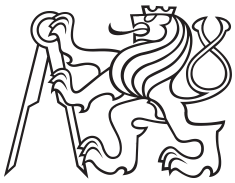
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Natural Language Generation from Knowledge-Base Triples

Ondřej Kobza

Supervisor: Ing. Petr Marek
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kobza** Jméno: **Ondřej** Osobní číslo: **457004**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Generování přirozeného jazyka ze znalostních databází

Název diplomové práce anglicky:

Natural Language Generation from Knowledge-Base Triples

Pokyny pro vypracování:

1.Prozkoumejte veřejně dostupné množiny dat vhodné pro natrénování modelu pro generaci přirozeného jazyka ze znalostních databází.
2. Prozkoumejte existující zdroje relevantní k dané úloze.
3. Na základě zdrojů z bodu 2. navrhněte možná řešení zadaného problému.
4. Implementujte a natrénujte navrhované modely na datové množině z bodu 1.
5. Najděte (popř. navrhněte) metriku pro hodnocení kvality jednotlivých modelů.
6. Porovnejte výsledky jednotlivých modelů/řešení, zhodnoťte jednotlivá řešení na základě metriky z předchozího bodu a na základě manuální evaluace.
7. Vytvořte API pro použití modelu (jenž dosahoval nejlepších výsledků v bodě 6.) tak, aby bylo možné daný model integrovat s dialogovým systémem.

Seznam doporučené literatury:

Ferreira, T. C., van der Lee, C., van Miltenburg, E., & Krahmer, E. (2019). Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. arXiv preprint arXiv:1908.09022.
Gardent, C., Shimorina, A., Narayan, S., & Perez-Beltrachini, L. (2017, September). The WebNLG challenge: Generating text from RDF data. In Proceedings of the 10th International Conference on Natural Language Generation (pp. 124-133).
Jurafsky, Daniel & Martin, James. (2008). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.
Zhu, Y., Wan, J., Zhou, Z., Chen, L., Qiu, L., Zhang, W., ... & Yu, Y. (2019, July). Triple-to-text: converting RDF triples into high-quality natural languages via optimizing an inverse KL divergence. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 455-464).
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Petr Marek, velká data a cloud computing CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **21.02.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2023**

Ing. Petr Marek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to express my sincerest gratitude and acknowledgment to my supervisor Ing. Petr Marek, for the guidance and helpful advices. Furthermore, I would like to thank Ing. Jan Pichl for helpful consultations.

Declaration

I, Ondřej Kobza, declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 21. 5. 2021

Abstract

The main goal of this master thesis is to create a machine-learning-based tool that is able to verbalize given data, i.e., from given RDF triples; it should be able to create a corresponding text in a natural language (English) such that the text must be grammatically correct, fluent, must contain all information from the input data and cannot have any additional information.

The thesis begins with examining the publicly available datasets; then, it focuses on the architectures of statistical machine learning models and their possible usage for natural language generation. The work is also focused on possible numerical text representation, text generation by machine learning models, and optimization algorithms for training the models.

The next part of the thesis proposes two main solutions to the problem and examines each of them. Automatic metrics evaluate all systems, and the best performing models are then passed to a human (manual) evaluation.

The last part of the thesis focuses on implementing the final application and its deployment for production.

Keywords: RDF Triple, Machine Learning, Natural Language Generation, LSTM, Transformer, T5, Roberta

Supervisor: Ing. Petr Marek

Abstrakt

Cílem této diplomové práce je vytvořit nástroj jenž za pomocí strojového učení dokáže verbalizovat data, t.j. ze vstupních dat ve formě RDF trojic dokáže vytvořit odpovídající text v přirozeném jazyce (angličtina) takový, že bude gramaticky a mluvnicky správný, bude obsahovat veškeré informace ze vstupních dat a nebude obsahovat žádné informace navíc.

Práce nejprve zkoumá dostupná data, poté se zabývá architekturami modelů pro statistické strojové učení a jejich možné použití pro generování přirozeného jazyka. Práce se taktéž zabývá numerickou reprezentací textu, generováním textu pomocí učících se modelů a optimalizačních algoritmů pro trénování těchto modelů.

V další části práce jsou navrženy dva rozdílné přístupy pro řešení zadání práce. Navržené přístupy jsou poté zhodnoceny pomocí automatických metrik a nejlepší systémy jsou zhodnoceny manuálně.

Závěr této diplomové práce je věnován nasazení výsledné aplikace pro produkční běh.

Klíčová slova: RDF Triple, Strojové Učení, Generování Přirozeného Jazyka, LSTM, Transformer, T5, Roberta

Překlad názvu: Generování Přirozeného Jazyka ze Znalostních Databází

Contents

Figures

Tables

ctuthesis t1606152353

Chapter 1

Introduction

In recent years one could witness a massive expansion of conversational systems. From simple dialog systems, such was Eliza (Weizenbaum, 1966), the technology developed into advanced conversational agents, such as Microsoft Cortana [1], Apple Siri [2] or Amazon Alexa [3].

Conversational agents consist of many parts. Thus, there are many challenges during the development of such software – one such challenge is to be able to use structural information (e.g., from knowledge bases) during a conversation with a client. The structural information is typically in the form of RDF triples and hence it is crucial to be able to generate natural texts from the RDF triples (i.e., to verbalize the RDF triples). In this thesis, the problem is further referred to as the 'triples-to-text' task.

One of many conversational agents that needs the ability to make natural texts from RDF triples is Alquist AI (Pichl et al. (2018), Pichl et al. (2020b), Pichl et al. (2020a)), which was developed at CTU CIIRC within the framework of the Amazon Alexa Prize. Alquist AI took second place in 2018 and 2019 and third place in 2020.

This thesis aims to solve the 'triples-to-text' task for Alquist AI to help it in the current Amazon Alexa Prize (2021).

[1] https://www.microsoft.com/en-us/cortana
[2] https://www.apple.com/siri/
[3] https://www.amazon.com/b?ie=UTF8&node=21576558011

1.1 Structure of the Thesis

The following sections of this chapter focus on a more detailed task definition and available data sets for the task.

The second chapter is describing theoretical frameworks which are necessary for solving the task with a machine-learning system.

Chapters three and four focus on two different main architectures of statistical machine-learning models. Both chapters propose solutions for the triples-to-seq task; each solution is based on the architecture described in the particular chapter.

The fifth and sixth chapters focus on an automatic evaluation of the earlier proposed systems, while the seventh and eighth chapters focus on a comparison of the proposed approaches. The former compares computational costs of various model–architectures, while the latter one is comparing their performance.

The ninth chapter is focused on a human (manual) evaluation of the best systems for each model–type.

The thesis ends with a description of the final application, its API, and with a final conclusion.

1.2 Task Definition

An RDF triple consists of three parts:

1. subject
2. predicate
3. object

Where predicate determines the relationship between subject and object. An example of an RDF triple can be:

```
subject: Abilene,_Texas
predicate: cityServed
object: Abilene_Regional_Airport
```

This thesis aims to tackle the problem of text generation from such RDF triples or triple sets. Thus given a set of RDF triples, the main task is to generate a natural language utterance, such that the utterance contains all information in the triple set and does not contain irrelevant information (information not present in the set). The utterance must also be fluent and grammatically correct. Both the triples and generated utterances must be in English.

1.3 Data

Since the topic of the thesis corresponds to the WebNLG challenge (Gardent et al. (2017), Castro Ferreira et al. (2020)), it is natural to use this dataset, so the performance of the proposed solutions in this thesis can be compared with the results of the systems that participated in the competition. Since many pieces of research which were done on the topic of verbalizing RDF triples are using one of the WebNLG datasets, one can easily compare the solutions presented in the thesis with the current state-of-the-art solutions.

The WebNLG dataset has two main versions — one used in the WebNLG Challenge 2017 and one used in WebNLG Challenge 2020:

1.3.1 WebNLG 2017 Dataset

The data consists of data pairs with 15 categories; 10 are seen (they are in training and validation set), five are unseen (they are present in the test set only). The test set contains all 15 categories (Gardent et al., 2017).

1.3.2 WebNLG 2020 Dataset

There are the 15 categories from the previous dataset (but all are seen) + there is one additional seen category. The test set contains data of seen categories, data of seen categories but with unseen entities (i.e., unseen subjects and objects), and three additional unseen categories (Castro Ferreira et al., 2020). Despite the low amount of unseen categories, 50% of the test data is of one of the unseen categories, and above 20% of the data are of a seen category, but an unseen entity.

For the first version of the WebNLG dataset, there also exists an enriched version (Castro Ferreira et al., 2020), which contains additional information:

1. **sorted tripleset:** The triples are sorted in the same order as is the corresponding information in the target sentence. Plus, the whole set is fragmented into subsets; each subset corresponds to one single sentence.
2. **entity map:** Entities (subjects and objects) are mapped to specific tags. There are also two types of target texts - raw texts and texts where the subjects or objects are replaced by the tags (AGENT-n, PATIENT-n, BRIDGE-n - which are standard tags in NLP).
3. **lexicalization:** It is the target sentence with the tags but in the form of the infinitive. Additionally, there are marks indicating to which form should be particular predicates transformed. For instance:

```
<lexicalization>
AGENT-1 which VP[aspect=simple,tense=present,voice=active,
    ↪ person=3rd,number=singular] be located in PATIENT-1 .
    ↪ AGENT-1 VP[aspect=simple,tense=past,voice=passive,
    ↪ person=null,number=singular] establish in PATIENT-2
    ↪ and VP[aspect=simple,tense=present,voice=active,
    ↪ person=3rd,number=null] fall under DT[form=defined]
    ↪ the category of PATIENT-3 .
</lexicalization>
```

The triplets can contain maximally seven triples in all WebNLG datasets. This restriction was discussed with the supervisor and other team members of Alquist AI. The discussion resulted in the conclusion that Alquist AI should use, in most cases, smaller sets of triples and that there is no necessity to use more than seven triples at once. Thus thresholding the set size to 7 is not a problem in this application.

1.4 Other Datasets

1. E2E: The input has a form of MR (meaning representation) instead of RDF (Novikova et al., 2017).
2. LDC2017T10: in the dataset, the source is an Abstract Meaning Representation (AMR) graph representing "who is doing what to whom," the target is a sentence (Knight).

Chapter 2

Related Work and Theory

Ferreira et al. (2019) shows that using traditional NLG (natural language generation) pipeline leads to a significant improvement in comparison with end-to-end approach. Ferreira et al. (2019) tried to train several different models, such as LSTM, GRU-RNN, and transformer. While all their models were trained from scratch on the task dataset, the current state-of-the-art approach is to fine-tune some pretrained transformers (attention-based model: (Vaswani et al., 2017)).

The participants of the first WebNLG challenge used either a template (rule-based) system, statistical machine translation or NMT (neural machine translation) systems with recurrent neural networks behind (Gardent et al., 2017) — note that a standard NMT system's architecture is equivalent to a standard seq-to-seq architecture. Similarly, Zhu et al. (2019) used a standard NMT system based (encoder-decoder architecture, where both encoder and decoder are LSTMs (Hochreiter and Schmidhuber, 1997)) but instead of using standard negative log-likelihood loss or cross-entropy loss, they proposed to optimize adjusted Kullback–Leibler divergence loss Joyce (2011) (so-called 'Inverse Kullback–Leibler divergence').

In the second WebNLG challenge, most participants used some transformer model, e.g., Kasner and Dušek (2020) fine-tuned MBART transformer (Liu et al., 2020) or Li et al. (2020) used a fine–tuned Google T5 model (Raffel et al., 2020) or Guo et al. (2020) used a pipeline, where they first ordered the triples, and then they fed the ordered triple set into a fine-tuned T5.

Besides Ferreira et al. (2019) some other researches proposed various

pipeline concepts. Harkous et al. (2020a) introduced amended GPT-2 (Radford et al., 2019) together with a ranker, which ranks topk outputs of the fine-tuned amended gpt-2, and the final output is based on the rankings. Other researches used various types of triples ordering as the first step and text generation as a second step, e.g., Guo et al. (2020).

The discussed work can be divided into three basic groups based on architecture of each system used to solve to triples-to-seq problem:

1. Template/rule-based approach
2. Encoder-decoder architecture build upon some recurrent neural networks (such as LSTM)
3. Attention model (transformer)

This thesis is focused on the last two mentioned groups. Hence the following sections of this chapter describe the two machine learning theoretical frameworks.

2.1 Sequence to Sequence Models Based on Recurrent Neural Networks

The standard RNN (recurrent neural network) sequence to sequence (seq-to-seq) framework consists of an encoder and a decoder; both of them are recurrent neural networks. The encoder takes in a series of discrete tokens $X = [x_1, x_2, ..., x_L]$ and outputs a series of hidden states. At the t-th step, the encoder takes in a token and updates the hidden state recurrently:

$$h_t^{enc} = f^{enc}(h_{t-1}^{enc}, e_x) \tag{2.1}$$

where e_x is the word embedding (Mikolov et al., 2013) of the t-th token, f^{enc} is a parametrized nonlinear function (e.g. LSTM (Hochreiter and Schmidhuber, 1997)). The output of an encoder are hidden states $H^{enc} = [h_1^{enc}, h_1^{enc}, ..., h_L^{enc}]$.

The decoder takes in the hidden states H^{enc} of the encoder as input and outputs a sequence of hidden states H^{dec}:

$$h_t^{dec} = f^{dec}(h_{t-1}^{dec}, e_{y_{t-1}}, c_t) \tag{2.2}$$

where $e_{y_{t-1}}$ is the embedding of the last output token from decoder/generator and c_t is either the last encoder's hidden state or it is an output from attention mechanism $c_t = g(h_1^{enc}, h_1^{enc}, ..., h_L^{enc})$ where g is a linear or nonlinear function. Typical choice of attention mechanism is a weighted average of the encoder's hidden states or a parametrized function, such as a multilayer perceptron (Bahdanau et al., 2016), (Wilson and Tufts, 1994), (Zhu et al., 2019).

The output of decoder is then fed into a generator, which outputs conditional probabilities of the output tokens:

$$P(y_t|x_1, x_2, ..., x_L, y_0, y_1, ..., y_{t-1}) = softmax(Wh_t^{enc} + b) \tag{2.3}$$

where W is the weight matrix of the generator and b is its bias.

If we have a bidirectional encoder, the number of hidden states for a sequence of length L would be $2L$, but the principle described above still holds.

2.1.1 LSTM

LSTM (Long short-term memory) is an artificial recurrent neural network architecture, which was developed to deal with the vanishing gradient problem and gradient explosion problem that occurred quite often in the standard RNN's architecture. The network consists of input, output, and forget gates and is defined as follows:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.4a}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.4b}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.4c}$$
$$c_t^{'} = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.4d}$$
$$c_t = f_t c_{t-1} + i_t c_t^{'} \tag{2.4e}$$
$$h_t = o_t \sigma_h(c_t) \tag{2.4f}$$

where x_t is the input vector, f_t is output of the forget gate, i_t is output of the input gate, o_t is output of the output gate, h_t is the hidden state of the whole LSTM unit, $c_t^{'}$ is cell activation vector, c_t is a cell state (the cell is supposed to be a 'memory' - it accumulates information of the whole input sequence from the beginning till the end), W, U are weight matrices and b is bias. σ is an activation function. Hochreiter and Schmidhuber (1997).

The main disadvantage of LSTM models is that they are not very suitable for transfer learning due to their tendency to 'catastrophic forgetting' (Schak

and Gepperth, 2019) (when an LSTM is fine-tuned, it tends to forget the knowledge gained during previous training). The main reason for this issue is the architecture, which does not allow high parallelization during training. Hence, LSTM-based models cannot be too large; otherwise, their training won't be possible (feasible) (generally small models are not suitable for transfer learning). Thus before the invention of transformers (Vaswani et al., 2017), transfer learning was not very common practice in NLP, unlike it is in computer vision, where Convolutional Neural Networks are widely used — they do not have such a high tendency to 'catastrophic forgetting' (Arora et al., 2019).

2.2 Sequence to Sequence Models Based on Transformers

Similar to the previous case 2.1, a seq-to-seq transformer model consists of an encoder and a decoder. The encoder comprises a Multi-Head Attention layer, residual connections, normalization layer, and generic feed-forward layer. While the decoder is similar to the encoder — it consists of the same type of layers but additionally contains several 'masked' Multi-Head Attention layers.

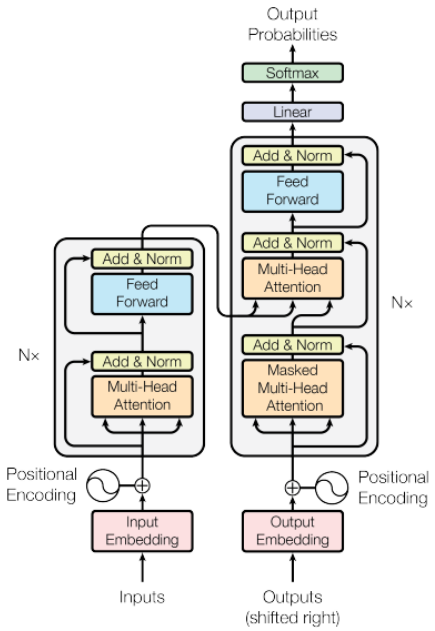


Figure 2.1: Visualization of Transformer architecture from 'Attention is all you need' paper by Vaswani et al. (2017)

Encoder: The encoder takes as the input token's embeddings summed with the positional encoding, which allows injecting information about positions of tokens in the input sequence. The positional encoding was defined by (Vaswani et al., 2017) as follows:

$$PE_{(pos,2i)} = sin(pos/10000^{(2i/d_{model})}) \tag{2.5a}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{(2i/d_{model})}) \tag{2.5b}$$

Where pos is the position of the word in the input sequence, i is the dimension (it can range from 0 to $d_{model} - 1$), d_{model} is the embedding dimension. The token's embedding layer maps a vector of real numbers to each token.
After embedding, layer and positional encoding layer follow N identical layers consisting of Multi-Head attention mechanism, normalization layer, and a standard feed-forward network. Around each those sublayers a residual connection is employed (He et al., 2015), (Vaswani et al., 2017).

Decoder: The decoder has an almost identical structure to an encoder but additionally uses 'Masked' Multi-Head attention.

2.2.1 Attention

Scaled dot-product Attention: Self-attention allows the Transformer to identify relevant words in the input with respect to the current token. It is defined as

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.6}$$

Which is a weighted average of the vector V. Given E is an output of a previous layer:

$$Q = EW_q \tag{2.7a}$$

$$K = EW_k \tag{2.7b}$$

$$V = EW_v \tag{2.7c}$$

where W_q, W_k, W_v are weight matrices (learnable parameters).

Multi–Head Attention: Instead of performing a single attention function with keys (K), values (V), and queries (Q), one can linearly project the queries, keys, and values h times with different learned projections and then on each of these projections perform the attention function in parallel. The particular outputs are concatenated and projected, which results in the final output of the multi-head attention (Vaswani et al., 2017).

Masked Multi–Head Attention: Masked Multi-Head Attention is multi-head attention adjusted to prevent positions from attending to subsequent positions (i.e., to prevent conditioning to future tokens). This masking ensures that the predictions for position i can depend only on the known outputs at positions less than i (Vaswani et al., 2017).

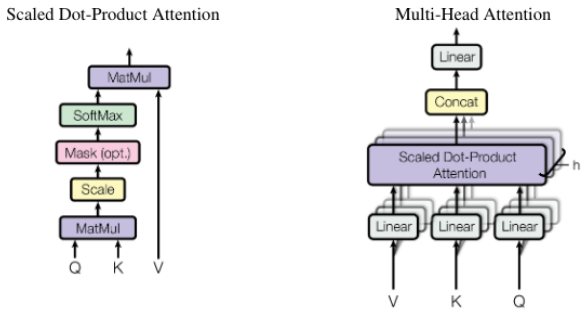


Figure 2.2: Masked Multi–Head Attention from 'Attention is all you need' paper by Vaswani et al. (2017)

The most important feature of the Transformer architecture is that it does not suffer so much from the 'catastrophic forgetting' (Chen et al., 2020) as LSTM and that training of transformers can be well parallelized (Vaswani et al., 2017). There are dozens of pretrained large transformer models, which are ready to be fine-tuned on various specific tasks. For instance, Huggingface Transformers (Wolf et al., 2020) offers many pretrained transformer models and implements different transformer-based architectures.

2.3 Text Representation in Machine Learning Models

There are many options, how to represent text numerically, from Bag–of–words, Tf-idf, word–embeddings, or byte pair encoding (BPE) (Sennrich et al., 2016). Since word–embeddings and Byte-Pair Encoding (BPE) are the state–of–the–art, this section will focus on these two.

2.3.1 Word Embedding

The idea behind word embedding is that words are defined by the context in which they occur. Thus, in an unsupervised setting, it should be possible to learn a dense vector for each word (in training data) Mikolov et al. (2013). Such representation ease in dealing with synonyms, ambiguity, and other challenges in natural language processing. Thus within word–embedding, similar vectors should belong to words that are also similar in their meaning or somehow related to each other.

However, a problem of the word–embedding are out-of-vocabulary words — thus, if such a word comes in the input sequence, it cannot be represented by any meaningful vector (its vector is typically initialized randomly). Here comes also tokenization challenge: when a sequence of words is being tokenized, one must decide, whether for instance *don't* will be treated as one word or two separate words: *do, n't* (Jurafsky and Martin, 2009).

BPE tries to tackle the tokenization and out-of-vocabulary problem. (Sennrich et al., 2016).

2.3.2 Byte-Pair Encoding

Byte-Pair Encoding was introduced by Sennrich et al. (2016) and relies on a pretokenizer that splits the training data into words (e.g., separating words by spaces). After pretokenization, BPE creates a base vocabulary consisting of all symbols that occur in the set of unique words and learns merge rules to form a new character composed of symbols from the base vocabulary. The new character is then added to the vocabulary. Merging is repeated until the size of the vocabulary is as desired. Thus, when a word is being encoded via BPE, if it is a frequent word, it might be in the vocabulary, but otherwise, it will be decomposed onto smaller, meaningful subwords.

There are also other algorithms based on BPE, such as WordPiece (Devlin et al., 2018) or Sentencepiece (Kudo and Richardson, 2018).

2.4 Decoding Output of an Sequence to Sequence Model

The outputs of a standard seq-to-seq model (or neural-machine-translation model) are conditional probabilities $P(y_t|x_1, x_2, ..., x_L, y_1, y_2, ..., y_{t-1})$ where x_i is an input token and y_i is an output token. The decoding problem can be formulate as:

Given the conditional probabilities $P(y_t|x_1, x_2, ..., x_L, y_1, y_2, ..., y_{t-1})$, find an output sequence of tokens from the target vocabulary.

One can use greedy–search and at t-th step select always token y_t with the highest conditional probability; however, such solution is not optimal — to find an optimal solution, one must try every possible combination of output tokens — unfortunately, this is usually not tractable (even with pruning). Besides greedy–search, there are other techniques, which can find a better output sequence than greedy–search — beam search and sampling.

2.4.1 Beam Search

Given a beam size N, beam search keeps track of N tokens with the highest conditional probabilities at each time step t. Then, the sequence with the highest probability is selected.

Note that if $N = 1$, beam search is equivalent to the previously mentioned greedy search.

2.4.2 Sampling

Sampling works as follows: at each time step a token y_t is sampled randomly (in fact pseudo–randomly) according to the conditional probability distribution $P(y_t|x_1, x_2, ..., x_L, y_1, y_2, ..., y_{t-1})$. In this pure form, sampling would generate incoherent sentences as stated by Holtzman et al. (2020). To solve the issue Holtzman et al. (2020) proposed so called 'temperature' which according to an input hyperparameter lowers low probabilities and puts higher high probabilities.

Further improvement can be achieved by *Top-k* and *Top-p* sampling.

Top-k Sampling

Fan et al. (2018) introduced a simple sampling scheme — the Top-k, where during sampling at each time step, only k tokens with the highest conditional probability are considered. Afterward, the probability mass is redistributed among only those k tokens, and a token is randomly sampled from the new distribution.

Top-p Sampling

Fan et al. (2018) also proposed Top-p (or nucleus) sampling — instead of sampling only from the most likely k words, Top-p sampling takes a parameter p and selects the smallest subset of words (from the vocabulary) such that their cumulative probability exceeds the value p. Thus the number of words from which one is sampling may vary in each time step. Then, the probability mass is again redistributed among the selected tokens, and the next output token is sampled from the new distribution.

Top-p sampling can also be combined with Top-k.

2.5 Optimization Techniques

In this work, two optimization algorithms are used for training machine-learning models: Stochastic gradient descent (SGD) (Ruder, 2017) and Adam (Kingma and Ba, 2017) and its adjusted version AdamW (Loshchilov and Hutter, 2019). These algorithms are widely used in deep–learning (Soydaner, 2020). Note, that standard optimization methods like the gradient descent or the Newton method (Werner, 2018) would be too slow for deep–learning — where we typically need a large dataset and the data are of high dimensionality — gradient descent needs to compute the derivatives with respect to each feature for each data point. This is often not feasible. Using even more computationally costly algorithms, such as the Newton method or Levenberg-Marquardt method (Werner, 2018) will not help.

2.5.1 SGD

Instead of considering the whole training dataset at each time step (this is considered by the standard gradient descent method), SGD picks M data points $I = \{i_1, ..., i_M\}$ and than the gradient is estimated (exact gradient would have to be computed from the whole training dataset) as:

$$\tilde{g_t} = \frac{1}{M} \sum_{i \in I} \nabla l_i(\theta_t) \tag{2.8}$$

Where θ_t are parameters, which are afterward updated as in standard gradient descent:

$$\theta_{t+1} = \theta_t - \alpha_t \tilde{g_t} \tag{2.9}$$

α_t is the step size. The subset $\{(x_i, y_i) | i \in I\}$ is called a batch (Ruder, 2017). There can be more variants, e.g. one can do sampling with or without replacement.

2.5.2 Adam

Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. It uses exponential weighted averages (EWA (Perry, 2010)) of gradients per parameter across the previous time steps:

$$\theta_{t+1,i} = \theta_i - \epsilon \frac{EWA_{\beta_1}(\tilde{g_{1:t,i}})}{\sqrt{EWA_{\beta_2}(\tilde{g^2_{1:t,i}})}} \tag{2.10}$$

where $\tilde{g_{1:t,i}}$ denotes the sequence of all past gradients.

Hence, each coordinate is rescaled differently. (Thus Adam is called an adaptive learning method (Kingma and Ba, 2017) (Boris Flach)).

2.5.3 AdamW

The difference between Adam and AdamW is how they implement regularization: while Adam implements L2 regularization (thus, an additional term is added into the loss function), AdamW implements weight decay instead. Note that while for many optimizers, weight decay and L2 regularization are the same things, it is not so for adaptive learning optimizers, as shown by Loshchilov and Hutter (2019).

2.6 Metrics

To be able to evaluate the performance of particular machine-learning models automatically and to compare them with the current state of the art, this work uses these for automatic metrics:

1. BLEU
2. METEOR
3. CHRF++
4. TER

All the metrics except TER should hold that the better the score is, the better the trained model.

2.6.1 Nomenclature

Let **hypothesis** be the result generated (and decoded) by a trained seq-to-seq model, i.e., given a triple set, the generated text from this triple set is called a *hypothesis*.

Let **reference** be a human annotation for each input triple set. I.e., *references* are the targets, and triple sets are the source sequences in the training dataset.

BLEU

BLEU (Bilingual Evaluation Understudy) was proposed by Papineni et al. (2002) and is based on matching n-grams. The BLEU score is a real number between $0 - 1$ and holds that the higher score with respect to the reference, the better the inference is.

Given a hypothesis and N references (each reference belong to the one hypothesis — e.g., an RDF triple can be verbalized in different synonymous

ways), the metric is computed as follows:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C' \in \{Candidates\}} \sum_{n-gram' \in C'} Count_{clip}(n-gram')} \tag{2.11}$$

$$BP = \begin{cases} 1 & if \quad c > r \\ e^{(1-r/c)} & if \quad c \leq r \end{cases} \tag{2.12}$$

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n log(p_n)\right) \tag{2.13}$$

1. p_n is the modified n-gram precision on blocks of text.
2. c is the length of the hypothesis.
3. r is the effective reference corpus length.
4. BP is the brevity penalty.
5. w_n are positive weights summing to one.
6. N is a hyperparameter, determining, what n-grams should be used. $n = 0, 1, 2, ..., N$.

In this work, two implementations of the BLEU score are used. They differ mainly in tokenization, smoothing method, and in handling fringe cases. The two implementations are:

1. MOSES BLEU [1]
2. NLTK BLEU [2]

In the later chapters, MOSES BLEU is referred to as 'BLEU', and NLTK BLEU is referred to as 'BLEU NLTK'.

[1] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu-detok.perl

[2] https://www.nltk.org/_modules/nltk/translate/bleu_score.html

METEOR

Unlike BLEU, METEOR also considers semantic similarity, and instead of n-grams, it is based just on unigrams. The last version of the metric is proposed by Denkowski and Lavie (2014) — the white paper states that METEOR can deal with some downsides of BLEU score and have a higher correlation with human judgments.

Assume a hypothesis and n corresponding ground truth references. METEOR computes a score of how well the sample matches each reference and then considers only the maximum score.

The metric first creates a word alignment (word matching) between the two strings (hypothesis x reference). Every word in each string can have matched only with one corresponding word in the other string. The alignments are incrementally produced by several word-mapping modules:

1. Exact match (e.g., word 'computer' matches 'computer' in reference string).
2. Porter stem module: the words in both strings are stemmed by Porter algorithm (Jurafsky and Martin, 2009). The matching is then based on matching the stems of the words.
3. 'WN synonymy' - this module aligns synonymous words if both words belong to the same WordNet's 'synset' (Fellbaum, 1998).

From all possible matches, only the largest alignment is considered. If more than one maximal cardinality alignment is found, METEOR selects the alignment with the least amount of 'crossing' unigram mappings.

Given that

1. m — is the number of mapped unigrams.
2. t — is the total number of unigrams in the hypothesis.
3. r — is the total number of unigrams in the reference.
4. $P = \frac{m}{t}$ — is the unigram precision.

5. $R = \frac{m}{r}$ — is the unigram recall.

6. $F_{mean} = \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R}$ is the parametrized harmonic mean of P and R (van Rijsbergen, 1979).

7. ch — is the number of chunks. METEOR divides the sequence of matched unigrams into 'chunks'. A chunk is a sequence of matched unigrams, such that the unigrams in a chunk are adjacent in both (hypothesis and reference) strings and are in identical word order as in the two strings. METEOR divides the alignemnt into the fewest possible amount of chunks.

8. $frag = \frac{ch}{m}$ — is a fragmentation fraction.

9. $Pen = \gamma \cdot frag^{\beta}$ — is the penalty for different word order in the hypothesis vs reference. γ, β are hyperparameters.

The METEOR score is computed as

$$score = (1 - Pen) \cdot F_{mean} \tag{2.14}$$

TER

TER (Translation Edit Rate) was proposed by Snover et al. (2006). Unlike METEOR and BLEU, TER measures the quality of inference by the editing distance between reference and hypothesis. Hence, the lower the TER score is, the better.

According to Snover et al. (2006), "TER is defined as the minimum number of edits needed to change a hypothesis so that it exactly matches one of the references, normalized by the average length of the references." When there are more references available, TER is measured on each pair, and only the minimal score is considered.

$$TER = \frac{\#\ of\ edits}{average\ \#\ of\ reference\ words} \tag{2.15}$$

Algorithm 1 Calculate Number of Edits

input: HYPOTHESIS h
input: REFERENCES R
$E \leftarrow \infty$
for all $r \in R$ **do**
  $h' \leftarrow h$
  $e \leftarrow 0$
  repeat
    Find shift, s, that most reduces min-edit-distance(h', r)
    if s reduces edit distance **then**
      $h' \leftarrow$ apply s to h'
      $e \leftarrow e + 1$
    end if
  until No shifts that reduce edit distance remain
  $e \leftarrow e+$ min-edit-distance(h', r)
  if $e < E$ **then**
    $E \leftarrow e$
  end if
end for
return E

Figure 2.3: Algorithm proposed by Snover et al. (2006) to calculate the number of edits.

CHRF++

Popovic (2017) presents the general formula for n-gram based F-scores:

$$ngrF\beta = (1 + \beta^2)\frac{ngrP \cdot ngrR}{\beta^2 \cdot ngrp + ngrR} \tag{2.16}$$

where

1. $ngrP$: is n-gram precision — percentage of n-grams in the hypothesis text that has a counterpart in the given reference.
2. $ngrR$: is n-gram recall — percentage of n-grams in the reference which also have a counterpart in the hypothesis.

WORDF is calculated on word n-grams, and CHRF is calculated on character n-grams. The proposed CHRF++ score is obtained by combining word n-grams with character n-grams and averaged afterward. According to Popovic (2017) the best maximum n-gram for CHRF is $N = 6$, for WORDF it is $N = 2$ or $N = 1$. The optimal value for the β parameter is $\beta = 2$.

Chapter 3

Proposed LSTM-based Systems for RDF-to-Text Generation Problem

This thesis proposes two types of solutions — while both are based on the sequence-to-sequence framework, one type is based on LSTM recurrent neural networks, and the other group is based on transformers. As shown by Ferreira et al. (2019) training a transformer from scratch on a specific task would not result in any significant improvement with respect to LSTMs (for the RDF-to-Text task), therefore building a solution based on transformers requires transfer–learning, i.e., to use a suitable transformer model for this particular task and fine-tune it. The results published by Ferreira et al. (2019) suggest that incorporating a model into some pipeline instead of an end-to-end setting might lead to significantly better results (this holds for transformers as for LSTMs).

From the results of the WebNLG challenge 2017 (Gardent et al., 2017) and WebNLG challenge 2020 (Castro Ferreira et al., 2020) it seems clear that a fine-tuned transformer (which has appropriate architecture, e.g., MBART (Liu et al., 2020)) should outperform any LSTM seq-to-seq model, especially in out-of-domain data. This is so due to the pretraining phase, where typically a transformer is trained on some huge dataset (often in an unsupervised setting). Thus a fine-tuned transformer should have knowledge that goes beyond the dataset used for fine-tuning. This helps it to generalize better than a model trained from scratch.

However, transformers have two main disadvantages — the input's length cannot be greater than some threshold (typically 512 tokens (Wolf et al., 2020), while in the case of LSTMs, the length of input can be arbitrary). The

transformer models are huge to be able to cover the data from the pretraining phase, their training is feasible on sufficient amount of GPUs because training transformers (also as inferencing from them) can be highly parallelized. Thus the second disadvantage is that running transformers on CPUs is slow in general. Hence, while LSTMs cannot be too large (otherwise, their training would not be feasible) because their training nor their inference is highly parallelizable, counterintuitively, a typical LSTM-based seq-to-seq model has less hardware demands than a typical transformer and is usually faster, especially when running on CPU (because the parallelization advantage of transformers is marginalized when running on CPU).

The thesis examines how much time requires both LSTMs and transformers for inference in Chapter 7.

Despite the advantages of transformers, when one does not have access to GPUs and needs a quick inference, it might be better (in some scenarios) to use the LSTM seq-to-seq model instead of the transformer seq-to-seq model. The performance of the LSTM model should not differ 'a lot' on in-domain data, while it may decrease significantly on out-of-domain data. Hence, depending on the task where one wants to use the RDF-to-Text system and what hardware is available, the LSTM model could still be a good choice. The final application (outcome of the thesis) uses both types of models, and a user can, through API, select which model to use and consider the trade-off between performance and hardware requirements.

3.1 Seq-to-Seq LSTM Models

Each proposed model uses this basic pipeline:

1. Delexicalization
2. Text generation from triples
3. Relexicalization

where delexicalization can be viewed as data preprocessing step and relexicalization as data postprocessing.

This work also examines the impacts of adding other steps into the pipeline, such as triples ordering (in the dataset, the triples are in a set, thus are not ordered) or structuring (to determine what triples will be in what sentence, how many sentences should at least a model generate). Both steps are proposed by Ferreira et al. (2019).

3.1.1 Data Preprocessing and Postprocessing

Data preprocessing can be defined as a delexicalization task as proposed by the Melbourne solution of the WebNLG challenge 2017 (Distiawan) or by the baseline system of the first challenge (Gardent et al., 2017). This step aims to reduce the size of vocabulary and reduce the amount of out-of-vocabulary cases.

Since a trained model on a delexicalized dataset outputs delexicalized utterances, one must perform a reverse delexicalization step (hence relexicalization) to transform the output utterances to natural utterances.

3.1.2 Delexicalization

Assume the following triple set containing just one triple:

```
"triples": [
[
[
"Adolfo_Suarez_Madrid_Barajas_Airport",
"runwayName",
"14L/32R"
]
]
]
```

During preprocessing, each predicate is being tokenized (i.e., in this example, 'runwayName' would be split into 'runway name'), then each subject and object is associated with 'ENTITY-N' placeholder. For our case, this would be:

```
"Adolfo_Suarez_Madrid_Barajas_Airport": ENTITY-1
"runway name",
"14L/32R": ENTITY-2
```

Afterward each subject and object is associated with its type, for instance:

```
"Adolfo_Suarez_Madrid_Barajas_Airport": AIRPORT
"runway name",
"14L/32R": NOT_FOUND
```

The entity types are gathered from DBpedia (dbp). Dates and numbers are discovered by regular expressions instead of DBpedia; For unknown entity, the type is 'NOT_FOUND.'

Next, each part of each triple is marked by the function it has, i.e., three additional tokens are added to each triple:

1. <subject>
2. <predicate>
3. <object>

Finally, the triples in the triple set are concatenated and separated by '&&' symbol. The shown triple would be finally delexicalized as:

```
<subject> ENTITY-1 AIRPORT <predicate> runway name <object>
    ↪ ENTITY-2 NOT_FOUND
```

or an input triple

```
"triples": [
[
[
"Buzz_Aldrin",
"occupation",
"Fighter_pilot"
],
[
"Buzz_Aldrin",
"was a crew member of",
"Apollo_11"
]
]
```

would be delexicalized as

```
<subject> ENTITY-1 ASTRONAUT <predicate> occupation <object>
    ↪ ENTITY-2 PERSON && <subject> ENTITY-1 ASTRONAUT <
    ↪ predicate> was a crew member of <object> ENTITY-3
    ↪ ARTIFICIALSATELLITE
```

For the training purpose, 'targets' are delexicalized as well — the 'ENTITY-N' placeholder replaces the entities in targets, hence while the original target sentence for the given 2-triplet is

```
Buzz Aldrin was a fighter pilot and crew member of Apollo 11 .
```

The delexicalized version is:

```
ENTITY-1 was ENTITY-2 and crew member of ENTITY-3 .
```

To be able to replace entities in the targets, the enriched version of the WebNLG dataset is used, and hence all models from this chapter are trained on the first version of the WebNLG dataset (because the enriched version does not exist for later versions of the WebNLG dataset).

3.1.3 Relexicalization

The 'ENTITY-N' placeholders in the outputs are paired with corresponding subjects and objects from the input and replaced.

3.2 Ordering and Text Structuring

We propose a seq-to-seq model and a heuristic for triple ordering and for text structuring. We decided not to train any model for text structuring because when such model makes an error, this error would be propagated into the triples-to-text model, and the final output could be therefore significantly distorted. This is not the case with the ordering model, where it is easy to determine that the order is incomplete or has some other flaws and can be eventually rejected — and replaced by a random or default order.

For comparison, we trained triples-to-text models with and also without ordered data.

3.2.1 Ordering

We used two different approaches: a seq-to-seq system, where the input triples are linearized into a text and output is a text (a similar technique used Ferreira et al. (2019)), and another approach based on a custom heuristic.

Seq-to-Seq Ordering

The input consists of delexicalized tripleset with additional placeholder type 'PREDICATE-N', e.g.

```
<subject> ENTITY-1 AIRPORT <predicate> PREDICATE-1 runway name
    ↪ <object> ENTITY-2 NOT_FOUND
```

The output is a sequence of the placeholders, for instance:

```
PREDICATE-1 PREDICATE-3 PREDICATE-2
```

This output means, that the new order of some 3-tripleset would be

```
triple-1 triple-3 triple-2
```

The reason for the target consisting of 'PREDICATE-N' instead of 'TRIPLE-N' tags is that the decoder in the ordering seq-to-seq model can have a shared embedding layer with encoder, and thus the representation of 'PREDICATE-N' tags should be more meaningful. Moreover, a Pointer–Network can be used (Vinyals et al., 2017).

Heuristic for Ordering

The ideas behind the ordering heuristic are that:

1. If there are two triples t_1, t_2 and $t_1[object] == t_2[subject]$ then t_2 should be right after t_1, because then in the output, $t_2[subject]$ can be substituted by a pronoun, which should increase fluency of the generated output.
2. If there are triples $t_1, t_2, ..., t_j$ such that both have the same subject, they should follow each other, so again in the output, after the common subject is mentioned for the first time, it can be then referenced by a pronoun, so the output should look more natural.
3. The rule n.1 has higher priority than rule n.2.

The following pseudo-code can describe the algorithm:

```
let grouper be a function, that groups the triples \
       with same subject
let tripleset be a list of given triples
let ret_triples = list()
then:
tripleset = grouper(tripleset)
for i in range(tripleset.length){
       if i==0: {ret_triples.append(tripleset[i]);
              remove(triplesets[i]);continue;}
       candidates = all triples t from triplesets:
              st. t[object]==ret_triples[i][subject] ;
       ret_triples.append(candidates[0]);
       remove(tripleset[i]);
}
return ret_triples;
```

Ordering Seq-to-Seq Model Setting

Encoder consists of 2-layer bidirectional LSTM, has 128 hidden units and the embedding vector has dimensionality $d = 100$.
Dencoder consists of 2-layer LSTM, with 128 hidden units and the embedding is shared with encoder.

Both the encoder and decoder are using pretrained embeddings — GloVe (Pennington et al., 2014).

As the attention layer we use a multi-layer perceptron attetntion proposed by Bahdanau et al. (2016). Because in this setting, the trained model would have problems with sorting large triple sets (discovered empirically — in case of 7-triples, in 87.4% cases some 'PREDICATE-i' was missing in the output or some placeholders were repeated), additional attention mechanism is used - the coverage attention, which was proposed by Tu et al. (2016).

Coverage attention

Common problems in the seq-to-seq framework are the so-called under–translation and over–translation, where the first term refers to the problem

when a word should be translated, but the seq-to-seq system did not translate it. The latter term refers to when a word is translated multiple times, but it should not be so. Note that the above-described problem with repeating placeholders is over-translation.

Tu et al. (2016) suggested to use *coverage vector*, that is summarizing the history of attention of particular encoder's hidden state h_j till the current time step. Hence if we consider all encoder's hidden states, the coverage is represented by a matrix at each time step t. This matrix is passed as an additional argument into the standard attention's (Bahdanau et al., 2016) function.

Tu et al. (2016) proposed also to adjust the objective function by adding an additional coverage term which is scaled by a hyperparameter λ:

$$objective = loss - \lambda \cdot coverage \tag{3.1}$$

Tu et al. (2016) proposed to set the λ to 1.

3.2.2 Text Structuring

Text structuring aims at splitting the given triple set such that the final verbalized outcome will be fluent. Splitting the triple set into subsets can be beneficial, especially for larger triple sets, e.g., 7-triple sets. The heuristic is based on the following ideas:

1. A good split is the one, after which the subsets have similar sizes.
2. The number of splits should be minimal (not to harm fluency).
3. If there are two triples t_1, t_2 and $t_1[object] == t_2[subject]$ then t_2, then t_1, t_2 should remain in a same group.
4. If there are triples $t_1, t_2, ..., t_j$ such that both have the same subject, they may be split into subsets, but since the subsets are ordered — the first subset must contain at least triples t_1, t_2.
5. Triplesets below five items should not be split.
6. Triplesets of sizes 5, 6, and 7 should have only one split.
7. Triplesets with more than seven triples are not expected.

Pseudo-code:

```
# let partition be a split defining how the set is splitted
# let n-partition be a partition splitting a set into n parts
# let 'is_ok' be a function controlling, whether a partition
#  satisfies conditions 3. and 4. (mentioned above).
let splitter(triples, partition):
trans = list();
j = 0;
for p in partition{
ls = [];
for i in range(p){
ls.append(triples[i + j]);
}
j += (i + 1);
trans.append(ls);
}
return trans;

let tripleset be the set of input triples;
partitions = all possible 1-partitions;
sort partitions by abs(diff):\
where diff is the difference of subset's \
sizes coming from the partition;

for each partition{
if is_ok(partition){
return splitter(partition, tripleset);
}
}
```

3.3 Triples-to-Text Model

We tried to train five different models; three of them were trained on the objective of negative-loss likelihood, while the fourth and the fifth was trained in an adversarial setting with 'inverse Kullback-Leibler loss' proposed by Zhu et al. (2019).

The first three models differ in attention mechanisms: $model_1$ is using only Bahdanau et al. (2016) attention, $model_2$ is using Bahdanau et al. (2016) attention with the coverage term (Tu et al., 2016) and $model_3$ uses the same

coverage attention as $model_2$ but additionally uses copy attention (See et al., 2017). Other parameters are as follows (for all models):

1. Embedding size: 100 (Glove (Pennington et al., 2014) pretrained embeddings was used)
2. RNN type: Bidirectional LSTM
3. Number of hidden units: 128
4. Dropout: 0.5
5. Number of layers: 4
6. Lambda coverage ($model_2, model_3$): 1
7. Optimizer: SGD for models 1,2,3 (as suggested by the baseline model of WebNLG challenge (Gardent et al., 2017)); AdamW for model trained by inverse KL divergence (as suggested by Zhu et al. (2019))
8. Batch size: 64
9. Decoding strategy: beam search (we empirically found out, that beam search outperforms sampling)

The hyperparameters of the two models trained within the 'Inverse Kullback–Leibler Divergence' framework are the same as presented in the white paper (Zhu et al., 2019). The last two models differ only in copy attention — one is using it, and one is not.

3.3.1 Copy Attention

Copy attention is a possible way of dealing with out-of-vocabulary words — it allows the model to use the words from the input at the output, even if the source words are not part of the model's vocabulary. This can help the model to deal with out-of-domain samples.

Let $P_{vocab}(w)$ is the vocabulary distribution at time step t (conditional probability distribution dependent on the input, last output and attention), let a^t be the attention distribution as described by Bahdanau et al. (2016), let the extended vocabulary be the union of the vocabulary and all words appearing in the source document (input), let p_{gen} be defined as follows:

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{tr}) \tag{3.2}$$

$w_{h^*}, w_s, w_x, b_{ptr}$ are learnable parameters, all are vectors except the scalar b_{ptr}, s_t is the decoder's hidden state, h_t^* is the context vector computed as weighted sum of encoder's hidden states, where the weights comes from the softmaxed attention (Bahdanau et al., 2016). x_t is the input token at time step t. Then the distribution over the extended vocabulary is defined as:

$$P(w) = p_{gen}P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t \tag{3.3}$$

Hence the copy attention works as a switch determining whether to consider only the words from the vocabulary or from the source (i.e., from the input sequence) (See et al., 2017).

3.3.2 Inverse Kullback–Leibler Loss

Zhu et al. (2019) claims that instead of considering minimizing the Kullback-Leibler (KL) divergence between the target distribution P and the learned distribution G, it is beneficial to take the inverse KL divergence. I.e., instead of minimizing:

$$KL(P||G) = \mathbb{E}_{Y\sim P}\left[log\frac{P(Y)}{G(Y)}\right] \tag{3.4}$$

Zhu et al. (2019) suggest to minimize

$$KL(G||P) = \mathbb{E}_{Y\sim G}\left[log\frac{G(Y)}{P(Y)}\right] \tag{3.5}$$

for the following reasons:

1. If $P(y) > 0$ and $G(Y) \to 0$, then $KL(P||G)$ goes to infinity, thus for very rare patterns, if the model fails to cover them, the penalty is extremely high and thus sensitive for outliers.
2. If $P(y) \to 0$ and $G(Y) > 0$ the penalty will be low; however, this case means that the trained model may assign a quiet high probability to very unlikely samples (hence may assign a high probability to some nonsense and would be 'punished' only little during training).

The inverse KL loss does the opposite with respect to the mentioned two points above.

While normally, instead of minimizing the KL divergence, one minimizes the negative log-likelihood or the cross-entropy (both are equivalent). Hence, one does not need to know the true distribution in this setting of inverse KL loss; the true distribution (or its approximation) is needed. Zhu et al. (2019) proposed adversarial algorithm, where the true distribution is approximated by a model trained on negative log-likelihood:

Algorithm 1: Triple-to-Text algorithm

Input: a corpus of knowledge bases and its corresponding natural sentences $\{(\mathcal{F}, S)\}$, hyper-parameters m and g
Output: a generator G_θ, a judger M_ϕ

Pre-process the knowledge bases corpus $\{(\mathcal{F}, S)\}$ into discrete token sequence pairs $\{(X, Y)\}$
Initialize G_θ and M_ϕ with random parameters θ and ϕ
Pre-train G_θ using Maximum Likelihood Estimation (optional)
while G_θ *not converge* **do**
  for m *steps* **do**
    Sample from sequence pairs $\{(X, Y)\}$
    Update judger M_ϕ via maximizing $\mathbb{E}_{(X,Y)\sim P}[\log M_\phi(Y|X)]$
  end
  for g *steps* **do**
    Sample conditional context $\hat{X}$ from pairs $\{(X, Y)\}$
    Generate the estimated target sentence $\hat{Y}$ given $\hat{X}$ according to G_θ
    Update generator G_θ via minimizing $\mathbb{E}_{(\hat{X},\hat{Y})\sim G_\theta}\left[\log \frac{G_\theta(\hat{Y}|\hat{X})}{M_\phi(\hat{Y}|\hat{X})}\right]$ by Eq. 15
  end
end
return G_θ, M_ϕ

Figure 3.1: Adversarial algorithm proposed by Zhu et al. (2019).

The hyperparameters m, n are both set to 1.

3.4 Implementation Details

The implementation consists mainly of Python scripts for data preprocessing, model builders, trainers, data loaders, and a pipeline that performs ordering, text structuring, and text generation.

For more details, one can see the 'README.md' in the attached CD.

The repository is organized into several folders and packages (the following list is describing the whole structure of the repository — including Python packages used only for the transformer-based system described in Chapter 4):

1. **utils:** data preprocessing and postprocessing. Tools to help manipulation with data.
2. **onmt_based:** Python package for training seq-toseq LSTM models (compatible with OpenNMT versions 1.x and 2.x).
3. **onmt_based_2:** Python package for training seq-toseq LSTM models (compatible with OpenNMT versions 2.x only).
4. **webnlg3:** Contains Python scripts for extracting data from the third WebNLG dataset and for data preprocessing.
5. **train_t5:** Python package for training and using the Google T5 transformer (Raffel et al., 2020).
6. **ranker:** To train Roberta ranler (see 4.5).
7. **root directory:** Despite Readme, requirrements.txt etc. there are Python files implementing pipeline for transformer part, LSTM part and a pipeline using both of them. All LSTM models should have a record of their specification in the *specs_of_models* file.

All code is compatible with the Python version of 3.6.

3.4.1 Data Preprocessing and Postprocessing

The Python package called 'utils' contains all necessary scripts for data preprocessing and postprocessing. The main tasks needed to do are described below:

1. **Data extraction:** The WebNLG dataset is provided in XML format (for each category and triple set size, there is a special XML file), Gardent et al. (2017) also provides Python utilities to convert the dataset into a JSON. One must extract the data source and target files (source files are containing linearized triple sets while target files contain the corresponding verbalizations).
2. **Data delexicalization:** The objects and subjects are replaced by entity placeholders, which are taken from DBpedia.
3. **Data relexicalization:** This is a necessary postprocessing step — the 'ENTITY-N' placeholders are replaced back to the original entities.

The 'utils' package provides the following functionalities:

1. Load data from a file into a list and Write data (list) to a file [inputters.py]
2. Data preprocessing (loading a given json file and output delexicalized triples) [data_preprocessing.py]
3. Delexicalization of input triplesets [delex_src.py]
4. Create references for measuring BLEU, METEOR, TER and CHRF++ [create_references.py]
5. Find types of entities from DBpedia [utils.py]
6. Relexicalization [relexicalize.py]
7. Restructuring - if a structured input is given to a model, the output must be restructured [restruct.py]
8. Heuristic ordering and structuring [struct_for_lstm.py]
9. Create targets for training and validation [create_tgts.py]

The preprocessed data are stored in *data_melbourne* folder, where are other preprocessing scripts, e.g., for data shuffling and clearing.

3.4.2 LSTM Models

All models are implemented as PyTorch models (Paszke et al., 2019) and all models are compatible with the PyTorch version of the OpenNMT library (Klein et al., 2017).

Because of the adversarial training of one of the models, we implemented new model builders, the inverse KL loss, the inverse KL loss for the model using copy attention, data loaders, and adjusted training method (new training class inherits from OpenNMT's training class). However, the core models (i.e., the encoder-decoder architecture and attention mechanisms) are imported from the OpenNMT library. In the attached code, the implementation of the adversarial training and data loaders, model builders, and translator can be found in the 'onmt_based' directory. The system is compatible with OpenNMT version 1.x and 2.x — there are some duplicate classes, each compatible either with 1.x or 2.x version (there are some big changes between these two versions). Concretely, the package 'onmt_based' implements the following functionality:

1. Data loader and Data iterator.
2. Trainer for training the generator for the adversarial mode; Trainer for training classical seq-to-seq model. Both trainers are compatible with OpenNMT 1.x and 2.x versions. They take the version of the library as a parameter.
3. Model builder — to build a seq-to-seq LSTM model.
4. The inverse KL loss and the modified inverse KL loss for copy attention.
5. Translator.
6. Script for training a seq-to-seq model.
7. Script for training a model by the inverse KL loss .

The 'onmt_based' package can also be used to train a seq-to-seq model with other losses than the inverse KL loss. Also, note that when one is training a model on inverse KL loss, an additional seq-to-seq model is being trained simultaneously on the negative log-likelihood loss. That model is also being stored (by default, it is named 'judger' as proposed by Zhu et al. (2019)).

Chapter 4

Proposed Transformer-Based Systems for RDF-to-Text Generation Problem

As already stated in the Chapter 2 and 3, the transformer-based solution should be (thanks to the transfer learning method) more robust. Transformers should have better performance, especially on unseen data (i.e., should generalize better) than the solution based on Recurrent neural networks trained from scratch.

The disadvantage is that the pretrained transformers are 'heavy' and slow on CPUs (see 7).

This chapter proposes to use the pretrained transformer 'Google T5' Raffel et al. (2020) for converting triple sets into natural texts. Additionally, our T5 transformer retrieves more outputs per each input; a 'ranker' (Roberta) then ranks the outputs, and if a retrieved sample is ranked as 'ok', that sample is finally retrieved by the system. If there are more 'ok' retrieved samples, the one with the highest probability is considered. Otherwise (no sample was classified as 'ok'), the sample with the highest probability of being 'ok' is selected. The ranker is the 'Roberta' transformer (Liu et al., 2019) and is inspired by Harkous et al. (2020b).

4.1 Google T5

T5 model aims to solve text-to-text tasks; hence, the input of the model is a string, and the output is a string. While the T5 model is aimed at text-to-text tasks, it can also be applied for classification tasks, etc. — but the output is always a text. Moreover, the model can be trained on various data such that a single model can perform various tasks, as shown in the following picture:

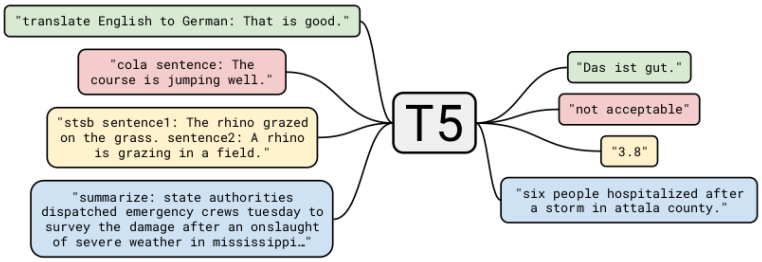


Figure 4.1: A diagram of the text-to-text framework proposed by Raffel et al. (2020).

The model uses standard encoder-decoder Transformer architecture as described in chapter 2. Both the encoder and decoder consist of 12 blocks (each consisting of self-attention, optionally with encoder-decoder attention, and feed-forward network). As an embedding layer, T5 uses Sentencepiece (Kudo and Richardson, 2018). The T5-base model has about 220 million parameters. Raffel et al. (2020) proposed other variants of the T5 model, which varies by the number of parameters:

1. T5-small: 60 million paramters
2. T5-large: 770 million parameters
3. T5-3B: 2.8 billion paramters
4. T5-11B: 11 billion parameters

This work explores T5-small, T5-base, and T5-large only since the two largest models require too many resources for fine-tuning as well as for inference.

4.2 Roberta

RoBERTa is a shortcut for 'A Robustly Optimized BERT pretraining Approach' and was proposed by Liu et al. (2019). The model is basically a standard BERT Transformer (Devlin et al., 2018), but trained in a bit different manner than the original BERT model, namely:

1. The model is trained longer (more epochs), with bigger batches and over more data.
2. The 'next sentence prediction objective' (Devlin et al., 2018) is removed.
3. The sequences in datasets are longer in average.
4. Dynamically changing the masking pattern (Devlin et al., 2018), (Liu et al., 2019).

Since the Roberta is pretrained more robustly than the original BERT, it should generalize better not just before fine-tuning but also after fine-tuning.

4.2.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) was introduced by Devlin et al. (2018). BERT also uses the Transformer architecture described in the chapter 2, but unlike T5, it uses only an encoder and requires fewer parameters. There are several variants of the model varying by their size. This work uses only ROBERTA-base (equivalent for BERT-base), which has ca. 110 million parameters and ROBERTA-large with ca. 336 million parameters.

BERT uses WordPiece embeddings and can decode a single sequence of tokens or a pair of sequences. The two sequences are distinguished by the $[SEP]$ special token and also by the so-called segment embeddings (an embedding layer for distinguishing sequenceA from sequenceB). Additionally, for each input, a special token ($[CLS]$) is prepended. The output of the last hidden layer corresponding to the $[CLS]$ token comprises the sentence embedding of the input. Thus, the token's hidden state can be fed to other additional layers (e.g., dense layer) for classification, etc. (we use it for the

Roberta ranker, where the hidden state is fed into a single dense layer, and the output of the ranker is softmaxed output of the dense layer).

Devlin et al. (2018) suggests two approaches how to pretrain the BERT model: *Masked LM* and *Next Sentence Prediction*. Since Roberta uses only the first one, this chapter describes *Masked LM* only:

4.2.2 Masked LM

To train a bidirectional representation of a language model instead of one-directional, Devlin et al. (2018) propose to mask some percentage of the input tokens at random (replacing them by either $[MASK]$ token or a random token) and then predict the masked tokens. The loss during the training is computed w.r.t. the prediction.

4.3 Data Preprocessing

The Transformer-based system requires a little data preprocessing and does not require any data postprocessing.

Given a triple set, the input must be preprocessed as follows:

1. **Linearization:** converting the triple set into a sequence of tokens.
2. **Tokenization:** Each subject and object is in snake case form in the WebNLG data. We split each such 'word' into single words. Predicates are in camel case form (in WebNLG), and they are split into single words as well.
3. **Adding special tokens:** Triples are concatenated by '&&' token. Moreover, each subject is prepended by '<*subject*>' token, and similarly, each object and predicate is prepended by '<*object*>' and '<*predicate*>' respectively.

Thus given an input:

```
"triples": [
[
[
"Adolfo_Suarez_Madrid-Barajas_Airport",
"runwayName",
"14L/32R"
]
]
]
```

the preprocessed sample would look as follows:

```
<subject> Adolfo Suarez Madrid-Barajas Airport <predicate>
    ↪ runway name <object> 14L/32R
```

Note that the same preprocessing is done for the triples-to-seq model (T5) as for the ranker (Roberta).

4.4 T5 for Triples-to-Text Setup

The models are trained on the preprocessed third version of the WebNLG dataset (i.e., the dataset used for WebNLG challenge 2020).

Each T5 model is trained with these hyperparameters:

1. **Optimizer:** AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $lr = 2e - 4$)
2. **Early stopping:** *patience* = 10, evaluation metric on validation set is BLEU
3. **Batch size:** 16 for T5-small and T5-base, 8 for T5-large
4. **Beam size:** 6
5. **Maximum number of epochs:** 100
6. **Learning rate decay:** Linear learning rate decay with $\gamma = 0.9$

7. **Decoding strategy:** beam search (we empirically found out, that beam search outperforms sampling)

The additional tokens (*<subject>*, *<predicate>*, *<object>*, *&&*) are added into the vocabulary and their embeddings are learned during fine-tuning.

4.5 Roberta for Ranker Setup

For training the ranker, the third version of the WebNLG dataset is used as well. However, the data are transformed to fit in the classification task of the ranker. Particularly, the model takes two sequences in the input:

1. A linearize preprocessed triplet
2. Target sequence

The output of the model is one of the following classes (as proposed by Harkous et al. (2020b)):

1. Accurate
2. Omission
3. Repetition
4. Hallucination
5. Value errors

The WebNLG dataset is transformed into the dataset for ranker as follows (note that in the input there are two sequences: linearized preprocessed triple set and target, the output is expected to be a class; the linearized preprocessed triple sets are unchanged in the dataset for ranker; also note, that for all other classes except 'Accurate', it is expected that the triple set in the input has a size larger than one):

1. **Accurate:** The target in the input pair is the target from the WebNLG dataset.
2. **Omission:** The shortest sentence in the target is removed.
3. **Repetition:** Select a random sentence from the target and insert it before another random sentence in the target.
4. **Hallucination:** Select a random sentence from another sample and insert it before a random sentence in the target.
5. **Value errors:** Select a random entity in the triple set and replace that entity in the target with a random entity from the triple set.
6. **Omission+Hallucination:** This case is treated as Hallucination class.

The scheme of the ranker model can be visualized as follows:

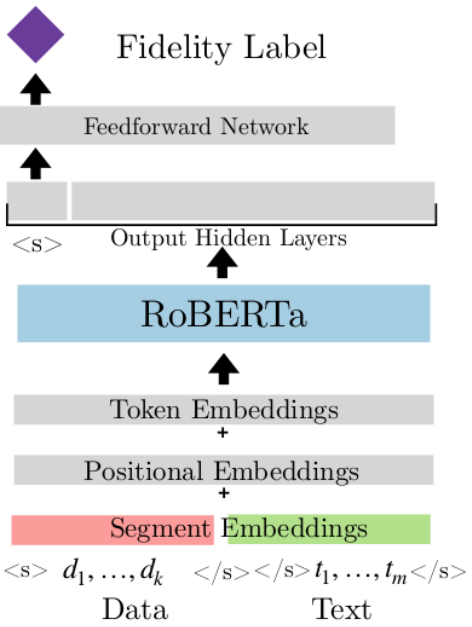


Figure 4.2: A diagram of the ranker by Harkous et al. (2020b).

The model is trained with the following parameters:

1. **Optimizer:** AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $lr = 2e - 5$)
2. **Early stopping:** *patience* $= 10$, evaluation metric on validation set is the negative lielihood loss

3. **Batch size:** 32
4. **Maximum number of epochs:** 25
5. **Learning rate decay:** Linear learning rate decay with $\gamma = 9$

Similarly to the T5 models, the additional tokens are added into the model's vocabulary, and their embeddings are learned during fine-tuning.

4.6 Implementation Details

Both (pre-trained) models (T5 and Roberta) are obtained from Huggingface Transformers [1] (Wolf et al., 2020) and both models are Pytorch (Paszke et al., 2019) models.

4.6.1 T5 Model

The training function uses a dataset class, which is copied from [2] and the fine-tuning step is done by a custom training function which is inspired by [3]. The model is an instance of the class 'transformers.T5ForConditionalGeneration', which is the standard T5 model with an additional *language model head* on top. The language model head is represented by a single dense layer.

Additionally, a script for inference is provided. All training scripts regarding this model can be found in the 'train_t5/' Python package, the scripts for inference can be found in the root directory.

4.6.2 Roberta Model

The Roberta model is an instance of the 'transformers.RobertaForSequenceClassification' class, and the implementation of fine-tuning as well as of inference can be found in the 'ranker' Python package. All implementation is custom.

[1] https://huggingface.co/transformers/index.html
[2] https://github.com/znculee/finetune-transformers
[3] https://github.com/znculee/finetune-transformers

Chapter 5

Results of LSTM Models

The computation of BLEU, TER, METEOR, and CHRF++ is done by a package provided within the WebNLG challenge [1].Because the LSTM models are trained on the enriched WebNLG dataset, we should compare them to the participants' systems from the WebNLG challenge 2017.

The table below is showing the results of the WebNLG challenge 2017 (except baseline):

model	BLEU	METEOR	chrF++	TER
Melbourne	45.13	0.37	0.625	0.47
TILB-SMT	44.28	0.38	0.647	0.53
PKUWRITER	39.88	0.31	0.5452	0.55
UPF-FORGE	38.65	0.39	0.645	0.55
TILB-NMT	34.60	0.34	0.586	0.6
TILB-PIPELINE	44.34	0.38	0.646	0.48
ADAPT	31.06	0.31	0.560	0.84
UIT-VNU	7.07	0.09	0.184	0.82

Table 5.1: Results of BLEU, METEOR, and TER are the official results of the WebNLG challenge 2017. The CHRF++ metric is measured on the output of the test set that is provided for each of the systems on the website of the challenge.

[1] https://github.com/WebNLG/GenerationEval

The next figure depicts the performance of LSTM models proposed in this work:

model	BLEU	BLEU NLTK	METEOR	chrF++	TER
$model_1$	39.78	39.78	0.378	0.620	0.517
$model_2$	40.21	40.23	0.369	0.601	0.502
$model_3$	43.11	43.11	0.390	0.642	0.486
ikl	41.5	41.5	0.385	0.633	0.505
ikl_{cpa}	42.19	42.19	0.396	0.648	0.502

Table 5.2: Results of the proposed models on the delexicalized dataset. $Model_1$ does not contain copy attention nor coverage attention, $model_2$ contains coverage attention and $model_3$ contains both coverage and copy attention. The ikl model is the small model trained by the inverse Kullback-Leibler divergence loss in adversarial mode, and ikl_{cpa} is the same model with additional copy attention.

5.1 Heuristic Ordering and Text Structuring

To examine the effect of ordering and text structuring, we took the architectures of $model_3$ and ikl, and trained the two models on ordered data. For comparison, we also used the same model but trained on unordered data and measured its performance on structured test data. The performance of the latter model is shown in the table below:

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
$model_3$	42.74	42.76	0.394	0.648	0.491
ikl_{cpa}	42.13	42.13	0.40	0.653	0.513

Table 5.3: The two best performing models trained on unordered data. The results are improved on METEOR and chf++ metrics while BLEU and TER are worse. A possible explanation is that both reordering and text structuring would break some n-grams in the output, which could harm all metrics, especially TER and BLEU. The text structuring should help to prevent information omission, however necessarily the generated sentences would have fewer references to the given entities by pronouns, which should not have a significant impact on METEOR and chrf++, but should impact BLEU and TER negatively.

The next table shows the performance of the same models trained on ordered data. The inference is made on ordered and structured data:

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
$model_3$	42.81	42.81	0.393	0.646	0.489
ikl_{cpa}	42.15	42.15	0.399	0.651	0.510

Table 5.4: The two best performing models trained on ordered triple sets. The table shows that the heuristic ordering does not lead to better performance.

5.2 Seq-to-seq Ordering

Seq-to-seq ordering model presented by Ferreira et al. (2019) does not have very good results, since (as Ferreira et al. (2019) presents in their paper) the model is not significantly better than just random ordering.

Unfortunately, the seq-to-seq model proposed in this work does not work well either — below are indicators, implying the model did not learn the task (measurements were done on the test set):

1. When there are n triples in the input, the output should have n tokens (each token corresponds to one particular triple as described in section 3.2.1), i.e., after n tokens, the model should generate the end-of-sentence token. However, in 43.26% cases, it does not do so.

2. On validation data, the model has perplexity of 29.9; on test data, the perplexity is 61.48. Note that $perplexity = 2^{crossentropy}$, hence the cross-entropy of the model is very high. For comparison, the models used for text generation from triples have perplexity on validation data around 4.7 and on test data around 11.3. (Both values are medians across $model_1$, $model_2$ and $model_3$).

3. The model frequently outputs tokens corresponding to $i-th$ triples, even the input triple set contains only n triples, where $n < i$. This happens in 38.4% cases.

Based on these results, we conclude that the model is not very good for the ordering task and is further considered to be a 'blind alley' — hence the model is not used by the triples-to-seq system.

5.3 Unseen test data

The results of the WebNLG 2017 challenge on unseen test data are shown in the table below:

model	BLEU	METEOR	TER
Melbourne	33.27	0.33	0.55
TILB-SMT	29.88	0.33	0.61
PKUWRITER	25.36	0.24	0.67
UPF-FORGE	35.70	0.37	0.55
TILB-NMT	25.12	0.31	0.72
TILB-PIPELINE	20.65	0.21	0.65
ADAPT	10.53	0.19	1.4
UIT-VNU	0.11	0.03	0.87

Table 5.5: Official results of the WebNLG challenge 2017 — unseen part of the test dataset.

While the results of the two best performing models proposed in this thesis are:

model	BLEU	METEOR	TER
$model_3$	33.86	0.37	0.56
ikl_{cpa}	34.11	0.37	0.55

Table 5.6: Results of the two best models (with text structuring) on the unseen part of the test set.

5.4 Discussion

Let's compare the results of the *ikl_cpa* model (with text structuring) with the systems from the challenge. One can observe that the *ikl_cpa* outperforms all solutions (measured on the complete test set) on METEOR and CHRF++ metrics, while it is fourth on BLEU score and third on TER. Considering the unseen data only, *ikl_cpa* model performs equally to UPF-FORGE system on METEOR and TER metrics, while it would be second (after UPF-FORGE) on the BLEU score.

Since UPF-FORGE is the winner of the challenge (outcome of human evaluation) and *ikl_cpa* outperforms UPF-FORGE on 3 out of 4 metrics on

the whole test dataset and performs equally on two out of three metrics on the unseen data (except BLEU, where it is worse — but with a relatively low difference), we would conclude that *ikl_cpa* should have similar quality as UPF-FORGE. Note that the maximum number of references of the whole test dataset is 8, while for the unseen part, it is only 5, which also affects the results. It is hard to say whether UPF-FORGE scores are affected by this comparably, more or lower than *ikl_cpa*.

UPF-FORGE is, however, a rule-based system, while all other competitors in the challenge used some kind of machine-learning models (systems). Hence one can also conclude that from the ML systems, *ikl_cpa* has the best performance (even it is not so good on BLEU measured on all data — because very high BLEU on seen data and abysmal results on unseen data indicates bad generalization of the model) and that it can generally be hard to outperform manually tuned rule-based systems by a machine learning models.

Chapter 6

Results of Transformer Models

Since the transformer models are trained on the latest version of the WebNLG dataset, their performance should be compared to the results of the last WebNLG challenge. The following table presents results (automatic metrics) of the first ten competitors (sorted by METEOR, measured on the whole test dataset):

system id	BLEU	BLEU NLTK	METEOR	CHRF++	TER
id18	53.98	53.5	0.417	0.690	0.406
id30	53.54	53.2	0.414	0.688	0.416
id30_1	52.07	51.8	0.413	0.685	0.444
id34*	52.67	52.3	0.413	0.686	0.423
id5	51.74	51.7	0.411	0.679	0.435
id35*	51.59	51.2	0.409	0.681	0.431
id23	51.74	51.4	0.403	0.669	0.417
id2	50.34	50.0	0.398	0.666	0.435
id15	40.73	40.5	0.393	0.646	0.511
id28	44.56	43.2	0.387	0.637	0.479
id12	40.29	39.3	0.386	0.634	0.504

Table 6.1: Eleven best systems of the WebNLG 2020 (all data, sorted by METEOR). Full results are available at https://beng.dice-research.org/gerbil/webnlg2020results. * refers to late submissions.

We trained three T5 models, their perofrmance is shown below:

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-small	46.32	46.21	0.399	0.657	0.476
T5-base	53.60	53.24	0.411	0.684	0.428
T5-large	53.70	53.46	0.417	0.686	0.420

Table 6.2: T5 end-to-end models trained on WebNLG 3.0 data set.

Since T5-small has significantly lower performance than T5-base and T5-large, we did not examine this architecture any further. The following two tables show results of the T5-base and T5-large models together with the Roberta-base and Roberta-large rankers:

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	53.79	53.42	0.412	0.685	0.412
T5-large	53.79	53.53	0.417	0.692	0.411

Table 6.3: T5 + Roberta-base ranker

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	53.79	53.43	0.413	0.686	0.412
T5-large	53.81	53.54	0.418	0.692	0.410

Table 6.4: T5 + Roberta-large ranker

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	53.74	53.41	0.414	0.687	0.413
T5-large	53.75	53.49	0.419	0.693	0.410

Table 6.5: T5 + Roberta-large ranker + text structuring

6.1 Unseen data

The table below shows ten best (according to METEOR) systems of the WebNLG challenge 2020 on unseen categories — a subset of the full test set:

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
id18	49.15	49.1	0.404	0.660	0.413
id30_1	45.34	45.5	0.398	0.651	0.471
id30	47.4	47.4	0.397	0.652	0.437
id34*	46.2	46.3	0.394	0.647	0.444
id5	43.98	44.1	0.393	0.636	0.470
id23	45.57	45.4	0.388	0.632	0.438
id35*	43.84	44.0	0.387	0.637	0.458
id15	35.85	36.4	0.384	0.617	0.512
id28	40.87	40.5	0.379	0.615	0.486
id4	43.82	43.6	0.379	0.618	0.472

Table 6.6: Official results of the WebNLG challenge 2020 — unseen part of the test dataset. * refers to late submissions.

Below, the two tables show the performance of T5-base and T5-large with Roberta-large ranker without and also with text structuring.

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	45.98	46.07	0.392	0.646	0.431
T5-large	48.12	48.36	0.403	0.656	0.421

Table 6.7: T5 + Roberta-large ranker

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	45.88	45.95	0.396	0.650	0.432
T5-large	48.04	48.29	0.405	0.661	0.422

Table 6.8: T5 + Roberta-large ranker + text structuring

6.2 Evaluation of Roberta Ranker

The performance of the trained Roberta rankers is measured by the F1-score:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} \tag{6.1}$$

model	F1
Roberta-base	0.92
Roberta-large	0.95

Table 6.9: Performance of Roberta classifiers (rankers).

6.3 Discussion

While expectably T5-large has better performance than T5-base, the difference is relatively low, and hence the API 10 allows to choose between the two models. Note that T5-large can be significantly slower as shown in Chapter 7.

To compare the WebNLG challenge 2020 results with the results in this thesis, it is fair to consider the T5-large only, since the 'id18' model (the best performing model) is also based on T5-large (Guo et al., 2020).

The T5-large+Roberta-large combination outperforms all models on the whole test dataset on BLEU NLTK, METEOR, and CHRF++ metrics, while it is second on TER and BLEU metrics. Note that the difference is very small — on the BLEU score, one implementation maps a higher score to the 'id18' system while the other to the system proposed in this thesis. Similar results are achieved with an additional step — the text structuring (but the differences are slightly higher).

On unseen data, if T5-large+Roberta-large + structuring is considered, then the system would be the best on METEOR and CHRF++, while it would be second on both BLEU scores implementations and TER.

Due to the fact that the overall differences between the scores of the system proposed in this thesis and the best performing 'id18' system are very low, we would suggest that the systems are +- equally good.

Note: the 'id18' system was proposed and is described by Guo et al. (2020).

Chapter 7

Speed Comparison

The Chapters 2 and 3 concluded that there should be a significant difference in computational demands of the two different architectures: the LSTM and the Transformer.

We measured the speed on both — CPU and GPU:

1. **CPU:** Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
2. **GPU:** Nvidia GeForce GTX 1080 Ti

The next table shows the results and is confirming the claims from Chapters 2 and 3. The reported time costs are averages made on WebNLG 1.4 test data set, and they tell us how much time does the model on average requires to transform a single triple set into a text (i.e., the batch size was set up to one, during the experiment).

Model	GPU	CPU
LSTM	121ms	192ms
Delexicalization	-	129ms
Relexicalization	-	$<$ 1ms
T5-base	1022ms	1414ms
T5-large	2003ms	3468ms
Ranker Roberta-base	81ms	265ms
Ranker Roberta-large	163ms	950ms

Table 7.1: Comparison of the speed of different systems. If LSTM is used, the total average time should be computed as the average time for the LSTM model plus the average time for delexicalization and relexicalization. For the transformer-based solution, one should add the average time of the ranker only if one intends to use it. Clearly, T5 models are much slower than the LSTM model, although if the data are feed-in batches, the difference would be lower for a run on GPU.

Chapter 8

Comparison of Performance by Automatic Evaluation metrics

This chapter aims to compare the ikl_{cpa} model with the T5-base and T5-large models. For the best objectivity, the performance is measured on the unseen part of the last release of the WebNLG test dataset (so it is out-of-domain for both models) and on the seen part of the WebNLG 1.4 (in-domain for both models) test dataset. Moreover, for better evaluation of the ikl_{cpa} model, the results include the performance of the UPF-FORGE system from WebNLG challenge 2017 (this system was used as a baseline system in the last challenge).

This comparison incorporates only the ikl_{cpa} model and T5-base and T5-large models. Hence, no ordering, text structuring, nor ranker is used.

8.0.1 WebNLG 3.0 Unseen Test Data

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	45.89	45.91	0.391	0.639	0.435
T5-large	48.03	48.27	0.402	0.648	0.426
ikl_{cpa}	23.23	23.23	0.346	0.563	0.682
UPF-FORGE	34.63	34.3	0.347	0.565	0.544

Table 8.1: T5 vs LSTM. The table shows, that both T5 models generalize significantly better, than the best LSTM model. Scores for UPF-FORGE are taken from Castro Ferreira et al. (2020)

8.0.2 WebNLG 1.4 Seen Test Data

model	BLEU	BLEU NLTK	METEOR	chrf++	TER
T5-base	67.11	67.11	0.468	0.775	0.291
T5-large	68.27	68.27	0.474	0.783	0.284
ikl_{cpa}	48.47	58.47	0.425	0.700	0.454
UPF-FORGE	40.88	-	0.40	-	0.55

Table 8.2: T5 vs LSTM. The table shows, that both T5 models perform significantly better, than the best LSTM model. Scores for UPF-FORGE are taken from Gardent et al. (2017)

8.1 Discussion

On both datasets: out-of-domain and in-domain, the transformers significantly outperformed the LSTM model. However, on the in-domain data, the scores are higher than the scores for transformers measured on the whole test set. However, the scores on in-domain data suggest that ikl_{cpa} might perform sufficiently on these data since the values of the scores are better than are scores of transformers measured on the whole test set.

If one focuses on the WebNLG 3.0 unseen test data, the performance of the ikl_{cpa} may seem very low; however, it has still better or comparable performance than some competitor's (participants of the challenge) systems that use transformers Castro Ferreira et al. (2020). Comparing to UPF-FORGE, ikl_{cpa} has similar performance on METEOR and CHRF++, but significantly worse results on BLEU and TER, which may suggest that the generated sentences are semantically more-less correct, but they may be disfluent or not grammatically correct.

The next chapter provides the same comparison but uses human evaluation.

Chapter 9

Human Evaluation

For the human evaluation, only the best performing models are taken, i.e., the ikl_{cpa} + text structuring (even the text structuring has a negative impact on TER and BLEU, we consider METEOR and CHRF++ more relevant metrics because especially METEOR correlates better with the results of human evaluation of the WebNLG 2017 and 2020 challenge) and the T5-large with Roberta-large ranker. (The performance of the ikl_{cpa} could be further improved by using the ranker; however, the idea is to compare a non-transformer-based system to a transformer-based system).

To make a fair comparison between the Transformer-based system and the LSTM-based system, 100 random samples were sampled from the unseen part of the third version of the WebNLG dataset (so it is ensured that none of the models have seen the entities in the given data during training/fine-tuning) and also 100 random samples were sampled from the seen part of the WebNLG 1.4 (test) dataset (hence both models have seen the entities during training).

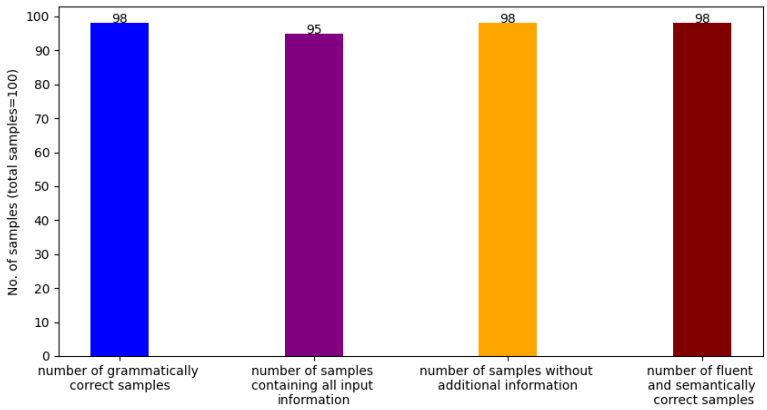
The human evaluation is focusing on these features:

1. Grammar correctness.
2. Whether the generated text captures all information from the input.
3. Whether the generated text does not contain any information that was not in the input.
4. Whether the generated text is fluent and semantically correct.

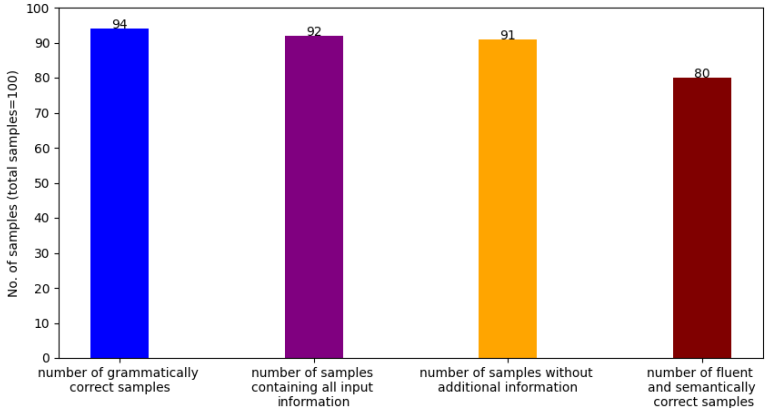
Each sentence is evaluated on 'binary' basis, i.e., a text can be either grammatically correct or incorrect, it can contain all information from the input or not, it has additional information, or it does not, and it is fluent and semantically correct, or it is not.

9.1 Results

The results of the human evaluation are provided in a form of two figures (one for each model — LSTM/T5), each containing two bar charts (one for seen and the other for unseen data):

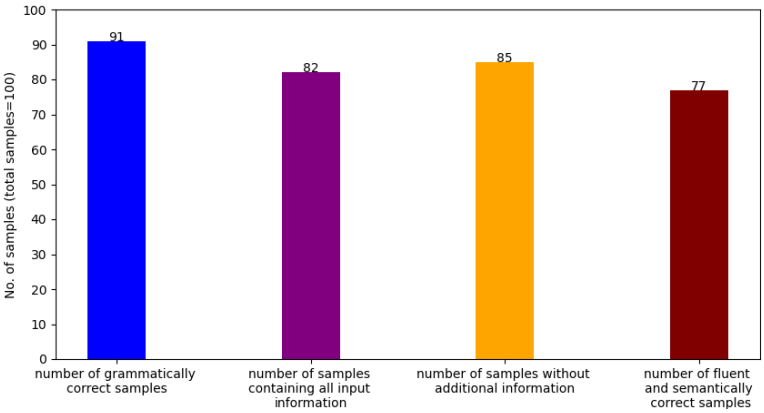


(a) : Human evaluation on in-domain (seen) part of the test data.

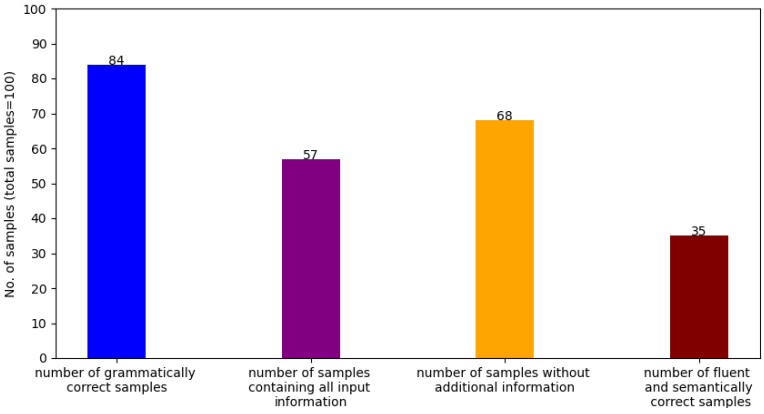


(b) : Human evaluation on out-of-domain (unseen) part of the test data.

Figure 9.1: Human evaluation of T5-large model + Roberta-large ranker.



(a) : Human evaluation on in-domain (seen) part of the test data.



(b) : Human evaluation on out-of-domain (unseen) part of the test data.

Figure 9.2: Human evaluation of ikl_{cpa} model.

9.2 Discussion

The manual human evaluation proved that the expectations from Chapter 3 were correct, i.e., the LSTM model has lower performance in general, moreover, while on in-domain data, the performance is decent; on out-of-domain data, the quality of generated texts drops significantly.

During the manual evaluation, we have found that both models sometimes mismatched *subjects* for *objects* and vice versa. The LSTM model did so more often. However, in some input triples (in both data sets), the objects and subjects seem to be swapped, as shown here:

```
No-hair_theorem | knownFor | Brandon_Carter
Anthropic_principle | knownFor | Brandon_Carter
```

In these cases, the triples seem to be wrong — i.e., *Brandon Carter* should be a subject, and the *thing* he is known for should be an object. Note that the error when subjects and objects are swapped in the hypothesis affects negatively the *fluency+semantic correctness* (dark-red bar in the reported figures).

The T5 model is better in dealing with such cases probably because it has seen many data during pretraining. Hence, it can 'know' the meaning of the entities inside the triples, which can help in generating the target text.

Perhaps if no subjects-objects were swapped, the results would be improved more for the LSTM model than for the T5.

Chapter 10

API

The final system uses KFServing [1] for serving the models for serverless inference. The idea is that the system will run in a Docker (hence the repository is containing a docker file to create a docker image) [2] and that the Docker image will be deployed onto a Kubernetes [3] cluster.

10.1 KFServing

KFServing uses two cloud technologies:

1. **Knative:** is a platform for Kubernetes to deploy and manage serverless workloads.

2. **Istio:** is a mesh that helps to manage networking microservices.

The whole system based on KFServing can be visualized as follows:

[1] https://github.com/kubeflow/kfserving
[2] https://www.docker.com/
[3] https://kubernetes.io/

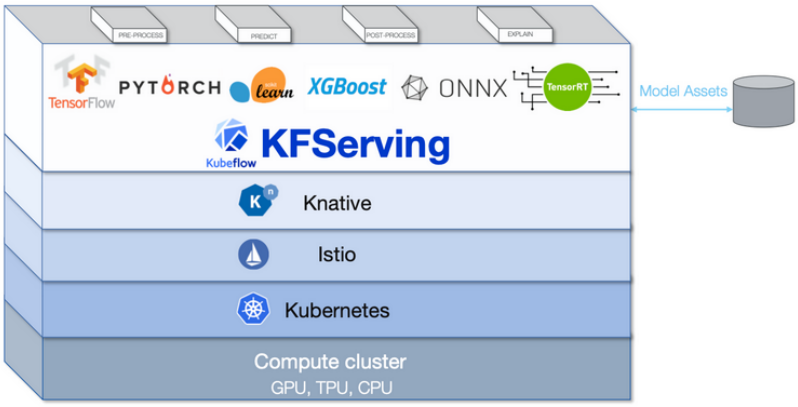


Figure 10.1: KFServing visualization from *https : //github.com/kubeflow/kfserving*.

10.2 Usage of the System

The system can be used by a POST request, e.g., by curl [4]. The request must contain the linearized input triple set (string) and can include different options:

1. Whether to use LSTM or T5.
2. Whether to use Roberta ranker or not.
3. Whether to use text structuring.
4. If using T5 - whether to use T5-base or T5-large.
5. If using Roberta - whether to use Roberta-base or Roberta-large.

The output of the system is a string — the verbalized input data.

10.3 Implementation Details

The system uses functions for delexicalization, relexicalization and structuring as described in 3. During initialization, the system loads all pretrained models,

[4] https://curl.se/

i.e. the *ikl_cpa*, *T5-base*, *T5-large* and the *Roberta ranker*. The main class implements all necessary methods required by KFServing, namely

1. ___init___
2. load (to load the models)
3. predict

Except for the name of the model and the running port, the system does not take any other arguments during initialization.

Delexicalization, relexicalization, structuring and all other preprocessing and postprocessing steps are done inside the main class (including restructuring and inference).

The code can be found in the root directory in $final\_system.py$.

Chapter 11

Conclusion

The goal of this thesis was to create a system that is able to make natural texts from given triple sets. We examined existing datasets and chose to use two different versions of the WebNLG dataset. The thesis described two main approaches to solve the main task, where one approach is based on LSTM neural networks and has low computational demands while the other is based on pretrained Transformers. The thesis examined the computational costs of each of the models as well as the performance of each of the models. Additionally, the thesis proposed further steps, which could improve text generation, such as triples' ordering, text structuring, and ranking. While ordering did not bring any improvement, text structuring brought improvement on some automatic metrics, and ranker improved performance on all measured metrics (but only a little).

Automatic metrics evaluated all proposed solutions, and the two best various systems were evaluated manually. The outcome of the automatic evaluation and the manual evaluation is that the Transformer-based system (T5 + Roberta ranker) works well on in-domain as well as on out-of-domain data. On the other hand, the LSTM-based system performs well on in-domain data but cannot sufficiently perform on out-of-domain data. Hence the thesis found a trade-off between good generalization on one side (Transformers) and between low computational requirements on the other side (LSTM).

All trained models are standard PyTorch (Paszke et al., 2019) models compatible with PyTorch 1.5.0 and newer versions. The LSTM is built upon the OpenNMT library, while the T5 and Roberta are built upon the Huggingface Transformers.

The last part of the work puts both models and other possible steps (text structuring, ordering, ranking) together into a single system that is virtualized in a Docker image and served by KFServing. The final system provides an API, which allows selecting a particular model type (transformer/LSTM), for transformers a specific size of the model (base/large) and whether to use text structuring, (heuristic) ordering and ranking (this is not dependent on the selected model type). The system then takes a single triple set or a batch of triple sets and outputs a list of texts.

Bibliography

URL https://www.dbpedia.org/.

Gaurav Arora, Afshin Rahimi, and Timothy Baldwin. Does an LSTM forget more than a CNN? an empirical study of catastrophic forgetting in NLP. In *Proceedings of the The 17th Annual Workshop of the Australasian Language Technology Association*, pages 77–86, Sydney, Australia, 4–6 December 2019. Australasian Language Technology Association. URL https://www.aclweb.org/anthology/U19-1011.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

Alexander Shekhovtsov Boris Flach. Bev033dle – deep learning - lecture slides. URL https://cw.fel.cvut.cz/b192/courses/bev033dle/start.

Thiago Castro Ferreira, Claire Gardent, Nikolai Ilinykh, Chris van der Lee, Simon Mille, Diego Moussallem, and Anastasia Shimorina. The 2020 bilingual, bi-directional WebNLG+ shared task: Overview and evaluation results (WebNLG+ 2020). In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 55–76, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.webnlg-1.7.

Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. Recall and learn: Fine-tuning deep pretrained language models with less forgetting, 2020.

Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Bayu Distiawan. URL `https://webnlg-challenge.loria.fr/files/melbourne_report.pdf`.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation, 2018.

Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures, 2019.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-3518. URL `https://www.aclweb.org/anthology/W17-3518`.

Qipeng Guo, Zhijing Jin, Ning Dai, Xipeng Qiu, Xiangyang Xue, David Wipf, and Zheng Zhang. $\sqrt{}^{\in}$: A plan-and-pretrain approach for knowledge graph-to-text generation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 100–106, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.webnlg-1.10`.

Hamza Harkous, Isabel Groves, and Amir Saffari. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity, 2020a.

Hamza Harkous, Isabel Groves, and Amir Saffari. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity, 2020b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

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

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020.

James M. Joyce. *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_327. URL https://doi.org/10.1007/978-3-642-04898-2_327.

Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, Upper Saddle River, N.J., 2009. ISBN 9780131873216 0131873210. URL http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

Zdeněk Kasner and Ondřej Dušek. Train hard, finetune easy: Multilingual denoising for RDF-to-text generation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 171–176, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.webnlg-1.20.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P17-4012.

et al. Abstract Meaning Representation (AMR) Annotation Release 2.0 LDC2017T10. Web Download. Philadelphia: Linguistic Data Consortium 2017. Knight, Kevin.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.

Xintong Li, Aleksandre Maskharashvili, Symon Jory Stevens-Guille, and Michael White. Leveraging large pretrained models for WebNLG 2020. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 117–124, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.webnlg-1.12.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

Jekaterina Novikova, Ondrej Dušek, and Verena 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*, Saarbrücken, Germany, 2017. URL `https://arxiv.org/abs/1706.09254`. arXiv:1706.09254.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL `https://www.aclweb.org/anthology/P02-1040`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. 'Alch' e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

Marcus Perry. *The Exponentially Weighted Moving Average*. 06 2010. doi: 10.1002/9780470400531.eorms0314.

Jan Pichl, Petr Marek, Jakub Konrád, Martin Matulík, Hoang Long Nguyen, and Jan Šedivý. Alquist: The alexa prize socialbot, 2018.

Jan Pichl, Petr Marek, Jakub Konrád, Petr Lorenc, Van Duy Ta, and Jan Šedivý. Alquist 3.0: Alexa prize bot using conversational knowledge graph, 2020a.

Jan Pichl, Petr Marek, Jakub Konrád, Martin Matulík, and Jan Šedivý. Alquist 2.0: Alexa prize socialbot based on sub-dialogue models, 2020b.

Maja Popovic. chrf ++ : words helping character n-grams. 09 2017. doi: 10.18653/v1/W17-4770.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:Paper No. 140, 67, 2020. ISSN 1532-4435. doi: 10.1080/15502287.2020.1772903. URL `https://doi.org/10.1080/15502287.2020.1772903`.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

Monika Schak and Alexander Gepperth. *A Study on Catastrophic Forgetting in Deep LSTM Networks*, pages 714–728. 09 2019. ISBN 978-3-030-30483-6. doi: 10.1007/978-3-030-30484-3_56.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks, 2017.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. 01 2006.

Derya Soydaner. A comparison of optimization algorithms for deep learning. *International Journal of Pattern Recognition and Artificial Intelligence*, 34 (13):2052013, Apr 2020. ISSN 1793-6381. doi: 10.1142/s0218001420520138. URL `http://dx.doi.org/10.1142/S0218001420520138`.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation, 2016.

C. J. van Rijsbergen. *Information retrieval*. Butterworths, London, 2 edition, 1979. URL `http://www.dcs.gla.ac.uk/Keith/Preface.html`.

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

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017.

Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1): 36–45, January 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL `https://doi.org/10.1145/365153.365168`.

Tomas Werner. Optimalizace (elektronicka skripta predmetu a4b33opt), 2018. URL `https://cw.fel.cvut.cz/old/_media/courses/a4b33opt/opt.pdf`.

E. Wilson and D. W. Tufts. Multilayer perceptron design algorithm. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 61–68, 1994. doi: 10.1109/NNSP.1994.366063.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

Yaoming Zhu, Juncheng Wan, Zhiming Zhou, Liheng Chen, Lin Qiu, Weinan Zhang, Xin Jiang, and Yong Yu. Triple-to-text. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Jul 2019. doi: 10.1145/3331184.3331232. URL `http://dx.doi.org/10.1145/3331184.3331232`.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kobza** Jméno: **Ondřej** Osobní číslo: **457004**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Generování přirozeného jazyka ze znalostních databází

Název diplomové práce anglicky:

Natural Language Generation from Knowledge-Base Triples

Pokyny pro vypracování:

1.Prozkoumejte veřejně dostupné množiny dat vhodné pro natrénování modelu pro generaci přirozeného jazyka ze znalostních databází.
2. Prozkoumejte existující zdroje relevantní k dané úloze.
3. Na základě zdrojů z bodu 2. navrhněte možná řešení zadaného problému.
4. Implementujte a natrénujte navrhované modely na datové množině z bodu 1.
5. Najděte (popř. navrhněte) metriku pro hodnocení kvality jednotlivých modelů.
6. Porovnejte výsledky jednotlivých modelů/řešení, zhodnoťte jednotlivá řešení na základě metriky z předchozího bodu a na základě manuální evaluace.
7. Vytvořte API pro použití modelu (jenž dosahoval nejlepších výsledků v bodě 6.) tak, aby bylo možné daný model integrovat s dialogovým systémem.

Seznam doporučené literatury:

Ferreira, T. C., van der Lee, C., van Miltenburg, E., & Krahmer, E. (2019). Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. arXiv preprint arXiv:1908.09022.
Gardent, C., Shimorina, A., Narayan, S., & Perez-Beltrachini, L. (2017, September). The WebNLG challenge: Generating text from RDF data. In Proceedings of the 10th International Conference on Natural Language Generation (pp. 124-133).
Jurafsky, Daniel & Martin, James. (2008). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.
Zhu, Y., Wan, J., Zhou, Z., Chen, L., Qiu, L., Zhang, W., ... & Yu, Y. (2019, July). Triple-to-text: converting RDF triples into high-quality natural languages via optimizing an inverse KL divergence. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 455-464).
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Petr Marek, velká data a cloud computing CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **21.02.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2023**

Ing. Petr Marek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

