universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Neural Machine Translation
- How machines learn to translate patent language -
An overview, evaluation and tutorial“

verfasst von / submitted by

Christian Lang BA

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Arts (MA)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 070 331 342
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Translation Deutsch Englisch
Betreut von / Supervisor:	ao. Univ.-Prof. Mag. Mag. Dr. Werner Winiwarter
Mitbetreut von / Co-Supervisor:	

Table of Contents

Abbreviations 6

Acknowledgments 7

1. Introduction 8

2. A brief history and the current state of machine translation 11

3. Rule Based Machine Translation (RBMT) 14

4. Statistical Machine Translation (SMT) 21

4.1 Word-based SMT 21

4.1.1 The word-alignment model 24

4.1.2 The translation model 26

4.1.3 The IBM models 27

4.1.4 The language model 29

4.1.5 Combining the models: The noisy-channel model 31

4.1.6 The limits of word-based SMT 32

4.2 Phrase-based SMT 33

4.2.1 The phrase translation table (phrase-based translation model) 35

4.2.2 Weighted models (log-linear modeling) 39

4.3 Tree-based SMT 40

4.4 Translation as decoding 41

4.5 SMT today (end of 2019) 43

5 Neural Machine Translation (NMT) 44

5.1 A closer look 44

5.1.1 Linear regression and the Perceptron 45

5.1.2 Training of neural networks: Backpropagation 49

5.1.3 Vectors: How neural networks “think” 50

5.2 Types and variants of neural networks 52

5.2.1 Multi-Layer Perceptron (MLP) 54

5.2.2 Convolutional Neural Network (CNN) 55

5.2.3 Recurrent Neural Networks (RNN) 56

5.2.3.1 The LSTM RNN (Long Short-Term Memory) 58

5.2.3.2 Encoder-Decoder modeling 61

5.2.3.3 Attention mechanism 63

5.2.4 The Transformer model 65

5.3 Summary 68

6 NMT in patent translation 70

6.1 A look at patent machine translation 70

6.2 The EPO's Patent Translate ... 72
6.3 WIPO Translate ... 75
6.4 Summary ... 78
7 Creating a Japanese-English Patent NMT model ... 79
7.1 Procuring the training data ... 82
7.1.1 Pre-processing of data ... 83
7.1.2 Tokenization ... 84
7.2 Which NMT toolkit to use ... 86
7.2.1 Tensorflow NMT ... 87
7.2.2 OpenNMT-py ... 88
7.2.3 OpenNMT-tf ... 89
7.2.4 nmt-Keras ... 90
7.2.5 Joey NMT ... 91
7.2.6 Selecting the most adequate toolkit ... 92
7.3 General recommendations before starting ... 95
7.4 Preparing the data (OpenNMT-tf) ... 100
7.4.1 Preparing datasets ... 100
7.4.1.1 Converting files to UTF-8 ... 102
7.4.1.2 Separate languages and store the sentences in line-aligned files ... 103
7.4.1.3 Tokenization of the files ... 104
7.4.1.4 Creating the vocabulary files ... 106
7.4.1.5 Creating the domain-controlled training data ... 106
7.4.1.6 Additional training data ... 109
7.5 Training (OpenNMT-tf) ... 110
7.5.1 Configuration and hyperparameters ... 111
7.5.2 Monitoring training ... 114
7.5.2.1 The BLEU metric ... 114
7.5.2.2 Loss ... 116
7.5.2.3 Perplexity ... 117
7.5.3 Training observations ... 118
7.6 Translating with the trained models and evaluation ... 120
7.6.1 BLEU evaluation ... 121
7.6.2 Human evaluation ... 123
7.7 Conclusion ... 141
8 Summary ... 144
8.1 Summary in German ... 145

8.2 Discussion and outlook 147
Bibliography 150
I. Appendix I: SAE J2450 evaluation 159
II. Appendix II: Code and scripts used in this thesis 162
Abstract 165
Abstract auf Deutsch 166

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BPE	Byte-Pair Encoding
CLI	Command Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
EM	Expectation Maximization
GPU	Graphics Processing Unit
HDD	Hard Disk Drive
LM	Language Model
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
MT	Machine Translation
NLP	Natural Language Processing
NMT	Neural Machine Translation
ODWS	Overall Document Weighted Score
OM	Omission
PSD	Parallel Sentence Data
RBMT	Rule-based Machine Translation
RNN	Recurrent Neural Network
SA	Word Structure or Agreement Error
SE	Syntactic Error
SL	Source Language
SMT	Statistical Machine Translation
SSD	Solid State Drive
ST	Source Text
TF	TensorFlow
TL	Target Language
TQA	Translation Quality Assessment
TT	Target Text
UA	Universal Attributes
UR	Universal Relations
UW	Universal Word
WM	Wrong Meaning
WT	Wrong Term

Acknowledgments

My heartfelt gratitude goes to my primary supervisor, Prof. Werner Winiwarter. He helped me troubleshoot several issues and gave me valuable insights into Computer Linguistics and the mathematics behind the concepts powering machine translation. He also meticulously checked several versions of my thesis while drawing from a seemingly endless pool of patience and knowledge. My thanks also go to my co-supervisor, Prof. Gerhard Budin, who was always quick to respond to my queries and provided valuable input for translation studies references and methods that would fit the requirements of this thesis.

I would like to thank all my colleagues who shared their experience with, observations of and questions regarding machine translation with me. The possibility to converse about the topic with other people who are confronted with translation every day was a very big help. Big thanks go to fellow translator, Patrick Hiehs, who helped with the human evaluation part of this thesis and gave valuable opinions on the performance of the trained models. Likewise, talks with those who come from the field of Computer Linguistics, like Bartholomäus Wloka, who is now a senior researcher at the Center for Translation Studies in Vienna, really helped shape the way I approached this thesis.

Thanks also go to my girlfriend, Maria Katzlinger, for enduring my endless ramblings about the workings of NMT, her participation in the discussion and the understanding for my late-night tinkering at the PC.

Last but not least, I would like to thank my parents, Sabine Raffeiner and Roland Lang, who never stopped me from doing the things I love (tinkering with computers since the age of 5, amongst other things) and enabled me to follow my passions with the study of Japanese and Translation.

1. Introduction

Machine Translation (MT) has recently gained a lot of momentum again, as deep learning methods based on artificial neural networks were (re-)introduced into the MT research sphere.

During my master's course at the Center for Translation Studies in Vienna, the way that Machine Translation (MT) and Computer Assisted Translation (CAT) tools were approached varied wildly from one lecturer to the other. One part of lecturers was very open to exploiting all tools available in order to provide the most efficient translation; the others warned us from overly relying on computer tools and focused on teaching us 'traditional' translation tactics and know-how for manual translation. This dualism is something even observed in studies regarding the acceptance of machine translation (see Cadwell et al. 2018). I believe that the more conservative stance was positive in the sense that it allowed us students to gain practical knowledge, skills and an understanding of the many possible issues one might face when translating, especially when not relying on supporting tools. However, I also believe that ignoring or antagonizing the developments in MT research is of no benefit and in fact very dangerous for translators to be. While we were taught basic concepts of machine translation, not a lot of time was spent with the concrete workings of how machine translation systems operate, and this may create a sense of alienation and threat coming from MT amongst Translation Studies students. Adding to this, we keep reading a fair share of doomsayer articles about MT systems replacing translators, as, on the face of it, it (once again) looks like machine translation is only a few steps away from being solved. While the rise in quality cannot be dismissed, it is mostly agreed upon by experts, that machine translation still has many glaring issues that need tackling and during our studies, we were generally promised that, as human translators, we will remain 'relevant' as machines will not be able to provide high-quality output any time soon. But aside from anecdotal evidence, no proper explanation as to why that is was provided.

While the evaluation of translation quality is in itself a very complex topic, neural network-based translation methods made big strides when it comes to a fluent and therefore 'perceived' high-quality output. I believe it is therefore imperative for translators to be at least somewhat informed on how these systems work, where they still don't, and how the Computer Linguistics approach the development of such systems. In fact, I believe, that giving translators a deeper knowledge about machine translation systems and development can benefit not only the translators but also MT research and the translation industry in general. Ideally, a more constructive collaboration between the two camps, translators and MT developers, should be strived for.

Translators may be able to contribute greatly to improving these systems and, conversely, stay relevant in this time of instant translations and high-quantity, subpar-quality texts (think "the internet"). For this, translators need to understand and to some degree work with these systems. As a patent-translator myself, I have found that recent advances in neural machine translation are really shaping the form in which patent organizations like the EPO or the WIPO can offer quite usable translations by automatic means on their websites. Instantaneously and for free. At the same time, using these services or just perusing many of the translated content available today, I have also witnessed plenty of times where the automatic translation fails to convey the correct meaning or downright obfuscates the original content. The risk involved in using automatic translations remains very high, especially if there is no bi-lingual postediting step involved. In fact, the risk is even higher today as it may not be immediately recognizable as a (faulty) machine translation, since the text may look correct in the target language. Problematic smaller nuances may even pass by undetected in a bilingual revision made by editors or translators, especially if the workings of neural networks are not well known. This is something that can be observed outside of the translation paradigm as well, with questionable overreliance on the so called "AI" or "Deep-Learning" solutions, that really seem to be omnipresent.

While there may be some talk about these "AI-Systems" replacing hitherto human expertise, this scenario should be considered as rather unlikely as rather than disappearing, the job of translators will more likely adapt to the necessities of the times, with more focus on pre- and post-editing. An area where both the linguistic as well as the translational expert knowledge can be put to good use and will be more important as reliance on MT increases in the industry. However, this may prove a daunting task and turn out as pure menial work for translators that have no insight into how artificial neural networks and the machine translation built upon them works.

This work therefore represents an attempt to have a more in-depth look into machine translation, especially the one powered by artificial neural networks or "AI". The attempt is to form an overview from the point of view of a Translation Studies student and not an IT student. In this work I try to learn how neural translation models are trained and which data preparation steps would help with the creation of a better translation model. Since one of my main sources of income are patent translations for the EPO, the thesis will focus on this very specific domain. While in general my working languages are German-Japanese for Patents, I will focus on English-Japanese as this should broaden the accessibility of this thesis and, on a more pragmatic note, simply more parallel text data is freely available in that language pair.

It is not the aim of this thesis, to create a new state-of-the-art translation model, but rather I hope to be able to shine some light on the practices of MT research and find some conclusive answers as to how translators as experts in their field can be more involved in the research regarding this topic. The thesis will take a more down-to-earth look at the recent developments in MT research and should also fulfill the purpose of a sort of tutorial for anyone with little to no IT background and interest in exploring (N)MT.

The thesis also aims to challenge the common conception of the MT research community, that more data will always equal in better results: The final part of this thesis will be experimental and focus on the creation of several NMT models, using freely available open source NMT/NLP-toolkits and patent text data provided by the NTCIR[1]. For creating the models, the data used will be manipulated in a pre-processing step to find out whether it is helpful to specialize a neural translation model onto a specific domain, or whether it is better to provide the network with as much data as possible. Furthermore, aside from the common automatic evaluation of the models' performance (through the BLEU metric), a human evaluation based on the SAE J2450 metric will also be provided.

In order to give a general idea about MT, all the major MT approaches as well as a short historical overview of MT will be provided before the more practical part of this thesis.

[1] http://research.nii.ac.jp/ntcir/permission/ntcir-10/perm-en-PatentMT.html (accessed on January 20, 2019)

2. A brief history and the current state of machine translation

Machine Translation (MT) as a concrete concept arose in the late 1940's when mostly political tensions between countries of different languages created the need for fast translations of texts from one natural language to the other. While some concepts were already quite promising and indeed close in theory to state-of-the-art machine translation approaches of today (see Weaver 1949, Hutchins 2007), the first systems, like the one developed by IBM in Georgetown (Hutchins 2004), were mostly dictionary- and word-based, disregarding most of the fundamental linguistic problems translators face (see Kaiser-Cooke 1993, Hutchins 2010). As such, they failed to live up to the (unreasonably) high expectations culminating in the publications of the ALPAC report in 1966[2], which prompted government(s) and many supporters to cut their funding, basically halting development of MT in the US and Britain for several years.

Canadian and European investment continued however, as the need for translation within their bi-/multi-lingual communities became ever more prominent. In Europe this led to the adoption of an English-French version of the SYSTRAN[3] system and the establishment of EUROTRA[4], while Canada developed the METEO system specifically designed for translation of weather forecasts. These systems were built on the foundation laid in the 1960's, however with linguistic rules added for the machine to follow during the translation decision. Such systems are regarded as Rule-based Machine Translation (RBMT) or as the 'Classical Approach' of MT and by their nature of following linguistic rules that have been defined manually by linguists, they offer very consistent and predictable translation quality, however at the cost of high-effort maintenance, sub-optimal handling of exceptions (especially exceptions to the fed rules) and usually rather poor *fluency*.

Research regarding RBMT continued at a steady rate in Europe and Japan (Carbonell et al. 1994), however different approaches to MT started to emerge. While in Europe the research-focus shifted to realizing an interlingua-based system, a system which could represent meaning by means of an interlingua independent of a specific language, Japan's focus shifted to what might be considered the opposite of RBMT: example-based translation, which uses a bilingual

[2] The ALPAC report of 1966 harshly judged the performance of MT-Systems of the time, especially pointing out the low return despite massive funding by government and other supporters and famously cited as 'there is no immediate or predictable prospect of useful machine translation" (see ALPAC 1966; Hutchins 1996:6; Cooke 1993:18)

[3] SYSTRAN, the company, was founded by Dr. Peter Toma in 1968. SYSTRAN, the MT-System, was one of the few machine translation systems to survive the major decrease of funding after the ALPAC report of 1966 and an English-Russian version of it was adopted by the US Air Force in 1970 (Koehn 2010:16).

[4] EUROTRA was an ambitious machine translation project that was funded by the EC and ran from 1978 to 1992 (Cooke 1993:43).

corpus with parallel texts as its main knowledge base at run-time (see Nagao 2003; Koehn 2010:17).

Japan's data-driven translation approach might be considered a forerunner to the next big step that was to occur in MT-history…

Approaching the end of the 1980s, the mathematical approach of IBM-scientists (Brown et al. 1993) of the IBM Candide project (Berger et al. 1994) laid the groundwork to the Statistical Machine Translation (SMT). Also based on parallel texts, machines would use statistical likelihood to evaluate translation possibilities. However, while the work was of major importance to the creation of SMT as we know it today, it was founded on a word-based approach. Once the source-code for IBM's MT approach was made public, researchers soon realized that a word-based MT would not be able to deliver results of satisfactory quality and so follow-up efforts soon developed the so called phrase based models (see Marcu & Wong 2002, Koehn, Och, & Marcu 2003), which became the de facto standard in MT research for over a decade. Thanks to these advancements, the increased processing power of computers, readily available huge amounts of data to create viable statistical results and freely accessible SMT-Frameworks (e.g. Moses[5]), studies regarding statistical machine translation gained a lot of momentum in the 2000's and many very promising approaches were laid out for further improving SMT.

However, the continuous adding of features to SMT also lead to stagnation in the development of SMT as systems started to incorporate more and more components to improve translation quality, but also needed a lot more maintenance to function properly. This posed a problem, as during recent years the need for instantaneous and effortless translation was bolstered by an ever-growing userbase of the internet and specifically social media as well as globalization in general.

Methods to make use of highly parallelized workloads, especially such that can be run on GPUs (Graphics Processing Units) instead of the conventional CPUs (Central Processing Units) allowed a lot more performance to be extracted from current computational equipment. While this development at first also allowed for a more efficient way of statistical data elaboration, this shift in technology also made previously only theorized approaches to Machine Translation feasible, especially those that were mostly coined 'AI-approaches'. However, rather than painstakingly trying to 'teach machines human knowledge' (see Kaiser-Cooke 1993:48), the research paradigm has shifted towards making machines 'learn language' on their own terms by using large corpora of 'real-word' data.

[5] http://www.statmt.org/moses/ (accessed March 03, 2019)

Enter Neural Machine Translation (NMT). Based on neural machine learning, which is often used synonymously with Deep Learning and AI in the broader media, this 'new' approach to machine translation is, at the time of writing, generally regarded as the new state-of-the-art in machine translation and the research surrounding it is booming (M.-T. Luong 2016:13; Koehn 2017:6; Dabre et al. 2017:1). While it actually still is a sort of Statistical Machine Translation – the machine is provided with a vast amount of data, based on which it will recognize statistical occurrences within said data – in a broader sense it will use this data to essentially learn how a given language is constructed on its own. Additionally, it can do so while looking at a whole sentence (or a sequence) instead of segregated phrases as was the case with SMT.

While NMT brings many improvements to the table, especially in terms of *fluency* of the output and manageability of big corpora, it does also come with its fair share of problems as well, especially in terms of managing and controlling the output provided by the machine: While a lot of the time-consuming manual intervention that was needed for prior methods of MT is no longer required with NMT, as the machine basically learns the language by itself from mono- and/or bilingual corpora, a lot of the decision making during the translation process and training of translation models happens in what is sort of a black-box for the maintainer of the system, which results in errors being very hard if not impossible to track down. Approaches to refine NMT are therefore mostly made in a pre- or post-processing step of the translation as, for example, unknown words are replaced via an external dictionary look-up (Luong 2016).

In this thesis I try to explore the impact manual preselection of training data can have on the results of an NMT output, especially whether it is sensible for patent translation where a very controlled language is used and therefore data should mostly only vary on a semantic and lexical level.

In order to provide an adequate background to the experimental part of this thesis, the following chapters will give an overview of the major MT methods that have been described above. A more in-depth look at how state-of-the-art NMT models operate will be provided in Chapter 5.

3. Rule Based Machine Translation (RBMT)

Rule-based machine translation (RBMT), or the 'Classical Approach' of machine translation, relies on linguistic information about the source and target text/language that is gathered from mono-, bi- or multilingual dictionaries and grammars covering the main semantic, morphological and syntactic regularities of each language respectively. It represents the logical evolution of the first MT systems developed back in the 1950s. RBMT systems may be divided into three major approaches:

1) Dictionary-based or direct translation
2) Transfer-based translation
3) Interlingua-based translation

The **direct translation** method simply maps the input of one language to the matching output of another language based on a dictionary lookup (hence dictionary-based) and by following some basic hand-written rules of word-reordering and possibly morphology. These systems are not built on any particular linguistic theory, instead they mostly rely on a sequential flow of word analysis and subsequently text generation (morphology/rearrangement). Typical stages in a direct translation system are as follows (Tucker 1987:23 in Kaiser-Cooke 1993:24):

- Source text dictionary look-up and morphological analysis
- Identification of homographs
- Identification of compound nouns
- Identification of noun and verb phrases
- Processing of idioms
- Processing of prepositions
- Subject-predicate identification
- Syntactic ambiguity identification
- Generation and morphological processing of target text
- Re-arrangement of words and phrases in target text

While direct translation is arguably the least sophisticated approach, it is ideally suited for translation of long lists of phrases on a sub-sentence level, like inventories or simple catalogs of products and services. The limitations of such direct mapping strategies become apparent once the task is to translate full sentences, especially in languages with vastly different grammars. Many grammatical constructs cannot be directly mapped into a different language and cannot be generally treated in the 'idiom processing' stage. For example, a specific

language might lack a certain case used in the source text or use a different grammatical construct to express the same meaning.

Because of these problems, RBMT development moved to different strategies that try to interpret the 'meaning' of a sentence by means of linguistical rules inherent to syntax and lexical components in context. Two main strategies are employed for this kind of approach: the **transfer-based** method and the **interlingua-based** method. See Figure 1 for an illustrated representation of these methods as proposed by Hutchins & Somers (1992:107).

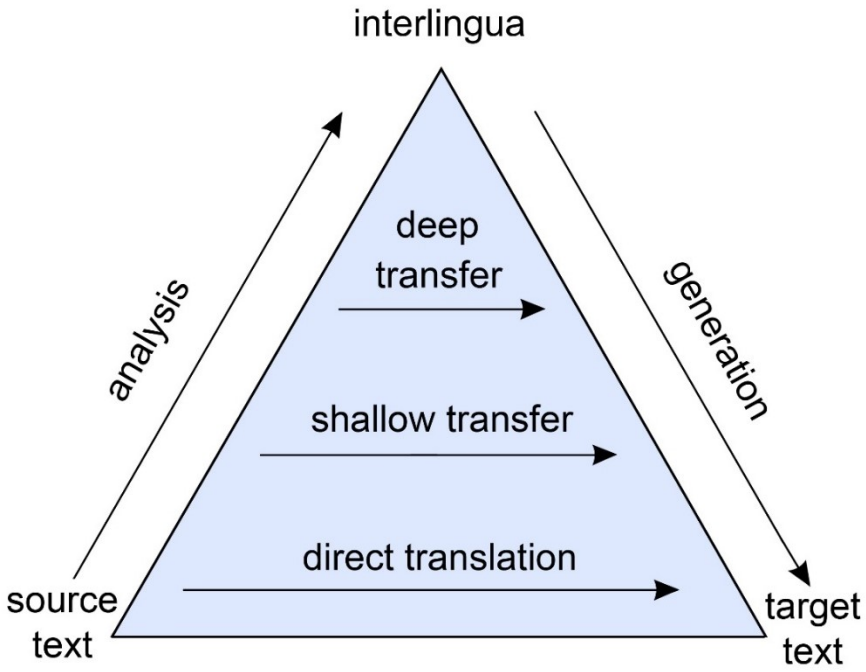


*Figure 1: Direct-, transfer- and interlingua-'pyramid' (*Hutchins & Somers 1992*: 107); base image taken from wiki-media* [6].

The **transfer-based translation** performs a morphological and syntactic analysis, trying to capture the 'meaning' of a source language (SL) on an abstract level, which is then transferred by applying the corresponding rules and 'equivalent' translations for the target language (TL). As such, the transfer strategy is generally language-pair specific. During the translation process the source language is parsed into an abstract structural representation and then transferred using this information as well as lexical information. During this transfer process, a bilingual dictionary forms the center piece of the method, as it provides 'translation equivalents' for the two languages (Kaiser-Cooke 1993:25). The translation basically consists of three steps (Popa 2008:152-153, Tyers 2013:5):

[6] https://en.wikipedia.org/wiki/Transfer-based_machine_translation#/media/File:Direct_translation_and_transfer_translation_pyramid.svg (accessed March 30, 2019)

1) *Analysis*: Describes the SL text linguistically while also relying on a dictionary, forming an SL intermediate representation of the text (SL IR).
2) *Transfer:* Transforms the results of the analysis step (SL IR) into an intermediate representation for the TL (TL IR) by determining the linguistic and structural equivalents between the two languages.
3) *Generation/Synthesis:* Produces a text in the TL based on the TL IR using a dictionary.

ST—> SL IR—>TL IR—>TT

Although it is a big step up from a direct translation, it still aims at providing a translation that represents a full 'equivalent' match of the source text in terms of lexical and structural units and the use of a simple bilingual dictionary with fixed 'equivalent' translations does limit the scope of this approach. Depending on the level of abstraction of the intermediate representation, we may group transfer-based RBMT into two groups: **shallow transfer** and **deep transfer.** Shallow transfer, where the intermediate representation is usually based on morphology or shallow syntax, may suffice for related language-pairs (Tyers 2013:4; Forcada et al. 2011). However, for more distant languages, like, for example, English and Japanese, a deeper analysis, including full syntactic or even semantic information is likely needed. Such a transfer would be called a deep transfer.

The **interlingua-based** translation goes one step further by trying to represent meaning in an interlingua that is independent of the translated languages or in fact any language, making the intermediate representation of the text language agnostic.

ST—> Interlingua—>TT

This idea of representing meaning in an explicitly formal way harkens back to the Chomskyan notion of *linguistic universals*, where it is assumed that there is a definite underlying meaning within a linguistic construct (*deep structure*), no matter what the language or grammatical structure (*surface structure*) is. This sparked a research trend in the 1980s and 1990s that was picked up by both researchers from the fields of artificial intelligence and computational linguistics alike (see Koehn 2010:16). The appeal of such a system makes sense, as translating involves the expression of meanings in different languages. However, as Koehn pointed out "the problem of representing meaning in a formal way is one of the grand challenges of artificial intelligence with interesting philosophical implications." (Koehn 2010:17). Cooke also argues that it is difficult to assess just how much of the 'meaning' a translator, or in this

case the machine, would need to grasp in order to provide an adequate translation, arguing that "the old division between subject knowledge, linguistic knowledge and real-world knowledge" in MT research remained, and the approach still disregards the choices human translators make while relying on their expert knowledge as a translator (Kaiser-Cooke 1993:50). Of course, it seems plausible that it would be very difficult, if not impossible to hard-code general rules for the decision making a translator follows, as such decisions are mostly made on a case-by-case basis and indeed are quite subjective.

Two notable projects for formal semantic representation are the **UNL (Universal Networking Language)**[7] and **AMR (Abstract Meaning Representation)**[8] projects. The UNL effort was started in 1996, as an initiative of the Institute of Advanced Studies (IAS) of the United Nations University (UNU) in Tokyo, Japan. In 2001 the UNDL Foundation was formed, constituted out of a world-wide network of universities and research institutes from 14 countries responsible for the further development and management of the UNL project. The mission of the project was to provide the methods and tools for overcoming the language barrier on the World Wide Web in a systematic way (see Hong & Streiter 1999 and [9]). As such this language is meant to express meanings in the same standardized way as HTML presents its content. It is built around the concept of Universal Words (UWs), Universal Relations (URs) and Universal Attributes (UAs). The system therefore aims to convert the ST to the UNL by forming nodes (the UWs) that are a human-language independent and machine-tractable representation of the core meaning of a certain word and subsequently adding information about relations between these concepts and specific attributes of the nodes (i.e. grammatical annotation, connotations, etc.); this process is called *enconverting*. Because the process is so similar to the analysis step in RMBT systems, slightly modified RBMT parser and recognizer modules may be used for the *enconversion* (Hong & Streiter 1999:3-4). Usually this process is still mostly handled by humans that have to hand-edit the automatically generated annotations, so that they fully and correctly include all the needed mark-ups (Martins 2010:2). Still, some information inherent to the ST may be lost, as the UNL only conveys concepts that are believed to be universally available in all languages and that have, at some point, been modeled by hand. For example, the UNL did originally not possess the capability to model the speech style, the tenor (the speaker-hearer relation) nor the 'channel of communication', so that such properties of a natural language expression simply disappeared (Hong & Streiter 1999:4).

[7] http://www.unlweb.net/unlweb/ (Accessed March 30, 2019)
[8] https://amr.isi.edu/ (Accessed April 4, 2019)
[9] https://web.archive.org/web/20040602215955/http://www.iai.uni-sb.de/iaien/en/unl.htm (Accessed April 4, 2019)

In summary, UNL can be seen as a kind of mark-up language which represents not the formatting but the core information of a text. The core information being the most common interpretation of the text, according to the hand-modeled UWs, UAs and URs. As such it can be embedded in the eXtensible Mark-up Language (XML), and as HTML/XML annotations already can be realized differently in the context of different applications, machines, displays, etc., so UNL expressions can have different realizations in different human languages. By using a language-specific *deconverter* that can generate a human-language output from the annotated UWs, one of the fundamental ideas behind the UNL was the creation of "globalized" versions of web-pages or documents, that could then be easily displayed in several languages by having the browser or a plug-in *deconvert* the UNL into a specific chosen language. So, while the UNL can be used as an interlingua, its primary objective is to serve as an infrastructure for handling knowledge in a natural-language agnostic way, rather than acting as an anchor point for translation between individual languages. In many ways, the UNL is not fit to cover all aspects of a good translation, as it always only represents one reading of an ST, which is in fact reliant on the interpretation hard coded in the knowledge base of the UNL.

AMR on the other hand was from the get-go never intended to be used specifically in machine translation, as an interlingua between SL and TL or to generate natural language with a computer. That said, the effort 'hopes to spur new research in natural language understanding, generation and translation'[10]. The idea is that one AMR representation should cover a variety of different formulations of sentences which convey the same meaning. It is a fairly recent effort, with many of the latest contributions dating back to 2016/2017. AMR is heavily geared towards English grammar and vocabulary and does therefore not share the strife for universality as UNL does. Because of that, the researchers behind AMR do officially not refer to the AMR as an interlingua (see [11]). The AMR Bank is manually constructed by human annotators at the Linguistic Data Consortium, SDL, the University of Colorado's Center for Computational Language and Education Research (CLEAR) and the University of Southern California's Information Sciences Institute (ISI) and Computational Linguistics at USC. It exists to standardize annotation formatting and is still actively developed. The current AMR 1.2 (May 2019) "is over-simple in many ways", as, for example, it drops grammatical number, tense, aspect, quotation marks, etc., does not deeply capture many noun-noun or noun-adjective relations and does not include deep frames for words and concepts such as 'earthquake' (with roles for magnitude, epicenter, casualties, etc.) or 'pregnancy' (with roles for mother, father, baby gender, time since inception, etc.).[12]

[10] https://amr.isi.edu/ (accessed April 4, 2019)
[11] https://github.com/amrisi/amr-guidelines/blob/master/amr.md (accessed April 4, 2019)
[12] https://github.com/amrisi/amr-guidelines/blob/master/amr.md#amr-slogans (accessed April 4, 2019)

While promising strides were made, interlingua-based RBMT systems therefore have yet to reach the results that are expected from machine translation today. In fact, as we saw, a true interlingua has yet to be fully developed. However, such an endeavor will only get increasingly complex the broader the language-spectrum the interlingua aims to cover. This is why AMR started small by mainly focusing on an abstract representation of the meaning of texts in English, with the hopes that the concepts and standards formulated can then be brought over to other languages and projects (see for example Damonte & Cohen 2018; Moreda et al. 2018). Still, the development of interlingua systems was important for the success of bigger projects like the German Verbmobil, because they allowed for a standardized representation of semantic and syntactic features and a better interface between natural languages and machine. However, in later phases, Verbmobil also relied on statistical machine translation for its translation part (see Ney et al. 2000).

As written in the introductory overview of the MT history, RBMT systems offer a few advantages over the other major MT solutions (for reference, see SYSTRAN[13] Website; Tyers 2013:5-6):

+ Consistent and predictable quality
+ Out-of-domain translation quality
+ Are aware of grammatical rules
+ High performance and robustness
+ Consistency between versions
+ Do not require vast amounts of data

Conversely, there are some intrinsic disadvantages compared to the data-driven approaches:

- Lack of fluency
- Hard to handle exceptions to rules
- High development and customization costs

[13] http://www.systransoft.com/systran/translation-technology/what-is-machine-translation/ (accessed March 20, 2019)

RBMT was used in many commercial solutions and would later go on to be implemented in SMT-systems as well. Some of the first systems in actual use include:

- SYSTRAN (Used by the European Commission until 2010)[14]
- METAL[15]
- MÉTÉO (Used until 2001 by Environment Canada for weather forecast translation)
- EUROTRA[16]

With the advent of the internet and the very active computational linguistics community, many open-source RBMT systems also emerged. Two notable efforts are:

- Apertium: A shallow-transfer-based RBMT (Forcada et al. 2011)[17]
- GramTrans: Deep-transfer-based RBMT[18] (deep-transfer only for Danish)

Today, RBMT systems are still used in specialized environments, but in the broad sense, they were replaced by or are used to enhance their data-driven 'successors' for most commercial solutions.

[14] https://webgate.ec.europa.eu/fpfis/mwikis/thinktank/index.php/European_Commission_Machine_Translation (accessed March 20, 2019)
[15] http://www.ccl.kuleuven.ac.be/about/METAL.html (accessed March 20, 2019)
[16] http://www-sk.let.uu.nl/stt/eurotra1.htm (accessed March 20, 2019)
[17] https://www.apertium.org/ (accessed April 10, 2019)
[18] https://gramtrans.com/ (accessed April 10, 2019)

4. Statistical Machine Translation (SMT)

This chapter about Statistical Machine Translation is based mostly on Phillip Koehn's excellent *Statistical Machine Translation* (2010) and is an attempt to explain the workings of the different variations in SMT in simpler terms.

Knowing about the basic concepts of how SMT systems work and how they are fed with data is going to be beneficial in understanding how state-of-the-art NMT systems work. In fact, many key concepts still play an important role for the neural models. With regards to NMT, this chapter shall also provide some insight into how SMT is used today and how it was used prior to the advent of NMT.

As written in the introductory chapter, the advent of SMT – first models were pioneered by IBM in 1993 (Brown et al. 1993) – laid the groundwork for a new wave of research enthusiasm, but the wave of general hype around SMT took a little longer to take off as interlingua-based research was still in the center of attention. In fact, while Koehn goes as far as to call the emergence of SMT in the 1990s "groundbreaking", he also notes that "in retrospect it seems the world was not quite ready for it" (Koehn 2010:17).

SMT truly took off around the year 2000, as tools which made use of the IBM algorithms developed in 1993 were made widely available under a toolkit named *GIZA* (Al-Onaizan et al. 1999). Koehn further points out, that funding by DARPA[19] as well as the US response to the events of 9/11 played a role in the renewed interest of automatic translation of foreign languages, especially Arabic (2010:18).

Another important factor, as was also mentioned in the introduction, was the increase in computing power, data storage and general availability of large text resources thanks to the growth of the internet. Generally, anyone with a somewhat recent computer system at home could build their own machine translation system based on these freely available resources. This was further bolstered by the availability of refined and free open-source frameworks like *GIZA++*[20] (Och & Ney 2003) and *Moses*[21] (Koehn et al. 2007).

4.1 Word-based SMT

While word-based SMT has been all but replaced by its phrase-based successor, it still introduced many of the basic concepts that form the foundation of SMT in general. Thanks to its simplicity it is easier to understand than more recent models and many key ideas are still relevant for phrase-based SMT and in fact NMT. This section will provide a general overview

[19] Leading funding agency in the US.
[20] http://www.statmt.org/moses/giza/GIZA++.html (accessed April 19, 2019)
[21] http://www.statmt.org/moses/index.php?n=Main.HomePage (accessed April 19, 2019)

of the tactics used in word-based SMT. For an in-depth discussion and simple enough to understand mathematical explanation of word-based models, Koehn's Chapter 4 (pp. 81-125) in *Statistical Machine Translation* (2010) is highly recommended; the interested and mathematically adept reader may also refer to IBM's original paper on their SMT models (Brown et al. 1993).

Word-based SMT is based on lexical translation, that is the translation of isolated words. Basically, this requires a dictionary that maps words from one language to another. So, just as with direct- and transfer-based RBMT, we need clear word alignments between the language-pairs. However, SMT does not use a fixed bilingual dictionary, but theoretically considers all the possible translations as they are found in the data. The data, in that case, is generally a large parallel corpus between two languages that is used to create a dictionary and in fact the word-alignment on a statistical basis. But let's first stick to Koehn's example of *Haus* (Koehn 2010:81) in a German-English dictionary[22]:

Haus – house, building, home, household, shell.

The challenge is to choose the right translation out of a variety of translations for a specific word; human translators would likely look at the context, domain, the definitions or eventually even follow their feel for a language and then decide. RBMT used a bilingual dictionary and then made use of hand-crafted rules to decide. Word-based SMT however relies on a so-called **lexical translation probability distribution** for the decision. What this means is that the SMT system first needs to align each SL word to TL word(s) in a sentence-aligned parallel corpus by statistical likelihood. This is achieved by the means of a statistical **word-alignment model.** After that a **translation model** needs to give a probabilistic score to each word-alignment. In formulaic terms we would look at the following function:

$$p_f: e \rightarrow p_f(e)$$

Equation 1: Probability function p_f

The function p_f returns a probability for each choice of TL translation e for a certain SL word f (in this example: *Haus*) (Koehn 2010:82). The function has two properties shown in Equation 2 and Equation 3:

[22] Koehn quotes this as a simplified version of an entry in Harper-Collins (1998) (Koehn 2010:81)

1) The sum of all probabilities must be 1.

$$\sum_{e} p_f(e) = 1$$

Equation 2: The sum of all probabilities $p_f(e)$ must be 1

2) All probabilities for the different variables *e* (the translations) in the function must be in the range of 0 to 1.

$$\forall e: 0 \leq p_f(e) \leq 1$$

Equation 3: The probabilities are expressed between 0 and 1

With these requirements fulfilled, it is possible to find the most probable translation by simply counting the occurrences of a certain word-pair in a vast parallel corpus of text. If the corpus was several dictionaries, the word which appeared more often as the translation of *Haus* (which would probably be *house)* would get the highest score, while all other solutions would get a lower score. The idea here is simple: The more often a certain translation was chosen for a specific word, the more likely the meaning may be considered as 'equivalent'. In statistical terms, the **maximum likelihood estimation** method is used for estimating the statistical parameters of each word.

So, in order to generate a basic translation, we would need at least two statistical models:

- A word-alignment model
- A translation model

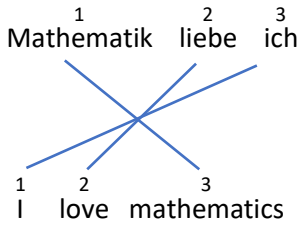
4.1.1 The word-alignment model

From a mathematical perspective, SMT systems work with incomplete data, where the alignment of words is considered to be a hidden variable (Koehn 2010:88). As we saw, RBMT-systems use a dictionary to find the alignment during the analysis phase and reorder the words during the final generative phase of the process. SMT, however, does generally not rely on a ready-made dictionary. Instead, probabilistic methods are used to find the most probable alignment of a given word by looking at parallelized text (preferably translations of a text). Through an alignment function a (see Equation 4) it is possible to formalize the mapping of every TL output word at the position i to an SL input word at the position j. In our examples, we will use English as our TL and German as our SL.

$$a: i \rightarrow j$$

Equation 4: The alignment function

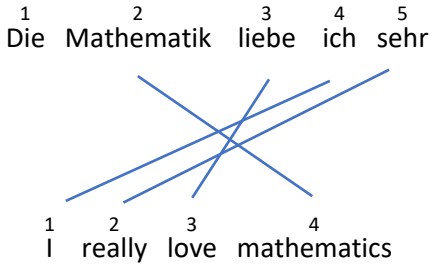
Let's explore this in a couple of examples: In Figure 2 the alignment is straight forward, as every SL word can be aligned to a TL word.



$a: \{1 \rightarrow 3; 2 \rightarrow 2; 3 \rightarrow 1\}$

Figure 2: Every SL word has a TL word alignment

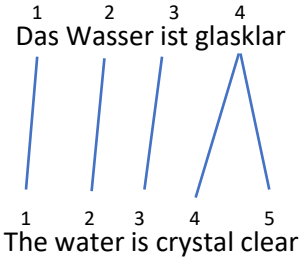
But let's look at some examples where this is not the case. In Figure 3 we see that the article in front of *Mathematik*, which is common in German, is missing in English. This was fittingly dropped in the translation and therefore no word aligns to the female article *Die*; it can simply be dropped in the alignment since the function is mapping **English to German;** i.e. the mapping happens opposite to the translation direction**.**



$$a: \{1 \rightarrow 4; 2 \rightarrow 5; 3 \rightarrow 3; 4 \rightarrow 2\}$$

Figure 3: Missing article in the TT

In Figure 4, we have more words in the SL text, because the German compound *glasklar* is expressed in two words in English. We can align both position *4 (crystal)* and position *5 (clear)* of the TL to the position *4 (glasklar)* of the SL. Note, that it is not possible to do it the other way around (i.e. have one English word align to two German words; this can be a major problem down the road of course, and will have to be solved with additional models[23]).

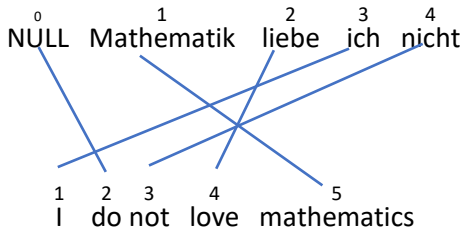


$$a: \{1 \rightarrow 1; 2 \rightarrow 2; 3 \rightarrow 3; 4 \rightarrow 4; 5 \rightarrow 4\}$$

Figure 4: ST word with multiple alignment in the TT

Lastly, in Figure 5 we see that the position 2 (*do*) of the TL text does not have an equivalent in the SL, as this verbal construct is not necessary for a negation in German. Also, (*do*) does not correspond to (*nicht*) in any way (i.e. has a very low probability for aligning to *nicht*), however the position still needs to be mapped to something. For these occasions a special NULL token is introduced for any word that is left over.

[23] Namely the fertility model.



$$a: \{1 \rightarrow 3; 2 \rightarrow 0; 3 \rightarrow 4; 4 \rightarrow 2; 5 \rightarrow 1\}$$

Figure 5: Less words in the ST

All in all, this function makes it possible to have word-to-word alignments, SL word to multiple TL word alignments, no and dropped alignments and the so-called NULL token alignments (see, Koehn 2010:84-85 for a more elaborate explanation).

But how are these alignments found? In order to find the most probable alignment of words, an algorithm called the **Expectation Maximization algorithm** (**EM algorithm**) is adopted and used with the data of a parallel corpus.

The EM algorithm is an iterative learning method that approaches the most probable result in alternating steps. It works as follows (taken from Koehn 2010:88, with additional annotations):

1) Initialize the model, typically with uniform distributions (all hidden variables are filled with equal values). Random distribution is also possible.
2) Apply the model to the data (expectation step).
3) Learn the model from the data using maximum likelihood estimation (maximization step; the values are changed according to occurrence in the data).
4) Iterate steps 2 and 3 until convergence.

Letting a computer elaborate data in a way like this is commonly called **training** and the results of such a training are then saved as a **model**. Once convergence is reached and therefore word-pairs are defined on a statistical basis, we have trained an **alignment model**. Note that there have been several different approaches to alignment in SMT over the years and this is just one of the possible ways to achieve a simple word-alignment.

4.1.2 The translation model

With an alignment model trained, the system may start counting the most likely matches of SL to TL words in the provided data using the **maximum likelihood estimation** (as seen in step 3 of the EM-algorithm) and from those results create a **translation model**. The resulting translation model for a word-based SMT is basically a probabilistic bilingual dictionary (M.-T.

Luong 2016:4) that may be represented in lexical translation probability tables as shown in Table 1 (taken from Koehn 2010:84).

Table 1: Example of probabilistic dictionary (Koehn 2010:84)

das		Haus		ist		klein	
e	$t(e\|f)$	e	$t(e\|f)$	e	$t(e\|f)$	e	$t(e\|f)$
the	0.7	house	0.8	is	0.8	small	0.4
that	0.15	building	0.16	's	0.16	little	0.4
which	0.075	home	0.02	exists	0.02	short	0.1
who	0.05	household	0.015	has	0.015	minor	0.06
this	0.025	shell	0.005	are	0.005	petty	0.04

Since this dictionary is taken from "real" sentences, the dictionary does in fact provide the corresponding surface forms of the words, i.e. inflected words / the morphology. While this sounds very promising and indeed makes basic SMT systems work with very little effort (i.e. seemingly no need for hand-crafted morphological rules), in the long run it is a shortcoming of simple word-based systems as the different surface forms of words might lead to data sparsity, which in turn leads to a less than ideal statistical interpretation of data (see Vuong et al. 2015). The first models presented by IBM do indeed suffer from this problem but do work as a showcase for how a simple word-based SMT can generate valid translations.

4.1.3 The IBM models

With the alignment model and the translation model in mind, we may now look at a simplified representation of the first working SMT model originally proposed: The IBM Model 1. The algorithm trains an alignment and translation model to subsequently translate a new SL-sentence in two basic steps:

In Figure 6, the first step, the length of the translation as well as the mapping of the ST-words to the TT-positions is chosen.

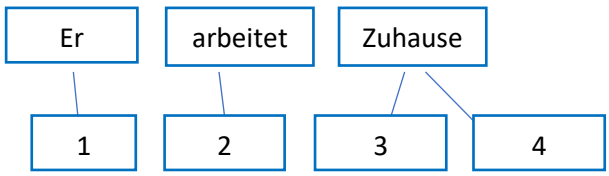


Figure 6: Alignment step

In Figure 7, the second step, it produces a translation by selecting the best translation for each ST-word according to the translation model (i.e. the probabilistic bilingual dictionary).

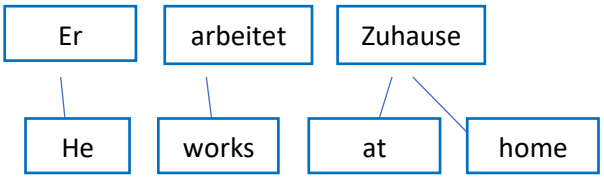


Figure 7: Translation step

For this translation no reordering was necessary and in fact, no reordering takes place with IBM Model 1. While a multiple alignment is present, the single constituents for *(Zuhause), (at)* and *(home),* just happened to fall into the right place in this case. However, for IBM Model 1 the following translations would be just as probable: 'At home he works', 'Home works at he' or any other combination.

To solve this and many other evident issues, IBM proposed 4 further models, that all build on each other and refine the translation by mostly improving the alignment model and a better reordering structure through additional statistical modelling. In total, IBM originally proposed five models (as taken from Koehn 2010:96-97):

- IBM Model 1: lexical translation;
- IBM Model 2: adds an absolute alignment model (alignment probability distribution);
- IBM Model 3: adds a fertility model (distortion probability distribution);
- IBM Model 4: adds a relative alignment model;
- IBM Model 5: fixes deficiency.

In simple terms this means:
The first model is pure lexical translation according to the probabilistic translation table.
The second model adds a statistical model for better reordering of translated words.
The third model adds a statistical model that expresses the probability of certain SL words generating multiple (or NULL) TL words, which is called the *fertility* of a word.
The fourth model further refines the reordering by adding a statistical model that takes some context into consideration.
The fifth model fixes a problem of all prior models called *deficiency,* where multiple output words in the TL would be placed on the same position, effectively wasting probability mass on impossible alignments.

4.1.4 The language model

To ensure a fluent output, SMT systems adapt an additional probabilistic model, called the **language model (LM)**. Unlike the alignment or translation model, this model is built exclusively on mono-lingual data. The model essentially counts the instances of words occurring together and thus learns to prefer a certain word order and selection. This can help in achieving more "natural" sounding translations. The LM should therefore prefer something with a correct order over something with non-sensical or unnatural order:

$$p_{LM}(He\ works\ at\ home) > p_{LM}(Home\ he\ works\ at)$$

Equation 5: LM prefers correct word combination

It also helps resolving some issues with ambiguity. Ambiguity may easily crop up with the probabilistic dictionary when words have equal or similar probability within the used text data. Take for example the translation for the German word 'Hahn'[24]: 'Rooster', 'faucet' and 'valve' are all valid translations for the single word and may appear with the same or similar probability in data. Let's assume we have the German phrase 'Drehen sie den Hahn' or 'Öffnen sie den Hahn'; while it is a relatively common sequence of words in German, it is unlikely we would encounter a sentence that spells 'turn the rooster' or 'open the rooster' in English. A well-trained **language model (LM)** would therefore give a higher score to something like 'turn the valve' or 'open the faucet'. Since LMs are mono-lingual, they are usually built only from TL sentences, that may also be sourced from additional text resources in that same TL, rather than just the parallel data used for the general training. More data generally results in higher statistical precision.

Commonly so-called **n-gram language models** are used for the language modeling (Koehn 2010:95), where the *n* stands for the amount of words included in the statistics. Essentially, the language model is then a function that predicts the probability for a certain word e_n to appear in the context of *n* words in a specific language. For example, a **trigram language model** rates the probability of a certain word *n* to appear after two specific words prior to that (e_{n-2}; e_{n-1}). The more data is used for training the LM, the longer the n-gram model can theoretically be made; and longer n-gram models make for better *fluency*. However, it is necessary to keep n-gram models relatively short in order to get usable statistics, as longer word sequences have a smaller likelihood to be found in data. Commonly the trigram language model is used but it can be supplemented by longer n-grams (Koehn 2010:183). As mentioned, the training of language models commonly consists of at least the TL-part of the parallel corpus

[24] This example was chosen as a homage to the mascot of the Austrian translator's association *UNIVERSITAS*. https://www.universitas.org/de/ueber-uns/#getsub_p3 (Accessed April 10, 2019)

but can also include separate text resources in the TL (For an in-depth explanation of how language models are built, see Koehn 2010:95, 181-214).

Equation 6 through Equation 8 put all of this in formulaic terms. The probability *p(e)* for the TL word *e* therefore represents the likelihood of *n* words appearing together in a specific order and context.

$$p(e) = p(e_1, e_2, \dots, e_n)$$

Equation 6: Basic function for language model p(e)

This probability prediction can be broken up into single word predictions using the **chain-rule:**

$$p(e_1, e_2, \dots, e_n) = p(e_1)p(e_2|e_1) \cdots \mathrm{p}(e_n|e_1, e_2, \cdots, e_{n-1})$$

Equation 7: Chain-rule applied to p(e)

Which would look like Equation 8 for a trigram model:

$$\mathrm{p}(e_1)\mathrm{p}(e_2|e_1) \cdots \mathrm{p}(e_n|e_{n-2}, e_{n-1})$$

Equation 8: Trigram language model

This type of model, which goes through a sequence (in this case words), while considering only a limited number of steps (for a trigram it is three steps), is known as a **Markov chain**. One key issue this kind of model has, is that it assumes that the probability of the word e_n depends only on the limited history before it, but in language we know that longer dependencies in sentences can equally influence the likelihood of a certain word appearing later in the sentence[25]. The inherent problems and the mathematical formulation aside, what a trained LM essentially does, is predict how likely the specific word e_n is to appear after *n* words and this prediction can be used to give a score to the translated phrases. This scoring can be applied to all of the possible translations found by the translation model, giving high scores to coherent and 'natural-sounding' translations, i.e. such translations with statistically more likely word order and context, and lower scores to less likely and therefore 'unnatural' translations. In order to create an SMT system with somewhat consistently natural sounding results therefore at least 3 models need to be trained on available text data:

1) an alignment model
2) a translation model
3) a language model

[25] Take, for example, the sentence: "I was raised in Japan and therefore I am fluent in Japanese"; in a sentence like this, the long-term dependency of "raised in Japan" and "fluent in Japanese" would not likely be captured by an n-gram model.

4.1.5 Combining the models: The noisy-channel model

To efficiently apply the language model in an SMT system, it is advantageous to combine it with the translation model. This may be achieved by applying the Bayes rule to add the language model *p(e)* to the translation model *p(e|f)*.This results in the following Equation 9:

$$argmax_e p(e|f) = argmax_e \frac{p(f|e)p(e)}{p(f)} = argmax_e p(f|e)p(e)$$

Equation 9: Noisy-channel model

This way to combine the language and translation model results in the so-called **noisy-channel model** and in fact, IBM's SMT models are referred to as an instance of the noisy-channel model (Collins 2011:2). What this basically does, is apply the knowledge we have about our TL, thanks to the LM *p(e)* and also the knowledge about what sort of 'distortion' might be possible when going from SL to TL thanks to our translation model *p(f|e)* (note that the mathematical direction here is actually TL to SL, opposite to the translation direction). As Koehn jokingly puts it, we basically assume that the foreign speaker wants to say something in the TL (message *e*), but the message gets distorted in a noisy-channel and out comes a sentence in the SL (message *f*) (Koehn 2010:96) as seen in Figure 8. Thus, what actually happens in the noisy channel model is a 'reconstruction' of the actual message *e* (which is assumed to be uttered in the TL) from the message *f* that was 'received' in the SL. [26]

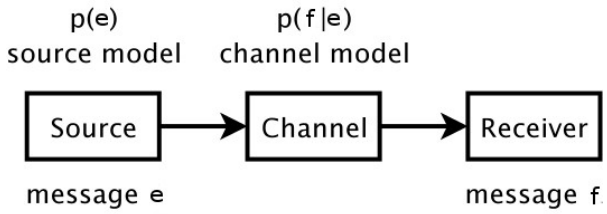


Figure 8: The noisy-channel model (diagram taken from Sokolov 2015)

We saw until now, that SMT is basically a combination of several statistics-based models combined to form the SMT model; since an output is generated at the end of each model and in fact at the end of the whole process, this type of modelling is called **generative modeling**. Generative modeling offers the advantage, that it is possible to break the translation problem up into several simpler problems, as each model generates its own output and we have the

[26] Interestingly, this idea is very similar to Warren Weaver's conclusion regarding translation in one of his 1947 letters: *'When I look at an article in Russian, I say 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode'."*(quoted in Weaver 1949:4) and indeed the interpretation of what translation was to most pioneering MT researchers.

ability to change only parts of the whole SMT model by keeping estimation and model definitions independent from each other (see Sokolov 2015). This allows to pinpoint translation errors to individual steps within the training of the SMT-Model. Additionally, expanding the whole SMT pipeline by adding other models that refine the alignment, translation or ranking tasks is possible. The disadvantage is that every model has the same importance and the addition of more models is not easily achieved; so, for example, putting an emphasis on better fluency by putting more weight on the language model or even adding a second language model is not possible (Sokolov 2015:25).

4.1.6 The limits of word-based SMT

While all these concepts are key to the advancement of SMT (as the IBM-researchers put it: "The lesson to be learned (...) is that simple, statistical methods can be surprisingly successful in achieving linguistically interesting goals." (Brown et al. 1993:2)), the underlying idea of word-based translation itself is a major bottleneck when trying to approach translation in general. In a sense, Brown et al. realized this, stating that the aim of SMT was never to replace the linguistic effort, but only to provide another approach in the machine translation paradigm.

Coming from Translation Studies, one of the first things one learns is to watch out not to stick to the exact words too much and the same can be said for machine translation. Aside from the fundamental problem of the lack of a proper 'equivalence' between languages, the word-based approaches have several other issues to contend with, even from a purely statistical point of view: The need to have a word to word or word-to-multiple-word correspondence ('solved' by the arbitrary addition of a null-token and the concept of *fertility*) is an inherent flaw of the system and while language models provide reasonable results when confronted with contextual ambiguities, n-gram models can still often not account for special translation of words in context. This, of course, is something that linguists as well as translators would know all too well, given that meaning of words changes drastically depending on the context (idioms are a prime example, but also prepositions pose a big problem for word-based approaches).

Taking idioms as an example, "Hals- und Beinbruch!" would likely translate to "Break a leg!", meaning "Good Luck!"; however, since there has to be an alignment between the words, if a word-based model was trained on this parallel text, "Hals", meaning "neck" would likely be aligned to "break", "a" might be aligned to "und" and the composite word "Beinbruch" (meaning the "breaking of a leg") would likely be aligned to leg. This would result in a quite nonsensical dictionary that would also not have a high probability in any bigger parallel corpus. For that reason, it was clear that there was still much work to be done. Brown et al. theorized of bringing more linguistic rules into the mix (Brown et al. 1993:296). However, it was still by

looking at the problem from an IT/data-engineering point of view, where researchers quickly found that limiting themselves to the word as the atomic unit for meaning might not be the best option and therefore the phrase-based approach was developed.

4.2 Phrase-based SMT

We saw that word-based SMT quickly falls apart with longer sentences and complex interrelations between these words, especially if the count of words changes from ST to TT.

To solve these issues, phrase-based models segment the ST and TT into several so-called **phrases**. This is the approach that several MT providers, like Google, Microsoft or SYSTRAN, employed for their machine translation services until the switch to neural network systems occurred in 2016. The alignment for the single words composing the phrases is basically still the same as we saw in the word-based models**.** Och and Ney found that models with a first-order dependence (i.e. a bigram model) and a fertility model (i.e. Model 3 and Model 4) give better results than more simple models like the IBM Model 1 or 2 (Och & Ney 2000:1), while Koehn et al. found that Model 2 already provides similar performance to the much more complex Model 4, as long as the right alignment heuristics are chosen (Koehn et al. 2003:7). Whichever model is used to define the initial word-alignment, phrases are extracted from the resulting alignments. **These phrases are not linguistically motivated**; in fact, it can be any multiword unit. Limiting extracted phrases to linguistic phrases has even been proven to yield lower quality translations (see Koehn et al. 2003). In the translation process, the extracted phrase-pairs allow translation of ST phrases into TT phrases, with a final reordering of the phrases. To see how a standard phrase-based model operates, let's look at the example provided by Koehn in Figure 9 (Koehn 2010:128).

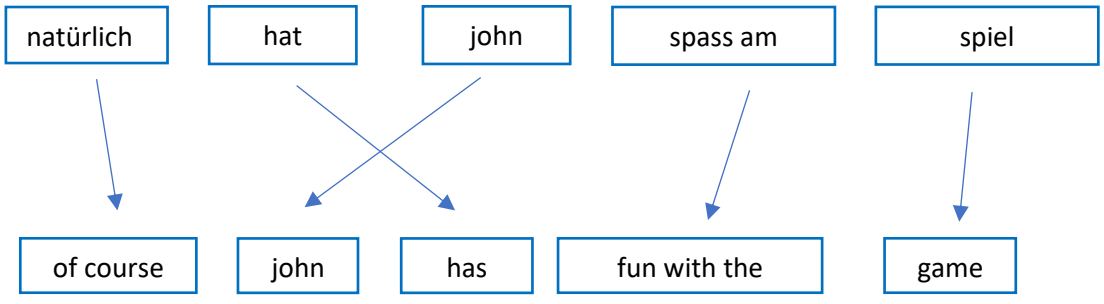


Figure 9: Basic phrase-based SMT

We can see that with phrase translations we remove the need for the concept of fertility and/or deletions/additions like null-tokens. Some single ST-words, like "natürlich", that are best

translated into multiple words ("of course"), are simply mapped as an ST-phrase to another translated phrase in the TT, doing away with the need of single to multiple unit mappings. Looking at the phrase-pair "spass am" and "fun with the" we can also see what was shortly mentioned above: the phrases are not linguistically motivated. Most syntactic theories would likely classify "spass" / "fun" as a noun phrase and "am Spiel" / "with the game" as a prepositional phrase. However, learning the translation of "spass am" as "fun with the" is more useful data for statistical translation as German and English prepositions do not match very well. Looking at the statistical occurrence of "am Spiel" would likely result in a mis-translation of the preposition "am", resulting in something like "at the game". On the other hand, the statistical data for "spass am" would likely converge to the correct but less likely translation of "am", e.g. "with the". In a word-based SMT "am" would not have a high chance to be translated as "with the" at all.

Recapping, two major pitfalls of the word-based approach are solved with phrase-based SMT:

1) Words are no longer necessarily the smallest unit in a sentence, so single word to phrase alignments are equally possible in both directions, solving the issue of single to multi-unit mappings. This also alleviates the need for insertion, deletion and the concept of fertility.
2) By translating word-groups instead of single words, we often manage to clear up ambiguities that would result in a mis-translation in word-based SMT.

A generative phrase-based model therefore starts out 'simpler' than word-based models would, as there are less evident problems to work around. Additionally, the larger the parallel text corpora, the more likely we are to find phrase-pairs that can be learned and, ideally, we might even be able to learn full sentence pairs from a parallel corpus. So even a simple phrase-based SMT will work better than most word-based translation models (Sokolov 2015:23; Koehn 2010:136).

For reordering the phrases, a new **distance-based reordering model (*d*)** is introduced, which is formulated quite similarly to the **alignment probability distribution** in the IBM Model 2. The ST is considered for calculating how expensive a reordering is, i.e. how many words *x* a phrase has to be moved so that a given ST phrase aligns to the matching TT phrase:

$$x = start_i - end_{i-1} - 1$$

Equation 10: Movement cost function

In Equation 10 $start_i$ stands for the position of the first word in the ST phrase corresponding to a TT phrase at the *i*th position and end_{i-1} stands for the last word of an ST phrase corresponding to the previous TT phrase.

While it would be possible to learn the statistical probability of this cost, in most phrase-based models data is not used to train the reordering model, but instead a fixed exponentially decaying cost function is applied:

$$d(x) = \alpha^{|x|}\ ;\ 0 < \alpha < 1$$

Equation 11: Distance-based decaying cost function

This approach makes longer distance reordering more expensive and therefore less likely and as such would not work well for language pairs with big syntactic differences. However, that is alleviated by the fact, that phrase-based SMT usually works with **weighted models** and**,** as we will see later, that allows a **language model** to "override" the basic reordering model if needed.

4.2.1 The phrase translation table (phrase-based translation model)

Learning a phrase translation table (the translation model/lexicon model for phrase-based SMT) generally happens in three stages: 1) word alignment, 2) phrase pair extraction and finally 3) phrase pair scoring.

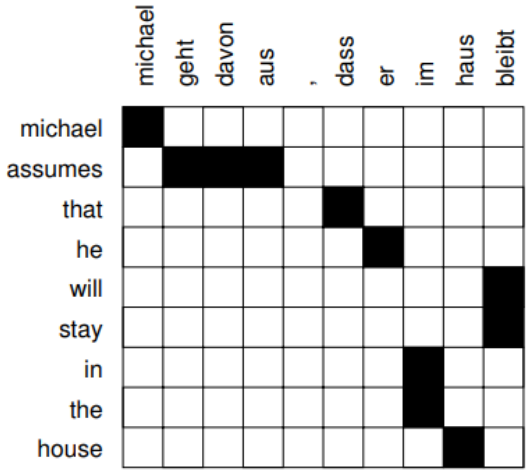


Figure 10: Building a phrase translation table - Stage 1: Word alignment (Koehn 2010:134)

Figure 10 shows the word alignment represented in a table. With this knowledge, the machine can now extract phrase pairs given specific conditions. The most important condition, which is part of basically every phrase-pair extraction process, is that of **consistency**. Basically, every word inside of an SL phrase must align to a word within the targeted TL phrase. Unaligned

words may also be included since they don't break the notion of consistency, as they have no other alignment point outside of the targeted phrase. This is needed to capture commas or articles that are nonexistent in the TL. Since these SL words are not explicitly aligned to any TL words, unaligned words (and punctuation marks) within a phrase-pair may lead to multiple translations (in the example above, we have the German comma being unaligned).

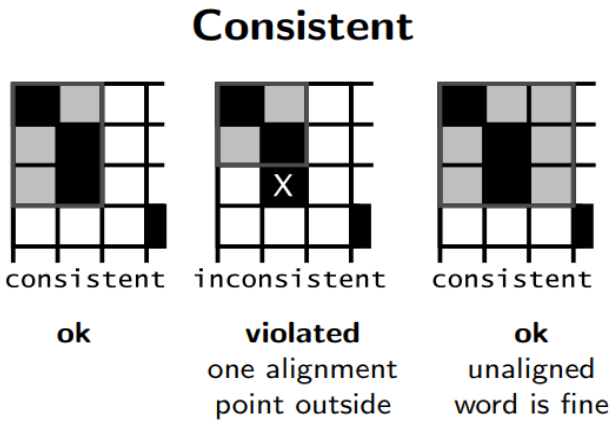


*Figure 11: The grey boxes symbolize the to be extracted phrase. In the first example, all words are aligned within the phrase and are therefore consistent. The second phrase would not be extracted, as one word would be aligned outside the phrase. The last box shows a phrase incorporating an unaligned word. (*Koehn 2010*:132)*

Let's examine the example by looking at Figure 12 for the second stage of the table creation:

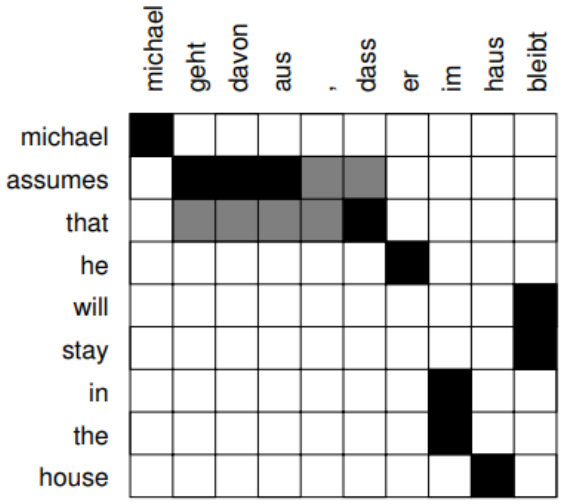


*Figure 12: Building a phrase translation table - Stage 2: Extraction (*Koehn 2010*:131)*

In this case "geht davon aus, dass" and "assumes that" may be extracted as a phrase pair, as all these words are aligned to each other. Other possible alignments of the marked words can be seen in Table 2 (Koehn 2010:134).

Table 2: Possible alignments

TL	SL
assumes	geht davon aus / geht davon aus ,
that	dass / , dass

Looking at the whole sentence, however, we may extract a lot of different pairs, shorter ones (Table 3) and longer ones (Table 4) (also taken from Koehn 2010:134):

Table 3: Short phrase pairs

TL	SL
michael	michael
assumes	geht davon aus / geht davon aus ,
that	dass / , dass
he	er
will stay	bleibt
in the	im
house	haus

Table 4: Longer phrase pairs

TL	SL
michael assumes	michael geht davon aus / michael geht davon aus ,
assumes that	geht davon aus , dass
assumes that he	geht davon aus , dass er
that he	dass er / , dass er
in the house	im haus
michael assumes that	michael geht davon aus , dass
michael assumes that he	michael geht davon aus , dass er
michael assumes that he will stay in the house	michael geht davon aus , dass er im haus bleibt
assumes that he will stay in the house	geht davon aus , dass er im haus bleibt
that he will stay in the house	dass er im haus bleibt / dass er im haus bleibt ,
he will stay in the house	er im haus bleibt
will stay in the house	im haus bleibt

On closer inspection it can be observed, that the shorter pairs are indeed not so different from word pairs. Yet they retain their usefulness, as shorter phrases will occur more frequently and therefore are going to be more applicable to hitherto unseen sentences during a translation process. Longer phrases on the other hand contain a lot more local context and can eventually

help with the translation of bigger, interrelated parts of a text. In theory at least, extracting all possible phrases was thought of improving the translation quality. For that reason, instead of just simply relying on the total count of occurrences, the length of the extracted phrase is considered, so the **frequency of appearance relative to the length of the phrase** is scored instead of just the absolute frequency of appearance. This way, longer phrase pairs are also likely to be selected during the translation process even though they may appear less often than the shorter pairs they are composed of.

Of course, having such a large translation table also significantly increases the memory requirements for SMT. For large parallel corpora with millions of sentences, the extracted table may well be several gigabytes in size. Luckily, disk storage capacity was one of the fastest growing aspects of computer hardware and as such, even if the table could not be loaded completely into memory (RAM), it was possible to efficiently estimate the probability distribution by storing and sorting the extracted phrases on disk (Koehn 2010:136). It therefore would not pose a real problem, except for less performant and/or mobile devices like PDAs and later smartphones/tablets. These changes in computer hardware trends led researchers to question the common wisdom of extracting all the possible phrase-pairs to use in the translation table.

Quirk and Menezes later pointed out, that extracting only the shortest phrases that map a whole sentence, would not hurt performance (Quirk & Menezes 2006). Indeed, it might even help with performance, as unlikely solutions could be excluded from the search. This so-called **pruning** of the extracted data, essentially deleting certain phrases according to predetermined or statistical conditions, was therefore considered a viable option by several researchers; the n-gram based variant of phrase-based SMT was also founded on this idea of reducing the extracted phrases (see Mariño et al. 2006; Costa-Jussà & Fonollosa 2007).

While phrase-based models were developed, it was also soon found, that putting more emphasis on the results of a certain model might help with the quality of the output. If, for example, *fluency* was insufficient, it would seem helpful to put more emphasis on the language model to provide a more natural and fluent output. In the standard generative model however, all the models have the same importance or 'weight' and would therefore not easily allow adding or making good use of additional models. For that reason, phrase-based SMT moved away from generative modeling, like what was used in the noisy-channel model, to a new modeling strategy, which is 'borrowed' from the machine learning domain: the **log-linear modeling** or **weighted models.**

4.2.2 Weighted models (log-linear modeling)

Log-linear models introduce the concept of weight functions to the modeling process. Essentially weight functions make it possible to give some elements within an SMT system more 'weight' or influence on the results than other elements have. This holds true when combining the separate elements by performing a sum, integral, or average. In simpler terms, this enables the SMT system to place higher emphasis on a particular model, for example, prioritizing correct word order by giving a higher bias or weight to the language model. By this point, phrase-based SMT essentially also consists of three separate models:

- The phrase translation table $\phi(\bar{f}_i|\bar{e}_i)$
- The reordering model d
- The language model $p_{LM}(e)$

However, instead of combining them with generative modeling as before, where each model generates its own output and gives an absolute result ("what is the BEST solution"), the models here are treated as so-called feature functions and each data-point (i.e. word or phrase translation) is fed into the system as a feature vector. The result is therefore a probability distribution that can be influenced by the weights ($\lambda_\phi, \lambda_d, \lambda_{LM}$) and as such can no longer be seen as an absolute result. In Equation 12 we can see the formulaic expression of such a weighted model.

$$p(e,a|f) = exp\left[\lambda_\phi \sum_{i=1}^{I} log\, \phi(\bar{f}_i|\bar{e}_i) + \lambda_d \sum_{i=1}^{I} log\, d(a_i - b_{i-1}) + \lambda_{LM} \sum_{i=1}^{|e|} log\, p_{LM}(e_i|e_1, \dots, e_{i-1})\right]$$

Equation 12: Log-linear phrase-based model

This way of modeling gives the benefit of control by managing the importance of the separate models. The models remain independent from each other and can thus be trained individually. It also makes it much easier to add other, yet again independent models to the system, without disrupting the contribution of the other models in the pipeline.

It was a big step forward for SMT, as this way of modeling would also allow rules and other more deliberate translation tactics to be applied in the system. In retrospect, however, it may also be one of the weaknesses of the system, as managing many separate models trying to achieve the same thing can be very hard and time-consuming.

Yet, log-linear modeling has and still is widely used in the machine learning community. As we will see in Chapter 5, NMT is based on the very same principle. In fact, Koehn also allures to this in his 2010 Book about SMT, when mentioning the **Perceptron learning methods** as an example for log-linear modeling (Koehn 2010:138).

4.3 Tree-based SMT

As seen in word-based SMT (Section 4.1) and phrase-based SMT (Section 4.2), both approaches were created without really taking linguistic rules into consideration for the translation. Without additional models, next to no consideration is given to linguistic concepts like word-classes, cases, flections and so on. While linguistic rules could be added as feature functions in the log-linear modeling, the whole pipeline itself was not designed to take linguistic rules into account. Relatively late in the existence of SMT a more linguistic point of view was taken and an attempt to build SMT models on top of linguistic ideas was made.

As a result, a new form of SMT emerged, where the idea of syntactic trees and the notion of formal grammar was worked into the foundation of the statistical system by SMT researchers. This way, the models for statistical translation would no longer only operate on flat sequence representations of sentences (i.e. strings of words) but would also be able to statistically capture the syntactic relationships between words and phrases by considering the word-classes and connections of the words; essentially the grammar structures of the sentence. This method was coined **tree-based SMT**. In order to extract such information from sentences and represent it as a tree diagram, linguistic parsers have been worked on since the early 1990s like, for example, the hand-crafted **Penn tree bank** (Koehn 2010:47) or the statistical parser by Michael Collins (see Collins 2003)[27]. These parsers represent sentences as trees, motivated by formalisms that are called **grammars**; for SMT even sentence pairs may be represented as an aligned tree pair, in such a case the formalisms are called **synchronous grammars**.

[27] The source-code for the parser is freely available at: http://www.cs.columbia.edu/~mcollins/code.html

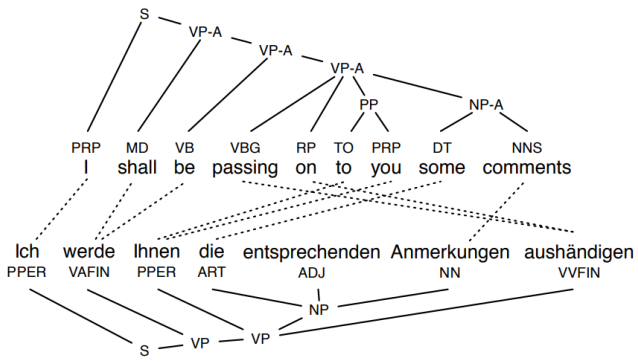


*Figure 13: Two phrase structure grammar trees with word alignment for a German-English sentence pair (*Koehn 2010*:334)*

First attempts proved promising, with tree-based SMT systems performing similarly or slightly better than some state-of-the-art phrase-based systems (Koehn 2010:331).

However, the jump in translation quality was not evident enough to warrant the amount of effort to manage the extracted translation and grammar rules. This made them less viable than a more automatic phrase-based system. Tree-based SMT therefore remained mainly in the academic realm, never really catching on with commercial use. Phrase-based SMT remained the de-facto standard in MT applications until the wide-spread adoption of NMT systems.

4.4 Translation as decoding

In the three prior sections, we have seen how word-, phrase-based and by extension tree-based SMT systems find the probability for certain translations of an SL word or phrase to occur in the TL. The models then provide translation suggestions or hypotheses for the individual parts of the whole sentence. This is the **encoding** part of the SMT approach. However, while it may seem trivial to then just pick the best translation out of all the suggested solutions, the process of finding the best combination of translation hypotheses and the best reordering for the whole sentence is one of the hardest parts in SMT, because there is an exponential number of choices given a specific input sentence (Koehn 2010:155). This means, that while good **decoding**, i.e. the search for the best scoring sentence translations, is crucial for optimal translation quality, it would be computationally too expensive to really look through all the possible translations to find the optimal result; even for input sentences of modest length.

SMT research has therefore either opted for so-called heuristic beam-search [28] algorithms or different channel models (Knight 1999:615). The problem therein lies in the fact that these approaches are not optimal searches and do not guarantee that they will find the best

[28] Beam-search essentially keeps a number of top-scoring hypotheses and continues the search for the best hypothesis by following each choice individually. This is in contrast to the so-called greedy search, where only the highest probability solution according to the model is used.

translation the models may have proposed. Because of the heuristic nature of the decoding, so called search-errors may lead to failure in finding the best translation according to the models in the SMT pipeline. In fact, the more models are added and the bigger the data within the models grows, the harder the decoding will be. Conversely, the statistically most probable translation of all the combined phrases, i.e. the best translation according to the models, might not be a good translation at all, as SMT fails to capture long-distance relationships of words within the whole sentence. These issues are often cited as the main reasons why SMT translation quality was stagnant even though a lot of research effort was put into the systems (Chiang et al. 2009; Galley & Manning 2008; Green et al. 2013).

4.5 SMT today (end of 2019)

Many large companies like IBM, Microsoft and Google, but also formerly rule-based machine translation providers like SYSTRAN, started to implement the probabilistic SMT algorithms in the early 2000s to provide efficient automatic translation services for a vast number of languages[29]. Most companies used SMT in a very controlled environment and domain, where statistical likelihood was strongly favored by the controlled language of the training corpus and the source for the translated texts. Microsoft, for example, used its SMT service for translating their knowledge-base into several languages from an English source text. However, anyone who has ever used Google Translate prior to the year 2016, probably recalls that it often delivered hilarious results, especially once the sentences got longer or grammatically more complex and ambiguous.

Long sentences illustrate two major weaknesses that SMT systems have. The first weakness is that decisions are locally determined, as they translate phrase-by-phrase or word-by-word and so long-distance dependencies are often ignored or insufficiently captured. Secondly, the decoding for longer sentences would get exponentially harder, as more and more possible solutions would need to be searched; sub-optimal search results are therefore the logical consequence.

This is especially detrimental for languages that follow completely different grammars and word-order, as would be the case between English (an analytic language) and Japanese (an agglutinating language). Additionally, as was allured to in the introduction and should have become sufficiently clear when looking at the several models described for even the simplest of SMT systems, the entire SMT pipeline was becoming increasingly complex as more and more features were added to the log-linear framework (Chiang et al. 2009; Galley & Manning 2008; Green et al. 2013).

[29] Google today claims to support over 100 languages. https://www.blog.google/products/translate/ten-years-of-google-translate/ (accessed May 03, 2019)

5 Neural Machine Translation (NMT)

This chapter provides a deeper look into the current state of the art machine translation approach, neural machine translation (NMT) and the artificial neural networks behind the method. In a way, NMT is similar to SMT in that it is corpus-based machine translation or data-driven machine translation. Just like SMT, it is trained on huge corpora of parallel texts (ideally sentence-aligned translations). What differs, however, is the computational approach: the whole translation task is performed by artificial neural networks. As Luong summarizes in his thesis about NMT, the big advantage this new approach has over SMT is that the whole translation process can be contained in one single neural machine learning model (Luong 2016:7). This means that in theory each step of the translation process, which in NMT is usually performed on a sentence-level, can have full access to the information of the specific words in that sentence and find the most likely translation of the sentence by mathematically analyzing each word in relation to all the other words of the sentence.

While NMT is still corpus-based, just like SMT, and does in fact rely on statistical probabilities in the final classification step, the decision making and learning of translation probabilities is quite different and part of the reason why NMT is often used in conjunction with the term AI, or artificial intelligence. But how do these neural models look like and why are they often related to AI and coined with this intriguing term "neural"?

5.1 A closer look

Neural machine translation is based on so-called artificial neural networks (or ANNs, for simplicity, from now on I will mostly stick to the term *neural networks*), a term that may summon quite scary thoughts for most people, especially with all the talk of AI taking over several fields of hitherto human expertise. And while recent achievements in machine learning technologies are in fact quite impressive, we should take a short look at the rather long history of neural networks to understand why such an ominous name was chosen and just why this seems to now basically have made any other form of AI research obsolete. Andrey Kurenkov shared an excellent and easy to read overview in four parts of the history of ANNs on his web-site[30]. In order to give readers an overview and in an attempt to demystify ANNs, I would like to provide a short historical overview myself, based in part on Kurenkov's work.

[30] https://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/ (accessed May 15, 2019)

5.1.1 Linear regression and the Perceptron

It is important to remember that artificial neural networks, while only lately coming to be at the center of research attention, do have a long history in academic research. In fact, artificial neural networks are nothing new, the main principle they and machine learning in general are based on is well over 200 years old! Let's take a look at the following graph in Figure 14.

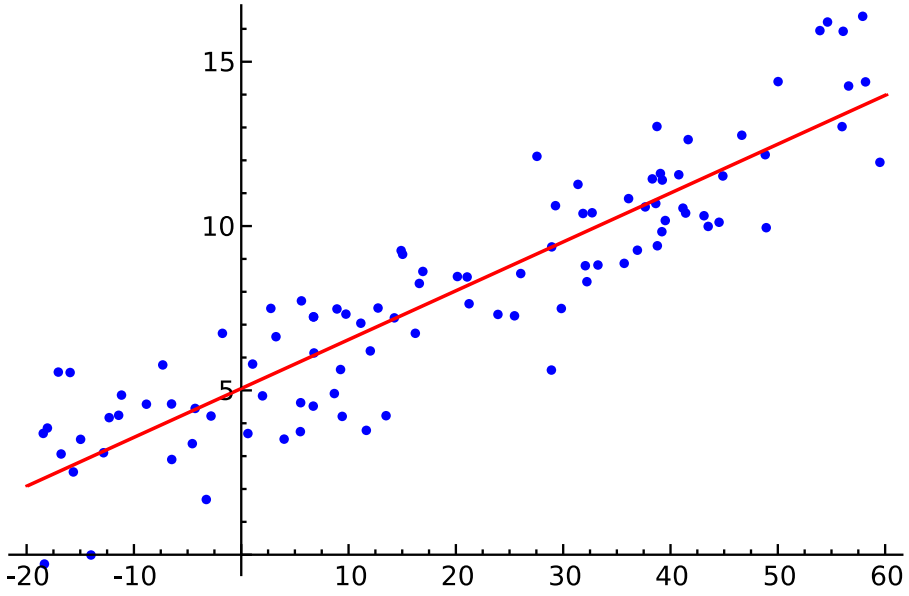


Figure 14: Linear regression[31]

The red line basically shows a general function that best approximates the location of the dots on the graph and therefore the relation between pairs of input values (x) and output values (y). What we are seeing here is a visual representation of a linear regression, a technique from statistical mathematics that was introduced well over 200 years ago by Legendre (1805) and Gauss (1809) (Yan & Su 2009:2). The great thing about this technique is that we can extrapolate a general function for data that is easy to observe but would have an incalculable amount of functions behind it and formulating equations directly for these would be very hard. As Kurenkov writes, this generalization would be very useful for finding a function that maps, for instance, the input of a recorded spoken word to the written output of said word. However, it is clear that the linear regression as we see it in Figure 14, is too simple to handle such a complex problem. Yet, it does show what is essentially **supervised machine learning**: 'learning' a function given a so-called **training set** of examples, where each example is a pair of an input and output from the function (Kurenkov 2015:3). In fact, all of this will appear quite similar to what has been done in SMT with the parallel corpora and indeed this means that also SMT is a

[31] https://upload.wikimedia.org/wikipedia/commons/3/3a/Linear_regression.svg (accessed May 16, 2019)

form of machine learning: After all, the machine learns the probabilities of certain translations reliant on the data provided. Linear regression itself has been used to improve several SMT constituents as well (see Yan & Su 2009; Biçici 2011).

It makes sense then, that one of the very first attempts to make machines learn bears quite some resemblance to the linear regression. One of those attempts is known under the name **Perceptron** and was so coined by the psychologist Frank Rosenblatt, who tried to create a simplified mathematical model of how neurons, in other words the subunits of the neural network in our brains, work (Rosenblatt 1958). As Kurenkov perfectly explains, the model "takes a set of binary inputs (nearby neurons), multiplies each input by a continuous valued weight (the synapse strength to each nearby neuron) and thresholds the sum of these weighted inputs to output a 1 if the sum is big enough and otherwise a 0 (much in the same way neurons either fire or do not)" (Kurenkov 2015:8). Rosenblatt's seminal work was based on the so-called McCulloch-Pitts Model, that showed that a (biological) neuron could in that way, much like a simple logic gate with binary outputs, model the basic OR/AND/NOT functions (see McCulloch & Pitts 1943; Figure 15).

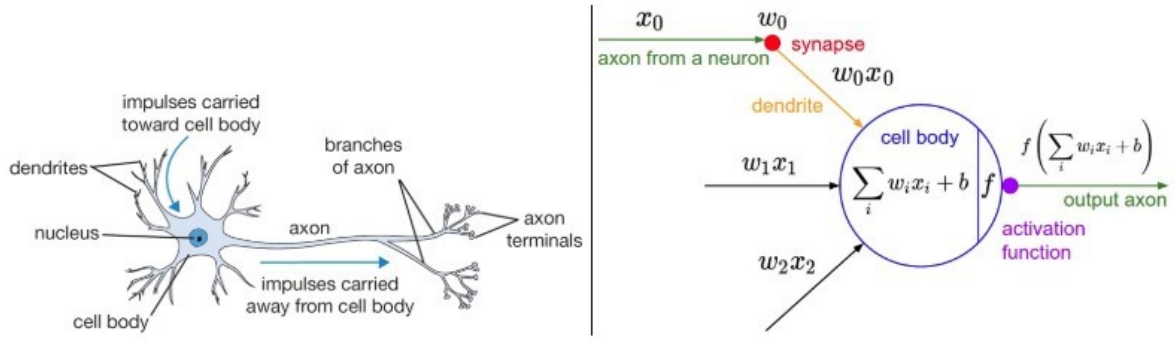


Figure 15: Drawing of a biological neuron (left) and the mathematical perceptron model (right).[32]

However, while the McCulloch-Pitts Model showed that a neuron could model the OR/AND/NOT functions, it did not explain a mechanism for learning. Rosenblatt solved this by making the weights of the inputs updatable: whenever the neuron would provide a result which was known to be incorrect (remember, it uses a known training set of input and output data for the supervised training), the weights are updated to adjust for the error. This in turn was based on a hugely influential theory by Donald Hebb, that stated that knowledge and learning occurs in the brain, primarily through the formation and change of synapses between neurons (Hebb 1950 as found in Kurenkov 2015:9). As Kurenkov describes, the Perceptron's learning algorithm works as follows (Kurenkov 2015:11):

[32] http://cs231n.github.io/neural-networks-1/ (accessed May 18, 2019)

1) Start off with a Perceptron having random weights and a training set.
2) For the inputs of an example in the training set, compute the Perceptron's output.
3) If the output of the Perceptron does not match the output that is known to be correct for the example:
   a) If the output should have been 0 but was 1, decrease the weights that had an input of 1.
   b) If the output should have been 1 but was 0, increase the weights that had an input of 1.
4) Go to the next example in the training set and repeat steps 2-4 until the Perceptron makes no more mistakes.

With this, one Perceptron can learn to provide a specific output, given a specific input. Of course, with only one Perceptron, almost no real-world problem can be solved, but by networking several Perceptrons together, it is possible, depending on the task, to linearly or exponentially increase the amount of possible outputs.

Take, for example, the classic task of recognizing handwritten digits: To be able to classify each of the ten separate digits (fed to the Perceptrons as a digital scan, i.e. certain arrangements of pixels), we need to have 10 Perceptrons networked together. Each Perceptron corresponds to one of the possible digits and is trained to output a '1' whenever the matching input (i.e. a certain arrangement of pixels representing the corresponding digit) is provided. Such a neural network would look a little bit like what we can see in Figure 16.

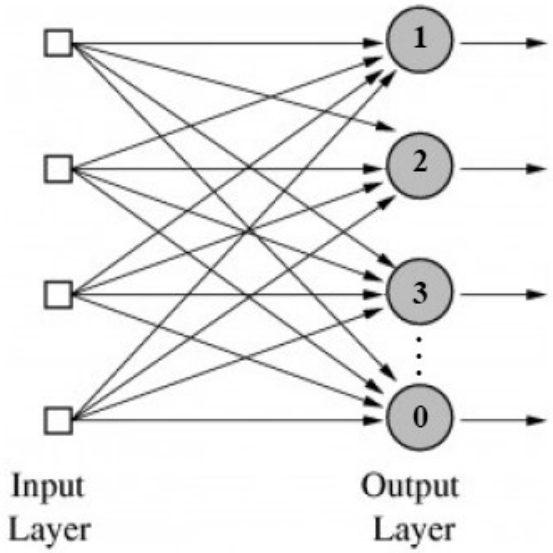


Figure 16: A simple neural network

A problem with the Perceptron and its thresholding activation function is that it does not exactly find the best function for generalizing the data, as it stops just when all problems are solved as expected. While this works for the training data, real data might be more fine-grained (just think at all the possible handwriting variations or in fact possible word combinations in a translation task) and therefore the trained model would fail to properly classify certain inputs. Bernard Widrow and Tedd Hoff made the discovery, that it is not strictly required for the artificial neuron to have a thresholding activation function that simulates the binary nature of biological neurons like the Perceptron does. In 1960, they introduced a different neuron model called ADALINE (see Widrow 1960), where the output was no longer constrained to be a 'digital' 0 or 1 and instead was kept 'analog' so that the fine-grained changes in that data would remain visible. With this 'analog' data, it is possible to measure how much the error changes when each weight is updated. This change of the error is called **partial derivative** and can be used to drive the error down and find optimal weight values.

While this is essentially the right approach to make neural networks learn, two major limitations had to be overcome: First, the basic Perceptron is only able to learn a function for data that is linearly separable, ergo it can only model the functions AND/OR/NOT, but cannot model the simple Boolean function XOR (exclusive OR). In Figure 16 we can see, that it is possible to easily separate the first two examples (OR/AND-functions) with a linear line, but for data that requires an XOR expression this is no longer possible (Kurenkov 2015:23).

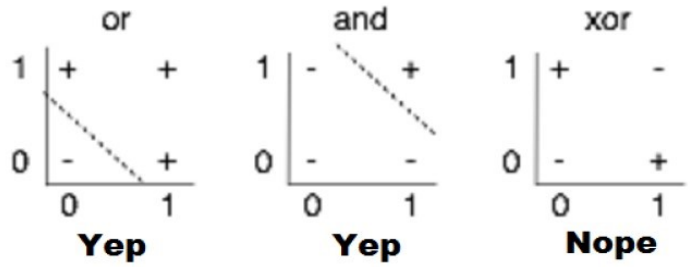


Figure 17: What can be expressed through a linear function

To overcome this issue, neural networks nowadays consist of not just one or several Perceptrons (i.e. neurons, nodes or units) with one input and one output each, but rather form a network of multiple layers (i.e. the neural network) of up to millions of neurons/units, that may also be placed in-between the input and output layers, as so-called hidden layers. These hidden layers allow a neural system to elaborate vastly more complex problems, by basically breaking up the very complex and possibly noisy data into several smaller features, allowing a more fine-grained approximation function and in fact the generation of XOR functions and others. Mathematically neural networks were therefore quickly coined as being "universal

approximators" (Hornik et al. 1989), meaning that with them one can theoretically express any (mapping) function in a multilayer configuration.

Figure 18 shows a simple feedforward multi-layer neural network with two hidden layers and one single output node in the output layer (depending on the model it can be more). We can see that with a network like this, the training method described above would no longer be possible, as the weights from the input-layer to the first hidden layer do not directly influence the result in the output layer. Conversely the first hidden layer is also not directly connected to the actual output node, meaning that the weights from hidden layer 1 to hidden layer 2 do not directly influence the value in the output layer.

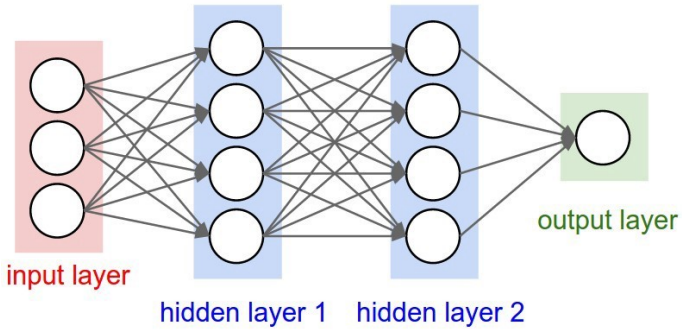


Figure 18: A simple feedforward multi-layer neural network [33]

5.1.2 Training of neural networks: Backpropagation

As we've found out above, once we create a network with layers in-between the input and output layer the training devised for a single node is no longer applicable, because, for example, hidden layer 1 has no direct access to the output layer. While this put research behind artificial neural networks into a somewhat extended hiatus, in 1986 Rumelhart, Hinton and Williams popularized a method to apply the error-rate across the whole network and as such update each weight in the network; this method is called **backpropagation** (Rumelhart et al. 1986). The realization was made, that with a non-linear but also more differentiable activation function in the neurons (so something closer to ADALINE than the Perceptron), it is not only possible to use the derivative to minimize the error, but it is also made possible to backpropagate[34] this derivative over all of the layers before it, essentially "splitting up the blame" for the error over each neuron. It is therefore possible to calculate in which way to adjust weights for each individual neuron. Further, to minimize the error typically the so-called **stochastic gradient descent** is used (Kurenkov 2015:27). What happens here, is that the network learns from its

[33] https://medium.com/@rajatgupta310198/getting-started-with-neural-network-for-regression-and-tensorflow-58ad3bd75223 (accessed May 29, 2019)
[34] Thanks to what is essentially the chain-rule of calculus: https://en.wikipedia.org/wiki/Chain_rule (accessed May 29, 2019)

own mistakes, by slightly adjusting the weights of each node until the closest approximation on the final output-layer is found. For a more picturesque explanation, I highly recommend Luis Serrano's unlisted YouTube video to linear regression and stochastic gradient descent, where he describes the process of minimizing the error by metaphorically comparing the process of the stochastic gradient descent to climbing down a rather steep "Mt. Errorest" (Serrano 2016)[35]. While this way of learning is the great strength of neural networks, it is also one of their weaknesses: Because neural networks can be several layers deep and we never actually see the output of the hidden layers (and even if we would, it would not be intelligible, as the network creates **vectors** as abstract representations of the data captured), it is essentially impossible to tell which weight is adjusted how and therefore manual tweaking of neural networks is more or less a matter of trial and error; **all we see is whether the produced output is correct or not**.

5.1.3 Vectors: How neural networks "think"

The adjustment of the weights within the network by minimizing the error in the output is still how neural networks work and learn today. Essentially, **supervised learning** is learning by **trial and error**. It is important to note at this point, that all operations within the network are strictly mathematical, more specifically calculus based. There are little to no statistical concepts applied, let alone linguistic rules. Neural networks operate on a list of numbers called **vectors,** that are created by the network itself to represent the features of the input in an abstract way. We have seen this in SMT, but SMT uses these vectorized inputs to generate statistical data out of, while NMT operates on the vectors from input to output. NMT may operate on vectorized single words, phrases or even individual symbols (for example, in the form of byte pair encoding). There are several ways to represent meaning-units, such as words, as a vector. One option would be the so called "**One-hot-encoding**", which essentially generates vectors where only one "1" in a certain position encodes the word ("one hot" means "one 1"). For example, it would be possible to represent the words "translator" and "interpreter" as the following vectors:

Translator: [1, 0, 0 … 0]
Interpreter: [0, 1, 0 … 0]

While the one-hot-encoding is fairly straight-forward, it does not capture any relations or similarities between the words. Nothing in the encoding would suggest that "translator" and "interpreter" are indeed both working with languages or in fact generally human beings. The

[35] Linear Regression Answer: https://www.youtube.com/watch?v=L5QBqYDNJn0 (Serrano 2016)

one-hot-encoding is therefore only used as a **starting point** for the vocabulary in NMT. In fact, before training an NMT generally a fixed vocabulary size must be defined for the network, as the total number of 0's and 1's in the vectors depends on the total number of words stored in the vocabulary. This means, that the bigger the vocabulary, the longer the training of the network will take.

However, with the one-hot-encoding, no real translation task could be solved. NMT therefore creates a **word embedding** from the vocabulary: During training, the network attributes a list of numbers (or a vector) to every word in the vocabulary to define its position in a theoretical "meaning space", however, to position the words in that space the list of numbers no longer only consists out of "0s" and "1s", but of decimals in-between. This way, shared features and similarities between words can be represented as well.[36] Since decimals can be used, the vector can also be a lot smaller than the actual number of words stored in the vocabulary. However, the longer the vector the more fine-grained the embedding can be. In practical use, vectors with a size of 256, 512, 1024 or similar may be chosen for the word embedding. "Interpreter" and "Translator" could, for example, be represented as follows:

Translator: [0.33, 0.44, ...]
Interpreter: [0.33, 0.45, ...]

Since vectors can be read as "spatial coordinates", you could say that the network generates a sort of meaning space or meaning cloud, in which related words are embedded closer together and words which represent other concepts are embedded farther apart from each other. Hence the term **word embedding**. Luckily, tools like *tensorboard*[37] exist, that enable a 3D visualization of said embeddings. While not terribly useful for debugging a neural network, it is fascinating to see where and how the machine places words within said word-embedding space.

Figure 19 shows us a three-dimensional representation of a word-embedding from the training presented in Chapter 7. Each dot represents a word (or "*token*").

[36] https://www.yamagata-europe.com/en-gb/blog/neural-machine-translation-what-s-under-the-hood-part-2 (accessed August 13, 2019)
[37] https://www.tensorflow.org/tensorboard/get_started (accessed March 03, 2020)

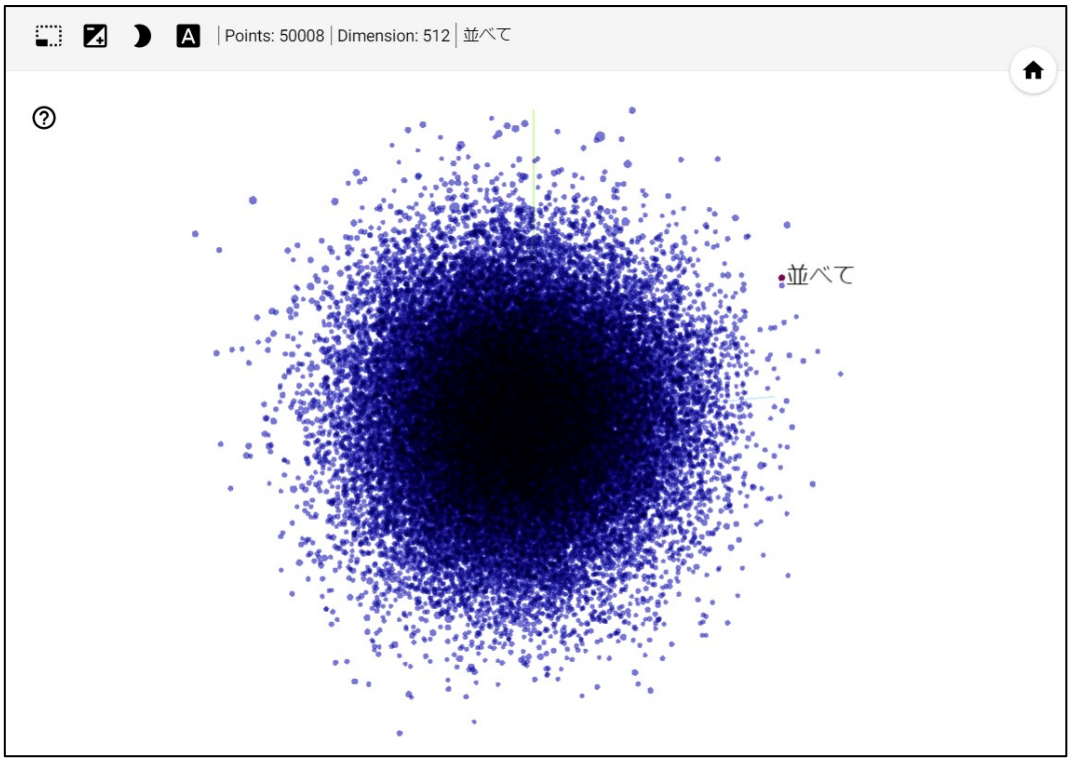


Figure 19: A three-dimensional representation of word-embeddings in tensorboard. Each dot represents a word

It is important to note here, that the machine which processes the text through the network, does NOT understand what it is reading. In fact, it "merely" analyzes the words by looking at their context in the training corpus and finds above mentioned patterns. This approach is based on the theories of **distributional semantics**[38] and while similar to the statistical approach, it has the advantage that the extracted features are more fine-grained than mere statistical occurrence data. However, for it to work properly, an even bigger amount of parallel text data is required than was the case with SMT.

The great thing about operating on vectors, is that they can, by definition, be combined to other vectors according to vector algebra. This would enable neural networks to generate a vector that essentially represents the whole sentence instead of each individual word or phrase. However, all the networks we have seen up until now were simple feedforward neural networks, which would not allow an easy connection between the individual outputs. In the next section we will therefore look at the different types of neural networks and see which of those network architectures work best for translation tasks.

5.2 Types and variants of neural networks

With the explanations above, it should be clear that artificial neural networks operate strictly on numbers, i.e. vectors, on the basis of several, relatively simple mathematical operations, like

[38] (see Boleda 2020 for a good overview of research in this area)

matrix multiplications. These are linked together to map one input to a certain form of output. They learn to output the desired values by "automatically" adjusting the parameters (weights) of the network in such a way, that the desired result is found or at least as close as possible (**backpropagation of error**).

While all neural networks learn by these means in one way or the other, there are several variants of networks for several specific tasks. What we saw up until now, was mostly feedforward neural networks. A feedforward neural network computes a function *f* on a fixed size input *x* such that $f(x) \approx y$ for training pairs (x, y) (McGonagle et al. 2020). This means, that all connections are going from the input towards the output layer. It can therefore only present an output for each individual input and not for all the inputs taken together. Such a system would not be of big use for translation tasks. For this reason, other networks were designed, that allow different mappings, as seen in Figure 20.

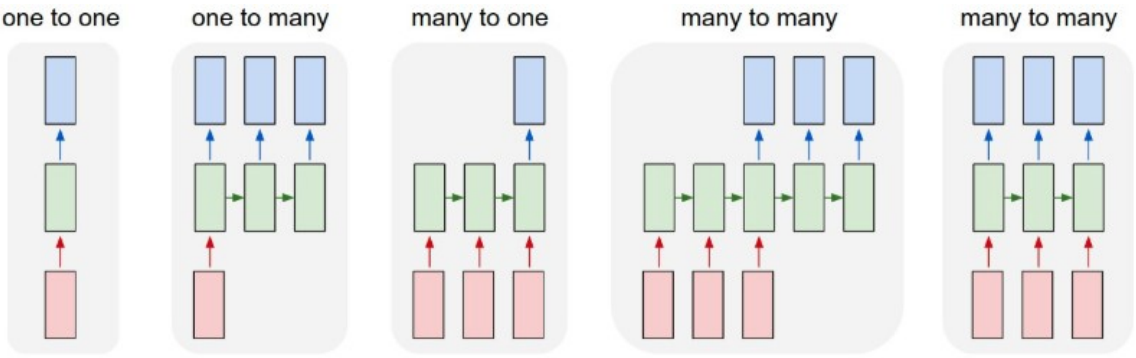


Figure 20: Some of the mapping possibilities with modern artificial neural networks[39]

The 'one to one' and 'one to many' mapping is possible with feedforward networks. This is ideal for image classification, as you might want to recognize a certain object (the output) within a frame (the input). Say for example, if you provide the network with an image of a cat, you would like to receive the text output 'cat'; likewise, you could train a feedforward network to output 'a white cat' for a picture that shows a white cat through the one-to-many mapping. One might imagine that this could work in a way similar to phrase translation tables in SMT. While this is true and feedforward neural networks were in fact used for creating phrase-tables (Cho, van Merrienboer, Gulcehre, et al. 2014), this would not fully use the potential of this new technique. In fact, it is the 'many to many' mapping enabled by the so-called recurrent neural networks (RNNs), that is a natural fit for sentence translations, as it allows the mapping of a whole sequence (of words) to another sequence (of words). That said, words do not need to be the smallest signifier for neural networks. The networks can operate on inputs that are on a sign

[39] https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912 (accessed on November 15, 2019)

level (i.e. letters or characters, like the Japanese *Kanji*) or could in fact be trained on phrases (Huang et al. 2017) or even full sentences as input. Either way, neural machine translation is a very active research domain at the time of writing this thesis and it would be impossible to cover all the different approaches that were developed for improving the translation performance.

In order to get a general understanding of NMT, we may look at the following four major architectures/models and elaborate in respect to their workings for translation:

1) The MLP (Multi-Layer Perceptron) or simple feedforward network
2) The CNN (Convolutional Neural Network) feedforward network
3) The RNN (Recurrent Neural Network) recurrent network
4) The Transformer

5.2.1 Multi-Layer Perceptron (MLP)

The term **Multi-Layer Perceptron** or **MLP** is used ambiguously, sometimes loosely to refer to any feedforward neural network, sometimes strictly to refer to networks composed of multiple layers of Perceptrons (with threshold activation). In this thesis, it is used to refer to multi-layer feed forward networks in general. As stated earlier, this means that the neural network's flow is strictly into one direction and therefore each layer is only fed with the output from the layer before it. However, unlike the original Perceptron, the activation function does generally not have a threshold activation that changes the output of each layer to a 0 or 1 in order to allow for backpropagation. This means the model works very well for classification or labeling predictions on, for example, tabular datasets, but not too well for translation tasks. The reason for this is that feedforward networks are based on several major assumptions that do not work well for language translation or language processing in general. One of the major issues of this type of neural network is that the size of the input layer is fixed to the length of the input sequence and thus the output length also must be fixed.

The other major issue here is the idea of independence - that different training examples (like the single words that make up a sentence) are independent of each other. So even if you were to process word for word and somehow find an equivalent solution every time, the context and meaning that develops over time in a sentence would be lost. Simple feedforward networks therefore inherently disregard two main aspects of translation: short and long temporal dependencies within a text and the fact that input and output length is not guaranteed to be the same (Agrawal & Sharma 2017:65). Additionally, even if you were to overcome these intrinsic issues by, for example, taking sentences as the base unit for meaning representations, this would

render MLPs unviable, as the amount of data for training would need to be unrealistically large and the network itself would have enormous memory (RAM) and processing demands because of the very large vectors required for capturing all that information.

5.2.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks or **CNNs** are also feed forward networks, however, they work by making the first and some subsequent hidden layers of the network **convolutional**. This means that each hidden layer only looks at a small subset of the input data and finds certain features within. These extracted features are then passed on to another set of hidden convolutional layers, which then can work on much leaner and cleaner data than the "noisy" input data, again finding certain features within. These high-order features can then finally be used to classify the data by the last two layers (one hidden layer and the output layer) as was happening before with regular feed forward neural networks. This makes the networks much more memory efficient and feasible for big amounts of data, especially because parallelization is possible within layers. For that reason, it works especially well on high-dimensional data like image data. Yet it was traditionally not used for language processing or translation, because CNNs do not maintain an internal state other than the network's own parameters. This means, that whenever a single sample (for example, a word from a sentence) is fed into a CNN, the network's internal state, or the activation of the hidden units, is computed from scratch and is not influenced by the state computed from the previous sample, just like with any other feedforward network (Agrawal & Sharma 2017:66). We are therefore still computing a function f on a fixed size input x such that $f(x) \approx y$ for training pairs (x, y). The network basically still has no 'memory' as to what it has just read and can therefore not account for short and long temporal dependencies, if a sentence were to be processed sequentially (i.e. word for word). To capture context the network would need to look at the whole sentence at once and therefore run into the same issues as a regular MLP network, like memory constraints but more importantly general data sparsity in the training corpus.

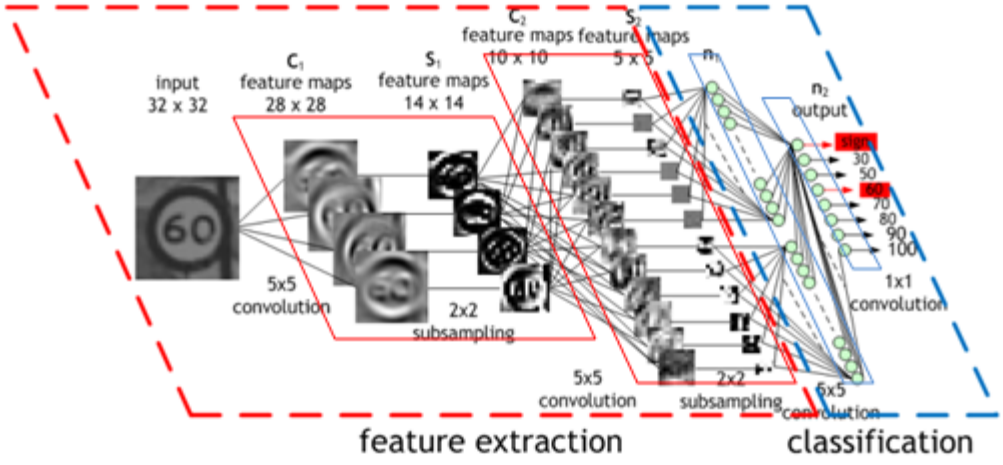


Figure 21: Visualization of a CNN[40]

As a side note, however, researches have been successful in using exclusively convolutional networks for neural machine translation (Gehring et al. 2017). Instead of producing an encoding of the whole source sentence by ingesting the embeddings of source words one by one, Gehring et al. devised an encoder that produces representations of each word by taking into account a few words (for example 2) to the left and to the right of it (similar to n-gram language models as seen in phrase-based SMT) (Forcada 2017:299-300; Gehring et al. 2017). This enables the CNN to capture context in a vectorized representation. Gehring et al. observe better BLEU scores (see Section 7.5.2.1) in English-French, English-German and English-Romanian tasks than with competing networks based on the recurrent neural network architecture that is actually capable of looking at the whole sentence and keep a sort of "context-memory" as shown in Section 5.2.3. Most importantly, however, Gehring et al argue that convolutional networks offer a higher level of parallelism than the more conventionally used recurrent neural networks because individual words don't have to be processed in sequence. This enables shorter training times on modern GPUs, which are very fast in parallelized workloads. More about their effort can be found on their *Facebook Engineering* publication online.[41]

5.2.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks or **RNNs** are an approach to solve the memory issue mentioned above by simply looping the output of the network back into the network. This enables RNNs to 'remember' data from the input over time, all the while handling inputs of 'any' length. One

[40] https://sites.google.com/site/5kk73gpu2013/assignment/cnn (accessed on December 03, 2019)
[41] https://engineering.fb.com/ml-applications/a-novel-approach-to-neural-machine-translation/ (accessed on November 18, 2019)

of the main reasons that RNNs are more exciting than their feedforward counterparts is that they can operate over a sequence of vectors: Sequences in the input, the output, or in the most general case (translation) in both. Unlike feedforward networks, recurrent neural networks learn sequential data by computing g on variable length input $X_t = \{X_1, \dots, X_t\}$ such that $g(X_t) \approx y_t$ for the training pairs (X_n, Y_n) for all $1 \leq t \leq n$.[42] Essentially, this enables RNNs to add the variable of "time" into the mix, by sequentially going through the individual inputs of a sequence in timesteps *t*, while carrying over information from each individual input/timestep. Each word in a sentence can therefore be learned in context of everything that came before it. In Figure 22 we see a representation of an unrolled RNN, i.e. where we see each timestep as an individual network. Essentially, we are looking at the RNN as several feed-forward networks, where for each new input from the sequence (X_t) a copy of the current network is generated. This new network is then fed the output of the network before it (called hidden-state h_{t-1}, as it is not a visible output) and the new input from the sequence (X_t). The exciting thing about this is that in RNNs each output of each individual timestep is combined into a new vector and finally output as one combined final vector. That vector should contain all of the information before it and therefore also encode what the sentence represents over time. In other words, this final hidden-state vector may be thought of representing the "meaning"[43] each individual word represents in the specific order and form in which they are set in that sentence. The resulting vector was therefore called the **thought vector** or **meaning vector**.

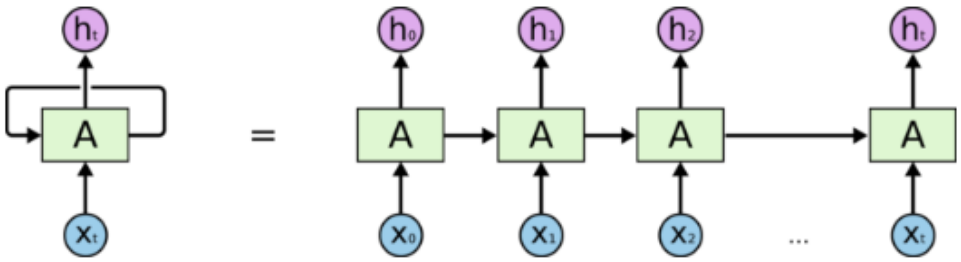


Figure 22: An unrolled recurrent neural network[44]

In theory, this way classic RNNs can keep track of arbitrarily long-term dependencies over variable length sequences, making them ideal for translation tasks. However, in practice memory still posed a problem with classic RNNs: Because each individual timestep of an RNN

[42] *Feedforward Neural Networks.* https://brilliant.org/wiki/feedforward-neural-networks/ (accessed on November 15, 2019)
[43] "meaning" has to be understood in a very abstract way here; the network basically captures very intricate patterns of languages by analyzing the vast corpus of sentences that it is fed.
[44] https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912 (accessed on December 08, 2019)

adds information to the next through multiplication, training through backpropagation[45] of the error leads to “vanishing” gradients (they become smaller and smaller, approaching 0) or "exploding" gradients (they increase in size on every iteration towards infinity). This makes training with the conventional methods very difficult if not impossible. In practical terms a classic RNN network would therefore learn well only on smaller sequences, but long-term dependencies and indeed longer sequences would never be learned properly as the weights of the network would just not adjust well through backpropagation in longer sequences.

Hochreiter (1991) and Yoshua Bengio et al. (1994) analyzed the problem in detail and proposed several approaches to try and mitigate the problem. It was not until 1997 when Hochreiter and Schmidhuber came up with what might be considered the solution to the problem: the so-called long short-term memory (LSTM) RNN model (Hochreiter & Schmidhuber 1997). Any real application of RNN in machine translation is based on LSTMs or variants thereof. In addition to the original authors, a lot of people contributed to modern LSTMs. In the next section we will take a short look at this variant of RNNs.

5.2.3.1 The LSTM RNN (Long Short-Term Memory)

Just as the name states, the LSTM RNN model (commonly just referred to as LSTM) enables the network to maintain a short-term memory of the context of each individual word over a longer sequence of words, especially over long distances within the sequence. It was specifically developed to mitigate the issue of exploding and vanishing gradients that may be encountered when training classic RNNs on longer sequences.

LSTMs achieve this by keeping a separate flow of information outside the normal flow of the recurrent network in a so-called **gated cell**. This cell allows for information to be stored, written, or read and this information can then be carried over from one timestep to the next through the so-called **cell state**. Figure 23 shows a standard RNN, whereas in Figure 24 we can see the somewhat more complex LSTM. The gated cell of the LSTM is represented by the four gates: three sigmoid gates (σ), a forget gate, an input gate and an output gate, and one additional output gate at the top horizontal line, which is the cell state (c_t). The cell state may also be called a **context vector** as it transports the context of all words over a long distance.

[45] For RNN an extension called Backpropagation Through Time (BPTT) is used.

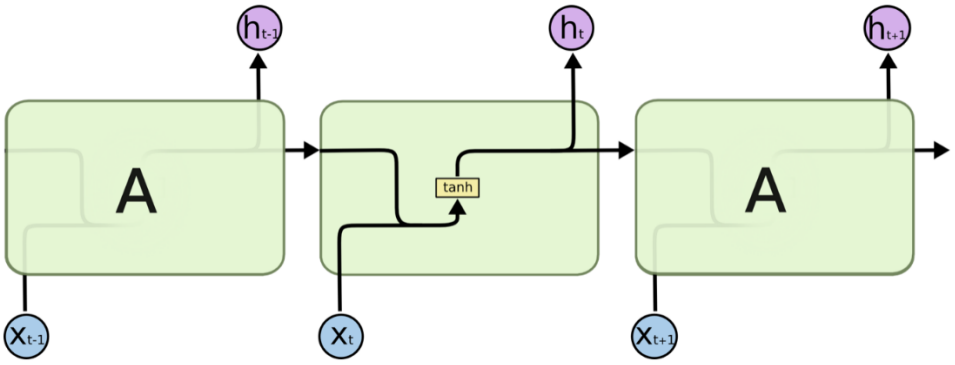


Figure 23: A recurrent unit in a standard RNN unit[46]

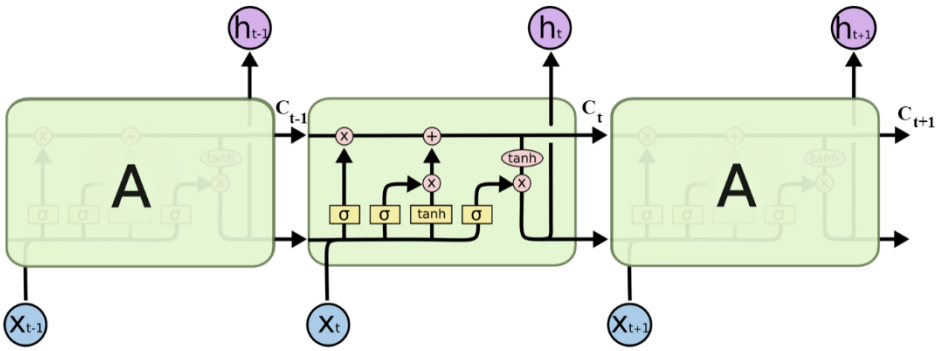


Figure 24: A recurrent unit in LSTMs[47]

The gates allow the cell to make decisions about what to store, and when to allow reads, writes or deletions. This information is then passed on to the next timestep of the network as the cell state. This allows for information to pass through the network relatively unchanged, meaning that the error can also remain more constant throughout the whole flow of the neural network.

Just like regular neural network nodes pass on certain information, the gates block or pass on information to the cell state, which is filtered through the gates' own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent network's learning process. Basically, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, **backpropagating the error**, and adjusting weights via gradient descent (see Nicholson 2019).

Essentially, the network has more parameters (the additional weights in the LSTM) and data (the cell-state/context vector) to work with. Because the flow of information in the cell-

[46] https://miro.medium.com/max/1606/1*wXEZTk3g_UiOgL6VutuBGA.png (accessed on December 03, 2019)
[47] https://miro.medium.com/max/1676/1*WOGNu3QcmDipMVPF2yA9wA.png (accessed on December 03, 2019, edited to reflect c_t)

state is controlled through the gates, vanishing or exploding gradients are much less likely. The network therefore manages to keep a much longer memory of words that are farther apart.

However, since more data and more mathematical operations must be processed, LSTMs tend to be quite a bit slower both in training, as well as in the translation process itself. Additionally, since each new timestep is dependent on the output of the prior timestep, parallelization, which is the most notable reason for the increase in computing power of the last decade, is not easily leveraged for improving performance. For that reason, more simplified **gated recurrent units (GRUs)** are often used in the real world and research (see Bahdanau et al. 2014:12). These only have two gates and one final output gate, which takes away some of the more granular control the network can exert over what data is transferred to the next timestep of the network but reduces processing load and therefore increases speed.

While LSTMs highly improved the capacity of RNNs to work over longer sequences, they still have issues besides performance. For one, the network still tends to forget words or parts of the context of words that are far apart, since even though the context vector is updated in a more controlled way than the meaning vector in regular RNNs, it is still updated as a single vector. Therefore, the probability of finding the context of a word that is far away from the word currently being processed decreases exponentially with the distance between the two.[48]

What we saw until now, was how the networks learn to represent meaning as vectors, but we haven't quite covered how these vectors are finally translated into the TL. In the next sub-section, we will look a bit more closely at how the majority of these translation models do the actual translation under the hood.

[48] https://towardsdatascience.com/transformers-141e32e69591 (accessed on December 10, 2019)

5.2.3.2 Encoder-Decoder modeling

Like with SMT, in NMT we are, in most cases, talking about encoding and decoding. Encoding is what we saw up until now: The network recognizes patterns in the training data and encodes these as vectors that should reflect the meaning of the sentence. The task of decoding, then, is to transform these meaning vectors into sentences in the TL.

The decoder in most NMT systems is built and trained in such a way that it resembles a text completion device (like the word prediction feature of some smartphone keyboards), which is informed by the meaning vector computed by the encoder. The decoder therefore provides, at each position of the target sentence being built, and for every possible word in the target vocabulary, the likelihood that the word is a continuation of what has already been produced (Forcada 2017:296).

This Encoder-Decoder model for NMT was concurrently introduced by Sutskever et al. and Cho et al. in 2014 (Cho et al. 2014; Sutskever et al. 2014). These models may be seen as the break-through for NMT as they first managed to outperform statistical machine translation models on large translation tasks (Sutskever et al. 2014).

Sutskever's RNN NMT architecture consists of two LSTM models: An **encoder model** and a **decoder model**. In the encoder model an input sequence is read in its entirety (up to the end-of-sequence token (<EOS>) and encoded to a fixed-length internal representation: the meaning vector. The decoder part of the network then uses this internal representation to output a variable-length sequence of words in the TL until the end-of-sequence token is reached. Essentially, the encoder part is trained to create fixed-length vector representations from the SL training data, whereas the decoder part is an RNN language model conditioned on the vector representations of the encoder model and the word embeddings created from the training data. The two models are therefore always trained in tandem.

A trained network can then be fed with new input data, which it would encode into fixed-length vectors (the meaning vector and the context vector in LSTMs). The encoder model stops the encoding once it reaches a pre-determined "end-of-sequence" token (<EOS>). From there, the vectorized sentence (meaning vector) and the context vector are input into the decoder model of the network: the meaning vector and the <EOS> token act as the input to the network and, by using the probability patterns that were learned during training, the decoder model maps the vector back into a sequence of words of another language. Since this whole process remains all within a single RNN, even the final timestep can theoretically still use information from the first step in the whole network. This is represented in Figure 25, where an SL sentence "ABC" is translated to the TL sentence "WXYZ". We can see that the initial data essentially flows

through the whole neural network and may therefore still be updated in the decoder part of the model.

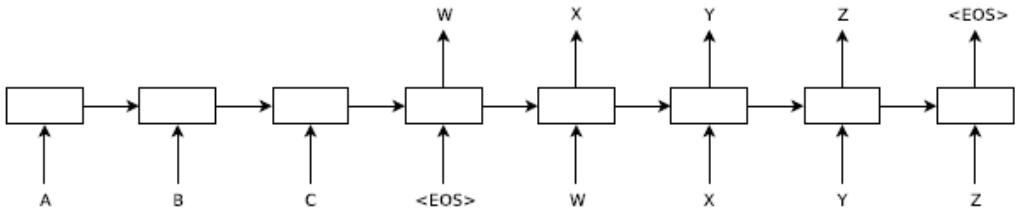


*Figure 25: Sutskever's Encoder-Decoder workflow (*Sutskever et al. 2014*:2)*

As noted in sub-section 5.2.3.1, the issue here is that the probability of keeping the context of a word that is far away from the word currently being processed decreases exponentially with the distance between them. So, for longer input sentences or even if the output sentence length would increase, the likelihood of the network still referring to and considering the right context for an earlier word in the timeline diminishes significantly.

To combat this issue Sutskever et al. realized that inverting the input sentence's word-order would bring words in similar language-pairs[49] closer together. In other words, when translating a sentence *a,b,c* to w*,x,y* the network would learn to translate *c,b,a* to *w,x,y* instead. This way, they found, the performance of the network greatly increased as *a* is in close proximity to *w, b* is still fairly close to x, and so on (Sutskever et al. 2014:3).

However, it stands to reason that such a solution only works well for related language-pairs and would likely present issues with languages that have a vastly different word order. Several empirical studies found that the fixed-length meaning vector acts as a bottleneck for the Encoder-Decoder approach (Pouget-Abadie et al. 2014; Cho, van Merrienboer, Bahdanau, et al. 2014). Because of this a much more robust solution was proposed to solve the memory issue LSTMs were still facing: the attention mechanism.

[49] In their case: English-French

5.2.3.3 Attention mechanism

The attention mechanism was first proposed by Bahdanau et al. 2014 and Luong et al. 2015. The technique described in these papers essentially allows the **decoder model** to focus on relevant parts of the input sequence as needed. This may be compared to how translators focus their attention on the ST words and context that they are trying to translate at the moment.

To achieve this the **encoder** does not pass the fully assembled meaning vector (i.e. the last hidden state) to the **decoder**, but instead passes **all the hidden states** (e.g. h_1, h_2, h_3) along to the decoder, so that a more controlled **context vector** may be computed by the decoder itself. Additionally, a feedforward neural network **alignment model**[50] is jointly trained with the network and added to the RNN decoder for deciding on what to pay attention to. Essentially, this allows the decoder to search through the ST and focus on certain parts of it while decoding.

Once the decoder is fed with the <EOS> token, it is reinitialized with a new hidden state (h_{init}) and based on that calculates an output and new hidden state of the network (e.g. h_4). The output is discarded, as the decoder uses the hidden states from the encoder for the **attention steps** to create a new **context vector**:

1) First the decoder looks at the whole set of hidden states it received from the encoder. Each of these hidden states is naturally most associated with the input that generated it.
2) The hidden states are then scored according to the alignment learned in training by the alignment model (i.e. how well the inputs around a certain ST position fit the TT position).
3) The score is then softmaxed and multiplied, which amplifies the hidden states with high scores, while the hidden states with low scores are drowned.
4) Finally, these scores are summed together to create the **context vector** for that specific timestep.

The context vector C_4 is then concatenated with the hidden state h_4 of the decoder and passed through another jointly trained feedforward network that finally outputs the TT word (see also Figure 26).

[50] Note that unlike in SMT, the alignment is not considered to be a latent variable. Instead, the alignment model directly computes a soft alignment between ST and TT words which also allows the gradient of the cost function to be backpropagated. This means that it is not a separately trained model, but part of the network and therefore trained towards a better log-probability of producing correct translations (Bahdanau et al. 2014).

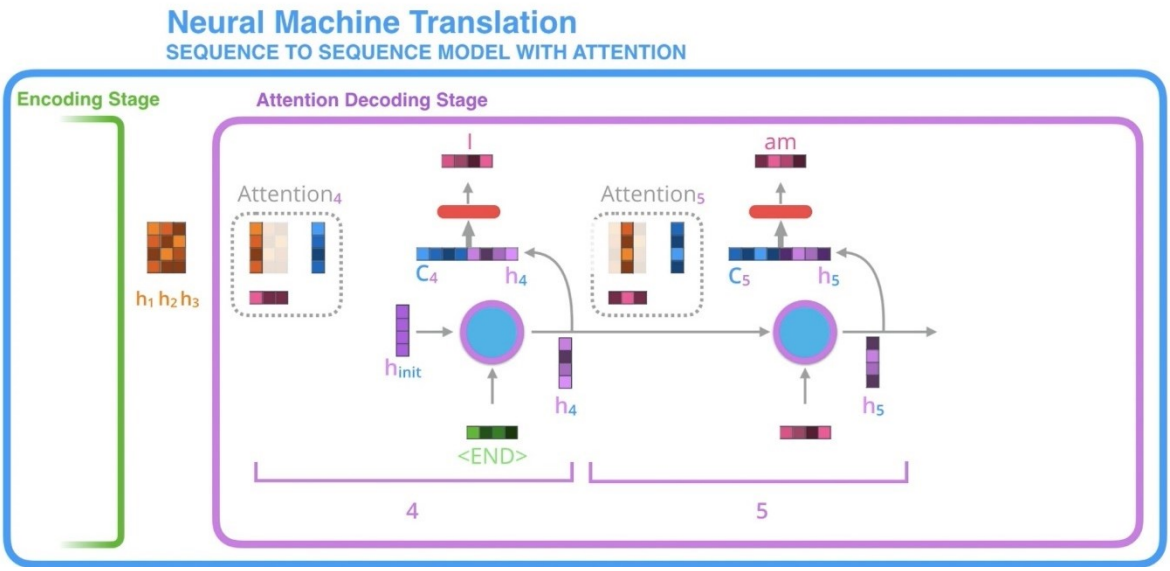


*Figure 26: RNN with attention translating DE-EN "Ich bin Student" to "I am a student".(*Alammar 2018b*)*

The big advantage of the attention mechanism is that the encoder is relieved from having to create one monolithic vector that contains all the information within a sentence. All the hidden states are passed on to the decoder and the attention mechanism allows it to pick out the parts that seem most relevant according to the alignment model. Bahdanau further improved upon this idea by deploying a Bidirectional RNN (BiRNN) for the Encoder that would read the sentence from left-to-right and right-to-left in order to store both the context before and after a certain timestep within the hidden state[51] (Bahdanau et al. 2014:3).

For a beautifully animated explanation of the attention mechanism, I highly recommend Jay Alammar's blog post regarding the technique[52] as the animated visualizations help a lot with understanding.

[51] Bahdanau calls the hidden states *annotations*

[52] https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/ (accessed on January 04, 2020)

5.2.4 The Transformer model

Attention proved to be a very robust solution for maintaining context over long distances and drastically improving the quality of machine translation through RNNs. However, training the networks and running the translation through all of these steps is computationally quite expensive and, because of the sequential nature of RNNs, parallelization cannot easily be employed to improve performance. Therefore, there is a practical limit to network complexity and scalability as the individual steps are reliant on the result of the calculations before them.

In 2017, Vaswani et al. seek to solve the performance issue with their seminal work titled *Attention is all you need* (Vaswani et al. 2017). In their paper they introduce what is now essentially the state-of-the-art architecture for NLP and machine translation: The Transformer model.

In essence, they suggest an Encoder-Decoder network architecture that does away with both convolutional and recurrent networks and is instead based solely on the idea of attention. In order to enable parallelized encoding and decoding, the input sequence is no longer provided sequentially to the network, but instead the inputs are fed into the network all at once as a list of vectors. These vectors are created by the learned word embeddings during training. The paper suggests creating vectors that have a size of 512 positions for their "Transformer Base" model, but also explores using larger embeddings in their "Transformer Big" model, which results in a noticeable improvement in model accuracy at the expense of higher memory and computational requirements (Vaswani et al. 2017:9).

In order to provide enough weights for handling all that input data, the paper suggests the use of six equal encoders and decoders stacked upon each other; however, while they are equal, they do not share weights with each other. Six is also an arbitrary number, in fact it is possible to use more, but also less stacked encoders and decoders. Each encoder is made up of two sub-layers: a so-called **multi-headed self-attention layer** and a **feedforward neural network layer**. Self-attention is a new concept, that allows the encoder to look at the different words within the sequence. It can be seen as a replacement to the hidden-states we had in the RNN Encoder-Decoder architecture.

This is a needed addition as in the Transformer model each word or in fact vector in each position of the list of vectors flows through its own path in the encoder. In other words, each vector in the list is passed through the same feedforward neural network, but individually. The output of the feedforward network layer is therefore not dependent on the words that came before or after, so they can be computed in parallel. Additionally, thanks to the context encoding of the self-attention layer, the feedforward network operates on vectors which contain

information about other relevant words in the sequence. Figure 27 shows the flow of a simple example sentence “Thinking Machines”, where *x1* and *x2* are the vectors resulting from the word embedding algorithm, *z1* and *z2* are the self-attention annotated vectors and *r1* and *r2* are the output of the first encoder after the feedforward neural network pass. These outputs are then used by the second encoder to process as inputs.

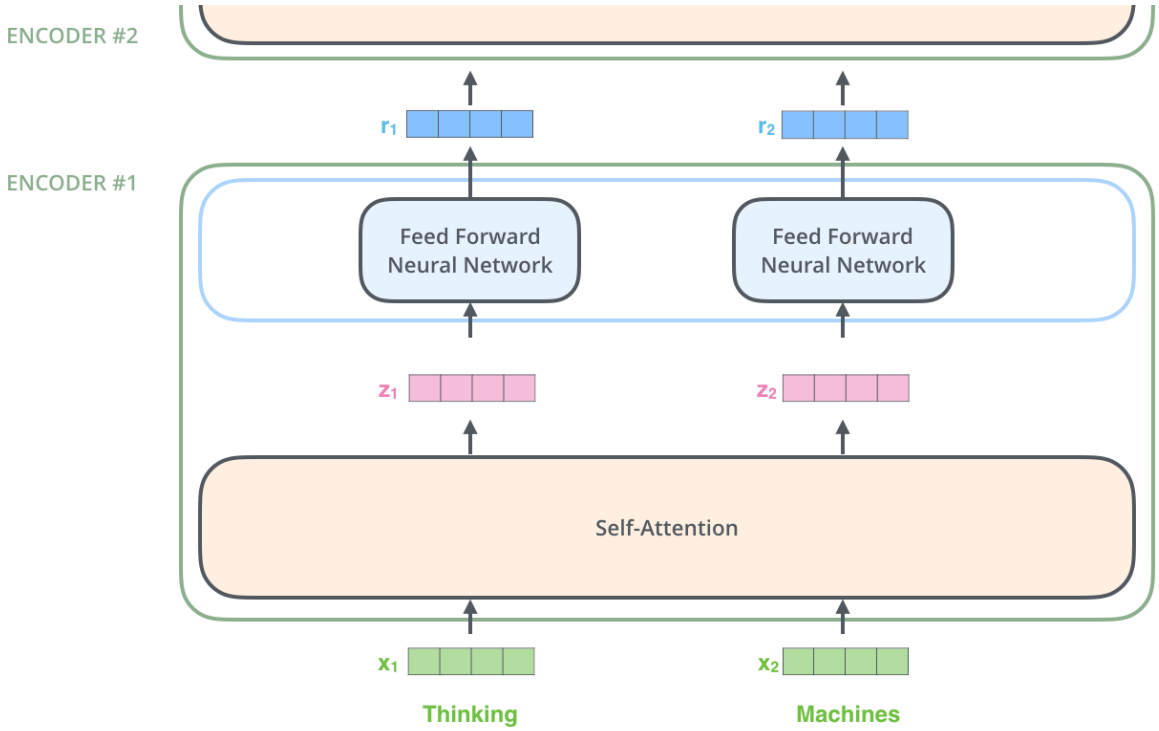


*Figure 27: The vectors in the list flow through their own path in the Encoder and provide individual outputs each; the connection between the different parts of the sequence is made in the self-attention layer. (*Alammar 2018a*)*

For the most part, the decoder is made up in the same way, but between the self-attention layer and the feedforward layer there is an additional attention layer (called Encoder-Decoder attention), which is fed with the output from the encoder. This layer helps the decoder to focus on relevant parts of the input sentence and predict the most likely output (similar to the original attention mechanism described in Sect. 5.2.3.3).

Figure 28 shows the Transformer model’s architecture from input to output. The encoder and decoder part are represented as single units but can be considered as stacks *N*x, where *N* stands for the number of units used.

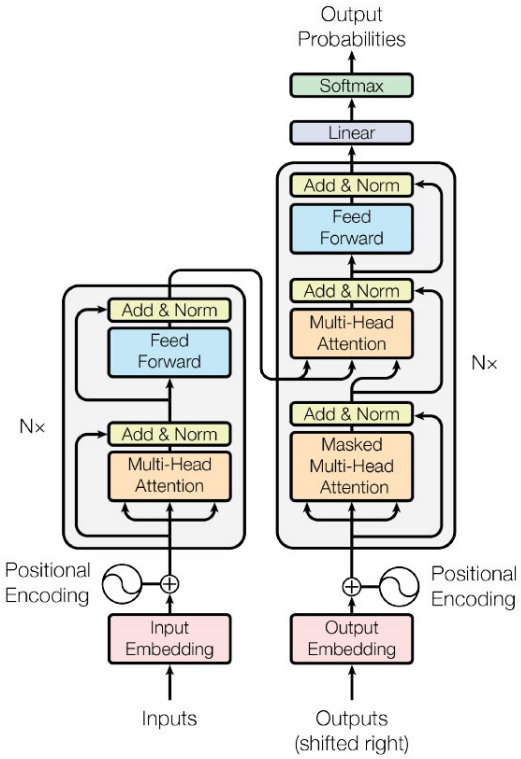


*Figure 28: The Transformer architecture as suggested by Vaswani et al. (*Vaswani et al. 2017*:3)*

Notice how the illustration adds a **positional encoding** node between the embedding layer and encoder/decoder stacks. Since the list of words is processed individually and all at the same time, the time information we had gained through RNNs' sequential processing is lost. Therefore, an additional positional encoding is required and added to the word embedding representations, so that the encoder and decoder can operate on positional information as well.

In summary, the Transformer model eschews recurrent and convolutional networks in favor of simpler and more parallelizable feedforward networks, by "annotating" the input vectors with context information through the attention mechanisms and the positional encoding. This allows the model to be trained much faster than regular RNN Encoder-Decoder models, while also providing a more robust context representation than the attempted CNN models (see Section 5.2.2).

At the time of writing this paper, most NMT systems, and, in the broader scope, NLP systems based on neural networks, operate through the Transformer architecture.

Note that this subsection provided a highly simplified explanation of the Transformer architecture, as describing it in detail would be beyond the scope of this thesis. Vaswani et al.'s paper provides an exhaustive but very mathematical explanation. For those interested in the

deeper workings of the self-attention mechanisms and the Transformer in general, I would again recommend reading Jay Alammar's excellent blog posts about the Transformer architecture. [53]

5.3 Summary

This chapter offered a deep look into the workings of neural networks and presented the different neural network architectures developed with a main focus on machine translation. We have learned, that neural networks operate on **input embeddings** (often on a word-level), essentially vectorized representations of the elements that make up a sentence, i.e. the input. This reliance on abstract distributed representations forms the pillar of the strength of neural networks: to mathematically and automatically **find patterns within data** and learn **mapping probabilities** that are much more sophisticated than mere statistical occurrence data of individual words or phrases.

We have learned that through **LSTM RNNs** neural networks managed to incorporate both the temporal aspect inherent to language (when seen as a sequence over time) as well as providing a means for the network to **learn long distance relationships** between the individual constituents of a sentence. We now know, that NMT mostly relies on **Encoder-Decoder models**, that act similarly to text completion devices: An encoder generates a vectorized representation of a sentence based on distributional semantics and the decoder predicts the most likely output based on that representation.

Finally, we saw how networks learned to "**pay attention**" to certain parts of the input by using soft-alignment models and the highly annotated data from **BiRNNs**, which relieved the networks from the constraint of having to represent the whole meaning of a sentence within a single vector, the "**meaning vector**".

This culminated in the development of the **Transformer model**, which relies on the aforementioned attention-mechanism to annotate each vector representation with what the mechanism regards to be relevant for that specific embedding. The main advantage of the Transformer is highly improved efficiency thanks to now parallelized computing and a high degree of contextual representation in the meaning encodings of the network.

We must however also remind ourselves that all of these processes for now work **solely on a sentence-level**. This means, that context beyond the sentence-level is generally NOT considered by neural networks at this point. Additionally, since the calculations performed by the neural networks are guided by the weights found through **backpropagation of error** along several layers of neural networks (i.e. the deep learning moniker), the actual "thought process"

[53] https://jalammar.github.io/illustrated-transformer/ (accessed January 14, 2020)

or the parameters of the neural networks (i.e. the single weights within the deep neural networks) remain mostly a black box for researchers. This means, that while the training examples may be solved correctly and many real examples may be as well, it is often very hard to pinpoint the actual source of issues an NMT system might run into with certain translations. The solution is generally tweaking of hyperparameters[54] (i.e. length of the embedding vectors, learning rate of the network, search algorithms [greedy, or beam search size], etc.), different input handling (i.e. word-embedding, symbol-embedding, phrase-embedding, etc.) or pre- and post-processing of data.

In Chapter 6 I want to look at patent translation as a specific use case for NMT. Finally, in Chapter 7, I will follow this up with a practical experiment by creating two differently tweaked translation models using one of the several available open source NMT frameworks and compare their performance. The idea behind this, is to provide a sort of tutorial for other translators looking at getting a deeper understanding of NMT and to practically examine whether a translator like myself could actively contribute to NMT research.

[54] Hyperparameters can be set by humans before training; whereas the parameters, i.e. the weights, are automatically computed by the network through training.

6 NMT in patent translation

One sector where NMT has found resounding success is the domain of patent translations. Major patent offices, like the JPO (Japan Patent Office), the EPO (European Patent Office) and the WIPO (World Intellectual Property Organization) are relying on NMT systems to provide quick translations of patent claims and descriptions to their customers. The quality of these translations is often astoundingly good, but at times fails spectacularly. In the following section, I will look at some of the major patent translation systems and make some educated guesses as to why quality issues remain, while also pointing out why translation works so well in certain other cases.

6.1 A look at patent machine translation

Patent language is highly standardized. There are certain expressions and syntactic structures that will be found over and over in different patents. And while there are differences depending on the formalities established by patent offices around the world, they are all generally structured as follows:

- **Title page:** Contains most of the bibliographical data about the patent. This includes patent number, dates of application and approval, classification (domain) and the inventors or applicants. May also contain a short abstract.
- **Claims:** The central part of patents, that is essential for getting a patent approved and therefore the most translated part of patents. These claims need to express what is new about the invention or what makes the invention "unique". The novel part of the invention is often expressed after the short sentence "…characterized in that…". Each claim can only be one sentence long (but there can be more than one claim per invention).
- **Detailed description of invention:** Further elaborates the specifics of the invention in slightly more "human" language. The description is no longer limited to single sentences, as was the case with claims, and it can be further divided into the following sections:
    - **Technical field of the invention:** This section describes the technical field to which the invention pertains, generally using the paragraph: "The present invention relates to a semiconductor manufacturing device, and more specifically relates to …".
    - **Prior Art:** This section describes patents and inventions that have been previously made public (i.e. prior art) and generally also describes the problems with this prior art.

    - **Problem to be solved by the invention:** In essence, this section elaborates further on the problem(s) already described in "Prior Art".
    - **Means for solving the problem:** In this section, the central concepts of the invention are described, and often the language of claims is repeated. However, since the claims are recited in rather abstract expressions this may be accompanied by an easier to read explanation.
- **Preferred Embodiment:** Concretely describes an invention by providing example realizations of the invention. For example, when describing a new type of neural machine translation network, the patent would describe exactly how many layers to use in the network, what hyperparameters to use, etc.
- **Drawing Sheets:** Illustrations to visualize the invention and complement the textual description.

Japanese patents have essentially the same structure, although obviously different on a surface level (i.e. the language is different):

- Claims → 請求の範囲 (*seikyū no han'i;* literally: scope of claims)
- Detailed description of Invention → 発明の詳細な説明 (*hatsumei no shōsai na setsumei*; quite literal*)*
- Technical field of the invention → 発明の属する技術分野 (*hatsumei no zoku suru gijitsubunya;* quite literal)
- Prior Art → 従来技術 (*jūraigijitsu;* quite literal)
- Problem to be solved by the invention → 発明が解決しようとする課題 (*hatsumei ga kaiketsu shiyō to suru kadai*; quite literal)
- Means for solving the problem → 課題を解決するための手段 (*kadai wo kaiketsu suru tame no shudan;* quite literal)
- Preferred Embodiment → 発明の実施の形態 (*hatsumei no jisshi no keinō*; literally: form of embodiment of the invention)

Of course, what is shown above can only be regarded as the most common formulations (Okuda 2015). As mentioned, there are slight variations in the exact wording of these titles, but they always denote the same sections/chapters with regards to content.

As we have seen so far, corpus-based machine translation, like SMT or NMT, works by recognizing patterns in parallel text data. Therefore, since essentially all patents use very uniform language (especially on a semantic level) and a very similar structure, the content is easy to line up and pattern recognition should work exceptionally well. Until recently, however, SMT would have a lot of issues coping with the very long sentences presented by patents. The

marked improvement with the switch to NMT is the ability to "capture the whole structure of the sentence", as Ian Wetherbee from Google Patents[55] puts it. This should help tremendously when translating long claims, which must be formulated in a single sentence. Another big advantage is that with patents we have large collections of parallel texts available. Patents are generally written in the inventor's native tongue (say Japanese) and then translated to the accepted languages of the foreign patent offices. The patent offices can therefore work with vast amounts of data to train the NMT system, as they have access to the original and translation of many patents. The EPO (European Patent Office), for example, only approves patents in English, German and French, but it is possible to find foreign patents on their patent search engine *Espacenet*[56] as well. All patents discoverable on Espacenet can be translated by the translation system on the site, either into English from one of 30 languages or from English into one of the other 30 languages (including Japanese).

This makes sense as according to Martin Schaller, a member of the Enrichment Application Services department in the EPO[57], the EPO has been partnered with Google since back in 2011 to offer this service and therefore, together with the general Google translation service, also switched to Google's NMT system in 2017. The NMT system used for patents is however trained individually and not part of the same model as the one used for the general-purpose translations on Google's own website.

Like the EPO, other patent offices, like the WIPO or the JPO, have already switched to NMT and are offering their respective "instant translation" for patents on their site.

Testing these translation systems reveals that more often than not, the translations are actually very usable and especially read quite fluently (at least as fluent as patent texts can be read). However, on closer inspection, some issues can still become apparent. In the following two sub-sections a very quick look at the EPO's *Patent Translate* service and the WIPO's *WIPO Translate* service will be taken, in order to see how they compare.

6.2 The EPO's Patent Translate

As shortly allured to above, the EPO works together with Google, to provide instant translation services on their website. According to the EPO[58] a comprehensive and up-to-date corpus of patent (parallel-)texts is provided to Google, which is organized into abstracts, claims and descriptions as well as being sorted according to the international patent classification system. Google then trains the NMT system based on that data. However, as a customer of the service,

[55] https://www.youtube.com/watch?v=-ZVplhqhyYM (accessed on December 20, 2019)
[56] https://worldwide.espacenet.com/ (accessed on January 05, 2019)
[57] The source is a personal conversation over e-mail.
[58] Source is the E-Mail conversation with EPO's Martin Schaller.

the EPO does not have insight into how the exact workflow of Google is. It is likely that Google draws from even more data than just the EPO's texts.

The most important point here, however, is that it is a closed system. This means, that as a user of the service, the EPO has little influence on how the translation is output. In fact, when using *Patent Translate*, it is not possible to see which choices the system made or to influence the system by, for example, choosing a specific domain. All is handled automatically, and the only thing provided is the finished translation of sentences; in the case of claims, potentially very long sentences. The system does also not allow for translation of text that is not in the *Espacenet* database, meaning that the system only provides translations for texts found through the search engine on the *Espacenet* website.

Following is a short analysis of the translation of one claim from the patent JP B1 6507295, a Japanese patent about knitting machines and needles. In order to keep it easier to read, color coding will be provided for respective sentence parts of the translated Claim 3. The translation was made in February of 2020; updates to the translation model may change the output of the *Patent Translate* service in the future.

「前記 第１の段部及び前記第２の 段部が、前記柄部の長手方向において、前記柄部 の基端 側から前記フック部側に向かうにつれて前記配列方向の幅が小さくなるテーパー状をしている請求項１又は２に記載の編機用編針。」

Text example 6.2-1: Claim 3 from JP B1 6507295

Text example 6.2-2 shows the EPO's NMT system's English translation:

"The first step portion and the second step portion have a tapered shape in which the width in the arrangement direction decreases from the base end side of the handle portion toward the hook portion in the longitudinal direction of the handle portion. The knitting needle for a knitting machine according to claim 1 or 2, wherein:"

Text example 6.2-2: EPO's NMT translation to English for claim 3 in JP B1 6507295

For reference, Text example 6.2-3 shows my own German translation, that I recently delivered to the EPO:

„Strick - oder Wirknadel für Strick- oder Wirkmaschinen nach Anspruch 1 und 2, bei denen der erwähnte Stufenabschnitt 1 und der erwähnte Stufenabschnitt 2 eine sich verjüngende Form aufweisen, bei der die Breite in der erwähnten Anordnungsrichtung, in Längsrichtung des

erwähnten Stielabschnitts, von der Basisendseite des erwähnten Stielabschnitts zur Seite des erwähnten Hakenabschnitts hin abnimmt.“

Text example 6.2-3: My own German translation of claim 3 in JP B1 6507295

There are two observations that can be made just by looking at the surface form of the machine translated text. First is the fact that it added a second sentence in a claim, which is not allowed in patent language. This is curious, as generally neural machine translation models are trained on a sentence by sentence basis; meaning Google might do something different here. Second, it seemingly had a hard time reversing the order of the sentence; notice how the second sentence in the machine translated text is actually the start of the sentence in the human translation. Within the individual sentences, however, the system did properly rearrange the order in which the words (and relative sentences) relate to each other and indeed produces a translation that reads very much like my human translation.

Looking closer, we can see that while the translation is actually quite good, some words have been dropped by the system, for example, the word 前記 (*zenki*), meaning “said” in English or “erwähnt” in German. This word is generally quite important to be translated, as it tells the reader whether this particular item has already been listed in the patent, but it does indeed hinder fluency and readability of the text. In fact, the word is often dropped in the final wording of English patents, as these are rewritten by patent lawyers to conform to the structure of the particular (national) patent standard. It is therefore likely, that Google uses published patent translations as training data, therefore the network learns to drop these adverbs. The same also happens with another word like 側 *(gawa)* meaning “side” which is once dropped as semantically it appears to make little difference.

By testing longer sentences, like the first claim[59], we can further exacerbate the issues observed above. The first claim of the same patent is, for example, split into 4 separate sentences. It can be observed, that the split happens whenever the system is not able to reorder the sentence, which generally seems to happen when relative sentences are quite long. Just as with the shorter claim, the same dropping of words can also be observed in the middle of the sentence, where the network fails to find the subject of a verb (有する, *yū suru*; meaning “possess/have”) and therefore drops the translation of the word completely, changing the meaning of the sentence drastically.

[59] The claim is almost a page long, so it won’t be quoted in this thesis. A copy of the ST and the translation will be provided on a Google Doc however, for anyone interested in comparing the two: https://docs.google.com/document/d/1AbF3BMOdrc3uAO5otMDCZ74zL6G7cdFeDSC3PSGitp0/edit?usp=sharing

The system produces translations that are more than good enough to give experts a good hunch on what the invention is about, and experts may even be able to extract the full meaning by looking at the illustrations. However, since the general structure is not conformant with patent requirements, important words are sometimes dropped and word relations may be lost, a post-edit would most certainly be required before the translation could be used. A mono-lingual post edit would likely not suffice (as generally grammar is not the problem). Arguably the machine translation can help experienced translators, too, as the word-choice and most of the inter-word relations are quite well selected by the system. On the other hand, the very high fluency of the text may make it harder for the translator or post-editor to see the flaws. This is however a whole different point of discussion, which has been elaborated and examined in other literature (see for example, Jia et al. 2019a, 2019b; Peris, Cebrián, et al. 2017; Sánchez-Gijón et al. 2019; Knowles et al. 2019).

6.3 WIPO Translate

The WIPO also provides an instant translation service on their website called *WIPO Translate*. It can be accessed directly from the website as a stand-alone service[60]. In contrast to the EPO, the WIPO uses an open source NMT framework called ***Marian NMT***[61]. The main advantage of using an open source framework would be that the service can be highly customized by the maintainer, i.e. in this case the WIPO itself.

In fact, it is rather apparent that some of this transparency is carried over to the users of the translation service on the website: Unlike the EPO's solution, the user is able to simply copy and paste a text into the translation mask and either let the system choose a domain or choose one manually. Additionally, once a text is translated, it is possible to see what segment of the sentence in the ST is translated to what segment of the sentence in the TT. Essentially, it is possible to see the **attention mechanism** in action.

What is even more interesting for us as translators, is that the system provides several translation suggestions when clicking on the provided translation and even accepts edits of the translation. Furthermore, the system allows to look up translation suggestions for individual terms in the ST by double-clicking on the term in question. It is also possible to segment long sentences at specific points, chosen by the user.

It is clear, that the open-source approach taken by the WIPO offers some tangible advantages in usability and transparency compared to what is offered on the EPO's website. The open-source framework in question, Marian NMT, is, however, also backed by a big

[60] https://www.wipo.int/wipo-translate/en/ (accessed April 25, 2020)
[61] https://marian-nmt.github.io/ (accessed February 27, 2020)

company from Silicon Valley: **Microsoft**. In fact, the short introduction to the framework on its GitHub-page reads: "Marian is an efficient, free Neural Machine Translation framework written in pure C++ with minimal dependencies. It is **mainly being developed by the Microsoft Translator team**." [62] Interestingly, amongst the users of this framework, the European Commission is one of them.

That said, let's look at how the system performs with the example sentence from the Japanese patent JP B1 6507295 that we looked at in Section 6.2, starting out again with the color-coded original claim 3 from said patent in Text example 6.3-1. Like with the EPO evaluation, the test was done in February of 2020; results may vary as the translation models are updated.

「前記 第１の段部及び前記第２の 段部が、前記柄部の長手方向において、前記柄部 の基端 側から前記フック部側に向かうにつれて前記配列方向の幅が小さくなるテーパー状をしている請求項１又は２に記載の編機用編針。」

Text example 6.3-1: Claim 3 from JP B1 6507295

" the knitting needle for knitting machine according to claim 1 or 2, wherein the first step part and the second step part have a tapered shape in which the width in the arrangement direction becomes smaller from the base end side of the handle part toward the hook part side in the longitudinal direction of the handle part. "

Text example 6.3-2: Claim 3 as translated by WIPO NMT

And for reference again, my own German translation of the same claim in Text example 6.3-3:

„Strick - oder Wirknadel für Strick- oder Wirkmaschinen nach Anspruch 1 und 2, bei denen der erwähnte Stufenabschnitt 1 und der erwähnte Stufenabschnitt 2 eine sich verjüngende Form aufweisen, bei der die Breite in der erwähnten Anordnungsrichtung, in Längsrichtung des erwähnten Stielabschnitts, von der Basisendseite des erwähnten Stielabschnitts zur Seite des erwähnten Hakenabschnitts hin abnimmt."

Text example 6.3-3: My own German translation of claim 3

Right away, we will notice that in Text example 6.3-2 the network managed to rearrange the sentence in a way that seems more natural to the English language and is in fact very similar to the way I personally chose to restructure the sentence in German. Staying on the surface-form of the sentence, we can notice that capitalization of letters has been removed (this can be

[62] See for more details https://marian-nmt.github.io/ (Accessed on April 15, 2020)

verified when testing the system with phrases or words that are generally capitalized in English, like languages or city names). This gives us an indication on how the model was trained, as removing capitalization may be beneficial by reducing the vocabulary size. In a language pair where casing is not essential, like English-Japanese, this makes a lot of sense; testing the System in German and English, however, reveals that in that case the WIPO trained the model to be case-sensitive.

With regards to repeating words, like "前記" *(zenki)* "said", we can observe the same behavior as with EPO's Patent Translate: The word is dropped for better fluency and readability, suggesting that models are trained on published translations rather than word-accurate translations.

Similarly, looking at the longer claim 1 from the same JP B1 6507295 patent as before, we can observe, that the WIPO NMT system appears to be slightly more resilient to the issues observed in the EPO's system.[63]

Interestingly, the WIPO also offers their earlier non-NMT translation models, based on a phrase-based SMT Model. Unfortunately, those models do not offer the Japanese to English pair, instead offering only the English to Japanese pair. For the sake of comparison, let's look at the following phrase pair from the test corpus that will be used in Chapter 7, by simply using the English sentence as the ST and the Japanese sentence as the TT.

(ST) FIG . 3 is a circuit diagram showing a construction of the frequency multiplication circuit in the second embodiment .
(TT) 図 3 は 、 この 実施 の 形態 に 係る 周波数 逓倍 回路 の 構成 を 示す 回路 図 で ある 。
(WIPO SMT) 図(3)の回路図に示す構成において、周波数逓倍回路の第 2 の実施の形態
(WIPO NMT) 図(3)は、第 2 の実施形態における周波数逓倍回路の構成を示す回路図である

Text example 6.3-4: Comparing SMT to NMT output with WIPO Translate

While this is obviously too small of a sample size to come to a definite conclusion, Text example 6.3-4 shows a general trend that can be observed with any other number of sentences and that has been largely observed by MT research (see Bentivogli et al. 2016; Moorkens 2018; Daems & Macken 2019). The SMT system provided solutions that are clearly recognizable as machine translations and can convey the gist of the meaning at best, but at worst they end up

[63] Find the ST and translation here: https://docs.google.com/document/d/1TjNkLHZMagiEs_S5-C_qPBJn8r1M57y2shCTIcE6W_8/edit?usp=sharing

completely unintelligible and unrelated semantically to the ST. The example above shows this quite well, as while the SMT system translates parts of the sentence correctly (likely the phrases learned during training), it fails to reorder them in a meaningful and grammatically correct way, obfuscating the original message of the sentence (it translates to something like: “In the circuit diagram of Fig. 3., the frequency multiplication circuit of the second embodiment”). The NMT solution on the other hand is quite close to the reference text and does in fact only vary from it, because the ST states “second embodiment” instead of “this embodiment” as the reference text does (この実施形態; *kono jisshi no keitai*). These kinds of divergence between ST and reference texts can often be observed in parallel text corpora meant for MT training. We will see in Section 7.6, that it might indeed be an issue, maybe less so for training, but more so for the evaluation of translation models’ output.

6.4 Summary

The jump in quality through neural machine translation can’t be denied. Many of the downfalls of rule based and statistical machine translation seem to have been solved. However, it is still dangerous to blindly rely on the output of the NMT systems: While the results often appear correct and well formulated (almost like if written by a human), at times, very important items are either mistranslated or left out altogether. Nonetheless, the output reads very fluently and therefore perceived translation quality may appear quite high.

It was interesting to analyze the different approaches the WIPO and EPO have taken towards implementing NMT into their systems, with arguably the WIPO system being more adequate for translators to work with, as the translation can be tuned slightly by the user. The output also seems to be slightly more precise, but the sample size used for this short overview is by no means big enough to give a scientific opinion about it.

The next chapter will be about the creation of an NMT-model, in order to try and recreate the findings above and see how different training variables and data selection influence the final output of the network.

7 Creating a Japanese-English Patent NMT model

In this chapter, the creation of a neural machine translation model is presented. The aim is to verify the hypothesis, that NMT favors *fluency* over *adequacy* and that training the system on domain-specific texts will enhance its performance. While these hypotheses have been confirmed in a broader sense by other publications before (Junczys-Dowmunt et al. 2016; Koehn & Knowles 2017), as of my knowledge this was not yet concretely tested in the rather controlled language of patent translation. In other words, in the cited publications, "domains" like Law, Medicine, IT and so on would also have used vastly different text-types for training, whereas in this thesis' case, the language will be limited to patent texts.

To achieve a controlled testing environment, a variety of NMT models using the state-of-the-art **Transformer model** will be trained on strictly patent parallel texts and subsequently the translation output of these models will be compared. In order to restrict the training to a certain domain, the international patent classification will be used to extract a specific domain from the corpus. The evaluation of the training results will be done both automatically by using the de-facto standard for automatic translation evaluation, the BLEU metric[64] (Papineni et al. 2001), but also through human evaluation based on the SAE J2450 automotive evaluation metric.

While this chapter provides an in-depth overview of the methodology, it may also be considered as a sort of tutorial for translators interested in getting familiar on a practical basis with NMT. The models will be trained on a regular gaming/multimedia personal computer. As mentioned often in the preceding chapters, neural networks benefit highly from a powerful GPU as the architecture of graphics processing units lends itself nicely to the computational requirements of neural networks. Gaming computers therefore offer a good platform for venturing into NMT research.

For reference, all training and subsequent experiments in this chapter will be performed on the hardware and software listed in Table 5.

[64] See Section 7.5.2.1 for further explanation of the BLEU metric

Table 5: Computer hardware used for this chapter

System specifications used for the experiments	
CPU	Intel Core i7 2600K @ 4.4Ghz
System memory	16GB DDR 3 1866Mhz
GPU	Nvidia GeForce RTX 2060
Video memory	6GB GDDR6
OS	Manjaro Linux (4.19.108-1)
Storage	256GB SSD (operating system, applications and training data) 3TB HDD (additional data, data preparation)

It is generally recommended to use a recent Nvidia GPU (GeForce GTX 9xx series or higher), as the GPU-acceleration in most NMT frameworks is coded using Nvidia's proprietary CUDA API (Application Programming Interface). The GPU should have access to as much video memory (VRAM) as possible, as memory capacity is often a big bottleneck when training neural networks. Additionally, making sure to have an ample amount of system memory (RAM) (at least 16GB) is equally important, whereas the performance of the CPU itself is not that important when training on the GPU.

While most of the toolkits and applications that will be presented in the next section are available for most current operating systems (Windows, Mac, Linux), the high transparency, ease of use and non-commercial nature of Linux as well as the very efficient on-board tools it provides, make it hard not to recommend using it. All the experiments and step-by-step guides found in the following sections will be based on Manjaro Linux[65], which in turn is based on Arch-Linux. For a slightly more accessible or Windows/Mac-like experience, a distribution like Ubuntu[66] may be used and most of the steps should still apply. The big advantage of Ubuntu is that most of the applications and dependencies used in the following experiments will be available in a pre-packaged form.

If no access to suitable computer hardware is available, Google offers a service called Google Colaboratory[67], where it is possible to use a virtual machine environment free of charge. The service even provides Free GPU acceleration for 12 hours, so it is well suited for running (smaller) NMT projects on it. The service can be accessed through any web-browser and runs completely on Google's servers. The Vienna Scientific Cluster (VSC)[68] offers a similar service

[65] https://manjaro.org/
[66] https://ubuntu.com/
[67] https://colab.research.google.com/ (Accessed on March 03, 2020)
[68] http://vsc.ac.at/home/

for, amongst others, students of Viennese universities. However, exploring these solutions would go beyond the scope of this thesis.

For the following sections, a basic understanding and interest in learning about the usage of and coding in Python will also be highly beneficial, as most frameworks tested are based on that programming language. Before writing this thesis, I had to learn some of the basics of Python to be able to more efficiently prepare the data and use the NMT toolkits. In the following sections, the scripts and programs I have written and used will be provided (see also Appendix II: Code and scripts). Table 6 is a list of online documentation and resources highly recommended for learning the basics of the underlying concepts.

Table 6: Python and NLP learning documents

Publication	Summary	Citation	URL
NLTK Book	A book about Natural Language Processing. Covers many of the Python basics with regards to NLP; great for people with even just very little pre-existing knowledge about coding.	Bird et al. 2009	https://www.nltk.org/book/
The Python Tutorial	General Tutorial on Python to get familiar with the way the language works and understand how to write code.	The Python Software Foundation	https://docs.python.org/3/tutorial/index.html

While the documentation available online is often very thorough, I will try to elaborate on issues I personally found difficult to wrap my head around, in the hopes of providing easier access to certain tools necessary for creating a neural machine translation model. When stuck, it is also very recommended to create an account on the website https://stackoverflow.com/ or at least consult it when needed. Often other people may have had exactly the same questions, so an answer may already be available. The community is very helpful, as long as the question is well researched and formulated.

All of the example code and commands in the following sections will be based on Manjaro Linux, so for recreating the coding environment of this thesis it is recommended to use the Manjaro Linux distribution. It is possible to use other Linux distributions as well, with the caveat of having to look up or know the respective commands for that distribution. Since most of the work happens on the command line interface (CLI), the major difference will be

how to access packages (i.e. applications and programs) as each distribution uses different repositories (servers were the packages are stored) and package managers (applications that download and install the packages). For further information consult the documentation of the respective distribution available online.

7.1 Procuring the training data

As mentioned in the theoretical part of this thesis, NMT is a corpus-based machine learning approach and therefore is dependent on vast amounts of input data; even more so than SMT before it. Before starting our exploration of NMT, we should therefore make sure that we have access to a large corpus of parallel text. Depending on the language-pair this can be a daunting task, as NMT requires corpora containing at least a few million words in order to even "get off the ground" as Koehn and Knowles put it (Koehn & Knowles 2017:4).

Generally, the bigger the dataset the better the results should generalize over new, unseen data. Using too small of a dataset also bears the risk of what is called *overfitting* a model, which means that the model will not generalize well over new data, because it is too specialized on the data that was provided during training.

Browsing the web, one may find many useful text corpora; some of them will be quite well known by translators, as best practice suggests looking at parallel texts for coherent translations. Keep in mind that corpora are not necessarily parallel texts. Mono-lingual corpora exist as well and are often used for unsupervised training or language-model training in MT studies. For NMT training, we will require sentence aligned parallel texts, so this is what we will be looking for. Luckily, SMT already required vast amounts of parallel text data for peak performance, so the data collection efforts that have been ongoing ever since are perfectly suitable for NMT training as well. For languages of the European countries (including English) some very useful parallel texts exist and can be easily found online. For example, the *European Parliament Proceedings Parallel Corpus 1996-2011* (**Europarl**)[69] offers a great starting point for a large variety of European languages. However, this thesis focuses on patent translation for the language pair English-Japanese, so the corpus had to be limited to patents of that specific language pair.

Luckily MT research is still very active in Japan and the *NTCIR-10 PatentMT (Patent Machine Translation) Test Collection*[70] was published for research use by the National Institute of Informatics (NII) Japan. It offers a parallel patent corpus with over 3.1 million sentence pairs in English and Japanese. Procuring the dataset required a written personal enquiry (by mail) to

[69] https://www.statmt.org/europarl/
[70] http://research.nii.ac.jp/ntcir/permission/ntcir-10/perm-en-PatentMT.html

the NII and was only possible through the backing of the project by my principal advisor Prof. Werner Winiwarter. The data was received in the form of a link with time-limited access and the download size of the complete (compressed) data was around 256GB (Gigabyte) in size; decompressing doubled the data size. It is therefore recommended to use a big hard-disk drive (HDD) to archive the data and only copy the relevant training data to a solid-state drive (SSD) for faster data preparation.

7.1.1 Pre-processing of data

While one big strength of the NMT paradigm is that no extensive pre-processing of data is required for the machine to learn, it is still recommended to prepare the data so that the system may optimally ingest it (see Domingo et al. 2018). First and foremost, it is paramount that the data is stored in a sentence-aligned format. Luckily, this is essentially standard in parallel text corpora.

However, most parallel text corpora store the aligned sentences in one single file, while most NMT toolkits will require separate TT and ST files, that keep the alignment by keeping the same line count for each file (i.e. line 1 in the TT file corresponds to the meaning of line 1 in the ST file). These two files are generally dubbed as the **training data** and are usually labeled **train** with the extension representing the language that the files contain. For this thesis' English and Japanese training set, we will therefore prepare a **train.en** file and a **train.jp** file. The training files should contain a vast number of sentences with several millions of words to be effective. Additionally, we will find that we need to prepare further data for the network to validate the training. This data must be different from the one we use in training, as it is used for testing the inference of the model on hitherto unseen data. Such data is often called **validation data** or **development data** and therefore shortened in a similar fashion to the training data as **val.xy** or **dev.xy**, with *xy* being the language of the file. Furthermore, in order to test the completed model, we will want to have yet another set of files, that has not been used in training or validation to check the translation quality of the model. These files are often referred to as the **test data** and are therefore generally labeled as **test.xy**, again with *xy* referring to the language they contain. The validation data should not be exceedingly large, as having too many sentences would only increase the training time and add significantly to the memory requirements of training.

Most NMT toolkits will also require two vocabulary files (one for the source and one for the target language). These files can be generated from the training data with the tools provided by the NMT toolkits or reused from other projects. The vocabulary files will be used by the network to create the **word-embedding** and should therefore be of a good size but also

not too large. The default maximum vocabulary size for most toolkits is 50.000 “tokens”. 50.000 tokens often don’t cover all the available tokens in training data, so tokens that appear less often are not stored in the vocabulary and simply replaced by an <unk> (unknown) token during training and translation.

Now, what is a token? Token refers to a single distinguishable unit in our data. Tokens can therefore be words, but also phrases, word-stems, symbols, spaces, letters or other, more abstract sub-word units like the Byte-pair encoding (BPE). In BPE the most common pair of consecutive bytes of data is replaced with a byte that does not occur within that data, allowing for compression of the language data and therefore enabling larger vocabularies while not using more tokens. The abstract nature of this approach lends itself well to Neural Machine Translation, which is why new tokenizers, i.e. programs that segment sentences into sub-word tokens, started to appear for this approach. Solutions, like ***sentencepiece***[71], therefore offer a language agnostic way to perform tokenization of text. This has been proven as efficient or even more efficient than classic, linguistics based tokenizers, at least for certain language pairs (see Sennrich et al. 2016, Kudo & Richardson 2018 and [72]). On the other hand, other experiments came to the contrary conclusion where classic tokenizers like the one from the SMT framework ***Moses***[73] or the tokenizer for the Japanese language ***mecab***[74] perform better (see Domingo et al. 2018). Combinations of both approaches (i.e. pre-tokenizing text with a language-specific tokenizer and then running a second BPE pass on top of it) also yielded good results in most publications. While this is a very exciting topic, for our testing we will stick to the classical tokenization as it keeps data human-readable throughout and performs very well.

7.1.2 Tokenization

So why is tokenization important? Looking at how NMT systems learn the language, the network should be able to figure out the patterns by itself. After all, most languages naturally come in a pre-tokenized form, as there are spaces in-between the words that help separating meaning units. However, especially for the Japanese language, this base form of tokenization is not present naturally.

If we were to ingest a Japanese sentence as is, the system would treat the whole sentence as a single token, as there are no spaces in-between words or symbols. Most tokenizers for Japanese therefore add spaces between Japanese words; this alone already makes the data ready

[71] https://github.com/google/sentencepiece
[72] https://github.com/google/sentencepiece/blob/master/doc/experiments.md
[73] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl
[74] https://github.com/taku910/mecab

for processing in NMT systems, but more can be done. For instance, *mecab* finds known *kanji*[75]-composita (i.e. the meaning of the word) and *hiragana*[76]-sequences in the text and separates them. Let's look at the first example sentence from the training corpus we will be using later on:

Original:

流体圧シリンダ３１の場合は流体が徐々に排出されることとなる。[77]

And here is the segmentation after we let *mecab* process the sentence[78]:

流体 | 圧 | シリンダ | ３ | １ | の | 場合 | は | 流体 | が | 徐々に | 排出 | さ | れる | こと | と | なる | 。

We can see that known composita, like 流体 literally standing for "fluid body" meaning "liquid" or "fluid", or the *katakana*[79] word シリンダ (cylinder) have been kept together; while some less known combinations like 流体圧 (liquid pressure) had a space added between the first two characters and the last. Curiously the number 31 was also spaced out to "3" and "1", which theoretically enables the system to learn the pattern of digit combinations itself.

Latin text tokenizers work in much the same way, separating punctuation from words and automatically marking hyphenation and other special typesetting, so the system knows the hyphen or typesetting is not part of the word per se.

It is important to note here, that many NMT toolkits offer their own form of tokenization or have some of the tools mentioned above as a part of the whole package. It also appears that different toolkits have varying performance, depending on what tokenization was used. For that reason, it is recommended to choose one toolkit to work with and stick to one form of tokenization. In this thesis' case the classic linguistic tokenization was chosen, so that it is easier to attain comparable results that are human-readable throughout the whole workflow.

[75] The adopted logographic Chinese characters used in Japanese writing, that also carry several meanings depending on usage.
[76] One of the 3 components of the Japanese writing system. Syllabary used mainly to write conjugational endings following a kanji root, various function words, including particles, and other native words for which there are no kanji or whose kanji form is not commonly used.
[77] The sentence translates to: "When the fluid pressure cylinder 31 is used, fluid is gradually applied."
[78] Spaces marked manually with " | "
[79] The last of the 3 components in the Japanese writing systems. Used mainly to write foreign words and to represent onomatopoeia.

7.2 Which NMT toolkit to use

Thanks to the thriving open source community around NMT and NLP in general, we have a large selection of tools at our disposal that we may use for creating our own NMT model. Table 7 lists some of the most popular and updated NMT toolkits on the development platform GitHub[80], that also offer good documentation. For this thesis all the listed tools were tested, and the annotations are based on personal experiences with the toolkits. Please keep in mind, that results may vary depending on the hardware, the operating system or even the data that is used. Also, since most of these toolkits are still being developed, the features they offer, as well as their efficiency may change over time.

Table 7: A selection of NMT toolkits analyzed

Toolkit	**Citation/URL**	**Framework**	**Advantages/Disadvantages**
Tensorflow NMT	M.-T. Luong et al. 2017 https://github.com/tensorflow/nmt	TensorFlow	+ Good documentation + Replicates (old) Google NMT + Automatic BLEU evaluation + Highly tweakable - Only attention-based RNN NMT architecture (no Transformer) - No longer actively developed - Setup rather complex
OpenNMT-py	Klein et al. 2017 https://github.com/OpenNMT/OpenNMT-py	PyTorch	+ Great documentation + Offers Transformer model + Good balance of customization and accessibility + Actively developed + Tensorboard monitoring + Made for research + Efficient (works well on low memory GPUs) + Provides many useful tools for data preparation - No automatic BLEU evaluation - Slightly lower scores for same amount of training time as TF models
OpenNMT-tf	Klein et al. 2017 https://github.com/OpenNMT/OpenNMT-tf	TensorFlow	+ Offers Transformer model + Highly tweakable + Actively developed + Tensorboard monitoring + Automatic BLEU evaluation during training + Fast, if configured correctly + Allows mixed-precision training on newest GPUs with dedicated Tensor-cores (very fast!) - Less efficient (needs more memory) - Setup complicated - Lacks some useful external tools - Documentation a bit scarce

[80] Popularity as measured by github. More stars on github indicate higher popularity

Toolkit	Citation/URL	Framework	Advantages/Disadvantages
nmt-Keras	Peris & Casacuberta 2018 https://github.com/lvapeab/nmt-keras	TensorFlow Theano (deprecated)	+ Good documentation with background explanation + Made for research + Automatic BLEU evaluation + Easy set up and configuration - Rather slow - Not geared towards creating big models - TensorFlow complications - Theano backend no longer developed
JoeyNMT	Kreutzer et al. 2019 https://github.com/joeynmt/joeynmt	PyTorch	+ Very accessible documentation + Scripts and configs well commented + Made for novices + Very simple setup + Automatic BLEU evaluation - Does not include some useful pre-processing tools (tokenization, length-filtering, etc.) - Slightly less tweakable - Tutorials meant for learning and less for quickly building a model - Does not cope well with large data sizes (requires more memory than other toolkits)

Some of the toolkits, like OpenNMT, also explicitly allow training of cross-mediatic inputs, like image/audio to text or vice-versa. The NMT toolkits are in fact a means of facilitating the interface between human and machine, providing useful features like checkpoint saving, automatic evaluation and data preparation, as well as communicating with the deep learning framework so that it performs the calculations needed to train the network. As long as we have features (the input) that we can attribute to labels (the output), the underlying architecture can start finding patterns and associate said features to the labels (or the ST to the TT for that matter).

As may become clear from the annotations in Table 7, the toolkits are based on different frameworks (TensorFlow, Theano, PyTorch). These frameworks essentially provide the support for neural networks/deep learning architectures, meaning they manage the vector calculations neural networks are based on. Since Theano is no longer actively developed, the only two frameworks tested in this thesis are Google's TensorFlow[81] and Facebook AI Research's PyTorch[82]. In the next sub-sections I will provide a short overview over the tested toolkits in order to explain why I decided for one particular toolkit amongst them.

7.2.1 Tensorflow NMT

Tensorflow NMT was authored by Thang Luong, Eugene Brevdo and Rui Zhao (M.-T. Luong et al. 2017). The aim was to provide a toolkit that reached state-of-the-art translation quality,

[81] https://www.tensorflow.org/
[82] https://pytorch.org/

while also offering a tutorial that "gives readers a full understanding of seq2seq models and shows how to build a competitive seq2seq model from scratch". The system is based on an attention RNN Encoder-Decoder model and aimed to replicate Google's NMT (GNMT) system. Both Multi-GPU as well as CPU training are supported through the TensorFlow framework.

The tutorial offers a great overview of NMT and the workings behind the offered RNN encoder-decoder architecture with attention. However, the toolkit was last updated in February of 2019[83] and is therefore the least up-to-date of the tested toolkits. For that reason, it does not offer the state-of-the-art Transformer model. While the tutorial is great for theoretical background, it is rather geared towards users with IT background and does not offer a real high-level API to communicate with TensorFlow. Setting up a model is therefore much more complicated than with other frameworks and in that respect not well suited for our needs. Since we also want to test our hypothesis on the newest available model architecture, the Transformer, this toolkit will not be used for our experiments.

Yet, it is still worthwhile reading through the documentation and following along with the tutorial if possible. However, the installation of TensorFlow needed for following the tutorial may be rather tedious, as the toolkit is built around an older nightly release and therefore not compatible with the newest stable releases of TensorFlow.[84]

7.2.2 OpenNMT-py

OpenNMT-py is a Python port of the original OpenNMT toolkit based on the now deprecated Lua version of Klein et al. (2017) and was initially created by Adam Lerer and the Facebook AI research team (Klein et al. 2017:3). However, OpenNMT is now generally developed as completely open-source at http://github.com/opennmt. The OpenNMT project is described as following:

> The system prioritizes efficiency, modularity, and extensibility with the goal of supporting NMT research into model architectures, feature representations, and source modalities, while maintaining competitive performance and reasonable training requirements. The toolkit consists of modeling and translation support, as well as detailed pedagogical documentation about the underlying techniques. OpenNMT has been used in several production MT systems, modified for numerous research papers, and is implemented across several deep learning frameworks.
>
> (Klein et al. 2017:1)

The toolkit is therefore very approachable, and installation is quite simple as all dependencies are automatically fetched by the installation command. The toolkit offers many

[83] As of this writing, March 2020.
[84] A TensorFlow 1.4 (stable) version of the tutorial is available: https://github.com/tensorflow/nmt/tree/tf-1.4, that however needs a work around for a beam-search bug in the TF framework.

templates for recreating known working architectures, like LSTM RNN Encoder-Decoder architecture or the Transformer. In my case, everything worked right out of the gate and I had model training up and running in no time after following the excellent online tutorial[85]. OpenNMT-py also offers many useful tools as part of the package. Rather than being implemented as direct commands, these external tools are stored in separate folders of the project directory and must be called separately from the main OpenNMT program. On one hand, that makes it easier to run the tools separately as part of a script, but conversely it also means that BLEU evaluations can't be performed during training.

The toolkit uses PyTorch as the underlying Framework and therefore also supports CPU and Multi-GPU training. In my personal testing, I have found that PyTorch is less difficult to set up and requires much less memory than TensorFlow. Initially I therefore gravitated towards OpenNMT-py, but testing has shown, that the sheer speed of TensorFlow makes up for the higher memory requirements. A short overview over my findings will be provided in sub-section 7.2.6.

7.2.3 OpenNMT-tf

OpenNMT-tf is a recent addition to the OpenNMT project "focusing on large scale experiments and high performance model serving using the latest TensorFlow features" (Klein et al. 2017:4).

Just like OpenNMT-py, it is part of the ongoing open-source OpenNMT project and offers many of the same functions and features. While it also offers many pre-made configuration templates recreating architectures like the Transformer, I found that the setup was slightly more involved than with OpenNMT-py. First and foremost, the online documentation[86] is less fleshed out than that of the PyTorch counterpart and second, TensorFlow requires more external and more specific dependencies than PyTorch. It also required more hyperparameter tuning than PyTorch, as Tensorflow apparently requires more memory than an equivalent architecture running on PyTorch (see sub-section 7.2.6).

However, using the latest TensorFlow features enables very fast training, especially if using the latest hardware. By leveraging TensorFlow's automatic mixed-precision training it was possible to make use of the novel GPU architecture in the training computer's RTX 2060, accessing its tensor cores. This provided a boost in training performance of almost 50%, which is quite a lot considering model training can take up to several days depending on architecture, hyperparameters and training data. At the time of writing this thesis, OpenNMT-py offered no

[85] Full documentation for OpenNMT-py available here: https://opennmt.net/OpenNMT-py/
[86] Documentation for OpenNMT-tf is available here: https://opennmt.net/OpenNMT-tf/

such option according to documentation. Additionally, the toolkit is more integrated into the command line and so enables us to use BLEU evaluation during training, which in turn enables us to stop training automatically when the BLEU score no longer significantly improves.

7.2.4 nmt-Keras

nmt-Keras was developed by Álvaro Peris and Francisco Casacuberta (Peris & Casacuberta 2018) and is based on the high-level deep-learning API Keras and their own "Multimodal Keras Wrapper" written in Python, which enables easy management of models and datasets as well as automated evaluation. The calculations were originally performed on Theano, but this has since shifted to TensorFlow in later releases. While theoretically the reliance on the high-level API Keras makes the whole package very accessible and easy to setup, complications with the no longer officially tested Theano framework, some not yet implemented TensorFlow features, the reliance on an older TensorFlow version (1.15.2 at the time of writing) and not implemented functions for the Transformer architecture can be detrimental when trying to work with the toolkit. I believe it is a shame, as the main focus of this toolkit is putting "particular emphasis on the development of advanced applications of neural machine translation systems, such as interactive-predictive translation protocols and long-term adaptation of the translation system via continuous learning." (Peris & Casacuberta 2018:1), so it would appear to be the closest that any of the toolkits get to including post-editing and translators as a part of the equation.

The documentation is very well made and much less IT-centric than some of the other toolkit's documentations. Likewise, the comments in the different files of the toolkit, like the config.py, are very exhaustive and helpful. As mentioned, while the toolkit also supports the Transformer architecture, it does not yet fully support all of the needed functions[87] and it was therefore not possible for me to get the toolkit up and running at an acceptable speed. In fact, training on the GPU was off the table for the larger patent dataset, as the toolkit used a lot more memory than any other toolkit for the same training data and would regularly throw up error messages and abort training.

I can therefore not recommend using this toolkit for more involved projects until it is fully featured and uses more up-to-date dependencies. Just like with Tensorflow-NMT however, it is worthwhile reading the documentation and following along with the examples provided by the authors as they give great insight into how neural machine translation modeling works. The

[87] At the time of writing, 27 March 2020, it did not support the "noam" reducer function that is used by default in the Transformer architecture.

authors even provide iPython notebooks[88] on Google Colab, where it is possible to run all of the code in a web-browser while reading the explanations of the authors.[89]

7.2.5 Joey NMT

The final toolkit I examined is **Joey NMT**, which is the most recent effort amongst the tested toolkits. It was made by Julia Kreutzer and Stefan Riezler of the Heidelberg University and Joost Bastings of the University of Amsterdam. The aim of the toolkit was to create a "minimalist neural machine translation toolkit based on PyTorch that is specifically designed for novices. Joey NMT provides many popular NMT features in a small and simple code base, so that novices can easily and quickly learn to use it and adapt it to their needs" (Kreutzer et al. 2019).

Installation is quite simple and can be achieved quickly by following the excellent documentation either online or in the handbook that was crafted for students as a script for lectures at university[90]. Despite focusing on simplicity, Joey NMT offers support for most important NMT architectures including LSTM RNNs and Transformer architectures. Usage is very similar to OpenNMT but slightly more streamlined, with a lot of configuration examples/templates that emulate the best models of big machine translation workshops.

I highly recommend testing Joey NMT and following the excellent examples in their tutorial for really getting familiar with how NMT works. Joey NMT also incorporates many useful tools to visualize training progress and model characteristics: Besides tensorboard integration, it allows to visualize attention weights and learning curves as well, which can be very useful for understanding NMT.

While theoretically it would be possible to use Joey NMT for larger projects like the experiment in this thesis as well, at the time of writing it is not yet optimized enough to run big datasets on a small memory footprint and has a bug regarding the batch_multiplier[91], which would enable more efficient training on limited memory capacity. The toolkit has a lot of promise in regard to educational usage and while it will not be used for creating the models in this thesis, it is worth considering as an educational tool for introducing NMT to Translation Studies students.

---

[88] Interactive notebooks that allow the execution of code.
[89] Find the tutorial by Peris and Casacuberta here: https://colab.research.google.com/github/lvapeab/nmt-keras/blob/master/examples/tutorial.ipynb
[90] https://readthedocs.org/projects/joeynmt/downloads/pdf/latest/ (accessed on March 12, 2020)
[91] I pointed out the issue and workaround was already suggested by one of the authors here: https://github.com/joeynmt/joeynmt/issues/90#issuecomment-605427609 (accessed on March 28, 2020)

7.2.6 Selecting the most adequate toolkit

From the roundup above, the two OpenNMT versions provided the most accessible and feature-complete package. They also provide high performance when creating NMT models using the Transformer architecture, even on relatively small video memory. In this sub-section a direct comparison between the two OpenNMT toolkits will be made. The section will be comparatively technical, but any technicalities that appear unclear now, will be further elaborated in Section 7.3 and up. The two toolkits were tested using the NTC7 optics-only dataset (see sub-section 7.4.1.5) and standard parameters for the "Transformer Base" architecture (Vaswani et al. 2017), with only the batch size[92] being adjusted.

Since the task is to create several models, performance and efficiency needed to be evaluated. The main difference is with memory consumption, which is higher on the TensorFlow powered OpenNMT-tf. This means, that to run training on TF, we would either have to use a GPU with more memory or reduce the training batch size. Popel and Bojar empirically found that reducing the batch size means that training will take longer and potentially deliver slightly worse results for the same amount of training time. However, if the *effective* batch size[93] is adjusted, the results should be the same as if using a larger batch size (Popel & Bojar 2018). In order to get comparable results, effective batch size was therefore kept to the common lowest denominator of 4096 in the first test. To get the best results of each individual toolkit however, OpenNMT-py was also tested at an effective batch size of 6144 (batch size 3072 x 2; the maximum that would fit the 6GB of VRAM), and in OpenNMT-tf mixed-precision training was utilized to speed up the training process (memory requirements actually grew with mixed-precision training, opposite to what might be expected; the reason for this seems to be, that both FP16 (Floating-point 16-bit) and FP32 (Floating-point 32-bit) values are kept in memory, something that may be resolved in a future TensorFlow version)[94].

[92] Essentially the number of words/tokens the system ingests at a time.
[93] Essentially multiplying gradient updates of x individual batch sizes to get the same result as using a larger batch size.
[94] https://forum.opennmt.net/t/settings-for-training-transformerbig-with-mixed-precision-on-single-gpu/3532/8?u=dixxy (accessed on 23.03.2020)

Table 8: OpenNMT-tf and OpenNMT-py memory consumption, accuracy and speed compared

Transformer Base model training memory consumption and speed				
Toolkit	**Batch size (Effective)**	**Speed (Tokens/s)**	**Memory-consumption**	**BLEU after 60k/40k* *steps*[95] (time needed)**
OpenNMT-tf (2.8.1)	1024 (4096)	ST: ~6400 tok/s TT: ~6000 tok/s	~ 5100MB	38.25 (~9.8h)
OpenNMT-py (1.0.2)	2048 (4096)	ST: ~6500 tok/s TT: ~6050 tok/s	~4800MB	32.19 (~9.4h)
OpenNMT-tf (2.8.1) Mixed-precision	1024 (4096)	ST: 9500 tok/s TT: 8900 tok/s	~ 5800 MB Non-critical OOM errors after saving first checkpoint	37.25 (~6.5h)
OpenNMT-py (1.0.2)	3072 (6144)	ST: 7200 tok/s TT: 6700 tok/s	~ 5900MB	33.47 (~8.6h)[96]

As Table 8 illustrates, OpenNMT-py performs slightly faster when calculations are done using classical full-precision (FP32) training, since it allows a bigger batch size to fit the GPU memory. However, OpenNMT-tf is very fast even at half the actual batch size, suggesting that the TensorFlow framework is capable of very efficient calculations through the GPU, as long as enough video memory is available. On the other hand, OpenNMT-py's efficient memory usage allows a batch size of 3072 (resulting in an effective batch size of 6144), which speeds up training by approximately 10.5 % from ST 6500 tok/s to 7200 tok/s and TT 6050 tok/s to 6700 tok/s respectively and slightly improves model accuracy according to the BLEU score. On the other hand, OpenNMT-tf supports mixed-precision training (Micikevicius et al. 2017), which makes use of special hardware on the latest Nvidia GPUs (Nvidia Volta and Turing architectures used in the Titan V, Tesla- and RTX-series respectively) and TensorFlow's automatic mixed-precision optimization to accelerate the training process. This gives an incredible performance increase of approximately 48.5% from ST 6400 tok/s to 9500tok/s and TT 6000 tok/s to 8900 tok/s, with only a slight loss in model accuracy according to the BLEU score (38.25 to 37.25). We can also observe that general model accuracy, as measured by BLEU, is higher for OpenNMT-tf than it is for OpenNMT-py. The reason for this is not quite clear, but

[95] 1 Step = One effective batch size of tokens (4096) was processed. 60k steps means the full data has been seen approximately 5-6 times (5-6 *Epochs* completed) on a data-set with approx. 47m words.
[96] BLEU score and time for the effective batch size of 6144 was evaluated after 40k steps, which provides the same data coverage as 60k steps on 4096.

it could be down to the used frameworks, the way the toolkits prepare the input data (vocabulary) or slight variances in the way the Transformer architecture is implemented in the two toolkits. Regardless, "out-of-the-box" performance is higher for OpenNMT-tf.

The gain in speed thanks to mixed-precision training and the higher base-line accuracy is essential for creating many models in a relatively short amount of time and this makes OpenNMT-tf the better choice over OpenNMT-py[97] for this project. Additionally, OpenNMT-tf offers some features, like BLEU evaluation during training, that help to streamline the training process and enables us to stop training automatically once BLEU no longer significantly increases.

Unfortunately, TensorFlow offers some inherent challenges for the setup of OpenNMT-tf. For one, as was shown in the small benchmark above, it requires more memory than PyTorch for the same model parameters, so some tuning will be required. In addition, there seem to be some unresolved bugs with newer Nvidia GPUs (RTX series), that result in aborted training unless the "allow_growth"-flag is specified[98].

```
--gpu_allow_growth
```

CLI 1: Command argument needed for RTX GPUs on TensorFlow framework

Adding the arguments in CLI 1 to our training command, something that is not documented well, allows training to start on RTX GPUs with OpenNMT-tf. However, since this flag essentially enables the framework to gradually build up memory usage on the GPU it might also result in an out of memory error (OOM) during training, if the GPU memory is otherwise utilized (by opening too many browser tabs for example) or particularly long sentences appear in the training batch. Therefore, utmost care must be taken when selecting training data and batch size.

[97] Mixed-precision training is still in experimental stage for openNMT-py: https://github.com/OpenNMT/OpenNMT-py/pull/1208 (accessed on March 25, 2020)
[98] https://github.com/tensorflow/tensorflow/issues/24496 (accessed on March 25, 2020)

7.3 General recommendations before starting

Since the installation process is well explained by the documentation of each individual toolkit presented in this thesis, I would rather like to present the reader with some additional general recommendations and some of the issues I encountered when preparing my working environment.

What follows are general recommendations when working with NMT toolkits and practical tips on how to deal with possible issues:

1) Use a Python virtual environment

Virtual environments enable you to install Python packages separated from the main system. Since most toolkits and dependencies are available as Python packages, it is possible to run them in different versions than what is running system-wide and this makes sure that the dependencies and the Python runtime of the environment remain unaffected from system updates. However, not all the necessary items are available through Python packages, so some care must be taken when updating the system (for example, Nvidia's CUDA toolkits are generally only available as system packages). With Python installed, the standard Python command (the command starts after the "$" prompt) for creating a virtual environment on Linux is shown in CLI 2.[99]

```
$ python -m venv /path/to/new/virtual/environment
```

CLI 2: Standard command for creating a virtual environment

However, to facilitate this process, I would recommend installing the *virtualenvwrapper*[100], which helps tremendously with creating and organizing the virtual environments. It is recommended to install *virtualenvwrapper* system-wide. To install it in Manjaro, first switch to the so-called root user (often also wrongly referred to as "super user") by typing the command in CLI 3.

```
$ su -
Password: (Type your password)
#
```

CLI 3: Switch to root user

[99] We assume that we will be using Python 3, so the "python" command is the same as the "python3" command: https://docs.python.org/3/library/venv.html (accessed on 26.03.2020)
[100] https://virtualenvwrapper.readthedocs.io/en/latest/ (accessed on 26.03.2020)

The prompt should now change to a "#" instead of the dollar sign "$", indicating that you are now executing commands as the root user.

To install *virtualenvwrapper*, type the command in CLI 4.

```
# pacman -S python-virtualenvwrapper
```

CLI 4: Install virtualenvwrapper

After the installation, you should return to the regular user by simply typing

```
# exit
```

CLI 5: Log-out of current terminal session

If you are using another distribution (like Ubuntu) you may need to use the *sudo* command instead. This installs the wrapper on a system level, enabling many useful commands for managing virtual environments. Some of the commands I used most during the experiment are listed in Table 9[101]:

$ mkvirtualenv env_name	create and activate virtual environment with "env_name"
$ workon env_name	activate virtual environment "env_name" (needs to be created first)
$ setvirtualenvproject	set project folder for active virtual environment (automatically jump to this folder when activating virtual environment)
$ deactivate	deactivate active virtual environment
$ lsvirtualenv	list all available virtual environments
$ rmvirtualenv env_name	irreversibly delete virtual environment

Table 9: Commands for virtualenvwrapper

2) Make sure to use a supported Python distribution for TensorFlow/PyTorch

By default, most Linux distributions come with the latest Python version installed. This is not always supported by the Deep Learning Frameworks. While PyTorch is relatively quick to adapt to a new Python release, TensorFlow is often one or two major versions (3.**x**) behind. At the time of writing, the stable build of TensorFlow is only compatible with Python 3.5 through

[101] For a full list of commands see: https://virtualenvwrapper.readthedocs.io/en/latest/command_ref.html (accessed on March 16, 2020)

to 3.7 while Python 3.8 is not supported. In order to check the Python version, just type the command shown in CLI 6.

```
$ python --version
```

CLI 6: Display Python version

The terminal should output the version in the next line as shown in CLI 7.

```
Python 3.8.2
```

CLI 7: Python version 3.8.2 is installed

If you have a supported Python version, all is well. If not, you will have to locally install the specific Python version you want (make sure not to install it as a system-package, as that may break many other things) and then specify the Python version you want to create the virtual environment with. What should work on Manjaro Linux (and most other Linux distributions) [102] is the following set of commands shown in CLI 8:

```
Switch to downloads folder
$ cd /home/user/Downloads
Download Python version from python.org
$ wget https://www.python.org/ftp/python/3.7.7/Python-3.7.7.tar.xz
Extract and change into directory
$ tar xf Python-3.7.7.tar.xz
$ cd Python-3.7.7
run configuration (with optimizations)
$ ./configure --enable-optimizations
make alternate installation as root user, will be 'python3.7'
$ sudo make altinstall
```

CLI 8: Step-by-step install of Python 3.7.7 as non-system package

This should install Python 3.7 to the /usr/local/bin/python3.7 binary folder, while leaving the system Python install alone. If /usr/local/bin is in $PATH[103], then simply create a new virtual environment with Python 3.7 through virtualenvwrapper by typing the commands in CLI 9:

```
$ mkvirtualenv --python=python3.7 env_name
```

CLI 9: Create virtual environment with virtualenvwrapper using Python 3.7 runtime

[102] Linux distributions like *Gentoo*, will allow you to install different Python versions easily, in a so-called slot-system. In most other distributions (and especially OS's like Windows) the process is a bit fiddlier.
[103] $PATH is an environment variable, that indicates to the system where to look for programs.

If this does not work, because /usr/local/bin/ is not in $PATH, either add it to your $PATH by following guides online or type out the absolute path to the Python 3.7 installation as shown in CLI 10:

```
$ mkvirtualenv --python=/usr/local/bin/python3.7 env_name
```

CLI 10: Alternate command for creating Python 3.7 environment

When running the "python --version" command in the virtual environment, it should now state the same line as shown in CLI 11:

```
Python 3.7.7
```

CLI 11: Python 3.7.7 is installed

3) Text encoding (use UTF-8)

Text encoding (or code page) may seem like a small thing, but it can make working with different languages very challenging on computer systems. Unfortunately, text encodings were established in the early days of computers and are generally not inter-compatible. Even worse, there is no 100% accurate way to detect the text-encoding used in a document.

For example, English documents may use the ASCII standard (American Standard Code for Information Interchange) which only supports encoding of 128 (!) symbols or the extended ANSI (American National Standards Institute) standard with support for 256 symbols. Asian languages would obviously never fit this limitation. Japanese texts therefore tend to use encodings like EUC-JP (Extended UNIX Coding – Japanese) or Shift-JIS, which extend the encoding possibilities but result in compatibility issues and unreadable symbols if the wrong decoding code page is chosen. Text example 7.3-1 shows an example of this issue.

Decoder	**EUC-JP encoded Japanese Sentence**
ANSI	¤½¤ÎÂ¾¡¢¥¨¥Ý¥-¥·¼ù»é¡¢¥¢¥¯¥ê¥ë¼ù»é¡¢¥·¥ê¥³¡¼¥ó¼ù»é¡¢¥¦¥ì¥¿¥ó¼ù»éÅù¤¬ÊÝ¸îÁØ£±£µ¤ÎºàÁ¤È¤·¤ÆÍÑ¤¤¤é¤ì¤ë¡£
EUC-JP	その他、エポキシ樹脂、アクリル樹脂、シリコーン樹脂、ウレタン樹脂等が保護層１５の材質として用いられる。

Text example 7.3-1: Decoding with wrong code-page compared to the right code-page

Luckily, a standardized way of encoding exists with the UTF standard. The most common is UTF-8, which is also the standard for Linux and Mac OSX Operating Systems[104]. Unfortunately, often text-corpora are still stored in local encodings like EUC-JP, as was the case for the patent data used in this thesis and this can be a major headache for working between different languages. It is therefore recommended to convert all text-files into UTF-8 before using the data in NMT training. There are several ways to do this, but for this thesis the Python script convert_encoding.py[105] was used.

4) Test small data-sets first and then run the system on real data

Generally, it is advised to first get familiar with the toolkits, hyperparameters and NMT architectures by testing toy-examples or smaller datasets. This way, test trainings don't take too long and it is much quicker and less convoluted to troubleshoot a problem. Once training runs without errors, it is possible to simply change the input data and, if needed, adjust hyperparameters like the batch size to account for the higher memory requirements of the bigger training data.

5) Plan ahead, allow for enough checkpoints to be saved and be patient

Training can take several hours and up to days depending on the used toolkit, the training parameters, the hardware and the training data. In order to avoid losing the whole progress of a training when the computer crashes or there is a power-outage, make sure to allow the toolkit to regularly save checkpoints, so training can resume at a later date.

[104] Windows is based on UTF-16, but Microsoft implemented system-wide UTF-8 support as BETA and is working on making Windows more UTF-8 friendly in the future. (https://blogs.msdn.microsoft.com/commandline/2018/07/20/windows-command-line-inside-the-windows-console/; accessed on 30.03.2020)
[105] https://github.com/goerz/convert_encoding.py (accessed on 28.03.2020)

7.4 Preparing the data (OpenNMT-tf)

In Section 7.2 it was established that the toolkit of choice would be OpenNMT-tf, as it showed the best performance of all the available toolkits and the most convenient feature-set. Installation is fairly straight-forward, and I would like to point the reader towards the excellent QuickStart-guide of the toolkit[106]. Instead of the suggested *virtualenv* command I would suggest using the *virtualenvwrapper* to create the virtual environment as explained in Section 7.3.

Once OpenNMT-tf is installed in the virtual environment, create a working folder for your toolkit, where you can store all of the models, configuration files, etc. Notice that a lot of dependencies and external tools are automatically installed along OpenNMT-tf, for instance, TensorFlow and Python-wrappers for Mecab[107] as well as an extended version of BLEU (SacreBLEU)[108] should be installed automatically. Unfortunately, the additional tools are not well documented in the OpenNMT documentation, so if unsure it is best to resort to external tools and follow the respective documentation.

For preparing our datasets, we will rely on the Moses Tokenizer (Perl-script) and MeCab to create tokenized versions of our training files and then use the OpenNMT command to generate the vocabulary out of these files.

7.4.1 Preparing datasets

In this sub-section, a step-by-step preparation of the parallel sentence data (PSD) from the *NTCIR-10 PatentMT (Patent Machine Translation) Test Collection*[109] will be presented. In this thesis, the NTC7 training subset of the collection was used, which provides about 1.8 million EN-JP parallel sentences from a vast variety of patents and therefore domains. After decompression, the training and patent files are stored in a very deep but well-organized folder structure. The original patent documents themselves are stored by year and then in several sub-directories within those year-folders. The only other folder that is relevant for this thesis is the "ntc8-patmt-train"-folder which contains both NTC7 and NTC8 training data. See Figure 29 for a visual representation of the folders in question.

[106] https://opennmt.net/OpenNMT-tf/quickstart.html (accessed on January 14, 2020)
[107] https://pypi.org/project/mecab-python3/
[108] https://github.com/mjpost/sacreBLEU
[109] http://research.nii.ac.jp/ntcir/permission/ntcir-10/perm-en-PatentMT.html (accessed on January 20, 2019)

The NTC7 training data is provided as a compressed file named “train.tgz”, that in turn contains seven separate files: train.txt, dev1.txt, dev2.txt, dev3.txt, pat-ids.txt, training-ids.txt and readme.txt. The reason for choosing NTC7 over NTC8, is that the training data is of a smaller size and more adequate for the scope of this thesis.

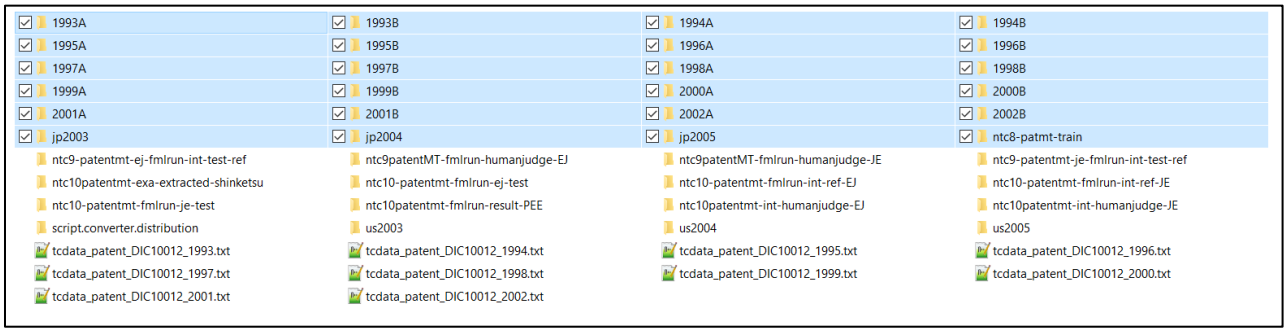


Figure 29: Root folder structure of the NTCIR-10 PatentMT Test Collection

The “readme”-file contains important information about where the data is from and how it is presented. All the data was taken from *A Japanese-English patent parallel corpus* (Uchiyama & Isahara 2007) and each sentence pair in the files is stored in five columns like shown below:

SSR ||| DOCID ||| TID ||| JA ||| EN

These fields have the following meanings:

SSR: Sentence-alignment score
DOCID: ID of the document from which the sentence pair is extracted
TID: ID of the sentence pair in document DOCID
JA: Japanese sentence
EN: English sentence

The DOCID, JA, and EN columns of the files will be used for generating the several files needed for the NMT toolkit to operate correctly (see Section 7.1.1). First, however, the files must be converted into UTF-8, as they are stored in the EUC-JP code-page. This goes for both the train and dev files, as well as all of the Japanese original patent files, that we will be using later on to determine the domain of the individual sentences in the PSD.

7.4.1.1 Converting files to UTF-8

There are many ways to convert files from one code-page to another. In this thesis the Python script/tool convert_encoding.py[110] was used, as it works great for batch processing. Usage is as simple as storing the convert_encoding.py in one of the directories on your $PATH[111] and typing the command shown in CLI 12.

```
$ convert_encoding.py [options] file1 file2 ...
```

CLI 12: Convert encoding command with arguments

However, since the script was written in Python 2, it might be necessary to change the so called *hashbang* or *shebang*[112] at the beginning of the script. Use a text editor of your choice and change the first line of the script from the content shown in CLI 13 to what is shown in CLI 14.

```
#!/usr/bin/python
```

CLI 13: Hashbang pointing to system Python runtime

```
#!/usr/bin/python2
```

CLI 14: Hashbang pointing to Python 2 runtime

Alternatively, it is possible to store convert_encoding.py in the same directory as the files, go to that directory and just type the command shown in CLI 15.

```
$ python2 convert_encoding.py [options] file1 file2 ...
```

CLI 15: Force python2 runtime when executing script

Check the documentation of the script for further information regarding the possible arguments in [options]. For the data used in this thesis, i.e. the NTC7 PSD, the bold options in CLI 16 were used for the conversion.

```
$ convert_encoding.py -f euc_jp -t utf_8 -r -o #.utf8 input_file(s)
```

CLI 16: Arguments used for converting NTC7 PSD to UTF-8

This tells the script to assume EUC-JP (-f or --from=) as the source encoding and tells it to convert the files to UTF-8 (-t or --to=), while also going into the subdirectories to convert the files (-r or --recursive) and adding a ".utf8" to the newly converted files (-o or --out=).

Since there are over 50,000 files in all the directories, selecting each file individually would be very time consuming, but thanks to the recursive argument, it is possible to simply pass the wildcard "*.txt" or the individual folder name(s) as files. The script will then convert

[110] https://github.com/goerz/convert_encoding.py (accessed on 28.03.2020)
[111] Type "echo $PATH" to see all the directories in PATH
[112] This line indicates which program to run the script with

all files within the current working directory or indicated folder(s) respectively. For a more fine-grained selection of files, it is possible to use the Linux *find* command with the -exec argument. For the thesis, Script 1 was written and used to find all relevant files in /path/ and convert them:

```
#!/bin/bash
find /path/ -type f -name "*.txt" -exec \
convert_encoding.py -f euc_jp -t utf_8 -r -o \#.utf8 {} +
```

Script 1: File selection through find command

With all data converted to UTF-8, the next step will be preparing the training (*train*), development or validation (*val*) and test (*test*) files for each language.

7.4.1.2 Separate languages and store the sentences in line-aligned files

In this sub-section, the original "train.txt", "dev1.txt" and "dev3.txt"[113] will be used to create "train.jp" and" train.en", "val.jp" and "val.en" and "test.jp" and "test.en" respectively. The easiest way to automate this, is by creating a Python script, that separates each line at a predetermined marker (in this PSD's case the triple pipe symbol "|||") and stores them in different output files. The following Script 2 was written and used in the thesis to create one output file for each of the columns in the original file (passed as the first argument to the script) and append the extension defined in "langs" to each output.

```
#!/usr/bin/python
import sys
with open(sys.argv[1], encoding="utf8") as f:
      columns = zip(*(l.split("|||") for l in f))

langs = ('SSR', 'DOCID', 'TID', 'jp', 'en')
for lang, data in zip(langs, columns):
       with open('output.' + lang, 'w', encoding='utf8') as f:
             f.writelines(line.strip("\n") + '\n' for line in data)
```

Script 2: Split text into separate output files

Since the alignment is maintained in the output files, it is now possible to simply rename "output.jp" and "output.en" to the file needed, i.e. "train.jp" and "train.en" if "train.txt" was the input file. The ID files can subsequently be deleted as they will not be needed. This was done separately for each file, in order to keep the whole process controlled and the script as simple

[113] dev2.txt will not be used at this point but will be used later for the creation of the domain-specific test data.

as it is. “dev1.txt” was used to create “val.jp” and “val.en”, while “dev3.txt” was used to create “test.jp” and “test.en”.

With this, the full NTC7 PSD would already be ready for deployment in NMT training, but since the original text files are not tokenized, it is highly advised to tokenize them before continuing. In the next sub-section, a step-for-step guide for tokenizing the files will be presented.

7.4.1.3 Tokenization of the files

As written in Section 7.3, tokenization is essential for Asian texts, but can also be very helpful for Latin text. For this project, the Japanese files were tokenized with *MeCab*[114] and the English files with the *Moses Tokenizer* Perl-script[115].

Preparing the *Moses Tokenizer* is fairly simple: simply copy the code provided on the git-hub site in a file named “tokenizer.perl”, save it in a location of your choosing and make it executable by typing

```
$ chmod +x tokenizer.perl
```

CLI 17: Make script executable

Installing MeCab can be a little more involved. On Manjaro it is only available via the AUR repository[116]. Installing MeCab and its dependencies therefore also requires a different procedure than regular packages. We will be using *pamac*, one of the many AUR helpers, to assist in the manual build process and to install all the needed packages, simply by following the commands shown in Table 10.

$ pamac search -a mecab	Searches for the package on AUR
$ pamac build mecab	Installs the most recent MeCab version
$ pamac build mecab-ipadic	Installs the IPA dictionary for MeCab

Table 10: Commands for installing mecab from AUR

Before the packages are built and installed, it is necessary to answer two questions and enter your password (see CLI 18, bold letters represent user input).

```
Edit build files ? [y/N] n
Apply transaction ? [y/N] y
```

CLI 18: Questions when installing from AUR

[114] https://taku910.github.io/mecab/ (accessed on 20.04.2020)
[115] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl (accessed on 20.04.2020)
[116] The Arch User Repository (AUR) is a community-driven repository for Arch users.

Unfortunately, packages on AUR can be outdated or just simply not working, so it is possible that the installation will fail. Troubleshooting would be too much to cover for this thesis. Luckily, it worked well for MeCab and most major dependencies[117].

With both tokenizers prepared, the files can be tokenized. The command used for the English text is shown in CLI 19 (change only the bold arguments; type "-h" after tokenizer.perl to see all possible arguments):

```
$ perl tokenizer.perl -a -no-escape -l en < inputfile.en > outputfile.en.tk
```

CLI 19: Command used to tokenize English text files

For the Japanese text, MeCab was called with the arguments shown in CLI 20.

```
$ mecab -O wakati -o outputfile.jp.tk inputfile.jp
```

CLI 20: Command used to tokenize Japanese text files

In order to speed up the process, Script 3 was written and used for this thesis ("#" represent comments; bold characters are variables to be changed for each different dataset):

```
#!/bin/bash
# Select working Folder, Dataset, Variant and Languages here
workdir="$(pwd)" # "$(pwd)" stands for current working directory; change if you want to indicate specific directory
dataset="ntc7"
variant=""
src="jp"
tgt="en"

# English / Latin language tokenization
for l in en; do for f in data/$dataset$variant/*.$l; \
do perl tools/tokenizer.perl -a -no-escape -l $l -q < $f > $f.tk; done; done

# Japanese tokenization
for l in jp; do for f in data/$dataset$variant/*.$l; \
do mecab -O wakati < $f > -o $f.tk; done; done
```

Script 3: Tokenization script EN-JP

The resulting filenames for the full NTC7 PSD after the tokenization were therefore: train.en.tk, train.jp.tk, val.en.tk, val.jp.tk, test.en.tk and test.jp.tk.

[117] At the time of writing, the neologism dictionary would not install via AUR; if it is required, it can be installed following the guide here: https://github.com/neologd/mecab-ipadic-neologd (accessed on 20.04.2020)

7.4.1.4 Creating the vocabulary files

With the files tokenized, all that remains is to create the two vocabulary files from the English and Japanese "train"-files. In the case of OpenNMT-tf, the base commands shown in CLI 21 were used in a slightly modified way to create the vocabulary files:

```
onmt-build-vocab --size 50000 --save_vocab train.en.tk vocab.en.tk
onmt-build-vocab --size 50000 --save_vocab train.jp.tk vocab.jp.tk
```

CLI 21: Commands for creating the vocabulary files with OpenNMT-tf

With this, two vocabulary files ("vocab.en.tk" and "vocab.jp.tk") are created from the corresponding "train" files while limiting the size of the vocabulary to 50,000 + 1 (the "unk") token. As will be further elaborated later, mixed-precision training was used in this thesis, so the additional argument "--size_multiple 8" was specified. This automatically increases the vocabulary limit to 50,007 + 1, as FP16 calculations need a vocabulary divisible by 8 for optimal performance. [118] Mixed precision training is only advantageous on GPUs that perform tensor operations in dedicated hardware, like the Nvidia Volta and Turing architectures do.

With this, the full NTC7 PSD dataset, that contains all domains and all 1.8 million parallel sentences is prepared and ready for training. The final files are: train.en.tk, train.jp.tk, val.en.tk, val.jp.tk, test.en.tk, test.jp.tk, vocab.en.tk and vocab.jp.tk.

In the next section, I will demonstrate how a specific domain was extracted from the full dataset and how five different sets of data were prepared for training five different neural translation models.

7.4.1.5 Creating the domain-controlled training data

For creating the domain-controlled data, the starting point was the original, UTF-8 converted "train.txt.utf8" file. Only the "train" file was modified for creating the domain-specific and shortened datasets.

First a broad domain was chosen according to the International Patent Classification[119]. For this experiment, *optics* was chosen as the domain, so the four major optics classifiers that appeared in most optics related patents that I had previously translated were selected. These were "G01", "G02", "G03" and "G06".

The idea here was to find the domain of a specific sentence by looking at the original patent documents that are linked to the individual training sentences in "train.txt" via the

[118] https://opennmt.net/OpenNMT-tf/training.html?highlight=mixed%20precision (accessed on 14.03.2020)
[119] https://www.wipo.int/classifications/ipc/ipcpub/ (accessed on 15.04.2020)

DOCID in “training-ids.txt”. The lines in “training-ids.txt” refer to an absolute path in the folder structure of the NTCIR 10 Test Collection, leading to the original patent document(s).
For making the code simpler, this process was divided into three separate steps.

1) **Step 1:**

First, a script that opens and searches each and every patent document of the collection and exports the absolute path of only those files that contained one of the aforementioned optics-classifiers had to be created. For this purpose, the following Script 4 was written in Python and used for this thesis.

```
#!/usr/bin/python
import glob
#Define Domains
domains = ('G01', 'G02', 'G03','G06')
#Search for Domains and extract absolute path of file to outputfile
outputfile = "step1out.txt"

with open(outputfile,'w') as f:
    for filename in glob.iglob('./' + '**/*.TXT', recursive=True):
        if any(x in open(filename, encoding='utf-8').read() for x in domains):
            print (filename, file=f)
```

Script 4: Extract absolute path of files that contain domain classifier

This writes all the absolute paths of files containing the optics classifiers in (almost) the same format as found in the “training-ids.txt” to the file specified under “outputfile” (“step1out.txt”). The script takes quite a while to run, as it needs to open each individual file.

2) **Step 2:**

The next step was to have a script create another file that contains the DOCIDs of the individual “train.txt” sentences that align with the documents listed in the output file from Step 1. Additionally, the “./” in the beginning of the extracted absolute paths from Step 1 had to be removed, as “./” is not written in the paths in “training-ids.txt”. Script 5 outputs a file containing

all lines which match the paths listed in the output file from Step 1. The output filename is defined under “outputfile” (“step2out.txt”):

```
#!/usr/bin/python
#Pathlist to find ID
paths = [line.rstrip("\n") for line in open("step1out.txt")]
#Remove "./" from paths
spath = [s.strip("./") for s in paths]

outputfile = "step2out.txt"
with open(outputfile,'w') as f:
        for line in open("other/ntc8-patmt-train/ntc7/train/training-ids.txt"):
                if any(x in line for x in spath):
                        print(line, end='', file=f)
```

Script 5: Create id-list from “training-ids.txt” and the step1 output

However, since the output contains the whole line, everything but the DOCID must be removed from each line. Step 2 therefore actually uses a second script, Script 6, to remove the unnecessary information and keep one DOCID per line while outputting all of it as the file defined in “outputfile” (“idlist.txt”):

```
#!/usr/bin/python
#Keep only DOCIDs before first space
l = []
outputfile = "idlist.txt"
with open(outputfile,'w') as f:
  for line in open("IDs-Optics.txt"):
    if line.strip():
      l.append(line.split()[0][1:])
  l = '\n'.join(l)
  print(l, file=f)
```

Script 6: Only keep DOCID before first space on each line

3) Step 3:

The final step is to write a new file, which only contains the sentences that correspond to optics patents. The file will be created from the “train.txt.utf8” document prepared in Section 7.4.1.1. The list generated in step 2 (“idlist.txt”) was used to identify which sentences to extract from “train.txt.utf8”. Originally another Python script was used to create the file, but it turned out to be too slow. Luckily, Linux offers a great option for performing this operation with the search

tool *grep*. The command used for finding each line in "train.txt.utf8", that matches one of the DOCIDs in "idlist.txt", is shown in CLI 22.

```
$ grep -f idlist.txt train.txt.utf8
```

CLI 22: Find patterns from file in another file using grep

To output the result to a file named "trainoptics.txt.utf8", the bold command in CLI 23 was added:

```
$ grep -f idlist.txt train.txt.utf8 > trainoptics.txt.utf8
```

CLI 23: Save grep standard output to file

With this, the optics training file is created, resulting in a document containing 684,693 lines. In order to use the data, the preparation steps mentioned in Section 7.4.1.2 through to 7.4.1.4 were applied to the optics training file (trainoptics.txt.utf8) as well, meaning that vocabularies for the optics dataset were created separately, on the basis of the optics training file.

7.4.1.6 Additional training data

In order to create comparable models, three further datasets were created:

1) The full NTC7 dataset, shortened to the same line count as the optics data, by randomly deleting 1,113,878 lines.
2) A dataset, where specifically all the optics patent sentences were removed (line count: 1,113,878).
3) The same dataset as in 2) but shortened to optics data line count by randomly deleting 429,186 lines.

The random deletion script used for 1) and 3) is presented in Script 7. The script outputs a list of lines (amount specified under "number") between 1 and the number of lines in the file specified under "filename" (inputfile.txt) as a file named "delete.lines":

```
#!/bin/bash
filename=inputfile.txt
number=429186

line_count="$(wc -l < "$filename")"
line_nums_to_delete="$(shuf -i "1-$line_count" -n $number)"

printf '%d\n' $line_nums_to_delete > delete.lines
```

Script 7: Create list to randomly delete lines

Then, to actually create a file without said lines the command shown in CLI 24 was used.

```
awk 'FNR == NR { h[$1]; next } !(FNR in h)' delete.lines inputfile.txt \
 > outputfile.txt
```

CLI 24: Command to exclude lines specified in delete.lines and write output to separate file

In this way, both shortened versions of the full NTC7 PSD and the NTC7 PSD without optics data could be created. CLI 25 shows how the NTC7 PSD without optics data announced in 2) was generated by simply inverting the pattern-matching (-v) of the *grep* tool as presented in Step 3 of Section 7.4.1.5:

```
$ grep -v -f idlist.txt train.txt.utf8 > trainwithoutoptics.txt.utf8
```

CLI 25: Inverse pattern match with grep -v to exclude optics sentences

Like with the optics data, the resulting training data would go through the preparation described in Section 7.4.1.2 through to 7.4.1.4, resulting in individually created vocabularies for each dataset.

7.5 Training (OpenNMT-tf)

With all the data prepared, training was started mainly following the official QuickStart guide for OpenNMT-tf. Since the GPU used for this thesis has a relatively small amount of video memory, some tweaking of hyperparameters had to be made, before the training could properly function. In general, the command for starting training with a Transformer architecture is the one listed in CLI 26:

```
onmt-main --model_type Transformer --config data.yml --auto_config \
train --with_eval
```

CLI 26: Default training command for OpenNMT-tf

For this thesis, the following, slightly modified command was used:

```
onmt-main --model_type Transformer --config ntc7/datatransmixedJPEN.yml \
config/transformer.yml --auto_config \
--gpu_allow_growth --mixed_precision train --with_eval
```

CLI 27: Training command used for this thesis

The bold arguments represent the modified parts of the command. Starting from the end, "--mixed_precision" enables the aforementioned FP16 optimization for higher training performance on GPUs with hardware-support for tensor calculations. The "--gpu_allow_growth" argument is necessary for the training to start, seemingly because of a bug

with memory allocation in the TensorFlow Framework, on the newer Nvidia RTX-Series GPUs. As can be noticed, it is possible to add multiple configuration files to the command, which was used to streamline the training of the multiple models. Luckily, OpenNMT-tf offers the "--auto_config" argument, so only the parameters of the model that are of interest to us and the location of the data have to be adjusted manually in the config files.

In the next sub-section, I will present the configuration files used for training the full NTC7 PSD model. The other models had the training data location adjusted accordingly.

7.5.1 Configuration and hyperparameters

Aside from the "--auto_config" parameters defined by the OpenNMT-tf toolkit for the base Transformer model [120], the following parameters were defined in the "transformer.yml" configuration file:

```
train:
  batch_size: 1024
  effective_batch_size: 4096
  save_checkpoint_steps: 10000
  average_last_checkpoints: 5
  keep_checkpoint_max: 10
  max_step: 150000
  valid_steps: 10000
  warmup_steps: 8000
  report_every: 100

eval:
  batch_size: 32
  steps: 10000
  save_eval_predictions: true
  external_evaluators: bleu
#  early_stopping:
#    metric: bleu
#    min_improvement: 0.01
#    steps: 4
```

Script 8: Transformer configuration used for this thesis

The **batch_size: *n*** parameter defines the maximum number (*n*) of tokens provided to the GPU at one time. The amount of 1,024 tokens was empirically found to be the maximum amount the

[120] Find the model catalog here: https://github.com/OpenNMT/OpenNMT-tf/blob/master/opennmt/models/catalog.py (accessed on 14.01.2020)

GPU could hold without running into critical OOM errors. A higher batch size generally means faster training, as more data can be calculated in parallel.

The **effective_batch_size: *n*** parameter defines the effective number of tokens to consider for one gradient update. Essentially, it multiplies the results of individual, smaller batch size in order to get the same result as if the GPU was able to look at the defined effective batch size in one pass. In this thesis' case, the effective batch size of 4,096 was chosen, which means that the GPU needs to run 4 passes to get to the effective batch sizes. While this slows training considerably, it was found that the networks perform noticeably worse with smaller batch sizes, so a higher effective batch size is highly recommended (Popel & Bojar 2018:12).

The **save_checkpoints_steps: *n*** parameter should be self-explanatory. It tells the toolkit to save a model checkpoint after *n* steps. One step is finished, when the effective batch size *n* has been calculated. Depending on training time, setting this to 5,000 or 10,000 is a good value.

The **average_last_checkpoints: *n*** parameter allows the toolkit to automatically average the results of the last *n* checkpoints saved during training, which usually improves model accuracy[121].

The **keep_checkpoint_max: *n*** tells the toolkit when to start discarding old checkpoints. Model checkpoints can be several gigabytes in size, so saving space by deleting the oldest checkpoints seems sensible.

The **max_step: *n*** and **valid_step: *n*** define the maximum training steps (n = 150,000 ~ 18.5h of training in this thesis) and when to perform validation of the models.

The **warmup_step: *n*** parameter defines when the *learning rate* of the network changes from linear decay to inverse square root decay. The default as found in the TransformerBase configuration of OpenNMT-py was used (n=8,000), in accordance with recommendations found in Popel & Bojar (2018).

The **report_every: *n*** parameter tells the toolkit to report training progress every *n* steps. It reports details like *speed* (steps/s and tokens/s), *learning rate* and *training loss*.

Under **eval:** several parameters may be set, to enable the toolkit to automatically evaluate the model after **steps: *n*** with the evaluation metric chosen under **external_evaluators: *x***. This allows for early stopping, once the result no longer improves. Notice the early stopping was commented out, as in this thesis using a fixed training time was also very interesting for comparison. See [122] for more details about these parameters.

[121] https://opennmt.net/OpenNMT-tf/inference.html?highlight=average (accessed on 18.12.2019)
[122] https://opennmt.net/OpenNMT-tf/training.html?highlight=external_evaluators (accessed on 18.12.2019)

The data configuration file ("datatransmixedJPEN.yml") is much more self-explanatory:

```
model_dir: /home/chris/NMT/openNMT-tf2.8.1/modelsfp16/ntc7-jpen/

data:
  train_features_file: /home/chris/NMT/data/ntc7/train.jp.tk
  train_labels_file: /home/chris/NMT/data/ntc7/train.en.tk
  eval_features_file: /home/chris/NMT/data/ntc7/val.jp.tk
  eval_labels_file: /home/chris/NMT/data/ntc7/val.en.tk
  source_vocabulary: /home/chris/NMT/data/ntc7/onmt-tf/vocab.jp.tk
  target_vocabulary: /home/chris/NMT/data/ntc7/onmt-tf/vocab.en.tk
```

Script 9: Data configuration file example used for this thesis

The **model_dir** is the directory where the model progress and all the checkpoints are saved, whereas the paths under **data:** lead to the different training, validation and vocabulary files. Notice the ST is identified as "features" and the TT is identified as "labels". So, the model is trained in one direction, in this case being JP→EN.

To summarize, as shown in Table 11, five Transformer NMT models were trained, with a separate model saved after the training step where the highest model performance was expected (see Section7.3).

Table 11: Trained model description, names and training time

Dataset	**Model name**	**Training time**
Full (~1,8m sentences) JP→EN	ntc7-jpen	150k steps: ~18.5h 120k steps: ~14.7h
Small (684,693 sentences) JP→EN	ntc768k-jpen	150k steps: ~18.5h 80k steps: ~9.8h
Optics (684,693 sentences) JP→EN	ntc7o-jpen	150k steps: ~18.5h 80k steps: ~9.8h
Without Optics (~1,1m sentences) JP→EN	ntc7wo-jpen	150k steps: ~18.5h n/a[123]
Small Without Optics (684,693 s.) JP→EN	ntc7wo68k-jpen	150k steps: ~18.5h 90k steps: ~11h

[123] Model metrics were best at 150k steps

7.5.2 Monitoring training

While it is not possible (or feasible) to see and understand every single calculation occurring in all the nodes of the neural network during training, it is possible to visualize some of the data through the tensorboard integration of OpenNMT-tf. The command to run tensorboard is:

```
$ tensorboard --logdir "path/to/model_dir"
```

CLI 28: Monitor models in „model_dir" with tensorboard

This will serve a website to http://localhost:6006/, on which it is possible to see training data visualized in easy to understand graphs. With this it is possible to get an idea of model performance even before proper external evaluation and to see when the model is likely to perform best. In order to display these graphs, tensorboard visualizes the scores calculated during the evaluation process in training; adding external evaluators will therefore also result in additional graphs reported on tensorboard.

Before looking at the results, some of the metrics have to be explained, so that it is clear what is being measured.

7.5.2.1 The BLEU metric

The term BLEU stands for *Bilingual Evaluation Understudy* and was used several times during this thesis. It was shortly introduced as the de-facto standard metric for automatic translation quality evaluation, but this is a hotly debated definition in and of itself. The BLEU score was proposed by IBM researchers Kishore Papineni, et al. in their 2001 paper *BLEU: a Method for Automatic Evaluation of Machine Translation* (Papineni et al. 2001). It is widely used in most publications regarding machine translation as a metric to compare translation output of different machine translation architectures and models. The main rationale behind the metric is: "The closer a machine translation is to a professional human translation, the better it is." (Papineni et al. 2001:1). But how does it determine "the translation closeness" and express it as a score?

BLEU works by comparing a generated sentence, like a translation hypothesis, to one or many reference sentences (or translations). What is measured is the so-called *precision* of the match, essentially expressing how many of the reference sentences' features the generated sentence matches. A perfect match, i.e. all the same features were used, would result in a 1, while a perfect-mismatch, i.e. no feature from the reference sentences was used, would result in a 0[124]. The main idea is that a good machine translation will have a surface form very similar to a professional human translation. Note, however, that getting a score of 1 is almost

[124] Note that most publications, as does this thesis, multiply the result by 100 for better readability. A BLEU score of 0.353 would therefore read as 35.3.

impossible, as that would mean the candidate sentence is the same as the reference-sentence(s). As translators well know, it is unlikely that even human translations ever match perfectly and that there are many ways to correctly translate one specific sentence.

One big advantage is that this way of evaluation is language agnostic, as BLEU only checks the surface form of texts, i.e. the tokens that make up a sentence (on a lower level, the bytes of the characters in the sequence for the computer[125]). The system essentially analyzes *n-grams* in the sentences, comparing tokens in the candidate sentence (the generated sentence) to the tokens appearing in the reference sentences. The n-gram matching is *position-independent,* so word order is NOT considered. The authors of the paper argue that this still tends to cover the *adequacy* of a text. Longer n-grams (bigrams, trigrams) are used to account for *fluency* by looking at phrase-matches, somewhat accounting for word-order. Since unigram or 1-gram matches are more likely than bi- or trigram matches, the BLEU metric also gives higher scores to sequential matching words. That is, if a string of 3 or 4 words (i.e. a 3- or a 4-gram) in the MT translation matches the human reference translation, it will have more of a positive impact on the BLEU score than a string of two matching words or a single matching word. However, this also means that an accurate translation will receive a lower score if it uses synonyms, or matching words in a different word order.

The BLEU algorithm addresses some common problems of MT, like the "overgeneration" of words, by penalizing the precision score if a matching n-gram is generated more often than in it appears in the reference sentence (see Text example 7.5-1 taken from Papineni et al. 2001:2), through the so-called *modified n-gram precision.*

Candidate: the the the the the the the.
Reference 1: The cat is on the mat.

Text example 7.5-1: Overgeneration by MT systems

Sentence length (or brevity) is also penalized in order to not inflate scores for shorter sentences that are made up of only reference sentences features (because only a part of the sentence is predicted).

It is clear, that the BLEU metric is a very simple algorithm that has no way to examine the deeper semantics of a sentence. So, a perfectly adequate translation might score poorly, because it uses less or other words than the reference translation. In fact, the score is highly dependent on the reference translations and can easily be artificially increased by simply providing more reference translations to the system.

[125] This is also why it is important to use a standardized code-page for all the text in a project.

BLEU also only calculates its scores on individual sentences, yet, the score is usually reported for a larger corpus of (generally independent) sentences, by averaging the individual scores of each sentence over the whole corpus. "Quantity leads to quality" is the mantra followed by the authors of the paper (Papineni et al. 2001:8), which is why generally corpora of 1000 or more individual sentences are used for the BLEU scoring. It is safe to assume, that since MT systems usually only work on a sentence level, sentence-level evaluation was accepted as a safe measure of quality.

While there are more issues apparent from a translator's point of view, it should already be abundantly clear, that the algorithm has no way of really assessing the quality of a translation. In fact, it was first created "as an inexpensive automatic evaluation that is quick, language-independent and correlates highly with human evaluation" in order "to monitor the effect of daily changes to their systems [MT/NLP systems] in order to weed out bad ideas from good ideas." (Papineni et al. 2001:1). It was therefore designed to compare similar models of the same MT-architecture, for incremental, global changes and not as a universal translation quality metric. While this may make it usable for the closed nature of this thesis' experiment and in fact MT development in general, readers and especially researchers should not take the BLEU score as an absolute indication for translation quality.

7.5.2.2 Loss

Most NMT toolkits provide a *loss* value during training, as this is the metric used in standard autoregressive NMT models to tune the individual weights and find the most probable translation of a word in a certain position (essentially the error, that is then backpropagated). It is similar to BLEU in that it expresses the precision of a specific prediction by the network compared to the reference text (i.e. the TT/TL data used in training), but instead of providing a score on a sentence-level, the sentence-level log-likelihood is decomposed as a sum over word/token-level log-likelihoods. The training of most NMT models is hence optimized to finding the next perfect output token given the previous perfect output token.

There are many ways to calculate the loss, but a common approach is the cross-entropy loss function shown in Equation 13[126].

[126] Function taken from https://towardsdatascience.com/neural-machine-translation-15ecf6b0b (accessed on 05.04.2020)

$$-\sum_{w=1}^{|S|}\sum_{e=1}^{|V|} y_{w,e}\log(\hat{y}_{w,e})$$

|S| = Length of Sentence
|V| = Length of Vocabulary
$\hat{y}_{w,e}$ = predicted probability of vocab entry e on word w
$y_{w,e}$ = 1 when the vocabulary entry is the correct word
$y_{w,e}$ = 0 when the vocabulary entry is not the correct word

Equation 13: Cross-entropy loss function

In essence, the closer the model gets to giving a probability of 100% to the correct word in the vocabulary at the point where it appears in the reference sentence, the lower the loss value will be. Likewise, the loss value increases exponentially, the more unlikely the model classifies the correct word at that specific point. This value is then commonly summed up for the whole sentence resulting in the full loss metric of the sentence. It may however also be summed up for all the tokens in a batch (i.e. multiple sentences). Lower loss values therefore typically correspond to better translations and the training of neural networks aims to reduce the loss of the predictions. Depending on the toolkits used, the loss reported may be obtained differently. OpenNMT-tf reports the cross entropy that is computed at the current training step (i.e. the total loss for all tokens in a batch) and can also report a separate loss value for predictions on the validation (val) dataset.[127]

7.5.2.3 Perplexity

Another value often reported by NMT toolkits and in fact machine translation or NLP tools in general is the *perplexity* metric. The perplexity is a measurement of how well a probability model predicts a sample. So, in the context of NLP, perplexity is one way to evaluate language models, and is closely related to the cross entropy or loss function described above. In fact, the perplexity reported by OpenNMT-tf is simply the exponential function of the loss value, $perplexity = exp\ (loss)$ [128]. Perplexity therefore gives us a linear value to how many average choices the network would have for a prediction (i.e. how "unsure" the network is). A lower perplexity generally points to a more accurate translation model. OpenNMT-tf only reports perplexity for predictions against the validation dataset, not for the training data.

[127] https://github.com/OpenNMT/OpenNMT-tf/issues/50#issuecomment-358582413 (accessed on 05.04.2020)
[128] https://forum.opennmt.net/t/perplexity-in-opennmt-tf/2290/2

7.5.3 Training observations

As stated in Section 7.5.2, *tensorboard* is a great tool for monitoring training progress. It will present colored graphs for each model and allow for great comparability (unfortunately colors cannot be customized and are somewhat ugly). Table 12 shows the color legend used in following graphs and a short summary of the models.

Table 12: Graph color legend and model summary

Model	Dataset used	Training loss at 150k steps
ntc7	Full (~1,8m sentences) JP→EN	2.165
ntc768k	Small (684,693 sentences) JP→EN	2.034
ntc7o	Optics (684,693 sentences) JP→EN	2.025
ntc7wo	Without Optics (~1,1m sentences) JP→EN	2.089
ntc7wo68k	Small Without Optics (684,693 sentences) JP→EN	1.997

Generally training for a longer amount of time will improve model accuracy for the training data, i.e. the loss function will continue to fall, and the model is more likely to predict the same sentences as the reference sentences. This can also be clearly seen in Figure 30, where all models yield a progressively lower loss value. Notice how the model based on the largest and most varied dataset (ntc7) has the highest loss value, while the optics dataset (ntc7o) has the lowest.

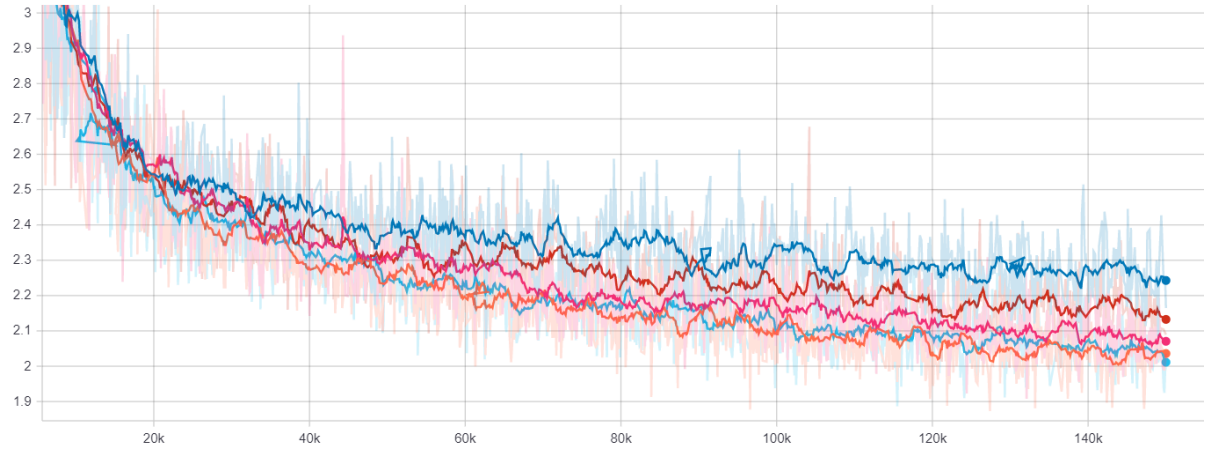


Figure 30: Training loss[129]

[129] Graph smoothed by 0.9 for better readability; desaturated lines show actual value, nicely showing how the gradient descent works.

What we are seeing on the smaller models, is an effect that may result in the so-called *overfitting* of a model, which means that it would generalize less well on hitherto unseen text data. This is why a separate set of validation files is created for training. As a reminder, the "val.en" and "val.jp" were created from the "dev1.txt" and therefore contain completely different sentences than the "train" files. This enables evaluation of the model on hitherto unseen data during training.

In Table 13 and Figure 31 it is possible to observe how, during the limited training time of 150.000 steps per model (~18.5h per model), the previously described effect of ***overfitting*** occurred only in the smaller datasets, like the "Optics" dataset (ntc7o), the "Small Without Optics" dataset (ntc7wo68k) and the "Small" dataset (ntc768k). We can see how the *loss* and *perplexity* continuously increases for these datasets after their lowest point at 80k steps. The BLEU score also slightly reflects this change in model accuracy. Generally, it is recommended to stop training once the *loss* and *perplexity* values start rising continuously during validation or when the BLEU score stops improving for several validations.

Table 13: Effects on model accuracy / BLEU score on the validation data

Model	BLEU / Loss / Perplexity at 80k Steps	BLEU / Loss / Perplexity at 150k Steps [130]
ntc7	39.09 / 1.211 / 3.421	39.35 / 1.207 / 3.344
ntc768k	38.13 / 1.329 / 3.754	38.15 / 1.391 / 4.019
ntc7o	36.96 / 1.446 / 4.202	36.38 / 1.524 / 4.593
ntc7wo	38.02 / 1.279 / 3.638	37.26 / 1.305 / 3.688
ntc7wo68k	37.78 /1.370 / 3.908	37.63 / 1.444 / 4.241

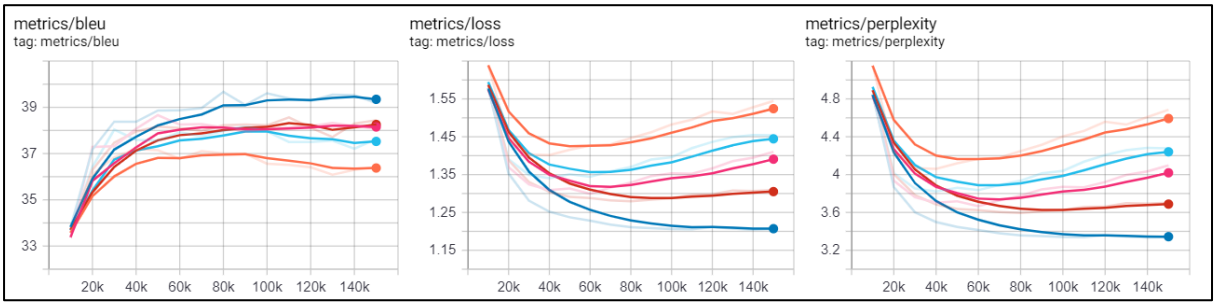


Figure 31: Validation BLEU, loss and perplexity in tensorboard

The rise in perplexity and loss, however, only reflects unfavorably on the BLEU score of the two smaller and domain filtered datasets (ntc7o and ntc7wo68k), while it shows no variation in

[130] Values smoothed with a ratio of 0.6

the larger datasets (ntc7 and ntc7wo) or the "Small" dataset without domain filtering (ntc768k). Interestingly, the "Small" dataset (ntc768k) and the "Without Optics" datasets (ntc7wo) perform almost the same in the BLEU metric, even though the latter corpus has 38% more data to work with. The tendency of overfitting by the smaller datasets does clearly reflect in the loss/perplexity metric. These observations confirm that more data does indeed seem to result in more generalized models, while it also shows that higher variety in the training data leads to lower model perplexity on unseen sentences of the same domains. How this explicitly influences the final output is, however, not easily visible.

It is likely, that the model based on the "Optics" data (ntc7o) performs the worst in BLEU, because it is "specialized" (or overfitted) on the optics domain. It is therefore not as adept at predicting the validation sentences' translations (or the exact words used), because the validation reference files contain sentences and words from all other domains as well. This would also explain why the model based on the more varied "Without Optics" dataset, both full-size and shortened (ntc7wo and ntc7wo68k), consistently performs better than the "Optics" model (ntc7o) and why the "Small" unfiltered dataset (ntc768k) performs about the same as the much larger "Without Optics" dataset (ntc7wo).

In the next sub-section, the different models will be tested on the mixed domain patent test file created in Section 7.4, to validate the BLEU and loss metric results seen during training with the validation data. Additionally, the models will be compared on domain-specific texts (i.e. sentences from optics patents) to find out whether the optics model can provide better scores and translations for domain-specific test sentences, or if neural networks generally benefit more from larger or more varied training corpora. The test should also help us see, whether overfitting can prove useful for domain specialization when there is not enough data available. Both test corpora will be evaluated with BLEU and later through random sampled human evaluation by the author.

7.6 Translating with the trained models and evaluation

With all the models trained, it is possible to translate single sentences or full documents[131] using the translation models for inference. The command used for creating the translation hypotheses of a specific source text file with OpenNMT-tf is listed in CLI 29.

[131] However, translation will only occur on a sentence level!

```
$ onmt-main --config /path/to/dataconfigofmodel --auto_config \
--gpu_allow_growth --checkpoint /path/to/modelcheckpoint infer \
--features_file /path/to/sourcetextfile > /path/to/outputfile
```

CLI 29: Translate sentences with OpenNMT-tf

Since a Japanese to English model was trained, Japanese source texts will be used:
The **first source text** will be the "test.jp.tk" file created in Section 7.4, which contains 899 lines.

The **second source text** will be a domain-specific test-file "testdom.jp.tk" created from the "dev1.txt", "dev2.txt" and "dev3.txt" files of the *NTCIR 10 PatentMT Test Collection* using the same method as presented in Sub-section 7.4.1.5 for creating the domain-controlled training data. The file was shortened to 899 lines by randomly deleting 175 lines. Note, that including the "validation" set (made from "dev1.txt") is not ideal, but since it is not used for training the network per se, the sentences are still essentially "unseen" by the model.[132]

For each source text the corresponding reference text in English was also created as described above, resulting in the two files "test.en.tk" and "testdom.en.tk". All the files were tokenized as described in Sub-Section 7.4.1.3 and have one sentence per line.

In order to see whether the overfitting of the model adversely affects the prediction ability of the models, the translation tests were also run with an earlier checkpoint by passing the "--checkpoint" argument. The checkpoint was chosen in accordance with the highest BLEU score for each model during training validation.

Each output file contains only one translation prediction per line, which is the prediction that has the highest probability found by the network. Also, the BLEU evaluation compares each prediction to only one reference sentence. For the human evaluation, eight (8) total sentences from both source-texts (4 each) are randomly selected and evaluated for each model and prediction separately. Texts are not detokenized before evaluation[133].

7.6.1 BLEU evaluation

To run the BLEU evaluation after training, it must be run through an external command or script. For this, the "multi-bleu perl-script" provided by the Moses SMT toolkit was used.[134] The script is also provided in the package of the openNMT-py toolkit.

Using the external BLEU script is fairly easy and was done by typing the command shown in CLI 30:

```
$ perl multi-bleu.perl reference.txt < predictions.txt
```

[132] Since otherwise not enough data would have been available for creating the domain-specific test corpus, dev1.txt was included, but again, ideally only data not used in training or validation are used.
[133] Tokenization artifacts will be annotated.
[134] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

Table 14 shows the BLEU scores of all models at 150k training steps for the predictions of both the mixed domain and the domain specific source texts (test.en.tk and testdom.en.tk).

Table 14: BLEU scores for the models trained for 150k Steps

Model	BLEU for mixed domain	BLEU for optics domain
ntc7	40.29	37.57
ntc768k	38.63	35.69
ntc7o	36.95	35.45
ntc7wo	39.85	35.68
ntc7wo68k	38.52	34.51

As expected, the biggest and unfiltered dataset (ntc7) performed the best of all models for the mixed domain test with 40.29 BLEU. Interestingly, it did also perform best in the optics domain test, something that will be interesting to verify in the human evaluation. The smaller unfiltered dataset (ntc768k) performed only about as well as the dataset of the same size without optics sentences (ntc7wo68k) in the mixed domain test, but outperformed it in the optics domain test by a margin of 1.18 BLEUs, performing as well as the bigger dataset without optics (ntc7wo) and the optics dataset.

However, another interesting note is, that the optics model (ntc7o) only lost 1.5 BLEU in the optics domain test, while all other models' average loss was 3.46 BLEU, with the highest loss seen on the ntc7wo model with a loss of 4.17 BLEU. It would be interesting to see, if an optics model with the same amount of data as the full ntc7 model would perform similarly to it or even better in the optics test.

Table 15 now shows us what happens when training is stopped early, in order to prevent overfitting. Each model was stopped at a different time, where the BLEU evaluation on validation data was most favorable. The steps at which the training was stopped are listed next to the model in brackets.

Table 15: BLEU scores for trained at their best BLEU score

Model	BLEU for mixed domain	BLEU for optics domain
ntc7 (120k steps)	40.42	37.80
ntc768k (80k steps)	38.73	36.24
ntc7o (80k steps)	37.84	36.12
ntc7wo (150k steps)[135]	39.85	35.68
ntc7wo68k (90k steps)	38.81	34.94

It can clearly be seen, that all models perform better in the BLEU metric, when training is stopped early and therefore before running into overfitting. While the models based on larger datasets expectedly show lesser variation in the BLEU score, we can see that the smaller models (ntc768k, ntc7o, ntc7wo68k) are more affected. Interestingly the gains are most pronounced for the optics dataset, which gained 0.89 BLEU in the mixed domain test and 0.67 BLEU in the optics domain test. This would suggest that overfitting is not helpful even for domain-specific applications and indeed rather destructive for general model performance.

The next sub-section will provide a human evaluation of a selected number of sentences from the same texts and models.

7.6.2 Human evaluation

Most of the papers researching neural machine translation are content with reporting the BLEU scores and don't dive deeper into analyzing the actual output of the systems. Big part of this, is that translation quality assessment (TQA) is no trivial matter and there is still no agreed upon standard in place that covers all possible text types (Koehn 2010:217-218; Mateo 2014; Bawden 2018). Additionally, human evaluation is very time consuming and therefore expensive. However, there is no lack of works that point out the many insufficiencies that BLEU has as a TQA metric and criticize the overreliance on BLEU by the MT research community (Callison-Burch et al. 2006; Bojar et al. 2017; Bawden 2018). This sub-section therefore aims to provide a manual qualitative and quantitative analysis of a very small sample of sentences predicted by each model. The predictions will be compared not just to a reference text, but also verified against the source text. Since looking at all the 899 sentences for each test corpus would be beyond the scope of this thesis, eight random sentences are chosen from the test corpora (four from test.jp.tk and four from testdom.jp.tk).

Clearly, this is where Translation Studies theories can best be applied. Notably, the *skopos theory* of Reiß/Vermeer (Lindquist et al. 1985) can be applied to machine translation

[135] ntc7wo test was not stopped earlier, as BLEU score was still increasing at 150k steps

and in fact the evaluation itself. We can ask what the purpose (or *skopos*) of the output is and what purpose our evaluation should have (Lo Presti 2016:4). While *gisting*[136] was generally the main purpose of MT translation up to SMT, NMT aims to be an end-to-end translation system, meaning that the output of the system is essentially to be used as-is with only slight corrections at best. Therefore, the purpose of the output in the patent translation paradigm, should be to be as informative and adequate as possible, while maintaining reasonable fluency and very importantly, grammatical correctness in the TT. In this thesis, the evaluation will have the purpose of validating or falsifying the results of the BLEU scores reported in Sub-Section 7.6.1 and will therefore limit itself to concretely score the translations by the different models, like a bi-lingual post-editor would score a human translation, while keeping in mind the *skopos* of the translation as explained above, with the *skopos* being the clear and unaltered conveying of the original message.

To provide the findings in an easier to read metric, an overall translation quality score, that leans on the SAE J2450[137] metric first established for assessing translation quality in the automotive industry, will additionally be provided and its scoring reported for each individual hypothesis in Appendix I.[138] In order to better account for issues on a semantic level, the improved SAE metric by Hui Liu (Liu 2017)[139] will be used. See Appendix I for a full definition of the error-types and error-weights used for the quantitative scoring.

For the final reported score, the overall document weighted score (ODWS) will be reported by combining the findings of all eight analyzed sentences into a single score for each individual machine translation and the reference texts. Please consider that even so, the evaluation is purely sentence based (as is the machine translation) and will not be taking extra-sentential context into consideration (which the reference sentences might have). This means that arbitrary additions or omissions (that change the meaning) in the reference and translations will be treated as an error, even if extra-sentential context would warrant those decisions. Errors need to have an impact on the meaning of the sentence to be classified as such. Major errors are marked in red, minor errors in blue. Wrong terms (WT) are marked bold and color-coded by severity. Omissions (OM) will be additionally visualized with an underscore "_", while Additions (AD) are marked by underlining. Syntactic errors (SE) are marked by italics, whereas misspellings (SP) are marked by putting the expression in question in angular brackets ("[]"). Word structure and agreement errors (SA) are expressed by a double wave around the

[136] Being able to understand the essence of a sentence
[137] https://www.sae.org/standardsdev/j2450p1.htm (accessed April 20, 2020)
[138] A short explanation on how score is calculated, and a definition of errors will also be provided in Appendix I.
[139] With the addition of "Wrong Meaning" error category, which is arguably the most important for our purpose, but missing from the original SAE J2450 metric. Style errors (WS) are not considered in the scoring.

expression (“~expression~”) whereas miscellaneous errors (ME) are expressed by a single wave in front of the expression (“~expression”). Style errors (WS) are not counted towards the total score and therefore not color-coded. A minor style error is shown with a dotted line and a major style error with a dash-dotted line. Wrong Meaning (WM) scores are assigned on a sentence level and evaluated on the basis of how close to the ST the meaning of the translation comes.

First, the mixed-domain test corpus will be analyzed. The first line (ST) will be the source text, the second line (TT) is the reference used for BLEU scoring. The following lines represent the hypotheses by the trained models; the model used will be listed in parenthesis at the beginning of each sentence. By default, the models trained to 150,000 steps will be used, if there is a difference between the longer trained model and the shorter trained model, the less trained model’s hypothesis will also be added in the same line. The broad domain of the patent where the sentence is from, as well as the patent number is listed in the caption. A short textual analysis of each individual example will also be provided after each example.

(**ST**) 1 1 は 7 の 回 動 範囲 を 制限 する ストッパ で ある 。
(**TT**) Numeral 11 indicates a stopper ~**for** restricting the range of rotation of the **lever** 7 . **(SAE J2450 Score) AD: 1*0 + 1*2; ME: 1*1; Weighted Score: 3**
(**ntc7**) ~**A** stopper 11 limits the range of rotation of the **stopper** 7 . **(SAE J2450 Score) AD: 1*4; ME: 1*1; WS:1*0; (WM:1*5); Weighted Score: 10**
(**ntc768k**) Numeral 11 denotes a stopper ~**for** limiting the range of rotation of seven . **(SAE J2450 Score) AD: 1*0; ME: 1*1; Weighted Score: 1** (**ntc768k 80k**) Reference numeral 11 denotes a stopper ~**for** limiting the range of rotation of 7 . **(SAE J2450 Score) AD: 1*0; ME: 1*1; Weighted Score: 1**
(**ntc7o**) Reference numeral 11 denotes a stopper ~**for** limiting a rotation range of 7 . **(SAE J2450 Score) AD: 1*0; ME: 1*1; Weighted Score: 1** (**ntc7o 80k**) A stopper 11 limits the range of **pivotal movement** of 7 . **(SAE J2450 Score) WT: 1*2; WS: 1*0; (WM: 1*2); Weighted Score: 4**
(**ntc7wo**) Reference numeral 11 denotes a stopper ~**for** limiting the range of *rotation 7* . **(SAE J2450 Score) AD: 1*0; SE: 1*4; ME: 1*1; (WM: 1*2); Weighted Score: 7**
(**ntc7wo68k**) Reference numeral 11 designates a stopper ~**for** limiting the range of *rotation 7* . **(SAE J2450 Score) AD: 1*0; SE: 1*4; ME: 1*1; (WM: 1*2); Weighted Score: 7** (**ntc7wo68k 90k**) Reference numeral 11 denotes a stopper ~**for** limiting the range of rotation of 7 . **(SAE J2450 Score) AD: 1*0; ME: 1*1; Weighted Score: 1**

Text example 7.6-1: Line 14 from test.jp.tk; Domain: B65H - PERFORMING OPERATIONS, HANDLING OF THIN/FILAMENTARY MATERIALS
(JP7251966A 1995)

Text example 7.6-1 starts off the comparison with a short sentence taken from classification B65H, which includes “performing operations”, “conveying, packing, storing and handling of thin/filamentary materials”. Right away, we can observe a characteristic of neural machine translation: Each model uses different words for expressing broadly the same meaning, i.e. the use of synonyms is prominent, something that would be rather unusual for SMT. Interestingly, the “best” model as measured by BLEU, the full-dataset ntc7 model, is the only model that provides an objectively wrong translation. It arbitrarily adds a word that is not present in the ST. By looking at the reference sentence, it can be observed that it wrongly infers “7” to stand for another stopper, when in fact it appears to be a lever. This is however NOT visible in the ST were talk is simply of a numeral 7. The degree of paraphrasis is also the highest for ntc7, showing an example of NMT to prefer *fluency* over *adequacy*. This is further accentuated by the fact, that all models add words not found in the source text (like “reference numeral*”* or “numeral”) and stray quite far from a literal translation. For instance, the ST does not explicitly state that the “stopper 11” is there **for** limiting a range of rotation, but rather that it simply does it. This will be counted as a small miscellaneous error (ME), as it does not change the meaning of the sentence significantly.

Additionally, the models (as does the reference) change the sentence structure quite deeply. It may be argued, that human translators, like myself, prefer a more literal translation

when working with patents and would rather produce something like the following sentence: “[Reference numeral] 11 is a stopper that limits the rotation range of 7“. This will however not be counted as an error, as the general meaning is unchanged by the reformulation of the sentence.

It is interesting to observe, that all models appear to correctly find the correlation between the “numeral 11” and the “stopper”, as some models simply write “a stopper 11” as the subject. On the other hand, some models appear to have problems to find the relation of the numeral 7, as it is not explicitly defined in the text. As pointed out, the ntc7 model assumed 7 to be another stopper, while models ntc7wo and ntc7wo68k assume the 7 to refer to the range of rotation, which is also not correct (and grammatically implied not to be so in the ST). While the error in the ntc7 sentence will be counted as a major addition error (AD), the errors in the other two models are counted as major syntax errors (SE). However, the meaning is arguably more obfuscated in the ntc7 model’s hypothesis, so this will be counted as a major wrong meaning (WM), while the ntc7wo variants will be counted as minor wrong meanings. The ntc7o68k model also appears to have gone for a debatable word-choice, choosing to translate 回動 (*kaidō;* rotation) as “pivotal movement”, which actually changes the meaning somewhat depending on how you interpret the term “pivotal”. This is counted as a minor wrong term, resulting in a minor wrong meaning.

Another common issue appears to be the selection of which article to use for the different nouns of the sentence. The definite article “the” or the indefinite “a”: In patent-language choosing “a” is quite common, as it is the safer translation, but when a numeral is explicitly pointed out, “the” seems like the more sensible choice. For that reason, “a stopper 11” feels somewhat off, but is an acceptable solution and will therefore only be separately reported as a minor style error (WS). However, the ntc7 sentence is inconsistent in its use of articles, as it refers to “a stopper 11” but then later to “the stopper 7”; this will be classified as a minor miscellaneous error (ME).

(**ST**) データの書込／読出が行なわれた後には、ローカルＩＯ線およびグローバルＩＯ線は、列選択動作完了時に所定電位にプリチャージ／イコライズされる。
(**TT**) After data is written / read , the local IO line and global IO lines are precharged / equalized to a **prescribed** potential at the time of the completion of the column select operation . **(SAE J2450 Score) WT: 1*2; Weighted Score: 2**
(**ntc7**) After data writing / reading , local IO lines and global IO lines are precharged / equalized to a predetermined potential at the completion of the column selecting operation . **(SAE J2450 Score) Weighted Score: 0** (**ntc7 120k**) After data writing / reading , the local I / O lines and global I / O lines are precharged / equalized to a predetermined potential at the completion of the column selecting operation . **(SAE J2450 Score) Weighted Score: 0**
(**ntc768k**) After data is written / read , ~ local IO line and ~ global IO line are precharged / equalized to a predetermined potential upon completion of ~ column selecting operation . **(SAE J2450 Score) ME: 3*1; Weighted Score: 3** (**ntc768k 80k**) After data is written / read , local I / O lines and global I / O lines are precharged / equalized to a predetermined potential upon completion of a column select operation . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7o**) After data is written / read , the local I / O lines and the global I / O lines are precharged / equalized to a predetermined potential when the column selecting operation is completed . **(SAE J2450 Score) Weighted Score: 0** (**ntc7o 80k**) After data is written / read , the local I / O line and the global I / O line are precharged / equalized to a predetermined potential when the column selecting operation is completed . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7wo**) After writing / reading of data , ~ local IO line and ~ global IO line are precharged / equalized to a predetermined potential upon completion of a column selecting operation . **(SAE J2450 Score) ME: 2*1; Weighted Score: 2**
(**ntc7wo68k**) After data writing / reading , local IO lines and global IO lines are precharged / equalized to predetermined potentials at the completion of a column selecting operation . **(SAE J2450 Score) Weighted Score: 0**

Text example 7.6-2: Line 380 from test.jp.tk; Domain: G11C - PHYSICS; STATIC STORES (JP9288888A 1997)

Text example 7.6-2 shows a longer sentence, more representative for most sentences in patent documents. The classification, G11C, is somewhat related to the Optics domain, as it is part of the same overarching domain “Physics”. Here most models perform almost the same, providing a translation very close indeed to the reference text and in fact, the source text. There is only a slight variation in the morphological form in the beginning of the sentence (“After data is written / read” or “After data writing / reading”) and the choice in how to express “完了時に” (*kanryō no toki ni;* “at the time of completion”): The neural models all prefer a more paraphrased translation than the human reference translation. Additionally, some models add a slash between “IO”, which is an accepted abbreviation of “Input/Output”. Finally, another variation can be observed in the choice of how “列選択動作” (*retsu sentaku dōsa;* column select operation) is translated. Valid possibilities include “column select operation”, “column selection operation” and “column selecting operation”. The models varied between “column

select operation" and "column selecting operation", it would be interesting to see whether the models would maintain consistency over a whole document.

The only error that can be observed in the context of the sentence, is the omission of articles, which will be counted as miscellaneous errors (ME) as they have no significant effect on the meaning of the sentence. These errors are found once in front of the expression "column select operation" on the ntc768k model and in several other models in front of the "local IO" / "global IO".

In that regard another very subtle issue can be observed, which allures to the problem of using only sentence-level translation. In the source text, it is unclear whether there are multiple IO lines or only one IO line and whether there is only one predetermined potential or multiple potentials. We can see that the translator of the reference text chose to write it as one "local IO line" but multiple "global IO lines" and a single potential. This is something that can only be correctly translated if a broader context is taken into consideration, or, in many cases, the patent illustrations are studied by the translator. It can be observed that the NMT models chose either the singular "line" or the plural "lines" but never arbitrarily mixed the two. While this means that no translation is strictly speaking correct in the context of the whole patent, it won't be counted as an error as it would virtually be impossible to know the correct answer without looking at the whole patent / illustrations. As a positive note, the choice was kept consistent within the hypothesis of each model, making it easier to correct in a post-editing step. Additionally, the choice of translation for "所定" (*shotei;* prescribed, fixed, predetermined) seems more sensible in the machine translations than the reference, at least when exclusively viewed in the context of this sentence. The reference appears to have relied on the first listing of most dictionaries ("prescribed") and the translation is therefore regarded as a minor wrong term error (WT).

(**ST**) ドライバトランジスタ ＱＤ ２ は 、 記憶 ノード ＳＮ ２ と ディプリーション 型 トランジスタ ＱＤＰ ２ と の 間 に 設け られ 、 ゲート は 記憶 ノード ＳＮ １ に 接続 さ れる 。
(**TT**) Driver transistor QD2 is placed between storage node SN2 and depletion type transistor QDP2 and **has its** gate connected to storage node SN1 . **(SAE J2450 Score) AD: 1*4; (WM: 1*2); Weighted Score: 6**
(**ntc7**) Driver transistor QD.sub.2 is provided between storage node SN2 and depletion type transistor **<unk>** , and **has** a gate connected to storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 1*2; (WM: 1*2); Weighted Score: 8** (**ntc7 120k**) The driver transistor **DB2** is provided between the storage node SN2 and the depletion @-@[140] mode transistor **<unk>** , and **its** gate is connected to the storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 1*2 + 1*5; (WM: 1*2); Weighted Score: 13**
(**ntc768k**) Driver transistor **<unk>** is provided between storage node SN2 and depletion @-@ mode transistor **<unk>** , and **its** gate is connected to storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 2*2; (WM: 1*2); Weighted Score: 10**
(**ntc7o**) Driver transistor **<unk>** is provided between storage node SN2 and depletion @-@ type transistor **<unk>** , and a gate is connected to storage node SN1 . **(SAE J2450 Score) WT: 2*2; Weighted Score: 4** (**ntc7o 80k**) The driver transistor **<unk>** is provided between the storage node SN2 and the depletion @-@ type transistor **<unk>** , and **its** gate is connected to the storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 2*2; (WM: 1*2); Weighted Score: 10**
(**ntc7wo**) Driver transistor **<unk>** is provided between storage node SN2 and depletion transistor **<unk>** , and **its** gate is connected to storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 2*2; (WM: 1*2); Weighted Score: 10**
(**ntc7wo68k**) Driver transistor QD.sub.2 is provided between storage node SN2 and depletion type transistor **<unk>** , and **its** gate is connected to storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 1*2; (WM: 1*2); Weighted Score: 8** (**ntc7wo68k 90k**) Driver transistor QD.sub.2 is provided between storage node SN2 and depletion type transistor **<unk>** , and has **its** gate connected to storage node SN1 . **(SAE J2450 Score) AD: 1*4; WT: 1*2; (WM: 1*2); Weighted Score: 8**

Text example 7.6-3: Line 436 from test.jp.tk; Domain: G11C - PHYSICS; STATIC STORES (JP10154393A 1998)

Text example 7.6-3 comes from the same domain of “Physics - Static Stores” as the example above and shows some of the inherent issues of NMT: Note the “<unk>” token used instead of the “QD” labels. Clearly, the “QD2” and “QDP2” label was not part of the vocabulary and as such will be covered by the “<unk>” token. As can be observed, the English tokenization did not separate the numeral from the letters and therefore “QD2” and “QDP2” would be stored as a single token in the English vocabulary, explaining why not even the numeral was written in the hypotheses. Only the full-size “ntc7” model and the smaller “ntc7 without optics model” translate one of the labels in the text: the “QD2”. Notice the added “.sub.” token, which was added through tokenization and is therefore not to be counted as an error. Indeed, the label appears in different writing form in the training data (like QD_2) which is expressed through the

[140] Hyphen marked by the Tokenizer

“.sub.” token by the Moses tokenizer. The “@” signs around hyphens are also added by the tokenizer and therefore not counted as errors as these issues could be easily resolved by detokenizing the data or, even better, resolving the inconsistencies in a preprocessing step for a more efficient vocabulary. The <unk> tokens, will be regarded as minor wrong term errors as there already exist several post-processing approaches to tackle this issue, by either copying the term in question from the ST and/or or replacing it by external dictionary look-up (see “copy mechanism” in M.-T. Luong 2016:40-54). However, a downright wrong denomination like the ntc7 120k model provides (where does the “DB2” come from?) will be regarded as major wrong term error (WT).

Aside from these issues, the translation hypothesis of all models is very close to the reference text and, interestingly, the ntc7o model provides the most grammatically accurate translation of the source text: While the reference text and all other hypotheses assume that the “gate” is in fact of the “driver transistor QD2”, this is grammatically not stated in the Japanese source text, making the indefinite formulation “…and a gate is connected to …” the most correct translation in this context-free analysis. Since NMT translation is still sentence-based, seemingly arbitrary additions like these could drastically change the meaning of a sentence, while remaining undiscernible by monolingual revision and are thus regarded as major addition errors (AD) as well as being counted as minor wrong meanings (WM).

(**ST**) 図 3 は 、 この 実施 の 形態 に 係る 周波数 逓倍 回路 の 構成 を 示す 回路 図 で ある 。
(**TT**) FIG . 3 is a circuit diagram showing a **construction** of the frequency multiplication circuit in **the second** embodiment . **(SAE J2450 Score) WT: 1*2 + 1*5; Weighted Score: 7**
(**ntc7**) FIG . 3 is a circuit diagram showing the configuration of a frequency multiplier _ according to this embodiment . **(SAE J2450 Score) OM: 1*2; WS: 1*0; Weighted Score: 2**
(**ntc768k**) FIG . 3 is a circuit diagram showing the configuration of a frequency multiplying circuit according to this embodiment . **(SAE J2450 Score) WS: 1*0; Weighted Score: 0** (**ntc768k 80k**) FIG . 3 is a circuit diagram showing the configuration of the frequency multiplying circuit according to this embodiment . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7o**) FIG . 3 is a circuit diagram showing the **structure** of a frequency multiplication circuit according to this embodiment . **(SAE J2450 Score) WT: 1*2; WS: 1*0; Weighted Score: 2**
(**ntc7wo**) FIG . 3 is a circuit diagram showing the configuration of a frequency multiplier _ according to this embodiment . **(SAE J2450 Score) OM: 1*2; WS: 1*0; Weighted Score: 2**
(**ntc7wo68k**) FIG . 3 is a circuit diagram showing the configuration of a frequency multiplier _ according to this embodiment . **(SAE J2450 Score) OM: 1*2; WS: 1*0; Weighted Score: 2** (**ntc7wo68k 90k**) FIG . 3 is a circuit diagram showing the configuration of the frequency multiplier _ according to this embodiment . **(SAE J2450 Score) OM: 1*2; Weighted Score: 2**

Text example 7.6-4: Line 777 from test.jp.tk; Domain: H03K - ELECTRICITY; BASIC ELECTRONIC CIRCUITRY (JP10322174A 1998)

Text example 7.6-4 is taken from a patent with domain-classification H03K, which represents electricity and more specifically basic electronic circuitry. The issue of context is again apparent in this example. However, in this case the reference sentence contains information absent from the actual source text: The reference refers to a “second embodiment”, while the source text only states “this embodiment” (この実施の形態; *kono jisshi no keitai*). All models translated/inferred this correctly. Likewise, the word choice for 構成 (*kōsei*; constitution, configuration, structure) seems more adequate in most machine translated sentences than the reference sentence, undoubtedly and ironically resulting in a lower BLEU score for this sentence. What is interesting to note, is that the two largest models, ntc7 and ntc7wo, and the smaller ntc7wo68k model are again displaying the tendency of NMT to prefer fluency over *adequacy*, by omitting the word “circuit” from the “frequency multiplication circuit”. Otherwise different inflections of “multiplication” are used in the same expression. A varying use of the definite article “the” and the indefinite article “a” can also be observed throughout. While generally the safer way to translate is using the indefinite article “a” unless it is specified

that the object in question already appeared in the text before (前記; *zenki;* said, aforementioned), in this case the circuit according to a specific embodiment is cited, so “the” seems more appropriate, but this will only be counted as a stylistic error and not calculated into the score.

(**ST**) ＣＰＵ （中央処理装置） １７ は ＲＯＭ （リードオンリメモリ） １８ に記憶された プログラム に基づいて 装置 全体 の 動作 を 制御 する 。
(**TT**) The CPU 17 controls ***an overall operation*** of the apparatus **~on a~** program stored in a ROM 18 . **(SAE J2450 Score) SA: 1*2; SE: 1*2; WS: 2*0; Weighted Score: 4**
(**ntc7**) A CPU (Central Processing Unit) 17 controls the operation of the entire apparatus based on a program stored in a ROM (Read Only Memory) 18 . **(SAE J2450 Score) Weighted Score: 0**
(**ntc768k**) A CPU (central processing unit) 17 controls the operation of the entire apparatus on the basis of a program stored in a ROM (read only memory) 18 . **(SAE J2450 Score) Weighted Score: 0** (**ntc768k 80k**) A CPU (central processing unit) 17 controls the operation of the whole apparatus based on a program stored in a ***read @-@ only memory (ROM)*** 18 . **(SAE J2450 Score) SE: 1*2 Weighted Score: 2**
(**ntc7o**) A CPU (central processing unit) 17 controls the operation of the entire apparatus on the basis of a program stored in a ROM (read only memory) 18 . **(SAE J2450 Score) Weighted Score: 0** (**ntc7o 80k**) A CPU (central processing unit) 17 controls the operation of the entire apparatus on the basis of a program stored in a ROM (read @-@ only memory) 18 . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7wo**) A CPU (central processing unit) 17 controls ***the overall operation*** of the apparatus in **accordance** with a program stored in a ROM (read only memory) 18 . **(SAE J2450 Score) WT: 1*2; SE: 1*2; Weighted Score: 4**
(**ntc7wo68k**) A CPU (central processing unit) 17 controls ***the overall operation*** of the apparatus based on a program stored in a ROM (read only memory) 18 . **(SAE J2450 Score) SE: 1*2; Weighted Score: 2** (**ntc7wo68k 90k**) A CPU (Central Processing Unit) 17 controls ***the overall operation*** of the apparatus on the basis of a program stored in a ROM (Read Only Memory) 18 . **(SAE J2450 Score) SE: 1*2; Weighted Score: 2**

Text example 7.6-5: Line 17 from testdom.jp.tk; Domain: G03G - PHYSICS - PHOTOGRAPHY; ELECTROGRAPHY
(Patent JP05088550A 1993)

Text example 7.6-5 shows the first sentence taken from the optics-domain specific corpus testdom.jp.tk. It is a line taken from a patent with the classification G03G, which stands for Photography/Electrography. This is a rather interesting example, as the machine translation varies significantly from the reference text. Notice how the reference text omits the explanation in the brackets present in the source text, while it is marked as an omission it will not be added to the score as it does not change the meaning of the sentence, but will be reported as a style error (WS). All machine translations correctly translate this, albeit with some variation in typesetting (capital letters, hyphens) and a curious exception of the order being flipped by the small domain-mixed model ntc768k at 80k steps.

More importantly, however, some of the machine translated sentences appear both grammatically more correct and closer to the original meaning and wording of the sentence, than the reference text. The main action of the sentence in the source text is 装置全体の動作

を制御する (*sōchi zentai no dōsa wo seigyō suru)*, meaning “to control the whole apparatus’ operation”, whereas the models based on data without optics (ntc7wo and ntc7wo68k) adhere to the reference translation by referring 全体 (*zentai;* “entire”, “whole”, “overall”) to the operation instead of the apparatus. On the other hand, the models that had optics sentences to learn from (ntc7, ntc768k, ntc7o) did translate the apparatus as the object of the sentence and refer the adverb “whole” to it. Note that this is a somewhat ambivalent utterance in the ST, as the meaning is almost unchanged, but grammatically a further possessive pronoun の would be required to refer 全体 (*zentai;* entire, whole) to the operations instead of the apparatus. As such, this is regarded as a minor syntactic error (SE). Reading the whole patent did not provide a conclusive answer as to what might have been intended, but in doubt sticking to grammatical cues in the ST is the way to go.

One further divergence from the reference text favors all the machine translated results. The reference text omits the “based on” in front of a “a program stored in ROM”, which, while still conveying the meaning to humans, makes no sense grammatically speaking. This is therefore considered as a minor agreement error (SA) in the reference text. The solutions proposed by most neural models are objectively both grammatically and semantically better (“on the basis of” and “based on a”), but would again, ironically, result in a lower BLEU score than a solution closer to the reference text. The solution proposed by the ntc7wo model (“in accordance with”) is quite paraphrased from the ST (に基づく , *ni motodsuku,* on the basis of), and in this case rather unfitting. It is therefore regarded as a minor wrong term (WT).

(**ST**) また 、 ゴム が 感光 体 と の 接触 で 磨耗 し て ロール 表面 の 凹凸 が 初期 状態 から 大きく 変化 し て 帯電 の 均一 性 が 損なわ れる という 問題 が 生じる 。
(**TT**) Further , _ the rubber is worn away by contact with the photoreceptor **~to largely change~** the unevenness of the surface of the roll from the initial state , thereby impairing the uniformity of charging . **(SAE J2450 Score) OM: 1*2; SA: 1*2; Weighted Score: 4**
(**ntc7**) Further , there is a problem that the rubber is abraded due to the contact with the photosensitive member and the unevenness of the surface of the roll is largely changed from the initial state , thereby impairing the uniformity of charging . **(SAE J2450 Score) Weighted Score: 0** (**ntc7 120k**) Further , there arises a problem that the rubber is worn out by contact with the photosensitive member and the unevenness on the surface of the roll largely changes from the initial state , thereby **damaging** the uniformity of charging . **(SAE J2450 Score) WT: 1*2; Weighted Score: 2**
(**ntc768k**) Further , there is a problem that the rubber becomes worn due to contact with the photosensitive member and the unevenness of the roll surface largely changes from the initial state , thereby deteriorating the uniformity of the charging . **(SAE J2450 Score) Weighted Score: 0** (**ntc768k 90k**) In addition , there arises a problem that the rubber wears away due to contact with the photosensitive member , and the unevenness of the surface of the roll largely changes from the initial state , thereby deteriorating the uniformity of charging . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7o**) Further , there arises a problem that the unevenness of the surface of the roll largely changes from the initial state due to abrasion of the rubber due to contact with the photosensitive body and the uniformity of charging is deteriorated . **(SAE J2450 Score) WS: 1*0; Weighted Score: 0** (**ntc7o 80k**) Further , there arises a problem that the rubber is ***worn away from the photosensitive body*** by contacting with the photosensitive body and the roughness of the roll surface is largely changed from the initial state so that the charging uniformity is deteriorated . **(SAE J2450 Score) SE: 1*4; (WM: 1*5); Weighted Score: 9**
(**ntc7wo**) In addition , there arises a problem that the rubber is worn out by contact with the photosensitive body , and the unevenness of the roll surface largely changes from the initial state , thereby damaging the uniformity of charging . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7wo68k**) In addition , there arises a problem that when the rubber is worn in contact with the photosensitive body , the **projections and recesses** on the surface of the roll greatly change from the initial state , thereby impairing the uniformity of charging . **(SAE J2450 Score) WT: 1*2; Weighted Score: 2** (**ntc7wo68k 90k**) In addition , there is a problem that the unevenness on the surface of the roll is largely changed from the initial state by the abrasion of the rubber in contact with the photoreceptor , resulting in ~ loss of the uniformity of the **electrification** . **(SAE J2450 Score) WT: 1*2; ME: 1*1; Weighted Score: 3**

Text example 7.6-6: Line 432 from testdom.jp.tk; Domain: : G03G - PHYSICS - PHOTOGRAPHY; ELECTROGRAPHY
(Patent JP08062939A 1996)

Text example 7.6-6 is another sentence from the optics test corpus, again with the classification G03G, photography and electrography. It is an interesting example, in that it shows just how flexibly the neural models arrange the syntax and choose words/prepositions, while keeping the

same meaning. By and large, all models convey the meaning successfully, while also being grammatically correct and indeed very close to the reference text (which omits the “there arises a problem” part present in the source text and makes it seem like this is something that happens on purpose). The only exception being the early-stopped ntc7o model, which wrongly infers that “the rubber is worn away from the photosensitive body”. In the source text, the rubber (ゴム) is not directly related to any of the other nouns in the sentence, but one would assume from the context available that, if anything, the “rubber” is part of the “roll”, as the “roll’s surface” unevenness is changed. An assumption that can only be verified when reading the patent’s context, which confirms that the roll possesses a conductive rubber layer (Patent JP08062939A 1996:2). This results in a major syntactic error (SE) and subsequently a minor meaning error (WM).

Again, the results for this example imply that the BLEU score cannot be taken at face value, as the early stopped ntc7o model consistently performed better in the BLEU metric, despite being the only model that mistranslated this example. That said, subjectively, the translations hypotheses found by both ntc7o models are arguably the least refined, as the sentence reordering is somewhat confusing. This is once again noted as minor wrong style (WS), but not considered in the overall score.

(ST) 写真乳剤層に入射すべき光の分光組成を制御することが必要なとき、写真感光上の写真乳剤層よりも支持体から遠い側に着色層が設けられる。
(TT) A colored layer can be formed on the side further from the support than the photosensitive photographic emulsion layer , ~where~ it is necessary to control the spectral composition of the light *which falls* on the photographic emulsion layer . **(SAE J2450 Score) SE: 1*2; SA: 1*2; Weighted Score: 4**
(ntc7) When it is necessary to control the spectral composition of the light to be incident on the photographic emulsion layer , a colored layer is provided on the side farther from the support than the photographic emulsion layer on the photographic photosensitive layer . **(SAE J2450 Score) Weighted Score: 0** (ntc7 120k) When it is necessary to control the spectral composition of light to be incident on the photographic emulsion layer , a colored layer is provided on the side farther from the support than the photographic emulsion layer on **the photographic light** . **(SAE J2450 Score) WT: 1*5; Weighted Score: 5**
(ntc768k) When it is necessary to control the spectral composition of the light to be incident on the photographic emulsion layer , a colored layer is provided farther from the support than the photographic emulsion layer on **the photographic light** . **(SAE J2450 Score) WT: 1*5; Weighted Score: 5** (ntc768k 80k) When it is necessary to control the spectral composition of light to be incident on the photographic emulsion layer , a colored layer is provided on the side farther from the support than the photographic emulsion layer on the photographic light @-@ sensitive surface . **(SAE J2450 Score) Weighted Score: 0**
(ntc7o) When it is necessary to control the spectral composition of the light incident on the photographic emulsion layer , a colored layer is provided **on a portion** farther from the support than the photographic emulsion layer _. **(SAE J2450 Score) WT: 1*2; OM: 1*2; Weighted Score: 4** (ntc7o 80k) When it is necessary to control the spectral composition of light incident on the photographic emulsion layer , a colored layer is formed **on a portion** farther from the support than the photographic emulsion layer _. **(SAE J2450 Score) WT: 1*2; OM: 1*2; Weighted Score: 4**
(ntc7wo) When it is necessary to control the spectral composition of the light to be incident on the photographic emulsion layer , a colored layer is provided at a position **remote** from the supporting member *rather than* the photographic emulsion layer on **the photographic image** . **(SAE J2450 Score) WT: 2*5; SE: 1*2; Weighted Score: 12**
(ntc7wo68k) When it is necessary to control the spectral composition of the light to be incident on the photographic emulsion layer , a *coloring* layer is provided on the side farther away from the support member than the photographic emulsion layer on **the photographic toner** . **(SAE J2450 Score) WT: 1*5; SE: 1*2; Weighted Score: 7** (ntc7wo68k 90k) When it is necessary to control the spectral composition of the light to be incident on the photographic emulsion layer , a *colorant* layer is provided on the side farther from the support than the photographic emulsion layer on **the photographic light** . **(SAE J2450 Score) WT: 1*5; SE: 1*2; Weighted Score: 7**

Text example 7.6-7: Line 651 from testdom.jp.tk; Domain: G03C - PHOTOSENSITIVE MATERIALS FOR PHOTOGRAPHIC PURPOSES (JP5323501A 1993)

Text example 7.6-7 shows an example from classification G03C, photosensitive materials for photographic purposes. This sentence stands out as one of the more difficult to translate, as it

spans over a long main clause and a separate, equally long subordinate clause. It is interesting to note, that all models chose to adhere to the Japanese sentence order, while the human reference translation reversed the sentence order. This in itself does not obfuscate the meaning in any way.

The main issue appears to be with the expression 写真感光上の写真乳剤層, which might well be a typo of the original patent document (***shashinkankōjō*** *no shashinnyūzaizō;* "the photographic emulsion layer **on the photographic light-sensitive (…)** ")[141]. In this occasion, some models provide completely non-sensical solutions like "photographic light" (ntc7 120k, ntc768k and ntc7wo68k 90k), "photographic image" (ntc7wo) or "photographic toner" (ntc7wo68k), the optics-based models simply omit the expression while keeping the essential meaning (ntc7o, ntc7o 80k)[142], while the inference of the two models based on the varied domain corpus proposes the arguably most adequate solutions with "photographic light-sensitive surface" and "photographic photosensitive layer" (ntc7, ntc768k 80k). It is fascinating to see how the optics-data based ntc7o model provides a very consistent output, while all other models are perplexed by the uncommon (and as a matter of fact, incorrect) expression. Clearly, in this case having less perplexity on the training data helped the model. On the other hand, the largest model (ntc7) achieved a solution closely resembling the reference text, while being closest to the ST wording. In this case, the omission of the ntc7o models seems preferable to the arbitrary wrong term of the other models, which is why only a minor omission (OM) was deducted from the ntc7o models, while the other models had a major wrong term (WT) deducted.

What is also interesting, is that all models chose to translate the ambiguous 設けられる (*mōkerareru;* "to be set up" or "can be set up") as a passive, while it can be either the passive or potential form of 設ける (*mōkeru;* set up). Only the reference text interpreted it as the potential form. This is again an ambiguity in the Japanese language, which can only be cleared up upon further inspection of the broad context in the patent and will therefore not be counted as an error of any sort.

[141] In fact, the expression only appears one more time in the patent, in a carbon copy of above sentence. All other appearances state 写真感光材料 (*shashinkankōzairyō*) which stands for „photographic light-sensitive material".
[142] Interestingly, the training corpus also contains the expression twice, also in the form of a carbon copy of the same sentence in the test data but from different patent documents. One reference (or label) sentence in the training corpus omits the expression in question and keeps the Japanese sentence order; the other is the same as the reference used in the test corpus.

(ST) 次いで、 Ｓ ３ １ において、 モード 設定 ダイアル ７ の ポジション が 変更 さ れ た か 否 か を 判断 する 。
(**TT**) Next , in **step** S31 , it is determined whether or not the position of the mode setting dial 7 has been changed . **(SAE J2450 Score) AD: 1*2; Weighted Score: 2**
(**ntc7**) _ In S31 , it is determined whether the position of mode setting dial 7 has been changed . **(SAE J2450 Score) OM: 1*2; Weighted Score: 2** (**ntc7 120k**) _ In S31 , it is determined whether the position of the mode setting dial 7 has been changed . **(SAE J2450 Score) OM: 1*2; Weighted Score: 2**
(**ntc768k**) Then , in S31 , it is determined whether or not the position of the mode setting dial 7 has been changed . **(SAE J2450 Score) Weighted Score: 0** (**ntc768k 90k**) Next , in S31 , it is determined whether or not the position of the mode setting dial 7 has been changed . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7o**) Next , in S31 , it is determined whether the position of the mode setting dial 7 has been changed . **(SAE J2450 Score) Weighted Score: 0**
(**ntc7wo**) _ In S31 , whether or not the position of mode setting dial 7 ***is changed*** is determined . **(SAE J2450 Score) SE: 1*2; OM: 1*2; Weighted Score: 4**
(**ntc7wo68k**) Next , at **step** S31 , it is determined whether the position of the mode setting dial 7 ***is changed*** . **(SAE J2450 Score) SE: 1*2; AD: 1*2; Weighted Score: 4**

Text example 7.6-8: Line 508 from testdom.jp.tk; Domain: G03B - PHYSICS - APPARATUS OR ARRANGEMENTS FOR TAKING PHOTOGRAPHS OR FOR PROJECTING OR VIEWING THEM (JP103115A 1998)

Finally, Text example 7.6-8 shows a shorter sentence from the classification G03B, indicating the invention is regarding an apparatus or arrangements for taking photographs or projecting/viewing them. This example just shows some seemingly arbitrary omissions of the conjunction 次いで (*tsuide;* then, next…), by some of the models, notably the ones based on more data (ntc7 and ntc7wo). The other thing that can be observed is the addition of the word "step" by the ntc7wo68k model, although it never appears in the source text. Interestingly the reference sentence also adds this word to the translation, indicating that at some point the numeral "S31" must have been identified as a step in the patent. This could however be an invalid addition, as there is no way to tell what S31 exactly is just by looking at the ST, so a minor addition error (AD) is calculated.

7.7 Conclusion

The training of the translation models worked unexpectedly well. In fact, the results of the translation models are very close to the performance of commercially used systems like the EPO's *Patent Translate* and *WIPO Translate* for the small sampled texts. This is great news for translators who would like to explore the option of maintaining their own neural machine translation system with standard home-computing equipment.

On one hand, the automatic evaluation confirmed what is common knowledge in MT research: Training on bigger data appears to be generally better than on smaller, even specialized data-sets, and training for too long, leading to a higher model perplexity for general tasks, leads to worse results. However, the human evaluation also proved that it is quite dangerous to rely only on the BLEU score for assessing model performance. This is shown in the ranking presented in Table 16, which ranks the models by comparing the BLEU scores (higher is better) and the SAE overall weighted document scores (OWDS[143], lower is better).

Table 16: Ranking BLEU and SAE scores between the different models and test domains

Model	BLEU Rank mixed domain	SAE Rank mixed domain	BLEU Rank optics domain	SAE Rank optics domain
ntc7	40.29 (2nd)	0.047281324 (6th)	37.57 (2nd)	0.008810573 (1st)
ntc7 (120k)	40.42 (1st)	0.059101655 (8th)	37.80 (1st)	0.039647577 (4th)
ntc768k	38.63 (6th)	0.033096927 (3rd)	35.69 (5th)	0.022026432 (3rd)
ntc768k (80k)	38.73 (5th)	0.026004728 (2nd)	36.24 (3rd)	0.008810573 (1st)
ntc7o	36.95 (9th)	0.016548463 (1st)	35.45 (7th)	0.017621145 (2nd)
ntc7o (80k)	37.84 (8th)	0.037825059 (4th)	36.12 (4th)	0.057268722 (5th)
ntc7wo [144]	39.85 (3rd)	0.049645390 (7th)	35.68 (6th)	0.088105727 (8th)
ntc7wo68k	38.52 (7th)	0.040189125 (5th)	34.51 (9th)	0.066079295 (6th)
ntc7wo68k (90k)	38.81 (4th)	0.026004728 (2nd)	34.94 (8th)	0.070484581 (7th)

Ironically, the ntc7o model, which has the lowest BLEU scores, happened to be the model with best overall SAE J2450 score. Conversely, the ntc7 model at 120k steps, which performed best in the BLEU metric, is only slightly better than the worst model in the overall SAE J2450 analysis, which is the model based on the second largest dataset, the "without optics" ntc7wo model. However, the optics model (ntc7o) did not perform best in the optics-domain test, where it performed slightly worse than the two mixed-domain models (ntc7 and ntc768k). The errors

[143] The overall weighted document score (OWDS) is calculated as the total weighted score divided by the number of words of all tested sentences. As per convention, each character is counted as a word in Japanese.
[144] ntc7wo test was not stopped earlier, as BLEU score was still increasing at 150k steps

made by the optics model in the optics test, mainly stem from arbitrary omissions and structural errors, while in the mixed domain test it made more wrong term mistakes. Ironically, the optics model (ntc7o) was less prone to arbitrary additions or omissions in the mixed domain test. This would suggest, that the less perplex an NMT model gets, the likelier it is to omit or add arbitrary content to a translation. Keep in mind, that the analyzed sample size is quite small and therefore needs further testing on larger sample sizes for a more reliable conclusion. An overview of the error count sorted by error type can be seen in Table 17.

Table 17: Extended SAE J2450 based quantitative evaluation of all test sentences in Section 7.6.2.

	WT		SE		OM		AD		SA		SP		PE		ME		WM		WS		Weighted Score	OWDS
Weights (Serious/ Minor)	5	2	4	2	4	2	4	2	4	2	3	1	2	1	3	1	5	2	0	0		
	Number of Errors for Each Category																				Words: 423	
TT	1	2	0	2	0	1	1	1	0	3	0	0	0	0	0	1	0	1	0	2	30	0.07092199
ntc7	0	1	0	0	0	2	2	0	0	0	0	0	0	0	0	1	1	1	0	2	22	0.05200946
ntc7 120k	2	2	0	0	0	2	2	0	0	0	0	0	0	0	0	1	1	1	0	2	34	0.08037825
ntc768k	1	2	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	1	0	1	19	0.04491726
ntc768k 80k	0	2	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	13	0.03073286
ntc7o	0	4	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	2	11	0.02600473
ntc7o 80k	0	5	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	2	0	2	29	0.05673759
ntc7wo	2	3	1	3	0	2	1	0	0	0	0	0	0	0	0	3	0	2	1	1	41	0.09692671
ntc7wo68k	1	2	1	3	0	1	1	1	0	0	0	0	0	0	0	1	0	2	0	1	32	0.07565012
ntc7wo68k 90k	1	2	0	3	0	1	1	1	0	0	0	0	0	0	0	2	0	1	0	0	27	0.06382979

The overall result showing a large divergence between human evaluation and automatic evaluation does coincide largely with the findings of research that covers the topic in-depth (Callison-Burch et al. 2006; Lo Presti 2016; Bojar et al. 2017; Bawden 2018). There is in fact a need to again address one major issue of most machine translation systems and their evaluation: All the translation and subsequently evaluation is strictly performed on a sentence basis; this inherently fails to emulate a lot of the combinational work translators must do while translating and fails to correctly evaluate *adequacy* over a whole document. Even the small randomized sample that was analyzed showed many issues related to context outside of the sentence and made it very difficult to analyze it with the SAE metric. This is a well-known issue often ignored by the MT research sphere, as it is quite complex to realize a holistic context-aware machine translation and evaluation system, both in terms of general complexity of the concept of context and in fact the modeling capacity of even the highest performing neural network models.[145]

[145] An incredibly thorough overview of the situation regarding contextual machine translation and evaluation thereof, as well as some approaches to tackle the issue can be found in *Going beyond the sentence: Contextual Machine Translation of Dialogue,* by Rachel Bawden (Bawden 2018).

These issues can also be observed in the evaluation of this thesis as, interestingly, even the reference texts often referred to information that was not available in the actual ST.

This highlights another issue with machine translation training and evaluation. The human evaluation showed that the reference sentences were quite far removed from the ST more often than not, sometimes even containing major translation errors. The reason for this is likely that the parallel sentence data is built from original Japanese documents and the final published English patents. The published patents are not direct translations, but texts that have been further revised by a specialized patent lawyer. Generally, these revisions are mono-lingual, meaning the ST is not considered for the revision and thus it seems plausible that the texts might change significantly from the original ST.

Considering the varying accuracy of the parallel data in the test and training corpus, it is remarkable that the translation works as well as it does, but it was clearly shown, that the networks may learn arbitrary translation patterns, that may refer to context not available in the sentence that is being translated. It would be interesting to see how the models would perform, if they were trained on the translations of the actual translators instead of the patent documents revised by patent lawyers. A more exhaustive evaluation of full patent documents translated with NMT models trained on published, patent lawyer revised data and then compared to a model trained on more literal direct translations by translators would be very interesting.

8 Summary

This thesis aimed to provide an accessible introduction to machine translation and in specific neural machine translation, especially for Translation Studies scholars and translators. This was achieved by giving a broad overview of the major machine translation architectures, with a deep-dive into the current state-of-the-art technique dubbed neural machine translation or NMT. Furthermore, the thesis provides a step-by-step tutorial for creating machine translation models based on this recent modeling technique and accessibly introduces many concepts from the computational linguistics and generally IT to the reader so that they may understand the workings of machine translation, allowing them to make better use of the recent techniques in their own line of work and to eventually also contribute to research.

OpenNMT-tf was suggested as one of the most accessible and performant toolkits according to testing and benchmarking of several available open-source NMT toolkits.

In the final part of the thesis, OpenNMT-tf was used to create 5 different neural machine translation models, which were subsequently evaluated by translating several patent sentences. The models differed only in the data used for training, where the data of a single large text corpus (the NTC7 parallel sentence data of the *NTCIR 10 PatentMT Test Collection*) was filtered by domain using the international patent classification numbers. Two larger datasets, one which contained all domains (named ntc7) and the other which contained all domains except for "optics" (ntc7wo) were used with approx. 1.8 million parallel sentences and approx. 1.1 million sentences respectively. Since the dataset which only contained the optics data (ntc7o) had only around 684,700 sentences, two shortened versions of the above-mentioned datasets were created by randomly deleting lines from the corpus (ntc768k and ntc7wo68k). The aim was to find out how such a domain specialization might affect the neural translation models, observe how data size and variety influence neural machine translation models and whether the neural models really provide better translation with more data than with more specifically selected data. The translation output of the models was evaluated both according to common practice of MT research, automatically with the BLEU metric, and manually through human evaluation of a small sample of sentences.

It was shown that the data amount, variety and selection of data clearly influenced the output of the neural machine translation models. As expected, the model based on the largest dataset had the best BLEU score in all tests and also showed the most promising progress during training (model perplexity continued to get lower on validation dataset). However, the BLEU score did not converge well with the human evaluation, where, in fact, the domain specialized model (ntc7o) provided the best results.

This strengthens the belief, that the intervention by translators or Translation Studies experts might indeed be of great benefit to the MT research paradigm, as many of the common practices in the development and especially automatic evaluation of machine translation do not consider common practices in translation quality assurance (TQA). This also explains, why the automatic evaluation correlates poorly with the findings of the human evaluation. Additionally, solid theoretical frameworks for translation, like the *skopos theory*, are rarely considered for the modeling of translation architectures and data pre-selection steps, while this thesis has proven that data pre-selection can strongly influence the output of neural machine translation and in fact likely trim the models towards a specific *skopos*.

By providing this encompassing analysis of state-of-the-art machine translation in a very practical and hopefully easy to understand approach, this thesis hopes to lower the hesitation of Translation Studies scholars, as well as translators, to deeply engage with the topic and provide constructive research in regards to improving machine translation, evaluation of machine translation, integration of the translators into the workflow and also integration of key translational theories, into the modeling and data-selection of NMT research.

8.1 Summary in German

Ein Ziel dieser Masterarbeit war es, eine zugängliche, aber dennoch umfassende und tiefreichende Einführung in die maschinelle Übersetzung und spezifisch in die neuronale maschinelle Übersetzung anzubieten. Dies soll insbesondere Translationswissenschafts-studierenden und ÜbersetzerInnen ansprechen, die sich mit dem Thema tiefgehend auseinandersetzen möchten. Zu diesem Zweck wurde ein Überblick über die verschiedenen Architekturen der maschinellen Übersetzung (MT) gegeben, von der regel-basierten Übersetzung (RBMT) zur statistischen Übersetzung (SMT) bis hin zum momentanen Stand der Technik, der neuronalen maschinellen Übersetzung (NMT). Neben einer tiefergehenden theoretischen Auseinandersetzung mit der Funktionsweise von NMT bietet die Arbeit eine Schritt-für-Schritt-Anleitung zum Erstellen neuronaler Übersetzungsmodelle an, die dank open-source Software auf handelsüblichen PCs erstellt werden können. Dabei werden viele Konzepte aus der Computerlinguistik und generellen IT vorgestellt, die der/m LeserIn dabei helfen die Logik hinter maschineller Übersetzung zu verstehen und diese somit besser zum eigenen Vorteil zu verwenden und eventuell eigene wissenschaftliche Beiträge dazu zu verfassen.

Nach tieferer Auseinandersetzung mit einer Vielzahl von open-source NMT Toolkits und Testung dieser Toolkits, wurde OpenNMT-tf als eines der zugänglichsten und leistungsfähigsten Toolkits empfohlen. Im letzten Teil dieser Masterarbeit wurden mit Hilfe

von OpenNMT-tf fünf verschiedene neuronale Übersetzungsmodelle trainiert, welche folglich anhand der Übersetzung von Patentsätzen evaluiert wurden. Die Modelle unterscheiden sich lediglich in den Daten die zum Training (oder zur Erstellung) dieser genutzt wurden, wobei die Daten aus dem NTC7 Parallelsatzcorpus der *NTCIR 10 PatentMT Test Collection* stammen und anhand der internationalen Patentklassifikationsnummer nach Domäne sortiert wurden. Zwei größere Datensätze, einer der alle Domänen im Corpus enthält (ntc7, ca. 1,8 Mio. Satzpaare) und einer der alle Domänen außer der „Optik"-Domäne enthält (ntc7wo, ca. 1,1 Mio. Satzpaare), wurden erstellt. Zusätzlich wurden noch ein „Optik"-Datensatz (ntc7o) mit ca. 684.700 Parallelsätzen erstellt. Um die Größe der Datensätze einheitlich zu halten, wurden noch zwei weitere Datensätze (ntc768k und ntc7wo68k) aus den oben genannten, größeren Datensätzen erstellt, bei denen jeweils Sätze zufällig aus dem größeren Datensatz gelöscht wurden, um auf dieselbe Satzanzahl zu kommen wie beim „Optik"-Datensatz.

Das Ziel war es, herauszufinden wie sich diese Domänenspezialisierung auf die Übersetzung der neuronalen Übersetzungsmodelle auswirken würde und zu beobachten, wie Datensatzgröße und Datenvarietät die Modelle beeinflussen würde bzw. ob sich die Annahme bewahrheitet, dass Datenmenge wichtiger als Datenselektion sei. Die Ergebnisse der Übersetzung zweier Testcorpora aus Patentsätzen im NTC7 Parallelsatzcorpus, wurde folglich nach üblichem Vorgehen der MT-Forschung automatisch mit der BLEU-Metrik und zusätzlich anhand einer kleinen, zufälligen Auswahl durch manuelle, humane Evaluierung ausgewertet.

Die Ergebnisse der Evaluierung zeigen, dass die Menge der Daten, die Varietät der Daten und die Selektion der Daten den Output der neuronalen Modelle klar beeinflusst haben. Wie per allgemeiner Annahme in der MT-Forschung zu erwarten war, hatte das Modell mit dem größten Datenset (ntc7) die besten BLEU-Ergebnisse. Allerdings wurde auch gezeigt, dass die BLEU-Wertung sich kaum mit den Ergebnissen der humanen Evaluation deckte, wo tatsächlich das auf die Optikdomäne spezialisierte Modell (ntc7o) die besten Ergebnisse lieferte.

Dies bestärkt weiter die Annahme, dass ein Beitrag von ÜbersetzerInnen und der Übersetzungswissenschaft große Vorteile für die MT-Forschung haben kann. Insbesondere sollte eine engere Zusammenarbeit der verschiedenen Felder in der Auswertung des maschinellen Outputs in Erwägung gezogen werden und theoretische Grundlagen aus der Übersetzungswissenschaft (wie z.B. die *Skopostheorie*) tiefer in die MT-Systeme und/oder Trainingsvorkehrungen eingearbeitet werden.

Durch das Bereitstellen dieser praxisbezogenen und umfassenden Analyse des Stands der Technik der MT-Forschung und der zugänglichen Anleitung zum Erstellen solcher MT-Systeme, erhofft sich diese Masterarbeit, Studierenden, ÜbersetzungswissenschaftlerInnen und

ÜbersetzerInnen einen leicht verständlichen Einstieg in die Materie zu ermöglichen und weitere konstruktive Beiträge zur MT-Forschung zu erleichtern.

8.2 Discussion and outlook

Machine translation and especially neural machine translation is still a hot research topic at the time of writing this thesis and seemingly every day a vast number of new works regarding the topic is published. Since NMT is still relatively fresh, a lot of room is left for experimentation and many new interesting concepts appear at the horizon every day. For example, researchers at Google found it is possible to train parameters of a single model on several languages, by only slightly modifying the ST to reflect which TL is expected. This enables them to drastically reduce the need of model training, as a single model can be used to cover several languages[146] and also provide higher quality output on low-resource languages (i.e. languages where only a small amount of parallel texts is available for the specific language pair) (Johnson et al. 2017). An unexpected side effect of that effort was the discovery of so-called ***zero-shot translation***, which enables the network to translate between language pairs that it has not explicitly seen before in training. This suggests that the abstract representation through the trained parameters and word embeddings does in fact capture some form of semantics or meaning[147]. The function of the multi-lingual models and zero-shot translation hinges on an abstract representation of sentences at the tokenization step; i.e. the paper uses a tokenizer which is very similar to the BPE tokenization described in this thesis (it uses the so-called word-piece tokenizer).

Google goes as far as to call the abstract representation calculated by the network to be a hint of an "**interlingua**" that is being extracted by the network from the training datasets, an interesting idea that would harken back to the classical approach in MT. In a way, it might be argued that NMT comes closer to the ideal of what translation should be than its predecessor, as it appears to capture more than just the surface form of texts. The results of such a zero-shot translation are, however, reported to be generally significantly lower than translating through so-called "explicit bridging", i.e. through an intermediate language (for example, Japanese→English, English→Korean gives better results than Japanese→Korean). Yet, the zero-shot approach shows promise, as trained models can act as a baseline for incremental training by using comparatively small amounts of actual parallel data in a low resource language to enable translation into that specific language (Johnson et al. 2017:9).

[146] For example, instead of training two models for English→Japanese and English→ Korean, it is possible to train a single English→Japanese, Korean model. It was proven to work best for related languages.
[147] In fact, the paper also presents the idea of mixing the ST languages arbitrarily while still getting a proper translation result.

Going into the same direction of applying neural networks to capture the meaning of texts and generating a baseline model to fine-tune later, ***BERT*** or the ***Bidirectional Encoder Representation for Transformers*** recently made a big splash in the NLP research community (Devlin et al. 2018). BERT is a very large pre-trained language model based on the same Transformer architecture, that was also used for this thesis' experiment. The BERT language model considers bi-directional word context and can be used as a baseline for fine-tuning a model and specialize it on a specific task like Question-Answering, text-understanding, etc. Attempts to incorporate BERT into NMT are currently underway and appear to have promising results (see Zhu et al. 2020).

However, one of the biggest strengths of the neural approaches is also a disadvantage that can be observed throughout most of the research: The language understanding and translation process is generally fully automatic, and the models are "static" once they have been trained unless further "offline" training is performed. This renders NMT rather unapproachable to translators, as system transparency is low, and tuning is difficult to accomplish without in-depth knowledge of neural networks and deep-learning toolkits. Even then, results can sometimes be quite arbitrary, as was also seen in this thesis' experiment.

Rare exceptions like the ***Interactive NMT*** approach (Peris, Domingo, et al. 2017) show that the idea of NMT as tool for the translator, that adapts to the translator's choices rather than simply providing pre-determined solutions could help improving the workflow of translators, while at the same time improving the NMT model underneath. However, there is a strong need for research into the better application of NMT for post-editing by human translators and its integration within existing translation tools, as, for example, Daems & Macken (2019) show that while interactive NMT tools may provide suggestions with less errors than a similar SMT solution, little to no improvement can be observed in the actual translation time or effort made by the translators (measured in key-strokes and mouse-actions). Similar results were found by other studies, like Castilho et al. 2017, Jia et al. 2019a and Knowles et al. 2019. Like in this thesis, these studies also showed that while automatic evaluation results look very promising, human evaluation often showed mixed results, with NMT providing noticeable increases in fluency, but inconsistent results for adequacy and the post-editing effort.

This disparity between automatic evaluation and human evaluation often remains unmentioned in papers regarding NMT. Translators and Translation Studies might be able to contribute significantly in the development of NMT systems by both raising the awareness to this gap and by proposing theoretical frameworks that can be applied specifically to MT evaluation and that can also be modeled mathematically or realized in data preparation steps to

improve the translation systems, similar to what Kenny and Doherty already suggested for SMT in 2014 (Kenny & Doherty 2014).

Obviously, one might pose the question, whether such an effort is desirable from the translator's point of view. Wouldn't translators basically help with making themselves disposable by aiding in the creation of better machine translation systems? That argument could especially be made considering that many research papers on NMT explicitly seek the comparison to human translators as the gold standard, something that becomes quite clear when looking at how titles for NMT papers, backed by tech-giants Google and Microsoft, include statements like "bridging the gap between human and machine translation" (Wu et al. 2016) or "human parity on automatic Chinese to English news translation" (Hassan et al. 2018). It is sensational statements like these, that are quickly picked up by the media, which write and talk about the alleged universal uses of deep-learning and AI (i.e. neural networks) in a quite sensationalist way and see MT displacing the human translator in the near future. Especially financial magazines, like the *Wall Street Journal*, were quick to pick up the potential of flawless machine translation, by exclaiming that "The Language Barrier Is About to Fall" (Ross 2016), which in turn lead *The economist* to explain to its readers "Why translators have the blues" (Johnson 2017).

Ripplinger wonderfully explores these sentiments in her essay "Is this the end of the era of human translation?", finding that a "brave new world" for translation seems all but unavoidable, suggesting that "in a world, where the amalgamation of machine and human beings has already been realized in many fields, Translation Studies and translation practice need to reinvent themselves in order to survive and stay relevant" (Ripplinger 2020).

I can't help but agree with Ripplinger's statement, that Translation Studies and translation practice needs to adapt to these new changes in the industry and remain relevant by providing its expertise to enable a better implementation of the systems. An implementation that doesn't aim to compete with the human translator but aims to be a powerful tool for the future translator in the continuously growing translation market.

Bibliography

Agrawal, R., & Sharma, D. M. 2017. Experiments on Different Recurrent Neural Networks for English-Hindi Machine Translation. In: *Computer Science & Information Technology (CS & IT)* (pp. 63–74). Academy & Industry Research Collaboration Center (AIRCC). https://doi.org/10.5121/csit.2017.71006

Al-Onaizan, Y., Curin, J., Jahr, M., Knight, K., Lafferty, J., Melamed, D., … Yarowsky, D. 1999. Statistical Machine Translation - Final Report. *StatMT '08 Proceedings of the Third Workshop on Statistical Machine Translation*, 1–42. Retrieved from http://mt-archive.info/JHU-1999-AlOnaizan.pdf (Accessed on: February 26, 2019)

Alammar, J. 2018a. The Illustrated Transformer. Retrieved from https://jalammar.github.io/illustrated-transformer/ (Accessed on: February 1, 2020)

Alammar, J. 2018b. Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention). Retrieved from https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/ (Accessed on: February 1, 2020)

Bahdanau, D., Cho, K., & Bengio, Y. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15. Retrieved from http://arxiv.org/abs/1409.0473 (Accessed on: February 11, 2020)

Bawden, R. 2018. *Going beyond the sentence : Contextual Machine Translation of Dialogue*. Université Paris-Saclay. Retrieved from https://tel.archives-ouvertes.fr/tel-02004683 (Accessed on: March 18, 2020)

Bengio, Y., Simard, P., & Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*/2, 157–166. https://doi.org/10.1109/72.279181

Bentivogli, L., Bisazza, A., Cettolo, M., & Federico, M. 2016. Neural versus Phrase-Based Machine Translation Quality: a Case Study. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 257–267). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/D16-1025

Berger, A. L., Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., Gillett, J. R., Lafferty, J. D., … Ureš, L. 1994. The Candide system for machine translation. In: *Proceedings of the workshop on Human Language Technology - HLT '94* (p. 157). Morristown, NJ, USA: Association for Computational Linguistics. https://doi.org/10.3115/1075812.1075844

Biçici, M. E. 2011. *The Regression Model of Machine Translation*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.708.2672&rep=rep1&type=pdf (Accessed on: June 30, 2019)

Bird, S., Klein, E., & Loper, E. 2009. NLTK Book. *Natural Language Processing with Python*. O'Reilly Media Inc. Retrieved from http://www.nltk.org/book/ (Accessed on: September 14, 2019)

Bojar, O., Graham, Y., & Kamran, A. 2017. Results of the WMT17 Metrics Shared Task. In: *Proceedings of the Second Conference on Machine Translation* (pp. 489–513). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/W17-4755

Boleda, G. 2020. Distributional Semantics and Linguistic Theory. *Annual Review of Linguistics*, *6*/1, 213–234. https://doi.org/10.1146/annurev-linguistics-011619-030303

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., & Mercer, R. L. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, *19*/2, 263–311. Retrieved from https://www.aclweb.org/anthology/J93-2003/ (Accessed on: February 19, 2019)

Cadwell, P., O'Brien, S., & Teixeira, C. S. C. 2018. Resistance and accommodation: factors for the (non-) adoption of machine translation among professional translators. *Perspectives*, *26*/3, 301–321. https://doi.org/10.1080/0907676X.2017.1337210

Callison-Burch, C., Osborne, M., & Koehn, P. 2006. Re-evaluating the role of BLEU in machine translation research. In: *EACL 2006 - 11th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*. Retrieved from https://www.aclweb.org/anthology/E06-1032 (Accessed on: April 12, 2020)

Carbonell, J. G., Rich, E., Masaru, T., Johnson, D., Vasconcellos, M., & Wilks, Y. 1994. Machine Translation in Japan. Retrieved from http://www.wtec.org/loyola/ar93_94/mt.htm (Accessed on: February 20, 2019)

Castilho, S., Moorkens, J., Gaspari, F., Calixto, I., Tinsley, J., & Way, A. 2017. Is Neural Machine Translation the New State of the Art? *The Prague Bulletin of Mathematical Linguistics*, *108*/1, 109–120. https://doi.org/10.1515/pralin-2017-0013

Chiang, D., Knight, K., & Wang, W. 2009. 11,001 new features for statistical machine translation. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics on - NAACL '09* (pp. 218–226). Morristown, NJ, USA: Association for Computational Linguistics. https://doi.org/10.3115/1620754.1620786

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation* (pp. 103–111). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/v1/W14-4012

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1724–1734). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1179

Collins, M. 2003. Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, *29*/4, 589–637. https://doi.org/10.1162/089120103322753356

Collins, M. 2011. *Statistical Machine Translation: IBM Models 1 and 2*. Retrieved from http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/ibm12.pdf (Accessed on: April 11, 2019)

Costa-Jussà, M. R., & Fonollosa, J. A. R. 2007. Analysis of statistical and morphological classes to generate weighted reordering hypotheses on statistical machine translation systems. In: *Proceedings of the Second Workshop on Statistical Machine Translation* (pp. 171–176).

Dabre, R., Cromieres, F., Nakazawa, T., & Dabre, R. 2017. Neural Machine Translation: Basics, Practical Aspects and Recent Trends. In: *Proceedings of the IJCNLP 2017, Tutorial Abstracts* (pp. 11–13). Taipei, Taiwan: Asian Federation of Natural Language Processing. Retrieved from https://www.aclweb.org/anthology/I17-5004 (Accessed on: March 4, 2020)

Daems, J., & Macken, L. 2019. Interactive adaptive SMT versus interactive adaptive NMT: a user experience evaluation. *Machine Translation*, *33*/1–2, 117–134. https://doi.org/10.1007/s10590-019-09230-z

Damonte, M., & Cohen, S. B. 2018. Cross-lingual Abstract Meaning Representation Parsing. *Proceedings of NAACL-HLT 2018*, *1*/6, 1146–1155. Retrieved from http://arxiv.org/abs/1704.04539 (Accessed on: June 5, 2019)

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from http://arxiv.org/abs/1810.04805 (Accessed on: May 4, 2020)

Domingo, M., Garcıa-Martınez, M., Helle, A., Casacuberta, F., & Herranz, M. 2018. How Much Does Tokenization Affect Neural Machine Translation? Retrieved from http://arxiv.org/abs/1812.08621 (Accessed on: March 24, 2020)

Forcada, M. L. 2017. Making sense of neural machine translation. *Translation Spaces*, *6*/2, 291–309. https://doi.org/10.1075/ts.6.2.06for

Forcada, M. L., Ginestí-Rosell, M., Nordfalk, J., O'Regan, J., Ortiz-Rojas, S., Pérez-Ortiz, J. A., … Tyers, F. M. 2011. Apertium: a free/open-source platform for rule-based machine translation. *Machine Translation*, *25*/2, 127–144. https://doi.org/10.1007/s10590-011-9090-0

Galley, M., & Manning, C. D. 2008. A simple and effective hierarchical phrase reordering model. In EMNLP. *Association for Computational Linguistics*, 848–856. Retrieved from https://www.aclweb.org/anthology/D08-1089 (Accessed on: May 25, 2019)

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. 2017. Convolutional Sequence to Sequence Learning. *34th International Conference on Machine Learning, ICML 2017*, *3*, 2029–2042. Retrieved from http://arxiv.org/abs/1705.03122 (Accessed on: November 7, 2019)

Green, S., Wang, S., Cer, D., & Manning, C. 2013. Fast and Adaptive Online Training of Feature-Rich Translation Models. *51st Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, *1*, 311–321. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.2756&rep=rep1&type=pdf (Accessed on: February 26, 2019)

Harper Collins German Dictionary: German-English, English-German, Concise Edition. 1998. Glasgow: Harper Resource.

Hassan, H., Aue, A., Chen, C., Chowdhary, V., Clark, J., Federmann, C., … Zhou, M. 2018. Achieving Human Parity on Automatic Chinese to English News Translation. Retrieved from http://arxiv.org/abs/1803.05567 (Accessed on: January 4, 2020)

Hebb, D. 1950. *The organization of behavior: A neuropsychological theory*. New Your: John Wiley And Sons.

Hochreiter, J. 1991. *Untersuchungen zu dynamischen neuronalen Netzen*. *Diplomarbeit im Fach Informatik*. Technische Universität München. Retrieved from https://www.bioinf.jku.at/publications/older/3804.pdf (Accessed on: December 11, 2019)

Hochreiter, S., & Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, *9*/8, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hong, M., & Streiter, O. 1999. Overcoming the language barriers in the Web: The UNL-Approach. In: J. Gippert & P. Olivier (Eds.), *Multilinguale Corpora. Codierung, Strukturierung, Analyse. 11. Jahrestagung der Gesellschaft für Linguistische Datenverarbeitung (Frankfurt, 7.-10. Juli 1999).* (pp. 253–262). Prague: Enigma.

Hornik, K., Stinchcombe, M., & White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*/5, 359–366.

Huang, P.-S., Wang, C., Huang, S., Zhou, D., & Deng, L. 2017. Towards Neural Phrase-based Machine Translation. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, /2016, 1–22. Retrieved from http://arxiv.org/abs/1706.05565 (Accessed on: January 21, 2020)

Hutchins, W. J. 2004. The Georgetown-IBM Experiment Demonstrated in January 1954. In: R. E. Frederking & K. B. Taylor (Eds.), *Machine Translation: From Real Users to Research* (pp. 102–114). Washington: Springer. https://doi.org/10.1007/978-3-540-30194-3_12

Hutchins, W. J. 2010. Machine translation : a concise history. In: C. S. Wai (Ed.), *Journal of Translation Studies* (Vol. 13, pp. 29–70). Hong Kong: Chinese University of Hong Kong.

Hutchins, W. J., & Somers, H. L. 1992. *An Introduction To Machine Translation*. London: Academic Press. Retrieved from http://www.hutchinsweb.me.uk/IntroMT-TOC.htm (Accessed on: February 19, 2019)

Jia, Y., Carl, M., & Wang, X. 2019a. How does the post-editing of neural machine translation compare with from-scratch translation? A product and process study. *Journal of Specialised Translation*, /31, 60–86.

Jia, Y., Carl, M., & Wang, X. 2019b. Post-editing neural machine translation versus phrase-based machine translation for English–Chinese. *Machine Translation*, *33*/1–2, 9–29. https://doi.org/10.1007/s10590-019-09229-6

Johnson. 2017. Why translators have the blues. *The Economist*. Retrieved from https://www.economist.com/books-and-arts/2017/05/27/why-translators-have-the-blues (Accessed on: May 7, 2019)

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., … Dean, J. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Transactions of the Association for Computational Linguistics*, *5*, 339–351. https://doi.org/10.1162/tacl_a_00065

Junczys-Dowmunt, M., Dwojak, T., & Hoang, H. 2016. Is Neural Machine Translation Ready for Deployment? A Case Study on 30 Translation Directions. Retrieved from http://arxiv.org/abs/1610.01108 (Accessed on: March 27, 2020)

Kaiser-Cooke, M. 1993. *Machine Translation and the human Factor*. Doctoral thesis, University of Vienna.

Kenny, D., & Doherty, S. 2014. Statistical machine translation in the translation curriculum: overcoming obstacles and empowering translators. *The Interpreter and Translator Trainer*, *8*/2, 276–294. https://doi.org/10.1080/1750399X.2014.936112

Klein, G., Kim, Y., Deng, Y., Senellart, J., & Rush, A. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In: *Proceedings of ACL 2017, System Demonstrations* (pp. 67–72). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/P17-4012

Knight, K. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, *25*/4, 606–615. Retrieved from https://www.aclweb.org/anthology/J99-4005 (Accessed on: January 21, 2020)

Knowles, R., Sanchez-Torron, M., & Koehn, P. 2019. A user study of neural interactive translation prediction. *Machine Translation*, *33*/1–2, 135–154. https://doi.org/10.1007/s10590-019-09235-8

Koehn, P. 2010. *Statistical Machine Translation*. Cambridge: Cambridge University Press.

Koehn, P. 2017. Neural Machine Translation. Retrieved from http://arxiv.org/abs/1709.07809 (Accessed on: February 19, 2019)

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C. Federico, M., Bertoldi, N., Cowan, B., … Constantin, A. Herbst, E. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. *ACL*. Retrieved from https://www.aclweb.org/anthology/P07-2045 (Accessed on: February 26, 2019)

Koehn, P., & Knowles, R. 2017. Six Challenges for Neural Machine Translation. In: *Proceedings of the First Workshop on Neural Machine Translation* (pp. 28–39). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/W17-3204

Koehn, P., Och, F. J., & Marcu, D. 2003. Statistical Phrase-Based Translation. *Proceedings of HLT-NAACL 2003*, *May-June*, 48–54. Retrieved from https://www.aclweb.org/anthology/J99-4005 (Accessed on: February 19, 2019)

Kreutzer, J., Bastings, J., & Riezler, S. 2019. Joey NMT: A Minimalist NMT Toolkit for Novices. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations* (pp. 109–114). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-3019

Kudo, T., & Richardson, J. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. Retrieved from http://arxiv.org/abs/1808.06226 (Accessed on: February 19, 2019)

Kurenkov, A. 2015. A brief history of neural nets and deep learning. Retrieved from https://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/ (Accessed on: June 30, 2019)

Lindquist, H., Reiss, K., & Vermeer, H. J. 1985. Grundlegung einer allgemeinen Translationstheorie. *Language*, *61*/3, 737. https://doi.org/10.2307/414441

Liu, H. 2017. *A Study on Translation Quality Management of Automotive Information Based on SAE J2450 Translation Quality Metric*. Doctoral thesis, University of Vienna.

Lo Presti, R. 2016. *Menschliche und automatische Evaluation von Übersetzungen von Fachtexten in Google Translate*. Master's thesis, University of Vienna.

Luong, M.-T. 2016. *Neural Machine Translation - A Dissertation*. Standford University. Retrieved from https://github.com/lmthang/thesis (Accessed on: February 19, 2019)

Luong, M.-T., Brevdo, E., & Zhao, R. 2017. Neural Machine Translation (seq2seq) Tutorial. Retrieved from https://github.com/tensorflow/nmt (Accessed on: March 24, 2020)

Luong, T., Pham, H., & Manning, C. D. 2015. Effective Approaches to Attention-based Neural Machine Translation. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412–1421). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1166

Marcu, D., & Wong, W. 2002. A phrase-based, joint probability model for statistical machine translation. In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - EMNLP '02* (Vol. 10, pp. 133–139). Morristown, NJ, USA: Association for Computational Linguistics. https://doi.org/10.3115/1118693.1118711

Mariño, J. B., Banchs, R. E., Crego, J. M., de Gispert, A., Lambert, P., Fonollosa, J. A. R., & Costa-jussà, M. R. 2006. N -gram-based Machine Translation. *Computational Linguistics*, *32*/4, 527–549. https://doi.org/10.1162/coli.2006.32.4.527

Martins, R. 2010. UNL About. Retrieved from http://www.unlweb.net/unlweb/index.php?view=article&catid=54%3Aunlweb&id=58%3Aunl&format=pdf&option=com_content (Accessed on: May 23, 2019)

Mateo, R. M. 2014. A deeper look into metrics for translation quality assessment (TQA): A case study. *Miscelánea: A Journal of English and American Studies*, *49*, 73–94.

McCulloch, W. S., & Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*/4, 115–133. https://doi.org/10.1007/BF02478259

McGonagle, J., Alonso García, J., & Saruque, M. 2020. Feedforward Neural Networks. Retrieved from https://brilliant.org/wiki/feedforward-neural-networks/ (Accessed on: November 15, 2019)

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., … Wu, H. 2017. Mixed Precision Training. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*. Retrieved from http://arxiv.org/abs/1710.03740 (Accessed on: March 31, 2020)

Moorkens, J. 2018. What to expect from Neural Machine Translation: a practical in-class translation evaluation exercise. *The Interpreter and Translator Trainer*, *12*/4, 375–387. https://doi.org/10.1080/1750399X.2018.1501639

Moreda, P., Suárez, A., Lloret, E., Saquete, E., & Moreno, I. 2018. From sentences to documents: Extending abstract meaning representation for understanding documents. *Procesamiento de Lenguaje Natural*, *60*, 61–68. https://doi.org/10.26342/2018-60-7

Nagao, M. 2003. A Framework of a Mechanical Translation between Japanese and English by Analogy Principle. In: *Readings in Machine Translation*. The MIT Press. https://doi.org/10.7551/mitpress/5779.003.0038

Ney, H., J. Och, F., & Vogel, S. 2000. Statistical Translation Of Spoken Dialogues In The Verbmobil System. Retrieved from https://www.researchgate.net/publication/2440700_Statistical_Translation_Of_Spoken_Dialogues_In_The_Verbmobil_System (Accessed on: February 26, 2019)

Nicholson, C. 2019. A Beginner's Guide to LSTMs and Recurrent Neural Networks. Retrieved from https://pathmind.com/wiki/lstm#recurrent (Accessed on: January 13, 2020)

Och, F. J., & Ney, H. 2000. Improved statistical alignment models. In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics - ACL '00* (pp. 440–447). Morristown, NJ, USA: Association for Computational Linguistics. https://doi.org/10.3115/1075218.1075274

Och, F. J., & Ney, H. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, *29*/1, 19–51. https://doi.org/10.1162/089120103321337421

Okuda, M. 2015. Description matters of the specification [Blog post]. Retrieved from https://www.interbooks.co.jp/column/jpatent/20150219/ (Accessed on: February 11, 2020)

Papineni, K., Roukos, S., Ward, T., & Zhu, W. 2001. BLEU: a method for automatic evaluation of machine translation. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02* (p. 311). Morristown, NJ, USA: Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135

Patent JP05088550A. 1993. Japan: MURATA MACH LTD.

Patent JP08062939A. 1996. Japan: Fuji Xerox Co., Ltd.

Peris, Á., & Casacuberta, F. 2018. NMT-Keras: a Very Flexible Toolkit with a Focus on Interactive NMT and Online Learning. *The Prague Bulletin of Mathematical Linguistics*, *111*/1, 113–124. https://doi.org/10.2478/pralin-2018-0010

Peris, Á., Cebrián, L., & Casacuberta, F. 2017. Online Learning for Neural Machine Translation Post-editing, 1–12. Retrieved from http://arxiv.org/abs/1706.03196

Peris, Á., Domingo, M., & Casacuberta, F. 2017. Interactive neural machine translation. *Computer Speech & Language*, *45*, 201–220. https://doi.org/10.1016/j.csl.2016.12.003

Popa, L. 2008. Machine translation: A survey. *Professional Communication and Translation Studies*, 151–158.

Popel, M., & Bojar, O. 2018. Training Tips for the Transformer Model. *The Prague Bulletin of Mathematical Linguistics*, *110*/1, 43–70. https://doi.org/10.2478/pralin-2018-0002

Pouget-Abadie, J., Bahdanau, D., van Merrienboer, B., Cho, K., & Bengio, Y. 2014. Overcoming the Curse of Sentence Length for Neural Machine Translation using Automatic Segmentation. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation* (pp. 78–85). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.3115/v1/W14-4009

Quirk, C., & Menezes, A. 2006. Do We Need Phrases? Challenging The Conventional Wisdom In Statistical Machine Translation. In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference* (Proceeding, pp. 9–16). ACL/SIGPARSE. Retrieved from https://www.microsoft.com/en-us/research/publication/do-we-need-phrases-challenging-the-conventional-wisdom-in-statistical-machine-translation/ (Accessed on: November 27, 2019)

Ripplinger, M. C. 2020. Is this the end of the era of human translation? Thoughts of a human translator at this curious time. In: M. En (Ed.), *Truths, trust and translation (to be released in 2020).* Peter Lang.

Rosenblatt, F. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*/6, 386–408. https://doi.org/10.1037/h0042519

Ross, A. 2016. The Language Barrier Is About to Fall. *The Wall Street Journal.* Retrieved from https://www.wsj.com/articles/the-language-barrier-is-about-to-fall-1454077968 (Accessed on: May 7, 2020)

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature*, *323*/6088, 533–536. https://doi.org/10.1038/323533a0

Sánchez-Gijón, P., Moorkens, J., & Way, A. 2019. Post-editing neural machine translation versus translation memory segments. *Machine Translation*, *33*/1–2, 31–59. https://doi.org/10.1007/s10590-019-09232-x

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1715–1725). Stroudsburg, PA, USA: Association for Computational Linguistics. https://doi.org/10.18653/v1/P16-1162

Serrano, L. 2016. *Linear Regression Answer [Video]*. Udacity. Retrieved from https://www.youtube.com/watch?v=L5QBqYDNJn0 (Accessed on: September 3, 2019)

Sokolov, A. 2015. Noisy Channel model Phrase-based SMT. Retrieved from https://www.cl.uni-heidelberg.de/courses/ss15/smt/scribe4.pdf (Accessed on: April 3, 2019)

Sutskever, I., Vinyals, O., & Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR*, *abs/1409.3*/January, 3104–3112. Retrieved from http://arxiv.org/abs/1409.3215 (Accessed on: May 25, 2019)

Tyers, F. M. 2013. *Feasible lexical selection for rule-based machine translation/Selecció lèxica factible per a la traducció automàtica basada en regles*. Universidad de Alicante. Retrieved from https://rua.ua.es/dspace/bitstream/10045/35848/1/thesis_FrancisMTyers.pdf (Accessed on: February 19, 2019)

Uchiyama, M., & Isahara, H. 2007. A Japanese-English patent parallel corpus. In: *Proceedings of MT summit XI*.

Van Bui, V., Tran, T. T., Nguyen, N. B. T., Pham, T. D., Le, A. N., & Le, C. A. 2015. Improving Word Alignment Through Morphological Analysis. In: *Integrated Uncertainty in Knowledge Modelling and Decision Making* (pp. 315–325). https://doi.org/10.1007/978-3-319-25135-6_30

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. 2017. Attention Is All You Need. Retrieved from http://arxiv.org/abs/1706.03762 (Accessed on: February 19, 2019)

Weaver, W. 1949. Translation [Weaver Letter]. Retrieved from http://www.mt-archive.info/Weaver-1949.pdf (Accessed on: February 20, 2019)

Widrow, B. 1960. Adaptive “adaline” Neuron Using Chemical “memistors.”. Stanford: Stanford University.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., … Dean, J. 2016. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 1–23. Retrieved from http://arxiv.org/abs/1609.08144 (Accessed on: February 19, 2019)

Yan, X., & Su, X. G. 2009. *Linear Regression Analysis*. World Scientific. https://doi.org/10.1142/6986

Zhu, J., Xia, Y., Wu, L., He, D., Qin, T., Zhou, W., … Liu, T.-Y. 2020. Incorporating BERT into Neural Machine Translation, 1–16. Retrieved from http://arxiv.org/abs/2002.06823 (Accessed on: May 4, 2019)

I. Appendix I: SAE J2450 evaluation

According to the expanded SAE J2450 by Hui Liu (Liu 2017), there are 9 different error types, However, especially important for evaluating neural machine translation, I would like to specify another error-type: The "arbitrary addition" (AD). This will be added as a 10th error type to the evaluation metric and treated similarly to an omission (OM). The error definitions are as follows:

- **Wrong term (WT)**:
  1) a term that denotes a concept in the TL which is obviously different from the concept denoted by the SL term
  2) a term that is not consistent with other translations of the SL term in the same document or type of document only when the context for the source language term can justify the use of a different target language term (e.g. due to ambiguity of the source language term)
  3) a term that is in clear conflict with the present standard translation(s) of the SL term in the automotive field
  4) a term that is in violation with a client term glossary
- **Syntactic error (SE):**
  1) the target language words are correct, but the linear order based on the syntactic rules of TL is wrong
  2) the TT contains an incorrect phrase structure
  3) a source term which is assigned a wrong part of speech in its TL counterpart
- **Omission (OM):**
  1) a graphic with ST has been removed from the TL deliverable
  2) a continuous block of text in SL which has no counterpart in TL text and therefore the semantics of ST is lost in the translation
     - At the same time, it is noteworthy that omission does not mean that the source and target language words should be in correspondence.
- **Word structure or agreement error (SA):**
  1) an error of incorrect word structure occurred if an otherwise correct target language word (or term) is expressed in an incorrect morphological form (e.g. tense, case, number, gender, prefix, suffix, infix or any other inflections)
  2) a mistake related with agreement, which occurred when two or more TL words disagree in any form of inflection as would be required according to the grammatical rules of that language.
- **Misspelling (SP):**
  1) a term in the TL violates the spelling as already stated in a client glossary
  2) a term in the TL violates the accepted norms of spelling in the TL
  3) a term in the TL is written in an incorrect or inappropriate writing system for the TL

❖ **Punctuation (PE):**

1) the TL text contains an error according to the punctuation rules of that language
2) Missing decimal points or commas are also regarded as punctuation errors

❖ **Miscellaneous error (ME):**

1) a linguistic error related to the TL text but cannot be clearly attributed to any other error categories

❖ **Wrong Meaning (WM):**

1) the meaning of TT varies greatly from that of ST.

❖ **Wrong Style (WS):**

1) there is a deviation or violation from the Style guide required in TT
2) phraseology of TT is not idiomatic
3) the construction of sentences is cumbersome or clumsy
4) the translation is only a literal one.

❖ **Arbitrary Addition (AD):**

1) An incorrect and/or arbitrary term is added in the TT, which changes the meaning of the ST
2) An assumption is made about word relations within the TT sentence, that is not explicitly specified by the ST and may be wrong

Different error types are weighted differently depending on severity of the error. Weights for the ten error categories are:

- 5 (a serious error) or 2 (a minor error) for a wrong term (WT)
- 4 (a serious error) or 2 (a minor error) for a syntactic error (SE)
- 4 (a serious error) or 2 (a minor error) for an omission (OM)
- 4 (a serious error) or 2 (a minor error) for an incorrect addition (AD)
- 4 (a serious error) or 2 (a minor error) for a word structure or agreement error (SA)
- 3 (a serious error) or 1 (a minor error) for a misspelling (SP)
- 2 (a serious error) or 1 (a minor error) for a punctuation error (PE)
- 3 (a serious error) or 1 (a minor error) for a miscellaneous error (ME)
- 5 (a serious error) or 2 (a minor error) for wrong meaning (WM)
- No scores are given to style errors, as they are often argued to be very subjective (WS)

The overall weighted document score (OWDS) is calculated as the total weighted score divided by the number of words of all tested sentences. As per convention, each character is counted as a word in Japanese. The error count for the mixed-domain, optics and all sentences combined is shown in Table 18, Table 19 and Table 20 respectively.

Table 18: Extended SAE J2450 based quantitative evaluation of mixed-domain sentences in Section 7.6.2

	WT		SE		OM		AD		SA		SP		PE		ME		WM		WS		Total Score	OWDS
Weights (Serious/ Minor)	5	2	4	2	4	2	4	2	4	2	3	1	2	1	3	1	5	2	0	0		
	Number of Errors for Each Category																				Words: 196	
TT	1	2	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	16	0.03782506
ntc7	0	1	0	0	0	1	2	0	0	0	0	0	0	0	0	1	1	1	0	2	20	0.04728132
ntc7 120k	1	1	0	0	0	1	2	0	0	0	0	0	0	0	0	1	1	1	0	2	25	0.05910165
ntc768k	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	1	0	1	14	0.03309693
ntc768k 80k	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	11	0.02600473
ntc7o	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	7	0.01654846
ntc7o 80k	0	4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	2	16	0.03309693
ntc7wo	0	2	1	0	0	1	1	0	0	0	0	0	0	0	0	3	0	2	0	1	21	0.04964539
ntc7wo68k	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	0	2	0	1	17	0.04018913
ntc7wo68k 90k	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	11	0.02600473

Table 19: Extended SAE J2450 based quantitative evaluation of optics sentences in Section 7.6.2

	WT		SE		OM		AD		SA		SP		PE		ME		WM		WS		Total Score	OWDS
Weights (Serious/ Minor)	5	2	4	2	4	2	4	2	4	2	3	1	2	1	3	1	5	2	0	0		
	Number of Errors for Each Category																				Words: 227	
TT	0	0	0	2	0	1	0	1	0	3	0	0	0	0	0	0	0	0	0	2	14	0.06167401
ntc7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0.00881057
ntc7 120k	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0.03964758
ntc768k	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0.02202643
ntc768k 80k	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0.00881057
ntc7o	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0.01762115
ntc7o 80k	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	13	0.04405286
ntc7wo	2	1	0	3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	20	0.08810573
ntc7wo68k	1	1	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	15	0.0660793
ntc7wo68k 90k	1	1	0	3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	16	0.07048458

Table 20: Extended SAE J2450 based quantitative evaluation of all test sentences in Section 7.6.2.

	WT		SE		OM		AD		SA		SP		PE		ME		WM		WS		Weighted Score	OWDS
Weights (Serious/ Minor)	5	2	4	2	4	2	4	2	4	2	3	1	2	1	3	1	5	2	0	0		
	Number of Errors for Each Category																				Words: 423	
TT	1	2	0	2	0	1	1	1	0	3	0	0	0	0	0	1	0	1	0	2	30	0.07092199
ntc7	0	1	0	0	0	2	2	0	0	0	0	0	0	0	0	1	1	1	0	2	22	0.05200946
ntc7 120k	2	2	0	0	0	2	2	0	0	0	0	0	0	0	0	1	1	1	0	2	34	0.08037825
ntc768k	1	2	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	1	0	1	19	0.04491726
ntc768k 80k	0	2	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	13	0.03073286
ntc7o	0	4	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	2	11	0.02600473
ntc7o 80k	0	5	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	2	0	2	29	0.05673759
ntc7wo	2	3	1	3	0	2	1	0	0	0	0	0	0	0	0	3	0	2	1	1	41	0.09692671
ntc7wo68k	1	2	1	3	0	1	1	1	0	0	0	0	0	0	0	1	0	2	0	1	32	0.07565012
ntc7wo68k 90k	1	2	0	3	0	1	1	1	0	0	0	0	0	0	0	2	0	1	0	0	27	0.06382979

II. Appendix II: Code and scripts used in this thesis

For the convenience of the reader, this appendix provides all of the larger scripts used by the author in this thesis. They may be used as is or taken as inspiration for how to tackle similar problems when preparing data for model training.

Scrip1: Bash-script for file selection through the *find* command and subsequent code page conversion with *convert_encoding.py*

```
#!/bin/bash
find ./1993A -type f -name "*.txt" -exec \
python2 convert_encoding.py -r -o \#.utf8 -f euc_jp -t utf_8 {} +
```

Script 2: Python-script for splitting text into separate output files (split occurs at "|||")

```
#!/usr/bin/python
import sys
with open(sys.argv[1], encoding="utf8") as f:
    columns = zip(*(l.split("|||") for l in f))

langs = ('SSR', 'DOCID', 'TID', 'jp', 'en')
for lang, data in zip(langs, columns):
    with open('output.' + lang, 'w', encoding='utf8') as f:
        f.writelines(line.strip("\n") + '\n' for line in data)
```

Script 3: Bash-script for tokenization of EN and JP text using Moses and MeCab

```
#!/bin/bash
# Select working Folder, Dataset, Variant and Languages here
workdir="$(pwd)"
dataset="ntc7"
variant=""
src="jp"
tgt="en"

# English / Latin language tokenization
for l in en; do for f in $dataset$variant/*.$l; \
do perl tools/tokenizer.perl -a -no-escape -l $l -q < $f > $f.tk; done; done

# Japanese tokenization
for l in jp; do for f in $dataset$variant/*.$l; \
do mecab -O wakati -o $f.tk $f; done; done
```

Script 4: Python-script to extract the absolute path of files that contain domain classifier

```
#!/usr/bin/python
import glob
#Define Domains
domains = ('G01', 'G02', 'G03','G06')
#Search for Domains and extract absolute path of file to outputfile
outputfile = "step1out.txt"
with open(outputfile,'w') as f:
    for filename in glob.iglob('./' + '**/*.TXT', recursive=True):
        if any(x in open(filename, encoding='utf-8').read() for x in domains):
            print (filename, file=f)
```

Script 5: Python-script for creating an id-list from "training-ids.txt" and Script 4 output

```
#!/usr/bin/python
#Pathlist to find ID
paths = [line.rstrip("\n") for line in open("step1out.txt")]
#Remove "./" from paths
spath = [s.strip("./") for s in paths]

outputfile = "step2out.txt"
with open(outputfile,'w') as f:
        for line in open("other/ntc8-patmt-train/ntc7/train/training-ids.txt"):
                if any(x in line for x in spath):
                        print(line, end='', file=f)
```

Script 6: Python-script for only keeping DOCID before the first space on each line

```
#!/usr/bin/python
#Keep only DOCIDs before first space
l = []
outputfile = "idlist.txt"
with open(outputfile,'w') as f:
  for line in open("IDs-Optics.txt"):
     if line.strip():
        l.append(line.split()[0][1:])
  l = '\n'.join(l)
  print(l, file=f)
```

Script 7: Bash-script for creating list to randomly delete lines from parallel sentence files

```
#!/bin/bash
filename=inputfile.txt
number=429186

line_count="$(wc -l < "$filename")"
line_nums_to_delete="$(shuf -i "1-$line_count" -n $number)"
printf '%d\n' $line_nums_to_delete > delete.lines
```

Abstract

This work strives to be an easy to understand overview of how the current state-of-the-art in machine translation (MT), neural machine translation (NMT), works. Using the example of patent translation, the thesis aims to both demystify the terms "AI" and "deep-learning", that are often associated with NMT, and aims to provide an accessible guide for translators and Translation Studies scholars to work with, create and understand their own NMT models.

A theoretical foundation to MT is provided on which the work presents the creation and evaluation of five *Transformer* NMT models to determine the impact of data selection before model training. For this purpose, the five models were trained on five different patent datasets sorted by domain using the International Patent Classification: A mixed dataset, an optics dataset, a dataset containing all domains but optics and two smaller versions of the mixed and optics-free dataset.

It was found that the network's performance varied noticeably depending on how much and which data was used for training. While the common conception that more data equals better results held true in the automatic evaluation, it was shown that the domain specific training can help with improving results in the human evaluation, even when using less data. In fact, a large discrepancy between the automatic evaluation (*BLEU* metric) and the human evaluation (extended *SAE J2450* metric) could be observed, with the worst performing model in the automatic metric having the best results in the human evaluation. The analysis of the NMT output with reference to the source text also highlights several issues that post-editors would have to contend with when post-editing NMT generated texts.

Abstract auf Deutsch

Diese Arbeit soll einen leicht verständlichen Überblick darüber geben, wie maschinelle Übersetzung (MT) und insbesondere die neuronale maschinelle Übersetzung (NMT) funktioniert. Am Beispiel der Patentübersetzung soll die Arbeit sowohl die Begriffe „KI“ als auch „Deep Learning“, die häufig mit NMT assoziiert werden, entmystifizieren und einen zugänglichen Leitfaden für ÜbersetzerInnen und ÜbersetzungswissenschaftlerInnen bereitstellen, mit dem sie ihre eigenen NMT-Modell erstellen können, diese verstehen und damit arbeiten können.

Es wird eine theoretische Grundlage für MT bereitgestellt, auf Basis derer die Erstellung und Bewertung von fünf *Transformer* NMT-Modellen vorgestellt wird, um die Auswirkung der Datenauswahl vor dem Modelltraining zu bestimmen. Zu diesem Zweck wurden die fünf Modelle auf fünf verschiedene Patentdatensätze trainiert, die über die Internationale Patent Klassifizierung nach Domänen sortiert wurden: Ein gemischter Datensatz, ein Optikdatensatz, ein Datensatz, der alle Domänen außer Optik enthält, und zwei kleinere Versionen des gemischten und optikfreien Datensatzes.

Es wurde festgestellt, dass die Leistung des Netzwerks je nachdem, wie viel und welche Daten für das Training verwendet wurden, erheblich schwankte. Während die gängige Auffassung, dass mehr Daten zu besseren Ergebnissen führen, bei der automatischen Auswertung zutrifft, wurde gezeigt, dass das domänenspezifische Training dazu beitragen kann, die Ergebnisse bei der menschlichen Auswertung zu verbessern, selbst wenn weniger Daten verwendet werden. Tatsächlich konnte eine große Diskrepanz zwischen der automatischen Bewertung (*BLEU*-Metrik) und der menschlichen Bewertung (erweiterte *SAE J2450*-Metrik) beobachtet werden, wobei das Modell mit der schlechtesten Leistung in der automatischen Metrik die besten Ergebnisse bei der menschlichen Bewertung erzielte. Die Analyse des NMT Outputs unter Bezugnahme auf den Quelltext hebt auch einige der Probleme hervor, mit denen sich Post-EditorInnen bei der Nachbearbeitung von NMT-generierten Texten auseinandersetzen werden müssen.