This is my title

Bachelorarbeit

Christopher Schütz | 2462248

Bachelor Wirschaftsinformatik

TECHNISCHE UNIVERSITÄT DARMSTADT

WIRTSCHAFTS INFORMATIK

Christopher Schütz
Matrikelnummer: 2462248
Studiengang: Bachelor Wirtschaftsinformatik

Bachelorarbeit
Thema: “This is my title”

Eingereicht: 18. März 2020

Betreuer: M.Sc. Felix Peters

Prof. Dr. Peter Buxmann
Fachgebiet Wirtschaftsinformatik | Software & Digital Business
Fachbereich Rechts- und Wirtschaftswissenschaften
Technische Universität Darmstadt
Hochschulstraße 1
64289 Darmstadt

Ehrenwörtliche Erklärung gemäß §22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Chrisopher Schütz, die vorliegende Bachelor-Thesis gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Darmstadt, den 18. März 2020

Abstract

This is my abstract.

Table of contents

List of figures

List of tables

1 Introduction

Here comes a beautiful introduction.

2 Background

This chapter serves to explain the foundations of natural language processing (NLP), especially the subpart of language modeling, needed to understand the problem and methodology used in this thesis. The two main building blocks of this thesis are the examination of a state-of-the-art language model (LM) and its ability to generate high quality, human-like text on the one hand and methods to distinguish such generations from human written text on the other hand. In order to understand the problem and methodology used in this thesis, this chapter explains the foundations of NLP (ch. 2.1), especially the subpart of language modeling, and the foundations of deep learning (ch. 2.2) especially under the aspect of sequence classification.

2.1 History of language modeling

Natural language processing (NLP) is “an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things”[1]. The applications of NLP such as speech recognition, sentiment analysis, question answering and others are numerous[2]. Moreover, these applications are already being heavily used by industry and consumers alike e.g. in the forms of digital voice assistants, sentiment analysis for recommender systems and browser search bars[3]. The subcomponent of NLP needed when it comes to tasks like machine translation, predictive typing or summarization that involve either generating text or estimating the probability of text is called language modeling. The following notation mostly follows the one from the CS224 Stanford Natural Language Processing with Deep Learning lecutre by Chris Manning [4]. The following chapters will cover word representation (ch. 2.1.1) as the foundation for language models, followed by a brief explanation of the basics of language models (ch. 2.1.2) and will end with the most common and most used language model types and architectures (ch. 2.1.3 - ch. 2.1.4).

2.1.1 Word representation

The starting point for most NLP related tasks lies within the preprocessing of the textual input data. These must first be converted into a semantically meaningful numerical representation to allow for effective computation. The simplest way to represent a word numerically is by treating words as *one-hot* vectors. Using this format, one word is encoded into an n-dimensional vector of numbers where n is the vocabulary size and each entry takes on the value 0 except for the one that corresponds to the indexed word which takes on the value 1. This process generates very sparse feature vectors for each input word, which is no problem for simple classification tasks, but is unsuitable for larger problems as the dimension size increases for each word added to the vocabulary. Another issue of one-hot vectors is the fact that these representations hold no notion of similarity: If we take a look at the representations for the words “plane“ and “airplane“ we

[1] Chowdhury (2003).
[2] Gatt & Krahmer (2017).
[3] Valdivia; Luzón; Herrera (2017); Klopfenstein et al. (2017); Pandu (2019).
[4] https://web.stanford.edu/class/cs224n/index.html

would get two vectors that are orthogonal, just as any two vectors in the whole vocabulary. This poses a problem as every word's similarity to other words is the same and we can not encapsulate the meaning of the word. Yoav Goldberg writes in his primer on neural networks for language processing that one-hot vectors should only be considered for problems where the model has only a small amount of input features, the inputs do not have to share model parameters and where there is a lot of data to learn from[5].

Otherwise, the currently dominant approach in the field is to use word embeddings as feature vectors. Word embeddings follow the so called "distributional semantics", which state that a word's meaning is given by the words that tend to occur in a similar context. This idea of context-dependent nature of meaning was first introduced by english linguist J.R. Firth [6] with the famous quote:

> *You shall know a word by the company it keeps.*

Following this idea we represent each word by a distribution of weights across many dimensions. Now, instead of having a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the entries in the vector, and each element in the vector contributes to the definition of many words. Having this new form of word vectors allows us to capture meaningful semantic and syntactic regularities between words in an expressive way. A simple way to illustrate this is depicted in figure 1.
Here, our word vectors contain the knowledge that the difference between a "king" and a "queen" primarily lies within the gender of a person. Thus, if we subtract the word vector of "man" from "king" and we add the word vector of "woman" we get a word vector that is very similar to the one of "queen".

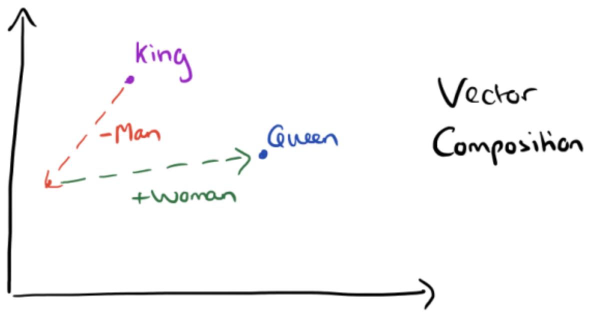

Figure 1: Word2vec king queen composition

The most famous realization of these embeddings was introduced by the researchers of Google around Tomas Mikolov in their paper "Distributed Representations of Words and Phrases and their Compositionality"[7]. Hereafter, their proposed approach, Word2Vec, along with their used notation will briefly be presented, as its core ideas are found throughout other popular word

[5] Goldberg (2015).
[6] https://web.stanford.edu/class/linguist236/materials/ling236-handout-05-09-vsm.pdf
[7] Mikolov et al. (2013).

embedding frameworks as well.
Given a sequence of training words $w_1, w_2, \dots, w_T$, the objective (ch 3.3.3 [HARDCODED, ref to objective function]) is to minimize the average negative log likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta) \tag{1}$$

where m is the fixed window size containing the context words around the center word w_j. Enlarging this window size results in more training examples and can lead to a higher accuracy, at the expense of training time. In order to compute the term $P(w_{t+j}|w_t;\theta)$ a regular softmax function (ch 3.3.3 [HARDCODED, ref to softmax function]) can be applied

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \tag{2}$$

where we use two vectors per word w: v_w when w is a center word and u_w when w is a context word. Optimizing the objective function results in high-quality distributed vector representations. It should be noted that for computational efficiency modified versions of these functions such as negative sampling or hierarchical softmax are used, as the presented ones do not scale well.

Even though word2vec embeddings are a powerful representation they do face certain limitations, which is why *Stanford University* [8] took it upon itself to create an improved variant called GloVe[9]. While word2vec ignores that some context words appear more ofthen than others, GloVe stresses the importance of frequencies of co-occurrences and that these should not be "wasted“ as additional training examples. Therefore, GloVe builds word embeddings in a way that a combination of word vectors relates directly to the probability of these words' co-occurrence in the corpus.
Another alternative to word2vec is *fasttext* by Facebook, which generates word vectors that generalize better, need less training data and can be trained "on more than one billion words in less than ten minutes using a standard multicore CPU, and classify half a million sentences among 312K classes in less than a minute“ according to the authors[10]. FastText also takes word parts into account, i.e. FastText not only stays on a word level of depth but also goes into the character level.

[8] https://www.stanford.edu
[9] Pennington; Socher; Manning (2014).
[10] Joulin et al. (2016).

2.1.2 Foundations of language modeling

As the necessary preprocessing has been dealt with, we can now get to the actual task of language modeling, which is the task of predicting “what word comes next“. More formally, this means: Given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, compute the probability of the next word $x^{(t+1)}$:

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) \tag{3}$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$.
Having a system that does so allows us to assign a probability to a text excerpt of length T:

$$\begin{aligned} P(x^{(1)}, \dots, x^{(T)}) &= P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \cdots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)}) \\ &= \prod_{t=1}^{T} P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)}) \end{aligned} \tag{4}$$

Developing and improving language models is a task central to language understanding by which we can measure how well machine learning systems actually comprehend natural language. This is demonstrated by the fact that “often (although not always), training better language models improves the underlying metrics of the downstream task (such as word error rate for speech recognition, or BLEU score for translation), which makes the task of training better LMs valuable by itself”[11].
Since the first significant language model was proposed back in 1980[12], language models and their architectures have gone through many changes. Especially the rise of deep learning and new network models such as recurrent neural networks (RNNs) or transformers have fueled language modeling research in the past few years.

2.1.3 N-Gram models

One solution in dealing with the problem of predicting a word after a sequence of $(n-1)$ words in the form of a Markov model, i.e. the probability of each event depends only on the state attained through the previous event, is called an n-gram model. An “n-gram” hereby denotes a chunk of n consecutive words. The core idea is that the probability of a word w_i occurring in the i^{th} instance after a sequence of $(i-1)$ preceding words can be approximated by observing only the preceding context of $(n-1)$ words.

[11] Józefowicz et al. (2016).
[12] Rosenfeld (2000).

Following this insight we can compute the probability of all n-grams in a corpus of text by simply counting their occurrences. Doing so allows us to calculate these conditional probabilities like so:

$$
\begin{aligned}
P(x^{(t+1)}|x^{(t)},\ldots,x^{(1)}) &= P(x^{(t+1)}|x^{(t)},\ldots,x^{(t-2+2)}) \\
&= \frac{P(x^{(t+1)},x^{(t)},\ldots,x^{(t-2+2)})}{P(x^{(t)},\ldots,x^{(t-2+2)})} \\
&\approx \frac{\text{count}(x^{(t+1)},x^{(t)},\ldots,x^{(t-n+2)})}{\text{count}(x^{(t)},\ldots,x^{(t-n+2)})} \text{(statistical approximation)}
\end{aligned} \tag{5}
$$

N-Gram Models where n=1 are called Unigram, n=2 Bigram and n=3 respectively Trigram.

[Talk about the text generation process]

Even though n-gram models have been widely used especially due to their simplicity and scalability they do face certain limitations that have led to a decrease in their popularity. One problem that can arise when computing n-gram frequencies/probabilites is that n-grams encountered in a test setting do not appear in the corpus that the model was trained on. This leads to the probability of that n-gram being 0 [insert formula]. In order to counter this different smoothing techniques can be applied.

Sparsity: what if "students opened their w" or "students opened their" never occurred? solutions -> smoothing, backoff [cite stanford textbook chapter]

Storage: Need to store counts for all n-grams you saw in the corpus. NO SOLUTION.

Lack of understanding: Perhaps the biggest drawback of n-gram models, though, is their very limited context size. In practice no n>5 usually. While produced output is often grammatically correct, there is an evident [lack] of coherence.

"today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share"

Even though n-gram language models are still widely used in speech recognition due to their high efficiency in inference, their limitations caused by poor generalization to unobserved n-grams and inability to capture long range dependencies led to the rise of neural language models [ice paper 8].

2.1.4 Neural language models

Neural Net Description Neural networks avoid the curse of dimensionality problem (the number of possible sequences of words increases exponentially with the size of the vocabulary) that arises with language modeling by representing words in a distributed way, as non-linear combination of weights.

One solution could be a fixed window neural language model:

[insert graphic]

This architecture foregoes the need of storing all possible n-grams. However there are other problems that remain: Firstly, the fixed window will always be too small, there still is no way of processing inputs of arbitrary length. Secondly, there is no symmetry in the processing of outputs. This means that x(1)and x(2)are multiplied by completely different weights [obwohl] their output could be correlated. The idea of applying the same weight matrix repeatedly leads to a new architecture called recurrent neural networks.

2.1.4.1 Recurrent neural networks

With their paper "Learning representations by back-propagating errors" [cite ice paper 11] from 1986, David Rumelhart, Geoffrey Hinton and Ronald Williams laid the foundations for current recurrent neural network architectures. This work showed that these networks can learn an internal 'hidden' state - which is neither part of the input nor the output - that represent important features of the task domain. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs, which makes them so valuable for language modeling.

[insert architecture of an rnn]

There are a few benefits that arise from the RNN architecture. As the same weight matrix is used for every input, model size does not increase for longer input. Also, the model can process inputs of every length. And finally, this architecture allows for usage of information from the past and for symmetric input processing by using the same weight matrix. Because RNNs synthesise and reconstitute the training data in a complex way they rarely generate the same thing twice. The predictions are "therefore much better at modelling real-valued or multivariate data [rather] than exact matches" [cite ice paper 10].

The core reason that recurrent nets are more exciting is that they allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both. [use karpathy images for visualization, ice paper 9]

The form of RNNs that has been described so far is known as the so called "vanilla RNN". It is the most basic form of the RNN class of artificial neural networks. Because of the heavy computations involved in adjusting the weight matrix vanilla RNNs are susceptible to the problem of vanishing gradients. To tackle the short-comings of vanilla RNNs different RNN-based architectures have been developed: LSTMs, GRUs and others.

It is worth inspecting the details of Long-Short Term Memory RNNs, introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [ice paper 12], as their real-world usage has been crowned with success.

LSTMs

Big companies like Apple, Facebook, Google and IBM have been known to use LSTM based architectures in their products [cite all the wikipedia listed references for big companies in wikipedia] for features like “smart reply” and “quicktype”.

The underlying core concept of LSTMs is the introduction of the so called cell state in addition to the hidden state. This cell is used to store long-term information. The LSTM can hereby erase, write and read information from the cell.

[insert formulas from stanford, pictures from “understanding lstms”] [explain each cell in detail with its according formulas]

This architecture provides a better way for preserving information over many timesteps, because the forget gate provides a way of excluding ‘irrelevant’ information for the optimization.

GRUs

2.1.4.2 Convolutional neural networks

This is some text.

2.1.4.3 Attention and transformers

This is some text.

2.2 Deep Learning

This is deep learning.

3 Methodology

This chapter explains the methodology.

3.1 Dataset

In order to create a dataset suited for the classification task a few aspects had to be considered. As deep learning methods with many parameters that need to be optimized were going to be used for the classification task, many labeled training samples were needed. The dataset should ideally be comprised of equal amounts of synthetically and human generated texts in order to improve the accuracy of training.

The first idea was to look for already built datasets that are freely available as these would not only reduce the time and amount of work needed for the creation of the dataset, but also provide a benchmark against other models used on them. Because the examined classification task is fairly novel and powerful language models have just started to emerge in recent years, there is a lack of standardized data sets - prior research often focused on detection of artificially generated academic papers instead of short texts [reference to papers that use academic papers]. Furthermore, the incentive of using metadata related to text snippets and inspecting the changes in detection accuracy through it led to the motivation of building a new dataset.

For this purpose, the following sources of human created text were inspected - taking different aspects like data availability, extensibility by metadata, potential for use of text generation and minimal overlap with the pretrained GPT2 Model into account: Wikipedia, Twitter, reviews (e.g. Amazon, IMDb), Reddit comments, Reuters Corpus.

With everything taken into consideration, Wikipedia articles were chosen as the best fit for this work. Reasons for this choice were the ease of access, the vast amount of data entries (currently there are more than 6 million articles in the english Wikipedia[13]), the extensibility by metadata such as pageviews or categories and most importantly the fact that the ready-to-use GPT-2 model was explicitly not trained on any Wikipedia article since ”it is a common data source for other datasets and could complicate analysis due to overlapping training data with test evalutation tasks.”[14].
The reviews and reddit comments datasets were not chosen, because the metadata was not seen as decisive in improving detection quality. The disadvantage of the Reuters Corpus was that the training dataset used for the ready-to-use GPT2 model is comprised of many newspaper sources and thus is more likely to generate results that are rathere similar to their human written counterparts. Twitter was seen as a promising data source, but the access to its public api was shut down which is why it could not be further considered.

[13] https://en.wikipedia.org/wiki/Wikipedia:Size_comparisons#Wikipedia
[14] Radford et al. (2019).

3.1.1 Extraction Process

The two usual ways of scraping Wikipedia articles and other data like article metadata or media files from the Wikimedia Foundation[15] are through the APIs or the database dumps provided by the MediaWiki platform[16]. Both of these channels were used for different purposes:

- **Database Dumps:** Titles, clear text and article ids were extracted using a python library called WikiExtractor[17]. This tool parses the compressed and xml-formatted dumps and extracts the aforementioned fields. The output is stored in ".jsonl" (json lines) files, where each line denotes a complete JSON object.
- **Api:** The MediaWiki Api was used to retrieve metadata such as pageviews, edits, the latest edit timestamp and namespaces (categories) linked to each article via the extracted page ids, but also to access the latest news listed in WikiNews to generate exemplary news messages.

The extension of text samples by metadata was made in order to examine the following hypotheses:
1. The detection accuracy varies (significantly) across different categories.
2. The higher the edits and/or views on a page are, the lower the detection accuracy will be as the human text will be more sophisticated and better worded.
3. The more recent the last edit timestamp of an article is the lower the detection rate will be as newer information will be less likely to be present in the training data used for GPT-2.

It should be noted that only articles with a minimum length of 1000 characters were taken into consideration in order to filter out many entries that would have diminished the quality of the dataset (e.g. entries that only have a redirect notice to another article). For each article entry the last edit timestamp, the aggregated amount of page views over the last 30 days and the namespace was retrieved.

3.1.2 Generation Parameter Combination

The output style of GPT-2 differs depending on the chosen parameter combination. The parameters that influence the produced output the most are input length, maximum output length, repetition penalty and temperature.

In order to determine a parameter combination that generates convincing text all possible permutations between the parameters given the values listed in table 1 were used to generate samples for 50 articles. As a metric for evaluation a fine-tuned large (1.5GB weights size as a .pt file) RoBERTa-based model with a mixture of temperature 1 and nucleus sampling outputs was chosen. This configuration was elected as it generalizes well to outputs generated using different

[15] https://wikimediafoundation.org/
[16] https://www.mediawiki.org/wiki/MediaWiki
[17] https://github.com/attardi/wikiextractor

Table 1: **Value ranges for all modified parameters. The sampling method indices refer to nucleus sampling (1), top k (2) and beam search (3).**

Parameter name	Value range		
max. input length (in chars)	20	40	60
max. output length (in tokens)	50	100	200
temperature	0.7	1.0	1.3
repetition penalty		1.0	1.3
sampling method	1	2	3
number of beams	5	7	

Table 2: **Exemplary word and character split inputs and according outputs**

character split input

Input	tennis is a spo
Output	tennis is a spoilt brat by all accounts, but can he work with the likes of Krajicek? Can he become a player on the big stage who can keep his cool with the big boys?

word split input

Input	tennis is a sport
Output	tennis is a sport which requires a lot of energy and runs at a very fast pace. Getting tired can really impair your performance.

sampling methods [citation gpt2 paper]. On top of that, human evaluation was performed while reading through samples created by the best performing parameter combinations according to the RoBERTa evaluation.
The main findings were that a higher repetition penalty as well as shorter output length were key factors for better text generation. In addition to that, feeding the LM word split input sentences as opposed to character split input sequences also improved quality (table 2).

Thus, the finally selected parameter values were:

Maximum Input Length - 60 characters
The input that was fed into GPT-2 XL was the first sequence of plain text in a Wikipedia article, i.e. no infobox or content table text was considered. Additionally, before feeding the 60 character String into the LM it was split at the last word. This was done because the quality of the generated output tended to increase when input was given in full words rather than in characters and thus having many times split words.

Maximum Output Length - 50 Tokens
This corresponds to an average of about 240 - 260 characters per text (in comparison: the max tweet length is 280 characters). As this is a size that occurs a lot especially in social media or breaking news headlines (with the subtitle)[18], the focus was placed on shorter text snippets. Also, the amount of needed compute power increased in a linear fashion with the output length, which would have doubled or quadrupled the dataset creation time of 14 days.

[18] Lee (2014).

Temperature - 1.0
Both lowering and increasing (to 0.7 and 1.3) the temperature led to an increased detection of synthetically generated text, which is why the temperature was left at its original value.

Repetition Penalty - 1.3
As repetitions were not desired in the generated output the repetition penalty was increased from its default value of 1.0 to 1.3. under the assumption that most Wikipedia articles do not contain repetitions (especially in their abstracts) unlike for instance dramatic text, where text repetition can be used as a stylistic device.

Sampling method - beam search (3)
The beam search algorithm for decoding of the LM output was chosen as the best fit.

Number of beams - 5
Initially thought of as a parameter that would have a big impact on the quality of the generated text, it was found out that altering the number of beams used in the beam sampling strategy when predicting the next most probable tokens was had only little impact on the LM text generation. The values chosen ranged between 5 and 10 as this is currently the de facto standard [cite Stanford lectures] in research.

3.1.3 Data Building

After downloading the Wikipedia dumps with a compressed size of 17GB the natural text was extracted via the *wikiextractor*[19] library (filtering out texts with a character length less than 1000) and the metadata was extracted and parsed using *SAX Parser*[20] ("Simple Api for XML"). In order to finalize the data set creation, synthetic text had to be generated for each article's first 60 characters twice. The double generation was performed in order to select the sample that GPT-2 felt more confident on and improve the data set quality.

Given the aforementioned parameter configurations, the first pipeline runs took place in Google Colaboratory[21], a free jupyter notebook cloud hosting platform which provides users with a GPU, in this case an Nvidia Tesla K80[22]. For the generation of synthetic text, the large GPT-2 model (774M parameters) was used. The generation of each text snippet took about 11 seconds. As soon as the pipeline was fully functioning the jupyter notebooks were converted into python modules and the code was transferred to a Google Cloud[23] deep learning virtual machine[24] (vm) with 13GB of memory, a 100GB standard persistent disk, and 2 Nvidia K80 GPUs. Google Cloud was elected for the reason of providing users with a free credit of 300$, while other cloud computing

19 https://github.com/attardi/wikiextractor
20 https://docs.python.org/3/library/xml.sax.reader.html
21 https://colab.research.google.com/notebooks/intro.ipynb
22 https://www.nvidia.com/en-gb/data-center/tesla-k80/
23 https://cloud.google.com/
24 https://console.cloud.google.com/marketplace/details/click-to-deploy-images/deeplearning?_ga=2.99839453.-2121832178.1549923708

providers such as AWS[25] or Azure[26] only provide a free credit of 100$. Furthermore, the specialized deep learning vm was chosen as it is configured to support common GPU workloads out of the box and removes the task of setting up a high-performance computing environment by having all the needed libraries (e.g. PyTorch, CUDA) and corresponding drivers preinstalled and with a cost of 295.20$ per month also being less expensive than choosing the same hardware configuration oneself with a cost of about 500$ per month.

The creation of the data points was parallelized and the total time needed for the dataset creation was approximately 14 days. The data points and log files were stored daily in a Google Storage Bucket[27] and could then be downloaded into a local machine for manual inspection.

The finalized JSON format of a datapoint is shown in Listing 1. The key ”*label*” indicates whether the text entry was created by a human (0) or by GPT-2 (1).

Listing 1: Example of a data point

```
{
        "meta": {
                "id": "565318",
                "input": "Lost in Space is a 1998 American science-
                   fiction adventure",
                "pageviews": 57062,
                "touched": "2020-02-26T06:53:05Z",
        },
        "label": 0,
        "title": "Lost in Space (film)",
        "text": "Lost in Space is a 1998 American science-fiction
           adventure film directed by Stephen Hopkins, and starring
           William Hurt, Matt LeBlanc, and Gary Oldman. The plot is
           adapted from the 1965-1968 CBS television series \"of the
           same name\". Seve"
}
```

3.2 Approach

This is the approach part.

3.3 Infrastructure

This is the infrastructure part.

25 https://aws.amazon.com/
26 https://azure.microsoft.com/en-us/features/azure-portal/
27 https://cloud.google.com/storage/docs/json_api/v1/buckets

4 Future work

Here I have to talk about improvements through nucleus sampling proposed in[28].

[28] Holtzman et al. (2019).

References

Chowdhury, Gobinda G. (2003): *Natural language processing*, In: Annual Review of Information Science and Technology, 37 (1), pp. 51–89 https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440370103.

Gatt, Albert & Krahmer, Emiel (2017): *Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation*, In: CoRR, abs/1703.09902 http://arxiv.org/abs/1703.09902.

Goldberg, Yoav (2015): *A Primer on Neural Network Models for Natural Language Processing*, In: CoRR, abs/1510.00726 http://arxiv.org/abs/1510.00726.

Holtzman, Ari et al. (2019): *The Curious Case of Neural Text Degeneration*, In: CoRR, abs/1904.09751 http://arxiv.org/abs/1904.09751.

Joulin, Armand et al. (2016): *Bag of Tricks for Efficient Text Classification*, In: CoRR, abs/1607.01759 http://arxiv.org/abs/1607.01759.

Józefowicz, Rafal et al. (2016): *Exploring the Limits of Language Modeling*, In: CoRR, abs/1602.02410 http://arxiv.org/abs/1602.02410.

Klopfenstein, Lorenz Cuno et al. (2017): *The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms*, In: Proceedings of the 2017 Conference on Designing Interactive Systems, New York, NY, USA Association for Computing Machinery, DIS '17 https://doi.org/10.1145/3064663.3064672, ISBN 9781450349222, p. 555–565.

Lee, Kevan (2014): *The proven ideal length of every tweet, Facebook post, and headline online*, In: Fast Company, Apr.

Mikolov, Tomas et al. (2013): *Distributed Representations of Words and Phrases and their Compositionality*, In: CoRR, abs/1310.4546 http://arxiv.org/abs/1310.4546.

Pandu, Nayuk (2019): *Understanding searches better than ever before*, No address in https://www.blog.google/products/search/search-language-understanding-bert/, Last accessed: 20.02.2020.

Pennington, Jeffrey; Socher, Richard & Manning, Christopher (2014): *Glove: Global Vectors for Word Representation*, In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar Association for Computational Linguistics https://www.aclweb.org/anthology/D14-1162, pp. 1532–1543.

Radford, Alec et al. (2019): *Language models are unsupervised multitask learners*, In: OpenAI Blog, 1 (8), pp. 2–3.

Rosenfeld, R. (2000): *Two decades of statistical language modeling: where do we go from here?* In: Proceedings of the IEEE, 88 (8), pp. 1270–1278, ISSN 1558–2256.

Valdivia, A.; Luzón, M. V. & Herrera, F. (2017): *Sentiment Analysis in TripAdvisor*, In: IEEE Intelligent Systems, 32 (4), pp. 72–77, ISSN 1941–1294.