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 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).

2.1.1 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.

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 streľby 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 githube tvorcov datasetu [1] Tabuľka 2.1 z validačného datasetu môže poslúžiť ako dobrý príklad.

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

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.

2.1.2 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 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` nesplňa 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

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

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

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

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.

2.1.3 Relation of summaries and tables

V tejto kapitole na jednoduchom príklade ukážem, ako sú štatistiky z tabuľky previazané so sumárom z RotoWire. Vo figure 2.1 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.

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 2.1: 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

To generate text from structured data I choose the Deep Neural Networks and specifically the Encoder-Decoder (ED) neural architecture (section 4). The ED suited to process sequential one dimensional data, however we cope with two dimensional tables. In this chapter I describe how I handle the problem. In addition I further analyze the data and propose cleaning methods and reason about motivations for selected techniques.

3.1 Transforming Tables to Records

At first let's define a table. A table is a two dimensional data structure, where the information is stored not only in the actual values in cells, but also in the positional information. Values in the same column have the same type, whereas values in the same row belong to the same entity. An example of the table as we have defined it is in figure 3.1.

	type_1	type_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

I use the same notation as [Liang et al., 2009]. Table $\mathcal{T}$ is transformed into a sequence of records $\mathbf{s} = \{r_i\}_{i=1}^{J}$, where r_i denotes i-th record. To fulfill our goal of keeping the most of the positional information from the table, each record contains field *r.type* denoting the type of the value, the actual value *r.value* and the entity *r.entity* to which the record belongs.

3.2 RotoWire

Firstly I present the statistics of the dataset *before any preprocessing*. Next I elaborate about the particularities of the input tables. After working with the dataset for a while I found many of its deficiencies. I'm not the first one who spotted the flaws and there exist datasets based on RotoWire, which contain cleaner data and summaries corresponding better to input tables [Wang, 2019], [Thomson et al., 2020]. However I choose to continue with the RotoWire dataset, as I am already accustomed to the format of the data.

3.2.1 Dataset Statistics

I believe that the challenges posed by the RotoWire dataset can be summarized in a set of statistics. In this subsection I want to present the most important ones to help reader to understand the nature of the problem.

Firstly, the target summaries as well as the sequences of input records are really long compared to other datasets modelling the same task (e.g. WikiBIO, WeatherGOV, RoboCup *XXXXXX*).

Secondly, many words occur rarely and the generation system cannot learn a good representation of them. It should be noted that the problem can be resolved by the means of clever preprocessing (section 3.2.3) or with help of advanced neural architectures (section 4.3).

Thirdly, there are many values which represent facts, e.g. values that cannot be deduced from the context (e.g. points a player scored in a match etc.) but must be selected and copied from the table.

The original dataset as prepared by [Wiseman et al., 2017], is already divided to train (3398 samples), development (727 samples) and test (728 samples) sets. *In the statistics presented below I state that there are only 3397 samples in the train set because one of the samples is the famous Lorem Ipsum template.*

Length-wise Statistics

The input tables contain huge amount of information. 2 teams and up to 30 players participate in a match of basketball. After transformation to a sequence, a player is represented by 24 and a team by 15 records. The type field *r.type* is the only trait distinguishing the team and player records. Table 3.2 summarizes the length statistics of the input sequences.

The generation system should be able to understand the meaning of a record and shouldn't rely on specific organisation of the table. This is modelled by emplacing the team records at the end, so that the system will need to search for the team statistics. Since the size of the input sequence isn't uniform, the team records can start anywhere between 500th and 720th record. The organization of the table will be further explained in the chapter about the experiments 6.

Set	Max Number of Records	Min Number of Records	Avegage Number of Records	Size
train	750	558	644.65	3397
development	702	582	644.66	727
test	702	558	645.03	728

Table 3.2: Statistics of tables as used by [Wiseman et al., 2017]

There is much greater variance in the lengths of output summaries. While the longest input sequence is 134% of the shortest one, from the data in table 3.3, we can see that the factor between the outputs is 511%. The size of the inputs and the outputs places high memory and computation demands on the GPUs used for training, and needs a special treatment (as will be explained in section 5.1.1).

Occurrences of Unique Tokens

While the length of the inputs and the outputs increases computational demands, the low occurrency of certain token causes the *rare word problem*. After discussions with my advisor I think it is reasonable to expect that the system should

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 3.3: Statistics of summaries as used by [Wiseman et al., 2017]

learn a good representation of a token if it appears at least 5 times in the train set.

There is about 11 300 unique tokens in the dataset (in the union of train, development and test set). In table 3.4 I present the statistics regarding the occurrences of the tokens. We can see that only 42 % of all the tokens appear at least 5 times in the train part of the dataset.

However I expect that even if some anomaly in the real word happens (e.g. team scores 200 points, although at the time of writing no team in the history of NBA scored more than 186) the system should be able to simply *copy* the value of *TEAM-PTS* record without reasoning about the actual value. Consequently I present also the statistics of tokens that cannot be copied from the table. At last, since most of the named entities can be directly copied, there is no need to keep casing as it doesn't represent any valuable information.

In the end we see that under our assumptions about 60 % of all the unique tokens cannot be learned by the generation system.

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 3.4: Occurrences of tokens in summaries from dataset RotoWire

In table 3.5 we can see how many of the unique tokens learned during training can be found in the respective development and test datasets. Under our assumptions we can expect the generated text to share less than 65 % of the vocabulary with the gold references.

3.2.2 Transformations of Input Tables

Firstly I want to talk about the preprocessing of actual values stored in the cells of the table. After that I present the format of a record which is fed to the generation system.

Set	Unique Tokens	Train Overlap	$\text{Train}_{>=5}$ 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: $\text{train}_{>=5}$ 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 3.4

Table 3.5: Overlap of train dataset summaries and valid/test dataset summaries

Tabular Types

There are 39 types (different headers of columns as discussed in section 3.1). A type is associated to textual or integer value which describes either a team or an individual. There are only 7 types bound to textual values (*TEAM-NAME*, *TEAM-CITY*, *FIRST_NAME**, *SECOND _NAME**, *PLAYER_NAME**, *START_POSITION**, *TEAM_CITY**)[1]

Numerical values

The other 33 types desribe absolute integer values (*TEAM-PTS*, *FTM*[2] ...) or relative integer values (*TEAM-FT_PCT*, *FT_PCT*[3], ...). During preprocessing no changes are made to any tabular numerical value[4].

Textual values

Regarding the textual values, I consider each as a single token. Since the names of teams are already one word long (with one exception needing a transformation *Trail Blazers* → *Trail_Blazers*) the transformation is rather trivial. The similar observation applies to names of cities (with 6 exceptions: *Oklahoma_City*, *San_Antonio*, *New_Orleans*, *Los_Angeles*, *Golden_State*, *New_York*), and start positions.

All the player names consist of at least 2 words (with one exception: *Nene*). As being discussed earlier, we want as many values from the table to be copiable as possible.

The authors of the dataset [Wiseman et al., 2017], as well as the authors of one of the more successful approaches to the dataset [Puduppully et al., 2019] make use of three special types, *PLAYER_NAME*, *FIRST_NAME* and *SEC-*

[1] * asterisk labels types bound to an individual

[2] Number of converted free throws by an individual, *"free throws made"*

[3] *player free throw percentage*

[4] As described in [Wiseman et al., 2017] all the relative values are converted to integers. I don't consider this as a preprocessing since only the converted values are available in the dataset.

OND_NAME which allow to distinguish if *James* refers to the first name of star player *James Harden* or to a second name of the legend of *LeBron James*.

My approach is based on the idea of making data as dense as possible to make the learning of the generation system easier. Therefore I transform the name of each player to a single token. To make the copying possible a preprocessing of the output summaries as described in 3.2.2 must take place.

Consequently *FIRST_NAME* and *SECOND_NAME* records aren't needed anymore which results in another benefit, shorter inputs. (as there are 22 - 30 players involved in the match, the size of inputs becomes about 10 % (44 - 60 records) shorter)

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 3.2.2 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

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

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 3.1: 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 3.1:

- **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*

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

3.2.3 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[5], ktorá sa dá stiahnuť aj ako samostatný python package. Minimálny príklad možnej implementácie v pythone je ukázaný ako algoritmus 1.

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

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[6]. 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%.[7]Výsledné štatistiky po všetkých transformáciách sú v tabuľkách 3.6 a 3.7.

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

Table 3.6: Occurrences of tokens in transformed summaries from dataset RotoWire

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

Table 3.7: Overlap of transformed train dataset summaries and valid/test dataset summaries

[6]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`

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

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

```
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.

Notation

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'*).

A Note About Embeddings

An embedding is a low dimensional continuous representation of discrete variables[1]. In this work I don't experiment with pretrained word embeddings, and I tune only the embedding-dimension hyperparameter, which adjusts the dimensionality of the trained continuous representation of the input.

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.

[1] I took the sentence from the accepted answer on stack overflow as it is the best description I have found. It is the first and the last time I "cite" from stack overflow. Source : `https://datascience.stackexchange.com/questions/53995/what-does-embedding-mean-in-machine-learning`

4.1.1 Recurrent Neural Network

Let $\mathbf{x} = (x^{(1)}, \dots, x^{(t)})$ 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.

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

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

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 visualized either as a loop, or as a feed-forward network with shared weights 4.1.

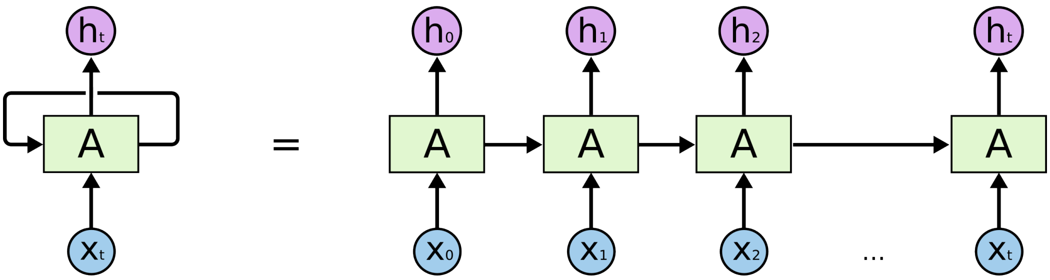


Figure 4.1: Possible visualizations of RNN [Olah, 2015]

The network is trained by back-propagation through time (BPPT) [Werbos, 1990]. It has been shown that the RNN suffers from vanishing / exploding gradient problems [Hochreiter and Schmidhuber, 1997].[2] 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[3] [Cho et al., 2014b] or Long Short-Term Memory [Hochreiter and Schmidhuber, 1997].

4.1.2 Long Short-Term Memory

The Long Short-Term Memory (*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

[2] 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.

[3] Although it is one of the most known architectures I used only LSTM for my experiments.

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].

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

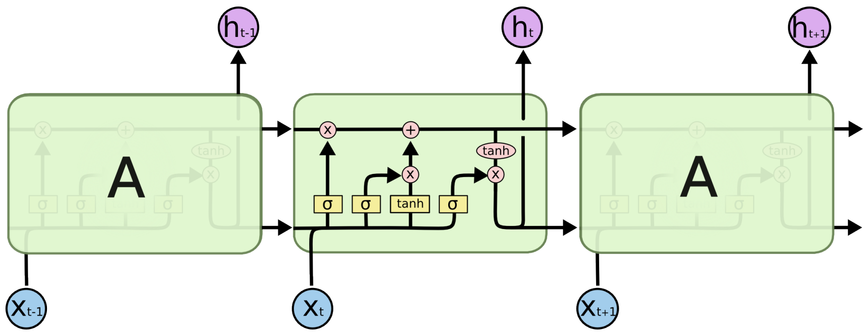


Figure 4.2: Visualization of LSTM [Olah, 2015]

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*[4]. 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.

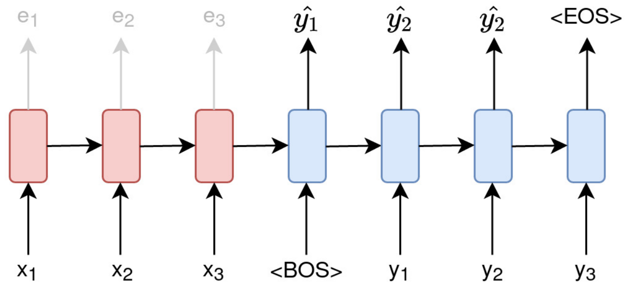
$$p(y_1, \ldots, y_n|x_1, \ldots, x_m) = \prod_{t=1}^{n} p(y_t|r, y_1, \ldots, y_{t-1}) \tag{4.9}$$

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)

[4]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.

As visualized in figure 4.3, 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.



Note: Encoder is red, Decoder is blue
none of the Encoder's outputs is used

Figure 4.3: Visualization of the training of the Encoder-Decoder Architecture with *teacher forcing*.

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.

Fixed-dimensional Representation of the Input Sequence

It has been shown by [Cho et al., 2014a] 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. Consequently this particular problem should be seen in our task.

Rare-word Problem and Hallucinations

In the standard Encoder-Decoder, the output is a distribution over the output vocabulary. At the start of the training each word is equally probable in any given context (assuming reasonable weights initialization). During training, model learns the language model on the training data. There are several flaws in the design:

1. 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).

2. 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"[5]

3. Networks tend to *hallucinate* the facts. E.g. from the record {*type:transformer; value:GPT*} network generates a sentence "The famous example of transformer architecture is BERT." To put it simply, the network knows it should talk about some transformer, therefore each word describing some kind of a transformer is somewhat probable, even if it wasn't seen in the actual input.

4. *The softmax function* which is used for calculation of the distribution is computationally expensive. This doesn't matter in the task over the RotoWire dataset, where we see only about 11 300 words[6]. However in WikiBIO task with more than 400 000 unique tokens it is simply unfeasible to compute the softmax.

I've already shown one way to overcome some of the issues in subsection 3.2.3. However in the section 4.3 I'll show how to get around these problems with a clever neural network architecture.

4.2 Attention Mechanism

At first I show the Attention mechanism which should cope with the issue of fixed-dimensional representation of the input sequence 4.1.4. 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".

The encoder is an recurrent neural network[7]. 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].[8]

[5]The problem is called *The Rare-Word Problem.*

[6]The number before any preprocessing.

[7]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.

[8]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.

4.2.1 Computation

The Attention mechanism calculates its outputs at each time-step. Let $\mathbf{s_t} = (s_{t,1}, \ldots, s_{t,m})$, which is called the *score vector*. Its elements are computed using *a score function* (which we will talk about below 4.2.2):

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

e_i are the outputs of the encoder and d_t is the actual output of the decoder. 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 called a *context vector* for time-step t.

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

Unlike in the standard Encoder-Decoder architecture, the output of the decoder also depends on the context vector.

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

$$p(y_t|y_{<t}, x) = softmax(W_y att_t) \tag{4.14}$$

4.2.2 Score Functions and Input Feeding

[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 $[y_t, att_t]$.

4.3 Copy mechanism

The copy mechanism is a further extension of the attention mechanism. In this section I discuss 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], [Yang et al., 2016] which model the decision making (whether to copy from the pointed location or to generate from the actual context).

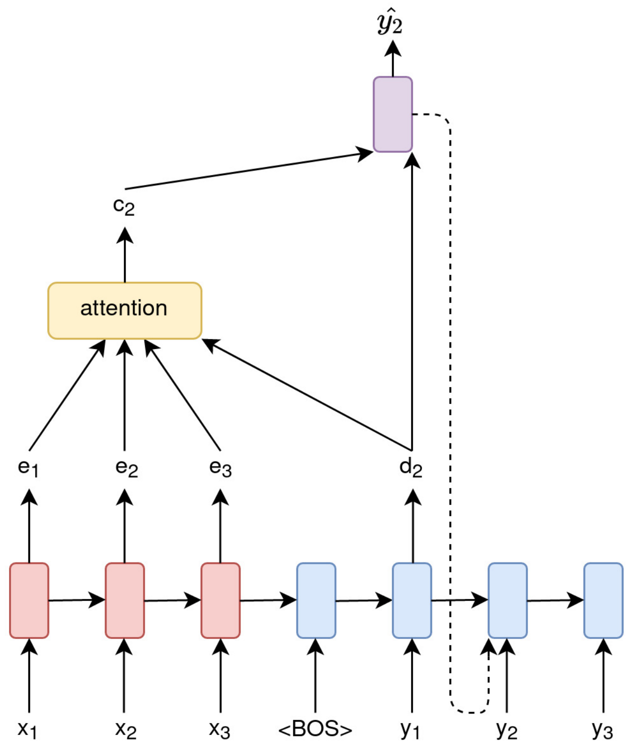


Figure 4.4: The Attention mechanism at the second time-step. Dotted line represents the input feeding approach.

4.3.1 Pointer Networks

The Pointer networks [Vinyals et al., 2015] leverage the fact that the alignment vector $\mathbf{a_t}$ can be seen as *a pointer* to the input sequence. Consequently instead of computing the weighted sum (*context vector*) and an MLP on top of that as the Attention models, they utilize the *alignment vector* as an output.

4.3.2 Approaches to Copying

[Gulcehre et al., 2016] notes that the ability to point is useless in the generation task if the network is always forced to point. Therefore they introduce a new switching network that outputs a binary variable z_t , which models the probability of the required action being pointing or generating. [Yang et al., 2016] propose similar approach but he marginalizes out the switch output z_t. Let $\mathbf{e} = (e_1, \ldots, e_m)$ be the encoder outputs, $\mathbf{s} = (r_1, \ldots, r_m)$ the sequence of input records, d_t be the decoder output at the actual time-step, and $ATTN$ be the Attention as presented in the previous section. The predicted word is calculated as follows:

$$\mathbf{a_t} = ATTN(\mathbf{e}, d_t) \tag{4.15}$$
$$p^{copy}(y_t|\mathbf{s}) = \mathbf{a_t} \tag{4.16}$$
$$\mathbf{c_t} = \sum_{i=1}^{m} e_i * a_{t,i} \tag{4.17}$$
$$p^{switch}(z_t|\mathbf{s}) = sigmoid(W_{switch}[\mathbf{c_t}, d_t]) \tag{4.18}$$
$$p^{gen}(y_t|\mathbf{s}) = softmax(W_{gen}[\mathbf{c_t}, d_t]) \tag{4.19}$$
$$p(y_t|\mathbf{s}) = p^{gen}(y_t|\mathbf{s})(1 - p^{switch}(1|\mathbf{s})) + p^{copy}(y_t|\mathbf{s})p(1|\mathbf{s}) \tag{4.20}$$

Since $p^{copy}(y_t)$ needs to be a distribution over the output vocabulary and in both tasks the output and the input vocabularies are the same, I compute the sum of *the one-hot encoded inputs*, weighted by the alignment vector.

$$p^{copy}(y_t) = \sum_{i=1}^{m} a_{t,i} * x_i \tag{4.21}$$

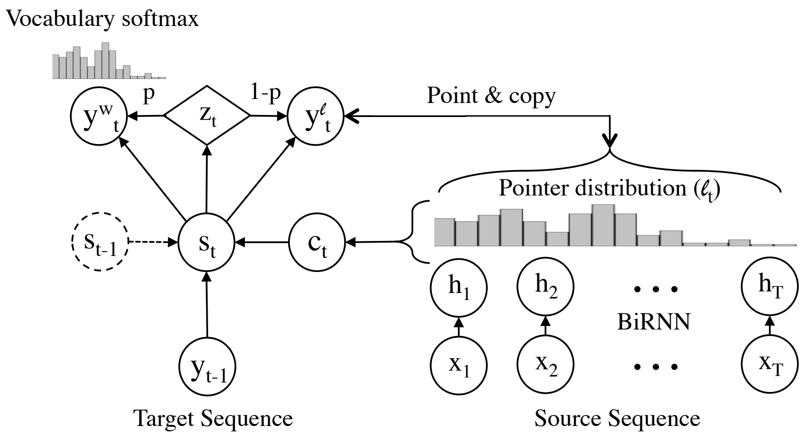


Figure 4.5: Excerpt from **Pointing the Unknown Words** paper by [Gulcehre et al., 2016], showing how the attention alignment can be utilized as a pointer information

5. Models

In this chapter I aim to clarify how I set up the neural network models for the experiments (which will be presented in the next chapter 6). Despite both datasets (WikiBIO, RotoWire) being created for modeling the same task (generation of text from structured data), I believe their characteristics are quite different (as shown in chapter about preprocessing 3). Therefore I present different set of models used for each dataset. At first I present a *baseline model*. Model, which is sufficiently competitive to be able to generate reasonable texts. Then I present changes to the architecture, which aim to improve the generation.

5.1 Rotowire

At first I try purely end-to-end approach by following the path set up by [Wiseman et al., 2017]. I create a baseline model, then improve it with the copy mechanism 4.3. Next I choose to divide the task to *content planning* and *text generation* as proposed by [Puduppully et al., 2019].

5.1.1 Truncated Backpropagation Through Time

It was really challenging to set up even the baseline model. Since the input sequences are about 600 records long and output sequences are more than 300 tokens in average (and the outputs are padded with $<PAD>$ token to approximately 800 tokens), it wasn't possible to fit the model into the GPU memory. (The GPUs at `https://aic.ufal.mff.cuni.cz` have about 8GBs of memory[1])

Computing and updating gradient for each time-step of a sequence of 800 tokens has approximately the same cost as forward and backward pass in a feed-forward network that has 800 layers. [Williams and Peng, 1998] proposes a less expensive method. Truncated Backpropagation Through Time (TBPTT) processes one time-step at a time, and every t_1 timesteps it runs BPTT for t_2 timesteps.

My implementation of the algorithm cuts the original sequence of size N to $(N - t_2)/t_1 + 1$ chunks (with possibly one more chunk if $(N - t_2)$ isn't divisible by t_1). Then I run *"normal"* BPTT on the first chunk, keeping the hidden state of the network after t_1 steps. Then I fed in the hidden state as the initial state at the start of processing second chunk. Similarly afterwards.

5.1.2 Baseline model

I use a non-recurrent Encoder and a two-layer LSTM decoder with Luong-style attention as the baseline model. At first I present how the encoding of input records works, then I talk about the text generation. I highlight all the differences between my approach and the one taken by [Wiseman et al., 2017]. The visualization of the model is shown in the figure 5.1.

[1]The exact GPU used on AIC cluster is NVIDIA GeForce GTX 1080

Encoder

The Encoder should process the input records created from the input table (section 3.1) to create the partial outputs at each time-step and the initial hidden state for the Decoder.

I described in section 3.2.2 that each record r consists of 4 different features: *type*, *value*, *entity* and a *home/away flag* depicting if the record belongs to home or away team.

At first each feature is embedded to a fixed-dimensional space. Next the embeddings are concatenated. I choose the same approach as [Wiseman et al., 2017] (who was inspired by [Yang et al., 2016]), and I pass the concatenation through a one layer feed-forward network (*tf.keras.layers.Dense*) with ReLU activation[2], to create the encoding e of the record r. Consequently, the input sequence of records $\{r_i\}_{i=1}^m$ is transformed to $\mathbf{e} = \{e_i\}_{i=1}^m$.

To create the initial state of the decoder, [Wiseman et al., 2017] calculated the mean of the records belonging to the particular entity and linearly transformed the concatenation of the means. (Simply, the concatenation is passed through *tf.keras.layers.Dense* without any activation function).

To make the implementation simpler I observe that each player entity is represented by 22 records and each team entity by 15 records 3.2.2. Consequently I approximate the approach taken by [Wiseman et al., 2017] and mean pool over $\mathbf{e}$ with stride 22. (Which means that while means of players are exact, the team records are less than approximated) During my experiments I haven't seen any indication that this modification became the performance bottleneck of the model.

Decoder

The Decoder is a 2-layer LSTM network with attention mechanism. The LSTMs are initialized with states prepared in the previous section. I opted to use the Luong style attention with input feeding [Luong et al., 2015]. I described in section 4.2.2 that we use the concatenation of the last attentional decision and the last gold output as the actual input to the first layer of LSTMs. However at the first time-step, when the input is the *<BOS>* token, there is no *last attentional decision*. Hence for this purpose I use one of the initial states prepared by the Encoder.

Training and Inference

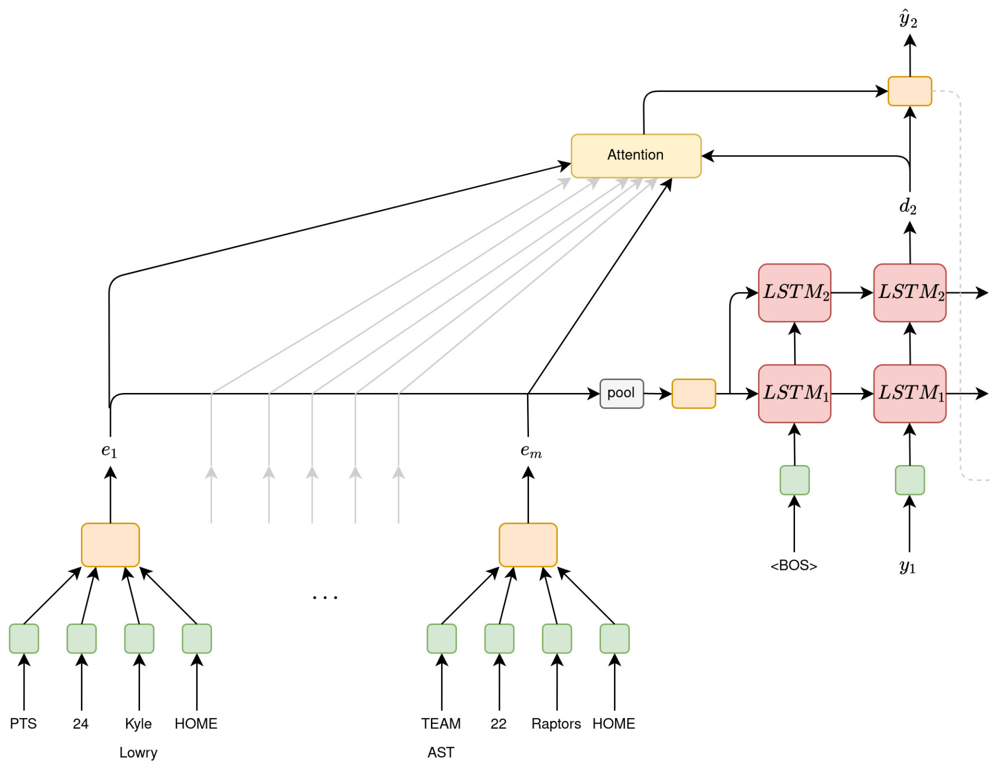
The model is trained end-to-end to minimize the cross-entropy loss (negative-log likelihood) of the gold output summary $\mathbf{y}$ given the sequence of input records $\mathbf{s}$.

$$\min \sum_{(\mathbf{s},\mathbf{y})\in\mathcal{D}} -log\ p(\mathbf{y}|\mathbf{s}) \tag{5.1}$$

We use the *teacher forcing* (as explained in section 4.1.3) during training and during the inference, we greedily approximate

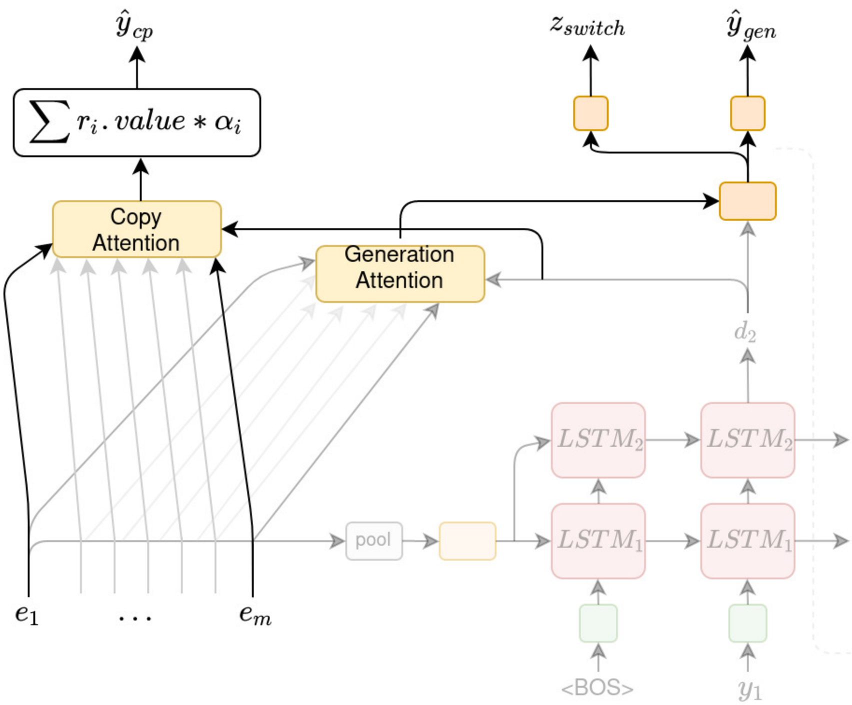
$$\arg\max_{\hat{\mathbf{y}}} p(\hat{\mathbf{y}}|\mathbf{s}) \tag{5.2}$$

[2]the Rectified Linear Unit activation function $f(x) = max(0, x)$



Note: green cells are embedding layers, orange cells are feed-forward networks

Figure 5.1: The Rotowire baseline model at the second time-step.



Note: green cells are embedding layers, orange cells are feed-forward networks
α is the alignment vector produced by the copy attention; r is the input record.

Figure 5.2: The Joint-Copy extension of the baseline model at second time-step.

5.1.3 Improving the Baseline with Copy Mechanisms

In the previous chapter 4.1.4, I underlined *hallucinations* as one of the problems of the Encoder-Decoder architecture. In the task on RotoWire dataset it basically means that during training, the model learns that it should put *some number* at a specific place in the sentence. We would like it to generate the *exact value*.

Therefore as the next step I incorporate the *Joint Copy mechanism* [Gu et al., 2016] (already described in 4.3). The encoder rests intact, however in the decoder I use another attention module to point to specific record from the input table and a feed-forward network which models if the model should generate or copy.

The generational path is the same as in the baseline model. In the copy path I use the alignment vector from the newly added attention as weights, to compute the weighted sum of the value portion of input records. The visualization can be seen in the figure 5.2.

5.1.4 Content Selection and Planning

Rather than training end-to-end, [Puduppully et al., 2019] suggest to divide the task to *content planning* and *text generation*. During the *content planning* stage the model selects some records from the input table and organizes them to a *content plan*. The *content plan* describes ”what to say, and in what order”. During the *text generation* stage, the model generates the text based on the records pointed to by the *content plan*.

Both tasks are modelled with *the same* Encoder and with *separate* Decoders.

Content Selection

[Puduppully et al., 2019] improves the baseline encoder by incorporating *context awareness*. At first the input records are encoded in the same way as in the baseline model 5.1.2, the self-attention is used to model the ”importance vis-a-vis other records in the table”.

Specifically, we compute

$$
\begin{aligned}
\forall k \neq t : \alpha_{t,k} &= score(\hat{r}_t, \hat{r}_k) && \textit{the score vector} \\
\boldsymbol{\beta_t} &= softmax(\boldsymbol{\alpha_t}) && \textit{the alignment vector} \\
\boldsymbol{\gamma_t} &= \sum_{i=1}^{m} \beta_{t,i} * \hat{r}_i && \textit{the context vector} \\
\hat{r}_t^{att} &= sigmoid(W_{cs}[\hat{r}_t, \boldsymbol{\gamma_t}]) && \textit{gating mechanism} \\
\hat{r}_t^{cs} &= \hat{r}_t^{att} \bigodot \hat{r}_t && \textit{the content selected representation}
\end{aligned}
$$

The Encoder thus creates a sequence of *context aware* representations $\{\hat{r}_t^{cs}\}_{t=1}^{m}$ (figure 5.3).

Content Planning

[Wiseman et al., 2017] experimented with conditional copy approach, in which the latent *switch* probability isn’t marginalized out. Hence, there exists pointer sequence for each summary in the train and validation dataset. The sequence corresponds to the order in which entities and values from the sequence of input

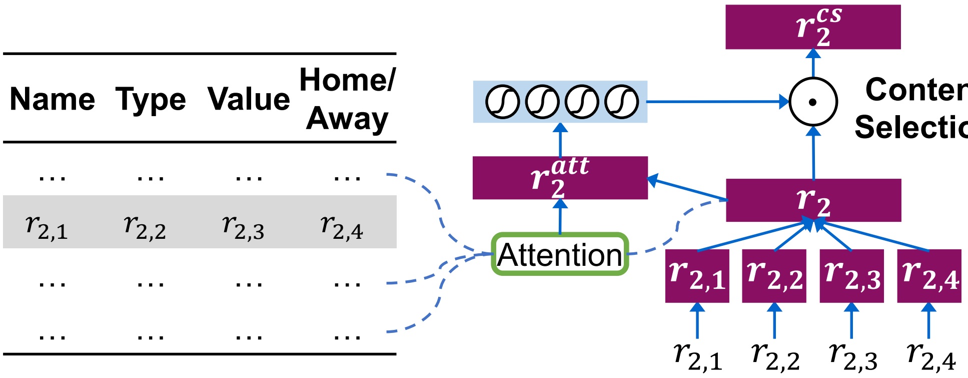


Figure 5.3: Content Selection mechanism (image is directly from [Puduppully et al., 2019])

Type	**Entity**	**Value**	**H/A flag**
<<BOS>>	<<BOS>>	<<BOS>>	<<BOS>>
TEAM-CITY	Raptors	Toronto	HOME
TEAM-NAME	Raptors	Raptors	HOME
TEAM-PTS	Raptors	122	HOME
TEAM-CITY	76ers	Philadelphia	AWAY
TEAM-NAME	76ers	76ers	AWAY
TEAM-NAME	76ers	76ers	AWAY
TEAM-PTS	76ers	95	AWAY
...	...	...	...

Note: The extract corresponds to sentence: ”The host Toronto Raptors defeated the Philadelphia 76ers , 122 - 95. . . ”

Table 5.1: An extract from the content plan corresponding to the summary from the figure 2.1

records appear in the output summary. [Puduppully et al., 2019] suggested that instead of just modelling the switch probability, we can train a Decoder, which extracts these pointers from the original table.

As suggested, the Decoder is a one layer LSTM which operates on the *context aware* representations $\hat{\mathbf{r}}^{cs}$. Its hidden states are initialized with $avg(\{\hat{r}_t^{cs}\}_{t=1}^{m})$. [Puduppully et al., 2019] haven't elaborated on the exact approach, hence it is probable that I have diverged a little bit from the intentions of original authors.

The Decoder from the baseline model is trained to start the generation when it sees the $<BOS>$ token. Since the Content Plan Decoder operates on $\hat{\mathbf{r}}^{cs}$ I've choosen to prepend a special $<BOS>$ record to the sequence of input records, and also a pointer to the $<BOS>$ record is prepended to each content plan. Therefore instead of teaching the Content Plan Decoder to start generating content plan when seeing a special value, I teach it to do so when seeing *the encoded representation of the special value*. The same approach is taken at the end, with the $<EOS>$ record.

Either baseline Decoder or Joint-Copy Decoder can operate on the generated content plan, to generate the output summary.

5.1.5 Training and Inference

As oposed to Baseline and Joint-Copy models, the Content Planning model has multiple outputs. Therefore we train the model to minimize the joint negative log-likelihood of the gold summary $\mathbf{y}$ and the gold content plan $\mathbf{z}$ conditioned on the sequence of input records $\mathbf{s}$. It deserves a note that we use the gold content plans as the inputs to the text generation part of the model.

$$\arg\min_{\mathbf{y},\mathbf{z}} p(\mathbf{y}|\mathbf{z},\mathbf{s}) + p(\mathbf{z}|\mathbf{s}) \tag{5.3}$$

6. Experiments

Firstly I would like to elaborate on the selection of the hyperparameters. Then I present few of the generated biographies (based on the input infoboxes from WikiBIO dataset) and some generated summaries (based on the input tables from RotoWire dataset).

6.1 Development Methods

At first I want to talk about the methods taken to find the best hyperparameter settings during the development of models.

To avoid the famous underfitting and overfitting problems (more in section 5.2 in [Goodfellow et al., 2016]), I use the validation (development) part of the respective dataset

6.2 BLEU

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

6.3 Manual evaluation

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

6.4 Other evaluation approaches

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

6.5 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.

6.6 Dropout

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

6.7 Scheduled Sampling

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

6.8 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, 2014a. URL `https://arxiv.org/abs/1409.1259`.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014b. URL `https://arxiv.org/abs/1406.1078`.

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/`.

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`.

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.

Craig Thomson, Ehud Reiter, and Somayajulu Sripada. Sportsett: Basketball - a robust and maintainable dataset for natural language generation. August 2020. URL https://intellang.github.io/. IntelLanG : Intelligent Information Processing and Natural Language Generation ; Conference date: 07-09-2020 Through 07-09-2020.

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.

Hongmin Wang. Revisiting challenges in data-to-text generation with fact grounding. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 311–322, Tokyo, Japan, October–November 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-8639. URL https://www.aclweb.org/anthology/W19-8639.

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.

Ronald Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2, 09 1998. doi: 10.1162/neco.1990.2.4.490.

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

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. Reference-aware language models, 2016. URL https://arxiv.org/abs/1611.01628.

List of Tables