UNIVERSITY OF LEIPZIG

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Neural Machine Translation with Transformers - Leveraging the Pivot Technique for Low-Resource Language Pairs

Johannes Michael Tauscher

Master's Thesis

in Data Science

Supervisor: Prof. Dr. Andreas Maletti

Submission: Leipzig, September 18, 2024

Acknowledgments

I would like to express my gratitude to Yasmin Moslem (Senior NLP Researcher at Bering Lab, Dublin), who not only provided illustrative examples for low-resource machine translation on her blog but also offered invaluable assistance to a newcomer on the OpenNMT platform and forum. Ms. Moslem addressed multiple issues I encountered, which I was able to overcome with her help. I am grateful for the welcoming and helpful online community in the field of neural machine translation. On top of that, I want to point out the patience of my supervising Professor Andreas Maletti, whom I have to thank for guiding me through the thesis writing process and the vast amount of computational resources on his machine learning cluster, which I occupied during the writing of my thesis. In terms of technical issues, I am grateful for both Ms. Yasmin Moslem's and Professor Andreas Maletti's patience and guidance. Furthermore, I want to thank my family, first and foremost my parents Ulrike and Detlef Tauscher, for their ongoing and unwavering support of my studies. My sister, Dr. Gabriele Tauscher, deserves special mention for her valuable criticism and feedback on my scientific endeavors, always finding time to offer guidance despite her busy schedule. Finally, I want to acknowledge Anna Koehne and Simon Jakob, who I currently share a flat with. Their support in providing a stress-free working environment at home has been invaluable. I appreciate their hospitality and understanding, especially during times when my research demanded most of my attention.

Abstract

Neural machine translation (NMT) has been a cornerstone in the development of large language models, which have recently gained significant attention in the domain of artificial intelligence. This thesis explores the effectiveness of pivot-based approaches in low-resource NMT scenarios, with a focus on the French-German language pair using English as a pivot language. We investigate the relationship between the amount of pivot resources and translation quality, utilizing the transformer architecture and the ONMT-Library. Our experiment demonstrates that increasing the size of source-pivot and pivot-target corpora consistently improves translation quality. The best-performing pivot model, trained on 400,000 pivot sentences, showed substantial improvements over the baseline model, with increases of 15.4% in BLEU, 5.8% in chrF, and -4.7% in TER scores. Notably, this model achieved $70 - 80\%$ of the upper-bound model's performance while in comparison using only 1% of direct source-target sentence pairs and 4% of additional pivot sentence pairs. Visual analysis of the training process revealed stable learning dynamics, with some anomalies in validation accuracy and early stopping mechanisms. The results suggest a clear relation between pivot resource availability and translation quality, with a saturation point emerging as the pivot corpus size increases. Additionally, we observed a performance gap between in-domain and out-of-domain test sets, highlighting the importance of both, domain-specific data in achieving optimal translation results and addressing robustness between in-domain and out-of-domain testing scenarios. This research contributes to the growing body of knowledge on low-resource NMT and provides insights into the efficient use of pivot strategies. The findings hold significance for future studies in NMT and the extensive domain of large language models, providing a fundamental basis for comprehending the core principles of these sophisticated A.I. systems.

Keywords: Neural Machine Translation, Machine Translation, Low-Resource Languages, Pivot-Based Translation, Transformer Architecture, OpenNMT-Library, French-German Translation, English Pivot Language, Large Language Models, Artificial Neural Networks

Contents

1 Introduction

LLaMA - The Impact Of A Leak On the 24^{th} of February 2023 Meta (formerly Facebook) released its latest LLaMA[1] foundational 65-billion-parameter large language model. Like other large language models, LLaMA works by taking a sequence of words as input and predicting the next word to generate text, recursively. The model whose architecture is based on artificial neural networks, introduced in Section 3.1, had previously been trained on 1.4 trillion tokens [2]. The calculation of the model parameters, an optimization approach originating in machine learning which we introduce in Section 3.1.3, was carried out on approximately 2000 GPUs (Nvidia A100) and took 21 days, which, at a data center renting price tag of around $4 per hour, cost approximately $4 million [3]. The release of LLaMA occured in response to OpenAI's ChatGPT, which was released to the public three months earlier in November 2022, putting pressure on major technology companies such as Google and Meta in the race for market dominance in the "Artificial Intelligence (AI) sector" [4]. Due to the high upfront research and development investment, companies have corresponding financial interests. By granting access to Application Programming Interfaces (APIs) for developers or subscription-based web services accessed through a user interface, companies financially leverage the power of their "AI" models and computational resources. Thus, even with Meta's prior notion and intention of supporting the Open Source community by publishing all kind of advances in the language model and "A.I" domain research, only the source code and model architecture, the transformer architecture which we are introducing in Section 3.2.3, but not the trained weights, i.e., the trained coefficients, for the model are published to the Open Source community by Meta. Without a comparable amount of computational and financial resources, open-sourcing the blueprints for their advances has limited impact. Yet, only a week after LLaMA's release, its weights were disclosed without authorization (i.e., leaked) on 4chan[5], the Internet's most trafficked imageboard known for its influential yet controversial role in internet culture. It is still up for debate whether Meta strategically published their weights to generate a user and developer platform surrounding its model's architecture, whether publishing the weights was an accident, or whether it was done intentionally but unauthorized by an individual. A month later a supposedly also

[1] acronym: Large Language Model Meta AI

[2] a token, i.e., in the domain of language processing the basic unit of text a model processes

[3] https://cloud.google.com/products/calculator/

[4] the quotation marks around "AI" indicate skepticism by the author towards the designation of the term by the mathematics and computer science domain as the "AI" term is blurry, of philosophical nature, and not defined but yet far spread and common in practice

[5] `https://4chan.org/`

leaked internal document (Patel, 2023) by a Google employee titled "We Have No Moat, And Neither Does OpenAI" describes the timeline of these events and what follows after:

> "From this point forward, innovations come hard and fast. (*a week later*) Language models on a Toaster. (*another week later*) Fine Tuning on a Laptop (...) within hours on a single RTX 4090." (Patel, 2023, March 3^{rd}-13^{th}, 2023)

Patel continues to describe how the leak has and will forever have changed the language model field of research as a whole, now that the costs for training compatible large language models are no more than a few hundred dollars instead of a few million. The saved resources by not having to estimate and train coefficients for a large foundational model but using the previously trained (*abbr.:* pre-trained) LLaMA and its existing coefficients for specific experiments from within the research and Open Source community amounts to that much. All kinds of research and experiments can now be undertaken by the many and not only by a few big, private corporations, e.g., fine-tuning the pre-trained foundational model on domain specific data while exploring efficient fine-tuning strategies. Within just 10 days after the leak, the University of Stanford published Alpaca, an instruction understanding and answering model, and more importantly, a framework for further development and experimentation, significantly impacting the Open Source community (Taori & Gulrajani, 2023). The academic and public interest is high and ever rising, as indicated in Figure 1. The previous developments, which are highlighted by giving a general and historical insight to the Machine Translation domain in Chapter 2, in the field already spark the interest of many researchers in the field. As Figure 1 shows, the number of papers published on arxiv.org[6] from year 2000 to 2023-06 having one of the keywords "LLM, Large Language Model, Transformer, Neural Machine Translation, Encoder-Decoder" in their title only indicate the currently ongoing hype surrounding large language models.

Relevance of Large Language Model Technology What is a *language model*, a large one at that, and where did the term suddenly come from? What is the hype all about? Research conducted by UBS[7], a systemically important multinational investment bank, dubbed ChatGPT the "fastest Growing App In The History Of Web Applications" (Gordon, 2023), reaching 100 million users within the first two months of its public release, showing the relevance of language model technology to the general public and the corresponding financial potential for corporations. The conversational bot ChatGPT produces human-like

[6] `arxiv.org`

[7] UBS is not/no longer considered an acronym and does not stand for Union Bank of Switzerland but became a designation

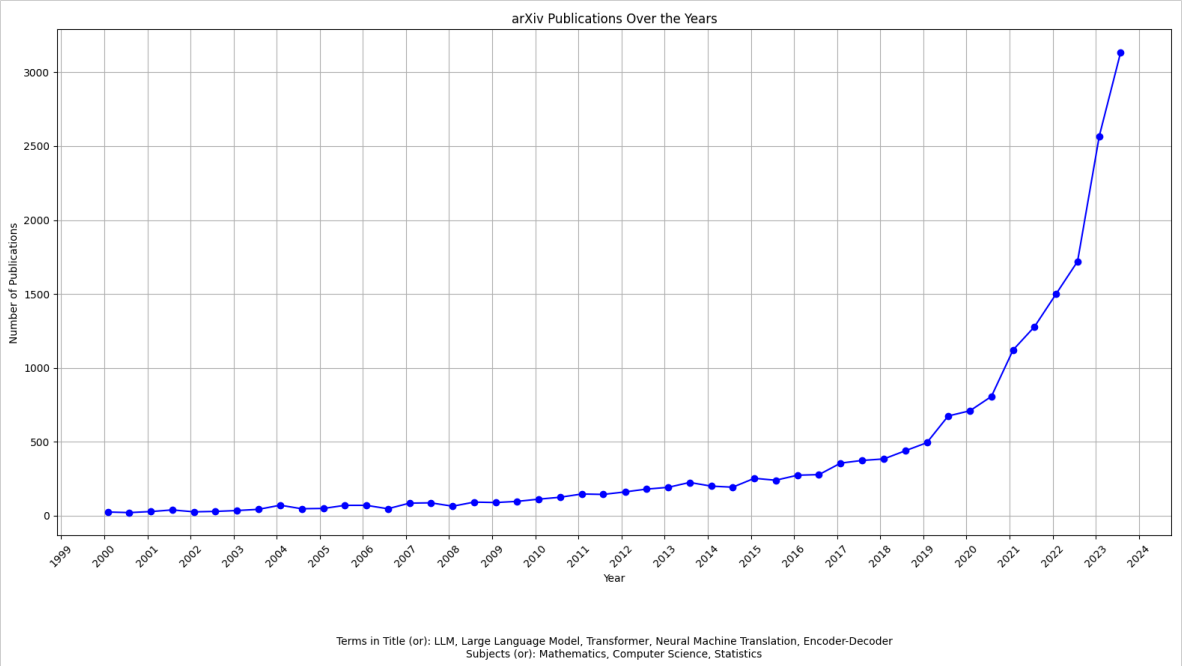


Figure 1: Number of Publications in specific Computer Science and Natural Language Processing topics from 2000 to 2023-06, Source: crawled from arxiv.org

text, including answering questions, writing short stories, composing music, solving math problems, performing basic computer coding, and translating languages. The quality of these artificially generated information is of such level that extensive debates now question points of contention, the fear, and the potential regarding the substitution of human workers with "AI" and its consequences for society. There have been numerous hype cycles in the artificial intelligence domain in the past, and the hype around language model technology undoubtedly shares similarities. However, the implications of finding a universal signal-approximating model for language are utopian. Are we on track to solve this matter with artificial neural networks, which we provide both an intuitive introduction to in Section 3.1 and a more rigor introduction in Section 3.1.3, the transformer architecture introduced in Section 3.2.3, and large language models? In a race for *artificial general intelligence* (AGI), Big Tech corporations compete in bringing forth the most powerful foundation model. The term AGI refers to the kind of intelligent behaviour that matches or surpasses human capabilities across a wide range of cognitive tasks. In the context of language models, a foundation model is believed to be a solid basis for representing the collective human text-based knowledge upon which AGI is (potentially) built. In the context of language models, to create a foundation model, learning algorithms produce specific coefficients based on digitized or digital books, texts, and other textual content from the internet, such as web pages and blogs. These coefficients are components of specific artificial neural networks. Equipped with these specific

coefficients, the neural network's task is to approximate the representation of collective human text-based knowledge "at the touch of a button" and even to evolve beyond reproducing learned representations by finding novel insights within the vast encoded space of knowledge, e.g., by discovering and interpolating new connections between thought streams or scientific domains. As the amount and quality of data increase, along with computational resources, the foundation models improve [8] — consequently improving the quality and impact of applications such as ChatGPT. But language is not the only domain where artificial neural networks and learning representations of some knowledge are on the rise. In micro-biology, a recently published model by Google DeepMind called "AlphaFold3" opens new research horizons, accurately predicting the structure of proteins, DNA, RNA, ligands and beyond proteins to a broad spectrum of biomolecules, and how they interact. Based on these predictions researchers hope to transform their understanding of the biological world and drug discovery, including the high financial potential in combination with societal prosperity. Earlier iterations of the AlphaFold model (AlphaFold and AlphaFold2) and research have already been cited more than 20,000 times (Google DeepMind AlphaFold team, 2024), and their scientific impact has been recognized through numerous awards. Many of the techniques and components used in the neural network architectures for applications such as ChatGPT and AlphaFold3 are based on findings originating from neural machine translation. The historical development of neural machine translation is highlighted in Chapter 2, while the concept of translation is discussed in Section 3.2.1. GPT is an acronym for Generative Pre-trained Transformer, indicating several key technologies that ChatGPT is composed of. The notion of finding a specific, trainable task to obtain a general-purpose representation, which serves as a starting point for various related subsequent tasks (downstream tasks), is called pre-training. The concept of training artificial neural networks and the meaning of a trainable task are introduced in Section 3.1.3. The Transformer is a recently developed neural network architecture component that efficiently produces information-dense, high-quality representations. This innovation has significantly impacted the rapid progression of machine learning and artificial intelligence. But what do these concepts have to do with low-resource machine translation scenarios, i.e., translation scenarios in which there exists a scarcity of (parallel) source and target language data? In the context of language, when we attempt to approximate a plausible target sequence based on a source sequence, we call this process *translation*. The ability to translate between languages, e.g., reduces global and intercultural boundaries, enabling interaction that foster understanding, collaboration, and communication among diverse communities while

[8] Philosophical and controversial. Trends indicate a saturation and upper performance limit for LLMs. However, *visionaries and evangelists* believe in the potential to overcome these saturation effects.

preserving intercultural lingual and linguistic differences. This alone is a valid motivation to improve existing translation solutions for smaller languages. Moreover, much of the impact of applications such as ChatGPT is limited by one's ability to express and understand the language in which the majority of collective human knowledge is stored. Both culturally and linguistically underrepresented facts and languages may get lost or not fully represented within the general representation compared to more strongly represented languages, such as English. Tackling the low-resource situation and improving translation models can not only provide a bilateral bridge between cultures and their potentially small languages but also grant access to collective knowledge for people belonging to those cultures, as well as offer the possibility to enrich the collective knowledge with translated knowledge from within those smaller languages. Regional folklore, customs, fairy tales, lyrics, and songs are among the types of knowledge that are potentially not even written down and hence completely hidden from the collective written knowledge. By making translation more accessible and versatile for smaller languages, we can counteract the loss of knowledge by providing means to preserve and document information originating from smaller language communities. In text generation we try to find the most plausible *next token* based on a source sequence and auto-regressively repeat that prediction based on the source sequence including the already predicted plausible next token. The text generation task with Transformers is closely related to text translation, and comprehension of one leads to understanding of the other, as we will see in Section 3.2.

From the Age of Retrieval to the Age of Generation Upon closer examination, we can expand the concept of translation between natural languages to the domain of computer science, with human language as one part and computer instructions as the other part of the language pairing. In our modern world, computers are ubiquitous, from personal computers in offices and cellphones in our pockets to cash register systems in supermarkets. The ability to communicate with and understand the language of computers distinguishes individuals with a background in computer science-related fields from others. Could it become reality for anyone without a background in computer science to give instructions to computers using natural language? What are the implications if harnessing the power of computation and the seemingly magical capabilities of computers for mundane tasks no longer requires extensive education in computer science-related fields? To what level of productivity will civilization rise, and how will fields of research beyond computer science thrive once scientists and innovators can command computers with ease using natural language? Recently, on June 3, 2024, at COMPUTEX, the Taipei International Information Technology Show, Jensen Huang, CEO of Nvidia, declared a new industrial revolution following the age of data and

information — one focused on generation rather than retrieval (NVIDIA, 2024). This new era is characterized by instructing computers to generate information rather than merely retrieving it. Instead of needing to learn another language, we simply instruct the computer to generate what we want to say in another language for us, empowering us to communicate with whomever we want wherever we are. These generative systems are data-hungry and require vast amounts of resources. Improvements in the domain of low-resource machine translation, which this thesis' experiment explores in Chapter 4, can potentially be adapted in non-low-resource scenarios, where the investigated exploits lead to even bigger amounts of data and even denser representations, hence a even stronger representation of our collective knowledge which consequently may lead to even better performing systems.

Outline - Unveiling Large Language Models through Neural Machine Translation The domain of language model technology has its roots in machine translation. This master's thesis provides insight into the field of large language models and demonstrates how a third pivot[9] language (English) can be used to improve translation results in a low-resource scenario, using the French-German language pair as an example. First, we explore the history of the field in Chapter 2. At the end of Chapter 2, we explore various research directions and approaches in the domain of low-resource machine translation, e.g., the so-called *pivot strategy* which is utilized in the thesis experiment. The pivot strategy exploits a situation where there are more resources between the source and pivot, or pivot and target language, or both, than between source and target language pair. The third language is called *pivot language*. It is introduced to potentially improve the translation quality between the source and target language. Chapter 3 introduces important concepts and components necessary for our experiment, which exploits additional language resources during neural network training. In Chapter 3, we first introduce the concept of artificial neural networks on a high level in Section 3.1. Building on this intuitive understanding, we explore the mathematical concepts behind neural networks in Section 3.1.3. We then examine related components in Section 3.2.2 that were crucial for the development of the current state-of-the-art Transformer architecture, which we introduce in Section 3.2.3. Through this process, we systematically uncover the basic building blocks that, when further developed and combined in specific ways, form the powerful modern neural network architecture known as the *Transformer*. These theoretical and mathematical components, however, need to be combined and enriched with the domain of *translation* and natural language. As the domain of neural networks is rooted in mathematics

[9]to pivot (verb): to turn or rotate, change in direction. Also, pivot (noun): a person, thing, or factor having a major or central role, function, or effect

and computer science, the connection to natural language processing is non-trivial and will be explored in Section 3.2. By continuously connecting the components, we naturally conclude our methodology with introducing said Transformer in Section 3.2.3. All these methodological foundations enable us to conduct our experiment, which we introduce in Chapter 4. In Chapter 4, we describe the research goal of the thesis and outline the practical considerations, and introduce the data and framework in which the experiment is conducted, and finally the framework its results are assessed in. We hypothesize that increasing the size of the pivot corpus will lead to measurable improvements in BLEU, chrF, and TER scores, with diminishing returns as the pivot data size increases. The training of large artificial neural networks requires vast amounts of computational resources. An advantage of neural networks is the potential of *transfer learning*, especially in the domain of language modeling. The pivot experiment leverages transfer learning by combining specific parts of existing models and fine-tuning them to a limited extent, as opposed to building one single large model. At the core of our approach is a *pivot*-strategy, which exploits the data availability of a specific pivot language to bridge two languages with limited direct parallel data. We exemplary conduct said experiment based on our methodology on the language pair French and German, using English as the common pivot language. The low resource situation is artificially created by heavily undersampling the existing data, and thus limiting the access of sentence pairs for training. The experiment's results are found in Chapter 5. First, we visually analyze the training procedure with the help of Tensorboard, a library specifically built to observe or log specific reporting metrics, such as the training and validation accuracy during training. The visual analysis serves as an initial indication of the experimental concept's validity. We demonstrate through examples that our experiment was generally successful; however, we also highlight shortcomings, anomalies, and difficulties encountered during the experimentation. We observe a slight improvement in translation quality as pivot resource availability increases, which we demonstrate by comparing translation quality metrics, such as BLEU, TER, and chrF in Chapter 5. The thesis is of practical nature in the field of computer science. To ensure reproducibility, the code is published on Github[10]. However, large parts of the source code consist of existing elements, such as the OpenNMT and PyTorch libraries, which we recombined and utilized for our specific needs. In Section 5.3, we discuss the results, explore them in depth, and compare our results to closely related research in Section 5.3. Finally, we conclude the thesis in Chapter 6 with an outlook on future research directions based on the novel insights we provide. This thesis is intended for students and interested readers in the fields of mathematics and computer science who seek introductory material on the

[10] `github.com`

subject, with some additional depth. While textbooks on this subject often cover a broad range of topics, we focus specifically on introducing the *transformer architecture* alongside an appropriate experiment, aiming for both scientific value and comprehensibility.

2 Literature Review

This chapter reviews the history and key developments in large language models, including findings and research advancements, leading to the current state-of-the-art Machine Translation approaches. First, we provide a brief review of the major contributions that have influenced the progression of Machine Translation, starting with early rule-based approaches, moving through phrase-based and data-driven statistical approaches, and culminating in neural network-based Machine Translation. Second, we introduce the current state-of-the-art in Neural Machine Translation (NMT). Third, we introduce the sub-domain of low-resource Neural Machine Translation. We provide a brief overview of selected topics, including (1) Exploitation of Monolingual Data, (2) Exploitation of Data from Auxiliary Languages, and (3) Exploitation of Multi-modal Data.

2.1 The History of Machine Translation

The history of trying to find specific rules between a pair of arbitrary languages and using rule-based mechanisms or machines for language translation dates back to as early as the 9^{th} century in the origins of Cryptology. "Cryptology is the study of cryptography and cryptanalysis. Cryptography is comprised of encryption and decryption. Encryption and decryption are the processes of substituting, ordering, and permuting discrete inscriptions. Decryption is the precise reversal of encryption. Cryptanalysis, on the other hand, is a form of intuitive guesswork that reveals plaintext from ciphertext and therefore admits the possibility of error (even when rationalized, systematized, or mechanized)" (DuPont, 2018). In The Codebreakers published in 1977, Kahn describes how "cryptography was born amongst the Arabs" and that they were the first to discover and document their findings on paper. This historical anecdote and the origins of cryptology, however, are of archaeological nature and in itself their field of research. Instead, we fast-forward about a thousand years to focus on more modern translation systems and mention their early predecessor only for the sake of completeness and trivia. The modern precursors for translation machines were developed by Georges Artsrouni and Petr Petrovič Trojanskij (Hutchins, 2002). They both developed mechanical translation machines around the year 1933. These purely mechanical devices, the "mechanical brain" (*cerveau mécanique*) proposed for patent 1933 by the French engineer Artsrouni and the "translating machine" proposed for patent 1933 by the Soviet Union engineer Trojanskij were limited by the progress and possibilities of their time (Hutchins, 2002). Artsrouni's machine can be described as a large dictionary capable of producing simple word-for-word "rough translations" by having access to up to 40m paper-band (i.e., memory or dictionary) and a selection mechanism based on perforations. Hutchins writes

that Artsrouni did not envision fully automatic or high-quality translation. He was no linguist and had no awareness of important linguistic concepts such as, e.g., polysemy [11], idioms [12], or syntactic ambiguity [13]. A prototype of Artsrouni's machine was built and successfully demonstrated at the Paris Universal Exhibition in 1937. Orders were placed for commercial production, but the Nazi occupation of France in 1940 ended any further developmentDuPont (2018). Nevertheless, Astrouni's work is in retrospect identified by many as an inspiration and foundation stone for future work in the field (Hutchins, 2002). Trojanskij's ambition, on the other hand, was to develop "a machine for selecting and typing words when translating from one language into another or several others simultaneously" (Hutchins, 2002). His proposal went beyond dictionary mechanization, unlike his contemporary Artsrouni. He articulated fundamental translation processes and introduced *logical parsing symbols*. These symbols were designed to denote *universal* grammatical relationships, making them applicable to any language. In a number of respects, his *logical parsing* resembles the kind of interlingual syntactic representation found in later Machine Translation work (Hutchins, 2002).

A possible modern originator of the computer-aided Machine Translation domain however is arguably Warren Weaver who, unaware of the earlier techniques and patents for translation machines (Hutchins, 1999), discussed the possibility of Machine Translation and in a July 1949 published memorandum on how to develop a "cryptographic-translation" technique. Weaver in his famous (Hutchins, 1999) memorandum "Translation" put forward four proposals, that Hutchins an English linguist and information scientist who specialized in machine translation summarised on the occasion of the 50th anniversary of the memorandum:

> The *first proposal* was that the examination of immediate context might tackle the problem of multiple meanings: *If one examines the words in a book, one at a time through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of words. "Fast" may mean "rapid"; or it may mean "motionless"; and there is no way of telling which.* (Weaver, 1949, p. 8)
>
> The problem was determining how much context would be required, and Weaver expected this to vary from one subject to another.

[11] Polysemy refers to the phenomenon where a single word has multiple related meanings.

[12] An idiom is a fixed phrase or expression that has a figurative meaning different from its literal meaning. Idioms often originate from cultural or historical contexts and can be difficult for non-native speakers to understand because the meaning isn't directly inferred from the individual words.

[13] Syntactic ambiguity, or structural ambiguity, occurs when a sentence can be parsed in multiple ways due to its syntax, leading to different interpretations.

> His *second proposal* started from the assumption that there are logical elements in language. He drew attention to a theorem proved by McCulloch and Pitts — developed in fact in the context of research on the mathematical modeling of the neural structure of the human brain — that *"a robot (or computer) constructed with regenerative loops of a certain formal character is capable of deducing any legitimate conclusion from a finite set of premises."* (Weaver, 1949, p. 10) The mathematical possibility of computing logical proofs suggested to Weaver that *"insofar as written language is an expression of logical character,"* (Weaver, 1949, p. 10) the problem of translation is formally solvable.
>
> The *third proposal* concerned the possible applicability of cryptographic methods. Weaver had been impressed at the success of cryptography based on, as he put it, *"frequencies of letters, letter combinations, intervals between letters and letter combinations, letter patterns, etc. which are to some significant degree independent of the language used."* (Weaver, 1949, p. 2) Weaver's ideas on cryptography were linked to information theory, which had recently been advanced by Claude Shannon. Weaver was writing a book about information theory with Shannon at the time (Shannon and Weaver 1949). The theory is concerned with the basic statistical properties of communication, including the effects of noise in telecommunication channels and of relative frequencies of signals. In particular, it embraced *"the whole field of cryptography."* (Weaver, 1949, p. 2)
>
> For his *fourth proposal*, Weaver became more utopian. It was based on the belief that, just as there may be logical features common to all languages, there may also be linguistic universals. Earlier in his memorandum, he commented on a paper by a sinologist Erwin Reifler, who had remarked that "the Chinese words for 'to shoot' and 'to dismiss' show a remarkable phonological and graphic agreement." Weaver's comment was: *This all seems very strange until one thinks of the two meanings of 'to fire' in English. Is this only happenstance? How widespread are such correlations?* (Weaver, 1949, p. 4) Weaver thought that such universals may be very common. (Hutchins, 1999, p. 1-3)

In the conclusion of the memorandum, Weaver expresses his conviction of linguistic universals, which Hutchins describes as one of the most famous metaphors in the field of Machine Translation:

> *Think, by analogy, of individuals living in a series of tall closed towers, all erected over a common foundation. When they try to communicate with one another,*

> *they shout back and forth, each from his own closed tower. It is difficult to make the sound penetrate even the nearest towers, and communication proceeds very poorly indeed. But, when an individual goes down his tower, he finds himself in a great open basement, common to all the towers. Here he establishes easy and useful communication with the persons who have also descended from their towers. Thus it may be true that the way to translate from Chinese to Arabic, or from Russian to Portuguese, is not to attempt the direct route, shouting from tower to tower. Perhaps the way is to descend, from each language, down to the common base of human communication — the real but as yet undiscovered universal language—and—then re-merge by whatever particular route is convenient.* (Weaver, 1949, p. 11)

Weavers' ideas pioneer the field from rule-based Machine Translation towards statistical approaches and still influence works today. Hutchins concludes, that in the long term, however, perhaps the most significant outcome of the Weaver memorandum was the decision in 1951 at the Massachusetts Institute of Technology to appoint the logician Yehoshua Bar-Hillel to a novel research position created explicitly for contributing to the domain and ideas of creating *machine translation*. Bar-Hillel wrote the first report on the state-of-the-art (Bar-Hillel, 1951) and organized the first conference on Machine Translation in June 1952, which is regarded as the first true milestone in the history of modern (western) Machine Translation. Thus, research on Machine Translation continued after Weaver. Figure 2 shows a flowchart that describes just a part of a complex rule-based translation algorithm from 1955 that was used in IBM TYPE 701, a famous machine translation computer endeavor in the 1950s. In the 1960s, however, Bar-Hillel surveyed the field of Machine Translation and critically concluded not only that the prospect of fully automatic high-quality translation was unrealistic given the current state of technology, but that the entire project was *impossible in principle* (Bar-Hillel, 1960). Sponsors to Machine Translation research grew sceptical and started being concerned over the stagnating advances, as expressed in the Automatic Language Processing Advisory Committee (ALPAC) report issued in 1964 (Hutchins, 1996). "Pre- and post-editing Machine Translation by humans was the norm, which required considerable time and effort." (Hutchins, 1996) The limitations of early Machine Translation methods stemmed primarily from their reliance on rule-based approaches, which attempted to codify linguistic rules and patterns into computational algorithms (e.g., Figure 2). Published in November 1966, the ALPAC report (Pierce et al., 1966) brought an end to substantial funding for Machine Translation research in the United States for nearly twenty years. More significantly, perhaps, was the clear message to the general public and the rest of the scientific community that Machine

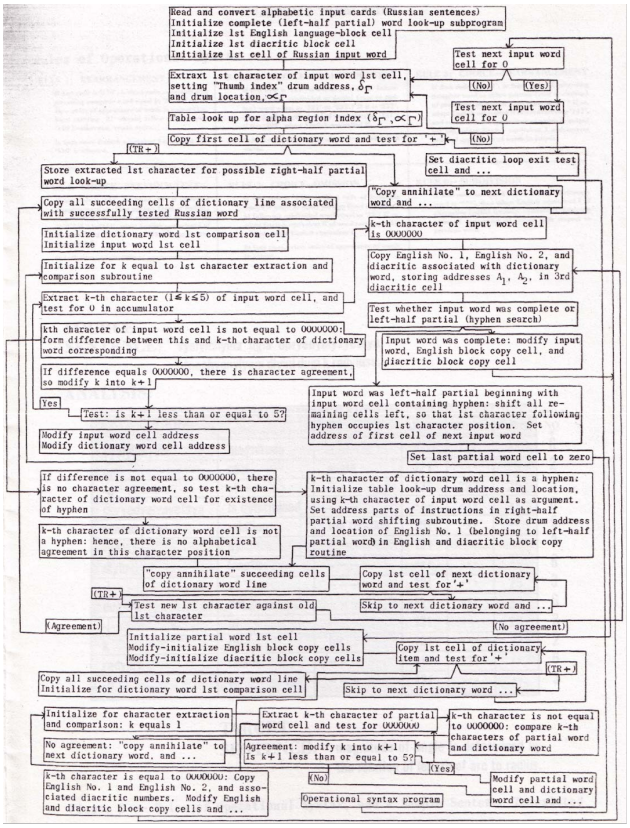


Figure 2: Flowchart of part of the dictionary lookup procedures (from Sheridan 1955), Source: Hutchins (2006)

Translation was hopeless (Hutchins, 1996). However, the committee agreed that research should continue ...

> *... in the name of science, but that the motive for doing so cannot sensibly be any foreseeable improvement in practical translation. Perhaps our attitude might be different if there were some pressing need for machine translation, but we find none.* (Pierce et al., 1966, p. 24)

The ALPAC report ends most of the Machine Translation research endeavors, preventing funding and dampening research interests in general. Machine Translation became part of and was in parts responsible for the first *AI winter* [14], which started with the ALPAC report 1966, to strongly paralyzing the domain from 1974 to approximately 1980 (Wikipedia, 2024). However, advances in Machine Translation did not stop completely and gained

[14] In the history of artificial intelligence, an 'AI' winter refers to a period of reduced funding and interest in artificial intelligence research. The field has experienced several hype cycles, followed by disappointment and criticism, followed by funding cuts, followed by renewed interest years or even decades later Wikipedia (2024).

momentum over time outside of the USA, i.e., in Japan, the Soviet Union, and EuropeHutchins (1989). By the 1980s, fully automatic high-quality machine translation had become practical and was being widely adopted by both governments and private organizations DuPont (2018). Enhanced methods and models improved translation quality, and faster, cheaper microcomputers further increased success while reducing costs DuPont (2018). In the 1990s, research moved away from hand-crafting linguistically motivated rules, and instead used example-based, data-oriented large language corpora and sophisticated statistics. A development that was sparked by exploits in artificial intelligence research and general advances in hardware and technology in addition to IBM research by Brown et al. who in 1990 published A Statistical Approach to Machine Translation - which set the field of Statistical Machine Translation in motion. The shift from rule-based to statistical approaches in Machine Translation brought about several implications. Statistical approaches reduce the reliance on manually encoded linguistic rules, allowing for more flexible and scalable translation systems. Instead of explicitly defining rules, Statistical Machine Translation systems learn patterns and associations from data, making them more adaptable to diverse language pairs and domains (Brown et al., 1990)(Brown et al., 1993). While statistical approaches excel in capturing common translation patterns, they may struggle with rare or ambiguous phrases that lack sufficient training data. Addressing this challenge requires techniques such as domain adaptation, data augmentation, and hybrid approaches combining statistical and rule-based methods. Since the 1990s statistical machine learning has flourished and was only recently dominated by the new state-of-the-art in Machine Translation. In the Year 2013, Kalchbrenner and Blunsom introduced "Recurrent Continuous Translation Models" (Kalchbrenner & Blunsom, 2013), ushering in the era of Neural Machine Translation by achieving state-of-the-art results in several experiments. Kalchbrenner's and Blunsom's model encodes a given source text into a continuous vector using a (convolutional) neural network, and then uses a recurrent neural network as the decoder to transform the state vector into the target language, freeing the need for excessive feature engineering compared to Statistical Machine Translation (Zhang, 2017). A year later Sutskever et al. (2014) and Cho et al. (2014) proposed the first pure sequence-to-sequence models for machine translation. Today, the Machine Translation domain has little role for linguists or rational, rule-based models of language DuPont (2018). "Number crunching" [15] for Machine Translation is now the norm. As of the Year 2024, the prior state-of-the-art Statistical Machine Translation approach has nearly completely evolved into *Neural Machine Translation* which utilizes

[15]"Number crunching" refers to the large number of calculations that take place during training and inference of large neural networks which Neural Machine Translation is based on

massive computational capabilities employing large and complex artificial neural networks, dominating and surpassing all other machine-based methods of language translation.

2.2 The Current State of the Art - Neural Machine Translation

Koehn in his book Neural Machine Translation reports that during the wave of neural network research in the 1980s and 1990s, *machine translation* had been in the sight of researchers exploring these methods. The models proposed by Forcada & Ñeco (1997) and Castaño et al. (1997) are "strikingly similar" (Koehn, 2017) to the current dominant neural machine translation approaches. Koehn adds, however, that none of these models were trained on data sizes large enough to produce reasonable results for anything but toy examples. He further states, that the computational complexity involved by far exceeds the computational resources of that era, and hence ideas have been abandoned for almost two decades (Koehn, 2017, p. 9-10). Koehn presents that the modern resurrection of neural methods in machine translation started with the integration of neural language models into traditional statistical machine translation systems and makes the necessity to use GPUs for training, which was expensive and difficult to do, responsible for the slow exploitation by the academic domain. Nevertheless, Kalchbrenner & Blunsom (2013), Sutskever et al. (2014) and Cho et al. (2014) were soon able to produce translations for short sentences. The addition of the *attention mechanism* Bahdanau et al. (2016) finally yielded competitive results. Koehn adds an anecdote, which further emphasizes the speed in which the field of Machine Translation developed:

> *Within a year or two, the entire research field of machine translation went neural. To give some indication of the speed of change: At the shared task for machine translation organized by the Conference on Machine Translation (WMT), only one pure neural machine translation system was submitted in 2015. It was competitive but outperformed by traditional statistical systems. A year later, in 2016, a neural machine translation system won in almost all language pairs. In 2017, almost all submissions were neural machine translation systems.* Koehn (2017, p.10)

Today (2024), Neural Machine Translation models dominate the field of Machine Translation. They are on par with statistical approaches[16] and usually even surpass them, depending on the language pairing and the linguistic aspects or availability of data. Achieving human-level translation quality with Neural Machine Translation approaches seems to be within reach, as

[16]Phrase-based Machine Translation is a representative of the Statistical Machine Translation discipline

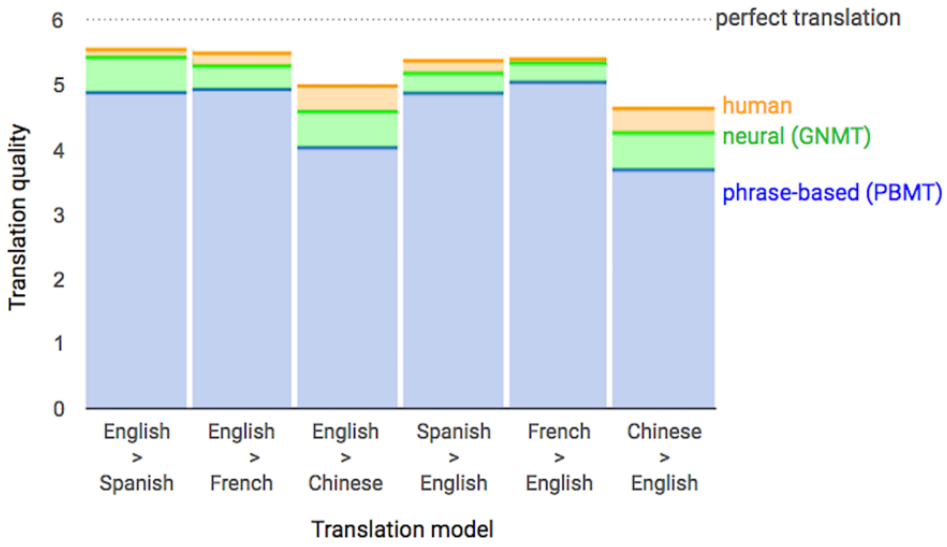


Figure 3: The comparison between the translation quality (score from 1 to 6) of GNMT (Google Neural Machine Translation, green), PBMT (Phrase-Based Machine Translation, blue) and human translation (yellow) for different language pairings. The comparison depicts the mean translation quality on the y-axis on a scale from 0 to 6 where human raters have repeatedly evaluated and compared the quality of two translations presented side by side for a given source sentence (Wu et al., 2016). The x-axis reports these values for each pair of languages. The evaluation data consisted of 500 randomly sampled sentences from Wikipedia and news websites, and the corresponding human translations to the target language for each language pair. Source: Human vs. AI: An Assessment of the Translation Quality Between Translators and Machine Translation, (Wu et al., 2016)

seen in Figure 3.

2.3 Low-Resource Neural Machine Translation

The primary task of the thesis is to explore a pivoting strategy for a low-resource neural machine translation scenario. While there are thousands of languages in the world, the major popular commercial translators (e.g., Google translator, Microsoft translator, Amazon translator) only support tens or a hundred languages due to the lack of large-scale parallel training data for most languages (Wang et al., 2021). In recent years, many algorithms have been designed for low-resource Neural Machine Translation. Following the survey conducted by the Microsoft Research Group Wang and colleagues, theoretical streams can be divided into three major groups:

1. Exploitation of Monolingual Data

2. Exploitation of Data from Auxiliary Languages

3. Exploitation of Multi-modal Data.

While all three groups have a similar goal - to eliminate the most limiting factor for Neural Machine translation: its need for large amounts of parallel data for model training - there are clear differences between their strategies. Since the field is expanding and developing quickly (as indicated by the current exponential growth in Figure 1), exploitations in groups (1) and (3) beyond the findings in Wang et al. (2021) are not or are not necessarily further discussed in the scope of this thesis. In this thesis, one idea belonging to the group (2), the exploitation of data from auxiliary languages, is being explored primarily. For completeness and overview, all three groups are introduced briefly. Readers who are familiar with the domain of statistical machine translation may find parallels to known approaches and strategies that have been explored in the past in the field of statistical machine translation. *Pivoting* or using an intermediate language, e.g., has been proposed by Schubert as early as 1988. Neural Machine Translation can be understood as a descendant of Statistical Machine Translation, so many ideas that improved Statistical Machine Translation may also inspire current and upcoming improvements and advances in Neural Machine Translation. Also, advances in low-resource machine translation can lead to general advances in machine translation, as the abundance of training data seems to increase the translation quality (Johnson et al., 2017), so techniques and exploits may be applied to rich-resource scenarios as well. Later, in Chapter 3, we will conduct a more rigorous and formal introduction to nomenclature, core concepts, and definitions regarding Neural Machine Translation.

2.3.1 Exploiting Monolingual Data of Source and/or Target Languages

Monolingual data contains a wealth of linguistic information (e.g., grammar and contextual information) and is more abundant and easier to obtain than bilingual parallel data. In the following strategies, making use of the abundance of monolingual data is exploited to improve the translation quality especially in low-resource scenario. Sennrich et al. (2016a) propose using monolingual data as an addition to the parallel data, with one-sided empty pairs that can be conceived as a form of dropout. Additionally, Sennrich et al. propose *Back-Translation* massively augmenting their corpus with synthetical data for the source side while the target side comes from the monolingual corpus. Sennrich et al. (2016a) report considerable improvements against models that do not exploit monolingual data (+1 to 3 BLEU points). The BLEU metric is the most common metric for benchmarking machine translation approaches. A higher BLEU metric is understood to correlate with a higher translation quality. The BLEU metric is thoroughly introduced in Chapter 4. Hoang et al. (2018) propose iterative back-translation as an improvement over (Sennrich et al., 2016a).

Another improvement in BLEU is reported by Zhang & Zong, who propose exploiting not the target but the source side of the monolingual data. Zhang & Zong (2016) propose a novel self-learning algorithm to generate synthetic large-scale parallel data and on top of that use a multi-task learning framework with two Neural Machine Translation models to predict the translation and the reordered source-side monolingual sentences simultaneously (Zhang & Zong, 2016). Furthermore, the concept of dual learning is proposed by Xia et al. (2016). Xia et al.'s approach utilizes both the source- and target-side of the monolingual data, which again leads to substantial improvements over the baseline (+1 to 5 BLEU points). Dual learning is achieved by a novel two-agent communication game and a warm-up phase with only 10% of parallel/bi-lingual data (Xia et al., 2016). The results indicate that working with small fractions ("only 10%") can indeed already suffice to generate models that are on par with models trained on more abundant data resources. In general, exploiting one or both sides of source- and/or target language has been and is explored by different actors in the domain. Ideas range from unsupervised Neural Machine Translation, back- and forward-translation, joint training on both translation directions, language model pre-training, and mining parallel information from comparable corpus, to enhancing the training with bilingual dictionaries (Wang et al., 2021). These promising techniques can be and are combined to achieve and strive for even better results in low-resource Neural Machine Translation.

2.3.2 Exploiting Data from Auxiliary Languages

Languages with similar syntax and/or semantics are helpful to each other when training Neural Machine Translation models. Wang et al. report three distinct categories for exploiting data from auxiliary languages by leveraging similarities in vocabulary, word order, or grammar. (1) multi-lingual training, where the low-resource pair is jointly trained with other language pairs in one model, (2) transfer learning, where a parent Neural Machine Translation model containing a rich-resource language pair is later fine-tuned and (3) pivot translation, where one or more pivot languages are selected to serve as bridges between source and target language. Johnson et al. describe pairing different low and high resource languages (1) into one large dictionary. They introduce an artificial token *('<2target-language>')* as a source data prefix. All other components (encoder, decoder, attention, shared wordpiece vocabulary) stay the same (Johnson et al., 2017). In doing so, they train a multilingual model that can translate two or more source languages into the specified target language, therefore leveraging the rich resources of the auxiliary language during training. Johnson et al. report that models trained on $A \longrightarrow B$ and $B \longrightarrow C$ pairings deliver reasonable results for $A \longrightarrow C$ (*zero-shot translations*), even though the corpus does not contain any $A \longrightarrow C$ pairs. To Johnson et al. knowledge, it is the first work to demonstrate the possibility of

zero-shot translation and a successful example of transfer learning in machine translation. In their experiments, the quality of the translations (BLEU Score) is improved when multiple source languages are translated into one target language (Many-to-One). In other scenarios (One-to-Many, Many-to-Many) the quality does not improve or even declines - even though other improvements such as the high redundancy of many different models for individual language pairs become apparent (Johnson et al., 2017). In another work Zoph et al. propose a transfer learning approach (2). Their idea is to train a *parent model* and transfer some amount of parameters to the *child model* for initialization and constraining training (Zoph et al., 2016) [17]. The low-data child model will thus not start with random weights and has either target or source language embeddings "already in place". These embeddings can be fixed or frozen and do not need to be trained anew. Zoph et al. increase the amount of BLEU for a given language pair by an average of 5.6 BLEU for 4 low-resource language pairs against their neural machine translation baseline and report being close to a strong syntax-based machine translation model[18]. On top of reporting on transfer learning in general, Zoph et al. state the importance of identifying sensible *parent* languages as similarity seems to correlate with a stronger positive impact on the final low-resource neural machine translation model. Auxiliary language selection is a field of interest and different approaches are being investigated. Wang et al. list several authors that have tried to determine if languages from the same language family, languages based on closely clustered embeddings, or language-level/sentence-level similarity between languages result in the most helpful starting point for model training. Furthermore, different manipulation approaches, such as re-ordering words in a sentence and re-designing parts of the auxiliary language to align with the low-resource language have shown improvements (Wang et al., 2021). The third exploitation (3) is the usage of one or more pivot languages in training. This strategy involves using one or more rich-resource languages as a *bridge* between the source and target language. An improved source-target translation can be formed by creating a source-pivot and pivot-target corpus and model. There are mainly three ways to exploit the pivot strategy. The first and most naive approach is, to first translate the source into the pivot language, and then to translate the pivot into the target language - essentially building a pseudo-parallel corpus between source and target language. This simple and effective approach suffers from the problem of error propagation: mistakes made in source-to-pivot translation will be propagated to pivot-to-target translation. For cases where the source-to-target models are of poor translation

[17]The parent and child nomenclature is based on a data structure (Tree) and refers to a subnode of a given node in a tree (graph structure) which is called a child node, and the given node, in turn, is the child's parent.

[18]short: SBMT, statistical machine translation approach)

quality because, e.g., the language pair is distant in family and grammar, this naive pivoting method can lead to improvements (Leng et al., 2019). In most cases, one pivot language is selected based on prior knowledge (e.g., language family, similar syntax, grammar). Leng et al. (2019) propose a learning-to-route (LTR) method to automatically select one or several pivot languages to translate via multiple hops to increase translation quality to up to +5 BLEU points (Leng et al., 2019) against naive low-resource source-target neural translation models. The second approach is, to train a source-pivot and pivot-target model directly and to find a *combination* of their results leading to one model, e.g., using pivot-word-embeddings as a common connection term (Cheng et al., 2017), or, e.g., using a teacher-student approach as introduced in (Chen et al., 2017). The latter is interesting as it does not use any parallel corpus between the source and target language. Instead, the method assumes that parallel sentences in pivot-to-target have similar probabilities as pivot-to-source, therefore training the source-to-target model directly while guiding it with a pivot-to-target teacher model, should suffice. Chen et al. (2017)'s approach not only allows for direct parameter estimation but also improves efficiency while completely avoiding the problem of error propagation due to pivoting. Their results show an improvement of up to +3 BLEU points across multiple language pairs against their baseline (Chen et al., 2017). The third way involves leveraging the parameters of the source-pivot and pivot-target models. Kim et al. (2019) transfer the encoder of the source-pivot model and the decoder of the pivot-target model to the source-target model. Re-using parts from closely related models is similar to pre-training or the transfer-learning strategy introduced by Zoph et al. (2016). While in general similar to pre- and transfer learning, the exploitation of pivot languages potentially greatly influences the translation quality Wang et al. (2021).

2.3.3 Exploiting Multi-modal Data

Multi-modal data (e.g., parallel data between text and image) has also been used in low-resource Neural Machine Translation. Pseudo-parallel corpora are at the base of the exploit (Chen et al., 2019). E.g., descriptions for images in source and target language can be interpreted as describing the same context and, therefore be used to learn from. Chen et al. even use the term "image-pivot", drawing clear parallels to the "exploitation of pivot languages" strategies introduced in Subsection 2.3.2. Chen et al. propose using respective image caption models that describe an image in both the source and target languages to build a parallel corpus, primarily to increase the availability of data for rare or low-resource languages. The strategy can potentially be applied to other sources of data as well. However, due to the difficulty of data acquisition in low-resource scenarios, the exploitation of multi-modal data is currently limited (Wang et al., 2021). In other literature, exploiting

multi-modal data, e.g., image-pivoting is summarized under *pivot-based* strategies (Chen et al., 2017) and not a theoretical stream on its own.

2.4 Pivot based Transfer Learning for Neural Machine Translation

The subject of this thesis is to reproduce the results and findings based on the pivot strategy introduced by Kim et al. (2019) and Mhaskar & Bhattacharyya (2021) while exploring and presenting the transformer architecture. Mhaskar & Bhattacharyya introduce a single source-target model that counteracts the two major drawbacks the common *naive* pivoting strategy has. Firstly, the source sentence is passed through two different neural machine translation models to produce the target sentence. Passing through multiple models multiplies the decoding time for generation accordingly, which is inefficient. Secondly, the errors in the first translation step (source-pivot) propagate into the second translation (pivot-target), which leads to a reduction of translation quality. Mhaskar & Bhattacharyya describe the method that counteracts these drawbacks as *direct pivoting*. Their approach is to train two separate models, source-pivot and pivot-target, on their respective high-resource parallel corpus. Afterward, they detach the encoder from the source-pivot model and the decoder from the pivot-target model to create a third encoder-decoder model, on which a final fine-tuning with the low-resource corpus is performed. Their hypothesis is that the initialized encoder and decoder of the source-target model have already learned some representations or knowledge from the previous tasks, and that this knowledge aids in the source-target translation task. Their approach is closely related to Kim et al. (2019), who theorized a similar method but pointed out a caveat: the source encoder is trained to be used by a pivot decoder, while the target decoder is trained to use the outputs of a pivot encoder—not a source encoder. Kim et al. instead propose step-wise training of a source-pivot model on a source-pivot corpus, followed by continued training with a pivot-target corpus while freezing the encoder parameters. They hypothesize that, in the second step, the target decoder is trained to use the outputs of the pre-trained source encoder as input. However, freezing the pre-trained encoder ensures that even training on the pivot-target corpus in the second step the final model's encoder encodes the source language. They reason that without the freezing, the encoder completely adapts to the pivot language input and is likely to forget source language sentences (Kim et al., 2019). This approach requires a joint vocabulary of the source and pivot languages so that the encoder effectively represents both languages. Kim et al. (2019)'s approach produces interesting results and formulates a framework in which we plan to embed our experiment: Does a relation between translation quality and availability of pivot resources exist, and if so, is there a point of diminishing returns? We explore our experiment in greater

detail in Chapter 4.

3 Methodology

In this Chapter, we present the comprehensive methodology employed to investigate the impact of varying the amount of resources within the exploitation of using a pivot language strategy for machine translation. The methodology is structured on a foundation of both theoretical and practical considerations, to ensure a nuanced understanding of the interactions within the translation process. At the core, neural networks need to be revisited, including the introduction of the back-propagation algorithm, which is further built upon to introduce the transformer architecture. Also, a detailed introduction to neural machine translation is provided, as neural machine translation is the main theme of this study.

3.1 Neural Networks

In revisiting neural networks, this chapter delves into fundamental concepts and introduces the backpropagation algorithm, a key advancement in the training of neural networks. Understanding the intricacies of neural networks is essential for grasping the subsequent sections that delve into sequence to sequence learning which is the basis for translation. The basis, however, evolves into a more modern concept called *Transformer*, which is the neural network architecture the thesis' experiment is utilizing. The explanation begins with the basic concepts of neural networks. Neural networks are introduced in a variety of books and sources. In the introduction of notation and concepts for artificial neural networks, we primarily use (Hastie et al., 2001) and (Bishop & Bishop, 2024) as a source. In this section, first, historical remarks are given, then neural networks are introduced on a conceptual level, and finally, a more formal and rigorous definition is proposed.

3.1.1 The Origin of Artificial Neural Network

Neural networks are the backbone of modern machine learning, mimicking a simplified structure and functioning of mammal brain neurons (compare Figure 4). The mammal or human brain consists of interconnected neurons, the fundamental units of the nervous system. Neurons communicate through synapses using electrical impulses and chemical signals. Each neuron receives electrochemical inputs from other neurons in the dendrites (Williams et al., 2016). If the sum of these electrical inputs is sufficiently powerful to activate the neuron, i.e., if the action potential threshold is surpassed, it transmits an electrochemical signal along the axon and passes this signal to the other neurons whose dendrites are attached at any of the synaptic terminals. These attached neurons may then fire. The brain has intricate neural networks with multiple layers of neurons and complex connectivity patterns and exhibits

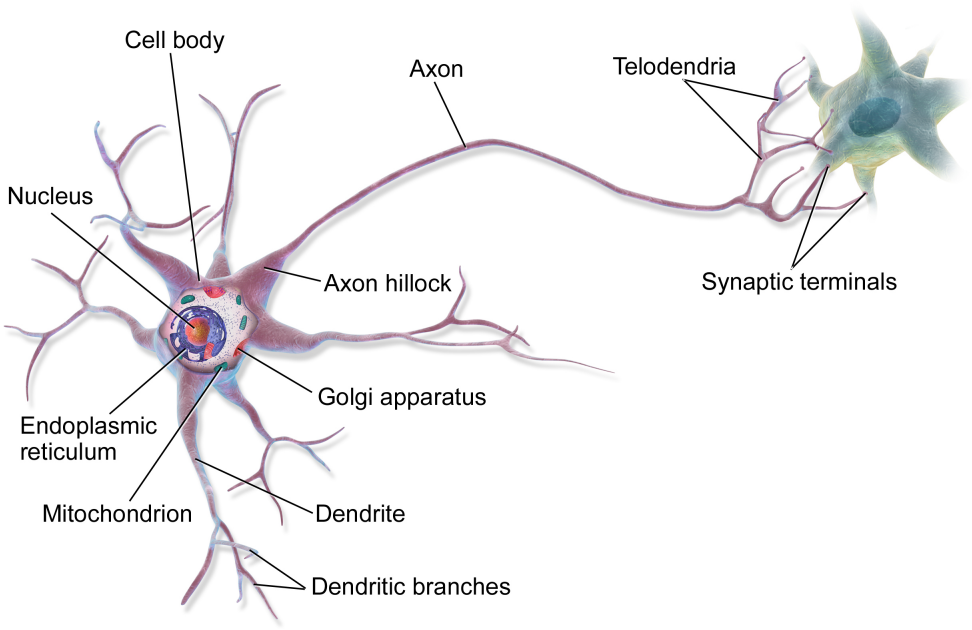


Figure 4: Model of a biological neuron. Source: Wikimedia/Neuron [19]

memory through the strength of synaptic connections and patterns of neural activation. A human brain contains around 90 billion neurons in total, each of which on average has several thousand synapses with other neurons, creating a complex network having a total of around 100 trillion (10^{14}) synapses (Bishop & Bishop, 2024).

In the computer science domain, these biological concepts were first abstracted by McCulloch & Pitts. In their 1943 paper "A logical calculus of the ideas immanent in nervous activity" (McCulloch & Pitts, 1943) the McCulloch-Pitts neuron is introduced to the scientific community. The two scientists make "physical assumptions which are most convenient for the calculus" (McCulloch & Pitts, 1943) and show that simple, linear gates (AND, OR, NOT, and others) can be constructed from those assumptions using the simplified artificial *nets* and their neurons. In 1949, Hebb's work (The Organization of Behavior) highlights the reinforcement of neural pathways through usage, a concept integral to (human) learning. Hebb proposes that the simultaneous firing of two nerves strengthens their connection. The widespread quote in the machine learning community "What fires together wires together" later is derived from Hebb's theories (Shatz, 1992) [20]. The quote suggests that neurons that simultaneously show high activation should be also reinforced simultaneously. The statement

[20]not by Donald Hebb himself but most likely by Carla Shatz an American neurobiologist who wrote "In a sense, then, cells that fire together wire together." in the Journal Article "The Developing Brain" from 1992

is an oversimplification and should therefore be used and interpreted with caution.

> *While mnemonic, this summary bares the risk of obscuring the importance of causation in Hebb's actual work: if two neurons literally fire together, i.e., at the same time, the firing of one cannot cause that of the other. Temporal precedence, rather than simultaneity, is the signature of causality and would indicate that 'one took part in firing the other'.* (Keysers & Gazzola, 2014)

Nevertheless, despite the many inaccuracies and oversimplifications trying to mimic the biology of the brain, these two early works lay the early foundation for the development of modern artificial neural networks that drive complex applications, such as the face recognition of media management software, artificially generative text applications or even autonomously driving cars.

3.1.2 Artificial Neural Networks

We first explore artificial neural networks on a high level to introduce the concept and build up intuition. Hastie et al. gives a good foundation for this intuition. Later in this section, we introduce the notation and definitions given by Bishop & Bishop in their latest work Deep Learning - Foundations and Concepts, 2024. A computational graph notion is provided by Goldberg (2017).

Artificial Neural Networks: Intuition At their core, artificial neural networks consist of layers of interconnected nodes. Each layer contributes to the extraction and transformation of information. The input layer receives data, hidden layers process it, and the output layer produces a final result. The central idea is to extract (without loss of generality) linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features (Hastie et al., 2001). Oversimplified, the goal is to deduce some true target signal $\mathbf{Y}$ by transforming information encapsulated in $\vec{x}$, one of many observations with $1, \ldots, D$ different measurements (also called *features*), and coefficients $\vec{w}$ in a specific way: $\mathbf{Y} = h(\mathbf{X}, \vec{w})$, where h is called an activation function. $\mathbf{Y}$ can be virtually anything as long as it can be expressed as a number: Class membership probabilities (e.g., probability of an image belonging to a category such as "cat" or "no cat" in image classification problems), or a (positive) real number (e.g., the price of an apartment in $). In the case of language translation modeling, e.g., $\mathbf{Y}$ is a set of vectors with probabilities linked to a vocabulary dictionary. These probabilities indicate the likelihood of each respective word being the a proper translation candidate or the next word in the translated sentence conditioned on the input sentence $\vec{x}$ from text corpus $\mathbf{X}$. $\vec{x}$, on the other hand, is what we *hope* carries

the information for this deduction to work. Most of the time $\vec{x}$ does not come alone but is part of a series of observations $\mathbf{X} = ([\vec{x}]_1, \ldots, [\vec{x}]_N)$. This object can be understood as a collection of information in matrix form, where the $[\vec{x}]_i$ are the rows (one observation corresponds to one row) and, where each entry in the D-dimensional vector $\vec{x} = (x_1, \ldots, x_D)$ is a feature. Henceforth, the data matrix $\mathbf{X}$ is the collection of information. In other words, for an output signal $\mathbf{Y}$ we are interested in what information exactly $\mathbf{X}$ carries to, as close as possible, reconstruct $\mathbf{Y}$, and how exactly the reconstruction process needs to look like, i.e, what kind of function, parameters or coefficients we need to express $f : \mathbf{X} \mapsto \mathbf{Y}$. And that is where the *weights* come into play. The *weights* are the coefficients that decide which part of some $\vec{x} \in \mathbf{X}$ needs to be recognized less or more for reconstructing some $\vec{y} \in \mathbf{Y}$. In artificial neural networks, the *weights* are learnable parameters that change during the training process. Initially, the *weights* are random values that throughout the training procedure strategically converge to a fixed estimation of $\vec{w}$ that lead to $\hat{\mathbf{Y}}$ (the reconstructed signal) that is close to $\mathbf{Y}$ (the true output signal), i.e., the optimization goal is $argmin_{\vec{w}|\mathbf{X}}(\mathbf{Y} - \hat{\mathbf{Y}})$. In our case, the reconstruction process f is an artificial neural network, our information is carried by $\vec{x} \in \mathbf{X}$ and our signal of interest is $\mathbf{Y}$. Similar concepts can be found in other statistical or machine learning-related approaches, such as a simple linear regression. The more information the many $\vec{x}$ related to their corresponding $\vec{y}$ carry, the better the reconstruction potential. Caution: There are cases in which $\mathbf{X}$ does not encapsulate enough information or the information is *by chance* related to $\mathbf{Y}$. In the first case, the reconstruction of $\mathbf{Y}$ fails, which is indicated by bad performance metrics when testing the reconstruction on data examples for which we know the *true* $\mathbf{Y}$, i.e., when the discrepancy between our estimate $\hat{\mathbf{Y}}$ and the true signal $\mathbf{Y}$ is big. For this reason, it is common practice to *test* a reconstruction model and calculate how well the model *fits* the input signal onto the output signal. In the second case, the reconstruction of $\mathbf{Y}$ might succeed and the so-called *spurious relationship* might even stay undetected. The famous saying "correlation does not imply causation" therefore needs to be in the back of one's mind when approaching any sort of signal reconstruction.

> *Another example of a spurious relationship can be seen by examining a city's ice cream sales. The sales might be highest when the rate of drownings in city swimming pools is highest. To allege that ice cream sales cause drowning, or vice versa, would be to imply a spurious relationship between the two. In reality, a heat wave may have caused both. The heat wave is an example of a hidden or*

unseen variable, also known as a confounding variable. [21]

It becomes apparent that, on top of that, the input data $\mathbf{X}$ plays an essential role in the reconstruction of the signal. The interplay of these factors, the nature of the reconstruction approach, the aspired output signal, and the quality and relevance of the source data for the output signal are the difficulties when building approximation models for signal reconstruction.

Single-layer Perceptron The simplest form of a neural network is a single-layer perceptron (sometimes called the single hidden layer back-propagation network (Hastie et al., 2001)). Figure 5 on the left-hand side shows the neural network *input layer*, in this example a

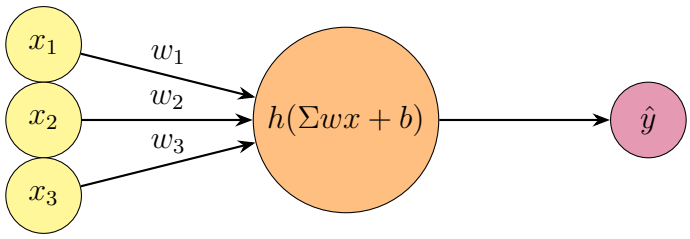


Figure 5: Single-Layer Perceptron

three-dimensional, real-valued vector $\vec{x} = (x_1, x_2, x_3)$. The edges towards the sigma symbol are called *weights* $\vec{w} = (w_1, w_2, w_3)$ and determine the magnitude the corresponding input is incorporated into the *activation function* h, i.e., the amount the specific weighted input value is adding or removing towards a threshold needed to be surpassed for the neuron to activate. The term b is called bias and can be introduced to dampen or amplify the neuron's ability to overcome its activation threshold. Each neuron produces an output that is either the input of the next neuron or the final output signal of the network. In Figure 5 the final node is shown as $\hat{y}$. The choice of notation is not common but emphasizes that evaluating the network brings forth a reconstruction estimation at its final output layer, not the true signal. Another possible choice of notation would be to denote the output node of the neural network building block as a function of the input layer and the weights. Since the neural network's nature is recursive, there is no difference between the intermediate and the final output. The recursive nature of the procedure is promoted using the function notation as depicted in Figure 6. In other words, the output of a neural network component may be the input for the component, like a building block. These blocks can be combined, intertwined, and stacked horizontally, or vertically for varying artificial neural network architectures and configurations.

[21] Wikipedia: Spurious Relationship, `https://en.wikipedia.org/wiki/Spurious_relationship`

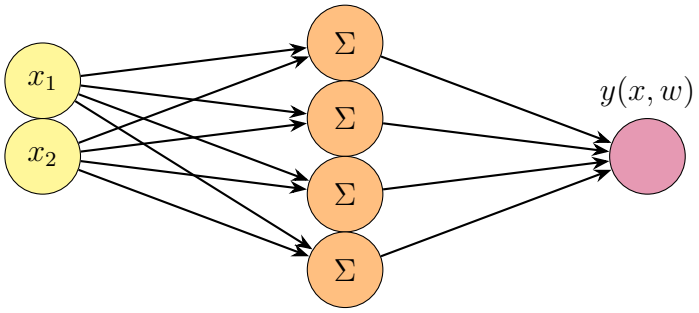


Figure 6: Single-Layer Multiple-Hidden-Nodes Variation

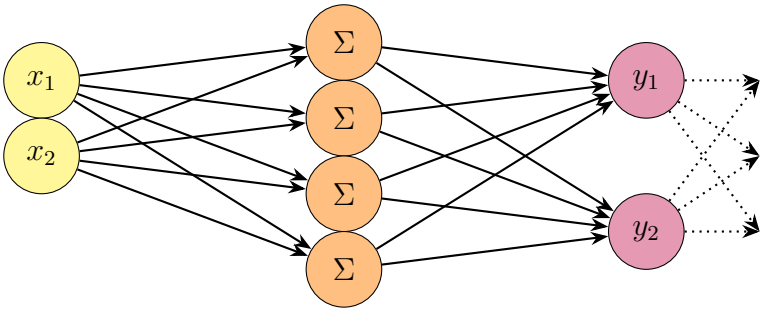


Figure 7: Multi-Layer Multiple-Hidden-Nodes Variation

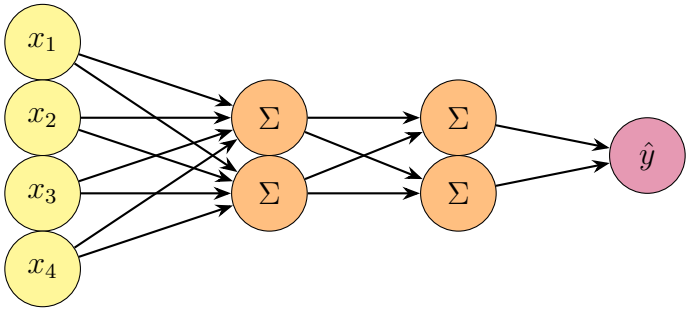


Figure 8: Fully connected feedforward artificial neural network

The core idea, in principle, stays the same. The examples in Figure 5 to Figure 8 depict small variations to simple neural networks and should be intuitive to follow. In these examples, because each node is connected to all nodes in the following layer, the architecture is characterized as 'fully-connected'. It is also characterized as "feedforward" because the computation proceeds iteratively from one layer of units to the next.

The setting of weighted paths between nodes can vary depending on the context. The width, depth, and interconnectedness are the most basic variations of the artificial neural network architecture. More complex variations are introduced in Section 3.2.2 about recurrent neural networks and Section 3.2.3 about the transformer architecture.

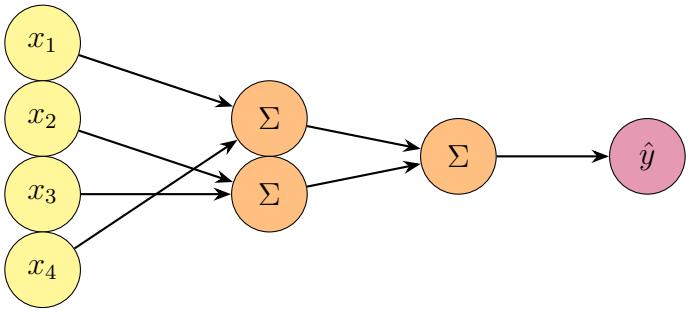


Figure 9: Artificial neural network Variation with multiple hidden Layers but not fully-connected

Activation Functions Activation functions play a crucial role in determining the output of a neuron. They control if and to what extent a neuron influences the artificial neural network calculation. Activation functions are simple, differentiable, often-times bounded parameterized functions that take $a_j = b_j + \sum_{i=0}^{D} w_{ij}^{(m)} \cdot x_i,\ x \in \mathbb{R}^{N \times D},\ m = 1, \ldots, max\_depth$, (*max_depth* represents how many hidden layers the architecture has, b_j is the bias term of the neuron in the m-th layer) as their input and map them to a single-valued, real output. $w_{ij}^{(m)}$ describes the weighted path connecting neuron$_i$ in the $(m-1)$ layer to neuron$_j$ in the (m-th) layer. As i goes up to D (the number of neurons in layer $(m-1)$), the neuron at a_j accumulates the sum of the current input nodes x_i multiplied by their respective weight $w_{ij}^{(m)}$. There is no rigorous definition for these functions as they are often tailored to fit a specific problem and are an area of research on their own. The activation functions can introduce non-linearities, allowing neural networks to learn non-linear relationships within the data. Otherwise, if the activation function is linear, only linear or affine mappings can be represented. Figure 10 shows common activation functions including (without loss of generality) hyperbolic tangent (a), a hyperbolic tangent derivation (b), rectified linear unit (d) and a rectified linear unit derivation (e), each serving specific purposes in different contexts. Bishop & Bishop (2024) also lists softplus (c) and absolute (f). Their predecessor, the logistic sigmoid function (see Eq. 1), is widely used in the early works on multi-layer neural networks. Sigmoid functions are inspired by studies on the properties of biological neurons (Bishop & Bishop, 2024). As mentioned before, the role of the activation function is to either suppress or control the amount a specific neuron is *firing*. Mathematically, the behavior can be constructed, e.g., by mapping input value $x \in \mathbb{R}$ to representative values that are by definition interpreted as *suppression*, e.g. $f(x_0) = 0$, where the value 0 implies "x_0 does not produce enough activation potential", or *activation* and its *potential*, e.g., $f(x_0) = \theta + \epsilon$, implying there is $\theta + \epsilon$ activation potential, which is equal to or stronger than the minimum threshold of excitation θ needed to *activate* the neuron. Biologically, there exists a neuronal excitation curve over time. If

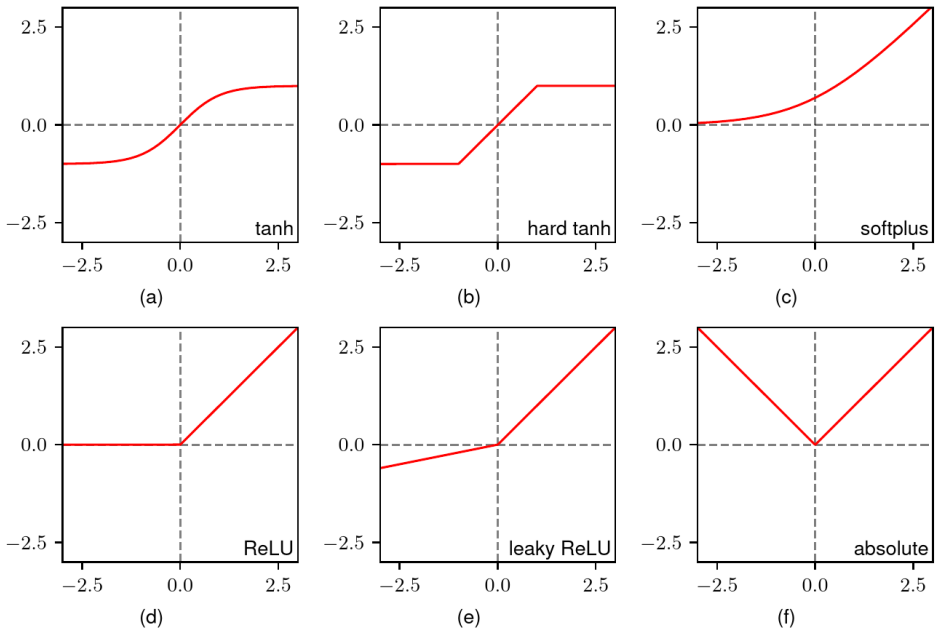


Figure 10: A variety of nonlinear activation functions. Source: Bishop & Bishop (2024, p.184, Figure 6.12)

the excitation threshold is met, the electrochemical signal first spikes due to its respective electrochemical composition, peaks at the activation potential, and vanishes after passing the activation potential onto the next neuron. The time component in the artificial replica is modeled by evaluating the activation function and is negligible. Passing the potential on is modeled by the activation function. The electrochemical spike and its maximum, the activation potential, can mathematically be constructed by bounding $\theta + \epsilon \in (0, 1]$ (without loss of generality). The sigmoid functions are a known class of differentiable functions to achieve such feats: they take any real-valued integer and smoothly map it to $(0, 1)$.

Sigmoid Function:

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{1}$$

Mathematically, logistic sigmoid functions have the characteristic of mapping $(-\infty, \infty) \mapsto (0, 1)$, whereas other activation functions, e.g., tanh can map to ranges such as $(-1, 1)$. Allowing for a negative value range can be conceptualized as inhibitory potential. At this point the biological model is gradually being neglected and mathematical sensibility is prioritized. Theoretically, for any network with logistic sigmoid activation functions, there is an equivalent network with tanh activation functions, as the difference is just a linear transformation of input and output values.

Hyperbolic Tangent (tanh):

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{2}$$

However, the first derivative of the hyperbolic tangent $tanh'(a) = 1 - tanh(a)^2$ has a stronger gradient in comparison to the first derivative of the logistic sigmoid function $\sigma'(a) = \sigma(a)(1 - \sigma(a))$, which can be useful since the backpropagation algorithm, which is introduced in Section 3.1.3, relies on the activation function's gradients. Stronger gradients imply faster parameter convergence due to larger, steeper learning steps.

Hard Hyperbolic Tangent (hard tanh):

$$h(a) = max(-1, min(1, a)) \tag{3}$$

Derivatives of the activation functions arise due to specific drawbacks of the original function, e.g., limiting behavior in training due to the range the real values are mapped to, or computational performance gains during the backpropagation algorithm. The derivative for the hard hyperbolic tangent, as one can deduce from graph (b) in Figure 10, is 0 for $-1 \leq a$ and $a \geq 1$, and 1 for values in $a \in [-1, 1]$, which computationally is cheap and easy to implement. Otherwise, it has similar characteristics as the tanh activation function.

Softplus:

$$h(a) = ln(1 + exp(a)) \tag{4}$$

The softplus function, for values $a \gg 1$, has the characteristic $h(a) \simeq a$, i.e, h has approximate equality and proportionality for positive, large values a helping to alleviate the vanishing gradients problem which we introduce in Section 3.1.3.

Rectified Linear Unit (ReLU):

$$\mathrm{h}(a) = \max(0, a) \tag{5}$$

The rectified linear unit function (ReLU) is even simpler. Empirically, it is the best-performing activation function, and on top of that, it is extremely cheap to compute. The derivative is 1 for $a > 1$ and defined[22] to be 0 at $a \leq 0$. Consequently, neurons can potentially get no backpropagated error signals, effectively halting the learning process for those neurons. In extreme cases, neurons with consistently negative activations may become dead neurons that

[22] undefined at $a = 0$ but this is ignored in practice (Bishop & Bishop, 2024)

remain inactive throughout training and do not contribute to the model's predictive power[23].

Leaky Rectified Linear Unit (leaky ReLU):

$$\mathrm{h}(a) = max(0, a) + \alpha\ min(0, a),\ \text{where } 0\ < \alpha <\ 1. \tag{6}$$

A modification of ReLU that seeks to avoid the issue of dying neurons is called a leaky ReLU. It has nonzero gradients for $a < 0$, which ensures that there is a signal to drive training. The absolute function denoted as (f) in Figure 10 is a modification of leaky ReLU where α is set to -1, in which case $h(a) = |a|$. It is worth noting that α is a trainable parameter, allowing the artificial neural network to determine how strong these inhibiting effects are. To further emphasize ReLU's practical relevance, Bishop & Bishop remark

> *... the introduction of ReLU gave a big improvement in training efficiency over previous sigmoidal activation functions (...) [M]any practical applications simply use ReLU as the default unless the goal is explicitly to explore the effects of different choices of the activation function.* (Bishop & Bishop, 2024, p. 185)

In Figure 6, the activation function's f input is $\vec{x}\vec{w}^t$, which is the scalar product $\langle \vec{x}, \vec{w} \rangle$ and therefore denoted with the large sigma symbol Σ. The vector notation is a common choice since it is much shorter than using the many indices. Figure 5 depicts (x_1, x_2, x_3) weighted by (w_1, w_2, w_3). The notion of $\langle \vec{x}, \vec{w} \rangle = \sum_{i=1}^{3} w_i \cdot x_i$ does expand to the general case for other architectures. When handling multiple neurons in multiple layers we need to introduce subscript j and superscript (m) to map the necessary additional information to their respective mathematical expression. The summarized weighted input is used to determine with the help of the *activation function* if a specific threshold Θ is met. The hidden layer neuron can be equipped with a bias term dampening or lightening (depending on the bias magnitude and sign) the neuron's activation. If the bias $b \neq 0$ the input for the activation changes to $xw^t + b$. There are different ways to implement the bias term. One way is to introduce a constant input signal (per biased layer l), e.g., a $\text{neuron}_0^{(l-1)}$ that is connected to the biased output $\text{neuron}_j^{(l)}$ with a respective weight $b_{0,j}^{(l-1)}$. Depending on the goal and architecture, the bias vectors $b^{(l)}$ can be a learnable parameter or not. Another way is to configure the activation function in such a way that a implicit general bias is present. E.g., shifting the activation function left or right has an overall dampening or lightening effect on the activation behavior of neurons. An activation function centered around zero can be called unbiased. If a specific threshold θ is met, meaning $f(xw^t + b) \geq \theta$, the activation

[23] known as dying ReLU problem

functions result is communicated as the output. Depending on the activation function or architecture of the network, θ might not be needed. While traditional neural networks use threshold-based activation, modern neural network architectures, such as those based on rectified linear units (ReLU), do not explicitly use a threshold parameter. Instead, they rely on the properties of the activation function itself to determine the firing behavior. With classical, non-leaky ReLU, e.g., the neuron fires if the input is greater than zero and remains inactive otherwise, making any θ obsolete.

Output Layer After the activation function of a hidden layer is computed, the value is transported along the path towards the final output layer. In Figure 5 the output $\hat{y}$ is on the right-hand side. The final equation for our example looks like this: $f(\vec{x}, \vec{w}) = f(\langle \vec{x}, \vec{w} \rangle) = f(\sum_{i=1}^{3} w_i^t \cdot x_i) = \hat{y}$. Lets, e.g., use $(1, 2, 3)$ as our inputs $\vec{x}$ and $(0.2, 0.4, 0.6)$ as our weights $\vec{w}$. For our activation function, we use ReLU $h(a) = max(0, a)$.

$$\begin{aligned} \hat{y} &= max(0, 1 \cdot 0.2 + 2 \cdot 0.4 + 3 \cdot 0.6) \\ &= max(0, 2.8) \\ &= 2.8 \end{aligned} \tag{7}$$

Summary of the Intuition Compared to the biological counterpart of brain neurons the artificial input layer mimics a biological neuron receiving stimuli at its dendrites. The activation potential is then collected at an artificial neuron in the hidden layer, where a threshold has to be surpassed for the hidden layer neuron to "fire". The neuron's incoming weights control the magnitude of the activation potential while the strength of the signal that is directed towards the artificial output neuron is influenced by the choice of activation function, which could be interpreted as the shape of the axon and synaptic terminal. The construction of Artificial Neural Networks seems complex. The different notations and representations have an intimidating effect but when broken down into the essential components, the complexity breaks down also into repetitive and small calculations. In the next chapter, we will learn more about these calculations.

3.1.3 Artificial Neural Networks: Expansion to Generality

Based on the examples and intuition we built in the previous Section, we are now going to introduce a more rigorous notation and notion of artificial neural networks. On top of that, we are explaining how exactly the *weights* converge by introducing the concept of *training*

and *backpropagation* of neural networks and the notion of *learning* in Section 3.1.3. In general, neural networks can have arbitrary architectures, much like electronic grids. On top of that, new methods and ideas are added to build even more complex relationships between the nodes and layers, such as residual connections between layers and neurons. Specifically, we introduce recurrent neural networks in Section 3.2.2, the self-attention mechanism3.2.2, and finally the transformers in Section 3.2.3.

Mathematical representation of Artificial Neural Networks As introduced in the previous section, an artificial neural network can be viewed as a function $f : \mathbf{X} \mapsto \mathbf{Y}$. Depending on the intricacies of the network architecture, the function f itself can be a composition of different functions g_i (functions of the g-th layer), which themselves may again be compositions of h_j (functions of the j-th layer) and so on. We can extend the notion to any finite number L of layers *(network depth*[24]*)*, in which layer l computes the following function:

$$\mathbf{z}^{(l)} = h^{(l)}(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)}) \tag{8}$$

where $h^{(l)}$ denotes the activation function associated with layer l, $\mathbf{W}^{(l)}$ denotes its weight matrix. $\mathbf{z}^{(0)} = \vec{x}$ is the networks input vector and $\mathbf{z}^{(L)} = \hat{\mathbf{y}}$ the output vector. Thus, the reconstructing function $\hat{f}(\vec{x}, \theta) = \hat{Y}$, where θ stores the networks parameters such as depth L and weight matrix $\mathbf{W}$, has the following form:

$$\begin{aligned} \hat{f} &= \mathbf{z}^{(L)} \\ &= h^{(L)}(\mathbf{W}^{(L)}\mathbf{z}^{(L-1)}) \\ &= h^{(L)}(\mathbf{W}^{(L)}h^{(L-1)}(\mathbf{W}^{(L-1)}\mathbf{z}^{(L-2)})) \\ &= h^{(L)}(\mathbf{W}^{(L)}h^{(L-1)}(\mathbf{W}^{(L-1)} \dots h^{(1)}(\mathbf{W}^{(1)}\mathbf{z}^{(0)}) \dots)) \end{aligned} \tag{9}$$

These functions do not necessarily have the same parametrization or constraints. The compositional nature of arbitrarily complex neural networks leads to graph representations both mathematically and for practical implementation in the form of computational graphs. We without further explanation introduced a graph representation of neural networks in Figure 5, while trying to build an intuitive foundation for the structure of neural networks and their recursive nature. Taking a look at the smallest unit of a neural network, we observe that any unit can indeed be represented as a graph. Mathematically, graphs are described as follows:

[24]For $l \gg 2$, these architectures are also called deep neural networks.

Definition 3.1 (Graph)**.** Let V be a set of vertices and E be a set of edges. Then, a *graph* G can be represented as a pair $G = (V, E)$.

Definition 3.2 (Edge)**.** An *edge* $e \in E$ is a tuple $e = (u, v)$ where vertices $u, v \in V$, indicating that there is a directed path [25] from u to v.

Graphs can have many characteristics and attributes. For representing neural networks, we define the following:

Definition 3.3 (Weighted Edge)**.** An *weight* $w \in \mathbb{R}$ can be introduced by expanding $G = (V, E, w)$ to a triple, where $w : V \times V \mapsto \mathbb{R}$.

Definition 3.4 (Vertex Threshold)**.** Furthermore, $b : V \mapsto \mathbb{R}$ introduces *vertex thresholds* (bias), expanding the graph to $G = (V, E, w, b)$.

Additionally, we define a vertex attribute we call *state* $h(v) \in \mathbb{R}$ to each vertex $V \in G$. We can manipulate the state of a vertex in our graph mathematically transforming $h(v)$ arbitrarily, e.g., $h(v) + 5$, which simply adds 5 to vertex v's state. In artificial neural networks, we want information to flow from a specific point of input to a specific point of output. Hence, we introduce a set vertices and call them *inputs* which we can connect to specific vertices in our graph G.

Definition 3.5 (Input)**.** An *input* to a graph is a subset $G \subset G^* = (V^*, E^*, w^*, b^*)$, where $V^* \cap V = \emptyset$ and $\neg\exists\ u^*, v^* \in V^* : (u^*, v^*) \in E^*$.

Note that input vertices also have *states* $h(v) \in \mathbb{R}$. However, the state of the *input vertices* are concrete instances i of $h(v^*) \in \mathit{Inputsignal}_i$ with i representing, e.g., the sampling frequency or number of features of the input signal. Input G^* can be interpreted as one instance or row of data, that we serve to our neural network as input. By constructing *edges* $E_{input} \subset E = \{(u, v) | u \in V^*, v \in V\}$ we can connect the inputs to the neural network graph. For modeling the *output* of the neural network, we use the internal state $h(v)$ of nodes in graph G and define transformation a on it:

Definition 3.6 (Output)**.** $h'(v) = a(w \cdot h(v) + b)$,

We call a activation function and $h'(v)$ output of vertex v. The graph representation encapsulates arbitrary artificial neural network architectures. As long as the graph design specifies the defined requirements, we can construct any sub-graph in arbitrary detail and complexity,

[25] Note: e not being a tuple but a set $e = \{u, v\}$ would indicate an undirected graph

and finally connect it into one large complex neural network architecture represented by that graph. Note that we can use the outputs of sections for inputs of others by simply accessing the *state* of the output vertices.

Computational Graph An additional representation of neural networks is the representation of neural networks by their computation graph. A computation graph is a representation of an arbitrary mathematical computation as a graph. It is a directed acyclic graph (DAG) in which nodes correspond to mathematical operations or (bound) variables and edges correspond to the flow of intermediary values between the nodes. The graph structure defines the order of the computation in terms of the dependencies between the different components. The graph is a DAG and not a tree, as the result of one operation can be the input of several continuations (Goldberg, 2017, p. 51). Operations can range from simple arithmetic operations to more complex functions, such as activation functions like ReLU or a sigmoid function. Figure 3.1.3 shows a simple example of a computational graph.

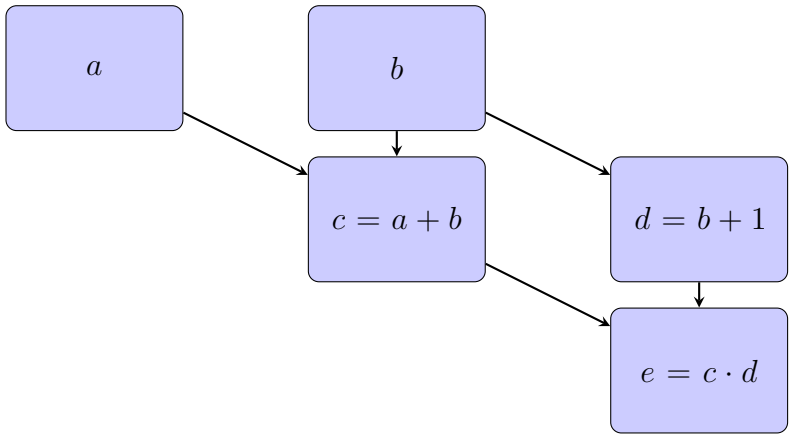


Figure 11: Computational Graph. Source: own illustration based on (Olah, 2015)

In Figure 3.1.3 the computation $(a + b) \cdot (b + 1)$ is expressed as a computational graph. That representation of neural networks with graphs differs from the representation as their computational graph by a margin. The graph representation represents the structure of components in a specific neural network from which we can statically access its information, e.g. by looking at the weights w of the edges or the state $h(v)$ of its vertices at any point in time. On the other hand, the computational graph represents the *flow of data* through the neural network by representing the mathematical operations that occur in the neural network. A computational graph is a much more detailed version that grants access to specific units and variables during the flow of data through the neural network. We have yet to introduce the notion of *flow*. Since a neural network is essentially a long, complex mathematical expression,

it can be represented as a computation graph. Dedicated software libraries and APIs, such as TensorFlow and Pytorch are making the task of constructing such graphs possible, which is why representing components as nodes in a graph has high practical value.

Flow of Data The flow of data happens during both the forward and backward passes. A forward pass refers to the evaluation of the network state at time t and corresponding inputs $G^*(t)$. The forward direction is trivial and occurs when evaluating layer for layer starting with the weighted input $\mathbf{x}$ in the first hidden layer and their neurons and activation functions. In Definition 3.6, we defined $h^{'}(v)$ to be our outputs. The forward flow takes $h^{'}(v)$ and broadcasts it to the nodes v^* connected to v dependent on the weight $w(v, v^*)$. Each of the vertices v^* gathers these incoming values on which then $h'(v^*)$ is computed and again broadcast to the outgoing paths v^* is connected to. That procedure stops when the vertices do not have any outgoing edges, in which case the state of these final vertices is called *final output*. The procedure is one *feed forward* evaluation also called *forward pass*. Since each node's output depends only on itself and on its incoming edges, it is trivial to compute the outputs of all nodes by traversing the nodes in topological order and computing the output of each node given the already computed outputs of its predecessors. Based on (Goldberg, 2017, p. 51), in a computational graph of N nodes, we associate each node with an index i according to their topological ordering.

Definition 3.7 (Forward pass)**.** Let f_i be the function computed by node i (e.g., multiplication, addition, etc.). Let $\pi(i)$ be the parent nodes of node i, and $\pi^{-1}(i) = \{j | i \in \pi(j)\}$ the children nodes of node i (these are the arguments of f_i). Now, $\upsilon(i)$ is the output of node i, that is, the application of f_i to the output values of its arguments $\pi^{-1}(i)$. For variable and input nodes, f_i is a constant function, and $\pi^{-1}(i)$ is empty. The computation graph *forward pass* computes the values $\upsilon(i)$ for all $i = 1, \ldots, N$.

Algorithm 1 Computation graph forward pass.

for $i = 1$ to N **do**
  $a_1, \ldots, a_m \leftarrow \pi^{-1}(i)$
  $\upsilon(i) \leftarrow f_i(\upsilon(a_1), \ldots, \upsilon(a_m))$
end for

The backward pass introduces the opposite to the forward flow of information and with it comes the notion of learning. We have not introduced the reason the *weights* w converge and why $\hat{f}(w)$ paired with $\mathbf{X}$ is a good approximator for $\mathbf{Y}$. Initially, all weights w are initialized randomly or based on a specific heuristic, e.g., weights from a prior related previously trained task. Most likely any input $\mathbf{x}_i$ is not going to reconstruct the respective signal $\mathbf{y}_i$ very well

with $\hat{f}(\mathbf{X}, \theta)$ without some amount of *training* the parameters θ which include the *weights* $\vec{w}$.

Training a neural network For neural networks to *learn*, we have to provide an objective goal. Having a goal presents us with the opportunity to define if our neural network reached the goal or not, or introduce a concept describing how well or bad a network performed while trying to reach a specific goal. This goal is called the objective function. As long as we can mathematically express $\hat{f}(x, \theta)$ does something better or worse with respect to a change in θ, we can base an optimization approach on that expression. There are different ways to formulate an objective function for different domains of deep learning. In the supervised learning context that kind of function is commonly known as loss function or cost function. Machine translation lies within the domain of supervised learning. Objective functions for, e.g., unsupervised learning serve the same purpose but their construction differs from objective functions in supervised learning. For unsupervised training, it is common practice to construct *scores* which the neural network can improve upon by internalizing behaviour that is connected to positive conditioning. Supervised learning, on the other hand, means that we have access to known (assumed to be correct) input-output signal pairings on which we can test our reconstruction approach to determine how off it is or how well it works. Thus, we construct objective functions as a measure of model performance for the task at hand and, to guide the learning process through optimization. The quantification of 'how far of' or 'how close' can be mathematically represented in a metric.

Definition 3.8. Let X be a set. A *metric* on X is a function $d : X \times X \mapsto \mathbb{R}$ satisfying the following properties for all $x, y, z \in X$:

1. **Non-negativity**: $d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$.

2. **Symmetry**: $d(x, y) = d(y, x)$.

3. **Triangle Inequality**: $d(x, z) \leq d(x, y) + d(y, z)$.

Consequently, we can formulate finding the minimum of a constructed metric as an optimization problem.

Definition 3.9 (Loss Function)**.** Given *metric* d, a loss function is defined as

$$L : \theta \mapsto d\left(\hat{f}(\theta, x), y\right), \tag{10}$$

where (x, y) is the input-output pair and $\hat{f}$ is a reconstruction approach with parameters θ.

Now, $\theta^* = \arg\min_\theta L(\theta)$ describes finding the minimum of the loss function as our optimization

problem as a function of θ. We can now introduce ideas and algorithms that optimize the parameter estimation of a reconstruction model in θ. The optimization procedure that specifically adjusts the coefficients or *weights* for artificial neural networks is called *training*, where the artificial network *learns* a specific set of coefficients. We did specifically construct our problem as a function of θ. In other approaches, such as ensemble learning, not only the model's parameters are constructed as an optimization problem but the *type* of model (reconstruction approach) itself is part of the optimization process. In that case, we are looking for $f^* = \arg\min_f L(f)$ with $f \in \mathcal{F}$, where $\mathcal{F}$ is the set of functions (reconstruction approaches) for which $\hat{f}(\mathbf{X}, \theta) = \mathbf{Y}$. Usually, well studied approaches are compared to one another with specific heuristics as parameter estimations as a result of such an approach. The models are either combined into one large *ensemble model* or the best performing model is chosen as $\hat{f}_{opt}$. In the scope of this thesis, however, we are focusing on introducing artificial neural networks. Other than that, there are optimization approaches that tune parameters, such as model architecture parameters (i.e., depth, width, specific components (i.e., GRU units or RNN units)) or, e.g., the learning rate, amount of training epochs, batch size[26], regularization methods [27], and more. On that level, the neural network optimization is called *hyperparameter optimization* or *hyperparameter tuning* and typically involves a strategy to traverse the space of possible artificial neural network *hyperparameters*, which are a subset of θ, to find a set that works better than others for a given problem. The hyperparameter optimization follows the same loss-function minimization principle. We are not going to take a look at *hyperparameter optimization* as it is not within the scope of the thesis. Instead, we rely on scientifically estimated heuristics that promise good results in a specific scenario. For training a concrete neural net $\hat{f}(w)$, however, we introduce the so-called *backpropagation algorithm*. In supervised learning, after the *forward pass* the neural network produces an estimation $\hat{y}$. During the *backward pass* (backward flow of data) the weights of a network are optimized. In the supervised learning scenario we know for a given input its corresponding

[26]it is common practice to not update (i.e. train) parameters based on single instances but on batches of data. The batch size can have an impact on the generalization performance of the trained model. Smaller batch sizes provide a more noisy estimate of the gradient, as they are based on fewer samples. This noise can act as a regularizer, helping to prevent overfitting and improving the generalization performance of the model. However, using very small batch sizes can also introduce instability in the training process, as the gradient estimates become more sensitive to individual samples. On the other hand, larger batch sizes provide a smoother estimate of the gradient, which can help converge to a better solution. However, they may also increase the risk of overfitting, especially when the training data is limited Academy (2023). Using a batch size $n > 1$ equals to using the so called stochastic gradient descent (SDG) variant, in which learning the parameters is based on sampling n times to generate a learning step.

[27]dropout layers, i.e., probabilistic on/off-switching of neurons. Using dropout layers has the consequence that the neural network cannot rely on specific neurons to produce an appropriate estimate but instead have to 'work out' how to circumvent the single-point-of-failure notion artificially introduced to the architecture.

true output, the one we try to reconstruct. By determining the difference (distance) between $|y - \hat{y}| = d(y, \hat{y})$ we can derive the magnitude of the reconstruction error of the current model and its current weights. Based on the magnitude of the reconstruction error, we adjust the weights of the neural network to improve the objective function value.

Definition 3.10 (Gradient Descent)**.** Mathematically, this adjustment approach is called *gradient descent* and can be represented as follows:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \nabla \mathbf{L}(w_{\text{old}})$$

where w_{old} is the vector of old weights, w_{new} is the vector of updated weights, η is the parameter controlling the intensity of the nudge (also called learning rate), and $\nabla L(w_{\text{old}})$ is the gradient of the objective function L with respect to the weights w_{old}.

We note the sign of the gradient. Since we want to find smaller values for L, we are moving in opposite direction of the greatest rate of fastest increase, i.e., the gradient of L. The *backpropagation algorithm* that is used in the *backward pass* does just that. The backward pass begins by designating a node N with scalar (1×1) output as a loss-node in the computational graph, which computes $|y - \hat{y}|$. Having access to the forward computation up to that node, the backward computation computes the gradients of the parameters with respect to that node's value.

Definition 3.11 (Backward Pass)**.** Denote by $d(i)$ the quantity $\frac{\partial N}{\partial i}$, i.e., the gradient of computational graph node N with respect to node i. The backpropagation algorithm is used to compute the values $d(i)$ for all nodes i.

The backward pass fills a table of values $d(1), \ldots, d(N)$ as in Algorithm 2 (Goldberg, 2017, p. 54).

Algorithm 2 Computation graph backward pass (backpropagation).

$d(N) = 1$ ▷ $\frac{\partial N}{\partial N} = 1$
for $i = N - 1$ to 1 **do**
  $d(i) \leftarrow \sum_{j \in \pi(i)} d(i) \cdot \frac{\partial f_j}{\partial d_i}$ ▷ $\frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \cdot \frac{\partial j}{\partial i}$
end for

Based on the chain-rule of differentiation, the quantity $\frac{\partial f_i}{\partial i}$ is the partial derivative of $f_j(\pi^{-1}(j))$ with respect to argument $i \in \pi^{-1}(j)$.

Definition 3.12 (Chain Rule)**.** The chain rule states that for a composite function $f(g(x))$,

the derivative is given by:

$$\frac{d}{dx} f\left(g(x)\right) = f'\left(g(x)\right) \cdot g'(x)$$

where f' and g' denote the derivatives of f and g respectively.

In simple terms, the backpropagation starts from the output and calculates the partial derivatives of each parameter, expressing it only based on the gradients of the later layers. The value depends on the function f_j and the values $\upsilon(a_1), \ldots, \upsilon(a_m) = \pi^{-1}(j)$ of its arguments, which were computed in the forward pass. What makes this procedure so outstanding is that the error is distributed proportionately to the node that caused it, i.e., weights that did not contribute towards the magnitude of $|y - \hat{y}|$ stay unchanged while weights that contributed a significant amount to the $|y - \hat{y}|$ will change towards the direction that makes them contribute less in future evaluations proportionally to their contribution. To put it simply, the backward pass allows the neural network to iterativly *learn* how its parameters should be adjusted in order to improve its predictions. In the scheme of the computational graph, we can now define *any node* i as long as we can construct a method to calculate the forward value $\upsilon(i)$ based on the node's inputs and $\frac{\partial f_i}{\partial x}$ for each $x \in \pi^{-1}(i)$. Repeating the forward and backward pass over and over again on input-output pairs is called *training*.

Exploding and Vanishing Gradients In Section 3.1.2 we hinted at *vanishing gradients*. In deep neural networks with many layers it is common for error gradients to either vanish (become exceedingly close to 0) or explode (become exceedingly high) as they propagate through the computational graph Goldberg (2017). When gradients less than 1 are repeatedly multiplied during backpropagation through many layers of a the neural network, they may diminish exponentially. Consequently, earlier layers receive increasingly smaller updates during training, slowing down learning or hindering convergence. For values larger than one, we have exponential growth on the other hand, leading to exploding gradients. Dealing with the vanishing gradients problem is an open research question, and usually successful architectures involve components that address it. Solutions include making networks shallower or stepwise training, i.e., training specific layers and freezing or fixing their weights before continuing to train others, batch normalization (zero mean and unit variance for the current batch of training data) or using specialized neural network components that are designed to assists gradient flow (e.g. LSTM and GRU architectures for recurrent neural networks which are introduced in 3.2.2). Dealing with the exploding gradients, on the other hand, has a simple but very effective solution: norm clipping the gradients whenever they explode, i.e., if their norm exceeds a given threshold. Let $\hat{g}$ be the gradients of all parameters in the network and $||\hat{g}||$ be their L_2 norm. (Pascanu et al., 2013) suggest to set: $\hat{g} \leftarrow \frac{\text{threshold}}{||\hat{g}||} \cdot \hat{g}$ if

$||\hat{g}|| > threshold$.

Summary: Mathematics of Artificial Neural Networks Artificial neural networks are arbitrarily complex, and often look like black boxes, because they provide little explanatory insight into the contributions of independent variables in the prediction process (Olden et al., 2004). Exploring techniques for improving the interpretability and explainability of neural network models on its own is an area of research. In this section, we learned about what for, and how artificial neural networks are constructed. We have introduced the core components and building blocks from which every neural network architecture is derived. On top of that, we introduced a generalized notion of *learning* an objective function by looking at how the network generates output based on input and compares it to a specific reference, and how to formulate such objectives. Based on two representations, the graph and the computational graph of a neural network, we are now able to express any neural network architecture we can think of, and perform arbitrary calculations to estimate parameters for our neural network that sufficiently solve the task we specify. Finally, we addressed common problems when working with artificial neural networks. In the next section, we combine these concepts to introduce the specific neural network architecture, the transformer architecture, that we use in the experiment of the thesis. We will explore more modern and complex artificial neural network components that allow us to understand the components used in modern neural machine translation model architectures.

3.2 Neural Machine Translation

In this section, we will examine the requirements for neural machine translation with respect to artificial neural networks. In the previous Section 3.1, we have learned the concepts of *learning* functions $f(\theta) = \mathbf{Y}$ based on artificial neural networks and their core components. However, we will learn that these core components need to be extended to map the complex relationships between languages. We are going to take a look at the requirements when it comes to working with natural language and artificial networks. What does the data look like? How are words and sentences mapped to mathematical expressions so that a formal optimization problem can be constructed? How can we apply a loss function and metrics on sequences of words? What kind of components are needed in the artificial neural architecture to perform the translation task? Finally, we are going to introduce the modern *Transformer* architecture in 3.2.3 and, based on the prior components, connect the concepts introduced into one large model that we are going to utilize for our translation task. Having access to these elements, we design and propose our experiment in Chapter 4.

3.2.1 Prerequisites for Machine Translation

Translation Natural Language Processing (NLP) encompasses various tasks, such as Named Entity Recognition (identifying and categorising entities like names, organisations, and places), Sentiment Analysis (assessing the tone of text), Text Summarisation, and many more. One of these tasks is Machine Translation. Translation involves finding a semantically equivalent set of symbols in another language, allowing the receiver to understand the original context. The effectiveness of a translation is determined by how closely the semantic meaning is preserved. For instance, describing a picture with spoken words to a blind person or using sign language to convey music to a deaf person are forms of translation.
These examples highlight that translation is not limited to written or spoken languages but involves conveying meaning across different modes of communication. Describing a picture with spoken words to a deaf person is also a translation but fails to achieve the implicit goal of adding to the understanding of the observer compared to without the translation. In computer science, translation typically refers to converting written text from one language to another while preserving semantic meaning. Table 1 shows an example for a translation between the language pair German and English, with German as the source language and English as the target language: We call the symbol-pair source and target language and

Source: German	**Target: English**
Ich	I
Haus	House
Goldenes Tor Brücke	Golden Gate Bridge
Lichtgeschwindigkeit	Speed of Light
Das Haus am See ist schön.	The house at the lake is beautiful.
Ich kann Peter mit dem Fernglas sehen.	I can see Peter who is holding binoculars.
Ich kann Peter mit dem Fernglas sehen.	With binoculars I can see Peter.

Table 1: Translations for different units of symbols, words, phrases and sentences from German to English, and an examples of semantic ambiguity.

specific structured sets of symbols with semantic meaning *token* which represent a word, phrase, sentence or document, depending on the window of context (i.e., other symbols that appear before and after a specific token). The set of tokens is known as the vocabulary of a language. The vocabulary can be derived from a corpus, i.e., a collection of symbols, e.g., a text, multiple texts or books, in which different words, phrases and sentences reside, potentially carrying different semantics with respect to different words, phrases and sentences in their surrounding context and size of the context. E.g., *hitting a bat* either means hitting a baseball-bat with a ball or hitting a animal-bat with something, or something else entirely, depending on the context. Also, as shown in Table 1, a set of symbols in one language

does not necessarily have a corresponding set of symbols in another language. The term *Lichtgeschwindigkeit* translated to English is represented as a phrase *speed of light*, whereas the English phrase *speed of light* could aswell be represented as the German phrase *die Geschwindigkeit von Licht*, which is not an incorrect translation on a semantic level, but rather an unusual German expression for the term *Lichtgeschwindigkeit*.
Furthermore, migration between cultures and regions with differing languages creates the need to name recent inventions, e.g., cell-phones or laptops. These invented words may differ completely within the very same language with respect to the region a person resides in. The domain of linguistics makes these and other problems its subject matter. Finding specific linguistic rules or strategies, i.e., a model to automate a translation between one or more tokens from one written language to another with the help of a mechanical device (usually a computer), is called Machine Translation. As described in Chapter 2, known approaches to Machine Translation can assume the form of either naive rule-based systems, where words in source and target dictionaries are linked via some kind of look-up table, or more sophisticated rule-based systems, where specific linguistically derived grammatical relationships or structures of phrases or sentences are deduced as additional context for a more sophisticated set of rules that further improve the quality of the translation, or statistical models that assume probabilities of known phrases based on frequencies derived from corpora, which maximize the probabilities of the target symbols conditioned on the source symbols, or neural network based models, which estimate coefficients based on a quantity of examples to reproduce linguistic behaviour.

Word Representations In section 3.1 we introduced a number of methods and neural network components working based on calculations on real numbers $x \in \mathbb{R}$. The subjects of machine translation, however, are words and sentences, or textual features related to those, e.g., character- or word counts. This section introduces the concept of representing words and sentences as numerical vectors, which is essential for processing them in machine learning models.

One-Hot-Encoding One simple solution is to represent words and word-based features as one-hot-encoded vectors. E.g., the most simple feature describes the presence of a word in a sequence. For a specific vocabulary $\mathbf{V}$ of size N a specific word $v \in \mathbf{V}$ can be represented as a vector $\vec{u} \in [0, 1]^N$, where 1 and 0 in the N-dimensional vector indicate a specific word from the vocabulary $\mathbf{V}$. The representation is sparse but unique. Sentences and documents can also be represented. E.g., a sentence with ten words results in a vector $\vec{u}$ with 1 in at most ten dimensions and a 0 in the other. Additional information or features can be encoded as

additional dimensions in $\vec{u}$, with 0 and 1 indicating the absence or presence of that particular feature. By construction, the dimensions of the resulting vector $\vec{u}$ are independent from another. This representation is called *one-hot-encoding*. A downside of this representation is the lack of continuous semantic information. While one-hot encoding preserves the categorical nature of words and allows for easy comparison and manipulation, it does not capture any inherent relationships or similarities between words and/or features.

Word Embedding That downside is addressed by instead *learning dense encodings* (feature embeddings). In *dense encodings* the dimensionality is reduced to a fixed constant d and a parameterized model is *learned*. The model's goal is to represent similar entities or similar features as similar vectors. There are two major different approaches to formulate such models. Supervised (task-specific) pre-training and unsupervised training. The supervised, task specific training requires a reference, e.g., vectors from a prior task that was completed satisfactorily. We can either use those vectors for the new task or fine-tune the vectors for our task as an optimization problem. The more common case is the unsupervised approach. The distributional hypothesis (Harris, 1954) states: words are similar if they appear in similar contexts (Goldberg, 2017, p. 118). The different methods for the unsupervised approach all create supervised training instances in which the goal is to either predict the word from its context, or predict the context from the word. One fundamental approach to learning dense representations comes from autoencoders, a type of neural network architecture used for unsupervised learning. An autoencoder consists of an encoder, which compresses the input data into a smaller representation (known as the latent space or bottleneck), and a decoder, which reconstructs the original data from this compressed form. The latent space is a lower-dimensional, continuous space that encodes important features of the input data. By reducing dimensionality, the model forces the encoder to capture the most salient patterns or information about the input, enabling similar inputs to be represented by vectors close together in this latent space. Using this concept of latent space, we can create dense embeddings for words or other features, reducing the high-dimensional representations of inputs while preserving essential relationships. Common methods are Word2Vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014). We are briefly and only vaguely introducing how the Word2Vec method works based on (Bishop & Bishop, 2024, p. 375). The interested reader will find more detailed information in advanced literature in the domain of information retrieval, especially. Word2Vec can be characterized as *self-supervised*. By spanning a context-window around a specific word in a sentence, the context for the word is presented. Now, by *masking* that specific word from a sentence in our data, we can construct a supervised scenario in which, e.g., a neural network, *learns* the masked word as its objective. This way, we obtain

vector estimates for words given a fixed context window [28]. Generally, these embeddings are calculated to represent words in a continuous vector space based on their distributional properties in large corpora of text data. The new vector space has dimension d, which is a user-defined constant, and generally is much smaller than the sparse, one-hot-encoding based on the vocabulary size and additional features (where depending on the language vocabulary size $N \approx 40.000$). Common languages, such as English, French or German, have pre-trained word embeddings. These word vectors are generally context-independent in that a word always has the same vector no matter what context it occurs in.

Remarks on Word Representations Once the words are embedded, the cosine similarity is used to determine the similarity between u_{word_a}, v_{word_b}.

Definition 3.13 (Cosine Similarity)**.** The similarity between two vectors u and v expressed as the angle between the vectors is called the *cosine similarity*:

$$\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}} \tag{11}$$

Meaningful[29] similarities indicate proper embedding and the preservation of semantic information. A result of a proper embbeding is that, e.g., similar words and synonyms can be inferred with the help of the cosine similarity. By simply calculating the similarity from v to all other vectors u and presenting the n most close ones, we can access the closely embedded *word space* around v. Furthermore, for embeddings generated with Word2Vec *analogy-recovery* can been observed[30], e.g.,

$$w_{king} - w_{man} + w_{woman} \approx w_{queen}$$

or,

$$w_{France} - w_{Paris} + w_{London} = w_{England}$$

Although this analogy-recovery task is possible due to the nature of the vector space and the learned word representations, success on a benchmark of analogy task has little to no practical implication for the suitability of an embedding for a specific task (Goldberg, 2017, p. 138).

[28] when the word is given and the context is masked, the method is called skip-grams
[29] from a subjective human perspective
[30] the phenomenon is not exclusive to Word2Vec, however Word2Vec is a known example.

Character- and Subword Encodings Just as we can break down documents into sentences and analyze the position and role of words within those sentences, we can also distinguish between the characters within a word and their relative positions. This approach becomes potentially necessary when dealing with misspellings or words that are not contained within our assumed vocabulary. Similar to the word embedding techniques, we can also formulate the same problem on character-level. Instead now, we get a vector for each letter in the alphabet of the corresponding language. Nevertheless, addressing language at the character level presents significant difficulties due to the weak correlation between the structure (characters) and the meaning (syntax, semantics) of language (Goldberg, 2017). A middle ground between characters and words is breaking up words into *meaningful units*. (Sennrich et al., 2016b) propose subword encodings for *rare words* for languages with complex morphology or in scenarios where handling rare or out-of-vocabulary words is essential.

> *Byte Pair Encoding (BPE) (Gage, 1994) is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. We adapt this algorithm for word segmentation. Instead of merging frequent pairs of bytes, we merge characters or character sequences.* (Sennrich et al., 2016b)

BPE captures common prefixes, suffixes, and stems shared units across words, allowing for more compact and efficient representation of vocabulary. BPE begins with a vocabulary of individual characters. It then merges the most frequent pair of consecutive characters. The user defined vocabulary size determines the number of times the process is repeated (Rothman, 2024). The result is a set of merged characters that can be an individual character, subwords, or words. BPE and the Google's derivative/implementation SentencePiece (Kudo & Richardson, 2018) have become a standard and baseline in a variety of tasks, including neural machine translation. The output of the BPE algorithm is a model and a vocabulary. The model describes the merging operations performed during training and indicates how the text has to be transformed so it is represented as a so called subworded text, in which words are eventually seperated into their corresponding subword-units. The vocabulary represents the atomic units from which text, using that specific BPE model, can be built. The numerical representation is the subword unit's position in the BPE vocabulary (sometimes referred to as (*hidden*) input ID). Note that this is not the vector representation that satisfies the dimensions of the model's neural network architecture. The vector representation for a specific input token is a learned parameter with respect to a specific linear mapping that corresponds to the meaning of the word. In other words, finding the specific linear mapping to represent inputs as vectors is part of the neural network (called the *embedding*). Once the

text is *tokenized*, i.e., turned into its subword representation based on a BPE model, it is presented to the neural network architecture as its input. Usually, the output of a neural network prediction is again in tokenized form. To reverse or undo the *tokenization*, Sennrich et al. propose applying a specific stream editor operation for filtering and transforming text (commonly known as *sed*) where the whitespaces and special characters that separate the words into their subword units are filtered to restore words from subwords. Alternativley, the SentencePiece library[31] implements a lossless decompression method to revert the subwording process.

Special Tokens: UNK, EOS, SOS and PAD During the encoding, special sequences are reserved for specific operations during the training and inference of the neural network. *UNK* signifies tokens that are unknown or out of vocabulary, allowing the model to handle previously unseen or ambiguous input. *EOS* marks the end of a sequence. Conversely, *SOS* denotes the start of a sequence. By incorporating these special sequences into the encoding scheme, the neural network gains enhanced adaptability, robustness, and efficacy in processing and generating sequential data, which is important in the context of language translation. The *PAD* token is a technical token used to shift sequences in time or fill up space in case it is required for specific matrix operations. They are specified and designed by the user and are context-specific. For more complex experiments, researchers may introduce novel tokens to, e.g., provide additional context to specific vocabulary units or positions in a sentence. One example of user-specified novel tokens is the introduction of tokens that imply the translation target when training with multiple source languages. We mentioned this example in the literature review in Section 2.

Summary: Prerequisites for Machine Translation After applying a specific encoding to the textual data, we are left with representatives from a vector space (Word2Vec, GloVe) or a numerical (positional) representation representing specific units (BPE). Depending on the encoding goal and algorithm, the level of detail may vary between units or a combination of units representing phrases *(e.g., New York$_{(1)}$)*, words *(e.g., New$_{(1)}$ York$_{(2)}$)*, characters *(e.g., N$_{(1)}$ e$_{(2)}$ w$_{(3)}$ Y$_{(4)}$ o$_{(5)}$ r$_{(6)}$ k$_{(7)}$)* or something in between *(e.g., New$_{(1)}$ Yo$_{_(2)}$ _rk$_{(3)}$)*, and even as fine-grained as literal numerical byte-encoding. These units are called tokens, and the procedure that turns words into tokens is called tokenization. The component within a larger neural network architecture is usually called the *tokenizer*. The space in which these tokens reside from the perspective of the neural network is known as an embedding. Through

[31] GitHub/SentencePiece `https://github.com/google/sentencepiece`

the embedding process, textual data is transformed into a structured numerical format that conforms to the requirements of the specific neural network architecture, paving the way for subsequent analysis and mathematical modeling. These units are the representatives of the textual data that flows back and forth through the artificial neural network. Ultimately, our goal is to decipher numerical units back into subwords or words to obtain a meaningful output with respect to our task. Interestingly, the computer never really sees words, only the numerical representation of specific vocabulary which is contained in a given corpus[32]. We will learn more about the process in the subsequent sections.

3.2.2 Neural Network Components for Neural Machine Translation

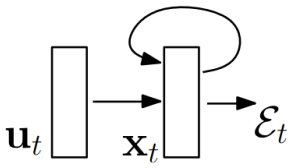


Figure 12: Schematic of a recurrent neural network. The recurrent connections in the hidden layer allow information to persist from one input to another. Source: (Pascanu et al., 2013).

Viewing sentences as *sequential* input signals introduces temporal dependencies. These temporal dependencies require the introduction of additional artificial neural network components, namely recurrent neural networks. Recurrent neural networks (RNN) were introduced as a model in the 1980s by Rumelhart et al., 1986; Elman, 1990; Werbos, 1988 (Pascanu et al., 2013). Their architecture resembles that of a conventional multilayer perceptron, with the distinction that it includes time-delayed connections among the hidden units. As depicted in Figure 12, these connections form a circular path back to the hidden layer x_t to serve as an additional input for that layer in time step_{j+1}. These links allow the network to preserve information from previous inputs, thus identifying temporal relationships between events that may be distant in time — a key feature for effective time series learning (Pascanu et al., 2013). Werbos phrased it in simpler terms:

> *[Predictions)]at time t will be more accurate if we can account for what we saw at earlier times.* (Werbos, 1990)

Werbos elaborates the concept in a network diagram shown in Figure 13. Reading the diagram

[32]Hence the notion of AI *learning* or *understanding* language is debatable.

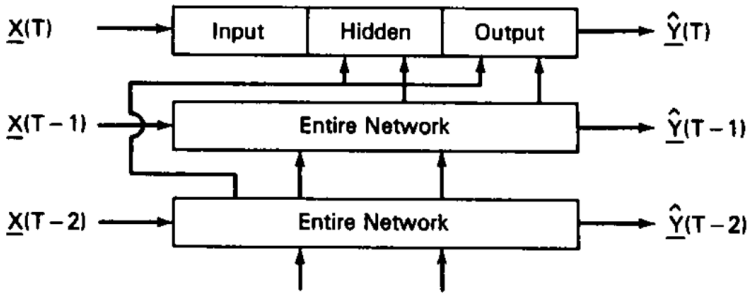


Figure 13: Generalized network design with time lags. Source: Werbos (1990)

from bottom to top, we observe the influence earlier time steps have on the ones in the future. The diagram shows the same *entire network* at three distinct time steps. Three distinct input signals on the left $\underline{X}(T-2), \underline{X}(T-1), \underline{X}(T)$ produce $\hat{\underline{Y}}(T-2), \hat{\underline{Y}}(T-1), \hat{\underline{Y}}(T)$ respectively based upon some $f(\theta)$ (annotated as *entire network*). It is shown that previous states of the network influence both the hidden and the output layers of the future states. The goal is to control the magnitude of that influence with respect to both the moment of time of a specific input and input itself. By *unrolling* the generic representation as depicted in Figure 12 into the repetitive theme depicted in Figure 13, we see that that a RNN is a deep neural network that furthermore can be mapped to a large computational graph with somewhat complex nodes, in which the same parameters are shared across many parts of the computation, and additional input is added at various layers Goldberg (2017).

Recurrent Neural Networks A generic definition of a recurrent neural network is given by

Definition 3.14 (Recurrent Neural Network Component)**.** Let $\mathbf{u}_t$ be an input layer (vector) and $\mathbf{x}_t$ be the state for time step t. A recurrent neural network component is defined by

$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta) \tag{12}$$

where θ is the current neural network parameters (weights). Deconstructing the recurrent neural network parameter component, we observe:

$$\mathbf{x}_t = \mathbf{W}_{rec} h(\mathbf{x}_{t-1}) + \mathbf{W}_{in} h(\mathbf{u}_t) + \mathbf{b} \tag{13}$$

where $\mathbf{W}_{rec}$ is the recurrent weight matrix constructed by the circular, recurring weighted paths and $\mathbf{b}$ is the bias. The initial input $\mathbf{x}_0$ is either set to 0, learned or constructed by the

user, and h is the hidden layer's activation function.

In section 3.1.3, we claimed that as long as we can construct a method to calculate the forward value $v(i)$ based on the node's inputs and $\frac{\partial f_i}{\partial x}$ for each $x \in \pi^{-1}(i)$, we may introduce any kind of node to the neural network architecture. The forward pass is trivial and its evaluation is shown in equation 13. As Werbos notes, backpropagation can be applied to any system with a well-defined order of calculations. For the backward pass, all we need to do is to create the *unrolled* computation graph for a given input sequence, add a loss node to the unrolled graph, and then use the backward (backpropagation) algorithm to compute the gradients with respect to that loss. Unrolling the network architecture in this way introduces the derivative of Werbos of the backpropagation algorithm called *backpropagation through time* (BTT). BTT extends the backpropagation algorithm so that it applies to dynamic systems. This allows us to calculate the derivatives needed for optimizing an iterative analysis procedure, a neural network with memory, or a control system that maximizes performance over time Werbos (1990). Figure 14 depicts a more annotated version of 13. Another insightful representation

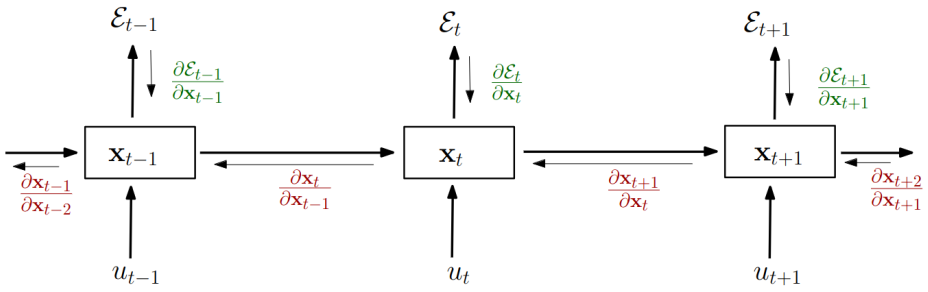


Figure 14: Unrolling recurrent neural networks in time by creating a copy of the model for each time step. We denote by $\mathbf{x}_t$ the hidden state of the network at time t, by $\mathbf{u}_t$ the input of the network at time t and by ε_t the error obtained from the output at time t. Source: (Pascanu et al., 2013)

is given by (Goldberg, 2017, p.164-166).

Remarks on RNN On their own RNN's are not able to produce models that solve $f(\mathbf{X}, \theta) = \mathbf{Y}$. Rather, RNN's serve as trainable components in a larger neural network. Also, the supervision signal is not applied to the RNN directly, but through the larger network.The final prediction and loss computation are performed by that larger network, and the error is back-propagated through the RNN. This way, the RNN components learn to encode properties of the input sequences that are useful for the further prediction task (Goldberg, 2017). One large drawback is that RNNs error signals tend to vanish or explode due to the recursive

nature of the component. During backpropagation through time, gradients are multiplied at each time step as they are propagated backward through the network. If these gradients are less than 1, they can diminish exponentially as they traverse through numerous time steps. Conversely, gradients can become exceedingly large during backpropagation through time. This phenomenon occurs when the gradients are greater than 1, leading to exponential growth as they propagate backward through the network. We already discussed the consequences and some possible solutions of vanishing and exploding gradients in section 3.1.3. The overarching problem has inspired specific architectural refinements for recurrent neural networks that we introduce in the next section 3.2.2.

Sequence-to-Sequence Currently, we have introduced components that predict y_t based on $x_0 \ldots, x_{t-1}, x_t$. By varying t between 0 and t, we retrieve a sequential output signal $\mathbf{y} = (y_0, \ldots, y_t)$. A common approach to Neural Machine Translation are RNN based sequence-to-sequence models [33]. Translation, from a probabilistic perspective, aims to maximize the conditional probability of the target sentence $\mathbf{y}$ given the source sentence $\mathbf{x}$: $\arg\max_y p(y|x)$. "In neural machine translation, we fit a parameterized model to maximize the conditional probability of sentence pairs using a parallel training corpus (input and output pairs). Once the conditional distribution is learned by a translation model, given a source sentence a corresponding translation can be generated by searching for the sentence that maximizes the conditional probability" (Bahdanau et al., 2016). Cho et al. propose a RNN Encoder–Decoder neural network architecture that is able to encode a variable-length sequence into a fixed-length vector representation and to decode a given fixed-length vector representation back into a variable-length sequence. They assume RNN can learn a probability distribution over a sequence by being trained to predict the next symbol in a sequence. In that case, we can infer the output at each time step t by the conditional distribution $p(x) = \prod_{t=1}^{T} = p(x_t|x_{t-1}, \ldots, x_0)$ They suggest that from this learned distribution, it is straightforward to sample a new sequence by sampling a symbol at each time step (Cho et al., 2014), proposing a valid model to achieve machine translation.

RNN Encoder-Decoder Cho et al. describe a model that calculates $p(y_1, .., y_{T_y}|x_1, .., x_{T_x})$, where the sequence lengths T_y and T_x do not necessarily have to be of the same size.

Definition 3.15 (RNN:Encoder)**.** The encoder is an RNN that reads each symbol of an input sequence $\mathbf{x}$ sequentially. As it reads each symbol, the hidden state of the RNN changes

[33] other architectures, e.g., a hybrid of an RNN and a de-convolutional neural network have been proposed by Kalchbrenner & Blunsom

according to

$$h_{(t)} = f(h_{(t-1)}, x_t), \tag{14}$$

where f is the activation function.

After reading the end of the sequence (marked by an end-of-sequence symbol), the hidden state of the encoder RNN is a *summary* $\mathbf{c}$ of the whole input sequence.

Definition 3.16 (RNN:Context Vector)**.** A vector generated from a sequence of hidden states $h_t \in \mathbb{R}^n$ at time $t = 1, \ldots, T_x$ is called *context vector*

$$c = q(\{h_{(1)}, \ldots, h_{(T_x)}\}) \tag{15}$$

where q is some non-linear function. The context vector can also simply, e.g., be $c = h_{(T)}$.

The summary $\mathbf{c}$ of the input sequence serves as a form of context or additional information that guides the decoding process. It potentially encapsulates the input sequence's features and provides context for generating the output sequence. This conditioning mechanism allows the decoder to generate output symbols that are influenced by both the input sequence and the previously generated symbols.

Definition 3.17 (RNN:Decoder)**.** The hidden state at time t is computed by

$$h_{(t)} = f(h_{(t-1)}, y_{t-1}, \mathbf{c}) \tag{16}$$

The *decoder* is a RNN which is trained to model the output sequence by predicting the next symbol y_t given the hidden state $h_{(t)}$.

$$p(y_t|\{y_{t-1}, \ldots, y_1\}, \mathbf{c}) = g(y_{t-1}, h_{(t)}, \mathbf{c}). \tag{17}$$

Both y_t and $h_{(t)}$ at are conditioned on y_{t-1} and on the *summary* $\mathbf{c}$ of the input sequence. In other words, the decoder defines a probability over the translation y by decomposing the joint probability into the ordered conditionals solving the task of predicting the next word based on the previously predicted words Bahdanau et al. (2016):

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t, |\{y_1, \ldots, y_{t-1}\}, \mathbf{c}) \tag{18}$$

where $\mathbf{y} = (y_1, \ldots, y_{T_y})$. Bahdanau et al. use a RNN to model the conditional probabilities

$$\prod_{t=1}^{T} p(y_t, |\{y_1, \ldots, y_{t-1}\}, \mathbf{c}) = g(y_{t-1}, s_t, c), \tag{19}$$

where g is a potentially multi-layered, non-linear function that outputs the probability of y_t and s_t is the hidden state of the RNN. Without loss of generality, g is often chosen to be a derivative of the so-called *soft max* that for all possible symbols $j = 1, \ldots, K$ calculates their probability with respect to each other symbol[34].

$$\frac{\exp(w_j h_{(t)})}{\sum_{j'=1}^{K} \exp(w_{j'} h_{(t)})} \tag{20}$$

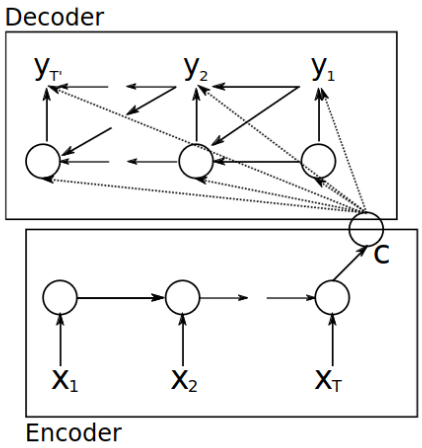


Figure 15: An illustration of a generic RNN Encoder–Decoder. Source: (Cho et al., 2014)

Figure 15 shows a generic encoder-decoder network as proposed by (Cho et al., 2014). The input sequence $x_1, x_2, \ldots, x_T$ produces some state $\mathbf{c}$ which is included as additional input in the decoding phase. Additionally, the figure schematically points at the influence of subsequent outputs y_{t_j}.

Gated Recurrent Units Furthermore, Cho et al. introduce gated recurrent units (GRU) as easier to evaluate and analyse than the Long-Short-Term-Memory neural network components. Long-Short-Term Memory (LSTM) was introduced by Hochreiter & Schmidhuber (1997) to specifically tackle the problem of the error signal being backpropagated through time either

[34] the concept is commonly known as a possible solution for calculating the class probabilities for a multi-class problem

blowing up or vanishing, as well as to find a solution for RNNs overestimating the influence of recent inputs versus non recent inputs (hence the notion of memory). By controlling the flow of gradients, these gated architectures alleviate the vanishing and exploding gradient problems. To solve the difficulty of capturing and retaining relevant signals over long sequences, Hochreiter & Schmidhuber (1997) introduced architectures with memory cells that are designed to retain information over longer time horizons. These memory cells, equipped with gating mechanisms, enable the network to selectively update and access information from previous time steps, allowing it to maintain relevant signals over extended sequences. Additionally, techniques such as skip connections and residual connections facilitate the flow of information through the network, helping to address the issue of information degradation over time. Many of the LSTM mechanisms introduced by Hochreiter & Schmidhuber have revolutionized and inspired the deep learning domain. It is not within the scope of this thesis to discuss all of them in detail. For further information, we refer to (Hochreiter & Schmidhuber, 1997) and (Schmidhuber et al., 2011). The more recent GRU achieve a similar

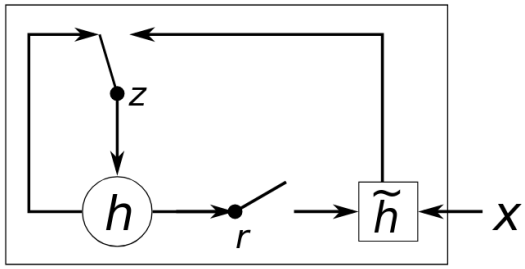


Figure 16: An illustration of the hidden activation function proposed by Cho et al. (2014). The update gate z selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$. The reset gate r decides whether the previous hidden state is ignored. Source: (Cho et al., 2014)

feat. Gating units regulate the flow of information and gradients, allowing the network to selectively update or retain relevant information over time. Cho et al. (2014) called their gate the "Hidden Unit that Adaptively Remembers and Forgets" which was later adapted as GRU. GRU, illustrated in Figure 16, feature two gates: the reset gate (r) and the update gate (z). Cho et al. describe that when the reset gate is close to zero (see Equation **??**), the hidden state is forced to ignore the previous hidden state and reset with the current input only. This effectively allows the hidden state to drop any information that is found to be irrelevant later in the future, thus, allowing a more compact representation. On the other hand, the update gate controls how much information from the previous hidden state will carry over to the current hidden state. This acts similarly to the memory cell in the LSTM network

and helps the RNN to remember long-term information. These gating units counteract the exploding/vanishing gradient problem and accommodate the hidden units of a layer$_j$.

Alignment by Attention Inspired by (Cho et al., 2014), Bahdanau et al. (2016) develop a neural network architecture that jointly learns to align and translate. Bahdanau et al. introduce a novel way to condition on the context vector c. Instead of conditioning on c as a whole, the probability is conditioned on one distinct c_i for each target word y_i. The c_i depend on a sequence of *annotations* $(h_1, \ldots, h_{T_x})$. The *annotations* contain information about the whole input sequence. The new distinct context vector c_i is calculated as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_i \tag{21}$$

The weight α_{ij} of each annotation is computed by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{22}$$

where $e_{ij} = a(s_{(i-1)}, h_j)$. The state at t of the RNN is denoted by $s_{(t)}$.

Definition 3.18 (Alignment Model)**.** Scores e_{ij} that indicate how well the inputs around position j and the output at position i match are called *alignment model*.

> *Intuitively, this implements a mechanism of attention in the decoder.* (Bahdanau et al., 2016)

Summary: Neural Network Components for Neural Machine Translation In section 3.2.2 we have introduced the necessary core components for artificial neural network based machine translation. We highlighted these components by introducing the recurrent neural network component (RNN) in conjunction with sequence-to-sequence modeling. We introduced the encoder-decoder structure of the modeling approach, where a encoder reads input and summarizes, and the decoder takes the summary to produce a translation. We learned about important subtleties when it comes to *learning* with RNNs and the vanishing gradient problem, and how to counteract it with gated units. Finally, we introduced the concept of alignment and attention between input and output sequences that we put in place to achieve a more rich connection between the source and target sequence.

3.2.3 Neural Machine Translation with Transformers

The introduction of the alignment model to the encoder-decoder architecture sparked researchers at Google to come up with an architecture that combines the introduced components for sequence to sequence modeling into a novel architecture. In their paper Attention Is All You Need (2017) they introduce the so called *Transformer*. In this section, we are going to dive into the components of the Transformer architecture. Furthermore, we document a paradigm shift that replaces old components. Based on the previous sections, we have a very good conceptual understanding of key components needed for the machine translation task. This section answers the questions, what the Transformer architecture is and why it is superior to RNN based architectures.

The Transformer Architecture Crucial about the Transformer is that it dispenses of recurrence entirely, which increases the training efficiency *enormously*. In other models, e.g., models with RNN components, the sequential computation of attention based on the hidden states $h_{(t)}$ based on $h_{(t)-1}$ completely preclude parallelization. Transformers, however, introduce *multi-head self-attention* via so called *scaled dot-product attention*. Today, Transformers based on attention have completely superseded RNNs in almost all applications (Bishop & Bishop, 2024, p.358). In Section 3.2.2 we introduced the concept of an alignment model e_{ij} that relies on both input and output sequences by generating context vector c_i based on attention weights α_{ij} and their *annotations* h_i. Novel at the time of publishing, the Transformer architecture, to draw *global dependencies* between input and output, instead uses the so-called self-attention (sometimes also called intra-attention) without using any alignment model between *input and output* sequence. Figure 17 shows the components used in their neural network architecture. The left side of Figure 17 depicts the encoder block of the architecture, the right side the decoder block. All components are introduced in the subsequent paragraphs. Note that the headline of the paragraph indicates the specific component of investigation. Components being part of both the decoder and encoder stack are indicated as such. Furthermore, a color schema indicates mentions of components or their potential synonyms. The components include feed forward network components (blue), multi-head attention components (orange), normalization components (yellow), embedding components (red), a linear projection component (purple) and a softmax component (green). The architecture also utilizes residual connections between specific components.

Transformer: Encoder+Decoder - Input/Output Embedding As mentioned in section 3.2.1, the inputs that are either vector representations of words or IDs with respect to their encoding vocabulary have to be mapped to a model-conforming vector. The Trans-

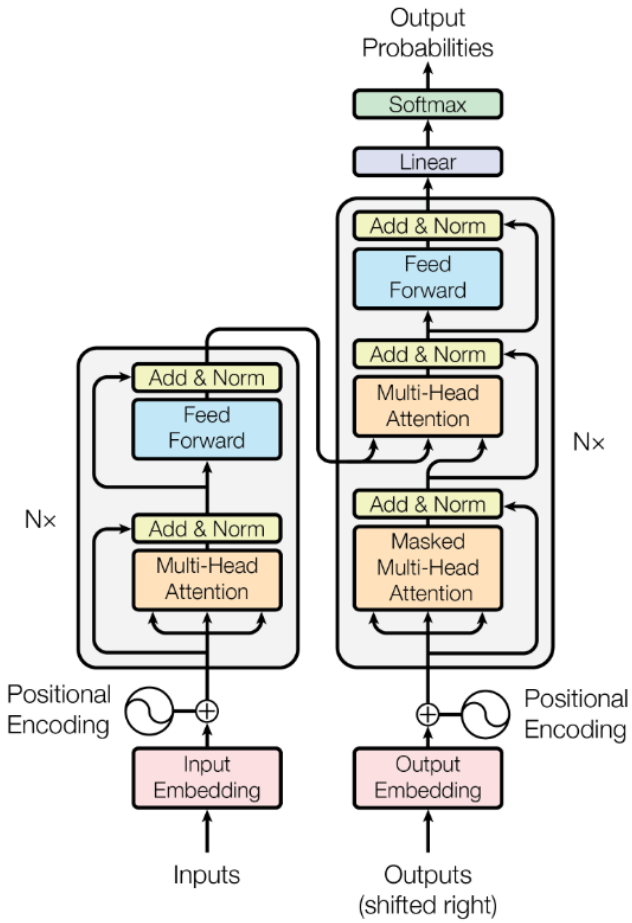


Figure 17: The Transformer - model architecture. Source Vaswani et al. (2017)

former's d_{model} parameter specifies the embedding vector dimension (default [35] $d_{model} = 512$). The mapping itself is part of the neural network's learnable parameters. The resulting d_{model} dimensional encoder and decoder embedding spaces encapsulate the learned linguistic nuances and relations between the encoder input tokens with respect to the decoder tokens.

Transformer: Encoder+Decoder - Positional Encoding Since the Transformer is not using recurrent neural networks, the time component of the input sequence is not naturally encoded by the choice of neural network architecture. Instead, an additional vector is introduced encoding the relative position of the current word in the sequence. The positional encoding procedure is the same for input and output sequences. In Figure 17, the positional encoding is indicated by the circle which encapsulates a sinoid curve positioned above the

[35] *defaults* to the Transformer architecture as proposed by (Vaswani et al., 2017)

input and output embedding. The positional encoding vector has the same dimension as the embedding space. These positional encoding vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence.

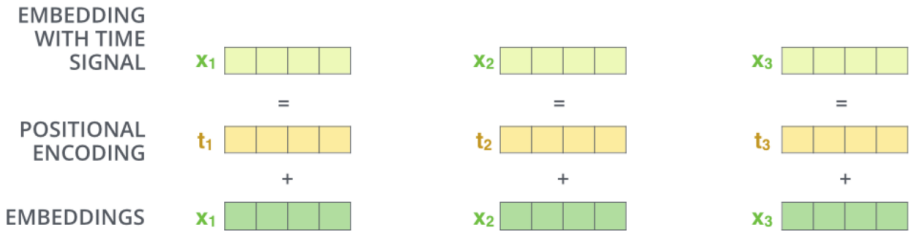


Figure 18: To give the model a sense of the order of the words, we add positional encoding vectors. Source: (Alammar, 2018)

The positional encoding function (Vaswani et al., 2017) propose is:

Definition 3.19 (Positional Encoding)**.** A *Positional encoding* is a function used to embed positional information in order to differentiate between different positions in a sequence.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{n^{\frac{2i}{d}}}\right) \tag{23}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{n^{\frac{2i}{d}}}\right), \tag{24}$$

where in the context of the Transformer architecture $\text{pos} = 0, \ldots, X_T$ is the position (zero-based index) of the input token from sequence X, $d = d_{model}$ the dimension of the embedding space, and i corresponds to the i-th embedding dimension. n is a fixed scalar defined by the user (default: 10,000).

In other words, for each i of the d embedding dimensions we calculate a positional encoding based on the position of the current token within the sequence, resulting in an $1 \times d$ positional encoding vector. By construction, even i are encoded with a sine and odd i are encoded with a cosine. Vaswani et al. hypothesize that by using sinusoidal functions it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training (Vaswani et al., 2017). Another advantage is, that the sine and cosine functions are constrained to $[-1, 1]$, keeping the values of the positional encoding matrix in a normalized range. Furthermore, as the sinusoid for each position is different by construction, we obtain a unique way of encoding each position of a sequence. Consequently, we now have a way of measuring or quantifying the similarity between different positions, hence enabling us

to encode the relative positions of words using these mappings. Note that the positional encoding Vaswani et al. introduce is one of many ways to achieve a similar feat. Now, we shift our focus towards the model's attention mechanism. The attention mechanism conceptually replaces the context vector $\mathbf{c}$ and alignment model e_{ij} of the RNNs sequence-to-sequence model.

Transformer: Encoder+Decoder The encoder and decoder block are composed of a stack of N (default $N = 6$) identical layers. In the encoder stack, each layer has two sub-layers: the multi head attention layer and the feed forward layer. In the decoder stack, each layer consists of three sub-layers: two multi head attention layers and the feed forward layer. The decoder is conditioned on the output of the encoder by ingesting the final encoder hidden state partially into its second multi head attention layer. First, we inspect the Multi Head Attention.

Transformer: Encoder+Decoder - Multi Head Attention The attention mechanism Transformers use can be viewed as a richer form of embedding in which a given vector is mapped to a location that depends on *all other* vectors in the sequence *simultaneously* (Bishop & Bishop, 2024). Previously, with RNNs, we were able to grasp the context of a sentence by evaluating states sequentially, and representing the context as some context vector $\mathbf{c}$. Now, instead, we look at all positions of the sequence and the corresponding vectors simultaneously, and encode additional information accordingly to each vector in sequence independent from time and state. In a sense, we assume an additional encoding step to embed our already embedded words once more, enabling more nuanced meanings for words that, e.g., have ambiguous meaning, enabling differentiation based on the embeddings of the whole sequence. On top of that, networks based on RNNs show weak support for longer sequences and large contexts. Vaswani et al. assume that the additional embedding space and assigning a vector for each token in that space counteracts relying on one context vector $\mathbf{c}$ for one whole sequence. To describe their mechanism, Vaswani et al. borrow both terminology and methodology from the information retrieval domain. They describe their attention as mapping a query and a set of key-value pairs to an output. For each token_i in a sequence, we create a *Query* vector, a *Key* vector, and a *Value* vector based on that token. These vectors can be interpreted as their information retrieval counterpart. For example, when you search for videos on YouTube, the search engine will map your query (text in the search bar) against a set of keys (video title, description, etc.) associated with candidate videos in their database, then present the best matched videos (values). Section 3.2.2 introduces a similar notion. Annotations h can be understood as our values that correspond to a potential match candidate. h_i relevance is

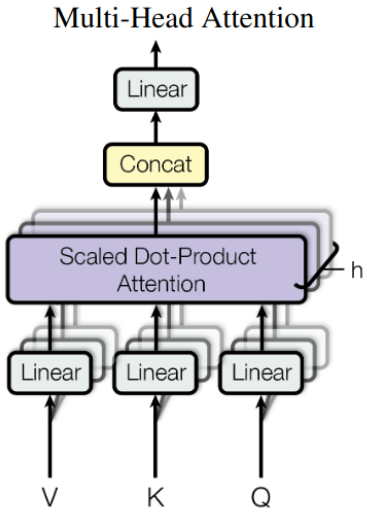


Figure 19: Attention consists of several attention layers running in parallel. Source:(Vaswani et al., 2017)

indicated by weighting with α_i to obtain context vector c_i. In other words α_i is the probability of h_i being relevant for the output sequence in the RNN model. Computationally, calculating this α is expensive. For sequences of size x_{T_x} and y_{T_y}, we have to evaluate the whole artificial neural network $x_{T_x} \cdot y_{T_y}$ times to find all attention scores e_{ij}. A more efficient way is to first project state s and annotations h onto a common space, then choose a similarity measure (metric) as the attention score (dontloo, 2019).

$$e_{ij} = f(s_i)g(h_j)^T \tag{25}$$

Equation 25 is essentially the approach proposed by (Vaswani et al., 2017), where the two linear projection vectors are called query (for decoder) and key (for encoder) and the similarity measure is given by their *dot product*. But why dot product? We already introduced another measure of similarity, the cosine similarity for vectors in equation 3.13. The cosine similarity expresses the angle between two vectors. Cosine similarity ranges from $[-1, 1]$, where 1 indicates that the vectors are pointing in the same direction, -1 indicates that they are pointing in opposite directions, and 0 indicates orthogonality (perpendicularity) between the vectors. The dot product of two vectors computes the similarity by quantifying how much two vectors point in the same direction.

Definition 3.20 (Dot Product)**.** The dot product of two vectors $\mathbf{a} = (a_1, a_2, \cdots, a_n)$ and

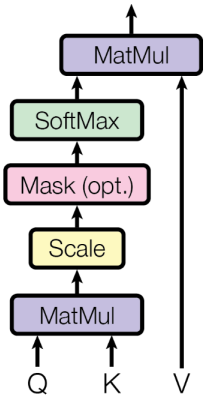


Figure 20: Scaled Dot-Product Attention. Source:Vaswani et al. (2017)

$\mathbf{b} = (b_1, b_2, \cdots, b_n)$, specified with respect to an orthonormal basis, is defined as

$$a \cdot b = a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_n \cdot b_n = \sum_{i=1}^{n} a_i \cdot b_i \tag{26}$$

The dot product is a measure of the projection of one vector onto the other that is *not* irrespective of magnitude, i.e., preserving more information than the cosine similarity.

Transformer: Multi Head Attention - Scaled Dot Product Attention Vaswani et al. call their attention mechanism *scaled dot product attention*. They describe any input can be represented as *query* and *key* with dimension $d_{key} = d_{query}$ and value of dimension d_{value}. The dimension usually is smaller than the dimension of d_{model} (dimension of the input embedding). Figure 19 has a 3-D depth to it, indicating that h (default=8) attention layers run in parallel. Vaswani et al. report finding it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_{key}, d_{key} and d_{query} dimensions, respectively, where d_{key} and d_{value} are much smaller than d_{model}. As a heuristic they propose $d_{key} = d_{value} = \frac{d_{model}}{h}$. Important to note is that key, value and query are vectors that are representations of the input encoding by some linear transformation:

$$Key = x_i \times \mathbf{W}_K \tag{27}$$

$$Query = x_i \times \mathbf{W}_Q \tag{28}$$

$$Value = x_i \times \mathbf{W}_V \tag{29}$$

$\mathbf{W}_K \in \mathbb{R}^{d_{model} \times d_{key}}, \mathbf{W}_V \in \mathbb{R}^{d_{model} \times d_{value}}$, and $\mathbf{W}_Q \in \mathbb{R}^{d_{model} \times d_{value}}$, the linear projection

matrices, are learned in the neural network. The matrices and their interplay represent a compatibility function. The compatibility of a token with regard to the rest of a sequence is a vector derived from computing the dot products of the *Query* (current token) with all *Keys* (rest of the sequence), divided by $\sqrt{d_{key}}$ (scale), and apply a softmax function (see 20) to obtain the compatibility weights of the query with all possible keys. The vector normalization with $\frac{1}{\sqrt{d_{key}}}$ is believed to lead to better control over the magnitude of gradients with respect to the softmax Vaswani et al. (2017). The above describes the left track of Figure 20. The output of the attention component is computed as a weighted sum of the *Values*, where the weight assigned to each *Value* is computed by the compatibility function of the query with the corresponding key, that we just described. This describes the right track of Figure 20. As Figure 19 implies, the scaled dot-product attention is calculated in parallel for h attention heads and different estimates for the linear projection matrices each yielding a d_{value} dimensional output with attention scores. Intuitively, multiple attention heads allow for attending to parts of the sequence differently (e.g., longer-term dependencies versus shorter-term dependencies). But the feed-forward layer is not expecting h matrices – it is expecting a single matrix (a vector for each word). Therefore, a final concatenation layer is used to concat the attention scores. Using another learnable projection matrix $\mathbf{W}_{output}$ to linearly project the output back into $\mathbb{R}^{x_{T_x} \times d_{model}}$ (or $\mathbb{R}^{y_{T_y} \times d_{model}}$, depending on the position of the attention layer in the overall architecture). Two similar multi-head attention components can be found in the decoder layer of the artificial neural network on the right in Figure 17. One component is called *masked* multi-head attention and the other simply multi-head attention. Noteworthy about the second (upper) multi-head attention layer in the decoder is that the encoder layer's output is the input for that attention layer (Encoder-Decoder attention). That mechanism established itself as the so-called *Cross-attention* and allows the model to consider information from different parts of the input sequence while generating the output sequence. Unlike self-attention, which focuses within the same sequence, cross-attention enables interactions between the input and output sequences. Here, *Key* and *Value* are computed as projections from the encoder output. The *Query* projection comes from the decoder input that was first directed through the masked multi-head attention layer and the layer normalization.

Transformer: Decoder - Masked Multi Attention Head What do we mean by masking and what is the purpose of it? In terms of connectivity, the encoder is fully transparent and connected to the decoder, leading to proper conditioning on all encoder inputs at all times. During training, we inform the decoder what the complete target sequence looks like by handing the true target sequence to it as its input - shifted to the right by one

position. Therefore, while predicting a word at a specific position, the decoder has available to it the target words preceding that word as well as the target words following that word. This allows the decoder to *cheat* by using target words from future time steps as additional context Doshi (2021). However, to preserve the auto-regressive property, we must not have access to the model target, i.e., the next word in the sequence. In other words, we need to *hide* future decoder input tokens since they *give away* the answer to our problem at the current step. We got rid of the time component all together and we are presenting the whole target sequence at once to the decoder. Now, in combination with the shifting, *masking* is the mechanism that achieves to hide these future tokens. The masked elements which correspond to illegal connections are set to negative infinity, so that softmax turns those values to zero when calculating their attention score.

Transformer: Encoder+Decoder - Feed Forward Neural Network In both the encoder and decoder layer there is a feed forward neural network. On the encoder side, before directing the encoder's output into the decoder multi-head attention layer, the outputs of the self-attention layer are fed to a fully connected feed-forward neural network consisting of two layers with ReLU activation functions. In Vaswani et al. specific architecture, the network depth is 2 and its width, the amount of neurons in each hidden layer, is 2048. The input and output vector of the feed-forward neural network has dimension d_{model}. The same architecture with different weights can be found on the decoder side.

Transformer: Decoder - Linear Projection Layer After the feedforward layer, a layer normalization takes place, from which the decoder stack finally outputs a vector of floats. Now, we want to derive the next word of a sequence. We borrow the same linear projection concept introduced in the attention mechanism. By linear transforming the output vector into a $1 \times vocab\_size$ dimensional vector, called logit vector, we obtain logit values for each entry in our vocabulary.

Transformer: Decoder - Softmax The final Softmax layer of the Transformer takes these logits and translates them via a softmax function into probabilities. Now, we have a probability distribution over our vocabulary, from which we can sample our next word. From here on out, we can choose to always get the most probable word resulting in deterministic predictions, or we choose to randomly sample from our distribution in any other way we specify.

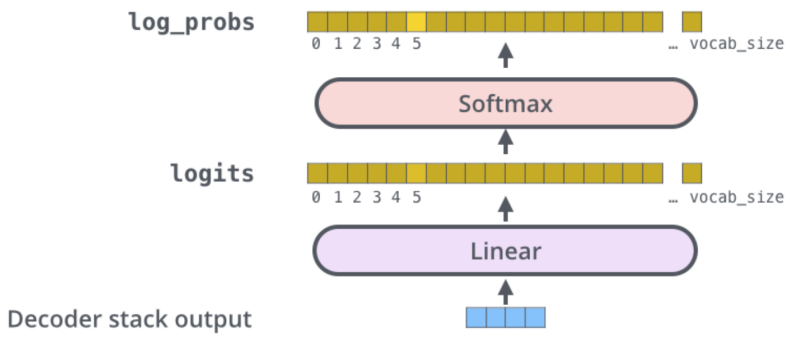


Figure 21: The Vector produced as the output of the decoder stack turned into probabilities. Source: (Alammar, 2018)

Transformer: Encoder+Decoder - Residual Connections The last component left to introduce is the residual connection. In Figure 17 we can find them in the encoder side indicated by the arrow that flows from before the multi-head attention to the normalization layer, and again after that before the feed forward layer to the next normalization layer. The same type of residual connections can be found on the decoder side, respectively. These residual connections *preserve* the inputs, i.e., they are identity preserving components. With the help of residual connections the layers in the neural network architecture can perform arbitrary operations and permutations on the input $\mathbf{x}$ without the need of preserving its identity, because we *connect* the output of those arbitrary operations back together with its identity. A residual connection changes a components mapping from $f(\mathbf{x}) = \mathbf{y} = h(\mathbf{x})$ into

$$f(\mathbf{x}) + \mathbf{x} = \mathbf{y} + \mathbf{x} = h(\mathbf{x}), \tag{30}$$

where f in this context is some neural network component applied to $\mathbf{x}$ and $\mathbf{x}$ is the residually connected identity. The residual block $h(\mathbf{x})$ represents the desired mapping. This operation is only valid if the dimensions of $\mathbf{x}$ and $f(\mathbf{x})$ are the same. Its common for neural network architectures to introduce manipulations to the signal that also change the dimensionality of that signal. To counteract these changes, we introduce linear projection $\mathbf{W}_s$

$$f(\mathbf{x}, \{\mathbf{W}_i\}) + \mathbf{W}_s\mathbf{x} = \mathbf{y} + \mathbf{W}_s\mathbf{x} \tag{31}$$

The residual connections *remind* the representation of what the original state was. Intuitively, we guarantee that the contextual representations of the input tokens represent the original input tokens (Libovický, 2022). Another task of the residual connections is to help mitigating the vanishing gradient problem through residual learning (He et al., 2015). Without the residual connections, a large part of the training signal can get lost during back-propagation.

The summation operations of residual connections form a path in the computation graphs where the gradient does not get lost. (He et al., 2016) show for a residual block harbouring a residual connection:

$$h(\mathbf{x}) = f(\mathbf{x}) + \mathbf{x}$$

When computing the derivative of the loss function L with respect to the input $\mathbf{x}$ of the residual block h:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \mathbf{x}}$$

Using the chain rule we observe:

$$= \frac{\partial L}{\partial h}\left(\frac{\partial f}{\partial \mathbf{x}} + \frac{\partial \mathbf{x}}{\partial \mathbf{x}}\right)$$

Since $\frac{\partial \mathbf{x}}{\partial \mathbf{x}} = 1$, the equation simplifies to:

$$= \frac{\partial L}{\partial h}\frac{\partial f}{\partial \mathbf{x}} + \frac{\partial L}{\partial h} \tag{32}$$

The result from Eq.32 indicate that $\frac{\partial L}{\partial x}$ can be decomposed into two additive terms:

1. $\frac{\partial L}{\partial h}\frac{\partial f}{\partial \mathbf{x}}$ represents the gradient of the loss propagated through the residual block, i.e., through the weight layers.

2. $\frac{\partial L}{\partial h}$ represents the direct propagation of the gradient through the identity connection, i.e., directly without concerning any weight layers.

In summary the residual connections counteract the problem of vanishing gradients, which when addressed has proven to produce better results in the context of deep neural networks in general. Additionally, the computational cost of the residual connection breaks down to a simple addition, which is easy to realize in both forward and backward pass of the computational graph. (He et al., 2016) and others show that empirically residual connection greatly improve the capabilities of neural networks in several applications. (Vaswani et al., 2017) imply that the Transformer architecture and neural machine translation is one of them.

Transformer: Inference But how exactly does the Transformer learn to map between inputs and outputs? How can it predict a sequence based on some input sequence? First of all, we need to distinguish the training procedure from the inference procedure as they are different. At inference, in the beginning at t_0 the predicted output sequence is empty ([]). To populate the output sequence, we need to provide context to the Transformer. We initiate context on both the encoder and decoder side. On the decoder side, we initiate the first token as the *SOS* (*start*

of sentence) token at t_0. The *SOS* token ID is embedded by the Output Embedding, onto it is added the positional encoding vector and then processed in the decoder stack accordingly. Here, in the decoder block, the decoder context meets the encoder context. For the encoder context, we initiate the whole input sequence which is fully transparent to the model already at t_0. At t_0, the whole input sequence is processed by the whole encoder stack to produce what is often referred as the final encoder hidden states. These states are of dimension $1 \times T_x \times d_{model}$, where the first dimension, e.g., 1 denotes one single sequence, T_x denotes the length of that sequence and d_{model} the embedding dimension. In other words, all encoder hidden vectors based on the complete input sequence are used to condition the decoder. Now, with the initial context from the decoder conditioned on the final encoder hidden states, we can predict a probable first token based on the decoder final hidden state. Once we obtain the first predicted token ID by applying the linear mapping (also referred to as language modeling head) and the Softmax layer, we start populating the output sequence. So now, in addition to the initial *SOS* token ID, the new output sequence is used as context that is given to the decoder stack. The same encoder hidden state vectors are re-used to generate outputs populating the output sequence autoregressively until the *EOS* token ID is predicted, which stops the prediction process for that specific sequence. The final decoder hidden state vectors, on the other hand, grow successively from $1 \times 1 \times d_{model}$ to $1 \times T_y \times d_{model}$, where T_y denotes the length of the output sequence. Finally, since the IDs correspond to entries in our vocabulary we can reconstruct a word representation for the generated output sequence.

Transformer: Training Training with the Transformer does not work exactly the same. The training procedure requires both input sequence and the corresponding output sequence (also called labels). Both source and target sequence are tokenized into their token ID representation based on the respective vocabulary. However, before being provided as context to the decoder block the output sequence is shifted by one position to the right by prepending a decoder *SOS* token ID. The calculation of the final encoder hidden state is identical to the calculation at inference. But now, instead of iteratively populating the output sequence, the Transformer model directly generates the final decoder hidden state with $1 \times T_y \times d_{model}$ based on the provided context. To optimize the Transformer parameters the discrepancy between the predicted tokens and the true tokens is calculated based on the specified loss function. The loss function for comparing source to target translation (and other classification tasks) usually is the cross-entropy loss over the entire sequence. The cross-entropy penalizes probability distributions that do not match with the true label, i.e., the less mass of the softmax distribution is given to the true label the higher the penalty. That cross-entropy penalty is calculated for each predicted token and its true label. The sequence

cross-entropy's mean indicates how *far off* or *close* the predicted sequence to the real sequence is. Finally, the error is back-propagated as introduced in Section 3.1.3 and the network's weights are adjusted accordingly. The whole training procedure is characterized as *teacher forcing*. Looking at Figure 17, the only components that are not trainable are the positional encoding vector, the Add & Norm layer, the residual connections and the Softmax layer. All other components in the Transformer are either some kind of linear projection matrix (Multi-Head Attention, Input/Output Embedding, Linear Projection Layer) or a neural network sublayer (Feed Forward Layer) and therefore trainable parameters.

Summary: Transformer In section 3.2.3, we learned about the Transformer architecture and its benefits over RNN based sequence to sequence models. The Transformer greatly enhances parallelization by getting rid of the dependency of network states. Additionally, all computations in the Transformer can further be undertaken on tensor-level, meaning training can be conducted on whole batches of input sequences at once, further improving efficiency. Above all, the multi-head self-attention mechanism via the dot product leads to richer encoding of contextual and semantic information that improve the quality of predictions with the Transformer architecture versus RNN and other approaches. In the next section, we are going to explore how we are utilizing the Transformer architecture for our translation task.

4 Experiment - Impact of Data Availability for the Pivoting Strategy in Low-Resource Scenarios

Having access to the components we previously introduced, we now propose an experiment to specifically gain insights in the dependency between data availability and translation quality for the pivoting strategy in a selected low-resource scenario. To conduct an experiment, we make use of the concepts introduced previously by (Kim et al., 2019) and (Mhaskar & Bhattacharyya, 2021). Additionally, we are going to introduce sampling fractions of the pivoting resource, to investigate the dependency of the amount of *additional* data a model is trained on against its translation quality. There are multiple factors of influence to consider. The number of language pairs between

1. source-pivot
2. pivot-target
3. source-target

by repeatedly training the same architecture (including hyperparameters) on multiple sample sizes, a clear dependency may become visible, helping to identify specific numeric thresholds for specific qualities for the translation task. We specifically test the hypothesis of larger source-pivot and pivot-target corpora resulting in better translation qualities, as the pivot language is the only variable factor when considering a source-target translation using the pivot strategy. The low-resource language pair, on the other hand, is the fixed component. By introducing artificial data deficiency between source-pivot and pivot-target, we may uncover latent importance and requirements that potentially change the selection of the pivoting language based on the availability of data between the pairs. We use two base lines to compare our approach to. Firstly, we train a benchmark model on the WMT19 [36] data. The WMT in 2019 introduced a French-German and German-French news translation task whose test set will be used for evaluating and comparing our models internally and externally. For the WMT19 model, we do not use any pivot strategy and are not restricting the language pair resources to 100,000 sentence pairs. Instead, the WMT19 data consists of about 10 million (9,589,357) sentences between French and German from different corpora [37]. Also, we choose hyperparameters which are suitable for non-low-resource scenarios, i.e., the

[36]The Workshop on Machine Translation (WMT) is the main event for machine translation and machine translation research. The conference is held annually in connection with larger conferences on natural language processing.

[37]commoncrawl, bicleaner07, dev08_14, europarl-v7

default parametrization as proposed by (Vaswani et al., 2017). The resulting model described

Parameter	WMT19 Transformer Model
encoder_type	transformer
decoder_type	transformer
position_encoding	true
enc_layers	6
dec_layers	6
heads	8
hidden_size	512
word_vec_size	512
transformer_ff	2048
dropout	0.1
dropout_steps	[0]
attention_dropout	[0.1]

Table 2: WMT19 Transformer Model Configuration

in Table 2 can be regarded as an strong baseline or upper-bound baseline. It resembles a model that is not constrained by limited language resources. Secondly, we use the plain source-target translation model as a lower-bound baseline, and assume a larger pivot corpus leads to better translation quality. The plain source-target model is trained on 100,000 sentence pairs only, to emulate the low-resource scenario in which no other exploit is utilized to improve the translation quality. The resulting source-target model can be understood as the lower-limit baseline. We adapt configuration parameters that are heuristically suited for low-resource scenarios, as described further down in the Practical Experiment Considerations paragraph. We compare these baselines to our pivot approach, which is are inspired and adapted from (Kim et al., 2019) and (Mhaskar & Bhattacharyya, 2021). A similar approach for the cascading pivot strategy has been conducted for statistical machine translation systems by (Paul et al., 2009) where a comparison is made between the optimal pivot language for a 10k and 80k source-pivot and pivot-target language pair, focusing on language selection and not selection by data availability. Their results show that 75.5% of the choices for an optimal pivot language do not change when increasing the amount from 10k to 80k sentence pairs, hinting at the possibility that rather than being a high resource source of data, instead the pivot strategy succeeds based on other circumstances, such as linguistic similarity between source-pivot or pivot-target language. To be precise, we want to investigate the impact of or point of diminishing returns for changes in data availability when pre-training with a pivot language.

Transfer Learning A large-scale model that can subsequently be adapted to solve multiple different tasks is known as a foundation model (Bishop & Bishop, 2024). Central to our approach is the theory of transfer learning, where knowledge gained from a general task *(pre-training)* enhances a model's adaptability to a specific task, e.g., translation. The theoretical foundation of transfer learning is rooted in the idea that models trained on one task can capture general features and representations that are transferable to related tasks. In the context of machine translation, this means that a model pre-trained on a diverse dataset containing multilingual content can acquire valuable linguistic knowledge applicable to various language pairs. Kim et al. (2019) and Mhaskar & Bhattacharyya (2021) utilize transfer learning by initially training specific components of the final model on data that are intermediate and related to the target task, before introducing the final task and its respective data. We specifically design the experiment to make use of the notion of transfer learning. By first training a transformer model between source and pivot languages, we obtain an encoder-decoder model where the encoder trained to encode the source language. By freezing the weights of the encoder and only training the decoder in the next transformer model between pivot and target language, we guarantee that the encoder does not unlearn encoding the source language while being presented with the task to encode the pivot language. Without freezing the encoder component, the model is exposed to (catastrophical) forgetting, and is likely perform worse the more pivot data it is trained on (Kim et al., 2019). The second stage of the training produces a decoder that decodes the target language based on *pseudo-source* encodings. The pseudo-source encodings are furthermore the result of the BPE based on a shared vocabulary between source and pivot language. By using shared subword units as tokens for the decoder, we allow the encoder-decoder training in the second step (pivot-target) to perceive similar encodings with respect to the source language. Finally, we retrieve the frozen encoder and decoder from the second transformer model and assemble a third transformer model. The task of this third transformer model is the source-target translation, i.e., training to translate between the low-resource language pair. We hypothesize that by utilizing the notion of transfer learning the translation quality improves above the lower-bound baseline. We furthermore hypothesize that an increasing size of pivot resource yields diminishing returns. To test our hypothesis, we build respective BPEs and conduct the training procedure as described above for different fractions of the data:

We use resource limitations (fewer than 800,000 pivot pairings) that are representative for out of simulation scenarios. The resource between source and target is fixed at 100,000 sentence pairs. For the first and second stage, it is important to note that the pivot BPE encodings are calculated on the corresponding level of resource. In other words, even thought we may have

Samples	Description
0	The baseline in which no pivot language is used
50,000	Pivot resources weaker than those of low-resource pair
100,000	Equal pivot resources
200,000	Canonical intermediate step
400,000	Medium resources for pivot pairings
800,000	Large resources for pivot pairings

Table 3: Sample sizes and their corresponding descriptions for pivot language resources

a corpus of 3 million English and 2 million French sentences we only use, e.g., 50,000 source sentences + 50,000 pivot sentences from which the BPE models and vocabulary for stage 1 are derived. Stage 3 always uses a BPE model based on 100,000 sentence pairs between source and target language. We argue that artificially limiting access to resources better simulates a realistic low-resource scenario and enhances the robustness of our findings. Consequently, increasing the pivot resource sample size is not simply controlled by using checkpoints at specific points in training and adding additional data, but by completely training each model based on its respective BPE from scratch. The training procedure itself is deterministic by specifying a random seeds, using the same initialization parameters for the transformer architecture parameters and other training parameters, such as the learning rate and so on. In doing so, we guarantee comparability across experiment settings and reproducibility for potential future work.

Comparison Now, the third, final stage is to compare the results of fine-tuning the source to target translation task between the pivot 50k, 100k, 200k, 400k and 800k. By varying the amount of data on which the first and second stage models are trained, we may observe the relevance of pre-training for the source and target translation task. We evaluate the quality of each model based on the BLEU, TER and chrF score, which are introduced after the upcoming paragraph. The WMT provides the industry standard testsets on which translation tasks can be benchmarked and compared against results of other streams of research and their approaches.

Practical Experiment Considerations The procedural framework, described in the following section, outlines the steps taken to achieve our research objectives. The model architecture for our experiment is not identical to (Mhaskar & Bhattacharyya, 2021) and (Kim et al., 2019). The authors do not work specifically on low-resource scenarios. Araabi & Monz (2020) shows using smaller and more shallow models yield better results in scenarios where data is not abundant. Araabi & Monz hypothesize that larger architectures and

therefore a larger amount of weights generally need more training iterations to converge. Based on Lankford & Afli's findings, using an unigram subword model instead of BPE has no significant impact on the translation results in low-resource scenarios, ergo BPE is used as the subword model for our experiment. Lankford & Afli (2021) furthermore suggest a vocabulary size of 16k over vocabulary sizes of 8k and 32k. The hypotheses between Araabi & Monz, van Biljon et al. (2020) and Lankford & Afli (2021) are consistent with each other and suggest that smaller and shallower transformer architectures, combined with BPE, may yield more promising results in low-resource scenarios. Heuristically motivated, we employ a more shallow 3-layer (*default: 6-layer*) base transformer described in (Vaswani et al., 2017) for all experiments. The batch size is set to 2,048 tokens per batch Araabi & Monz (2020). We use the Adam Optimizer with initial learning rate 0.0001, which is multiplied by 0.7 whenever perplexity on the validation set does not improve for three checkpoints as described by Kim et al. (2019). When it does not improve for 8 checkpoints, we early stop the training. Table 4 shows the comparison between training with the default transformer parameters

Parameter	**Default**	**Low-Resource**
encoder_type	transformer	transformer
decoder_type	transformer	transformer
position_encoding	true	true
enc_layers	6	3
dec_layers	6	3
heads	8	8
hidden_size	512	256
word_vec_size	512	256
transformer_ff	2,048	2,048
dropout	0.1	0.3
attention_dropout	[0.1]	[0.1]
label smoothing	0.1	0.1
batch size	$\sim$ 25,000 tokens	2,048 tokens
vocabulary size	32,000	16,000
subword model	BPE	BPE
optimizer	Adam	Adam
β_1	0.9	N/A
β_2	0.98	0.998
warmup steps	4,000	6,000
decay method	N/A	noam

Table 4: Comparison of Default Transformer Parameters and Adjustments for Low-Resource Scenarios

described in Kim et al. (2019) and Vaswani et al. (2017) versus the adjusted parameters for low-resource scenarios as described above for 100k sentences between French (source)

and German (target). We use the same low-resource parametrization for the different pivot models. The training pipeline is built with the OpenNMT library[38], which lets us explore these different parameters based on configuration files. The OpenNMT library, similar to the well known statistical machine translation system "Moses", provides functionality to train transformer and other artificial neural networks, e.g., "Image to Text", "Video to Text" or "Speech to Text" neural networks. To make use of the OpenNMT library, specific requirements have to be met. As soon as the proper type of data is provided, we can call the "onmt_train" procedure, which in turn starts a neural network training loop based on the pytorch[39] library. The procedure is further guided by a configuration file which is an input to the "onmt_train" script. In the configuration file, we define the paths for the target and source files for training and validation data, as well as specific hyperparameters for training such as the learning rate, type of optimization, number of warmup and training steps or early stopping. With the help of the configuration file, we can further influence the architecture of the transformer. E.g., we can set the number of attention heads to 6 instead of 8 or change the d_{model} size to 256 instead of 512. Consequently, running the experiments can be achieved by specifying the respective configuration files. However, the data pre-processing is only partially covered by OpenNMT. While the OpenNMT provides its own tokenization procedure we opted to use the SentencePiece library by Google to build a BPE. The library implements a industry-standard, fast BPE algorithm that creates both BPE model and vocabulary based on a specific corpus. Google's SentencePiece library, however, is only partially compatible with OpenNMT. The vocabulary file which provides the transformer with input token ids has to be either transformed with a compatibility script or built using the OpenNMT tooling. We opted for relying on the OpenNMT tooling and the "onmt_build_vocab" procedure to produce a OpenNMT compatible vocabulary. It is noteworthy, that the token ids have to be updated between steps 1 to 3. There may be various ways to achieve a similar feet, however, when using the same transformer model file in the form of a checkpoint [40] to continue the training for each consecutive step from, the OpenNMT implementation of the transformer internally saves representation of the vocabulary. When switching between step 1, the training between source and pivot, to step 2, the training between pivot and target, the token ids for the target language are unknown to the transformer or worse, already occupied by subword units from the source and pivot language leading to faulty trainings and results. To prevent

[38]OpenNMT Homepage `https://opennmt.net/`. Remark: On July the 4^{th}2024, it was announced in the OpenNMT community forum that the OpenNMT project is discontinued and instead eole (`eole-nlp.github.io/eole`) is to be used

[39]PyTorch is an optimized tensor library for deep learning using GPUs and CPUs: `https://pytorch.org/`

[40]a save file in machine learning or deep learning is referred to as checkpoint

overloading the token ids with multiple entries, we use the "-update_vocab" option provided by the OpenNMT library.

Preparation of Datasets Datasets for the selected language pairs are acquired via `https://opus.nlpl.eu/` (Opus), ensuring sufficient volume and quality for training, validation, and testing. Opus contains about 1,200 different corpora for different language pairs, from and to German, from and to English, and from and to French, among others. The source (French) to target (German) language pairing has 360,890,261 sentence pairs [41]. The source-to-pivot (English) language pairing has 1,088,160,191 sentence pairs. The pivot to target language pair has 898,280,702 sentence pairs. Those numbers are far from "low-resource". Neural Language Translation is a data-hungry undertaking. A low-resource scenarios is typically classified as such if the amount of parallel data in that particular translation scenario comprises fewer than 100,000 sentence pairs. Therefore, we constrain the source-target corpus to 100,000 of randomly sampled pairs to artificially generate a low-resource scenario. We are going to further compare results for 0 % (i.e., source-target baseline model), 50 %, 100 %, 200 %, 400 % and 800 % of 100,000 for the size of the pivot corpus, allowing us to inspect changes in translation quality with respect to the availability of pivot language pairings as described in the previous section. We use the publicly available Europarl corpus [42] provided by Opus to obtain high quality French, English and German parallel sentences. The data is cleaned to remove missings, duplicates, and rows that were copied (i.e., an exact match or copy in source and target corpus), and normalized for special characters and HTML sequences. Lastly, we shuffle the sentences as oftentimes datasets come with sentence pairs sorted by length. Afterwards, we split the data into training, validation (i.e., development) and testset, and reserve 6,000 sentence pairs for validation and finally hold out 12,000 sentences for testing the translation models. The validation set is used during the training of the different models. While the training set is being processed, every 10,000 steps the OpenNMT procedure reports on accuracy, cross-entropy and perplexity based on the validation set, where the current state of the training is tasked to translate the source side of the validation set. The resulting translation attempt is compared against the target side of the validation set.

```
    def accuracy(self):
        """compute accuracy"""
```

[41] Sentence Pair Data inquired on June 26, 2024

[42] The Europarl parallel corpus is extracted from the proceedings of the European Parliament. It includes versions in 21 European languages: Romanic (French, Italian, Spanish, Portuguese, Romanian), Germanic (English, Dutch, German, Danish, Swedish), Slavik (Bulgarian, Czech, Polish, Slovak, Slovene), Finni-Ugric (Finnish, Hungarian, Estonian), Baltic (Latvian, Lithuanian), and Greek. (European Commission (last), 2011)

```
        return 100 * (self.n_correct / self.n_words)

    def xent(self):
        """compute cross entropy"""
        return self.loss / self.n_words

    def ppl(self):
        """compute perplexity"""
        return math.exp(min(self.loss / self.n_words, 100))
```

Listing 1: Validation Metric Calculation, Source: github/OpenNMT-py

In Listing 1, we illustrate the calculation steps for the accuracy, entropy (xent), and perplexity (ppl) metric. The *self* Object is a Statistics class object that gets initialized in the *trainer.py*-onmt procedure with the respective information *n_correct*, i.e., the number of correctly translated words, the current *loss* of the model that is being validated, and the total number of words in the validation set. These calculations are conducted on batch-level, meaning, since our validation batch size is $4,096$ tokens and we use $6,000$ sentences for validation consisting of about 26 words on average per sentence, and with about 0.72 tokens per word, we accumulate 55 batches over which the statistics are gathered. On that level, we are able to get robust insights to performance metrics during the model training. After the training is concludes, to finally evaluate the performance of the output model, in addition to our holdout Europarl test set, we consult the BLEU Score on the WMT19 fr-de news translation task test data, which was not used in training and furthermore has a slightly different overall vocabulary than the Europarl (politics) vocabulary. We expect the models to perform slightly worse on the out-of-training-domain vocabulary. To counteract an overfitting behaviour, dropout rates on attention-head and feedforward level are set to 0.1 and 0.3, which aligns with findings from (Lankford & Afli, 2021).

The BLEU-Score Using the BLEU-Score as a benchmark is a domain and industry standard. In July 2002 IBM's research group led by Kishore Papineni introduced BLEU (acronym: bilingual evaluation understudy) as a measure to algorithmically evaluate translations and their quality (Papineni et al., 2001). In their study, the proposed BLEU procedure correlates strongly (96% to 98%) with the judgment of both mono- and bi-lingual human judges (10 judges per group) when ranking proposed reference translations to a given source sentence. Compared to human evaluation, the BLEU is fast and cheap. Noteworthy, in the paper introducing BLEU, Papineni et al. assume their metric to be used for ranking different translation outcomes to multiple reference translations. Papineni et al. discuss the use of a single reference translation to be correct only under some circumstances:

> *... we may use a big test corpus with a single reference translation, provided that the translations are not all from the same translator.* (Papineni et al., 2001)

Nevertheless, the BLEU metric has been adopted as the dominant metric for Machine Translation research (Post, 2018). Different scientific groups have since criticized the widespread use of BLEU and many alternatives have been proposed. However, the scoring approach is used as a benchmark in virtually every research paper as it is a widely known, simple to implement, fast, and common metric for comparison within domain-specific tasks. Designed to be used for several reference translations, in practice it is used with only a single reference translation since often large corpora contain bijective sentence pairs in source and target language. The success of the BLEU score is polarizing. One of the most cited works questioning, e.g., the correlation of BLEU with human judgments is the work by (Osborne & Koehn, 2006) from 2006: "Re-evaluating the Role of BLEU in Machine Translation Research" (Osborne & Koehn, 2006).

> *... BLEU is not sufficient to reflect a genuine improvement in translation quality, and in other circumstances that it is not necessary to improve Bleu in order to achieve a noticeable improvement in translation quality.*

The authors claim that although BLEU has significant advantages, there is no guarantee that an increase in BLEU score is an indicator of improved translation quality (Osborne & Koehn, 2006).

> *The main principle behind BLEU is the measurement of the overlap in unigrams (single words) and higher order n-grams of words, between a translation being evaluated and a set of one or more reference translations. The main component of BLEU is n-gram precision: the proportion of the matched n-grams out of the total number of n-grams in the evaluated translation. Precision is calculated separately for each n-gram order, and the precisions are combined via a geometric averaging. BLEU does not take recall into account directly. Recall – the proportion of the matched n-grams out of the total number of n-grams in the reference translation, is extremely important for assessing the quality of MT output, as it reflects to what degree the translation covers the entire content of the translated sentence.* (Banerjee & Lavie, 2005)

Banerjee & Lavie (2005) describe that BLEU does not use recall because the notion of recall is unclear when matching against multiple references simultaneously. But in reality, BLEU is rarely used with multiple references, so not considering recall is a weakness. Furthermore,

BLEU is dependent on the tokenization technique, and scores achieved with different ones are incomparable. The tokenization technique dependency is a common flaw among evaluation metrics for natural language processing tasks. In order to improve reproducibility and comparability, the SacreBLEU package was designed in 2018 by Matt Post, an Amazon Researcher (Post, 2018) who himself attributes the idea to Rico Sennrich, Professor at the University of Zurich working on natural language processing. SacreBLEU tries to solve the problem of inconsistency in the reporting of BLEU scores (Post, 2018).

> *Although people refer to "the" BLEU score, BLEU is in fact a parameterized metric whose values can vary wildly with changes to these parameters. These parameters are often not reported or are hard to find, and consequently, BLEU scores between papers cannot be directly compared.*

SacreBLEU therefore is no derivative or next-generation metric, that replaces BLEU but an extension to make working with the metric more consistent. On top of that, the SacreBLEU library provides access to the Translation Error[43] Rate (TER) metric and character n-gram F-score (chrF).

The TER Metric The TER is defined as the minimum number of edits needed to change a hypothesis (i.e., the predicted sequence) so that it exactly matches one of the references, normalized by the average length of the references (Snover et al., 2006).

$$\text{TER} = \frac{\text{\# of edits}}{\text{average \# of reference words}} \tag{33}$$

TER corresponds to post-editing effort.

> *Possible edits include the insertion, deletion, and substitution of single words as well as shifts of word sequences. A shift moves a contiguous sequence of words within the hypothesis to another location within the hypothesis. All edits, including shifts of any number of words, by any distance, have equal cost. In addition, punctuation tokens are treated as normal words and mis-capitalization is counted as an edit.* Snover et al. (2006)

The metric is straight forward and interpretable. In contrast to BLEU, a higher TER indicates a worse translation result. A perfect fit between hypothesis and reference results in 0 edits,

[43] originally named Translation Edit Rate by Snover et al. (2006) but commonly referred to as Translation Error Rate

yielding a translation error rate of 0.

The chrF Metric The chrF proposed by Popović (2015), on the other hand, computes

$$\text{chrF} \cdot \beta = (1 + \beta^2) \frac{\text{chrP} \cdot \text{chrR}}{\beta^2 \cdot \text{chrP} \cdot \text{chrR}} \tag{34}$$

where chrP and chrR stand for character n-gram precision and recall arithmetically averaged over all n-grams.

- chrP: percentage of n-grams in the hypothesis
- chrR: which have a counterpart in the reference
- β: Parameter to scale the importance between recall and precision.

In contrast to BLEU, the computation is evaluated on character and not on word level, making it more sensitive to smaller differences, such as word forms, spelling, and morphological variations. Furthermore, in contrast to BLEU, chrF computes both precision and recall, addressing a common flaw of the BLEU metric. However, as they operate on different levels, they measure different aspects of translation quality. BLEU is more focused on word-level matching and precision, with an emphasis on fluency and overall sentence structure, whereas chrF focuses on character-level matching, balancing precision and recall, is more sensitive to finer details like spelling and morphology. The two additional metrics, TER and chrF, are common variants besides the BLEU metric which enable a broader comparability.

5 Results

In the following chapter, we report on the results of our experiment. First, we conduct a visual analysis of the training procedure and its results. A visual inspection provides insight to the model's learning dynamics, allowing us to identify trends, detect anomalies, and assess the overall performance and convergence of the training process. After visually confirming the plausibility of our experiment, we provide the translation quality results across the different scenarios, followed by an excerpt of translation examples. We conclude in a comparison of the translation qualities for the different scenarios.

5.1 Visual Analysis

To analyze the training procedure, we use Tensorboard[44]. Tensorboard is a visualization tool for monitoring and debugging machine learning processes, providing interactive visualizations of metrics such as, e.g., loss and accuracy during training and evaluation. To access this information, Tensorboard provides a graphical user interface. Tensorboard is supported by the OpenNMT lirbrary, natively. In the following paragraph, we describe findings and observations during the training procedure of the models. In total, we trained 15 models[45], along with 2 baseline models. While it is unnecessary to visually inspect all 17 training procedures, we focus on patterns and provide examples. The extensive log-files can be found in the evaluation directory for further inspection.

Visual Analysis - Observations In Figure 22 we observe a variety of information. There are distinct 3 graphs for the 3 stages in our experiment setup. The first (left to right) sub-graph belongs to the language pair source-pivot (fr-en), the second to pivot-target (en-de), and the third to source-target (fr-de). In the Table below, there are several metrics reported. The initial and final training accuracy of the respective model (Min or Start Value and Max or End Value), the absolute and relative change in accuracy (Δ Value, Δ %), and finally, the Start and End Step. Alongside the graphs, these metrics provide a meaningful insight into the training procedure. The first sub-graph begins at step 10,000. In the training configuration, we set the validation to run every 10,000 steps, so we observe the expected behaviour. We also observe the training curves to generally increase during the training procedure, indicating our loss function and architecture generally optimizes the model for the training data. Interestingly, we observe gaps of 20,000 steps between stages 1 and 2

[44]Tensorboard: `https://www.tensorflow.org/tensorboard`
[45]15 models from 3 stages for 5 pivot resource levels

and stages 2 and 3. We expect these jumps to be 10,000 steps wide. A training procedure concludes in reporting its final parameters and by forming a step-specific checkpoint, that reflects the training process up to that specific step. We observe a reporting at step 90,000, and in our model files we find the corresponding 'model.fr-en_step_90000.pt', also. File 'model.fr-en_step_90000.pt' then serves as the starting point for training the en-de model. During process, the pytorch optimizer (which is saved in the checkpoint alongside, e.g., the model weights or the model architecture) preserves both learning rate and progression. We expect the training procedure to continue from 90,000 steps, hence expect the first validation procedure to occur after 10,000 steps at 100,000 steps. During training, however, an issue occurred if the *early stopping* mechanism triggered. The training configuration involved an early stopping mechanism that concluded the training as soon as for 8 consecutive validations there is no meaningful improvement in perplexity. We expected the procedure to save the models' checkpoint at the early stopping. However, due to an pytorch memory allocation error [46], that last checkpoint including its logging seems to be lost. That memory error occurred consistently on early stopping in the other model training procedures but is negligible since due to the nature of the mechanism there must have been either negative, or very small to no changes in model quality for the last 8 steps anyway. We now shift our focus to the

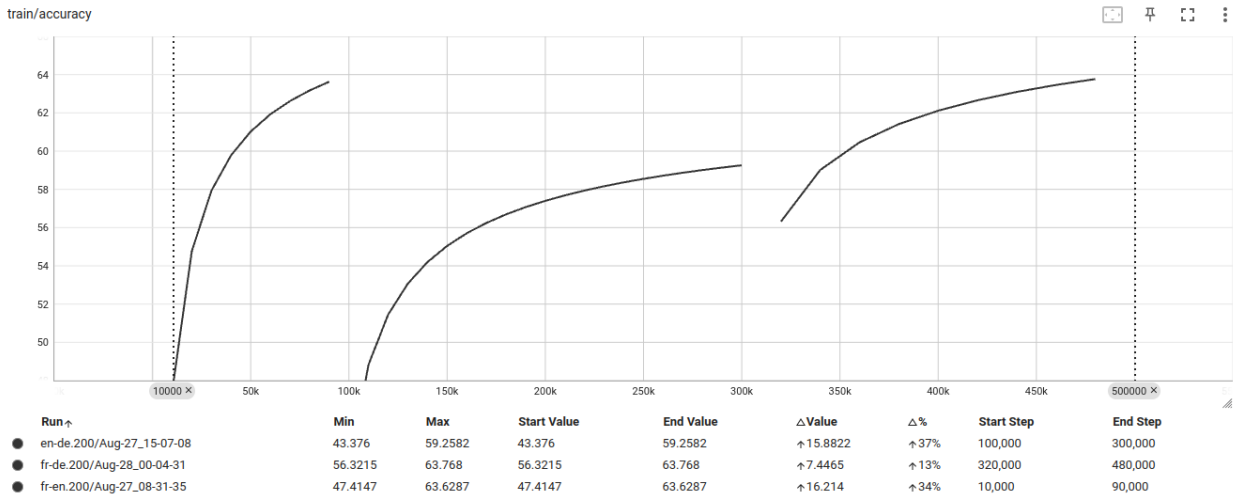


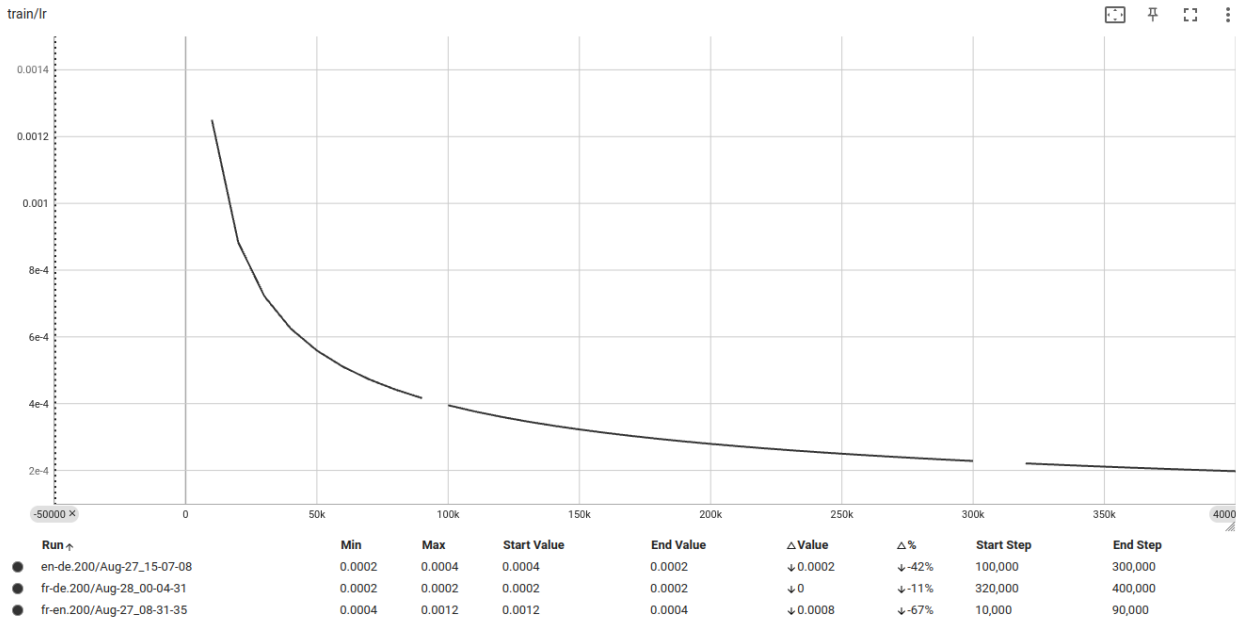
Figure 22: Training Accuracy Graph, Pivot Resource Level: 200,000 Sentences

absolute training accuracy. To compare the metric between the three sub-graphs has little

[46] An error in cpython/Lib/multiprocessing/popen_fork.py, resulting in *14 leaked semaphore objects to clean up at manual shutdown*, hinting at a deadlock during memory allocation in a low-level library during runtime which we were not able to address. Further inspection revealed that the process tries to allocate hundreds of Gigabytes of RAM for no apparent reason. The error only occurs when the checkpoint saving procedure is initiated by early stopping.

meaning. sub-graph 1 and 2 are intermediate steps *enabling* sub-graph 3. Worthy of note in sub-graph 3 (fr-de) is the absolute training accuracy, which begins at 56.3 compared to 47.41 for fr-en and 43.3 for en-de. The high starting accuracy value could already hint to the overall success of the pivot approach, putting the pivot model at advantage over the non-pivot variant.

In Figure 23, we observe the declining learning rate during the training. The decline is specified in the training config file and provided to the pytorch optimizer, which handles the learning rate adjustments accordingly. In the training config, we used the "adam" optimizer, starting at a learning rate of 2 with 6000 warmup steps, a $adam\_\beta_2$ of 0.998 with the 'noam' decay method (Lukasz Kaiser, 2017). We observe, that the state of the optimizer persists

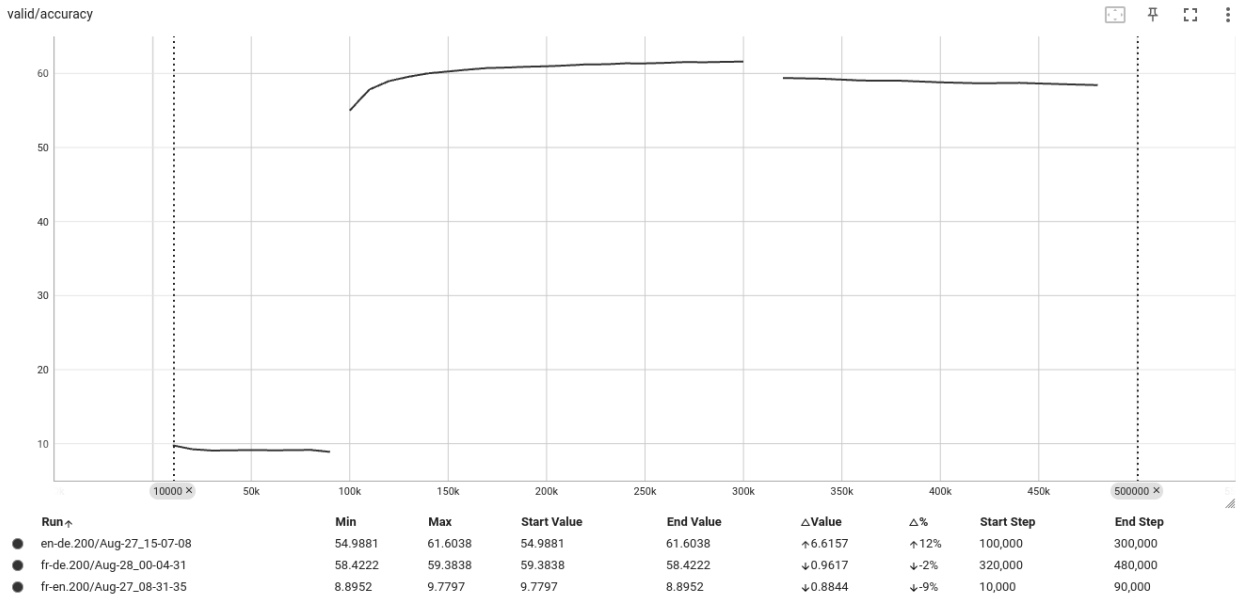


Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
en-de.200/Aug-27_15-07-08	0.0002	0.0004	0.0004	0.0002	↓0.0002	↓-42%	100,000	300,000
fr-de.200/Aug-28_00-04-31	0.0002	0.0002	0.0002	0.0002	↓0	↓-11%	320,000	400,000
fr-en.200/Aug-27_08-31-35	0.0004	0.0012	0.0012	0.0004	↓0.0008	↓-67%	10,000	90,000

Figure 23: Learning Rate Graph, Pivot Resource Level: 200,000 Sentences

across training stages 1, 2 and 3. As expected, the jumps between the stages occur. At these jumps, learning rates are not reset. We included the decaying mechanism described in Chapter 4, which should have let the learning rate decay on validation perplexity not improving over three steps, but the behaviour can not be observed from the logging nor visually. Most likely, our implementation of the decaying mechanism proposed by Kim et al. (2019) did not work as intended and was overridden by the default 'adam' optimizer's behaviour. The learning rate is a parameter to control the convergence and convergence speed of the training. Since our training still shows steady improvement and seems to converge, we conclude the missing mechanism to be negligible.

In Figure 24, we can see the accuracy metric which was previously shown in Figure 22 but now for the validation data. As the validation data is not part of the neural network optimization, we expect the metric to be lower than the training accuracy. Interestingly, the



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
en-de.200/Aug-27_15-07-08	54.9881	61.6038	54.9881	61.6038	↑6.6157	↑12%	100,000	300,000
fr-de.200/Aug-28_00-04-31	58.4222	59.3838	59.3838	58.4222	↓0.9617	↓-2%	320,000	480,000
fr-en.200/Aug-27_08-31-35	8.8952	9.7797	9.7797	8.8952	↓0.8844	↓-9%	10,000	90,000

Figure 24: Validation Accuracy Graph, Pivot Resource Level: 200,000 Sentences

pivot-target validation accuracy is higher than the pivot-target training accuracy. Usually, a higher validation than training accuracy indicates data leakage, i.e., situations where to model has access to data it should have no access to, or in simpler terms: the validation data was part of the optimization procedure. However, we thoroughly checked the file paths in the training configuration, the data splitting and selection process, and were not able to find any issues. The behaviour is anomalous but did not occur in other resource levels besides 200,000 pivot sentence pairs. Besides the anomaly, we can observe that the validation accuracy overall increases between stage 1 and stage 3 but decreases slightly between stage 2 and stage 3, even decreasing between the start and end of stage 3. At this point, the learning rate is decayed to 0.0002 (rounded), allowing only for small adjustments of the overall model. The increasing training accuracy during the third phase, on the other hand, indicates adequacy of the learning rate and overall model. Also, it is important to consider that the final stage 3 only sees 100,000 sentence pairs between source and target language to simulate the low resource scenario. If we, on the other hand, take a look at Figure 25, we instead see a steady improvement between stages. However, in this scenario, the additional resources amount only to half of the fixed number of 100,000 source-target sentence pairs. The resulting improvement in performance may be attributed to the larger amount of avaliable resources in the final stage. On a neutral resource level as seen in figure 26, where pivot data and source-target

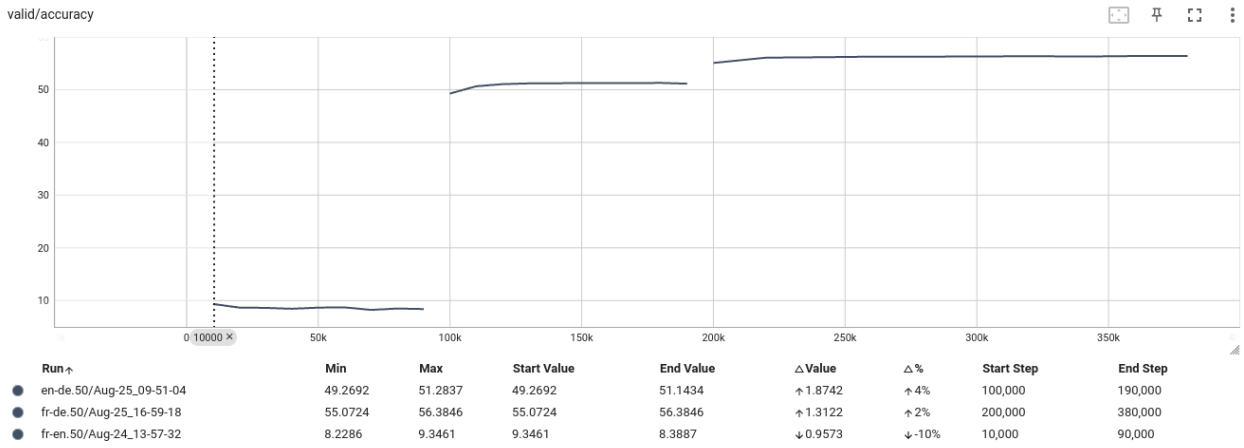


Figure 25: Validation Accuracy Graph, Pivot Resource Level: 50,000 Sentences

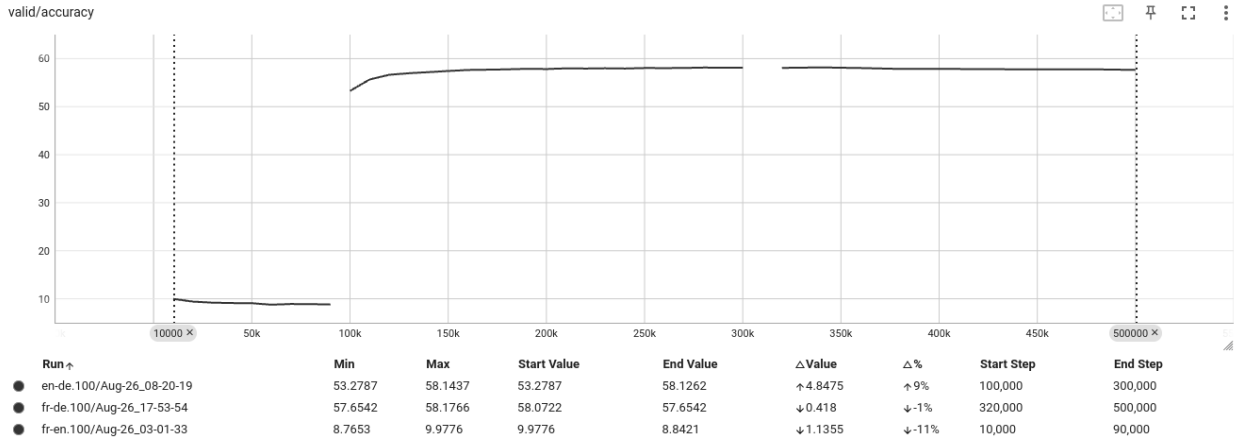


Figure 26: Validation Accuracy Graph, Pivot Resource Level: 100,000 Sentences

data are available equally (100,000 sentence pairs), we even observe a decrease of validation accuracy of -0.48 with overall little to no change to the model's performance during the source-pivot training phase. However, the observed behaviour, consistent improvement in training accuracy, is similar to the 200,000 sentence scenario. Here we have to consider that it is possible and likely that due to the limited amount of source target sentence pairs $(100,000)$, the intermediate training between source to pivot or pivot to source is resulting in better performing intermediate models on their validation data. That better performance is to be expected and the discrepancy even stronger the larger the gap between the resources available for training. The discrepancy between validation accuracy in stage 1 and 2 and validation accuracy in stage 3 is displayed in 27.

To argue for or against the success of the pivot strategy, we have to inspect the perfor-

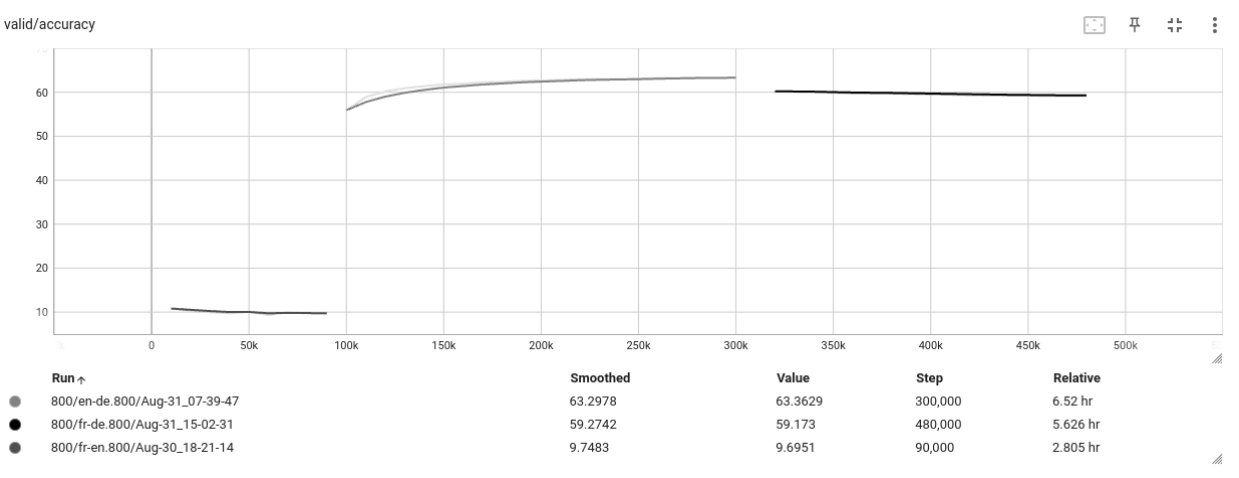


Figure 27: Validation Accuracy Graph, Pivot Resource Level: 800,000 Sentences

mance of the plain source to target transformer model in which no pivot strategy was used. With respect to the validation accuracy during training, we we can see in Figure 28, the pivot strategy outperforms the baseline source to target transformer model. The validation accuracy of the baseline strategy reaches 5.68 whereas the accuracy of the different pivot strategies are about 10× that high. Not only is the validation accuracy outperformed by the

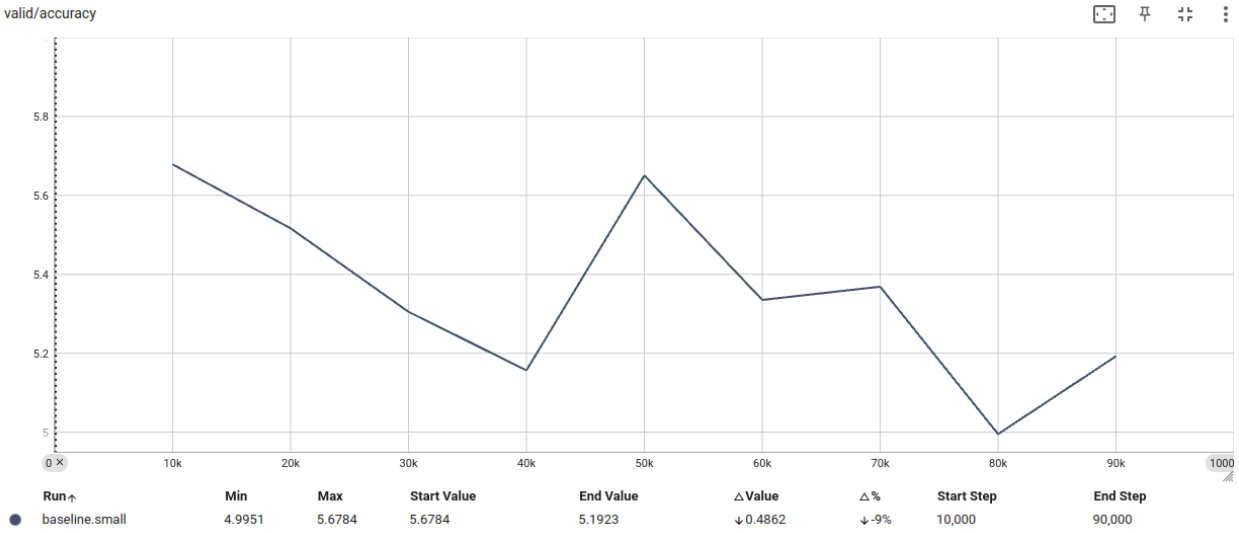


Figure 28: Validation Accuracy Graph, lower-bound Baseline Model

pivot strategies validation accuracy but the baseline validation accuracy does not stabilize or increase, even after 90,000 training steps between 100,000 sentence pairs from source and target language. The behaviour seems odd, however, the training accuracy shown in Figure 29 reveals that the baseline model is learning the transfer between source and target language

sufficiently, i.e., the model is slowly but steadily optimizing. Noteworthy, the early stopping

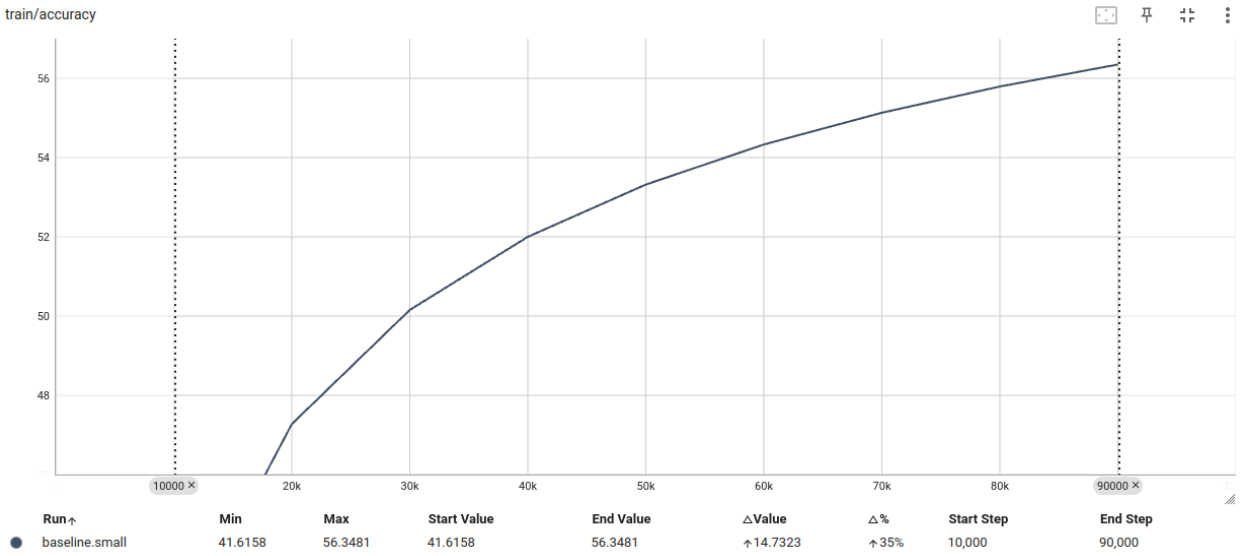


Figure 29: Training Accuracy Graph, lower-bound Baseline Model

mechanism stops the training for the baseline model after 8 validation iterations (90,000 training steps), because the perplexity did increase instead of decreasing.

Finally, we report the training progress for the strong, upper-bound baseline model, which was trained on source to target sentences provided in the WMT19 challenge. The model's parameter are set to the default parameters described in (Vaswani et al., 2017). Since the WMT19 data situation is no low-resource scenario, the parameters do not have to be adapted for such. The curve shown in Figure 30 reveals, that the transformer architecture picks up on the optimization goal and is able to increase the accuracy on its validation data up to 61.1 after 170,000 training iterations. Interestingly, the validation accuracy is about as high as for the pivot strategies, which were trained on comparably small amounts of resource. The visual analysis serves as an indicator but more reliable are the translation quality metrics.

5.2 Translation Quality

From our experiments, we obtain 7 trained models: lower-bound baseline (zero pivot resources, i.e., naive source-target model), 50,000, 100,000, 200,000, 400,000 and 800,000 pivot sentence pair models, and finally the strong baseline model which is trained on the WMT19 data. We report BLEU, TER and chrF for the standardized WMT19 test-set aswell as for the hold out testset based on the Europarl data. The results are shown in Table 5.

The leftmost column "Pivot Resource" describes the amount of resources used, where 0

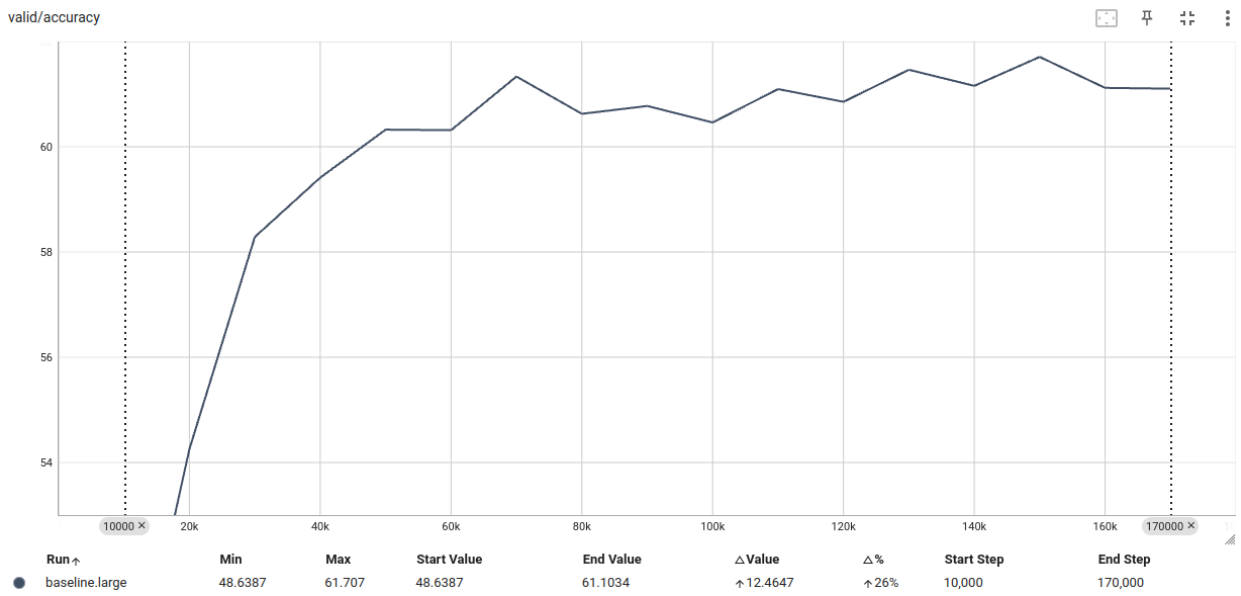


Figure 30: Validation Accuracy Graph, upper-bound Baseline Model

Pivot Resource	**Europarl**			**WMT19**		
	BLEU	**chrF**	**TER**	**BLEU**	**chrF**	**TER**
0	19.48	50.68	70.87	12.82	44.97	81.88
50k	20.97	52.25	69.00	15.09	48.04	77.84
100k	21.76	52.77	68.06	15.99	48.64	75.81
200k	22.09	53.22	67.74	16.38	49.20	75.94
400k	**22.48**	**53.61**	67.55	**17.31**	**50.29**	**74.68**
800k	22.29	53.39	**67.54**	16.95	49.84	74.85
WMT19 Baseline	27.85	57.92	62.17	24.19	56.46	66.10

Table 5: Translation Quality: BLEU, ChrF, and TER metrics for Europarl and WMT19 test sets

corresponds to the naive strategy in which pivotting was not used, 50k corresponds to 50,000 sentence pairs between source to pivot and pivot to target language, and so on. The last row, WMT19 Baseline, corresponds to the results of the model trained on the WMT19 data. The columns "Europarl" and "WMT19" encapsulate three subcolumns for the translation quality metrics BLEU, chrF and TER. The higher the BLEU and chrF score, the better. On the contrary, the lower the TER (Translation Error Rate), the better. The best performing model in each metric is indicated by a bold markup for each dataset. The WMT19 strong baseline, however, is out of competition which is indicated by the two horizontal lines. The WMT19 strong baseline model outperforms every other model in each metric for each test set. It is no strict rule nor proven that a model trained on less resources cannot surpass a model trained on more resources. Nevertheless, the WMT19 strong baseline model is a heuristical *soft upper*

bound and corresponds to "what could potentially (at least) be achieved". We observe various other things from the translation quality table. First and foremost, the naive baseline model, row "0", scores worst in each metric in both datasets. However, compared to the unstable, small validation accuracy and seemingly faulty training procedure, as indicated by the visual analysis, the translation quality metrics of the naive baseline are within reasonable range of the other models which exploited additional data. Between the worse, row "0", and best BLEU score, row "400k", there is a difference of +3 BLEU or a 15.4% improvement in BLEU over naive baseline. The chrF differs 2.93 Points (+5.8% improvement over baseline), and the TER shrinks 3.33 points from 70.87 to 67.54, which corresponds to a 4.7% improvement over baseline. Vice versa, the pivot strategy outperforms the naive strategy in all pivot resource scenarios. Noticeable, the model based on 400,000 pivot sentences in row "400k" has the highest score in 5 out of 6 comparisons, and is beat by the 800k pivot sentence pair model on the Europarl test data in Translation Error Rate by −0.01 points, with values ranging from 70.87 ("0") to 62.17 (WMT19 Baseline). In other words, the 800k model outperforms the 400k only by a margin while being trained on two times the resources. In general, there is a trend noticeable where less resources indicate a worse performance in translation quality and vice versa. Figure 31 illustrates the potential relation between the amount of pivot resources and translation quality. The trend is apparent in all three translation quality metrics, BLEU, chrF and TER. The trend saturates with higher resources. The graphs in Figure 31 and Table 5 furthermore illustrate the discrepancy between in-domain and out-of-domain neural machine translation training and its quality. The translation quality on the Europarl test set is on average +5.76 BLEU points, +4.16 chrF points, and −8.37 TER points better than on the WMT19 test set at the same level of resource.

5.3 Discussion

Based on our findings, we are going to discuss our research hypothesis that larger source-pivot and pivot-target corpora result in better translation qualities. In other words: Does a relation between translation quality and the amount of pivot resources exist. First and foremost, our findings suggest that using the pivot strategy improves the translation quality over not using a pivot strategy in low resource scenarios. Neural machine translation is a data hungry endeavor, and in our experiments the baseline model, for which no pivot strategy is used, is outperformed by the pivot approach that exploits additional language resources to better satisfy the models *need for data*. As expected, the assumption that our pivot strategy improves the translation quality seems to hold. The visual analysis of the training progress for the naive baseline model shows that it is debatable whether a wrong configuration of the

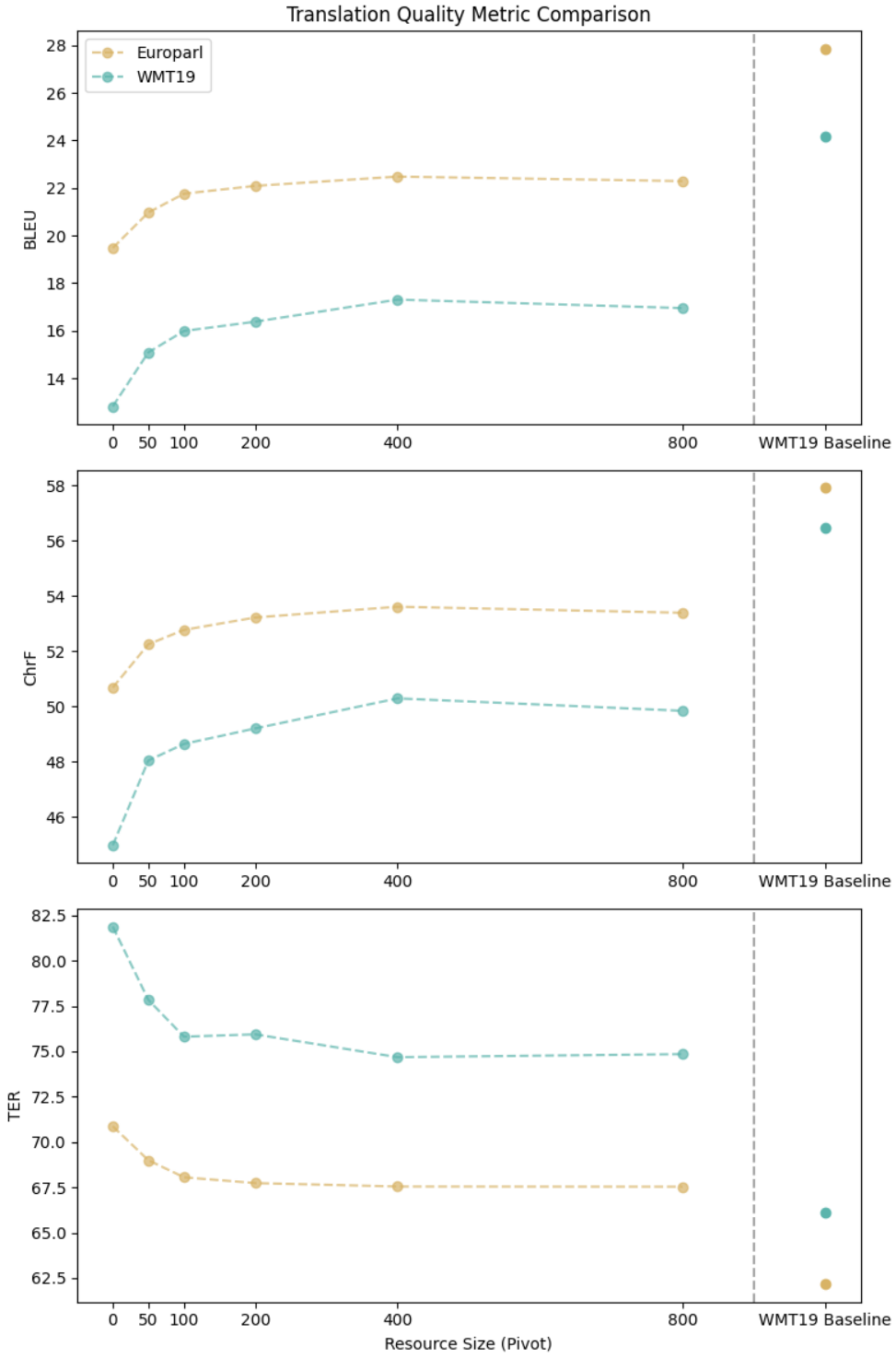


Figure 31: Relation between Resource Availability and Translation Quality

training and network led to the early stopping because of the unstable validation accuracy. We base our configuration parameters on assumptions by Mhaskar & Bhattacharyya (2021),

Kim et al. (2019), Araabi & Monz (2020), van Biljon et al. (2020), Lankford & Afli (2021), and Vaswani et al. (2017) as described in Section Practical Considerations in Chapter 4 but we cannot rule out implementation errors completely, such as the potential (but negligible) failure in adapting the perplexity based learning rate decaying mechanism. From our visual analysis we deduct that in general the training procedure for the naive baseline model picks up on the optimization goal and shows steady improvement in accuracy on the training data. On the other hand, the validation accuracy decreases. There are no mention worthy deviations from the training pipeline used to train the models which include the pivot strategy. The pivot strategy involves additional, more complex steps. The naive baseline model trains between source and target model with 100,000 sentence pairs. In our pivot strategy, a model is trained in 3 stages, firstly between source and pivot, secondly with a frozen encoder between pivot and target, and finally between source and target. The validation accuracy varies between the 3 stages but ends up being $10\times$ higher than the validation accuracy of the baseline model for every scenario. Realistically, the zero pivot resource baseline model should not fail to that extend during training, hence we conclude that our comparison is apparent in nature but questionable. The deduction that our pivot approach improves the translation quality over the lower-bound baseline may be exaggerated by comparing them to a potentially mis-configured lower-bound baseline. On the other hand and to our surprise, the translation quality metrics suggest that even though the validation accuracy for the naive baseline model is just $5-6\%$ compared to $50-60\%$ for the pivot strategy validation accuracies, the discrepancy between translation capabilities is not as high ($+15.4\%$ BLEU, $+5.8\%$ chrF, and -4.7% TER over naive baseline versus $\sim 1,000\%$ increased validation accuracy over naive baseline). However, we also hypothesised the strong baseline model which was trained on the WMT19 data to be an strong, upper-bound baseline. The WMT19 data consists out of 10 million sentence pairs which is $100\times$ the amount of our source to target resources (100,000 sentence pairs). In other words, in our pivot experiment we use 1% of source-target resource sentence pairs while exploiting the effect of 0%, 0.5%, 1%, 2%, 4% and 8% additional pivot resources in comparison to the strong, upper bound model. Our experiment suggests, that while there is a large discrepancy between the availability of data the discrepancy between translation quality is not that big. E.g., the overall best performing model with 400,000 pivot sentence pairs achieves with 1% source-target $+4\%$ pivot of data 80.72% of the potential *strong, upper bound* baseline of 27.85 BLEU points on the Europarl test set and 70.01% of the potential 24.19 BLEU points on the WMT19 test set. In general, the translation quality on the Europarl test set is on average $+5.76$ BLEU points, $+4.16$ chrF points, and -8.37 TER points better than on the WMT19 test set at the same level of resource. The discrepancy is expected, since all models besides the WMT19 baseline model are trained exclusively on the Europarl

dataset, which contains specific vocabulary and topics, primarily in the domain of politics. The WMT19 data, however, consists of multiple data sources as described in Chapter 4. One of the four sources for the WMT19 data is a Europarl-v7 corpus consisting of 1,726,419 sentence pairs, 18.00% of the total 9,589,357 WMT19 fr-de sentence pairs. Arguably, a portion of the translation quality metrics is overestimated because there may be similarities in the vocabularies between training and test data even though a seemingly unrelated and "industry standard" benchmark test set was used. In general, the WMT19 translation quality results are the more robust results as they generalize to the other 82.00% of the WMT19 data, which are not necessarily topics revolving around politics but news and other crawled online resources. Vice versa, the translation quality metrics for the Europarl test set are overestimated and overfit to the specific domain of european politics. Finally, we address the question whether larger source-pivot and pivot-target corpora result in better translation qualities. So far we have concluded that a pivot strategy generally leads to better translation quality. The trend shown in Table 5 and Figure 31 point towards a relation between pivot resource availability and translation quality. The trend, however, seems to saturate at around 400,000 sentence pairs, where no noteworthy improvement takes place when again doubling the amount of pivot resources. The curves of improvement for the different quality metrics as illustrated in 31 are evident of this observation. We emphasize, however, that the breaking point of 400,000 sentences is vague and also strictly connected to the low-resource scenario setting of 100,000 source to target sentences. Other experiments with different amounts of available resource may find other points of diminishing returns but our findings suggest that using a pivot strategy in low resource scenarios with 3to5$\times$ the amount of source-pivot and pivot-target data yields the best results and trying to exploit further additional data only yields little to no improvements.

Comparison to Related Research We describe (Kim et al., 2019) as influential to our work. In their paper, Kim et al. (2019) report the following metrics:

In Figure 32 we find a table that reports on the BLEU and TER for specific approaches. Their parallel fr-de corpus consists of 35,000,000 sentence pairs, their pivot en-de of 9,100,000 sentence pairs, and their source to target fr-de corpus of 270,000 sentence pairs. In their experiment, they are not specifically examining low-resource scenarios, however, when comparing our results to theirs we find similarities. The first row, "Direct source $\rightarrow$ target", corresponds to our naive baseline model in which no pivoting strategy is used. The fourth row, "Plain Transfer", corresponds to the pivot strategy utilized in our approach. Kim et al. report their metrics on two testsets, "newstest2012" and "newstest2013", while using the "News Commentary v14" data as training material for their models. As discussed, using a specific

	French→German			
	newstest2012		newstest2013	
	BLEU [%]	TER [%]	BLEU [%]	TER [%]
Direct source→target	14.8	75.1	16.0	75.1
Multilingual many-to-many	18.7	71.9	19.5	72.6
Multilingual many-to-one	18.3	71.7	19.2	71.5
Plain transfer	17.5	72.3	18.7	71.8
+ Pivot adapter	18.0	71.9	19.1	71.1
+ Cross-lingual encoder	17.4	72.1	18.9	71.8
+ Pivot adapter	17.8	72.3	19.1	71.5
Step-wise pre-training	18.6	70.7	19.9	70.4
+ Cross-lingual encoder	**19.5**	**69.8**	**20.7**	**69.4**

Figure 32: Models fine-tuned with source-target parallel data. Source: (Kim et al., 2019, Table 2),

domain (e.g., topics, vocabulary) for training and testing may lead to an overestimation of translation quality metrics. In our results, that kind of overestimation is potentially embedded within the quality assertion for the Europarl testset. On the other hand, we report on a second, industry benchmark test set, which is in parts out-of-domain. Our results are therefor more robust compared to (Kim et al., 2019) or at least address the issue of in-domain and out-of-domain training and test scenarios. This robustness, however, typically comes at the cost of better translation quality metrics. Hence, averaging our results between Europarl and WMT19 compared to Kim et al.'s results does not seem far fetched. With a BLEU score of 19.48 and 12.82 for our plain naive model compared to their BLEU scores of 14.8 and 16.0 for their direct source to target model, we on average achieve a competitive result of 16.15 BLEU. Not using the exact same training configuration and artificial neural network setting may lead to vastly different results. Our configuration specifically addresses concerns with respect to low resource scenarios. These concerns may given our models an overall advantage above Kim et al. when trained and tested on in-domain data. Comparing the additional gain through the pivoting strategy within the same frame of reference (i.e., testset), we find a 18% and 16.9% increase in BLEU score over baseline from their results, which is close to our 15.4% BLEU score improvement of our 400k pivot model above baseline for the Europarl testset. The improvement of 35.02% of BLEU score above naive baseline from 12.82 to 17.31 BLEU points for our 400k pivot model is somewhat of an outlier that can be explained by the overall bad performance and missing robustness of our naive baseline line model on the WMT19 test data. Also, their baseline TER of 75.1% for newstest2012 and newstest2013

matches our average TER of 76.38% (Europarl and WMT19) for the naive baseline scenario. Their improvements due to their plain transfer pivot strategy result in a relative decrease in TER of 3.87% for newstest2012 and 4.6% for newstest2013 which are close to our 4.91% on Europarl and 9.64% WMT19, with the WMT19 improvement again seeming strong because of the initial weak baseline. Overall, the results are within reasonable range of reference even though Kim et al. (2019) did not specifically construct a low-resource scenario for their experiments.

6 Conclusion

Revisiting the Motivation In our motivation for the thesis in Chapter 1, we describe neural machine translation as the starting point for large language technology and set out to explore the technology by learning about its predecessor: Neural Machine Translation. We revisited the past of neural machine translation in Chapter 2 and learned about the theoretical streams with respect to low resource machine translation. By providing a general intuition of the concepts of artificial neural networks in Section 3.1, insights into the intricacies of artificial neural networks in Chapter 3.1, sequence-to-sequence recurrent neural networks in Section 3.2.2, and the transformer architecture especially in Section 3.2.3, we unlocked the necessary components which allowed us to understand and build our own neural translation machine translation system for our experiment which we described in Chapter 4. With the help of the OpenNMT-Library and pytorch, we succesfully trained a reasonably capable translation model for translating between French and German, and were able to provide evidence for a saturating relation between the amount of pivot resources and translation quality which we showed in Chapter 5. In our motivation, we linked neural machine translation to large language models. Unlike traditional Neural Machine Translation (NMT) models, which are specifically designed to translate text from one language to another, LLMs operate with a different approach. LLMs are typically trained as self-supervised learners using techniques such as masked language modeling, where portions of a sentence are masked, and the model is trained to predict the missing words or sequences. This training method allows LLMs to learn rich representations of language and context across a broad range of tasks, including but not limited to text generation, question answering, and summarization based on vast amounts of training material. While NMT models focus on mapping between source and target languages to produce translations, LLMs generate text based on patterns learned from extensive datasets. Despite this difference, both types of model leverage similar underlying architectures and principles, such as the transformer which we explored in our thesis. With the help of our introduction to neural machine translation using the transformer architecture in Section 3.2, we are now able to grasp the general concept of LLMs and furthermore are able to connect new topics with respect to important existing concepts and components, including details such as the context vector c which was introduced in the context of RNNs in Section 3.2.2 which was evolved into the concept of attention heads in the Transformer introduced in Section 3.2.3.

Conclusion of the Experiment and its Results Additionally, our research successfully demonstrates the effectiveness of using a pivot-based approach in low-resource neural machine

translation. Through both visual and quantitative analyses, we observe that increasing the size of source-pivot and pivot-target corpora results in consistently improved translation quality. Models trained with the pivot strategy significantly outperform the baseline model, which uses no pivoting, particularly in low-resource scenarios. The best-performing pivot model, trained on 400,000 pivot sentences, showed an improvement of approximately 15.4% in BLEU, 5.8% in chrF, and 4.7% in TER over the naive baseline, and reached between 70.01% to 80.72% of the potential *assumed best case scenario* with only 1% of sentence pairs between source and target language while exploiting only 4% of additional pivot sentence pairs compared to the 10,000,000 sentence pair WMT19 baseline model. The results implicate great potential for finding trade-off points for diminishing returns. Moreover, the visual analysis of the training process indicates stable learning dynamics, though we encountered some anomalies, such as discrepancies in validation accuracy and potentially process-inhibiting training interruptions due to an strict early stopping mechanism. Despite these minor issues, the overall convergence and performance trends confirmed the robustness of our experiment configuration. Our results suggest a clear relation between the availability of pivot resources and the quality of translations, with a saturation point emerging as the size of the pivot corpus increases. Additionally, the performance gap between in-domain (Europarl) and out-of-domain (WMT19) test sets further highlights the importance of domain-specific data in achieving optimal translation results, and emphasize the importance of mindfully selecting training and evaluation corpora, potentially addressing conflicting aspects during analyses. In conclusion, our experiment and findings support the hypothesis that using pivot strategies in low-resource neural machine translation enhances translation quality, particularly when leveraging a specific factor of additional language resources. While during our research of existing material on the matter we came across experiments varying different parameters, such as architecture configuration or the optimal choice of pivot language, we did not come across an attempt to exclusively vary the level of available pivot resources. It might be seemingly obvious and one might argue "more is always better", but there are reasons, e.g., the limited availability of data or computational resources, why carefully planned compliance with constraints is beneficial over "aimlessly enlarging each and every configuration parameter, because one can". We add *the amount of pivot resources* to the list of potentially beneficial parameters to the quality of neural machine translation models when carefully formulated.

Outlook Our results support our hypothesis and a relation between pivot data availability and translation quality, however, the amount of evidence is limited and based on one language pair. By extending the number of language pairings, we could further solidify our hypothesis, and by varying not only the amount of pivot resources but also the amount of source to

target sentence pairs, specific points of diminishing returns can potentially be unraveled. A heuristic based on strong, potentially statistical evidence, such as "3to5× as much pivot data as source to target data for optimal results using pivoting strategies", could reveal itself, providing a road mark for further research endeavors. Road markings and heuristics such as these are substantial for researchers as they allow for great savings in time and effort. With a carefully planned grid of language pairs, models, and their evaluation, our hypothesis does not seem far fetched - on the contrary, our evidence already provides the reasonable insight that is needed to formulate consecutive studies on the matter. However, the models themselves are far from optimized. Although in comparison to the other models trained within the same experimental framework, our hypothesis holds, the same might not necessarily be true when training models on far more optimized configurations. These far more optimized configurations are either achievable by extending research in the domain of low resource language translation or gaining more hands-on experience on the matter. In general, the thesis served as a great framework to gain that hands-on experience, and while bumping into difficulties here and there, the author is now overflowing with ideas for improvements to the experiment design and new sections for thesis script. Neural Machine Translation and the technology revolving around large language models is complex, yet an exciting field of research to get lost in.

7 General Addenda

7.1 Mathematical Notation

Vectors if not obvious are for clarity indicated as $\vec{x}$ and are assumed to be column vectors. If obvious, $\vec{x}$ is simplified to just $\mathbf{x}$. Matrices $\mathbf{A}$ are uppercase bold roman letters. The transpose of a mathematical object is denoted by a superscript T, e.g., $\mathbf{M}^T$ or $\mathbf{x}^T$. $(w_1, \ldots, w_N)$ denotes a row vector with N elements, respectively $(w_1, \ldots, w_N)^T$ a column vector. Indices are written in subscript with i, j for column and row. Superscript (i), like $\mathbf{M}^{(\mathbf{1})}$ denotes the layer position in an artificial neural network setting. Subscripts (i), such as $h_{(3))}$ denote the time stept of a time-dependent process. We adopt common statistical notation, e.g., $\hat{x}$ is an estimation for x, and $x \sim p(x)$ means x is a sample from distribution $p(x)$. θ are (vector of) parameters, usually in context of estimations. θ is also a common symbol for threshold values. The difference will be obvious given the context. For identically distributed (i.d.d) $\vec{x}_1, \ldots, \vec{x}_N \sim p(x)$, where $\vec{x}_i$ is a D-Dimensional the column vector the notation $\mathbf{X}$ is used and represents a matrix of $\mathbf{X} \sim p(x)$ of dimension $N \times D$. When we speak about $x_{\bullet,j}$, we mean the j-th column vector of $\mathbf{X}$, the j-th feature of a data matrix. When we speak of $x_{i,\bullet}$, we mean the i-th row vector of $\mathbf{X}$, in other words, the i-th observation of a data matrix. The sets of numbers are written in $\mathbb{R}$ whereas sets in general are upper-case letters M. Where needed, matrices (and row/column vectors respectively) will be written out as

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

7.2 Source Code and GitHub

The thesis' code can be inspected under `git@github.com:thejonnyt/nmt-pivot-strat.git`. Primarily, *bash* and *python* were used to conduct the experiments. To run the experiments on a device, the docker environment has to be initialized. The Dockerfile lists all the necessary installs and sets up a sufficient environment. From within the Docker environment one has to call the *run.sh* script. Due to the error described in 5, the runtime is likely to crash during operation, leading to the memory allocation error, which has no consequence as the program stops at that point anyway. However, the error prevents the program to be written as a loop, looping through the different experiment sizes (0, 50, 100, 200, and 800), resulting in manual labour: after each training, the bash variables have to be adjusted accordingly, and the script

has to be manually resumed at the next stage (e.g., after finishing the source to pivot training, the pivot to target training has to be started manually). First, the *run.sh* procedure will attempt to download the language pairs. The code produces a new, shared folder, local:*nmt-experiment* or within-docker: *share*, in which the script's progress and results under *data* reside. With the help of the OpenNMT script *onmt_translate* and the *compute_metrics.py* script, which primarily wraps the sacrebleu library, the models can be evaluated. Feel free to contact the author of the thesis and its experiment via jonnytauscher@gmail.com (or, e.g., researchgate) for additional instructions if needed.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das

elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Leipzig, September 18, 2024 Author

List of Figures

[47] https://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png

List of Tables

References

Academy, EITCA. 2023. How does the batch size parameter affect the training process in a neural network? Online: https://eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/tensorflow/using-more-data/examination-review-using-more-data/how-does-the-batch-size-parameter-affect-the-training-process-in-a-neural-network/.

Alammar, Jay. 2018. The Illustrated Transformer. Online: http://jalammar.github.io/illustrated-transformer/.

Araabi, Ali, und Christof Monz. 2020. Optimizing Transformer for Low-Resource Neural Machine Translation. *Proceedings of the 28th International Conference on Computational Linguistics*, 3429–3435. Barcelona, Spain (Online): International Committee on Computational Linguistics. Online: https://www.aclweb.org/anthology/2020.coling-main.304.

Bahdanau, Dzmitry; Kyunghyun Cho; und Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473 [cs, stat]. Online: http://arxiv.org/abs/1409.0473.

Banerjee, Satanjeev, und Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, hrsg. von Jade Goldstein, Alon Lavie, Chin-Yew Lin, und Clare Voss, 65–72. Ann Arbor, Michigan: Association for Computational Linguistics. Online: https://aclanthology.org/W05-0909.

Bar-Hillel, Yehoshua. 1960. The present status of automatic translation of languages.

Bar-Hillel, Yehoshua. 1951. The present state of research on mechanical translation. *American Documentation* 2.229–237. Online: https://onlinelibrary.wiley.com/doi/10.1002/asi.5090020408.

van Biljon, Elan; Arnu Pretorius; und Julia Kreutzer. 2020. On Optimal Transformer Depth for Low-Resource Language Translation. arXiv:2004.04418 [cs]. Online: http://arxiv.org/abs/2004.04418.

Bishop, Christopher M., und Hugh Bishop. 2024. *Deep Learning: Foundations and Concepts*. Cham: Springer International Publishing. Online: https://link.springer.com/10.1007/978-3-031-45468-4.

Brown, Peter E; Vincent J Della Pietra; Stephen A Della Pietra; und Robert L Mercer. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics* 19.

Brown, Peter F.; John Cocke; Stephen A. Della Pietra; Vincent J. Della Pietra; Fredrick Jelinek; John D. Lafferty; Robert L. Mercer; und Paul S. Roossin. 1990. A Statistical Approach to Machine Translation. *Readings in Machine Translation*, *Computational Linguistic*, Ausg. 16, 79–85. The MIT Press. Online: https://direct.mit.edu/books/book/2694/chapter/72868/A-Statistical-Approach-to-Machine-Translation.

Castaño, M Asunción; Francisco Casacuberta; und Enrique Vidal. 1997. Machine translation using neural networks and finite-state models.

Chen, Shizhe; Qin Jin; und Jianlong Fu. 2019. From Words to Sentences: A Progressive Learning Approach for Zero-resource Machine Translation with Visual Pivots. arXiv:1906.00872 [cs]. Online: http://arxiv.org/abs/1906.00872.

Chen, Yun; Yang Liu; Yong Cheng; und Victor O.K. Li. 2017. A Teacher-Student Framework for Zero-Resource Neural Machine Translation. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1925–1935. Vancouver, Canada: Association for Computational Linguistics. Online: http://aclweb.org/anthology/P17-1176.

Cheng, Yong; Qian Yang; Yang Liu; Maosong Sun; und Wei Xu. 2017. Joint Training for Pivot-based Neural Machine Translation. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Online: https://www.ijcai.org/proceedings/2017/0555.pdf.

Cho, Kyunghyun; Bart van Merrienboer; Caglar Gulcehre; Dzmitry Bahdanau; Fethi Bougares; Holger Schwenk; und Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078 [cs, stat]. Online: http://arxiv.org/abs/1406.1078.

dontloo. 2019. Answer to "What exactly are keys, queries, and values in attention mechanisms?". Online: https://stats.stackexchange.com/a/424127.

Doshi, Ketan. 2021. Transformers Explained Visually: How it works. Online: towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step.

DuPont, Quinn. 2018. THE CRYPTOLOGICAL ORIGINS OF MACHINE TRANSLATION. Online: https://amodern.net/article/cryptological-origins-machine-translation/.

European Commission (last). 2011. Europarl Parallel Corpus. Online: https://www.statmt.org/europarl/.

Forcada, Mikel L., und Ramón P. Ñeco. 1997. Recursive hetero-associative memories for translation. *Biological and Artificial Computation: From Neuroscience to Technology*, Ausg. 1240, 453–462. Berlin, Heidelberg: Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science. Online: http://link.springer.com/10.1007/BFb0032504.

Goldberg, Yoav. 2017. *Neural Network Methods for Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Cham: Springer International Publishing. Online: https://link.springer.com/10.1007/978-3-031-02165-7.

Google DeepMind AlphaFold team. 2024. AlphaFold 3 predicts the structure and interactions of all of life's molecules. Online: https://blog.google/technology/ai/google-deepmind-isomorphic-alphafold-3-ai-model/.

Gordon, Cindy. 2023. ChatGPT Is The Fastest Growing App In The History Of Web Applications. Section: AI. Online: https://www.forbes.com/sites/cindygordon/2023/02/02/chatgpt-is-the-fastest-growing-ap-in-the-history-of-web-applications/.

Harris, Zellig S. 1954. Distributional Structure. *WORD* 10.146–162. Online: http://www.tandfonline.com/doi/full/10.1080/00437956.1954.11659520.

Hastie, Trevor; Robert Tibshirani; und Jerome Friedman. 2001. *The elements of statistical learning*. Springer series in statistics. New York, NY, USA: Springer New York Inc. tex.added-at: 2008-05-16T16:17:42.000+0200 tex.interhash: d585aea274f2b9b228fc1629bc273644 tex.intrahash: f58afc5c9793fcc8ad8389824e57984c tex.timestamp: 2008-05-16T16:17:43.000+0200.

He, Kaiming; Xiangyu Zhang; Shaoqing Ren; und Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs]. Online: http://arxiv.org/abs/1512.03385.

He, Kaiming; Xiangyu Zhang; Shaoqing Ren; und Jian Sun. 2016. Identity Mappings in Deep Residual Networks. arXiv:1603.05027 [cs]. Online: http://arxiv.org/abs/1603.05027.

Hebb, Donald O. 1949. *The organization of behavior: a neuropsychological theory*. 11. [print.] Ed. New York: Wiley.

Hoang, Vu Cong Duy; Philipp Koehn; Gholamreza Haffari; und Trevor Cohn. 2018. Iterative Back-Translation for Neural Machine Translation. *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, 18–24. Melbourne, Australia: Association for Computational Linguistics. Online: http://aclweb.org/anthology/W18-2703.

Hochreiter, Sepp, und Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9.1735–1780. Online: https://doi.org/10.1162/neco.1997.9.8.1735.

Hutchins, John. 1996. ALPAC: The (In)Famous Report. *Readings in Machine Translation*, 131–136. Online: https://direct.mit.edu/books/book/2694/chapter/72843/ALPAC-The-In-Famous-Report.

Hutchins, John. 1999. From the archives... Warren Weaver memorandum July 1949. *MT News International* 22.5–6. Online: https://aclanthology.org/1999.eamt-1.18.

Hutchins, John. 2002. Two precursors of machine translation: Artsrouni and Trojanskij.

Hutchins, John. 2006. The first public demonstration of machine translation: the Georgetown-IBM system, 7th January 1954. *MT News International*.

Hutchins, W John. 1989. Out of the shadows: a retrospect of machine translation in the eighties.

Johnson, Melvin; Mike Schuster; Quoc V. Le; Maxim Krikun; Yonghui Wu; Zhifeng Chen; Nikhil Thorat; Fernanda Viégas; Martin Wattenberg; Greg Corrado; Macduff Hughes; und Jeffrey Dean. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Transactions of the Association for Computational Linguistics* 5.339–351. Online: https://direct.mit.edu/tacl/article/43400.

Kahn, David. 1973. *The Codebreakers - The Story of Secret Writing*. Sphere.

Kalchbrenner, Nal, und Phil Blunsom. 2013. Recurrent Continuous Translation

Models. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1700–1709.

Keysers, Christian, und Valeria Gazzola. 2014. Hebbian learning and predictive mirror neurons for actions, sensations and emotions. *Philosophical Transactions of the Royal Society B: Biological Sciences* 369.20130175. Online: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4006178/.

Kim, Yunsu; Petre Petrov; Pavel Petrushkov; Shahram Khadivi; und Hermann Ney. 2019. Pivot-based Transfer Learning for Neural Machine Translation between Non-English Languages. arXiv:1909.09524 [cs]. Online: http://arxiv.org/abs/1909.09524.

Koehn, Philipp. 2017. Neural Machine Translation. arXiv:1709.07809 [cs]. Online: http://arxiv.org/abs/1709.07809.

Kudo, Taku, und John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. arXiv:1808.06226 [cs]. Online: http://arxiv.org/abs/1808.06226.

Lample, Guillaume, und Alexis Conneau. 2019. Cross-lingual Language Model Pretraining. arXiv:1901.07291 [cs]. Online: http://arxiv.org/abs/1901.07291.

Lankford, Seamus, und Haithem Afli. 2021. Transformers for Low-Resource Languages: Is Fe´idir Linn!

Leng, Yichong; Xu Tan; Tao Qin; Xiang-Yang Li; und Tie-Yan Liu. 2019. Unsupervised Pivot Translation for Distant Languages. arXiv:1906.02461 [cs]. Online: http://arxiv.org/abs/1906.02461.

Libovický, Jindřich. 2022. Answer to "Why are residual connections needed in transformer architectures?". Ph.D. in Computational Linguistics, Charles University, Faculty of Mathematics and Physics (2013 – 2019). Online: https://stats.stackexchange.com/a/565203.

Lukasz Kaiser. 2017. Why does transformer use noam as the learning rate decay scheme? · Issue #280 · tensorflow/tensor2tensor. Online: https://github.com/tensorflow/tensor2tensor/issues/280.

McCulloch, Warren S., und Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity.

Mhaskar, Shivam, und Pushpak Bhattacharyya. 2021. Pivot Based Transfer Learning

for Neural Machine Translation: CFILT IITB @ WMT 2021 Triangular MT. *Proceedings of the Sixth Conference on Machine Translation (WMT)* November.336–340.

Mikolov, Tomas; Kai Chen; Greg Corrado; und Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs]. Online: `http://arxiv.org/abs/1301.3781`.

NVIDIA. 2024. NVIDIA CEO Jensen Huang Keynote at COMPUTEX 2024. Online: `https://www.youtube.com/watch?v=pKXDVsWZmUU`.

Olah, Christopher. 2014. Deep Learning, NLP, and Representations - colah's blog. Online: `http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/`.

Olah, Christopher. 2015. Calculus on Computational Graphs: Backpropagation – colah's blog. Online: `http://colah.github.io/posts/2015-08-Backprop/`.

Olden, J.; M. Joy; und R. Death. 2004. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling* 178.389–397.

Osborne, Chris Callison-Burch Miles, und Philipp Koehn. 2006. Re-evaluating the Role of BLEU in Machine Translation Research. *Proceedings of the Conference*, Ausg. 11th Conference of the European Chapter of the Association for Computational Linguistics, 249–256. Trento, Italy: The Association for Computer Linguistics. Online: `https://aclanthology.org/E06-1032/`.

Papineni, Kishore; Salim Roukos; Todd Ward; und Wei-Jing Zhu. 2001. BLEU: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, 311. Philadelphia, Pennsylvania: Association for Computational Linguistics. Online: `http://portal.acm.org/citation.cfm?doid=1073083.1073135`.

Pascanu, Razvan; Tomas Mikolov; und Yoshua Bengio. 2013. On the difficulty of training Recurrent Neural Networks. arXiv:1211.5063 [cs]. Online: `http://arxiv.org/abs/1211.5063`.

Patel, Dylan. 2023. Google "We Have No Moat, And Neither Does OpenAI". Online: `https://www.semianalysis.com/p/google-we-have-no-moat-and-neither`.

Paul, Michael; Hirofumi Yamamoto; Eiichiro Sumita; und Satoshi Nakamura. 2009. On the importance of pivot language selection for statistical machine translation.

Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers on - NAACL '09, 221. Boulder, Colorado: Association for Computational Linguistics. Online: http://portal.acm.org/citation.cfm?doid=1620853.1620914.

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

Pierce, John R.; John B. Carroll; Eric P. Hamp; David G. Hays; Charles F. Hockett; Anthony G. Oettinger; und Alan Perlis. 1966. *Language and Machines: Computers in Translation and Linguistics*. Washington, D.C.: National Academies Press. Pages: 9547. Online: http://www.nap.edu/catalog/9547.

Popović, Maja. 2015. chrF: character n-gram F-score for automatic MT evaluation. *Proceedings of the Tenth Workshop on Statistical Machine Translation*, hrsg. von Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, und Pavel Pecina, 392–395. Lisbon, Portugal: Association for Computational Linguistics. Online: https://aclanthology.org/W15-3049.

Post, Matt. 2018. A Call for Clarity in Reporting BLEU Scores. *Proceedings of the Third Conference on Machine Translation: Research Papers*, 186–191. Belgium, Brussels: Association for Computational Linguistics. Online: http://aclweb.org/anthology/W18-6319.

Rothman, Denis. 2024. *Transformers for natural language processing and computer vision: explore generative AI and large language models with Hugging Face, ChatGPT, GPT-4V, and DALL-E3*. Third edition Ed. Birmingham, UK: Packt Publishing Ltd. OCLC: 1424949941.

Schmidhuber, Jürgen; Kristinn R. Thórisson; Moshe Looks; David Hutchison; Takeo Kanade; Josef Kittler; Jon M. Kleinberg; Friedemann Mattern; John C. Mitchell; Moni Naor; Oscar Nierstrasz; C. Pandu Rangan; Bernhard Steffen; Madhu Sudan; Demetri Terzopoulos; Doug Tygar; Moshe Y. Vardi; und Gerhard Weikum (Hg.) 2011. *Artificial General Intelligence: 4th International Conference, AGI 2011, Mountain View, CA, USA, August 3-6, 2011. Proceedings*, *Lecture Notes in Computer Science*, Ausg. 6830. Berlin, Heidelberg: Springer Berlin Heidelberg. Online: https://link.springer.com/10.1007/978-3-642-22887-2.

Schubert, Klaus. 1988. Implicitness as a guiding principle in machine translation. *Proceedings of the 12th conference on Computational linguistics* -, Ausg. 2, 599–601. Budapest, Hungry: Association for Computational Linguistics. Online: http://portal.acm.org/citation.cfm?doid=991719.991761.

Sennrich, Rico; Barry Haddow; und Alexandra Birch. 2016a. Improving Neural Machine Translation Models with Monolingual Data. arXiv:1511.06709 [cs]. Online: http://arxiv.org/abs/1511.06709.

Sennrich, Rico; Barry Haddow; und Alexandra Birch. 2016b. Neural Machine Translation of Rare Words with Subword Units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1715–1725. Berlin, Germany: Association for Computational Linguistics. Online: http://aclweb.org/anthology/P16-1162.

Shatz, Carla J. 1992. The developing brain. *Scientific American* 267.60–67, Publisher: Scientific American, a division of Nature America, Inc. Online: http://www.jstor.org/stable/24939213.

Snover, Matthew; Bonnie Dorr; Rich Schwartz; Linnea Micciulla; und John Makhoul. 2006. A Study of Translation Edit Rate with Targeted Human Annotation. *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, 223–231. Cambridge, Massachusetts, USA: Association for Machine Translation in the Americas. Online: https://aclanthology.org/2006.amta-papers.25.

Sutskever, Ilya; Oriol Vinyals; und Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. *arXiv preprint arXiv:1409.3215* abs/1409.3215. Online: http://arxiv.org/abs/1409.3215.

Taori, Rohan, und Ishaan Gulrajani. 2023. Stanford CRFM. Online: https://crfm.stanford.edu/2023/03/13/alpaca.html?ref=the-batch-deeplearning-ai.

Vaswani, Ashish; Noam Shazeer; Niki Parmar; Jakob Uszkoreit; Llion Jones; Aidan N. Gomez; Lukasz Kaiser; und Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs]. Online: http://arxiv.org/abs/1706.03762.

Wang, Rui; Xu Tan; Renqian Luo; Tao Qin; und Tie-Yan Liu. 2021. A Survey on Low-Resource Neural Machine Translation. arXiv:2107.04239 [cs]. Online: http://arxiv.org/abs/2107.04239.

Weaver, Warren. 1949. Translation. Online: https://aclanthology.org/1952.earlymt-1.1.pdf.

Werbos, P.J. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78.1550–1560. Online: http://ieeexplore.ieee.org/document/58337/.

Wikipedia. 2024. AI winter. Page Version ID: 1228768505. Online: https://en.wikipedia.org/w/index.php?title=AI_winter&oldid=1228768505.

Williams, Stephen; Pankaj Sah; Victor Anggono; und Frederic Meunier. 2016. Action potentials and synapses. Online: https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/action-potentials-and-synapses.

Wu, Yonghui; Mike Schuster; Zhifeng Chen; Quoc V. Le; Mohammad Norouzi; Wolfgang Macherey; Maxim Krikun; Yuan Cao; Qin Gao; Klaus Macherey; Jeff Klingner; Apurva Shah; Melvin Johnson; Xiaobing Liu; Łukasz Kaiser; Stephan Gouws; Yoshikiyo Kato; Taku Kudo; Hideto Kazawa; Keith Stevens; George Kurian; Nishant Patil; Wei Wang; Cliff Young; Jason Smith; Jason Riesa; Alex Rudnick; Oriol Vinyals; Greg Corrado; Macduff Hughes; und Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144 [cs]. Online: http://arxiv.org/abs/1609.08144.

Xia, Yingce; Di He; Tao Qin; Liwei Wang; Nenghai Yu; Tie-Yan Liu; und Wei-Ying Ma. 2016. Dual Learning for Machine Translation. arXiv:1611.00179 [cs]. Online: http://arxiv.org/abs/1611.00179.

Zhang, Jiajun, und Chengqing Zong. 2016. Exploiting Source-side Monolingual Data in Neural Machine Translation. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 1535–1545. Austin, Texas: Association for Computational Linguistics. Online: http://aclweb.org/anthology/D16-1160.

Zhang, Mos. 2017. History and Frontier of the Neural Machine Translation | by Synced | SyncedReview | Medium. Online: http://webcache.googleusercontent.com/search?q=cache:https://medium.com/syncedreview/history-and-frontier-of-the-neural-machine-translation-dc981d25422d.

Zoph, Barret; Deniz Yuret; Jonathan May; und Kevin Knight. 2016. Transfer

Learning for Low-Resource Neural Machine Translation. arXiv:1604.02201 [cs]. Online: http://arxiv.org/abs/1604.02201.