VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

STROJOVÉ UČENÍ PRO ODPOVÍDÁNÍ NA OTÁZKY V PŘIROZENÉM JAZYCE

MACHINE LEARNING FOR NATURAL LANGUAGE QUESTION ANSWERING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JONÁŠ SASÍN**
AUTHOR

VEDOUCÍ PRÁCE **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Sasín Jonáš**
Program: Informační technologie
Název: **Strojové učení pro odpovídání na otázky v přirozeném jazyce**
Machine Learning for Natural Language Question Answering
Kategorie: Databáze

Zadání:

1. Seznamte se s metodami odpovídání na otázky v přirozeném jazyce se zaměřením na techniky strojového učení pro extrakci odpovědí, případně přenos výsledků napříč jazyky
2. Shromážděte data pro testování průběžného vývoje i pro závěrečné vyhodnocení úspěšnosti zvolených metod v českém jazyce.
3. Navrhněte a realizujte systém, který s využitím existujících implementací dokáže odpovídat na otázky nad českou wikipedií, případně nad doplňkovými zdroji informací.
4. Vyhodnoťte výsledky systému na shromážděných datech a diskutujte možná zlepšení.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle domluvy s vedoucím

Pro udělení zápočtu za první semestr je požadováno:

- Funkční prototyp řešení

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Práce se zabývá odpovídáním na otázky v přirozeném jazyce s pomocí strojového učení a využitím české Wikipedie jako báze znalostí. Popsány jsou dostupné metody, state of the art a výběr vhodných metod pro daný úkol. Prozkoumány jsou dostupné datové sady pro trénink i vyhodnocení výsledného řešení. Poté je představen návrh výsledného systému a popis implementace jeho jednotlivých komponent. Diskutovány jsou metody vyhledávání relevantních dokumentů a vhodné předzpracování prohledávaných textů i nalezení odpovědi v získaném kontextu. Celý systém je vyhodnocen na známém datasetu s použitím základních metrik. Výsledkem je systém schopný odpovídat na otázky v přirozeném jazyce.

Abstract

This thesis describes natural language question answering using machine learning and searching Czech Wikipedia as its knowledge base. Existing methods, state of the art and choosing a suitable method for the given task is described. Available datasets for training and final system evaluation are explored. The system design is then introduced and the implementation of its individual components is described. Methods for relevant document retrieval, preprocessing the searched corpus as well as finding the answer span in the acquired context are discussed. The whole system is then evaluated on a well known dataset using the basic metrics. The result is a system being able to answer natural language questions.

Klíčová slova

zpracování přirozeného jazyka, NLP, odpovídání na otázky, strojové učení, Wikipedia, otevřená doména, ALBERT, dolování znalostí

Keywords

natural language processing, NLP, question answering, machine learning, Wikipedia, open domain, ALBERT, knowledge mining

Citace

SASÍN, Jonáš. *Strojové učení pro odpovídání na otázky v přirozeném jazyce*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Strojové učení pro odpovídání na otázky v přirozeném jazyce

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže Ph.D. Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jonáš Sasín
11. dubna 2021

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Obsah

Kapitola 1

Úvod

Odpovídání na otázky je dobře prozkoumaný a populární úkol z oblasti zpracování přirozeného jazyka. Využití kolem nás, v běžném životě, můžeme vidět třeba u vyhledávacích nástrojů, komunikačních agentů nebo hlasových asistentů.

Výzkum se v dané oblasti soustředí na metody strojového učení pro vyznačení odpovědi na otázku v daném textu. Pro trénink neuronové sítě, která by byla schopná spolehlivě vykonávat takový úkol, je zapotřebí velké množství anotovaných dat.

Kromě porozumění textu jsou v oblasti důležité také algoritmy pro zhodnocení relevantnosti dokumentů používané všemi vyhledávacími nástroji a metody zpracování textu, které jsou užitečné pro optimální funkci vyhledávacích algoritmů.

Pro češtinu je množství i velikost dostupných datových sad oproti angličtině malé a je tedy obtížné dosáhnout s ní podobných výsledků, jako u jazyků s mnoha zdroji (angličtina, němčina, francouzština ...). Obzvlášť v kontextu toho, že pro většinu úkolů na poli zpracování přirozeného jazyka se v posledních letech nejlépe osvědčily velké modely předtrénované na velmi rozsáhlých korpusech textu. Ty pro češtinu prozatím bohužel nejsou dostupné v dostatečné kvalitě.

V této práci se snažím otestovat přístup, který využívá strojový překlad češtiny do angličtiny, aby mohlo být použito jednoho z robustních předtrénovaných modelů pro porozumění textu. Kromě metod pro porozumění textu a extrakci odpovědi v práci rozebírám algoritmy řazení dokumentů dle jejich relevance k otázce. Jako báze znalostí pro systém schopný na otázky odpovídat je použita česká Wikipedie, jejíž články jsou pro nalezení vhodné odpovědi prohledávány.

Začal jsem popsáním metod pro zpracování textu v kapitole 2. Popsány jsou zde metody reprezentace slov a způsoby zpracování textu vhodné pro porozumění textu a pro algoritmy řazení relevantních dokumentů.

Kapitola 3 se zabývá popisem neuronových sítí pro porozumění textu a extrakci odpovědi. Popsány jsou zde moderní architektury a modely použité v práci. V kapitole 5 jsou poté popsány datové sady vhodné pro trénink a vyhodnocení výsledného systému. Následuje kapitola 6, kde je popsán návrh, kapitola 7 o implementace systému založeného na zmíněných postupech a kapitola 8, kde je výsledný systém vyhodnocen s použitím standardních metrik.

Kapitola 2

Reprezentace slov a zpracování textu

Předtím, než se vrhneme na vysvětlení poměrně složitých jazykových modelů a vyhledávacích algoritmů, bude v této kapitole vysvětleno pár základních pojmů a technik.

Počítač lidské řeči na elementární úrovni příliš nerozumí, jsou mu ale mnohem bližší čísla a statistika. Proto musíme nejdříve porozumět tomu, s jakou reprezentací slov moderní jazykové modely pracují a do jaké podoby je tedy potřeba zdrojový text dostat. Vysvětleny jsou tedy termíny související s reprezentací slov pomocí vektorů, neboli „word embeddings“ a tokenizace textu.

Také je potřeba prozkoumat možnosti zpracování textů a slov na jejich morfologické úrovni. Tato úloha je na pomezí informatiky a lingvistiky a používají se pro ni speciální nástroje. Použité techniky jako lemmatizace a odstranění stop-slov zajistí optimální funkčnost některých použitých algoritmů.

Po přečtení kapitoly by měl být čtenář teoreticky vybaven pro pochopení konceptů vysvětlených v kapitolách 3 a 4, které na zde popsaných technologiích staví a částečně se s nimi překrývají.

2.1 Reprezentace slov

Jak už bylo řečeno, je složité pracovat s textem v takové podobě, na jakou jsou lidé zvyklí. Počítač v ní nedokáže vidět důležité sémantické souvislosti, které jsou pro přirozený jazyk tak důležité. Musíme tedy z textu a slov v něm získat nějakou matematickou reprezentaci, která je počítači bližší a se kterou dokáží dále pracovat neuronové sítě.

Pro takovou reprezentaci jsou nejpoužívanější vektory pevně dané dimenze známé jako „word embeddings“. Pro představu například (běžně) 300 desetinných čísel pro popis každého slova v textu. Vektory slov ve slovníku nesou důležitou informaci o sémantické příbuznosti jednotlivých slov. Po promítnutí do spojitého prostoru jsou si vektory, které reprezentující sémanticky podobná slova, blízko, jak zobrazuje obrázek 2.1.

Po aplikaci těchto myšlenek zmíněných v článcích [14] a [15] byl zaznamenán výrazný pokrok v úlohách z oblasti zpracování přirozeného jazyka, jako strojový překlad, porozumění textu, odhad emocí, klasifikace atp. Vektory reprezentující slova jsou většinou získávány analýzou rozsáhlých textových korpusů. Snaží se zachytit sémantickou blízkost slov na základě jejich koexistence v podobných kontextech.

Například slova *pohovka* a *gauč* se nejspíše budou vyskytovat v podobných situacích, protože jsou to synonyma a jejich sémantika je téměř totožná.

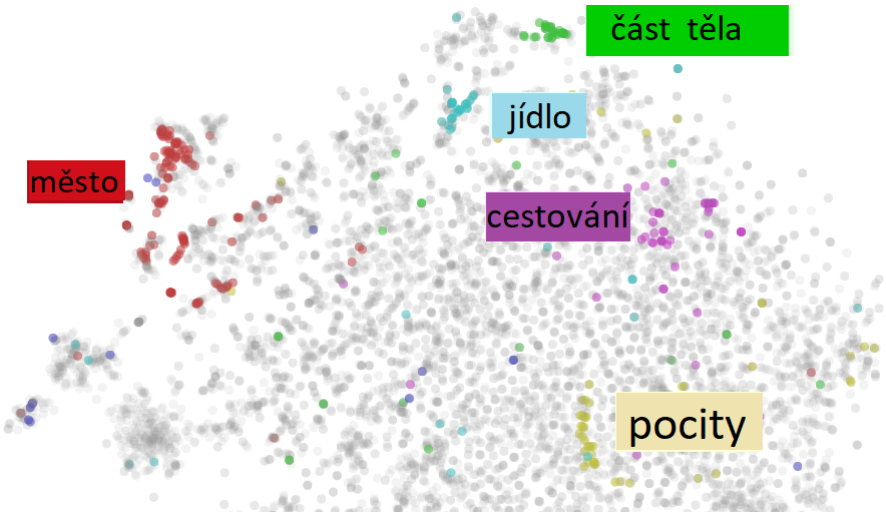
Výsledné vektory nám poté umožňují provádět se slovy také velmi zajímavé matematické operace, jako je sčítat, nebo odčítat a je schopen zachytit velmi zajímavé vztahy jako například:

$$[\text{Král}] - [\text{Muž}] + [\text{Žena}] = [\text{Královna}]$$

nebo

$$[\text{Paříž}] - [\text{Francie}] + [\text{Čína}] = [\text{Peking}]$$

V následující části budou popsány vlastnosti a nejznámějších metod pro získání vektorových reprezentací.



Obrázek 2.1: Část vektorového prostoru znázorňující reprezentace slov, které spojuje nějaké téma. (https://ruder.io/word-embeddings-1/))

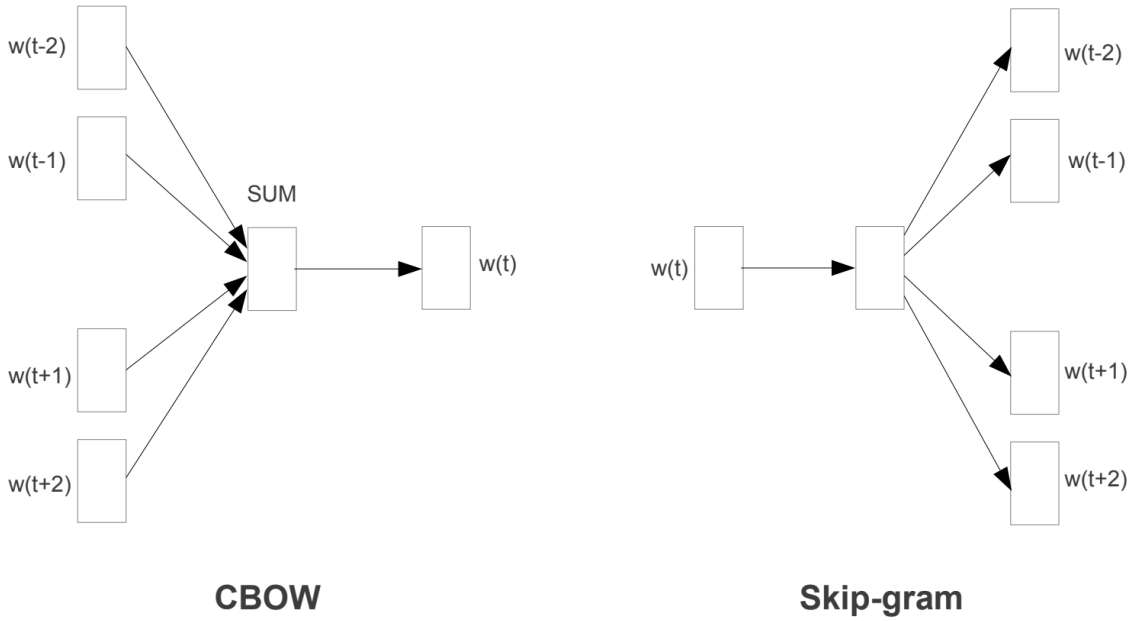
2.1.1 Word2vec

Word2vec je technika prezentována v článku [14] schopná naučit se vektorové reprezentace pomocí prediktivního modelu neuronové sítě. Základním konceptem dvou přístupů, které se pro trénink modelu používají, je predikce slov na základě sousedních slov v tzv. kontextovém okně, které se postupně posouvá. Dostáváme tedy pro každé slovo pravděpodobnost, se kterou se bude vyskytovat v blízkosti slova cílového.

První model používaný pro trénink word2vec vektorů se nazývá Continuous Bag of Words, neboli CBOW. Neuronová síť se trénuje hádáním chybějícího slova v kontextu známých sousedním slov kontextového okna.

Druhá metoda funguje funguje na opačném principu a nazývá se Continuous Skip-gram Model. Na základě současného slova se tedy model snaží předpovědět slova v určitém vzdálenosti před, i po současném slově, v závislosti na velikosti kontextového okna. Tato metoda se z článku [14] jeví jako o něco úspěšnější a je dále zdokonalena v článku [15].

Na obrázku 2.2 můžeme vidět rozdíl v tom, jak projekční vrstva transformuje vstup na výstup v případě obou architektur.



Obrázek 2.2: Obrázek architrektury CBOW přepovídající slovo na základě kontextu a Skip-grap předpovídající kontext na základě slova. Převzato z [14].

2.1.2 GloVe

GloVe, neboli Global Vectors for Word Representation je technika prezentována v článku [17]. Výsledkem je vektorový prostor o pevně dané dimenzi stejně, jako u word2vec reprezentace. Liší se však způsobem získání daných reprezentací.

Zatímco word2vec je prediktivní model, GloVe je spíše modelem statistickým. Klade tedy důraz hlavně na počet výskytů slov a počet jejich výskytů v podobných kontextech. Dostaneme tedy nějakou matici společných výskytů slov v korpusu.

Jeho hlavní výhodou je, že se nesoustředí pouze na slova v těsném kontextu, jak už slovo *Global*, ukryté v GloVe, napovídá. Reprezentace je tedy schopná zachytit sémentické souvislosti v širším kontextu, ale základní idea není konceptu word2vec příliš daleko.

2.1.3 Wordpiece embeddings

Wordpiece je prediktivní technika prezentována článkem [26] v roce 2016 (původně představeno pro strojový překlad) a používána modely popsanými v sekci 3.3.

Hlavním záměrem přístupu je lépe se vypořádat s ojedinělými slovy tak, že jsou slova rozdělena do již známých *podslov*. Nabízí tím kompromis mezi *word embeddings* pro celá slova a tzv. *character embeddings* pro jednotlivé znaky.

Slovník je nejprve naplněn všemi znaky v textu a jazykový model je na tomto slovníku trénován. Z výsledného slovníku modelu je poté sestaven slovník nových kombinací jednotlivých znaků tak, aby se zlepšila úspěšnost na trénovací datové sadě. Postup se opakuje, dokud nenarazíme na limit minimální velikosti slovníku, nebo je rozdíl mezi úspěšností jednotlivých iterací modelu příliš malý.

2.2 Tokenizace a předzpracování

Zpracování textu je důležitou součástí oblasti zpracování přirozeného jazyka. S textem se totiž pracuje lépe, pokud jsme schopni odstranit některé záludnosti a nadbytečnosti přirozeného jazyka a dostat ho do podoby, který je pro automatické zpracování vhodnější. Jednotlivé nástroje pro efektivní zpracování textu v práci jsou poté uvedeny v části 7.1.

První část této sekce navazuje na část 2.1 o reprezentaci jednotlivých slov textu a zabývá se postupem rozdělení textu na tzv. *tokeny*, které mohou být později převedeny na jejich vektorovou reprezentaci.

Druhá část je věnována postupům pro zpracování a pokud možno normalizaci textu pro optimální využití vyhledávacích algoritmů. Statistické metody využívány těmito algoritmy jsou efektivnější, pokud jsou z textu odstraněny například slova nesoucí zanedbatelnou informaci. Mohou být také zmateny různými tvary daných slov.

2.2.1 Tokenizace vstupního textu

Tokenizace je převod textu jako posloupnosti jednotlivých znaků na posloupnost jednotlivých tokenů. Například rozdělení textu „Adam šel po škole domů.“ na posloupnost jednotlivých tokenů:

$$\_\text{Adam} + \_\text{šel} + \_\text{po} + \_\text{škole} + \_\text{domů} + \_.$$

Primitivní přístup je rozdělení textu pomocí bílých znaků a interpunkčních znamének, který ale nebere ohled např na tečky uprostřed zkratek v angličtině a těžko se vypořádává se složeninami. Získané tokeny je pak taky třeba normalizovat pro následný převod na word embeddings.

V přístupu uvedeném v sekci 2.1.3 jsou slova děleny na tokekeny podle toho, která podslova v daném textu jsou slovníku známy. Můžeme si uvést příklad z angličtiny, s jímž podobným se ještě jistě setkáme i v sekci 3.3. Věta „I like playing.“ by tedy byla převedena na následující tokeny:

$$\_\text{I} + \_\text{like} + \_\text{play} + \_\text{ing} + \_.$$

Na uvedeném případě je vidět hlavně převedení posloupnosti znaků „playing“ na dvě podslova (tokeny) *play* a *ing*.

2.2.2 Normalizace délky textu a rozdělení do vět

V kontextu práce je normalizací délky textu myšlena jakási standardní maximální délka dokumentu, který je později předložen readeru k vyznačení správné odpovědi. Tento přístup je užitečný ze několika důvodů.

Algoritmy pro řazení dokumentů dle relevance sice berou na relativní délku každého textu ohled, nicméně některé články jako [10] nasvědčují, že můžou být příliš dlouhé dokumenty znevýhodněny. Článek [10] sice představuje úpravu klasického algoritmu pro minimalizaci tohoto problému, v práci jsou ale i další důvody, proč je vhodné délku jednotlivých odstavců normalizovat.

Model pro vyznačení správné odpovědi v textu je trénován na datové sadě, která příliš dlouhé dokumenty nebere v úvahu[1] a jak je naznačeno v článku [27], není na dlouhých dokumentech příliš úspěšný.

[1] Pro každou otázku je jen jeden odstavec kontextu, ne celý článek z Wikipedie.

Dalším důvodem je snaha nepřekládat pro účel vyznačení správné odpovědi příliš dlouhé úseky textu.[2]

Proč rozdělit text do vět? Předpokládejme, že je text již rozdělen do jednotlivých odstavců. Pro každý odstavec je kontrolována jeho délka, jestli nepřesahuje délku maximální. Pokud bude maximální délka odstavce překročena, může být sice rozdělen jednoduše na poloviny, na třetiny a tak dále podle potřeby, bude tím ale ztracena velká část kontextu při rozdělení některých vět v půlce.

Pro takový případ je vhodné vědět, kde jednotlivé věty začínají a končí, aby bylo možné jednotlivé odstavce případně smysluplně rozdělit. Vhodné je také zařídit, aby se jednotlivé rozdělené „pododstavce“ částečně překrývaly a obsahovaly třeba 3 poslední věty odstavce předchozího, pro maximální zachování původního kontextu.

2.2.3 Převod na malá písmena

Převod na malá písmena je sice velmi jednoduchý, ale přesto užitečný úkol. Při psaní otázky na klávesnici je velmi pravděpodobně, že pomyslný uživatel napíše „sssr“ místo „SSSR“ nebo „Albert“ místo „ALBERT“, ale bylo by vhodné, aby byly nalezeny dokumenty obsahující obě varianty. Stejně tak může dojít k nestandardnímu užití malých/velkých písmen v některém z dokumentů.

Pro některé případy může převod na malá písmeny znamenat ztrátu části kontextu. Například *Malá Strana* X *malá strana*, kde velké písmeno indikuje vlastní jméno (městskou část). Vhodné je to tedy pouze v některých případech.

2.2.4 Odstranění stop slov

Ve zpracování přirozeného jazyka jsou *stop slova* většinou souborem nejčastěji používaných slov v daném jazyce[3]. Většinou jsou odstraňována, protože kvůli jejich vysokému výskytu nesou menší informační hodnotu a jejich odstranění vede k lepší funkci vyhledávacích algoritmů [23].

Nejužívanější slova daného jazyka je možno například ve frekvenčních seznamech jako [25], který obsahuje 100 nejpoužívanějších slov v češtině. Získaný seznam je vhodné kriticky zhodnotit a vyčlenit z něj některá slova, která by pro vyhledávání mohla mít kritický význam.[4] Většinou je zanedbatelná alespoň většina spojek, předložek či zájmen. Typickými příklady stop slov pro češtinu jsou: být, a, se, v, že ...

2.2.5 Získání lemmat

Lemmatizace je druh zpracování textu, při kterém je pro každé slovo nalezen základní tvar, tzv. *lemma*.[5]

ulicí běžely děti ⟶ ulice běžet dítě
ostrovy Středozemního moře ⟶ ostrov Středozemní moře

[2] Hlavně kvůli určitým omezením dostupných nástrojů pro překlad.
[3] https://en.wikipedia.org/wiki/Stop_word
[4] Příklad: „první“, „země“, „člověk“ ...
[5] Podobným postupem je *stematizace*, která ale algoritmicky pouze odstraní koncovky a předpony pro nalezení slovního kmene. Lemmatizátor pro nalezení základních tvarů používá morfologický slovník.

Pro získání lemmat se používají speciální nástroje pro zpracování přirozeného jazyka. *Lemmatizátory* mohou poskytovat také některé doplňkové informace o mluvnichkých kategoriích jako rod, pád nebo slovní druh. Hlavním úskalím lemmatizátorů je mnohoznačnost jazyků jako je čeština, kdy může být více základních tvarů daného slova dle kontextu [6].

Využití lemmatizace je různé, ale v případě této práce je podobné, jako účel postupu uvedeného v sekci 2.2.3. Jednoduše se hodí, když pro dotaz obsahující „. . . Liparských ostrovů“ budou nalezeny i dokumenty obsahující základní tvar hledaného výrazu „Liparské ostrovy“

Lemmatizace je také užitečným nástrojem při odstranění stop slov, kdy můžeme například odstranit všechny tvary slovesa „být“, zatímco seznam stop slov může obsahovat pouze jeho základní tvar.

[6] https://cs.wikipedia.org/wiki/Lemmatiz%C3%A1tor

Kapitola 3

Strojové učení pro extrakci odpovědi

Rozmach strojového učení umožnil obrovský pokrok v oblastech NLP jako strojový překlad, rozpoznání entit, generování textu nebo také odpovídání na otázky. Proto na tomto přístupu staví všechny nejmodernější metody a tímto směrem se také ubírá výzkum.

Účelem této kapitoly je vysvětlit čtenáři, jak fungují neuronové sítě a jakým způsobem dokáže počítač porozumět textu a naučit se vyznačit v textu odpověď na danou otázku.

Nejprve budou ve zkratce vysvětleny principy fungování a učení neuronových sítí. Poté budou rozebrány nejznámější architektury neuronových sítí používané pro vypořádání se s úkoly na poli zpracování přirozeného jazyka a nakonec budou popsány nejmodernější modely, které na nich staví.

3.1 Neuronové sítě

V této části bude vysvětlen princip fungování neuronových sítí. Informace v sekci byly převzaty z článku [22] a z prezentací Stanfordského kurzu NLP.[1]

Neuronové sítě si můžeme zjednodušeně představit jako velkou funkci s obrovským množstvím parametrů, která přetváří vstupy na výstupy. V průběhu tréninku neuronové sítě se postupně všechny parametry optimalizují tak, aby síť přetvářela vstupy na výstupy co nejlépe.

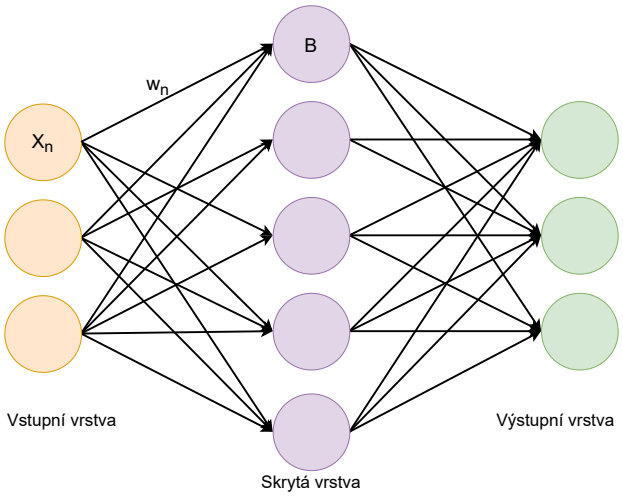
Fungují tu tři základní jednoduché vrstvy(3.1). Vstupní, skrytá (zde probíhá proces přetváření vstupů) a výstupní, na které se přetvořený vstup objeví. Skrytá vrstva se navíc běžně skládá z mnoha dalších vrstev.

Každá z těchto vrstev, ať už je vstupní, výstupní, nebo skrytá, se skládá z dalších primitiv a ty jsou nazývány jak jinak, než neurony. Každý neuron v každé vrstvě je propojen s každým neuronem vrstvy další, a tak je propojena celá síť.

3.1.1 Umělý neuron a aktivační funkce

Umělý neuron je primitivní jednotka inspirována neuronem skutečným. Ten vypadá asi tak, že má tělo, do kterého přichází krátkými výběžky, tzv. *dendrity* elektrické signály z ostatním neuronů. Z těla tohoto neuronu pak vychází jediný dlouhý výběžek, tzv. *axon*.

[1] Dostupné z: http://web.stanford.edu/class/cs224n/



Obrázek 3.1: Schéme neuronové sítě s jednou skrytou vrstvou. Znázorněn je také práh neuronu B, jeden vstup z předchozí vrstvy x_n a váha jednoho propojení w_n.

Neuron se tedy na základě signálů, které do něj přichází dendrity rozhodne (na základě nějaké minimální hranice přijatého signálu), zda-li bude, nebo nebude vysílat signál dál [2].

Funkce umělého neuronu je analogická. Jak už bylo popsáno v úvodu sekce 3.1, každý neuron je propojen s každým neuronem vrstvy předcházející i vrstvy další. Na rozdíl od skutečného neuronu má tedy „více axonů“, kterými je propojen s další vrstvou.

Každé propojení mezi každým neuronem má svoji váhu w, která ovlivňuje důležitost jednotlivého vstupu do neuronu. Každý neuron má pak svůj práh B a aktivační funkci. Výstupem neuronu je tedy vážená suma vstupů do neuronu s přičteným prahem a aplikovanou aktivační funkcí, neboli

$$F(\sum_{i=1}^{n}(x_n w_n) + B) \tag{3.1}$$

kde x_n je neuron předcházející vrstvy, w_n je váha jeho propojení, B je práh aktuálního neuronu a F je aktivační funkce. Znázornění vah propojení a jednotlivých neuronů je vyznačeno na obrázku 3.1.

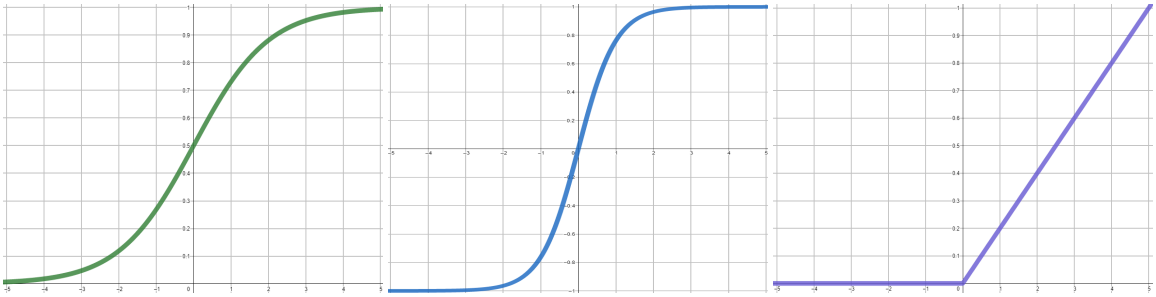
Všechny propojení jednotlivých neuronů a jejich prahy v dávají dohromady parametry neuronové sítě, které jsou v průběhu tréninku optimalizovány pro každý neuron zvlášť. Pro síť, jako je na obrázku 3.1 je to třeba 30 jednotlivých vah propojení a 8 prahů jednotlivých neuronů (skryté a výstupní vrstvy), tedy 38 parametrů v tak jednoduché síti.

Aktivační (přenosová) funkce rozhoduje, jestli bude daný neuron aktivován. Skutečný neuron má, zdá se, funkci skokovou. Umělé neurony však používají funkce s postupnou aktivací. Běžné jsou tři (čtyři) základní varianty aktivačních funkcí:

- (Skoková)
- Sigmoidální
- Hyporbolické tangenty
- ReLU (Rectified Linear Unit)

[2] https://cs.wikipedia.org/wiki/Neuron

Funkce jsou znázorněny na obrázku 3.2. Vhodnost použití každé z nich může být různé. Výhodou sigmoidální funkce je normalizace výstupu do rozmezí $(0, 1)$. Nejpoužívanější funkcí pro skrytou vrstvou je však funkce ReLU. Kvůli její charakteristice je aktivováno méně neuronů, což vede k rychlejšímu tréninku a konvergenci. Použití ReLU funkce je také méně výpočetně náročné [1].



Obrázek 3.2: Jednotlivé aktivační funkce, zleva: sigmoidální ($H(f) = \langle 0, 1 \rangle$), hyperbolické tangenty (tanh $H(f) = \langle -1, 1 \rangle$) a ReLU ($H(f) = \langle 0, \text{inf} \rangle$).

3.1.2 Dopředná a zpětná propagace

Proces, kdy se opakovaně ze vstupů počítá výstup každého neuronu v dané vrstvě a stejně tak pro každou následující vrstvu, se nazývá dopředná propagace. Tímto postupem se postupně hodnoty vstupní vrstvy přetvoří na hodnoty ve vrstvě výstupní a může se určit, jak byla síť úspěšná.

Pro určení úspěšnosti každé takové iterace neuronové sítě se používá nákladová funkce (anglicky cost function pro jednu iteraci, loss function pro průměr všech), která udává jediné číslo charakterizující úspěšnost modelu. Nejznámější nákladovou funkcí je Střední kvadratická chyba (anglicky Mean Squared Error - MSE). Cílem je hodnotu této nákladové funkce postupně minimalizovat.

Postup optimalizace jednotlivých parametrů neuronové sítě pro zajištění minimalizace nákladové funkce se nazývá **zpětná propagace**. Je to v podstatě opačný průchod neuronovou sítí, který má za úkol zjistit, které neurony se největší mírou podílely na výsledné chybě. Snížením významu těchto neuronů můžeme dosáhnout lepší úspěšnosti celé sítě. Naopak můžeme zvýšit význam propojení s neurony, které výstup ovlivnily pozitivně.

Základem tohoto výpočtu je, že chyba každého výstupního neuronu je vstupem parciální derivace vzhledem ke každému z jeho vstupů. Celý postup zpětné propagace je velmi složitý a pro intuitivní pochopení problematiky není klíčový, proto tu nebude podrobně rozebrán.

Celý postup zpětné propagace je opakován a parametry neuronové sítě jsou postupně optimalizovány.

Trénink neuronové sítě tedy probíhá následovně. Síti s náhodně inicializovanými parametry je předložena část dat z trénovací datové sady. Ty jsou pomocí dopředné propagace prostupem přes skrytou vrstvu přetvořeny na výstupy a tím získány predikce neuronové sítě. Predikce jsou porovnány se základní pravdou (tzv. *ground truth*) a nákladovou funkcí je vyjádřeno, jak byly predikce daleko od pravdy. Pomocí zpětné propagace jsou postupně upraveny všechny parametry neuronové sítě.

Postup je opakován, dokud úbytek nákladové funkce mezi jednotlivými iteracemi nezačne dlouhodobě stagnovat, nebo dokud nedosáhneme požadovaného počtu epoch.

Při nesprávném tréninku neuronové sítě může dojít k podtrénování (underfitting), nebo přetrénování (overfitting). Problémy při tréninku souvisí s tzv. hyperparametry, mezi které patří počet skrytých vrstev, velikost modelu nebo *learning rate*[3]. Pomoct může také množství trénovacích dat nebo počet trénovacích epoch.

3.2 Nejlepší architektury pro porozumění textu

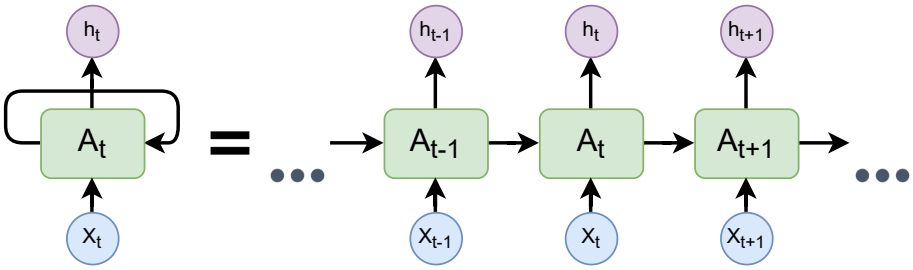
V této sekci budou popsány dvě nejznámější a nejpoužívanější architektury používané pro zpracování přirozeného jazyka. Rozšiřují koncepty prezentované v sekci 3.1 a snaží se překonat problémy, které tradiční model představuje.

První model v části 3.2.1 využívá k překonání ztráty kontextu rekurentní neuronové sítě. Druhý model v části 3.2.2 spoléhá na paralelní zpracování a vzájemnou pozornost mezi otázkou a kontextem.

3.2.1 Long short-term memory

Informace v této sekci byly převzaty z originální práce [6], která architekturu představuje a ze článku [4].

LSTM, neboli Long Short-term Memory je speciální druh *rekurentní* neuronové sítě, který se částečně vypořádává s problémem ztráty kontextu v delším úseku textu. Jednoduše řečeno, při učení neuronové sítě bychom byli rádi, aby pochopení současné věty dokázala ovlivnit i věta předcházející.



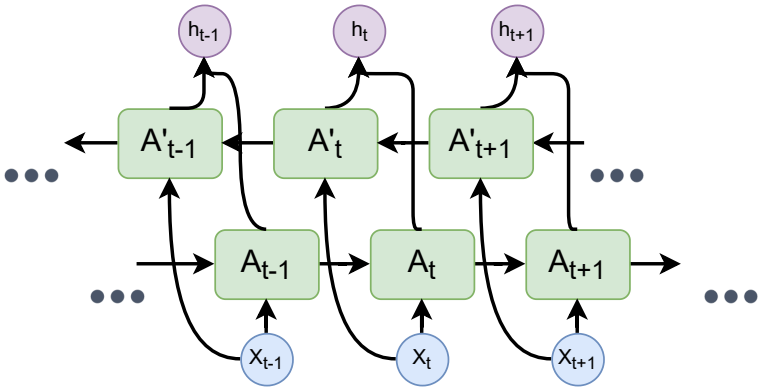
Obrázek 3.3: Schéma rekurentní neuronové sítě inspirován obrázkem z [4]. X_t je vstupem neuronové sítě A_t v čase t a výstupem h_t.

Nejprve bude vysvětlen princip **rekurentní neuronové sítě** (dále RNN - Recurrent Neural Network). Jejich hlavním úkolem je vypořádat se se zpracováním sekvenčních dat, jako je text. Oproti například rozpoznávání vzorů v obrázcích není možné pohlížet na každé slovo v textu zvlášť. Slovo/slova předcházející by měly ovlivnit slovo současné. Stejně tak by měly následující slova ovlivnit význam slova současného (obousměrné rekurentní sítě).

Schéma jednoduché RNN je vidět na obrázku 3.3. Zjednodušeně je to tedy více jednoduchých neuronových sítí propojených za sebe. Vstupem každé sítě h_n je výstup předchozí sítě h_{n-1} a slovo x_n. Výstup y_n je potom vstupem do další jednotky h_{n+1}.

Analogicky funguje obousměrná RNN, které je propojená i v opačném směru. Vstup je zpracováván od konce a pro současné slovo je tedy brán v úvahu i kontext slova nadcházejícího, jak je vidět na obrázku 3.4.

[3]Konstanta určující velikost kroku při každé iteraci zpětné propagace při úpravě parametrů neuronové sítě.



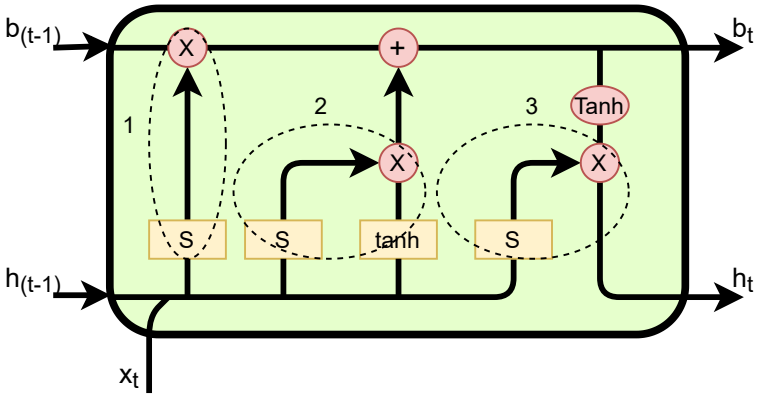
Obrázek 3.4: Schéma obousměrné rekurentní neuronové sítě.

LSTM (Long Short-term Memory) představuje vylepšení standardní RNN.

Mějme model, který má za úkol předpovědět následující slovo ve větě. Pokud se kontext potřebný pro predikci nachází v těsné blízkosti předpovídaného slova (př.: „vychlazené mléko z *lednice*“), je i běžná RNN schopná si souvislost propojit. Pro příklad, kdy je kontext „dále v minulosti“ (př.:„Učím se od 6 let anglicky . . . Vzhledem k těmto okolnostem plynule ovládám *angličtinu*“) není RNN schopná naučit se tuhle souvislost mezi kontextem a hledaným slovem.

LSTM je tedy narozdíl od klasické RNN navržená tak, aby byla schopná pamatovat si dlouhodobé souvislosti v textu při sekvenčním zpracování. Řeší tím tzv. *problém mizejícího gradientu*.

Model má stejnou opakující se strukturu jako RNN, rozdíl je však v komplexnosti opakující se jednotky (či buňky). Struktura naznačená na obrázku 3.5 je poměrně složitá a jejím hlavním komponentem jsou tzv. *brány*.



Obrázek 3.5: Schéma LSTM buňky převzato z článku [4]. b_t je vnitřní stav buňky, $b_{(t-1)}$ předcházející buňky, h_t výstup buňky, $h_{(t-1)}$ výstup předcházející. x_t je vstup současné buňky. „1“ je *zapomínací*, „2“ *vstupní* a „3“ *výstupní* brána.

Brány jsou v LSTM buňce celkem tři. Každá je složena z vrstvy neuronové sítě se sigmoidální funkcí („S“ v 3.5) a násobícího („×“ v 3.5) operátoru. Brána je komponentem určujícím, kolik informace je možno propustit. Brány budou popsány zleva doprava.

První, tzv. zapomínací brána určuje, kolik vstupní informace je zahozeno[4]. Druhá, tzv. vstupní brána, určí, které hodnoty skrytého stavu buňky budou aktualizovány a vrstva s funkcí *tanh* („tanh“ v 3.5) rozhodne o nových kandidátních hodnotách, které po kombinaci s výstupem sigmoidální vrstvy vytvoří nový skrytý stav buňky. Poslední bránou je výstupní, která spojuje informaci z brány zapomínací a vnitřním stavem buňky a vytváří tím tedy výstup dané buňky.

Jednotlivé LSTM buňky jsou za sebou sekvenčně propojeny stejně jako RNN. Při použití dvou LSTM sítí s tou druhou propojenou v opačném směru získáme obousměrnou LSTM síť. Výstupy obou sítí jsou poté pro získání výstupu zřetězeny.

3.2.2 Transformers

Informace v této sekci byly převzaty z originální práce *Attention Is All You Need* [24], práce [2] a článku [7]. Sekce popisuje Transformer model používaný pro zpracování přirozeného jazyka a mechanismus pozornosti, který s ním úzce souvisí.

Přístup použitý tímto modelem se dokázal oprostit od rekurentních modelů jako LSTM a nahradil je na poli zpracování přirozeného jazyka jako nová state of the art architektura. Transformers nepoužívá sekvenční zpracování slov, ale zpracování paralelní.

Jeho nejdůležitější myšlenkou je tzv. **mechanismus pozornosti** (anglicky *attention mechanism*) představený článkem [2] původně pro strojový překlad. Mechanismus pozornosti se však dále rozvíjel a popularizoval do dalších odvětví strojového učení. Stručně bude tedy popsán.

Mechanismus vezme vektory dvou vět přirozeného jazyka naráz a sestaví z nich *key(**K**)-value(**V**)* a *query(**Q**)* matice (řádek – vektor pro každé vstupní slovo). Ty vzniknou vynásobením vstupní word embedding matice speciální váhovou maticí, jejíž parametry se natrénují. Na matice jsou poté aplikovány následující operace pro získání výstupu attention mechanismu

$$softmax(\frac{Q \times K}{k}) \times V \tag{3.2}$$

kde k je podle [24] konstanta 8 (mohly by být stanoveny i jiné hodnoty) a softmax funkce pro normalizaci výstupu do rozmezí (0,1).

Vektory pro všechny vstupní slova respektive vstupní matice K, V a Q mohou být ze stejné věty. Potom se používá termín **self-attention**.

Tím je určeno, která část vstupu je relevantní (self-attention), nebo například která část dokumentu je relevantním kontextem dané otázky.

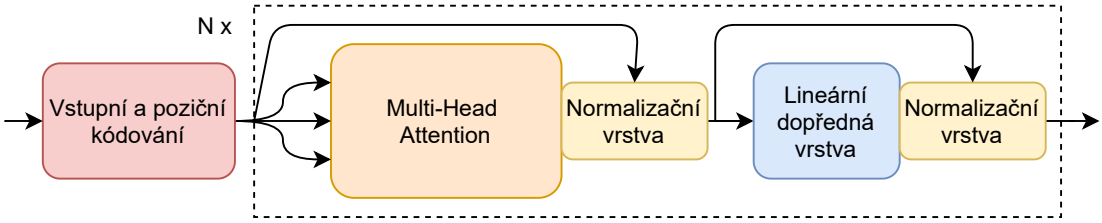
Jednoznačnou výhodou mechanismu pozornosti je, že je schopen podívat se na celý kontext naráz (narozdíl od RNN, která prochází vstup sekvenčně) a vyznačit v něm důležité pasáže. Tímto se zbavuje problému mizejícího gradientu.

Transformer enkodér[5][6] je model, jehož architektura je kombinací dopředné neuronové sítě a mechanismu pozornosti. Jak už bylo popsáno, jeho hlavní výhodou je vlastnost *mechanismu pozornosti* zpracovávat vstupní slova paralelně, a tím se naučit i vztahy mezi slovy na dlouhé vzdálenosti v textu.

[4] Výstupem je číslo v rozmezí (0,1), kde 0 znamená „všechno zahodit“ a 1 znamená „všechno propustit“.

[5] Enkodér je část modelu, která vytvoří z vektorů jednotlivých slov na vstupu vektor celého vstupního textu, reprezentující jeho význam.

[6] V původním článku představen model pro *sequence to sequence* model vhodný pro překlad, který obsahuje i dekodér pro generování textu.



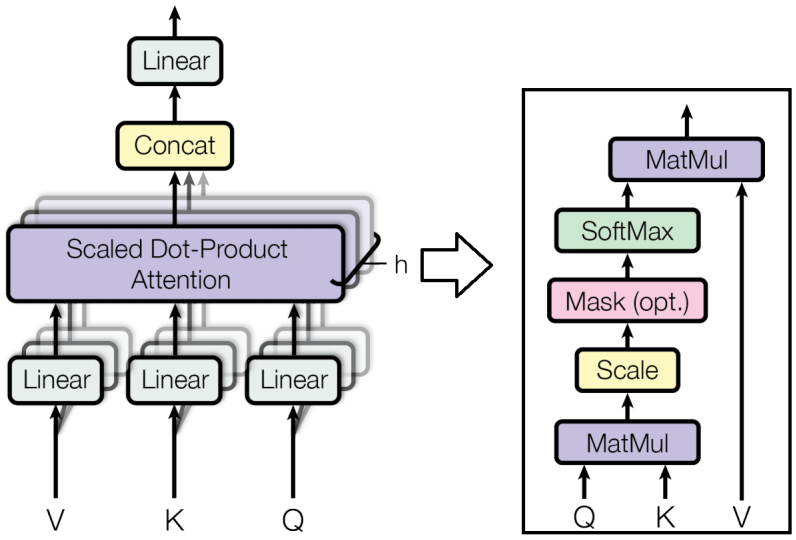
Obrázek 3.6: Schéma architektury transformer enkodéru převzaté z článku [24]. V článku je uvedeno $N = 6$ pro počet identických vrstev ekodéru.

Na obrázku 3.6 můžeme vidět zjednodušenou strukturu jednotky enkodéru. V článku [24] tvoří enkodér 6 takových bloků propojených za sebe.

Před vstupem do prvního bloku jsou jednotlivé tokeny vstupního textu převedeny na vektory (word embeddings), ke kterým je přidáno poziční kódování[7].

Každý blok se pak skládá z „*multi-head attention*“ modulu a plně propojené dopředné vrstvy. Obě části jsou následovány normalizační vrstvou.

Nejzajímavější částí bloku enkodéru je multi-head attention modul, který je dále rozkreslen na obrázku 3.7. Modul je sofistikovanou implementací self-attention mechanismu popsaném dříve, který dokáže vyzdvihnout souvislost jednotlivých částí celého kontextu.



Obrázek 3.7: Schéma *Multi-Head* attention vlevo a *Scaled Dot-Product* attention vpravo. *Multi-head* attention je aplikováno paralelně v h vrstvách (pro práci [24] $h = 8$). Obrázky byly převzaty z článku [24].

Mechanismus ne obrázku 3.7 vpravo je popsaný rovnicí (3.2) a je hlavním komponentem multi-head attention modulu znázorněném v 3.7 vlevo.

Multi-head znamená, že do modulu vstupuje 8 (dle práce [24]) trojic V,K,Q. Každá matice trojice V,K,Q je transformována lieární vrstvou a na celou trojici je potom aplikován *scaled dot-product attention* (self-attention) mechanismus. Tento postup je aplikován na každou z osmi trojic paralelně, pokaždé však s jinými váhami lineární vrstvy.

[7] identifikuje pořadí tokenu ve vstupním textu.

3.3 Předtrénované modely BERT a ALBERT

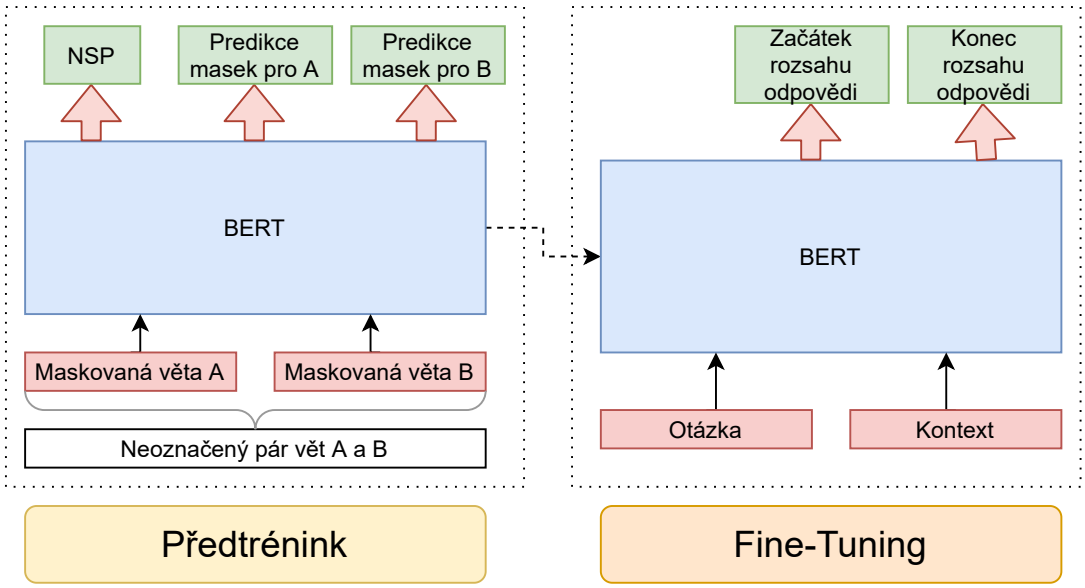
V této sekci budou představeny modely založené na architektuře Transformer popsané v sekci 3.2.2. Informace v této sekci jsou převzaty z původních článků [5] a [8].

Předtrénované modely stavějící na této architektuře určily nový standard pro přicházející state of the art metody. Kromě jejich vysoké úspěšnosti je obrovskou výhodou také znovupoužitelnost předtřénovaných modelů. Modely je možno jednoduše dotrénovat (provést tzv. *fine-tuning* po načtení modelu s předtrénovanými parametry) pro celou řadu úkolů na poli NLP, jako je odpovídání na otázky, rozpoznání entit, klasifikace textu a další.

3.3.1 BERT

BERT (Bidirectional Encoder Representation from Transformers) je model pro reprezentaci přirozeného jazyka představený v roce 2018 prací [5]. Jeho hlavním přínosem je aplikace obousměrného tréninku Transformer architektury. Ukazuje, že model trénovaný v obou směrech dokáže plně využít schopnost Transformerů zpracovávat vstupní text naráz, ne sekvenčně.

Hlavním přínosem práce [5] je způsob, kterým probíhá předtrénink na velkých textových korpusech. Trénink na úkolech, které nemusí přímo souviset s cílovým použitím dovolí modelu získat základní pochopení vlastností jazyka pro jeho správnou reprezentaci vytvořenou enkodérem. Pro předtrénink modelu BERT byly použity následující úkoly.



Obrázek 3.8: Vlevo předtrénink ekodéru BERT na úlohách *ML modeling* a *NSP*. Vpravo předtrénovaný BERT dotrénován na konkrétní úlohu odpovídání na otázky. Obrázek byl inspirován obrázkem 1 z článku [5].

Maskované modelování jazyka (ML modeling) je úloha velmi podobná již dříve zmíněné úloze predikce následujícího slova ve větě.Je ale použit přístup, kdy je určité procento slov v textu *maskováno*, aby nebyl původní úkol kvůli obousměrné reprezentaci transformerů triviální.

Při tréninku je 15 % slov každé věty nahrazeno [MASK] tokenem, jehož původní význam je poté na základě okolního kontextu hádán.

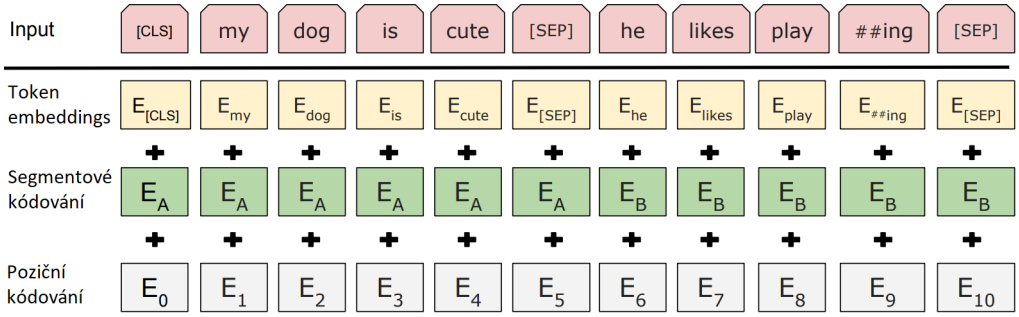
Pro tento úkol je na enkodér BERTa napojena klasifikační vrstva, pomocí které je určena pravděpodobnost pro každého slovo ve slovníku, které má [MASK] token nahradit.

Predikce následující věty (NSP) je druhým typem úlohy pro předtrénink BERTa. Jeho snahou je naučit model porozumět vztahu nejen mezi jednotlivými slovy, ale taky mezi významem celých vět.

Trénovací datová sada je rozdělena na dvojice vět A a B. Model má za úkol předpovědět, je-li věta B větou následující větu A v kontextu. V 50 % případů je věta B opravdu větou následující. Ve zbytku je to náhodně zvolená věta z korpusu.

Práce [5] také uvádí, že navzdory jednoduchosti tohoto úkolu je jeho přínos pro pozdější aplikaci modelu pro odpovídání na otázky velký.

Formát vstupu modelu BERT je koncipován tak, aby mohl být použit pro větší spektrum aplikací, pro které je zamýšlen. Proto je umožněno na vstupu pomocí jediné sekvence tokenů reprezentovat jak jednu, tak i dvě vstupní sekvence, jako třeba otázku a odpověď.



Obrázek 3.9: Příklad formátu vstupu „*input*“ modelu BERT převzatý z [5]. Poziční (*position*), segmentové a token *embeddings* a speciální tokeny [CLS] a [SEP].

Pro vektorovou reprezentaci jednotlivých tokenů jsou použity Wordpiece embeddings [26] popsány v 2.1.3. K těm je přidáno poziční kódování kvůli použití Transformer architektury a segmentační kódování, které identifikuje, které sekvenci vstupu jednotlivé tokeny náleží. Příklad je naznačen na obr. 3.9.

Pro popis vstupu jsou navíc použity dva speciální tokeny [CLS] a [SEP]. [CLS] token je přidán vždy na začátek vstupu. Token [SEP] je přidán na konec každé sekvence. [CLS] token je využit také pro klasifikační úlohy (například předtrénink na NSP).

3.3.2 ALBERT

Informace v této sekci jsou převzaty ze článku [8] *A Lite BERT for self-supervised learning of language representations.*

Hlavním problémem modelu jako BERT je jeho velikost – asi 110 milionů parametrů v jeho základní verzi. Zvětšováním modelu se většinou dá dosáhnout zlepšení, je to však problém kvůli omezené paměti grafických karet a množství času potřebného pro trénink.

ALBERT – A Lite BERT, je model představený v roce 2019 článkem [8]. Vychází z modelu BERT a dosahuje lepších výsledků na datových sadách pro porozumění textu, přestože má mnohem menší počet parametrů, než velká verze BERTa. V následujících odstavcích jsou shrnuty hlavní přínosy modelu ALBERT.

Faktorizace parametrů reprezentací – *Factorized embedding parameterization*
Wordpiece embeddings se učí reprezentovat tokeny nezávisle na kontextu, zatímco skryté vrstvy modelu se učí reprezentaci závislou na kontextu. Zvětšení velikosti skryté vrstvy způsobí u BERTa stejný nárůst velikosti reprezentace tokenů, jejichž význam není tak podstatný a model to zbytečně zatěžuje. Díky odstranění provázanosti velikosti dimenze *wordpiece embeddings* a dimenze skryté vrstvy modelu, je možno lépe zvětšovat velikost skrytých vrstev bez toho, aby to ovlivnilo velikost reprezentací ve slovníku.

Sdílení parametrů mezi vrstvami – *Cross-layer parameter sharing*
Parametry napříč vrstvami jsou sdíleny, což zamezuje růstu počtu parametrů při zvětšování hloubky modelu a způsobuje jejich efektivnější využití. Ve výsledku má velká verze modelu ALBERT mnohonásobně méně parametrů, než velká verze modelu BERT.

Trénink návaznosti dvou vět – *Inter-sentence coherence loss*
Trénink modelu BERT pomocí predikce následující věty (NSP) se částečně překrýval s úkolem predikce maskovaného tokenu, protože mohla být nesouvislost vět odhadnuta na základě nepříslušnosti slov stejnému tématu.

Pro trénink modelu ALBERT je pro tento úkol použita stejná dvojice vět A a B, u kterých se určuje, zda-li B navazuje na A. Při nenávaznosti však není věta B náhodně vybrána z jiného korpusu, ale je pouze prohozena s větou A. Model je tak lépe schopen naučit se logickou souvislost vět, místo odhadu tématu, ke kterému se věty vztahují bez toho, aby na sebe logicky navazovaly.

ALBERT (resp. BERT) pro odpovídání na otázky Pro využití modelu ALBERT (resp. BERT) na konkrétní úkoly vyžadující porozumění textu je potřeba provést *fine-tuning* modelu.

Pro odpovídání na otázky je na výstup modelu napojena lineání vrstva, která je natrénována na určení indexu začátku a konce odpovědi v rámci celé vstupní sekvence. Pro začátek a konec je poté vybrána validní[8] dvojice indexů označující rozsah odpověď.

Pro *fine-tuning* na daný úkol jsou optimalizovány parametry lineární klasifikační vrstvy i enkodéru ALBERT (resp. BERT) – vyžaduje delší trénink, nebo pouze lineární vrstvy.

[8] podmínky jako: start idx > end idx, odpověď se nenachází v první sekvenci (otázce)

Kapitola 4

Řazení dokumentů dle relevance

Tato kapitola popisuje metody pro indexaci a řazení dokumentů dle relevance, které jsou využity při vyhledávání relevantního dokumentu popsaného v části 7.4. Nejprve bude vysvětlen význam indexu v kontextu řazení dokumentů a poté algoritmy, které ho využívají k vyhledávání. Nakonec budou diskutovány problémy velké báze dat jako je Wikipedie.

Informace v této kapitole jsou převzaty z prezentací Stanfordského kurzu *Information Retrieval and Web Search*[1] a knihy *Introduction to Information Retrieval* [12].

4.1 Indexování dokumentů

Pro vyhledávání dokumentů s použitím webových zdrojů jako Wikipedie jsou typické rozsáhlé texty které by bylo náročné procházet sekvenčně. Proto je důležité vytvořit strukturu, která je dále použita metodami pro řazení dokumentů a umožní optimalizované vyhledávání. Taková struktura se nazývá **index**.

Index poskytuje informace o výskytu jednotlivých termů v prohledávaných dokumentech a je vytvořen ještě před začátkem vyhledávání. Nejpoužívanější variantou je tzv. *invertovaný index*.

Před vytvořením indexu je užitečné provést co nejlepší normalizaci prohledávaných dokumentů. Metody pro zpracování textu byly podrobně popsány v kapitole 2.2 (převod na malá písmena, lemmatizace, odstranění stop slov).

invertovaný index je struktura obsahující všechny slova (termy) prohledávaného korpusu. Pro každý term je uložen seznam dokumentů (označených *docID*), které daný výraz obsahují. Může obsahovat doplňující informace o počtu výskytů termu v dokumentu a jeho délce. Informace jsou poté využity při řazení dokumentů, tzv. *ohodnoceném vyhledávání*.

4.1.1 Úskalí velké báze dat jako Wikipedie

Česká Wikipedie obsahuje velké množství článků (asi 3,5 GB v nekomprimonavé podobě). Pro prohledávání celé Wikipedie je potřeba vytvořit a udržovat index, což je pro takový objem dat poměrně výpočetně a paměťově náročné. Je složité pomocí běžných nástrojů sestavit index pro celou Wikipedii.

Dalším problémem je různá délka dokumentů, nestrukturovanost některého textu a dynamicky se měnící prostředí, kdy jsou články na Wikipedii pravidelně aktualizovány.

Způsoby prohledávání Wikipedie a vypořádání se se zmíněnými problémy jsou popsány v části 6, konkrétně 6.2 a kapitole o implementaci 7.

[1] Dostupné z: https://web.stanford.edu/class/cs276/

4.2 Algoritmy pro řazení dokumentů

Při dotazování na vytvořený index nestačí výčet dokumentů, které termy přítomné v dotazu obsahují. Ohodnocené vyhledávání umožňuje získání pouze množiny seřazených n nejvíce relevantních dokumentů.

4.2.1 TF-IDF (Term frequency – Inversed document frequency)

Četnost termu (TF) je počet výskytů vyhledávaného termu v daném dokumentu. Vycházíme z předpokladu, že dokument s deseti výskyty termu bude pro dotaz více relevantní, než dokument obsahující vyhledávaný term pouze jednou. Neznamená to však, že bude dokument 10× více relevantní – růst relevance není přímo úměrný TF.

Jednotlivé termy nenesou stejnou informační hodnotu (např. stop slova 2.2.4) a ojedinělé výrazy jsou často zásadní pro relevanci k dotazu (což souvisí i s neúměrným růstem relevance s růstem TF).

$$TF_{t,d} = log(1 + tf_{t,d}) \tag{4.1}$$

Četnost dokumentů (DF) je počet dokumentů, které obsahují vyhledávaný term. Účel je v identifikaci termů vyskytujících se jenom ve zlomku dokumentů, které nejspíš ponesou zásadní informační hodnotu a adresovat tím neúměrnou závislost růst relevance dokumentu a četnosti termu. Obrácená četnost dokumentů (IDF) dělí celkový počet dokumentů N četností dokumentů pro daný term. Hodnota IDF je tedy vyšší pro vzácnější termy.

$$IDF_d = log(\frac{N}{df_t}) \tag{4.2}$$

TF-IDF je statistická metoda, která použitím těchto dvou metrik hodnotí relevanci dokumentu. Pro výpočet váhy TF-IDF pro daný term t v dokumentu d platí tedy:

$$\text{TF-IDF}_{t,d} = TF_{t,d} \cdot IDF_d \tag{4.3}$$

Ohodnocení S dokumentu d pro dotaz q je potom sumou jednotlivých ohodnocení termů t dané otázky q.

$$S(q,d) = \sum_{t \in q \cap d} \text{TF-IDF}_{t,d} \tag{4.4}$$

4.2.2 BM25 (Best Match 25)

(Okapi) BM25 představeno v [20] je vylepšením klasické hodnotící funkce TF-IDF. Využívá poznatky statistiky a pravděpodobnosti pro řazení dokumentů dle jejich relevance s ohledem na vztah délky dokumntu a četnost výskytů termu v něm.

Delší dokumenty budou mít pravděpodobně větší $tf_{t,d}$. To je potřeba ve výsledném ohodnocení zohlednit, aby nebyly delší dokumenty nutně zvýhodněny. Výsedný vzorec pro ohodnocení dokumentu d pomocí BM25 vzhledem k otázce q je:

$$S(q,d)^{BM25} = \sum_{t \in q} \left(IDF_d \cdot \frac{(k_1 + 1) \cdot tf_t}{k_1 \cdot ((1-b) + b \cdot \frac{dl}{avdl}) + tf_t} \right) \tag{4.5}$$

kde tf a idf jsou metriky vysvětlené v sekci 4.2.1, dl je délka dokumentu d, $avdl$ je průměrná délka dokumentu v kolekci a b, k_1 jsou konstanty.

- k_1 je konstanta ovlivňující dopad TF na výsledné skóre dokumentu. Obvykle je volena v rozmezí $k_1 \in (1,2;2)$. Pro nízké hodnoty k_1 roste význam dokumentu s rostoucí TF poměrně rychle.
- b ovlivňuje normalizaci délky dokumentu a obvykle je volena hodnota $b = 0,75$. Hodnota $b = 1$ znamená úplnou a $b = 0$ žádnou normalizaci délky dokumentu.

Výhodu hodnotící funkce BM25 oproti obyčejnému TF-IDF lze demonstrovat na následujícím příkladu.
Mějme vyhledávaný dotaz „*term frequency*“ a dva dokumenty d_1 a d_2:

	term	frequency
d_1	1024	1
d_2	16	8

	TF-IDF	**BM25**
d_1	**87**	31
d_2	75	**43**

Tabulka 4.1: Výskyt termů „term“ a „frequency“ v dokumentech d_1 a d_2 a ohodnocení jednotlivých dokumentů podle funkcí TF-IDF a BM25 ($k_1 = 2$)

Tabulka 4.1 ukazuje, jak BM25 lépe zohlednilo počet výskytů termů a skutečně ohodnotilo lépe relevantnější dokument.

Pro BM25 existují také modifikace, jako BM25+, BM25L a další, které byly porovnány v [23].

- BM25L [10] adresuje znevýhodnění příliš dlouhých dokumentů klasickou BM25 funkcí.
- BM25+ [9] adresuje stejný problém jako BM25L, ale implementuje vylepšení odlišně

Závěr článku [23] ale ukazuje, že neexistuje modifikace, která by systematicky zlepšila dosažené výsledky vyhledávání.

Kapitola 5

Dostupné datové sady a výběr

Tato kapitola se zabývá popisem a výběrem datových sad (datasetů) pro realizaci a evaluaci systému.

Datová sada je v oblasti strojového učení velká anotovaná (člověkem popsaná) kolekce dat pro trénink a vyhodnocování neuronových sítí. Obvykle obsahuje pro každý příklad vstupní data a jejich popis (anotaci).
Datové sady je možno rozdělit podle toho, v jaké fázi vývoje s nimi pracujeme.

- **Trénovací** Používá se pro naučení neuronové sité. Při zpracovávání trénovací datové sady se model učí a upravuje své parametry. Na konci tréninku by měl model konvergovat ke 100 % úspěšnosti na trénovací datové sadě.
- **Validační (dev)** Slouží pro průběžné vyhodnocení existujícího modelu v průběhu vývoje. Model se z validačních dat neučí. Může být využit například k úpravě hyperparametrů nebo zastavení tréninku modelu, když se jeho úspěšnost na validační datové sadě přestane zlepšovat. Vytvořen může být vybraným zlomkem trénovacího datasetu nepoužitým pro trénink.
- **Testovací** Obsahuje příklady z reálného světa, na kterých je model/systém vyhodnocen po dokončení jeho vývoje. Testovací dataset je často shodný s validačním.

Pro trénink neuronových sítí pro odpovídání na otázky by měla datová sada obsahovat hlavně:

- otázku
- kontext (dokument obsahující odpověď)
- odpověď
- případně doplňující informace (jako začátek odpovědi, délku odpovědi . . .)

Vytvoření takové datové sady je složitý úkol vyžadující velké množství lidských zdrojů. Je potřeba nasbírat dostatečné množství dokumentů, pro které někdo musí vymyslet otázky a vyznačit v dokumentech příslušné odpovědi. Tvorba datových sad obvykle probíhá pomocí *crowdsourcingu*[1].

[1] *Crowdsourcing je nové slovo označující proces získání potřebné služby, nápadů, příspěvků, pomoci při řešení problémů od velké skupiny lidí.* (https://wikisofia.cz/wiki/Crowdsourcing)

5.1 Datové sady pro angličtinu

Pro angličtinu existuje celá řada datových sad zaměřených na odpovídání na otázky. Nejznámější z nich je však SQuAD (*Stanford Question Answering Dataset*) [19]. Dataset obsahuje asi 100 000 anotovaných dvojic otázka–odpověď z více jak 500 různých článků z anglické Wikipedie. Existuje také rozšířená verze SQuAD datasetu. SQuAD2.0 [18] obsahuje navíc 50 000 otázek, na které v kontextu neexistuje odpověď. Model tedy musí umět určit, když dokument odpověď neobsahuje.

5.2 Datové sady pro češtinu

Čeština nabízí poměrně omezené zdroje oproti anglickým datovým sadám.

Nejznámější je datová sada SQADv3 (Simple Question Answering Database) [21]. Obsahuje přes 13 000 příkladů otázek s odpovědí v přiloženém kontextu pocházejícím z české Wikipedie. Datová sada otázky směřované na časový údaj, entitu, lokaci, osobu ... včetně 16 % ano/ne otázek.

Existuje také česká verze anglického datasetu SQuAD prezentovaná v práci [11]. Datová sada je strojovým překladem původního datasetu SQuAD 1.1 a SQuAD2.0. Oproti anglické verzi však obsahuje asi 70 000 otázek pro první (1.1) a 110 000 otázek pro druhou (2.0) verzi datové sady SQUAD (tabulka 5.1). Ztráta části rozsahu datové sady je způsobena problémy s vyznačením odpovědi ve strojově přeložené verzi dokumentu.

5.3 Výběr trénovacích a testovacích dat

Pro systém je potřeba vybrat data pro natrénování modelů pro vyznačení odpovědi v získaném textu a pro validaci výsledného systému.

Pro trénink anglického modelu byl vybrán anglický SQuAD 1.1 (obsahující i validační část datasetu). Po zvážení bylo rozhodnuto použít první verzi datasetu, protože budou testovací data obsahovat pouze otázky, pro které existuje na otevřené doméně odpověď.

Pro trénink českého modelu byl vybrán český překlad anglického datasetu SQuAD 1.1 [11] kvůli jeho velikosti a porovnatelnosti s anglickou verzí datové sady.

Jako testovací datová sada byl zvolen český dataset SQADv3, ze kterého bylo odstraněno 16 % otázek očekávajících ano/ne odpovědi. Datová sada nejlépe odpovídá reálnému scénáři užití systému, protože obsahuje otázky a odpovědi z článků z české Wikipedie (narozdíl od trénovacích datasetů).

V době použití byly zjištěny jiné hodnoty pro velikost datových sad, které jsou uvedeny v tabulce 5.2:

Dataset		Anglické otázky	České otázky
SQuAD 1.1	Trénink	87,599	64,164
	Validace	10,570	8,739
SQuAD 2.0	Trénink	130,319	107,088
	Validace	11,873	10,845

Tabulka 5.1: Velikost datasetů SQuAD 1.1 a 2.0 pro češtinu a angličtinu podle [11].

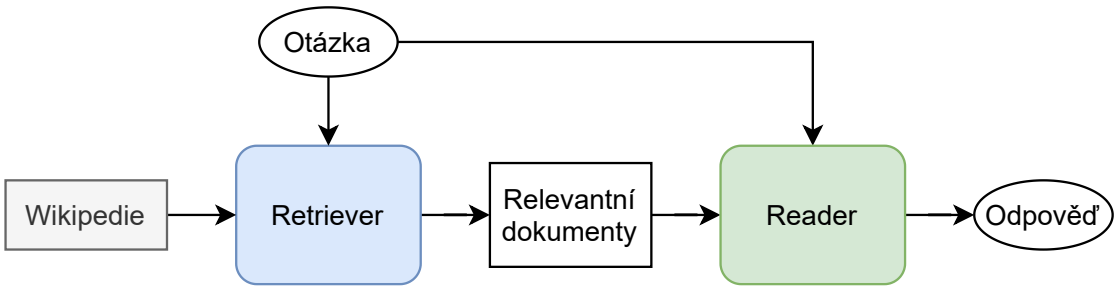
Dataset		Anglické otázky	České otázky
SQuAD 1.1	Trénink	88,638	69,169
	Validace	10,808	9,615
SQuAD 2.0	Trénink	131,958	115,415
	Validace	12,171	12,066

Tabulka 5.2: Velikost datasetů SQuAD 1.1 a 2.0 zjištěná při jejich použití.

Kapitola 6

Návrh systému a jeho komponent

V této kapitole je rozebrán návrh systému, než přijde popis implementace jeho jednotlivých částí (7). Nejprve je popsán generický návrh modelu pro odpovídání na otázky, zvolený přístup k řešení problému a motivace za finálním návrhem. Následuje rozdělení problému na části a jejich návrh včetně popisu blokových schémat hlavních komponent. Na závěr kapitoly je diskutován návrh vyhodnocení výsledného systému v kontextu otevřené domény.



Obrázek 6.1: Generické schéma (prezentované například v [3]) open-domain QA systému

6.1 Zvolený přístup k problému

Odpovídání na otázky na otevřené doméně (open-domain QA) má obvykle dvě hlavní (na sobě téměř nezávislé) části, jak je naznačeno na obrázku 6.1. První částí je *Retriever*, který má za úkol získat z internetu (pro nás z Wikipedie) relevantní dokumenty, které by mohly obsahovat odpověď a druhou je *Reader*, který provede extrakci odpovědi ze získaného kontextu [3].
Retriever části systému je věnována pozornost v sekci 7.4.

Pro návrh systému bylo však největší dilema v realizaci *Reader* části systému pro kvalitní extrakci odpovědi v češtině. Chtěl jsem využít některého z populárních předtrénovaných modelů (3.3) založených na Transformers [24] architektuře (3.2.2). Proto jsem se rozhodl vyzkoušet přístup, kdy je nativně anglickému ALBERT modelu, který dosahuje špičkových výsledků[1], strojově přeložen vstupní dokument i otázka z češtiny do angličtiny.

[1] https://rajpurkar.github.io/SQuAD-explorer/

Pro porovnání jsem zvolil vícejazyčný model BERTa (m-BERT[2]), který podporuje 104 světových jazyků s nejrozsáhlejší Wikipedií, trénovaný na českém překladu datové sady SQuAD 1.1 [11] [19].

6.2 Návrh jednotlivých částí systému

V této sekci bude popsán návrh systému, tedy hlavně *Retrieveru*, *Readeru* a části pro zpracování datové sady SQAD v3.0 [21] pro závěrečné vyhodnocení.

6.2.1 Návrh Retrieveru

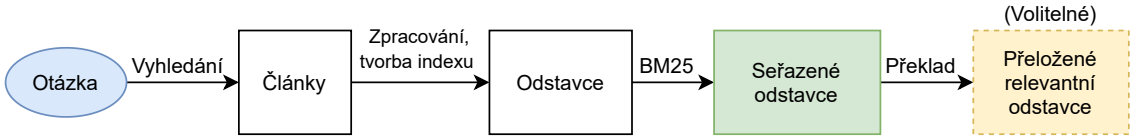
Získání n relevantních dokumentů kandidujících pro zodpovězení otázky vyžaduje několik na sebe navazujících kroků (obrázek 6.2). V návrhu a implementaci jsou použity metody předzpracování textu a řazení dokumentů popsané v kapitolách 2 a 4.

Retriever dostává na vstupu otázku, která je také hlavním vstupem do celého systému. Dříve, než se na základě otázky začnou přímo prohledávat odstavce české Wikipedii, bude dobré z otázky získat co nejvíce informací, které by mohly vést ke *spraveným článkům*. Až po jejich získání jsou jednotlivé články rozděleny na odstavce a ty prohledávány.
K získání článku jsou použity tři metody:

- **základní Wiki Search** nezpracované otázky, který někdy přináší uspokojivé výsledky, ale většinou je nedostačující,
- **získání relevantního titulku článku** (pomocí BM25 4.2.2) z *dumpu* titulků[3] a abstraktů české Wikipedie po lemmatizaci otázky a odstranění stop slov,
- **rozpoznání (pojmenované) entity**, která by se mohla vyskytovat v titulku článku a usnadnit tím *Wiki Search*.

Tímto tedy získáme seznam článků z Wikipedie, který je následně upraven tak, aby se v něm každý získaný článek vyskytoval pouze jednou.

Jednotlivé články je následně třeba rozdělit na odstavce, normalizovat jejich délku, lemmatizovat je, odstranit stop slova a indexovat je. Odstavce jsou poté seřazeny algoritmem BM25 a z nich vybrány první tři[4] nejvhodnější. Pro vyhledávání mezi odstavci je použita lemmatizovaná otázka s odstraněnými stop slovy. Tři nejvhodnější odstavce jsou získány v jejich původní, nezpracované podobě a případně (při použití nativně anglického *readeru*) jsou včetně otázky přeloženy do angličtiny.



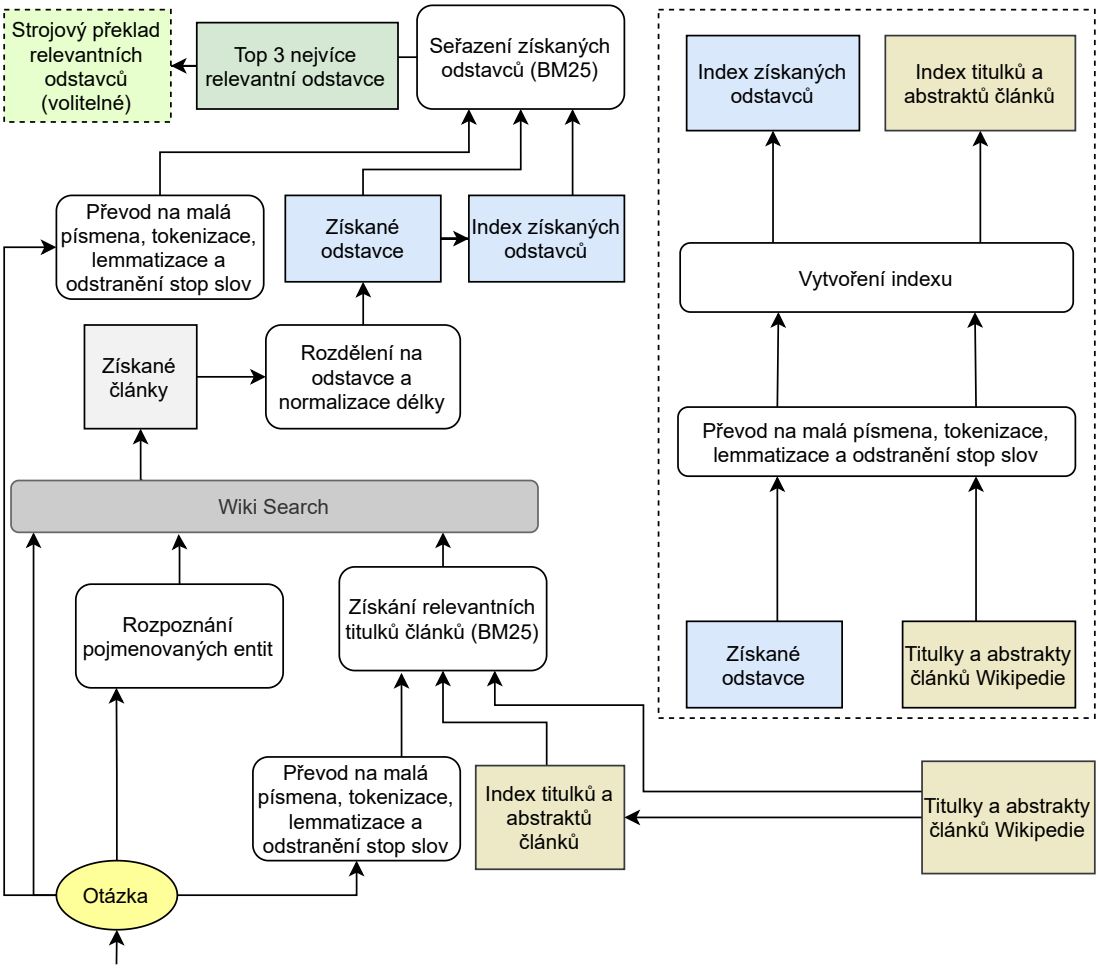
Obrázek 6.2: Zjednodušené schéma *retriever* části systému

[2] https://github.com/google-research/bert/blob/master/multilingual.md – Zjednodušeně řečeno je to model BERT předtrénován na korpusech mnoha jazyků.

[3] https://cs.wikipedia.org/wiki/Wikipedie:St%C3%A1hnut%C3%AD_datab%C3%A1ze

[4] V našem případě experimentálně určeno. Dále byly zkoušeny hodnoty 5, 8 a 10, nebylo však zaznamenáno prokazatelné zvýšení úspěšnosti.

Následuje podrobný návrh struktury *retrieveru* znázorňující podrobně jednotlivé komponenty na obrázku 6.3.



Obrázek 6.3: Podrobné schéma *retriever* části systému

6.2.2 Návrh Readeru

Část systému pro extrakci odpovědi ze získaného dokumentu byla navržena ve dvou variantách.

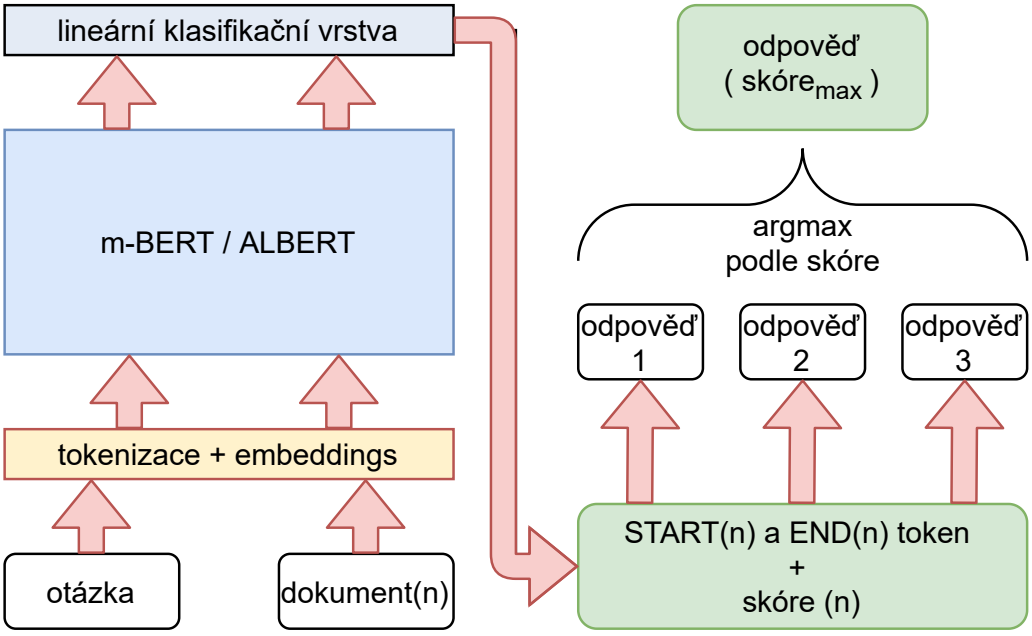
- **Anglický ALBERT** s lineární klasifikační vrstvou pro určení začátku a konce odpovědi, pro který jsou vstupní dokumenty a otázka strojově přeloženy do angličtiny. Výsledná odpověď je přeložena opět zpět do češtiny.
- **Vícejazyčný BERT (m-BERT)** opět s lineární klasifikační vrstvou pro určení začátku a konce odpovědi, který dokáže odpovídat přímo na českých dokumentech.

Pro obě varianty je však postup (až na překlad pro anglický ALBERT, který je však součástí *retrieveru*) shodný.

Model dostane k přečtení tři nejvíce relevantní odstavce získané *retrieverem*. Pro každý odstavec je provedena extrakce odpovědi, kdy je vybrána vždy právě jedna validní odpověď

s největším „skóre“(nenormalizovaná logaritmická pravděpodobnost) pro daný dokument. Ze získaných odpovědí je vybrána ta, kterou si je model nejvíce jistý (ta s největším skóre) a případně (při použití ALBERT-en modelu) přeložena zpátky do češtiny.
Ostatní odpovědi jsou také uloženy pro získání statistických dat při vyhodnocení. *Reader* tak vybere kromě odpovědi také finální dokument, ze kterého je odpověď získána, který může být použit pro získání některých dalších informací/kontextu uživatelem.

Schéma *readeru* pro extrakci finální odpovědi je naznačeno na obrázku 6.4. Modely použité k implementaci a naznačeny v návrhu byly blíže popsány v kapitole 3, konkrétně v sekci 3.3.



Obrázek 6.4: Schéma *readeru* (kombinovaně pro oba navržené systémy) pro 3 relevantní dokumenty, kde $n \in (1, 3)$. Pro model ALBERT by musel proběhnout ještě strojový překlad finální odpovědi do češtiny.

6.2.3 Návrh vyhodnocení

Pro vyhodnocení byl vybrán český dataset SQAD v3.0, jak už bylo několikrát zmíněno. Pro vyhodnocení musely být navrženy metriky respektující charakteristiku datové sady i to, že se pohybujeme v rámci otevřené domény. Kromě tradičních metrik EM (*exact match*) a F1 (popsány blíže v kapitole 8.1) jsem se při návrhu vyhodnocení systému snažil respektovat několik věcí:

- Kvůli otevřené doméně může být odpověď nalezena z jiného kontextu v jiném tvaru.
- Potřebujeme metriku pro ohodnocení samotného *retrieveru* v úspěšnosti nalezeného dokumentu.
- Metriku pro zohlednění i správných odpovědí, které byly nalezeny, i když nebyly například vybrány *readerem* jako nejvhodnější.

Pro podrobnější popis zvolených metrik viz 8.1. Metriky budou použity pro porovnání systému využívající dva navržené přístupy v *reader* části a také aspektů ovlivňující *retriever*.

Kapitola 7

Implementace a použité technologie

Obsahem této kapitoly je popis použitých nástrojů a implementace jednotlivých částí systému. Je popsáno jak a pomocí jakých knihoven byl návrh z kapitoly 6 realizován a jaké problémy bylo potřeba vyřešit. Rozebrán je taky postup zpracování dat potřebných pro funkci systému i jeho vyhodnocení.

7.1 Použité nástroje a technologie pro implementaci

V této sekci budou popsány použité technologie a knihovny použité při implementaci výsledného systému.

Pro implementaci byl vybrán jazyk **Python** (3.7.10), který nabízí největší množství knihoven a nástrojů vhodných pro realizaci systému. Zároveň je vhodný ke zpracování dat a je sympatický svou jednoduchou syntaxí. Nabízí také prostředí *Jupyter notebook*, který je velmi pohodlný pro vývoj.

Některé knihovny implementují poznatky popsané v kapitolách 2, 3 a 4. Knihovny běžného charakteru, jako je například `json`, ve výčtu nejsou uvedeny.

- **PyTorch**[1] je open-source framework pro strojové učení, zejména pro zpracování obrazu a přirozeného jazyka.

  Nabízí implementace mnohých druhů neuronových sítí a tensorových operací s GPU akcelerací (CUDA).

- **HuggingFace – Transformers**[2] je populární (open-source) implementací nejznámějších Transformers architektur jako BERT nebo ALBERT (dále RoBERTa, XLM ...) vyvinuta společností HuggingFace.

  Nabízí nástroje pro jednoduchou práci s modely, předzpracování jejich vstupu, trénink atd. Pro implementace modelů obecného využití jako BERT nabízí také jejich verze pro konkrétní úlohy, jako je odpovídání na otázky (extrakce odpovědi). Pro každou architekturu je k dispozici typicky několik předtrénovaných verzí (BERT base, BERT large ...).

[1] https://pytorch.org/
[2] https://huggingface.co/transformers/

Je implementována jak pro PyTorch, tak i pro velmi podobný framework TensorFlow[3].

- **HuggingFace – Datasets**[4] umožňuje snadnou práci s veřejně dostupnými datasety. Usnadňuje jejich načtení a efektivní předzpracování.

  Díky dostupnosti zdrojových kódů Umožňuje také úpravu skriptu pro načtení datové sady pro načtení jiné, která doposud v oficiálním seznamu knihovny není dostupná.

- **DeepPavlov**[5] je open-source framework, postavený na Pytorch, TensorFlow a Keras, pro usnadnění vývoje a výzkumu v oblasti zpracování přirozeného jazyka, konkrétně dialogových systémů.

- **Rank-bm25**[6] je knihovnou implementující hodnotící algoritmy z rodiny BM25 [23] (Okapi, BM25+, BM25L).

  Nabízí velmi jednoduché API pro indexaci a řazení dokumentů.

- **NumPy**[7] nabízí jako knihovna velmi efektivní operace pro práci s vektory a maticemi a vícerozměrnými poli. Je nabízena pod licencí BSD (open-source).

- **NLTK**[8] (Natural Language Toolkit) je open-source knihovna nabízející nástroje pro zpracování textu a přirozeného jazyka.

  Prostřednictvím jednoduchého API umožňuje tokenizaci (včetně dělení na věty), stematizaci, tagování částí textu a jiné.

- **MorphoDiTa**[9] **(corpy)**[10] je open-source nástrojem pro morfologickou analýzu umožnující například tokenizaci nebo lemmatizaci textu.

  Pro použití je potřeba také natrénovaný jazykový model, který je nabízený na webových stránkách v poznámce pod čarou pro češtinu a angličtinu. Jednoduché API pro práci s MorphoDiTou nabízí knihovna **corpy**.

- **Majka**[11] je podobně jako MorphoDiTa morfologickým analyzátorem.

  Majka provádí analýzu za použití morfologických slovníků. Slovníky jsou dostupné pro celou řadu jazyků, jako je čeština, slovenština, polština, angličtina, ...

- **Wikipedia (api)**[12] je knihovna usnadňující vyhledávání, získávání a parsování článků z Wikipedie.

  Nabízí velmi jednoduché API pro vyhledávání článků a získávání jejich textového obsahu.

- **Googletrans**[13] nabízí API pro přístup k funkcionalitě Překladače Google.

  Je neoficiální implementací přístupu k překladači a kvůli omezením na množství překládaných znaků nezajišťuje naprosto stabilní funkčnost.

[3] https://tensorflow.org/
[4] https://huggingface.co/docs/datasets/
[5] https://deeppavlov.ai/
[6] https://github.com/dorianbrown/rank_bm25
[7] https://numpy.org/
[8] https://www.nltk.org/
[9] https://ufal.mff.cuni.cz/morphodita
[10] https://corpy.readthedocs.io/en/stable/api/morphodita.html
[11] https://nlp.fi.muni.cz/czech-morphology-analyser/
[12] https://github.com/goldsmith/Wikipedia
[13] https://github.com/ssut/py-googletrans

- **Scikit-learn – metrcis**[14] je část knihovny **sklearn** určená k usnadnění výpočtu základních vyhodnocovacích metrik.
- **Jupyter Notebook**[15] je open-source webová aplikace pro interaktivní spouštění Python kódu v tzv. buňkách.

  Kromě zrojového kódu umožňuje také vkládání textu, vizualizaci dat, vkládání obrázků a grafů nebo spouštění BASH (Shell) skriptů.
- **Google Colab**[16] je služba nabízející přístup k virtuálnímu stroji, který zprostředkovává přístup k Jupyter Notebooku v cloudu.

  Virtuální stroje nabízí (bez rozšířené placené verze) asi 12 GB RAM, úložiště s možností připojení a přístup k Google Drive a výkonné GPU s pamětí 16 GB vhodné pro akceleraci výpočtů. Dokonce jsou dostupné i běhová prostředí s TPU pro paralelizaci výpočtů.

  Běhová prostředí vždy disponují množstvím předinstalovaných, běžně používaných knihoven (PyTorch, TensorFlow a další).

7.2 Zpracování dat v práci

Následující sekce popisuje, jak byla zpracována data, se kterými se v práci pracuje. Nejprve je popsáno zpracování datasetu SQuAD a jeho českého překladu, na kterých je trénován *reader*. Poté je popsáno zpracování dumpů obsahující titulky a abstrakty článků české Wikipedie. Vysvětleno je také zpracování českého datasetu SQAD v3.0 a také formát uložení odpovědí při vyhodnocení testování systému, aby na nich mohla být provedena evaluace.

7.2.1 Předzpracování datové sady SQuAD

Pro načtení datasetu SQuAD byla použita knihovna Huggingface `datasets` , která načte trénovací i dev dataset dostupný ve formátu JSON. Pro načtení je použit skript *squad.py*, který lze díky open-source povaze knihovny upravit a použít i pro načtení české verze datasetu SQuAD[17] (skripty *czech_squad.py* a *czech_squad2.py*). Stejně jednoduchý postup lze použít i pro načtení verze 2.0 obou datasetů.

Po načtení obsahuje dataset položku *train* a *validation* pro trénink a vyhodnocení natrénovaného modelu pro extrakci odpovědi. Velikosti jednotlivých částí jsou zmíněny v tabulce 5.2. Každá otázka ve SQuAD datasetu obsauje id, titulek, kontext(dokument), otázku a odpověď. Odpovědí může být i více, pro odpověď je také uveden index znaku, na kterém začíná.

Z knihovny `transformers` je poté potřeba načíst tokenizér, který převede otázku a kontext každé položky v datasetu do podoby stravitelné modelem (tokeny reprezentovány token ID pro předtrénovaný slovník modelu a speciální tokeny jako [CLS] a [SEP]). Pro vícejazyčný BERT lze použít `BertTokenizerFast`, pro ALBERT zas `AlbertTokenizerFast`.

[14] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
[15] https://jupyter.org/
[16] https://colab.research.google.com/notebooks/intro.ipynb
[17] https://lindat.cz/repository/xmlui/handle/11234/1-3249?show=full

Modely BERT a ALBERT založené na Transformer architektuře mají pevně danou maximální délku vstupní sekvence tokenů. Proto je potřeba tokenizér nastavit tak, aby byly příliš dlouhé vstupy rozděleny na sekvenci kratších, ideálně s nějakým přesahem za sebou následujících oddělených částí (`return_overflowing_tokens=True`).

Při práci s odpovědí je v datasetu vyznačen pouze index počátečního znaku. Nastavením tokenizéru `return_offsets_mapping=True` získáme pro každý token také indexy vstupního řetězce znaků, na kterých se nachází.

S použitím popsaných metod jsou k původnímu datasetu přidány informace o indexu *počátečního a koncového tokenu* každé odpovědi pro daný tokenizér daného modelu.

Funkce, která zpracování implementuje, byla vypůjčena z HuggingFace notebooku[18] a pomocí metody datasetu `map()` je aplikována na všechny vzorky datasetu.

7.2.2 Práce s dumpem Wikipedie

V práci byly použity dva různé soubory pro získání relevantních článků Wikipedie. Prvním je dump názvů článků a druhým je dump abstraktů.

Soubor s názvy článků je pouze jednoduchý text snadný pro zpracování. Na každý řádek souboru připadá jeden název článku. Před vytvořením indexu je tedy třeba pouze nahradit podtržítka v názvech mezerou a uložit názvy jednotlivých článků do seznamu.

Pro abstrakty článků je to o něco složitější, protože jsou uloženy v poměrně rozměrném XML souboru. Každý záznam obsahuje titulek, url, text abstraktu a odkazy. Po načtení je dump zpracován tak, že je ponechána pouze informace o názvu článku a text jeho abstraktu. Výsledný zpracovaný dump je uložen jako soubor JSON s asi čtvrtinovou velikostí souboru původního. Získání a zpracování konkrétních článků je popsáno až v sekci 7.4.

7.2.3 Extrakce informací z datasetu SQAD

Dataset SQAD v3.0[19] je dostupný ve velmi specifické struktuře. Jednotlivé otázky datové sady jsou rozesety po 13,476 složkách, z nichž každá obsahuje 7 – 9 souborů „vert“. Důležité pro nás jsou pouze soubory obsahující otázku a extrakci odpovědi. Na obrázku 7.1 je příklad formátu, ve kterém jsou otázky a odpovědi uloženy.

```
<s>
Kdy     kdy     k6eAd1  kdy
se      se      k3xPyFc4        se
narodil narodit k5eAaPmAgMnS    narodit
herec   herec   k1gMnSc1        herec
Ivan    Ivan    k1gMnSc1        Ivan
Trojan  Trojan  k1gMnSc1        Trojan
<g/>
?       ?       kIx.    ?
</s>
```

Obrázek 7.1: Příklad formátu položky (otázky) datasetu SQAD v3.0.

[18] https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/question_answering.ipynb#scrollTo=xcE7HKQsxr5S

[19] https://lindat.cz/repository/xmlui/handle/11234/1-3069

Pro každý záznam musí být tedy otevřeny a zpracovány soubory obsahující otázku a odpověď. První sloupec souboru obsahuje na každém řádku *token* otázky/odpovědi, druhý sloupec jejich lemmata. Při zpracování je také potřeba brát ohled na speciální znaky.

Pro každý záznam je při zpracování uložena otázka, extrakce odpovědi a lamma extrakce odpovědi. Záznamy obsahující „ano/ne“ odpovědi byly ze zpracování vynechány. Zpracovaný dataset je uložen ve formátu JSON (obr. 7.2) pro jeho pozdější snadnější a rychlejší zpracování.


```
{
    1: {
        question: "Kdo je autorem novely Létající jaguár? ",
        answer: "Josefa Formánka ",
        answer_lemma: "Josef Formánek "
    },
    2: {
        question: "Jak se nazývá věda zabývající se houbami? ",
        answer: "mykologie ",
        answer_lemma: "mykologie "
    },
```


Obrázek 7.2: Příklad formátu JSON zpracovaného datasetu SQAD v3.0.

7.2.4 Ukládání dat pro vyhodnocení systému

Pro uložení výsledků byl opět zvolen formát JSON, aby mohla být evaluace provedena až po získání odpovědí.


```
2: {
    question: "Jak se nazývá věda zabývající se houbami? ",
    answer: "mykologie",
    answer_orig: "mykologie",
    answer_sqad: "mykologie ",
    answer_sqad_lemma: "mykologie ",
    articles: [
        "Mykologie",
        "Houby"
    ],
    document: "Houby (Fungi, Mycetalia) představují velkou skupinu
    all_answers: [
        "mykologie",
        "Mykologie",
        "saprofytické houby"
    ],
    all_log_probs: [
        "18.43",
        "13.74",
        "-1.10"
    ]
},
```


Obrázek 7.3: Příklad formátu JSON uložené odpovědi s doplňkovými informacemi.

Pro každý zodpovězený záznam je uloženo číslo otázky „2“, otázka *question*, odpověď *answer*, originální nepřeložená odpověď *answer_orig* (při použití m-BERT modelu, jako v příkladu, shodná s *answer*), extrakce odpovědi podle datasetu SQAD *answer_sqad* a její

lemma *answer_sqad_lemma*. Dále jsou pro podrobnější analýzu výsledků k dispozici názvy článků *articles*, ve kterých systém odpověď vyhledával, kontext *document*, ze kterého byla finální odpověď vybrána, první tři nejlepší odpovědi *all_answers* a (nenormalizované) pravděpodobnosti *all_log_probs* jednotlivých odpovědí z *all_answers*.

Soubor zodpovězených záznamů je pak pojmenován podle toho, jaký rozsah otázek obsahuje a přidáno je i časové razítko.

answers_1-3000___04-04-2021_09:53.json

7.3 Implementace a trénink readeru

Pro implementaci *readeru* systému byla použita knihovna `transformers` od HuggingFace, která nabízí velmi jednoduché rozhraní pro trénink modelů.

7.3.1 Trénink

Pro trénink ALBERT modelu byla použita třída `AlbertForQuestionAnswering` a předtrénovaný model `albert-base-v2`[20] (11 milionů parametrů). Model není příliš velký a dosahuje podobných výsledků, jako jeho větší verze `large` nebo `xlarge`. Pro *fine-tuning* byl použit dataset SQuAD 1.1 a 2.0.

Pro druhý model byl použit `BertForQuestionAnswering` z knihovny `transformers`. Předtrénovaný model `bert-base-multilingual-cased`[21] (110 milionů parametrů). Pro fine-tuning byl použit český překlad datasetu SQuAD 1.1 a 2.0. Pro načtení obou datasetů byl použit upravený skript *squad.py*.

Všechny modely byly trénovány pomocí třídy `Trainer` s následujícími parametry:

- *batch size* (velikost dávky) – 16
- *learning rate* (rychlost učení) – $2e-5$
- *weight decay* – $0,01$
- *počet trénovacích epoch* – 3

Po *fine-tuningu* je úspěšnost modelů (F1 a EM, viz 8.1) na jejich validačních datasetech následující (tabulka 7.1). Oba modely byly trénovány na GPU ve službě Google Colab.

ALBERT	SQuAD 1.1	SQuAD 2.0
EM	82,14	78,33
F1	89,58	81,55

BERT	cz_SQuAD 1.1	cz_SQuAD 2.0
EM	68,52	66,47
F1	77,72	69,54

Tabulka 7.1: Úspěšnost modelů na validačních datasetech.

7.3.2 Extrakce odpovědi

Třída zajišťující extrakci odpovědi se nazývá `Reader`. Pro extrakci odpovědi z kontextu je nejprve potřeba vytvořit její instanci. Většina parametrů pro inicializaci jsou volitelné, musíme však zadat cestu k natrénovanému modelu (`model_checkpoint`) v úložišti a jeho

[20] https://huggingface.co/albert-base-v2
[21] https://huggingface.co/bert-base-multilingual-cased

typ (`model_type`). Můžeme také upravit maximální délku odpovědi (`max_answer_length`) a maximální počet validních odpovědí (`n_best_size`), které model vybere. *Reader* načte tokenizér a model, který bude pro extrakci odpovědi využívat a použije GPU, pokud je k dispozici.

Pro extrakci odpovědi je poté možné volat jedinou instanční metodu `get_answers`, která vyžaduje otázku s kontextem a vrací pole o velikosti `n_best_size` obsahující nejlepší kandidáty pro odpověď. Každá odpověď se skládá z textu "text"a skóre "score".

Pro každý dokument je poté vybrána nejlépe hodnocená odpověď s neprázdným textem.

7.4 Implementace retrieveru

Pro získání relevantních dokumentů je implementována třída `Retriever`. Pro její inicializaci je třeba zadat cestu k modelu (`dita_file`) pro nástroj MorphoDiTa, který provádí lemmatizaci, cestu k souboru s titulky článků Wikipedie (`wiki_titles`) a ke zpracovanému JSON souboru s abstrakty Wikipedie (`wiki_abstracts`). Místo MorphoDiTy může `Retriever` použít i nástroj Majka (`majka=True`), pro který je potřeba zadat cestu k souboru s morfologickým slovníkem (`majka_file`). Pokud není k dispozici zpracovaný soubor s abstrakty, lze původní dump abstraktů převést na formátovaný JSON pomocí funkce `parse_abstracts()`.

Při vytváření instance `Readeru` je načten lemmatizátor, sestaven index titulků článků a index abstraktů článků Wikipedie, načten tokenizér (pro rozdělení textu na věty) a načten model pro rozpoznávání entit.

Použitá stop slova – Pokud je potřeba při zpracování textů v průběhu vyhledávání relevantního dokumentu odstranit stop slova, je pro to využit následující seznam:

> kdy být a se v na ten on že s z který mít do já o k i jeho ale svůj jako za moci pro tak po tento co když všechen už jak aby od nebo říci jeden jen můj jenž ty stát u muset chtít také až než ještě při jít pak před však ani vědět hodně podle další celý jiný mezi dát tady tam kde každý takový protože nic něco ne sám bez či dostat nějaký proto

Dále je k dispozici seznam pro odstranění interpunkce:

> . , ? ! ... "( ) ; - /

Tvorba indexu – Pro vytvoření indexu pro prohledávání je použita třída `BM25Okapi` z dříve zmíněné knihovny `rank-bm25`. Každý titulek/abstrakt je převeden na malá písmena, lemmatizován a poté jsou z něj odstraněny stop slova. Získány jsou zpracované tokeny každého titulku/abstraktu. Index titulků/abstraktů je poté vytvořen pomocí `BM25Okapi` z takto zpracovaných titulků/abstraktů.

Rozpoznání pojmenovaných entit – Rozpoznání pojmenovaných entit (NER - *Named Entity Recognition*) je další úloha na poli NLP. Uveďme alespoň příklad jejího použití:

$$\text{Kdy vynalezl Alfred Nobel dynamit?} \longrightarrow \text{Alfred Nobel}$$

NER může usnadnit vyznačení klíčových slov pro vyhledávání.

Pro rozpoznání pojmenované entity v otázce je použita knihovna `deeppavlov`. Využívá natrénovaný model `ner_ontonotes_bert_mult`, který je vytvořen funkcí `build_model()`.

7.4.1 Získání relevantních dokumentů

Inicializace třídy `Retriever` trvá hlavně kvůli tvorbě indexů a načtení modelu pro NER poměrně dlouho[22]. Po jeho inicializaci je potřeba pro získání relevantních dokumentů pouze jediná instanční metoda `retrieve()`, která pro zadanou otázku získá n (v práci 3) relevantních dokumentů.

Získání dokumentů probíhá následovně:

1. Nejprve jsou získány názvy článků metodou `get_doc_list()`, které jsou výsledkem vyhledávání metody `search()` knihovny `wikipedia`. Jednotlivé vstupy do vyhledávání jsou následující:

   - V otázce jsou rozpoznány pojmenované entity
   - Otázka je převedena na malá písmena, lemmatizována a jsou z ní odstraněny stop slova.
     - Pomocí metody vytvořeného indexu titulků `get_top_n()` je získán nejvíce relevantní titulek článku vzhledem k otázce.
     - Pomocí metody vytvořeného indexu abstraktů `get_top_n()` je získán nejvíce relevantní titulek článku vzhledem k otázce.
   - Otázka je pro vyhledání použita nezpracovaná.

   Pokud jsou některé výsledky vyhledávání stejné, je použit pouze jeden z nich.

2. Pro každý získaný název článku je získán jeho textový obsah metodou `page()` knihovny `wikipedia`. Ten je rozdělen na jednotlivé odstavce a jsou z něj odstraněny nepotřebné části jako reference a odkazy.

3. Je zkontrolováno, jestli některý z odstavců nepřesahuje maximální délku. Je-li potřeba odstavec dále rozdělit, je použita metoda `normalize_length()`, která využívá tokenizéru knihovny `nltk`. Ten rozdělí odstavec na věty a spojuje jednotlivé věty, dokud není dosaženo maximální délky. Pro zbývající věty je vytvořen další odstavec, který obsahuje také poslední 2 věty odstavce předcházejícího.

4. Po normalizaci délky získaných odstavců je potřeba získat jejich zpracovanou verzi, aby mohl být sestaven index pro vyhledávání. Každý odstavec je tedy převeden na malá písmena, lemmatizován a jsou z něj odstraněna stop slova (a interpunkce). Pro každý odstavec je získán seznam takto normalizovaných tokenů, které jsou použity pro tvorbu indexu `BM25Okapi` pro vyhledávání.

5. Původní lemmatizovaná otázka převedená na malá písmena bez stop slov je použita pro vyhledávání v indexu získaných odstavců pomocí metody `get_top_n()`. Tím jsou získány (nezpracované[23]) nejvíce relevantní odstavce – dokumenty pro danou otázku.

[22] Asi 10 minut, pokud není potřeba stáhnout `deeppavlov` model pro NER.
[23] Zpracování probíhá pouze pro tvorbu indexu.

Nalezení odpovědi na otázku

Celý tok programu systémem je následující:

- Je vytvořena instance třídy pro extrakci odpovědi `Reader`, instance třídy pro získání relevantních dokumentů `Retriever` a také instance překladače `Translator` knihovny `googletrans`.
- Po zadání otázky je volána funkce `find_answer()`, která získá odpověď na otázku.
  - Pomocí *retrieveru* jsou získány relevantní dokumenty
  - V případě použití nativně anglického modelu ALBERT jsou dokumenty spolu s otázkou přeloženy pomocí funkce `transtale()` do angličtiny.
  - Pro každý získaný dokument je provedena extrakce odpovědi a vybrána nejlépe ohodnocená neprázdná odpověď.
  - Ze získaných extrakcí odpovědi z jednotlivých dokumentů je vybrána ta, která má nejvyšší skóre. Použita je metoda `argmax()` knihovny `numpy`. Pomocí získaného indexu je vybrán dokument i odpověď.
  - Vybraná odpověď je při použití anglického modelu pro extrakci přeložena zpět do češtiny.
  - Funkce vrací odpověď a pro podrobné vyhodnocení dále: (odpověď před překladem), dokument, ve kterém byla odpověď nalezena, seznam nejlepších odpovědí, seznam skóre nejlepších odpovědí, seznam sum skóre nejlepších odpovědí a seznam článků Wikipedie, které byly pro nalezení odpovědi procházeny.

Kapitola 8

Vyhodnocení systému a rozbor chyb

V této kapitole jsou popsány výsledky dosažené vytvořeným systémem. Popsán je vliv použitého přístupu pro *reader*, trénovacího datasetu a použitého nástroje pro morfologickou analýzu. Nejprve jsou popsány standardní i speciální metriky, na nichž je úspěšnost systému prezentována. Výsledky jsou poté porovnány s jiným existujícím řešením. Jsou diskutovány hlavní nedostatky a příčiny chyb a možnosti dalšího zlepšení.

8.1 Vysvětlení základních metrik

V této sekci jsou popsány metriky použité pro vyhodnocení systému. Popsány jsou standardní metriky jako EM a F1 používané pro QA systémy a také metriky zachycující specifika otevřené domény.

Metody EM, F1, MRR a částečně T3M jsou standardní metriky používané pro vyhodnocení většiny podobných systémů. Ostatní metriky jsou úpravou těch standardních, navržené specificky pro vyhodnocení tohoto systému.

8.1.1 Exact Match (EM)

Exact match, neboli přesná shoda, je jednoduchá standardní metrika, které ale může diskriminovat dostačující odpovědi s pouze malými rozdíly oproti referenčním výsledkům. Její hodnota může být buď 1 (pokud je shoda dosažena) nebo 0 (pokud se oproti referenční hodnotě výsledek liší, byť jen v jednom znaku). Na příkladu je ukázka penalizace odpovědi, která byla dokonce přesnější, než referenční odpověď.

otázka: Kde se narodil Gabriel Jonas Lippmann?
referenční odpověď: Bonnevoie
odpověď systému 1: Bonnevoie, Lucembursko – **EM** – 0
odpověď systému 2: Bonnevoie – **EM**– 1

Pro vyhodnocování je referenční i získaná odpověď převedena před porovnáním na malá písmena. Při vyhodnocování systému s readerem ALBERT je brána v úvahu i původní, nepřeložená odpověď.

8.1.2 Exact Lemma Match (ELM)

ELM je speciálně navržená metrika pro toleranci drobných rozdílů oproti referenční odpovědi. Kromě převedení na malá písmena jsou odpovědi před porovnáním také lemmatizovány. Skóre ELM je tedy typicky vyšší, než EM.

> **otázka:** Kdy se koná dostihový závod Velká pardubická?
> *referenční odpověď:* druhou říjnovou neděli
> *odpověď systému:* druhá říjnová neděle
> **EM** – 0 , **ELM** – 1

Skóre ELM nabývá opět hodnoty buď 1 (správně) nebo 0 (špatně), stejně jako pro EM. ELM pro příklad uvedený v 8.1.1 by bylo také 0.

8.1.3 Naivní skóre (NS)

Naivní skóre je nejvíce benevolentní metrikou. Zjišťuji u ní, zda-li je odpověď nebo její lemma podřetězcem referenční odpovědi nebo jejího lemma, případně naopak. Snaha je tedy zachytit co nejvíce alespoň částečně správných odpovědí. Naivní skóre (NS) příkladu uvedeného v podsekci 8.1.1 nebo 8.1.2 by tedy bylo 1.

8.1.4 Top 3 Match (T3M)

T3M je obdobou EM 8.1.1, pro shodu je však testována každá z nejlepších třech (3) nalezených odpovědí. Můžeme tedy určit, v kolika případech oproti EM systém našel správnou odpověď, ale nevybral ji, protože nedosáhla nejvyššího skóre. T3M nabývá opět hodnoty 0 nebo 1.

8.1.5 MRR

MRR (*Mean Reciprocal Rank*) je obdobou T3M, zohledňuje však, na kolikátém místě byla odpověď nalezena (T3M zohledňuje pouze to, jestli byla vůbec nalezena). Pokud dosáhla odpověď shodná s referenční odpovědí nejvyšší skóre (byla vybrána), hodnota MRR je 1, pokud byla na druhém místě, hodnota je 1/2 a pokud na třetím, hodnota MRR je 1/3. Pokud nebyla odpověď nalezena vůbec, hodnota MRR je 0.

8.1.6 F1 skóre

Je jakýmsi kompromisem mezi naivním skóre a EM. Při jeho výpočtu jsou použity dvě metriky — *recall* (úplnost) a *precision* (přesnost).

$$\mathbf{F1} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{8.1}$$

$$recall = \frac{tp}{tp + fn} \quad precision = \frac{tp}{tp + fp} \tag{8.2}$$

- tp – *true positive* – znaky (počet) společné pro zíkanou i referenční odpoěď
- fn – *false negative* – znaky, které jsou v referenční odpovědi, ale chybí v získané
- fp – *false positive* – znaky, které nejsou v referenční odpovědi, ale přebývají v získané

Obvykle jsou definice tp, fn a fp uváděny pro tokeny, ne znaky, chceme ale více zohlednit morfologii češtiny a dropné rozdíli v odpovědích vztahující se k otevřené doméně – například augustiánský/augustiánských – při porovnávání tokenů bychom nedosáhli žádné shody.

Pro příklad:

> **otázka:** Odkud byla anglická kapela The Beatles?
> *referenční odpověď:* z Liverpoolu
> *odpověď systému:* Liverpoolu

je tedy $precision = 1$, $recall = 0,83$ a $\mathbf{F1} = 0,91$.

Pro příklad z 8.1.1

> **otázka:** Kde se narodil Gabriel Jonas Lippmann?
> *referenční odpověď:* Bonnevoie
> *odpověď systému:* Bonnevoie, Lucembursko

je $precision = 0,41$, $recall = 1$ a $\mathbf{F1} = 0,82$.

F1 tedy zohledňuje přesnost i úplnost odpovědi a je velmi směrodatnou metrikou pro odpovídání na otázky.

8.1.7 Lidské vyhodnocení

Otevřená doména je problematická pro automatické vyhodnocování. Jak je uvedeno například v [16], otázka může mít více možných odpovědí, pokud není jednoznačně položena. Odpověď může být také obecnější, nebo specifičtější (př. z 8.1.1), ale stále odpovídat na otázku. Z článku [16] tedy vyplývá, že systémy jsou hodnoceny lépe při vyhodnocení lidmi. Konkrétně, procento odpovědí označených jako „rozhodně správně“ bylo u systémů v průměru o 17 – 25 % lepší, než pouhé porovnání řetězců odpovědí pomocí EM.

Proto je pro každý experiment také náhodně vybráno 300 otázek pro lidské vyhodnocení, aby se vzal alespoň minimální ohled na diskriminační povahu automatické evaluace.

8.2 Vyhodnocení výsledného systému

Pro vyhodnocení celého systému byla vybrána datová sada SQAD v3.0 [21], ze které byly odebrány ano/ne otázky. (??a otázky, které nedávají bez přiloženého kontextu smysl.)

Pro každou metriku v 8.1 bylo použito vzorce 8.3

$$\text{skóre} = \frac{1}{N} \cdot \sum_{k=0}^{N} S_k \tag{8.3}$$

kde S_k je skóre S (např. EM) pro odpověď k a N je celkový počet odpovědí. Skóre na celém datasetu je tedy průměrem jednotlivých skóre S_k všech zodpovězených otázek.

V této sekci jsou shrnuty a porovnány výsledky výsledného systému na testovací sadě.

8.2.1 Model ALBERT a strojový překlad

ALBERT	EM	ELM	NS	T3M	MRR	F1	člověk
SQuAD 1.1+MD	35,53	37,33	?	?	?	37,70	?
SQuAD 2.0+MD	33,53	35,33	?	?	?	35,51	?
SQuAD 1.1+Maj	?	?	?	?	?	?	?
SQuAD 2.0+Maj	?	?	?	?	?	?	?

Tabulka 8.1: Úspěšnost systému s readerem ALBERT a strojovým překladem textů (prozatím 500 otázek).

8.2.2 Model vícejazyčný BERT

mBERT	EM	ELM	NS	T3M	MRR	F1	člověk
SQuAD 1.1+MD	48,49	50,98	61,12	55,73	51,99	57,44	?
SQuAD 2.0+MD	51,50	53,60	64,17	56,33	53,63	59,92	?
SQuAD 1.1+Maj	?	?	?	?	?	?	?
SQuAD 2.0+Maj	?	?	?	?	?	?	?

Tabulka 8.2: Úspěšnost systému s readerem ALBERT a strojovým překladem textů (prozatím 500 otázek).

8.2.3 Porovnání dosažených výsledků

8.3 Porovnání výsledků se současným stavem poznání

Porovnat s AQA atp. [13]

8.4 Rozbor chyb a možnosti dalšího vývoje

Popsat příčiny chyb, nejlepší systém, nejčastější nedostatky, co by se dalo zlepšit...

Kapitola 9

Závěr

Literatura

[1] Avinash Sharma V. *Understanding Activation Functions in Neural Networks* [online]. 2017. [cit. 20-03-2021]. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[2] Bahdanau, D., Cho, K. a Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv*. Září 2014, sv. 1409. Dostupné z: https://arxiv.org/abs/1409.0473.

[3] Chen, D., Fisch, A., Weston, J. a Bordes, A. Reading Wikipedia to Answer Open-Domain Questions. *CoRR*. 2017, abs/1704.00051. Dostupné z: http://arxiv.org/abs/1704.00051.

[4] Christopher Olah. *Understanding LSTM Networks* [online]. 2015. [cit. 20-03-2021]. Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[5] Devlin, J., Chang, M., Lee, K. a Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. 2018, abs/1810.04805. Dostupné z: http://arxiv.org/abs/1810.04805.

[6] Hochreiter, S. a Schmidhuber, J. Long Short-term Memory. *Neural computation*. Prosinec 1997, sv. 9, s. 1735–80. DOI: 10.1162/neco.1997.9.8.1735. Dostupné z: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory.

[7] Jay Alammar. *The Illustrated Transformer* [online]. 2018. [cit. 17-03-2021]. Dostupné z: http://jalammar.github.io/illustrated-transformer/.

[8] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P. et al. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *CoRR*. 2019, abs/1909.11942. Dostupné z: http://arxiv.org/abs/1909.11942.

[9] Lv, Y. a Zhai, C. Lower-Bounding Term Frequency Normalization. In:. New York, NY, USA: Association for Computing Machinery, 2011, s. 7–16. CIKM '11. DOI: 10.1145/2063576.2063584. ISBN 9781450307178. Dostupné z: https://doi.org/10.1145/2063576.2063584.

[10] Lv, Y. a Zhai, C. When Documents Are Very Long, BM25 Fails! In:. New York, NY, USA: Association for Computing Machinery, 2011, s. 1103–1104. SIGIR '11. DOI: 10.1145/2009916.2010070. ISBN 9781450307574. Dostupné z: https://doi.org/10.1145/2009916.2010070.

[11] Macková, K. a Straka, M. *Reading Comprehension in Czech via Machine Translation and Cross-lingual Transfer*. 2020. Dostupné z: https://arxiv.org/abs/2007.01667.

[12] Manning, C. D., Raghavan, P. a Schütze, H. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. ISBN 978-0-521-86571-5. Dostupné z: http://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[13] Medved', M. a Horák, A. AQA: Automatic Question Answering System for Czech. In: Sojka, P., Horák, A., Kopeček, I. a Pala, K., ed. *Text, Speech, and Dialogue*. Cham: Springer International Publishing, 2016, s. 270–278. ISBN 978-3-319-45510-5.

[14] Mikolov, T., Chen, K., Corrado, G. a Dean, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Dostupné z: https://arxiv.org/abs/1301.3781.

[15] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. a Dean, J. Distributed Representations of Words and Phrases and their Compositionality. *CoRR*. 2013, abs/1310.4546. Dostupné z: http://arxiv.org/abs/1310.4546.

[16] Min, S., Boyd Graber, J., Alberti, C., Chen, D., Choi, E. et al. *NeurIPS 2020 EfficientQA Competition: Systems, Analyses and Lessons Learned*. 2021.

[17] Pennington, J., Socher, R. a Manning, C. Glove: Global Vectors for Word Representation. In:. Leden 2014, sv. 14, s. 1532–1543. DOI: 10.3115/v1/D14-1162. Dostupné z: https://www.researchgate.net/publication/284576917_Glove_Global_Vectors_for_Word_Representation.

[18] Rajpurkar, P., Jia, R. a Liang, P. Know What You Don't Know: Unanswerable Questions for SQuAD. *CoRR*. 2018, abs/1806.03822. Dostupné z: http://arxiv.org/abs/1806.03822.

[19] Rajpurkar, P., Zhang, J., Lopyrev, K. a Liang, P. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. *CoRR*. 2016, abs/1606.05250. Dostupné z: http://arxiv.org/abs/1606.05250.

[20] Robertson, S., Walker, S., Jones, S., Hancock Beaulieu, M. a Gatford, M. Okapi at TREC-3. In:. 1994. Dostupné z: https://www.researchgate.net/publication/221037764_Okapi_at_TREC-3.

[21] Sabol, R., Medveď, M. a Horák, A. Czech Question Answering with Extended SQAD v3.0 Benchmark Dataset. In: Horák, A., Rychlý, P. a Rambousek, A., ed. *Proceedings of the Thirteenth Workshop on Recent Advances in Slavonic Natural Languages Processing, RASLAN 2019* [tištěná verze "print"]. Brno: Tribun EU, 2019, s. 99–108. ISBN 978-80-263-1530-8. [cit. 17-03-2021]. Dostupné z: https://nlp.fi.muni.cz/raslan/2019/paper14-medved.pdf.

[22] Tony Yiu. *Understanding Neural Networks* [online]. 2019. [cit. 19-03-2021]. Dostupné z: https://towardsdatascience.com/understanding-neural-networks-19020b758230.

[23] Trotman, A., Puurula, A. a Burgess, B. Improvements to BM25 and Language Models Examined. In: *Proceedings of the 2014 Australasian Document Computing Symposium*. New York, NY, USA: Association for Computing Machinery, 2014, s. 58–65. ADCS '14. DOI: 10.1145/2682862.2682863. ISBN 9781450330008. Dostupné z: https://doi.org/10.1145/2682862.2682863.

[24] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention Is All You Need. *CoRR*. 2017, abs/1706.03762. Dostupné z: http://arxiv.org/abs/1706.03762.

[25] Wikipedia contributors. *Příloha:Frekvenční seznam (čeština) — Wikislovník* [online]. 2020. [cit 17-03-2021]. Dostupné z: https://cs.wiktionary.org/wiki/P%C5%99%C3%ADloha:Frekven%C4%8Dn%C3%AD_seznam_(%C4%8De%C5%A1tina).

[26] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M. et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*. 2016, abs/1609.08144. Dostupné z: http://arxiv.org/abs/1609.08144.

[27] Zhu, M., Ahuja, A., Juan, D.-C., Wei, W. a Reddy, C. K. Question Answering with Long Multiple-Span Answers. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Listopad 2020, s. 3840–3849. DOI: 10.18653/v1/2020.findings-emnlp.342. Dostupné z: https://www.aclweb.org/anthology/2020.findings-emnlp.342.