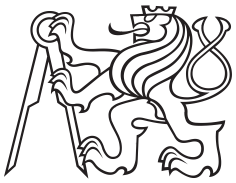
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Natural Language Generation From Structured Data

Bc. Martin Matulík

Supervisor: Ing. Jan Šedivý
Field of study: Computer Science
Subfield: Data Science
May 2018

Acknowledgements

I would like to thank Ing. Jan Šedivý for supervising me and providing me with advice on this thesis. I would also like to thank my family for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.
Prague, 23. May 2018

Abstract

Natural language generation is one of the hardest tasks of machine learning. Usually, the task is to convey some information stored in a structured form. In this work, we implement and test a system based on a neural language model which attempts to generate natural language sentences from data contained in a table.

Keywords: natural language generation, language model, structured data

Supervisor: Ing. Jan Šedivý
Czech Institute for Informatics, Robotics and Cybernetics,
Jugoslávských partyzánů 1580/3, 160 00 Dejvice

Abstrakt

Generování přirozeného jazyka je jedna z nejtěžších úloh strojového učení. Jejím cílem je obvykle prezentovat informaci původně uloženou ve strukturované podobě. V této práci implementuji a zkoumám systém založený na principu jazykového modelu, který generuje věty v přirozeném jazyce z dat uložených v tabulce.

Klíčová slova: generování přirozeného jazyka, jazykový model, strukturovaná data

Překlad názvu: Generování přirozeného jazyka ze strukturovaných dat

Contents

Figures

Tables

ctuthesis t1606152353

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Matulík** Jméno: **Martin** Osobní číslo: **420292**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Datové vědy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Generování přirozeného jazyka ze strukturovaných dat

Název diplomové práce anglicky:

Natural language generation from structured data

Pokyny pro vypracování:

The student will research current methods of generating natural language (specifically, a dialogue) based on information stored in structured form (for example RDF triples). The student will implement one or more of these systems and measure their performance during live testing with users. THe student will describe thier implementation and findings in a report.

Seznam doporučené literatury:

[1] Lebret, Rémi - Neural Text Generation from Structured Data with Application to Biography Domain -2016
[2] Sharma, Shikhar - NLG in Dialogue using Lexicalized and Delexicalized Data -2017
[3] Sha, Lei - Order- Planning Neural Text Generation from Structured Data - 2017
[4] Mei, Hongyuan - Selective Generation using LSTMs with Coarse-to-Fine Alignment-2016

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Šedivý, CSc., velká data a cloud computing CIIRC

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

Datum zadání diplomové práce: **15.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

Ing. Jan Šedivý, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. 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 _____

Chapter 1

Introduction

The amount of data worldwide grows fast every day. There are many problems associated with this phenomenon, be it storage, processing or proper usage. In this flood of information, it is hard to decide how to present it, pick relevant or interesting pieces, or summarize it. Often, there is a need to convert the data stored in a structured format such as tables or knowledge graphs into a form which allows easy interpretation and provides comfort to the user. The fields which deal with this problem are for example data visualization or natural language generation. It is the last field mentioned that we will explore in this thesis. The goal of natural language generation is creating a sentence in natural ('human') language based on information stored in a structured form. For instance, the personal assistants (such as Amazon Alexa, Microsoft Cortana or Google Assistant) which have recently been becoming more and more popular have to convert information which the user desires into a single utterance. One solution is hand-crafted responses where the information is inserted as a substitution for delexicalized tokens, but with great volume and variety in the data, it is laborious at best and impossible at worst to cover all possible cases. Therefore, data-driven approaches to natural language generation are on the rise (with the popular neural networks in the lead), eliminating the need for human interventions as well as utilizing the ever-growing heaps of data.

1.1 Motivation

The work on this thesis was conducted as part of Alquist, a conversational socialbot participating in the Alexa Prize competition organized by Amazon[RPK+18]. The competition is intended for university teams and its goal is building a socialbot on the Amazon Alexa platform which will converse with a user coherently and engagingly about different popular topics such as sports, movies or music. The conversation should go on for as long as possible (Amazon set a duration of 20 minutes as the main milestone to be overcome). To keep the conversation going, the socialbot needs to present a piece of information to the user from time to time (for example to find a new subject to talk about, provide details about the current subject or simply answer the user's questions). This task is best suited for a natural language generating system, so the goal of this thesis was to explore the current state of this task and test if a certain system could improve the socialbot.

1.2 Goals

The first goal of this thesis is to research the topic and compile available approaches to it. The next goal is to implement a natural language generation system based on machine learning. The experiments consist of measuring BLEU and perplexity over multiple datasets and also observing how the system reacts to various subsets of information contained in the structured data (for example, if we train the system on data which include name, birth date and an occupation of a person, we want to know how it would perform and what sentences it would generate on data which include just a name and an occupation).

1.3 Structure

The thesis is structured as follows: In chapter 1 we describe the task and set goals for this thesis. Next, in chapter 2, we provide background for the task. We talk about language models, neural networks, and natural language generation. In chapter 3, we present currently utilized approaches to our problem. In chapter 4 we describe our own solution: a language generation system implemented in Python language. In the next chapter, 5, we discuss

our experiments with this system. We sum up our findings in chapter 6. All pictures are our own, created in the Inkscape editor[1].

[1] https://www.inkscape.org

Chapter 2

Theoretical background

2.1 Language model

Language model is a probability distribution which assumes that any sequence $s = \{w_1, w_2, ..., w_m\}$ of words in a given language (not necessarily a sentence) that has length m can be assigned a probability $P(s) = P(w_1, w_2, ..., w_m)$. Being able to assign probabilities to sequences of words and compare them is useful since this way the computers can identify and distinguish "nonsensical" sequences (those which have low probabilities) and sequences which "make sense" (those with high probabilities). This method is applied in problems such as machine translation[LOW12], natural language processing[KJ13], speech recognition[JMRS91] or information retrieval[PC98]. In machine translation, it is impossible to simply change words from one language to another since the result will be garbled. With language model in use, the translation system can determine the word order, choose more appropriate word translations or correct shapes of words.

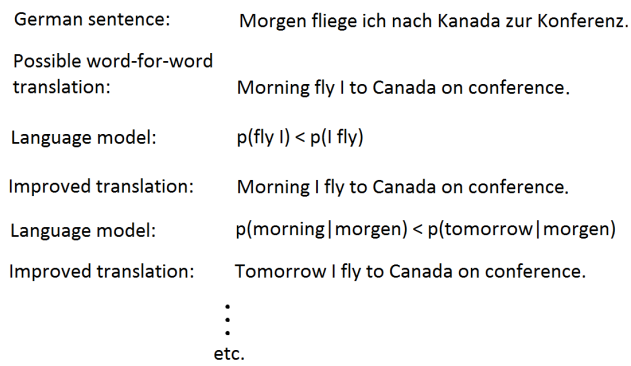


Figure 2.1: Examples of how a language model can improve machine translation

Part-of-speech tagging is one of the tasks of natural language processing where each word is assigned a tag from a predetermined set based on the word's role in a sentence. The language model can help determine the correct tag in case a word can be interpreted in more ways. In figure 2.2, it can be seen that the word 'watch', while unchanged, performs a completely different role in each of the two sentences.

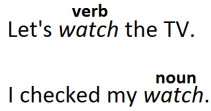


Figure 2.2: Ambiguity of part of speech tagging

In speech recognition, the input is a sound recording and output is text. Certain phonemes sound alike and different text sequences can be constructed from the same recording. The language model can improve recognition accuracy by determining which sequence is more likely to happen in the recognized language. Finally, in information retrieval, we need to compare documents to our query. The information about how likely the query belongs to a given document's language model can contribute to the score based on which the documents are ranked.
We can obtain the best estimate of output Y of the language model by maximizing the a posteriori probability of Y given input X which is, for fixed X, equivalent to a joint probability

$$P(Y, X) = P(Y) \cdot P(X|Y)$$

where $P(X|Y)$ is a conditional probability and $P(Y)$ is the a priori probability of the output sequence Y occurring in the given language. The method of obtaining $P(Y)$ is discussed in the following section.

2.2 N-gram language model

This type of model [BdM+92] deals with the task of obtaining $P(Y)$, the a priori probability of a word in a language. It represents the language model by conditional probability of the next word $P(w_k)$ given words which precede it in a sequence of words. The formula to obtain this probability is a product of conditional probabilities

$$P(w_k) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2, w_1) ... \cdot P(w_k|w_{k-1}, ..., w_1)$$

The words $w_1, ..., w_{k-1}$ are called history or context. This approach takes advantage of word order and assumes that the words appearing close together are statistically dependent. Since more conditions in a conditional probability give more accurate results, we would ideally want to compute a probability of the word given arbitrarily long context, but there are several problems with that. First of all, the probabilities need to be computed (trained) on a corpus of text written in the language we want to model (Exact training method is described below). We would need a very large dataset to obtain a reliable representation of a language. Next problem is with model storage. All probabilities are stored in a table, e.g.

Context	*this old*	*my little*
house	0.8	0.7
man	0.6	0.1

Table 2.1: Example slice of a table containing probabilities. The value in each cell is the probability of a word (to the left) given two context words (above) - this is a trigram model.

Size of such table is V^k which means it grows exponentially with regard to vocabulary size V given increasing context length k, which becomes computationally infeasible for large k. Another problem with very long histories is that they might occur only a few times at best in the training set and never occur during inference time.
The n-gram model attempts to approximate this conditional probability by considering only the contexts of length $n-1$. Commonly used values of n are 2 (the model is then called bigram) or 3 (trigram model). This simplification eliminates the storage problem since the size of the table (or the number of parameters) is only V^2 or V^3.
As mentioned before, the probabilities of n-grams are computed by examining a text in the given language. For unigram (that is, $n = 1$), the maximum likelihood estimate of the probability $P(w_n)$ is simply

$$P(w_n) = \frac{C(w_n)}{T}$$

where $C(w_n)$ is the number of occurrences of the word in the training text and T is the length of the training text. To obtain the estimate for an n-gram, we need to count all occurrences of the n-gram as well as all occurrences of the $(n-1)$-gram which appears as the context for the last word in the sequence and divide both numbers. For example, if we wanted an estimate of the conditional probability of the trigram phrase *this old house*, we would need to count how many times this three-word phrase appears in the training text and then how many times the two-word (bigram) phrase *this old* appears. The resulting estimate would be the quotient of these two numbers. To put it into a formula

$$P(w_n) = \frac{C(w_1, w_2, ..., w_{n-1}, w_n)}{C(w_1, w_2, ..., w_{n-1})} \simeq \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}$$

We will use this representation for n-gram counts from now on.

2.3 Smoothing

Although shortening the length of the context helps with the problem of sequences not occurring often enough or at all, it does not solve it altogether. Certain n-grams might not be seen during training anyway and therefore the model does not assign them a probability - or rather, it assigns them a zero probability. We want to assign each possible n-gram a probability, even if it is a very small one, which is where smoothing (or discounting) comes in. One of the approaches to smoothing is adding a constant to all n-gram counts. The method which consists of adding 1 to every count is called Laplace smoothing [Lid20]. This changes the nominator of the formula but not the denominator, which we need to increase as well. We "observed" each n-gram once more, so value by which the denominator is increased is the size of the vocabulary V.

$$P(w_n) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$$

General constant-adding method is called the add-k smoothing (based on that, the Laplace smoothing can be called add-one smoothing). The constant k is usually a real number greater than 0 and lower than 1. The denominator needs to be adjusted again.

$$P(w_n) = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + kV}$$

The disadvantage of this approach is that we need a way to determine the best value of k. Generally, the add-k methods do not work well when applied to language modeling tasks.
A different way of smoothing is reverting to a lower order, that is, utilizing an

n-gram model with lower n on the same text. There are two main approaches: Backoff and interpolation [PH08]. When using backoff smoothing, we always revert exclusively to a lower n-gram in case of zero probability. For example, if we are unable to find a trigram (w_{n-2}, w_{n-1}, w_n) in the training text corpus, we substitute in the count of bigram (w_{n-1}, w_n) instead. If that count is also zero, we lower n further and further until we find an n-gram whose count is non-zero. If necessary, we go as far as unigram which has always non-zero count else the word would not appear in the vocabulary at all and we would not need to compute its probability. The other approach, interpolation[JM80], also utilizes the lower order n-grams, but instead of using only one it bases the probability on all n-grams with n having a value from the originally intended n-gram model up to possibly the unigram. The formula (here for trigram) is a linear combination

$$P(w_n|w_{n-1}, w_{n-2}) = \lambda_1 P(w_n|w_{n-1}, w_{n-2}) + \lambda_1 P(w_n|w_{n-1}) + \lambda_1 P(w_n)$$

The linear coefficients λ_i need to be computed. For that, we can use EM algorithm which fixes the n-gram counts and finds hyperparameters λ_i that maximize the probabilities over a validation dataset.

2.4 Neural networks

Artificial neural networks[Hay98] are machine learning systems based on phenomena observed in nature, specifically on animal and human brains. They can learn on their own (i.e. without any specific programming, hand-crafted rules or prior knowledge) approximate solutions to various tasks. Nowadays, they are widely applied in fields such as pattern recognition, sequence recognition, data mining or machine translation. The neural networks manage to grasp the underlying logic quite well but they have several disadvantages. They require a lot of training data (usually including labels) to learn the parameters reliably, the learned models are hard to interpret and they have high computational requirements. On top of that, the architecture and hyperparameters of a neural network need to be determined empirically.
The most basic building block of an artificial neural network is a neuron. Same as its biological counterpart, it has several inputs, processing core, and an output. The inputs are real numbers (they can be understood as one vector). The processing core contains a function which is applied to the inputs whose result becomes the output of the neuron. The function is usually a dot product of the input vector and weights (that is, a real number-valued vector) stored also in the neuron as a stand-in for a memory.

$$f(x) = f(x_1, x_2, ..., x_m) = \langle x; w \rangle = x_1 \cdot w_1 + x_2 \cdot w_2 + ... + x_m \cdot w_m$$

In this formula and in the following ones, w_i stands for a weight value. The weights are randomly initialized and periodically updated during training to learn the task - details about that come below. In that case, the neuron is basically equivalent to standard perceptron[Ros58].

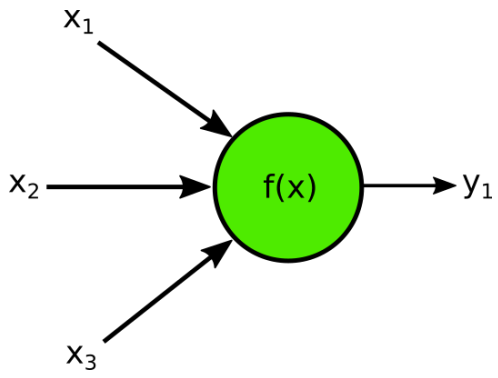


Figure 2.3: A depiction of a neuron

The neurons are gathered into neural layers. The layer is an array of neurons, in this context also called units or 'hidden units'. The units share all layer inputs, but each of them has its own weight vector. From mathematical point of view, instead of multiplying the input vector by another vector, we multiply it by a matrix of size $m \times u$, where m is the length of an input and u is number of units in a layer. Often a bias vector b is additionally used.

$$f(x) = \langle x; W \rangle + b = \begin{pmatrix} x_1 \cdot w_{11} + x_2 \cdot w_{12} + ... + x_m \cdot w_{1m} + b_1 \\ x_1 \cdot w_{21} + x_2 \cdot w_{22} + ... + x_m \cdot w_{2m} + b_2 \\ \vdots \\ x_u \cdot w_{u1} + x_2 \cdot w_{u2} + ... + x_m \cdot w_{um} + b_u \end{pmatrix}$$

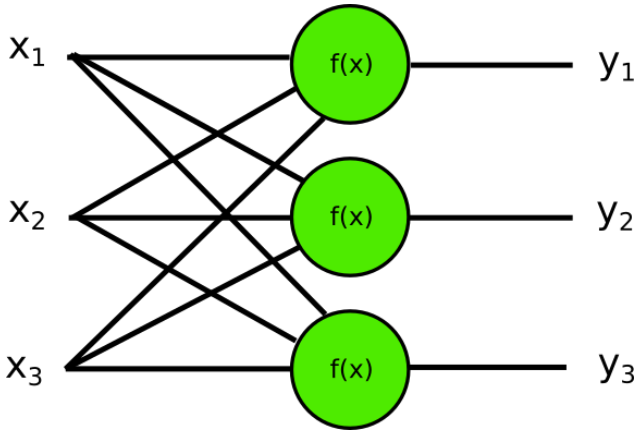


Figure 2.4: A neural layer

In networks with multiple layers, the layers are connected so that one layer's output becomes the next layer's input.

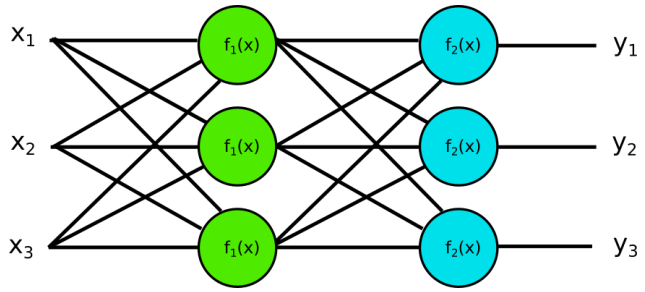


Figure 2.5: Neural network with two layers

If we want to modify the output of a neuron, an activation function can be applied to it. Activation functions' purpose ranges from normalizing the output - e.g. softmax

$$\sigma(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

to thresholding it - e.g. rectified linear unit (ReLU)

$$f(x) = max(0, x)$$

where x is the unit output.
Above, we mentioned that the artificial neural networks are able to "learn". By that, we mean that the network is able to minimize a given error function (also called loss function) over a training dataset by updating the weights of neural layers using an optimization algorithm, usually a gradient descent. The training dataset is a set of example inputs and outputs. When we apply the neural network to an input, we want to obtain the output value associated with that input. By computing the error function we know how much the neural network deviates from this value and we can update the weights accordingly. The value by which the weights are updated is found by multiplying the input with the difference between the predicted value and the true value. A hyperparameter which is utilized here is the learning rate. It is a coefficient by which is the update multiplied so that optimum can be found. The update is performed in all neural layers in the network. The mechanism which is used to compute the updates in all layers is called backpropagation [WJPJ74].

2.5 Neural language model

When we are attempting to model a language, we are effectively searching for a joint probability between words, that is, discrete random variables. However, the size of vocabulary V for various tasks can easily reach tens or even hundreds of thousands of words. This means the model has a great number of parameters to learn. As was already mentioned in section 2.2, the number of parameters of n-gram model is $V^n - 1$ which, for example, in trigram model with a relatively small vocabulary of 10,000 words reaches astonishing $10^{12} - 1$ parameters. This problem is referred to as "curse of dimensionality" and language modeling is not the only learning task which suffers from it.
Another problem of statistical language models is that they do not consider the semantic similarity between words which, if utilized, could help with generalization and improve performance. For example, if the training corpus contains the sentence *A woman was walking on a street*, the model should be able to generate sentence *A man was running on a road* since there are pairs of words that have similar semantical meaning and even similar grammatical roles in a sentence - e.g. *man* and *woman*, *walking* and *running* or *street* and *road*.
A neural language model proposed by [BDVJ03] attempts to solve both of these problems by utilizing a neural network. The joint probability function

$$P(w_t|w_1^{t-1}) = f(w_t, ..., w_{t-n+1})$$

is split into two parts. The first part which counters the curse of dimensionality is distributed word feature vectors. Each word w is mapped using a function $C(w_i)$ to a real-valued vector with a fixed length which associates the word with a point in a vector space. The advantage of the distributed feature vector which should bring the desired improvement to the language model is that semantically similar words should map to points close to each other, that is, their feature vectors should be similar as well. Another improvement is that the length of feature vector can be several orders smaller than a usually used one-hot encoding (which has a length equal to the size of the vocabulary V) and we can tune it to obtain better results. The second part of the neural language model is a function g which maps the distributed word feature vectors $C(w_i)$ to a probability distribution. This function is implemented by feed-forward or recurrent neural network. Both parts put together form the function f:

$$f(w_t, ..., w_{t-n+1}) = g(w_t, C(w_{t-1}), ..., C(w_{t-n+1}))$$

The authors state that it is best to train both the word feature vectors and probability function parameters at the same time. The training is performed by finding parameters θ which minimizes the log-likelihood

$$L = \frac{1}{T} \sum_t log f(w_t, ..., w_{t-n+1}; \theta)$$

over all samples from the training dataset.

2.6 Natural Language Generation

The task of generating natural language [GK18] can be defined as transforming input data (structured or unstructured) into output sentence written in natural language. It is often split into six subtasks:

1. content determination
2. text structuring
3. sentence aggregation
4. lexicalization
5. referring expression generation
6. linguistic realization

The tasks are usually performed roughly in the order they were listed here, especially in systems with pipeline structure. One thing to note is that further down the list, the subtasks are less and less connected to the domain they are applied to, i.e. the content determination is closely interlinked with the application while lexicalization or linguistic realization can be researched independently on a task and the methods for them can be applied to various problems. It can be generally said about all of the steps that historically, they were initially performed using hand-crafted rules and the preferred methods later moved towards data-driven approaches.

2.6.1 Content determination

The content determination is the subtask of choosing which pieces of information we want to include in the output sentence and which should we drop. Typically, the data from which we plan to generate a natural text from contains much more information than what we desire to convey in a text, or it is too detailed (which would result in the sentence being convoluted).

The selection of data is based on several factors. One factor is the possible audience of the generated text. If a piece of information requires expert knowledge (for instance in medical data), but the target audience is composed of novices or laymen, we may want not to include it. Another factor is the purpose of the text: chosen content will be different in a text whose goal is just to inform the reader about certain facts and in a text which aims to convince the audience about something. An obvious factor is the relevance and importance of the information. For example, if we collect medical data from continuously running sensors about patient's physical parameters such as heart rate, temperature or blood pressure, the data will have a lot of the details but most of it would be of no interest to us. What we are interested in are sudden changes or abnormal values. Therefore, the data must be filtered as a part of the selection process.

2.6.2 Text structuring

Text structuring is the subtask of deciding the order of information in which it is presented to the reader, that is, constructing a temporal sequence. Again, there are multiple approaches to ordering the information. One of them is starting with general information and going into finer and finer details further into the text. For example, in a text about an ice hockey match, we would like the result to be generated first and then who scored the goals and other highlights of the match. If we have data which contain information about the chronological order of events (like the goals in the previous example), we obviously want to retell the events in chronological order as well. Another method is ordering the information by importance, beginning with the most important bits and only later introducing the less interesting pieces of information. As certain relations between pieces of data may arise during their processing, we also need to regard them when solving the text structuring.

2.6.3 Sentence aggregation

If we want the text to be coherent and not just a collection of sentences, we may need to observe the sentences whether it is not possible to cluster multiple pieces of the same kind of information together into a single sentence instead of several ones which are almost the same. For example, the sentences

1. David Pastrnak scored for Boston Bruins in a match against Toronto Maple Leafs at 5:26.

2. David Pastrnak scored for Boston Bruins in a match against Toronto Maple Leafs at 12:34.

3. David Pastrnak scored for Boston Bruins in a match against Toronto Maple Leafs at 18:24.

can be merged into one sentence *David Pastrnak scored three goals for Boston Bruins in a match against Toronto Maple Leafs.* This subtask is called sentence aggregation. It is one of the most difficult subtasks of natural language generation because of several complications. It is very application-dependent and in some cases, it is debatable whether it should be performed. Another problem is that what is referred to as text aggregation can be interpreted in different ways: eliminating redundant words or even sentences, combining the underlying linguistic structure of a sentence, etc.

2.6.4 Lexicalization

When the choice of content is finalized by the previous steps, it is time to choose proper words and phrases to transform the data into natural language text. This is solved in lexicalization subtask. Main hurdle to overcome is that same concepts can be expressed in different ways in natural language and it is up to our system to decide which way is the best one. For example, when we want to generate a sentence about a player scoring a goal, it could be expressed by phrases *to score a goal*, *to score* or *to have a goal noted*. The more possibilities there are for the language model to generate, the more complex the lexicalization process becomes. Also, when implementing a lexicalization system we need to decide if we prefer the sentences to be more varied or to be homogeneous. This is again application-dependent, for instance, readers of a report from a sports match would prefer the text to have variety while the summary of medical data should be concise and direct. One way to create a lexicalization model is mapping the data domain concepts directly to phrases. This seems straightforward but in fact is rather difficult even on well-defined domains. One problem is vagueness which arises for instance in gradable adjectives where we need some sort of point of reference - can the system say that a house is *small* when it is still likely taller than a *tall* human?

2.6.5 Referring expression generation

A subtask of natural text generation which has been given probably the most attention in the recent years is referring expression generation. One of the reasons for that is the fact that it could be researched as a separate topic. Referring expression generation is defined as the task of "communicating enough information to distinguish one domain entity from other domain entities". The expressions which the system will use to describe an entity are dependent on several factors. If the entity has been previously mentioned, a pronoun might suffice to refer to it. If there are other entities of the same category as the described entities in the examined domain, "our" entity needs to be told apart from them and therefore the system has to find features of the entity which characterize it and make it stand out. When referring to entities, the system needs to decide on two parts: referential form and referential content. The choice of referential form means the system selects whether to refer to the entity using a proper name, pronoun or (in)definite description. The choice of referential content is usually carried out if the chosen referential form is a description, definite or indefinite. It requires the system to find the combination of properties of the entity which it does not share with other entities (which are put into a role of "distractors") so the audience of the generated text is able to recognize which entity it is about. The algorithms which solve referential content choice are based on finding the "best" combination of properties. This combination should contain neither too few (as the distinguishing information about the entity can be lost) or too many properties (as too many details describing an entity could appear artificial and even boring). There are several approaches to solving this problem:

- Building a set of all possible combinations of properties, then performing an exhaustive search which finds the smallest possible set which will reliably identify the target entity[Dal89]
- Building the combination of properties incrementally by adding a property which eliminates the most distractors in each step. [CFDGBT09]
- Similar to the algorithm in the previous point, but instead of most distinguishing property, the selection is based on knowledge about the domain.[DR95]

All of these algorithms prioritize minimizing the number of properties. However, in some cases, it is desirable to actually include redundant information [JW05]. Another method which aims to expand the expressive possibilities of a text generation system considers plurals and relations between objects (e.g. their relative position).

2.6.6 Linguistic realization

The final task of the imagined pipeline is linguistic realization. In this step, all of the selected and processed input is transformed into a natural language text. This consists not only of mapping the entities to correct words and choosing the right morphological forms but also of inserting punctuation, functional words (such as prepositions and auxiliary words) and other elements required for the text to be fluid and coherent - elements often not included in the input data at all. The main approaches to this problem are *human-crafted templates*, *grammar-based systems* and *statistical methods*. Human-crafted templates are suitable to be used in closed domains. Returning to the ice hockey example

- David Pastrnak scored for Boston Bruins in a match against Toronto Maple Leafs at 5:26.

could be the result of applying the template

- <player> scored for <own_team> in a match against <other_team> at <minutes>:<seconds>.

which uses mapping *player*:*David Pastrnak*, *own_team*:*Boston Bruins*, etc. The advantage of this approach is that it offers complete control over the linguistic realization and the resulting sentences are very convincing. However, if we want to cover as many cases as possible, the templates need to be laboriously created by hand and as mentioned above, are not suitable for domains which require linguistic variety. Grammar-based are more advanced, but still require human-created grammar rules to generate sentences. The rules are based on the grammar of the given language. Statistical methods are those with the most variety and least control offered. They rely on large text corpora to determine the parameters of statistical models and also on human labor, but not as extensively as templates and grammars. One approach is generating all possible realizations using hand-crafted grammar, then choosing the best realization based on a statistical model. This is, however, computationally expensive. The other approach utilizes the information from the statistical model already during the generation step, aiding the human-created generator with its choices.

Chapter 3

Related work

3.1 Neural text generation

The method described in [LGA16] is applied to the problem of generating a first sentence of a Wikipedia article. The task is constrained to biography articles about people since the first sentences of them are often very similar. The authors create a neural language model to solve this task. The language model is based on standard n-gram language model, that is, the next word of the sentence is generated based on previous (context) words. In a simple model, the probability (score) of the next word would be

$$P(w_i|c_i) = \prod_{t=i-n}^{i-1} P(w_t|c_t)$$

where w_i is the words being generated and c_i represents the context words. However, in this article the authors add several layers of conditioning (derived from the field-value pairs from the infobox for such article) added on top. The probabilities are obtained from a neural network instead of a text corpus. The model looks like this

$$P(w_i|c_i, z_{c_i} g_f, g_w) = \prod_{t=i-n}^{i-1} P(w_t|c_t, z_{c_t}, g_f, g_w)$$

The meaning of c_i, z_{c_i}, g_f and g_w is explained in the following paragraphs.

Field	Value
name_1	Albert
name_2	Einstein
birth_date_1	14
birth_date_2	March
birth_date_3	1879
known_for_1	General
known_for_2	relativity
known_for_3	,
known_for_4	Special
known_for_5	relativity

Table 3.1: Example infobox structured the same way it is in the source data, taken from Albert Einstein Wikipedia article.

Same as the standard n-gram language model, c_i stands for context words preceding the generated word. These words are embedded into fixed-length vectors. As the corpus for training these vectors, the set of first sentences of the articles is used. Only W most common words are used, the rest is removed from the sentences or, where possible, replaced with keys from the tables. All sentences are prepended with n tokens representing the beginning of the sentence so that the first word of the actual sentence also has context. The embeddings are trained as part of the training of the whole network. They can be initialized randomly or with pre-trained vectors(such as the word2vec).
The local conditioning (z_{c_t})is computed from occurrences of a context word in the table. For each context word, list of fields where it occurs as well as the indexes is formed. The indexes are counted not only from the beginning but also from the end and they are capped to given length for fields with too many words. For example in the infobox depicted in table 3.1, the word relativity occurs in field *known_for* on indexes 2 and 5 counted from the beginning and on indexes 9 and 5 counted from the end (the actual field on the Wikipedia page contains much more entries which were omitted for brevity's sake). The end-indexes help to capture the information that the field terminates the sentence. Symbols such as comma are also included. The tables undergo preprocessing - only fields occurring more than a certain number of times in the dataset are used, the rest is disregarded. Pairs formed from field name and index are embedded into a vector. The embeddings are stored in two matrices, one for beginning indexes and another for end indexes.
The global conditioning does not depend on the context words but on all of the fields (g_f) and content words (g_w) available in the infobox. The intention behind including table fields is that people with different occupations will have different fields in their infoboxes, for example, politicians will have a field for their political affiliation while athletes will have a field for the sports team they are playing for.
As mentioned above, infrequent words are dropped from the vocabulary, which

means it will be impossible for the model to generate them. The copy actions are used to deal with these out-of-vocabulary words. Field names are added to words as additional classes so instead of the actual word, delexicalized field name (for example *name_2* instead of *Einstein*) is generated by the model. The authors also offer the dataset they used for experiments. It consists of about 730,000 Wikipedia biography articles.

3.2 Order-planning with hybrid attention

This method [SML+17] also attempts to solve the problem from the previous section. It is based on encoder-decoder architecture. The key component is a dispatcher which based on computed attention decides what will be generated next.
The input to the encoder is the Wikipedia infobox. Field-content pairs are split based on content words (for example *Occupation: writer, politician* turns into *occupation_1: writer, occupation_2: politician*). Both field and content are then embedded into vectors f, c. The embedding matrix is different for field and for content. The two vectors for each row are concatenated, forming the i-th row's representation.

$$x_i = [f_i, c_i]$$

The embedding is then encoded using standard LSTM recurrent neural network.
The dispatcher uses what the authors describe as "hybrid attention", a linear combination of content-based attention and link-based attention. The content-based attention is dependent not only on content embeddings but also on table field embeddings. It is computed as

$$\alpha_{t,i}^{content} = \frac{exp(\alpha_{t,i}^{(f)} \alpha_{t,i}^{(c)})}{\sum_{j=1}^{C} exp(\alpha_{t,j}^{(f)} \alpha_{t,j}^{(c)})}$$

where $\alpha_{t,i}^{(f)}$ and $\alpha_{t,i}^{(c)}$ are attentions of a field and a content word, respectively. These "marginal" attentions are based on words generated in a previous step. The function is basically a softmax over the rows of the table. The idea behind the link-based attention is that the generated words should be in some kind of a preferred order, for example, the name of a person should come before their birth date or the nationality should be generated before occupation. The links are represented as a matrix where each element $a_{i,j}$ contains a probability that the i-th field comes before the j-th field. The link-based attention is obtained by multiplying this matrix with the hybrid attention computed in the previous step, then softmaxing the result again. The matrix is similar to a Markov chain but due to multiple occurrences of a

field is not a probability distribution. The hybrid attention is obtained by linear combination of content-based and link-based attention:

$$\alpha_t^{hybrid} = z_t \alpha_t^{content} + (1 - z_t)\alpha_t^{link}$$

where z_t is a coefficient based on the previous state of decoder RNN, previously generated word and a sum of field embeddings.
The input to the decoder module is

$$x_t = tanh(W_d[a_t; y_{t-1} + b_d])$$

where W_d and b_d are weights, a_t is dot product of table encoding h_t and attention vector α_t and y_{t-1} is the word generated in the previous step. The neural network then transforms this input x_t into output h_t which is then used in a standard linear layer to compute the score. The score is tweaked by copy mechanism which helps with dealing with unseen words. This mechanism is basically an additional scoring function which computes the likelihood that the content word will be a part of the target output. The score from the linear layer and from the copy mechanism are added and softmaxed, resulting in a probability vector of each word. The objective function used in training is negative log likelihood of a sentence based on this probability.

3.3 Lexicalized and delexicalized data

As the title suggests, this system [SHS+16] takes into consideration both the lexicalized (values) and delexicalized (fields) parts of structured data. It is based on encoder-decoder architecture as well. The authors call their model "lexicalized delexicalized semantically controlled LSTM". They applied it to the task of generating dialogue for making a reservation in a restaurant.
The field-value pairs are transformed into a vector in the following way: the field is encoded into a one-hot vector. The value is translated into pre-trained word embeddings. If the field contains multiple words, mean of the embedding is used. The vectors for field and value are then concatenated and the result is used as an input for the encoder, which is a bi-directional LSTM neural layer.
The decoder is based on sc-LSTM which contains a "dialogue act vector". Here, this vector acts as a memory of which dialogue acts need to be included in the output sentence. The encoder output serves as an initialization for the decoder hidden state and memory cell. On the input of the decoder is the embedding of the word generated in the previous timestep. The output of the decoder is either a word or a delexicalized field name. The final sentence is produced using beam search.

Chapter 4

Implementation

We based our system on the system proposed by [LGA16]. The system is composed of three parts: data preprocessor, neural network and decoder. The task of the neural network is to generate a word based on n previous words and the structured table. This way, it learns the probabilities of the language model. The decoder can then use this model to generate natural language sentences word-by-word. The shape and processing of data required by the system is described in the section 4.1. The neural network architecture is shown in section 4.2.1. In sections 4.2.2 and 4.2.3 we explain how this data is processed into a neural network input and output. Finally, in section 4.3 we describe how the sentences are inferred and section 4.4 contains brief documentation of implementation code.

4.1 Preprocessing

4.1.1 Structured data

The system accepts the structured data in the shape of a table of field-value pairs. If a field contains more than one word, the field is split into as many fields as there are words in it and these new fields are numbered with indexes of the words. Fields that contain only one word are numbered with 1.

Field	Value
address	Main Street 16
food	pizza

Field	Value
address_1	Main
address_2	Street
address_3	16
food_1	pizza

Table 4.3: Transformation of field-value pairs to format acceptable by the system

4.1.2 Natural language data

For training, the system requires a set of sentences in natural language which are based on the structured data. From them it learns the parameters of the neural language model.

The restaurant at Main Street 16 serves pizza .

Figure 4.1: Sentence based on the data shown in table 4.3

4.1.3 Processing

All words are lowercased and sentences split into space-separated lists of words and punctuation. All numbers are replaced with a special token and the same thing is done with years, but with a different token. To make the number of possible outputs of the model smaller, we limit the vocabulary to V words which are most frequent in the training set. This means that certain words are replaced with an 'unknown' token. If an out-of-vocabulary word appears in the table, it is replaced with a name of the field where it occurs instead.
We are not interested in rare fields so we drop all those that do not occur at least f times.

4.2 Neural language model

4.2.1 Architecture

The neural network has a standard feed-forward architecture used usually for classification with one hidden layer and an output layer. The hidden layer's activation function is hyperbolic tangent and the output layer's activation function is softmax, which is commonly used for classification.

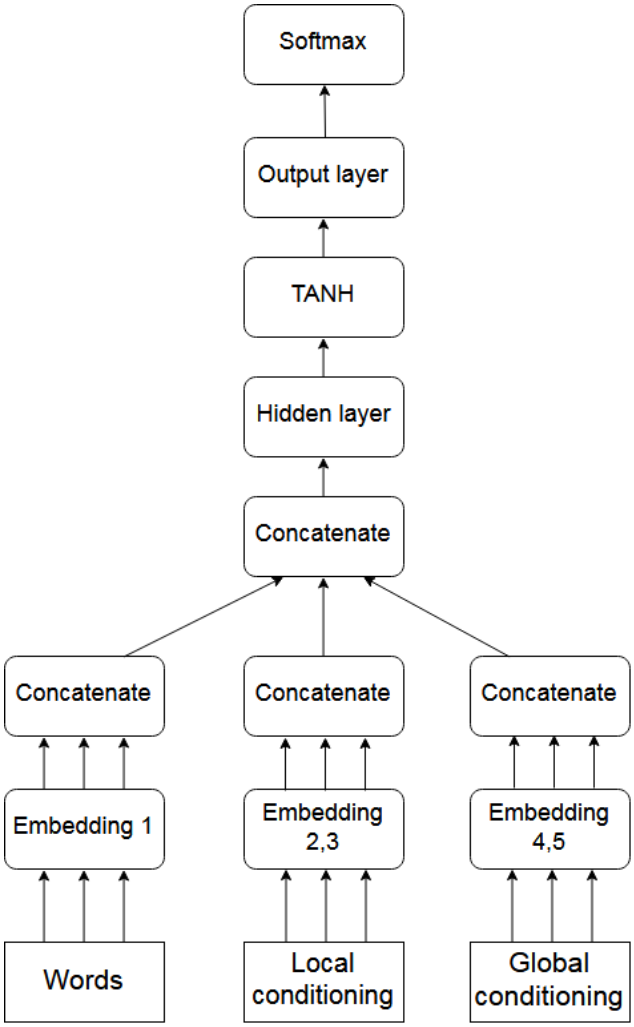


Figure 4.2: Neural network architecture

The function which is optimized during the training phase is negative log

likelihood of a sentence.

$$\min -\sum_{t} \log P(w_t|c_t, l_s, l_e, g_f, g_w)$$

Therefore, one training batch consists of one sentence. Each batch has a different length from the others due to this.

4.2.2 Input

There are three types of input to the neural network: context words, local conditioning, and global conditioning.
The context words are n previous words in a sentence. On the input, they are represented as fixed length vectors of dimension d. The vectors are trained simultaneously to the main model. They can be initialized randomly or with pre-trained vectors. We experimented with random initialization and with using vectors trained with FastText[JGBM16].
The local conditioning is based on the context words as well as on the structured table. First, considering all the tables from training dataset, fields that appear at least f times are chosen and encoded, obtaining F unique fields. Then, for each of the context words, all its occurrences in the table are listed - in which fields and on what index they appear. The index is capped at L - words with higher indexes are not discarded but their index is lowered. Combining the field encoding and the index, we obtain a number representing

Field	**Value**
known_for_1	General
known_for_2	relativity
...	...
known_for_16	Brownian
known_for_17	motion

Value	**Field**	**Start**
General	known_for	1
relativity	known_for	2
...	...	...
Brownian	known_for	10
motion	known_for	10

Table 4.6: Example: from infobox (left), local conditioning is obtained (right), index is capped to L (in this case $L = 10$)

'position' of the context word in the table. As a word can appear in multiple fields, each of the context words is assigned a list of these numbers which has length at most W (based on the word which appears in the most fields in a certain table from the training dataset). This serves as the address into an embedding matrix, which has $F \cdot L \times W \times d$ elements. Note that if we obtain an embedding for a context word using this matrix, it would be a 2-dimensional vector (of dimension $W \times d$) while we need a 1-dimensional one (which is also of size d, same as context word embeddings) for the input. The reshaping is done by choosing maximal values of features over W. So far, when discussing indexes in fields, we've been considering only indexes from

the beginning of the field. Another tweak to the model is considering also the indexes from the end of the field. The idea behind this modification is that low "end-index" indicates that the word ends with one piece of information and another one should come next.

Field	Value
birth_date_1	12
birth_date_2	May
birth_date_3	1967

Value	Field	Start	End
12	birth_date	1	3
May	birth_date	2	2
1967	birth_date	3	1

Table 4.9: Example: from table (left), local conditioning is obtained (right)

The global conditioning disregards the context words completely and is based solely on the table. This means that all words in one sentence share this information. There are two possibilities: Field conditioning and word conditioning. Both simply take a set of fields or a set of words from field values and encode each member of these sets into embeddings. For words, different embedding matrix is used. Therefore, the dimension of these global embeddings can differ from d and is noted as g.
To sum it up, from the context words we obtain n embedding vectors of dimension d, from local conditioning $2n$ vectors having also dimension d and from global conditioning two vectors of dimension g. All of these are concatenated to form a vector x

$$x \in \mathbb{R}^{3 \cdot n \cdot d + 2 \cdot g}$$

which serves as the input to the neural network.

4.2.3 Output

As the system approaches the task basically as a classification problem, on the output we want one word to be chosen from the vocabulary as the next one in the sentence. However, as our vocabulary is limited and it is possible that certain words will not be seen during the training phase at all, we need to find a way to include them. The solution lies in the structured data, that is, the table. In section 4.2.2, we describe that we keep F fields that occur at least f times. We append those common fields to the vocabulary so that they are included in the output. For example, even if the model does not know the word *Einstein*, it can still generate the field name *name_2*, which is during inference substituted with the correct value from the table. This mechanism is called copy action by [LGA16] and was inspired by work of [LPM15].

4.3 Decoding

Decoder is used after the training ends to generate the natural language sentences. The input to the decoder is just the structured data table. The sentence is initialized with a sequence of n starting tokens. An algorithm called beam search is utilized to find the best sentence.

4.3.1 Beam search

If we want to find the most likely sentence, we cannot simply choose the best word in each step, but searching the whole state space and keeping all possible sentences could be inefficient. As a compromise, we use a heuristic known as beam search [DLP16]. This method is based on breadth-first search. Using a state space search analogy, it expands all current nodes but keeps only b most likely candidates (where b is the beam size). In our system, we compute a score for all words in each step for every unfinished sentence, then choose b sentences with highest scores.

4.4 Code documentation

The implementation was done in Python 3. For constructing and training the neural network, Keras library[C$^+$15] with Tensorflow[AAB$^+$15] backend was used. The system is split into several scripts:

- **config.py** - stores all parameters for other scripts, mainly for model training.
- **data_loader.py** - contains methods for loading both structured data and natural language sentences
- **data_process.py** - processes the data as described in section 4.1.3 and stores the output
- **main.py** - creates training dataset based on the data created in the previous step, then constructs and trains the neural network
- **testing.py** - serves for testing the trained model using beam search

Chapter 5

Experiments

5.1 Experiment description

5.1.1 Task

The task on which we perform the experiments is generating a sentence given a set of key-value pairs. We have two datasets and the exact goal slightly differs between them; details are explained in the respective sections.

5.1.2 Data

The first dataset we used consists of about 730,000 articles from Wikipedia. They are exclusively biography articles, that is, articles that describe lives and deeds of famous or notable people. The dataset is divided into training (80%), validation (10%) and testing (10%) splits. The information provided is infobox contents in structured form, first several sentences of the article and their count, URL, article IDs and a list of contributors, each of these in a separate file. The key components to our system are the infobox and the first sentence of each article, since the first serves as an input, be it for training or testing, and the other as an example output for training which our system will attempt to replicate. The infoboxes are stored as tab-separated pairs

containing a key and a value, where the key is a field name with appended index and value is a word or punctuation mark appearing in the field at the index. There is one article infobox per line. For natural language sentences, there is one sentence per line (as space-separated words), but there might be several sentences extracted from the beginning of a Wikipedia biography article. However, we are interested only in the very first sentence. The goal is to generate this sentence based on the data contained in the article's infobox.

Field	**Value**
name_1	craig
name_2	starcevich
birth_date_1	16
birth_date_2	May
birth_date_3	1967
debut_date_1	round
debut_date_2	1
debut_date_3	:
debut_date_4	1987

Table 5.1: Example table from Wikipedia dataset.

The other dataset was compiled from a goal-based conversational system whose task was to find and recommend a restaurant based on user's requirements. It was constructed by [WGM+16]. We filtered and transformed the original dataset to match the format of the Wikipedia biography data. After this processing, there are 3 307 sentences, divided into training (60%), validation (20%) and testing (20%) splits. The goal is to generate an utterance based on several pieces on information which are available to the system (example in table5.2).

Field	**Value**
name_1	alamo
name_2	square
name_3	seafood
name_4	grill
address_1	803
address_2	fillmore
address_3	street

Table 5.2: Example table from the restaurant dataset.

5.1.3 Training environment

The experiments were performed on two virtual machines provided by Amazon Web Services (AWS)[1]. Both of them used Linux Ubuntu 16.04 from Deep Learning template - this template comes with pre-installed commonly used machine learning frameworks and libraries such as Tensorflow, Theano and Torch. The hardware parameters are listed in tables 5.3 and 5.4. The hardware is not the same but since the operating system and all software is identical, the difference should have no impact on testing results.

Parameter	**Value**
GPU name	Nvidia Tesla K80
GPU memory (GB)	12
CPU name	Intel Xeon E5-2686
number of CPU cores	4
CPU frequency (GHz)	2.30
RAM (GB)	64

Table 5.3: Hardware parameters of the first instance

Parameter	**Value**
GPU name	Nvidia Tesla K80
GPU memory (GB)	8
CPU name	Intel Xeon E5-2686
number of CPU cores	16
CPU frequency (GHz)	2.30
RAM (GB)	122

Table 5.4: Hardware parameters of the second instance

5.1.4 Experiment variables

The experiment variables can be split into two groups: boolean and numerical. The first group consists of certain components of the system which might be used in data processing and model training but also can be skipped or disregarded. Two of these are the global and local conditioning of the model, which serve to augment the model with additional information. By testing a model trained with these turned off, we can measure how great improvement they bring, if any. In all experiments, at least one of them needs to be included since otherwise the structured data would be disregarded and all

[1] https://aws.amazon.com/

generated sentences would be the same. Another boolean parameter which modifies the data processing is removing the punctuation from the natural language sentences. As the punctuation is quite common, removing it might help the model focus on the more rare words.
The first numerical variable is the size of the vocabulary. With lowering the size, more words in sentences get replaced with field names and the sentences might become more general. This might help the model to generalize. Another hyperparameter is the size of the context n. The higher this value becomes, the better should the model perform, but lower values might speed up the training process. The variables which come from the neural network hyperparameters are learning rate and number of epochs. The last variable is used during inference, and it is the beam size in beam search. All of the hyperparameters are summarized in tables 5.5 and 5.6 for boolean and numerical variables respectively. The values which are experimented on are in the "Values" column. Default values, which are fixed whenever we experiment on the others, are highlighted in bold.

Name	**Values**
Local, global cond.	True/True, **False/True**, True/False
Remove punctuation	True, **False**

Table 5.5: Boolean hyperparameters

Name	**Values**
Vocabulary size	5 000, 10 000, **20 000**
n	5, **10**, 15
beam size	5,**10**,15,20

Table 5.6: Numerical hyperparameters

In table 5.7, we show hyper parameters that were fixed during training. All of them were taken from the original paper [LGA16]. Most of them are hyperparameters of the neural network. Hyperparameters d and g are embedding sizes for local and global conditioning respectively, nhu is the number of units in the hidden layer, α is the learning rate and f is the minimum number of times a table key appears in a dataset to be included.

Name	Value
d	64
g	128
nhu	256
iterations	20
α	0.025
f	100

Table 5.7: Fixed hyperparameters

A question which we would also like to answer is, how the system reacts to differences in the structured data? For example, how will the generated sentence be different if we include person's birth date, name, and occupation and if we include only their name and occupation? The expected result would look like *John Doe, born January 1, 1982, is a British actor*, respective *John Doe is a British actor*, but we have to experiment to find out if the system can generalize that much. Also, what could happen if only a name was inputted into the system?

5.1.5 Metrics

The first metric which we will measure is BLEU[PRWjZ02]. Originally developed for evaluating machine translation models, it is intended to mimic human judgment as close as possible. The key idea is replacing previously used standard n-gram precision with modified n-gram precision. The standard n-gram precision counts all n-grams in candidate sentence which occur at least once in any reference sentence, then divides this count by the length of the candidate sentence. In the modified n-gram precision, MC_{ref},the maximum number of times each candidate n-gram occurs in any reference sentence is computed first. Then, the count of each n-gram in the candidate sentence is clipped to MC_{ref}. These clipped counts are added and divided by candidate sentence length to obtain the result. The improvement is illustrated in the following example.

Candidate: The the the the the the the.
Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.

Figure 5.1: Example candidate and reference sentences.

The candidate translation seen in figure 5.1 is obviously not accurate at all, it is just a very probable word generated over and over. However, due to precision metric imperfection, it achieves score of 1 since all candidate unigrams occur in both reference sentences. $R(w_i)$ is the function that returns 1 if w_i occurs at least in one reference, 0 otherwise.

$$p = \frac{R(w_1) + ... + R(w_7)}{len(s_c)} = \frac{R(the) + ... + R(the)}{7} = \frac{7}{7} = 1$$

This implies a perfect translation which is not the case as (apart from other inconsistencies) many reference words are missing. The modified n-gram precision achieves score of $\frac{2}{7}$:

$$MC_{ref}(the) = \max(2, 1) = 2$$

$$p_1 = \frac{\min(MC_{ref}(the), C_c(the))}{len(s_c)} = \frac{\min(2, 7)}{7} = \frac{2}{7}$$

To get more accurate values, BLEU combines modified n-gram precisions for different values of n. It does so by computing geometric mean (standard mean is a worse fit for this since the decay in values with increasing n is exponential). Another problem BLEU needs to deal with is too long or too short candidate sentences. By design, BLEU already penalizes overly long sentences. For candidate sentences which are too short, the authors introduce a brevity penalty, which is 1 if the reference and candidate have the same number of words, and increases with decreasing candidate length. The authors suggest computing the brevity penalty over the whole corpus instead of sentence by sentence in order to allow the model some freedom. The brevity penalty (BP) is

$$BP = \begin{cases} 1, if\ c > r \\ e^{(1-r/c)}\ otherwise \end{cases}$$

where c is candidate length and r is reference length. The final BLEU formula is

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

where p_n are the modified n-gram precisions and w_n are weights. The reason for using BLEU in our experiments is that our system outputs a candidate sentence which we compare to a reference sentence, similarly to machine translation. However, the resulting BLEU values might appear low since the metric can improve with a higher number of reference sentences, but we have only one. We examine modified n-gram precisions with n ranging from 1 to 4 as well as their combination (with uniform weights) - total BLEU.
The other metric which we use is perplexity. The perplexity of language model $q(X)$ over a sequence of length N is defined as

$$2^{-\frac{1}{N}\sum_{i=1}^{N} log_2 q(X_i)}$$

Perplexity[JM00] is indirectly proportional to conditional probability and therefore we desire to minimize it since that maximizes the probability. Perplexity can be interpreted as a weighted average branching factor, that is, the number of words which could come next in the sequence. The lowest value of perplexity is therefore 1 and it has no upper bound.

5.2 Experiment results

5.2.1 Results

Variable	BLEU	Perplexity
default	15.1%	1.0
Local only	**23.1%**	**1.13**
Local + global	0.05%	1.0
Remove punctuation	16.1%	1.0
5k words	20.7%	1.25
10k words	21.9%	1.32
5-gram	19.2%	1.0
15-gram	15.1%	1.0
5 beam	22.6%	1.33
15 beam	22.3%	1.31
20 beam	22.3%	1.31

Table 5.8: Experimenting on Wikipedia dataset, emboldened are the best (meaningful) results.

Variable	BLEU	Perplexity
default	27.9%	**1.16**
Local only	26.9%	1.30
Local + global	26.5%	1.26
Remove punctuation	**28.2**%	1.19
5-gram	27.3%	1.20
15-gram	27.9%	**1.16**
5 beam	27.8%	1.19
15 beam	27.8%	1.19
20 beam	27.8%	1.19

Table 5.9: Experimenting on restaurant dataset, emboldened are the best results.

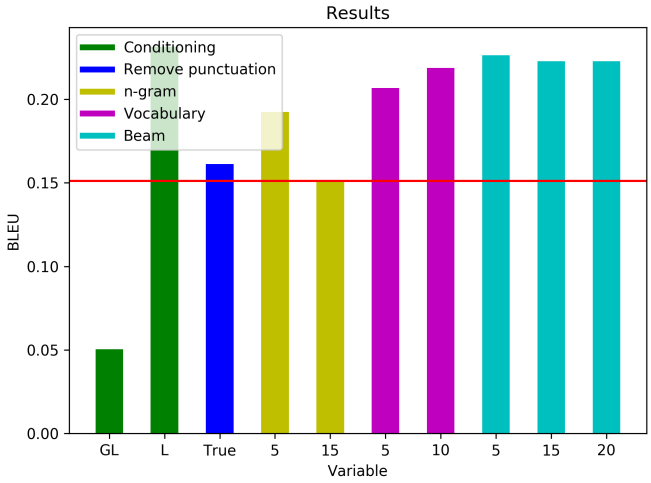


Figure 5.2: Plot showing BLEU results on Wikipedia dataset. Result of model with default parameters is represented by horizontal line.

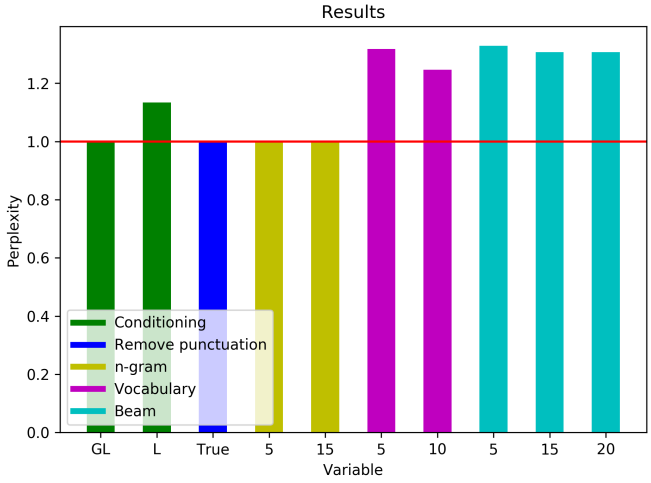


Figure 5.3: Plot showing perplexity results on Wikipedia dataset. Result of model with default parameters is represented by horizontal line.

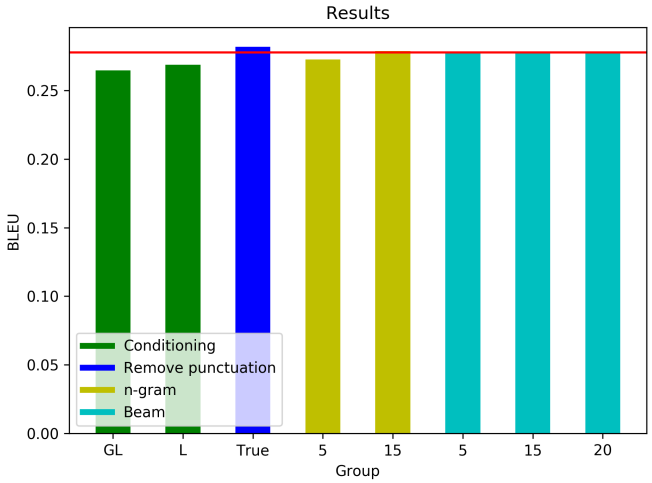


Figure 5.4: Plot showing BLEU results on restaurant dataset. Result of model with default parameters is represented by horizontal line.

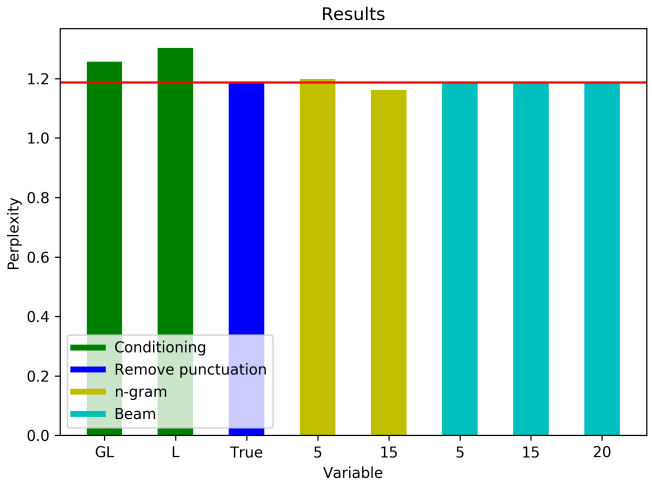


Figure 5.5: Plot showing perplexity results on restaurant dataset. Result of model with default parameters is represented by horizontal line.

5.2.2 Analysis

Judging from the low BLEU and perplexity scores, the system was unable to learn the model of the Wikipedia sentences in most training settings. Only the model containing just the local conditioning produced a little better results than the rest. It appears that some sort of error occurred during training and the network overfit the model.
Experiments with vocabulary were not performed on the restaurant dataset

due to its small size which resulted in the vocabulary containing around only 700 words. The best result was achieved by the model which disregarded punctuation in the original training sentences. Since punctuation appears often in the sentences, removing it likely caused the probability to be distributed more evenly over words and improving performance. The perplexity is quite low in all cases, which hints at training set overfitting. Modifying the beam size had no effect on the result.

5.2.3 Human evaluation

In this section, we compare sentences generated by various models from the same table.

Field	Value
name_1	miroslav
name_2	popov
nationality_1	cze
nationality_2	czech
birth_date_1	14
birth_date_2	june
birth_date_3	1995
birth_place_1	dvur
birth_place_2	kralove
birth_place_3	nad
birth_place_4	labem
birth_place_5	,
birth_place_6	czech
birth_place_7	republic
article_title_1	miroslav
article_title_2	popov

Table 5.10: A table from the testing split of the Wikipedia dataset

Sentence
of of of of of of of of of of of of of of of of of of of of
is is is is is is is is is is is is is is is is is is is is
popov “ “ ” ” ” , <UNK> -rrb- , and , <UNK> , <UNK> , , and argentinian <UNK>
14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
-lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb- -lrb-
is is is is is is is is is is is is is is is is is is is is
miroslav fullname_2 is an american film actor , who is currently playing for the <UNK> <UNK> of the university
bernhard popov name_3 -lrb- born june 14 , <UNK> -rrb- is an american singer and film actor , best known

Table 5.11: Sentences generated by various models on one of the tables from Wikipedia dataset.

Field	Value
name_1	red
name_2	door
name_3	cafe
phone_1	4152828283
match_1	yes

Table 5.12: A table from the testing split of the restaurant dataset

Sentence
red door cafe is a nice restaurant that allows kid -s and are good for breakfast and is located at
red door cafe is a good place for dinner and their phone number is 4152828283
colibri mexican bistro is moderate -ly priced restaurant that allows kid -s in the hayes valley area
red door cafe phone number is 4152828283 and allows child -s and is good for breakfast and allows child -s
red door cafe is a nice restaurant that is between price_2 and price_4 phone number of red door cafe is 4152828283
red door cafe is a nice restaurant that is good for breakfast cafe ’s phone number is 4152828283

Table 5.13: Sentences generated by various models on one of the tables from restaurant dataset.

The table 5.11 confirms that the models were not trained successfully. In most of the cases, the sentence consists only of the most probable word which

is repeatedly generated. Here, an increase of the beam size slightly improves the results, but there is still no coherent sentence.
In table 5.13 we see sentences generated by different models based on a table 5.12 from restaurant dataset. The order of the models is the same as in the result table (the beam sizes are excluded). The generated sentences are basically grammatically correct, which means that the core n-gram language model learned its parameters quite well. However, it can be observed that there are discrepancies between the sentences and the structured data. On the other hand, the generated sentences are not the same and therefore the features extracted from the structured data must have some influence on the output (if they had not, all sentences would be the same since they are initialized in the same way). The low perplexity again hints at overfitting the model on the training data. The name is correctly included in all of them except the third one which is wrong altogether. That model works only with local conditioning and context (no global conditioning) but this resulted in all sentences in the testing set being almost the same. The key piece of information, the phone number, is included in four sentences. However, all the sentences contain redundant information (in some cases even repeated), which is undesired as it might be incorrect. In the fifth sentence, delexicalized tokens *price_2* and *price_4* appear because they were not in the source table and could not be swapped with words.

5.3 Testing on subsets of information

Since the system failed to produce a model with any meaningful results over the Wikipedia dataset, we will include only the restaurant data in this section. As the input to be experimented on we chose a table containing information about a "Red Door Cafe", one over which all models performed rather well.

Field	**Value**
name_1	red
name_2	door
name_3	cafe
address_1	1608
address_2	bush
address_3	street
match_1	yes
sentence	the address of red door cafe is 1608 bush street

Table 5.14: First table and sentence from the restaurant dataset.

The first example is just a name and address, which the system handles without fail. The sentence is grammatically correct and contains all provided information. It is shown that the system learned to provide the construct *The address of X is* to the beginning of the sentence when an address is provided in the table.

Field	**Value**
name_1	red
name_2	cafe
address_1	1608
address_2	bush
address_3	street
phone_1	4152828283
match_1	yes
sentence	the address of red door cafe is 1608 bush street and the phone number is 4152828283

Table 5.15: Second table and sentence from the restaurant dataset. Note that the name is incomplete

In the second example, we removed a part of the establishment's name. Despite that, the system generated the name in full. This was likely due to the language model overruling the structured data conditioning. The added piece of information (phone number) was appended successfully to the end of the sentence.

Field	**Value**
name_1	red
name_2	door
name_3	cafe
kidsallowed_1	no
goodformeal_1	brunch
match_1	yes
sentence	red door cafe is good for brunch and does not allow kid -s

Table 5.16: Third table and sentence from the restaurant dataset.

The *-s* token here is a modifier which indicates that the previous word should be transformed to plural form. Again the system performs well, correctly stating the meal and adding a negative form.

Field	Value
name_1	red
name_2	door
name_3	cafe
goodformeal_1	brunch
pricerange_1	cheap
match_1	yes
sentence	red door cafe is cheap good and does not allow kid -s

Table 5.17: Fourth table and sentence from the restaurant dataset.

In the next example, the system runs into a bit of trouble. The expression *cheap good* might look wrong but this might be just missing a comma as this dataset does not contain punctuation. However, the information about the meal is not included in the sentence and the restaurant is only described as "good". On top of that, information about children which does not appear in the table is generated. Only correct parts are the name and the information about price.

Field	Value
name_1	red
name_2	door
name_3	cafe
near_1	lower
near_2	pacific
near_3	heights
match_1	yes
sentence	red door cafe is near the lower pacific heights is moderate -ly priced

Table 5.18: Fifth table and sentence from the restaurant dataset.

The name and location are provided correctly but there is redundant, grammatically wrong and possibly incorrect mention of price. The reason this happened is probably the way of generating the sentences - the beam search is terminated when the candidates reach a fixed length, which sometimes results in sentences longer than needed.

Chapter 6

Conclusion and future work

We researched currently used approaches for natural language generation. We chose one of the methods and implemented a natural language generation system based on it. We experimented with the system over two datasets, Wikipedia biography articles and dialogues focused on restaurant reservation. The system, when using the best model, reached 23.1 average BLEU on the first dataset and average 28.2 BLEU on the second dataset. We failed to reproduce the BLEU of 34.7 from the original paper but in many cases the output is correct, as illustrated by the following examples.

Field	**Value**
name_1	red
name_2	door
name_3	cafe
address_1	1608
address_2	bush
address_3	street
match_1	yes
sentence	the address of red door cafe is 1608 bush street

Table 6.1: First example of correct sentence from the restaurant dataset.

Field	Value
name_1	red
name_2	door
name_3	cafe
kidsallowed_1	no
goodformeal_1	brunch
match_1	yes
sentence	red door cafe is good for brunch and does not allow kid -s

Table 6.2: Second example of correct sentence from the restaurant dataset.

The evaluation by hand showed that while the system is able to learn the language model and generate coherent sentences, there is usually erroneous or redundant information included. Further experimentation and changes to the implementation are probably required. Once these issues are solved and the performance improves, the next step is integrating the system into the socialbot Alquist, possibly trained on data collected from its previous chat sessions with its users.

Appendix A

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, Software available from tensorflow.org.

[BdM+92] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai, *Class-based n-gram models of natural language*, Comput. Linguist. **18** (1992), no. 4, 467–479.

[BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin, *A neural probabilistic language model*, J. Mach. Learn. Res. **3** (2003), 1137–1155.

[C+15] François Chollet et al., *Keras*, `https://keras.io`, 2015.

[CFDGBT09] Michael C Frank, Noah D Goodman, and Joshua B Tenenbaum, *Using speakers' referential intentions to model early cross-situational word learning*, 578–85.

[Dal89] Robert Dale, *Cooking up referring expressions*, Proceedings of the 27th Annual Meeting on Association for Computational Linguistics (Stroudsburg, PA, USA), ACL '89, Association for Computational Linguistics, 1989, pp. 68–75.

[DLP16] Suranjan De and Anita Lee-Post, *Performance analysis of beam search with look ahead*, Journal of computing and information technology **5(4)** (2016), 136 – 140.

[DR95] Robert Dale and Ehud Reiter, *Computational interpretations of the gricean maxims in the generation of referring expressions*, CoRR **cmp-lg/9504020** (1995).

[GK18] Albert Gatt and Emiel Krahmer, *Survey of the state of the art in natural language generation: Core tasks, applications and evaluation*, J. Artif. Intell. Res. **61** (2018), 65–170.

[Hay98] Simon Haykin, *Neural networks: A comprehensive foundation*, 2nd ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov, *Bag of tricks for efficient text classification*, CoRR **abs/1607.01759** (2016).

[JM80] F Jelinek and Robert Mercer, *Interpolated estimation of markov source parameters from sparse data.*, 381–397, 401.

[JM00] Daniel Jurafsky and James H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, 1st ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[JMRS91] F. Jelinek, B. Merialdo, S. Roukos, and M. Strauss, *A dynamic language model for speech recognition*, Proceedings of the Workshop on Speech and Natural Language (Stroudsburg, PA, USA), HLT '91, Association for Computational Linguistics, 1991, pp. 293–295.

[JW05] Pamela W. Jordan and Marilyn A. Walker, *Learning content selection rules for generating object descriptions in dialogue*, J. Artif. Intell. Res. **24** (2005), 157–194.

[KJ13] Dinesh Kumar Kashyap and Gurpreet Singh Josan, *A trigram language model to predict part of speech tags using neural network*, Intelligent Data Engineering and Automated Learning – IDEAL 2013 (Berlin, Heidelberg) (Hujun Yin, Ke Tang, Yang Gao, Frank Klawonn, Minho Lee, Thomas Weise, Bin Li, and Xin Yao, eds.), Springer Berlin Heidelberg, 2013, pp. 513–520.

[LGA16] Rémi Lebret, David Grangier, and Michael Auli, *Neural text generation from structured data with application to the biography domain*, Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2016, pp. 1203–1213.

[Lid20] G. Lidstone, *Note on the general case of the Bayes–Laplace formula for inductive or a posteriori probabilities.*, Transactions of the Faculty of Actuaries **8** (1920), 182–192.

[LOW12] Gennadi Lembersky, Noam Ordan, and Shuly Wintner, *Language models for machine translation: Original vs. translated texts*, Comput. Linguist. **38** (2012), no. 4, 799–825.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning, *Effective approaches to attention-based neural machine translation*, CoRR **abs/1508.04025** (2015).

[PC98] Jay M. Ponte and W. Bruce Croft, *A language modeling approach to information retrieval*, Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (New York, NY, USA), SIGIR '98, ACM, 1998, pp. 275–281.

[PH08] Bo-June Paul) Hsu, *Generalized linear interpolation of language models*, 136 – 140.

[PRWjZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu, *Bleu: a method for automatic evaluation of machine translation*, 2002, pp. 311–318.

[Ros58] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review (1958), 65–386.

[RPK+18] Ashwin Ram, Rohit Prasad, Chandra Khatri, Anu Venkatesh, Raefer Gabriel, Qing Liu, Jeff Nunn, Behnam Hedayatnia, Ming Cheng, Ashish Nagar, Eric King, Kate Bland, Amanda Wartick, Yi Pan, Han Song, Sk Jayadevan, Gene Hwang, and Art Pettigrue, *Conversational AI: the science behind the alexa prize*, CoRR **abs/1801.03604** (2018).

[SHS+16] Shikhar Sharma, Jing He, Kaheer Suleman, Hannes Schulz, and Philip Bachman, *Natural language generation in dialogue using lexicalized and delexicalized data*, CoRR **abs/1606.03632** (2016).

[SML+17] Lei Sha, Lili Mou, Tianyu Liu, Pascal Poupart, Sujian Li, Baobao Chang, and Zhifang Sui, *Order-planning neural text generation from structured data*, CoRR **abs/1709.00155** (2017).

[WGM+16] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young, *Multi-domain neural network language generation for spoken dialogue systems*, Proceedings of the 2016 Conference on North American Chapter of the Association for Computational Linguistics (NAACL), Association for Computational Linguistics, June 2016.

[WJPJ74] Paul Werbos and Paul J. (Paul John, *Beyond regression : new tools for prediction and analysis in the behavioral sciences /*.

Appendix B

Code manual

This section describes the usage of the system implementation. The implementation is done in Python 3 and requires the following libraries: Numpy, Tensorflow, Keras, and NLTK. First, parameters need to be specified in the script *config.py*. To process the input and generate intermediate data, run *data_process.py*. This creates a folder labeled with hash encoding selected parameters. The hash needs to be specified on the input of the next script, *main.py*. In it, the model is trained and saved with an extended hash (additional parameters are included). Finally, you can test the model by running *testing.py* with the model name as a parameter, which saves the generated sentences to a file and outputs the resulting BLEU and perplexity values.

Appendix C

CD contents

- natural_language_generation_from_structured_data.pdf - text of this thesis
- natural_language_generation_from_structured_data.zip - source code of this thesis in LaTeX
- StructuredDataNLG.zip - source code of implementation in Python and data from the restaurant dataset