ZHAW
Zurich University of Applied Sciences

Bachelor thesis

FS 2018

Natural Language Generation from Structured Data

Authors:
Tobias Fierz
Jerome Schaub

Supervisors:
Dr. Mark Cieliebak
Jan Deriu

June 6, 2018

Zurich University
of Applied Sciences

School of Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirm that this work is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the project work have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date: Signature:

....

.....

.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all project works submitted.

Zusammenfassung

Es hat sich mehrfach gezeigt, dass neuronale Netzwerke gut im Gebiet der Natural Language Generation funktionieren. Ein Problem, welches sie zurzeit aber noch haben, ist, dass eine grosse Menge an speziell vorbereiteten Daten benötigt werden.

Das Ziel dieser Arbeit ist es festzustellen, ob ein bereits existierendes neuronales Netz, das gut darin ist, Restaurant Reviews zu generieren, auch mit keinen oder nur geringen Anpassungen in der Lage ist ähnlich gute Resultate im Generieren von Laptop Reviews zu erreichen. Ein weiteres Ziel ist es herauszufinden, wie schwer es ist ein passendes Datenset für ein neuronales Netz zu erstellen.

Zuerst wurden Daten von einem Onlinehändler gesammelt, die wichtigsten Informationen extrahiert und das Ganze in einem Datenset zusammengefasst. Anschliessend wurde das existierende neuronale Netz auf dem erstellten Datenset trainiert.

Die Resultate zeigten, dass die Outputqualität eines neuronalen Netzwerks in grossem Zusammenhang mit der Qualität des Datensets steht. Nicht nur die Grösse des Datensets selber, sondern auch Eigenschaften wie die Grösse des Vokabulars (Anzahl einzigartiger Wörter) spielen eine grosse Rolle. Da das erstellte Datenset noch sehr "unsauber" war, führte es dazu, dass auch der Output schlecht war. Ausgehend von diesen Resultaten musste die Problemstellung der Arbeit vereinfacht werden. Die zweite Problemstellung beinhaltete die Frage, wie gut ein neuronales Netz beim Ordnen von ungeordneten Sätzen und beim Füllen von Lücken in Sätzen ist.

Abstract

Neural networks work well in the field of natural language generation, the problem being that they require a great number of data and usually have to be specifically adapted for the task at hand.

This thesis aims to determine whether an already established network that performs well in one domain (restaurant reviews), also does so in another one (laptop reviews) with no or only minor adjustments to it. It also aims to establish how difficult it is to create a suitable dataset for a neural network.

First, data was gathered from an online laptop retailer. Then the most important information was extracted and summarized in a dataset. Finally the already established neural network is trained on said dataset.

Results illustrated that the quality of the dataset is one crucial part of a well performing neural network. While the total size of the dataset is clearly important, properties such as the vocabulary size (number of unique words) also play a significant role. The output quality of the neural network quickly drops when trained on a dataset created out of "noisy" data. In the end it transpired that given the created dataset, the neural network did not perform as expected and the task had to be simplified to trying to solve the problem of bringing shuffled sentences back into the right order and filling gaps in sentences.

Contents

1. Introduction

In the project preceding this bachelor thesis it was shown that a relatively basic neural network already performs well in the task of generating sentences when given a feature vector [11]. The dataset used in said project consists out of approximately 42000 restaurant reviews and was specifically prepared for a natural language generation challenge, meaning it was a very "clean" dataset. The similarities in the sentence structures and a small vocabulary simplified the learning process for the neural net.

The goal of this bachelor thesis was to determine whether the neural network built in the previous project can also be used in other domains. It was decided to focus on laptop reviews, because the challenge is similar to restaurant reviews. The difference was that the features which can be generated are now for example CPU, GPU, RAM and other laptop specifications instead of food type, area, price and other attributes. One of the main challenges was the choice of the dataset. Since there was no already existing set available one had to be created. This meant that review data had to be crawled from the internet with the high risk of getting a lot of "noise" in the created dataset and a most likely large vocabulary compared to the restaurant review one. These factors could have a great influence on the neural nets ability to recognize patterns and learn efficiently.

This thesis aimed to answer the following questions:

- What are the challenges when creating a dataset for a neural network out of data crawled from the internet with the goal of generating natural language?
- Does the network built in the project thesis work at all?
  - If so, how well does it perform?
  - If not, what adjustments need to be made for it to work / perform well?

This thesis is structured as follows: First the planning chapter includes the meeting summaries that explain what was done and when it was discussed. Then some basics needed for this thesis are explained followed by the creation and analysis of the dataset used. The next chapter explains the architectures of the models created for this thesis. Finally the results of the individual models are displayed and analyzed followed by a short conclusion of the thesis.

2. Related Work

Different approaches exist to build a Natural Language Generator. One approach that is currently popular is to use Neural Networks to build language models, which was done for this bachelor thesis and is based on the project "Natural Language Generation using Neural Networks"[11]. The goal of the preceding project was to set up a neural network for the E2E NLG challenge [10], which in the end took a feature vector into consideration and generated meaningful text. In this thesis said neural net was used as a base and adjustments were made where needed. It consisted of two inputs, one with the already generated text that was feed to an LSTM and a second with the features to be generated. Wen et al.(2015) [14] used a similar approach. They created a semantically controlled LSTM cell where they extended the LSTM cell with the information on which features it has to generate. This approach led to good results. Juraska et al. [8] used an ensemble of 3 different sequence-to-sequence models as the generator. Two of the models used bidirectional LSTM as an encoder, the third model relied on a CNN encoder. In order to have different results from the 2 LSTM encoders, they were trained individually for a different number of epochs. In the end the output probabilities of the top 10 candidates from the models were combined and the utterance with the highest score was predicted. This ensemble approach increased the overall output quality of the system.

The second part of this thesis deals with the challenge of bringing words of a sentence into the right order, which is a difficult task. It depends on different factors such as "syntactic structure, selective restrictions, subcategorization,and discourse considerations" according to Elman[5]. There are different approaches to solve the task. A simple approach is to use n-gram language models. These models can already perform well as Liu and Zhang showed [9]. They compared the n-gram word modeling and syntactic language models for word ordering. Syntactic language models also rely on statistical language models such as for example n-grams but they are extended with a discriminative language model which is able to consider syntactical information according to Kaufmann and Pfister[13]. Liu and Zhang found that the syntactic language model performs better then a simple n-gram model, but a combination of both of them gets even better results[9].

Another approach to the problem is the usage of LSTM lanugage models, which was done by Schmalz, Rush and Shieber [12]. For this task no explicit syntactic information was provided to the model. They also compared their model to a n-gram model and showed, that the LSTM solution outperformed the n-gram approach. Hasler et al. [1] came up with a bag-to-sequence model that was inspired by a sequence-to-sequence model with attention. They compared their model with a normal seq2seq model and with n-gram models. Even though it outperformed both of them, they called the seq2seq model a strong baseline.

A question for both of these problems is how does one evaluate the output quality. A measurement that is often used is the so called BLEU score, which originated from the field of machine-translation. It is however only considered to be a sufficient meassurement to give a vague idea on how well the output is, as Juraska et al. [8] concluded in their paper. The same conclusion was found by Gatt and Krahmer [2] in a study where they did a survey on the start of the art in Natural Language Generation. It was found that the BLEU score and human evaluation differ and the correlation between these two measurements is not always given. Their conclusion was that it would be helpful to look at different measurements to assess the output quality.

3. Planning

This chapter contains some information on how the planning of this thesis was done. The first three weeks mainly consisted out of crawling the necessary data, creating and analyzing the dataset. For the remaining weeks the topics addressed can be viewed in the meeting summaries. After the last meeting the main focus was the writing of this thesis.

3.1. Meeting Summaries

3.1.1. Meeting Week 4

- Create tree to visualize features of CPU, GPU, etc. (powerful, energy saving, ...)
- Determine most important specs
  - Find text parts that describe said specs
  - Find values / keywords of specific features (e.g. price → high, moderate, ...)
- Fix missing product names in Notebookcheck dataset
- Extract section titles of Notebookcheck reviews
- Try to find sentences on Google describing specific CPUs

3.1.2. Meeting Week 5

- Check if a rule of thumb for battery runtime exists
- Extract price expression using Regex
- Remove $ prices from dataset
- Create first dataset out of extracted sentences for feature
- Write down analysis details for report

3.1.3. Meeting Week 6

- Check out external reviews (e.g. Techradar,...)
- Compare seed words for prices (e.g. moderate) to actual price → mapping?
- Create price distribution statistics (e.g. for specific CPU / GPU models)
- Test generation with CPU and GPU features

- Check number of sentences extracted for all features

3.1.4. Meeting Week 7

- 1st Phase: Try generation without feature vector
- 2nd Phase: Add feature vector
- Try CNN stack instead of LSTM
- Calculate word count leaving out words with less than e.g. 5 occurrences
- Switch from character to word-based
- Try sentence generation for e.g. GPU given keywords
  - Try with large example dataset beforehand (e.g. Shakespeare, Bible, ...)
  - Given a vector of words generate a sentence containing them all

3.1.5. Meeting Week 9

- Try a 2D CNN instead of 1D (set length or width to matrix length)
- Start generation with word instead of empty string
  - Try with different words and use beam search
- "Sanity-checks":
  - Non-sorted input
  - Assign word-to-int randomly instead of in order
- Set up NN that tries to complete one or multiple words
- Calculate BLEU score or similar (e.g. Rouge, Meteor,...) on output
- Try with restaurant dataset at first
- Find spell-checker API

3.1.6. Meeting Week 10

- Simplify problem: Give the NN a choice of two words to fill a gap
- Leave out the dense layer after one-hot
- Use vector with word ids instead of one-hot encoding
- Analyze the softmax output
  - Correct word should be high up
  - Try setting up a choice between the correct and a low value word
- Set up or find a NN for the dot product

3.1.7. Meeting Week 11

- Try prediction in different direction - are there different results?

- Try 2-grams with 2 gaps (input = bag of 2-grams)
- Try seq2seq model for prediction
- Analyze Keras LSTM cell code - check possibility of automatically setting a generating word to 0 in the input vector

3.1.8. Meeting Week 12

- Check correlation between BLEU score and sentence length
- Attention mechanism in Keras?
- Set up seq2seq model with word-dropout to randomly set gaps by adapting the dropout layer of keras
- "Sanity-Checks":
  - Calculate F1-Score over all words
  - Are all words present?
  - Are additional words added?
- Check if the bad BLEU scores are in correlation with the content of the dev- / trainset
- Input words in non sorted order (random shuffle)
- Analyze what kind of gaps the NN can fill and where it has problems

3.1.9. Meeting Week 13

- Take a closer look at softmax output of wrongly replaced words (list top 5)
- Compare word statistics of train- and devset
- Fix bug in NN (e.g. Riverside appears way too often)

4. Basics

In this chapter specific terms, technologies and methods mentioned or used in this thesis are explained in greater detail. This includes the used classes of neural networks such as the **CNN** and the **RNN** and also specific models such as the **sequence-to-sequence** model. A more general introduction to neural networks can be found online[1].

4.1. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specific class of a neural net that is most often used in the area of image recognition. Other applications include video recognition, recommender systems and also natural language processing. CNNs take inspiration from biology and how the 'visual cortex' works[2]. They use a set of special layers, starting with the **Convolutional layer**.

Convolutional layer The convolutional layer (short: conv layer) is always the first layer in a CNN and applies a convolution operation to the input, which reduces the number of free parameters and therefore resolves the problem of vanishing or exploding gradients.

Pooling layer The pooling layer is another special layer used in CNNs. There are different kinds of pooling layers, such as the **max pooling layer**, which uses the maximum value from each of a cluster of neurons at the prior layer, or the **average pooling layer**, which is similar to max pooling but uses the average value. The goal of this layer is to down-sample an input representation and reduce its dimensionality.

Fully connected layer The fully connected layer does, as its name implies, connect every neuron in one layer to every neuron in another layer. The output of the conv and pooling layers represent high-level features of the input data. The goal of the fully connected layer

[1] https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
[2] https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

is to to use these features to classify the input into a set of classes based on the training dataset.

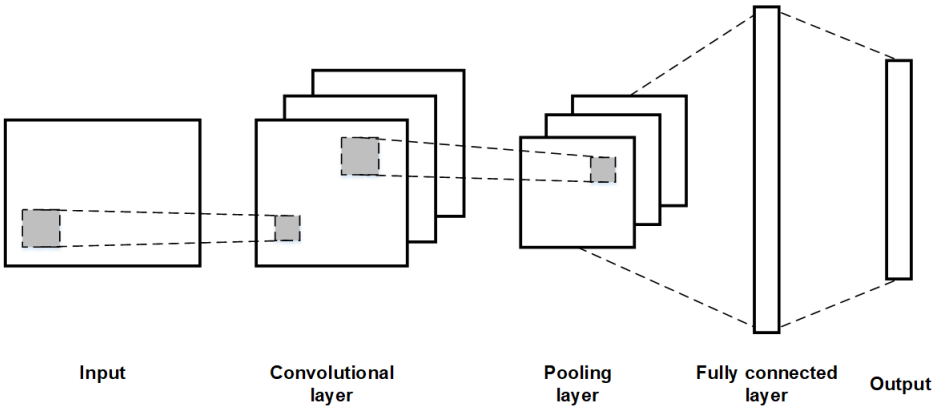

Figure 4.1.: Visualization of a simple CNN model

4.2. Recurrent Neural Network

A Recurrent Neural Network (RNN) is another specific class of a neural net that can use its internal state to process sequences. RNNs do not only take the current input example as their input but also what they have perceived previously in time. This 'memory' allows the net to preserve sequential information that normal feedforward networks cannot. As the equation 4.1 and figure 4.2 show, at the time step t the current hidden layer h_t is not only influenced by the current input x_t but also by the state of the previous hidden layer h_{t-1}. In equation 4.2 the output is then simply calculated using the softmax function, which squashes the outputs of each unit to be between 0 and 1 and divides each output so that the total sum will be 1.

$$h_t = f(W_{xh}x_t + W_{hh}h_{h-1}) \tag{4.1}$$

$$o_t = softmax(W_{hy}h_t) \tag{4.2}$$

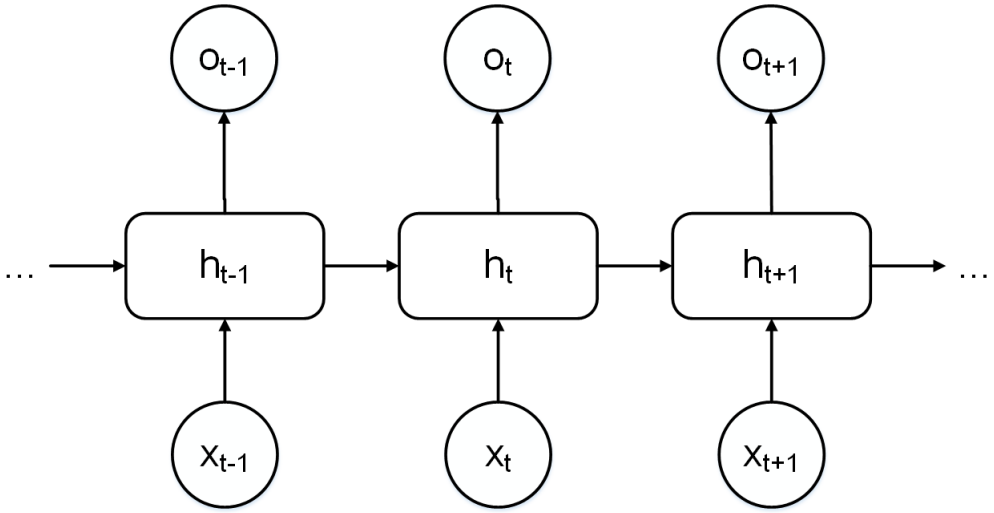

Figure 4.2.: Basic structure of a RNN

4.3. Long Short-Term Memory Network

Long short-term memory (LSTM) networks are a special form of RNNs. They were introduced in 1997 by Hochreiter & Schmidhuber [7] in order to solve the vanishing gradient problem of vanilla RNNs. They are very useful in the area of natural language processing, due to being able to understand long-term dependencies in sentences. LSTM networks are composed of so called LSTM units. LSTM units possess an internal cell state C_t, an input gate i_t, an output gate o_t and a forget gate f_t.

Forget gate At the beginning this layer decides how much information from the cell state should be forgotten. The layer outputs a number between 0 and 1 (0 = forget all information, 1 = keep all information).

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{4.3}$$

Input gate The input gate decides what new information is going to be stored inside the cell state C_t.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{4.4}$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C) \tag{4.5}$$

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{4.6}$$

Output gate The output gate is responsible for the LSTM units output and will be based on a filtered version of the cell state C_t.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{4.7}$$

$$h_t = o_t * tanh(C_t) \tag{4.8}$$

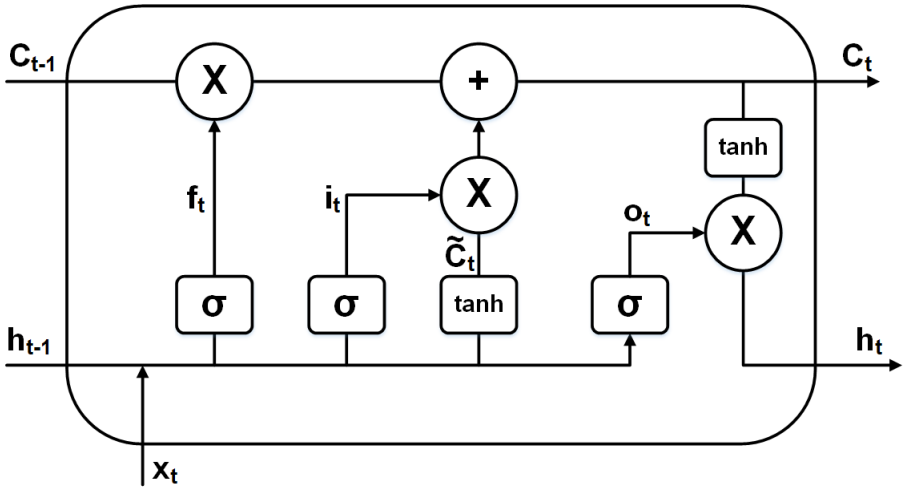

Figure 4.3.: Structure inside of an LSTM unit

4.4. Sequence-to-sequence model

Sequence-to-sequence (seq2seq) is a special type of neural network training model where sequences from one domain are converted to sequences in other domains. This model is useful for a variety of problems such as machine translation, where the input might be a sentence in German and the output the translated sentence in English. A seq2seq model is usually built out of two or more RNN layers.

Encoder One RNN layer (or multiple) act as an encoder that processes the input and returns its own internal state while the output itself is ignored.

Decoder Another RNN layer (or multiple) then acts as a decoder and is trained to predict the next characters or words of the output sequence given what has been generated up to now.

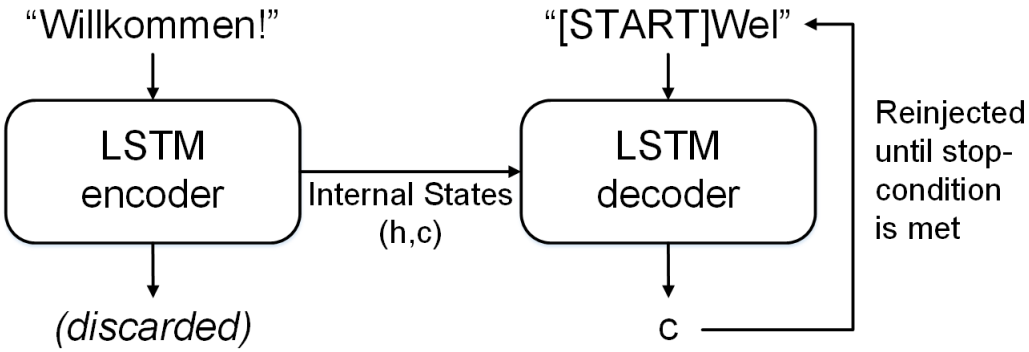

Figure 4.4.: Visualization of the sequence-to-sequence model

5. Dataset Analysis

The following chapter takes a closer look at the datasets used in this thesis. First, the sources of the dataset and the approach to crawling them are explained. Afterwards, the created datasets are analyzed and statistics such as vocabulary size are displayed.

5.1. Dataset Source

A thorough internet search revealed that there was no existing dataset suitable for the task, so one had to to be created. The requirements for the dataset were as follows: It has to contain structured data regarding the specifications of a laptop (CPU / GPU model, RAM, screen size, ...) and some sort of text that is related to said specifications. Two popular websites presented themselves as useful to gather said data: **Newegg** and **Notebookcheck**.

Newegg The website "newegg.com" is a popular US online retailer selling computer hardware and consumer electronics. There were 9412 laptops available on that site, which usually contain a short description, a commercial block from the manufacturer and a block with the specifications of the given model.

Notebookcheck "notebookcheck.net" is a website that contains in-depth reviews about laptops, tablets and smartphones. The structure of a review is as follows: After the title there is a short intro text followed by the detailed specifications about the model. The review itself is then structured into several subsections containing detailed information about the different specifications or aspects of the model such as the display, case or performance. A total of 1942 English reviews were crawled from said website. An example review can be found in the appendix.

5.2. Crawling

For the crawling of data from the websites mentioned in the Dataset Source section, two similar Python scripts were created which work as follows: In a first step all the links

to the individual model pages are collected. While this was uncomplicated on Newegg because it is possible to display all laptops at once, the task turned out to be a challenge on Notebookcheck. The site does not contain the functionality the display all reviews on one single or multiple pages, so the search function has to be used with different parameters to get as many distinctive reviews as possible. Using different manufacturers and price ranges 1942 review links could be gathered. Afterwards the HTML files of the individual model pages are collected and in a final step the data is extracted from said HTML files with the help of certain attribute values and saved to a CSV file. Additionally simple cleanup of the text, such as the removal of certain HTML tags, was also done. Certain HTML tags were left in the dataset in hopes of the neural net generating similar text structure and formatting with the help of for example <b>bold or <p>paragraph tags.

5.3. Newegg Analysis

The reviews consisted of a total of **92 unique characters** and **25657 unique words**, which is significantly more than the restaurant review dataset from the preceding project work. The large disparity between the amount of unique words and the number of texts was a finding which turned out to be problematic later on during the learning phase. One reason for the large quantity of unique words were the often occurring unique product or part names and their specifications. If only words appearing more than once were counted, the number is reduced to **17800**. Table 5.1 shows additional more detailed results from the text analysis.

Unique characters	92
Unique words	25657
Unique words with more than 1 appearance	17800
Unique words with more than 2 appearances	13511
Unique descriptions	5259
Unique titles	6352
Unique specifications	4991
Average description text length	≈ 1624

Table 5.1.: Detailed results from the Newegg dataset analysis

5.4. Notebookcheck Analysis

The reviews consisted of a total of **85 unique characters** and **32200 unique words**. For the same reason as with the Newegg dataset, this is considerably more than the restaurant review dataset. If only words appearing more than once were counted, the number was reduced to **19765**. Table 5.2 shows additional more detailed results from the text analysis.

Unique characters	85
Unique words	32200
Unique words with more than 1 appearance	19765
Unique words with more than 2 appearances	13586
Unique reviews	1073
Unique titles	1710
Unique specifications	1346
Unique intro texts	796
Average review text length	≈ 2775
Average intro text length	≈ 281
Number of authors	34
Number of translators	26

Table 5.2.: Detailed results from the Notebookcheck dataset analysis

5.5. Dataset Choice

After a short analysis it was decided to simply work with the Notebookcheck dataset at first, because the style of the written text was more suitable for the goal of this thesis. The dataset contained specific information about the specifications of a laptop (e.g. screen size, brightness, response time, etc.) compared to the Newegg dataset, where most of the texts were 'commercial texts' written by the manufacturer and thus not containing very specific information.

5.6. Feature Analysis

For the learning and generation process a feature vector similar to the one used in the project work had to be introduced. The primary goal of the thesis is being able to generate a review text based on given specifications such as CPU, GPU and RAM, so the feature vector had to represent the wanted specifications. Each review from Notebookcheck contained structured data about the specifications of a laptop in form of a table. An example entry can be seen in table 5.3.

Processor	Intel Core i7-8550U (Intel Core i7)
Graphics adapter	NVIDIA GeForce GTX 1050 (Notebook) - 4096 MB, Core: 1354 MHz, Memory: 1752 MHz, GDDR5, 23.21.13.8792
Memory	8192 MB, DDR4 (Dual Channel 2 x 4 GB)
Display	15.6 inch 16:9, 1920 x 1080 pixel 141 PPI, yes, native pen support, Chi Mei CMN15D7, IPS, glossy: yes
Mainboard	Intel Kaby Lake-U iHDCP 2.2 Premium PCH
Storage	Micron 1100 MTFDDAV256TBN, 256 GB, + 1000 GB HDD (Western Digital WD10SPZX), 190 GB free
Weight	2.2 kg (= 77.6 oz / 4.85 pounds), Power Supply: 320 g (= 11.29 oz / 0.71 pounds)
Price	1299 Euro
Links	Acer homepage, Acer notebook section

Table 5.3.: Example of structured data available for the review of Acer Nitro 5 Spin NP515

Table 5.3 shows that the given data was very specific, meaning that most likely there was not enough data for each feature and the feature vector would have become too large. Table 5.4 confirms this claim, showing that the feature with the highest occurrence was "Operating System: Windows 10 Home 64 bit", which is not really a useful feature because the goal of the thesis was to concentrate on the hardware specifications. Table 5.5 shows that the top CPU was only present in 153 reviews and merely a total of 4 CPUs were present in more than 50 reviews. This was not nearly enough data for the neural network, meaning that not only did some features like operating system have to be omitted, but also that the remaining ones had to be generalized somehow.

Feature	**Value**	**# of Occ**
operating system	microsoft windows 10 home 64 bit	664
memory	8192 mb	531
memory	4096 mb	321
memory	16384 mb	294
operating system	microsoft windows 10 pro 64 bit	264
links	asus homepage	257
camera	webcam: hd	253
memory	2048 mb	241
links	acer homepage	237
storage	32 gb emmc flash 32 gb	234

Table 5.4.: Top 10 feature values according to occurrence

Feature	**Value**	**# of Occ**
processor	intel core i7-7700hq	153
processor	intel core i5-7200u	124
processor	intel core i7-7500u	75
processor	intel core i7-6700hq	67
processor	intel core i7-8550u	45
processor	intel core i5-8250u	45
processor	intel core i7-6500u	43
processor	intel core i5-6200u	41
processor	qualcomm snapdragon 625	31
processor	mediatek mt6750	30

Table 5.5.: Top 10 values for processor feature according to occurrence

5.6.1. Extending the Dataset

For a majority of the features there were not enough examples in the training set, so a set of different approaches were analyzed to extend the dataset. One of them was to crawl data from other sites that also write reviews for notebooks. The problem with that was that the writing style on other sites was very different from the Notebookcheck

style. It was often more subjective. A different idea was to just google for a specific CPU name that appears in the training set and get different sentences for this specific CPU. In the end, this approach had the same problem as above. Usually the sites found like this were either forums or vendor sites, which compared to Nootebookcheck had totally different styles of writing. That meant that there was no easy way to extend the dataset size.

5.7. The Feature Vector

To shorten the feature vector certain specifications such as CPU had to be generalized and other specifications had to be left out completely. In Figure 5.1 the chosen feature vector is visualized. For example the CPU models simply contains the generation number for Intel processors and AMD processors are generalized altogether due to their low occurrence. The rest of the features were chosen according to the importance in the view of a consumer. The rationale being that when reading a review a consumer is more likely to look at specifications such as screen size and battery lifetime instead of for example the exact number of USB ports.

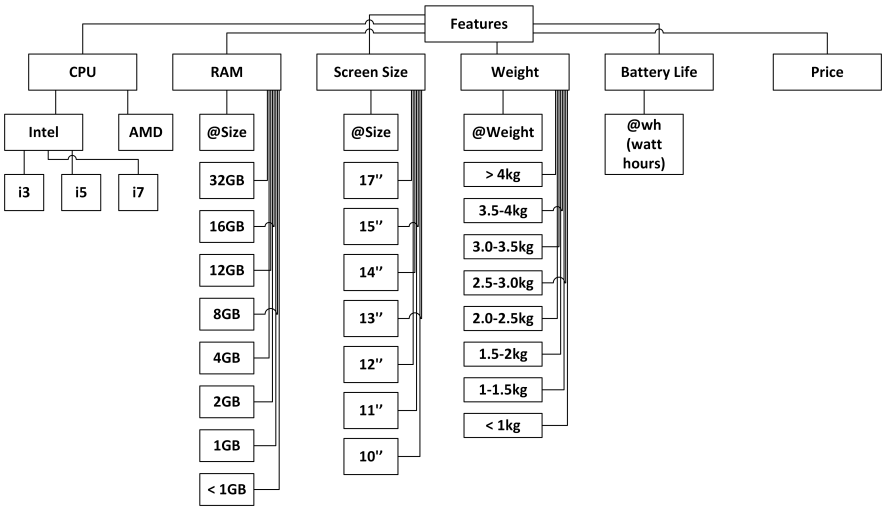

Figure 5.1.: Feature vector visualized

5.8. Dataset Creation

Although the crawled dataset already contained structured data (the specifications of the laptop) and the associated review, directly feeding it to the neural network would not have been a good idea due to the fact that there was still a lot of unrelated data ("noise") present. For example there were entire subsections in the review regarding the case or temperature and energy statistics of the reviewed laptop. This data was not represented in the chosen feature vector in any way, meaning that the sentences related to the

specifications in the feature vector had to be extracted somehow.

5.8.1. TFIDF Analysis

The goal of the TFIDF analysis was to find certain expressions that are used together with a specific CPU so that sentences containing said expression could be extracted and added to the dataset. Table 5.6 displays the top results of the TFIDF analysis of all reviews containing the Intel Core i7 CPU. Unfortunately the results turned out to be poor because most of the words in the list were not CPU related at all or can also be used in another context. The results for 2-grams as shown in Table 5.7 were not much better.

h3	0.41756883707808
h2	0.2522290804805468
performance	0.19659490907343738
display	0.14221094376536408
test	0.13908542851777367
core	0.1336157768344904
gaming	0.1303339858245205
15	0.111893445863737
gpu	0.11064323976470085
notebook	0.10564241536855618
cpu	0.10454848503189952
model	0.10314200317048383
battery	0.10001648792289342
i7	0.0998602121605139
device	0.0990788333486163
gb	0.09579704233864635
pro	0.09376545742771258
review	0.09345290590295353
power	0.09079621794250169
gtx	0.08923346031870646

Table 5.6.: TFIDF analysis for reviews containing a Intel Core i7 CPU

core i7	0.20620272689651564
h3 h3	0.13956724008771798
performance h3	0.12959815151002382
h2 h3	0.122777196167391
geforce gtx	0.11280810758969685
card reader	0.11175872984467641
current prices	0.11175872984467641
h5 current	0.11175872984467641
prices h5	0.11175872984467641
xps 13	0.1023143301394925
battery life	0.09969088577694141
original german	0.09496868592434944
kaby lake	0.09444399705183923
german review	0.09234524156179835
original german	0.09182055268928814
intel core	0.08867241945422683
input devices	0.08499959734665531
17 inch	0.08395021960163487

Table 5.7.: TFIDF analysis for reviews containing a Intel Core i7 CPU with 2-grams

5.8.2. K-Means

Additionally to the TFIDF analysis the K-Means algorithm was used. The rationale behind this was the assumption that similar terms should end up in the same cluster. The TFIDF scores were used as input values. It is then possible to show the dimensions which have the most influence on the cluster center. Example clusters are shown in Table 5.8. In the first row of the table is a sentence in which one of the top terms appears. It is then followed by the 10 most influential terms in this cluster.

the resolution of 1280x720 pixels is on the standard level for the price range	*as already mentioned the lenovo thinkpad t470p clearly ranges in the high-price segment*	*some less expensive contenders do a better job with that*
resolution	thinkpad	job
1280x720	x1	far
inch	yoga	perform
1920x1080	t470	convertibles
pixel	13	respect
1280x800	2017	rivals
moto	carbon	exist
1280	t470s	things
level	oled	look
density	l560	total

Table 5.8.: Top 10 of 3 different K-Means clusters

The first two examples in Table 5.8 both seem to represent a cluster with very specific vocabulary. The cluster on the left seems to focus on screen resolution while the one in the middle seems to be a Lenovo cluster that includes the different model names. Clusters like these were very helpful for the analysis, because they showed a lot of different terms for certain features. However, not all clusters seemed to be specific in respect to a certain vendor or specification. An example for a more general cluster is given on the right. Clusters like these did not provide additional information for this task.

5.8.3. Entity Recognition

Another approach was to use Spacy[1], a tool that is designed for entity recognition. Spacy comes with some pretrained entities such as organizations, countries, celebrities and more. Entities can then be detected in a sentence. For example in the sentence "Microsoft release a new Windows version today" Spacy would predict "Microsoft" to be an organization and would also return the start and end position of the entity, which in this case is 0 respectively 9. It is possible to train new entities, which was tried with GPUs. However, the problem with the training was that Spacy expects a sentence as training data and the start and stop position for the entity that is in in. The main goal of this approach was to extract exactly this information in order to create the Meaning Representation (mr) for the natural language representations. As a workaround it was tried to use the GPU name from the structured data, search its position in the sentences and then train Spacy with this information. The intention behind this was the assumption that Spacy might recognize certain structures in the sentences and then could detect GPUs even if the model name was different. Unfortunately Spacy did not generalize well with this training. For example in the sentence "There is a NVIDIA Quadro K2100M built in", it would detect "NVIDIA Quadro K2100M" to be the graphics card. If the sentence is now changed to

[1] https://spacy.io/

"There is a NVIDIA K2100M built in", it does not detect the GPU. Unfortunately the Spacy approach did not lead to any useful results because of the bad generalization and the lack of a special trainset for this task.

5.8.4. Test-Dataset for CPU

In order to conduct a test-run where the goal was simply to generate meaningful text, it was decided to first only create a dataset for sentences regarding the CPU. This was done by extracting sentences from reviews that met the following two conditions: They have to contain at least one of the terms under the paragraph *terms* but cannot contain one of the 'filter terms' listed under the paragraph *filter terms*. The two sets of these terms were constructed using k-means clustering (c.f Chapter 5.8.2). Table 5.9 shows some example sentences which resulted out of using this method. While most of the sentences are talking about the CPU there are also a few outliners that are referencing for example the gpu or network adapter. Another problem is that a lot of sentences still contained useless information such as comparisons with other CPUs.

Terms cpu, core, processor, ghz, i7, cache, 6m, i5, 3m, 7700hq, 8m, 4800mq, 6300u, 4900mq, 7820hq, 4700mq, 7th, 6th, 7200u, quad, 6700hq, 6820hq, 4200m, e3, v6, xeon, cache, ghz, 1505m, 8mb, 4m, 7500u, 6600u, 4600u, 6500u, 4ghz, 4500u, 3667u, 1ghz, 6th, 5ghz, turbo, ram, gpu, ssd, hdd, @nbname

Filter terms score, noise, benchmark, cd/m, temperaturesof, germanreview, keyboard, 4k, fan, compar, brightness, prime95, cinebench, usb, thunderbolt, kabylake, sandybridge, watt, observe, test, diagram, hyperthreading, behavior, framerate, batterymode, acpowermode, tdprestriction, alsooffers, coolboost, pascal, fasterthan, marketing, so-dimm, touchpad, pcmark, measure, pcie, alsoavailable, turbo, change, disassembled, exterior, cover, decline, ramslot, competitors, tablet, moreinformation, furtherinformation, degrees, outperform, defeats, %, predecessor, successor, apparently, appear, tdp, directx, unlike, rivals, reach, crystaldiskmark, 3dmark, furmark, depending, competition, plugged, thrott, warranty, cheaper, replace, identical, skylake, review, overclocking, xperia, smartphone, overclocked, marshmallow, landscape, panoram, camera, galaxy, -ak033ng, gsm, nfc, sim, phone, gorilla

specs-wise it is powered by a @ghz intel @cpu 8 gb of lpddr3 ram and a 256 gb ssd
at the heart of the predator helios 300 is an overclockable @gpu or 1050ti gpu combined with a 7th-gen intel @cpu(7700hq) or @cpu (7300hq) for outstanding performance
the current @cpu is probably the most widely used intel kaby-lake ulv processor which is often encountered in office or business and multimedia notebooks in particular
to include intel processors in the evaluation we also used the hp 250 g5 lenovo's ideapad v110-15ikb and the asus vivobook x510ua-br305t
intel has provided a fast quad-core processor in the form of the @cpu
even though it is not very up-to-date anymore thanks to the intel dual band wireless-ac 7265 mimo-2x2 module with bluetooth 4
intel specifies this rate as the maximum clock when four cores are loaded simultaneously
intel has specified a base clock of @ghz for this chip

Table 5.9.: Example sentences out of the created CPU dataset

When this data is compared to the sentences from the NLG Challenge, it becomes clear that the domain is much broader. For example, if the feature *family-friendly* in the NLG Challenge was set to 'yes', there was only a limited amount of expressions for that feature. In the current dataset however, this is not the case. If a review to an intel core i-3 should be generated, there are still plenty of different aspects such as the clock speed, if it is overclockable, the number of cores, the generation number, the exact model name and more, which can be mentioned. This would not be a problem, if there were enough mentions of all of these aspects. The feature vector could then simply be extended, so that this specific features can be generated. However, Table 5.4 shows that there was already a low amount of CPU references, so it was not possible to split them any further. The same problem applies for other features such as the GPU, RAM or screen as well. For the battery life there was an additional inconvenience: The structured data contained the battery life in the form of Watt hours of the battery. In the reviews however, the battery life was usually mentioned in hours. A direct mapping between these two different measurements is not possible, because how long a laptop can run depends on different factors, such as the hardware that is built in. An additional GPU for example can shorten the battery life by a large amount. It was decided to ignore this problem and focus on CPU generation at first.

5.9. NLG Challenge Dataset

The trainset of the NLG challenge was already analyzed during the project thesis related to this work [11]. The main focus there was to give an overview on how the different features in the dataset can be expressed. But more relevant for the task of word ordering is the comparison of the distribution of words between the train- and devset.

In the trainset there are 42061 different reviews. The devset contains 4672 reviews. For this analysis all restaurant names were replaced through the token "@name" and all places through the token "@near". Furthermore everything was converted to lowercase. After

the preprocessing there were 2760 different words in the trainset and 1000 different words in the devset.

Out of these 1000 words 103 did not appear in the trainset. Some of these words were just slightly different from the words of the trainset. As example *averaged-priced* was in the devset and *average-priced* would have been in the trainset, the same with *family-friend* and *family-friendly*. Other words were different due to the preprocessing. As an example there is a restaurant called *"Browns Cambridge"*. The way the preprocessing was done is that there was an exact substitution of the name through the token "@name". But one of the sentences in the devset was "*Brown Cambridge* is not family friendly, but offers a unique menu of Chinese foods." Due to the missing "s" at the end of brown this was not replaced. Some words were also newly introduced words. Examples for this include "passable", "stand-out" and "cheerful". There are also examples of words which are wrongly spelled like "costumer-rated" and were not in the trainset because of this.

Even though there are 103 new words, every new word appears at most 2 times in the devset. There are 16 words appearing twice, the other 87 appear once. The word distribution can be seen in Figure 5.2. The graphic compares the amount of appearances of all words in the trainset to the amount of appearances in the devset. It visualizes that an overwhelming part of the words only appears rarely.

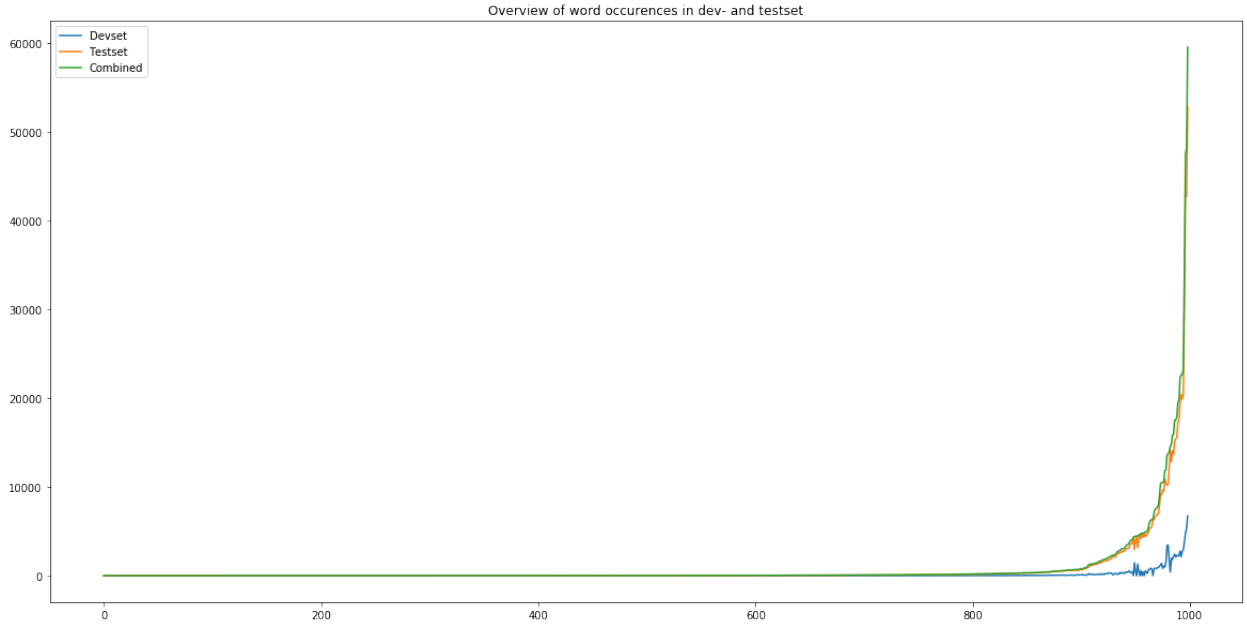

Figure 5.2.: Overview of word occurences in the dev and testset

The average number of words per sentence is *20.21* in the testset and *22.92* in the devset.

6. Architecture

In this chapter all neural networks used for the tasks at hand are listed. The chapter is split into two sections: The first section is related to the original task which was to generate reviews with the Notebookcheck dataset and the second section is related to solving the problems of sentence ordering and filling gaps in sentences with the restaurant review dataset. This switch of goals had to be done, because the results of all neural nets set up for the original task were poor.

6.1. Generation with CPU feature

As mentioned in the analysis chapter, the idea was to focus on generation text solely with the CPU feature first. If this test would perform well, the system could be extended to include all additional features seen in Section 5.7.

6.1.1. Model with feature vector

The idea of the first model was to simply have a feature vector of length 1 that decides whether the sentence includes a description about the CPU or not (0 = no CPU description, 1 = includes CPU description). The structure of the model can be seen in Figure 6.1.

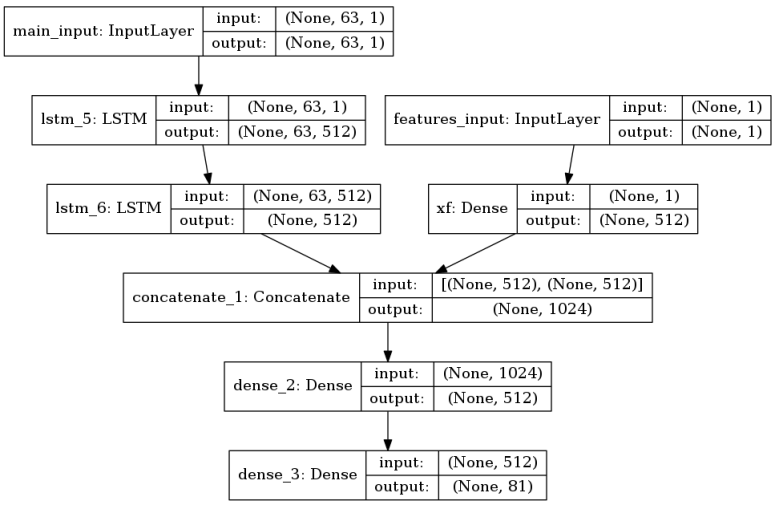

Figure 6.1.: First CPU generation model with a feature vector

6.1.2. Model for simple generation

The second model was structured even simpler and its goal was to verify if the neural network is even capable of generating meaningful text out of the given review data. To check this the feature vector was omitted completely, the full structure of the model can be seen in Figure 6.2.

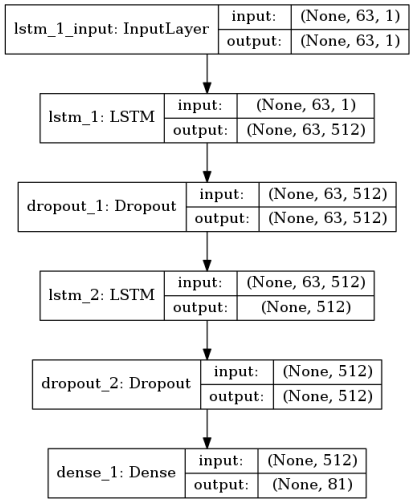

Figure 6.2.: Second CPU generation model without a feature vector

6.1.3. Model with multiple features

A third model was essentially the same as the first model. The difference was that it had more features and that the input characters were one-hot encoded. The structure of this model can be seen in Figure 6.3

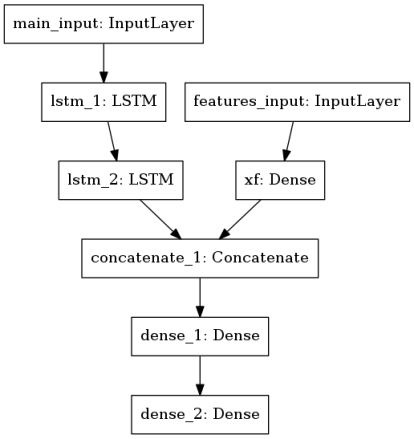

Figure 6.3.: Third CPU generation model with a feature vector and one-hot encoded input

6.2. Sentence Ordering

As seen in Section 7.1, the results of the text generation with the created dataset were not really promising, so the task had to be simplified somehow. One idea was that instead of using a feature vector containing specifications regarding the laptop, a set of words describing the laptop can be used to generate a descriptive text of it. (Example: overclocked, powerful, cpu → The laptop contains a powerful CPU that can be overclocked.) Before using this approach, two separate tests had to be done in order to check the feasibility of such a system. One test was to get an unordered sentence and bring it back into the right order, the other test was trying to fill pre-defined gaps in sentences. Different approaches were used to try and solve these tasks.

6.2.1. Sentence order with CNN

One of the models which were built for this task had two inputs. The first one got a one-hot encoded vector with words which were still available for generation. The second input had a sequence with the already predicted words. Both of those sequences were padded to the length of the longest sentence in the dataset. A mapping from the words in the dataset to an integer was taken as the input value for the vectors. These values were then translated with an embedding layer into dense vectors. It was decided to use this approach instead of one-hot encoding at the input layer, because otherwise there would have been huge sparse vectors. After that the output will then be fed to two convolutional layers with a maxpooling in between and then a global maxpooling layer. After the maxpooling layer there is a dense layer and then the two different branches will be merged. At the end there is another dense layers before the prediction is done. The output layer is then the same size as the vocabulary and uses one-hot encoding.

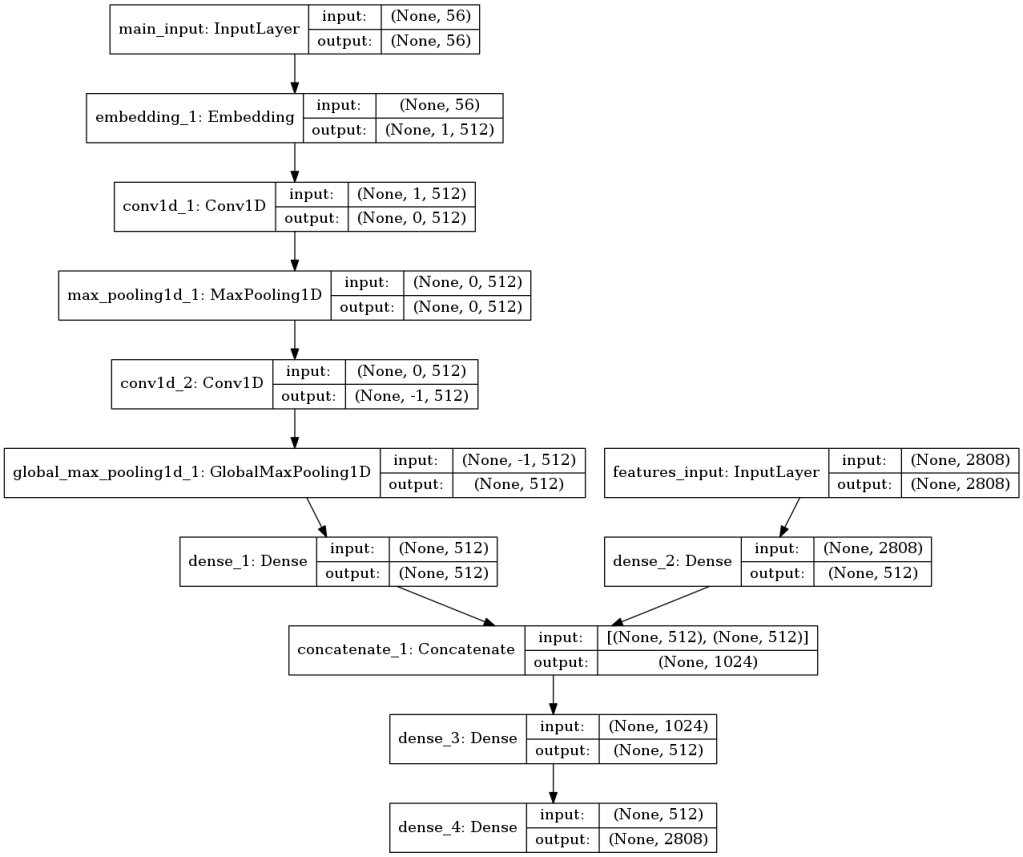

Figure 6.4.: CNN sentence order with multiple inputs

6.2.1.1. Filling the gaps with a CNN

The second approach which was initially used as a check if CNNs can master this task had only one input. It did not get the second input with the available words as the model above. That meant that it is also not able to bring a sentence back into the right order, it was just used to fill the gaps in a sentence. The input consisted of a sentence with placeholder tokens and a prediction token. The placeholders are words which are not yet known and the prediction token indicates the word that should be predicted in this run. For example the sentence ”The red car over there is very nice” could become ”The red @placeholder over @topredict @placeholder very nice”. All sentences were padded to the length of the longest sentence with empty tokens.

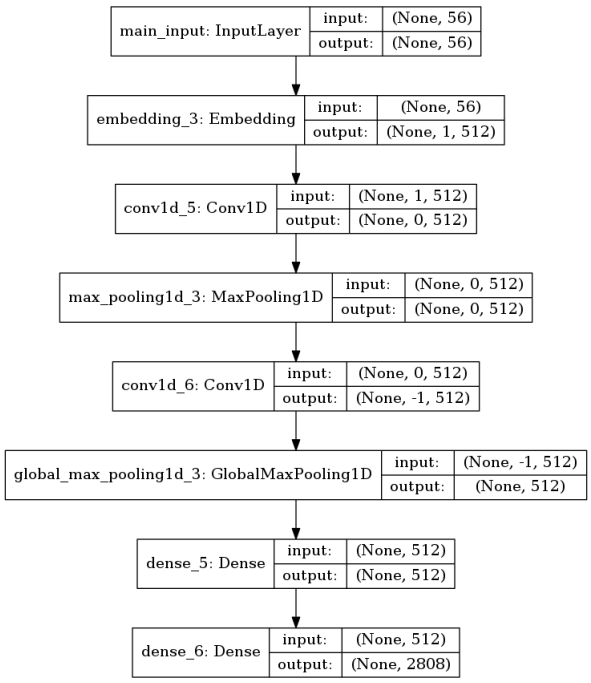

Figure 6.5.: CNN sentence order with single input

The performance of this model could be boosted by using a second input during sampling with the available words one-hot encoded. This second input is then multiplied with the results from the softmax layer above. This guarantees that all the words which are not available for the generation get a score of 0. This is especially helpful when the missing words have synonyms. For example if the network gives a high score to ”restaurant” and ”venue” the multiply layer could set ”restaurant” to 0, because it is not available for generation and thus ”venue” would be predicted.

6.2.2. Sentence order with Seq2Seq using word-to-int encoding

The template of the model was taken from a Keras blog [4] entry where they described how to build sequence-to-sequence models with Keras. The model worked on word-level and got the encoder integer sequences padded to the length of the longest sentence in the trainset as an input. For padding purposes a so called ”empty” token was defined. Both the encoder and decoder then had an embedding layer followed by an LSTM layer. While the encoder only consisted of these two layers, the decoder had an additional dense layer at the end with the softmax activation function.

Training

During training the input to the encoder was an unordered list of the integers which are padded to the maximum sentence length. The decoder got the encoder state as initial

state and then got a start token followed by a list of integers which represent the words in the sentence in order as input. If it is assumed that the max sentence length is 7 and the target sentence is *the car is green* the input sequence would be *[1,7,2,5,4,0,0]*. The used vocabulary for this example can be seen in the Table 6.1. The example mapping from every word to the integer is shown in Table 6.2. The target data is the one-hot encoded sentence.

Word	Integer
@empty	0
@start	1
car	2
fancy	3
green	4
is	5
nice	6
the	7
unique	8
@stop	9
@placeholder	10

Table 6.1.: Example vocabulary

@start	the	car	is	green	@empty	@empty
1	7	2	5	4	0	0

Table 6.2.: Example sequence

Sampling

During the sampling process the following steps are made: First the unordered sentence is fed to the encoder. The output of the encoder is discarded, but the internal states of the LSTM are kept. The internal state is then used as the initial state of the decoder LSTM. Now, if the first word should be predicted, the decoder gets the start token as input and predicts the first word. The input to the second round will then be the word that has just been generated and the internal state of the previous round. This process will continue until the decoder generates the stop token. If again the vocabulary from Table 6.1 is taken and the input sentence would be *"the car is green"* the input sentence to the decoder would therefore be *[2,4,5,7]*. The internal state of the encoder will be the initial state of the decoder and the input 1 (start token) would be given to the decoder. If the model predicts it correctly, the prediction would be 2 and the current decoder state would be used as initial state of the second round and 2 will be fed to the neural network. This process will continue until the sequence-to-sequence model predicts 9, which is the stop token.

6.2.2.1. Filling the gaps

The same sequence-to-sequence model that is used to predict the sentence order can be used to fill in gaps in a sentence. The only difference is in the training process: The

encoder input was adjusted and words were randomly substituted through a placeholder token. If the sentence *"the car is green"* is taken again to demonstrate this, the input would be *[7,10,5,4,0,0,0]* with the vocabulary from Table 6.1 and the assumption that the word "car" has been randomly replaced.

6.2.3. Sentence order with LSTM

6.2.3.1. Filling of gaps

Multiples Inputs One LSTM models to fill in the gaps in a given sentence worked as follows: It had two inputs, one input got a bag of words with the words which are available to fill in the blank spaces in the sentence, the other got a sentence. The sentence was padded to the length of the longest sentence with a "padding" token. Some of the words in the sentence were randomly replaced with a "placeholder" token. One of the words was the "predict" token, which told the neural network which word it should predict. Every word in the sentence was then mapped to an integer. There was a embedding layer connected to the input sentence, which was decided to be used instead of one-hot vectors, because the one-hot vectors would be high-dimensional vectors with only one entry (the current word) set to one and the rest would have been set to zero. After the embedding layer there were two bidirectional LSTM. Bidirectional was chosen to give the neural network some additional information, since the layers until here were supposed to learn the language model. After the LSTMs two dense layers followed. The output from the Dense layer was then concatenated with the input from the bag of words after a dense layer. After the concatenation two more dense layers followed, one had a softmax activation and was the output layer.

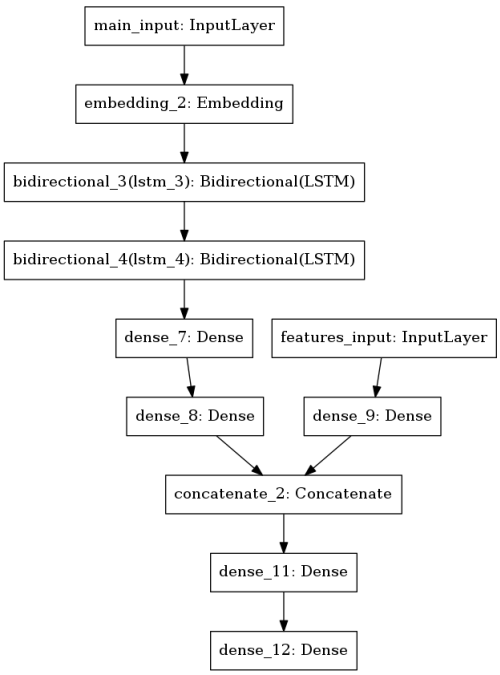

Figure 6.6.: LSTM sentence order with multiple inputs

During the sampling process a multiply layer could be added at the end. This layer takes the softmax output and the input from the bag of words and does an element by element multiplication. This was an additional safeguard to guarantee that only candidates available in the bag of words were predicted.

Single input The second LSTM was a simpler form of the model above. Instead of having two inputs, one for the available words and the other with the sequence, it just got the input with the sequence. This means that it just learns a language model in the end and it just generated suitable candidates. This could be used to generate new training samples.

6.2.4. Sentence order

For the task of bringing a sentence back into the right order, the same model as above was used.

7. Results

7.1. Generation with CPU feature

7.1.1. Model with feature vector

For this experiment the model described in Chapter 6.1.1 was trained. If the feature vector was set to 0 it should have generated a review containing a CPU. If it was set to 1, it should not talk about the CPU in the reviews. The output quality was rather poor, as can be seen in the results below. Very often the network was stuck at a local optimum, generating the same words over and over again. Also the same problem from the preceding project work that characters close to each other may get mixed up still existed. This can be seen in the first example below: The start pattern is 'o' and the network generated 'oumbers', most likely coming from 'numbers'.

Start pattern: o
Feature vector: [[1]]
umber of usb 3.0 ports: 1 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2
number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2 number of usb 3.0 ports: 2
number of usb 3.0 ports: 2</p

Start pattern: p
Feature vector: [[1]]
rocessor model: i7-7500u (dual core 2.7ghz 3th gen) intel core i7 (7th gen) 7200u) 1.50 ghz with turbo boost technology up to 3.5ghz) with intel hd graphics 620 ddr3 sdram - bluetooth - english keyboard - front camera/webcam - ieee 802.11ac - ethernet - network (rj-45) - hdmi - 3 x usb 3.0 ports - 1 x usb 3.0 ports - 1 x usb 3.0 ports - 4-cell lithium ion (li-ion) - 12 hours battery life (depending on configuration &amp&#59

usage) up to 12 hours bnd 40 minutes (mixed usage), up to 5 hours and 40 minutes (mixed usage), up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 mi

Start pattern: h
Feature vector: [[0]]
ard drive: 500gb sata hard drive + 1tb hard drive - 128gb ssd - 16gb ddr4 ram, 2133 mhz - 4g lte-a sierra wireless em7455 wwan card (amp&#59 olufsen sound technology. 59 intel core i7-4500u (3mb intel smart cache, up to 3.50 ghz) with intel vpro technology for easy access to your network or hotspots on your screen. *15.6 " fhd (1920 x 1080) display, and a new surface pen (platinum), 128gb micro sd card, mini displayport to hdmi/vga/dva cable adapter
wsb ports:usb 3.0 - 2, usb 3.0 - 2
built-in webcam:yes
backligh memory«/b>8192mb
processors: intel core i7-7500u dual-core 2.5ghz 4mb cpu
memory installed:
4096mb ddr3
hard drive:500gb 7.2k 3gb/s sata 2.5"" hdd
power supply:hp charger</l

Start pattern: b
Feature vector: [[1]]
attery life: up to 10 hours and 40 minutes (mixed usage), up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery

Start pattern: s
Feature vector: [[0]]
he system is powered by a 2.8 ghz intel core i7-8550u qrocessor and 16gb of memory, a 2tb hard drive, a multi-format sd digital media reader, 10/100/1000mbps gigabit ethernet, 802.11 wi-fi, an integrated webcam and microphone. there are also built-in speakers utilizing bang &amp&#59 olufsen sound technology. pptioal memory card formats. *next-generation wireless-ac connectivity for yireless nan built into the screen size and a thin dell inspiron 15 5000 series notebook from dell is a durable system that can withstand life nore than the thinkpad p50 features a blazing fast and enjoy the security features that allows you to responsive display and stay connected and entertainment features and enhanced security features to have the macbook pro model a computer upgrade king of the latest high-

Start pattern: b

Feature vector: [[1]]
attery life: up to 10 hours and 40 minutes (mixed usage), up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery life - up to 12 hours of battery life - up to 12 hours bnd 40 minutes (mixed usage) up to 10 hours bnd 30 minutes of battery

Start pattern: w
Feature vector: [[1]]
ireless connectivity: 802.11ac wi-fi &&#59 bluetooth 4.0interface: 2x usb 3.0 ports, 1x usb 3.0 (always on), headphone/mic combo, microsd, omnm, and an integrated dard tlot, and a new gigabit ethernet connection offers faster than wireless-n. the fast ethernet lan port also plugs into wired networks. *built-in hd webcam with dual digital microphone array microphones&#59 bluetooth 4.0interface: 1x usb 3.0 ports, 1x usb 3.0 port, 1x hdmi port, 1x mini displayport, 1x rj45 lan jack for lan (10/100), memory card reader and 10/100 ethernet port with dolby home theater and usb 3.0 ports, 1 usb 3.0 ports, 1 usb 2.0 ports, 1 usb 2.0, 1 headphone/microphone jack, 1 x usb 3.0, 1 headphone/microphone jack, 1 x usb 3.0, 1 headphone/microphone jack, 1 x usb 3.0, 1 headphone/microphone jack, 1 x usb

Start pattern: the hp
Feature vector: [[0]]
elitebook 850 g3 notebook from dell is a durable system that can withstand life nore than the thinkpad p50 features a blazing fast and enjoy the security features that allows you to responsive display and stay connected and entertainment features and enhanced security features to have the macbook pro model a computer upgrade king of the latest high-speed devices for security and music videos in corporate-clean qicture. *1– usb 3.0 ports, a firewire ””800”” port, two usb 3.0 ports, a ””thunderbolt”” port, an audio in/out, and an sdxc card slot, and an integrated digital microphone for extability and music videos in compatible with fast security and multimedia capabilities. the latest intel core i7-6500u processor and 16gb of memory, a 2tb hard drive, a multi-format sd digital media reader

7.1.2. Model for simple generation

Below are the results for the model, whose goal is simply to generate text without any feature vector. Even here the output quality was poor, probably due to the large vocabulary size of the training set.

Start pattern: l

ntel core i5-7200u processor model: i5 8200u 1.5ghz (tupport tize 1.1 gb (1360 x 108) w 1.0.. x 1.. im (

Start pattern: o
rocessor model: i5-6200u 2.5ghz - 8 gb ram - 220 gb ssd - windows 10 pro 64,bit - 8 gb ram - 156 gb ssd - 156 mb ssd hos wideo camera with the latest amd quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance

Start pattern: e
esigned fesigned with a 1tb had boost sp headphone ouititask with a 1.0 gb soleds for a hard deii processors and backlit keyboard with windows 10 pro operating system and aacess tested and leno you can with the latest amd quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance f

Start pattern: u
he macbook pro ”core i5 3320m 2.5ghz (tumxag baptery (ladbook pro ”core 2 duo” 2.2 13

Start pattern: r
he mew x1 carbon features a windows 10 pro opbrating system: windows 10 pro operating system and aacess tested and leno you can with the latest amd quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a seroection of the latest and quad-core processing performance for a s

7.1.3. Model with multiple features

This model seemed to have slightly better results than the other two. Especially for the generation of the CPU and SSD features. However there is still a lot of nonsense that is generated by the model. Individual words may make sense and some sentences are also more or less correct, but it is not coherent in general. It also gave an indication on what the neural network could learn about certain features.

The way that sampling worked for this model was, that different features could be chosen for generation. The model then generated either until the features were generated or until 800 characters were predicted. To check if a feature was generated, an external script was used where the features were hard coded. In the 4th example it can be seen that the model generated both features (GPU and CPU) and still continued to predict. This happened because the GPU value should have been "@gpu" instead of a GPU name. But in some cases the dataset had a disparity between the GPU model in the text and in the structured data. This led to the effect that some of the GPUs were not substituted during training and were learnt by the neural network.

Features to generate: CPU, SSD **Start pattern:** d
espite the fast processor a @cpu and an @gpu with 6 gb of video ram at its disposal. for a further 100 euros you can get a model with windows 10 operating system while 150 euros more will get you a 256 gb ssd as well as windows 10.

Features to generate: CPU, SSD **Start pattern:** h
igh performance with the optional @cpu 8 gb of working memory and a 128 gb ssd. the 8 gb of ram and the 256-gb ssd and the 1 tb hdd from seagate.

Features to generate: CPU, SSD **Start pattern:** a
model with @cpu processor 8 gb of ram 256 gb ssd storage and either 32 gb or 64 gb of storage which can be expanded by up to 200 gb.

Features to generate: CPU, SSD **Start pattern:** u
sers can place on the @nbname and its height feet somewhat portable programs and also achieved performance potential at least for a price of about 1000 euros but that is the campus program and cpu performance is at aboot 300 and up to 3.0ghz when encryption achieved a maximum enveloped @cpu (surrounding up to 4 gb) about the same contrast level as the ram can be expanded up to 32 gb. the @cpu is a dual core processor and an integrated and a dedicated radeon r5 m430 gpu. the cheapest configuration is an amd quad-core processor that clocks at a base speed of @ghz. the processor clocks at a base speed of @ghz. the processor clocks at a base speed of @ghz. the processor clocks at a base speed of @ghz. the processor clocks at a base speed of @ghz. the processor clocks at a base speed of @ghz.

Features to generate: CPU, SSD **Start pattern:** l
imited models can be configured with an @cpu 8 gb of ram and a 128 gb ssd.

Features to generate: CPU, RAM **Start pattern:** l
imited models can be configured with an @cpu 8 gb of ram and a 128 gb ssd efore the processor that games at least to a higher clock maintenance flap as to a fast ssd and a free m.2 solid-state drive. the @cpu is a mid-range phone with a high-end gpu and ssd solution is supported by a 4-cell display and 3 gb of ram.

Features to generate: CPU, SSD **Start pattern:** the dell
with a weither integrated graphics processor and an @cpu 16 gb of ram and qhd+ 1800p glossy touchscreen for $1750 usd. though slightly disappointing this which would put the xps 13 more in line with the tuxedo infinitybook pro sporting the same cpu. the three devices with the better cpu cannot really break away from the rest of the field. despite having only two cores the huawei is able to keep up well against the eurocom with excessively paid for the private scale. the @nbname is exactly 1354 mb/s while writing. the specifications fujitsu has a fingerprint-sensor for gaming. the @nbname is equipped with a 1 tb hdd and 256 gb ssd storage with 256 gb being standard.

Features to generate: CPU, Intel **Start pattern:** s
ince the @nbname behaves very differently. bass is a foreign word for the @nbname because the @gpu can run most games smoothly at low resolutions and low quality settings. the gpu's weaknesses are seen in the fast-paced racing game "asphalt 8 " without any problems. unfortunately the ssd does not support hyper-threading and the more powerful cpu has to run demanding tasks for a regular radeon r5 m430 gpu. the built-in nvme ssd ensures a fast-running system. the ssd's transfer rates are good. intel's hd graphics 520 gpu is responsible for video output.

Features to generate: CPU, Gpu **Start pattern:** m
ost people serfequently the less expensive @cpu offers a moderate performance increase. the cpu recurrently windows 10 pro includes the combination of a 32 gb ssd cache and separate (!) hdd is installed in dell's xps 13. due to the large range in the clock rate sitjer the 14-inch notebook with an @gpu with 6 gb of vram is plenty sturdy with a more powerful gpu.

7.2. Filling gaps in a sentence

Because the results for the language generation were not promising it was tried to assess if a neural network can fill gaps in a sentence. This was done on the restaurants dataset.

7.2.1. Softmax output

To check if the built neural networks learn the language model and are able to generate good candidate words for a placeholder in a sentence, the softmax output was analyzed. The multiply layer was omitted for this experiment so that it would not influence the output. The input was a sentence with one placeholder.

LSTM

Single Input LSTM In this section there are some examples for the output of the softmax layer from the model with a single input described in Chapter 6.2.3.1. On top of every example sentence the top 5 results according to the softmax are listed in order. This means that the candidate with the highest score is on the far left side on position one, the candidate from the top 5 with the lowest score is on the far right side at position 5.

Top 5: city, town, historical, mid-range, pub
Input: in the @tofind centre there is a venue name @name this is not a family-friendly venue
Expected: city

Top 5: restaurant, place, pub, venue, coffee-shop
Input: in the city centre there is a @tofind name @name this is not a family-friendly venue
Expected: venue

Top 5: riverside, river, @near, city, northern
Input: @name is a coffee shop in the @tofind area
Expected: riverside

Top 5: serves, offers, features, provides, sells
Input: it @tofind chinese food and has a customer rating of 1 out of 5
Expected: sells

Top 5: japanese, french, italian, fast, indian
Input: it sells @tofind food and has a customer rating of 1 out of 5
Expected: chinese

These examples show that the neural network generally did a good job at predicting a suitable candidate for the placeholder. There were cases like the last one, where the expected word for the placeholder was not in the top 5. But even in cases like this, the top 5 candidates consisted mainly out of words that would also be grammatically correct in the given context.

Another example shows how difficult it was to rank the proper word in the top 20 without

knowledge of eligible candidates:

> **Top 20:** popular, kids-friendly, fast-food, non-family-friendly, noodle, five-star, english, family, the, children-friendly, high-priced, expensive, italian, child-friendly, riverside, kid-friendly, french, cheap, family-friendly, japanese
> **Input:** near @near in city centre is the @tofind establishment @name
> **Expected:** adult

The only candidate that does not make any sense at all given the context is the word "the". While "Noodle" may sound strange in this sentence, when taken into account the neural network probably learns that restaurant and establishment can be used as synonyms, it starts to make more sense. The sentence "... is the noodle restaurant @name" for example would be a correct prediction.

Results The softmax results from all the examples in the devset from the NLG Challenge were analyzed. Table 7.1 shows how many times the expected word was not ranked under the top x words. There were a total of 93693 samples. That means that 76980 examples were correctly detected if only one placeholder is used in the sentence.

Not in top 1	Not in top 2	Not in top 5	Not in top 10	Not in top 20
16713	10447	5360	2425	1552

Table 7.1.: Number of times an expected word was not in the top x

Multi-input LSTM The multi-input LSTM that is described in Chapter 6.2.3.1 was analyzed in the same way as the LSTM model from above. Theoretically this neural network should have provided better results then the previous one, because it has the possible words to fill the gap as additional information.
In this experiment the available words were the word that should be predicted and an additional 50 random words. If the random words were not added, the task would be too trivial, because the model would have simply had to predict the only word that it gets in the available words. Below are some of the results of the LSTM.

> **Top 5:** is, has, moderate, serves, @name
> **Input:** there @tofind a place in the city centre @name that is not family-friendly
> **Expected:** is

> **Top 5:** riverside, town, in, @name, an
> **Input:** @name is an english coffee shop in @tofind
> **Expected:** riverside

> **Top 5:** @name, there, stars, the, that
> **Input:** @tofind is a place in the city centre @name that is not family-friendly
> **Expected:** there

Top 5: indian, italian, located, @name, @near
Input: this @tofind restaurant called @name is located in riverside near @near
Expected: kids-friendly

There was the problem that the neural network could not correctly detect phrases which were not present in the training set. For example the word "looks" only appeared once in the training set in the sentence "...it also looks good... ". In the devset there was also a sentence with the word "looks" in it, but the context was completely different:

Top 5: restaurant, also, families, they, offers
Input: @name @tofind like a location where people or families arent allowed
Expected: looks

It was not surprising that the network did not learn to predict words and phrases, which are rarely seen during training. Nevertheless this architecture performed well and produced useful results. In Table 7.2 the results over the whole devset are listed:

Not in top 1	Not in top 2	Not in top 5	Not in top 10	Not in top 20
10829	5935	3393	2292	1606

Table 7.2.: Number of times an expected word was not in the top x

CNN

Single Input CNN The same experiment was done for the CNN model that is described in Chapter 6.2.1.1. The two main differences to the model above are: First, the CNN cells instead of LSTM cells and second, that the only input is the sentence with the placeholder. The available words to fill the gap are not given to the neural network.

Number of samples: 93693

Top 5: there, it, @name, @near, this
Input: @tofind is a place in the city centre @name that is not family-friendly
Expected: there

Top 5: cheap, family-friendly, kid-friendly, child-friendly, children-friendly
Input: @name is a @tofind establishment near @near at the riverside
Expected: non-family-friendly

Top 5: near, the, beside, neat, nearby
Input: @tofind @near in the city centre @name is family-friendly
Expected: located

Top 5: without, scale, ranting, okay, establishment

Input: @name looks like a @tofind where people or families arent allowed
Expected: location

Top 5: rate, allows, popular, rated, enjoy
Input: @name looks like a location where @tofind or families arent allowed
Expected: people

Not in top 1	Not in top 2	Not in top 5	Not in top 10	Not in top 20
19605	12112	6269	3046	1853

Table 7.3.: Number of times an expected word was not in the top x

In Chapter 6.2.1.1 it was also mentioned that additionally a multiply layer could be added at the end to improve the results. In Table 7.4 the results of this model are listed.

Not in top 1	Not in top 2	Not in top 5	Not in top 10	Not in top 20
4763	2117	934	463	101

Table 7.4.: Number of times an expected word was not in the top x

As can be seen, it still does not predict every word correctly. Below are some of the examples where it failed:

Top 5: as, very, is, having, customer
Input: the customer rating is 1 out of 5 as well as @tofind a very high price range
Expected: having

Top 5: a, not, is, friendly, @name
Input: @name is @tofind a children friendly english coffee shop
Expected: not

Top 5: and, but, it, called, is
Input: it is called @name and it is family friendly @tofind it does have a 1 out of 5 rating
Expected: but

Top 5: family, a, style, friendly, is
Input: @name is a family friendly @tofind coffee shop
Expected: style

A reason that some of the examples (e.g the last one and the first one) are still wrong, is that some of the language constructs in the devset are very rare or non-existent in the trainset. The phrase ”as well as having” appears only once in the trainset and it is followed by ”a coffee shop”, unlike in the first example where it is followed by the high price range. Conclusively, it makes sense that the neural network was unable learn

phrases, which appeared that rarely. The same problem was already observed above with the LSTMs.

Multi Input CNN The same experiment was also done with the second CNN model, which was described in Chapter 6.2.1. This neural network got the input sentence with a placeholder and also the available words to fill the gap. For this experiment all the words which appear in the sentence, including the word that should be predicted and 50 random words were marked as available words in order to make it more difficult. There were a total of 93693 examples. In Table 7.5 the results can be seen.

Not in top 1	Not in top 2	Not in top 5	Not in top 10	Not in top 20
12274	6352	3225	2020	1291

Table 7.5.: Number of times an expected word was not in the top x

7.2.2. Sequence-to-Sequence

The seq2seq model cannot be compared directly to the models above, because the task for this model was more difficult. The model got an input sequence, which was unordered and additionally one word replaced with a placeholder. The goal was then to bring it back into the proper order and find the right word for the placeholder. The other models from above always got the sentence in right order and simply had to predict the missing word. Below are some examples:

Missing word: near
Input sequence: @near @name is @placeholder riverside located in
Target sentence: @name is located near @near in riverside
Decoded sentence: @name is located near @near in riverside

Missing word: in
Input sequence: a family area friendly @name place @placeholder the is riverside
Target sentence: @name is a family friendly place in the riverside area
Decoded sentence: @name is a family friendly place in the riverside area

Missing word: is
Input sequence: @placeholder @name venue a area the in family-friendly riverside there called
Target sentence: there is a family-friendly venue in the riverside area called @name
Decoded sentence: there is a cheap family-friendly venue in the riverside area called @name

Missing word: serves
Input sequence: chinese range @name shop high food that the price a @placeholder is coffee in
Target sentence: @name is a coffee shop that serves chinese food in the high price range
Decoded sentence: @name is a coffee shop that provides indian food in the high price range

Recall was chosen as a measurement to detect how many placeholders were replaced through the expected word. It ignores the word order and checks if all the expected words are included in the generated sentence. The recall score was 0.7864.

7.2.3. Conclusion

If the models are compared, it does not come to a surprise to see, that the models with a multiply layer do exceptionally well on this task. When the CNN, which just got the sentence with the placeholder and the one that also got the available words are compared, it can be seen that the one with the additional information outperforms the other. The same is true for the LSTM models. More interesting to see is that CNNs performed nearly as well as the LSTM models. It would have been interesting to see, if CNNs could outperform the LSTM model given properly tuned hyper-parameters. If this was the case, the advantage would be that training and prediction time for the CNN model would be significantly faster than for the LSTM model. Unfortunately there was not enough time to analyze said question.

Model	Not in top 1	Not in top 2	Not in top 5	Not in top 10	Not in top 20
CNN	19605	12112	6269	3046	1853
CNN with available words	12274	6352	3225	2020	1291
CNN with multiply layer	4763	2117	934	463	101
LSTM	16713	10447	5360	2425	1552
LSTM with available words	10829	5935	3393	2292	1606

Table 7.6.: Comparison of results from the different models

The sequence-to-sequence model can not be directly compared to these other models for this task. However it got clear during a evaluation by hand and with the use of the recall score, that if the goal is simply to fill one gap, a seq2seq model does a worse job than the other models.

7.2.4. Filling multiple gaps

7.2.4.1. CNN with multiple inputs

In the previous section it was shown, that the CNN described in Chapter 6.2.1 is able to fill a single gap in a sentence. It was evaluated, if the model could also fill multiple gaps in a sentence. For this multiple placeholder token were set in the sentence. To sample one of the placeholders, it was replaced through a prediction token. This was done until all placeholders were predicted. Below are some of the results:

Input: @placeholder is a @placeholder in @placeholder @placeholder @placeholder @name @placeholder is not @placeholder
Expected: there is a place in the city centre @name that is not family-friendly
Predicted: there is a place in the city centre @name that is not family-friendly

Input: @placeholder @placeholder @placeholder @near its customer @placeholder is @placeholder out @placeholder 5
Expected: it is near @near its customer rating is 3 out of 5
Predicted: it is near @near its customer rating is 3 out of 5

Input: @placeholder is @placeholder @placeholder @placeholder @placeholder @placeholder centre @name that @placeholder @placeholder family-friendly
Expected: there is a place in the city centre @name that is not family-friendly
Predicted: there is not place in the city centre @name that is a family-friendly

Input: in the city centre @placeholder @placeholder @placeholder @placeholder name @name @placeholder @placeholder @placeholder a @placeholder venue
Expected: in the city centre there is a venue name @name this is not a family-friendly venue
Predicted: in the city centre and there a venue name @name this is not a family-friendly venue

Input: in the @placeholder centre @placeholder @placeholder @placeholder venue name @tofind this is @placeholder @placeholder @placeholder @placeholder
Expected: in the city centre there is a venue name @name this is not a family-friendly venue
Predicted: in the city centre there a a venue name @name this is not is family-friendly venue

Input: @name coffee @placeholder is @placeholder friendly with @placeholder english food @tofind a @placeholder @placeholder rating
Expected: @name coffee shop is family friendly with cheap english food with a low customer rating

Predicted: @name coffee shop is family friendly with cheap english food with a low customer rating

Input: @name @tofind coffee @placeholder @placeholder @placeholder @placeholder price with @placeholder @placeholder @placeholder @placeholder @placeholder out of 5; @placeholder in the @placeholder area close @placeholder @near
Expected: @name chinese coffee shop is moderate in price with a customer rating of 3 out of 5; located in the riverside area close to @near
Predicted: @name chinese coffee shop in is moderate price with a customer rating of 3 out of 5; located in the riverside area close to @near

Input: @placeholder @placeholder @placeholder @placeholder @placeholder @placeholder @tofind @placeholder @placeholder @placeholder coffee @placeholder @placeholder @placeholder @placeholder located by @placeholder riverside customers have @placeholder @placeholder 1 out @placeholder 5
Expected: if youre looking for a kid free moderately priced english coffee shop check out @name located by the riverside customers have rated it 1 out of 5
Predicted: youre a moderately priced out check it if looking english coffee shop free free @name located by the riverside customers have rated for 1 out of 5

The examples above show that this neural network is able to fill in multiple gaps. For sentences were a pattern is visible (e.g "placeholder placeholder friendly") and available words would fit that pattern (e.g. "not family friendly"), it performs well and finds the right places for the available words. If however there are a lot of placeholders in succession like in the last example and not a lot of additional information is provided, the quality quickly drops. A possible approach to increase the performance for such examples could be to implement a beam search and for example follow the top 5 solutions, instead of using the best one from the beginning. To assess the output quality different measurements were calculated. The whole devset was taken and in every sentence at least 2 words were randomly replaced by placeholders and at most as much as that one word would still be in the sentence. E.g for the sentence *"it was a very nice day*" it means that the sentence would at least be something like *"it placeholder a placeholder nice day."* and at most *"placeholder placeholder a placeholder placeholder placeholder"*. Then the recall score, the precision and the BLEU score were calculated. The results are shown in Table 7.7

Recall	Precision	Bleu
0.9896886201791828	0.996647277624764	0.9741221735891562

Table 7.7.: Recall, Precision and Bleu score for CNN

7.2.4.2. LSTM with multiple gaps

The previous findings illustrated, that the LSTM model with multiple inputs from Chapter 6.2.3.1 performed best in the task of filling one gap. So it was also evaluated how well it performs in filling multiple gaps. Here are some examples:

Input: in the @placeholder centre @placeholder @near there is a @placeholder @placeholder called @placeholder its not family @placeholder serves @placeholder @placeholder has an average customer rating @placeholder @placeholder high price range
Expected: in the city centre near @near there is a coffee shop called @name its not family friendly serves chinese food has an average customer rating and a high price range
Predicted: in the city centre near @near there is a coffee shop called @name its not family friendly serves chinese food has an average customer rating and a high price range

Input: @name is a @placeholder shop that @placeholder @placeholder food @placeholder @placeholder @placeholder @placeholder
Expected: @name is a coffee shop that serves english food in the city centre
Predicted: @name is a coffee shop that serves english food in the city centre

Input: @placeholder @placeholder @placeholder coffee @placeholder @placeholder @placeholder @placeholder @placeholder @placeholder @placeholder low @placeholder range with @placeholder @placeholder customer rating
Expected: @name is a coffee shop that serves english food in the low price range with a low customer rating
Predicted: @name is a coffee shop that serves english food in the low price range with a low customer rating

Input: @name is a coffee @placeholder that serves @placeholder @placeholder in the low @placeholder @placeholder @placeholder @placeholder @placeholder @placeholder @tofind
Expected: @name is a coffee shop that serves english food in the low price range with a low customer rating
Predicted: @name is a coffee shop that serves english food in the low price range a customer rating with low

Input: in the city centre there @placeholder @placeholder @placeholder name @name this @placeholder not a @placeholder venue
Expected: in the city centre there is a venue name @name this is not a family-friendly venue
Predicted: in the city centre there is a family-friendly name @name this venue not a family-friendly venue

Input: for @placeholder adults-only @placeholder @placeholder 5 @placeholder @placeholder 5 by its @placeholder head to @name near @near @placeholder riverside **Expected:** for an adults-only venue rated 5 out of 5 by its patrons head to @name near @near in riverside **Predicted:** for an adults-only venue rated 5 out of 5 by its patrons head to @name near @near in riverside

Input: @placeholder @placeholder @placeholder @placeholder rated 5 out of 5 by @placeholder @placeholder @placeholder to @name near @near @placeholder riverside **Expected:** for an adults-only venue rated 5 out of 5 by its patrons head to @name near @near in riverside **Predicted:** adults-only venue for its rated 5 out of 5 by pounds pounds patrons to @name near @near in riverside

Input: for @placeholder @placeholder @placeholder @placeholder @placeholder out of @placeholder @placeholder @placeholder patrons head to @placeholder near @near in @placeholder **Expected:** for an adults-only venue rated 5 out of 5 by its patrons head to @name near @near in riverside **Predicted:** for adults-only venue rated an 5 out of 5 by its patrons head to @name near @near in riverside

As can be seen in the examples, simpler sentences can be rebuilt well, even if there are a lot of unknown words in the sentence. The last three examples show how the same sentence with different placeholders can result in better or worse outputs. Especially interesting in the last two examples is, that less gaps in a sentence do not necessarily lead to better results. The last example had 10 placeholder whereas the second last had 8 placeholders. The results of the example with more placeholders were better in this specific constellation. This lead to the same conclusion as for the CNN, that the output quality depends on the substituted word and if there are obvious expressions in the sentence.

Recall	Precision	Bleu
0.9936536957087293	0.9962966484055036	0.9868172936633186

Table 7.8.: Recall, Precision and Bleu score for multi input LSTM

7.2.4.3. Seq2Seq with multiple gaps

The sequence-to-sequence model that was described in Chapter 6.2.2 can also be used to fill multiple gaps. The part that makes this task hard for the model is again that it has to bring the sentence into the right order and at the same time predict the right words for the placeholders. Here are some examles:

Missing words: riverside, is
Input sequence: in @near @placeholder near @name located @placeholder
Target sentence: @name is located near @near in riverside
Decoded sentence: @name is located near @near in riverside
Bleu score: 1.0
Recall score: 1.0

Missing words: a, in, near
Input sequence: @placeholder family @placeholder there located friendly @placeholder is @name riverside @near place
Target sentence: there is a family friendly place @name located near @near in riverside
Decoded sentence: there is a family friendly place @name located near @near in riverside
Bleu score: 1.0
Recall score: 1.0

Missing words: is
Input sequence: @name riverside near the in @placeholder @near area family-friendly a location @placeholder
Target sentence: @name is a family-friendly location in the riverside area near the @near
Decoded sentence: the family-friendly location near @near in the riverside area is called @name
Bleu score: 0.8290090653071776
Recall score: 0.9090909090909091

Missing words: child, which, is, friendly, there
Input sequence: @placeholder a @placeholder @placeholder riverside called @name restaurant @placeholder @placeholder @placeholder
Target sentence: there is a riverside restaurant called @name which is child friendly
Decoded sentence: there is a fast food restaurant called @name by the riverside
Bleu score: 0.6603237968547442
Recall score: 0.7

Missing words: you, child, the, their, try, @name, meal, for, a, provide, they, riverside, to
Input sequence: @placeholder service take @placeholder @placeholder @placeholder @placeholder if friendly @placeholder in @placeholder @placeholder want @placeholder children setting @placeholder then @placeholder @placeholder @placeholder @placeholder
Target sentence: if you want to take the children for a meal then try @name they provide a child friendly service in their riverside setting
Decoded sentence: if you want a more than £30 price and children-friendly low customer service rating fast food restaurant @name would be in riverside near @near
Bleu score: 0.4884131772209672
Recall score: 0.36363636363636365

Missing words: with, its, @name, by, low
Input sequence: @placeholder friendly @placeholder is @placeholder ratings @placeholder customers @placeholder child
Target sentence: @name is child friendly with low ratings by its customers
Decoded sentence: it is child friendly and receives a low customer rating
Bleu score: 0.6445873357693387

Recall score: 0.6445873357693387

The examples show that the neural network is able to rebuild sentences in the correct order with the right words, if the sentences are simple and only a few words are missing. For longer sentences it is not possible to recover the original words, which makes sense, because the words are not provided to the neural network. To get a measurement of the output quality precision, recall and the BLEU score were calculated over all the samples:
Bleu score: 0.7417775417424143
Precision: 0.7757993592075457
Recall: 0.7871104677100901

Recall	Precision	Bleu
0.7871104677100901	0.7757993592075457	0.7417775417424143

Table 7.9.: Recall, Precision and Bleu score for multi input LSTM

However, the scores were not that meaningful for this task and the sequence-to-sequence model. It was more important to check, if the decoded sentences made sense. This had to be done by hand, because there did not seem to be any suitable tools available to solve this task. They either marked too many things as mistakes such as e.g country names written in lowercase, British English words given there were American English equivalents or the tools simply checked the words but not the grammar. During this evaluation it was seen that very long sentences were often grammatically wrong and incoherent. Below are some examples that show said problem:

> if you are looking for a cheap coffee shop with a 5 out of 5 customer rating you should try @name located near @near and has prices then visit @name

> if you want to get english food with a moderately priced coffee shop then go to @name it has still with customer rating for it 5 out of 5 and offer food near @near

> if youre looking for a high priced coffee shop that is children friendly but not so children is you should try @name near the @near the customer rating is average

> if you are looking for chinese food place @name near the @near in the city centre with a moderate price range and a customer rating of 1 out of 5 but not kid friendly

> in the city centre is a 5 out of 5 rated fast food coffee shop called @name they are cheap and inexpensive prices and terrible reviews

Shorter sentences in contrast usually were of high quality (grammatically correct) and had a similar style to the sentences in the training set.

7.2.5. Conclusion

Three different models were shown to be able to fill in gaps in a sentence. Depending on the task, the best model changes. If a sentence has to be rebuilt and the words are known, the LSTM with multiple inputs had the best performance. If the task is to build a sentence with only some given words, the sequence-to-sequence model showed promising results.

Model	Recall	Precision	Bleu
Seq2Seq	0.7871104677100901	0.7757993592075457	0.7417775417424143
CNN multi input	0.9896886201791828	0.996647277624764	0.9741221735891562
LSTM multi input	0.9936536957087293	0.9962966484055036	0.9868172936633186

Table 7.10.: Recall, Precision and Bleu score for multi input LSTM

The CNN model was slightly beaten by the LSTM in terms of recall and BLEU score. In precision however it was vaguely better.

7.3. Sentence ordering

7.3.1. Sequence-to-Sequence

7.3.1.1. Examples

In this section there are some examples of the seq2seq model. At first there is the input sequence, which consists of the words that were randomly shuffled. Next there is the target sentence, which is the sentence that the neural network should generate. The decoded sentence is the sentence which was actually generated by the neural network.

Input sequence: city a place in is the family-friendly that is there centre @name not
Target sentence: there is a place in the city centre @name that is not family-friendly
Decoded sentence: there is a place @name in the city centre that is not family-friendly
Bleu score: 0.9884373631740123

Input sequence: venue this @name in the family-friendly venue is a centre a is name city not there
Target sentence: in the city centre there is a venue name @name this is not a family-friendly venue
Decoded sentence: there is a venue not family-friendly name @name is this venue

in the city centre
Bleu score: 0.896366497899096

Input sequence: is place a city not @name family-friendly located in centre
Target sentence: @name is not a family-friendly place located in city centre
Decoded sentence: @name is not a family-friendly place located in city centre
Bleu score: 1.0

Input sequence: in place centre not the family-friendly is @name city a
Target sentence: @name is not a family-friendly place in the city centre
Decoded sentence: @name is a not family-friendly place in the city centre
Bleu score: 0.9557892810345545

Input sequence: is isnt riverside @name family-friendly it in but
Target sentence: @name isnt family-friendly but it is in riverside
Decoded sentence: @name is in riverside but isnt family-friendly
Bleu score: 0.9043437155197077

The examples above show that the model works well. It is able to bring simple sentences back into the right order. Sometimes it outputs a different order that would still be grammatically correct. This cannot be avoided if no extra information is given to the neural network and is in fact no problem for this task.

Input sequence: establishment near the @name in centre adult city is @near
Target sentence: near @near in city centre is the adult establishment @name
Decoded sentence: @name is adult establishment in the city centre near @near
Bleu score: 0.8960776018554231

Input sequence: rating 3 5 of @name budget a with for is a a great out on is those moderate which child-friendly restaurant of
Target sentence: @name is a child-friendly moderate restaurant with a rating of 3 out of 5 which is great for those on a budget
Decoded sentence: @name is a great restaurant for families which is in a mid range menu with a rating of 3 out of 5
Bleu score: 0.6512064825119718

Input sequence: near has adults happy @near children rating customer of is 5 located place 3 and to @name for very
Target sentence: @name is located near @near very happy place for children and adults has 3 to 5 of customer rating
Decoded sentence: @name coffee shop near @near located near to @near and has very high prices in the riverside atmosphere
Bleu score: 0.46600282322240805

Input sequence: a a offering medium-priced coffee meals is shop star @name rating

called three with
Target sentence: a coffee shop called @name is medium-priced offering meals with a three star rating
Decoded sentence: a coffee shop called @name is a three star with a pub that features
Bleu score: 0.5927274901448417

Input sequence: average heard @name @near have are friendly you they and families of the
Target sentence: have you heard of @near and @name they are the average friendly families
Decoded sentence: families are welcome at @name they have a average customer rating and are not family-friendly
Bleu score: 0.4992103170949989

These additional examples show cases, where the neural network did a rather poor job according to the BLEU score. But if the last example is analyzed, it can be seen that even the target sentence is not grammatically correct. So even though the decoded sentence did not contain all the words from the input and did not recreate the target sentence, it can be said that the generated sentence makes more sense from a grammatical standpoint.

For all the samples the BLEU score, recall and precision was calculated. The averages scores were:
Bleu score: 0.9286144193974197
Precision: 0.9630055907925232
Recall: 0.9596015752708394

It was also analyzed whether there is a visible correlation between the sentence length and the BLEU score, which according to the results in table 7.11 is not the case.

Correlation between input sentence length and BLEU score	0.03938729
Correlation between output sentence length and BLEU score	0.05036982
Correlation between difference of input and output sentence length and BLEU score	-0.11418943

Table 7.11.: Correlation of BLEU score and sentence length

To try to further improve the model an attention mechanism could be implemented on the decoder side, so that the output not just relies on the hidden states from the encoder. Another approach which could be helpful would be a beam search mechanism.

7.4. Conclusion

In this chapter the results from different tasks were discussed. First, it was shown that language generation with the self crawled dataset did not work well. Afterwards, the results from multiple models with the goal of filling a single gap in a sentence were presented, which turned out to be of good quality. Next, it was tried to fill in multiple gaps, which also showed promising results. In the end, the results of a sequence-to-sequence model, which is able to bring a sentence back into order, were displayed.

8. Conclusion

The first part of the thesis illustrated how crucial the quality of the dataset is for a neural network to perform well. A neural net, that has been proven to perform well in the project thesis given a specifically prepared dataset of restaurant reviews, was unable to learn and generate anything meaningful when given the self created dataset of laptop reviews. One issue that could be further analyzed is how the performance of the network could be improved by for example using a different approach during the learning process or by adapting the dataset. Another question that was unable to be answered during this thesis is how well the neural network from the previous project would perform on a dataset from another domain of similar quality to the restaurant dataset.

The second part of the thesis proved that it is possible for neural networks to learn to not only fill gaps in a sentence but also to bring unordered sentences back into the right order. Even a combination of both worked well when the given sentence contained a clear and correct structure. One open question would be how well this approach would work for the problem mentioned in the first part of the thesis. Instead of the current situation, where a feature vector that defines what features should be described in the output is given, a vector of words have be included in the output could be used as the input. For example the input vector containing of the words "laptop cpu powerful" could turn into the output sentence "The laptop comes with a powerful CPU.".

Further future work that could be done is to write a special LSTM cell for the task of sentence ordering, so that it gets a vector with the available words and sets generated words internally to zero, so that it would not generate them again. It could also help to write a beam search for the LSTM and CNN models, so that multiple solutions are considered. Additionally, it could be interesting to build an ensemble over the different models and see, if this boosts the output performance. An improvement for the sequence-to-sequence model could be to use an attention mechanism, so that the decoder not solely relies on the encoder state but gets additional information.

Appendix

A. How to use the software

A.1. Requirements

In order to use the developed system, the following software is required. The specified version does not necessarily have to be met, but the software was developed with this versions.

- python 3.5.3
- pip 9.0.1

Additionally the following python packages in the following list must be installed. If pip is used as python package manager the packages can be installed with the command

Listing A.1: pip package installation on Linux

```
$ pip3 install "packagename" --user
```

- numpy 1.13.3
- tensorflow-gpu 1.5.0
- h5py 2.7.0
- keras 2.1.4
- matplotlib 2.0.0

A.2. Software

The files containing the source code are in the folder "src" on the CD. Some of the scripts in there depend on files from the "utils" subfolder. In there are as example preprocessing scripts.

The Jupyter notebook seq2seq contains the sourcecode to the sequence to sequence model which brings sentences back in order. seq2seq-gaps is the model that has one placeholder and seq2seq-multigaps contains the model for multiple gaps.

To sample from the single input lstm or cnn models the training scripts need to be run first. For every notebook there is a python file with the same name and "training" behind the name. After training the notebooks can be executed to sample for the trained models. For the other models the training and sampling sections are defined in the same file.

B. Example Reviews

B.1. Notebookcheck

Here is an example review copied from notebookcheck [1].
Title: Acer Aspire 3 A315-41 (Ryzen 3 2200U, Vega 3, SSD, FHD) Laptop Review
Introtext: Simple office notebook. Acer's 15.6-inch laptop scores some major points with its hardware combination of a powerful Ryzen 3 2200U APU, a fast SSD and 8 GB of dual-channel RAM. However, the screen and thermal performance disappoint.

Review: The Aspire 3 A315-41 is an office notebook that is powered by a Ryzen 3 2200U APU. We have already reviewed several Aspire 3 models with the Aspire 3 A315-51-36YU, the Aspire 3 A315-51-55E4, the Aspire 3 A315-21-651Y and the Aspire 3 A315-51-30YA. The competing devices include: The Acer Aspire 3 A315-51, the HP 250 G6, and the Lenovo V330-15IKB.
Please note: In many online stores, the A315-41 is advertised with images that depict the A315-51, the A315-31 or the A315-21. The A315-41 comes with a different case. Acer Germany could not provide any product images for the A315-41 when we asked for them.
Case & Connectivity - Acer uses a plastic case
The Aspire 3 A315-21, the Aspire 3 A315-31 and the Aspire 3 A315-51 all use the same case. However, the Aspire 3 A315-41 comes with a different case. It is a little thicker and has different hinges. It also has larger fan grills. That being said, all Aspire 3 A315 models are made of black, brushed plastic.

All in all, the device has a good build quality. However, the display lid could have been somewhat more rigid: It can be twisted with little effort, and when it is twisted, it usually leads to image distortions. The images also become distorted when pressure is applied to the back of the display lid. The hinges can hold the display in a set position. However, there is some screen wobble. The display lid can be opened with one hand, but this will require some sleight of hand to achieve.

The Aspire 3 offers two maintenance hatches. The small hatch enables access to the RAM. The large hatch allows access to the 2.5-inch drive bay. However, this drive bay offers neither an installation frame for an HDD nor a SATA connector.

To reach the rest of the hardware you will have to remove the bottom cover. For this,

[1] https://www.notebookcheck.net/Acer-Aspire-3-A315-41-Ryzen-3-2200U-Vega-3-SSD-FHD-Laptop-Review.306442.0.html

you will have to first remove both maintenance covers. Then you will need to undo all the screws on the underside. Now the bottom cover can be removed with the help of a spatula or a (plastic) putty knife.

The A315-41 has the same selection of ports as its siblings. Acer provides the notebook with three USB Type-A ports (one USB 3.1 Gen 1 port, two USB 2.0 ports). The laptop does not offer any USB Type-C ports. An external monitor can be connected via HDMI. The SD card reader belongs to the faster specimens of its kind. When copying large chunks of data, a maximum transfer speed of 75.9 MB/s was achieved. The transfer of 250 JPG image files was completed with a speed of 86.2 MB/s. We test the SD card reader with the help of a reference SD card (Toshiba Exceria Pro SDXC 64 GB UHS-II).

The Aspire 3 comes equipped with a Wi-Fi module from Qualcomm (QCA9377). Besides the Wi-Fi standards 802.11 a/b/g/n, it also supports the fast Wireless-AC standard. The data transfer speeds that we measured under the optimal conditions (no other Wi-Fi-enabled devices in close proximity, a short distance between the notebook and the server PC) are quite average, because the device features a 1x1 MIMO antenna.

Input Devices - The Aspire 3 does not have a keyboard backlight The Aspire 3 comes with an unlit chiclet-style keyboard complete with a number pad. The keys have a slightly rough surface. They have a short travel distance and a clear actuation point. The keys are a little too mushy for our taste. During typing, the center of the keyboard exhibits some mild flex. However, this does not prove to be annoying. All in all, Acer delivers an "okay" keyboard that is well suited for regular typing.

The multitouch-enabled ClickPad occupies an area of some 10.5 x 7.8 cm (4.1 x 3 in). Therefore, there is enough space for the use of gesture controls. The smooth surface of the pad makes finger-gliding easy. The corners of the ClickPad register inputs well. The bottom of the pad, where the left and the right mouse buttons are usually located, exhibits a long travel distance and a vague actuation point.

Display - The dim display with poor viewing angles is not going to win any praise The matte 15.6-inch display of the Aspire 3 has a native resolution of 1920x1080. Both the contrast ratio (544:1) and the brightness (211 cd/m^2) are way too low.

Unfortunately, at 10% brightness and below, the screen exhibits PWM flickering with a frequency of 25000 Hz. However, such a high frequency should not lead to headaches and/or eye-strain amongst susceptible individuals.

Screen Flickering / PWM (Pulse-Width Modulation)

The display does not shine in terms of color accuracy. Straight out of the box, we measured a DeltaE 2000 color deviation of 11.27 (DeltaE less than 3 is the optimal value). Moreover, the display suffers from a noticeable bluish cast. The display can cover only 56% of the sRGB color space and 36% of the Adobe RGB color space.

By means of our color profile, the color reproduction can be improved. However, before downloading it, you should make sure that your laptop has the same display model (manufacturer + model number) as our review device, because otherwise our color profile can result in worse color reproduction. Displays from different manufacturers can often be found within notebooks from the same model range.

Verdict

The Aspire 3 A315-41 is a 15.6-inch office notebook that is powered by a Ryzen 3 2200U APU. The APU offers more than enough computing power for such usage scenarios as office work and Internet browsing. The notebook does not become hot even under full load. However, the fan is active both when the notebook is idle and when it is under load. This can be attributed to the fact that the fan tends to spin too fast even when the temperatures are relatively low.

An SSD creates a very responsive system. It can be replaced. However, to do this you will have to remove the entire bottom cover. The two maintenance hatches do not provide access to the SSD. The keyboard has left a positive impression and is fit for regular office work. It does not feature a backlight.

The battery life is okay. There is a fast SD card reader on board. The screen disappoints once again. Here, Acer uses a dim TN panel with poor contrast and viewing angles.

All in all, the built-in AMD APU has left a positive impression. It is a direct competitor to Intel's dual-core Kaby Lake CPUs. In both CPU and GPU benchmarks, the two processors are roughly on the same level.

B.2. Newegg

Here is an example review copied from Newegg [2].

Title: DELL Laptop Latitude 5580 (PXP7J) Intel Core i5 7th Gen 7200U (2.50 GHz) 4 GB Memory 500 GB HDD Intel HD Graphics 620 15.6" Windows 10 Pro 64-Bit

Text: Dell Latitude 5580: Feature-rich and versatile

A 15.6" laptop built for ultimate productivity and performance. Featuring top of the line security features and flexible docking options.

Operating System

[2] https://www.newegg.com/Product/Product.aspx?Item=1TS-000A-01MY0

Available with Windows 10 Pro - for a smooth, versatile PC experience.

Security you can rely on Log in with ease: Activate Windows Hello via optional infrared camera to facilitate facial recognition for easy and secure access.

Trusted authentication: The Latitude 5580 offers multiple security options to meet your diverse security needs. Features include essential multi-factor authentication hardware such as touch fingerprint reader, contacted FIPS 201 Smart Card Reader and Contactless Smart Card Reader NFC with Control Vault 2™ Fips 140-2 Level 3 Certification to prevent unauthorized access.

Protection from attacks: Dell™ ControlVault™ provides a more secure alternative for storing and processing passwords, biometric templates and security codes.

Manage with ease: Through Dell unique vPro extensions, you can remotely manage a fleet of devices, including diagnostics whether they are powered on or off.

Keeps up with you, and your work The power to perform: Leverage scalable performance using Intel's® latest 7th Gen Core™ i Dual (U) or Quad Core™ (H) processors, NVIDIA® graphics and a range of storage options from HDD to M.2 PCIe NVMe.

Designed with purpose: Work confidently on a laptop equipped with all-day battery life and if needed, bring your Dell Power Companion for an extended day.

Focus on work, not your notebook: Tested against 15 MIL STD 810G benchmarks, our incredibly durable systems make sure your work stays safe wherever you go.

Secure Only Dell offers industry-leading encryption, authentication including optional touch fingerprint reader and leading-edge malware prevention from a single source right out of the box. Plus, with Dell Data Protection | Protected Workspace, your data is safe across all endpoints, including external media, self-encrypting drives and in public cloud storage.

Manageable The world's most manageable laptop is built to allow flexible and automated BIOS and system configurations through Dell Client Command Suite free tools. We make it easy to deploy, monitor and update your Latitude fleet.

Reliable Features a durable, built-to-last chassis that has undergone extensive military-grade MIL-STD 810G testing that ensures your system can withstand real-world conditions.

Ports & Slots
1. Audio Combo Jack | 2. External SIM tray (optional) | 3. USB 3.0 Powershare | 4. VGA | 5. Nobel Wedge lock slot | 6. RJ45 | 7. HDMI | 8. USB 3.0 | 9. Power 10. Display Port Over Type C; optional Thunderbolt | 11. USB 3.0 | 12. SD slot

Dimensions & Weight

1. Height (front): 0.9” (23.25mm) | 2. Width: 14.8” (376.0mm) | 3. Depth: 9.87” (250.65mm) Weight: 4.19 lbs (1.90 kg)

List of Tables

List of Figures

Bibliography

[1] E. H. et al., *A comparison of neural models for word ordering*, 2017.

[2] E. K. Albert Gatt, "Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation", *Journal of Artificial Intelligence Research*, Jan. 2018.

[3] J. Brownlee. (2017). Define an encoder-decoder sequence-to-sequence model for neural machine translation in keras, [Online]. Available: https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras/ (visited on 05/13/2018).

[4] F. Chollet, *A ten-minute introduction to sequence-to-sequence learning in keras*, Blog, 2017. [Online]. Available: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html.

[5] J. L. Elman, *Finding structure in time*, 1990.

[6] E. Hasler, F. Stahlberg, M. Tomalin, A. de Gispert, and B. Byrne, "A Comparison of Neural Models for Word Ordering", *ArXiv e-prints*, Aug. 2017. arXiv: 1708.01809 [cs.CL].

[7] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, 1995.

[8] J. Juraska, P. Karagiannis, K. K. Bowden, and M. A. Walker, "A Deep Ensemble Model with Slot Alignment for Sequence-to-Sequence Natural Language Generation", *ArXiv e-prints*, May 2018. arXiv: 1805.06553 [cs.CL].

[9] J. Liu and Y. Zhang, *An empirical comparison between n-gram and syntactic language models for word ordering*, 2015.

[10] J. Novikova, O. Dušek, and V. Rieser, "The E2E dataset: New challenges for end-to-end generation", in *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, arXiv:1706.09254, Saarbrücken, Germany, 2017. [Online]. Available: https://arxiv.org/abs/1706.09254.

[11] J. Schaub and T. Fierz, *Natural language generation using neural networks*, 2017.

[12] A. Schmaltz, A. M. Rush, and S. M. Shieber, *Word ordering without syntax*, 2016.

[13] B. P. Tobias Kaufmann, *Syntactic language modeling with formal grammars*, 2012.

[14] T. Wen, M. Gasic, N. Mrksic, P. Su, D. Vandyke, and S. J. Young, "Semantically conditioned lstm-based natural language generation for spoken dialogue systems", *CoRR*, vol. abs/1508.01745, 2015. arXiv: 1508.01745. [Online]. Available: http://arxiv.org/abs/1508.01745.