



**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: In this thesis we examine ways of generating document-scale natural language text conditioned on structured input data. Specifically we train Deep Neural Network models on recently introduced RotoWire dataset containing statistical data about basketball matches paired with descriptive summaries. At first we analyse the dataset and propose several preprocessing methods (i.e. Byte Pair Encoding). Next we train a baseline model based on the Encoder-Decoder architecture on the preprocessed dataset. We discuss several problems of the baseline and explore advanced Deep Neural Network architectures that aim to solve them (Copy attention, Content Selection, Content Planning). We hypothesize that our models are not able to learn the structure of the input data and we propose a method reducing its complexity. Our best model trained on the simplified data manages to outperform the baseline by more than 5 BLEU points.

Keywords: text generation structured data natural language processing neural networks

Contents

| | |
|---|-----------|
| Introduction | 3 |
| 1 Data | 4 |
| 1.1 Data requirements | 4 |
| 1.2 Summaries | 4 |
| 1.2.1 Fantasy sports | 4 |
| 1.2.2 Fantasy sports news | 5 |
| 1.3 Structured Data | 5 |
| 1.3.1 Team Statistics | 5 |
| 1.3.2 Player statistics | 6 |
| 1.4 Relation of summaries and tables | 6 |
| 2 Preprocessing and Statistics of the Dataset | 9 |
| 2.1 Transforming Tables to Records | 9 |
| 2.2 Dataset Statistics | 10 |
| 2.2.1 Length-wise Statistics | 10 |
| 2.2.2 Occurrences of Unique Tokens | 10 |
| 2.3 Transformations of Input Tables | 11 |
| 2.3.1 Tabular Types | 12 |
| 2.3.2 Numerical values | 12 |
| 2.3.3 Textual values | 12 |
| 2.3.4 Entities | 13 |
| 2.3.5 Record Format | 13 |
| 2.3.6 Order of Records | 13 |
| 2.4 Preprocessing of Summaries | 14 |
| 2.4.1 Number Transformations | 14 |
| 2.4.2 Player name transformations | 15 |
| 2.5 Vocabularies | 16 |
| 2.6 Byte Pair Encoding | 16 |
| 2.6.1 Algorithm | 17 |
| 2.6.2 Application | 17 |
| 2.7 Cleaning | 18 |
| 2.8 Statistics of Transformed Dataset | 18 |
| 3 Neural Network Architectures | 21 |
| 3.1 The Encoder-Decoder Architecture | 21 |
| 3.1.1 Recurrent Neural Network | 21 |
| 3.1.2 Long Short-Term Memory | 22 |
| 3.1.3 High-level Overview of Encoder-Decoder Architecture | 23 |
| 3.1.4 Problems of the Encoder-Decoder Architecture | 26 |
| 3.2 Attention Mechanism | 27 |
| 3.2.1 Computation | 27 |
| 3.2.2 Score Functions and Input Feeding | 27 |
| 3.3 Copy mechanism | 28 |
| 3.3.1 Pointer Networks | 28 |

| | | |
|----------|---|-----------|
| 3.3.2 | Approaches to Copying | 29 |
| 4 | Experimental Setup | 31 |
| 4.1 | Truncated Backpropagation Through Time | 31 |
| 4.2 | Baseline model | 31 |
| 4.2.1 | Encoder | 31 |
| 4.2.2 | Decoder | 32 |
| 4.2.3 | Training | 32 |
| 4.2.4 | Regularization | 33 |
| 4.3 | Improving the Baseline with Copy Mechanisms | 33 |
| 4.4 | Content Selection and Planning | 34 |
| 4.4.1 | Content Selection | 34 |
| 4.4.2 | Content Planning | 35 |
| 4.4.3 | Training and Inference | 37 |
| 4.5 | Postprocessing | 37 |
| 5 | Experiments | 39 |
| 5.1 | Evaluation Methods | 39 |
| 5.1.1 | Manual Evaluation | 39 |
| 5.1.2 | Ommited Evaluation Methods | 39 |
| 5.2 | Baseline Model | 40 |
| 5.2.1 | Results | 40 |
| 5.3 | Joint-Copy Model | 41 |
| 5.3.1 | Results | 42 |
| 5.4 | Content Selection Encoder with Joint-Copy Decoder | 43 |
| 5.4.1 | Results | 43 |
| 5.5 | Content Selection and Planning | 44 |
| 5.5.1 | Results | 44 |
| 5.6 | Ordered Tables | 46 |
| 5.6.1 | Results | 46 |
| 5.7 | Shortened Tables | 47 |
| 5.7.1 | Results | 47 |
| 5.8 | Overall Comparison | 48 |
| | Conclusion | 50 |
| | Bibliography | 51 |
| | List of Tables | 54 |

Introduction

Language Modeling is the task of predicting what word comes next in the actual context. In this thesis we delve into a slightly more complicated task, into the Conditional Language Modeling, where the next generated word does not only depend on the context but also on some external factors.

Specifically, given tables of team and individual statistics (e.g. number of points scored by a team, number of minutes played by a player etc.) we aim to generate a document-scale text that is coherent (e.g. without contradictions), written in good English, and that contains a lot of factual data based on the input tables. We train multiple Deep Neural Network models on the RotoWire dataset [Wiseman et al., 2017]. It consists of the statistics of National Basketball Association matches from 2014–2017 and the corresponding professionally written summaries from the fantasy sports news site <https://www.rotowire.com/basketball/>.

The statistics as well as the manual analysis of the dataset show numerous discrepancies (chapter 2). We elaborate on various preprocessing and cleaning methods that should make the learning path of our models easier.

We explore the Encoder-Decoder architecture [Sutskever et al., 2014] along with many enhancements such as attention (section 3.2), copy attention (section 3.3), and content selection and planning (section 4.4).

However this kind of architecture is aimed at processing sequential input while the structured data from the RotoWire dataset is in form of tables. Therefore following Wiseman et al. [2017] we show the methods that keep most of the valuable positional information while transforming tabular data into a sequence of records. We also discuss the neural network architectures that learn what kind of information is stored in a particular record (section 4.2.1), and which records are complementary to each other (section 4.4.1).

We hypothesize that the biggest performance bottleneck of our approach is the complexity of the input tables. Therefore we propose two methods which aim to reduce the complexity.

The quality of generated texts is measured using the BLEU score, and a set of manual metrics (Entity Recall measures how many of the entities mentioned in the gold summaries are also in the generated ones, while Factual Correctness measures how many of the generated numerical facts are grounded on the statistics from the input tabular data). We show that enhancements of the Encoder-Decoder architecture as well as the reduction of complexity of the input tables help the performance of our models.

All the models and preprocessing methods are developed in *python 3.8* and *tensorflow 2.4.1*. However the code is compatible with *python 3.6* and *tensorflow 2.3* (the versions used on Artificial Intelligence Cluster (AIC) where the training was executed).

1. Data

The goal of the thesis is to explore the possibilities of generating natural language representation of a sports match. The differences in the number of players, the environment, the rules etc. prevent us from creating one system for all the sports. In the introduction we stated that we narrowed the task to one particular sport, the basketball.

In this chapter we explain our requirements on the output summaries associated with the structured data, reason why we choose the RotoWire dataset, and elaborate on the format of the input tables.

1.1 Data requirements

We have high demands on the quality of the texts which are used as targets during training of the Deep Neural Network models. The summaries should be connected to statistical data about the match, and contain the least possible amount of subjective, emotional information. Otherwise the model wouldn't have possibility to learn how to interpret its inputs, and instead it would learn how to generate some invaluable emotional phrases.

It is known that a lot of data is needed to train a deep neural network (e.g. Sennrich et al. [2016] trains the neural machine translation system on 4.2 million English-German sequence pairs). Therefore it's unfeasible to collect and prepare our own dataset, and we choose to select from a high variety of publicly available datasets.

We choose the RotoWire dataset [Wiseman et al., 2017]. It contains a set of statistics of NBA matches associated with summaries collected from the portal focused on fantasy sport news, <https://www.rotowire.com/basketball/>.

1.2 Summaries

In section 1.1 we have already discussed our demands on the summaries which will be used as targets.

Wiseman et al. [2017] experimented with one other summary origin, <https://www.sbnation.com/nba>, where articles are written by fans for fans. However such summaries do not meet our requirements (Wiseman et al. [2017] states that "many documents in the dataset focus on information not in the box- and line-scores") and the neural networks they trained on these kind of summaries "performed poorly".

Therefore we experiment only with the summaries from Rotowire fantasy sports news. In the following subsections we explain what is fantasy sport and why the summaries collected from Rotowire meet our expectations.

1.2.1 Fantasy sports

According to [Tozzi, 1999], the origins of the phenomenon of fantasy sports can be dated to the beginning of 1960s. The article tells a legend about a restaurant

called La Rotisserie Francaise. There a group of "sportswriters, accountants and lawyers" started the first fantasy sport league. Rotowire, which name is derived from the name of the restaurant, is the source of all the target summaries in the dataset.

Let's proceed and discuss the rules of the fantasy sport according to Rotowire¹. The fantasy league is created on top of a real world league, e.g. NBA. If you want to play, you choose a subset of NBA players and gain points based on their performance in the real-world NBA matches. You have limited resources and the better players cost more. The selection must contain a basketballer for each position (therefore it is not possible to draft e.g. 5 point guards and no power forward). The points are awarded according to the role, so that defensive and offensive players gain you points for different achievements in a match. The leagues differ in their specifics and we advise an interested reader to check the Rotowire fantasy league rules¹.

1.2.2 Fantasy sports news

To succeed in the fantasy sport, one must keep a good track of the player statistics and injuries, the tendencies of team performances, the trends causing the emergence of a future star or a burn-out. From the beginning there exists a form of news specializing directly on fantasy league players. <https://www.rotowire.com/> is one of the most popular ones. The articles tend to summarize the most important statistics as well as present a deeper understanding of the events that happened during the match. As noted by Wiseman et al. [2017], the articles are the ideal candidates for the target summaries of the structured data from the dataset.

1.3 Structured Data

The structured data is in form of multiple tables of statistics related to a particular NBA match. In this section we examine the tables.

To begin the examination, each datapoint contains a date when the match was played. This information isn't leveraged in the summaries², therefore we opt not to use it.

1.3.1 Team Statistics

A datapoint contains a separate table with statistics, called the *line score*, for each of the opposing teams. Each *line score* contains 15 fields, some of which we will discuss in this section. The full listing can be found on the official github of the dataset ³

Firstly there are fields with the team name and the city where the team is located.

¹<https://www.rotowire.com/basketball/advice/>

²This isn't technically true, as the majority of summaries contain a phrase like "*Team A defeated Team B on monday.*", however since the date is in form *DD/MM/YYYY*, there isn't any chance the network could deduce the day of the week from the date.

³<https://github.com/harvardnlp/boxscore-data>

Next there is a set of numerical statistics. We can divide them to the contextual statistics (the number of wins and losses prior to the actual match), the team-totals (*Team-total* means the overall team statistics. E.g. *TEAM-PTS* is the sum of all the points scored by the players of the team), and the per-period point statistics (*TEAM-PTS_QTR1* - *TEAM-PTS_QTR4*). Table 1.1 shows a portion of both *line scores* present in the first data-point from the development part of the RotoWire dataset.

| 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 1.1: An example of the team statistics from the development part of the Rotowire dataset

1.3.2 Player statistics

The player statistics are gathered in a table called *box score*. In each row there is an information about the player name, the city he plays for (since there are 2 teams originating in Los Angeles, the distinction is made by calling one *LA* and the other one "*Los Angeles*"), his position on the starting roster (or the *N/A* value if he isn't in the starting lineup), and another 19 fields with statistics summing up the impact of the player in the match (e.g. points, assists, minutes played etc.). Again, the full listing can be found on github³ and table 1.2 contains a portion of a *box score* from the first data-point from the development part of the RotoWire dataset.

| 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 1.2: An example of the player statistics from the development part of the Rotowire dataset

1.4 Relation of summaries and tables

In this section we would like to show the relationship between the structured data and the target summary. Let's observe the data-point from the development

part of the dataset in figure 1.1. The non-highlighted text shows one-to-one correspondence with the structured data. The **blue-highlighted text** marks the information which is present in the input data only implicitly. However the level of implicitness varies. It is relatively easy to see that since Terrence Ross's starting position is "N/A", he must have started off the bench. On the other hand the fact that "The Raptors came into this game as a monster favorite" requires comparison of the winning-loosing records of both teams. The **yellow-highlighted text** labels the information that isn't deducible from the input data and the network would have to learn to hallucinate to be able to generate such text.

Somebody may argue that the observations do not sound favorable for the dataset. We see it as a big challenge and a possibility to apply advanced preprocessing as well as interesting architectural design of the neural networks. We will further elaborate on the issue of too noisy data in section 2.7.

| 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 1.1: A data-point from the development part of the Rotowire dataset. Yellow-highlighted text isn't based on the input data. Blue-highlighted text implicitly follows from the input data.

2. Preprocessing and Statistics of the Dataset

To generate text from structured data we choose the Deep Neural Networks and specifically the Encoder-Decoder (ED) neural architecture (chapter 3). The ED is suited to process sequential one dimensional data, however we cope with two dimensional tables. In this chapter we present the statistics of the dataset *before any preprocessing*. Next we elaborate about the methods of preprocessing and cleaning of the data. In the end we show the statistics of the dataset with all the transformations applied.

2.1 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 a table as we have defined it is in figure 2.1.

| | type ₁ | type ₂ ... |
|---------------------|----------------------|--------------------------|
| entity ₁ | value _{1,1} | value _{1,2} ... |
| entity ₂ | value _{2,1} | value _{2,2} ... |
| ... | | |

Table 2.1: An example of structured data

We use the same notation as Liang et al. [2009]. Table \mathcal{T} is transformed into a sequence of records $\mathbf{x} = \{r_i\}_{i=1}^J$, 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. At the end, we transform table 2.1 to a sequence of records shown in figure 2.1.

$\{type: type_1; entity: entity_1, value: value_{1,1}\}$

$\{type: type_2; entity: entity_1, value: value_{1,2}\}$

$\{type: type_1; entity: entity_2, value: value_{2,1}\}$

...

Figure 2.1: An example of records obtained by transforming table 1.1

2.2 Dataset Statistics

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

Firstly, Wiseman et al. [2017] notes that the target summaries as well as the sequences of input records are really long compared to other datasets modelling the data-to-text task (e.g. WikiBio [Lebret et al., 2016] contains 13-times smaller gold summaries and 32-times smaller input tables, Wiseman shows a comparison to some other datasets as well).

Secondly, many words occur rarely and the generation system cannot learn a good representation of them (it is known as a *rare word problem*). It should be noted that the problem can be resolved by the means of clever preprocessing (section 2.6) or with help of advanced neural architectures (section 3.3).

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 we state that there are only 3397 samples in the train set because one of the samples is the famous Lorem Ipsum template.*

2.2.1 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 2.2 summarizes the length statistics of the input sequences.

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

Table 2.2: Statistics of tables as used by Wiseman et al. [2017]

There is much greater variance in the lengths of output summaries. The longest sequence of input records is 1.34 - times longer than the shortest one, while the factor between longest and shortest summaries is more than 5. The size of inputs and outputs places high memory and computation demands on the GPUs used for training, and needs a special treatment (as will be explained in section 4.1).

2.2.2 Occurrences of Unique Tokens

While the length of the inputs and the outputs increases computational demands, another common issue is the *rare word problem*. If a token appears only sporadi-

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

Table 2.3: Statistics of summaries as used by Wiseman et al. [2017]

cally in the train data, the generation system can’t recognize how to use it. After discussions with my advisor we think it is reasonable to expect that the system should 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 2.4 We present the statistics regarding the occurrences of the unique tokens. We can see that only 42 % of all the unique tokens appear at least 5 times in the train part of the dataset.

However we 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 we are interested in tokens that cannot be copied from the table. Since most of the named entities are directly copiable, there is no need to preserve casing. All the aforementioned statistics are summarized in table 2.4.

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 2.4: Occurrences of tokens in summaries from dataset RotoWire

In table 2.5 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.

2.3 Transformations of Input Tables

Firstly we want to present what kind of data is stored in the input tables, and how it is preprocessed. After that we show the final format of a record which is fed to the generation system.

| 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} overlap is a set of all the tokens from the development/test dataset with more than 5 occurrences in the train dataset summaries _{1, 2, 3, 4} have the same meaning as in table 2.4

Table 2.5: Overlap of train dataset summaries and valid/test dataset summaries

2.3.1 Tabular Types

There are 39 types (different headers of columns as discussed in section 2.1). 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, out of which 2 are related to teams (*TEAM-NAME*, *TEAM-CITY*) and 5 to individuals (*FIRST-NAME**, *SECOND-NAME**, *PLAYER-NAME**, *START-POSITION**, *TEAM-CITY**)

2.3.2 Numerical values

The other 32 types describe absolute (*TEAM-PTS*¹, *FTM*² ...) or relative integer values (*TEAM-FT-PCT*, *FT-PCT*³, ...). During preprocessing no changes are made to any tabular numerical value⁴.

2.3.3 Textual values

Regarding the textual values, we 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.

Out of three types connected to player credentials, only *PLAYER-NAME* (describing player full name with all the attributes e.g. *Johnny O'Bryant III*) is multi-token.

The original take on the problem is different to ours. The authors of the dataset [Wiseman et al., 2017], as well as the authors of one of the more successful approaches to the task [Puduppully et al., 2019] make use of three special types, *PLAYER-NAME*, *FIRST-NAME* and *SECOND-NAME* which allow to

¹Team Points

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

³team/player free throw percentage

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

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

Our approach is based on the idea that if only one token is associated to an entity then the generation system has easier job in learning what to copy. Therefore we transform the name of each player to a single token. To make copying possible a preprocessing of the output summaries as described in 2.4.2 must take place.

Consequently *FIRST_NAME* and *SECOND_NAME* records aren't needed anymore. Number of records belonging to a particular player is thus reduced from 24 to 22 and the overall number of records which describe one match is decreased by 8% (the maximal length of a table decreases from 750 to 690).

2.3.4 Entities

The type information tells us what the number or text in the value field represents. However it is the entity field (the row in table 2.1) which brings together all the records describing the same player or team. Let's show an example of records about a star player, *Stephen Curry*. His name is stored in a record of type *PLAYER_NAME*, and value *Stephen_Curry*. To link all the information about him together, each record has an entity field labelled *Stephen_Curry*. Similarly all the records connected to a team with name **A** have the same entity field, **A**.

At last, we should notice that the overall team information is the union of the accumulated team stats (e.g. the number of points scored by all the players of a team) and the collection of statistics of the individuals playing for the team. Therefore the record also contains *HOME/AWAY* field which brings together all the statistics about the home side and the away side.

2.3.5 Record Format

The records fed into the generation system contain the following fields:

- *Type*
- *Value*
- *Entity*
- *Home/Away flag*

2.3.6 Order of Records

The generation system should be able to understand the meaning of a record and shouldn't rely on a specific organization 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.

However during experiments we found out that the models we train have problems with extraction of information from the input table. Therefore we prepared two additional schemes of organization of the input records.

Ordered records

The first scheme is rather simple. We place the 15 home team records at the beginning of the ordered sequence followed by 15 away team records. The remaining N sequences of 22 records related to players are ordered according to their points-total in the corresponding match.

Shortened records

The second scheme leverages an observation about the output summaries. We can see that only few best players are thoroughly discussed and the information about the remaining ones is reduced to mentioning their point totals.

Therefore we order the records in the same way as previously. Only first 10 players according to their point-totals are mentioned. The information about top three players is reduced from 22 to 21 records (we cut off the information about starting position) and the information about the remaining players is reduced from 22 to 5 records (minutes they played, points and assists they scored, their team and name). The maximal size of the newly constructed sequence is 130 records (compared to 690, the size of table after transformations described in section 2.3.3).

$\{type: PTS; entity: Stephen_Curry, value: 25; ha: HOME \}$

$\{type: TEAM-PTS; entity: Warriors, value: 122; ha: HOME\}$

...

Figure 2.2: An example of a player and a team record.

2.4 Preprocessing of Summaries

We would like to reiterate that our motivation is to avoid the *rare word problem* and to make copying words from the sequences of input records as easy as possible. Therefore we opt for methods which reduce the number of tokens, increase their average frequency (because the system couldn't learn the most sporadic ones anyway), and transform the tokens describing the tabular data to the same form as is used in the table (so copying is trivial).

2.4.1 Number Transformations

Just as Wiseman et al. [2017] and Puduppully et al. [2019], we represent the numbers only by numerals. This preprocessing method partially fulfills both of our goals. Obviously it decreases the unique token count, but on top of that it makes copying easier. E. g. the sentence "*Isaiah Thomas once again excelled , scoring 23 points, **three** assists and **three** rebounds.*" is transformed to "*Isaiah Thomas once again excelled , scoring 23 points, **3** assists and **3** rebounds.*". Under this setting, the network still has to learn the correspondence between record type and the summary token "*AST*" \cong "*assists*" but without the need of linking

"three" to "3" the connection of the phrase with record {*AST; 3; Isaiah_Thomas; Home*} should be much clearer. However we preserve the word "three" when it forms a part of a basketball terminology (e.g. *three pointer*) to differentiate between these different meanings of the word. The transformations are done with the help of the *text2num* library⁵ which is also used by the authors cited above.

2.4.2 Player name transformations

The generation system should be able to create a summary of player's actions in the game based on the records describing his match-statistics. It is common that at first a player is mentioned by his full name (e.g. *Stephen Curry*) and after that only by his second name (*Curry*). Also more than 97 % of all the players have exactly 2 names (first, last). This leaves out 17 players with longer names the most extreme case being *Luc Richard Mbah a Moute*, who is represented in the whole dataset by 6 different combinations of ellipsis in his name.

Since only the full name concatenated to a single token is contained in the input records we created a simple algorithm which transforms all ⁶ the references to a player to that specific token.

We haven't measured the accuracy of the algorithm, however it passes an eye-test as during the development of neural models we have inspected a great amount of produced summaries which haven't contained any discrepancies.

At first we gather all the player names from the input tables. The transformation then happens in three steps, which are described on the example sentence from figure 2.3.:

- **1. Extraction of player names from the summary**

The summary is at first divided to sentences, using NLTK ⁷ library. Then we traverse each sentence and extract the longest subsequences of tokens which appear in the set of the player names. This way, one-token name *James* and two-token name *James Harden* is extracted. (Although *James Harden* hasn't played in the game and therefore the network cannot learn to copy his name, we extract it to densify the data, so the player is represented by the same token in all the summaries)

- **2. Resolution of one-token references and creation of transformation dictionary**

James Harden is a two-token name matched in the first phase, so we assume that it is already full name of the player and we add a trivial transformation *James Harden* \rightarrow *James_Harden*. *James* is a one-token name which needs resolution. At first we look if anyone whose second (third ...) name is *James* hasn't already been mentioned in the summary. If not we proceed to searching through all the players in the match statistics. There we spot *LeBron James* and add the transformation *James* \rightarrow *LeBron_James*. Note that we create a unique transformation dictionary for each summary and we assume, that no player is called only by his first name.

⁵<https://github.com/alloy-media/text2num>

⁶Although technically speaking this is not true as it doesn't transform any pronouns and the transformations follow simple path: *some part of a name* \rightarrow *full name*

⁷<https://www.nltk.org/>

- **3. Application of transformations**

The summary is traversed for the second time and the longest subsequences appearing in the transformation dictionary are substituted.

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 2.3: Example of transformation of player names leveraging the knowledge of players on the rosters as well as of all players from the train set.

2.5 Vocabularies

During preprocessing we collect the vocabulary of all the words from the training set. Each token is represented as an index to the vocab. This representation is fed to the initial layers of the neural network (embedding layers which will be discussed in section 3). Therefore the network learns to process only the tokens belonging to the vocabulary and no new tokens can be introduced during inference.

We collect 3 different vocabularies, one for record types, one for home/away flags and one for all the entities, number values and tokens from the summaries.

2.6 Byte Pair Encoding

The Byte Pair Encoding (BPE) [Sennrich et al., 2016] is a method of preprocessing of a text. It was developed to enable the Neural Machine Translation models to operate on subword units. It allows them to generate and accept sequences of subwords and thus handle words unseen during training. (E.g. there are words 'high', 'low', 'lower' in the training dataset, therefore under regular setting the network couldn't be able to generate or process word 'higher'. When operating on subwords 'high', 'low', 'er' this isn't an issue anymore since the network can learn to chain subwords 'high', 'er' to form the word unseen during training.)

Since in NBA there is a fixed set of cities, teams and players (only about 10 players from development and 10 from test set were unknown from training) this isn't our main concern. However we can use the BPE to lower the size of the summary token vocabulary (E.g. looking at the previous example, even if the word 'higher' had been a part of the input vocabulary, the vocabulary of subwords would have still contained only subwords 'high', 'low', 'er'.) Thus the average frequency of a token increases as well as the generational capacity of the network.

In this section we begin with a short explanation of the algorithm and conclude with the statistics of the fully transformed dataset.

2.6.1 Algorithm

We use the implementation of the algorithm from the authors of the BPE paper⁸, which is also downloadable as a standalone python package. The idea is to divide each token from the train set to characters, and add it to the *symbol vocabulary*. Iterating over the text each time we merge together the most common pair of succeeding characters to create a new symbol. The symbol is added to the vocabulary and each occurrence of the pair is substituted by it. At the end (after N iterations, where N is the hyperparameter of the algorithm), the symbol vocabulary contains all the characters and newly created symbols.

In practice it is more efficient to create a token vocabulary (tokens are weighted by the number of occurrences in the corpus), and iterate over it instead of over the whole corpus. Also to allow easy detokenization to the original text, a special `<eos>` token is appended to each token. An excerpt from the original paper shows the minimal possible implementation of such approach 2.6.

At the test time the text is divided to characters and the longest subsequences of characters appearing in the symbol vocabulary are transformed to the corresponding symbol.

2.6.2 Application

As stated previously, the BPE should transform the output summaries to a sequence of subwords. However, in certain situations it may be contraproductive. E.g. it may make copying harder by dividing single-token player name to multiple tokens. Therefore we apply the BPE to all the tokens except the ones which correspond to player/city/team names and numerical values. We set the number of iterations to 2000, which means that the overall vocabulary contains around 2800 tokens (there is about 700 players, 29 cities and 30 teams). An example of the final appearance of a summary after all the transformations mentioned in the chapter can be seen in figure 2.4.

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 fav★ or★ ite and they did n't le★ ave any doub★ t with this result . Toronto just continu★ ou★ s★ ly pi★ led it on , as they won each quarter by at least 4 points . the Raptors were l★ ights - 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 . fas★ t★ break 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 cle★ arly did n't have enough to compe★ te with a poten★ t Raptors squad . Robert.Covington had 1 of his best games of the season though , tallying 20 points , 5 rebounds , 2 assists and 2 steals on 7 - of - 11 shooting

Figure 2.4: A part of transformed summary corresponding to the sample from figure 1.1. ★ is used to mark that the following token formed the same word in the original text. Note that in the original implementation @@ is used instead.

⁸<https://github.com/rsennrich/subword-nmt>

2.7 Cleaning

During implementation of the Content Planning approach (discussed in section 4.4.2) we found out that the authors of the method, [Puduppully et al., 2019], have used only subset of training and validation data. They removed all the pairs summary-table where the "gold" summary contained information which wasn't linked to the input table. Figure 2.5 shows an example of such a summary.

We use a subset of the subset used by Puduppully. In addition we also removed all the samples from the dataset about matches between teams from Los Angeles (Clippers and Lakers) where the distinction between different teams is not shown, and every player is listed as playing for "Los Angeles" (thus it is impossible to tell if he played for Clippers or Lakers).

After removing all the non-valid pairs, the dataset size was reduced to 3369 train, 721 development and 727 test samples.

There exist datasets based on RotoWire, which contain cleaner data and summaries corresponding better to input tables [Wang, 2019], [Thomson et al., 2020]. However we choose to continue with the RotoWire dataset, as we are already accustomed to the format of the data.

Following a week filled with trade rumors , Paul George came out of the All-Star break in fairly unimpressive fashion . In his first 4 games following the break , Paul George shot a combined 16 - of - 54 (29 percent) from the field and was averaging just 14 points per game over that stretch . However , in his last 2 games , Paul George has flipped the script entirely and rattled off a pair of incredible offensive performances . Following Monday 's 36 - point performance in the Pacers ' loss to the Hornets , Paul George has now scored a combined 70 points over his last 2 games and did so while shooting a scorching 27 - of - 44 (61 percent) from the field and 12 - of - 23 (52 percent) from behind the arc . The performances from Paul George on Sunday and Monday were by far his best back - to - back shooting and scoring performances of the season .

Figure 2.5: An example of a faulty summary. To illustrate how hard it is to tell even which teams played we purposely do not show the input table.

2.8 Statistics of Transformed Dataset

In this section we want to summarize all the transformations applied to the summaries and tables and present the statistics of the summaries after transformations.

At first we converted all the tokens in the value and entity fields of a record to a single token. Next we transformed all the numerical values in the summaries to numerals and all the player names to a single token to allow direct copying from the input records. At the end we applied the Byte Pair Encoding to all the remaining tokens to decrease the overall number of tokens and increase the average frequency of a token.

In table 2.6 we can observe that almost 90 % of all the tokens occur more than 5 times therefore we can conclude that data is definitely much denser (compared to 42 % in the original data 2.4). As a nice side effect the intersection of development/test set and train set is almost 99 % (table 2.7). It is clear from

figure 2.4 that each factual information is represented by a single token and can be copied as is from the *r.value* field of a record.

| Set | Unique Tokens | ≥ 5 Absolute | ≥ 5 Relative |
|------------|--------------------------|---|---|
| train | 2839 | 2531 | 89.15% |

Table 2.6: Occurrences of tokens in transformed summaries from dataset RotoWire

| Set | Unique Tokens | Train Overlap | Train$_{\geq 5}$ Overlap |
|------------|--------------------------|--------------------------|--|
| valid | 2582 | 98.80% | 95.70% |
| test | 5741 | 98.69% | 95.45% |

Table 2.7: Overlap of transformed train dataset summaries and valid/test dataset summaries

Tables 2.8, 2.9 show that while the lengths of input records decreased (mainly because of player name transformations discussed in section 2.3.3), the lengths of output summaries increased (due to byte pair encoding).

| Set | Max Number of Records | Min Number of Records | Average Number of Records | Size |
|-------------|--------------------------------------|--------------------------------------|--|-------------|
| train | 690 | 514 | 593.41 | 3369 |
| development | 646 | 536 | 593.40 | 721 |
| test | 646 | 514 | 593.77 | 727 |

Table 2.8: Length statistics of the preprocessed tables.

| Set | Max Summary Length | Min Summary Length | Average Summary Length | Size |
|------------|-----------------------------------|-----------------------------------|---------------------------------------|-------------|
| train | 826 | 148 | 351.41 | 3397 |
| validation | 847 | 150 | 357.73 | 727 |
| test | 805 | 146 | 364.50 | 728 |

Table 2.9: Length statistics of the preprocessed summaries.

```

1 import re, collections
2
3 def get_stats(vocab):
4     pairs = collections.defaultdict(int)
5     for word, freq in vocab.items():
6         symbols = word.split()
7         for i in range(len(symbols)-1):
8             pairs[symbols[i],symbols[i+1]] += freq
9     return pairs
10
11 def merge_vocab(pair, v_in):
12     v_out = {}
13     bigram = re.escape(' '.join(pair))
14     p = re.compile(r'(?!\S)' + bigram + r'(?!\S)')
15     for word in v_in:
16         w_out = p.sub(''.join(pair), word)
17         v_out[w_out] = v_in[word]
18     return v_out
19
20 vocab = {'l_o_w</w>' : 5, 'l_o_w_e_r</w>' : 2,
21         'n_e_w_e_s_t</w>':6, 'w_i_d_e_s_t</w>':3}
22 num_merges = 10
23 for i in range(num_merges):
24     pairs = get_stats(vocab)
25     best = max(pairs, key=pairs.get)
26     vocab = merge_vocab(best, vocab)
27     print(best)

```

OUTPUT: r · → r·
 l o → lo
 lo w → low
 e r · → er·

Figure 2.6: Python code extracted from paper **Neural Machine Translation of Rare Words with Subword Units** by Sennrich et al. [2016]

Output represents the learned merge operations.

3. Neural Network Architectures

In previous chapter (2) we have shown how to transform the structured tabular data into a sequence of records, therefore we have reduced the problem to an instance of the famous sequence to sequence problem. Now, we 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 we used throughout this thesis. In this chapter we 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 we provide links to papers, books and tutorials which helped the author on his path to understanding.

Notation

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

A Note About Embeddings

"An embedding is a low-dimensional, learned continuous vector representation of discrete variables into which you can translate high-dimensional vectors." [Daylor, 2019]. In this work we do not experiment with pretrained word embeddings, and we tune only the embedding-dimension hyperparameter of *tf.keras.layers.Embedding* layer.

3.1 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 we 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.

3.1.1 Recurrent Neural Network

Let $\mathbf{x} = (x^{(1)}, \dots, x^{(t)})$ be the input. The standard Feed-Forward Network (*tf.keras.layers.Dense*) has a 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.

The Feed-Forward Neural Network

$$\mathbf{y} = \text{activation}(W\mathbf{x} + b) \quad (3.1)$$

On the contrary, the Recurrent Neural Network (RNN) [Rumelhart et al., 1988] (*tf.keras.layers.SimpleRNN*) shares the same set of weights between time-steps and in addition it keeps a hidden state. At each time-step the hidden state is updated and used to calculate the output as in equation 3.2. The computation can be visualized either as a loop, or as a feed-forward network with shared weights 3.1.

The Recurrent Neural Network

$$\begin{aligned} h_t &= f_h(x_t, h_{t-1}) \\ y_t &= f_t(h_t) \end{aligned} \quad (3.2)$$

Note: h_t is the hidden state; y_t is the output at t -th timestep; *tf.keras.layers.SimpleRNN* uses $f_t \cong id$ and $f_h \cong tanh$ as default.

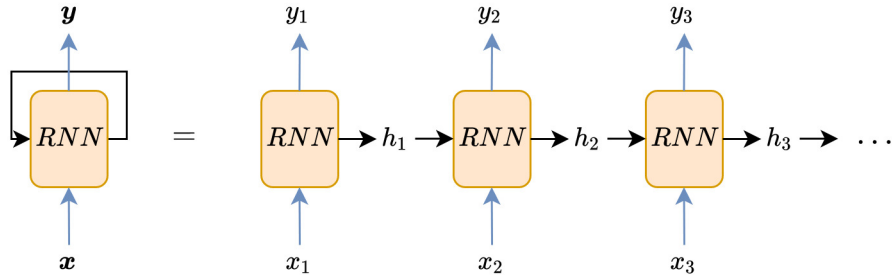


Figure 3.1: Visualizations of the RNN

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].¹ Which cause that either the RNN cannot learn anything or it is really unstable. These difficulties are addressed by more sophisticated architectures such as Gated Recurrent Unit² [Cho et al., 2014b] or Long Short-Term Memory [Hochreiter and Schmidhuber, 1997].

3.1.2 Long Short-Term Memory

The Long Short-Term Memory (*tf.keras.layers.LSTM*) addresses the vanishing gradient problem. It does so by adding a special cell state for capturing long range context and series of gating mechanisms. The latter update the cell state and regulate the flow of gradient through the network (as shown in figure 3.2). The more in-depth explanation can be found in [Olah, 2015].

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

²Although it is one of the most known architectures we used only LSTM for our experiments.

$$y_t = W_{hy}h_t + b_y \quad (3.3)$$

$$h_t = o_t \tanh(c_t) \quad (3.4)$$

$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o) \quad (3.5)$$

$$c_t = f_t * c_{t-1} + i_t * \tanh(W_c[h_{t-1}; x_t] + b_c) \quad (3.6)$$

$$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i) \quad (3.7)$$

$$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f) \quad (3.8)$$

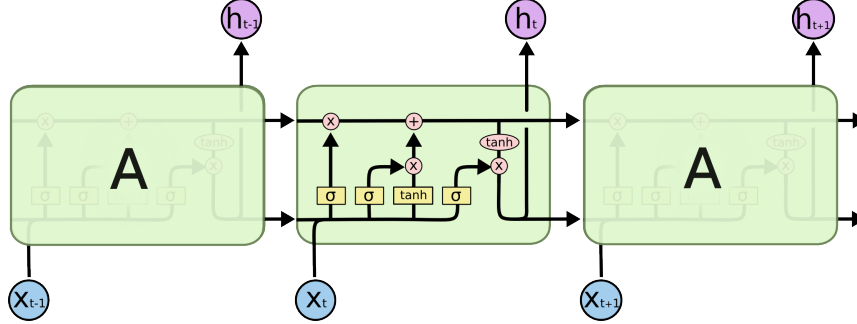


Figure 3.2: Visualization of LSTM [Olah, 2015]

3.1.3 High-level Overview of Encoder-Decoder Architecture

As stated by [Sutskever et al., 2014], the main goal of the encoder-decoder architecture is to estimate the conditional probability $p(y_1, \dots, y_n | x_1, \dots, x_m)$ of the output sequence y_1, \dots, y_n conditioned on the input sequence x_1, \dots, x_m . It uses two separate *recurrent networks*³. The first, called the Encoder, processes the input sequence. Its last hidden state represents a fixed-dimensional representation r of the input. r is then used to initialize the hidden state of the second recurrent network, called the Decoder, which models the conditional probability of the output sequence (equation 3.9).

$$p(y_1, \dots, y_n | x_1, \dots, x_m) = \prod_{t=1}^n p(y_t | r, y_1, \dots, y_{t-1}) \quad (3.9)$$

The dimensionality of the output of the Decoder at time-step t is the same as the size of the output vocabulary. The softmax (equation 3.10) over the outputs is used to represent the distribution $p(y_t | r, y_1, \dots, y_{t-1})$.

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (3.10)$$

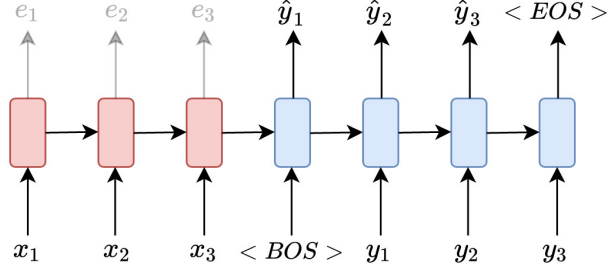
Training

The Decoder is trained under the *teacher-forcing* regime. The main gist of this approach is to feed the gold output y_{t-1} as the input at time-step t . Since there

³Here we refer to *recurrent network* as to a complex consisting of at least one RNN/LSTM/GRU/... rather than to a single recurrent layer.

is no zeroth gold output we use a special $\langle BOS \rangle$ (*beginning of sequence*) token as the first input to the Decoder.

Another special token, the $\langle EOS \rangle$ (*end of sequence*) token is appended at the end of each target sequence. This way we train the model to explicitly show when the generation is over (therefore the produced sequences can be of variable length). The visualization of the approach is shown in figure 3.3.



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

Figure 3.3: Visualization of the training of the Encoder-Decoder Architecture with *teacher forcing*.

Inference

In the inference phase, we want to find a sequence of tokens y_1, \dots, y_n which maximizes the probability

$$p_{model}(\mathbf{y}|\mathbf{x}) = p_{model}(y_1|\mathbf{x})p_{model}(y_2|y_1, \mathbf{x}) \dots p_{model}(y_N|y_1, \dots, y_{N-1}, \mathbf{x}) \quad (3.11)$$

$$= \prod_{t=1}^N p_{model}(y_t|y_1, \dots, y_{t-1}, \mathbf{x}) \quad (3.12)$$

Calculating all $\mathcal{O}(N^{|V|})$ sequences and choosing the maximal is surely the most accurate option, although computationally infeasible. Therefore we only approximate the optimal solution.

Greedy Decoding

Greedy Decoding provides the simplest approximation of the optimal sequence. At each time-step we take the most probable token under the model distribution as the output. The process ends when the $\langle EOS \rangle$ token is generated.

$$\begin{aligned} \hat{y}_1 &= \arg \max_{y'} p_{model}(y'|\mathbf{x}) \\ \hat{y}_2 &= \arg \max_{y'} p_{model}(y'|\hat{y}_1, \mathbf{x}) \\ &\dots \\ \langle EOS \rangle &= \arg \max_{y'} p_{model}(y'|\hat{y}_1, \dots, \hat{y}_{n'}, \mathbf{x}) \end{aligned}$$

The suboptimality of the algorithm can be seen on a simple example. E.g. let's say that we have a training corpus consisting of sentences describing the

eating habits of the author of this text. The corpus consists of sentences "I eat a banana", "I eat a peach", "I eat a goulash" and two repetitions of sentence "I eat an apple". Starting from the state after generating subsequence "I eat", the greedy decoder would pick "a" as the most probable continuation of the sequence. However the optimal solution would have picked "an", because none of the possible continuations of subsequence "I eat a" is as probable as "I eat an apple" which is the most occurring sentence in the corpus.

Beam Search Decoding

Beam Search builds on the greedy decoding approach. We keep track of k most promising *hypotheses* (and associated hidden states). A hypothesis is a sequence of generated tokens $y_1, \dots, y_{n'}$. We compute its score (equation 3.13).

$$\text{score}(y_1, \dots, y_{n'}) = \sum_{i=1}^{n'} \log p_{\text{model}}(y_i | y_1, \dots, y_{i-1}, \mathbf{x}) \quad (3.13)$$

At each time-step we expand all the hypotheses (take k most probable tokens under the respective hypothesis, which will result in k^2 possibilities), and choose k with the highest score. k is called the *beam size*. An example of the approach can be seen in figure 3.4. There exist several options what to do when some hypothesis expands to $\langle \text{EOS} \rangle$ token. The finished hypothesis can be put aside and the generation continues until T -th time-step (T is another hyperparameter of the algorithm), or until at least N hypotheses are finished. We choose yet another option, to end the generation right after the first $\langle \text{EOS} \rangle$ is generated⁴.



Figure 3.4: Beam-Search decoding, excerpt from the slides to lecture about Machine Translation, Seq2Seq and Attention on Stanford
<http://web.stanford.edu/class/cs224n/>

⁴This option may suffer from generating too short summaries (because some hypothesis at the beginning may expand to $\langle \text{EOS} \rangle$ token and beat all the remaining ones although they may have had better score), however we have not experienced this problem.

3.1.4 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 2, we know that there are 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 mention a transformer, therefore each word describing some kind of a transformer is somewhat probable, even if it wasn't seen in the actual input.

We have already discussed how to increase the average frequency of a token to minimize the Rare-Word Problem through preprocessing (section 2.6). In the following sections we show the methods that handle hallucinations (section 3.3).

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

3.2 Attention Mechanism

The Attention mechanism should cope with the issue of fixed-dimensional representation of the input sequence 3.1.4. 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 a recurrent neural network⁶. From the overall architecture (figure 3.3) 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] propose an architecture which takes advantage of this simple observation. In our work we use a little refinement, proposed by [Luong et al., 2015].⁷

3.2.1 Computation

Let's start with the description of the computation that produces the attention output.

As stated previously, the Encoder encodes the input sequence to *encoder outputs* $\mathbf{e} = (e_1, \dots, e_m)$ and the Decoder RNN is initialized with the last hidden state of the Encoder.

Let d_t be the output of the Decoder RNN at t -th time-step. At first we calculate the *score vector* $\mathbf{s}_t = (s_{t,1}, \dots, s_{t,m})$. Its elements are computed using a *score function* (which we will talk about below 3.2.2):

$$s_{t,i} = \text{score}(e_i, d_t) \quad (3.14)$$

According to [Bahdanau et al., 2014], the alignment vector \mathbf{a}_t

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t) \quad (3.15)$$

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

$$c_t = \sum_{i=1}^m a_{t,i} e_i \quad (3.16)$$

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

$$\text{att}_t = \tanh(W_c[c_t; d_t]) \quad (3.17)$$

$$p(y_t | y_{<t}, x) = \text{softmax}(W_y \text{att}_t) \quad (3.18)$$

3.2.2 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

⁶From 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.

⁷Since we do not experiment with the original Bahdanau attention, we 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.

directly extracted from the Luong’s paper)

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

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 account its previous decisions. It simply means that the next input is the concatenation $[y_t, att_t]$ (as shown in figure 3.5).

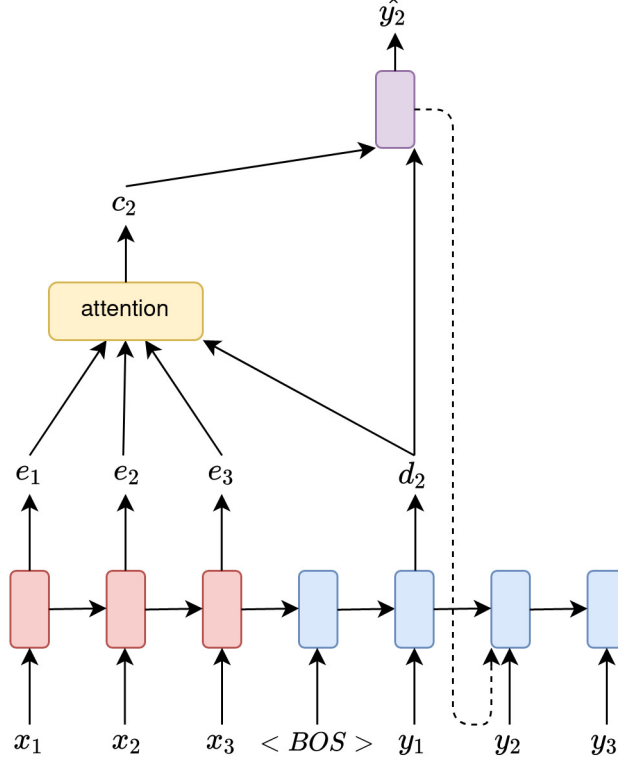


Figure 3.5: The Attention mechanism at the second time-step. Dotted line represents the input feeding approach.

3.3 Copy mechanism

The copy mechanism is a further extension of the attention mechanism. In this section we 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).

3.3.1 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.

3.3.2 Approaches to Copying

[Gulcehre et al., 2016] note 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.

Let $\mathbf{e} = (e_1, \dots, e_m)$ be the encoder outputs, $\mathbf{x} = (x_1, \dots, x_m)$ the input sequence, d_t the output of the Decoder RNN at the actual time-step, and $ATTN$ the Attention as presented in the previous section.

The probability of gold output y_t and gold switch decision z_t is decomposed to $p^{gen}(y_t|\mathbf{x})$ (the probability that y_t should be generated), $p^{switch}(z_t|\mathbf{x})$ (the probability that we should copy) and $p^{copy-pos}(y_t = x_i|\mathbf{x})$ (the probability that y_t should be copied from the input position i):

$$\mathbf{a}_t = ATTN(\mathbf{e}, d_t) \quad (3.19)$$

$$p^{copy-pos}(y_t = x_i|\mathbf{x}) = a_{t,i} \quad (3.20)$$

$$\mathbf{c}_t = \sum_{i=1}^m e_i * a_{t,i} \quad (3.21)$$

$$p^{switch}(z_t|\mathbf{s}) = \text{sigmoid}(W_{switch}[\mathbf{c}_t, d_t]) \quad (3.22)$$

$$p^{gen}(y_t|\mathbf{s}) = \text{softmax}(W_{gen}[\mathbf{c}_t, d_t]) \quad (3.23)$$

[Gulcehre et al., 2016] explicitly model each of these probabilities (therefore the targets contain 3 values for each time-step). [Yang et al., 2016] marginalize out the switch probability z_t , and they model $p = p^{copy} * p^{switch} + p^{gen} * (1 - p^{switch})$.

To be able to follow their path, we take the one-hot encoding of each input and compute the weighted sum:

$$p^{copy}(y_t|\mathbf{x}) = \sum_{i=1}^m p^{copy-pos}(y_t = x_i|\mathbf{x}) * x_i \quad (3.24)$$

Throughout our task the input and output vocabularies are shared, therefore the weighted sum of the inputs has the same dimensionality as the output generation distribution p^{gen} .

Consequently, the probability of the gold output y_t at time-step t is computed as follows:

$$p(y_t|\mathbf{x}) = p^{gen}(y_t|\mathbf{x}) * (1 - p^{switch}(1|\mathbf{x})) + p^{copy}(y_t|\mathbf{x}) * p^{switch}(1|\mathbf{x}) \quad (3.25)$$



Figure 3.6: Excerpt from **Pointing the Unknown Words** paper by [Gulcehre et al., 2016], showing how the attention alignment can be utilized as a pointer information

4. Experimental Setup

In this chapter we aim to clarify how we set up the neural network models for the experiments (which will be discussed in the next chapter 5). At first we present a *baseline model*. Model, which is sufficiently competitive to be able to generate reasonable texts. Then we present changes to the architecture, which aim to improve the generation.

We follow the path set up by Wiseman et al. [2017]. We begin with purely end-to-end approach (Encoder-Decoder with attention). Next we try to improve it with the copy mechanism 3.3. Inspired by Puduppully et al. [2019] we continue by dividing the task to *content planning* and *text generation*.

4.1 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 $\langle PAD \rangle$ 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 memory¹)

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] propose 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.

To illustrate our implementation of the algorithm, we show how we process an output sequence which is longer than 150 tokens. Let $t_1 = 50$, $t_2 = 100$. At first we let the network predict the first 100 outputs and run the BPTT. We keep aside the 50-th hidden state of the decoder. Next the network (initialized with the hidden state from the 50-th time-step) predicts positions 51 to 150, and again the BPTT is run. Similarly afterwards.

4.2 Baseline model

We use a non-recurrent Encoder and a two-layer LSTM decoder with Luong-style attention as the baseline model. At first we present how the encoding of input records works, then we discuss the text generation. We highlight all the differences between our approach and the one taken by Wiseman et al. [2017]. The visualization of the model is shown in the figure 4.1.

4.2.1 Encoder

The Encoder should process the input records (the formation of a record is explained in section 2.3.3) to create the partial outputs at each time-step and the initial hidden state for the Decoder.

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

We described in section 2.3.5 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. We choose the same approach as Wiseman et al. [2017] (who was inspired by Yang et al. [2016]), and we pass the concatenation through a one layer feed-forward network (*tf.keras.layers.Dense*) with ReLU activation², 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 we observe that each player entity is represented by 22 records and each team entity by 15 records 2.3.3. Consequently we approximate the approach taken by Wiseman et al. [2017] and mean pool over \mathbf{e} with stride 22. During our experiments we haven't seen any indication that this modification became the performance bottleneck of the model.

4.2.2 Decoder

The Decoder is a 2-layer LSTM network with attention mechanism. The LSTMs are initialized with states prepared in the previous section. We opted to use the Luong style attention with input feeding [Luong et al., 2015]. We described in section 3.2.2 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 $\langle BOS \rangle$ token, there is no *last attentional decision*. Hence for this purpose we use one of the initial states prepared by the Encoder.

4.2.3 Training

Given an input table \mathbf{x} and the corresponding gold output summary \mathbf{y} the model approximates the conditional probability of the latter conditioned on the former $p(\mathbf{y}|\mathbf{x})$. Following the maximum likelihood principle (e.g. section 5.5 in [Goodfellow et al., 2016]) the model is trained by minimizing the cross-entropy loss (negative log likelihood) on the training set \mathcal{D} .(equation 4.1).

$$- \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log p_{model}(\mathbf{y}|\mathbf{x}) \quad (4.1)$$

As explained in section 3.1.3 the training happens under *teacher-forcing* setting. The minimization is provided by stochastic gradient descent, specifically we opted to use one of the standard algorithms, Adam [Kingma and Ba, 2014]. Since this algorithm associates specific learning rate to each of the trainable network parameters, we modify only the initial learning rate parameter³.

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

³It means that we don't try e.g. learning rate decay.

We report the loss value as well as the accuracy⁴ of the model on the training set, to be able to detect *underfitting* (this happens when a model is unable to achieve sufficiently low loss value on the training part of the dataset).

The main challenge is to obtain good results on previously unseen data. Therefore we report the loss value, and the accuracy of the model on the validation part of the dataset (also collected under *teacher-forcing* setting). We consider the model with the lowest loss value and accuracy on the validation set to be the best⁵.

4.2.4 Regularization

Two regularization methods are used, Dropout and Scheduled Sampling [Bengio et al., 2015].

"The key idea of Dropout is to randomly drop units (along with their connections) from the neural network during training." [Srivastava et al., 2014]. A unit is dropped with probability p that can be set as a hyperparameter. (e.g. $p = 0$ would mean no dropout) We apply the dropout on the outputs of the LSTM cells.

The Scheduled Sampling aims to minimize the difference between training and inference. Bengio et al. [2015] note that using any of the two generation techniques described in sections 3.1.3, 3.1.3 the model can get to the "state space that is very different from those visited from the training distribution and for which it doesn't know what to do". They propose to "bridge the gap" by feeding either gold y_{t-1} or the prediction of the model \hat{y}_{t-1} as the input at t -th time-step. The decision whether to use gold is made independently for each input with probability p' which is another hyperparameter of the training. They propose to decay p' over time with similar strategies as used with learning rate.

4.3 Improving the Baseline with Copy Mechanisms

In section 3.1.4 we 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 we incorporate the *Joint Copy mechanism* [Gu et al., 2016], [Yang et al., 2016] (already described in 3.3). The encoder rests intact, however in the decoder we 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 we 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 4.2.

⁴the frequency with which the most probable token in the distribution generated by the model matches the gold token

⁵We also report the BLEU score [Papineni et al., 2002] of the text generated from a subset of the validation dataset and if there are multiple epochs where the model achieved similar performance we pick the one which generated the best text according to BLEU.



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

Figure 4.1: The Rotowire baseline model at the second time-step.

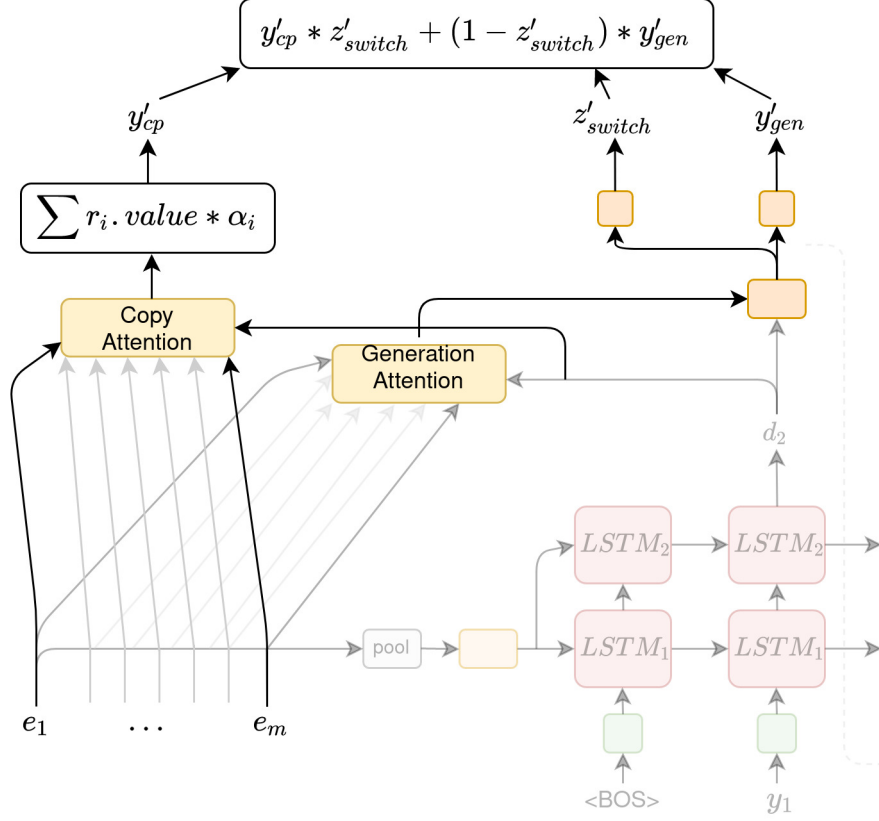
4.4 Content Selection and Planning

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

4.4.1 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 4.2, the self-attention is used to model the "importance vis-a-vis other records in the table".



Note: green cells are embedding layers, orange cells are feed-forward networks
 α is the alignment vector produced by the copy attention; $r_i.value$ is the value portion of the i -th input record.

Figure 4.2: The Joint-Copy extension of the baseline model at second time-step.

Specifically, we compute

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

The Content Selection Encoder thus creates a sequence of *context aware* representations $\{\hat{r}_t^{cs}\}_{t=1}^m$ (figure 4.3). We experiment with a model using Content Selection Encoder and Joint-Copy decoder (we call it *CopyCS*).

4.4.2 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 records appear in the output summary. Puduppully et al. [2019] suggested that

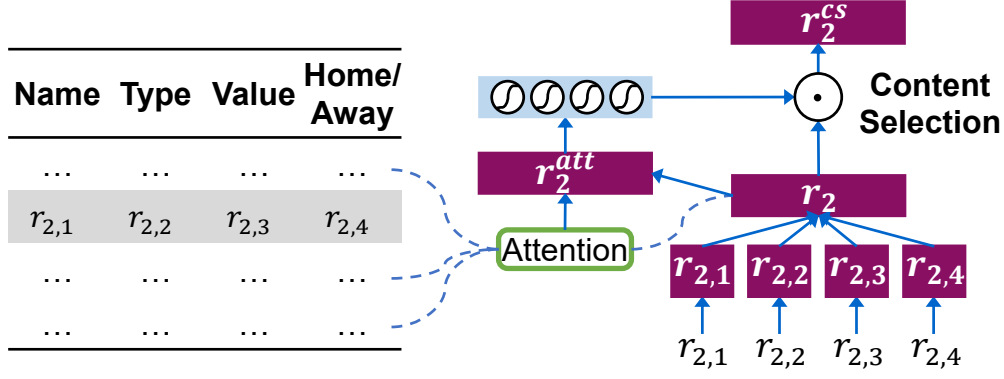


Figure 4.3: Content Selection mechanism (image is directly from [Puduppully et al., 2019])

instead of just modelling the switch probability, we can train a Decoder to extract these pointers from the original table.

As suggested, the Content Plan Decoder is a one layer LSTM which operates on the *context aware* representations \hat{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 we have diverged a little bit from the intentions of original authors.

| 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 4.1: An extract from the content plan corresponding to the summary from the figure 1.1

The Text Decoder is trained to start the generation when it sees the $\langle BOS \rangle$ token. Since the Content Planning Decoder operates on \hat{r}^{cs} we have chosen to prepend a special $\langle BOS \rangle$ record to the sequence of input records, and also a pointer to the $\langle BOS \rangle$ 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, we teach it to do so when seeing *the encoded representation of the special value*. The same approach is taken at the end, with the $\langle EOS \rangle$ record.

Puduppully et al. [2019] use a one layer bidirectional LSTM on top of the generated content plans as shown in figure 4.4.

Either baseline Decoder or Joint-Copy Decoder can operate as the Text De-

coder to generate the output summary.

4.4.3 Training and Inference

At first let me elaborate on the training. The model consists of two parts which are trained jointly. According to notation in figure 4.4 we calculate the content planning loss and the text decoding loss. These are added together to form the overall loss which is minimized by the optimizer. During training the inputs to the Content Plan Encoder are the encoded records pointed by the *gold content plan*. Equation 4.2 shows the loss value we aim to minimize during training.

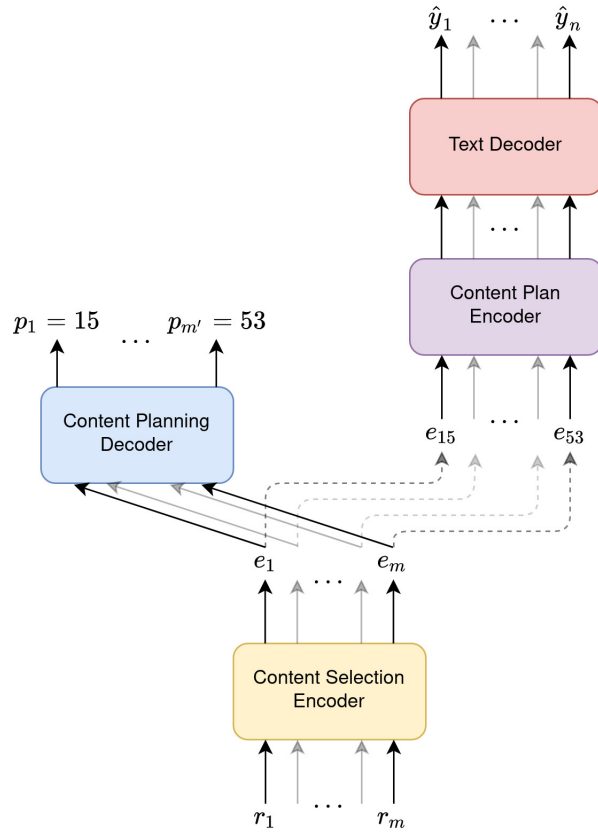
$$\sum_{(x,y,z) \in \mathcal{D}} -\log p(\mathbf{y}|\mathbf{z}, \mathbf{x}) - \log p(\mathbf{z}|\mathbf{x}) \quad (4.2)$$

There are many possibilities what to do during inference. In section 5.5 we mention five options. The outputs of the Content Planning Decoder can be obtained using Greedy or Beam Search decoding, similarly for the Text Decoder. At last there is possibility to ignore the Content Planning Decoder and use the encoded records pointed to by the gold content plan. (if we have any gold content plan available). Equation 4.3 shows the quantity we aim to maximize during the inference.

$$\arg \min_{\mathbf{y}', \mathbf{z}'} p(\mathbf{y}'|\mathbf{z}', \mathbf{s}) + p(\mathbf{z}'|\mathbf{s}) \quad (4.3)$$

4.5 Postprocessing

The models are trained to generate lowercased Byte Pair Encoded sequences. During the postprocessing we merge the tokens belonging to one word together (e.g. "contribu★tor" → "contributor"), split the tokens of named entities (e.g. "LeBron_James" → "LeBron James") and uppercase the first letter of a sentence. Since the vast majority of unique tokens (more than 98.6%) from the validation and test sets (section 2.8) is in the vocabulary collected from the training set, we opted to omit



Note: $p_1 = 15$ means that the first generated pointer points to 15-th encoded record (therefore e_{15} the first input to the Content Plan Encoder), similarly for $p_{m'}$

Figure 4.4: Overall architecture of the Content Selection and Planning model with example outputs.

5. Experiments

In previous chapters we have presented the dataset (chapter 1), and discussed challenges which arise from its properties (chapter 2). Next we described the neural network architectures (chapter 3) which served as building blocks for the models introduced in chapter 4. Now we want to connect all the parts together, elaborate on the evaluation methods and analyze the results of different models.

Let us recap the main challenges we face, and hypothesize about the ideal generation system. Firstly, the target summaries are really long. The ideal system should remember what has already been generated and shouldn't produce duplications. Secondly, the targets contain a lot of facts based on the input structured data. The ideal system should copy these facts from the input and reduce *hallucinations*. Lastly, the generated text should be as close to English as possible *while meeting the requirements described previously*.

5.1 Evaluation Methods

During evaluation I want to measure which model resembles the hypothetical ideal model the most. To do so, I report the BLEU score [Papineni et al., 2002] of the postprocessed generated summaries on the validation and test sets. Although it is the gold standard, there are many people arguing against its usage as a performance metric [Celikyilmaz et al., 2021], and I found that the networks producing more factually correct statements doesn't score better. Therefore I also manually evaluate a subset of the generated summaries.

5.1.1 Manual Evaluation

I used a pseudo random generator to pick 8 different data points, 4 from the validation and 4 from test set¹. I looked at the factual correctness (how many of the generated numbers describing team and player statistics are based on the input tabular data), and factual recall (how many of the entities mentioned in the gold summaries are present in the generated ones). Due to time-requirements of the process, only the best model² is manually evaluated. Therefore manual evaluation should show the differences between approaches and architectures rather than between variations of the same model.

5.1.2 Omitted Evaluation Methods

[Wiseman et al., 2017] proposed three custom automated metrics to evaluate the performance of their models. They call them *Content Selection* ("how well the generated document matches the gold document in terms of selecting which records to generate"), *Relation Generation* ("how well the system is able to generate text containing factual (i.e., correct) records") and *Content Ordering* ("how

¹Specifically samples 132, 319, 475 and 709 from the validation set and samples 16, 247, 585 and 671 from the test set

²In the respective sections I explain why I think the selected model is the best one.

well the system orders the records it chooses to discuss”). The metrics are implemented in outdated neural network framework *Torch* and I wasn’t able to execute it on my computers. After discussions with my advisor we agreed to not adopt these methods.

5.2 Baseline Model

At first I would like to summarize my expectations. [Wiseman et al., 2017] note that even models with explicit copy mechanisms tend to *hallucinate* the facts. However they do not show the results of any baseline without copying. We expect the baseline to generate a fairly good English, even though the summaries should not be factually correct.

I searched through the hyperparameter space (possible combinations of learning rates, dropout probability p , hidden state dimensionality etc.) until the generated language as well as the numerical results haven’t fulfilled the expectations at least by half.

The overall architecture of the baseline model is discussed in section 4.2. Figure 5.1 shows the hyperparameter choice for discussed baseline models.

| Hidden States Dimensionality | Learning Rate | Batch Size | Token | Embedding Dimensionality |
|---------------------------------|---------------|------------|-------------------------|-----------------------------|
| 600 | 0.001 | 16 | <i>record.type</i> | 300 |
| | | | <i>record.home_away</i> | 300 |
| | | | <i>record.value</i> | 600 |
| | | | <i>record.entity</i> | 600 |
| | | | <i>summary token</i> | 600 |

Figure 5.1: Hyperparameter settings for baseline models.

The first baseline model is trained without any regularization, the second one with dropout on the output LSTM units with $p_{dropout} = 0.3$ (the probability that the unit is dropped is 0.3) and scheduled sampling with constant rate of 0.8 (the probability that the gold output from the previous timestep is used as the actual input is 0.8).

5.2.1 Results

Baseline model wasn’t able to capture the relationships between the input tabular data and the output summaries (the best configuration achieved perplexity 10.59). We found that dropout helps to reduce overfitting while scheduled sampling improved the quality of the generated summaries. (We present only models which achieved reasonable performance.) The quality of generated summaries is further improved with beam search. Although the statement contradicts the calculated BLEU scores (as the score decreased after introducing beam search), the summaries contain more natural language and less repetitions. The expectations are fulfilled in terms of *hallucinations*, as the model makes up *almost all the factual statements*.

Figure 5.2 shows an example generated by the regularized baseline model

| Model | Validation Perplexity | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|------------------|--------------------------|--------------------|--------------|------------------|------------------|
| BN _G | 12.68 | 9.04 | 9.10 | – | – |
| BN _{B5} | | 8.73 | 8.55 | – | – |
| BR _G | 10.59 | 10.0 | 10.47 | – | – |
| BR _{B5} | | 9.99 | 10.6 | 37.35% | 8.03% |

_G - Greedy Decoding

_{B5} - Beam search decoding, beam size = 5

BN - Baseline, non-regularized

BR - Baseline, regularized (dropout 0.3, scheduled sampling 0.8)

Table 5.1: Performance metrics on the baseline models.

with beam search decoding. We observe³ that the score, the winning-losing records⁴ as well as all the individual statistics are *hallucinated*. We found that the model repeatedly (over multiple observed summaries) mentions the same players and uses the same numbers. This experiment thus supports our assumptions. Statistics in table 5.1 show that only about 8 % of the numbers aren’t *hallucinated* and that the recall is lower than 35 %. Therefore we can conclude that model learned to produce probable numbers and names of star players of the respective teams.

The Toronto Raptors defeated the Philadelphia 76ers , 109 - 95 , at air canada centre on tuesday . The Raptors (21 - 15) checked in to saturday ’s contest trying campaign , having won 7 of their last 7 games . However , Philadelphia was able to pull away in the second half , as they outscored Philadelphia by a 59 - 44 margin over the second and third periods . DeMar DeRozan led the way with a points - point , 10 - rebound double - double that also included an assist , a steal and a block . Kyle Lowry was right behind with a points - point , 10 - rebound double - double that also included an assist , a steal and a block . Jonas Valanciunas posted a near double - double with 17 points , 9 rebounds , 2 assists and a steal . Jonas Valanciunas posted a near double - double with 11 points , 9 rebounds , 2 assists and a steal . Ersan Ilyasova was productive in a reserve role as well with 11 points , 4 rebounds , an assists and a steal . The Raptors head back home to take on the Toronto Raptors on saturday night , while the Raptors remain home to tangle with the Brooklyn Nets on tuesday evening as well .

Figure 5.2: A summary generated by the baseline model. The corresponding gold summary and input table is shown in figure 1.1.

5.3 Joint-Copy Model

The Joint-Copy Model is expected to produce more factually correct statements. [Wiseman et al., 2017] select this model as their baseline. Due to massive preprocessing that we introduced, the results we obtained aren’t comparable to theirs. (E.g. Wiseman et al. report BLEU 10.41 while we obtained similar scores with *simpler baseline model*.)

³We opted to not show the same tabular information multiple times. A part of the table corresponding to the generated summary can be found in figure 1.1

⁴”The Raptors (21 - 15)” means that Raptors have won 21 matches and lost 15 matches this season.

The additional complexity of the model increases the memory demands. Therefore we use lower dimensional embeddings, hidden states and smaller batch size as presented in figure 5.3 to be able to train on the GPUs on the AIC. Compared to the baseline model we obtained the best results when training with 5-times lower learning rate. Regularization methods didn't provide any significant advantage over the unregularized case.

| Hidden States Dimensionality | Learning Rate | Batch Size | Token | Embedding Dimensionality |
|---------------------------------|---------------|------------|-------------------------|-----------------------------|
| 500 | 0.0002 | 8 | <i>record.type</i> | 300 |
| | | | <i>record.home_away</i> | 300 |
| | | | <i>record.value</i> | 500 |
| | | | <i>record.entity</i> | 500 |
| | | | <i>summary token</i> | 500 |

Figure 5.3: Hyperparameter settings for joint-copy models.

5.3.1 Results

We see the first major improvement in the BLEU score (more than 2 points) as well as in the manually evaluated metrics (almost 6-times better factual correctness). However the model fails in recognition of the structure of the table.

While it is able to copy facts, many times it copies statistics of wrong players (e.g. it learns that most of the time 5-th player from the input records is most important, therefore it mentions the 5-th player even if he scored unremarkable amount of points).

We also spotted that model learned that it is more probable that home team wins. Therefore it produces sentence similar to *"The Denver Nuggets (11 - 17) defeated the Los Angeles Lakers (5 - 23) 111 - 107 on friday at the pepsi center in Denver . "* even if the score was reversed (*107 - 111* in favour of the Lakers).

Also the model relatively often generates a sequence of tokens which cannot be seen in the training data which causes *cycling* (model generates one sentence multiple times), which results in exceeding the maximal allowed length of a summary (We don't allow the model to generate longer sequence than the longest one seen in the dataset (849 tokens)). The phenomenon can be seen in figure 5.4.

Contrary to observations on the baseline model neither greedy decoding nor beam search managed to significantly outperform the other method.

| Model | Validation Perplexity | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|--------------------|--------------------------|--------------------|--------------|------------------|------------------|
| Copy _G | 9.87 | 12.48 | 12.6 | 37.35% | 47.29% |
| Copy _{B5} | | 12.19 | 12.5 | 39.76% | 47.13% |

Copy - Joint-Copy model
_G - Greedy Decoding
_{B5} - Beam search decoding, beam size = 5

Table 5.2: Performance metrics on the joint-copy models.

The Toronto Raptors defeated the Philadelphia 76ers , 122 - 95 , at air canada centre on wednesday . The Raptors (11 - 14) were in the first quarter for the Raptors (4 - 14) , as the sixers (4 - 14) were able to pull away in the fourth quarter , out - scoring the 76ers 31 - 26 in the first quarter , while also holding the Raptors (4 - 14) by a 59 - 38 margin over the second and third quarters . However , the Raptors (11 - 14) shot 42 percent from the field and 68 percent from the three - point line , while the Raptors went 42 percent from the floor and 68 percent from the free - throw line . The Raptors also out - rebounded the Raptors 42 - 38 , while also holding the Raptors to just 42 percent . The Raptors were led by Joel Embiid , who finished with 24 points (7 - 14 fg , 6 - 6 ft) , 8 assists and 4 rebounds , in 32 minutes . Kyle Lowry was the only other starter to reach double figures , as he finished with 24 points (7 - 14 fg , 1 - 6 3pt , 4 - 5 ft) , 8 assists , 4 rebounds and 1 steal , in 32 minutes . Kyle Lowry was the only other starter to score in double figures , as he finished with 15 points (7 - 9 fg , 1 - 2 ft) , 5 rebounds and 8 assists , in 25 minutes . The other starter to reach double figures in scoring was DeMar DeRozan , who finished with 14 points (4 - 13 fg , 0 - 3 3pt , 6 - 6 ft) , along with 5 rebounds , 4 assists , 1 steal and 1 block , in 31 minutes . The other starter to reach double figures in scoring was DeMar DeRozan , who finished with 14 points (4 -

Figure 5.4: A summary generated by the joint-copy model. The corresponding gold summary and input table is shown in figure 1.1.

5.4 Content Selection Encoder with Joint-Copy Decoder

[Puduppully et al., 2019] introduced two new concepts. The first one, the Content Selection (CS) (discussed in section 4.4.1) uses self-attention to incorporate *context awareness*. Basically it should mean that the encoded record representation should be aware about the most important records related to itself. [Puduppully et al., 2019] report that CS alone contributes to the quality of the generated summaries. We expect the model with CS to resolve the problem of mixing statistical information of multiple players.

The hyperparameter configuration rests the same as in Joint-Copy model (figure 5.3).

5.4.1 Results

The summaries produced by the model are richer in structure, as they mention more entities from the input tables (44.58 % vs 39.76 % for the Copy model). However we observe that many times model doesn't copy the players name (it only learns to copy numbers). E.g. it generates a sentence *"The Raptors were led by DeMar DeRozan , who finished with a team - high of 26 points (9 - 16 fg , 0 - 2 3pt , 8 - 9 ft) , to go along with 7 rebounds and 5 assists ."* where there would be 8/9 numbers correct if the name wasn't *DeMar DeRozan* but *Marc Gasol*. This causes the model to score less on the factual correctness metric (42.22 % vs 47.13 % for the Copy Model). Regarding BLEU score we see an improvement of about 1 point.

I would like to note that I don't think I came even close to the apex of the performance of this configuration as many summaries still show indications of underfitting (e.g. sequences like *"TEAM-A defeated TEAM-A"*). Therefore I do not show an example of the summary generated by the model (as it contains

| Model | Validation Perplexity | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|----------------------|--------------------------|--------------------|--------------|------------------|------------------|
| CopyCS _G | 9.93 | 13.53 | 13.62 | – | – |
| CopyCS _{B5} | | 13.54 | 13.96 | 44.58% | 42.22% |

CopyCS - Joint-Copy decoder + Content Selection encoder
G - Greedy Decoding
B5 - Beam search decoding, beam size = 5

Table 5.3: Performance metrics on the joint-copy model with content selection encoder.

sentences of the same quality as the ones generated by the joint-copy model).

5.5 Content Selection and Planning

The second concept introduced by [Puduppully et al., 2019] is the Content Planning Decoder (discussed in section 4.4.2). The generation is divided into two parts. In the first one, we generate a sequence of pointers to the input records which contain the most essential information from the table. In the second phase we generate the summary according to records pointed to by the extracted pointers. Multitude of added architectures (content selection attention, content planning attention, content planning LSTM) increased even more the memory demands. Therefore we again reduced the embedding and hidden state dimensionalities (figure 5.5).

| Hidden States Dimensionality | Learning Rate | Batch Size | Token | Embedding Dimensionality |
|---------------------------------|---------------|------------|-------------------------|-----------------------------|
| 450 | 0.0002 | 8 | <i>record.type</i> | 300 |
| | | | <i>record.home_away</i> | 300 |
| | | | <i>record.value</i> | 450 |
| | | | <i>record.entity</i> | 450 |
| | | | <i>summary token</i> | 450 |

Figure 5.5: Hyperparameter settings for content selection and planning models.

Content Planning reduces the size of the table, and *arranges* the records into the same order in which they appear in the gold summary. There are two aspects of this solution which should help the model to generate better summaries. Firstly, the input sequences of records become less complex (the maximal length of content plan is 92 compared to more than 800 for the original tables). Secondly, the records are ordered in the same way as they should appear in the summary. [Puduppully et al., 2019] show that this approach led them to really promising results (e.g. they obtained BLEU score of about 16).

5.5.1 Results

During generation we applied two different settings. Firstly we ignore the outputs from the Content Planning Decoder and use the *gold content plan*. (Since we have the gold content plans only for the training and validation parts of the dataset, the reported metrics are calculated only on the validation part) We can see that

with gold content plans model mentions more than 70% of the entities that also appear in the gold summaries (we believe that the number would increase with better quality of the gold content plans⁵).

Figure 5.6 is a good example of the summaries which are generated from the *gold content plans*. Gold summary mentions 13 entities, out of which 12 are mentioned in the generated one. The generated text contains 39 numerical values out of which 31 are correct. The model learned to place the information about the entities in right⁶ order, and to copy numbers related to entities. Although still many flaws persist. I.e. the model generated 3 times the same information about *Richaun Holmes*⁷ Another problem demonstrated in the summary is that model mixed up records about Toronto and about Philadelphia (e.g. model generates sentence : *"Defense was key for Toronto , as they held Philadelphia to 55 percent from the field and 68 percent from three - point range ."*, however 55% shooting for 2pt and 68% shooting for 3pt are the statistics related to Toronto).

The Toronto Raptors defeated the Philadelphia 76ers , 122 - 95 , at philips arena on saturday . The Raptors were able to prevail with a win , as they outscored the sixers by 17 in the second quarter to pull out the win . Defense was key for Toronto , as they held Philadelphia to 55 percent from the field and 68 percent from three - point range . The sixers also dominated the rebounding , winning that battle , 44 - 30 . The sixers (4 - 14) have now lost 5 of their last 6 games , as this marks their fifth win in their last 6 games . Robert Covington was the lone bright spot for the sixers , as he tallied 20 points , 5 rebounds , 2 assists and 2 steals on 7 - of - 11 shooting . Jahlil Okafor was the only other blazer to score double figures , as he totaled 15 points and 5 rebounds . Sergio Rodriguez and Ersan Ilyasova each scored 11 points off the bench . Nik Stauskas was the only other starter in double figures , as he dropped 11 points . Richaun Holmes was the only other starter in double figures , as he dropped 11 points . Richaun Holmes was the only other starter in double figures , as he dropped 11 points . Richaun Holmes was the only other starter in double figures , as he dropped 11 points . Toronto will look to bounce back on sunday in a road matchup against the Orlando Magic . Toronto (11 - 6) have been dominant all season , but a win like this was their best performance of the season . Kyle Lowry once again carried the load , as he accumulated 24 points , 4 rebounds and 8 assists . Terrence Ross was second on the team , as he dropped 22 points on 8 - of - 11 shooting . DeMar DeRozan was the only other starter in double figures , as he accumulated 14 points , 5 rebounds and 5 assists . Jonas Valanciunas recorded a double - double , totaling 12 points and 11 rebounds .

Figure 5.6: A summary generated by the Content Selection and Planning model from the gold content plans. The corresponding gold summary and input table is shown in figure 1.1.

The other generation setting is to use the generated content plans. However this is where we failed to obtain the same results as [Puduppully et al., 2019]. We observed that the network fails to learn when to place *<EOS>* token in the content plan and the generated content plans end with sequence of repetitions. Therefore the part of the model responsible for text generation (Content Plan Encoder and Text Decoder) fails to generate reasonable texts. (Reasonable in

⁵[Puduppully et al., 2019] extract the content plans with specialized information extraction system, and they claim that "strictly speaking, we cannot talk about gold content plans".

⁶The same order as in the gold summaries.

⁷Let me note that there is a numerical value "11 *points*" which is counted 3 times. (Therefore 3 is added to both *all the numerical values* and *correct numerical values*. This causes wrong reduplicated information to decrease the manual metrics more.)

terms of the used language. Table 5.4 shows that model still achieved reasonable entity recall (in fact better than any previous model) and the best correctness, although the correctness is due to large number of repetitions of the accurate information.)

| Model | Validation Perplexity ₁ | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|---------------------|---------------------------------------|--------------------|--------------|------------------|------------------|
| CS&P _G | 8.76 | 13.07 | 13.08 | 44.58% | 67.76% |
| CS&P _G * | | 22.8* | – | 71.43%* | 54.69%* |

CS&P - Content Selection and Planning model
_G - Greedy Decoding
₁ - Validation Perplexity of text decoded from gold content plans
* - All the metrics are computed only on the validation part of the dataset, with *gold content plans*

Table 5.4: Performance metrics on the Content Selection and Planning model.

Since we do not implement the Beam Search decoder for the content plans (due to shortness of time) we try to artificially add *<EOS>* token at some position in the generated content plan and mask everything after, however the results do not show any improvement. The same applies for using Beam Search in Text Decoder.

5.6 Ordered Tables

Since we haven’t managed to obtain sufficient results using any of the previously proposed methods we try to make the task easier. Looking at the results of the CS&P model generating from the *gold content plans*, we may draw a conclusion that the performance bottleneck is in the understanding of the tabular structure. Therefore in following sections we organize and reduce the size of the input tables, and observe the effects of these operations.

Going through the gold content plans as well as the ones generated by Content Selection and Planning model we spotted that their structure follows a simple pattern. At first the teams are presented, then there is some information about the best players and about players who performed surprisingly well.

Therefore we order the input records in a similar way. The first records in the ordered sequence belong to home and away teams. The remaining records contain information about all the players, who are ordered by their point-totals in the respective match.

5.6.1 Results

We trained only the Joint-Copy model on the ordered tables, and we cannot make any clear conclusions. We see a raise in terms of the BLEU score (of about 2 points compared to Joint-Copy model trained on the original tables) but also a drop in both manually evaluated metrics. The overall performance of the model is comparable to Joint-Copy model with Content Selection Encoder. *THE TRAINING OF THE CONTENT SELECTION MODEL IS STILL RUNNING, MAYBE THE CONCLUSIONS MAY CHANGE.*

| Model | Validation Perplexity | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|---------------------------|--------------------------|--------------------|--------------|------------------|------------------|
| Copy _{orderedG} | 9.21 | 12.98 | 13.32 | – | – |
| Copy _{orderedB5} | | 13.75 | 14.01 | 33.73% | 46.84% |

Copy_{ordered} - Joint-Copy model operating on the ordered tables
G - Greedy Decoding
B5 - Beam search decoding, beam size = 5

Table 5.5: Performance metrics on the Content Selection and Planning model.

5.7 Shortened Tables

We took the second effort to make the input tables easier to understand for the network. We order the tables as in previous section. Next we keep all the information about both teams and top three players according to their point totals. The information about the other players is reduced to minutes they played, points and assists they scored, their team and their name. The length of the new sequence of input records is 130.

We opted to use an architecture similar to the one which achieved the best performance on the gold content plans. Following the notation from figure 4.4 we encode the records with Content Selection Encoder, next we use the Content Plan Encoder (bidirectional LSTM operating on the shortened sequences of records) and we feed its outputs to Text Decoder (which uses Joint-Copy mechanism). We call this model *Copy_{pruned}*, its overall hyperparameter configuration is shown in figure 5.7.

| Hidden States Dimensionality | Learning Rate | Batch Size | Token | Embedding Dimensionality |
|---------------------------------|---------------|------------|-------------------------|-----------------------------|
| 500 | 0.0002 | 8 | <i>record.type</i> | 300 |
| | | | <i>record.home_away</i> | 300 |
| | | | <i>record.value</i> | 500 |
| | | | <i>record.entity</i> | 500 |
| | | | <i>summary token</i> | 500 |

Figure 5.7: Hyperparameter settings for Copy_{pruned} models.

5.7.1 Results

This configuration (more sophisticated model, input table that is easier to understand) achieved the best BLEU score of all the models (excluding the one generating from the gold content plans). Table 5.6 shows that we managed to improve the BLEU score by more than 1 point compared to both CopyCS and Copy_{ordered} models. Model also learned to mention more players, however not with correct statistics. Since we reduced the amount of input records related to all but the best three players, the model learned to *hallucinate* the statements about the remaining ones. We haven’t experimented with other possible combinations but we expect that the model performance would improve when about five best players would be fully described in the input tables (compared to actual three).

| Model | Validation Perplexity | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|---------------------------|--------------------------|--------------------|--------------|------------------|------------------|
| Copy _{pruned} G | 11.24 | 14.31 | 14.45 | – | – |
| Copy _{pruned} B5 | | 15.18 | 15.63 | 50.6% | 49.87% |

Copy_{pruned} - the model introduced in section 5.7

G - Greedy Decoding

B5 - Beam search decoding, beam size = 5

Table 5.6: Performance metrics on the Content Selection and Planning model.

This is our final experiment, using the most advanced preprocessing as well as the most advanced model architecture. Therefore I would like to show how far is our best model from the ideal generation system described in the introduction. The generated text in figure 5.8 starts by introducing the teams that have played as well as the score and the location where the match was played. Next it contains some "interesting" events that happened, followed by statistics of the players. Many generated summaries (although not this one) end with a phrase about future schedule of both teams. Since *Jahlil Okafor* isn't between the best three players of the match, *all his statistics excluding his points and assists are hallucinated*. Model mixed up *Robert Covington* with *Terrence Ross*, thus the presented statistics are not correct. Overall we put our expectations too high and we are not able to come even close to any result that could have practical applications.

The Toronto Raptors defeated the Philadelphia 76ers , 122 - 95 , at wells fargo center on saturday evening . The Raptors (11 - 6) came away with a 33 - point second quarter , where the Raptors (11 - 14) opened up a 12 - point lead by the end of the first quarter . The game remained close in the third quarter , as the Raptors outscored the sixers , 31 - 24 . Toronto was led by Terrence Ross , who poured in 22 points on 7 - of - 11 shooting , to go along with 8 assists and 4 rebounds , in 32 minutes . Robert Covington followed up with 22 points on 8 - of - 11 shooting , including 3 - of - 5 from long range , in 23 minutes off the bench . Jonas Valanciunas was solid with 12 points and 5 assists , while Terrence Ross chipped in 22 points on 8 - of - 11 shooting , including 3 - of - 5 from long range , in 23 minutes off the bench . The Raptors shot 55 percent from the field , while the 76ers shot just 42 percent from the field and 41 percent from long range . Jahlil Okafor was the high - point man for Philadelphia , with 24 points on 7 - of - 9 shooting , to go along with 8 assists and 4 rebounds , in 32 minutes . Jahlil Okafor followed up with 15 points , 5 rebounds , 5 assists and 2 steals , in 32 minutes .

Figure 5.8: A summary generated by the Copy_{pruned} model. The corresponding gold summary and input table is shown in figure 1.1.

5.8 Overall Comparison

We start with the Baseline model (section 4.2) which produces fairly good language but many hallucinated statements (only 8% of the produced statements are correct).

Next we experiment with the Joint-Copy model (section 4.3). There we see the number of correct statements to increase to more than 40% and the BLEU score to rise by more than 2 points.

We move on to Joint-Copy model with Content Selection encoder (section 4.4.1). We see another improvement in the BLEU score (by 1 point) and the entity recall (by 5%) however the number of correct facts decreases and the produced language contains many contradictions because we did not manage to find the right combination of the hyperparameters.

The Content Selection and Planning model (section 4.4) is the last one we use for the task. We see that the model achieves great performance on reduced and reorganized tables (the BLEU score bigger than 22) although the part of the model creating the reduced tables does not cooperate well with the part generating text, therefore we do not see any improvement over the Joint-Copy model with Content Selection encoder.

Our last experiments aim to make the sequence of input records more organized and shorter. We use a model introduced previously (Joint-Copy model) and another model with Content Selection encoder with bidirectional LSTM layer on the top and Joint-Copy decoder. On shortened and ordered tables we manage to increase the BLEU score by another point and all the manually evaluated metrics by at least 2 %.

| Model | Validation Perplexity | Validation BLEU | Test BLEU | Entity Recall | Correct Facts |
|---------------------------|--------------------------|--------------------|--------------|------------------|------------------|
| BR _{B5} | 10.59 | 9.99 | 10.6 | 37.35% | 8.03% |
| Copy _{B5} | 9.87 | 12.19 | 12.5 | 39.76% | 47.13% |
| CopyCS _{B5} | 9.93 | 13.54 | 13.96 | 44.58% | 42.22% |
| CS&P _G | 8.76* | 13.07 | 13.08 | 44.58% | 67.76% |
| Copy _{orderedB5} | 9.21 | 13.75 | 14.01 | 33.73% | 46.84% |
| Copy _{prunedB5} | 11.24 | 15.18 | 15.63 | 50.6% | 49.87% |

BR - Baseline, regularized
Copy - Joint-Copy model
CopyCS - Joint-Copy decoder + Content Selection encoder
CS&P - Content Selection and Planning model
Copy_{ordered} - Joint-Copy model operating on the ordered tables
Copy_{pruned} - the model introduced in section 5.7
G - Greedy Decoding
B5 - Beam search decoding, beam size = 5
* - Validation Perplexity of text decoded from gold content plans

Table 5.7: Performance metrics on all the models and approaches discussed in this chapter.

Conclusion

In this thesis, we explored document-scale text generation conditioned on the basketball match statistics. In chapter 1 we presented the RotoWire dataset. We analysed the statistics of the dataset and came up with numerous preprocessing and cleaning methods (chapter 2). The rest of the thesis focused on creating building blocks, composing them into models (chapters 3, 4) and the evaluation and analysis of the experiments (chapter 5).

We followed the path set up by Wiseman et al. [2017]. At first we trained a baseline Encoder-Decoder model with Attention that produced texts of reasonable quality although completely hallucinated facts. Next we added joint-copy mechanism to the Decoder part of the model, which caused massive improvement in terms of entity recall as well as factual correctness (the model is better at selecting important entities and produces correct facts about them). We choose to not implement the Conditional Copy mechanism proposed by Wiseman.

Inspired by Puduppully et al. [2019] we experimented with Content Selection encoder + joint-copy decoder model (CopyCS), and with Content Selection and Planning model (CS&P). While both of the models helped to improve the quality of the generated summaries in terms of all the measured metrics, we are not satisfied with the texts as they contain a lot of repetitions and many times they are not properly finished (the models do not learn to place the $\langle EOS \rangle$ token). We think that it is caused by wrong hyperparameter settings, and a future work may be focused e.g. on applying learning rate decay, using layer normalization etc.

Analysing the content plans generated by CS&P model hypothesized that the performance bottleneck may be the complexity of the input sequence of the records. Therefore at first we ordered the records and then we removed many insignificant records from the inputs and trained CS&P as well as the Copy models on the new data. The results proved our hypothesis as we were able to obtain the biggest BLEU score (which improved from 10 for the baseline to 15 for the CS&P model trained on shortened tables), entity recall and second biggest factual correctness out of all the trained models.

However the generated summaries do not match any of our expectations. While our best model learned to extract information about the teams, it still suffers from selecting wrong players to talk about (as the entity recall for our best model is only about 50%), and copying wrong information (the factual correctness is on the same level as the entity recall).

We believe that future work should focus on improving the Content Planning mechanism, as we saw that CS&P model generating from gold content plans hugely outperformed our best model on the BLEU score (22.8 vs 15.1), the Entity Recall (71.4% vs 50.6%) as well as on the factual correctness (54.69% vs 49.87%). We are also interested in the Transformer architecture for text generation [Vaswani et al., 2017], specifically we would like to try the few-shot setting [Chen et al., 2020].

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>.
- Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks, 2015. URL <https://arxiv.org/abs/1506.03099>.
- Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. Evaluation of text generation: A survey, 2021.
- Zhiyu Chen, Harini Eavani, Wenhui Chen, Yinyin Liu, and William Yang Wang. Few-shot nlg with pre-trained language model, 2020.
- Kyunghyun Cho, Bart van Merriënboer, 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 Merriënboer, 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>.
- Sean Dailor. What does embedding mean in machine learning?, 2019.
- 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.

- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- Remi Lebrete, 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>.
- 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/>.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002. doi: 10.3115/1073083.1073135.
- 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>.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435.
- 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.

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.
- 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

| | | |
|-----|---|----|
| 1.1 | An example of the team statistics from the development part of the Rotowire dataset | 6 |
| 1.2 | An example of the player statistics from the development part of the Rotowire dataset | 6 |
| 2.1 | An example of structured data | 9 |
| 2.2 | Statistics of tables as used by Wiseman et al. [2017] | 10 |
| 2.3 | Statistics of summaries as used by Wiseman et al. [2017] | 11 |
| 2.4 | Occurrences of tokens in summaries from dataset RotoWire | 11 |
| 2.5 | Overlap of train dataset summaries and valid/test dataset summaries | 12 |
| 2.6 | Occurrences of tokens in transformed summaries from dataset RotoWire | 19 |
| 2.7 | Overlap of transformed train dataset summaries and valid/test dataset summaries | 19 |
| 2.8 | Length statistics of the preprocessed tables. | 19 |
| 2.9 | Lenght statistics of the preprocessed summaries. | 19 |
| 4.1 | An extract from the content plan corresponding to the summary from the figure 1.1 | 36 |
| 5.1 | Performance metrics on the baseline models. | 41 |
| 5.2 | Performance metrics on the joint-copy models. | 42 |
| 5.3 | Performance metrics on the joint-copy model with content selection encoder. | 44 |
| 5.4 | Performance metrics on the Content Selection and Planning model. | 46 |
| 5.5 | Performance metrics on the Content Selection and Planning model. | 47 |
| 5.6 | Performance metrics on the Content Selection and Planning model. | 48 |
| 5.7 | Performance metrics on all the models and approaches discussed in this chapter. | 49 |