FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Detekce podobností datových domén pomocí metod strojového učení
Student:	Bc. Andrej Oliver Chudý
Vedoucí:	Ing. Zdeněk Buk, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je navrhnout, implementovat a otestovat systém pro efektivní vyhledávání podobných sloupců napříč velkému množství databázových tabulek. Z důvodu velkého množství dat se zaměřte na řešení založené na vytvoření vektorových reprezentací pro každý sloupec (fingerprint). Tato nová reprezentace bude využívána při aplikaci dotazu na nejpodobnější sloupce. Řešení musí být připraveno na vstupy v různých kódováních a abecedách (azbuka, čínské znaky, apod.). Za účelem dosáhnutí co nejpřesnějších výsledků, experimentujte s různými modely. Všechny tyto přístupy porovnejte.

Seznam odborné literatury

[1] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, Y. Wu, and G. Brain: Exploring the Limits of Language Modeling, 2019, arXiv:1602.02410v2
[2] Chollet, F.: *Deep learning with Python*. Shelter Island, NY: Manning Publications, 2018
[3] Patterson, J., & Gibson, A.:Deep learning: A practitioners approach. Beijing, OReilly Media, 2017

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Detekce podobností datových domén pomocí metod strojového učení

Bc. Andrej Oliver Chudý

Katedra aplikovanej matematiky
Vedúci práce: Ing. Zdeněk Buk, Ph.D.

6. mája 2019

Poďakovanie

Veľké poďakovanie patrí firme Ataccama Software, s.r.o., ktorá mi umožnila pracovať na tejto téme a poskytla veľké množstvo dát. Osobitná vďaka patrí RNDr. Jakubovi Kúdelovi, ktorý viedol túto prácu po technickej stránke a všetkým ostatným ľuďom, ktorý sa podieľali na korekcií a verifikácií. Ďakujem.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 6. mája 2019 .....

Odkaz na túto prácu

Chudý, Andrej Oliver. *Detekce podobností datových domén pomocí metod strojového učení*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Cieľom tejto práce je navrhnúť a zostrojiť systém, na základe ktorého by bolo možne efektívne porovnávať podobnosť v stĺpcoch tabuľky. Bolo preukázané, že vektorová reprezentácia stĺpca vytvorená pomocou rekurentnej neurónovej siete je schopná dobre zakódovať vlastnosti domény, ktorú reprezentuje. V porovnaní s TF-IDF metódou, ktorá je na tento účel najčastejšie používaná, RNN dosiahli zlepšenie až o **14,5%**. Na základe výsledkov tejto práce bol implementovaný a nasadený systém na doporučovanie Business Terms v produkte Ataccama One.

Klíčová slova embedding, rekurentné neuronové siete, siamese, triplet, sesequence to sequence, seq2seq, gpt2, databáza, detekcia cudzích kľúčov, doporučovanie Business Terms, LSTM, GRU, cuDNNGRU.

Abstract

This thesis describes the design and implementation of a system for comparing the similarity of columns in an arbitrary database. We have shown that our system, based on recurrent neural networks, outperforms the industry standard TF-IDF method by **14.5%**. We therefore conclude that our system is capable of learning to effectively recognize the domain properties of data in

the database. We deployed the described system in Atacama One, where it is responsible for Business Terms recommendations.

Keywords embedding, recurrent neural networks, siamese, triplet, sesequence to sequence, seq2seq, gpt2, database, similarity, column, foreign key detection, Business Terms suggestion, LSTM, GRU, cuDNNGRU.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Väčšina dnešných organizácií produkuje obrovské množstvo dát, ktoré sa zhromažďujú do dátových skladov. Máloktorá organizácia však dokáže efektívne spájať dáta v rôznych dátových skladoch a získať tak kompletný obraz o povahe a prepojeniach všetkých existujúcich dat. V praxi často dochádza ku vzniku dátovo izolovaných systémov, ktoré spolu nedokážu komunikovať, čo môže byť spôsobené nekompatibilnými technológiami alebo manažovaním rôznymi tímami.

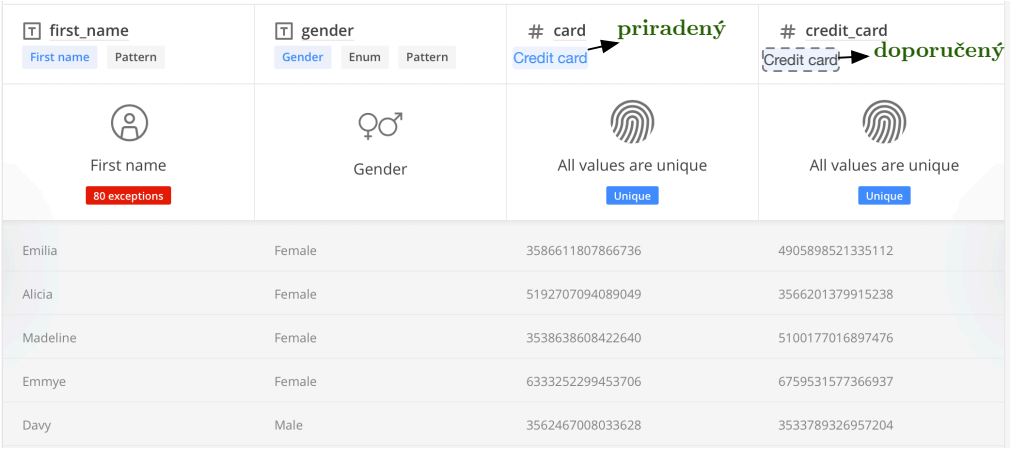
Tento problém sa organizácie snažia v dnešnej dobe riešiť vytvorením centrálnych systémov, ktoré kategorizujú, čistia a spájajú dáta z rôznych interných zdrojov tak, aby tie boli pripravené na ďalšiu analýzu. Avšak na to, aby centrálny systém dokázal odhaliť vzťahy medzi dátami v rôznych skladoch musí pochopiť, ktoré dáta reprezentujú rovnaké entity.

Napríklad, bežný internetový obchod bude mat databázovú tabuľku so všetkými užívateľmi uloženú v SQL databáze a zároveň môže mať log všetkých interakcií každého užívateľa uložený v Amazon S3. Centrálny systém umožni analytikom spojiť tieto heterogénne dáta a týmto získať komplexný pohlaď na správanie užívateľa.

Avšak, heterogénne systémy mnohokrát neukladajú sémantickú povahu dát alebo pri tom používajú rôzne konvencie. Napríklad, v jednej tabuľke môže byt emailová adresa používateľa označená ako `email_address` a v inej iba ako `mail`. Centrálny systém ich potom nedokáže spojiť, lebo nevie rozoznať, či ide o rovnaký druh dát. Riešením je implementovať systém, ktorý automaticky rozpozná podobnosť a dokáže na základe samotných dat odporučiť ich predpokladaný význam (Doporučovanie Business Terms).

Doporučovanie Business Terms

Informácie o podobnosti stĺpcov je možné využiť napríklad pri pridaní novej tabuľky, ktorú chceme obohatiť o metadáta priradením *Business Terms*.

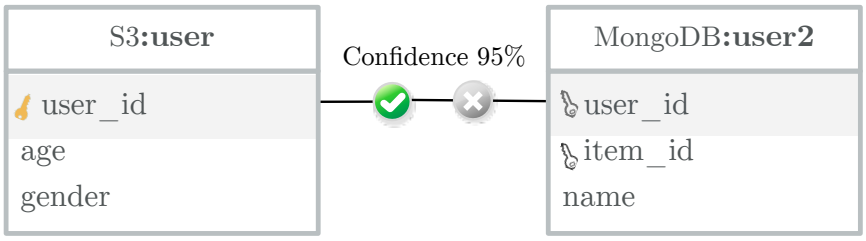


Obr. 0.1: Doporučovanie Business Terms. Prvým trom stĺpcom boli Business Terms priradené ručne. Štvrtý stĺpec `credit_card` disponuje doporučeným termom *Credit card* na základe podobnosti dát zo stĺpcom `card`

Ten vyjadruje význam dátovej domény v spojitosti s biznisom z ktorého dáta vznikli. Príkladom je sériové číslo produktu alebo emailová adresa. V organizácií typicky existuje preddefinovaný podnikový slovník, ktorý jednotlivé *Business Terms* obsahuje a vysvetľuje. Naivný prístup k automatizácií tohto procesu vedie na regulárne výrazy alebo porovnanie údajov s niektorými referenčnými vzorkami. Tento prístup však nie je veľmi flexibilný a pri rozsiahlom podnikovom slovníku je aj veľmi pomalý. Neodráža interakcie používateľa a zodpovedá vždy len tým istým údajom, ktorý je výsledkom pred-definovaného regulárneho výrazu. Doporučenie *Business Terms* na základe strojového učenia je schopné poučiť sa z používateľského vstupu (priatie/zamietnutie odporučeného termu) a odvodiť vhodné pravidlá bez explicitného zásahu do funkcie porovnania. Model rozpozná podobnosť medzi jednotlivými stĺpcami dátového skladu a návrhy na priradenie môže poskytovať na základe úrovne podobnosti s dátami, ktoré už *Business Term* obsahujú a tak distribuovať definovaný tento *Business Term* do ďalších častí dátového skladu. Príklad použitia je prezentovaný na obrázku 0.1.

Detekcia vzájomných vzťahov

Vo všeobecnosti podnik disponuje množstvom súvisiacich údajov. Existuje však nedostatočné množstvo možností na zachytenie vzťahov naprieč viacerými databázovými systémami. To značne komplikuje získanie komplexného pohľadu na systém ako celok. Na základe detekcie podobnosti by mohlo byť možné identifikovať vzťah primárneho a cudzieho kľúča, alebo vzťah tabuľky naprieč databázovými systémami či duplikáty v údajoch. Príklad nájdeného



Obr. 0.2: Detekcia podobnosti cudzieho a primárneho kľúča. V ľavej časti obrázku sa nachádza tabuľka `user` na Postgres a v pravej časti obrázku sa nachádza tabuľka user2 na MongoDB databázovom systéme. Detekcia vzájomných vzťahov ukázala vysokú zhodu preto existuje predpoklad, že sa jedná o dodatočné údaje, ktoré patria k tabuľke `user`

vzťahu medzi dvoma tabuľkami je prezentovaný na obrázku 0.2.

Cieľ práce

Táto práca si kladie za cieľ vytvoriť metódu porovnania databázových stĺpcov na základe dát. Práca sa zameria na metódy strojového učenia. Pri návrhu riešenia musí byť metóda pripravená na prácu zo stovkami tisíc stĺpcov a preto vyhľadávanie musí pracovať rýchlo.

Kapitola 1

Teoretický úvod do problematiky

Táto práca sa zaoberá riešením opísaného problému za využitia metód hlbokého učenia. Preto je dôležité na začiatok definovať, čo vlastne hlboké učenie je a aké ma črty.

1.1 Čo je hlboké učenie (Deep Learning)?

Je členom širšej rodiny metód strojového učenia založených na umelých neurónových sieťach. Počet týchto vrstiev je inak nazývaný aj hĺbka modelu. Moderné modely obsahujú desiatky až stovky vrstiev, ktoré sa dokážu úspešne natrénovať len na základe ukazovania tréningových príkladov dát. Túto vrstvovú reprezentáciu voláme neurónová sieť (NN). [1]

1.1.1 Tréning neurónových sietí

Tréningový proces v NN je zobrazenie vstupu (obrázky, text, a iné) na očakávaný výstup (kategória, spojitá hodnota a iné). Táto transformácia dát je zložená z jednoduchších transformácií, ktoré vykonávajú jednotlivé vrstvy, na základe naučených sád parametrov špecificky pre danú úlohu.[1]

Špecifikácia interakcie vrstvy so vstupnými dátami je definovaná vo váhach vrstvy, čo je v podstate n-dimenzionálna matica čísel. Z technického hľadiska by sme povedali, že transformácia realizovaná vrstvou je parametrizovaná váhami. V tomto kontexte učenie znamená nájdenie vyhovujúcich hodnôt pre váhy všetkých vrstiev v sieti, takže sieť bude správne mapovať príklady vstupov do svojich priradených cieľov. Avšak takáto neurónová sieť môže obsahovať desiatky miliónov parametrov. Hľadanie správnych hodnôt pre všetky z nich je veľmi náročná optimalizačná úloha.[1] [2]

Ak chcete posúdiť výstup neurónovej siete, musíte byť schopní merať, do akej miery je tento výstup správny. Toto je úloha takzvanej *loss function*,

ktorá môže byť reprezentovaná napríklad funkciou pre výpočet Euklidovej vzdialenosti. Jej úlohou je posúdiť ako dobre NN plní úlohu, ktorú od nej vyžadujeme. Na základe predpovedanej a skutočnej hodnoty vypočíta skóre, ktoré sa zmenšuje, keď sa sieti darí a zväčšuje keď sa sieti naopak nedarí. Základná metóda v hlbokom učení je použiť toto skóre ako signál spätnej väzby, aby sa hodnota váh trochu upravila v smere, ktorý zníži výstup *loss function* pre aktuálny príklad. Táto úprava je úlohou optimalizátora, ktorý môže predstavovať napríklad *Backpropagation*, ktorý je najznámejším algoritmom v strojovom učení pre optimalizáciu váh neurónovej siete.[1] [2]

1.2 Rekurentné neuronové siete (RNNs)

Princíp práce rekurentnej neurónovej siete (viď. obrázok 1.1) je viac podobný ľudskému mozgu ako klasická dopredná sieť. Biologická inteligencia spracováva informácie postupne, čo znamená, že predchádzajúca informácia dokáže ovplyvniť tú budúcu. Toto klasické dopredné siete nedokážu. RNNs sa snažia riešiť tento problém. Disponujú takzvaným vnútorným stavom c, ktorý operuje s variabilnými dĺžkami sekvencii vstupov $x = (x_1, ..., x_T)$. V každom čase t, je vnútorný stav c aktualizovaný ako

$$c_{<t>} = f(c_{<t-1>}, x_t) \tag{1.1}$$

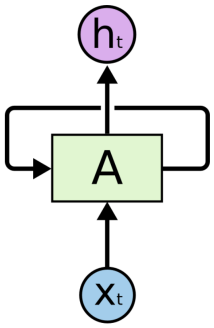
kde f môže byť jednoduchá ako logistická sigmoida alebo komplexná ako LSTM [3]. RNN sa môže naučiť rozdeleniu pravdepodobnosti v priebehu sekvencie tým, že sieť natrénujeme na úlohe predpovedania ďalšieho symbolu v sekvencii. V takom prípade výstup v čase t je podmienenou pravdepodobnosťou $p(x_t|x_{t-1}, ..., x_1)$. [1] [2]

Model potom môže pracovať s dátami, ktoré sa vyvíjajú v čase. Z historického hľadiska bolo ťažké tieto siete trénovať, ale v poslednom čase boli zaznamenané obrovské pokroky vo výskume oblastí optimalizácie, architektúr sietí, paralelizmu a grafických procesorov (GPU). Vďaka schopnosti zohľadňovať pri výpočte čas sa tento typ sietí najčastejšie využíva v oblastiach rozpoznávania reči, jazykového modelovania, prekladu a ďalších. [2]

1.2.1 Markov model vs RNN

Keď sa zaoberáme časovou dimenziou v našich modeloch, prirodzene by sme mohli zvážiť Markovove modely ako možnosť. Sú ďalšou triedou modelov strojového učenia, ktoré boli široko používané na modelovanie sekvencií. Avšak s rastúcim kontextovým oknom sa tieto modely stávajú výpočtovo náročnými pre modelovanie závislostí na dlhých vzdialenostiach.[2]

Rekurentné neurónové siete sú lepšie ako Markovove modely (a iné modely s časovým oknom), pretože vo vstupných údajoch dokážu zachytiť časové závislosti na dlhé vzdialenosti. Umožňuje im to ich skrytý stav zachytávajúc



Obr. 1.1: Rekurentná neurónová sieť [4]. Na obrázku je znázornené rekurentné prepojenie, ktorým disponuje každý druh RNN.

informácie z ľubovoľne dlhého kontextu. Počet stavov, rastie exponenciálne s počtom skrytých uzlov vrstvy. Vďaka tomu sú rekurentné neurónové siete dobre uspôsobené pri zachytávaní veľkého množstva časovo relevantných informácií v mnohých vstupných vektoroch.[2]

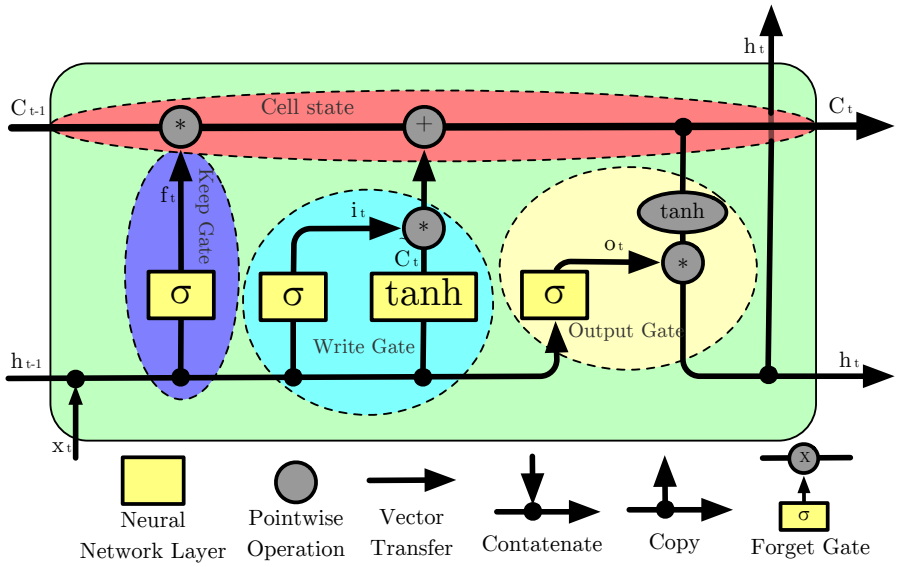
1.2.2 Long Short Term Memory (LSTM)

LSTM siete sú často používanou variantov rekurentných neurónových sietí. Prvýkrát sa objavili v roku 1997 v publikácii Long-Short Term Memory [3]. Je to špeciálny druh RNN, schopný učiť sa dlhodobým časovým závislostiam. Zapamätanie si informácií po dlhú dobu je prakticky predvolené správanie LSTM sieti vďaka ich architektúre. [4]

Štruktúra bunky LSTM je vizualizovaná na obrázku 1.2. Jej hlavná komponenta je *cell state*, horizontálna čiara prechádzajúca cez vrchol diagramu. Táto komponenta správaním pripomína dopravníkový pás. Posúva vnútorný stav c_{n-1} do stavu c_n. Ostatné komponenty LSTM predstavujú brány (gates), ktorých úlohou je riadiť tok informácii prechádzajúcich do nasledujúceho stavu. Vďaka tomuto prepojeniu je LSTM schopná prenášať dlhodobé závislosti naprieč svojimi stavmi. V nasledujúcej časti budú opísané funkčnosti všetkých brán, ktoré participujú na výslednom stave. [4] [1]

Keep Gate zvýraznená na obrázku 1.2 pozostáva zo sigmoind neurónovej vrstvy, ktorá sa vyznačuje výstupným intervalom hodnôt $< 0, 1 >$. V spojení s násobením, toto zapojenie pracuje ako zabúdacia komponenta. V prípade, že výstup je nula, hodnota Cell state z predchádzajúceho stavu je kompletne zabudnutá (vynulovaná). Naproti tomu ak je výstup jednotkový, celá informácia sa ponechá a nijako nemodifikuje Cell Statem na ktorý je Keep Gate napojený. To môžeme matematicky vyjadriť nasledovne: [4] [1]

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f), \tag{1.2}$$



Obr. 1.2: Bunka LSTM. Vo vrchnej časti obrázka sa nachádza *cell state*. Jednotlivé brány riadia, ktoré informácie sa odovzdajú do nasledujúceho stavu a ktoré nie. [4]

kde σ predstavuje aktivačnú funkciu sigmoid, h_{t-1} je rekurentný stav, x_t reprezentuje aktuálny vstup, W_i váhy a b_i je konštanta.

Nasledujúca komponenta je **Write Gate**. Táto komponenta je zodpovedná za rozhodnutie, ktoré relevantné informácie sa zapíšu do vnútorného stavu **Cell State**. Samotná komponenta sa skladá z dvoch častí. Prvá sigmoid vrstva zodpovedá za rozhodnutie, ktorá informácia bude posunutá a druhá časť je *tahn* funkcia, ktorá vytvára vektor nových kandidátnych hodnôt $\tilde{C}$. Matematicky možno tento vzťah popísať takto:

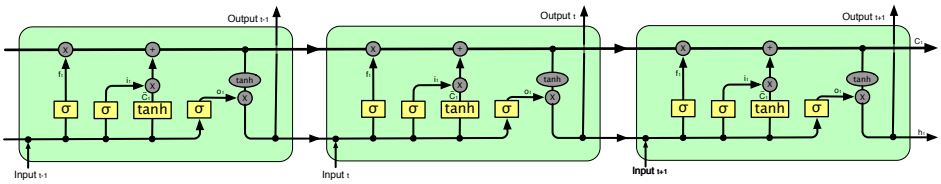
$$\begin{aligned} i_t &= \sigma(W_i * [h_{t-1}, x_t] + b_i), \\ \tilde{C}_t &= tanh(W_C * [h_{t-1}, x_t] + b_C), \end{aligned} \tag{1.3}$$

kde σ predstavuje aktivačnú funkciu sigmoid, h_{t-1} je rekurentný stav, x_t reprezentuje aktuálny vstup, W_i, W_c váhy vrstvy a b_i, b_c sú konštanty. [4] [1]

V tomto bode už vieme, ako veľmi chceme brať do úvahy informácie z predchádzajúceho stavu a aj to, ktoré informácie chceme posunúť do stavu nasledujúceho. Preto môžeme vypočítať výstupnú hodnotu **Cell State** C_t ako :

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{1.4}$$

Cieľom poslednej komponenty LSTM bunky je pripraviť jej výstup. Na tento výpočet je využitý vnútorný stav reprezentovaný výstupnou hodnotou komponenty **Cell State** a spojené vstupné hodnoty h_{t-1} a x_t. Výsledok sa



Obr. 1.3: Rozvinuté rekurentné prepojenie LSTM v čase. Obrázok nepredstavuje viacero LSTM prepojených vzájomne, len viacero stavou jednej vrstvy, ktoré sa menia v čase.

súčasne distribuuje ako výstup a zároveň rekurentný vstup do nasledujúceho stavu. Vďaka tomuto LSTM dokáže modelovať aj krátke závislosti, ktoré sú do veľkej mieri riadené práve h_t výstupom:

$$\begin{aligned} o_t &= \sigma(W_o * [h_{t-1}, x_t] + b_o), \\ h_t &= o_t * tanh(C_t), \end{aligned} \tag{1.5}$$

kde σ predstavuje aktivačnú funkciu sigmoid, h_{t-1} reprezentuje stav h z predchádzajúceho stavu, x_t reprezentuje aktuálny vstup, W_o váhy vrstvy a b_o je konštanta. [4] [1]

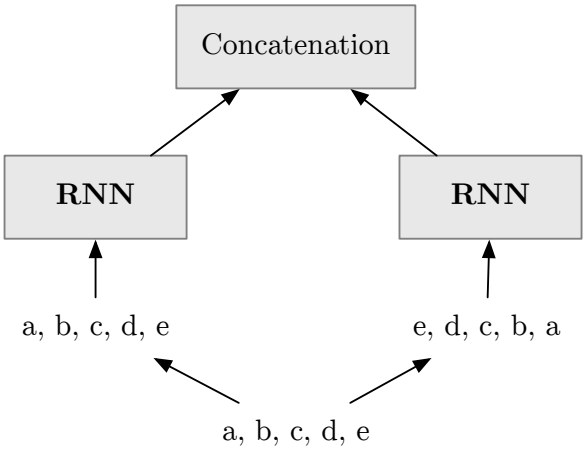
Kľúčovým poznatkom tohto typu rekurentnej vrstvy je spôsob, akým sa informácie šíria cez sieť. Pri rozložení LSTM vrstvy v čase (obrázok 1.3) vo vrchnej časti obrázka môžeme vidieť šírenie stavového vektora, ktorého interakcia je lineárna v čase. Táto technika sa vyhýba **Gradient vanishing** problému, ktorý je opísaný v ďalšom texte. LSTM je možné použiť na generovanie viet, klasifikáciu časových rád, rozpoznávanie reči, rozpoznávanie rukopisu, modelovanie polyfónnej hudby a iné úlohy v reálnom svete. [4] [1]

Výrazné zlepšenie fungovania LSTM modelov na NLP úlohy bolo zaznamenané v špeciálnom zapojení BLSTM (Bidirectional LSTM). Spájajú dve skryté vrstvy opačných smerov s rovnakým výstupom (viď. obrazok 1.4). S touto formou generatívneho hlbokého učenia môže výstupná vrstva získať informácie zo spätných aj popredných stavov súčasne. [1]

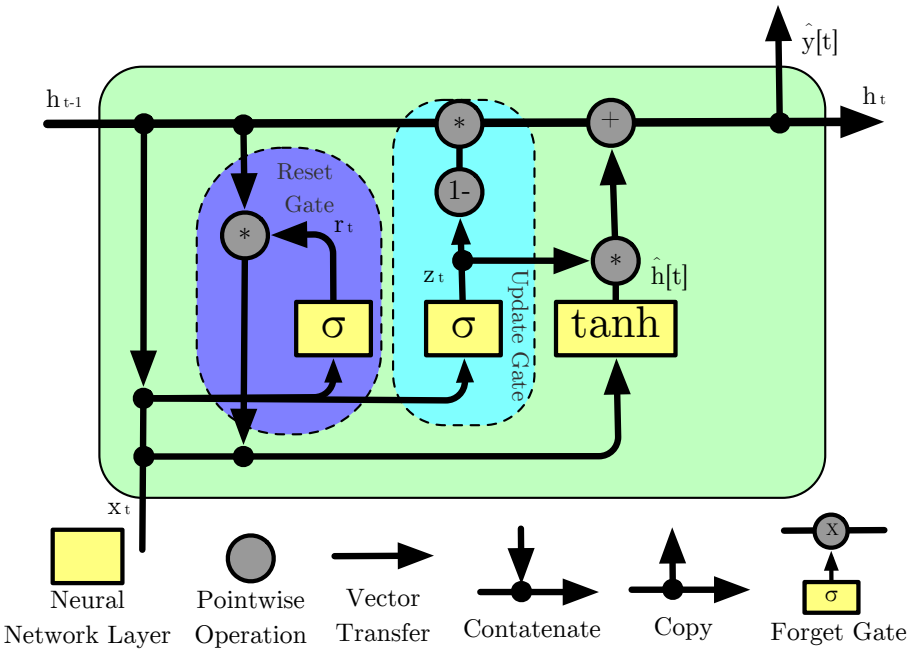
1.2.3 Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) [5] je novšia generácia rekurentných neurónových sietí a je veľmi podobná LSTM. V mnohých použitiach ponúka porovnateľnú presnosť ale je rýchlejšia.

Kľúčovou vlastnosťou, rovnako ako v LSTM, je riadenie vnútorného stavu pomocou *gates*. To znamená, že máme vyhradené mechanizmy na aktualizáciu skrytého stavu a tiež na jeho resetovanie (viď. obrázok 1.5). Napríklad, ak má prvý symbol veľký význam, vrstva sa naučí neaktualizovať skrytý stav po prvom vstupe. Celkovo GRU disponuje dvoma bránami:



Obr. 1.4: Obojsmerné (bidirectional) zapojenie RNN. Na obrázku je znázornené rozdelenie vstupu a následné privedenie do RNN, ktoré majú zdielané váhy. Výstupy sú konkatenované, preto dimenzionalita výstupného vektora je dvojnásobna oproti dimenzionalite samotného RNN.



Obr. 1.5: GRU bunka [4]. Na obrázku je znázornená GRU bunka, v ktorej sú vyznačené jej dve brány, ktoré používa na ovplyvnenie hodnoty vnutorného stavu.

1. **Update gate** - Aktualizačná brána rozhoduje o tom, koľko z predchádzajúcej pamäte sa má uchovávať.

2. **Reset Input** - Resetovacia brána definuje, ako kombinovať nový vstup s predchádzajúcou hodnotou.

Sústava rovníc, ktorá popisuje chovanie GRU je definovaná nasledovne:

$$\begin{aligned} z_t &= \sigma(W_z * [h_{t-1}, x_t] + b_z) \\ r_t &= \sigma(W_r * [h_{t-1}, x_t] + b_r) \\ \tilde{h}_t &= tanh(W * [r_t * h_{t-1}, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned} \tag{1.6}$$

Pri rozhodovaní, či je lepšie použiť LSTM alebo GRU neexistuje jednoznačná odpoveď. Preto obvyklý postup pri riešení tohoto problému je založený na experimentálnom otestovaní oboch variant.

GRU je často používaná v spolupráci s knižnicou NVIDIA CUDA® Deep Neural Network (cuDNN). Jedná sa o GPU akcelerovanú knižnicu pre hlboké neuronové siete. CuDNN poskytuje vysoko vyladené implementácie pre štandardné rutiny, ktorých súčasťou je aj implementácia GRU (CuDNNGRU).

1.2.4 Vanishing gradients

Vanishing gradients je problém objavujúci sa pri hlbokých sieťach. Vyplýva z podstaty fungovania optimalizačného algoritmu **Backpropagation**, ktorý šíri spätnoväzbový signál z výstupu do skorších vrstiev a tým riadi konvergenciu modelu k nájdeniu riešenia. Ak sa tento signál spätnej väzby musí šíriť hlbokým zapojením vrstiev, signál sa môže stať slabým alebo sa môže úplne stratiť, čím sa sieť stane netrénovateľnou. [1] LSTM aj GRU rieši tento problém za pomoci **Cell State**, fungovanie ktorého je opísané v sekcii 1.2.2, ktorý šíri informácie paralelne s hlavným prúdom spracovania. [1]

Kapitola 2

Návrh riešenia úlohy

Táto kapitola je zameraná na popis možných riešení daného problému. Venuje sa rozboru výhod a nevýhod jednotlivých prístupov, čoho výstupom je výber najvhodnejšej metódy.

2.1 Klasifikácia dát do jednotlivých domén

Jeden z možných pohľadov na riešenie problému podobnosti stĺpcov je pristupovať k úlohe ako ku klasickej muli-label klasifikácií. Vstupom modelu v takomto prípade by boli jednotlivé hodnoty stĺpca a výstupom by bola pravdepodobnosť priradenia do skupín, z ktorej každá definuje špecifickú kategóriu dát, ktorej budeme hovoriť doména. Za podobné stĺpce by sme potom mohli považovať tie, ktorým model pridelil rovnakú doménu.

Avšak na natrénovanie takéhoto modelu je potrebné disponovať trénovacími dátami, ktoré by obsahovali priradené dátové domény v žiadanej granularite (tréning s učiteľom). Ďalším problémom, ktorý prináša tento prístup je nutnosť pretrénovať model pri každom pridaní novej domény do podnikového slovníka. Najzásadnejší problém však spočíva v prípade keď sa modelu prezentujú dáta, ktoré predstavujú doposiaľ neexistujúcu kategóriu domény. Naivné očakávanie zachovania modelu k takejto situácii je informovať o vzniknutej situácii a ohodnotiť všetky kategórie domén nízkou pravdepodobnosťou. Toto je však extrémne náročné dosiahnuť pri takto definovanom modeli. Z jednoduchých pokusov, ktoré boli uskutočnené v rámci práce vyplynulo, že výsledný systém by bol ťažkopádny, pomalý, každé zmeny v katalógu obchodných názvov by si vyžadovali dlhý čas na pretrénovanie modelu a vynútenie schopnosti nevrátiť výsledok v prípadoch, ktoré si to vyžadujú, by bola skoro nemožná úloha.

2.2 Binárna klasifikácia

Ďalším prístupom, ktorý rieši mnohé nevýhody predchádzajúceho riešenia, je pristupovať k úlohe ako ku binárnej klasifikácií. Každá doména, definovaná v podnikovom slovníku, bude disponovať vlastným binárnym klasifikačným modelom, ktorý bude rozhodovať či vstupné dáta zodpovedajú danej doméne alebo nie. Inými slovami výsledný model sa bude skladať z takého počtu binárnych klasifikačných modelov, koľko je dátových domén v slovníku. Tento prístup značne eliminuje nutnosť pretrénovania celého systému pri pridaní novej domény, avšak stále je nutné natrénovať klasifikátor zodpovedný za pridanú doménu. Pri riešení otázky nepriradenia žiadneho termu je tiež tento prístup o niečo lepší ako v predchádzajúcom prípade, avšak stále nie dostatočne spoľahlivý.

Najzásadnejším problémom použitia tohto, na oko sľubného prístupu je však spôsob tréningu. Tréningové dáta by mali obsahovať pozitívne a negatívne vzorky. Avšak množina negatívnych vzoriek je vlastne doplnkom tých pozitívnych. Vybudovanie dostatočne variabilnej negatívnej množiny, ktorá by reprezentovala doplnok k tej pozitívnej je extrémne náročná.

2.3 Vytvorenie vektorovej reprezentácie

Predchádzajúce prístupy sa pokúšali problém podobnosti domén previesť na klasifikačné úlohy, na ktoré je možné celkom jednoducho natrénovať model. Avšak oba tieto prístupy si so sebou nesú požiadavku tréningu s učiteľom. V rámci existujúcich dátových skladov však nemožno predpokladať dostatočne veľkú tréningovú množinu dát.

V ideálnom prípade by sme potrebovali model, ktorý by nemusel byť trénovaný na cieľovom dátovom sklade, nepotreboval na svoj tréning množstvo anotovaných dát, umožňoval jednoduché pridanie alebo odstránenie novej domény či vysporiadal sa s požiadavkou detekcie neznámeho typu dát.

Spôsobom ako dosiahnuť všetky tieto kritéria je vytvorenie modelu, ktorý by bol schopný každému stĺpcu prideliť na základe syntaktických vlastnosti vektorovú reprezentáciu, na základe ktorej by bolo následné možné za pomoci použitia jednej z metrík (Cosinusova vzdialenosť, Euklidova vzdialenosť, atd.) posúdiť ako veľmi sú si dva stĺpce podobné. Riešenie úlohy automatického navrhovania Business Terms by mohlo vyzerať tak, že ak existuje stĺpec, ktorý má pridelený nejaký Business Term, stĺpce ktoré sú v jeho bezprostrednom okolí pravdepodobne tak isto pochádzajú z rovnakej domény a vhodne ich popisuje rovnaký Business Term. Takto postavený systém je veľmi efektívny pretože vektorová reprezentácia stĺpca sa môže počítať len raz a následne byť perzistentne uložená ako metainformácia stĺpca. To bude mať za následok, že pri porovnávaní podobností stĺpcov by nebolo nutné invokovať model a operácia by bola veľmi rýchla. Výnimočnou správou takisto je, že pridanie či

odoberanie domény (napr. v podobe Business Terms) nemá vplyv na funkčnosť modelu a nie je nutné žiadne pretrénovanie modelu.

2.4 Zhrnutie

V tejto kapitole boli zhrnuté základné možnosti prístupu k úlohe. Zistili sme, že najvhodnejšou metódou ako pracovať s podobnosťou dátových stĺpcov je vytvorenie vektorovej reprezentácie a následné použitie niektorej z vzdialenostným metrík, ktoré budú schopné posúdiť mieru podobnosti.

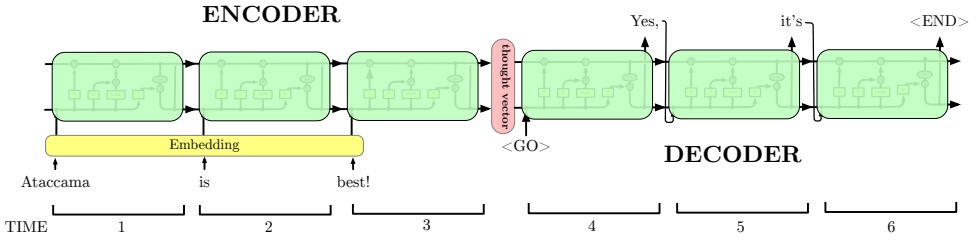
Kapitola 3

Topológie využitých sietí

V tejto kapitole budú popísané využité topológie pre vytvorenie vektorových reprezentácií stĺpcov. Ďalšom znení práce budú tieto prístupy podrobené testom a porovnaniu.

3.1 RNN Encoder-Decoder (Seq2seq)

Architektúra neurónovej siete známa ako RNN Encoder – Decoder (Seq2seq) sa skladá z dvoch rekurentných neurónových sietí, ktoré fungujú ako pár kodéra a dekodéra viď obrázok 3.1. Kodér mapuje zdrojovú sekvenciu s premennou dĺžkou na vektor s pevnou dĺžkou a dekodér mapuje vektorovú reprezentáciu späť na cieľovú sekvenciu s premenlivou dĺžkou. Obe siete sú spoločne trénované aby sa maximalizovala podmienená pravdepodobnosť cieľovej sekvencie danej zdrojovou sekvenciou. Príkladom môže byť preklad z anglického jazyka do francúzskeho jazyka. Práca *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation* [6] ukázala, že model Encoder-Decoder sa učí sémanticky a syntakticky zmysluplne reprezentovať text.



Obr. 3.1: Seq2seq topologia. V ľavej časti obrázka je Encoder, ktorý vytvára zo vstupného reťazca *tought vector*. Ten je nositeľom celej informácie, z ktorej následne Decoder rekonštruuje zodpovedajúci výstup.

Použitie tejto topológie preukázalo výborné výsledky naprieč rôznym úlohám NLP:

- **Strojový preklad** - publikácia spoločnosti Google [7] ukazuje, ako sa kvalita modelu Seq2seq prekladu približuje alebo prekonáva všetky aktuálne publikované výsledky
- **Rozpoznávanie reči** - ďalšia publikácia spoločnosti Google [8] porovnáva existujúce modely seq2seq na úlohe rozpoznávania reči
- **Generácia popisov filmov** - dokument [9] z roku 2015 ukazuje, ako saq2seq poskytuje skvelé výsledky pri vytváraní popisov filmov

3.1.1 Zapojenie autoencoder

Toto zapojenie pozostáva rovnako ako seq2seq z dvoch rekurentných neurónových sieti enkodéra a dekodéra (obr. 3.1). Špecifikum však spočíva vo forme vstupných dát privedených do topológie. Namiesto dvoch rôznych sekvenčných vstupov je privedený na vstup len jeden a sieť sa snaží dekódovať na výstupe rovnakú informáciu ako dostala na vstupe. Takto postavené optimalizačné kritérium dosiahne, že výstup enkodéra (**state vector**) bude obsahovať dostatočne relevantné informácie aby dekodér na druhej strane bol schopný iba na základe tohoto vektoru spätne rekonštruovať vstup. Veľká výhoda pri tréningu takejto siete spočíva v tom, že nepotrebujeme anotovať dáta. Ľubovolný vstup je zároveň aj očakávaným výstupom. Tréning teda môže prebiehať bez učiteľa. [10]

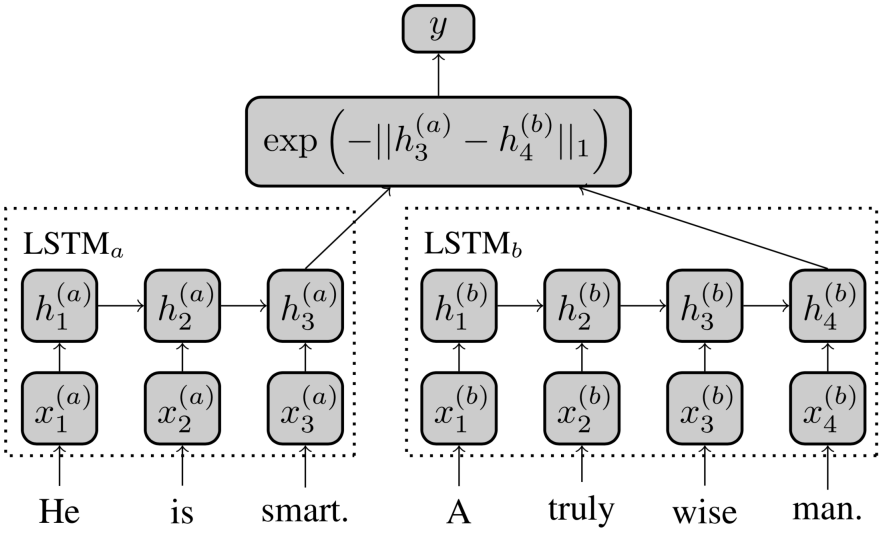
3.2 Siamese

Siamské siete sú špeciálnym typom architektúry neurónovej siete. Namiesto modelového učenia na klasifikáciu jeho vstupov sa neurónové siete učia rozlišovať medzi dvoma vstupmi. Sieť sa učí medzi nimi hodnotiť podobnosť. Skladajú sa z dvoch identických neurónových sietí, ktoré majú zdielané váhy. Každá z týchto sietí má na vstupe jeden z dvojice vstupných dát. Ich výstup sa následne vyhodnotí za pomoci niektorej zo vzdialenostných metrík ako napríklad Manhattan alebo Euclidian. [11]

3.2.1 Contrastive loss

Cieľom siamskej architektúry nie je klasifikovať vstupné dáta, ale rozlišovať medzi nimi. Preto je veľmi vhodné použiť ako stratovú funkciu *contrastive loss*[12]. Jej definícia vyzerá takto:

$$(1 - Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}max(0, m - D_2)^2 \tag{3.1}$$



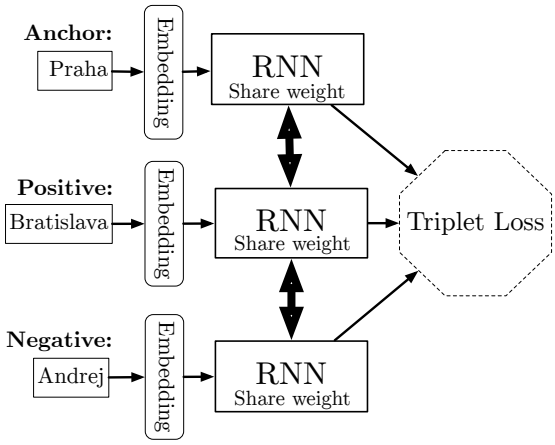
Obr. 3.2: MaLSTM [13]. Na obrázku sú znázornené dve LSTMa a LSTMb, ktoré majú vzájomne zdielané váhy. Výsledok oboch sietí je privedený do Manhatan metriky, ktorej výstupom je hodnota y.

kde D_w je definovaná ako euklidovská vzdialenosť medzi výstupmi siamesekých sietí a Y označuje či sa jedná o pozitívnu alebo negatívnu vzorku. Pozitívna vzorka reprezentuje pár, ktorý si je vzájomne podobný a je označený hodnotou 0, negatívna vzorka označuje nepodobný pár na vstupe a je označený 1. Najdôležitejšou premennou je hodnota $m > 0$. Určuje minimálnu vzdialenosť medzi pozitívnymi a negatívnymi vzorkami vo vektorovom priestore. Nastavenie tejto hodnoty vlastne umožňuje zvoliť hraničný bod, ktorý reprezentuje hranicu medzi pozitívnymi a negatívnymi podobnosťami už pri tréningu a nie je nutné ju následne štatisticky hľadať.[12]

3.2.2 MaLSTM

Adaptácia siamskej topológie pre RNN sa vyskytla v práci *Siamese Recurrent Architectures for Learning Sentence Similarity*[13], ktorá predstavila model MaLSTM, ktorý preukázal dobré výsledky na úlohe detekcie podobných viet. Model Manhattanu LSTM (MaLSTM) je znázornený na obr. 3.2. Skladá sa z dvoch vrstiev LSTMa a LSTMb, z ktorých každá spracováva jeden vstup z dvojice vstupných dát, avšak obe tieto vrstvy majú zdielané váhy.[13]

Každý token (slovo, písmeno) $x_1, ..., x_T$, ktorý je vstupom LSTM, aktualizuje vnútorný stav v každom indexe sekvencie. Konečná reprezentácia vety je použitá ako vstup do podobnostnej funkcie g, ktorá určí podobnosť na základe vzdialenostnej metriky.[13]



Obr. 3.3: Sieť, ktorá na základe triplet loss sa učí rozlišovať medzi mestami a menami.

Všimnite si, že na rozdiel od typického jazykového modelovania RNNs, ktoré sa používajú na predpovedanie ďalšieho slova uvedeného v predchádzajúcom texte, táto topológia jednoducho funguje ako kodér. Jediný chybový signál počas tréningu teda vychádza z podobnosti medzi reprezentáciami viet $h_T^{(a)}$, $h_T^{(b)}$, čo efektívne núti úplne zachytiť sémantické rozdiely dvojíc počas tréningu. Funkcia podobnosti potom môže vyzerať nasledovne: [13]

$$g(h_T^{(a)}, h_T^{(b)}) = \exp(-||h_T^{(a)} - h_T^{(b)}||_1) \in [0, 1] \tag{3.2}$$

3.3 Triplet

Táto topológia sa, rovnako ako siamese, učí na základe prezentovania rozdielnych a podobných typov dát. Avšak na rozdiel od Siamese sú prezentované trojice $x_i^a(anchor), x_i^b(positive), x_i^c(negative)$. Snaha procesu tréningu je minimalizovať vzdialenosť x_i^b od x_i^a a naopak maximalizovať vzdialenosť x_i^b od x_i^c súčasne (Obr 3.3). [14]

3.3.1 Triplet Loss

Od loss funkcie očakávame: [14]:

$$\begin{gathered} \|f\left(x_i^a\right) - f\left(x_i^p\right)\|_2^2 + \alpha < \|f\left(x_i^a\right) - f\left(x_i^n\right)\|_2^2 \\ \forall \left(f\left(x_i^a\right), f\left(x_i^p\right), f\left(x_i^n\right)\right) \in \mathcal{T}, \end{gathered} \tag{3.3}$$

kde α je *margin* (veľkosť vzájomnej vzdialenosti vo výsledkom priestore) medzi pozitívmi a negatívnymi pármi. τ je množina všetkých možných trojíc

v tréningovej množine s kardinalitou N. Potom stratová funkcia vyzerá nasledovne:

$$Loss = \sum_{i=1}^{N} [\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha]_+ \tag{3.4}$$

Vytvorenie všetkých možných trojíc by viedlo k mnohým tripletom, ktoré sú ľahko splnené (spĺňajú obmedzenia v rovnici 3.3). Takéto dáta by neobsahovali veľa informácií a viedlo by to pomalšej konvergencií tréningu modelu. Preto je vhodné vybrať trojice tak, aby obsahovali ťažko rozlíšiteľne trénovacie príklady (napríklad mená a mestá).

3.4 Hierarchická topológia

Základnou črtou tejto topológie je hierarchický prístup k vstupným dátam. Obvykle disponuje dvoma vrstvami, ktoré kooperujú na vytvorení reprezentácie, ktorá by odrážala charakter celku. Jedným zo zástupcov tohto typu modelu bol predstavený v práci [15], ktorá bola zameraná na klasifikáciu dokumentov. Výsledkom ich práce je model **HAN (Hierarchical Attention Network)**, ktorý dosiahol na úlohe klasifikácie dokumentov dobré výsledky.

3.4.1 Hierarchical Attantion network (HAT)

Model sa skladá zo štyroch základných častí, pričom každá má svoje špecifické poslanie pri tvorbe vhodnej reprezentácie dokumentu (viď. obrázok 3.4).

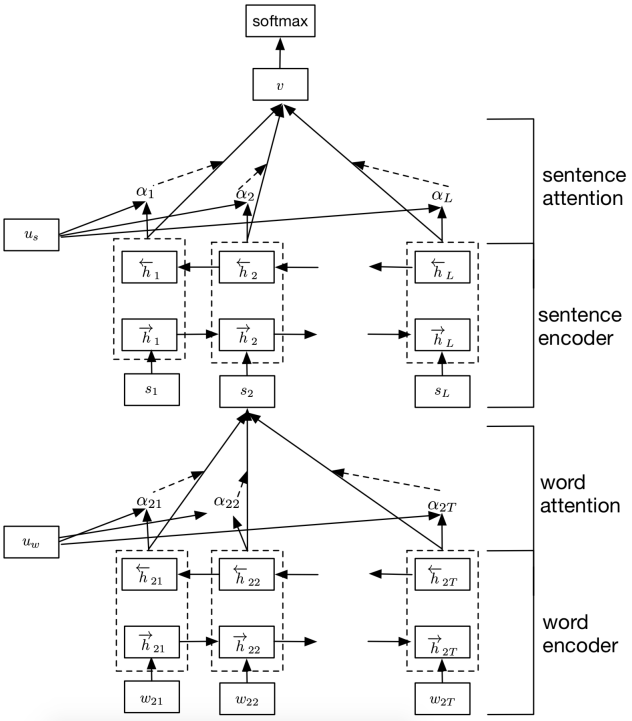
Word encoder - Predstavuje vrstvu zodpovednú za kódovanie jednotlivých slov vo vete. Prvý krok tejto vrstvy je vytvorenie vektorovej reprezentácie pre každé slovo. V druhom kroku sa snaží získať anotácie slov za použitia obojsmerného zapojenia GRU, ktorá sumarizuje informácie z oboch smerov pre slová a tak zapracúva kontextové informácie do anotácie. [15]

$$\begin{aligned} x_{it} &= W_e w_{ti}, t \in [1, T], \\ \overrightarrow{h}_{it} &= \overrightarrow{GRU}(x_{ut}), t \in [1, T], \\ \overleftarrow{h}_{i}t &= \overleftarrow{GRU}(x_{it}), t \in [1, T], \end{aligned} \tag{3.5}$$

kde, x_{it} je slovný vektor zodpovedajúci za slovo w_{it}, W_e je embedding matica a T je počet všetkých viet. Anotáciu pre dané slovo w_{it} získame konkatenáciou vnutorných stavov sietí v oboch smeroch nasledovne: [15]

$$h_{it} = [\overrightarrow{h}_{it}, \overleftarrow{h}_{i}t] \tag{3.6}$$

Word attention - Nie všetky slová prispievajú k reprezentácií významu vety rovnako. Preto je použitá *Attention* vrstva aby bolo možné extrahovať



Obr. 3.4: Hierarchival Attantion model [15]. Prvá časť siete je zodpovedná za kódovanie slov, z ktorého Attention vrstva vyberá najpodstatnejšie slová vety. Druhá časť RNN je zodpovedná za kódovanie viet a Attention vrsta je zodpovedná za výber relevantných viet dokumentu.

slová, ktoré sú dôležité a zoskupiť reprezentácie týchto informatívnych slov, z ktorých sa vytvorí relevantný vetný vektor. Toto chovanie možno popísať matematicky nasledovne:

$$\begin{aligned} u_{it} &= \tanh\left(W_w h_{it} + b_w\right) \\ \alpha_{it} &= \frac{\exp\left(u_{it}^\top u_w\right)}{\sum_t \exp\left(u_{it}^\top u_w\right)} \\ s_i &= \sum_t \alpha_{it} h_{it} \end{aligned} \tag{3.7}$$

v prvom kroku výpočtu je privedená slovná reprezentácia h_{it} na vstup Multilayer perceptron vrstvy, ktorá vytvorí vnútornú reprezentáciu u_{it} vstupu. Následne sa odmeria dôležitosť slova ako podobnosť u_{it} s kontextom na úrovni slov u_w. Po normalizácií za pomoci softmax vzniknú normalizované váhy dôležitosti. Nakoniec sa spočíta vetný vektor s_i ako vážený súčin slovných anotácií a váhového vektoru. Kontextový vektor u_w je náhodne inicializovaný a trénovaný. [15]

Sentence encoder - Postup na tejto úrovni modelu je veľmi podobný ako v jeho prvej vrstve. Pre získanie kontextových informácií pre každú vetu je použitá druhá BGRU.

$$\begin{aligned} \overrightarrow{h}_i &= \overrightarrow{GRU}(s_i), i \in [1, L], \\ \overleftarrow{h}_i &= \overleftarrow{GRU}(s_i), i \in [1, "] \end{aligned} \tag{3.8}$$

Následne sa $\overrightarrow{h}_i$ a $\overleftarrow{h}_i$ konkatenujú ako $h_{it} = [\overrightarrow{h}_i, \overleftarrow{h}_i]$. [15]

Sentence attention - Keďže aj na tejto vrstve chceme zvýrazniť vety, ktoré sú vodítkami na správne klasifikovanie dokumentu, znovu použijeme attention mechanizmus a v kontexte vetných vektorov vyberieme len najrelevantnejšie vety dokumentu.

$$\begin{aligned} u_i &= \tanh\left(W_s h_i + b_s\right) \\ \alpha_i &= \frac{\exp\left(u_i^\top u_s\right)}{\sum_i \exp\left(u_i^\top u_s\right)} \\ v &= \sum_i \alpha_i h_i \end{aligned} \tag{3.9}$$

kde v je vektor dokumentu, ktorý sumarizuje všetky informácie viet v dokumente. Podobne ako na leveli slov, kontext vektor je náhodne inicializovaný a trénovaný spoločne so sieťou. [15]

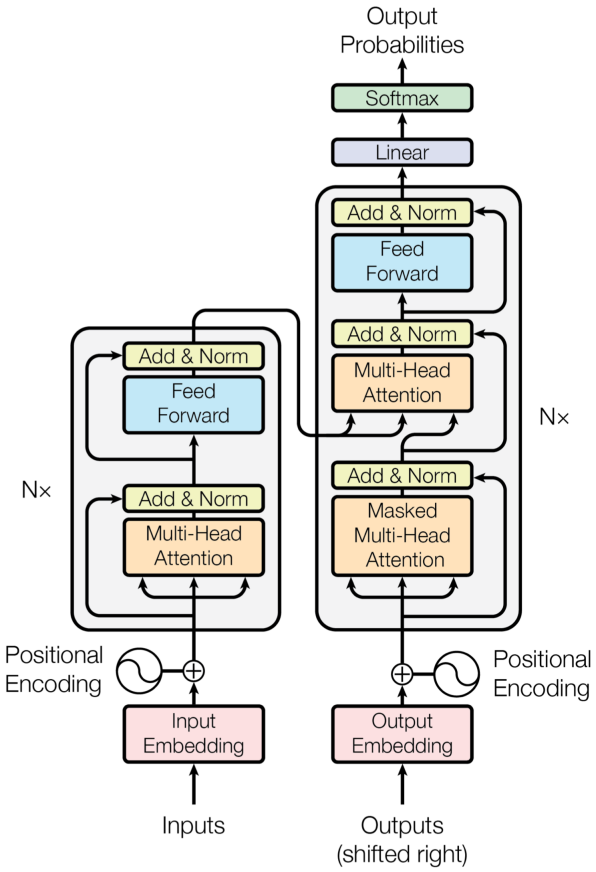
Prístup k spracovaniu štruktúrovaných dát takýmto spôsobom umožňuje na vstupe modelu vložiť celý dokument a na výstupe možno očakávať jeho vektorovú reprezentáciu, ktorá bude odrážať syntaktický či sémantický charakter dokumentu.

3.5 GPT2

Cieľom výskumu NLU (Natural Language Understanding) spoločnosti OpenAI je vytvorenie univerzálneho modelu, ktorý by bolo možné aplikovať na ľubovolnú NLU úlohu bez nutnosti nového tréningu. Na dosiahnutie tohto cieľa vytvorili model GPT2, ktorý bol natrénovaný na úlohe predpovedania ďalšieho slova v texte. Obsahuje 1.5 B parametrov a trénovací dataset obsahoval 8 miliónov webových stránok zozbieraných prehľadávaním kvalifikovaných odkazov od spoločnosti Reddit.[16]

3.5.1 Architektúra GPT2

Architektúra GPT-2 je variáciou slávnej architektúry *Transformer* navrhnutej tímom *Google Brain* vo svojom publikácií „Attention Is All You Need“ [17] . Architektúra *Transformer* (obrázok 3.5) vo svojom jadre poskytuje



Obr. 3.5: Transformer - architektura modelu [17]

všeobecný mechanizmus založený na topológií koder-dekodér. V modeli Transformer kódovač mapuje vstupnú sekvenciu symbolových reprezentácií $x_1, ..., x_n$ na sled súvislých reprezentácií $z = (z_1, ..., z_n)$. Vzhľadom na z, dekodér potom generuje výstupnú sekvenciu $(y_1, ..., y_m)$ symbolov po jednom prvku. V každom kroku je model automaticky regresívny, pričom spotrebuje predtým generované symboly ako dodatočný vstup pri generovaní ďalšieho.[16]

Tím OpenAI, vytvoril variáciu optimalizovanú pre multitaskové učenie NLU. Architektúra GPT-2 rozširuje základný Transformer model o možnosť vkladania optimalizácií pre špecifické úlohy a navyše optimalizuje prenos znalostí medzi vrstvami, ktoré sa stávajú robustnejšími v celom spektre úloh NLU.

3.5.2 Zero-Shot Transfer

Úlohou GPT-2 je výlučne jazykové modelovanie. Neexistuje žiadne špecifické dotrénovanie pre danú úlohu. Príklady formulácie úloh vyzerajú nasledovne: [16]

- **Generovanie textu** - je základná funkčnosť tohto modelu
- **Úloha strojového prekladu** - napríklad angličtina na čínštinu, je vyvolaná vložením na vstup siete páry „anglickej vety = čínska veta“, ... , „cieľová anglická veta =“ na konci.
- **Úloha QA** - je formátovaná podobne ako preklad s pármi otázok a odpovedí v kontexte.

3.5.3 Byte Pair Encoding (BPE)

GPT-2 používa BPE na UTF-8 bajtových sekvenciách. Každý bajt môže reprezentovať 256 rôznych hodnôt v 8 bitoch, zatiaľ čo UTF-8 môže použiť až 4 bajty pre jeden znak, pričom celkovo podporuje až 231 znakov. Preto pri reprezentácií bajtových sekvencií potrebujeme iba slovník s veľkosťou 256 a nemusíme sa starať o predspracovanie, tokenizáciu atď.

BPE zlučuje často existujúce bajtové páry viď. kapitola 4.5.2. Aby sa zabránilo vytváraniu viacerých verzií bežných slov (tj psa, psa! A psa? Pre slovo psa), GPT-2 bráni BPE v zlučovaní znakov naprieč kategóriami (teda pes by nebol zlúčený s interpunkciami ako.,! ?). Tieto triky pomáhajú zvyšovať kvalitu konečnej segmentácie bajtov.

Kapitola 4

Výber a predspracovanie dát

Tato kapitola bude zameraná na opis získania a predspracovania dát, ktoré boli následne použité pri experimentoch s rôznymi architektúrami sietí.

4.1 Požiadavky na dáta

Pre splnenie cieľa práce bolo potrebné nájsť kvalitný zdroj dát. Pri hľadaní správneho datasetu bol braný ohľad na pár základných vlastností, ktoré vyplývajú z definície problému:

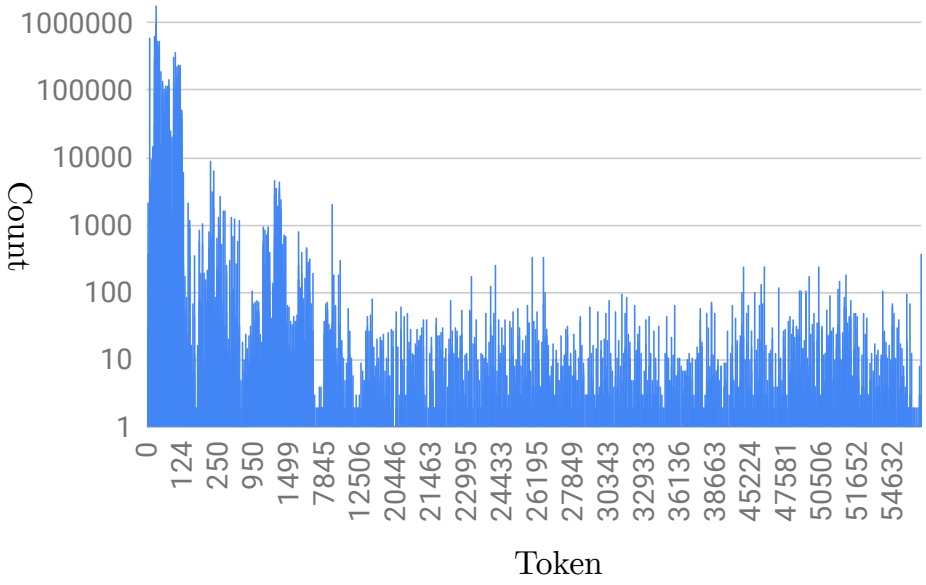
- **Data organizované do databázového systému**. Z dôvodu maximálneho priblíženia testovacieho prostredia tomu produkčnému, je táto požiadavka veľmi dôležitá. Organizácia dát do takejto štruktúry so sebou prináša rôzne problémy ako auto-inkrementálne dátové typy obsahujúce primárne kľúče, s ktorými sa musí systém vysporiadať.

- **Dostatočná veľkosť dátového zdroja**. Dátová množina musí byť dostatočne veľká za účelom obsiahnutia čo najväčšieho počtu dátových domén.

- **Minimalizácia počtu dát, ktoré vznikli syntetickou generáciou**. Dáta vzniknuté takouto cestou sú príliš čisté, čo vo väčšine prípadov nezodpovedá reálnemu stavu dát v cieľovom systéme.

4.2 Popis vybraných dátových zdrojov

Proces hľadania dátových zdrojov, ktoré by spĺňovali definované požiadavky bola veľmi náročná. Väčšina nájdených datasetov bola úzko definovaná pre istú doménu alebo podmnožinu podobných domén dát. Nakoniec boli identifikované dva.



Obr. 4.1: Histogram výskytu písmen v Ataccama S3 datasete. Osa x zodpovedá jednotlivým číselným reprezentáciám znaku v unicode a osa y reprezentuje počet v logaritmickom merítku.

4.2.1 Ataccama S3 dataset

Tento dataset bol získaný v spolupráci s firmou Ataccama. Obsahuje 32464 stĺpcov z rôznych domén. Dáta v nich obsahujú znaky z celého unikódového spektra (viď. histogram 4.1). Žiaľ, súčasťou dohody o práci s týmto datasetom je jeho nezverejniteľnosť, keďže sa jedná o vlastníctvo firmy.

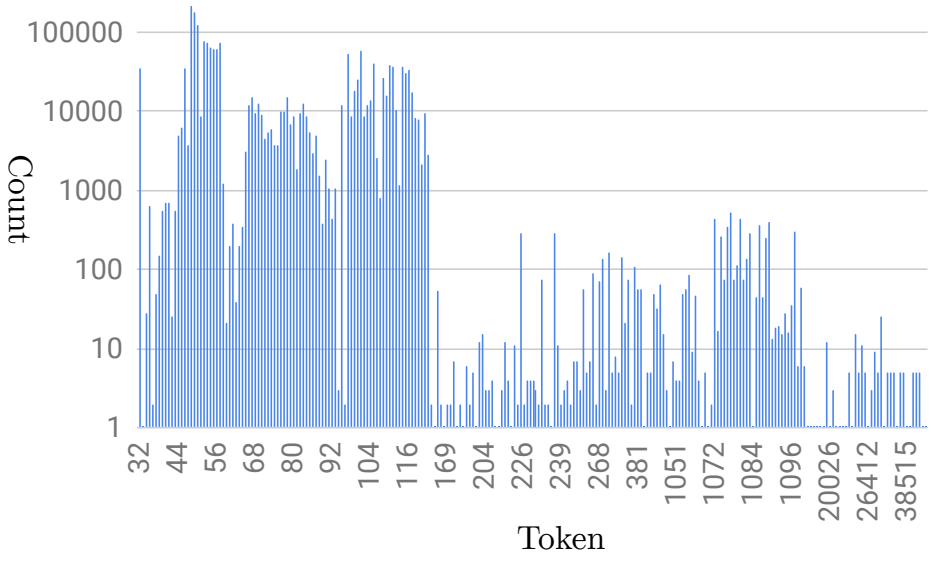
4.2.2 ČVUT dataset

Repozitár je vedený na verejnom MySQL serveri a spravuje ho ČVUT v Prahe. Každý súbor údajov je uložený ako MySQL databáza na serveri. Repozitár v súčasnosti obsahuje 73 databáz [18]. Na histograme 4.2 je vizualizované zastúpenie písmen v celom datasete. Na rozdiel od Ataccama S3 obsahuje predovšetkým dáta v prvej stovke spektra.

4.2.3 Výpočet podobnosti Ataccama S3 a ČVUT datasetov

Pre overenie nepodobnosti oboch dátových zdrojov bola vypočítaná Jaccard Similarity všetkých hodnôt, ktoré oba datasety obsahujú.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = 0.0319 \tag{4.1}$$



Obr. 4.2: Histogram výskytu písmen v ČVUT datasete. Osa x zodpovedá jednotlivým číselným reprezentáciám znaku v unicode a osa y reprezentuje počet v logaritmickom merítku.

pri čom platí, že

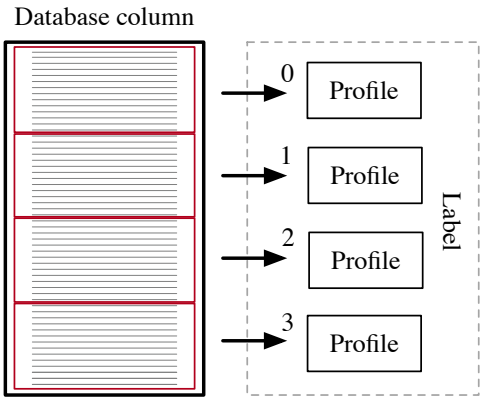
$$A \subseteq S3_{ataccama}, B \subseteq \check{C}VUT_{dataset}, 0 \leq J(A,B) \leq 1 \tag{4.2}$$

Na základe výsledku môžeme konštatovať, že 3% záznamov v datasetoch sú zhodné, čo je dostatočne malá zhoda na použitie.

4.3 Predspracovanie dát

Po dôslednom výbere dát, bolo nutné tieto dáta vhodne pred-spracovať, tak aby bolo možné zadefinovať optimalizačný ciel a následne zmerať ako dobre natrénovaný model plní svoju úlohu. Avšak ani jeden z datasetov neobsahuje Business Terms, ktoré by dáta rozdeľovali do dátových domén (mená, adresy, emaily, ...). Tieto informácie sú však nevyhnutne dôležité pri tréningu ale hlavne pri vyhodnotení presnosti natrénovaného modelu. Preto bolo nutné túto informáciu získať.

Prvá možnosť je ručne dátam priradiť doménu (Business Terms). Avšak to by znamenalo, že každý z vybraných datasetov, ktoré reprezentujú tisíce stĺpcov, by musel niekto ručne anotovať. Tento prístup sa ukázal ako neúnosný aj z pohľadu množstva aj z pohľadu samotného rozhodnutia o povahe dát v stĺpci.



Obr. 4.3: Proces tvorby profilu. Na obrázku je znázornené delenie stĺpca na 4 rôzne profily, pri ktorých poznačíme, že majú rovnaký pôvod.

Druhá možnosť, ako zaniesť informácie o doménach do dát sa opiera o základnú myšlienku vytvorenia n partícií ku každému stĺpcu, ku ktorým budeme následne pristupovať ako ku unikátnym a nezávislým stĺpcom. Z toho vyplýva, že počet stĺpcov v datasete sa n násobne zväčší. Pri vyhodnotení potom môžeme považovať za úlohu uspokojivo splnenú, keď vo vyhľadávaní najpodobnejších stĺpcov k danému stĺpcu sa objavia jeho partície. Problém tohoto prístupu však spočíva v tom, že ak sa partície vo výsledku neobjavia neznamená to nutne nesprávny výsledok. V datasete sa dáta naprieč stĺpcom môžu rôzne opakovať a preto nie je zaručené, že partície sú práve tie najpodobnejšie. Avšak takýto prístup nám poskytuje aspoň nejakú metriku, na základe ktorej je možné úspešnosť modelov porovnávať. Predstavuje spodnú hranicu riešenia problému.

4.4 Profiling

Profiling je proces, pri ktorom sa pre všetky partície stĺpcov vytvorí reprezentácia, **profile**, ktorý uchováva len redukovanú vzorku dát a ich jednoznačný identifikátor pôvodu (viď. obrázok 4.3). Identita dát je zložená z *názov databázy, názov tabuľky, názov stĺpca, číslo partície*. Na základe toho je možné dáta priradiť k sebe a tak zistiť, či majú rovnaký pôvod. Profil neuchováva všetky dáta nachádzajúce sa v partícií. Je uložená len vzorka dát z dôvodu zredukovania dátovej náročnosti. Bolo však potrebné zabezpečiť, aby redukčný proces zachoval všetky podstatné informácie o vlastnostiach zdrojových dát. Vhodnými prístupmi k tejto úlohe sú:

- n najfrekventovanejších hodnôt
- n najmenej frekventovaných hodnôt

- n najfrekventovanejších hodnôt ∪ n najmenej frekventovaných hodnôt
- Quantile [vybrané]

Z dôvodu najlepšieho zachovania distribúcie zdrojových dát sa v tejto práci zameriame na predspracovanie za pomoci quantilu. Ostatné metódy neboli predmetom skúmania z dôvodu nepravdepodobnosti pozitívneho efektu.

4.5 Metódy kódovania vstupných dát do NN

Na to aby sme mohli do neurónovej siete vložiť textové dáta, musia najprv prejsť procesom kódovania, pri ktorom sa každému slovu / písmenu priradí unikátna numerická reprezentácia. Toto priradenie sa uloží do slovníka, na základe ktorého následne môže prebehnúť proces dekódovania. Keďže vstupné dáta, s ktorými v tejto práci pracujeme sú prevažne technického charakteru, zameriame sa na metódy založené na kódovaní písmen. Avšak metódy musia byť schopné zachytiť celé unicode spektrum znakov, aby pri tomto procese nebola zanedbaná žiadna informácia a aby fungovala v každom jazyku.

4.5.1 Unicode kódovanie

Táto naivná metóda sa opiera o prevod každého znaku na jeho číselnú reprezentáciu. Je tak zaručené, že index obsahuje všetky znaky a navyše nemusí byť extra uchovávaný, keďže je obsiahnutý v samotnej reprezentácií daného písmena a prevod je možný na základe vstavanej funkcií operačného systému.

4.5.2 Byte pair encoding (BPE)

Jedná sa o jednoduchú techniku kompresie údajov, ktorá iteratívne nahradí najčastejší pár bajtov v poradí s jedným, nepoužívaným bitom. Avšak v kontexte textu namiesto spájania častých párov bajtov spájame znaky alebo sekvencie znakov. [19]

Prvým krokom algoritmu je inicializácia slovníka symbolov pomocou slovníka charakterov (písmená, znaky). Všetky páry symbolov sa spracovávajú postupne a každý výskyt najčastejšej dvojice („A“, „B“) nahradíme novým symbolom „AB“. Každá operácia zlúčenia vytvorí nový symbol, ktorý predstavuje znak n-gram. Časté n-gramy sa eventuálne spoja do jedného symbolu. Konečná veľkosť slovnej zásoby je rovnaká ako veľkosť počiatočného slovníka, plus počet operácií zlúčenia. Pre efektívnosť algoritmu nie sú uvažované páry, ktoré prekračujú hranice slov. Algoritmus tak môže byť spustený na slovníku extrahovanom z textu, pričom každé slovo je vážené jeho frekvenciou.[19]

Hlavný rozdiel oproti iným algoritmom kompresie, ako je napríklad Huffmanovo kódovanie, ktoré bolo navrhnuté na vytvorenie kódovania premenných dĺžok pre NMT [20] je, že tieto sekvencie symbolov sú stále interpretovateľné ako jednotky podslov (subword) a že sieť môže zovšeobecňovať preklad

a vytváranie nových slov (neviditeľných v čase tréningu) na základe týchto podjednotkových jednotiek. [19]

4.6 Metódy vektorizácie kódovaných dát

V predchádzajúcej časti boli opísané spôsoby kódovania textovej informácie do číselnej reprezentácie. V takejto forme by bolo možné dáta použiť na vstupe NN, avšak takto predspracované dáta by samé o sebe nepravdivo odrážali písmená / slová, ktoré zastupujú a to z dôvodu, že sú neporovnateľné na rozdiel od čísel. Inak povedané, slovami nemožno povedať, že písmeno $a \not> b$ avšak aj $a = 2$ a $b = 1$ tak môžeme tvrdiť, že $a > b$. Na odstránenie tohoto efektu používame vektorizačné metódy.

4.6.1 One-hot

Je metóda kódovania vstupných tokenov do vysoko dimenzionálneho binárneho vektora. Používa sa na reprezentáciu kategorického dátového typu dát. Veľkosť výslednej dimenzie vektora je rovná počtu kategórií, ktoré vektor kóduje. Keďže sa jedna o binárny vektor, obsahuje samé nuly a na indexe, ktorý reprezentuje danú kategóriu je jednotka. Veľká nevýhoda pri aplikácií na textové dáta je, že slovník býva veľký, čo ma negatívny dopad na dimenzionalitu výsledného vektora. [1]

4.6.2 Embedding

Ďalším populárnym a silným spôsobom transformácie tokenu (čísla) na vektor sa zameriava na použite hustých vektorov. Na rozdiel od One-hot, ktoré sú binárne, riedke a veľmi vysoko rozmerné, embedding vektory nadobúdajú pre každú dimenziu spojité hodnoty, čo umožňuje výraznú dimenzionálnu redukciu. Inak povedané, produkuje husté vektory. [1]

Existujú dva spôsoby použitia:

- Tréning embedding vrstvy spolu s hlavnou úlohou (napr. klasifikácia dokumentov alebo predikcia sentimentu). V počiatočnej fáze sa vektory náhodne na-inicializujú a postupným procesom tréningu sa vrstva naučí prideľovať vektory s ohľadom na optimalizačnú úlohu.

- Použitie embedding vrstvy, ktorá bola natrénovaná na inej úlohe. V takomto prípade váhy tejto vrstvy je nutné zafixovať, aby neboli menené počas tréningu cieľového problému. To umožňuje výrazné zrýchlenie procesu optimalizácie modelu.

V práci boli použité oba z týchto prístupov.

4.7 Aplikovanie na vstupné dáta

Pri príprave modelov, ktoré budú vytvárať vektorové reprezentácie stĺpcov treba zaistiť, aby všetky prevažne technické hodnoty, ktoré budú do modelu vstupovať mali vhodnú reprezentáciu naprieč celým unicodovým spektrom. Model musí predpokladať s najrôznejšími vstupnými hodnotami (azbuka, čínština, ...). Z týchto dôvodov pre všetky modely bude vektorizácia prebiehať na úrovni písmen. Iba tak je možné reprezentovať ľubovolné technické dáta v modeli bez zanedbania informácií.

4.7.1 Vstupná množina písmen

Pre dokonalú reprezentáciu všetkých možných písmen by bolo nutné obsiahnuť celé unicode spektrum, čo reprezentuje $17 * 2^{16} = 1114112 znakov$, kde 17 je počet rovín z čoho každá má 2^{16} možných znakov. To by viedlo k obrovskej dimenzionalite vstupných dát a vzniknutá sieť by bola veľmi náročná na tréning. Preto bola táto množina hodnôt obmedzená len na základnú viacjazyčnú rovinu (Basic Multilingual Plane BMP). Jedná sa o prvú rovinu. Obsahuje znaky pre takmer všetky moderné jazyky a veľký počet symbolov. Väčšina priradených kódových bodov v BMP sa používa na kódovanie čínskych, japonských a kórejských znakov. Jej celková veľkosť je $2^{16} = 65536$ *znakov*. [21]

4.8 Zhrnutie

V tejto kapitole sme si popísali základné požiadavky, podľa ktorých sme následne vybrali vhodné datasety. Oba vybrané datasety obsahujú veľké množstvo technických dát a disponujú rozmanitou škálou vstupných písmen. Podľa jednoduchého testu bolo overené, že tieto datasety disponujú rôznymi hodnotami a nie je medzi nimi korelácia. Následne bol opísaný proces profilingu, ktorý redukuje množinu dát pre každý stĺpec a proces rozdelenia slpca do partícií tak, aby informáciu o podobnosti jednotlivých profilov nebolo nutné anotovať ručne. Posledná časť kapitoly bola zameraná na riešenie problému ako priviesť textové dáta na vstup NN.

Všetky koncepty, ktoré boli v tejto kapitole opísané, boli následne aj využité pri konštrukcií modelov.

Kapitola 5

Metódy evaluácia a baseline problému

Táto kapitola práce bude zameraná na opis vyhodnotenia presnosti modelov, aby bolo možné jednotlivé modely vzájomne porovnávať. Popisuje, aké najväčšie problémy pri evaluácií bolo nutné riešiť a metódy ich riešenia. Druhá časť kapitoly je zameraná na otestovanie všeobecne známych metód pre riešenie tejto úlohy bez využitia hlbokého učenia neurónových sietí.

5.1 Evaluácia modelov

Aby sme mohli testované topológie modelov vzájomne porovnávať a vyhodnotiť, ktorá architektúra funguje najlepšie, musíme najprv jednoznačne definovať spôsob vyhodnotenia, ktorý bude čo najlepšie odrážať výsledné použitie na cieľových systémoch.

5.1.1 Data

Z dôvodu maximalizovania vierohodnosti vyhodnotenia, tréning a vyhodnotenie modelu používajú dáta, ktoré pochádzajú z dvoch rôznych systémov. Modely boli trénované na S3 datasete (viď. sekcia 4.2.1) a vyhodnotenie na ČVUT datasete (viď. sekcia 4.2.2).

5.1.2 Evaluačná funkcia

Vyhodnocovacia funkcia predpokladá na svojom vstupe vektorové reprezentácie pre každý profil z trénovacej množiny dát. Z týchto vektorov sa postaví index vzájomných vzdialenosti na základe *Euklidovej metriky*. Keď už disponujeme vyhľadávacím indexom nad profilami, môžeme nájsť pre každý profil n najpodobnejších susedov na základe vzdialenosti v hyper priestore vektorových reprezentácií profilov. V tomto bode evaluácia disponuje štruktúrou, v ktorej

každému profilu zodpovedá n iných najpodobnejších profilov. Tie však referujú len na m rôznych stĺpcov, keďže na základe každého stĺpca vzniklo až niekoľko profilov. Preto je nutné particiované profily spätne zgrupiť a vypočítať priemernú výslednú vzdialenosť a následne výsledky zoradiť od najmenšieho. V prípade, že zgrupovaný profil, ktorý referuje na pôvod profilu sa nachádza na jednom z prvých troch indexoch, je to považované za úspech a výsledok je zaznamenaný ako správny. V opačnom prípade je výsledok zaznačený ako nesprávny.

5.1.3 Aproximácia výsledkov

Takýto systém vyhodnotenia presnosti je na daných vstupných dátach len aproximatívny. V prípade, že sa label nachádza na indexe väčšom ako 3 je tento výsledok už uznaný ako nesprávny. Avšak z povahy dát nemusí nutne takéto chovanie znamenať chybu lebo nemáme garantované, že iné profily v testovacích dátach nepochádzajú z rovnakej domény (viď. sekcia 4.3). Keďže táto aproximácia nemá zásadný vplyv na kvalitu vzájomného porovnania modelov a bez labelovaného datasetu tento problém nie je možné rozumne riešiť, v práci sa bude toto zjednodušenie akceptovať a výsledok bude vnímaný ako spodný odhad miery funkčnosti.

5.2 Baseline problému

Na to, aby bolo možné vyhodnotiť prínos práce k aktuálnemu stavu riešenia, budú vyhodnotené najznámejšie prístupy k detekcii podobných domén na rovnakom testovacom datasete, ako aj všetky ostatné modely uvedené v ďalších častiach práce.

5.2.1 TF-IDF

TF-IDF je číselná štatistika, ktorá má odrážať, aké dôležité je slovo pre dokument v korpuse. Často sa používa ako váhový faktor pri vyhľadávaní informácií, ťažbe textov a modelovaní používateľov. Hodnota TF–IDF sa zvyšuje úmerne s počtom, koľkokrát sa slovo v dokumente nachádza a je vyvážené počtom dokumentov v korpuse, ktoré obsahujú slovo, čo pomáha prispôsobiť sa skutočnosti, že niektoré slová sa vo všeobecnosti vyskytujú častejšie. TF–IDF je dnes jednou z najpopulárnejších metód tvorby vektorovej reprezentácie dokumentu na základe váhovania výskytu slov. Až 83% textových doporučovacích systémov používa TF–IDF. [22]

V rámci práce bol aplikovaný tento prístup na testovaciu doménu dát a to tak, že z jednotlivých profilov bol vytvorený dokument spojením jednotlivých hodnôt quantilu za pomoci medzery. Výsledkom tejto operácie bol dokument obsahujúci hodnoty quantilu pre každý profil. Následne boli vytvorené 3-gramy s rešpektom na hranice slov. Výsledkom TF-IDF bola ma-

Tabuľka 5.1: Výsledky TF-IDF. Hodnoty stĺpcov predstavujú vzostupný výskyt správnych labelov na prvom, prvom až druhom a nakoniec aj na prvom až tretom indexe.

Name	**Index 1** [%]	**Index 1-2** [%]	**Index 1-3** [%]
TF-IDF	41,98	54,21	60,70

tica s veľkosťou 39104 x 44458, kde prvá dimenzia reprezentuje počet testovaných profilov a druhá celkový počet features (rôznych 3-gramov), ktoré nad testovacím datasetom vznikli. Jednotlivé riadky možno považovať za vektorové reprezentácie profilov a preto môžeme aplikovať na takto vzniknuté dáta evakuačnú metódu, ktorá bola navrhnutá v 5.1.2. Samozrejme bolo by efektívnejšie aplikovať na riedke vektory nejakú z aproximatívnych metód Nearest Neighbor Search a tým výrazne zvýšiť rýchlosť vyhľadávania, avšak bolo by to na úkor presnosti. Z toho dôvodu evaluačná metóda menená nebola. Výsledky vyhodnotenia sú prezentované v tabuľke 5.1.

Kapitola 6

Konštrukcia a vyhodnotenie výpočetných modelov

V tejto kapitole sa práca zameriava na jednotlivé experimenty s rôznym topológiami a ich aplikovanie na problém tvorby vektorovej reprezentácie stĺpca. Všetky experimenty boli pripravené v programovacom jazyku Python verzie 3.7. s využitím knižnice **Keras** s TensorFlow backend-om. Tréning prebiehal na grafickej karte **GeForce RTX 2080 Ti**.

6.1 Modely na úrovni hodnôt

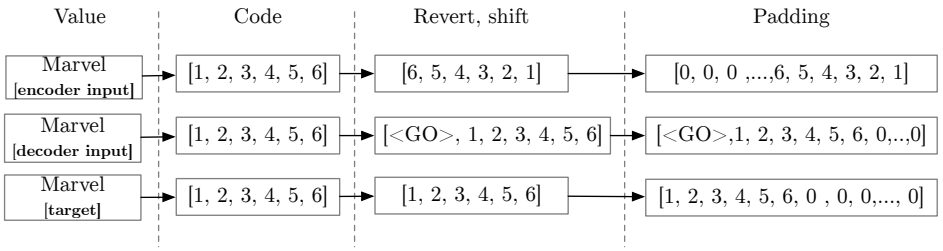
V tejto sekcií sa práca zameriava na modely, ktoré pracujú na úrovni hodnôt. To znamená, že model nemá o spracovávanom profile celkovú informáciu. Jednoducho každá hodnota je prevedená na jej vektorovú reprezentáciu, ktorá odráža jej syntaktický charakter. Takto získané hodnoty profilu sú následne priemerované a tým vzniká reprezentácia pre daný profil.

Všetky zostavené modely sa opierajú o koncept Seq2seq v zapojení autoenkoder opísaný v sekcií 3.1. Každý z prezentovaných modelov sa líši v type použitej rekurentnej siete alebo spôsobe predspracovania vstupných dát.

6.1.1 Zapojenie

Implementovaná sieť má svoje špecifické črty spôsobené najmä veľkou vstupnou abecedou 2^{16} (unicode). Embedding vrstva je súčasťou modelu, preto nie je možné pripraviť cieľové dáta v dobe pred začatím tréningu. Preto je výstupná hodnota siete zreťazená (concatenation) s výstupom *Embedding* vrstvy na cielovej hodnote. Obe tieto hodnoty sú následne privedené do *loss funkcie*. Inak povedané, vlastná implementácia *loss funkcie* získa y_{pred} aj y_{true} z argumentu y_{pred}, kde sú obe hodnoty zreťazené.

Všetky testované siete v rámci tejto topológie sú uvedené v tabuľke 6.1. Model je vždy zložený z Embedding vrstvy a jedného typu RNN. V tejto práci



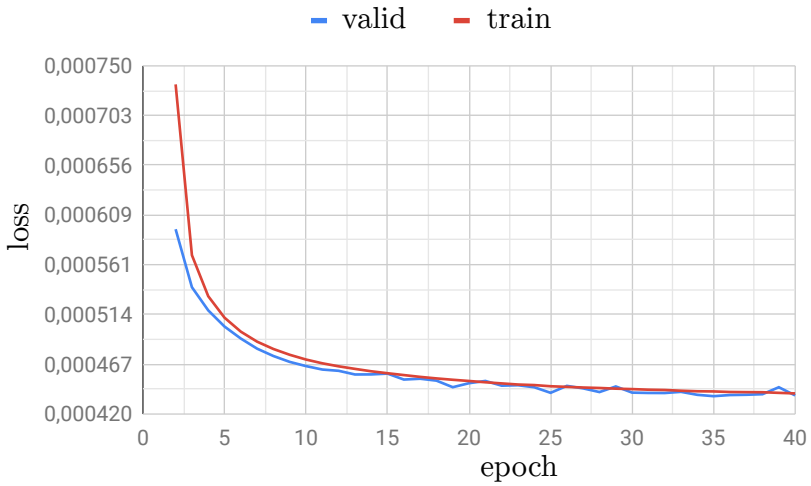
Obr. 6.1: Predspracovanie vstupných hodnot do seq2seq autoecoder. V prvom kroku sa pridelí každému písmenu unikátny kód, otočí sa poradie a vloží špeciálny token a nakoniec sa doplnia nuly do cieľovej dĺžky.

pracujeme vo všeobecnosti s troma variantmi embeddingu, všetky sú založené na Keras implementácií Embedding vrstvy opísanej v sekcií 4.6.2. Jednotlivé varianty sa líšia v metóde kódovania a teda aj vzniknutej vstupnej dimenzionalite. Hyper-parametre trénovaných sietí vychádzajú predovšetkým z predvolených nastavení jednotlivých komponentov. Veľkosť RNN naprieč všetkými modelmi je 128, dropout na úrovni 0.2 a ako optimalizátor bol použitý Adam. Názvy pre jednotlivé varianty vznikli pre účely referencie na konkrétny typ modelu a sú zložené zo začiatočných písmen embedding-u a typu RNN (napr. UL = Unicode embedding, LSTM RNN).

6.1.2 Predspracovanie vstupných dát

Keďže pracujeme so sieťou typu autoenkoder na hodnotovej úrovni, nepotrebujeme pri tréningu informácie o profiloch. Na tréning nám stačia hodnoty, ktoré budú reprezentovať čo najviac domén, ktoré sa môžu v cieľovom systéme vyskytnúť. Preto z každého profilu v trénovacej množine vyberieme kvantilové hodnoty a agregujeme ich do jedného pola, ktoré následne zbavíme duplikátov. Takto pripravené hodnoty predstavujú základ trénovacej množiny. Avšak ešte je nutné textové dáta previesť na vhodnú reprezentáciu pre NN. Na obr. 6.1 je detailná vizualizácia procesu predspracovania jednotlivých hodnôt.

Prvý krok je kódovanie. Tento proces bol opísaný v sekcií 4.5. V ďalšom kroku sa revertuje poradie tokenov v *input encoder*, z dôvodu uľahčenia prenosu informácie a posun vpravo *decoder inputu* z dôvodu oneskorenia. Posledným krokom je doplnenie tokenu $< pad >$ do fixnej dĺžky naprieč všetkým vstupom. V našom prípade je to 64. Je dôležité zdôrazniť, že zatiaľ čo pri *input encoder* sa dopĺňa token $< pad >$ spredu, u ostatných je to z opačnej strany. Dôvodom je minimalizácia dopadov efektu gradient vanishing (viď. sekcia 1.2.4) pri krátkych vstupoch.



Obr. 6.2: Priebeh tréningu modelu GCu. Train zodpovedá strate na trénovacej množine a valid zodpovedá strate na validačnej množine.

6.1.3 Priebeh tréningu

Modely boli natrénované na vzorke 508966 unikátnych hodnôt. Veľkosť **batch-u** bola nastavená na úrovni 64 a **validačná množina** bola 30% z trénovacích dát. Z dôvodu minimalizácie dopadu overfitingu je použitý **EarlyStop callback**, nastavený na ukončenie tréningu pri šiestich neúspešných zlepšeniach val-loss. Pre vizualizáciu priebehu tréning bol vybraný model **GCu**. Tréning ostatných modelov mal veľmi podobný priebeh a preto nebudú duplicitne prezentované. Z vývoju *validation loss* a *train loss* je zjavné, že model disponuje dobrou schopnosťou tréningu a generalizácie. Avšak je možné, že ak by bol zvýšený earlyStop parameter, model by bol schopný ešte pokračovať v tréningu.

Celkovo bolo trénovaných 8 sietí, ktoré sa líšia v Embedding vrstve a type RNN. V tabuľke 6.1 možno vidieť aký dopad mal typ zvolenej RNN na rýchlosť tréningu.

6.1.4 Vyhodnotenie

Evaluácia natrénovaných modelov sa striktne drží konceptu opísaného v sekcií 5.1. Avšak hodnotový enkodér nedokáže vytvoriť vektorovú reprezentáciu pre celý profil naraz, preto je nutné postupne vektorizovať hodnoty v quantile pre každý profil a následným spriemerovaním vytvoríme vektor odrážajúci vlastnosti celého profilu. Vektorový odtlačok profilu tak vznikne post procesne. V tabuľke 6.2 sú prezentované výsledky vyhodnotenia presnosti jednotlivých modelov.

Tabuľka 6.1: Trénované modely na hodnotovej úrovni. Stĺpec *Name* reprezentuje identifikátor siete pre účely tejto práce,*Embedding* definuje použitý typ embedding vrstvy, *Epoch* reprezentuje priemernú dĺžku epochy a *n* počet epoch do ukončenia tréningu.

Name	**Embedding**	**RNN typ**	**Epoch**	**n**
UL	$Embedding_{unicode}$	LSTM	1678s	47
BL	$Embedding_{bpe}$	LSTM	1698s	62
GL	$Embedding_{gpt2}$ [fix]	LSTM	1645s	40
UG	$Embedding_{unicode}$	GRU	1312s	47
BG	$Embedding_{bpe}$	GRU	1386s	60
GG	$Embedding_{gpt2}$ [fix]	GPT	1336s	39
UCu	$Embedding_{unicode}$	CuDNNGRU	101s	79
GCu	$Embedding_{gpt2}$ [fix]	CuDNNGRU	91s	45

Tabuľka 6.2: Výsledky modelov na hodnotovej úrovni. Hodnoty stĺpcov predstavujú vzostupný výskyt správnych labelov na prvom, prvom až druhom a nakoniec aj na prvom až tretom indexe.

Name	**Index 1** [%]	**Index 1-2** [%]	**Index 1-3** [%]
UL	40,74	55,08	63.73
BL	40,79	53,50	60.17
GL	42,66	56,36	63.33
UG	43,18	59,64	66.99
BG	43,34	57,50	64.65
GG	41,33	56,35	63.23
UCu	49,48	62,64	71.13
GCu	**50,16**	**63,5**	**71.95**

Z nameraných výsledkov vyplýva, že použitie CuDNNGRU ako typu RNN sa veľmi pozitívne odrazí na presnosti modelu. Navyše model, ktorý využíva túto vrstvu je 16x rýchlejší na tréningu oproti LSTM a 13x rýchlejší oproti GRU. Ďalšie zistenie z nameraných výsledkov sa týka spôsobu enkodingu. Využitie embedding vrstvy v spojení s BPE prinieslo zhoršenie oproti ostatným dvom metódam, ktoré dosiahli veľmi podobné výsledky na všetkých typoch RNN.

Hlavné zistenie z realizovaných experimentov sa týka CuDNNGRU, ktorá dokázala na daných hyperparametroch riešiť danú úlohu najpresnejšie a najefektívnejšie spomedzi všetkých ostatných testovaných variant modelov. Zdá sa, že Embedding vrstva v spojení s PBE neprináša pozitívny efekt. Je to spôsobené pravdepodobne špecifickými vstupnými dátami obsiahnutými v trénovacej množine, lebo natrénovaná embedding vrstva z modelu GPT2, ktorá rovnako používa BPE dosiahla lepšie presnosti na testovaných konfiguráciách

modelov.

6.2 Modely na úrovni profilov (Siamese)

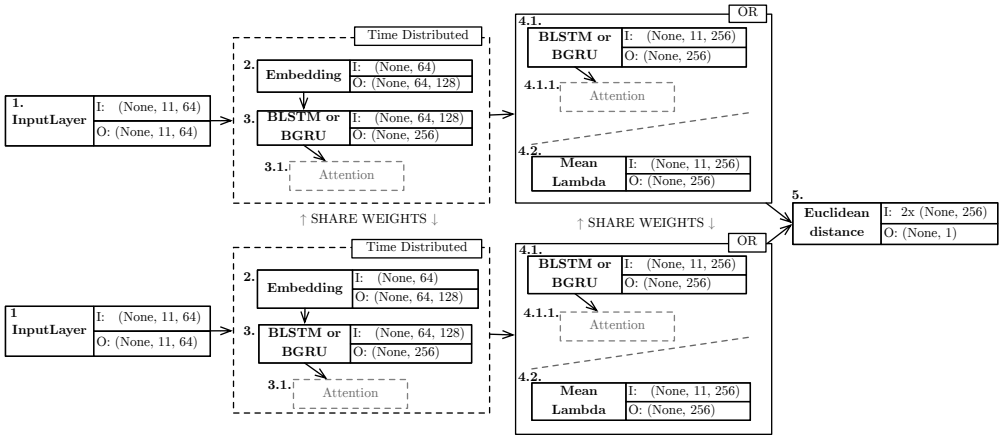
Modely, ktoré budú opísané v tejto časti, na rozdiel od modelov na úrovni hodnôt pracujú s profilom ako celkom. Aby bolo možné takýto model trénovať, musíme vhodne definovať optimalizačný cieľ. Keďže najdôležitejšia úloha celého modelu je vytvorenie vektorových reprezentácii tak, aby podobné dáta vytvárali zhluky, môžeme sa pokúsiť optimalizovať tréning modelu na základe pozitívnych a negatívnych párov profilov na vstupe. Budeme predpokladať, že ak obidva profily sú z rovnakej partície, ich vzájomná vzdialenosť vektorových reprezentácii by mala byť minimalizovaná, v prípade, že sú z rozdielnych partícií - by táto vzdialenosť naopak mala byť maximalizovaná.

6.2.1 Zapojenie

Zapojenie viacerých variant sietí je zobrazené na obrázku 6.3. Vstup do siete **InputLayer** (bunky 1 na obr. 6.3) predstavujú dva predspracované quantily vo formáte (11, 64). Keďže sa jedná o siamskú architektúru, všetky ďalšie vrstvy sú zapojené tak, aby obidva výpočetné prúdy pre obidva profily pracovali s vrstvami, ktoré majú vzájomne zdielané váhy. To znamená, že oba výpočetné prúdy optimalizujú rovnakú sadu parametrov. V prvom kroku výpočetnej časti modelu je nutné vektorizovať tokenizované dáta. K tomu modely využívajú **Embedding** vrstvu (bunka 2 na obr. 6.3), ktorá je za pomoci *TimeDistributed*[1] aplikovaná pre každú z 11-tich hodnôt profilu, čoho výsledkom sú vektorové reprezentácie každej hodnoty quantilu profilu. Takto pripravené dáta sú privedené do RNN, ktorú predstavujú **Bidirectional LSTM** alebo **Bidirectional GRU** (bunky 3 na obr. 6.3). Následne možno zapojiť za RNN ešte **Attention vrstvu** (bunka 3.1 na obr. 6.3) avšak nie je to nutné. Výsledkom je 11 vektorových reprezentácii pre každú hodnotu quantilu. V ďalšej vrstve modelu existujú dva základné prístupy ako vytvoriť z quantilových vektorov vektor profilový. Jedna z možností je pripojenie ďalšej úrovne RNN (bunka 4.1 na obr. 6.3) alebo využitie jednoduchej metódy priemerovacej vrstvy **Mean lambda** (bunky 4.2 na obr. 6.3). Posledným krokom celého výpočetného grafu je vrstva, ktorá vypočíta vzdialenosť oboch vzniknutých vektorových reprezentácii **Euclidean distance** (bunka 5 na obr. 6.3). Na základe jej výstupu potom *Contrastive loss* dokáže vhodne ohodnotiť výsledok.

Konfigurácia parametrov sietí sa predovšetkým opiera o predvolené hodnoty. Veľkosť použitých RNN je 128, dropout na úrovi 0.2 a ako optimalizátor bol použitý Adam. Testované modely boli inšpirované topológiu HAT (viď. sekciu 3.4). Jednotlivé varianty modelov, ktoré boli testované sú prezentované

[1] Funkcia knižnice keras - `https://keras.io/layers/wrappers/`



Obr. 6.3: Výpočetný graf Siamese modelov. Diagram zobrazuje varianty modelov: *Siamese ULL, Siamese ULLWA, Siamese UCC, Siamese UCCWA, Siamese ULM, Siamese UCM*

v tabuľke 6.3, kde v rámci práce boli vytvorené názvy jednotlivých sieti pre jednoduchšiu identifikáciu konkrétnej konfigurácie modelu.

6.2.2 Predspracovanie dát

Na tréning modelu bol využitý **generátor**, ktorý vytvára pozitívne a negatívne páry profilov. Avšak pri tomto procese je nutné zaistiť aby sa generované páry neopakovali. Zároveň je nutné sa vysporiadať s tým, že nemáme garantované, že dáta z rôznych partícii naozaj predstavujú rôzne domény, čo vyplýva zo spôsobu predspracovanie dát opísané v časti 4.3.

Algoritmus generátora pozostáva z týchto krokov:

1. Vyber dva rôzne stĺpce dát a skontroluj, či obsahujú aspoň dve partície. Ak nie, opakuje sa krok 1.
2. Z jedného z vybraných stĺpcov náhodne vyber dve jeho partície a označ ich ako pozitívny pár.
3. Z každého z vybraných stĺpcov náhodne vyber jednu partíciu a dvojicu označ ako negatívny pár.
4. Kontroluj, či negatívne páry majú rovnaký dátový typ, ak nie, začni znovu krokom 1.
5. Vypočítaj *Jaccard similarity* medzi negatívnymi pármi, ak je väčší ako 0.2, začni znovu krokom 1 (z dôvodu zamietnutia podobných dát v role negatívneho príkladu).
6. Pre každú patríciu vytvor hash z jeho quantilu.

7. Skontroluj, či pár s rovnakým hash-om už nebol vygenerovaný, ak áno, začni znovu krokom 1.

8. Ulož dvojicu hash-ov pozitívnych aj negatívnych particii do zoznamu videných.

9. Na všetky vybrané partície zavolaj *funkciu predspracovania*.

10. Prvé časti z oboch párov vlož do zoznamu *left* a druhé časti párov vlož do zoznamu *right* a do zoznamu *label* vlož hodnoty [1, 0] v správnom poradí, kde 1 je reprezentovaná pozitívnym párom a 0 negatívnym párom.

11. Opakuj pokiaľ veľkosť zoznamov *left*, *right* a *label* nebudú mať veľkosť rovnú alebo väčšiu ako **batch size**

12. Vráť zoznamy *left*, *right* a *label*

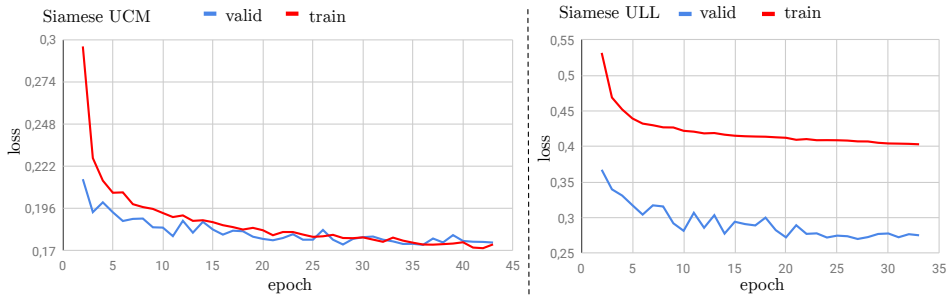
Ako môžeme v bode 9 vidieť, algoritmus používa predspracovaciu funkciu, ktorá má za úlohu previesť quantile profilu (partície) do tvaru vhodného pre NN. Toto predspracovanie pozostáva z podobných krokov ako bolo zobrazené na obrázku 6.1 s rozdielom, že sa vytvára verzia len pre *encoder input*.

6.2.3 Priebeh tréningu

Tréning modelov prebiehal za pomoci *fit generator*, ktorý bol opísaný v časti 6.2.2. Hodnota *batch size* bola nastavená na 128 a validačná množina predstavovala 30% trénovacích dát, ktorých rozdelenie bolo vykonané na úrovni tabuliek, aby sa zachovali jednotlivé partície a maximalizovala dátová unikátnosť oboch množín. Ďalej bol využitý EarlyStop callback, ktorý bol nastavený na ukončenie tréningu pri šiestich neúspešných pokusoch o zlepšenie loss na validačnej množine.

Priebeh tréningu modelov Siamese UCM a Siamese ULL je vizualizovaný na grafe 6.4. Na prvý pohľad si môžeme všimnúť, že v modeli Siamese ULL je validation loss menší ako training loss. Je to spôsobené dropoutom, ktorý LSTM sieť používa a v čase testovania na validačnej množine neaktivuje.

V tabuľke 6.3 je zoznam všetkých trénovaných sieti v rámci tejto topológie. Stĺpec **Name** označuje unikátny názov siete pre účely tejto práce, **Emb.** udáva použitú verziu embedding vrstvy. Nasledujúce dva stĺpce udávajú výpočetnú vrstvu na úrovni hodnôt a na úrovni quantilu. V prípade hodnoty *Mean* sa jedná o lambda vrstvu, ktorá agreguje vektory na základe priemeru. Stĺpec **Att.** konkretizuje, či boli použité Attention vrstvy alebo nie. Stĺpec **Epoch** disponuje hodnotami o priemernej dĺžke trvania jednej epochy a **n** zase udáva koľko bolo potrebných epoch na aktiváciu podmienky skorého ukončenia.



Obr. 6.4: Priebeh loss pri tréningu modelov Siamese UCM v ľavo a Siamese ULL v pravo.

Tabuľka 6.3: Trénované modely architektúry Siamese. Stĺpec *Name* reprezentuje identifikátor siete pre účely tejto práce,*Level 1 a 2* udáva typ použitej výpočetnej vrstvy, *Epooch* reprezentuje priemernú dĺžku epochy a *n* počet epoch do ukončenia tréningu.

Name	**Emb.**	**1. level**	**2. level**	**Att.**	**Epoch**	**n**
ULL	$Embedding_{unicode}$	LSTM	LSTM	N	1611s	32
ULLWA	$Embedding_{unicode}$	LSTM	LSTM	Y	1778s	34
ULM	$Embedding_{unicode}$	LSTM	Mean	N	1053s	40
UCC	$Embedding_{unicode}$	CuDNNGRU	CuDNNGRU	N	78s	20
UCCWA	$Embedding_{unicode}$	CuDNNGRU	CuDNNGRU	Y	104s	28
UCM	$Embedding_{unicode}$	CuDNNGRU	Mean	N	**72s**	**43**
GCC	$Embedding_{gpt2}$ [fix]	CuDNNGRU	CuDNNGRU	N	138s	18
GCM	$Embedding_{gpt2}$ [fix]	CuDNNGRU	Mean	N	134s	60

Tabuľka 6.4: Výsledky modelov Siamese architektúry. Hodnoty stĺpcov predstavujú vzostupný výskyt správnych labelov na prvom, druhom a nakoniec aj na tretom indexe.

Name	**Index 1[%]**	**Index 1-2[%]**	**Index 1-3[%]**
ULL	49,75	63,41	71.36
ULLWA	49,84	63,67	72.03
ULM	50,57	64,37	71,05
UCC	**50,77**	65,99	72.75
UCCWA	49,66	65,48	72.24
UCM	50,59	**66,20**	**72.93**
GCC	45,97	60,01	69,06
GCM	49,65	63,45	70,68

6.2.4 Vyhodnotenie

Výsledky jednotlivých modelov sú prezentované v tabuľke 6.4. Všetky varianty modelov dosiahli priemerne rovnaký výsledok. Možno konštatovať, že *Attention vrstva* v jednom prípade priniesla malé zlepšenie a v druhom zhoršila výsledok. Ďalej možno pozorovať, že použitie $Embedder_{gpt2}$ sa na výsledku odrazilo zhoršením oproti klasickému $Embedding_{unicode}$. Žiadna konfiguračná zmena v modeloch nepriniesla výrazné zlepšenie oproti ostatným. Celkovo výsledky potvrdzujú nestabilný tréning, ktorý by potreboval hyper-parametrickú optimalizáciu, ktorá by možno tréning stabilizovala. Napriek tomu je dôležité si uvedomiť, že aplikácia tejto architektúry umožnila použitie *contrastive loss*, ktorý dokáže optimalizovať vzdialenosť dvoch nepodobných entít vo vzniknutom multi-dimenzionálnom priestore na konkrétnu nastaviteľnú hodnotu. Tá môže byť následne použitá ako prahová hodnota pri odpovedaní na otázku, ktoré dva profily sú podobné a ktoré už nie sú.

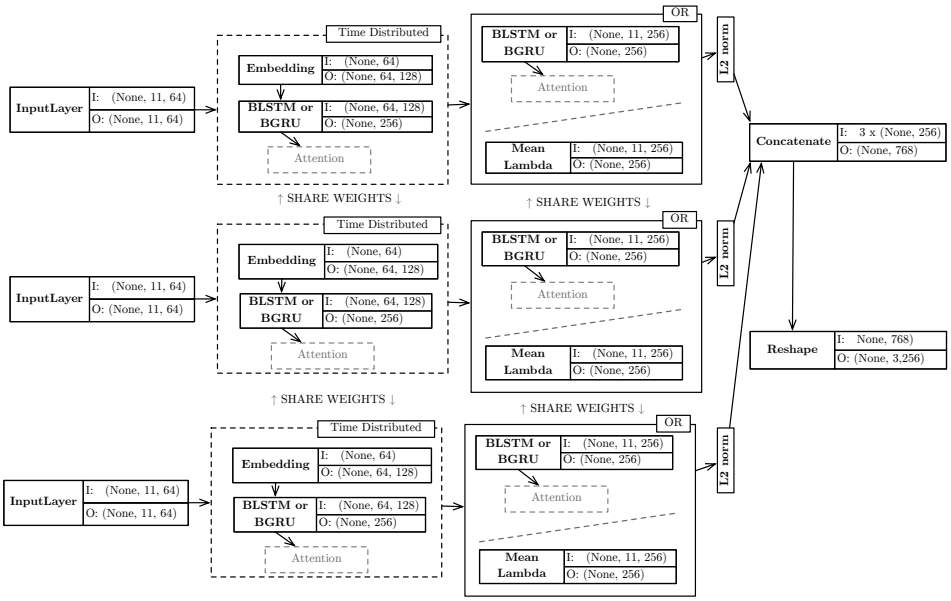
6.3 Modely na úrovni profilov (Triplet)

V tejto časti experimentov sa práca zameriava na topológiu Triplet opísanú v časti 3.3. Základným špecifikom architektúry je, že vstupné hodnoty sú trojice (anchor, positive, negative). Zoznam všetkých konfigurácií modelov je prezentovaný v tabuľke 6.5, kde okrem iného každému testovanému modelu je pridelený názov pre jednoduchú referenciu na konkrétny typ testovaného modelu.

6.3.1 Zapojenie

Zapojenie viacerých variant modelov je vizualizované vo výpočetnom grafe na obrázku 6.5. Princíp zapojenia jednotlivých vrstiev je identický so zapojením Siamese siete opísanej v časti 6.2.1, preto sa práca v tomto prípade zameriava len na opis hlavných rozdielov. Na prvý pohľad je evidentné, že Triplet topológia opiera svoj výpočet o trojicu výpočetných prúdov, ktoré majú vzájomne zdielané váhy na rozdiel od Siamese, ktorá pracuje len s dvojicou. Ďalšou zmenou je **l2 normalizácia**, ktorá ma za úlohu normalizovať vektory. Ak by normalizácia vektorov neprebehla, hodnotiaca funkcia *triplet loss*, by nebola schopná správne pracovať. Ďalšia zmena oproti Siamese spočíva v tom, že v tomto modeli sa nenachádza žiadna agregačná funkcia. Všetky tri vektory sú privedené do *triplet loss* (viz sekciu 3.3.1). Na to slúži **Concatenate vrstva**, ktorá vektory zlúči do jedného a **Reshape**. Posledný krok výpočetného modelu nie je nevyhnutný a slúži len na jednoduchú prácu s výsledným vektorom.

Hyperparametre siete ostali nezmenené oproti parametrom opísaným v sekcií 6.2.1.



Obr. 6.5: Zapojenie Siamese modelov. Diagram zobrazuje varianty modelov: *Triplet ULL, Triplet ULMWA, Triplet ULM, Triplet UCC, Triplet UCCWA, Triplet UCM*

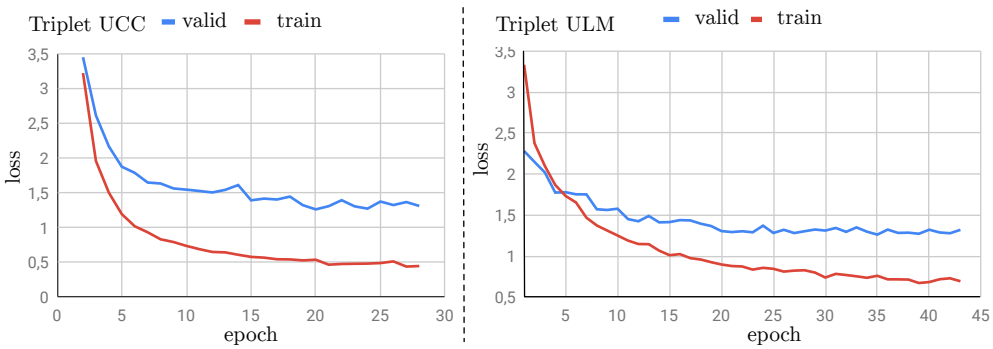
6.3.2 Predspracovanie dát

Príprava trénovacích trojíc je realizovaná rovnako ako v predchádzajúcom prípade cez **fit generátor**. Algoritmus generátora je prakticky identický ako v prípade opísanom v 6.2.2 s rozdielom, že sa generujú trojice profilov bez množiny label, lebo každá vzorka dát má svoj negatívny a aj pozitívny príklad.

6.3.3 Priebeh tréningu

Parametre tréningu ostali nezmenené oproti tréningu modelov v architektúre Siamese (viď. sekciu 6.2.3) Hodnota *batch size* bola rovná 128 a validačná množina bola zostrojená z rozdelenia trénovacej množiny na úrovni tabuliek. Priebeh tréningu modelov Triplet UCC a Triplet ULM je vizualizovaný na grafe 6.6 z ktorého vyplýva, že oba modely možno trpia overfiting-om a majú potenciál na zlepšenie uskutočnením hyper-parametrickej optimalizácie, ktorá však nebude predmetom tejto práce.

Zoznam trénovaných modelov a konkrétne použitie typov vrstiev na jednotlivých úrovniach je opísaný v tabuľke 6.5. Jedná sa o rovnakú tabuľku ako bola prezentovaná v experimentoch Siamese. Posledné dva stĺpce tabuľky prezentujú rýchlosť tréningu jednotlivých modelov. V porovnaní s tréningom rovnako konfigurovaných sietí v tabuľke 6.2.3 môžeme zhodnotiť, že Triplet architektúra potrebuje priemerne väčší čas na výpočet epochy a konverguje



Obr. 6.6: Priebeh loss pri tréningu modelov Triplet UCC naľavo a Triplet ULM napravo.

Tabuľka 6.5: Trénované modely architektúry Triplet, Stĺpec *Name* reprezentuje identifikátor siete pre účely tejto práce, *Level 1 a 2* udáva typ použitej výpočetnej vrstvy, *Epooch* reprezentuje priemernú dĺžku epochy a *n* počet epoch do ukončenia tréningu.

Name	**Emb.**	**1. level**	**2. level**	**Att.**	**Epoch**	**n**
ULL	$Embedding_{unicode}$	LSTM	LSTM	N	1717s	40
ULMWA	$Embedding_{unicode}$	LSTM	LSTM	Y	1645s	27
ULM	$Embedding_{unicode}$	LSTM	Mean	N	1554s	43
UCC	$Embedding_{unicode}$	CuDNNGRU	CuDNNGRU	N	101s	27
UCCWA	$Embedding_{unicode}$	CuDNNGRU	CuDNNGRU	Y	150s	37
UCM	$Embedding_{unicode}$	CuDNNGRU	Mean	N	101s	23
GCC	$Embedding_{gpt2}$	CuDNNGRU	CuDNNGRU	N	185s	33
GCM	$Embedding_{gpt2}$	CuDNNGRU	Mean	N	179s	23

o niečo pomalšie.

6.3.4 Vyhodnotenie

Výsledky konfigurácií trénovaných v topológií Triple sú prezentované v tabuľke 6.6. Keď výsledky porovnáme s rovnakými konfiguráciami optimalizovanými topológiou Siamese (tabuľka 6.4) vidíme, že na rovnakých hyper-parametroch triplet výrazne napomohol k zlepšeniu kvality tréningu a tým zlepšil presnosť všetkých testovaných konfigurácií modelov. Pravdepodobne je to spôsobené lepšie definovaným optimalizačným cieľom s ktorým triplet pracuje. Na prezentovaných výsledkoch ďalej môžeme pozorovať, že modely ktoré majú na pozícií *2. level* RNN miesto Mean, majú priemerne lepšie výsledky. Je to pravdepodobne spôsobené tým, že RNN na úrovni quantilu je schopná sa naučiť pracovať s distribúciou jednotlivých hodnôt v quantile, zatiaľ čo priemerova-

Tabuľka 6.6: Výsledky modelov Triplet architektúry. Hodnoty stĺpcov predstavujú vzostupný výskyt labelov na prvom, druhom a nakoniec aj na treťom indexe.

Name	Index 1 [%]	Index 1-2 [%]	Index 1-3 [%]
ULL	51,51	**67,31**	74,45
ULWA	50,02	66,54	74,10
ULM	50,22	65,33	72,41
UCC	**52,65**	66,45	**75,21**
UCCWA	51,94	67,25	74,11
UCM	47,87	64,07	71,53
GCC	52,18	67,06	73,79
GCM	48,27	62,23	70,88

cia vrstva sa neučí a tým sú jej možnosti zohľadnenia tejto distribúcie značne obmedzené.

6.4 Modely s využitím prístupu skladania

V tejto časti sa práca zameria na experimentálne spojenie predtrénovaného hodnotového enkodéra opísaného v 6.1 zapojeného do RNN na quantilovej úrovni. Motivácia takéhoto typu experimentu spočíva v redukovaní zložitosti tréningu tým, že ho rozložíme na dva samostatné behy:

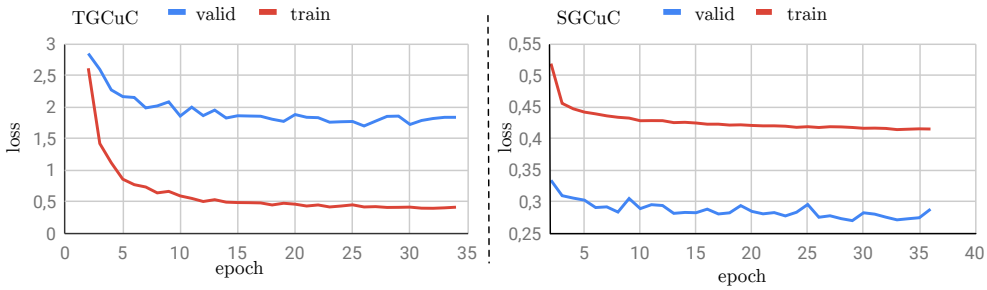
1. Tréning enkodéra na hodnotovej úrovni 6.1.
2. Zafixovanie natrénovaného hodnotového enkodéra, pripojenie pred enkodér na úrovni quantilov a následný spoločný tréning.

Spojený model možno následne efektívne trénovať za využitia Siamese alebo Triplet opísanej v 6.2 a 6.3. Počet trénovaných parametrov takéhoto zapojenia sa výrazne zmenší, keďže súčasťou hodnotového enkodéra je aj Embedding vrstva, ktorá výrazne zvyšuje počet trénovacích parametrov problému.

6.4.1 Tréning

Z predchádzajúcich výsledkov práce boli vybrané perspektívne konfigurácie. Prezentuje ich tabuľka 6.7. Obsahuje všetky základné informácie o zloženom modeli. Stĺpec **Arch.** definuje, na základe ktorej topológie bol výsledný model optimalizovaný a stĺpec **Value level** obsahuje názov jedného z enkodérov natrénovaných v sekcii 6.1, ktorý bude mať v výslednej sieti váhy nastavené na netrénovatelné. V posledných dvoch stĺpcoch tabuľky si opäť môžeme všimnúť výraznú úsporu času na tréningu.

Vývoj loss pri tréningu modelov **TGCuC** a **SGCuC** je zobrazený v grafe 6.7. Oba modely pravdepodobne trpia silným overfitingom a bolo by nutné



Obr. 6.7: Priebeh loss pri tréningu skladaných modelov TGCuC naľavo a SGCuC napravo.

Tabuľka 6.7: Trénované modely s využitím skladania. Stĺpec *Name* reprezentuje identifikátor siete pre účely tejto práce, *Arch.* udáva topológiu v ktorej bol model trénovaný, *Quantile leve* definuje využitý value enkoder z časti 6.1, *Att.* definuje, či boli využité Attention vrstvy, *Epooch* reprezentuje priemernú dĺžku epochy a n počet epoch do ukončenia tréningu.

Name	Arch.	Value level	Quantile level	Att.	Epoch	n
TGCuC	Triplet	GCu_{fix}	CuDNNGRU	No	85	35
TGCuCWA	Triplet	GCu_{fix}	CuDNNGRU	Yes	85	40
TUGC	Triplet	UG_{fix}	CuDNNGRU	No	42	13
TUGL	Triplet	UG_{fix}	LSTM	No	269	11
SGCuC	Siamese	GCu_{fix}	CuDNNGRU	No	57	36
SUGC	Siamese	UG_{fix}	CuDNNGRU	No	30	13

vykonať hyper-parametrickú optimalizáciu, ktorá má potenciál výrazne zlepšiť výslednú presnosť. Pri grafe tréningu modelu **SGCuC** si môžeme všimnúť nezvyčajne malý validation loss oproti tréning loss. Je to spôsobené dropoutom použitým v sieti. Ten sa pri invokácii na valid loss neaktivuje a tým vzniká tento nepríjemný efekt. Po odstránení dropout sa valid los dostal do normálu avšak model nedokázal byť natrénovaný do rovnakej presnosti.

6.4.2 Vyhodnotenie

Výsledky jednotlivých sietí postupne vyhodnotené na prvých troch indexoch sú prezentované v tabuľke 6.8, kde sa opäť potvrdil predpoklad, že triplet topológia na daných hyper-parametroch dosahuje stabilne lepšie výsledky na rovnakých konfiguráciach modelov. Obrazom toho je model **TGCuC** a **SGCuC**, ktoré sa líšia len vo zvolenej architektúre využitej pri tréningu. Tento prístup sa nejaví ako správny z dôvodu, že len v prípade modelov **TGCuC** a **TGCuCWA** bolo dosiahnuté zlepšenie oproti samotnému enkodéru. Dôvod môže spočívať v overfitingu modelov, avšak pri jednoduchých dodatočných

Tabuľka 6.8: Výsledky modelov s využitím skladania. Hodnoty stĺpcov predstavujú vzostupný výskyt label-ov na prvom, druhom a nakoniec aj na treťom indexe.

Name	Index 1 [%]	Index 1-2 [%]	Index 1-3 [%]
TGCuC	49,71	**66,73**	73,71
TGCuCWA	**51,2**	65,95	**73,82**
TUGC	39,11	51,42	57,34
TUGL	40,76	54,48	60,99
SGCuC	48,52	63,96	70,65
SUGC	40,99	55,06	61,68

experimentoch nebolo možné tieto dopady zmierniť bežnými metódami ako zväčšenie kapacity či zníženie dropout úrovne.

6.5 Predtrénované modely (GPT2)

V tejto časti sa práca zameriava na experimenty s predtrénovanou sieťou GPT2, ktorá bola opísaná v 3.5. Jedná sa o model, ktorý sa preslávil hodnovernou generáciou syntetického textu. Zároveň ale dokázala zlepšiť doposiaľ najlepšie riešenia naprieč širokým spektrom NLP úloh. Z toho dôvodu bola vybraná aj pre účely vyhodnotenia funkčnosti na riešenej úlohe podobností. Spomedzi všetkých podobných modelov (BERT, ELMo, Transformer) bola vybraná z dôvodu využitia Byte Pair Encoding (BPE). Vďaka tomu je táto sieť schopná vyjadriť ľubovolnú sekvenciu písmen, čo je kľúčovou vlastnosťou vo svete technických dát. V predchádzajúcich častiach práce sme experimentovali s extrahovanou sadou predtrénovaných parametrov Embedding vrstvy tohto modelu a v tejto časti sa práca pokúsi vyhodnotiť funkčnosť tohto modelu ako celku.

6.5.1 Interpretácia úlohy a predspracovanie

Hlavným problémom použitia GPT2 bolo nájsť spôsob, ako sieti interpretovať úlohu. V práci bola zvolená naivná stratégia, ktorá nezahŕňa žiadny fine-tune tréning, podľa opisu definície úlohy prekladu v publikácií o tejto sieti [16]. Cieľová požiadavka, ktorú očakávame od použitia siete je vytvorenie vektorových reprezentácií pre každý profil a z toho dôvodu potrebujeme s celým profilom pracovať ako s dokumentom. Za týmto účelom každý quantile profilu prevedieme do string-ovej reprezentácie tak, že každú hodnotu oddelíme špeciálnym znakom separovania (je treba zaistiť, že sa nevyskytuje v dátach). Výsledkom bude jedna stringová reprezentácia, ktorá je nositeľom celej informácie o profile (jeho quantile). Predspracované dáta následne môžeme priviesť na vstup funkcie, ktorá na základe BPE kódovania zaenkoduje znaky

Tabuľka 6.9: Výsledky modelu GPT2. Hodnoty stĺpcov predstavujú vzostupný výskyt labelov na prvom, druhom a nakoniec aj na treťom indexe.

Name	**Index 1** [%]	**Index 1-2** [%]	**Index 1-3** [%]
GPT2	40,75	54,72	61,63

do ich číselných reprezentácií a privedie na vstup natrénovanej siete. Pipeline enkodovania bola prebraná z exportu siete, ktorá spoločnosť OpenAI vydala spolu s modelom. Výstupom modelu je vektor tvaru *(počet profilov, maximálna dĺžka stringového vstupu, 768)*. Keďže každý vstup do siete má inú veľkosť, pozíciu výsledného vektoru je nutné vypočítať nasledovne:

$$index = len(str\_input) - 1 \tag{6.1}$$

6.5.2 Vyhodnotenie

V tabuľke 6.9 sú prezentované výsledky. Napriek žiadnemu tréningu, ktorý by viac špecifikoval doménu či úlohu, model dosiahol porovnateľné výsledky ako väčšina RNN, ktoré boli predmetom experimentov v predchádzajúcich častiach. Podľa našich experimentov tento model dosiahol o 13% horšiu presnosť ako doteraz najlepší prezentovaný model, UCC Triplet.

6.6 Zhodnotenie výsledkov

Táto kapitola práce bola zameraná na opis riešenia a vyhodnotenie navrhnutých stratégií, ktoré postupovali od jednoduchých seq2seq zapojení až po najmodernejšiu topológiu v oblasti strojového spracovania jazyka GPT2. Bolo preukázané, že topológie seq2seq v zapojení autoenkodér je možné spoľahlivo použiť na riešenie tejto úlohy. S využitím CuDNNGRU je rýchla a spoľahlivá. Avšak jej optimalizačný proces nedovoľuje vplývať na rozloženie vektorov v priestore a preto je skoro nemožné nájsť prahovú hodnotu, na základe ktorej by bolo možné rozhodnúť, či sú dva profily podobné alebo nie. Inak povedané, v indexe vytvorenom touto sieťou je možné nájsť len tie najpodobnejšie profily, avšak je len ťažko definovatelná hranica, kedy sú profily už vzájomne nepodobné. Z toho dôvodu bolo nutné preskúmať iné stratégie, ktoré by mali podobné chovanie a zároveň by vedeli vyriešiť tento zásadný problém.

Ďalšou testovanou topológiou bola Siamese. Sieť, ktorá pracuje s profilovými pármi, ktoré sa snaží optimalizovať pomocou *contrastive loss*. Výsledky tejto siete dosiahli veľmi podobný charakter ako najlepšie výsledky seq2seq topológie. Navyše táto topológia vyriešila problém s prahovou hodnotou, keďže parametrom contrastive loss, ktorou bola topológia optimalizovaná, obsahuje parameter *margin*, čo reprezentuje vzdialenosť, do ktorej budú od seba odtláčané vektory dvoch nepodobných profilov. Siamese však priniesla aj veľa pro-

blémov so samotným tréningom, ktorý javí známky možného overfitingu. Tento problém sa nepodarilo v rámci práce vyriešiť a preto pravdepodobne nebol dosiahnutý plný potenciál tohto zapojenia.

Ďalším prístupom, ktorý zdieľa podobné znaky s topológiou Siamese je architektúra Triplet. Vstupom tejto siete sú trojice (anchor, postive, negative) čo znamená, že každý vstup do siete obsahuje pozitívny a aj negatívny príklad. Celá sieť bola optimalizovaná na základe *triplet loss*, ktorá podobne ako v prípade contrastive loss disponuje margin parametrom. Triplet siete dosiahli na daných hyper-parametroch najlepšie výsledky a zlepšili presnosť oproti Siamese vo všetkých konfiguráciách. Avšak podobne ako topológia Siamese aj tu sa objavil problém s možným overfiting-om, preto výsledky oboch modelov nie sú úplne porovnateľné. Avšak na základe empirických skúseností s obidvoma topológiami v rôznych konfiguráciách sa zdá, že Triplet dokáže lepšie riešiť daný problém. Táto hypotéza bola navyše posilnená ručnou kontrolou výsledkov, kde v prípade Triplet bolo oveľa menej nezmyselných priradení. Na signifikantnejšie výsledky je žiaľ nutná hyperparametrická optimalizácia oboch topológií.

V snahe o ďalšie zlepšovanie presnosti výsledkov bola testovaná topológia, ktorá sa snaží využiť natrénované hodnotové enkodéry za pomoci seq2seq a zapojiť za nich ďalší level RNN a následne optimalizovať za pomoci topológie Siamese alebo Triplet. Motivácia takéhoto prístupu je zmenšenie priestoru parametrov pri tréningu a predpoklad lepšieho tréningu menšieho modelu. V tomto prípade sa však podarilo len v dvoch topológiach mierne zlepšiť presnosť modelu oproti samotnému seq2seq enkoderu, ktorý bol v nej použitý. Pravdepodobná príčina je znovu overfiting, napriek značnej snahe hyper-parametrickej optimalizácie.

Posledný testovaný model reprezentuje kategóriu modelov, ktoré sú určené pre široké spektrum NLP úloh a nepotrebujú tréning na cieľovom probléme aby dokázali uspokojivo daný problém riešiť. GPT2 bol vybraný z dôvodu BPE kódovania na svojom vstupe, vďaka čomu dokáže vyjadriť ľubovoľnú sekvenciu znakov. Výsledná presnosť aplikácie modelu na daný problém dosiahla úrovne najhoršieho riešenia modelu seq2seq, čo vzhľadom na náročný charakter vstupných dát je pomerne perspektívny výsledok no v kontraste ostatných prístupov, nepoužiteľný.

6.7 Budúca práca

Hlavná výzva pre budúcu prácu na tomto probléme spočíva v hyper-parametrickej optimalizácií jednotlivých modelov tak, aby boli uspokojivo natrénované. To môže ale nemusí zlepšiť výsledky modelov. Ďalší priestor pre prácu spočíva vo fine-tune modelu GPT2, ktorý ukázal, že princíp jeho fungovania má veľký potenciál a NLP nie je doménou len rekurentných sietí. Ďalšie zlepšenia môžu priniesť efektívnejšie reprezentácie vstupnej množiny symbolov, ktoré by ne-

museli byť také veľké ako boli použité v tejto práci. Malo by to za následok zníženie trénovateľných parametrov a možné rýchlejšie konvergovanie siete.

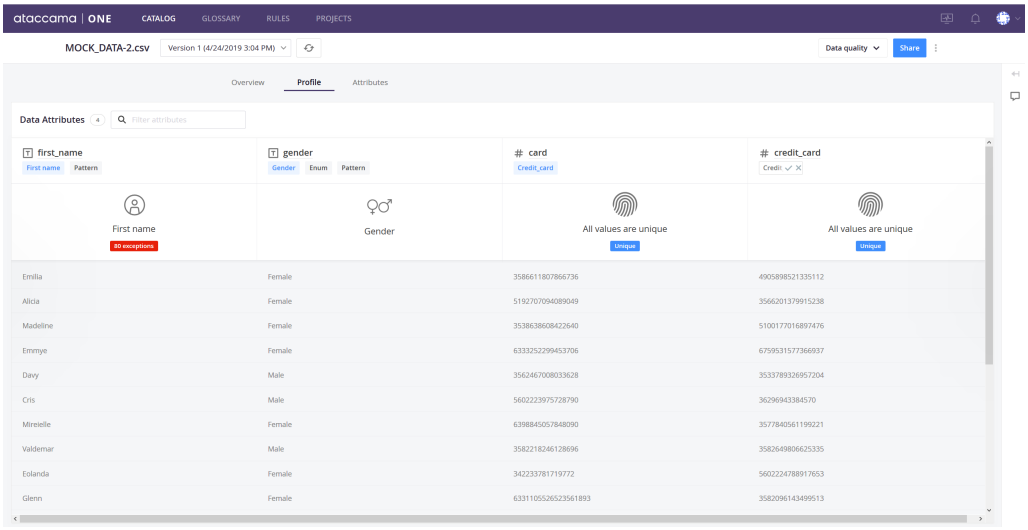
Záver

Táto práca bola zameraná na vytvorenie metódy pre porovnanie databázových stĺpcov na základe ich doménovej podobnosti. Ako najefektívnejší prístup k tomuto problému sa ukázalo vytvorenie vektorovej reprezentácie, ktorá obsahuje relevantné informácie. Mieru podobnosti v takomto vektorovom priestore potom možno ľahko zmerať za pomoci Euklidovej vzdialenosti. Veľká výhoda oproti ostatným prístupom je, že vektorová reprezentácia môže byť vytvorená len raz pri štarte systému a perzistentne uložená do metadát stĺpca. Rýchlosť vyhľadávania je tak limitovaná iba rýchlosťou implementácie Nearest Neighbor algoritmu, pre nájdenie najbližších susedov.

Pre dosiahnutie čo najlepších vektorových reprezentácií bolo otestovaných 5 rôznych prístupov: seq2seq, Siamese, Triplet, modely s využitím prístupu skladania, GPT2. V každom z nich boli otestované základné konfigurácie. Model, ktorý na zvolených parametroch fungoval najlepšie, bol natrénovaný na Triplet topológií a využíva dve za seba zapojené CuDNNGRU. Jeho výsledok prekonal doposiaľ najčastejšie používanú metódu na danú úlohu TF-IDF o **14,51%**. Ukážka výsledkov priamo na testovacích dátach je prezentovaná v prílohe A na obrázkoch A.1 a A.2. Tento výsledok potvrdil veľkú perspektívu využitia rekurentných neurónových sietí na danú úlohu. Je pravdepodobné, že presnosť modelu by bola ešte o niečo vyššia po dôkladnej hyperparametrickej optimalizácií, ktorá nebola súčasťou tejto práce z dôvodu obmedzených výpočetných prostriedkov. Pri testovaní s jednotlivými typmi rekurentných sietí bol odhalený výrazne pozitívny efekt pri použití CuDNNGRU namiesto klasickej GRU alebo LSTM. Modely s touto vrstvou boli 11-14x rýchlejšie pri tréningu aj invokácií a priemerne dosahovali aj lepšie výsledky.

Využitie práce

Výsledky tejto práce sa stali základom funkcionality doporučovania Business Terms, ktorá bola implementovaná v produktoch firmy *Ataccama Software*. Užívateľ môže definovať vlastný Business Term a priradiť ho ku existujúcemu



Obr. 6.8: Ukážka aplikácie Ataccama One. Na základe výsledkov tejto práce bola rozšírená funkčnosť tejto aplikácie o doporučovanie Business Terms.

stĺpcu. V prípade, že v systéme existujú podobné stĺpce, je k nim systém schopný následne doporučiť tento Business Term. Avšak potvrdenie alebo zamietnutie navrhovaného Termu je na samotnom užívateľovi. Najväčší prínos tejto funkcionality spočíva v minimalizácií užívateľskej interakcie pri výbere *Business Terms* z podnikového slovníku, pri katalogizácií nových dátových zdrojov. Užívateľovi vo väčšine prípadov stačí len potvrdenie návrhu. Ďalší pozitívny dopad na systém spočíva v objasňovaní povahy neznámych dát. Ak je nejaký Term doporučený, znamená to, že podobné dáta už v systéme existujú a užívateľ tak získava predstavu o charaktere neznámych dát.

Literatúra

[1] Chollet, F.: *Deep Learning with Python*. Manning Publications Co., 2017, ISBN 978-1-6172-9443-3.

[2] Patterson, J.; Gibson, A.: *Deep Learning: A Practitioner's Approach*. O'Reilly, 2017, ISBN 978-1-4919-1425-0. Dostupné z: https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/

[3] Hochreiter, S.; Schmidhuber, J.: Long Short-Term Memory. *Neural Computation*, ročník 9, 1997: s. 1735–1780.

[4] Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2018-04-18.

[5] Chung, J.; Çaglar Gülçehre; Cho, K.; aj.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, ročník abs/1412.3555, 2014.

[6] Cho, K.; van Merrienboer, B.; Çaglar Gülçehre; aj.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*, 2014.

[7] Sutskever, I.; Vinyals, O.; Le, Q. V.: Sequence to Sequence Learning with Neural Networks. In *NIPS*, 2014.

[8] Prabhavalkar, R.; Rao, K.; Sainath, T.; aj.: A Comparison of Sequence-to-Sequence Models for Speech Recognition. 2017. Dostupné z: http://www.isca-speech.org/archive/Interspeech_2017/pdfs/0233.PDF

[9] Venugopalan, S.; Rohrbach, M.; Donahue, J.; aj.: Sequence to Sequence - Video to Text. *CoRR*, ročník abs/1505.00487, 2015.

[10] Chung, Y.-A.; Wu, C.-C.; Shen, C.-H.; aj.: Audio Word2Vec: Unsupervised Learning of Audio Segment Representations using Sequence-to-sequence Autoencoder. In *INTERSPEECH*, 2016.

[11] Koch, G. R.: Siamese Neural Networks for One-Shot Image Recognition. 2015.

[12] Hadsell, R.; Chopra, S.; LeCun, Y.: Dimensionality Reduction by Learning an Invariant Mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, ročník 2, 2006: s. 1735–1742.

[13] Mueller, J.; Thyagarajan, A.: Siamese Recurrent Architectures for Learning Sentence Similarity. In *AAAI*, 2016.

[14] Schroff, F.; Kalenichenko, D.; Philbin, J.: FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015: s. 815–823.

[15] Yang, Z.; Yang, D.; Dyer, C.; aj.: Hierarchical Attention Networks for Document Classification. In *HLT-NAACL*, 2016.

[16] Radford, A.; Wu, J.; Child, R.; aj.: Language Models are Unsupervised Multitask Learners. 2019.

[17] Vaswani, A.; Shazeer, N.; Parmar, N.; aj.: Attention Is All You Need. In *NIPS*, 2017.

[18] Motl, J.; Schulte, O.: The CTU Prague Relational Learning Repository. *CoRR*, ročník abs/1511.03086, 2015, 1511.03086. Dostupné z: http://arxiv.org/abs/1511.03086

[19] Sennrich, R.; Haddow, B.; Birch, A.: Neural Machine Translation of Rare Words with Subword Units. *CoRR*, ročník abs/1508.07909, 2016.

[20] Chitnis, R.; DeNero, J.: Variable-Length Word Encodings for Neural Translation Models. In *EMNLP*, 2015.

[21] Cursive, M.: The Unicode Standard, Version 6.1. 2008.

[22] Beel, J.; Gipp, B.; Langer, S.; aj.: Research-paper recommender systems : a literature survey. *International Journal on Digital Libraries*, ročník 17, č. 4, 2016: s. 305–338, ISSN 1432-5012, doi:10.1007/s00799-015-0156-0.

Dodatok **A**

Ukážka výsledkov na dátach

Query UID: ('Accidents', 'nesreca', 'y', 4)
Query Quantiles: (32190, 71739, 91709, 101317, 105174, 117213, 123345, 136596, 151208, 157537, 190598)

Result Distance: 3.6708296373073055
Result UID: ('Accidents', 'nesreca', 'y', 1)
Result Quantiles: (32571, 71704, 91746, 101370, 105295, 117246, 123236, 136388, 151053, 157501, 191727)
Result Distance: 3.7574289842433406
Result UID: ('Accidents', 'nesreca', 'y', 2)
Result Quantiles: (32243, 71912, 91819, 101386, 105287, 117273, 123296, 136542, 151318, 157535, 191774)

Query UID: ('Accidents', 'nesreca', 'x_wgs84', 4)
Query Quantiles: (13.4201383435244, 14.0351061641884, 14.4085043016297, 14.5076989512322, 14.5814814049044, 14.9843141127715, 15.1914612317584, 15.4578509570473, 15.6485335217967, 15.8613237038254, 16.5267980673445)

Result Distance: 1.5541023236650573
Result UID: ('Accidents', 'nesreca', 'x_wgs84', 0)
Result Quantiles: (13.4191625067265, 14.0015685194445, 14.3836064233199, 14.5055697223914, 14.5744575852477, 14.9697133877581, 15.1887669134278, 15.449086876517, 15.6480765712332, 15.8586742528383, 16.5230477062534)
Result Distance: 1.6892986716406337
Result UID: ('Accidents', 'nesreca', 'x_wgs84', 1)
Result Quantiles: (13.4224444379385, 14.0280481073033, 14.4033306944944, 14.5075874735752, 14.5808191696905, 14.9834679884163, 15.1911294596606, 15.4659374072989, 15.6485335217967, 15.8593601193203, 16.5273252071308)

Obr. A.1: Ukážka hľadania podobností na dátach 1. V oboch prípadoch bolo hľadanie úspešné a medzi nepodobnejšími výsledkami sú particie hľadaného. `UID` reprezentuje identifikátor profilu vo formáte (`názov databázy, názov tabuľky, názov stĺpca, číslo partície`).

Query UID: ('Accidents', 'nesreca', 'tekst_cesta_ali_naselje', 1)
Query Quantiles: ('(RAZDRTO-) - VRTOJBA', 'DRAVOGRAD - ARJA VAS', 'KOPER - SEČOVLJE', 'LJUBLJANA', 'LJUBLJANA', 'MARIBOR', 'NEZNANO:00688', 'PTUJ', 'STRMEC - SOCKA - VITANJE', 'ČRNUČE - ZIDANI MOST', 'ŽVIRČE')

Result Distance: 2.85082594331207
Result UID: ('Accidents', 'nesreca', 'tekst_cesta_ali_naselje', 3)
Result Quantiles: ('(RAZDRTO-) - VRTOJBA', 'DRAVOGRAD - ARJA VAS', 'KOPER', 'LJUBLJANA', 'LJUBLJANA', 'MARIBOR', 'NEZNANO:00701', 'PTUJ', 'STRUNJAN', 'ČRNUČE - ZIDANI MOST', 'ŽVIRČE')
Result Distance: 3.0073777773500683
Result UID: ('Accidents', 'nesreca', 'tekst_cesta_ali_naselje', 2)
Result Quantiles: ('(RAZDRTO-) - VRTOJBA', 'DRAVOGRAD - ARJA VAS', 'KOPER', 'LJUBLJANA', 'LJUBLJANA', 'MARIBOR', 'NEZNANO:00696', 'PTUJ', 'STRMEC - SOCKA - VITANJE', 'ČRNOMELJ', 'ŽVIRČE')

Query UID: ('Accidents', 'upravna_enota', 'ime_upravna_enota', 3)
Query Quantiles: ('Gornja Radgona', 'Grosuplje', 'Hrastnik', 'Ljubljana Center', 'Logatec', 'Ravne na Koroškem', 'Sevnica', 'Sežana', 'Trbovlje', 'Zagorje ob Savi', 'Črnomelj')

Result Distance: 6.125604817770057
Result UID: ('Financial_ijs', 'districts', 'A2', 0)
Result Quantiles: ('Brno - mesto', 'Cesky Krumlov', 'Jesenik', 'Kutna Hora', 'Mlada Boleslav', 'Pardubice', 'Pelhrimov', 'Praha - zapad', 'Rychnov nad Kneznou', 'Tabor', 'Trebic')
Result Distance: 6.179335047234514
Result UID: ('financial', 'district', 'A2', 0)
Result Quantiles: ('Brno - venkov', 'Cesky Krumlov', 'Frydek - Mistek', 'Jesenik', 'Karvina', 'Pelhrimov', 'Prostejov', 'Semily', 'Trebic', 'Vsetin', 'Zdar nad Sazavou')

Query UID: ('AdventureWorks2014', 'AWBuildVersion', 'ModifiedDate', 0)
Query Quantiles: ('2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08', '2014-07-08')

Result Distance: 2.6359438444644328
Result UID: ('cs', 'ACCOUNTS', 'ACCH_CLOSE_DATE', 2)
Result Quantiles: ('2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01', '2014-07-01')
Result Distance: 2.7491770766781376
Result UID: ('Chess', 'game', 'event_date', 1)
Result Quantiles: ('2014-06-16', '2014-06-16', '2014-06-17', '2014-06-17', '2014-06-17', '2014-06-17', '2014-06-17', '2014-06-17', '2014-06-17', '2014-06-17', '2014-06-17')

Query UID: ('AdventureWorks2014', 'Address', 'AddressLine1', 4)
Query Quantiles: ('9900 2700 Production Way', '1874 Orchid Ct', '2725 Deerwood Court', '3788 Canyon Creek Drive', '4839 Belle Dr', '6055 Broadway Street', '6999 Yosemite Circle', '8042 Placer Dr.', '909, rue Saint Denis', 'Auf dem Ufer 764', 'Zur Lindung 78')

Result Distance: 5.29693068960766
Result UID: ('AdventureWorks2014', 'Address', 'AddressLine1', 2)
Result Quantiles: ('081, boulevard du Montparnasse', '198 Edie Ct.', '2894 Foothill Way', '3965 Stony Hill Circle', '4912 Roundhouse Place', '5898 Mt. Dell', '6948 Midway Ct', '8068 Villageoaks Dr.', '9052 Montgomery Avenue', 'Auf den Kuhlen Straße 7', 'Zur Lindung 764')
Result Distance: 5.720387072386671
Result UID: ('AdventureWorks2014', 'Address', 'AddressLine1', 0)
Result Quantiles: ('00, rue Saint-Lazare', '1933 Rock Creek Pl.', '2901 Sunny Ave', '388 Frayne Lane', '4832 Park Glen Ct.', '5919 Zartop Street', '692 Honey Trail Lane', '8 Sunnybrook Drive', '9001 Esperanza', 'Am Kreuz 409', 'Zur Lindung 787')

Obr. A.2: Ukážka hľadania podobností na dátach 2. V prvom príklade bolo hľadanie úspešné a v dvoj najbližších výsledkoch sa našli particie stĺpca. V druhom prípade particie neboli nájdené ale výsledok je napriek tomu správny. K mestám boli nájdené iné mestá. Podobná situácia nastala v dátumoch. V poslednom príklade bol vyhľadávanie na adresách znovu úspešné a boli nájdené partície. `UID` reprezentuje identifikátor profilu vo formáte `(názov databázy, názov tabuľky, názov stĺpca, číslo partície)`

Dodatok **B**

Zoznam použitých skratiek

LSTM Long short-term memory

GRU Gated recurrent units

TF-IDF Term frequency–inverse document frequency

GPT Generative Pre-training Transformer

NN Neural network

RNN Recurent Neural network

Seq2seq Sequence to sequence

HAT Hierarchical Attantion network

cuDNN NVIDIA CUDA Deep Neural Network

Dodatok C

Obsah priloženej SD karty

- `README.md` ..... popis ako spustiť experiment
- `requirements.txt` ..... zoznam potrebných baličkou
- `env.sh` ..... script pre nastavenie envirimentu
- `tasks` ..... súbor zo všetkými experimentami
  - `gpt2` ..... experimenty s topologiou gpt2
  - `seq2seq` ..... experimenty s topologiou seq2seq
  - `siamese` ..... experimenty s topologiou siamese
  - `triplet` ..... experimenty s topologiou triplet
  - `run_xxx.py` .. Programy určené pre štart experimentov kategórie xxx
- `outcome` ..... výstupy treńingu každého z modelov
  - `seq2seq` ..... výstupy modelov viď. tabulka 6.1
  - `siamese` ..... výstupy modelov viď. tabulka 6.3
  - `triplet` ..... výstupy modelov viď. tabulka 6.5
  - `other` ..... baseline (TF-IDF), pretrain (GPT2)
- `data` ..... data
  - `profiles` ..... pickle profilovaných dát
  - `models` ..... natrénovaný GPT2 model
- `custom_components` .. baličky s vlastnou implementáciou komponent NN
- `evaluation` ..... evaluačný baliček
- `preprocessor` ..... balíček obsahujúcu predspracovacie funkcie
- `latex` ..... zdrojový kód textu práce