvy je za předpokladu klasifikačního problému určena počtem tříd [17].

Nechť má neuronová síť L vrstev, kde $l = 1, 2, \dots, L$ označuje vrstvu a $n^{(l)}$ označuje počet neuronů v l -té vrstvě. x je vektor vstupních dat, který je přiveden na vstup sítě. Hodnoty neuronů ve vrstvě l jsou označeny jako vektor $a^{(l)}$ [6].

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

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

kde $W^{(l)}$ je matice vah neuronů mezi $l - 1$ a l vrstvou, $b^{(l)}$ je bias vektor pro vrstvu l a $a^{(0)} = x$ [6].

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

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

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 [10]:

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

Sigmoidní aktivační funkce má tvar [10]:

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

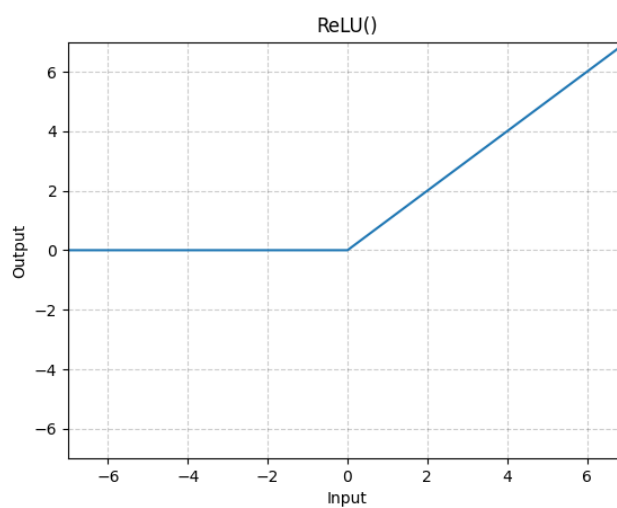
Tanh aktivační funkce má tvar [10]:

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

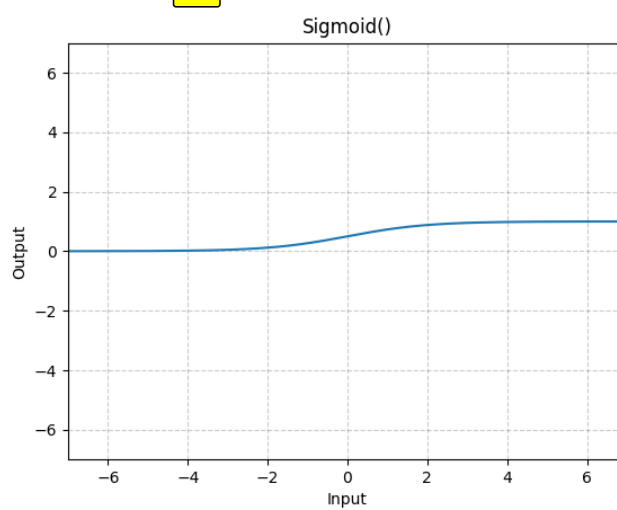
Funkce Softmax může být na rozdíl od sigmoidních funkcí, které se používají pro binární klasifikaci, použita pro problémy s klasifikací více tříd. Pokud má výstupní vrstva K neuronů, potom má Softmax funkce tvar [10]:

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

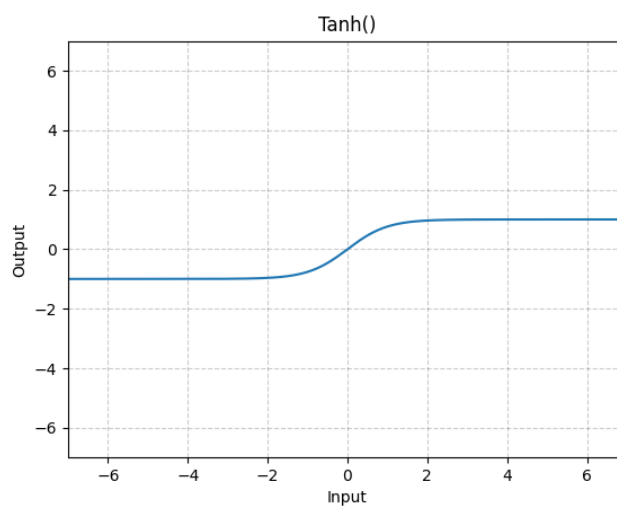
Aktivační funkce ReLU, Sigmoid a Tanh jsou zobrazeny na obrázku 2.2.



(A) ReLU



(B) Sigmoid



(C) Tanh

OBRÁZEK 2.2: Aktivační funkce (převzato z [15])



Cílem trénování neuronové sítě je nalézt takové hodnoty vah a biasu neuronů, aby síť byla schopna klasifikovat nebo regresovat vstupní data 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“ [1].

Algoritmus se skládá ze tří hlavních částí [3]:

- 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:

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

Zpětné šíření (Backpropagation):

C představuje tzv. ztrátovou funkci (Loss Function), označovanou také jako Cost function. Pro klasifikační problémy má C obvykle tvar kvadratické funkce [6]:

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



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

Zpětné šíření se týká pochopení, jak změna vah a biasů v síti ovlivňuje hodnotu ztrátové funkce. Dále jsou tedy uvedeny partiální derivace C vzhledem k vahám a biasům sítě. Pro neuron j ve vrstvě l jsou tyto derivace následující [6]:

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \quad (2.9)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.10)$$

kde δ_j^l je tzv. chyba neuronu j ve vrstvě l , která se vypočítá jako [6]:

$$\delta_j^l = \sigma'(z_j^l) \sum_k w_{jk}^{l+1} \delta_k^{l+1} \quad (2.11)$$

kde $\sigma'(\cdot)$ 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 [6]:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \quad (2.12)$$

Aktualizace vah a biasů:

Po výpočtu gradientu chyby se váhy a biasy sítě aktualizují pomocí gradientního sestupu. **Nova** hodnota váhy w_{jk}^l a biasu b_j^l je vypočtena jako [14]:

$$w_{jk}^l \leftarrow w_{jk}^l - \eta \frac{\partial C}{\partial w_{jk}^l} \quad (2.13)$$

$$b_j^l \leftarrow b_j^l - \eta \frac{\partial C}{\partial b_j^l} \quad (2.14)$$

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 [5]. 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 [3].


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 [9].

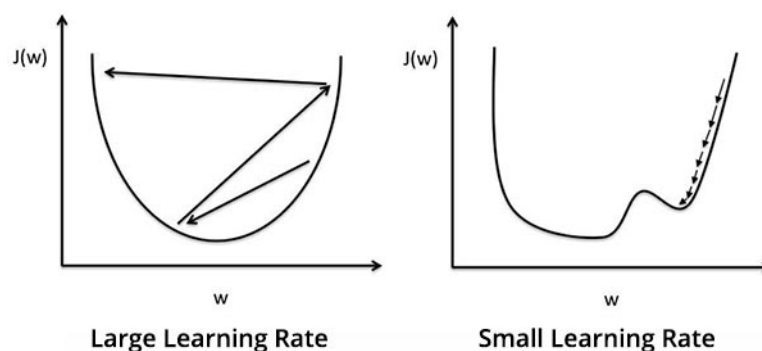
Hyperparametrické ladění je proces hledání optimálních hyperparametrů pro **danou danou** úlohu (např. maximalizace přesnosti modelu na validačním datasetu). Tento proces je často prováděn pomocí opakovaného trénování modelu s různými hodnotami hyperparametrů a vyhodnocováním výsledků pomocí validační sady dat.

Následuje výběr nastavitelných hyperparametrů:



- Počet skrytých vrstev: Počet skrytých vrstev představuje počet vrstev mezi vstupní a výstupní **vrstvnou**. Malý počet vrstev může vést k nedotrénování (Underfitting) modelu, zatímco příliš mnoho vrstev může vést k přetrénování (Overfitting) modelu [22]. 

- 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 [22]. 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 [9].
- 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 [9]. 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 [12])

- Velikost dávky (Batch Size): Velikost dávky je počet podvzorků, které jsou předány síti, po kterých nastane aktualizace parametrů [9]. Velikost dávky ovlivňuje rychlost trénování sítě. Menší velikost dávky urychluje proces učení, ale zvyšuje rozptyl přesnosti na validačním datasetu. Větší velikost dávky naopak zpomaluje proces učení, ale má nižší rozptyl přesnosti na validačním datasetu [22].
- Počet epoch: Počet epoch značí, kolikrát je celý trénovací dataset předložen neuronové síti během tréninku [9]. Příliš málo epoch může vést k nedotrénované síti, zatímco příliš mnoho epoch může vést k přeučení sítě [22].
- Optimalizační algoritmus: Optimizér je algoritmus používaný k úpravě vlastností neuronové sítě, jako jsou váhy a učící koeficient, s cílem snížit chybovost [16]. 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 uvíznout v lokálním minimu, může

být pomalý, pokud je dataset velký a má mnoho příznaků, a má vysoké paměťové nároky [16].

- 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 učicího koeficientu pro dosažení stejné konvergence jako u gradientního sestupu. Algoritmus může být navíc nestabilní [16].



– Momentum: Momentum bylo vynalezeno k redukci vysoké variance v SGD a zjemnění konvergence. Konverguje také rychleji než gradientní sestup. Nevýhodou však je přidání nového hyperparametru, který musí být nastaven manuálně [16].

- 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 datasey s chybějícími vzorky. Je však výpočetně náročný a snižující se učicí koeficient má za následek pomalé trénování [16].
- Adadelta: Jedná se o rozšíření metody Adagrad. Adadelta 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 [16].
- 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 [19].
- 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 datasey a je výpočetně efektivní. Výkon algoritmu Adam závisí na typu poskytnutých dat a jedná se o kompromis mezi rychlostí a generalizací [19].

2.2 Sentence transformery



Sentence Transformer je model strojového učení, který dokáže transformovat vstupní textové sekvence (např. věty) do vektorů s vysokou dimenzionalitou. Tyto vektory pak mohou být využity například pro kategorizaci, porovnávání podobnosti nebo další úlohy zpracování přirozeného jazyka.



Architektura zvaná transformer byla poprvé představena v roce 2017 v článku Attention Is All You Need. Tato architektura dokáže na rozdíl od tradičních rekurentních nebo konvolučních neuronových sítí ~~pracovat zcela bez opakování a konvolucí~~, navíc dosahují tyto modely lepší kvality, jsou více paralelizovatelné a vyžadují podstatně méně času na trénování. Klíčovou součástí transformeru je mechanismus pozornosti (Attention), který umožňuje modelu se zaměřit na důležité části vstupní sekvence a ignorovat méně významné informace [11].



Vzorec pro výpočet pozornosti (Scaled Dot-Product Attention) vypadá následovně [11]:

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

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 trénování

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 několika hlav pozornosti se umožňuje reprezentace několika souborů vztahů současně [23].



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 [18].

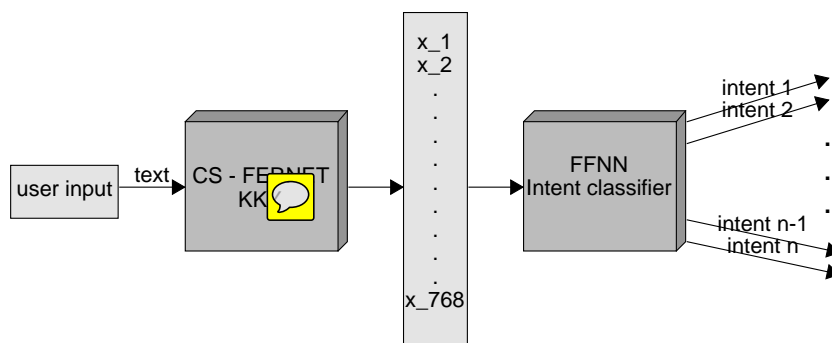
Model je veřejně k dispozici online a umožňuje integraci s různými aplikacemi, jako jsou chatboty, analýza sentimentu, nebo klasifikace textu.

2.3 Neuronová síť pro klasifikaci záměru

Neuronová síť pro klasifikaci záměru (Intent Classifier) je dopředná neuronová síť, viz 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 lze ji 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 2.2. Tato technologie umožňuje převádět texty na vektorové reprezentace zachycující sémantické vlastnosti.

Ilustrativní napojení Sentence Transformeru FERNET (viz 2.2.1) na dopřednou neuronovou síť pro klasifikaci záměru je zobrazeno na obrázku 2.4. Textový uživatelský vstup je pomocí FERNETU převeden na vektor délky 768. Tento vektor pak slouží jako vstup do neuronové sítě, jejíž výstupem je záměr textu uživatele.



OBRAZEK 2.4: Napojení Sentence Transformeru na neuronovou síť

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.



2.4 Jemné ladění (Fine-tuning)

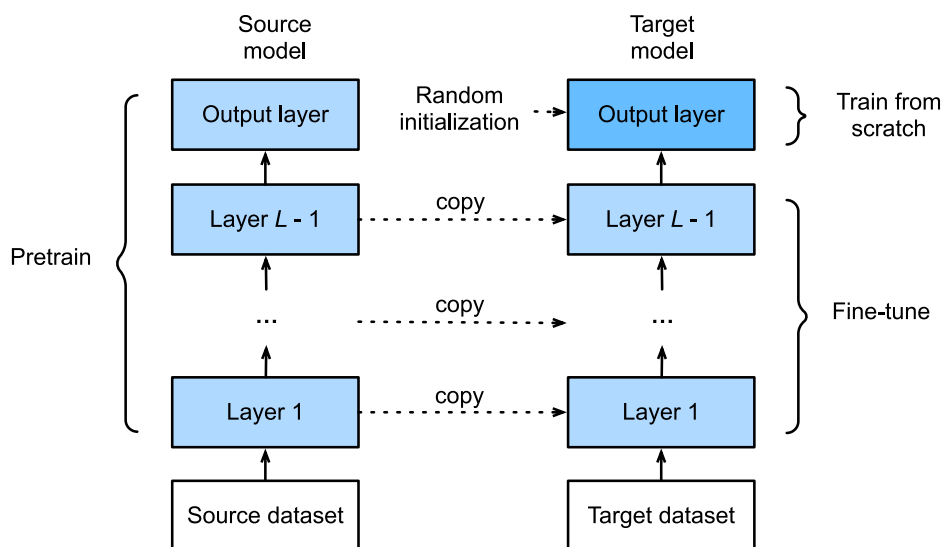
Jemné ladění neuronových sítí je proces přizpůsobení již natrénované sítě na nový úkol, který může být odlišný od původního úkolu, pro který byla síť trénována. Tento proces spočívá v použití vah již trénované sítě jako počátečních hodnot pro trénování nové sítě [8]. Při použití jemného ladění se předtrénovaný 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. Výjimkou je 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 [8]. Její inicializace je zpravidla náhodná [20].



Nakonec je nový model trénován na cílovém datasetu. Výstupní vrstva je trénována od začátku, zatímco parametry všech ostatních vrstev je jemně vyladěny na základě parametrů zdrojového modelu [20].

Nástin fungování jemného ladění je zobrazen na obrázku 2.5.



OBRÁZEK 2.5: Diagram jemného ladění (převzato z [20])

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.

2.5 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 [21, 7].

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 uživateli po celém světě v oblastech od domácích projektů v rámci koníčků, po velké průmyslové automatizované systémy [7]. 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 [13].

Základními koncepty implementace ROS jsou uzly (Nodes), zprávy (Messages), té-mata (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) [7, 4].

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 [7, 4].

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 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 [7, 4]. 

Fi rické 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 [7, 4].
- 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 [7, 4].
- 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 [7, 4].
- 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 [7, 4].
- Zdarma a Open-Source: Úplný zdrojový kód ROS je veřejně dostupný [7, 4].

2.6 Speechcloud




SpeechCloud je platforma pro zpracování řeči a analýzu hlasu vyvinutá společností **Speech Tech.** 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. [25].


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.

2.7 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. 

2.7.1 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 (**typický webový prohlížeč**) 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.

2.7.2 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 odebrat 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.

2.7.3 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 umožňují 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.

2.8 Implementace

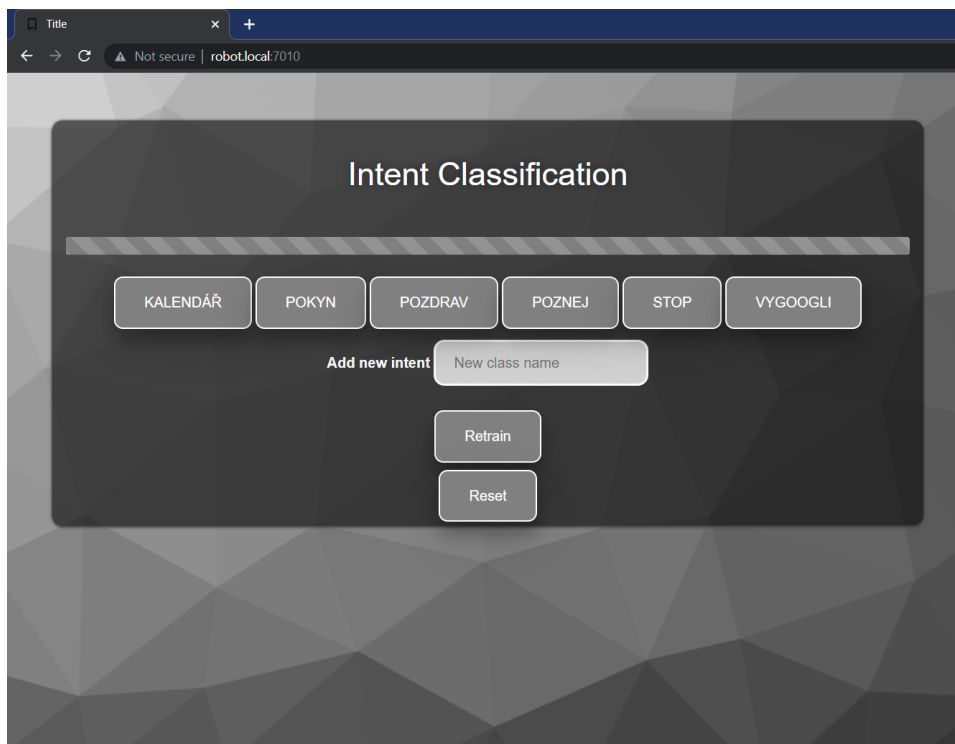
Diagram na obrázku 2.6 zobrazuje propojení jednotlivých stavebních bloků celé aplikace a tok informací mezi uživatelem a jednotlivými bloky. Běh všech skriptů v rámci robotické entity, konkrétně Raspberry Pi 4B, je zajištěn pomocí softwarového frameworku ROS (více viz 2.5). Pro každou funkční část aplikace 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.

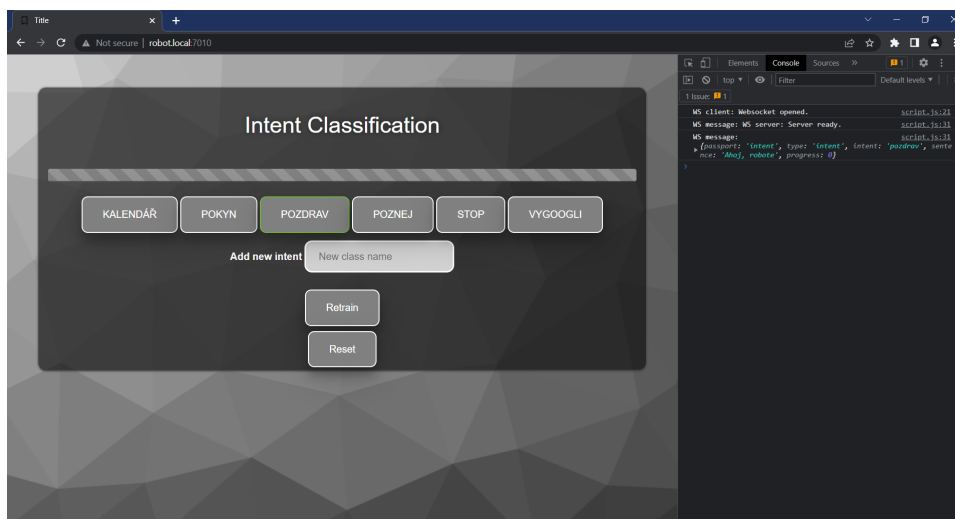
- Dialog package – Teto balík zajišťuje hlavní funkcionalitu celé dialogové smyčky. Běží zde dialogový server, v rámci nějž jsou registrováni odběratelé pro hlasový vstup od uživatele a pro zprávy z webové stránky (tj. zpětné vazby od uživatele). Dále dialogový server umožňuje publikování zpráv pro webovou 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 2.3), 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.

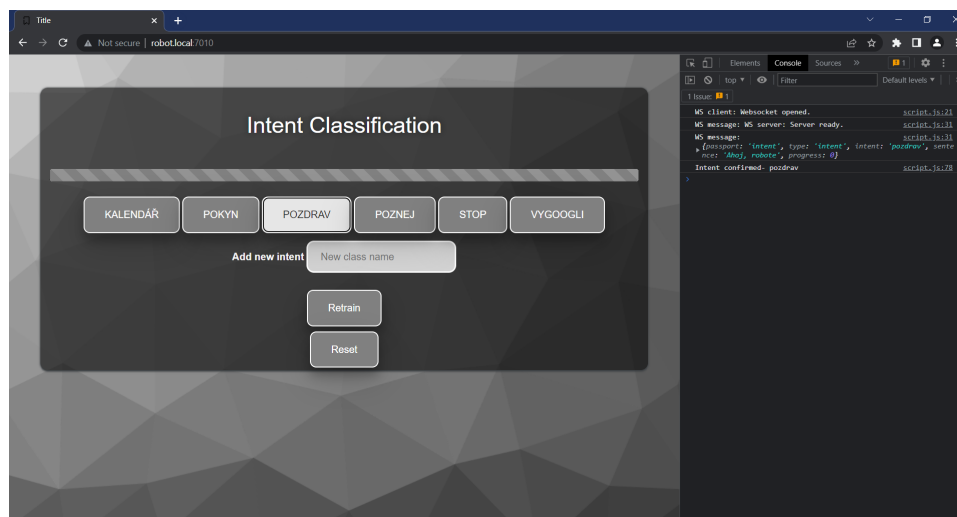
kdy se model přetrénovává je vidět na obrázku 2.12. Po přetrénování, obrázek 2.13, je zobrazena přesnost nového modelu a vzorky z celého datasetu, u nichž model při validaci chyboval. Uživatel má možnost tyto vzorky smazat kliknutím na ikonu koše.



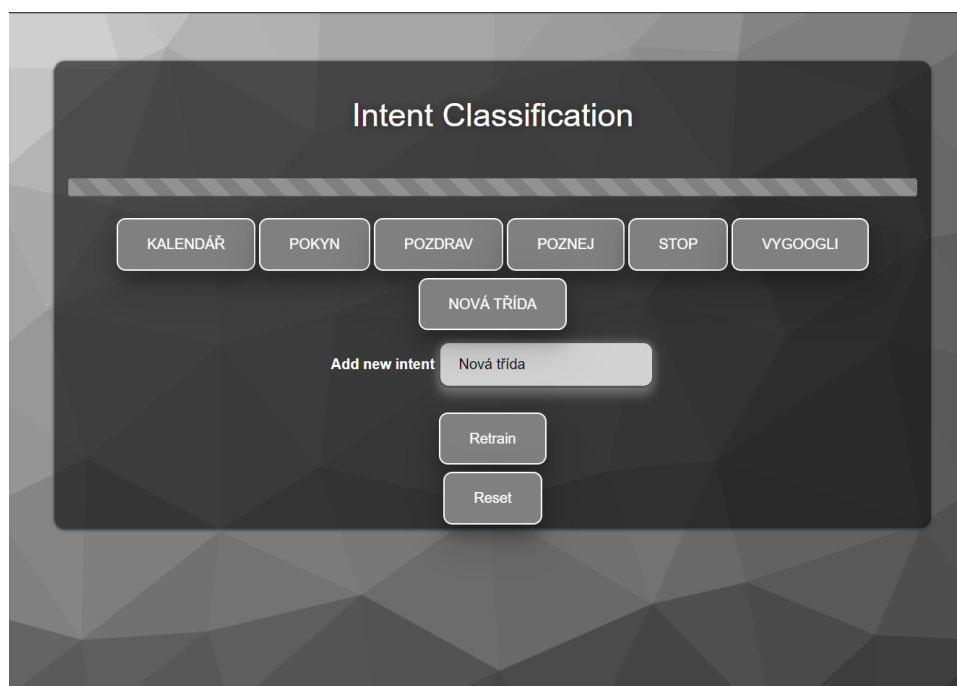
OBRÁZEK 2.7: Celkový náhled na uživatelské rozhraní



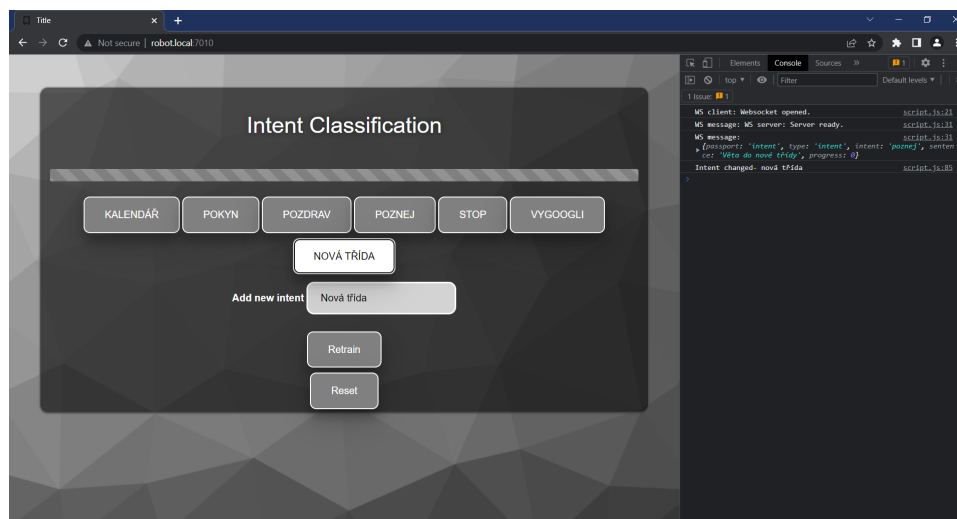
OBRÁZEK 2.8: Zvýraznění predikovaného záměru



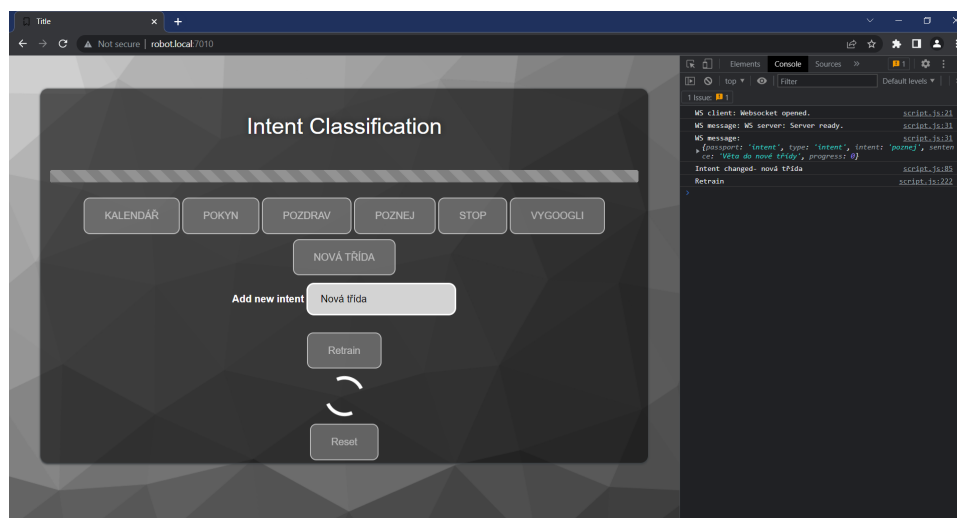
OBRÁZEK 2.9: Potvrzení predikce záměru „pozdrav“ uživatelem



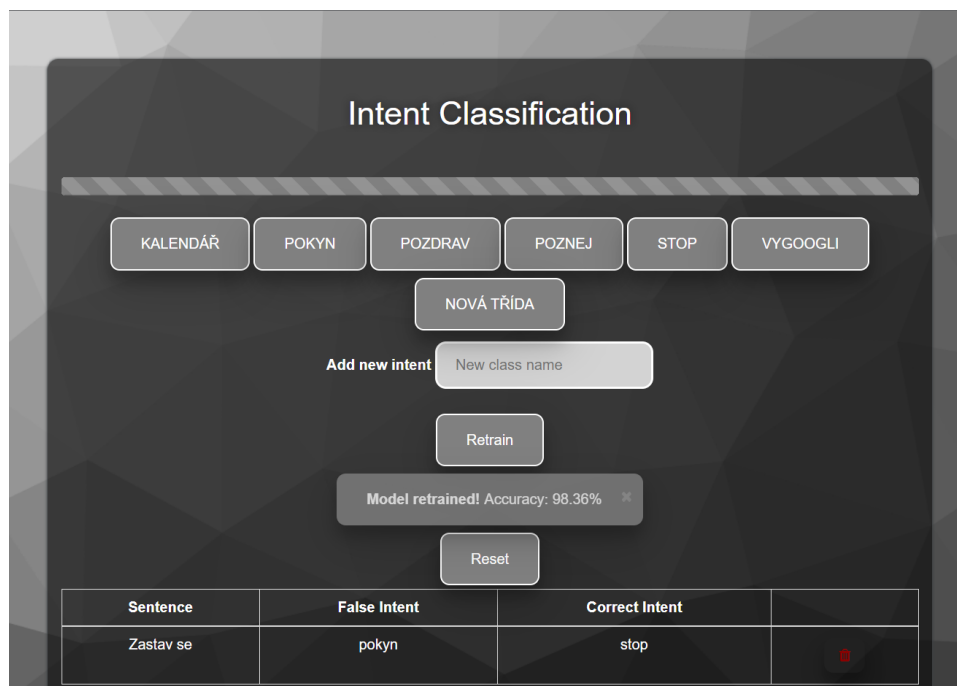
OBRÁZEK 2.10: Přidání nové třídy (nového záměru) s názvem „Nová třída“



OBRÁZEK 2.11: Korekce predikovaného záměru na záměr „Nová třída“



OBRÁZEK 2.12: Proces přetrénování modelu



OBRÁZEK 2.13: Dokončení přetrénování, možnost odstranění chybných vzorků

2.10 Robot.v1

2.11 Přetrénovací smyčka

Kapitola 3

Data

3.1 Malý dataset

Tato trénovací data obsahují 50 různých vzorků, které jsou rozděleny do 5 tříd. Počet vzorků pro každou třídu je tedy 10. Celý dataset je k nahlédnutí viz A1.

Kapitola 4

Examples

4.1 XOR Function

Kapitola 5

Discussion

Discussion starter...

5.1 Recapitulation of Methods

5.2 Summary of Results

Kapitola 6

Conclusion

Conclusion text...

6.1 Future Work

Outlook...

Literatura

- [1] David E. Rumelhart, Geoffrey E. Hinton a Ronald J. Williams. „Learning representations by back-propagating errors“. In: *Nature* 323 (1986), s. 533–536.
- [2] Daniel Svozil, Vladimir Kvasnicka a Jiri Pospíchal. „Introduction to multi-layer feed-forward neural networks“. In: *Chemometrics and Intelligent Laboratory Systems* 39 (1997), s. 43–62.
- [3] Eva Volná. *Neuronové sítě 1*. 2008.
- [4] Morgan Quigley et al. „ROS: an open-source Robot Operating System“. In: *ICRA Workshop on Open Source Software*. 2009.
- [5] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029 0262018020. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.
- [6] Michael A Nielsen. *Neural networks and deep learning*. Sv. 25. Determination press San Francisco, CA, USA, 2015.
- [7] 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.
- [8] 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.
- [9] 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>.
- [10] Sagar Sharma, Simone Sharma a Anidhya Athaiya. „Activation functions in neural networks“. In: *Towards Data Sci* 6.12 (2017), s. 310–316.
- [11] 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>.
- [12] Saugat Bhattarai. *What is gradient descent in machine learning?* 2018. URL: <https://saugatbhattarai.com/np/what-is-gradient-descent-in-machine-learning/>.
- [13] Jongkil Kim et al. „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>.
- [14] 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>.

- [15] 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>.
- [16] Sanket Doshi. *Various optimization algorithms for training neural network*. 2020. URL: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>.
- [17] Martin Bulín. „On Using Multi-Agent Technologies to Build Neural Networks“. Dis. pr. University of West Bohemia in Pilsen, 2021. URL: http://www.kky.zcu.cz/en/publications/MartinBulin_2021_OnUsingMulti-Agent.
- [18] 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>.
- [19] Nikita Sharma. *Exploring optimizers in machine learning*. 2021. URL: <https://heartbeat.comet.ml/exploring-optimizers-in-machine-learning-7f18d94cd65b>.
- [20] Aston Zhang et al. „Dive into Deep Learning“. In: *CoRR* abs/2106.11342 (2021). arXiv: 2106.11342. URL: <https://arxiv.org/abs/2106.11342>.
- [21] 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>.
- [22] 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/>.
- [23] James Briggs. *Sentence transformers and embeddings*. 2023. URL: <https://www.pinecone.io/learn/sentence-embeddings/>.
- [24] *Designing your neural networks*. URL: <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>.
- [25] *SpeechTech* — *speechtech.cz*. <https://www.speechtech.cz/>. [Accessed 25-Feb-2023].

Příloha A1

Malý 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

Záměr	Text
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

Příloha A2

Structure of the Workspace

```
root
├── officials
├── literature
├── data
│   ├── data_mnist
│   └── data_speech
├── py
│   ├── examples
│   │   ├── karnin
│   │   ├── mnist
│   │   ├── rpe
│   │   ├── speech
│   │   ├── train
│   │   └── xor
│   ├── kitt_lib
│   └── scripts
├── results
├── progress_reports
└── thesis
```