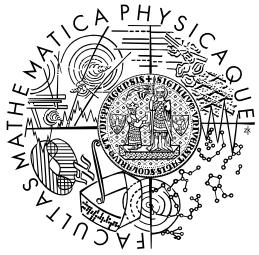


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

František Trebuňa

Generating text from structured data

Institute of Formal and Applied Linguistics (ÚFAL)

Supervisor of the bachelor thesis: Mgr. Rudolf Rosa, Ph.D.

Study programme: Computer Science (B1801)

Study branch: General Computer Science Bc. R9 (NIOI9B)

Prague 2021

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Generating text from structured data

Author: František Trebuňa

Institute: Institute of Formal and Applied Linguistics (ÚFAL)

Supervisor: Mgr. Rudolf Rosa, Ph.D., Institute of Formal and Applied Linguistics (ÚFAL)

Abstract: Abstract.

Keywords: text generation structured data natural language processing neural networks

Contents

Introduction

1. Problem statement

In september 2020 I read a blog by [Karpathy, 2015]. He created a neural network consisting of only one LSTM cell and trained it to predict a character based on all the previous ones. The network was trained on a corpus of all the plays by Shakespeare. During inference the last predicted character was fed as the input to the network and this way it could create a really good looking Shakespeare-like text. Then I began to explore the possibilities of generating a text conditioned on some input parameters. How to construct a network that could be told to generate a sad, happy, or sarcastic sounding text?

1.1 Data to text generation

Known datasets (WIKIBIO, WeatherGov, RoboCup) -¿ short description, the generated summaries are one-two sentences long. Rotowire -¿ really long summaries although only a fraction of the number of unique tokens from the WIKIBIO dataset. Only short description of the dataset, the statistics and observations are in the second chapter.

1.2 Fantasy sports

What it is, where it originates, the relation to basic optimisation problems, why NLGenerated summaries could be added value.

1.3 My goal

Fluent text which captures the important statistics from the table

2. Data

One needs a lot of data if he wants to train his neural network. E.g. [Sennrich et al., 2016] trains the neural machine translation system on 4.2 million English-German sequence pairs. The Data-to-Text generation task má vyššie nároky na kvalitu datasetu. Potrebujeme, aby boli vstupné dáta štandardizované a aby výstupné texty zodpovedali vstupným dátam. Existuje viacero datasetov, ktoré spĺňajú túto podmienku. V tejto kapitole predstavím datasety WikiBIO a Rotowire, ktoré používam pre svoje experimenty.

Tréning prebieha v plne supervised režime, teda potrebujeme ako vstupné štruktúrované dáta, tak aj výstupné sumáre. Zatiaľ čo vstupné štruktúrované dáta sú vo forme tabuliek, ktoré sú v preprocessingu (viac v kapitole 3) upravené na formu, ktorá sa dá použiť ako vstup pre generačný systém, výstupné dáta sú vo forme tokenizovaných textov.

2.1 WikiBIO dataset

Na to, aby som vedel rozoznať, čím je generovanie popisu športového zápasu náročné, rozhodol som sa pracovať ešte s jednou úlohou a to s generovaním krátkych životopisov na základe infoboxu z wikipédie. Touto úlohou sa zaoberali tvorcovia datasetu WikiBIO [Lebret et al., 2016].

2.1.1 Structured Data

Vstupné štruktúrované dáta sú vo forme infoboxu z Wikipedie. Infobox je tabuľka faktov opisujúca život osoby. Charakteristickou vlastnosťou infoboxov je ich rôznorodosť. Vyplýva to z toho, že osoby s inou kariérou majú výrazne iné charakteristiky. U niekoho hovoríme o titule, u iného o povolaní, u niekoho spomenieme významné práce, u niekoho tímy, v ktorých hraje. Zatiaľ čo zrejme pri každej osobe sa vyskytne napríklad položka "dátum narodenia", pri žijúcich sa zrejme nevyskytne "dátum úmrtia". Dá sa teda povedať, že štruktúrované dáta z tohoto datasetu sú veľmi ťažké, pretože generačný systém nemôže predpokladať štruktúru, ale musí sa naučiť rozoznať ju. Príkladom môže byť infobox pre T.G. Masaryka 2.1.

2.1.2 Summaries

Ako sumár infoboxu slúži prvá veta z článku na wikipedii, ku ktorému prislúcha daná tabuľka. Z pozorovaní vstupných dát vyplýva, že prvá veta väčšinou obsahuje meno, dátum narodenia, prípadne dátum úmrtia a najdôležitejšie činnosti a úspechy osoby, ktorej sa infobox týka. Už z pozorovaní tvorcov datasetu [Lebret et al., 2016] vyplýva, že okolo tretiny tokenov nachádzajúcich sa v prvej vete pochádza z tabuľky. Generačný systém pracujúci s týmto datasetom teda bude musieť správne určiť, kedy sa spoľahnúť na jazykové zručnosti a kedy kopírovať. Ako príklad možno použiť sumár prislúchajúci k infoboxu 2.1.

2.2 Rotowire dataset

Pre samotnú úlohu generovania textového popisu zápasu zo štruktúrovaných dát som si vybral dataset RotoWire [Wiseman et al., 2017]. Štruktúrované dáta sú vo forme tabuľky. Tabuľka obsahuje jednak hodnoty popisujúce celkové tímové štatistiky, jednak hodnoty, popisujúce štatistiky hráčov. Ako výstupné dáta slúžia sumáre jednotlivých zápasov z portálu venujúcemu sa real-time fantasy sports news `https://www.rotowire.com/basketball/` (see below).

2.2.1 Structured Data

V tejto podsekcii si rozoberieme, aké dáta sa v tabuľke nachádzajú, a čo z nich sa dá využiť pri vytváraní sumáru. Pre prehľadnosť ich rozdelím na tri časti, kontextové údaje, tímové údaje a štatistiky a hráčske údaje a štatistiky. Keďže kontextové údaje sú vo vstupnej tabuľke zastúpené len ako informácia v ktorý deň sa zápas konal a vo väčšine sumárov táto informácia nie je použitá, rozhodol som sa ju tiež úplne vynechať z trénovacích dát. Všetky číselné údaje v tabuľkách sú tvorcami datasetu upravené na celočíselné hodnoty.

tomáš garrigue masaryk , sometimes called thomas masaryk in english (7 march 1850 - 14 september 1937) , was a czechoslovak politician , sociologist and philosopher , who as an eager advocate of czechoslovak independence during world war i became the founder and first president of czechoslovakia .

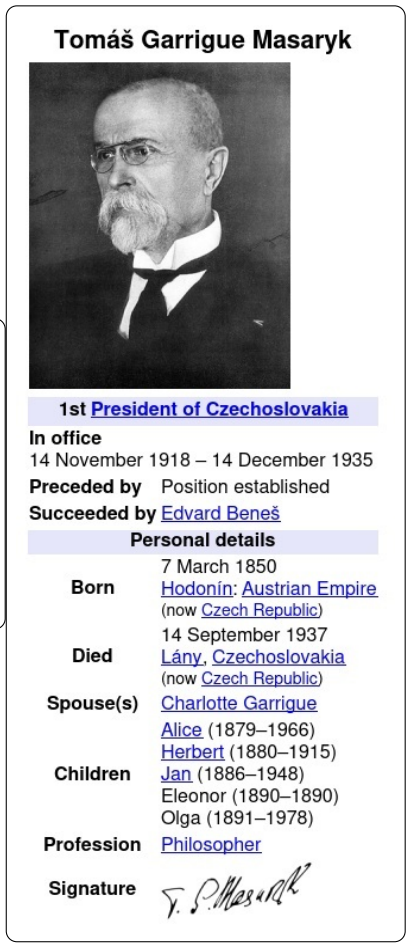


Figure 2.1: Príklad z trénovacích dát datasetu WikiBIO

Team Statistics

Medzi tímové údaje rátam jednak mená klubov, mestá, kde kluby sídlia. Štatistiky sú jednak kontextové (počet predchádzajúcich zápasov, ktoré klub vyhral, resp. prehral) jednak zápasové (napríklad počet získaných bodov, percentuálna úspešnosť strelby z poľa atp.). Tímové údaje a štatistiky obsahujú celkovo 15 položiek pre každý tím, tvorcovia datasetu túto časť označujú ako *line score* a ich celý výčet sa dá nájsť na github tvorcov datasetu [1] Tabuľka 2.1 z validačného datasetu môže poslúžiť ako dobrý príklad.

Name	City	PTS_1	AST_2	REB_3	TOV_4	Wins	Losses ...
Raptors	Toronto	122	22	42	12	11	6 ...
76ers	Philadelphia	95	27	38	14	4	14 ...

Note: The statistics are accumulated across all the team players

$_1$ Points; $_2$ Assists; $_3$ Rebounds; $_4$ Turnovers

Table 2.1: Príklad tímových štatistík z datasetu Rotowire

Player statistics

U každého hráča je údaj, za ktorý tím hrá, či je daný tím domáci, alebo či hráč patril do starting roster, alebo nastúpil z lavičky. Štatistiky sú len zápasové a medzi ne patria napríklad počet nahraných bodov, počet minút, ktoré hráč hral atp. Hráčske údaje a štatistiky čítajú 24 položiek, tvorcovia datasetu ich nazývajú *box score* a čitateľa, ktorého zaujíma kompletný výpis položiek opäť odporúčame na github tvorcov datasetu [1] a uvádzame aj príklad tabuľky z validačného datasetu 2.2.

Name	Team City	S_POS_1	PTS_2	STL_3	BLK_4 ...
Kyle Lowry	Toronto	G	24	1	0 ...
Terrence Ross	Toronto	N/A	22	0	0 ...
Robert Covington	Philadelphia	G	20	2	0 ...
Jahlil Okafor	Philadelphia	C	15	0	1 ...

Note: N/A means that the statistic couldn't be collected because it is undefined
(e.g. player didn't appear on starting roster therefore his starting position is undefined)

$_1$ Starting position ; $_2$ Points; $_3$ Steals; $_4$ Blocks

Table 2.2: Príklad hráčskych štatistík z datasetu Rotowire

2.2.2 Summaries

Kladieme vysoké nároky na sumáre. Jednak musia súvisieť s tabuľkou, jednak nechceme neurónovú sieť učiť niečo, čo by sa dalo rýchlo napísať pomocou jednoduchej šablóny. Preto potrebujeme sumáre, ktoré dokážu vytiahnuť z

[1] `https://github.com/harvardnlp/boxscore-data`

tabuľky hlbšie štatistiky a správne s nimi pracovať. Ako už vysvetľuje [Wiseman et al., 2017] obyčajné športové spravodajstvo zo stránok `https://www.sbnation.com/nba` nespĺňa tieto kritériá, keďže obsahuje priveľa informácií, ktoré sú založené na iných kontextoch, ako sú dáta v zápasovej tabuľke. Práve preto je zaujímavé sledovať fenomén fantasy športov.

Fantasy sports

Podľa [Tozzi, 1999] môžeme datovať počiatky fenoménu fantasy športu do šesťdesiatych rokov dvadsiateho storočia. Podľa stránky RotoWire[2] je základom fantasy športu vytvorenie tímov reálnych hráčov z ligy a získavanie bodov na základe ich výkonov v skutočných zápasoch. Bodovanie berie do úvahy, či ide o defenzívneho, alebo ofenzívneho hráča a je nutné vybrať hráča na každú pozíciu v hre. (čím sa zaručuje, že nemôžu existovať fantasy tímy, ktoré majú len point guardov). Disponujete obmedzenými zdrojmi na nákup hráčov a hráči, u ktorých je pravdepodobnejšie, že skórujú viac bodov, resp. získajú viac lôpt a tak podobne, stoja viac. Skutočné pravidlá sú jemne zložitejšie a tí, ktorých by fantasy ligy zaujímali odporúčame na dané stránky[2].

Fantasy sports news

Na to, aby hráč mohol uspieť vo fantasy lige, potrebuje mať dokonalý prehľad o štatistikách hráčov, o trendoch, o zraneniach, o tímoch, ktorým sa darí aj o tých, ktorým sa možno začne dariť neskôr. Práve preto už od začiatku organizovania fantasy líg existujú formy spravodajstva, ktoré sa špecializujú práve na hráčov fantasy športov. Podľa legendy popísanej [Tozzi, 1999] sa hráči jednej z prvých fantasy líg schádzali v reštaurácii La Rotisserie Francaise, a podľa tejto reštaurácie je pomenovaná aj stránka špecializujúca sa na spracovávanie štatistík pre fanúšikov fantasy líg, `https://www.rotowire.com/`. Články písané o zápasoch oveľa viac berú do úvahy, čo sa v zápase udialo a zároveň poskytujú aj hlbší náhľad do kontextu zápasu. Preto ako píše [Wiseman et al., 2017] je pre generačné systémy ideálnejšie učiť sa generovať práve články podobné tým z RotoWire.

2.2.3 Relation of summaries and tables

V tejto kapitole na jednoduchom príklade ukážem, ako sú štatistiky z tabuľky previazané so sumárom z RotoWire. Vo figure 2.2 je krásne vidieť, že väčšina údajov zo sumáru pochádza z tabuľky. Napriek tomu však text obsahuje niektoré údaje (zvýraznené žltou farbou), ktoré v tabuľke vôbec nie sú a niektoré údaje (zvýraznené modrou farbou), ktoré sú v tabuľke len implicitne a je ich potrebné odvodiť. Zatiaľ čo fakt, že keďže Terrence Ross má nedefinovanú štartovaciu pozíciu, tak musel nastúpiť do zápasu z lavičky je celkom zrejmý, informácia o tom, že Joel Embiid nehrá je dôležitá len vtedy, pokiaľ je Joel Embiid natoľko dôležitý hráč pre tím, že stojí za zmienku ho spomenúť. To je však kontext, ktorý v tabuľke spomenutý nie je a navyše informácia o tom, prečo nehrá nemôže byť jasná ani z kontextu celého korpusu dát.

[2] `https://www.rotowire.com/basketball/advice/`

TEAM	WIN	LOSS	PTS_1	FG_PCT_2	REB_3	AST_4 ...
Raptors	11	6	122	55	42	22
76ers	4	14	95	42	38	27

PLAYER	City	PTS_1	AST_4	REB_3	FG_5	FGA_6	S_POS_7 ...
Kyle Lowry	Toronto	24	8	4	7	9	G
Terrence Ross	Toronto	22	0	3	8	11	N/A
Robert Covington	Philadelphia	20	2	5	7	11	G
Jahlil Okafor	Philadelphia	15	0	5	7	14	C
DeMar DeRozan	Toronto	14	5	5	4	13	G
Jonas Valanciunas	Toronto	12	0	11	6	12	C
Ersan Ilyasova	Philadelphia	11	3	6	4	8	F
Sergio Rodriguez	Philadelphia	11	7	3	4	7	G
Richaun Holmes	Philadelphia	11	1	9	4	10	N/A
Nik Stauskas	Philadelphia	11	2	0	4	9	N/A
Joel Embiid	Philadelphia	N/A	N/A	N/A	N/A	N/A	N/A
...							

The host Toronto Raptors defeated the Philadelphia 76ers , 122 - 95 , at Air Canada Center on Monday . The Raptors came into this game as a monster favorite and they did n't leave any doubt with this result . Toronto just continuously piled it on , as they won each quarter by at least four points . The Raptors were lights - out shooting , as they went 55 percent from the field and 68 percent from three - point range . They also held the Sixers to just 42 percent from the field and dominated the defensive rebounding , 34 - 26 . Fastbreak points was a huge difference as well , with Toronto winning that battle , 21 - 6 . Philadelphia (4 - 14) had to play this game without Joel Embiid (rest) and they clearly did n't have enough to compete with a potent Raptors squad . Robert Covington had one of his best games of the season though , tallying 20 points , five rebounds , two assists and two steals on 7 - of - 11 shooting . Jahlil Okafor got the start for Embiid and finished with 15 points and five rebounds . Sergio Rodriguez , Ersan Ilyasova , Nik Stauskas and Richaun Holmes all finished with 11 points a piece . The Sixers will return to action on Wednesday , when they host the Sacramento Kings for their next game . Toronto (11 - 6) left very little doubt in this game who the more superior team is . Kyle Lowry carried the load for the Raptors , accumulating 24 points , four rebounds and eight assists . Terrence Ross was great off the bench , scoring 22 points on 8 - of - 11 shooting . DeMar DeRozan finished with 14 points , five rebounds and five assists . Jonas Valanciunas recorded a double - double , totaling 12 points and 11 rebounds . The Raptors next game will be on Wednesday , when they host the defensively - sound Memphis Grizzlies .

Note: $_1$ Points; $_2$ Field Goal Percentage; $_3$ Rebounds; $_4$ Assists; $_5$ Field Goals; $_6$ Field Goals Attempted; $_7$ Starting Position; N/A means undefined value

Figure 2.2: Príklad vstupných tabuliek a zlatého sumáru z datasetu. Žltou sú zvýraznené informácie nenachádzajúce sa v tabuľke, zatiaľ čo modrou sú zvýraznené informácie, ktoré z tabuľky a celého korpusu vyplývajú len implicitne.

3. Preprocessing and Statistics of the Datasets

Ako hlavný spôsob riešenia problému generovania prirodzeného textu zo štruktúrovaných dát volíme RNN. RNN sú uspôsobené na spracúvanie sekvenčných, 1D dát, avšak my potrebujeme spracovať 2D tabuľky. V tejto kapitole popíšem spôsob, ako sa s týmto problémom vyrovnávam, pričom sa pokúsim vysvetliť, prečo je postup rozdielny pre dataset WikiBIO a RotoWire. Ďalej budem rozprávať o tom, ako som ďalej upravoval vstupné a výstupné dáta a dôvody pre každú z aplikovaných zmien.

3.1 Transforming Tables to Records

Najprv sa pokúsim ukázať aký cieľ chcem naplniť pri transformácii tabuľky na sekvenčný vstup. Zadefinujme si tabuľku, s ktorou budeme pracovať. Povedzme, že stĺpce budú označovať typy hodnôt a riadky budú označovať entity. (ako v príklade 3.1). Chceme, aby čo najviac informácií ostalo v dátach. Konkrétne to znamená, že v sekvenčnom vstupe by malo ostať zachované, jednak ku ktorej entite daný vstup patrí, jednak aký typ hodnoty prislúchajúci k danej entite vyjadruje.

3.1.1 Notation

Podľa [Liang et al., 2009] zavádzam notáciu, ktorú budem ďalej používať. Tabuľku $\mathcal{T}$ transformujeme na postupnosť záznamov $\mathbf{s} = \{r_i\}_{i=1}^{J}$, kde r_i označuje i-ty záznam. Vzhľadom na ciele stanovené vyššie, každý záznam položku $r.f$ označujúcu typ hodnoty, položku $r.v$ označujúcu hodnotu daného záznamu, prípadne položku $r.e$, označujúcu entitu, ktorej prislúcha.

3.2 WikiBIO

Transformácie a štatistiky na datasete sa týkajú jednak štruktúrovaných dát, jednak sumárov, ktoré má generačný systém generovať. Štruktúrované dáta majú formu infoboxu, ako sumár slúži prvá veta z príslušného článku na wikipedii. Ako v príklade 2.1. Najprv predstavím štatistiky, následne na ich základe uvediem, aké transformácie a preprocessing som zvolil.

	field_1	field_2 ...
entity_1	$\text{value}_{1,1}$	$\text{value}_{1,2}$ ...
entity_2	$\text{value}_{2,1}$	$\text{value}_{2,2}$ ...

Table 3.1: An example of structured data

3.2.1 Dataset statistics

Pre tréning neurónovej siete používam origiálny train-valid-test split od autorov, teda 582 659 trénovacích infobox-sumár párov, 72 831 validačných a 72 831 testovacích.

Sumáre obsahujú celkovo 493 878 unikátnych tokenov, celkovo 18 981 222 tokenov. Tabuľky obsahujú 7 200 unikátnych typov.

3.2.2 Transformation of Infoboxes

Každý infobox sa týka práve jednej entity. To veľmi uľahčuje transformáciu na záznamy, kedže na tabuľku sa možno pozerať ako na zoznam dvojíc $(type, \{value_i\}_{i=1}^{|\mathbf{value}|})$. Ako už vyplýva z notácie, počet hodnôt prislúchajúcich jednomu typu môže byť rôznorodý.

Existujú minimálne dve možnosti ako sa s tým vysporiadať. Prvou je vyhlásiť hodnotu prislúchajúcu jednomu typu za jeden token (ako v príklade 3.1) a prípadne vytvoriť viacero typov lepšie reprezentujúcich to, čo sa v tabuľke nachádza. To je však vzhľadom na veľkosť datasetu (okolo 730 000 párov infobox-sumár, viac ako 7000 rôznych typov) nevhodný prístup.

(successor, edvard beneš)

Figure 3.1: An example of a record made from infobox in figure 2.1 by treating all the values as one token

Druhou možnosťou je prístup, ktorý zvolili aj tvorcovia datasetu [Lebret et al., 2016], či autori state of the art riešenia [Liu et al., 2017]. Každú hodnotu z množiny hodnôt považujeme za samostatný token. Tabuľka však nemá stálu štruktúru, preto je napríklad problém rozlíšiť, či záznam (successor, edvard) vyjadruje krstné meno, alebo priezvisko nástupcu Tomáša Garrigue Masaryka v jeho funkcii. Preto ku každému záznamu pridávame hodnotu *r.pos*, vyjadrujúcu poradie (číslované od 1) vrámci hodnôt prislúchajúcich jednomu typu, ako je ukázané v príklade 3.2.

(successor, edvard, 1),
(successor, beneš, 2)

Figure 3.2: An example of a record made from infobox in figure 2.1 by adding positional information and treating each value as a separate token

Zatiaľ čo nástupcom T.G. Masaryka bol Edvard Beneš, nástupcom amerického prezidenta Herberta Hoovera bol Franklin D. Roosevelt. Rozpoznať, že obidva záznamy (successor, beneš, 2) a (successor, roosevelt, 3) vyjadrujú priezvisko danej osoby môže byť ťažké. Preto pridávame aj informáciu o poradí od konca, *r.rpos*. Potom je na vybranom príklade už zrejmé, že záznamy s $r.f = successor$ a $r.rpos = 1$ vyjadrujú tú istú informáciu.

(name, tomáš, 1, 3), (name, garrigue, 2, 2), (name, masaryk, 3, 1), (image, tomáš, 1, 16), (image, garrigue, 2, 15), (image, masaryk, 3, 14), (image, ",", 4, 13),(image, bain, 5, 12) ...

Figure 3.3: An example of records made from infobox in figure 2.1 with all the additional information included

3.2.3 Preprocessing

Jednotlivé sety (train, valid, test) ešte prefiltrujem tak, aby žiadna tabuľka nebola dlhšia ako 100 recordov a žiadny sumár nebol dlhší ako 75 tokenov. (urobené na základe štatistík v tabuľke 3.2) Keďže tento dataset nie je nosným datasetom tejto práce, rozhodol som sa neexperimentovať s preprocessingom a použil som hodnoty hyperparametrov, ktoré zvolili [Liu et al., 2017]. V krátkosti zhrniem, čo konkrétne používam.

General

Celý dataset je lowercaseovaný, čiarky, zátvorky, bodky ...sú považované za samostatné tokeny. Všetky okrem najčastejších 20 000 tokenov vybraných [Liu et al., 2017] v sumároch a hodnotách tabuliek nahradím špeciálnymi UNK tokenmi.

Tables

Všetky záznamy, kde by hodnota činila *none*, resp. nevalidné hodnoty v tabuľke (prázdne, alebo s nesprávne zadefinovaným typom) odstránim. Taktiež všetky typy, ktoré sa nevyskytujú v slovníku typov od [Liu et al., 2017] (ktorý je zozbieraný zo všetkých typov vyskytujúcich sa aspoň 100-krát) nahradím UNK tokenmi.

Records$_1$	Percentile$_2$
50	56,56
75	82,6
100	93,4

(a) Lenght statistics of tables
Note: $_1$ Number of records, where (f, v, N, M) and (f, v, N+1, M-1) are treated as 2 distinct records
$_2$ Percentage of tables with lower number of records

Tokens$_1$	Percentile$_2$
25	55,76
50	96,84
75	99,68

(b) Length statistics of summaries
Note: $_1$ Number of tokens in a summary
$_2$ Percentage of summaries with lower number of tokens

Table 3.2: Statistics of the WikiBIO dataset

3.3 RotoWire

Na rozdiel od datasetu WikiBIO, v tabuľke charakterizujúcej jeden zápas NBA (teda vo vstupe pre generačný model) sa vyskytuje viacero entít a dokonca viacero

druhov entít (tímy, hráči). Tieto fakty je potrebné zobrať do úvahy pri spracúvaní tabuliek a ich transformácii na sekvenčný vstup pre RNN.

3.3.1 Dataset Statistics

Keďže transformácie sumárov sú výrazne ovplyvnené štatistikami datasetu, najprv si dovolím predstaviť to, ako vyzerajú tieto štatistiky. Rovnako ako pri datasete WikiBIO, aj teraz používam originálny train-valid-test split (3397, 727, 728). Pri spracovaní datasetu vychádzam z 2 predpokladov:

- Neurónová sieť sa môže naučiť generovať určitý token, pokiaľ sa v tréningových dátach vyskytuje aspoň päťkrát.
- Je jednoduchšou úlohou kopírovať dáta z tabuľky, ako generovať zo skrytého stavu.

Prezentované štatistiky sú s preprocessingom, ktorý používa [Wiseman et al., 2017] [1], neskôr, v sekcii 3.3.3 ukážem štatistiky, ktoré sú po aplikácii všetkých úprav na datasete.

Length-wise Statistics

Jeden hráč je v tabuľke zastúpený 24 záznamami, jeden tím 15 záznamami. Každá tabuľka obsahuje záznamy o 2 tímoch. Podľa dát z tabuľky 3.3 teda ostáva 528 až 720 záznamov o hráčoch, teda v jednej tabuľke sa hovorí o 22 - 30 hráčoch. Z toho vyplýva, že generačný systém nemá úlohu ešte sťaženú rôznorodosťou tabuliek, keďže úseky, kde sa rozpráva o hráčoch a o tímoch sú relatívne rovnaké vo všetkých tabuľkách.

Set	Max Summary Length	Min Summary Length	Avegage Summary Length	Size
train	750	558	644.65	3397
validation	702	582	644.66	727
test	702	558	645.03	728

Table 3.3: Statistics of tables as used by [Wiseman et al., 2017][1]

Na rozdiel od vstupných tabuliek, výstupné sumáre sú čo sa týka dĺžky oveľa rôznorodejšie (ako je vidieť v tabuľke 3.4). Tu je podstatné hlavne to, že priemerná dĺžka sumáru je viac ako 330 tokenov a pokiaľ chceme generačný systém trénovať aj na dlhších sumároch, tak dĺžka tabuľky a sumáru nám vytvorí netriviálne pamäťové nároky na grafickú kartu a na RAM. (*tu by mohol byť odkaz na sekciu o implementačných problémoch*)

[1] V datasete sa však vyskytla jedna nezdokumentovaná chyba, ktorú som odstránil, jeden sumár bol známym textom Lorem Ipsum, ten je z datasetu, ktorý používam odstránený a teda trénovacie dáta obsahujú 3397 položiek a nie 3398, ako uvádza [Wiseman et al., 2017]

Set	Max Summary Length	Min Summary Length	Avegage Summary Length	Size
train	762	149	334.41	3397
validation	813	154	339.97	727
test	782	149	346.83	728

Table 3.4: Statistics of summaries as used by [Wiseman et al., 2017][1]

Frequency of Unique Tokens

Teraz sa zameriam na štatistiky dôležité pre pochopenie motivácie pre ďalšie transformácie datasetu vzhľadom k skôr prezentovaným predpokladom 3.3.1. V tabuľke 3.5 je vidieť, že viac ako 61% tokenov, ktoré nie sú číslami, menami hráčov, tímov, alebo miest sa v tréningovom datasete vyskytne menej ako 5-krát. Z toho sme usúdili, že je zrejme potrebné urobiť ďalšie transformácie na datasete, čím by sme zvýšili pravdepodobnosť, že sa generačný systém lepšie datasetu uspôsobí.

Set	Unique Tokens	>= 5 Absolute	>= 5 Relative
train	9779	4158	42.52%
train_wop$_1$	8604	3296	38.31%
train_wopl$_2$	8031	3119	38.84%

Note: $_1$ train_wop is training set with all the player names, city names, team names and numbers extracted $_2$ train_wopl is train_wop lowercased

Table 3.5: Occurrences of tokens in summaries from dataset RotoWire

Podobné závery možno urobiť aj zo štatistík prekryvu slovníkov trénovacích a validačných (resp. testovacích) dát.

Set	Unique Tokens	Train Overlap	Train$_{>=5}$ Overlap
valid	5625	88.18%	66.63%
test	5741	87.46%	65.72%
valid_wop$_1$	4714	86.36%	61.92%
test_wop$_2$	4803	86.03%	61.13%
valid_wopl$_3$	4442	86.74%	62.36%
test_wopl$_4$	4531	86.32%	61.37%

Note: train$_{>=5}$ is a set of all the tokens with more than 5 occurrences in the train dataset summaries $_1$, $_2$, $_3$, $_4$ have the same meaning as in table 3.5

Table 3.6: Overlap of train dataset summaries and valid/test dataset summaries

3.3.2 Transformations of Input Tables

Ako je spomenuté v sekcii 2.2.1, hodnoty v tabuľke môžu byť buď mená tímov, hráčov, miest, alebo celé číslo oznamujúce percentuálnu, či absolútnu hodnotu nejakej štatistiky.

Preto má zmysel použiť rozdielny prístup ku spracovaniu hodnôt ako pri datasete WikiBIO. Konkrétne budeme každú hodnotu považovať za jeden token. Väčšina hodnôt je číselná (konkrétne $\frac{13}{15}$ z typov charakterizujúcich tímy a $\frac{19}{25}$ z typov charakterizujúcich hráčov. Následne mená tímov sú väčšinou dlhé práve jeden token, až na jednu výnimku, z ktorej sa vyrobí jeden token (*Trail Blazers* → *Trail_Blazers*). Podobne pre mená tímov a výnimky (*Oklahoma City* → *Oklahoma_City*, *Golden State* → *Golden_State*, . . .).

Transformations of Player Names

Mená hráčov sú všetky, až na jednu výnimku (*Nene*) viac-tokenové a autori, ktorých prístup ku danému datasetu som bral za referenciu ([Wiseman et al., 2017], [Puduppully et al., 2019]) zvolili odlišný prístup ako ten, ktorý používam ja.

Referenčný prístup používa 2 špeciálne typy, *first_name* a *last_name*. Vďaka nim sa úloha pochopiť, či token *James* odkazuje na krstné meno hviezdneho *James Harden*, alebo na priezvisko legendy *LeBron James* necháva na generačný systém.

Môj prístup je založený na myšlienke, že už na tak veľmi ťažkom datasete je dôležité vytvoriť čo najjednoduchšiu úlohu pre neurónovú sieť. Preto transformujem sumáre tak, aby meno každého hráča bolo reprezentované práve jedným tokenom. Práve preto strácajú typy *first_name* a *last_name* zmysel a vo vstupnej tabuľke ich nemám.

Entities

Keďže vstupná tabuľka obsahuje informácie o obidvoch hrajúcich tímoch a všetkých hráčoch na súpiskách, každý záznam musí obsahovať aj hodnotu $r.e$ popisujúcu, ktorú entitu záznam charakterizuje. Vďaka transformácii mien hráčov a tímov, môžeme ako entitu použiť názov tímu, resp. meno hráča. Okrem toho však ku každému záznamu pridáme špeciálnu položku $r.ha$, ktorá symbolizuje, či sa záznam vzťahuje ku domácemu, alebo hosťujúcemu tímu.

Record Format

Nakoniec teda používam záznamy, ktoré obsahujú položky $r.f$ - typ záznamu, $r.e$ - entita, $r.v$ - hodnota a $r.ha$ - domáci/hostia. (tu by mal byť pridaný príklad, ako to vyzerá)

Kvôli tomu vykonávame také transformácie, aby sa jednak vyskytovalo čo najviac tokenov čo najčastejšie, a aby v sumároch boli používané rovnaké tokeny ako v tabuľke.

Number Transformations

Rovnako ako [Wiseman et al., 2017] a [Puduppully et al., 2019], reprezentujeme čísla len pomocou číslic. Táto transformácia je motivovaná druhým predpokladom a teda tým, že dúfame, že väčšinu čísel bude neurónová sieť kopírovať a nie generovať. To znamená, že napríklad token *five* transformujeme na 5. Používame na to knižnicu text2num [1]. Niektoré čísla však majú v basketbalovej terminológii špeciálny význam. Špecificky teda netransformujeme číslo *three*, pretože sa často vyskytuje ako súčasť slovných spojení *three pointer, three pt range ...*

Player name transformations

Ako už vyplýva z 3.3.2 a z predpokladov, ktoré sme si stanovili, chceme v texte reprezentovať jednoho hráča práve jedným tokenom. To platí aj pre extrémne prípady ako *Luc Richard Mbah a Moute*, ktorý bol v datasete reprezentovaný šiestimi rôznymi kombináciami elíps v jeho mene. Najčastejšie však vychádza, že v texte sa najprv hráč spomenie a následne sa už oslovuje len priezviskom a až na 17 prípadov (čo činí 2.4% zo 668 hráčov spomenutých aspoň v jednej tabuľke) je meno hráča vždy zložené z 2 tokenov. Na transformáciu mien hráčov používam jednoduchý algoritmus, ktorý napriek tomu, že nerieši všetky referencie úplne presne, dosahuje na oko slušnú presnosť.

While King James struggled , James Harden was busy putting up a triple - double on the Detroit Pistons on Friday

↓

while king LeBron_James struggled , James_Harden was busy putting up a triple - double on the Detroit Pistons on friday.

Figure 3.5: Example of transformation of player names leveraging the knowledge of players on the rosters

Transformácia prebieha v troch krokoch. Teraz sa ich pokúsim popísať na príklade transformácie vety z figure 3.5:

- **1. Extrakcia mien hráčov zo sumáru**
  Vytiahneme zo sumáru mená *James* a *James Harden*, ktoré sú súčasťou mena nejakého z hráčov, ktorí sú známi z korpusu

- **2. Vyriešenie referencií pre daný sumár a vytvorenie slovníka transformácií**
  Pokúsime sa zistiť, na čo odkazuje jednotokenové meno *James*. Explicitne zakazujeme, aby sa vnímalo ako krstné meno a v boxscore daného zápasu nájdeme, že doň zasiahol niekto s menom *LeBron James*. Do slovníka transformácií teda umiestnime transformáciu *James* → *LeBron_James*. *James Harden* je viac tokenové meno, ktoré sme v prvom kroku rozoznali ako meno hráča, teda do slovníka pridáme len prepis *James Harden* → *James_Harden*

[1] `https://github.com/allo-media/text2num`

- 3. Aplikácia transformácií na sumár
  Prechádzame text a na najdlhšiu namatchovanú postupnosť aplikujeme transformáciu.

3.3.3 Byte Pair Encoding

Pokiaľ by sme sa rozhodli, že ako slovník použijeme len slová, ktoré generačný systém pozná z tréningového datasetu, museli by sme nahradiť viac ako 10% unikátnych slov z validačného, či testovacieho datasetu špeciálnymi *UNK* tokenmi. Namiesto toho, sme sa rozhodli použiť prístup, ktorý predstavil [Sennrich et al., 2016]. Najprv popíšeme algoritmus, následne ukážeme, ako pomohol prekonať problém, o ktorom rozprávame.

Algorithm

Používame implementáciu algoritmu priamo od autorov[2], ktorá sa dá stiahnuť aj ako samostatný python package. Minimálny príklad možnej implementácie v pythone je ukázaný ako algoritmus 1.

Ako prvý krok sa každé slovo rozdelí na znaky a na koniec každého slova sa pridá špeciálny znak <*eow*>, ktorý umožní detokenizáciu na pôvodný text. Vstupný slovník sa inicializuje na množinu unikátnych znakov v texte (okrem bielych znakov, algoritmus sa týka len slov). Algoritmus mnohokrát prechádza text a pri každom prechode nájde najčastejšiu dvojicu za sebou idúcich znakov a spojí ju do jednoho nového znaku. Tento znak sa pridá do slovníka a všetky výskyty daných dvoch znakov za sebou sa nahradia novým znakom.

Veľkosť výstupného slovníka je teda počet unikátnych znakov v pôvodnom texte (+1 kvôli <*eow*> tokenu) + počet prebehnutých iterácií.

Algoritmus má teda jeden hyperparameter a to počet iterácií, a teda veľkosť výstupného slovníka.

Statistics of Transformed Dataset

Vyskúšali sme tri možnosti nastavenia hyperparametru BPE[3]. Nakoniec sme zvolili 2000 iterácií algoritmu. Z procesu spájania tokenov sú vybrané čísla a mená hráčov, keďže chceme, aby tieto tokeny generačný systém priamo kopíroval zo vstupnej tabuľky. Tým pádom prienik tréningového a validačného (resp. testovacieho) datasetu nie je 100%.[4]Výsledné štatistiky po všetkých transformáciách sú v tabuľkách 3.7 a 3.8.

Set	**Unique Tokens**	>= 5 **Absolute**	>= 5 **Relative**
train	2839	2531	89.15%

Table 3.7: Occurrences of tokens in transformed summaries from dataset RotoWire

[2]`https://github.com/rsennrich/subword-nmt`

[3]Možnosti nastavenia počtu iterácií BPE spolu so štatistikami s nimi spojenými sú prístupné na `https://github.com/gortibaldik/TTTGen/blob/master/rotowire/dataset_stats.md`

[4]Konkrétne spolu 92 tokenov (31 unikátnych tokenov) z validačného a 87 tokenov (34 unikátnych tokenov) je nahradených UNK tokenom vo validačných dátach.

Set	Unique Tokens	Train Overlap	Train$_{>=5}$ Overlap
valid	2582	98.80%	95.70%
test	5741	98.69%	95.45%

Table 3.8: Overlap of transformed train dataset summaries and valid/test dataset summaries

Algorithm 1: Learn BPE operations
Extract from paper **Neural Machine Translation of Rare Words with Subword Units** by [Sennrich et al., 2016]

```
1
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
        pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
           'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)

> r · → r·
> l o → lo
> lo w → low
> e r· → er·
```

4. Neural Network Architectures

To generate text from structured data I make use of deep neural networks. In previous chapter (3) I've shown how to transform the structured tabular data into a sequence of records, therefore I've reduced the problem to an instance of the famous sequence to sequence problem. Now, I show how to create a system that transforms the input sequential data (structured records) to the output sequential data (natural language).

The most common way to tackle the sequence to sequence problem is to use the Encoder-Decoder architecture proposed by [Sutskever et al., 2014]. It is the main approach I used throughout this thesis. In this chapter I introduce the concepts behind the encoder-decoder architecture, its shortcomings (fixed vocabulary and thus problems with generation of words unseen during training, divergence and hallucinations) and ways to overcome these shortcomings (the attention mechanism, the copy mechanisms, the further transformations of input sequences). Since it is not the purpose of this work, only the basics of the concepts are presented, and I provide links to papers, books and tutorials which helped me on my path to understanding.

Many papers diverge on the notation and calling conventions of the architectures. Therefore I choosed to adopt the notation used in *Tensorflow Keras API, version 2.x* ([Abadi et al., 2015]). Specifically in the field of recurrencies it is discutable if the paper refers to *tf.keras.layers.RNNCell* or to *tf.keras.layers.RNN*. I believe that they can be used interchangeably in the context of this chapter, hence I (deliberately) choose the latter notation (*without 'Cell'*).

4.1 The Encoder-Decoder Architecture

Proposed by [Sutskever et al., 2014] the Encoder-Decoder is composed of 2 recurrent units, called Encoder and Decoder. In this section I briefly introduce the Recurrent Neural Network (*tf.keras.layers.SimpleRNN*), its modification, the Long Short-Term Memory (*tf.keras.layers.LSTM*)([Hochreiter and Schmidhuber, 1997]) and the high-level overview of the Encoder-Decoder architecture.

4.1.1 Recurrent Neural Network

Let $\mathbf{x} = (x^{(1)}, \ldots, x^{(t)})$ be the input. The standard Feed-Forward Network (*tf.keras.layers.Dense*) has different set of weights for each input time-step $x^{(t)}$, therefore the number of time-steps of the input needs to be known in advance.

On the contrary, the Recurrent Neural Network (RNN) ([Rumelhart et al., 1988]) (*tf.keras.layers.SimpleRNN*) shares the same set of weights for each time-step and in addition it keeps a hidden state. At each time-step we update the hidden state and compute the output as in equation 4.2. The computation can be visualised either as a loop, or as a feed-forward network with shared weights 4.1.

The network is trained by back-propagation through time (BPPT) [Werbos, 1990]. It has been shown that the RNN suffers from vanishing / exploding gra-

$$y = activation(W\mathbf{x} + b) \tag{4.1}$$

The Feed-Forward Neural Network

$$\begin{aligned} h_t &= f_h(x_t, h_{t-1}) \\ y_t &= h_t \end{aligned} \tag{4.2}$$

The Recurrent Neural Network
Note: h_t is the hidden state, and y_t is the output at t-th timestep

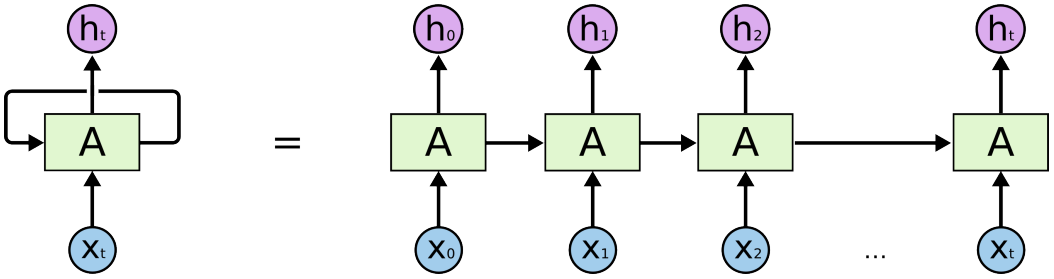


Figure 4.1: Possible visualisations of RNN, with $f_o \cong id$ ([Olah, 2015])

dient problems [Hochreiter and Schmidhuber, 1997].[1] Which cause that either the RNN cannot learn anything or it is really unstable. These difficulties are adressed by more sophisticated architectures such as Gated Recurrent Unit or Long Short-Term Memory.

4.1.2 Long Short-Term Memory

The Long Short-Term Memory ([Hochreiter and Schmidhuber, 1997])(*tf.keras.layers.LSTM*) addresses the problem of vanishing gradient. It does so by adding a special cell state for capturing long range context and a series of gating mechanisms. The latter update the cell state and regulate the flow of gradient through the network (as shown in figure 4.2). The more in-depth explanation can be found in [Olah, 2015].

$$y_t = W_{hy}h_t + b_y \tag{4.3}$$
$$h_t = o_t tanh(c_t) \tag{4.4}$$
$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o) \tag{4.5}$$
$$c_t = f_t * c_{t-1} + i_t * tanh(W_c[h_{t-1}; x_t] + b_c) \tag{4.6}$$
$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i) \tag{4.7}$$
$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f) \tag{4.8}$$

[1] I believe that discussion about BPPT and exploding/vanishing gradient problems is beyond the scope of this work. Therefore I refer the reader craving for further explanation to [Goodfellow et al., 2016] and to referenced papers.

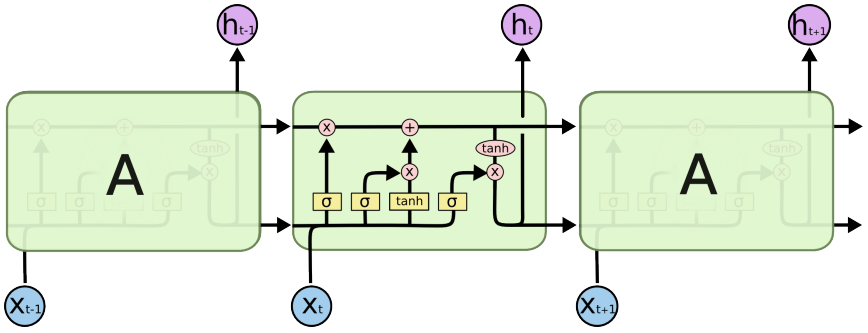


Figure 4.2: Visualisation of LSTM ([Olah, 2015])

4.1.3 High-level Overview of Encoder-Decoder Architecture

As stated by [Sutskever et al., 2014], the main goal of the encoder-decoder architecture is to model the conditional probability $p(y_1, \ldots, y_n | x_1, \ldots, x_m)$ of the output sequence $y_1, \ldots, y_n$ conditioned on the input sequence $x_1, \ldots, x_m$. It uses two separate *recurrent networks*[2]. The first, called Encoder, produces a fixed-dimensional representation r of the input sequence. r is then used to initialize the hidden states of the second recurrent network, called a Decoder. The Decoder then generates the output sequence as in equation 4.9.

$$p(y_1, \ldots, y_n | x_1, \ldots, x_m) = \prod_{t=1}^{n} p(y_t | r, y_1, \ldots, y_{t-1}) \tag{4.9}$$

The Encoder part is straight-forward, however in the Decoder part two important questions arise:

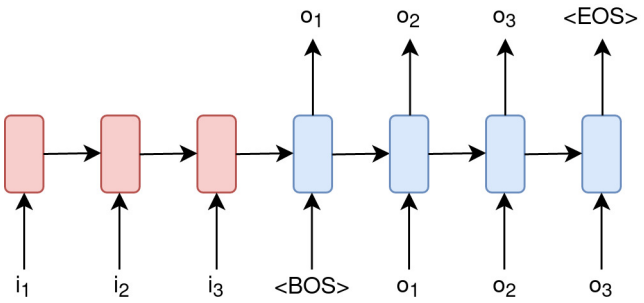
- 1. What should be the inputs to the Decoder
- 2. When should we stop decoding (generating the output sequence)

To tell the Decoder when to *start* generation, a special $<BOS>$ token (*beginning of sequence*) is fed in. The decoding phase is slightly different during training and inference. During training, the inputs of the decoder are targets from the *previous* time-step. This process is called *teacher forcing*. After the last time-step, the target is the special $<EOS>$ token (*end of sequence*). In the inference phase, the last output of the Decoder is fed in as the input, and the decoding finishes after producing the $<EOS>$ token.

4.1.4 Problems of the Encoder-Decoder Architecture

Despite having many advantages (variable length of the input and output sequence, possibility of extending the number of recurrent layers in the encoder and the decoder) there are some major flaws that need to be overcome in order to generate text from structured data.

[2]Here I refer to *recurrent network* as to a complex consisting of at least one RNN/LSTM/-GRU/... rather than to a single recurrent layer.



Note: Encoder is red, Decoder is blue

Figure 4.3: Visualisation of the computation of the Encoder-Decoder Architecture

Fixed-dimensional Representation of the Input Sequence

It has been shown by [Cho et al., 2014] that the performance of the Encoder-Decoder architecture "suffers significantly from the length of sentences". [Bahdanau et al., 2014] hypothesize that it may be because all of the information from the source sequence is encoded to the fixed-dimensional vector. Both mentioned papers understand word *"long"* as *longer than 30 tokens*. From the chapter about the preprocessing 3, we know that there are more than 40 records in the average input from the WikiBIO dataset and more than 300 records in the average input from the RotoWire dataset.

Named Entities and Unknown Words

In the standard Encoder-Decoder, the output is a distribution over the output vocabulary. The distribution is modelled with *the softmax function*. There are two flaws in the design.

Firstly, *the softmax function* is computationally expensive. Therefore only the most frequent words in the training dataset are included to the vocabulary and all the others are replaced by the $<UNK>$ token. Consequently we loose information about the words which were substitued out. I've already shown one way to overcome this issue in subsection 3.3.3. However in the subsection *reference should be there* I'll show how to overcome the issue in other ways than through clever preprocessing.

Secondly, it essentially means that e.g. words 'the' and 'Roberta' compete against each other, although one depends purely on the language skill (perhaps the next token after 'the' would be superlative) and the other one on the input sequence (which probably mentions some AI research).

Thirdly, as pointed out by e.g. [Gulcehre et al., 2016], (although not on this particular example) word 'Roberta' occurs less frequently in the training data than the word 'the', thus it is "difficult to learn a good representation of the word, which results in poor performance".

4.2 Attention

As stated by [Bahdanau et al., 2014] "The most important distinguishing feature of this approach from the basic encoder-decoder is that it does not attempt to

encode a whole input sequence into a single fixed-length vector". Therefore it solves the first problem (4.1.4) and as I will show later, I believe it partly solves the second one too.

The encoder is an recurrent neural network[3]. From the overall architecture (figure 4.3) we can see that only the last hidden state of the encoder is used, although there are encoder outputs for each time-step. [Bahdanau et al., 2014] proposes an architecture which takes advantage of this simple observation. In my work I use a little refinement, proposed by [Luong et al., 2015].[4]

Particularly, at time-step t, we compute the score vector.

$$s_{ti} = score(e_i, d_t) \tag{4.10}$$

where m is the length of the input sequence, e_i are the outputs of the encoder and d_t is the actual output of the decoder. Let us denote $\mathbf{s_t} = (s_{t1}, \ldots, s_{tm})$. According to [Bahdanau et al., 2014], the alignment vector $\mathbf{a_t}$

$$\mathbf{a_t} = softmax(\mathbf{s_t}) \tag{4.11}$$

"scores how well the inputs around position j and the output at position i match". The weighted sum of the outputs of the encoder, is callde a *context vector* for time-step t.

$$c_t = \sum_{i=1}^{m} a_{ti} e_i \tag{4.12}$$

The output of the decoder then also depends (unlike in the standard Encoder-Decoder architecutre) on the context vector.

$$att_t = tanh(W_c[c_t; d_t]) \tag{4.13}$$

$$p(o_t | o_{<t}, x) = softmax(W_o att_t) \tag{4.14}$$

4.2.1 Refinements of the Attention Mechanism

[Luong et al., 2015] experimented with three different types of score functions. We adopted two of them, the *dot* and *concat* ones. (The following equations are directly extracted from the Luong's paper)

$$score(e_i, d_t) = \begin{cases} e_i^\top d_t & dot \\ v_s^\top tanh(W_s[e_i; d_t]) & concat \end{cases}$$

The same author also states that the fact that the attentional decisions are made independently is suboptimal. Hence the *Input Feeding* approach is proposed to allow the model to take into acount its previous decisions. It simply means that the next input is the concatenation $[o_t, att_t]$.

[3]From now on, I'll stick to refer to *recurrent neural network* as to the neural network consisting of at least one *tf.keras.layers.RNN* or relatives.

[4]Since I don't experiment with the original Bahdanau attention, I only show the Luong's approach. [Luong et al., 2015] shows all the differences between his and Bahdanau's approach in section 3.1 of the paper.

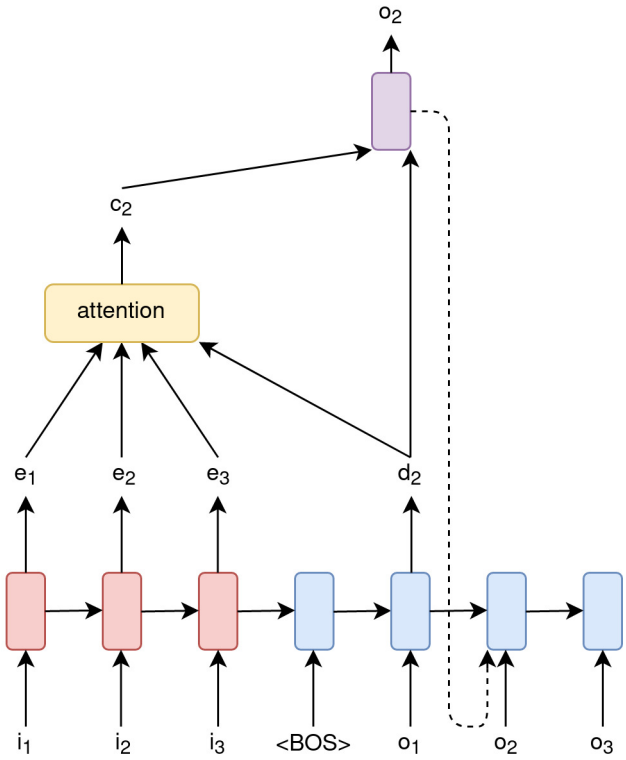


Figure 4.4: The Attention mechanism at the second time-step. Dotted line represents the input feeding approach.

4.3 Copy mechanism

The copy mechanism is a further extension of the attention mechanism. In this section I present the Pointer networks [Vinyals et al., 2015], which are trained to *point* to some position in the input sequence and the Copy Mechanisms ([Gulcehre et al., 2016], [Gu et al., 2016]) which model the decision making (whether to copy from the pointed location or to generate from the actual context).

4.4 Truncated Backpropagation Through Time

Why it is infeasible to generate sequences of average 350 tokens with full back-propagation. Which types of truncated BPTT exist and which I've chosen.

4.5 Beam search

Why greedy search isn't enough, what is beam search, when is it used.

4.6 Transformers

Right now I don't think I'll get this far in my exploration and implementation of DNN architectures.

5. Models

The main goal is to explore the architectures. Therefore each model is manually implemented making use of the tensorflow library

5.1 Tensorflow

What is it, mention the paper introducing it, mention other frameworks and motivation why this is the one which is used.

5.2 Sequence to Sequence architecture

Encoder, decoder. Encoder creates the representation of the input in some meta language, decoder creates the output from the representation.

5.3 Encoder

Mention the embedding and MLP encoding (the main approach used in the rotowire paper) - MLP is used instead of LSTM in the encoding process, then the 2 initial decoder states are obtained by mean pooling over the MLP encodings of the embedded source records.

5.4 Decoder

2 layer LSTM, embeddings, dimensionality.

5.5 Base Model

Seq2Seq architecture with attention, both Luong style Dot Attention and Bahdanau style Concat Attention are used, input feeding approach. Maybe some pictures.

5.6 Joint Copy model

Uses 2 attention mechanisms. Definitely some pictures.

6. Experiments

This chapter should present the observations about the generated data and the steps taken to improve the generations.

6.1 BLEU

What it is, why do I use such a metric for evaluating the data.

6.2 Manual evaluation

How the summaries for manual evaluation are chosen, how many people do evaluate the predicted summaries.

6.3 Other evaluation approaches

Which approaches are presented in the read papers, which improvements should be made.

6.4 Results of the baseline model

Learned which teams play, what are the greatest stars of each team, although the summaries diverge, only first few sentences from the generated summaries are relevant.

6.5 Dropout

What it is, where I apply the dropout - on the decoder LSTM cells.

6.6 Scheduled Sampling

What it is, how it solves the divergence of the summaries.

6.7 Copy methods

How do they help the model to choose more relevant data from the table, how do they fare in the concurrence of the baseline model.

Conclusion

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL `https://arxiv.org/abs/1409.0473`.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014. URL `https://arxiv.org/abs/1409.1259`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. Incorporating copying mechanism in sequence-to-sequence learning, 2016. URL `https://arxiv.org/abs/1603.06393`.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words, 2016. URL `https://arxiv.org/abs/1603.08148`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks, 2015. URL `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

Remi Lebret, David Grangier, and Michael Auli. Neural text generation from structured data with application to the biography domain, 2016. URL `https://arxiv.org/abs/1603.07771`.

Percy Liang, Michael Jordan, and Dan Klein. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 91–99, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P09-1011`.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. Table-to-text generation by structure-aware seq2seq learning, 2017. URL https://arxiv.org/abs/1711.09724.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015. URL https://arxiv.org/abs/1508.04025.

Christopher Olah. Understanding lstm networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Ratish Puduppully, Li Dong, and Mirella Lapata. Data-to-text generation with content selection and planning, 2019. URL https://arxiv.org/abs/1809.00582.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262010976.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016. URL https://arxiv.org/abs/1508.07909.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.

Lisa Tozzi. The great pretenders, 1999. URL http://weeklywire.com/ww/07-05-99/austin_xtra_feature2.html.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2015. URL https://arxiv.org/abs/1506.03134.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. Challenges in data-to-document generation, 2017. URL https://arxiv.org/abs/1707.08052.

List of Tables