FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Question Answering Algorithms in Natural Language
Student:	Matúš Žilinec
Supervisor:	doc. Ing. Pavel Kordík, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

Survey state of the art algorithms for question answering. Focus on natural question answering and recently released NaturalQuestions dataset. Explore and describe the BERT architecture, evaluate its performance on the NQ dataset and compare it to different machine learning approaches. Examine particular errors and describe the limitations of current algorithms. Explain the findings in the context of theoretical expectations.

References

1) Natural Questions, A Benchmark for Question Answering Research. Available at https://ai.google.com/research/NaturalQuestions/.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 11, 2019

Bachelor's thesis

Question Answering Algorithms in Natural Language

Matúš Žilinec

Department of Applied Mathematics
Supervisor: doc. Ing. Pavel Kordík, Ph.D.

May 15, 2019

Acknowledgements

First of all, I would like to express my appreciation to doc. Ing. Pavel Kordík, Ph.D., Ing. Stanislav Kuznetsov and everyone at the Data Laboratory for their invaluable help throughout my studies. I would also like to extend my deepest gratitude to my family for their unwavering support and patience.

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.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 15, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Žilinec, Matúš. *Question Answering Algorithms in Natural Language*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019. Also available from: ⟨`https://zilinec.me/pdf/bachelors-thesis.pdf`⟩.

Abstrakt

Tato bakalářská práce zkoumá nejmodernější algoritmy pro strojové odpovídání na dotazy v přirozeném jazyce se zaměřením na porozumění textu a modely založené na hlubokém učení. Architektura transformer je prozkoumána a vyhodnocena na nově vydaném datasetu NaturalQuestions. Práce analyzuje konkrétní chyby a omezení současných algoritmů a zabývá se jejich možnými vylepšeními.

Klíčová slova strojové odpovídání na dotazy, porozumění textu, strojové učení, neuronové sítě

Abstract

This bachelor's thesis surveys state-of-the-art algorithms for natural language question answering, focusing on machine reading comprehension and deep learning based models. The transformer architecture is explored and evaluated on the newly released NaturalQuestions dataset. The thesis analyzes particular errors and limitations of current algorithms and discusses their possible improvements.

Keywords question answering, machine reading comprehension, machine learning, neural networks

Contents

List of Figures

List of Tables

Introduction

Question answering is an important topic in the intersection of the fields of artificial intelligence and natural language processing. The ability to answer questions posed in natural language is a necessary prerequisite to building intelligent systems capable of having a meaningful conversation with a human. Since the advent of computing technology, most of human-computer interaction has been realized using graphical user interfaces, which is often an unnatural and ineffective communication channel for humans. Lately, question answering has become increasingly popular in commercial applications, including chatbots, voice assistants (such as Alexa), and web search. Given the exponential increase in data published on the internet, I believe that in the near future, question answering will be vital in the everyday use of computing technology.

The goal of this thesis is to survey state of the art machine learning based algorithms for question answering in natural language (focusing on recent advances in machine reading comprehension), to analyze their performance and limitations, and to use such an algorithm in practice. The thesis consists of five chapters. In chapter 1, I summarize different aspects of question answering and describe the required theoretical background. Then, in chapter 2, I describe different question answering datasets and neural network based models. In chapter 3, I explore the NaturalQuestions dataset and implement a question answering system. I evaluate the performance of this system and discuss its limitations in chapter 4. Finally, I deploy the trained question answering system as a web application in chapter 5.

Chapter 1

Theoretical background

1.1 Question answering

Question answering (abbreviated QA) is one of the oldest topics in computer science. Since the invention of the computer, researchers have been trying to build computer systems able to communicate with humans and answer their queries in natural language, most commonly being English. The key difficulty of question answering lies in the processing of language and transforming the knowledge represented in it into machine readable form, which is a non-trivial task due to the language's complexity and ambiguity.

We can classify present question answering systems into two categories based on the approach they use to produce the answer (Jurafsky et al., 2008). One option is to parse the question into a formal query in a database language and to search a structured database of knowledge. These systems are called **knowledge-based**. The second approach is to extract answers directly from an unstructured source of information, typically plain text documents, using information retrieval and natural language processing algorithms. This method is called **information retrieval-based**.

In this chapter, I will describe these two paradigms in more detail. After a brief introduction of the required machine learning background, I will focus on the recent progress in retrieval-based QA.

1.1.1 Knowledge-based QA

Knowledge-based QA deals with answering questions by reasoning over information stored in a structured knowledge base. This may be a relational SQL database or, more commonly, a knowledge graph that follows the Resource Description Framework (RDF). In the RDF format, knowledge about the world is represented in the form of triples $(subject, relation, object)$, where

each triple represents a single relation between two real-world concepts. As an example, consider the proposition *"Prague is located in Czech Republic"*. In the language of RDF, we would represent this piece of information with this triple: $(prague, located\_in, czech\_republic)$. We can then execute queries in the SPARQL query language, for example to retrieve all similar objects with the same relation. RDF databases can be thought of as a graph with vertices representing concepts and edges representing relationships between them. Examples of large knowledge graphs used in practice include Google's Freebase, DBpedia or Wikidata. (Lassila et al., 1999; Guu et al., 2015)

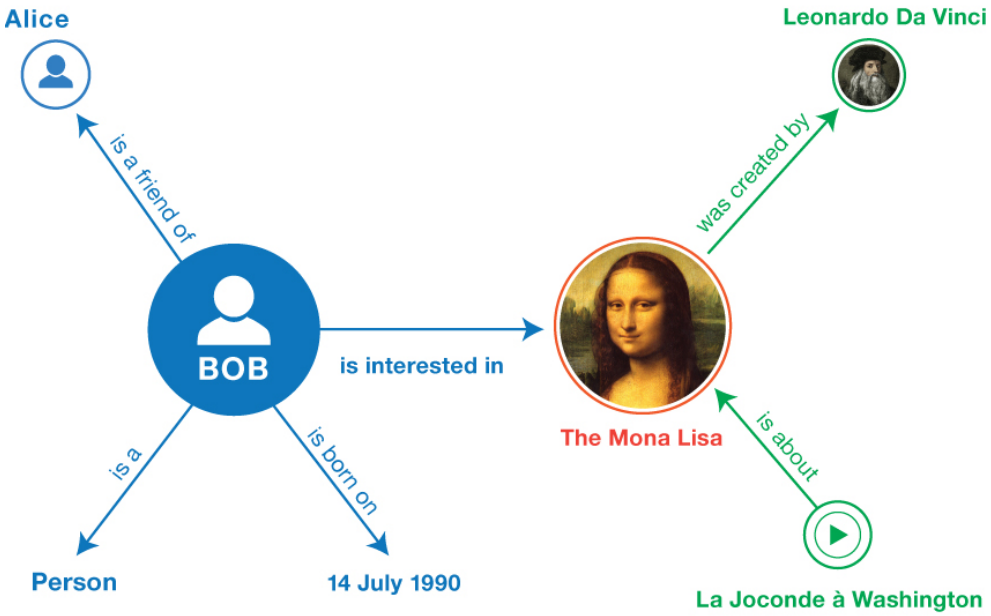

Figure 1.1: RDF knowledge graph

Source: https://www.w3.org/TR/rdf11-primer

The answering process usually involves mapping the question into a formal database query, executing this query in the database engine and processing its result to generate an answer. For example, Mohammed et al. (2018) have proposed a baseline system consisting of 4 steps:

- entity detection,
- entity linking,
- relation prediction,
- evidence integration.

In the first step, entity detection, the system scans the input and identifies the entity being queried as a substring of the question. This substring then has to be linked to a specific vertex in the knowledge graph. The next step is to find the potential relationship being queried. Finally, the system selects n top entities and relations and returns a score for each of these results. Another interesting approach, the Neural SPARQL Machine (Soru et al., 2017), uses recurrent neural networks to directly translate natural language questions to SPARQL queries.

Knowledge-based QA systems have several notable advantages. Firstly, because they use optimized databases, they are very computationally effective and scale well to large amounts of data. They are also well interpretable and explainable, as it is easy to verify what logical steps the algorithm took to answer the question.
These systems work well on factoid questions, questions answerable by an objective fact, such as "What is the speed of light in vacuum?", or "What is the highest mountain in Slovakia?", but fail on more complex non-factoid questions such as "When are hops added to the brewing process?" or "What does the word china mean in chinese?", that can't easily be represented by a knowledge graph. There are also other drawbacks to knowledge-based systems. Most importantly, the requirement that the information is represented in a structured database requires a lot of laborous human annotatation. Furthermore, the knowledge base needs to be periodically updated so that the system can provide up-to-date information.

1.1.2 Information retrieval-based QA

Knowledge-based QA systems can only answer questions about a particular domain for which a knowledge base exists. To be able to answer open-domain questions, a QA system needs large corpora of data from multiple domains. Usually, this data will be downloaded from multiple sources and stored in different formats. Furthermore, most of information on the web is present in unstructured formats, such as HTML pages, text documents or PDF files. It would be impractical, if not outright impossible, to manually convert all of this data to knowledge bases. Thus we need a way of processing unstructured documents and extracting knowledge from them without manual annotation.

Retrieval-based QA systems are able to answer natural language questions without a formal representation of the required knowledge. Instead, they are only given a corpus of text written in natural language, which they process and extract facts required to answer the question. This can range from string matching and extracting simple factoid answers to multi-step reasoning and answer text generation. (Elworthy, 2000; Ittycheriah et al., 2001; Weston et al., 2015)

Most present retrieval-based systems contain core components for **document retrieval** and **answer extraction**. In the document retrieval step, the system must choose candidate documents which are likely to contain information related to the question from a large dataset. This can be done using keyword matching, by ranking document similarities, or using machine learning. The selected documents are subsequently analyzed by an answer extraction component, which then outputs a text answer. Again, this component can be implemented in more ways, such as by simple comparison of strings, using linguistic features or, more recently, with machine learning.

In addition to these core components, a retrieval-based system may use additional modules for question analysis, answer processing and scoring.

1.1.3 General-purpose QA frameworks

Recently, a number of general-purpose QA frameworks have been developed. Examples include IBM Watson (Ferrucci et al., 2013), which won the *Jeopardy!* quiz show in 2011, and YodaQA (Baudiš et al., 2015), an open source QA system that collects open-domain information from structured and unstructured sources freely available on the web. This system uses an assortment of components to process knowledge bases and fulltext data and then ranks and refines possible answers returned from these components.

1.1.4 Machine reading comprehension

Machine reading comprehension is a new topic in question answering, enabled by recent progress in artificial intelligence and availability of datasets. It focuses on the ability of algorithms to read text and answer questions about its content. This task was first proposed by Hirschman et al. (1999), who conducted experiments on a dataset of 600 questions collected from children's reading tests. Clark et al. (2016) argue that machine reading comprehension is a good benchmark for evaluating general progress in AI, since it requires the computer to understand abstract concepts in text and to make conclusions.

In present machine reading comprehension tasks, an algorithm is given a natural-language question along with a paragraph of text (called "context"). The algorithm's goal is to select a span of words from the context that accurately answers the input question, or to indicate that the question is not answerable given any possible span of words from this context.

1.2 Evaluation metrics

In order to objectively compare different QA systems, it is important to set standard and objective numeric metrics of a system's performance. Usually,

the evaluation consists of running multiple systems on the same set of examples (or "dataset") and comparing the numbers of correct predictions. The most widely used metrics are (Powers, 2015):

Accuracy measures how often the system is correct in general.

$$Accuracy = \frac{true\ positives + true\ negatives}{number\ of\ examples} \tag{1.1}$$

Precision measures the chance that a predicted value is correct.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{1.2}$$

Recall measures the chance that the correct value will be predicted.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{1.3}$$

F_1 score (or F-measure) is the harmonic mean of precision and recall.

$$F_1 = (\frac{1}{2} \cdot (precision^{-1} + recall^{-1}))^{-1} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{1.4}$$

1.3 Machine learning

In order to understand natural language, a question answering system should ideally be able to adapt to different and possibly unseen formulations of the same sentence. Machine learning is a subfield of artificial intelligence that studies the ability of computers to learn patterns from data without being explicitly programmed (Murphy, 2012). In *supervised* machine learning, we are generally interested in finding a mapping from an input (feature) space X to an output space Y. This mapping is usually a function $\hat{y} = f(x, \theta)$, where $x \in X$ is the input, $\hat{y}$ is the predicted value and θ is a set of learnable parameters. We teach the algorithm to predict the output variable using a data set D of training examples (x, y) with $x \in X, y \in Y$ where the prediction targets y are known in advance. We select a suitable loss function $L(\hat{y}, y)$, a metric of error in prediction between $\hat{y}$ and y. The parameters θ are then optimized in a way that minimizes the loss function over all training examples. After training, the model should be able to generalize to unseen inputs and produce meaningful predictions.

We can further divide supervised learning problems into two categories based on the predicted variable y:

- classification - y is an element of a finite set of discrete classes,
- regression - y is a continuous variable.

I will now summarize a few important machine learning algorithms that are extensively used in question answering.

1.3.1 Linear regression

Linear regression is one of the most basic and widely used algorithms in supervised machine learning. As its name suggests, the algorithm works by approximating the independent variable y by a linear function of the vector of features $x \in \mathbb{R}^n$. The algorithm uses two parameters: a vector of weights $w \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$. The linear model predicts the dependent variable y as:

$$\hat{y} = \mathbf{w}^T\mathbf{x} + b = \sum_{i=0}^{n-1}(w_i x_i) + b \tag{1.5}$$

An obvious weakness of linear regression is its inherent simplicity, as it assumes the relationship between x and y is linear, which is sometimes an incorrect assumption and in these cases the model will give poor predictions. One way to alleviate this issue is by preprocessing the data manually by applying transformations to the input features. While this is a valid solution in most simple problems, it is often very time-consuming or outright impossible to do when dealing with complicated data.

1.3.2 Neural networks

A neural network is a biologically inspired machine learning model that is able to accurately model highly nonlinear data. There exist many different types of neural networks. For the purposes of question answering, the most significant ones are fully connected feedforward neural networks and recurrent neural networks.

1.3.2.1 Feedforward neural networks

Fully connected feedforward neural networks (Goodfellow et al., 2016) are a natural extension of the linear model. The basic building block of a neural network is the **perceptron**, a linear function of the inputs multiplied by certain weight coefficients followed by a nonlinear activation function σ. The output of a perceptron can be defined as:

$$y = \sigma(\mathbf{w}^T\mathbf{x} + b) \tag{1.6}$$

We can apply h perceptrons to the same input in parallel to form a **layer**. This layer will then output h different non-linear projections of the input features. A feedforward neural network contains multiple layers, with each one passing its outputs as the input to the following one. The predicted value $\hat{y}$ is the output of the last layer.

The numbers of layers and perceptrons in each layer are so-called **hyperparameters**, or parameters of the architecture itself. These need to be individually fine-tuned for each application.

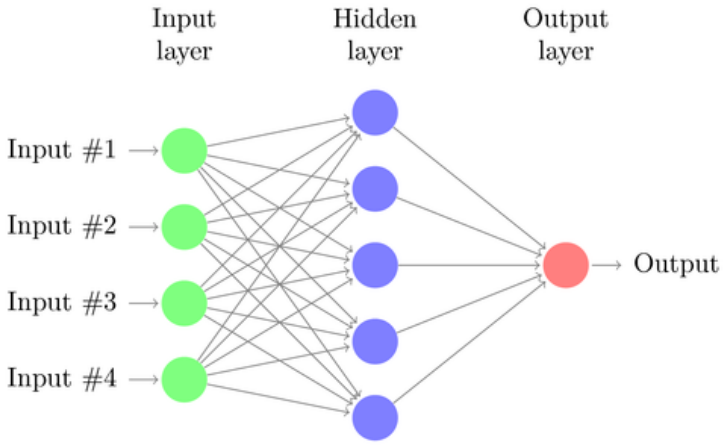

Figure 1.2: Feedforward neural network

It has been proven by Hornik et al. (1989) that feedforward networks are universal approximators and can model any function. However, in practice it is common to use deep neural networks with large numbers of layers.

1.3.2.2 Recurrent neural networks

The models discussed so far generate predictions for an input vector of a fixed size and have no sense of order between different positions of the input or output. In contrast to this, text is a sequence of words with each word depending on the other positions. To process this kind of sequential data, it is possible to use recurrent neural networks (RNNs). A recurrent network reads one word at a time from the beginning of the text, similarly to the human. At each time step, it outputs a value for the current position and updates its internal state ("thought vector") that contains information about what the network has read so far. The thought vector is passed along with the input to the next timestep and the process repeats (Schuster et al., 1997). It is possible to use the RNN in two ways: Firstly, the outputs for each position may be used to classify each token or to generate a new sentence. Secondly, the final internal state can be used as a vector representation of the sentence. The basic formulas for a recurrent network are:

$$h_t = tanh(W_h \cdot h_{t-1} + W_x \cdot x_t), \tag{1.7}$$

$$y_t = W_y \cdot h_t, \tag{1.8}$$

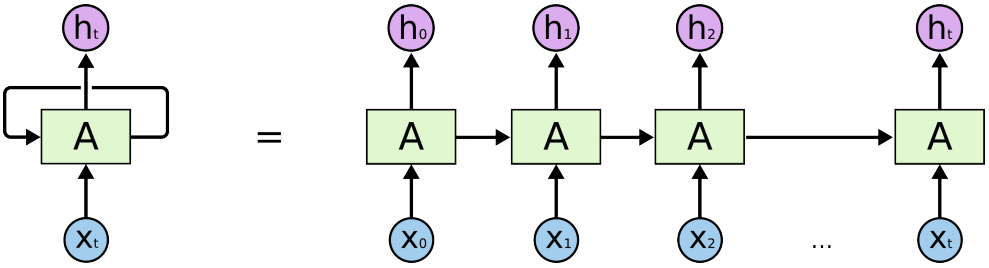

Figure 1.3: Recurrent neural network

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs

where W_h, W_x and W_y are sets of weights, x_t is the current input token and h_{t-1}, h_t and y_t are the previous and current hidden state and the output, respectively.

In practice, more complex RNN variants, such as long short-term memory network (LSTM), are used because of their better learning properties. (Hochreiter et al., 1997)

It is also possible to generate sentences by using two RNNs (Sutskever et al., 2014). The first one, also called **encoder**, accepts the input sentence $(x_1, x_2..., x_n)$ and outputs the final thought vector. The other RNN, the **decoder**, is then initialized with the hidden state and outputs the target sequence $(y_1, y_2, ..., y_m)$ word after word. This approach is called **encoder-decoder**. The probability of each output token y_i is modeled as a function of the previous output tokens $y_j, j < i$ and the thought vectors of the encoder and decoder c, s_i:

$$p(y_i|y_1, ..., y_{i-1}, c) = f_\theta(y_{i-1}, s_t, c). \tag{1.9}$$

1.3.2.3 Attention

Recurrent neural networks have a serious design flaw - they try to encode sequences of arbitrary length into a fixed-size vector! Indeed, it was shown by Cho et al. (2014) that RNNs tend to underperform with long sequences. Attention (Bahdanau et al., 2014) is a mechanism inspired by how humans process input. It allows the decoder to focus on important parts of text. When using attention, the decoder model changes to:

$$p(y_i|y_1, ..., y_{i-1}, \mathbf{x}) = f_\theta(y_{i-1}, s_i, c_i), \tag{1.10}$$

where c_i is a *context vector*, that contains information about the input sequence relevant for the specific position i. In additive (or Bahdanau) attention, the context vector is a weighted sum (interpretable as expectation) of the hidden states of the encoder.

$$c_i = \sum_{j=0}^{n} \alpha_{ij} h_j \tag{1.11}$$

The coefficients α are determined by a trainable similarity function a of h_j and s_i and normalized to (0; 1) using the *softmax* function:

$$\alpha_{ij} = \frac{\mathrm{e}^{a_\theta(h_j, s_i)}}{\sum_{j'=0}^{n} \mathrm{e}^{a_\theta(h_{j'}, s_i)}}. \tag{1.12}$$

This corresponds to the human concept of looking at the input tokens that are relevant to the current output position.

1.4 Preprocessing

When working with unstructured documents, it is very important to transform text into a numeric representation that can an algorithm can understand. A minimal text preprocessing pipeline should contain these steps:

- tokenization - the text is split into a list of words (tokens),
- stemming - word prefixes and suffixes are removed,
- vectorization - the words are encoded to numeric representations.

I will now briefly describe the most important techniques for text vectorization.

1.4.1 Bag-of-words

Bag of Words is the most straightforward algorithm used to map text into a vector. Firstly, we create the vocabulary V, a set that contains all tokens in the dataset. Then, we create a matrix of documents D, where each row corresponds to a document and each column corresponds to a single word from the vocabulary. We initialize all values in the matrix to zero. We then iterate over all documents and set the values in each word's column to the number of occurrences of the word in the document.

1.4.2 TF-IDF

Term frequency - inverse document frequency or TF-IDF (Ramos, 2003) is an extension of the Bag of Words that takes into account the relative importance of words. For example, in the sentence *"It is raining."*, the word *"it"* has clearly a lower information value than the word *"raining"*. Instead of simply counting occurrences, TF-IDF assigns weights to each words based on their relevance. If we denote the set of all documents D and the number of occurrences of a word (term) t in a document $d \in D$ as $f_{t,d}$, we can compute the

TF-IDF score of a word as:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in D} f_{t',d}}, \tag{1.13}$$

$$idf(t,d) = log \frac{|D|}{|\{d' | d' \in D \wedge t \in d'\}|}, \tag{1.14}$$

$$tfidf(t,d) = tf(t,d) \cdot idf(t,d). \tag{1.15}$$

TF-IDF is a simple and widely used algorithm. However, it has one drawback: it completely ignores semantic similarity, meaning the representations of words *"dog"* and *"puppy"* have no correlation whatsoever and the neural network will not be informed that they represent similar concepts.

1.4.3 Word embedding

Ideally, we would like text vectorization to preserve semantic information in text. This means that we should assign a vector to every word in such a way that the distance of each pair of vectors in the vector space is be proportional to the similarity of the words represented by these vectors.

Mikolov et al. (2013) demonstrated that this can be done by training a single-layered neural network. They proposed two models: The Continuous Bag of Words (CBOW) and the Skip-gram. The CBOW is trained to predict a word using its context – the words immediately preceding and following it in the document. Each input word is converted to a one-hot vector – a vector containing the value 1.0 under its vocabulary index and zeros everywhere else, similarly to the bag-of-words encoding. The words are then projected to a lower-dimensional space using a single layer.

This low-dimensional representation is used to predict the target word's vocabulary index. As a side effect, the representations of words are forced to reflect their similarity. In a similar manner, the Skip-gram is trained to predict the context of a word. These models can either be used to pre-compute word vectors in advance, or can be integrated as the first layer in a larger neural network.

Chapter 2

State of the art

In this chapter, I will give an overview of the present state of the art in the topics of machine reading comprehension and neural question answering.

2.1 Datasets

Because today's methods for machine reading comprehension are based on machine learning, research in this field is necessarily driven by the availability of large annotated datasets. (Rajpurkar; Zhang, et al., 2016)

One of the first datasets large enough to train machine learning models was released by Hermann et al. (2015), who collected a corpus of (a million) news articles from websites of CNN and Daily Mail, along with bullet points summarizing each article. They created training examples by removing a single named entity (name, location, ...) from the bullet points, making the incomplete bullet point the query and the named entity the answer. For example, the bullet point "– Mary visited England" would be converted to this question-answer pair: ("X visited England", "Mary"). The task (called cloze-style QA) is then to fill in the "X" placeholder using an answer (named entity) from the article. While the CNN/DailyMail dataset is large enough to train neural networks, it is still a simplification of the general QA problem both because of the simple structure of the questions and because of its factoid nature (Chen et al., 2016).

The more recent Stanford Question Answering Dataset (SQuAD) (Rajpurkar; Zhang, et al., 2016) contains approximately 100 000 questions written by a human annotator using information from English Wikipedia. The training set is composed of quadruples (question, context, answer_start, answer_end), where context is a short snippet of about 100 to 400 tokens from a Wikipedia article. The correct answer is a substring of the context between indices answer_start and answer_end. Additionally, the dataset contains 50 000 unanswerable ques-

tions (Rajpurkar; Jia, et al., 2018). This dataset is close to being solved and several models have in fact reached human performance of 89.452 F1 points. However, most questions in the dataset are simple reformulations of the answers and for this reason it can't be used to measure text understanding.

The newly released NaturalQuestions (Kwiatkowski et al., 2019) is the first dataset to use naturally occurring questions and focus on finding answers by reading an entire Wikipedia page containing tens of paragraphs. The model should provide a long answer - the correct paragraph in the page and a short answer - the correct answer span from the paragraph. The main improvement from the previous datasets is that the questions were not written by an annotator that has seen the answer in advance, but instead they are real users' queries from Google Search. About 50% of the questions do not have a correct answer. Each question was annotated by 5 humans, who chose a short and long answer span. Long answer is the smallest paragraph containing the desired short answer.

2.2 End-to-end models

After the release of the aforementioned datasets, there has been a large rise in interest in building end-to-end question answering systems using a single neural network. Since then, a plethora of (often unnecessarily complex) different architectures have been proposed. Most of these follow a common principle: they estimate the conditional probabilities of all possible answers a to a question q using information from a context c as $P(a|q,c) = f_\theta(q,c)$. In this section, I will try to summarize some of the most significant ones.

2.2.1 Attentive reader

With the release of the CNN/DailyMail dataset, Hermann et al. (2015) proposed three simple models. Firstly, they used an LSTM recurrent neural network to select the correct answer token. It works simply by reading the question and the context paragraph in sequence with a separator token in between. The network's final hidden state is then used to predict the answer token. Their second model, the attentive reader, is very similar, with the difference that it uses attention over all tokens when predicting the answer. Their third model, impatient reader, re-reads the entire document when processing each question token.

2.2.2 Attention sum reader

The attention sum reader (Kadlec et al., 2016) is a very simple architecture, also designed for cloze-style QA. The question is first passed through a re-

current network to obtain an embedding. The document is passed through a second recurrent network. The probability of each token being the answer is obtained by taking the dot product between the question embedding and the network's output for this position and normalizing the outputs using the softmax function. The probability of each word from the vocabulary is the sum of probabilities of its occurrences.

2.2.3 BiDAF

While the previous models work well for cloze-style QA, they aren't well suited for the more general span selection task of SQuAD. Seo et al. (2016) proposed the Bidirectional Attention Flow (BiDAF) that addresses some of the previous models' issues. Specifically, BiDAF uses a multi-stage process that avoids early summarization of the question and obtains a query-aware embedding of the context.

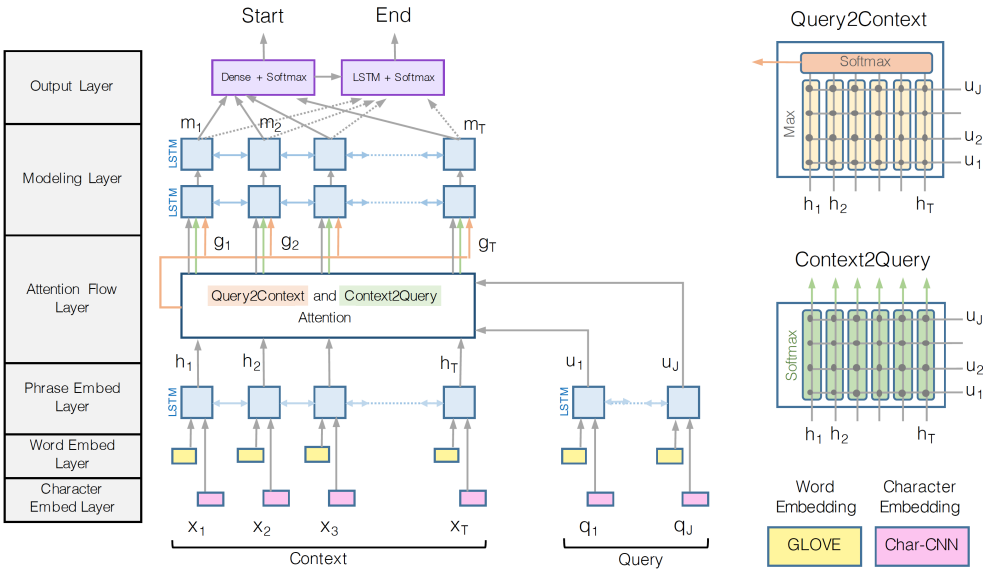

Figure 2.1: Bidirectional Attention Flow

Source: https://allenai.github.io/bi-att-flow

Firstly, BiDAF embeds the context and the query using independent character and word embedding layers. The purpose of the character embedding is to improve generalization on infrequent and new words that aren't contained in the vocabulary. The first key contribution is in the following contextual embedding layer that uses a recurrent network to refine the word embeddings based on the context the words are used in.
The actual question understanding is done in the attention flow layer, which

applies attention in both directions, query to context and context to query. The output of this layer is a sequence of *query-aware* context features that contain information about the relation of each context word to the query.
Lastly, the query-aware context features are processed by a multi-layered LSTM network and the outputs for each context token are passed through a fully-connected layer followed by softmax to obtain final start and end token probabilities.
Contextual word representations have proven to be quite important for text understanding. On the SQuAD dataset, BiDAF scored 62.093 F1 points. When enhanced with pre-trained ELMo word embeddings (Peters et al., 2018), the score increased to 66.251.

2.2.4 Transformer-based models

So far, all previous approaches used recurrent neural networks with attention to read sequences. Vaswani et al. (2017) discovered that it is in fact possible to use attention standalone. They proposed the transformer, a new encoder-decoder architecture based on attention, as an alternative to the RNNs. From an input sequence of tokens $\mathbf{x} = (x_1, x_2, ..., x_n)$, the encoder generates N intermediate sequences $\mathbf{z^{(i)}} = (z_1^{(i)}, z_2^{(i)}, ..., z_n^{(i)})$. The decoder is then conditioned on the final sequence $\mathbf{z^{(N)}}$ and similarly generates N refined representations of the output sequence $\mathbf{y}$.

The transformer uses an improved variant of attention called scaled dot product attention. It is defined as

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \tag{2.1}$$

Apart from this, the attention is *multi-headed*. This can be interpreted as the network focusing on more different aspects of the sequence at the same time. It is accomplished by first applying a linear layer to the features of each sentence token a number of times to obtain h different linear projections. The attention is then applied to each projection, the results are concatenated and passed through a fully-connected layer.

$$head_i = Attention(Q \cdot W_i, K \cdot W_i^k, V \cdot W_i^v) \tag{2.2}$$

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h) \cdot W^o \tag{2.3}$$

The encoder and decoder consist of N identical layers (or blocks). Each layer of the encoder is composed of a self-attention sub-layer, which computes multi-headed attention between tokens of the input sequence, followed by a feedforward network with two fully connected layers applied independently to each token. The decoder layers are similar with two differences. Firstly, the self-attention is masked so that each sequence position is only allowed to attend to

previous positions, otherwise the prediction would depend on future tokens. Secondly, each decoder layer contains a second attention sub-layer attending to the output of the encoder. (Vaswani et al., 2017)

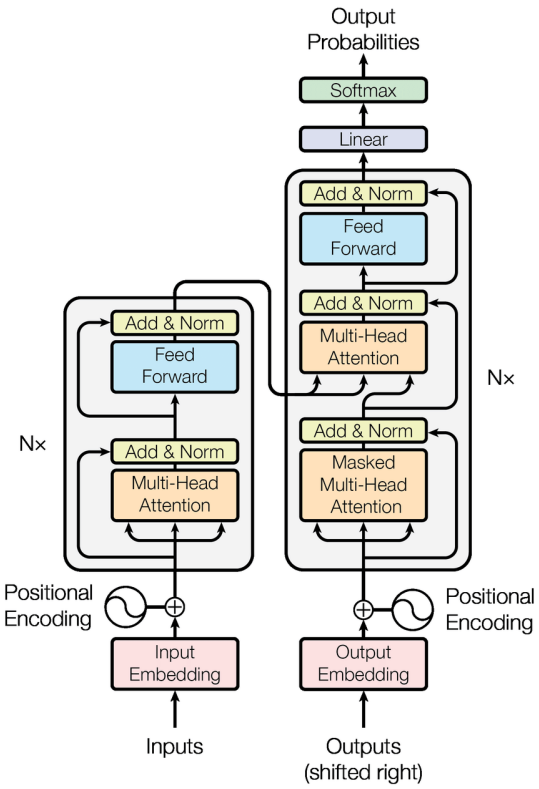

Figure 2.2: Layers of the transformer.

Source: Vaswani et al. (2017)

Unlike the RNN, which reads one token at a time, the transformer doesn't have an inherent representation of time and instead relies on augmenting the inputs with a positional encoding.

The transformer was originally created for machine translation, on which it outperformed previous state-of-the-art results while being faster and easier to train than recurrent networks. It was then applied to other NLP tasks including question answering. Recently, Devlin et al. (2018) introduced a model called **BERT** (Bidirectional Encoder Representations from Transformers). BERT is a large transformer encoder trained for two-tasks: masked language modeling and next sentence prediction. In masked language modeling, a specific number of tokens is randomly malformed or replaced (similarly to cloze-style QA) and the network is trained to predict the original words based on

their context. In sentence prediction, the network is given two sentences and it should predict if the second sentence logically follows the first one. Training on these tasks forces the network to create a *deep contextual word embedding* that can be used similarly as in Mikolov et al. (2013).
The pre-trained contextual embedding is then used for transfer learning and model is fine-tuned on different NLP tasks. BERT outperforms previous task-specific models in eleven tasks and achieves near-human performance (89.147 F1 points) on the SQuAD dataset.
Devlin et al. (2018) have released two pre-trained variants of BERT:

- BERT-base (12 layers, 768 hidden units, $110 \cdot 10^6$ parameters)
- BERT-large (24 layers, 1024 hidden units, $340 \cdot 10^6$ parameters)

One significant disadvantage of BERT is its size. It takes several weeks to pre-train a BERT model even on the best available hardware. A loaded BERT-large also requires more than 16 GB of memory, making it impossible to use on a standard GPU.

Chapter 3

Implementation

In the practical part of this work, I will implement multiple variants of a QA system using BERT. I will then train and evaluate them on the NaturalQuestions dataset. The dataset consists of two tasks - paragraph selection and answer span selection. From now on, these will be referred to as "long answer" and "short answer". There are two possible approaches to solving this dataset:

- **Joint** - All paragraphs are processed by a single neural network. The network outputs a possible answer for each paragraph and chooses as the answer the paragraph from which the most probable answer span was extracted.
- **Pipelined** - There are two separate components. The first component selects n most probable paragraphs and passes them to the second component that extracts the most probable answer. The answers still need to be scored, this can be done by either component.

I will first implement a single model that predicts short and long answers jointly in a single step. Then I will experiment with making the model more effective.

3.1 Exploratory data analysis

Before moving to implementation details, it will be beneficial to explore the NaturalQuestions dataset (Kwiatkowski et al., 2019). This dataset contains a training set of 307 373 examples (42 GB in total), a development set of 7830 examples and a testing set of 7842 examples of questions entered to Google search. Each example contains a tokenized question, a full Wikipedia page in HTML along with its tokens, a list of paragraph start offsets in the page and 5 annotations written by different human annotators. Each annotation contains a short answer and a long answer that the annotator chose subjectively. The

long answer is a HTML paragraph, form or other object that contains the information needed to answer the question. The short answer is a span (or multiple spans) of words that answer the question precisely. It is always a substring of the long answer. Both answers are defined by their start and end offsets in the HTML page. The answer is often not a factoid and may require some reasoning. The question can be answerable in one of these ways:

- LONG (only a long answer exists)
- SHORT (a full short answer exists)
- YESNO (the answer is a boolean value)
- NONE (the question is unanswerable)

The dataset is split into 50 gzip-compressed files in the JSON lines format. In this format, the examples are stored as JSON objects delimited by a newline. A single example from the dataset would look like this (expanded to multiple lines):

```
{
    "example_id": ...,
    "question_tokens": ["where", "is", "prague", "located"]
    "annotations": [{
        "long_answer": {
            "start_token": -1, "end_token": -1,
            "start_byte": -1, "end_byte": -1}
        "short_answers": [...]
    }, ...],
    "long_answer_candidates": [{
            "start_token": 100, "end_token": 110,
            "start_byte": 11454, "end_byte":11655
        }, ...
    ],
    "document_tokens": [...],
    "document_html": ...
}
```

In my analysis of the development dataset, I discovered a few notable facts.

About 50 % of all questions are unanswerable. The vocabulary contains 722 901 different tokens. 16.0 % of questions overlap with their answers in at least one token.

Although there are a few very long outliers, the mean question length in development set is 9 tokens and the mean paragraph length is only 60 tokens.

In fact, 91 % of the paragraphs contain less than 300 tokens. Each question in the development set has 137.1 paragraphs on average and out of these, 98% don't contain the correct answer.

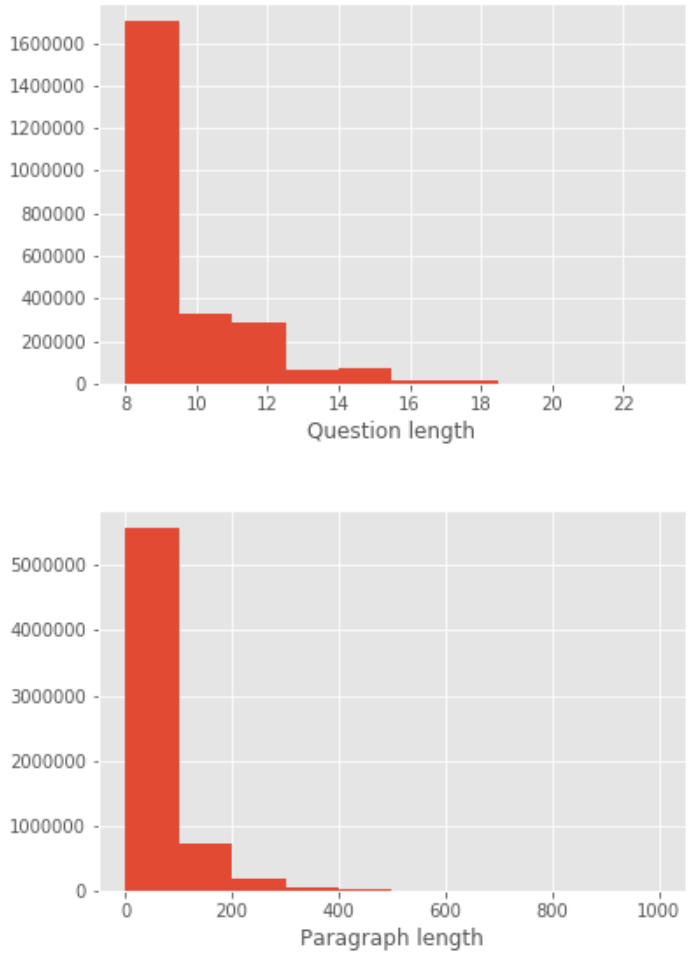

Figure 3.1: Histograms of question and paragraph lengths

Both because of the large number of unanswerable question-paragraph pairs and because of my limited computing resources, I will use a random balanced sample of the full dataset for training. This sample contains 263 921 valid training examples, out of which 97 459 (36.9 %) are answerable with a short answer.

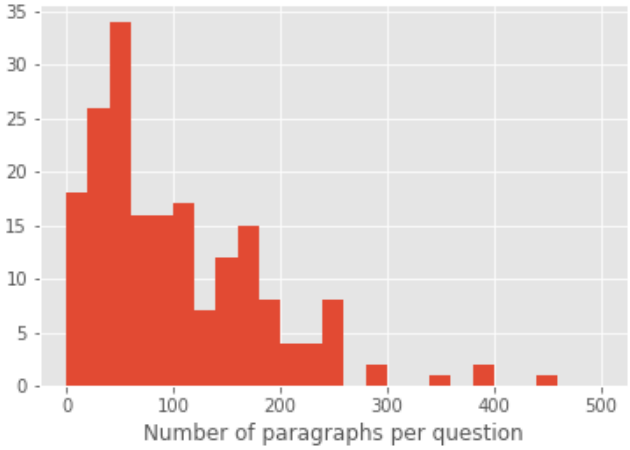

Figure 3.2: Histogram of paragraph counts

3.2 Setup

Deep neural networks are known for their high requirements on processing power and memory. With some optimizations, all experiments were performed on a desktop computer with Intel i5-7400 CPU with 8 GB of RAM and a single nVidia GTX 1080 GPU with 8 GB VRAM. The computer was running Ubuntu 18.04 and nVidia driver 410.73 with CUDA version 10.0. All code is written in Python 3.6, the industry standard programming language for machine learning. I use Google's deep learning framework Tensorflow v1.13 (Abadi et al., 2016). Tensorflow is a popular toolkit that defines neural networks as static graphs and uses highly optimized automatic differentiation on GPU.

The implementation consists of these parts:

- scripts for data preprocessing,
- definition of the tensorflow model,
- a script for training and evaluation,
- a script for answer postprocessing,
- a Jupyter notebook for tf-idf scoring.

3.3 Preprocessing

The first step of implementation is to transform the dataset into a format suitable for machine learning. The preprocessing pipeline thus needs to clean

up all text and convert it to a matrix of feature vectors readable by Tensorflow.

The text first needs to be tokenized. Even though the question and paragraphs are already provided as tokens, these can't be used directly, since they may not be contained in the pre-trained vocabulary of BERT. For this reason, I run a WordPiece tokenizer (Wu et al., 2016) (provided by the default BERT implementation) on every token from all questions and contexts. Each token is thus split into subword units. This eliminates the need for stemming or other out-of-vocabulary word handling.

The next step is to create a fixed-length sequence of input tokens that can be passed to the neural network. I use the recommended BERT input format

```
[CLS] query [SEP] context [SEP] padding
```

where the first token in input is always a special 'CLS' token that instructs the network to accumulate information about the whole input at this position. After this follow the query and the context delimited by special separator tokens. Finally, the remaining space is padded with zero tokens to indicate to the network that it shouldn't be processed. All tokens are then converted to integer token IDs using a hash map. The network also requires a second input of segment IDs, which are a way of specifying different logical parts of the input. In this case, the segment ids '1' and '2' are used for query and context positions, respectively.
Finally, the correct start and end positions are selected from the example's annotation and mapped to correct input indices. In training mode, the preprocessing script samples the examples so that more than 30 per cent are answerable. This greatly increases the training speed and prevents bias. Each preprocessed example is subsequently saved in the TFRecord file format directly readable by Tensorflow.
The mentioned preprocessing steps don't keep the original token positions in the paragraph, which we need to know in order to evaluate the model's performance. For this reason, in testing mode the script also saves a mapping between the original positions and token offsets in a JSON metadata file.

3.4 Modeling

I experiment with a neural network based on the BERT model followed by multiple variants of a domain-specific classifier. To the best of my knowledge, there is only one existing BERT implementation for the NaturalQuestions dataset at the time of writing, a baseline model by Google AI (Alberti et al., 2019) trained on a Cloud TPU. Throughout this chapter, I will point out

the similarities and differences with this implementation where necessary. In all experiments, I will be using the pre-trained BERT model by (Devlin et al., 2018), specifically the smaller BERT-base (uncased) with 12 transformer blocks and a hidden size of 768 (110M trainable parameters in total), in contrast to BERT-large with 3x more parameters used by Alberti et al. (2019). This decision is mainly motivated by memory and time constraints, and also because no experiments have yet been made with the smaller network at the time of the writing.

The network architecture is quite simple. Firstly, input tokens from the pre-processing step are processed by the pre-trained transformer encoder. The sequence outputted by the last transformer layer is then split into individual vectors for each position and these are passed to a classifier. For the first experiment, I use a single linear classifier with 2 hidden units. For each input position, the classifier performs the matrix multiplication $(batch\_size, hidden\_size) \times (hidden\_size, 2)$ and outputs a vector in $\mathbb{R}^2$. These two real values are interpreted as the scores of this position being the start or end of the short answer. Additionally, the output of the [CLS] token is passed through a linear classifier and normalized to $(0; 1)$. This number is used as the probability that the context is a valid long answer.

In addition to fine-tuning the entire transformer with a single linear layer on top, I experiment with freezing all layers and using only the pre-trained features to learn a multi-layer classifier. This way, the model can be used in production with significantly lower hardware requirements. For instance, it would be possible to run a single pretrained BERT model as a cloud server and use an API to get embeddings for different task-specific neural networks over the internet. This would allow the deployment of high-quality NLP models in standard web servers and embedded devices.

3.5 Training

The network is trained using gradient descent (Ruder, 2016). The most straightforward way to obtain predictions from the network is to use the *softmax* function and define a threshold probability, for example 0.9, above which the position is marked as the answer. It is then possible to minimize the standard *cross-entropy* loss. In contrast to this, Alberti et al. (2019) use the raw values of the network and minimize the loss function $\mathcal{L} = -log\ p_{start} - log\ p_{end}$, where p_{start} and p_{end} are raw outputs of the network. In case of no correct answer, the network is taught to predict the [CLS] token. I experimented with both methods and found that the choice had no visible effect on results, apart from slower training with the cross-entropy loss.

3.6 Hyperparameters

The network has a few hyperparameters that could influence its performance. Because the training of a single model is very time-consuming, it was infeasible to do a thorough hyperparameter search and the optimal values were determined empirically. I will now list the chosen values and justify their selection.

- batch size: 6
- learning rate: $1 * 10^{-5}$
- weight initialization: $\mathcal{N}(0, 0.05^2)$
- max sequence length: 312

The training batch size was set to 6 out of necessity, in order to fit the model into GPU memory. Better results could probably be obtained using higher batch sizes. The learning rate was chosen as recommended by (Devlin et al., 2018). All weights were initialized by the normal distribution, as is standard practice with neural networks. The transformer also requires to specify the maximum sequence length that can be loaded into memory. Alberti et al. (2019) solve this by splitting long paragraphs into multiple inputs with 512 tokens. However, during data analysis I found out that only 9 % of paragraphs are longer than 300 tokens. For this reason and also because of memory constraints I set the maximum input size to 312. I tested setting this value to 384 and splitting the remaining 9 % of paragraphs with a stride of 168 tokens, which seemed to have little impact on the results.

3.7 Paragraph selection

So far, the network is able to predict a short answer span. However, if there exist more contexts for a single testing example and the network predicts a different answer span for each one, we need a way of selecting the most probable one. I evaluated multiple scoring methods, such as using a separate "confidence" output or predicting the answer category (long/short/none/boolean). Surprisingly, the most effective method was to simply select the span with the highest sum of start and end predictions.

The selection of long answers is a related issue. If a short answer exists, it is logical to set the long answer to the HTML paragraph containing it. However, there are examples where the short answer doesn't exist. I solved this by adding a dedicated output score that predicts the long answer confidence. It is only used in case there is no predicted short answer. It would also make more sense to filter the contexts and possibly even determine the long answer

in advance. For this purpose, I implemented a pipelined system, adding a TF-IDF paragraph scorer that selects $n = 10$ best paragraphs to be sent to the neural network. This makes the prediction much faster, resulting in about 10x speedup.

Chapter 4

Evaluation

In this chapter, I will evaluate the performance of the implemented models on the NaturalQuestions dataset using standardized metrics. Because inference on the full development set takes more than one hour, I will use a random sample of 200 questions (10 000 question-context pairs) as the development set. I will then use the full development set as the testing dataset. First, I evaluate the ability of the models to correctly determine if the input question-context pair is answerable. Second, I compare the performances of various modifications of the model. Then I evaluate performance in the long and short answer prediction tasks using the original evaluation script by Google. Finally, I discuss the results, limitations and possibilities for future research.

4.1 Question answerability

I will now try to predict the answerability of question-context pairs from the sampled development dataset. I determine whether a question is answerable or not by selecting the highest scoring span from the context. If the highest scoring start and end indices are valid context indices, the question is answerable. In all other cases, such as when one of them points to the [CLS] token or to the question, I mark the question as not answerable.

Because this evaluation is done per-context and the contexts are generated during preprocessing, I use my own evaluation script. I find that the model has reasonable recall and can accurately flag answerable questions, but its precision is very low, giving a lot of false positives.

True positives: 82 True negatives: 9897	False positives: 768 False negatives: 11
Precision: 0.096	Recall: 0.882

Table 4.1: Answerability precision and recall

Based on these results, I argue that the joint prediction approach is not optimal and even though it finds the correct answer for a single training example, it is vulnerable to adversarial examples, which would seriously degrade its performance on real-world data.

4.2 Paragraph scoring

Because of the problems associated with joint long answer prediction, I tested a simple heuristic to select best paragraphs. The heuristic first weighs all tokens in the question and corresponding contexts and ranks them based on their tf-idf-weighted word overlap counts with the question. This is implemented as a dot product between the tf-idf vectors of the query and contexts. I experimented with 1-gram and 2-gram overlaps (using single words and two consecutive words). The following table shows the likelihoods of selecting the correct paragraph using different parameters of the heuristic.

tf-idf features	accuracy
1-grams, top-1	24.5
1,2-grams, top-1	25.0
1-grams, top-5	58.0
1,2-grams, top-5	55.5
1-grams, top-10	**73.5**
1-grams, top-20	**85.3**
1-2grams, BPE, top-10	70.0

Table 4.2: Paragraph scoring accuracy

The table shows that just by using a simple algorithm, it is possible to reduce the number of long answers that need to be processed with the neural network to just 10 with 73.5% accuracy. I experimented with other preprocessing steps, such as stemming the tokens and using subword tokenization (BPE), however, these had a negative effect on performance that I attribute to information loss.

4.3 Answer prediction

The following table shows the F1 scores of BERT with various modifications on the sampled development dataset. *BERT base* refers to the simple model with a linear classifier on top. *BERT freeze* is a model obtained by freezing all transformer layers and only training three fully connected classifier layers with 768 hidden units on top, added to increase the number of trainable parameters. Similarly, *BERT freeze + LSTM* contains an LSTM layer followed by two dense layers on top of a frozen BERT model. *BERT + tfidf* refers to the pipelined system consisting of the *1-grams, top-10* paragraph scorer and

the BERT base model for answer prediction. All models were trained for 2 epochs. On average, the training takes about 20 hours to complete.

	Long answer	Short answer
BERT base	49.9	**46.6**
BERT freeze	33.6	13.8
BERT freeze LSTM	35.4	15.1
BERT + tf-idf	**50.6**	44.8

Table 4.3: Comparison of different BERT variants

It is visible in the table that the tf-idf scorer had little negative impact on the performance and managed to filter out the unnecessary paragraphs. The BERT freeze models, on the other hand, perform quite poorly at present.

The following table shows the results of BERT-base compared to other available models on the tasks of long and short answer prediction on the full development set.

	Long answer			Short answer		
	P	R	F1	P	R	F1
Untrained						
First context (Google AI)	22.2	37.8	27.8			
Closest match (Google AI)	37.7	28.5	32.4			
Trained						
DocumentQA (Google AI)	47.5	44.7	46.1	38.6	33.2	35.7
DecAtt + DocReader (Google AI)	52.7	57.0	54.8	34.3	28.9	31.4
$\text{BERT}_{\text{base}}$ (this work)	47.5	54.9	50.9	50.4	40.0	44.6
$BERT_{joint}$ (Google AI)	61.3	68.4	64.7	59.5	47.3	52.7
Human						
Single annotator	80.4	67.6	73.4	63.4	52.6	57.5
Super-annotator	90.0	84.6	87.2	79.1	72.6	75.7

Table 4.4: Comparison of state-of-the-art results

The untrained baselines include *first context*, a simple baseline that always selects the first paragraph, and *closest match*, a heuristic that selects the paragraph with the largest TF-IDF word overlap. All compared results were reported in (Kwiatkowski et al., 2019) and (Alberti et al., 2019).

4.4 Extrinsic evaluation

I will now describe specific mistakes that the model makes when predicting short answers on the development dataset.

Following the metodology of (Seo et al., 2016), I manually classified incorrect answers into these categories:

Ambiguous answer boundary (47%)
Almost half of the mistakes are caused by ambiguity in what tokens should to be part of the answer. These errors are not related to language understanding and would be ambiguous even to a human.

Example: *"who got the first nobel prize in physics?"*
Predicted answer: *"wilhelm conrad rontgen"*
Reference answer: *"wilhelm conrad rontgen , of germany"*

Failure to understand meaning (25%)
The second most common mistake the network makes is also the most important one. In a quarter cases, the network is not able to distinguish the meaning of phrases in text or it is not able to perform reasoning based on these phrases.

Example: *"who wrote the first declaration of human rights"*
Predicted answer: *"mohammad reza pahlavi"*
Reference answer: *"cyrus"*
In this case, the context contained the phrase *"[...] the cylinder has also been referred to by mohammad reza pahlavi [...] as [...]"* and the network failed to understand that the verb *refer to* has a different meaning than the verb *to write*.

Example: *"who died in the plane crash grey's anatomy"*
Predicted answer: *"meredith"*
Reference answer: *"lexie"*
Here, the context contained the substring *"[...] meredith is relatively unscathed [...]"* and the network failed to understand and apply this piece of information.

Failure to understand the point of the question (10%)
In some cases, the network manages to find the correct answer, but doesn't understand what the user is looking for.

Example: *"what is the sentencing reform act of 1984"*
Predicted answer: *"to increase consistency in u.s. federal sentencing"*
Reference answer: *"u.s. federal statute intended to increase consistency in u.s. federal sentencing"*

Word used in different context (8%)
The network sometimes finds a relation also present in the question with a different meaning.

Example: *"the south west wind blows across nigeria between"*
Predicted answer: *"intertropical convergence zone"*
Reference answer: *"arrives in central nigeria in july [...] till september"*
Here, the context contained the string *"intertropical convergence zone swinging northward over west africa from the southern hemisphere"* and the network mistook the time relation *"between"* for the position.

Required world knowledge (5%)
In a few cases, the question is not answerable without basic knowledge about the world, such as geography in the following case.

For example: *"where do they grow hops in the us"*
Predicted answer: *"kent"* (located in UK)
Reference answer: *"the yakima (washington)"* (located in US)

Failure to parse HTML (5%)
Some contexts in the dataset, especially these that were extracted from Wikipedia forms, contain a large number of HTML tags, which the network can't understand. In my opinion, this is not a big issue and could be easily solved by further fine-tuning the model.

4.5 Discussion

Even though partial results can already be used in production, the machine reading comprehension task is still far from solved, and there still are many issues that will have to be addressed. For one, current models are seriously limited by design. So far, they only operate on a limited window of tokens and predict the answer by discriminating over all inputs. Furthermore, it can happen that the answer to a question is not a single span in text, but rather a set of such spans, which the network cannot handle yet. It is also unclear to what extent the present models are able to perform reasoning or if they merely make use of repeated patterns in text without language understanding.

Answer justification is yet another issue to be solved. Explainability and bias in AI are considered a big issue today and should be addressed before the deployment of such systems to production. It should always be clear and verifiable why an answer was returned and the user should always make their own judgement. Answer justification could possibly be realized by returning a list of reasoning steps, assumptions and supporting evidence along with each answer.

Future research topics also include augmenting models with world knowledge or "common sense" and being able to generate answers in natural language in addition to selecting them from text.

Finally, there is the issue of generalization to different languages.

4.5.1 Czech language

So far, machine reading comprehension has been limited to high-resource languages, specifically English and Chinese. With the recent improvements in neural machine translation (also thanks to the transformer), it is possible to translate individual sentences between languages with accuracy close to that of a human annotator (Bojar et al., 2018). This prompted my idea to create machine-translated datasets for machine reading comprehension in lower-resource languages. If perfected, this could open up a few interesting possibilities. In addition to making design of QA systems in different languages easier, one could for example compare the performances of QA systems in languages with difficult morphology or improve robustness of existing systems to content written by non-native speakers.

After experimenting with this idea, I translated the SQuAD dataset v2.0 to the Czech language using the best translation model available to date, CUNI-Transformer (Popel, 2018). Although some contexts are ill-formed due to being split into multiple inputs during the translation, the overall translation quality is surprisingly good. The project was realized at the Institute of Formal and Applied Linguistics, Charles University, whom I would like to thank for providing computing resources and the translation system. The dataset will be made available for download at `https://zilinec.me/dl`.

Chapter 5

Deployment

To demonstrate that neural question answering can be effectively used in production, I deployed a trained model on my website. I created a web application that can be used to answer questions about a custom context in real time. The application contains a backend REST API programmed in Flask, and a simple frontend programmed in static HTML and Javascript.

The backend accepts the question and context as JSON POST data and returns a JSON response containing a short answer along with a confidence score. The answer and score is not selected using same method as used for the NaturalQuestions dataset, but rather using a heuristic that yields longer, human-friendly answers that could be used in conversation with a chatbot. This heuristic selects 10 best answer spans and divides them into disjoint sets of overlapping answers. The set with the most elements is selected as the best and the final answer is formed as the union of the spans in the set.

The frontend contains a simple form along with a few example questions. It is is freely available for evaluation purposes at `https://zilinec.me/bert/`.

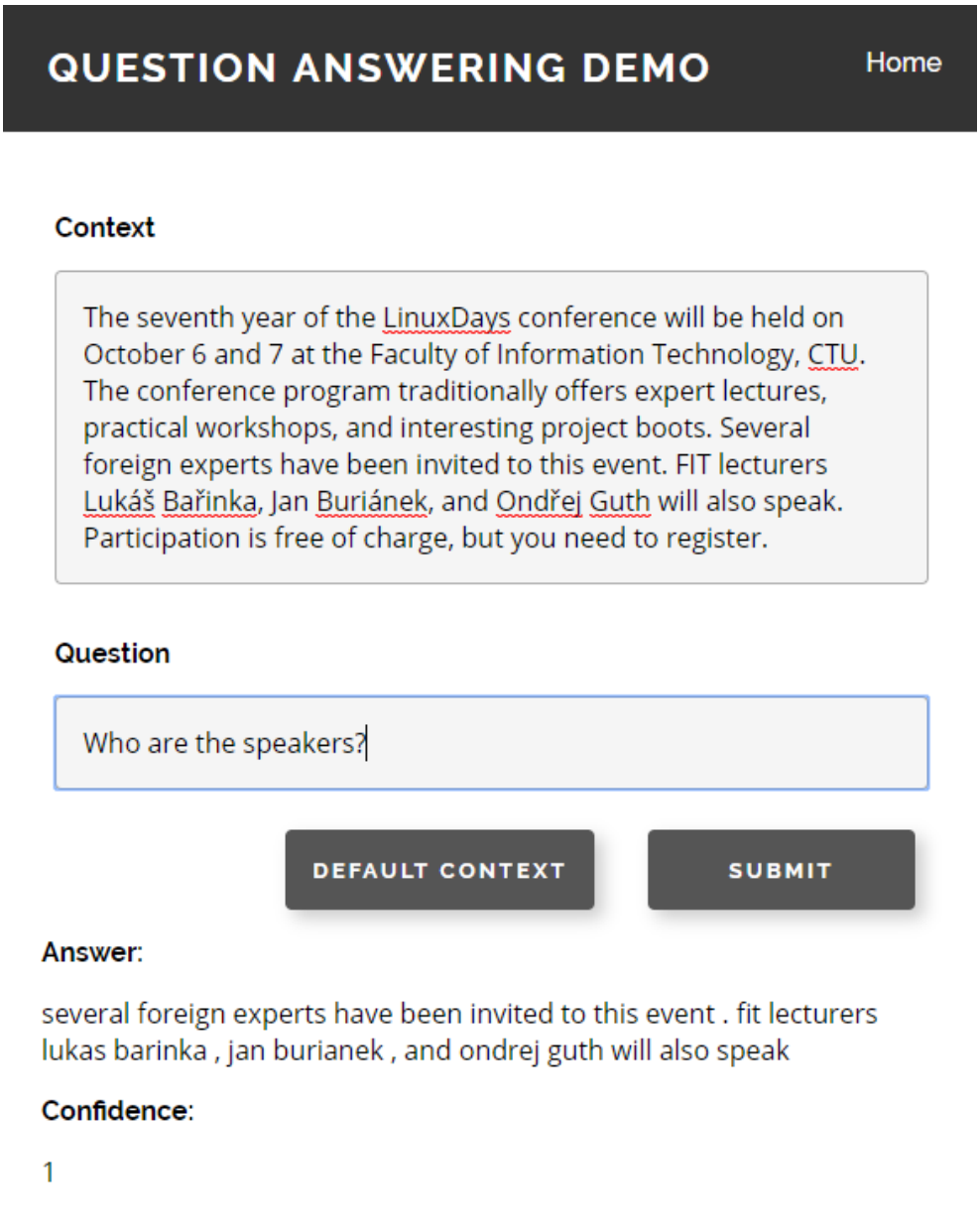

Figure 5.1: QA web application

Conclusion

The goals of the thesis were fulfilled. In the beginning of the thesis, I presented an introduction to question answering followed by an overview of necessary theoretical background. This includes the essentials of machine learning and neural network design, methods for text preprocessing and evaluation metrics. Building on this theory, I described selected state-of-the-art question answering datasets and neural network architectures, including the Transformer.

In the following practical part of the work, I implemented and trained a question answering system based on a BERT language model. This was followed by an evaluation of the results on the new NaturalQuestions dataset. To improve the performance of the system, I described and tested various possible modifications. In addition to this, the thesis explored and categorized the system's particular errors in prediction and reviewed the limitations of its design.

Finally, I discussed possibilities for future research and deployed a trained question answering system as a web application.

Bibliography

ABADI, Martin et al., 2016. TensorFlow: A system for large-scale machine learning. In: *TensorFlow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283. Available also from: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

ALBERTI, Christopher; LEE, Kenton; COLLINS, Michael, 2019. A BERT Baseline for the Natural Questions. *CoRR*. Vol. abs/1901.08634.

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua, 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*. Vol. abs/1409.0473. Available also from: https://arxiv.org/abs/1409.0473.

BAUDIŠ, Petr; ŠEDIVÝ, Jan, 2015. Modeling of the Question Answering Task in the YodaQA System. In: *Modeling of the Question Answering Task in the YodaQA System. Proceedings of the 6th International Conference on Experimental IR Meets Multilinguality, Multimodality, and Interaction - Volume 9283*. Toulouse, France: Springer-Verlag, pp. 222–228. CLEF'15. ISBN 978-3-319-24026-8. Available from DOI: 10.1007/978-3-319-24027-5_20.

BOJAR, Ondřej; FEDERMANN, Christian; FISHEL, Mark; GRAHAM, Yvette; HADDOW, Barry; KOEHN, Philipp; MONZ, Christof, 2018. Findings of the 2018 Conference on Machine Translation (WMT18). In: *Findings of the 2018 Conference on Machine Translation (WMT18). Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, pp. 272–303. Available also from: https://www.aclweb.org/anthology/W18-6401.

CHEN, Danqi; BOLTON, Jason; MANNING, Christopher D., 2016. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. In: *A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. Proceedings of the 54th Annual Meeting of the Association*

for Computational Linguistics (Volume 1: Long Papers). Berlin, Germany: Association for Computational Linguistics, pp. 2358–2367. Available from DOI: 10.18653/v1/P16-1223.

CHO, Kyunghyun; MERRIENBOER, Bart van; BAHDANAU, Dzmitry; BENGIO, Yoshua, 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In: *On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, pp. 103–111. Available from DOI: 10.3115/v1/W14-4012.

CLARK, Peter; ETZIONI, Oren, 2016. My Computer Is an Honor Student — but How Intelligent Is It? Standardized Tests as a Measure of AI. *AI Magazine*. Vol. 37, pp. 5. Available from DOI: 10.1609/aimag.v37i1.2636.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina, 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. Vol. abs/1810.04805. Available from arXiv: 1810.04805.

ELWORTHY, David, 2000. Question Answering Using a Large NLP System. In: *Question Answering Using a Large NLP System. TREC*.

FERRUCCI, David; LEVAS, Anthony; BAGCHI, Sugato; GONDEK, David; MUELLER, Erik T., 2013. Watson: Beyond Jeopardy! *Artificial Intelligence*. Vol. 199-200, pp. 93–105. ISSN 0004-3702. Available from DOI: https://doi.org/10.1016/j.artint.2012.06.009.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron, 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

GUU, Kelvin; MILLER, John; LIANG, Percy, 2015. Traversing Knowledge Graphs in Vector Space. In: *Traversing Knowledge Graphs in Vector Space. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 318–327. Available from DOI: 10.18653/v1/D15-1038.

HERMANN, Karl Moritz; KOCISKY, Tomas; GREFENSTETTE, Edward; ESPEHOLT, Lasse; KAY, Will; SULEYMAN, Mustafa; BLUNSOM, Phil, 2015. Teaching Machines to Read and Comprehend. In: CORTES, C.; LAWRENCE, N. D.; LEE, D. D.; SUGIYAMA, M.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., pp. 1693–1701. Available also from: http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.pdf.

HIRSCHMAN, Lynette; LIGHT, Marc; BRECK, Eric; BURGER, John D., 1999. Deep Read: A Reading Comprehension System. In: *Deep Read: A Reading Comprehension System. Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*. College Park, Maryland: Association for Computational Linguistics, pp. 325–332. ACL '99. ISBN 1-55860-609-3. Available from DOI: 10.3115/1034678.1034731.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen, 1997. Long Short-Term Memory. *Neural Computation*. Vol. 9, no. 8, pp. 1735–1780. Available from DOI: 10.1162/neco.1997.9.8.1735.

HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert, 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*. Vol. 2, no. 5, pp. 359–366. ISSN 0893-6080. Available from DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

ITTYCHERIAH, Abraham; FRANZ, Martin; ZHU, Wei-jing; RATNAPARKHI, Adwait; MAMMONE, Richard, 2001. IBM's Statistical Question Answering System.

JURAFSKY, Daniel; MARTIN, James, 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

KADLEC, Rudolf; SCHMID, Martin; BAJGAR, Ondřej; KLEINDIENST, Jan, 2016. Text Understanding with the Attention Sum Reader Network. In: *Text Understanding with the Attention Sum Reader Network. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 908–918. Available from DOI: 10.18653/v1/P16-1086.

KWIATKOWSKI, Tom et al., 2019. Natural Questions: a Benchmark for Question Answering Research. *Transactions of the Association of Computational Linguistics*.

LASSILA, Ora; SWICK, Ralph R., 1999. *Resource Description Framework (RDF) Model and Syntax Specification*. Available also from: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. W3C Recommendation. W3C.

MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey, 2013. Efficient Estimation of Word Representations in Vector Space. In: *Efficient Estimation of Word Representations in Vector Space. 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Available also from: http://arxiv.org/abs/1301.3781.

MOHAMMED, Salman; SHI, Peng; LIN, Jimmy, 2018. Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks. In: *Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks*, pp. 291–296. Available from DOI: 10.18653/v1/N18-2047.

MURPHY, Kevin P., 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press. ISBN 0262018020, 9780262018029.

PETERS, Matthew E.; NEUMANN, Mark; IYYER, Mohit; GARDNER, Matt; CLARK, Christopher; LEE, Kenton; ZETTLEMOYER, Luke, 2018. Deep contextualized word representations. In: *Deep contextualized word representations. Proc. of NAACL*.

POPEL, Martin, 2018. CUNI Transformer Neural MT System for WMT18. In: *CUNI Transformer Neural MT System for WMT18. Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, pp. 482–487. Available also from: https://www.aclweb.org/anthology/W18-6424.

POWERS, D. M. W., 2015. What the F-measure doesn't measure: Features, Flaws, Fallacies and Fixes. *arXiv e-prints*. Available from arXiv: 1503.06410 [cs.IR].

RAJPURKAR, Pranav; JIA, Robin; LIANG, Percy S., 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In: *Know What You Don't Know: Unanswerable Questions for SQuAD. ACL*.

RAJPURKAR, Pranav; ZHANG, Jian; LOPYREV, Konstantin; LIANG, Percy, 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In: *SQuAD: 100,000+ Questions for Machine Comprehension of Text. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2383–2392. Available from DOI: 10.18653/v1/D16-1264.

RAMOS, Juan Enrique, 2003. Using TF-IDF to Determine Word Relevance in Document Queries. In: *Using TF-IDF to Determine Word Relevance in Document Queries*.

RUDER, Sebastian, 2016. An overview of gradient descent optimization algorithms. *CoRR*. Vol. abs/1609.04747. Available from arXiv: 1609.04747.

SCHUSTER, Mike; PALIWAL, Kuldip K., 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*. Vol. 45, pp. 2673–2681.

SEO, Min Joon; KEMBHAVI, Aniruddha; FARHADI, Ali; HAJISHIRZI, Hannaneh, 2016. Bidirectional Attention Flow for Machine Comprehension. *CoRR*. Vol. abs/1611.01603. Available from arXiv: 1611.01603.

SORU, Tommaso; MARX, Edgard; MOUSSALLEM, Diego; PUBLIO, Gustavo; VALDESTILHAS, André; ESTEVES, Diego; NETO, Ciro Baron, 2017. SPARQL as a Foreign Language. In: *SPARQL as a Foreign Language. Proceedings of the 13th International Conference on Semantic Systems - SEMANTiCS2017 Posters and Demos*. Amsterdam, The Netherlands. Available also from: https://arxiv.org/html/1708.07624.

SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V, 2014. Sequence to Sequence Learning with Neural Networks. In: GHAHRAMANI, Z.; WELLING, M.; CORTES, C.; LAWRENCE, N. D.; WEINBERGER, K. Q. (eds.). *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pp. 3104–3112. Available also from: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N; KAISER, Łukasz; POLOSUKHIN, Illia, 2017. Attention is All you Need. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., pp. 5998–6008. Available also from: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

WESTON, Jason; CHOPRA, Sumit; BORDES, Antoine, 2015. Memory Networks. In: *Memory Networks. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Available also from: http://arxiv.org/abs/1410.3916.

WU, Yonghui et al., 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*. Vol. abs/1609.08144. Available from arXiv: 1609.08144.

Appendix A

Acronyms

AI Artificial intelligence

BERT Bidirectional Encoder Representations from Transformers

LSTM Long short-term memory

RNN Recurrent neural network

NLP Natural language processing

QA Question answering

TF-IDF Term frequency - Inverse document frequency

TPU Tensor processing unit

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `thesis.pdf` ..... the thesis text in PDF format
- `src` ..... the directory of source codes
  - `impl` ..... the directory of implementation source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis