NEURAL MACHINE TRANSLATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Minh-Thang Luong
November 2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Christopher D. Manning) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Dan Jurafsky)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Andrew Ng)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Quoc V. Le)

Approved for the Stanford University Committee on Graduate Studies

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> The Babel fish is small, yellow, leech-like, and probably the oddest thing in the universe. If you stick a Babel fish in your ear, you can instantly understand anything in any form of language.
>
> *The Hitchhiker's Guide to the Galaxy*. Douglas Adams.

Human languages are diverse with about 6000 to 7000 languages spoken worldwide (Anderson, 2010). As civilization advances, the need for seamless communication and understanding across languages becomes more and more crucial. Machine translation (MT), the task of teaching machines to learn to translate automatically across languages, as a result, is an important research area. MT has a long history (Hutchins, 2007) from the original phiosophical ideas of universal languages in the 17^{th} century to those first practical suggestions in the 1950s, most notably an influential proposal by Weaver (1949) which marked the beginnings of MT research in the United States. In that memorandum, Warren Weaver touched on the idea of using computers to translate, specifically addressing the language ambiguity problem by combining his knowledge of statistics, cryptography, information theory, as well as logical and linguistic universals (Hutchins, 2000). Since then, MT has gone through many periods of great development but also encountered several stagnant phases as illustrated in Figure 1.1. Despite several moments of excitement that led to hopes that MT will be solved "very soon" such as the 701 translator (Sheridan, 1955) developed by scientists at Georgetown and IBM in the 1950s and the popular Google Translate at the

beginning of the 21st century (Brants et al., 2007), MT remains an extremely challenging problem (Kelly, 2014; David, 2016). This motivates my work in the area of machine translation; specifically, in this thesis, the goal is to advance neural machine translation (NMT), a new promising approach for MT developed just recently, over the past two years. The results achieved in this thesis on NMT together with work from other researchers have eventually produced a significant leap in the translation quality as illustrated in Figure 1.1. Before delving into details of the thesis, we now walk the audience through the background and a bit of the development history of machine translation.

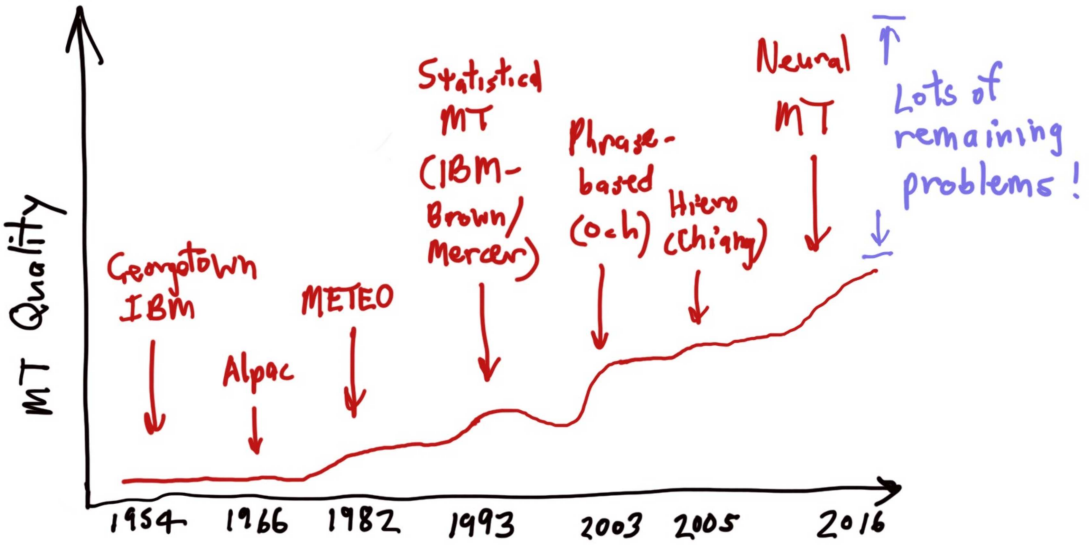

Figure 1.1: **Machine translation progress** – from the 1950s, the starting of modern MT research, until the time of this thesis, 2016, in which neural MT becomes a dominant approach. Image courtesy of Christopher D. Manning.

1.1 Machine Translation

Despite much enthusiasm, the beginning period of MT research in the 1950-60s, was mostly about direct word-for-word replacement based on bilingual dictionaries.[1] An MT winner quickly came right after the ALPAC report in 1966 pointing out that "there is no

[1]There are also proposals for "interlingual" and "transfer" approaches but these seemed to be too challenging to achieve, not to mention limitations in hardware at that time(Hutchins, 2007).

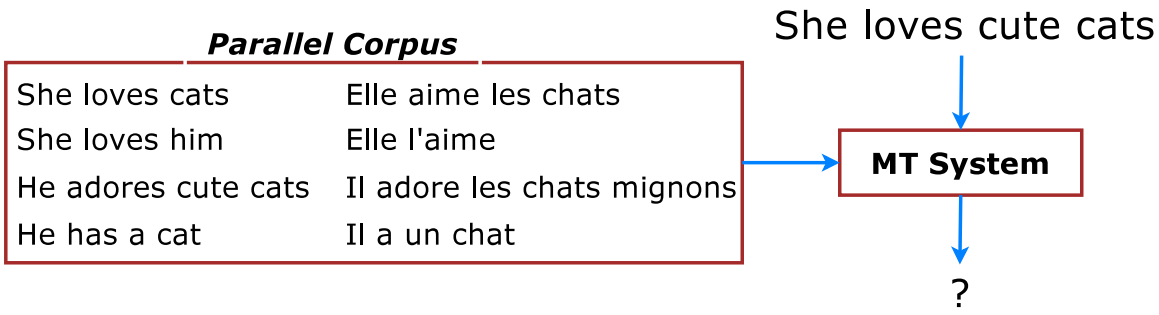

Figure 1.2: **Corpus-based approaches to machine translation** – a general setup in which MT systems are built from parallel corpora of sentence pairs having the same meaning. Once built, systems are used to translate new unseen sentences, e.g., "She loves cute cats".

immediate or predictable prospect of useful machine translation", which hampered MT research for over a decade. Fast-forwarding through the resurgence in the 1980s beginning with Europe, Japan, and gradually the United States, modern statistical MT started out with a seminal work by IBM scientists (Brown et al., 1993). The proposed *corpus-based* approaches require minimal linguistic content and only need a *parallel* dataset of sentence pairs which are translations of one another, to train MT systems. Such a language-independent setup is illustrated in Figure 1.2. In more detail, instead of hand building bilingual dictionaries which can be costly to obtain, Brown and colleagues proposed to learn these dictionaries, or *translation models*, probabilistically from parallel corpora. To accomplish this, they propose a series of 5 algorithms of increasing complexity, often referred as IBM Models 1-5, to learn *word alignment*, a mapping between source and target words in a parallel corpus, as illustrated in Figure 1.3. The idea is simple: the more often two words, e.g., "loves" and "aime", occur together in different sentence pairs, the more likely they are aligned to each other and have equivalent meanings.

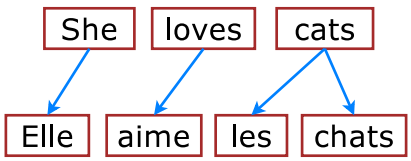

Figure 1.3: **Word-based alignment** – example of an alignment between source and target words. In IBM alignment models, each target word is aligned to at most one source word.

Once a translation model, i.e., a probabilistic bilingual dictionary, has been learned, IBM model 1, the simplest and the most naïve one among the five proposed algorithms, translates a new source sentence as follows. First, it decides on how long the translation is as well as how source words will be mapped to target words as illustrated in Step 1 of Figure 1.4. Then, in Step 2, it produces a translation by selecting for each target position a word that is the best translation for the aligned source word according to the bilingual dictionary. Subsequent IBM models build on top of one another and refine the translation story such as better modeling the reordering structure, i.e., how word positions differ between source and target languages. We refer the audience to the original IBM paper or Chapter 25 of (Jurafsky and Martin, 2009) for more details.

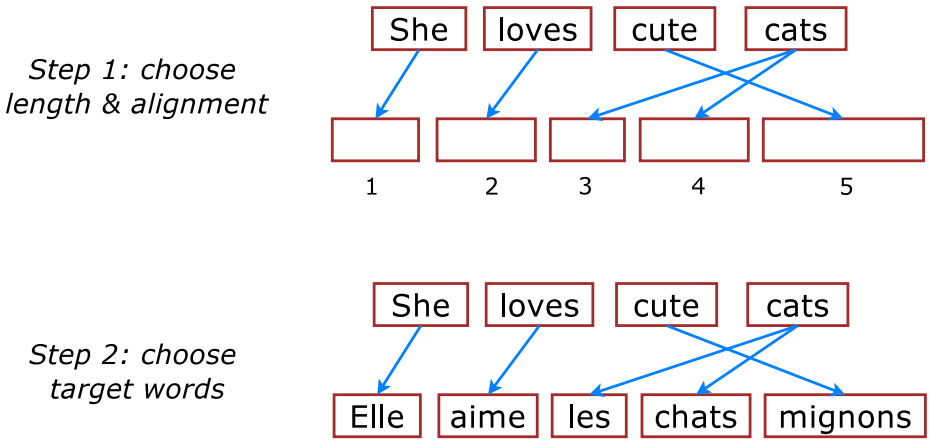

Figure 1.4: **A simple translation story** – example of the generative story in IBM Model 1 to produce a target translation given a source sentence and a learned translation model.

There are, however, two important details that we left out in the above translation story, the *search* process and the *language modeling* component. In Step 1, one might wonder among the exponentially many choices, how do we know what the right translation length is and how source words should be mapped to target words? The search procedure informally helps us "browse" through a manageable set of candidates which are likely to include a good translation; whereas, the language model will help us select the best translation among these candidates. We will defer details of the search process to later since it is dependable on the exact translation model being used. Language modeling, on the other hand, is an important concept which has been studied earlier in speech recognition (Katz, 1987). In

a nutshell, a language model (LM) learns from a corpus of monolingual text in the target language and collect statistics on which sequence of words are likely to go with one another. When applying to machine translation, an LM will assign high scores for coherent and natural-sounding translations and low scores for bad ones. For our example in the above figure, if the model happens to choose a wrong alignment, e.g., "cute" goes to position 3 while "cats" goes to positions 4 and 5, an LM will alert us with a lower score given to that incorrect translation "Elle aime mignons les chats" compared to the translation "Elle aime les chats mignons" with a correct word ordering structure.[2]

While the IBM work had a huge impact on the field of statistical MT, researchers quickly realized that word-based MT is insufficient as words require context to properly translate, e.g., "bank" has two totally different meanings when preceded by "financial" and "river". As a result, *phrase-based models*, (Marcu and Wong, 2002; Zens et al., 2002; Koehn et al., 2003), inter alia, became the de facto standard in MT research and are still being the dominant approach in existing commercial systems such as Google Translate until now. Much credit went to Och's work on *alignment templates*, starting with his thesis in 1998 and later in (Och and Ney, 2003, 2004). The idea of alignment templates is to enable phrase-based MT by first symmetrizing[3] the alignment to obtain many-to-many correspondences between source and target words; in contrast, the original IBM models only produce one-to-many alignments. From the symmetrized alignment, several heuristics have been proposed to extract phrase pairs; the general idea is that phrase pairs need to be "consistent" with their alignments: each word in a phrase should not be aligned to a word outside of the other phrase. These pairs are stored in what called a *phrase table*

[2] For completeness, translation and language models are integrated together in an MT system through the *Bayesian noisy channel* framework as follows:

$$\hat{t} = \underset{t}{\operatorname{argmax}}\, P(t|s) \approx \underset{t}{\operatorname{argmax}}\, P(s|t)P(t) \tag{1.1}$$

Here, we have a source sentence s in which we ask our *decoder*, an algorithm that implements the aforementioned search process, to find the best translation, the argmax part. $P(s|t)$ represents the *translation* model, the faithfulness of the translation in terms of meaning preservation between the source and the target sentences; whereas $P(t)$ represents the *language* model, the fluency of the translated text.

[3] Symmetrization is achieved by training IBM models in both directions, source to target and vice versa, then intersecting the alignments. There are subsequent techniques that jointly train alignments in both directions such as (Liang et al., 2006).

together with various scores to evaluate phrase pairs in different aspects, e.g., how equivalent the meaning is, how good the alignment is, etc. Figure 1.5 gives an example of how a phrase-based system translates.

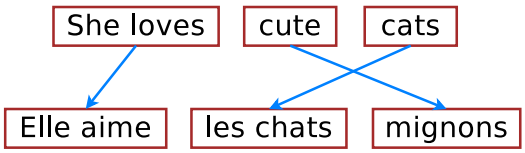

Figure 1.5: **Phrase-based machine translation** (MT) – example of how phrase-based MT systems translate a source sentence "She loves cute cats" into a target sentence "Elle aime les chats mignons": sentences are split into chunks and phrases are translated.

State-of-the-art MT systems, in fact, contain more components than just the two basic translation and language models. There are many knowledge sources that can be useful to the translation task, e.g., language model, translation model, reversed translation model, reordering model[4], length/unknown penalties[5], etc. To incorporate all of these features, modern MT systems use a popular framework in natural language processing, called the *maximum-entropy* or *log-linear* model (Berger et al., 1996; Och and Ney, 2002), which has as its special case the Bayesian noisy channel model that we briefly mentioned in Eq. (1.1).

Training log-linear MT models can be done using the standard *maximum likelihood estimation* approach. However, in practice, these models are learned by directly optimizing translation quality metrics such as BLEU (Papineni et al., 2002) in a technique known as *minimum error rate training* or *MERT* (Och, 2003). Here, BLEU is an inexpensive automatic way of evaluating the translation quality; the idea is to count words and phrases that overlap between machine and human outputs. Despite many criticisms, BLEU is still the most widely used evaluation metric up until now thanks to its simplicity.

Lastly, there has also been effort in adding *syntax* to machine translation through tree-based models such as work by Wu (1997); Yamada and Knight (2001); Chiang (2005), inter alia. As illustrated in Figure 1.1, these approaches do provide gains for several language

[4]Reordering models learn the patterns of how words move across source and target sentences and are trained based on the word alignment.

[5]To produce translations of appropriate lengths and with a reasonable amount of unknown words, e.g., unseen names and numbers at test time.

pairs, mostly those that are significant different in terms of sentence structures such as Chinese and English. However, the gains are often modest compared to the added complexity of tree-based models such as requirements to have good parsers and syntactic annotations.

For more information on evaluation metrics, tree-based models, and other topics in statistical machine translation, we refer the audience to an excellent book by Koehn (2010).

1.2 Neural Machine Translation

While statistical machine translation (SMT) has been successfully deployed in many commercial systems, it does not work very well and suffers from the following two major drawbacks. First, translation decisions are *locally determined* as we translate phrase-by-phrase and long-distance dependencies are often ignored. More problematically, the entire MT pipeline is becoming increasingly *complex* as more and more features are added to the log-linear framework such as in recent MT systems (Galley and Manning, 2008; Chiang et al., 2009; Green et al., 2013). Many different components need to be tuned separately, e.g., translation models, language models, reordering models, etc., which makes it difficult to combine them together and to innovate. As a result, the translation quality has saturated for SMT and big changes to the existing framework were in dire need.

Neural Machine Translation (NMT) is a new approach that addresses the aforementioned problems. First, NMT is a *single big neural network* (with millions of artificial neurons) that is designed to model the entire MT process (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). NMT requires *minimal domain knowledge*, just a parallel corpus of source and target sentence pairs, similar to SMT, but with far less preprocessing steps before a translation model can be built. The most appealing feature of NMT is that it can be trained *end-to-end* directly from the learning objective; hence, eliminating the problem of having to learn multiple components in SMT systems.

Unlike those intricate decoders (the search procedure we mentioned earlier) in popular SMT packages (Koehn et al., 2007; Chiang, 2007; Dyer et al., 2010; Cer et al., 2010), the translation story of NMT is conceptually simple. NMT translates as follows: an *encoder*

reads through the given source sentence to build a "thought" vector[6], a sequence of numbers that represents the sentence meaning; a *decoder*, then, processes the sentence vector to emit a translation, as illustrated in Figure 1.7. This is often referred to as the encoder-decoder architecture.[7] In this manner, NMT addresses the local translation problem in SMT; it does not do phrase-by-phrase translation. Instead, NMT gathers information from the entire source sentence before translating; as a result, it can capture *long-range dependencies* in languages, e.g., gender agreements; structural orderings of subject, verb, and object; etc.

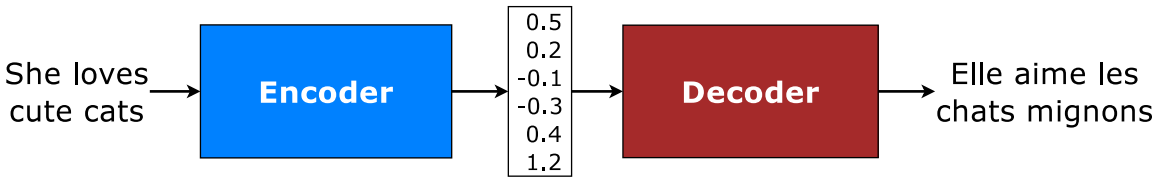

Figure 1.6: **Encoder-decoder architecture** – example of the general approach for NMT. An *encoder* converts a source sentence into a meaning vector which is passed through a *decoder* to produce a translation.

A realization of NMT is to use a powerful model for sequential data, namely recurrent neural network (RNN), for both the encoder and decoder (Sutskever et al., 2014; Cho et al., 2014). Interested readers can find details about RNNs in Section 2.2; in a nutshell, RNNs allow us to build representations for variable-length input – in our case, sentences – using a dynamic memory structure. In Figure 1.7, deep RNNs with two stacking layers are used to build a sequence-based NMT: an encoder first constructs a representation for a source sequence; a decoder, then, generates a target sequence, one symbol at a time until a special end-of-sequence symbol is produced.

Sequence-based NMT has several advantages. First, NMT beam-search decoders that generate words from left to right can be easily implemented, unlike the highly complex beam-search decoders in SMT (Koehn et al., 2003). More importantly, the use of RNNs allow NMT for *better generalization* to very long sequences while not having to explicitly

[6]This term was coined by Geoffrey Hinton in this article https://www.theguardian.com/science/2015/may/21/google-a-step-closer-to-developing-machines-with-human-like-intelligence.

[7]Allen (1987); Chrisman (1991) wrote the very first papers on encoder-decoder models for translation!

store any gigantic phrase tables or language models as in the case of SMT. As sequence-based NMT is currently the de facto approach, we will use NMT to generally refer to sequence-based NMT throughout this thesis unless otherwise stated.

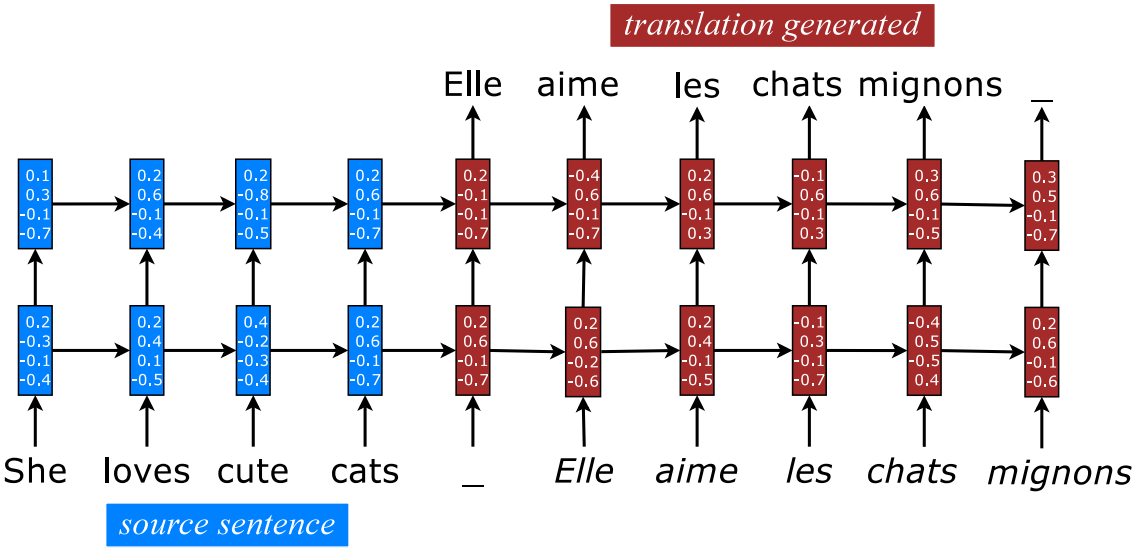

Figure 1.7: **Sequence Models for NMT** – example of a deep recurrent architecture for translating a source sentence "She loves cute cats" into a target sentence "Elle aime les chats mignons". On the decoder side, *words* generated from previous timesteps are used as inputs for the next ones. Here, "_" marks the end of a sentence.

1.3 Thesis Outline

Despite all the aforementioned advantages and potentials, the early NMT architecture (Sutskever et al., 2014; Cho et al., 2014) still has many drawbacks. In this thesis, I will highlight three problems pertaining to the existing NMT model, namely the *vocabulary coverage*, the *memory constraint*, and the *language complexity* issues. Each chapter is devoted to solving one of these problems. In each chapter, I will describe how I have pushed the limits of NMT, making it applicable to a wide variety of languages with state-of-the-art performance such as English-French (Luong et al., 2015c), English-German (Luong et al., 2015b; Luong and Manning, 2015), English-Vietnamese (Luong and Manning, 2015), and English-Czech (Luong and Manning, 2016). Towards the *future* of NMT, I answer two

questions: (1) whether we can improve translation by jointly learning from a wide variety of sequence-to-sequence tasks such as parsing, image caption generation, and auto-encoders or skip-thought vectors (Luong et al., 2016); and (2) whether we can compress NMT for mobile devices (See et al., 2016). In brief, this thesis is organized as follows. I start off by providing background knowledge on RNN and NMT in Chapter 2. The aforementioned three problems and approaches to the future of NMT are detailed in Chapters 3, 4, 5, and 6 respectively, which we will go through one by one next. Chapter 7 wraps up and discusses remaining challenges in NMT research.

Copy Mechanisms

A significant weakness in the first NMT systems is their inability to correctly translate very rare words: end-to-end NMTs tend to have relatively small vocabularies with a single `<unk>` symbol that represents every possible out-of-vocabulary (OOV) word. In Chapter 3, I propose simple and effective techniques to address this *vocabulary size* problem through teaching NMT to "copy" words from source to target. Specifically, I train an NMT system on data that is augmented by the output of a word alignment algorithm, allowing the NMT system to emit, for each OOV word in the target sentence, the position of its corresponding word in the source sentence. This information is later utilized in a post-processing step that translates every OOV word using a dictionary. My experiments on the WMT'14 English to French translation task show that this method provides a substantial improvement of up to 2.8 BLEU points over an equivalent NMT system that does not use this technique. With 37.5 BLEU points, this NMT system is the first to surpass the best result achieved on a WMT'14 contest task. *This chapter is based on the following paper (Luong et al., 2015c) in which I, Ilya Sutskever, and Quoc Le share the first co-authorship.*

Attention Mechanisms

While NMT can translate well for short- and medium-length sentences, it has a hard time dealing with long sentences. An attentional mechanism was proposed by Bahdanau et al. (2015) to address that *sentence length* problem by selectively focusing on parts of the

source sentence during translation. However, there has been little work exploring useful architectures for attention-based NMT. Chapter 4 examines two simple and effective classes of attentional mechanism: a *global* approach which always attends to all source words and a *local* one that only looks at a subset of source words at a time. I demonstrate the effectiveness of both approaches on the WMT translation tasks between English and German in both directions. With local attention, I achieve a significant gain of 5.0 BLEU points over non-attentional systems that already incorporate known techniques such as dropout. My ensemble model using different attention architectures yields a new state-of-the-art result in the WMT'15 English to German translation task with 25.9 BLEU points, an improvement of 1.0 BLEU points over the existing best system backed by NMT and an n-gram reranker. *This chapter is based on the following papers (Luong et al., 2015b; Luong and Manning, 2015).* Mention results from IWSLT, for TED talk English-German, English-Vietnamese

Hybrid Models

Nearly all previous NMT work has used quite restricted vocabularies, perhaps with a subsequent method to patch in unknown words such as the copy mechanisms mentioned earlier. While effective, the copy mechanims cannot deal with all the complexity of human languages such as rich morphology, neologisms, and informal spellings. Chapter 5 presents a novel word-character solution to that *language complexity* problem towards achieving open vocabulary NMT. I build hybrid systems that translate mostly at the *word* level and consult *character* components for rare words. My character-level recurrent neural networks compute source word representations and recover unknown target words when needed. The twofold advantage of such a hybrid approach is that it is much faster and easier to train than character-based ones; at the same time, it never produces unknown words as in the case of word-based models. On the WMT'15 English to Czech translation task, this hybrid approach offers an addition boost of $+2.1-11.4$ BLEU points over models that already handle unknown words. My best system achieves a new state-of-the-art result with 20.7 BLEU score. I demonstrate that my character models can successfully learn to not only generate well-formed words for Czech, a highly-inflected language with a very complex vocabulary, but also build correct representations for English source words. *This chapter is based on the following paper (Luong and Manning, 2016).*

NMT Future

Chapter 6 answers the two aforementioned questions for the future of NMT: whether we can utilize other tasks to improve translation and whether we can compress NMT models. The former question is important because of the fact that the first NMT systems only utilize parallel corpora despite an abundant amount of available data from monolingual and multilingual corpora as well as data from related tasks. The latter question is motivated by the indispensable role of mobile devices in nowadays society[8] and the fact that state-of-the-art NMT models are beyond the storage capacity of existing mobile gadgets.

For the first question, I examine three multi-task learning (MTL) settings for sequence to sequence models: (a) the *one-to-many* setting – where the encoder is shared between several tasks such as machine translation and syntactic parsing, (b) the *many-to-one* setting – useful when only the decoder can be shared, as in the case of translation and image caption generation, and (c) the *many-to-many* setting – where multiple encoders and decoders are shared, which is the case with unsupervised objectives and translation. My results show that training on a small amount of parsing and image caption data can improve the translation quality between English and German by up to 1.5 BLEU points over strong single-task baselines on the WMT benchmarks. Rather surprisingly, I have established a new *state-of-the-art* result in constituent parsing with 93.0 F_1 by utilizing translation data. Lastly, I reveal interesting properties of the two unsupervised learning objectives, autoencoder and skip-thought, in the MTL context: an autoencoder helps less in terms of perplexity but more on BLEU scores compared to skip-thought. *This section is based on the following paper (Luong et al., 2016).*

For the second question, I examine three simple magnitude-based pruning schemes to compress NMT models, namely *class-blind*, *class-uniform*, and *class-distribution*, which differ in terms of how pruning thresholds are computed for the different classes of weights in the NMT architecture. I demonstrate the efficacy of weight pruning as a compression technique for a state-of-the-art NMT system. I show that an NMT model with over 200

[8] In 2014, the number of mobile devices is more than the number of people in the world according to http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html.

million parameters can be pruned by 40% with very little performance loss as measured on the WMT'14 English-German translation task. This sheds light on the distribution of redundancy in the NMT architecture. My main result is that with *retraining*, I can recover and even surpass the original performance with an 80%-pruned model. *This section is based on the following paper (See et al., 2016) in which Abigail See and I share the first co-authorship.*

Wrap-up

In summary, this thesis has touched on a variety of aspects in which NMT can be significantly improved. My hope is to convince the audience at the end of this thesis that NMT models have successfully taken over the role of SMT models and will continue to be the de factor standard for several years to come. Still, there are many challenging and rewarding problems to be explored which I will summarize in the conclusion chapter. *The material is based on an NMT tutorial given by me, Kyunghuyn Cho, and Christopher D. Manning at ACL'2016.*[9]

[9]The tutorial website is https://sites.google.com/site/acl16nmt/.

Chapter 2

Background

> Give me a place to stand and I will move the earth.
>
> Archimedes.

In this chapter, we provide background knowledge on three main topics, namely language model (LM), recurrent neural network (RNN), and neural machine translation (NMT). Language modeling is an important concept in natural language processing to allow one to do *word prediction*, i.e., guessing which word will come next given a preceding context. As we shall see later, there is an interesting fact that for neural machine translation, it all started from language modeling. Before we get into NMT, we will go through the basics of recurrent neural network, the heart of sequence-based NMT, to explain how RNNs can naturally and effectively model *variable-length* inputs, or sentences in the context of the translation task. We cover in depth one particular type of RNN, the *Long Short-term Memory* (LSTM), that makes training RNNs easier. Interested readers can find all the details of how to implement LSTM "by hand" with detailed formulas on gradient computation as compared to the automatic differentiation feature given by nowadays deep learning frameworks. The understanding of language modeling will allow us to extend RNNs into recurrent neural language models which enable *language generation*, a key step in NMT. Lastly, with RNN as a basic building block, we describe key elements of an NMT system as well as tips and tricks for better training and testing NMT.

2.1 Language Model

As we have discussed in Section 1.1, language modeling plays an indispensable role in MT to ensure that systems produce fluent translations. Specifically, the job of an LM is to specify a probability distribution over sequences of symbols (often, words) so that one can judge if a sequence of words is more likely or "fluent" than another. To accomplish that, an LM decomposes the probability of a word sequence $y = y_1, \ldots, y_m$ as:

$$p(y) = \prod_{i=1}^{m} p(y_i|y_{<i}) \tag{2.1}$$

In the above formula, each of the individual terms $p(y_i|y_{<i})$ is the conditional probability of the current word y_i given previous words $y_{<i}$, also referred to as the *context* or the *history*. To model these conditional probabilities, traditional n-gram LMs have to resort to the Markovian assumption to consider only a fixed context window of $n-1$ words, effectively modeling $p(y_i|y_{i-n+1}, \ldots, y_{i-1})$. In fact, n-gram LMs have to explicitly store and handle all possible n-grams occurred in a training corpus, the number of which quickly becomes enormous. As a result, despite much research in this area (Rosenfeld, 2000; Stolcke, 2002; Teh, 2006; Federico et al., 2008; Heafield, 2011), inter alia, n-gram LMs can only handle short contexts of about 4 to 6 words, and does not generalize well to unseen n-grams.

Neural language models (NLMs), first proposed by Bengio et al. (2003) and enhanced by others such as Morin and Bengio (2005); Mnih and Hinton (2009); Mnih and Teh (2012), have addressed the aforementioned concerns using two ideas: (a) *dense distributed representations* for words which encourage sharing of statistical weights between similar words; and (b) *feed-forward neural networks* to allow for better composition of unseen word sequences at test time without having to explicitly store all enumerations of n-grams. These features function as a way to combat the "curse" of dimensionality in language modeling. As a result, NLMs are compact and can extend to longer context.

As a natural development, subsequent MT systems (Schwenk, 2007; Vaswani et al., 2013; Luong et al., 2015a), inter alia, started adopting NLMs alongside with traditional n-gram LMs and generally obtain sizable improvements in terms of translation quality. To make NLMs even more powerful, recent work (Schwenk, 2012; Son et al., 2012; Auli et al.,

2013; Devlin et al., 2014) proposes to condition on source words as well as the target context to lower uncertainty in predicting next words (see Figure 2.1).[1] These hybrid MT systems with NLM components, while better than statistical MT systems, still translate locally and fail to capture long-range dependencies. For example, in Figure 2.1, the source-conditioned NLM does not see the word "stroll", or any other words outside of its fixed context windows, which can be useful in deciding that the next word should be "bank" as in "river bank" rather "financial bank".

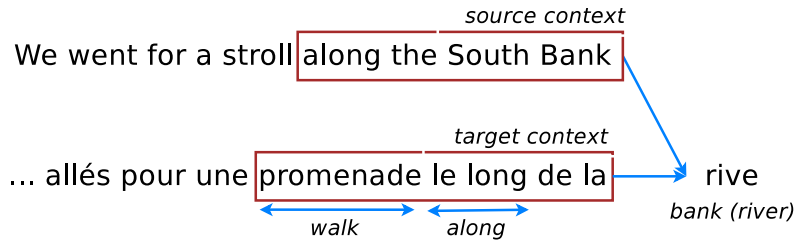

Figure 2.1: **Source-conditioned neural language models** (NLMs) – example of a source-conditioned NLM proposed by Devlin et al. (2014). To evaluate how likely a next word "rive" is, the model not only relies on previous target words (context) "promenade le long de la" as in traditional NLMs (Bengio et al., 2003), but also utilizes source context "along the South Bank" to lower uncertainty in its prediction. Plot a figure on NPLMs?

More problematically, the entire MT pipeline is already complex with different components needing to be tuned separately such as translation models, language models, and reordering models. Now, it becomes even worse as different neural components are incorporated in to the translation framework. This inspires the birth of neural machine translation with a goal of redesigning the entire MT pipeline completely. To start, we will first learn about recurrent neural network, a building block for NMT as well as a key component to address the local translation problem in statistical MT systems.

2.2 Recurrent Neural Network

Recurrent neural network (RNN) (Elman, 1990) is a powerful and expressive architecture

[1] In (Devlin et al., 2014), the authors constructed a model that conditions on 3 target words and 11 source words, effectively building a 15-gram LM.

that can handle sequential data and has been successfully applied to language modeling tasks (Mikolov et al., 2010, 2011; Mikolov and Zweig, 2012). Formally, an RNN takes as input a sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ and processes them one by one. For each new input $\boldsymbol{x}_i$, an RNN updates its memory to produce a hidden state $\boldsymbol{h}_i$ which one can think of as a representation for the partial sequence $\boldsymbol{x}_{\overline{1,i}}$. The key secret sauce is in the recurrence formula of an RNN that defines how its hidden state is updated. At its simplest form, a "vanilla" RNN defines its recurrence function as:

$$\boldsymbol{h}_t = f\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\right) \tag{2.2}$$

In the above formula, f is an abstract function that computes a new hidden state given the current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$. The starting state $\boldsymbol{h}_0$ is often set to $\mathbf{0}$ though it can take any value as we will see later in the context of NMT decoders. A popular choice of f is provided below with σ being a non-linear function such as $\mathrm{sigmoid}$ or $\tanh$.[2]

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}) \tag{2.3}$$

At each timestep t, an RNN can (optionally) emit an output symbol y_t which can either be discrete or real-valued. For the discrete scenario, which is often the case for linguistic applications, a probability distribution $\boldsymbol{p}$ over a set of output classes Y is derived as:[3]

$$\boldsymbol{s}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t \tag{2.4}$$

$$\boldsymbol{p}_t = \mathrm{softmax}(\boldsymbol{s}_t) \tag{2.5}$$

Here, we introduce a new set of weights $\boldsymbol{W}_{hy} \in \mathbb{R}^{|Y| \times d}$, with d being the dimension of the RNN hidden state, to compute a score vector $\boldsymbol{s}_t$, or *logits*, over different individual classes. Often, with a large output set Y, the matrix-vector multiplication in Eq. (2.4) is a major computational bottleneck in RNNs, which results in several challenges for neural language modeling and machine translation that we will address in later chapters. The $\mathrm{softmax}$

[2]There could also be an optional bias term in Eq. (2.3).

[3]For the real-valued case, we refer readers to mixture density models (Bishop, 1994) which have been applied to RNN training, e.g., for hand-writing synthesis (Graves, 2013).

function transforms the score vector $\boldsymbol{s}_t$ into a probability vector $\boldsymbol{p}_t$, which is defined for each specific element $y \in Y$ as below. For convenience, we overload our notations to use $\boldsymbol{p}_t(y)$ and $\boldsymbol{s}_t(y)$ to refer to entries in the vectors $\boldsymbol{p}_t$ and $\boldsymbol{s}_t$ that correspond to y.

$$\boldsymbol{p}_t(y) = \frac{e^{\boldsymbol{s}_t(y)}}{\sum_{y' \in Y} e^{\boldsymbol{s}_t(y')}} \tag{2.6}$$

With the above formulas, we have completely defined the RNN weight set $\boldsymbol{\theta}$ which consists of *input* connections $\boldsymbol{W}_{xh}$, *recurrent* connections $\boldsymbol{W}_{hh}$, and *output* connections $\boldsymbol{W}_{hy}$. These weights are shared across timesteps as illustrated in Figure 2.2. This is, in fact, the beauty of RNNs as they can capture the dynamics of arbitrarily long sequences without having to increase their modeling capacity. In contrast, feedforward networks can only model relationship over fixed-length segments.

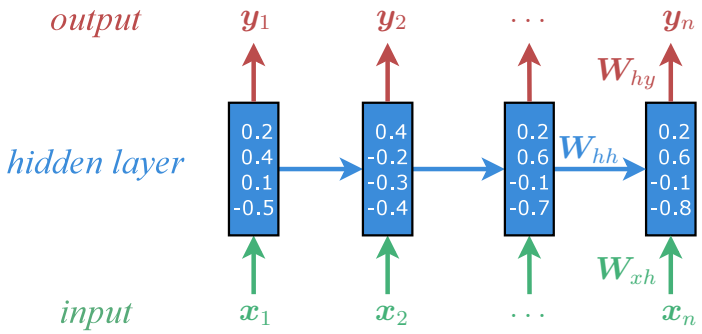

Figure 2.2: **Recurrent neural networks** – example of a recurrent neural network that processes a input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ to build up hidden representations as each input is consumed and produces an output sequence $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n$. The input $\boldsymbol{W_{xh}}$, recurrent $\boldsymbol{W_{hh}}$, and output $\boldsymbol{W_{hy}}$ weights are shared across timesteps.

Throughout this thesis, RNNs will be discussed from a language learning perspective. For more details on general RNNs, we refer readers to the following resources (Sutskever, 2012; Mikolov, 2012; Karpathy, 2015).

Recurrent Language Models As a special case of RNN, recurrent language model assumes that the input and output sequences consist of discrete symbols, often words in a language. Additionally, the input sequence is prepended with a special starting symbol

<s>, e.g., x = { <s>, "I", "am", "a", "student"}. Since the goal of a language model is to predict the next word, the output sequence is a shift-by-1 version of the input and ends with a special symbol </s> that marks the boundary, e.g., y = { "I", "am", "a", "student", </s>}. As we illustrate in Figure 2.3, the word emitted at one timestep is used as an input to the next timestep.

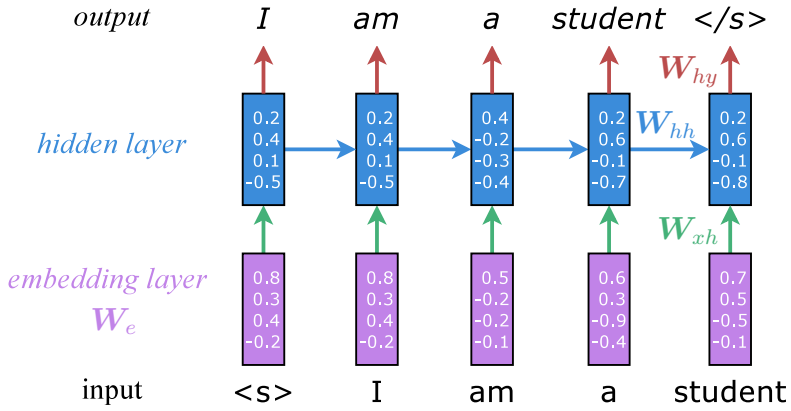

Figure 2.3: **Recurrent language models** – example of a recurrent language model that processes a sentence "I am a student" and predicts next words as it goes. Beside the shared recurrent $\boldsymbol{W}_{hh}$ and feed-forward $\boldsymbol{W}_{xh}$ weights, there is an additional shared embedding weight matrix $\boldsymbol{W}_e$ that needs to be learned as well.

To apply RNNs to sentences in languages, or generally sequences of discrete symbols, one can consider one-hot representations for words, i.e., $\boldsymbol{x}_i \in \mathbb{R}^{|V|}$, with V being the vocabulary considered. However, for a large vocabulary V, such a representation choice is problematic as it results in a large weight matrix $\boldsymbol{W}_{xh}$ and there is no notion of similarity between words. In practice, low-dimensional dense representations for words, or *embeddings*, are often used to address these problems. Specifically, an embedding matrix $\boldsymbol{W}_e \in \mathbb{R}^{d_e \times |V|}$ is looked up for each word x_i to retrieve a representation $\boldsymbol{x}_i \in \mathbb{R}^{d_e}$. As a result, a vanilla recurrent language model will generally have $\theta = \{\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_{hy}, \boldsymbol{W}_e\}$ as its weights.

2.2.1 Training & Backpropagation

Given a training dataset of N discrete output sequences $y^{(1)}, \ldots, y^{(N)}$ with lengths $m_1, \ldots, m_N$ accordingly. The learning objective is to minimize the negative log-likelihood, or the *cross-entropy* loss, of these training examples:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} -\log p\left(y^{(i)}\right) \tag{2.7}$$

$$= \sum_{i=1}^{N} \sum_{t=1}^{m_i} -\log p\left(y_t^{(i)} | y_{<t}^{(i)}\right) \tag{2.8}$$

RNN learning is often done using mini-batch stochastic gradient descent (SGD) algorithms in which a small set of training examples, a *mini-batch*, is used to compute the gradients and update weights one at a time. Using mini-batches has several advantages: (a) the gradients are more reliable and consistent than the "online" setting which updates per example, (b) less computation is required to update the weights unlike the case of full-batch learning which has to process all examples before updating, and (c) with multiple examples in a mini-batch, one can turn matrix-vector multiplications such as those in Eq. (2.3) and Eq. (2.4) into matrix-matrix multiplications which can be deployed efficiently on GPUs. The simplest weight update formula with η as a learning rate is given below:

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta}) \tag{2.9}$$

Here, $\nabla J(\boldsymbol{\theta})$ is the gradient of the loss that we are minimizing with respect to the model weights. Intuitively, what the formula does is to update the weights along the opposite direction of the gradient to minimize the loss objective. The learning rate η, sometimes referred as a *step size*, is a hyperparameter which controls how much we update the weights along the optimization direction.

Mathematical Helpers To simplify the maths for our backpropagation derivations in the next section, we present here a few simple remarks and lemmas on vector calculus and gradient computation.

Remark **1.** *Let $\boldsymbol{u}$, $\boldsymbol{v}$ be any vectors and $\circ$ be element-wise vector multiplication, we have:*

$$diag(\boldsymbol{u}) \cdot \boldsymbol{v} = \boldsymbol{u} \circ \boldsymbol{v} \tag{2.10}$$

Here $diag(\boldsymbol{u})$ refers to a diagonal matrix with its diagonal elements being $\boldsymbol{u}$.

Lemma **1.** *Let l be a loss value for which we already know how to compute its gradient $d\boldsymbol{v}$ with respect to a vector $\boldsymbol{v}$. Given that $\boldsymbol{v} = f(\boldsymbol{W}\boldsymbol{h})$, the gradients $d\boldsymbol{h}, d\boldsymbol{W}$ of the loss l with respect to the vector $\boldsymbol{h}$ and the matrix $\boldsymbol{W}$ can be derived as follows:*

$$d\boldsymbol{h} = \boldsymbol{W}^\top \cdot (f'(\boldsymbol{W}\boldsymbol{h}) \circ d\boldsymbol{v}) \tag{2.11}$$

$$d\boldsymbol{W} = (f'(\boldsymbol{W}\boldsymbol{h}) \circ d\boldsymbol{v}) \cdot \boldsymbol{h}^\top \tag{2.12}$$

Proof. Let $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{h}$, we have the following derivations:

$$\begin{aligned}
d\boldsymbol{h} &= \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{h}} \cdot \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{z}} \cdot d\boldsymbol{v} && \text{[Vector calculus chain rules]} \\
&= \frac{\partial \boldsymbol{W}\boldsymbol{h}}{\partial \boldsymbol{h}} \cdot \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} \cdot d\boldsymbol{v} \\
&= \boldsymbol{W}^\top \cdot \text{diag}\left(f'(\boldsymbol{z})\right) \cdot d\boldsymbol{v} \\
&= \boldsymbol{W}^\top \cdot (f'(\boldsymbol{W}\boldsymbol{h}) \circ d\boldsymbol{v}) && \text{[Remark 1]}
\end{aligned}$$

Let $\boldsymbol{w}_i^\top$ be the i^{th} row vector of matrix $\boldsymbol{W}$ and v_i, z_i be the i^{th} elements of vectors $\boldsymbol{v}, \boldsymbol{z}$. Also denoting $d\boldsymbol{w}_i, dv_i$ to be the gradients of l with respect to $\boldsymbol{w}_i, v_i$, we have:

$$\begin{aligned}
d\boldsymbol{w}_i &= \frac{\partial z_i}{\partial \boldsymbol{w}_i} \cdot \frac{\partial v_i}{\partial z_i} \cdot dv_i && \text{[Vector calculus chain rules]} \\
&= \frac{\partial \boldsymbol{w}_i^\top \boldsymbol{h}}{\partial \boldsymbol{w}_i} \cdot f'(z_i) \cdot dv_i \\
&= \boldsymbol{h} \cdot f'(z_i) \cdot dv_i \\
d\boldsymbol{w}_i^\top &= (f'(z_i) \cdot dv_i) \cdot \boldsymbol{h}^\top && \text{[Tranposing]} \\
d\boldsymbol{W} &= (f'(\boldsymbol{W}\boldsymbol{h}) \circ d\boldsymbol{v}) \cdot \boldsymbol{h}^\top && \text{[Concatenating row derivatives]}
\end{aligned}$$

□

Corollary **1.** *As a special case of Lemma 1, when f is an identity function, i.e., $\boldsymbol{v} = \boldsymbol{W}\boldsymbol{h}$, we have:*

$$d\boldsymbol{h} = \boldsymbol{W}^\top \cdot d\boldsymbol{v} \tag{2.13}$$

$$d\boldsymbol{W} = d\boldsymbol{v} \cdot \boldsymbol{h}^\top \tag{2.14}$$

Remark **2.** *Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{s}$ be any vectors such that $\boldsymbol{s} = \boldsymbol{u} \circ f(\boldsymbol{v})$. Also, let $d\boldsymbol{u}, d\boldsymbol{v}, d\boldsymbol{s}$ be the gradients of a loss l with respect to the corresponding vectors. We have:*

$$d\boldsymbol{u} = f(\boldsymbol{v}) \circ d\boldsymbol{s} \tag{2.15}$$

$$d\boldsymbol{v} = f'(\boldsymbol{v}) \circ \boldsymbol{u} \circ d\boldsymbol{s} \tag{2.16}$$

Remark **3.** *As a special case of Remark 2 when f is an identity function, i.e., $\boldsymbol{s} = \boldsymbol{u} \circ \boldsymbol{v}$. We have:*

$$d\boldsymbol{u} = \boldsymbol{v} \circ d\boldsymbol{s} \tag{2.17}$$

$$d\boldsymbol{v} = \boldsymbol{u} \circ d\boldsymbol{s} \tag{2.18}$$

Single-Time Backpropagation To compute the gradients for the loss $J(\boldsymbol{\theta})$, we first need to be able to derive the gradients of the per-timestep loss $l_t = -\log \boldsymbol{p}_t(y_t)$ with respect to both the RNN weights $\{\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_{hy}\}$ and the inputs $\{\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\}$. It is worth noting that $\boldsymbol{x}_t$ is a column vector in the embedding matrix $\boldsymbol{W}_e$. We denote these gradients as $\{d\boldsymbol{W}_{xh}, d\boldsymbol{W}_{hh}, d\boldsymbol{W}_{hy}, d\boldsymbol{x}_t, d\boldsymbol{h}_{t-1}\}$ respectively and define intermediate gradients $d\boldsymbol{s}_t, d\boldsymbol{h}_t$ similarly with $\boldsymbol{s}_t$ and $\boldsymbol{h}_t$ being used in Eq. (2.4) and Eq. (2.5). Starting with the loss l_t, we employ backpropagation through structures (Goller and Kchler, 1996) to derive each gradient one by one in the following order: $l_t \rightarrow \boldsymbol{s}_t \rightarrow \{\boldsymbol{h}_t, \boldsymbol{W}_{hy}\} \rightarrow \{\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}\}$.

First, from Eq. (5.2), with $\boldsymbol{p}_t(y) = \dfrac{e^{\boldsymbol{s}_t(y)}}{\sum_{y' \in Y} e^{\boldsymbol{s}_t(y')}}$, we have:

$$d\boldsymbol{s}_t = \frac{\partial l_t}{\partial \boldsymbol{s}_t} = \frac{\partial}{\partial \boldsymbol{s}_t}\left(\log \sum_{y'} e^{\boldsymbol{s}_t(y')} - \boldsymbol{s}_t(y_t)\right) \tag{2.19}$$

Computing per-coordinate gradient $\boldsymbol{s}_t(y)$ gives:

$$\frac{\partial}{\partial \boldsymbol{s}_t(y)} \left(\log \sum_{y'} e^{\boldsymbol{s}_t(y')} - \boldsymbol{s}_t(y_t) \right) = \begin{cases} \boldsymbol{p}_t(y_t) - 1 & y = y_t \\ \boldsymbol{p}_t(y) & y \neq y_t \end{cases} \tag{2.20}$$

The above gradients can be concisely written in vector form as:

$$d\boldsymbol{s}_t = \boldsymbol{p}_t - \mathbf{1}_{y_t} \tag{2.21}$$

Here, $\boldsymbol{p}_t$ is the probability distribution defined in Eq. (2.5) and has been calculated in the forward pass, so we simply reuse it. $\mathbf{1}_{y_t}$ is a one-hot vector with 1 at position y_t. Applying Corollary 1, noting that $\boldsymbol{s}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t$ in Eq. (2.4), we arrive at:

$$d\boldsymbol{h}_t = \boldsymbol{W}_{hy}^\top \cdot d\boldsymbol{s}_t \tag{2.22}$$

$$d\boldsymbol{W}_{hy} = d\boldsymbol{s}_t \cdot \boldsymbol{h}_t^\top \tag{2.23}$$

At this point, we have derived part of the backpropation procedure which can be applied to any hidden unit type, e.g., the aforementioned vanilla RNN or the LSTM unit that we will describe shortly in the next section.

Vanilla RNN Backpropagation First of all, we can simplify the notation to have $\boldsymbol{T}_{\text{rnn}} = [\boldsymbol{W}_{\boldsymbol{xh}} \boldsymbol{W}_{\boldsymbol{hh}}]$ and $\boldsymbol{z}_t = [\boldsymbol{x}_t; \boldsymbol{h}_{t-1}]$, so the RNN formulation in Eq. (2.3) becomes:

$$\boldsymbol{h_t} = \sigma\left(\boldsymbol{T}_{\text{rnn}} \boldsymbol{z}_t\right) \tag{2.24}$$

Applying Lemma 1, we have:

$$d\boldsymbol{z}_t = \boldsymbol{T}_{\text{rnn}}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}} \boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \tag{2.25}$$

$$d\boldsymbol{T}_{\text{rnn}} = (\sigma'(\boldsymbol{T}_{\text{rnn}} \boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{z}_t^\top \tag{2.26}$$

This is one of the *tricks* that we use to better utilize GPUs by creating larger matrices and vectors, i.e., $\boldsymbol{T}_{\text{rnn}}$ and $\boldsymbol{z}_t$. From Eq. (2.25) and Eq. (2.26), one can easily extract the following gradients: (a) $d\boldsymbol{x}_t$ – embedding gradients which we use to sparsely update the

embedding weights $\boldsymbol{W}_e$, (b) $d\boldsymbol{h}_{t-1}$ – gradients of the previous hidden state, which is needed by the backpropagation-through-time algorithm that we will discuss next, and (c) $d\boldsymbol{W}_{xh}$ as well as $d\boldsymbol{W}_{hh}$ – the RNN input and recurrent connections.[4]

Backpropagation Through Time (BPTT) Having defined a single-timestep backpropagation procedure, we are now ready to go through the BPTT algorithm (Rumelhart and McClelland, 1986; Werbos, 1990). Inspired by Sutskever (2012), we summarize the BPTT algorithm for RNNs below with the following remarks: (a) Lines 3, 5, 6, 7 accumulate the gradients of RNN weights $\{\boldsymbol{W}_{hy}, \boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_e\}$ over time; (b) In line 7, $d\boldsymbol{x}_t$ refers to gradients of words participating in the current mini-batch which we use to sparsely update $\boldsymbol{W}_e$;[5] and (c) Line 4 accumulates gradients for the current hidden state $\boldsymbol{h}_t$ by considering two paths, a "vertical" one from the current loss at time t and a "recurrent" one from the timestep $t+1$ which was set in Line 8 earlier.

Algorithm 1: BPTT algorithm for "vanilla" RNNs

1 **for** $t = T \rightarrow 1$ **do**
  `// Output backprop`
2 $d\boldsymbol{s}_t \leftarrow \mathbf{1}_{y_t} - \boldsymbol{p}_t$
3 $d\boldsymbol{W}_{hy} \leftarrow d\boldsymbol{W}_{hy} + d\boldsymbol{s}_t \cdot \boldsymbol{h}_t^\top$
4 $d\boldsymbol{h}_t \leftarrow d\boldsymbol{h}_t + \boldsymbol{W}_{hy}^\top \cdot d\boldsymbol{s}_t$
  `// RNN backprop`
5 $d\boldsymbol{W}_{xh} \leftarrow d\boldsymbol{W}_{xh} + (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{x}_t^\top$
6 $d\boldsymbol{W}_{hh} \leftarrow d\boldsymbol{W}_{hh} + (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{h}_{t-1}^\top$
  `// Input backprop`
7 $d\boldsymbol{x}_t \leftarrow \boldsymbol{W}_{xh}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t)$
8 $d\boldsymbol{h}_{t-1} \leftarrow \boldsymbol{W}_{hh}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t)$
9 **end**

[4] One can also separately derive these gradients as follows:

$$d\boldsymbol{x}_t = \boldsymbol{W}_{xh}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \tag{2.27}$$
$$d\boldsymbol{h}_{t-1} = \boldsymbol{W}_{hh}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \tag{2.28}$$
$$d\boldsymbol{W}_{xh} = (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{x}_t^\top \tag{2.29}$$
$$d\boldsymbol{W}_{hh} = (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{h}_{t-1}^\top \tag{2.30}$$

[5] In multi-layer RNNs, $d\boldsymbol{x}_t$ is used to send gradients down to the below layers.

2.2.2 Long Short-Term Memory

Even though computing RNN gradients is straightforward once the BPTT algorithm has been plotted out, training is inherently difficult due to the nonlinear iterative nature of RNNs. Among all reasons, the two classic problems of RNNs that often arise when dealing with very long sequences are the *exploding* and *vanishing* gradients as described by Bengio et al. (1994). In short, exploding gradients refers to the phenomenon that the gradients become exponentially large as we backpropagate over time, making learning unstable. Vanishing gradients, on the other hand, is the opposite problem when the gradients go exponentially fast towards zero, turning BPTT into truncated BPTT that is unable to capture long-range dependencies in sequences.

Let us try to explain the aforementioned problems informally and refer readers to more rigorous and in-depth analyses in (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Martens and Sutskever, 2011; Pascanu et al., 2013). The main cause of these two problems all lies in Line 8 of the BPTT algorithm which can be rewritten as $d\boldsymbol{h}_{t-1} = \boldsymbol{W}_{hh}^{\top} \cdot \text{diag}\left(\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t)\right) \cdot d\boldsymbol{h}_t$ (see Remark 1). We can try to understand the behavior of RNNs over time by assuming for a moment that there is no contribution from intermediate losses, i.e., Line 4 is "ignored". Given such an assumption, a signal backpropagated from the current hidden state over K steps will become $d\boldsymbol{h}_{t-K} = \prod_{i=1}^{K} \left(\boldsymbol{W}_{hh}^{\top} \cdot \text{diag}\left(\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_{t-i+1})\right)\right) \cdot d\boldsymbol{h}_t$. Assuming that the non-linear function σ is bounded, e.g., sigm and $\tanh$, and behaves "nicely", what we need to deal with now is the multiplication of the recurrent matrix over time. This leads to the fact that the behavior of RNNs is often governed by the characteristics of the recurrent matrix $\boldsymbol{W}_{hh}$ and most analyses examine it in terms of the largest eigenvalue of $\boldsymbol{W}_{hh}$ as well as the norms of these signals. Roughly speaking, if the largest eigenvalue is large enough, exploding gradients will be likely to happen. On the contrary, if the largest eigenvalue is below a certain threshold, vanishing gradients will occur, as clearly explained by Pascanu et al. (2013).

Gradient Clipping In practice, it is generally easy to cope with the exploding gradient problem by applying different forms of gradient clipping. The first approach was proposed by Mikolov (2012) through the form of temporal *element-wise* clipping. At each timestep during backpropagation, any elements of $d\boldsymbol{h}$ that are greater than a positive threshold τ or

smaller than -τ will be set to τ or -τ respectively. One can also perform gradient *norm* clipping as suggested by Pascanu et al. (2013). The idea is simple: given a final gradient vector $\boldsymbol{g}$ computed per mini-batch, if its norm $||\boldsymbol{g}||$ is greater than a threshold τ, then we will use the following scaled gradient $\frac{\tau}{||\boldsymbol{g}||}\boldsymbol{g}$ instead. The latter approach is widely used in many systems nowadays and can also be used in conjunction with the former. We take the combined approach in our implementations described later in this thesis.

Long Short-Term Memory The vanishing gradient problem, on the other hand, is more challenging to tackle. There have been many proposed approaches to alleviate the problem such as skip connections (Waibel et al., 1990; Lin et al., 1996), hierarchical architectures (El Hihi and Bengio, 1996), leaky integrators (Jaeger et al., 2007), second-order methods (Martens and Sutskever, 2011), and regularization (Pascanu et al., 2013), to name a few; also, see (Bengio et al., 2013) for a comparison of some of these techniques. Among all, Long Short-term Memory (LSTM), invented by Hochreiter and Schmidhuber (1997) and later refined by Gers et al. (2000), appears to be one of the most widely adopted solutions to the vanishing gradient problem. Graves and colleagues deserve credit for popularizing LSTM through a series of work (Graves and Schmidhuber, 2005, 2009; Graves, 2013). The key idea of LSTM is to augment RNNs with linear *memory* units that allow the gradient to flow smoothly through time. In addition, there are gating units that control how much an RNN wants to reuse memory (*forget* gates), receive input signal (*input* gates), and extract information (*output* gates) at each timestep. There are many implementation instances of LSTM, differing in terms of whether and which biases are used, how gates are built, etc; however, it turns out that these different choices do not matter much for most cases (Józefowicz et al., 2015; Greff et al., 2015). As such, in this section and throughout this thesis, we will stick to the formulation described in (Zaremba et al., 2014).

Instead of jumping directly into the detailed formulation, let us provide intuitions on how to gradually build up an LSTM architecture. First, we can construct a simple memory unit as follows:

$$\boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \sigma\left(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}\right) \tag{2.31}$$

$$\boldsymbol{h}_t = \boldsymbol{c}_t \tag{2.32}$$

This architecture can be viewed as a form of "leaky" integration mentioned in (Sutskever, 2012; Bengio et al., 2013) since it is equivalent to $\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \sigma(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$. Training this network over long sequences is easy since among the exponentially many backpropagation paths, there is exactly one path that goes through all the memory units $\boldsymbol{c}_i$ ($i = \overline{1, T}$) and is guaranteed to not vanish since $d\boldsymbol{c}_t = d\boldsymbol{c}_{t-1}$ along that path.

Such an architecture, however, does not account for the fact that certain inputs, e.g., function words or punctuations, are, sometimes, not relevant to the task at hand and should be downweighted. Occasionally, we might also want to reset the memory, e.g., at the beginning of each sentence in a paragraph. To add more flexibility and power to this architecture, the LSTM adds forget, input, and output gates as follows:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \sigma\left(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}\right) \tag{2.33}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \sigma\left(\boldsymbol{c}_t\right)) \tag{2.34}$$

We note that, in Eq. (2.34), the memory cell $\boldsymbol{c}_t$ is passed through a nonlinear function σ before the output gate $\boldsymbol{o}_t$ is used to extract relevant information in the hope for better information retrieval. As evidence, Greff et al. (2015) have shown that such an output nonlinearity is critical to the performance of an LSTM. Moving on, to ensure that the gates are adaptive, we build them from the information given by the current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$. We also want the gates to be in $[0, 1]$, so sigm will be used (here sigm refers to the *sigmoid* function defined as $f(x) = \frac{1}{1+e^{-x}}$). All of these desiderata lead to the below LSTM formulation described in (Zaremba et al., 2014) in which σ is chosen to be $\tanh$:

$$\begin{pmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \hat{\boldsymbol{h}}_{\boldsymbol{t}} \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \tanh \end{pmatrix} \begin{bmatrix} \boldsymbol{W}_{xi}\boldsymbol{W}_{hi} \\ \boldsymbol{W}_{xf}\boldsymbol{W}_{hf} \\ \boldsymbol{W}_{xo}\boldsymbol{W}_{ho} \\ \boldsymbol{W}_{xh}\boldsymbol{W}_{hh} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} \tag{2.35}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \hat{\boldsymbol{h}}_{\boldsymbol{t}} \tag{2.36}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) \tag{2.37}$$

Following the same spirit as Eq. (2.24), we can be GPU-efficient with Eq. (2.35) since the 8 different submatrices are grouped into a single big matrix, which we call $\boldsymbol{T}_{\text{lstm}}$. Let $\boldsymbol{z}_t = [\boldsymbol{x}_t; \boldsymbol{h}_{t-1}]$. What we do is first multiply $\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t$ and then apply different non-linear functions to corresponding parts of the output. For the ease of deriving backpropagation equations later, we can rewrite Eq. (2.35) as:

$$\boldsymbol{u}_t = g(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \tag{2.38}$$

$$= g(\boldsymbol{T}_{\text{x}}\boldsymbol{x}_t + \boldsymbol{T}_{\text{h}}\boldsymbol{h}_{t-1}) \tag{2.39}$$

Here, g is a non-linear function applied element-wise and we define g loosely in the sense that it uses $\tanh$ only for the vector part corresponding to $\hat{\boldsymbol{h}}_{\boldsymbol{t}}$ and sigm for the rest.

LSTM Training In the LSTM training pipeline, there are many components that are exactly the same or very similar to RNN training. We will now highlight some key differences. First of all, LSTM extends the recurrence function to have not just the hidden states but also the memory cells as both inputs and outputs. The definition is as below:

$$(\boldsymbol{h}_t, \boldsymbol{c}_t) = f\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}\right) \tag{2.40}$$

In our case, the abstract function f is implemented by Eq. 2.35-2.37. Once $\boldsymbol{h}_t$ is computed, the prediction process is the same as that of RNNs which is given by Eq. 2.4-5.2. The training objective in Eq. (2.8) remains unchanged as well.

LSTM Backpropagation Since the prediction procedure is the same, LSTM backpropagation pipeline mimics that of RNNs up to Eq. (2.22) and Eq. (2.23), which computes $d\boldsymbol{h}_t$ and $d\boldsymbol{W}_{hy}$ respectively.

Given $d\boldsymbol{h}_t$, we now work backward to derive other gradients. First, starting from Eq. (2.37) and by applying Remark 2, we have:

$$d\boldsymbol{o}_t = \tanh(\boldsymbol{c}_t) \circ d\boldsymbol{h}_t \tag{2.41}$$

$$d\boldsymbol{c}_t = \tanh'(\boldsymbol{c}_t) \circ \boldsymbol{o}_t \circ d\boldsymbol{h}_t \tag{2.42}$$

Before backpropagating Eq. (2.36), once must *remember* to update $d\boldsymbol{c}_t$ with the gradient sent back from $\boldsymbol{c}_{t+1}$, which is accomplished by Lines 6 and 10 of Algorithm 2. Given the updated $d\boldsymbol{c}_t$, we apply Remark 3 to derive:

$$d\boldsymbol{f}_t = \boldsymbol{c}_{t-1} \circ d\boldsymbol{c}_t \tag{2.43}$$

$$d\boldsymbol{c}_{t-1} = \boldsymbol{f}_t \circ d\boldsymbol{c}_t \tag{2.44}$$

$$d\boldsymbol{i}_t = \hat{\boldsymbol{h}}_{\boldsymbol{t}} \circ d\boldsymbol{c}_t \tag{2.45}$$

$$d\hat{\boldsymbol{h}}_{\boldsymbol{t}} = \boldsymbol{i}_t \circ d\boldsymbol{c}_t \tag{2.46}$$

Let $d\boldsymbol{u}_t = [d\boldsymbol{i}_t; d\boldsymbol{f}_t; d\boldsymbol{o}_t; d\hat{\boldsymbol{h}}_{\boldsymbol{t}}]$ (vertical concatenation), we are now ready to backpropagate through Eq. (2.39). In a similar manner as RNNs, Eq. 2.27-2.30, we arrive at:

$$d\boldsymbol{x}_t = \boldsymbol{T}_{\text{x}}^{\top} \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \tag{2.47}$$

$$d\boldsymbol{h}_{t-1} = \boldsymbol{T}_{\text{h}}^{\top} \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \tag{2.48}$$

$$d\boldsymbol{T}_{\text{x}} = (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{x}_t^{\top} \tag{2.49}$$

$$d\boldsymbol{T}_{\text{h}} = (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{h}_{t-1}^{\top} \tag{2.50}$$

All of these gradients can now be put together in the below BPTT algorithm for LSTM:

2.3 Neural Machine Translation

Having introduced recurrent language models, one can simply think of neural machine translation (NMT) as a recurrent language model that conditions on the source sentence. More formally, NMT aims to directly model the conditional probability $p(y|x)$ of translating a source sentence, $x_1, \ldots, x_n$, to a target sentence, $y_1, \ldots, y_m$. It accomplishes this goal through an *encoder-decoder* framework (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). The *encoder* computes a representation $\boldsymbol{s}$ for each source sentence. Based on that source representation, the *decoder* generates a translation, one target word at

Algorithm 2: BPTT algorithm for LSTM

for $t = T \rightarrow 1$ **do**

// Output backprop

$d\boldsymbol{s}_t \leftarrow \mathbf{1}_{y_t} - \boldsymbol{p}_t$

$d\boldsymbol{W}_{hy} \leftarrow d\boldsymbol{W}_{hy} + d\boldsymbol{s}_t \cdot \boldsymbol{h}_t^\top$

$d\boldsymbol{h}_t \leftarrow d\boldsymbol{h}_t + \boldsymbol{W}_{hy}^\top \cdot d\boldsymbol{s}_t$

// LSTM backprop

$d\boldsymbol{o}_t \leftarrow \tanh(\boldsymbol{c}_t) \circ d\boldsymbol{h}_t$

$d\boldsymbol{c}_t \leftarrow d\boldsymbol{c}_t + \tanh'(\boldsymbol{c_t}) \circ \boldsymbol{o_t} \circ d\boldsymbol{h}_t$; // Already included $d\boldsymbol{c}_{t+1}$

$d\boldsymbol{f}_t \leftarrow \boldsymbol{c}_{t-1} \circ d\boldsymbol{c}_t$

$d\boldsymbol{i}_t \leftarrow \hat{\boldsymbol{h}}_{\boldsymbol{t}} \circ d\boldsymbol{c}_t$

$d\hat{\boldsymbol{h}}_{\boldsymbol{t}} \leftarrow \boldsymbol{i}_t \circ d\boldsymbol{c}_t$

$d\boldsymbol{c}_{t-1} \leftarrow \boldsymbol{f}_t \circ d\boldsymbol{c}_t$; // Compute $d\boldsymbol{c}_{t-1}$

$d\boldsymbol{u}_t = [d\boldsymbol{i}_t; d\boldsymbol{f}_t; d\boldsymbol{o}_t; d\hat{\boldsymbol{h}}_{\boldsymbol{t}}]$

$d\boldsymbol{T}_{\text{x}} \leftarrow (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{x}_t^\top$

$d\boldsymbol{T}_{\text{h}} \leftarrow (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{h}_{t-1}^\top$

// Input backprop

$d\boldsymbol{x}_t \leftarrow \boldsymbol{T}_{\text{x}}^\top \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t)$

$d\boldsymbol{h}_{t-1} \leftarrow \boldsymbol{T}_{\text{h}}^\top \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t)$

end

a time, and hence, decomposes the log conditional probability as:

$$\log p(y|x) = \sum_{t=1}^{m} \log p\left(y_t|y_{<t}, \boldsymbol{s}\right) \tag{2.51}$$

NMT models vary in terms of the exact architectures to use. A natural choice for sequential data is the recurrent neural network (RNN), used by most of the recent NMT work and for both the encoder and decoder. The used RNN models, however, differ in terms of: (a) *directionality* – unidirectional or bidirectional; (b) *depth* – single or deep multi-layer; and (c) *type* – often either a vanilla RNN, an LSTM (Hochreiter and Schmidhuber, 1997), or a gated recurrent unit (GRU) (Cho et al., 2014). In general, for the encoder, almost any architecture can be used since we have fully observed the source sentence. For example, Kalchbrenner and Blunsom (2013) used a convolutional neural network for encoding the source. Choices on the decoder side are more limited since we need to be able to generate a translation. At the time of this thesis, the most popular choice is a unidirectional RNN, which simplifies the beam-search decoding algorithm by producing translations from left to right.

In this thesis, all our NMT models are deep multi-layer RNNs which are unidirectional and have an LSTM as the recurrent unit. We show an example of such model in Figure 2.4. In this example, we train our model to translate a source sentence "I am a student" into a target one "Je suis étudiant". At a high level, our NMT models consist of two recurrent neural networks as described in Section 2.2: the *encoder* RNN simply consumes the input source words without making any prediction; the *decoder*, on the other hand, processes the target sentence while predicting the next words.

In more detail, at the bottom layer, the encoder and decoder RNNs receive as *input* the following: first, the source sentence, then a boundary marker "_" which indicates the transition from the encoding to the decoding mode, and the target sentence. Given these discrete words, the model looks up the source and target embeddings to retrieve the corresponding word representations. For this *embedding layer* to work, a vocabulary is chosen for each language, and often the top V frequent words are selected. These embedding weights, one set per language, are learned during training. While one can choose to initialize embedding weights with pretrained word representations, such as word2vec (Mikolov et al., 2013) and

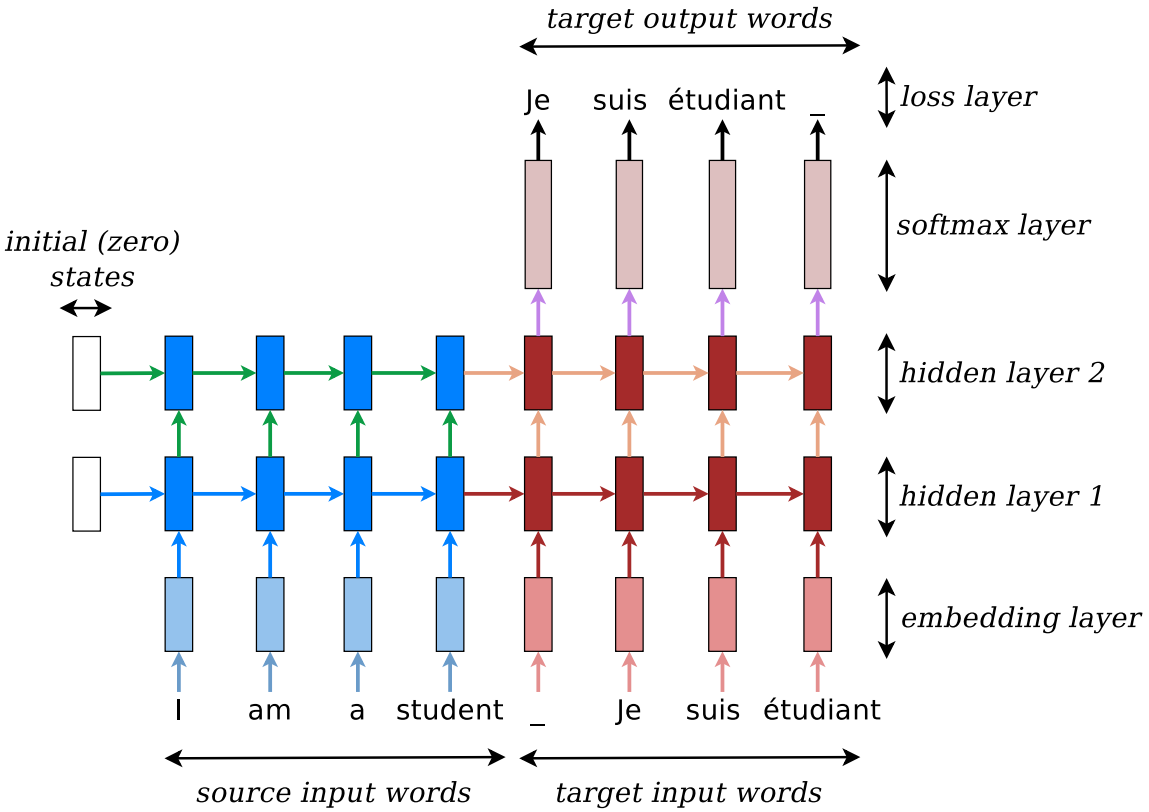

Figure 2.4: **Neural machine translation** – example of a deep recurrent architecture proposed by Sutskever et al. (2014) for translating a source sentence "I am a student" into a target sentence "Je suis étudiant". Here, "_" marks the end of a sentence.

Glove (Pennington et al., 2014), we found, in this thesis, that these embeddings can be initialized randomly and learned from scratch given large training datasets.

Once retrieved, the word embeddings are then fed as input into the main network, which consists of two multi-layer RNNs 'stuck together' — an encoder for the source language and a decoder for the target language. These two RNNs, in principle, can share the same weights; however, in practice, we found that having two different RNN parameters works better and less overfits to large training datasets. The encoder RNN uses zero vectors as its starting states. The decoder, on the other hand, needs to have access to the source information, so one simple way to achieve that is to initialize it with the last hidden state of the encoder.[6] In Figure 2.4, we pass the hidden state at the source word "student" to the

[6]This is not the only way to initialize the decoder, e.g., Cho et al. (2014) connect the last encoder state to

decoder side. The *feed-forward* (vertical) weights connect the hidden unit from the layer below to the upper one; whereas, the *recurrent* (horizontal) weights transfer the history knowlege from the previous timestep to the next one. Often, we use different weights across the encoder and decoder as well as across different layers; in the current example, we have 4 different LSTM weight sets $\boldsymbol{T}_{\text{lstm}}$, detailed in Eq. (2.38), over $\{\text{encoder, decoder}\} \times \{1^{\text{st}}, 2^{\text{nd}} \text{ layer}\}$. Finally, for each target word, the hidden state at the top layer is transformed by the *softmax* weights into a probability distribution over the target vocabulary of size V according to Eq. (2.4) and Eq. (2.5).

2.3.1 Training

Training a neural machine translation system is similar to training a recurrent language model that we have discussed in Section 2.2 except that we need to handle the conditioning on source sentences. The training objective for NMT is formulated as:

$$J = \sum_{(x,y)\in\mathbb{D}} -\log p(y|x) \tag{2.52}$$

Here, $\mathbb{D}$ refers to our parallel training corpus of source and target sentence pairs (x, y). Given the aforementioned NMT architecture, computing the NMT loss for (x, y) during the *forward* pass is almost the same as how we compute the regular RNN loss on just y. The only difference is that we have to first compute representations for the source sentence x to initialize the decoder RNN instead of just starting from zero states. For the *backpropagation* phase, computing gradients for the decoder is the same as what we have described in Algorithm 2 for regular RNNs. The last hidden-state gradient from the decoder is passed back to the encoder. We then continue backpropagating through the encoder in a similar fashion as that of the decoder but without any prediction losses.

More concretely, we present in Algorithm 3 details of the forward pass of an NMT model which uses a deep multi-layer LSTM architecture. Since the encoder and decoder share many operations in common, we combine the source sentence x (length m_x), the target sentence y (length m_y), and the end-of-sentence markers "_" together to form an input sequence s as shown in Line 1. We first start with the encoder weights and initial

every timestep in the decoder as an extra input.

states set to zero (lines 2-3). The algorithm switches to the decoder mode at time $m_x + 1$ (line 5). The same LSTM codebase (lines 8-11) is used for both the encoder and decoder in which embeddings are first looked up for the input s_t; after that, hidden states as well as LSTM cell memories are built from the bottom layer to the top one (the L^{th} layer). In Line 10, `LSTM` refers to the entire formulation in Eq 2.35-2.37, which one can easily replace with other hidden units such as RNN and GRU. Lastly, on the decoder side, the top hidden state is used to predict the next symbol s_{t+1} (line 13); then, a loss value l_t and a probability distribution $\boldsymbol{p}_t$ computed according to Eq 2.4-2.5 are returned.

Algorithm 3: NMT training algorithm – *forward* pass.

Input: source sentence x of length m_x, target sentence y of length m_y.
Parameters: encoder $\boldsymbol{W}_e^{\text{encoder}}$, $\boldsymbol{T}_{\text{lstm}}^{\text{encoder}}$; decoder $\boldsymbol{W}_e^{\text{decoder}}$, $\boldsymbol{T}_{\text{lstm}}^{\text{decoder}}$.
Output: loss $\boldsymbol{l}$ and other intermediate variables for backpropagation.

1 $s \leftarrow [x, \_, y, \_]$; `// Length of` s `is` $m_x + 1 + m_y + 1$
2 $\boldsymbol{W}_e, \boldsymbol{T}_{\text{lstm}}^{(1..L)} \leftarrow \boldsymbol{W}_e^{\text{encoder}}, \boldsymbol{T}_{\text{lstm}}^{\text{encoder}}$; `// Encoder weights`
3 $\boldsymbol{h}_0^{(1..L)}, \boldsymbol{c}_0^{(1..L)} \leftarrow \boldsymbol{0}$; `// Zero init`
4 **for** $t = 1 \rightarrow (m_x + 1 + m_y)$ **do**
  `// Decoder transition`
5 **if** $t == (m_x + 1)$ **then**
6 $\boldsymbol{W}_e, \boldsymbol{T}_{\text{lstm}}^{(1..L)} \leftarrow \boldsymbol{W}_e^{\text{decoder}}, \boldsymbol{T}_{\text{lstm}}^{\text{decoder}}$;
7 **end**
  `// Multi-layer LSTM`
8 $\boldsymbol{h}_t^{(0)} \leftarrow \texttt{Emb\_LookUp}(s_t, \boldsymbol{W}_e)$;
9 **for** $l = 1 \rightarrow L$ **do**
10 $\boldsymbol{h}_t^{(l)}, \boldsymbol{c}_t^{(l)} \leftarrow \texttt{LSTM}\left(\boldsymbol{h}_{t-1}^{(l)}, \boldsymbol{c}_{t-1}^{(l)}, \boldsymbol{h}_t^{(l-1)}, \boldsymbol{T}_{\text{lstm}}^{(l)}\right)$; `// LSTM hidden unit`
11 **end**
  `// Target-side prediction`
12 **if** $t \geq (m_x + 1)$ **then**
13 $l_t, \boldsymbol{p}_t \leftarrow \texttt{Predict}(s_{t+1}, \boldsymbol{h}_t^{(L)}, \boldsymbol{W}_{hy})$;
14 **end**
15 **end**

Next, we describe details of the backpropagation step in Algorithm 4. A quick glance through the algorithm reveals many similarities compared to the forward pass algorithm except that we have reversed the procedue. First, we initialize gradients of the recurrent layers

Algorithm 4: NMT training algorithm – *backpropagation* pass.

$d\boldsymbol{h}^{(1..L)}_{m_x+1+m_y}, d\boldsymbol{c}^{(1..L)}_{m_x+1+m_y} \leftarrow \mathbf{0}$; // Cell and state gradients

$d\boldsymbol{T}^{(1..L)}_{\text{lstm}}, d\boldsymbol{W}_e, d\boldsymbol{W}_{hy} \leftarrow \mathbf{0}$; // Model weight gradients

for $t = (m_x + 1 + m_y) \rightarrow 1$ **do**

  // Encoder transition

  if $t == m_x$ **then**

    $d\boldsymbol{W}^{\text{decoder}}_e, d\boldsymbol{T}^{\text{decoder}}_{\text{lstm}} \leftarrow d\boldsymbol{W}_e, d\boldsymbol{T}^{(1..L)}_{\text{lstm}}$; // Save decoder gradients

    $d\boldsymbol{T}^{(1..L)}_{\text{lstm}}, d\boldsymbol{W}_e \leftarrow \mathbf{0}$;

  end

  // Target-side prediction

  if $t \geq (m_x + 1)$ **then**

    $d\boldsymbol{h}, d\boldsymbol{W} \leftarrow \texttt{Predict\_grad}(s_{t+1}, \boldsymbol{p}_t, \boldsymbol{h}^{(L)}_t)$;

    $d\boldsymbol{h}^{(L)}_t \leftarrow d\boldsymbol{h}^{(L)}_t + d\boldsymbol{h}$; // Vertical gradients

    $d\boldsymbol{W}_{hy} \leftarrow d\boldsymbol{W}_{hy} + d\boldsymbol{W}$;

  end

  // Multi-layer LSTM

  for $l = L \rightarrow 1$ **do**

    // Recurrent gradients

    $d\boldsymbol{h}^{(l)}_{t-1}, d\boldsymbol{c}^{(l)}_{t-1}, d\boldsymbol{x}, d\boldsymbol{T} \leftarrow \texttt{LSTM\_grad}\left(d\boldsymbol{h}^{(l)}_t, d\boldsymbol{c}^{(l)}_t, \boldsymbol{h}^{(l)}_{t-1}, \boldsymbol{c}^{(l)}_{t-1}, \boldsymbol{h}^{(l-1)}_t\right)$;

    $d\boldsymbol{h}^{(l-1)}_t \leftarrow d\boldsymbol{h}^{(l-1)}_t + d\boldsymbol{x}$; // Vertical gradients

    $d\boldsymbol{T}^{(l)}_{\text{lstm}} \leftarrow d\boldsymbol{T}^{(l)}_{\text{lstm}} + d\boldsymbol{T}$;

  end

  $d\boldsymbol{W}_e \leftarrow \texttt{Emb\_grad\_update}(s_t, d\boldsymbol{h}^{(0)}_t, d\boldsymbol{W}_e)$;

end

$d\boldsymbol{W}^{\text{encoder}}_e, d\boldsymbol{T}^{\text{encoder}}_{\text{lstm}} \leftarrow d\boldsymbol{W}_e, d\boldsymbol{T}^{(1..L)}_{\text{lstm}}$; // Save encoder gradients

at the final time step (line 1) as well the model weights on the decoder size (line 2) to zero. At time m_x, we switch to the encoder mode by saving the currently accumulated LSTM and embedding gradients for the decoder (line 5) and starting to accumulate gradients for the encoder weights (line 6). Thanks to the backpropagation procedure presented earlier for LSTM, we can simplify the core NMT gradient computation (lines 8-18) by making the following two referents: (a) `Predict_grad` (lines 2-4 of Algorithm 2) which computes gradients for the target-side losses with respect to the hidden states at the top layer and the softmax weights $\boldsymbol{W}_{hy}$; and (b) `LSTM_grad` (lines 5-15 of Algorithm 2) which computes gradients for inputs to LSTM and the LSTM weights per layer $\boldsymbol{T}_{\text{lstm}}^{(l)}$. It is important to note that in Lines 10 and 15 of Algorithm 4, we add the gradients (flowing vertically from either the loss or the upper LSTM layer) to the gradient of the below layer (which already contains the gradient backpropagated horizontally) instead of overriding it. In Line 18, we perform sparse updates on the corresponding embedding matrix for participating words only. Lastly, implementation wise, one can save memory by using a single copy of $d\boldsymbol{h}^{(1..L)}$ and $d\boldsymbol{c}^{(1..L)}$ for all time steps and overwriting the values whenever we transition from timesteps t to $t-1$ (line 14).

2.3.2 Testing

Having trained an NMT model, we, of course, need to be able to use it to translate, or decode, unseen source sentences! This section explains a few different ways to accomplish this goal and how to decode with an ensemble of models.

The simplest strategy to translate a source sentence is to perform *greedy decoding* which we illustrate in Figure 2.5. The idea is simple: (a) we first encode the source sentence, "I am a student" in our example, similar to the training process; (b) the decoding process is started as soon as an end-of-sentence marker "_" for the source sentence is fed as an input; and (c) for each timestep on the decoder side, we pick the most likely word (a greedy choice), e.g., "moi" has the highest translation probability in the first decoding step, then use it as an input to the next timestep, and continue until the end-of-sentence marker "_" is produced as an output symbol. Step (c) is what makes testing different from training: unlike training in which correct target words in y are always fed as an input, testing, on the

other hand, uses words predicted by the model.

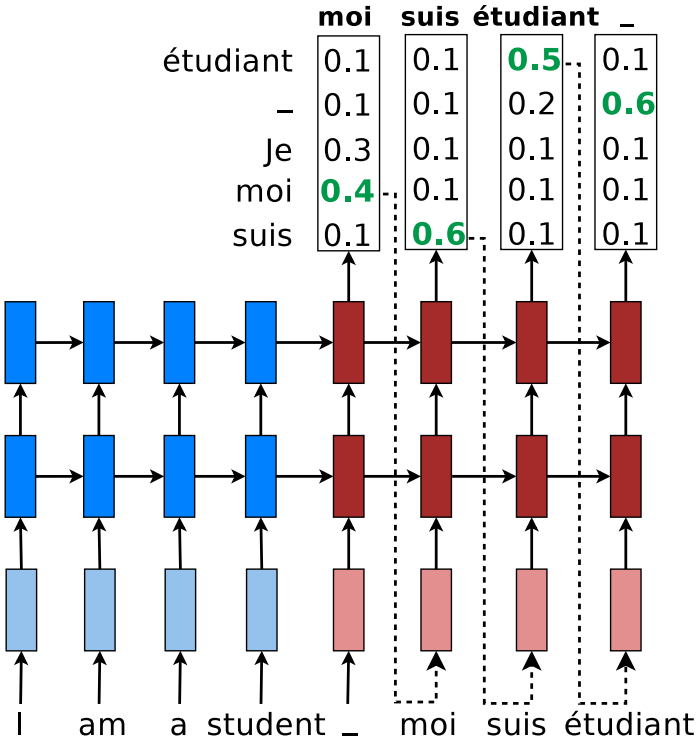

Figure 2.5: **Greedy Decoding** – example of how a trained NMT model produces a translation for a source sentence "I am a student" using greedy search.

More concretely, we adopt the NMT forward algorithm to arrive at the greedy decoding strategy in Algorithm 5. We present the greedy algorithm in a slightly more abstract way by reusing elements of the NMT forward pass in Algorithm 3. First, we run through the encoder in Line 1 to obtain a representation $\boldsymbol{h}_0, \boldsymbol{c}_0$ for the source sentence x (length m_x). We then use the end-of-sentence marker "_" as an input to start the decoding process and restrict the final translation to have a maximum length of $\alpha * m_x$.[7] At each timestep on the decoder side, we call `MultiLayerLSTM`, which refers to Lines 8-11 in Algorithm 3, to build up representations over L stacking LSTM layers. The hidden state at the top layer is used to compute the predictive distribution $\boldsymbol{p}_t$ from which we make a greedy choice to produce the index of the translation word at that timestep (line 7). The process ends

[7] We often set α to 1.5.

when we have produced the marker "_" as a translation word or when the translation length exceeds the length threshold.

Algorithm 5: NMT *greedy* decoding algorithm.

1 $\boldsymbol{h}_0, \boldsymbol{c}_0 \leftarrow \text{Encoder}(x, \boldsymbol{W}_e^{\text{encoder}}, \boldsymbol{T}_{\text{lstm}}^{\text{encoder}})$;
2 $t \leftarrow 1$;
3 $y_1 \leftarrow$ _ ;
4 **while** $t \leq \alpha * m_x$ **do** // Length factor $\alpha \geq 1$
5 $\quad \boldsymbol{h}_t, \boldsymbol{c}_t \leftarrow \texttt{MultiLayerLSTM}\left(\boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, y_t, \boldsymbol{W}_e^{\text{decoder}}, \boldsymbol{T}_{\text{lstm}}^{\text{decoder}}\right)$;
6 $\quad \boldsymbol{p}_t \leftarrow \texttt{Softmax}(\boldsymbol{h}_t^{(L)}, \boldsymbol{W}_{hy})$;
7 $\quad y_{t+1} \leftarrow \text{argmax}_i\, \boldsymbol{p}_t(i)$; // ***Greedy*** choice
8 $\quad$ **if** $y_{t+1} == \text{Index}(\_)$ **then** // Ending condition
9 $\quad\quad$ break;
10 $\quad$ **end**
11 $\quad t \leftarrow t + 1$
12 **end**
13 **return** $y_{2..t}$

For NMT, it turns out that such a simple strategy of greedy decoding can produce very good translations (Sutskever et al., 2014). However, to achieve a better result, a more popular strategy is to use a *beam-search* decoding algorithm which has been the core of phrase-based statistical machine translation for years (Koehn et al., 2003). Unlike phrase-based SMT, NMT has a much simpler beam-search decoding algorithm since it generates translations word-by-word from left to right and does not have to explicitly explore different places on the source side to pay attention to.[8] One can modify the greedy decoding algorithm as follows to build a beam-search decoder: (a) at each timestep on the decoder side, we keep track of the top B (the beam size) best translations together with their corresponding hidden states; (b) in Line 7 of Algorithm 5, instead of applying argmax, we select the top B most likely words; and (c) given B previous best translation $\times B$ best words, we select a new set of B best translations for the current timestep based on the combined scores (previous translation scores + current word translation scores). Extra care needs to be taken to make sure that in step (c) we select correct hidden states for the new set of B

[8] In SMT, a source coverage set is maintained to indicate which words have been translated. As translation progresses, an SMT system base on the coverage set and pick untranslated source words to continue. Such an idea of coverage set later re-emerges in NMT which we will describe more in Chapter 7.

best translations. Sutskever et al. (2014) observed that for NMT, a minimal beam size of 2 already provides a significant boost in translation quality. A beam of size 10 is often used, which is significant smaller that what phrase-based SMT tends to use, about $100 - 200$.

Furthermore, to achieve the very best result, one simple strategy which has been widely adopted for deep neural networks is to use an ensemble of models. For NMT decoding, using multiple models is pretty straightforward. The idea is that each model produces a distribution at each timestep in the decoder (line 6 of Algorithm 5). These different distributions are then averaged to produce a new ensemble distribution which we can use for both greedy and beam-search decoders as if we decode from a single model.

In summary, we have covered all the necessary background knowledge to understand this thesis entirely. We start with language modeling, an important concept in natural language processing, which turns out to be the basis of NMT. One can simply view NMT as a source-conditioned language model. To make NMT possible, we have covered the fundamentals of recurrent neural networks which allow us to handle variable-length sequences, in our case, the sentences. We have particularly studied Long Short-term Memory, a specific type of RNN that is more effective at handling long sequences, in depth. Finally, given these building blocks, language modeling and RNN, we have discussed NMT in details from training to testing.

Chapter 3

Copy Mechanisms

Neural Machine Translation (NMT) is a novel approach to MT that has achieved promising results (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015; Jean et al., 2015a). An NMT system is a conceptually simple large neural network that reads the entire source sentence and produces an output translation one word at a time. NMT systems are appealing because they use minimal domain knowledge which makes them well-suited to any problem that can be formulated as mapping an input sequence to an output sequence (Sutskever et al., 2014). In addition, the natural ability of neural networks to generalize implies that NMT systems will also generalize to novel word phrases and sentences that do not occur in the training set. In addition, NMT systems potentially remove the need to store explicit phrase tables and language models which are used in conventional systems. Finally, the decoder of an NMT system is easy to implement, unlike the highly intricate decoders used by phrase-based systems (Koehn et al., 2003).

Despite these advantages, conventional NMT systems are incapable of translating rare words because they have a fixed modest-sized vocabulary[1] which forces them to use the `unk` symbol to represent the large number of out-of-vocabulary (OOV) words, as illustrated in Figure 3.1. Unsurprisingly, both Sutskever et al. (2014) and Bahdanau et al. (2015) have observed that sentences with many rare words tend to be translated much more poorly than

[1] Due to the computationally intensive nature of the softmax, NMT systems often limit their vocabularies to be the top 30K-80K most frequent words in each language. However, Jean et al. (2015a) has very recently proposed an efficient approximation to the softmax that allows for training NTMs with very large vocabularies. As discussed in Section 5.2, this technique is complementary to ours.

sentences containing mainly frequent words. Standard phrase-based systems (Koehn et al., 2007; Chiang, 2007; Cer et al., 2010; Dyer et al., 2010), on the other hand, do not suffer from the rare word problem to the same extent because they can support a much larger vocabulary, and because their use of explicit alignments and phrase tables allows them to memorize the translations of even extremely rare words.

Motivated by the strengths of standard phrase-based system, we propose and implement a novel approach to address the rare word problem of NMTs. Our approach annotates the training corpus with explicit alignment information that enables the NMT system to emit, for each OOV word, a "pointer" to its corresponding word in the source sentence. This information is later utilized in a post-processing step that translates the OOV words using a dictionary or with the identity translation, if no translation is found.

Our experiments confirm that this approach is effective. On the English to French WMT'14 translation task, this approach provides an improvement of up to 2.8 (if the vocabulary is relatively small) BLEU points over an equivalent NMT system that does not use this technique. Moreover, our system is the first NMT that outperforms the winner of a WMT'14 task.

en: The *ecotax* portico in *Pont-de-Buis* , . . . [truncated] . . . , was taken down on Thursday morning

fr: Le *portique* *écotaxe* de *Pont-de-Buis* , . . . [truncated] . . . , a été *démonté* jeudi matin

nn: Le `unk` de `unk` à `unk` , . . . [truncated] . . . , a été pris le jeudi matin

Figure 3.1: **Example of the rare word problem** – An English source sentence (*en*), a human translation to French (*fr*), and a translation produced by one of our neural network systems (*nn*) before handling OOV words. We highlight *words* that are unknown to our model. The token `unk` indicates an OOV word. We also show a few important alignments between the pair of sentences.

3.1 Neural Machine Translation

A neural machine translation system is any neural network that maps a source sentence, $s_1, \ldots, s_n$, to a target sentence, $t_1, \ldots, t_m$, where all sentences are assumed to terminate

with a special "end-of-sentence" token <eos>. More concretely, an NMT system uses a neural network to parameterize the conditional distributions

$$p(t_j|t_{<j}, s_{\leq n}) \tag{3.1}$$

for $1 \leq j \leq m$. By doing so, it becomes possible to compute and therefore maximize the log probability of the target sentence given the source sentence

$$\log p(t|s) = \sum_{j=1}^{m} \log p\left(t_j|t_{<j}, s_{\leq n}\right) \tag{3.2}$$

There are many ways to parameterize these conditional distributions. For example, Kalchbrenner and B (2013) used a combination of a convolutional neural network and a recurrent neural network, Sutskever et al. (2014) used a deep Long Short-Term Memory (LSTM) model, Cho et al. (2014) used an architecture similar to the LSTM, and Bahdanau et al. (2015) used a more elaborate neural network architecture that uses an attentional mechanism over the input sequence, similar to Graves (2013) and Graves et al. (2014).

In this work, we use the model of Sutskever et al. (2014), which uses a deep LSTM to encode the input sequence and a separate deep LSTM to output the translation. The encoder reads the source sentence, one word at a time, and produces a large vector that represents the entire source sentence. The decoder is initialized with this vector and generates a translation, one word at a time, until it emits the end-of-sentence symbol <eos>.

None the early work in neural machine translation systems has addressed the rare word problem, but the recent work of Jean et al. (2015a) has tackled it with an efficient approximation to the softmax to accommodate for a very large vocabulary (500K words). However, even with a large vocabulary, the problem with rare words, e.g., names, numbers, etc., still persists, and Jean et al. (2015a) found that using techniques similar to ours are beneficial and complementary to their approach.

3.2 Rare Word Models

Despite the relatively large amount of work done on pure neural machine translation systems, there has been no work addressing the OOV problem in NMT systems, with the notable exception of Jean et al. (2015a)'s work mentioned earlier.

We propose to address the rare word problem by training the NMT system to track the origins of the unknown words in the target sentences. If we knew the source word responsible for each unknown target word, we could introduce a post-processing step that would replace each *unk* in the system's output with a translation of its source word, using either a dictionary or the identity translation. For example, in Figure 3.1, if the model knows that the second unknown token in the NMT (line *nn*) originates from the source word `ecotax`, it can perform a word dictionary lookup to replace that unknown token by `écotaxe`. Similarly, an identity translation of the source word `Pont-de-Buis` can be applied to the third unknown token.

We present three annotation strategies that can easily be applied to any NMT system (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). We treat the NMT system as a black box and train it on a corpus annotated by one of the models below. First, the alignments are produced with an unsupervised aligner. Next, we use the alignment links to construct a word dictionary that will be used for the word translations in the post-processing step.[2] If a word does not appear in our dictionary, then we apply the identity translation.

The first few words of the sentence pair in Figure 3.1 (lines *en* and *fr*) illustrate our models.

3.2.1 Copyable Model

In this approach, we introduce multiple tokens to represent the various unknown words in the source and in the target language, as opposed to using only one *unk* token. We annotate the OOV words in the source sentence with $\underline{unk}_1$, $\underline{unk}_2$, $\underline{unk}_3$, in that order,

[2]When a source word has multiple translations, we use the translation with the highest probability. These translation probabilities are estimated from the unsupervised alignment links. When constructing the dictionary from these alignment links, we add a word pair to the dictionary only if its alignment count exceeds 100.

en: The $\underline{unk}_1$ portico in $\underline{unk}_2$ ...

fr: Le $\underline{unk}_\emptyset$ $\underline{unk}_1$ de $\underline{unk}_2$ ...

Figure 3.2: **Copyable Model** – an annotated example with two types of unknown tokens: "copyable" $\underline{unk}_n$ and null $\underline{unk}_\emptyset$.

while assigning repeating unknown words identical tokens. The annotation of the unknown words in the target language is slightly more elaborate: (a) each unknown target word that is aligned to an unknown source word is assigned the same unknown token (hence, the "copy" model) and (b) an unknown target word that has no alignment or that is aligned with a known word uses the special null token $\underline{unk}_\emptyset$. See Figure 3.2 for an example. This annotation enables us to translate every non-null unknown token.

3.2.2 Positional All Model (PosAll)

The copyable model is limited by its inability to translate unknown target words that are aligned to *known* words in the source sentence, such as the pair of words, "portico" and "portique", in our running example. The former word is known on the source sentence; whereas latter is not, so it is labelled with $\underline{unk}_\emptyset$. This happens often since the source vocabularies of our models tend to be much larger than the target vocabulary since a large source vocabulary is cheap. This limitation motivated us to develop an annotation model that includes the complete alignments between the source and the target sentences, which is straightforward to obtain since the complete alignments are available at training time.

Specifically, we return to using only a single universal $\underline{unk}$ token. However, on the target side, we insert a positional token p_d after every word. Here, d indicates a relative position ($d = -7, \ldots, -1, 0, 1, \ldots, 7$) to denote that a target word at position j is aligned to a source word at position $i = j - d$. Aligned words that are too far apart are considered unaligned, and unaligned words rae annotated with a null token p_n. Our annotation is illustrated in Figure 3.3.

en: The $\underline{unk}$ portico in $\underline{unk}$ ...

fr: Le p_0 $\underline{unk}$ p_{-1} $\underline{unk}$ p_1 de $p_\emptyset$ $\underline{unk}$ p_{-1} ...

Figure 3.3: **Positional All Model** – an example of the PosAll model. Each word is followed by the relative positional tokens p_d or the null token $p_\emptyset$.

3.2.3 Positional Unknown Model (PosUnk)

The main weakness of the PosAll model is that it doubles the length of the target sentence. This makes learning more difficult and slows the speed of parameter updates by a factor of two. However, given that our post-processing step is concerned only with the alignments of the unknown words, so it is more sensible to only annotate the unknown words. This motivates our *positional unknown* model which uses $unkpos_d$ tokens (for d in $-7, \ldots, 7$ or $\emptyset$) to simultaneously denote (a) the fact that a word is unknown and (b) its relative position d with respect to its aligned source word. Like the PosAll model, we use the symbol $\underline{unkpos_\emptyset}$ for unknown target words that do not have an alignment. We use the universal $\underline{unk}$ for all unknown tokens in the source language. See Figure 3.4 for an annotated example.

en: The $\underline{unk}$ portico in $\underline{unk}$ ...

fr: Le $\underline{unkpos_1}$ $\underline{unkpos_{-1}}$ de $\underline{unkpos_1}$ ...

Figure 3.4: **Positional Unknown Model** – an example of the PosUnk model: only aligned unknown words are annotated with the $unkpos_d$ tokens.

It is possible that despite its slower speed, the PosAll model will learn better alignments because it is trained on many more examples of words and their alignments. However, we show that this is not the case (see §3.4.2).

3.3 Experiments

We evaluate the effectiveness of our OOV models on the WMT'14 English-to-French translation task. Translation quality is measured with the BLEU metric (Papineni et al., 2002) on the newstest2014 test set (which has 3003 sentences).

3.3.1 Training Data

To be comparable with the results reported by previous work on neural machine translation systems (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015), we train our models on the same training data of 12M parallel sentences (348M French and 304M English words), obtained from (Schwenk, 2014). The 12M subset was selected from the full WMT'14 parallel corpora using the method proposed in Axelrod et al. (2011).

Due to the computationally intensive nature of the naive softmax, we limit the French vocabulary (the *target* language) to the either the 40K or the 80K most frequent French words. On the *source* side, we can afford a much larger vocabulary, so we use the 200K most frequent English words. The model treats all other words as unknowns.[3]

We annotate our training data using the three schemes described in the previous section. The alignment is computed with the Berkeley aligner (Liang et al., 2006) using its default settings. We discard sentence pairs in which the source or the target sentence exceed 100 tokens.

3.3.2 Training Details

Our training procedure and hyperparameter choices are similar to those used by Sutskever et al. (2014). In more details, we train multi-layer deep LSTMs, each of which has 1000 cells, with 1000 dimensional embeddings. Like Sutskever et al. (2014), we reverse the words in the source sentences which has been shown to improve LSTM memory utilization and results in better translations of long sentences. Our hyperparameters can be summarized as follows: (a) the parameters are initialized uniformly in [-0.08, 0.08] for 4-layer models and [-0.06, 0.06] for 6-layer models, (b) SGD has a fixed learning rate of 0.7, (c) we train for 8 epochs (after 5 epochs, we begin to halve the learning rate every 0.5 epoch), (d) the size of the mini-batch is 128, and (e) we rescale the normalized gradient to ensure that its norm does not exceed 5 (Pascanu et al., 2013).

We also follow the GPU parallelization scheme proposed in (Sutskever et al., 2014), allowing us to reach a training speed of 5.4K words per second to train a depth-6 model

[3]When the French vocabulary has 40K words, there are on average 1.33 unknown words per sentence on the target side of the test set.

with 200K source and 80K target vocabularies ; whereas Sutskever et al. (2014) achieved 6.3K words per second for a depth-4 models with 80K source and target vocabularies. Training takes about 10-14 days on an 8-GPU machine.

3.3.3 A note on BLEU scores

We report BLEU scores based on both: (a) *detokenized* translations, i.e., WMT'14 style, to be comparable with results reported on the WMT website[4] and (b) *tokenized translations*, so as to be consistent with previous work (Cho et al., 2014; Bahdanau et al., 2015; Schwenk, 2014; Sutskever et al., 2014; Jean et al., 2015a).[5]

The existing WMT'14 state-of-the-art system (Durrani et al., 2014) achieves a detokenized BLEU score of 35.8 on the newstest2014 test set for English to French language pair (see Table 3.1). In terms of the tokenized BLEU, its performance is 37.0 points (see Table 3.2).

System	**BLEU**
Existing SOTA (Durrani et al., 2014)	35.8
Ensemble of 8 LSTMs + PosUnk	**36.6**

Table 3.1: **Detokenized BLEU on newstest2014** – translation results of the existing state-of-the-art system and our best system.

3.3.4 Main Results

We compare our systems to others, including the current state-of-the-art MT system (Durrani et al., 2014), recent end-to-end neural systems, as well as phrase-based baselines with neural components.

The results shown in Table 3.2 demonstrate that our unknown word translation technique (in particular, the PosUnk model) significantly improves the translation quality for both the individual (non-ensemble) LSTM models and the ensemble models.[6] For 40K-word vocabularies, the performance gains are in the range of 2.3-2.8 BLEU points. With

[4] `http://matrix.statmt.org/matrix`

[5] The `tokenizer.perl` and `multi-bleu.pl` scripts are used to tokenize and score translations.

[6] For the 40K-vocabulary ensemble, we combine 5 models with 4 layers and 3 models with 6 layers. For the 80K-vocabulary ensemble, we combine 3 models with 4 layers and 5 models with 6 layers. Two of the

larger vocabularies (80K), the performance gains are diminished, but our technique can still provide a nontrivial gains of 1.6-1.9 BLEU points.

System	Vocab	Corpus	BLEU
State of the art in WMT'14 (Durrani et al., 2014)	All	36M	**37.0**
Standard MT + neural components			
Schwenk (2014) – neural language model	All	12M	33.3
Cho et al. (2014)– phrase table neural features	All	12M	34.5
Sutskever et al. (2014) – 5 LSTMs, reranking 1000-best lists	All	12M	36.5
Existing end-to-end NMT systems			
Bahdanau et al. (2015) – single gated RNN with search	30K	12M	28.5
Sutskever et al. (2014) – 5 LSTMs	80K	12M	34.8
Jean et al. (2015a) – 8 gated RNNs with search + UNK replacement	500K	12M	37.2
Our end-to-end NMT systems			
Single LSTM with 4 layers	40K	12M	29.5
Single LSTM with 4 layers + PosUnk	40K	12M	31.8 (+2.3)
Single LSTM with 6 layers	40K	12M	30.4
Single LSTM with 6 layers + PosUnk	40K	12M	32.7 (+2.3)
Ensemble of 8 LSTMs	40K	12M	34.1
Ensemble of 8 LSTMs + PosUnk	40K	12M	36.9 (+2.8)
Single LSTM with 6 layers	80K	36M	31.5
Single LSTM with 6 layers + PosUnk	80K	36M	33.1 (+1.6)
Ensemble of 8 LSTMs	80K	36M	35.6
Ensemble of 8 LSTMs + PosUnk	80K	36M	**37.5 (+1.9)**

Table 3.2: **Tokenized BLEU on newstest2014** – Translation results of various systems which differ in terms of: (a) the architecture, (b) the size of the vocabulary used, and (c) the training corpus, either using the full WMT'14 corpus of 36M sentence pairs or a subset of it with 12M pairs. We highlight the performance of our best system in bolded text and state the improvements obtained by our technique of handling rare words (namely, the PosUnk model). Notice that, for a given vocabulary size, the more accurate systems achieve a greater improvement from the post-processing step. This is the case because the more accurate models are able to pin-point the origin of an unknown word with greater accuracy, making the post-processing more useful.

It is interesting to observe that our approach is more useful for ensemble models as compared to the individual ones. This is because the usefulness of the PosUnk model directly depends on the ability of the NMT to correctly locate, for a given OOV target word, its corresponding word in the source sentence. An ensemble of large models identifies these source words with greater accuracy. This is why for the same vocabulary size, better

depth-6 models are regularized with dropout, similar to Zaremba et al. (2014) with the dropout probability set to 0.2.

models obtain a greater performance gain our post-processing step. Except for the very recent work of Jean et al. (2015a) that employs a similar unknown treatment strategy[7] as ours, our best result of 37.5 BLEU outperforms all other NMT systems by a arge margin, and more importanly, our system has established a new record on the WMT'14 English to French translation.

3.4 Analysis

We analyze and quantify the improvement obtained by our rare word translation approach and provide a detailed comparison of the different rare word techniques proposed in Section 3.2. We also examine the effect of depth on the LSTM architectures and demonstrate a strong correlation between perplexities and BLEU scores. We also highlight a few translation examples where our models succeed in correctly translating OOV words, and present several failures.

3.4.1 Rare Word Analysis

To analyze the effect of rare words on translation quality, we follow Sutskever et al. (Sutskever et al., 2014) and sort sentences in newstest2014 by the average inverse frequency of their words. We split the test sentences into groups where the sentences within each group have a comparable number of rare words and evaluate each group independently. We evaluate our systems before and after translating the OOV words and compare with the standard MT systems – we use the best system from the WMT'14 contest (Durrani et al., 2014), and neural MT systems – we use the ensemble systems described in (Sutskever et al., 2014) and Section 6.1.2.

Rare word translation is challenging for neural machine translation systems as shown

[7]Their unknown replacement method and ours both track the locations of target unknown words and use a word dictionary to post-process the translation. However, the mechanism used to achieve the "tracking" behavior is different. Jean et al. (2015a)'s uses the attentional mechanism to track the origins of all target words, not just the unknown ones. In contrast, we only focus on tracking unknown words using unsupervised alignments. Our method can be easily applied to any sequence-to-sequence models since we treat any model as a blackbox and manipulate only at the input and output levels.

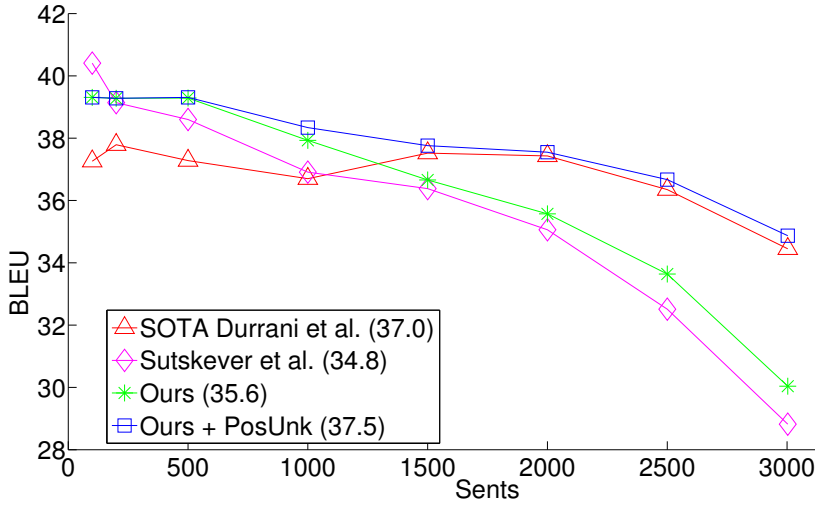

Figure 3.5: **Rare word translation** – On the x-axis, we order newstest2014 sentences by their *average frequency rank* and divide the sentences into groups of sentences with a comparable prevalence of rare words. We compute the BLEU score of each group independently.

in Figure 3.5. Specifically, the translation quality of our model before applying the postprocessing step is shown by the green curve, and the current best NMT system (Sutskever et al., 2014) is the purple curve. While (Sutskever et al., 2014) produces better translations for sentences with frequent words (the left part of the graph), they are worse than best system (red curve) on sentences with many rare words (the right side of the graph). When applying our unknown word translation technique (purple curve), we significantly improve the translation quality of our NMT: for the last group of 500 sentences which have the greatest proportion of OOV words in the test set, we increase the BLEU score of our system by 4.8 BLEU points. Overall, our rare word translation model interpolates between the SOTA system and the system of Sutskever et al. (2014), which allows us to outperform the winning entry of WMT'14 on sentences that consist predominantly of frequent words and approach its performance on sentences with many OOV words.

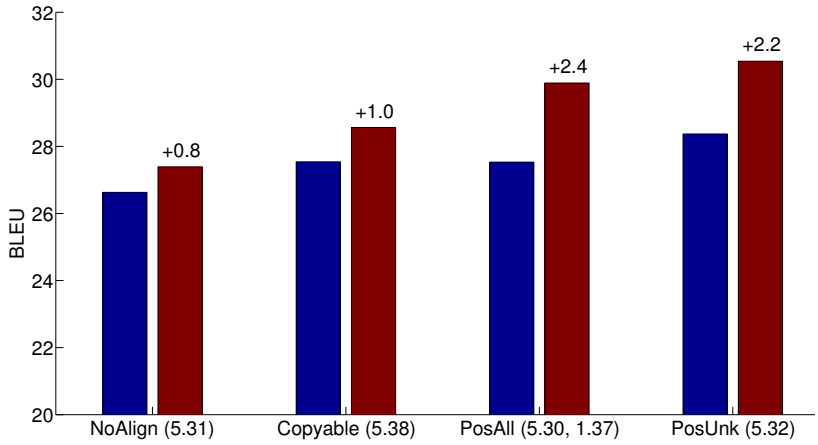

Figure 3.6: **Rare word models** – translation performance of 6-layer LSTMs: a model that uses no alignment (*NoAlign*) and the other rare word models (*Copyable, PosAll, PosUnk*). For each model, we show results before (*left*) and after (*right*) the rare word translation as well as the perplexity (in parentheses). For *PosAll*, we report the perplexities of predicting the words and the positions.

3.4.2 Rare Word Models

We examine the effect of the different rare word models presented in Section 3.2, namely: (a) *Copyable* – which aligns the unknown words on both the input and the target side by learning to copy indices, (b) the Positional All (*PosAll*) – which predicts the aligned source positions for every target word, and (c) the Positional Unknown (*PosUnk*) – which predicts the aligned source positions for only the unknown target words.[8] It is also interesting to measure the improvement obtained when no alignment information is used during training. As such, we include a baseline model with no alignment knowledge (*NoAlign*) in which we simply assume that the i^{th} unknown word on the target sentence is aligned to the i^{th} unknown word in the source sentence.

From the results in Figure 3.6, a simple monotone alignment assumption for the *NoAlign*

[8] In this section and in section 3.4.3, all models are trained on the unreversed sentences, and we use the following hyperparameters: we initialize the parameters uniformly in [-0.1, 0.1], the learning rate is 1, the maximal gradient norm is 1, with a source vocabulary of 90k words, and a target vocabulary of 40k (see Section 3.3.2 for more details). While these LSTMs do not achieve the best possible performance, it is still useful to analyze them.

model yields a modest gain of 0.8 BLEU points. If we train the model to predict the alignment, then the *Copyable* model offers a slightly better gain of 1.0 BLEU. Note, however, that English and French have similar word order structure, so it would be interesting to experiment with other language pairs, such as English and Chinese, in which the word order is not as monotonic. These harder language pairs potentially imply a smaller gain for the NoAlign model and a larger gain for the Copyable model. We leave it for future work.

The positional models (*PosAll* and *PosUnk*) improve translation performance by more than 2 BLEU points. This proves that the limitation of the copyable model, which forces it to align each unknown output word with an unknown input word, is considerable. In contrast, the positional models can align the unknown target words with any source word, and as a result, post-processing has a much stronger effect. The PosUnk model achieves better translation results than the PosAll model which suggests that it is easier to train the LSTM on shorter sequences.

3.4.3 Other Effects

Deep LSTM architecture – We compare PosUnk models trained with different number of layers (3, 4, and 6). We observe that the gain obtained by the PosUnk model increases in tandem with the overall accuracy of the model, which is consistent with the idea that larger models can point to the appropriate source word more accurately. Additionally, we observe that on average, each extra LSTM layer provides roughly 1.0 BLEU point improvement as demonstrated in Figure 3.7.

Perplexity and BLEU – Lastly, we find it interesting to observe a strong correlation between the perplexity (our training objective) and the translation quality as measured by BLEU. Figure 3.8 shows the performance of a 4-layer LSTM, in which we compute both perplexity and BLEU scores at different points during training. We find that on average, a reduction of 0.5 perplexity gives us roughly 1.0 BLEU point improvement.

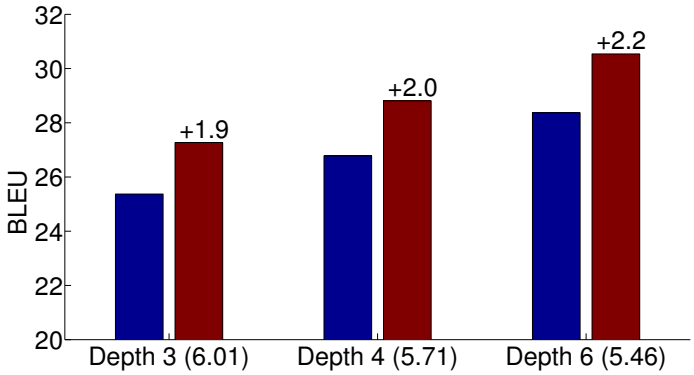

Figure 3.7: **Effect of depths** – BLEU scores achieved by *PosUnk* models of various depths (3, 4, and 6) before and after the rare word translation. Notice that the PosUnk model is more useful on more accurate models.

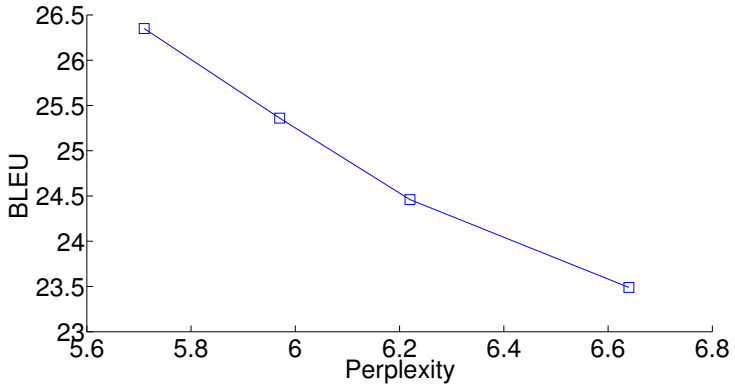

Figure 3.8: **Perplexity vs. BLEU** – we show the correlation by evaluating an LSTM model with 4 layers at various stages of training.

3.4.4 Sample Translations

We present three sample translations of our best system (with 37.5 BLEU) in Table 5.4. In our first example, the model translates all the unknown words correctly: *2600*, *orthopédiques*, and *cataracte*. It is interesting to observe that the model can accurately predict an alignment of distances of 5 and 6 words. The second example highlights the fact that our model can translate long sentences reasonably well and that it was able to correctly translate the unknown word for *JPMorgan* at the very far end of the source sentence. Lastly, our examples also reveal several penalties incurred by our model: (a) incorrect entries in

the word dictionary, as with *négociateur* vs. *trader* in the second example, and (b) incorrect alignment prediction, such as when $unkpos_3$ is incorrectly aligned with the source word *was* and not with *abandoning*, which resulted in an incorrect translation in the third sentence.

	Sentences
src	An additional *2600* operations including *orthopedic* and *cataract* surgery will help clear a backlog .
trans	En outre , $unkpos_1$ opérations supplémentaires , dont la chirurgie $unkpos_5$ et la $unkpos_6$, permettront de résorber l' arriéré .
+unk	En outre , *2600* opérations supplémentaires , dont la chirurgie *orthopédiques* et la *cataracte* , permettront de résorber l' arriéré .
tgt	2600 opérations supplémentaires , notamment dans le domaine de la chirurgie orthopédique et de la cataracte , aideront à rattraper le retard .
src	This *trader* , Richard *Usher* , left RBS in *2010* and is understand to have be given leave from his current position as European head of forex spot trading at *JPMorgan* .
trans	Ce $unkpos_0$, Richard $unkpos_0$, a quitté $unkpos_1$ en 2010 et a compris qu' il est autorisé à quitter son poste actuel en tant que leader européen du marché des points de vente au $unkpos_5$.
+unk	Ce *négociateur* , Richard *Usher* , a quitté RBS en *2010* et a compris qu' il est autorisé à quitter son poste actuel en tant que leader européen du marché des points de vente au *JPMorgan* .
tgt	Ce trader , Richard Usher , a quitté RBS en 2010 et aurait été mis suspendu de son poste de responsable européen du trading au comptant pour les devises chez JPMorgan
src	But concerns have grown after Mr *Mazanga* was quoted as saying *Renamo was* abandoning the 1992 peace accord .
trans	Mais les inquiétudes se sont accrues après que M. $unkpos_3$ a déclaré que la $unkpos_3$ $unkpos_3$ l' accord de paix de 1992 .
+unk	Mais les inquiétudes se sont accrues après que M. *Mazanga* a déclaré que la *Renamo était* l' accord de paix de 1992 .
tgt	Mais l' inquiétude a grandi après que M. Mazanga a déclaré que la Renamo abandonnait l' accord de paix de 1992 .

Table 3.3: **Sample translations** – the table shows the source (*src*) and the translations of our best model before (*trans*) and after (+*unk*) unknown word translations. We also show the human translations (*tgt*) and italicize words that are involved in the unknown word translation process.

3.5 Conclusion

We have shown that a simple alignment-based technique can mitigate and even overcome one of the main weaknesses of current NMT systems, which is their inability to translate words that are not in their vocabulary. A key advantage of our technique is the fact that it is applicable to any NMT system and not only to the deep LSTM model of Sutskever et al. (2014). A technique like ours is likely necessary if an NMT system is to achieve state-of-the-art performance on machine translation.

We have demonstrated empirically that on the WMT'14 English-French translation task, our technique yields a consistent and substantial improvement of up to 2.8 BLEU points over various NMT systems of different architectures. Most importantly, with 37.5 BLEU points, we have established the first NMT system that outperformed the best MT system on a WMT'14 contest dataset.

Chapter 4

Attention Mechanisms

Neural Machine Translation (NMT) achieved state-of-the-art performances in large-scale translation tasks such as from English to French (Luong et al., 2015c) and English to German (Jean et al., 2015a). NMT is appealing since it requires minimal domain knowledge and is conceptually simple. The model by Luong et al. (2015c) reads through all the source words until the end-of-sentence symbol $</\mathtt{s}>$ is reached. It then starts emitting one target word at a time, as illustrated in Figure 4.1. NMT is often a large neural network that is trained in an end-to-end fashion and has the ability to generalize well to very long word sequences. This means the model does not have to explicitly store gigantic phrase tables and language models as in the case of standard MT; hence, NMT has a small memory footprint. Lastly, implementing NMT decoders is easy unlike the highly intricate decoders in standard MT (Koehn et al., 2003).

In parallel, the concept of "attention" has gained popularity recently in training neural networks, allowing models to learn alignments between different modalities, e.g., between image objects and agent actions in the dynamic control problem (Mnih et al., 2014), between speech frames and text in the speech recognition task (Chorowski et al., 2014), or between visual features of a picture and its text description in the image caption generation task (Xu et al., 2015). In the context of NMT, Bahdanau et al. (2015) has successfully applied such attentional mechanism to jointly translate and align words. To the best of our knowledge, there has not been any other work exploring the use of attention-based architectures for NMT.

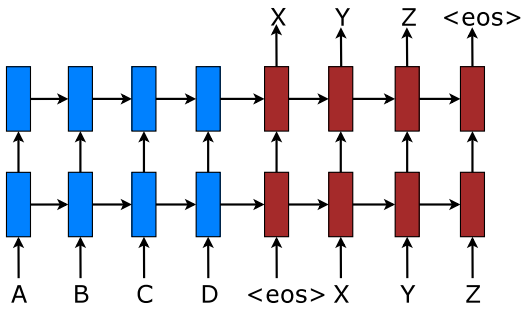

Figure 4.1: **Neural machine translation** – a stacking recurrent architecture for translating a source sequence `A B C D` into a target sequence `X Y Z`. Here, `</s>` marks the end of a sentence.

In this work, we design, with simplicity and effectiveness in mind, two novel types of attention-based models: a *global* approach in which all source words are attended and a *local* one whereby only a subset of source words are considered at a time. The former approach resembles the model of (Bahdanau et al., 2015) but is simpler architecturally. The latter can be viewed as an interesting blend between the *hard* and *soft* attention models proposed in (Xu et al., 2015): it is computationally less expensive than the global model or the soft attention; at the same time, unlike the hard attention, the local attention is differentiable almost everywhere, making it easier to implement and train.[1] Besides, we also examine various alignment functions for our attention-based models.

Experimentally, we demonstrate that both of our approaches are effective in the WMT translation tasks between English and German in both directions. Our attentional models yield a boost of up to 5.0 BLEU over non-attentional systems which already incorporate known techniques such as dropout. For English to German translation, we achieve new state-of-the-art (SOTA) results for both WMT'14 and WMT'15, outperforming previous SOTA systems, backed by NMT models and n-gram LM rerankers, by more than 1.0 BLEU. We conduct extensive analysis to evaluate our models in terms of learning, the ability to handle long sentences, choices of attentional architectures, alignment quality, and translation outputs.

[1]There is a recent work by Gregor et al. (2015), which is very similar to our local attention and applied to the image generation task. However, as we detail later, our model is much simpler and can achieve good performance for NMT.

4.1 Neural Machine Translation

A neural machine translation system is a neural network that directly models the conditional probability $p(y|x)$ of translating a source sentence, $x_1, \ldots, x_n$, to a target sentence, $y_1, \ldots, y_m$.[2] A basic form of NMT consists of two components: (a) an *encoder* which computes a representation $\boldsymbol{s}$ for each source sentence and (b) a *decoder* which generates one target word at a time and hence decomposes the conditional probability as:

$$\log p(y|x) = \sum_{j=1}^{m} \log p\left(y_j|y_{<j}, \boldsymbol{s}\right) \tag{4.1}$$

A natural choice to model such a decomposition in the decoder is to use a recurrent neural network (RNN) architecture, which most of the recent NMT work such as (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015; Luong et al., 2015c; Jean et al., 2015a) have in common. They, however, differ in terms of which RNN architectures are used for the decoder and how the encoder computes the source sentence representation $\boldsymbol{s}$.

Kalchbrenner and Blunsom (2013) used an RNN with the standard hidden unit for the decoder and a convolutional neural network for encoding the source sentence representation. On the other hand, both Sutskever et al. (2014) and Luong et al. (2015c) stacked multiple layers of an RNN with a Long Short-Term Memory (LSTM) hidden unit for both the encoder and the decoder. Cho et al. (2014), Bahdanau et al. (2015), and Jean et al. (2015a) all adopted a different version of the RNN with an LSTM-inspired hidden unit, the gated recurrent unit (GRU), for both components.[3]

In more detail, one can parameterize the probability of decoding each word y_j as:

$$p\left(y_j|y_{<j}, \boldsymbol{s}\right) = \mathrm{softmax}\left(g\left(\mathbf{j}\right)\right) \tag{4.2}$$

with g being the transformation function that outputs a vocabulary-sized vector.[4] Here, $\mathbf{j}$ is

[2] All sentences are assumed to terminate with a special "end-of-sentence" token $</\mathrm{s}>$.

[3] They all used a single RNN layer except for the latter two works which utilized a bidirectional RNN for the encoder.

[4] One can provide g with other inputs such as the currently predicted word y_j as in (Bahdanau et al., 2015).

the RNN hidden unit, abstractly computed as:

$$\mathbf{j} = f(\mathbf{j\text{-}1}, \boldsymbol{s}), \tag{4.3}$$

where f computes the current hidden state given the previous hidden state and can be either a vanilla RNN unit, a GRU, or an LSTM unit. In (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014; Luong et al., 2015c), the source representation $\boldsymbol{s}$ is only used once to initialize the decoder hidden state. On the other hand, in (Bahdanau et al., 2015; Jean et al., 2015a) and this work, $\boldsymbol{s}$, in fact, implies a set of source hidden states which are consulted throughout the entire course of the translation process. Such an approach is referred to as an attention mechanism, which we will discuss next.

In this work, following (Sutskever et al., 2014; Luong et al., 2015c), we use the stacking LSTM architecture for our NMT systems, as illustrated in Figure 4.1. We use the LSTM unit defined in (**?**). Our training objective is formulated as follows:

$$J_t = \sum\nolimits_{(x,y)\in\mathbb{D}} -\log p(y|x) \tag{4.4}$$

with $\mathbb{D}$ being our parallel training corpus.

4.2 Attention-based Models

Our various attention-based models are classifed into two broad categories, *global* and *local*. These classes differ in terms of whether the "attention" is placed on all source positions or on only a few source positions. We illustrate these two model types in Figure 4.2 and 4.3 respectively.

Common to these two types of models is the fact that at each time step t in the decoding phase, both approaches first take as input the hidden state $\boldsymbol{h}_t$ at the top layer of a stacking LSTM. The goal is then to derive a context vector $\boldsymbol{c}_t$ that captures relevant source-side information to help predict the current target word y_t. While these models differ in how the context vector $\boldsymbol{c}_t$ is derived, they share the same subsequent steps.

Specifically, given the target hidden state $\boldsymbol{h}_t$ and the source-side context vector $\boldsymbol{c}_t$, we

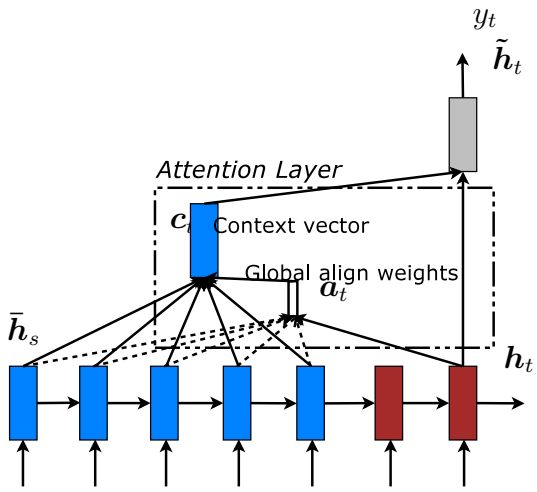

Figure 4.2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed as the weighted average, according to $\boldsymbol{a}_t$, over all the source states.

employ a simple concatenation layer to combine the information from both vectors to produce an attentional hidden state as follows:

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \tag{4.5}$$

The attentional vector $\tilde{\boldsymbol{h}}_t$ is then fed through the softmax layer to produce the predictive distribution formulated as:

$$p(y_t|y_{<t}, x) = \text{softmax}(\boldsymbol{W}_s\tilde{\boldsymbol{h}}_t) \tag{4.6}$$

We now detail how each model type computes the source-side context vector $\boldsymbol{c}_t$.

4.2.1 Global Attention

The idea of a global attentional model is to consider all the hidden states of the encoder when deriving the context vector c_t. In this model type, a variable-length alignment vector $\boldsymbol{a}_t$, whose size equals the number of time steps on the source side, is derived by comparing

the current target hidden state $\boldsymbol{h}_t$ with each source hidden state $\bar{\boldsymbol{h}}_s$:

$$\begin{aligned} \boldsymbol{a}_t(s) &= \text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \\ &= \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \end{aligned} \tag{4.7}$$

Here, score is referred as a *content-based* function for which we consider three different alternatives:

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_{\boldsymbol{a}} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_{\boldsymbol{a}}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

Besides, in our early attempts to build attention-based models, we use a *location-based* function in which the alignment scores are computed from solely the target hidden state $\boldsymbol{h}_t$ as follows:

$$\boldsymbol{a}_t = \text{softmax}(\boldsymbol{W}_a \boldsymbol{h}_t) \qquad \textit{location} \tag{4.8}$$

Given the alignment vector as weights, the context vector c_t is computed as the weighted average over all the source hidden states.[5]

Comparison to (Bahdanau et al., 2015) – While our global attention approach is similar in spirit to the model proposed by Bahdanau et al. (2015), there are several key differences which reflect how we have both simplified and generalized from the original model. First, we simply use hidden states at the top LSTM layers in both the encoder and decoder as illustrated in Figure 4.2. Bahdanau et al. (2015), on the other hand, use the concatenation of the forward and backward source hidden states in the bi-directional encoder and target hidden states in their non-stacking uni-directional decoder. Second, our computation path is simpler; we go from $\boldsymbol{h}_t \rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \tilde{\boldsymbol{h}}_t$ then make a prediction as detailed in Eq. (5.4), Eq. (4.6), and Figure 4.2. On the other hand, at any time t, Bahdanau et al. (2015) build from the previous hidden state **t-1** $\rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \boldsymbol{h}_t$, which, in turn, goes through a deep-output and a maxout layer before making predictions.[6] Lastly, Bahdanau et al. (2015) only

[5] Eq. (4.8) implies that all alignment vectors $\boldsymbol{a}_t$ are of the same length. For short sentences, we only use the top part of $\boldsymbol{a}_t$ and for long sentences, we ignore words near the end.

[6] We will refer to this difference again in Section 4.2.3.

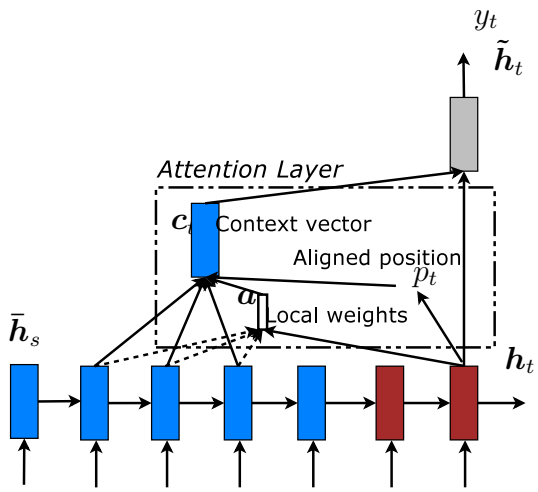

Figure 4.3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights $\boldsymbol{a_t}$ are inferred from the current target state $\boldsymbol{h}_t$ and those source states $\bar{\boldsymbol{h}}_s$ in the window.

experimented with one alignment function, the *concat* product; whereas we show later that the other alternatives are better.

4.2.2 Local Attention

The global attention has a drawback that it has to attend to all words on the source side for each target word, which is expensive and can potentially render it impractical to translate longer sequences, e.g., paragraphs or documents. To address this deficiency, we propose a *local* attentional mechanism that chooses to focus only on a small subset of the source positions per target word.

This model takes inspiration from the tradeoff between the *soft* and *hard* attentional models proposed by Xu et al. (2015) to tackle the image caption generation task. In their work, soft attention refers to the global attention approach in which weights are placed "softly" over all patches in the source image. The hard attention, on the other hand, selects one patch of the image to attend to at a time. While less expensive at inference time, the hard attention model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train.

Our local attention mechanism selectively focuses on a small window of context and is differentiable. This approach has an advantage of avoiding the expensive computation incurred in the soft attention and at the same time, is easier to train than the hard attention approach. In concrete details, the model first generates an aligned position p_t for each target word at time t. The context vector $\boldsymbol{c}_t$ is then derived as a weighted average over the set of source hidden states within the window $[p_t - D, p_t + D]$; D is empirically selected.[7] Unlike the global approach, the local alignment vector $\boldsymbol{a}_t$ is now fixed-dimensional, i.e., $\in \mathbb{R}^{2D+1}$. We consider two variants of the model as below.

Monotonic alignment (**local-m**) – we simply set $p_t = t$ assuming that source and target sequences are roughly monotonically aligned. The alignment vector $\boldsymbol{a}_t$ is defined according to Eq. (4.7).[8]

Predictive alignment (**local-p**) – instead of assuming monotonic alignments, our model predicts an aligned position as follows:

$$p_t = S \cdot \text{sigmoid}(\boldsymbol{v}_p^\top \tanh(\boldsymbol{W}_p \boldsymbol{h}_t)), \tag{4.9}$$

$\boldsymbol{W}_p$ and $\boldsymbol{v}_p$ are the model parameters which will be learned to predict positions. S is the source sentence length. As a result of sigmoid, $p_t \in [0, S]$. To favor alignment points near p_t, we place a Gaussian distribution centered around p_t . Specifically, our alignment weights are now defined as:

$$\boldsymbol{a}_t(s) = \text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right) \tag{4.10}$$

We use the same align function as in Eq. (4.7) and the standard deviation is empirically set as $\sigma = \frac{D}{2}$. Note that p_t is a *real* nummber; whereas s is an *integer* within the window centered at p_t.[9]

[7] If the window crosses the sentence boundaries, we simply ignore the outside part and consider words in the window.

[8] *local-m* is the same as the global model except that the vector $\boldsymbol{a}_t$ is fixed-length and shorter.

[9] *local-p* is similar to the local-m model except that we dynamically compute p_t and use a truncated Gaussian distribution to modify the original alignment weights $\text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)$ as shown in Eq. (4.10). By utilizing p_t to derive $\boldsymbol{a}_t$, we can compute backprop gradients for $\boldsymbol{W}_p$ and $\boldsymbol{v}_p$. This model is differentiable almost everywhere.

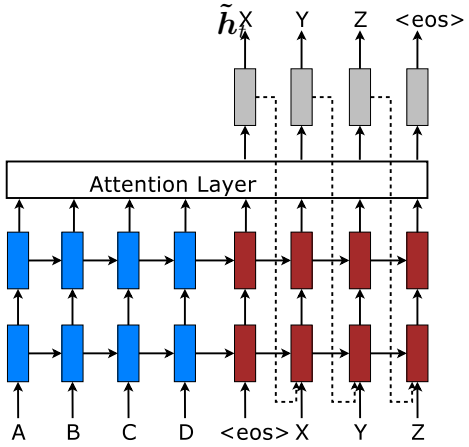

Figure 4.4: **Input-feeding approach** – Attentional vectors $\tilde{\boldsymbol{h}}_t$ are fed as inputs to the next time steps to inform the model about past alignment decisions.

Comparison to (Gregor et al., 2015) – have proposed a *selective attention* mechanism, very similar to our local attention, for the image generation task. Their approach allows the model to select an image patch of varying location and zoom. We, instead, use the same "zoom" for all target positions, which greatly simplifies the formulation and still achieves good performance.

4.2.3 Input-feeding Approach

In our proposed global and local approaches, the attentional decisions are made independently, which is suboptimal. Whereas, in standard MT, a *coverage* set is often maintained during the translation process to keep track of which source words have been translated. Likewise, in attentional NMTs, alignment decisions should be made jointly taking into account past alignment information. To address that, we propose an *input-feeding* approach in which attentional vectors $\tilde{\boldsymbol{h}}_t$ are concatenated with inputs at the next time steps as illustrated in Figure 4.4.[10] The effects of having such connections are two-fold: (a) we hope to make the model fully aware of previous alignment choices and (b) we create a very deep network spanning both horizontally and vertically.

[10]If n is the number of LSTM cells, the input size of the first LSTM layer is $2n$; those of subsequent layers are n.

Comparison to other work – Bahdanau et al. (2015) use context vectors, similar to our $\boldsymbol{c}_t$, in building subsequent hidden states, which can also achieve the "coverage" effect. However, there has not been any analysis of whether such connections are useful as done in this work. Also, our approach is more general; as illustrated in Figure 4.4, it can be applied to general stacking recurrent architectures, including non-attentional models.

Xu et al. (2015) propose a *doubly attentional* approach with an additional constraint added to the training objective to make sure the model pays equal attention to all parts of the image during the caption generation process. Such a constraint can also be useful to capture the coverage set effect in NMT that we mentioned earlier. However, we chose to use the input-feeding approach since it provides flexibility for the model to decide on any attentional constraints it deems suitable.

4.3 Experiments

We evaluate the effectiveness of our models on the WMT translation tasks between English and German in both directions. newstest2013 (3000 sentences) is used as a development set to select our hyperparameters. Translation performances are reported in case-sensitive BLEU (Papineni et al., 2002) on newstest2014 (2737 sentences) and newstest2015 (2169 sentences). Following (Luong et al., 2015c), we report translation quality using two types of BLEU: (a) *tokenized*[11] BLEU to be comparable with existing NMT work and (b) *NIST*[12] BLEU to be comparable with WMT results.

4.3.1 Training Details

All our models are trained on the WMT'14 training data consisting of 4.5M sentences pairs (116M English words, 110M German words). Similar to (Jean et al., 2015a), we limit our vocabularies to be the top 50K most frequent words for both languages. Words not in these shortlisted vocabularies are converted into a universal token `<unk>`.

[11] All texts are tokenized with `tokenizer.perl` and BLEU scores are computed with `multi-bleu.perl`.

[12] With the `mteval-v13a` script as per WMT guideline.

System	Ppl	BLEU
Winning WMT'14 system – *phrase-based + large LM* (Buck et al., 2014)		20.7
Existing NMT systems		
RNNsearch (Jean et al., 2015a)		16.5
RNNsearch + unk replace (Jean et al., 2015a)		19.0
RNNsearch + unk replace + large vocab + *ensemble* 8 models (Jean et al., 2015a)		**21.6**
Our NMT systems		
Base	10.6	11.3
Base + reverse	9.9	12.6 (+*1.3*)
Base + reverse + dropout	8.1	14.0 (+*1.4*)
Base + reverse + dropout + global attention (*location*)	7.3	16.8 (+*2.8*)
Base + reverse + dropout + global attention (*location*) + feed input	6.4	18.1 (+*1.3*)
Base + reverse + dropout + local-p attention (*general*) + feed input	5.9	19.0 (+*0.9*)
Base + reverse + dropout + local-p attention (*general*) + feed input + unk replace		20.9 (+*1.9*)
Ensemble 8 models + unk replace		**23.0** (+*2.1*)

Table 4.1: **WMT'14 English-German results** – shown are the perplexities (ppl) and the *tokenized* BLEU scores of various systems on newstest2014. We highlight the **best** system in bold and give *progressive* improvements in italic between consecutive systems. *local-p* referes to the local attention with predictive alignments. We indicate for each attention model the alignment score function used in pararentheses.

When training our NMT systems, following (Bahdanau et al., 2015; Jean et al., 2015a), we filter out sentence pairs whose lengths exceed 50 words and shuffle mini-batches as we proceed. Our stacking LSTM models have 4 layers, each with 1000 cells, and 1000-dimensional embeddings. We follow (Sutskever et al., 2014; Luong et al., 2015c) in training NMT with similar settings: (a) our parameters are uniformly initialized in $[-0.1, 0.1]$, (b) we train for 10 epochs using plain SGD, (c) a simple learning rate schedule is employed – we start with a learning rate of 1; after 5 epochs, we begin to halve the learning rate every epoch, (d) our mini-batch size is 128, and (e) the normalized gradient is rescaled whenever its norm exceeds 5. Additionally, we also use dropout with probability 0.2 for our LSTMs as suggested by (Zaremba et al., 2014). For dropout models, we train for 12 epochs and start halving the learning rate after 8 epochs. For local attention models, we empirically set the window size $D = 10$.

Our code is implemented in MATLAB. When running on a single GPU device Tesla K40, we achieve a speed of 1K *target* words per second. It takes 7–10 days to completely train a model.

4.3.2 English-German Results

We compare our NMT systems in the English-German task with various other systems. These include the winning system in WMT'14 (Buck et al., 2014), a phrase-based system whose language models were trained on a huge monolingual text, the Common Crawl corpus. For end-to-end NMT systems, to the best of our knowledge, (Jean et al., 2015a) is the only work experimenting with this language pair and currently the SOTA system. We only present results for some of our attention models and will later analyze the rest in Section 5.5.

As shown in Table 4.1, we achieve progressive improvements when (a) reversing the source sentence, +1.3 BLEU, as proposed in (Sutskever et al., 2014) and (b) using dropout, +1.4 BLEU. On top of that, (c) the global attention approach gives a significant boost of +2.8 BLEU, making our model slightly better than the base attentional system of Bahdanau et al. (2015) (row *RNNSearch*). When (d) using the *input-feeding* approach, we seize another notable gain of +1.3 BLEU and outperform their system. The local attention model with predictive alignments (row *local-p*) proves to be even better, giving us a further improvement of +0.9 BLEU on top of the global attention model. It is interesting to observe the trend previously reported in (Luong et al., 2015c) that perplexity strongly correlates with translation quality. In total, we achieve a significant gain of 5.0 BLEU points over the non-attentional baseline, which already includes known techniques such as source reversing and dropout.

The unknown replacement technique proposed in (Luong et al., 2015c; Jean et al., 2015a) yields another nice gain of +1.9 BLEU, demonstrating that our attentional models do learn useful alignments for unknown works. Finally, by ensembling 8 different models of various settings, e.g., using different attention approaches, with and without dropout etc., we were able to achieve a *new SOTA* result of 23.0 BLEU, outperforming the existing best system (Jean et al., 2015a) by +1.4 BLEU.

Latest results in WMT'15 – despite the fact that our models were trained on WMT'14 with slightly less data, we test them on newstest2015 to demonstrate that they can generalize well to different test sets. As shown in Table 4.2, our best system establishes a *new SOTA* performance of 25.9 BLEU, outperforming the existing best system backed by NMT

System	BLEU
Top – *NMT + 5-gram rerank* (Montreal)	24.9
Our ensemble 8 models + unk replace	**25.9**

Table 4.2: **WMT'15 English-German results** – *NIST* BLEU scores of the winning entry in WMT'15 and our best one on newstest2015.

System	Ppl.	BLEU
WMT'15 systems		
SOTA – *phrase-based* (Edinburgh)		**29.2**
NMT + 5-gram rerank (MILA)		27.6
Our NMT systems		
Base (reverse)	14.3	16.9
+ global (*location*)	12.7	19.1 (+2.2)
+ global (*location*) + feed	10.9	20.1 (+*1.0*)
+ global (*dot*) + drop + feed	9.7	22.8 (+*2.7*)
+ global (*dot*) + drop + feed + unk		24.9 (+*2.1*)

Table 4.3: **WMT'15 German-English results** – performances of various systems (similar to Table 4.1). The *base* system already includes source reversing on which we add *global* attention, *drop*out, input *feed*ing, and *unk* replacement.

and a 5-gram LM reranker by +1.0 BLEU.

4.3.3 German-English Results

We carry out a similar set of experiments for the WMT'15 translation task from German to English. While our systems have not yet matched the performance of the SOTA system, we nevertheless show the effectiveness of our approaches with large and progressive gains in terms of BLEU as illustrated in Table 4.3. The *attentional* mechanism gives us +2.2 BLEU gain and on top of that, we obtain another boost of up to +1.0 BLEU from the *input-feeding* approach. Using a better alignment function, the content-based *dot* product one, together with *dropout* yields another gain of +2.7 BLEU. Lastly, when applying the unknown word replacement technique, we seize an additional +2.1 BLEU, demonstrating the usefulness of attention in aligning rare words.

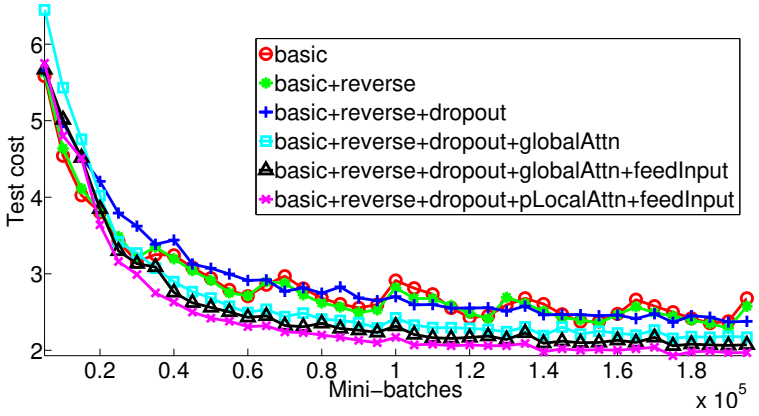

Figure 4.5: **Learning curves** – test cost (ln perplexity) on newstest2014 for English-German NMTs as training progresses.

4.4 Analysis

We conduct extensive analysis to better understand our models in terms of learning, the ability to handle long sentences, choices of attentional architectures, and alignment quality. All results reported here are on English-German newstest2014.

4.4.1 Learning curves

We compare models built on top of one another as listed in Table 4.1. It is pleasant to observe in Figure 4.5 a clear separation between non-attentional and attentional models. The input-feeding approach and the local attention model also demonstrate their abilities in driving the test costs lower. The non-attentional model with dropout (the blue + curve) learns slower than other non-dropout models, but as time goes by, it becomes more robust in terms of minimizing test errors.

4.4.2 Effects of Translating Long Sentences

We follow (Bahdanau et al., 2015) to group sentences of similar lengths together and compute a BLEU score per group. Figure 4.6 shows that our attentional models are more

effective than the non-attentional one in handling long sentences: the quality does not degrade as sentences become longer. Our best model (the blue + curve) outperforms all other systems in all length buckets.

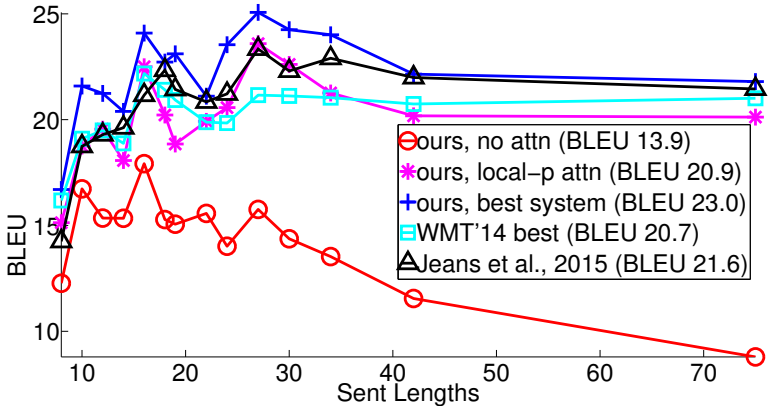

Figure 4.6: **Length Analysis** – translation qualities of different systems as sentences become longer.

4.4.3 Choices of Attentional Architectures

We examine different attention models (*global, local-m, local-p*) and different alignment functions (*location, dot, general, concat*) as described in Section 4.2. Due to limited resources, we cannot run all the possible combinations. However, results in Table 4.4 do give us some idea about different choices. The *location-based* function does not learn good alignments: the *global (location)* model can only obtain a small gain when performing unknown word replacement compared to using other alignment functions.[13] For *content-based* functions, our implementation *concat* does not yield good performances and more analysis should be done to understand the reason.[14] It is interesting to observe that *dot*

[13]There is a subtle difference in how we retrieve alignments for the different alignment functions. At time step t in which we receive y_{t-1} as input and then compute $\boldsymbol{h}_t, \boldsymbol{a}_t, \boldsymbol{c}_t$, and $\tilde{\boldsymbol{h}}_t$ before predicting y_t, the alignment vector $\boldsymbol{a}_t$ is used as alignment weights for (a) the predicted word y_t in the *location-based* alignment functions and (b) the input word y_{t-1} in the *content-based* functions.

[14]With *concat*, the perplexities achieved by different models are 6.7 (global), 7.1 (local-m), and 7.1 (local-p). Such high perplexities could be due to the fact that we simplify the matrix $\boldsymbol{W_a}$ to set the part that corresponds to $\bar{\boldsymbol{h}}_s$ to identity.

System	Ppl	BLEU	
		Before	After unk
global (location)	6.4	18.1	19.3 (+1.2)
global (dot)	6.1	18.6	20.5 (+1.9)
global (general)	6.1	17.3	19.1 (+1.8)
local-m (dot)	>7.0	x	x
local-m (general)	6.2	18.6	20.4 (+1.8)
local-p (dot)	6.6	18.0	19.6 (+1.9)
local-p (general)	**5.9**	**19**	**20.9 (+1.9)**

Table 4.4: **Attentional Architectures** – performances of different attentional models. We trained two local-m (dot) models; both have ppl > 7.0.

Method	AER
global (location)	0.39
local-m (general)	0.34
local-p (general)	0.36
ensemble	0.34
Berkeley Aligner	0.32

Table 4.5: **AER scores** – results of various models on the RWTH English-German alignment data.

works well for the global attention and *general* is better for the local attention. Among the different models, the local attention model with predictive alignments (*local-p*) is best, both in terms of perplexities and BLEU.

4.4.4 Alignment Quality

A by-product of attentional models are word alignments. While (Bahdanau et al., 2015) visualized alignments for some sample sentences and observed gains in translation quality as an indication of a working attention model, no work has assessed the alignments learned as a whole. In contrast, we set out to evaluate the alignment quality using the alignment error rate (AER) metric.

Given the gold alignment data provided by RWTH for 508 English-German Europarl sentences, we "force" decode our attentional models to produce translations that match the references. We extract only one-to-one alignments by selecting the source word with the

highest alignment weight per target word. Nevertheless, as shown in Table 4.5, we were able to achieve AER scores comparable to the one-to-many alignments obtained by the Berkeley aligner (Liang et al., 2006).[15]

We also found that the alignments produced by local attention models achieve lower AERs than those of the global one. The AER obtained by the ensemble, while good, is not better than the local-m AER, suggesting the well-known observation that AER and translation scores are not well correlated (Fraser and Marcu, 2007). Due to space constraint, we can only show alignment visualizations in the arXiv version of our paper.[16]

4.4.5 Sample Translations

We show in Table 5.4 sample translations in both directions. It it appealing to observe the effect of attentional models in correctly translating names such as "Miranda Kerr" and "Roger Dow". Non-attentional models, while producing sensible names from a language model perspective, lack the direct connections from the source side to make correct translations. We also observed an interesting case in the second example, which requires translating the *doubly-negated* phrase, "not incompatible". The attentional model correctly produces "nicht ... unvereinbar"; whereas the non-attentional model generates "nicht vereinbar", meaning "not compatible".[17] The attentional model also demonstrates its superiority in translating long sentences as in the last example.

4.5 Conclusion

In this paper, we propose two simple and effective attentional mechanisms for neural machine translation: the *global* approach which always looks at all source positions and the *local* one that only attends to a subset of source positions at a time. We test the effectiveness of our models in the WMT translation tasks between English and German in both directions. Our local attention yields large gains of up to 5.0 BLEU over non-attentional

[15] We concatenate the 508 sentence pairs with 1M sentence pairs from WMT and run the Berkeley aligner.

[16] `http://arxiv.org/abs/1508.04025`

[17] The reference uses a more fancy translation of "incompatible", which is "im Widerspruch zu etwas stehen". Both models, however, failed to translate "passenger experience".

English-German translations	
src	Orlando Bloom and Miranda Kerr still love each other
ref	Orlando Bloom und *Miranda Kerr* lieben sich noch immer
best	Orlando Bloom und *Miranda Kerr* lieben einander noch immer .
base	Orlando Bloom und **Lucas Miranda** lieben einander noch immer .
src	″ We ′ re pleased the FAA recognizes that an enjoyable passenger experience is not incompatible with safety and security , ″ said Roger Dow , CEO of the U.S. Travel Association .
ref	“ Wir freuen uns , dass die FAA erkennt , dass ein angenehmes Passagiererlebnis nicht im Widerspruch zur Sicherheit steht ” , sagte *Roger Dow* , CEO der U.S. Travel Association .
best	″ Wir freuen uns , dass die FAA anerkennt , dass ein angenehmes ist nicht mit Sicherheit und Sicherheit *unvereinbar* ist ″ , sagte *Roger Dow* , CEO der US - die .
base	″ Wir freuen uns über die <unk> , dass ein <unk> <unk> mit Sicherheit nicht **vereinbar** ist mit Sicherheit und Sicherheit ″ , sagte *Roger* **Cameron** , CEO der US - <unk> .
German-English translations	
src	In einem Interview sagte Bloom jedoch , dass er und Kerr sich noch immer lieben .
ref	However , in an interview , Bloom has said that he and *Kerr* still love each other .
best	In an interview , however , Bloom said that he and *Kerr* still love .
base	However , in an interview , Bloom said that he and **Tina** were still <unk> .
src	Wegen der von Berlin und der Europäischen Zentralbank verhängten strengen Sparpolitik in Verbindung mit der Zwangsjacke , in die die jeweilige nationale Wirtschaft durch das Festhalten an der gemeinsamen Währung genötigt wird , sind viele Menschen der Ansicht , das Projekt Europa sei zu weit gegangen
ref	The *austerity imposed by Berlin and the European Central Bank , coupled with the straitjacket* imposed on national economies through adherence to the common currency , has led many people to think Project Europe has gone too far .
best	Because of the strict *austerity measures imposed by Berlin and the European Central Bank in connection with the straitjacket* in which the respective national economy is forced to adhere to the common currency , many people believe that the European project has gone too far .
base	Because of the pressure **imposed by the European Central Bank and the Federal Central Bank with the strict austerity** imposed on the national economy in the face of the single currency , many people believe that the European project has gone too far .

Table 4.6: **Sample translations** – for each example, we show the source (*src*), the human translation (*ref*), the translation from our best model (*best*), and the translation of a non-attentional model (*base*). We italicize some *correct* translation segments and highlight a few **wrong** ones in bold.

models that already incorporate known techniques such as dropout. For the English to German translation direction, our ensemble model has established new state-of-the-art results

for both WMT'14 and WMT'15.

We have compared various alignment functions and shed light on which functions are best for which attentional models. Our analysis shows that attention-based NMT models are superior to non-attentional ones in many cases, for example in translating names and handling long sentences.

Chapter 5

Hybrid Models

Neural Machine Translation (NMT) is a simple new architecture for getting machines to translate. At its core, NMT is a single deep neural network that is trained end-to-end with several advantages such as simplicity and generalization. Despite being relatively new, NMT has already achieved state-of-the-art translation results for several language pairs such as English-French (Luong et al., 2015c), English-German (Jean et al., 2015a; Luong et al., 2015b; Luong and Manning, 2015), and English-Czech (Jean et al., 2015b).

While NMT offers many advantages over traditional phrase-based approaches, such as small memory footprint and simple decoder implementation, nearly all previous work in NMT has used quite restricted vocabularies, crudely treating all other words the same with an <unk> symbol. Sometimes, a post-processing step that patches in unknown words is introduced to alleviate this problem. Luong et al. (2015c) propose to annotate occurrences of target <unk> with positional information to track their alignments, after which simple word dictionary lookup or identity copy can be performed to replace <unk> in the translation. Jean et al. (2015a) approach the problem similarly but obtain the alignments for unknown words from the attention mechanism. We refer to these as the *unk replacement* technique.

Though simple, these approaches ignore several important properties of languages. First, *monolingually*, words are morphologically related; however, they are currently treated as independent entities. This is problematic as pointed out by Luong et al. (2013): neural networks can learn good representations for frequent words such as "distinct", but fail for

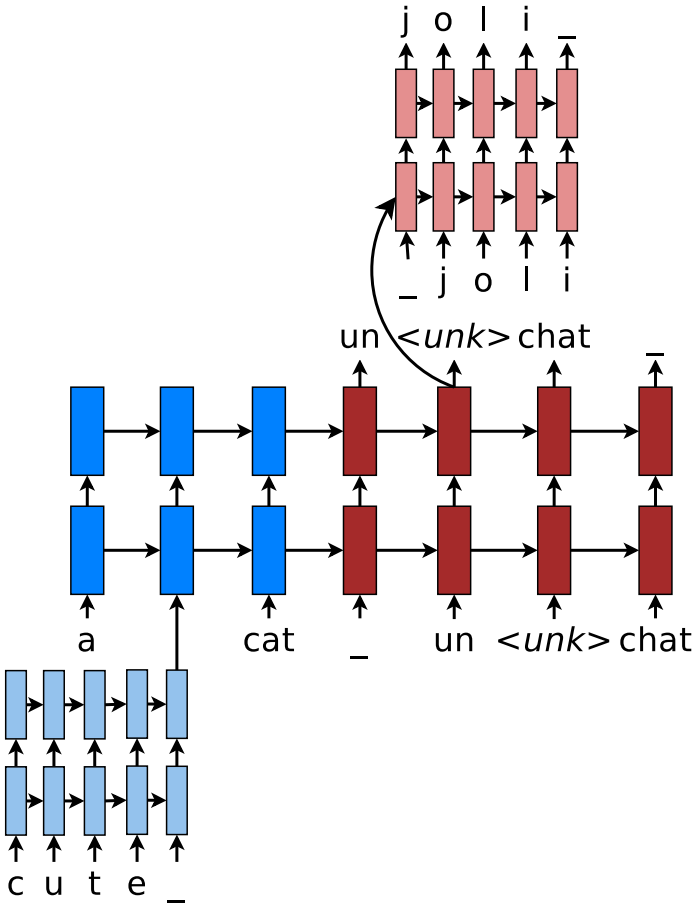

Figure 5.1: **Hybrid NMT** – example of a word-character model for translating "a cute cat" into "un joli chat". Hybrid NMT translates at the word level. For rare tokens, the character-level components build source representations and recover target <unk>. "_" marks sequence boundaries.

rare-but-related words like "distinctiveness". Second, *crosslingually*, languages have different alphabets, so one cannot naïvely memorize all possible surface word translations such as name transliteration between "Christopher" (English) and "Kryštof" (Czech). See more on this problem in (Sennrich et al., 2016).

To overcome these shortcomings, we propose a novel *hybrid* architecture for NMT that translates mostly at the word level and consults the character components for rare words when necessary. As illustrated in Figure 5.1, our hybrid model consists of a word-based NMT that performs most of the translation job, except for the two (hypothetically) rare words, "cute" and "joli", that are handled separately. On the *source* side, representations for rare words, "cute", are computed on-the-fly using a deep recurrent neural network that operates at the character level. On the *target* side, we have a separate model that recovers the surface forms, "joli", of <unk> tokens character-by-character. These components are learned jointly end-to-end, removing the need for a separate unk replacement step as in current NMT practice.

Our hybrid NMT offers a twofold advantage: it is much faster and easier to train than character-based models; at the same time, it never produces unknown words as in the case of word-based ones. We demonstrate at scale that on the WMT'15 English to Czech translation task, such a hybrid approach provides an additional boost of $+2.1-11.4$ BLEU points over models that already handle unknown words. We achieve a new state-of-the-art result with 20.7 BLEU score. Our analysis demonstrates that our character models can successfully learn to not only generate well-formed words for Czech, a highly-inflected language with a very complex vocabulary, but also build correct representations for English source words.

We provide code, data, and models at `http://nlp.stanford.edu/projects/nmt`.

5.1 Related Work

There has been a recent line of work on end-to-end character-based neural models which achieve good results for part-of-speech tagging (dos Santos and Zadrozny, 2014; Ling et al., 2015a), dependency parsing (Ballesteros et al., 2015), text classification (Zhang et al., 2015),

speech recognition (Chan et al., 2016; Bahdanau et al., 2016), and language modeling (Kim et al., 2016; Jozefowicz et al., 2016). However, success has not been shown for cross-lingual tasks such as machine translation.[1] Sennrich et al. (2016) propose to segment words into smaller units and translate just like at the word level, which does not learn to understand relationships among words.

Our work takes inspiration from (Luong et al., 2013) and (Li et al., 2015). Similar to the former, we build representations for rare words on-the-fly from subword units. However, we utilize recurrent neural networks with characters as the basic units; whereas Luong et al. (2013) use recursive neural networks with morphemes as units, which requires existence of a morphological analyzer. In comparison with (Li et al., 2015), our hybrid architecture is also a hierarchical sequence-to-sequence model, but operates at a different granularity level, word-character. In contrast, Li et al. (2015) build hierarchical models at the sentence-word level for paragraphs and documents.

5.2 Background & Our Models

Neural machine translation aims to directly model the conditional probability $p(y|x)$ of translating a source sentence, $x_1, \ldots, x_n$, to a target sentence, $y_1, \ldots, y_m$. It accomplishes this goal through an *encoder-decoder* framework (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014). The *encoder* computes a representation $\boldsymbol{s}$ for each source sentence. Based on that source representation, the *decoder* generates a translation, one target word at a time, and hence, decomposes the log conditional probability as:

$$\log p(y|x) = \sum_{t=1}^{m} \log p\left(y_t|y_{<t}, \boldsymbol{s}\right) \tag{5.1}$$

A natural model for sequential data is the recurrent neural network (RNN), used by most of the recent NMT work. Papers, however, differ in terms of: (a) architecture – from unidirectional, to bidirectional, and deep multi-layer RNNs; and (b) RNN type – which

[1] Recently, Ling et al. (2015b) attempt character-level NMT; however, the experimental evidence is weak. The authors demonstrate only small improvements over word-level baselines and acknowledge that there are no differences of significance. Furthermore, only small datasets were used without comparable results from past NMT work.

are long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and the gated recurrent unit (Cho et al., 2014). All our models utilize the *deep multi-layer* architecture with *LSTM* as the recurrent unit; detailed formulations are in (Zaremba et al., 2014).

Considering the top recurrent layer in a deep LSTM, with $\boldsymbol{h}_t$ being the current target hidden state as in Figure 5.2, one can compute the probability of decoding each target word y_t as:

$$p\left(y_t|y_{<t}, \boldsymbol{s}\right) = \text{softmax}\left(\boldsymbol{h}_t\right) \tag{5.2}$$

For a parallel corpus $\mathbb{D}$, we train our model by minimizing the below cross-entropy loss:

$$J = \sum\nolimits_{(x,y)\in\mathbb{D}} -\log p(y|x) \tag{5.3}$$

Attention Mechanism – The early NMT approaches (Sutskever et al., 2014; Cho et al., 2014), which we have described above, use only the last encoder state to initialize the decoder, i.e., setting the input representation $\boldsymbol{s}$ in Eq. (5.1) to $[\bar{\boldsymbol{h}}_n]$. Recently, Bahdanau et al. (2015) propose an *attention mechanism*, a form of random access memory for NMT to cope with long input sequences. Luong et al. (2015b) further extend the attention mechanism to different scoring functions, used to compare source and target hidden states, as well as different strategies to place the attention. In all our models, we utilize the *global* attention mechanism and the *bilinear form* for the attention scoring function similar to (Luong et al., 2015b).

Specifically, we set $\boldsymbol{s}$ in Eq. (5.1) to the set of source hidden states at the top layer, $[\bar{\boldsymbol{h}}_1, \ldots, \bar{\boldsymbol{h}}_n]$. As illustrated in Figure 5.2, the attention mechanism consists of two stages: (a) *context vector* – the current hidden state $\boldsymbol{h}_t$ is compared with individual source hidden states in $\boldsymbol{s}$ to learn an alignment vector, which is then used to compute the context vector $\boldsymbol{c}_t$ as a weighted average of $\boldsymbol{s}$; and (b) *attentional hidden state* – the context vector $\boldsymbol{c}_t$ is then used to derive a new attentional hidden state:

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \tag{5.4}$$

The attentional vector $\tilde{\boldsymbol{h}}_t$ then replaces $\boldsymbol{h}_t$ in Eq. (5.2) in predicting the next word.

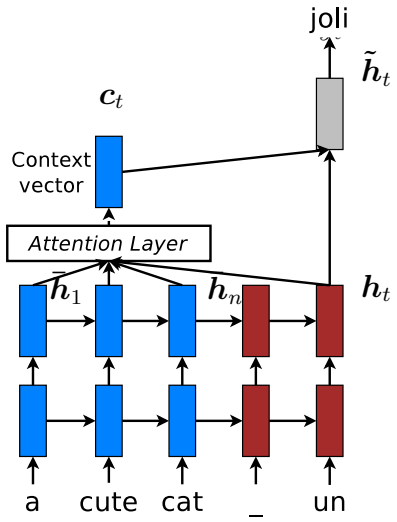

Figure 5.2: **Attention mechanism**.

5.3 Hybrid Neural Machine Translation

Our hybrid architecture, illustrated in Figure 5.1, leverages the power of both words and characters to achieve the goal of open vocabulary NMT. The core of the design is a *word*-level NMT with the advantage of being fast and easy to train. The *character* components empower the word-level system with the abilities to compute any source word representation on the fly from characters and to recover character-by-character unknown target words originally produced as <unk>.

5.3.1 Word-based Translation as a Backbone

The core of our hybrid NMT is a deep LSTM encoder-decoder that translates at the *word* level as described in Section 5.2. We maintain a vocabulary of $|V|$ frequent words for each language. Other words not inside these lists are represented by a universal symbol <unk>, one per language. We translate just like a word-based NMT system with respect to these source and target vocabularies, except for cases that involve <unk> in the source input or the target output. These correspond to the character-level components illustrated in Figure 5.1.

A nice property of our hybrid approach is that by varying the vocabulary size, one can

control how much to blend the word- and character-based models; hence, taking the best of both worlds.

5.3.2 Source Character-based Representation

In regular word-based NMT, for all rare words outside the source vocabulary, one feeds the universal embedding representing <unk> as input to the encoder. This is problematic because it discards valuable information about the source word. To fix that, we learn a deep LSTM model over characters of source words. For example, in Figure 5.1, we run our deep character-based LSTM over 'c', 'u', 't', 'e', and '_' (the boundary symbol). The final hidden state at the top layer will be used as the on-the-fly representation for the current rare word.

The layers of the deep character-based LSTM are always initialized with *zero* states. One might propose to connect hidden states of the word-based LSTM to the character-based model; however, we chose this design for various reasons. First, it simplifies the architecture. Second, it allows for efficiency through *precomputation*: before each mini-batch, we can compute representations for rare source words all at once. All instances of the same word share the same embedding, so the computation is per *type*.[2]

5.3.3 Target Character-level Generation

General word-based NMT allows generation of <unk> in the target output. Afterwards, there is usually a post-processing step that handles these unknown tokens by utilizing the alignment information derived from the attention mechanism and then performing simple word dictionary lookup or identity copy (Luong et al., 2015b; Jean et al., 2015a). While this approach works, it suffers from various problems such as alphabet mismatches between the source and target vocabularies and multi-word alignments. Our goal is to address all these issues and create a coherent framework that handles an unlimited output vocabulary.

Our solution is to have a separate deep LSTM that "translates" at the character level

[2]While Ling et al. (2015b) found that it is slow and difficult to train source character-level models and had to resort to pretraining, we demonstrate later that we can train our deep character-level LSTM perfectly fine in an end-to-end fashion.

given the current word-level state. We train our system such that whenever the word-level NMT produces an <unk>, we can consult this character-level decoder to recover the correct surface form of the unknown target word. This is illustrated in Figure 5.1. The training objective in Eq. (5.3) now becomes:

$$J = J_w + \alpha J_c \tag{5.5}$$

Here, J_w refers to the usual loss of the word-level NMT; in our example, it is the sum of the negative log likelihood of generating {"un", "<unk>", "chat", "_"}. The remaining component J_c corresponds to the loss incurred by the character-level decoder when predicting characters, e.g., {'j', 'o', 'l', 'i', '_'}, of those rare words not in the target vocabulary.

Hidden-state Initialization Unlike the source character-based representations, which are context-independent, the target character-level generation requires the current word-level context to produce meaningful translation. This brings up an important question about what can best represent the current context so as to initialize the character-level decoder. We answer this question in the context of the attention mechanism (§5.2).

The final vector $\tilde{\boldsymbol{h}}_t$, just before the softmax as shown in Figure 5.2, seems to be a good candidate to initialize the character-level decoder. The reason is that $\tilde{\boldsymbol{h}}_t$ combines information from both the context vector $\boldsymbol{c}_t$ and the top-level recurrent state $\boldsymbol{h}_t$. We refer to it later in our experiments as the *same-path* target generation approach.

On the other hand, the same-path approach worries us because all vectors $\tilde{\boldsymbol{h}}_t$ used to seed the character-level decoder might have similar values, leading to the same character sequence being produced. The reason is because $\tilde{\boldsymbol{h}}_t$ is directly used in the softmax, Eq. (5.2), to predict the same <unk>. That might pose some challenges for the model to learn useful representations that can be used to accomplish two tasks at the same time, that is to predict <unk> and to generate character sequences. To address that concern, we propose another approach called the *separate-path* target generation.

Our separate-path target generation approach works as follows. We mimic the process described in Eq. (5.4) to create a counterpart vector $\breve{\boldsymbol{h}}_t$ that will be used to seed the

character-level decoder:

$$\breve{\boldsymbol{h}}_t = \tanh(\breve{\boldsymbol{W}}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \tag{5.6}$$

Here, $\breve{\boldsymbol{W}}$ is a new learnable parameter matrix, with which we hope to release $\boldsymbol{W}$ from the pressure of having to extract information relevant to both the word- and character-generation processes. Only the hidden state of the first layer is initialized as discussed above. The other components in the character-level decoder such as the LSTM cells of all layers and the hidden states of higher layers, all start with zero values.

Implementation-wise, the computation in the character-level decoder is done per word *token* instead of per *type* as in the source character component (§5.3.2). This is because of the context-dependent nature of the decoder.

Word-Character Generation Strategy With the character-level decoder, we can view the final hidden states as representations for the surface forms of unknown tokens and could have fed these to the next time step. However, we chose not to do so for the efficiency reason explained next; instead, <unk> is fed to the word-level decoder "as is" using its corresponding word embedding.

During *training*, this design choice decouples all executions over <unk> instances of the character-level decoder as soon the word-level NMT completes. As such, the forward and backward passes of the character-level decoder over rare words can be invoked in batch mode. At *test* time, our strategy is to first run a beam search decoder at the word level to find the best translations given by the word-level NMT. Such translations contains <unk> tokens, so we utilize our character-level decoder with beam search to generate actual words for these <unk>.

5.4 Experiments

We evaluate the effectiveness of our models on the publicly available WMT'15 translation task from English into Czech with *newstest2013* (3000 sentences) as a development set and *newstest2015* (2656 sentences) as a test set. Two metrics are used: case-sensitive NIST

	English		Czech	
	word	char	word	char
# Sents	15.8M			
# Tokens	254M	1,269M	224M	1,347M
# Types	1,172K	2003	1,760K	2053
200-char	98.1%		98.8%	

Table 5.1: **WMT'15 English-Czech data** – shown are various statistics of our training data such as *sentence*, *token* (word and character counts), as well as *type* (sizes of the word and character vocabularies). We show in addition the amount of words in a vocabulary expressed by a list of 200 characters found in frequent words.

BLEU (Papineni et al., 2002) and chrF$_3$ (Popović, 2015).[3] The latter measures the amounts of overlapping character n-grams and has been argued to be a better metric for translation tasks out of English.

5.4.1 Data

Among the available language pairs in WMT'15, all involving English, we choose *Czech* as a target language for several reasons. First and foremost, Czech is a Slavic language with not only rich and complex inflection, but also fusional morphology in which a single morpheme can encode multiple grammatical, syntactic, or semantic meanings. As a result, Czech possesses an enormously large vocabulary (about 1.5 to 2 times bigger than that of English according to statistics in Table 5.1) and is a challenging language to translate into. Furthermore, this language pair has a large amount of training data, so we can evaluate at scale. Lastly, though our techniques are language independent, it is easier for us to work with Czech since Czech uses the Latin alphabet with some diacritics.

In terms of preprocessing, we apply only the standard tokenization practice.[4] We choose for each language a list of 200 characters found in frequent words, which, as shown in Table 5.1, can represent more than 98% of the vocabulary.

[3]For NIST BLEU, we first run `detokenizer.pl` and then use `mteval-v13a` to compute the scores as per WMT guideline. For chrF$_3$, we utilize the implementation here `https://github.com/rsennrich/subword-nmt`.

[4]Use `tokenizer.perl` in Moses with default settings.

	System	Vocab	Perplexity		BLEU	chrF$_3$
			w	c		
(a)	Best WMT'15, big data (Bojar and Tamchyna, 2015)	-	-	-	i18.8	-
	Existing NMT					
(b)	RNNsearch + unk replace (Jean et al., 2015b)	200K	-	-	15.7	-
(c)	iEnsemble 4 models + unk replace (Jean et al., 2015b)	200K	-	-	18.3	-
	Our *word-based* NMT					
(d)	Base + attention + unk replace	50K	5.9	-	17.5	42.4
(e)	iEnsemble 4 models + unk replace	50K	-	-	18.4	43.9
	Our *character-based* NMT					
(f)	Base-512 (600-step backprop)	200	-	2.4	3.8	25.9
(g)	Base-512 + attention (600-step backprop)	200	-	1.6	17.5	i46.6
(h)	Base-1024 + attention (300-step backprop)	200	-	1.9	15.7	41.1
	Our *hybrid* NMT					
(i)	Base + attention + same-path	10K	4.9	1.7	14.1	37.2
(j)	Base + attention + separate-path	10K	4.9	1.7	15.6	39.6
(k)	Base + attention + separate-path + 2-layer char	10K	4.7	1.6	i17.7	44.1
(l)	Base + attention + separate-path + 2-layer char	50K	5.7	1.6	19.6	46.5
(m)	iEnsemble 4 models	50K	-	-	**20.7**	**47.5**

Table 5.2: **WMT'15 English-Czech results** – shown are the vocabulary sizes, perplexities, BLEU, and chrF$_3$ scores of various systems on *newstest2015*. Perplexities are listed under two categories, word (w) and character (c). **Best** and iimportant results per metric are highlighed.

5.4.2 Training Details

We train three types of systems, purely *word-based*, purely *character-based*, and *hybrid*. Common to these architectures is a word-based NMT since the character-based systems are essentially word-based ones with longer sequences and the core of hybrid models is also a word-based NMT.

In training word-based NMT, we follow Luong et al. (2015b) to use the global attention mechanism together with similar hyperparameters: (a) deep LSTM models, 4 layers, 1024 cells, and 1024-dimensional embeddings, (b) uniform initialization of parameters in $[-0.1, 0.1]$, (c) 6-epoch training with plain SGD and a simple learning rate schedule – start with a learning rate of 1.0; after 4 epochs, halve the learning rate every 0.5 epoch, (d) mini-batches are of size 128 and shuffled, (e) the gradient is rescaled whenever its norm exceeds 5, and (f) dropout is used with probability 0.2 according to (Pham et al., 2014). We now detail differences across the three architectures.

Word-based NMT – We constrain our source and target sequences to have a maximum length of 50 each; words that go past the boundary are ignored. The vocabularies are limited to the top $|V|$ most frequent words in both languages. Words not in these vocabularies are converted into $<\texttt{unk}>$. After translating, we will perform dictionary[5] lookup or identity copy for $<\texttt{unk}>$ using the alignment information from the attention models. Such procedure is referred as the *unk replace* technique (Luong et al., 2015c; Jean et al., 2015a).

Character-based NMT – The source and target sequences at the character level are often about 5 times longer than their counterparts in the word-based models as we can infer from the statistics in Table 5.1. Due to memory constraint in GPUs, we limit our source and target sequences to a maximum length of 150 each, i.e., we backpropagate through at most 300 timesteps from the decoder to the encoder. With smaller 512-dimensional models, we can afford to have longer sequences with up to 600-step backpropagation.

Hybrid NMT – The *word*-level component uses the same settings as the purely word-based NMT. For the *character*-level source and target components, we experiment with both shallow and deep 1024-dimensional models of 1 and 2 LSTM layers. We set the weight α in Eq. (5.5) for our character-level loss to 1.0.

[5]Obtained from the alignment links produced by the Berkeley aligner (Liang et al., 2006) over the training corpus.

Training Time – It takes about 3 weeks to train a word-based model with $|V| = 50K$ and about 3 months to train a character-based model. Training and testing for the hybrid models are about 10-20% slower than those of the word-based models with the same vocabulary size.

5.4.3 Results

We compare our models with several strong systems. These include the winning entry in WMT'15, which was trained on a much larger amount of data, 52.6M parallel and 393.0M monolingual sentences (Bojar and Tamchyna, 2015).[6] In contrast, we merely use the provided parallel corpus of 15.8M sentences. For NMT, to the best of our knowledge, (Jean et al., 2015b) has the best published performance on English-Czech.

As shown in Table 5.2, for a purely *word-based* approach, our single NMT model outperforms the best single model in (Jean et al., 2015b) by +1.8 points despite using a smaller vocabulary of only 50K words versus 200K words. Our ensemble system *(e)* slightly outperforms the best previous NMT system with 18.4 BLEU.

To our surprise, purely *character-based* models, though extremely slow to train and test, perform quite well. The 512-dimensional attention-based model *(g)* is best, surpassing the single word-based model in (Jean et al., 2015b) despite having much fewer parameters. It even outperforms most NMT systems on chrF$_3$ with 46.6 points. This indicates that this model translate words that closely but not exactly match the reference ones as evidenced in Section 5.5.3. We notice two interesting observations. First, attention is critical for character-based models to work as is obvious from the poor performance of the non-attentional model; this has also been shown in speech recognition (Chan et al., 2016). Second, long time-step backpropagation is more important as reflected by the fact that the larger 1024-dimensional model *(h)* with shorter backprogration is inferior to *(g)*.

Our *hybrid* models achieve the best results. At 10K words, we demonstrate that our *separate-path* strategy for the character-level target generation (§5.3.3) is effective, yielding an improvement of +1.5 BLEU points when comparing systems *(j)* vs. *(i)*. A *deeper*

[6]This entry combines two independent systems, a phrase-based Moses model and a deep-syntactic transfer-based model. Additionally, there is an automatic post-editing system with hand-crafted rules to correct errors in morphological agreement and semantic meanings, e.g., loss of negation.

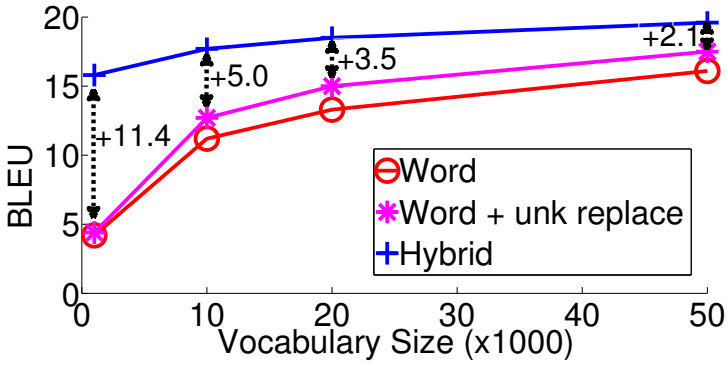

Figure 5.3: **Vocabulary size effect** – shown are the performances of different systems as we vary their vocabulary sizes. We highlight the improvements obtained by our hybrid models over word-based systems which already handle unknown words.

character-level architecture of 2 LSTM layers provides another significant boost of +2.1 BLEU. With 17.7 BLEU points, our hybrid system *(k)* has surpassed word-level NMT models.

When extending to 50K words, we further improve the translation quality. Our best single model, system *(l)* with 19.6 BLEU, is already better than all existing systems. Our ensemble model *(m)* further advances the SOTA result to i20.7 BLEU, outperforming the winning entry in the WMT'15 English-Czech translation task by a large margin of +1.9 points. Our ensemble model is also best in terms of chrF$_3$ with i47.5 points.

5.5 Analysis

This section first studies the effects of vocabulary sizes towards translation quality. We then analyze more carefully our character-level components by visualizing and evaluating rare word embeddings as well as examining sample translations.

5.5.1 Effects of Vocabulary Sizes

As shown in Figure 5.3, our hybrid models offer large gains of +2.1-11.4 BLEU points over strong word-based systems which already handle unknown words. With only a small vocabulary, e.g., 1000 words, our hybrid approach can produce systems that are better than

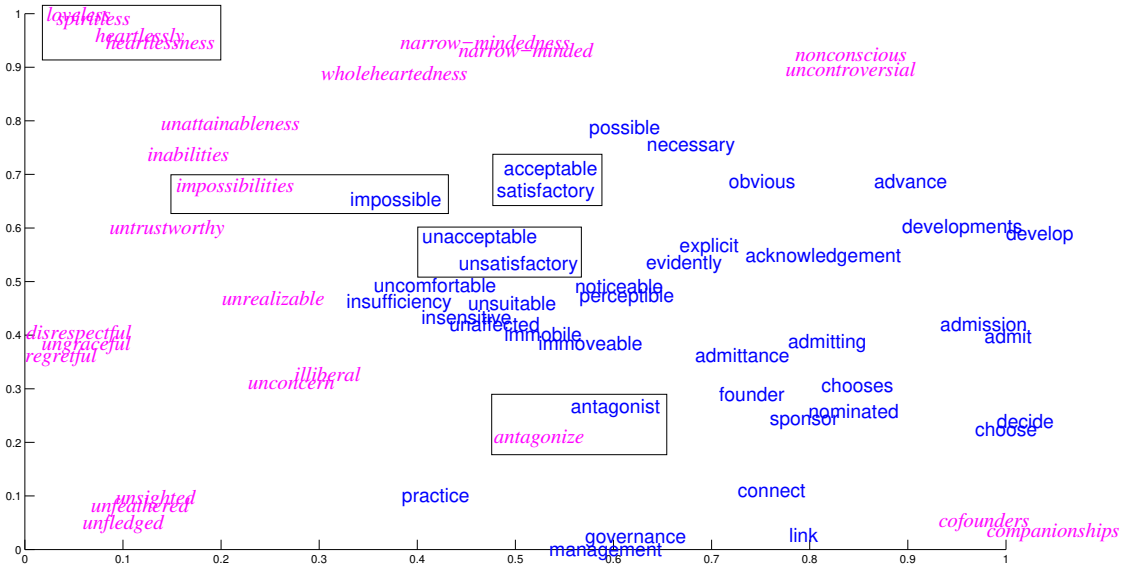

Figure 5.4: **Barnes-Hut-SNE visualization of source word representations** – shown are sample words from the *Rare Word* dataset. We differentiate two types of embeddings: frequent words in which encoder embeddings are looked up directly and *rare* words where we build representations from characters. Boxes highlight examples that we will discuss in the text. We use the hybrid model *(l)* in this visualization.

word-based models that possess much larger vocabularies. While it appears from the plot that gains diminish as we increase the vocabulary size, we argue that our hybrid models are still preferable since they understand word structures and can handle new complex words at test time as illustrated in Section 5.5.3.

5.5.2 Rare Word Embeddings

We evaluate the *source* character-level model by building representations for rare words and measuring how good these embeddings are.

Quantitatively, we follow Luong et al. (2013) in using the word similarity task, specifically on the *Rare Word* dataset, to judge the learned representations for complex words. The evaluation metric is the Spearman's correlation ρ between similarity scores assigned by a model and by human annotators. From the results in Table 5.3, we can see that source representations produced by our hybrid[7] models are significantly better than those of the word-based one. It is noteworthy that our deep recurrent character-level models can outperform the model of (Luong et al., 2013), which uses recursive neural networks and requires a complex morphological analyzer, by a large margin. Our performance is also competitive to the best Glove embeddings (Pennington et al., 2014) which were trained on a much larger dataset.

System		Size	$\|V\|$	ρ
(Luong et al., 2013)		1B	138K	34.4
Glove (Pennington et al., 2014)		6B	400K	38.1
		42B	400K	**47.8**
Our NMT models				
(d)	Word-based	0.3B	50K	20.4
(k)	Hybrid	0.3B	10K	42.4
(l)	Hybrid	0.3B	50K	**i**47.1

Table 5.3: **Word similarity task** – shown are Spearman's correlation ρ on the *Rare Word* dataset of various models (with different vocab sizes $|V|$).

Qualitatively, we visualize embeddings produced by the hybrid model *(l)* for selected

[7]We look up the encoder embeddings for frequent words and build representations for rare word from characters.

words in the Rare Word dataset. Figure 5.4 shows the two-dimensional representations of words computed by the Barnes-Hut-SNE algorithm (van der Maaten, 2013).[8] It is extremely interesting to observe that words are clustered together not only by the word structures but also by the meanings. For example, in the top-left box, the *character*-based representations for "loveless", "spiritless", "heartlessly", and "heartlessness" are nearby, but clearly separated into two groups. Similarly, in the center boxes, *word*-based embeddings of "acceptable", "satisfactory", "unacceptable", and "unsatisfactory", are close by but separated by meanings. Lastly, the remaining boxes demonstrate that our character-level models are able to build representations comparable to the word-based ones, e.g., "impossibilities" vs. "impossible" and "antagonize" vs. "antagonist". All of this evidence strongly supports that the source character-level models are useful and effective.

5.5.3 Sample Translations

We show in Table 5.4 sample translations between various systems. In the first example, our hybrid model translates perfectly. The word-based model fails to translate "diagnosis" because the second <unk> was incorrectly aligned to the word "after". The character-based model, on the other hand, makes a mistake in translating names.

For the second example, the hybrid model surprises us when it can capture the long-distance reordering of "fifty years ago" and "před padesáti lety" while the other two models do not. The word-based model translates "Jr." inaccurately due to the incorrect alignment between the second <unk> and the word "said". The character-based model literally translates the name "King" into "král" which means "king".

Lastly, both the character-based and hybrid models impress us by their ability to translate compound words exactly, e.g., "11-year-old" and "jedenáctiletá"; whereas the identity copy strategy of the word-based model fails. Of course, our hybrid model does make mistakes, e.g., it fails to translate the name "Shani Bart". Overall, these examples highlight how challenging translating into Czech is and that being able to translate at the character level helps improve the quality.

[8] We run Barnes-Hut-SNE algorithm over a set of 91 words, but filter out 27 words for displaying clarity.

1	source	The author ***Stephen Jay Gould*** died 20 years after ***diagnosis*** .
	human	Autor *Stephen Jay Gould* zemřel 20 let po *diagnóze* .
	word	Autor Stephen Jay <unk> zemřel 20 let po <unk> . Autor *Stephen Jay Gould* zemřel 20 let po **po** .
	char	Autor **Stepher Stepher** zemřel 20 let po *diagnóze* .
	hybrid	Autor <unk> <unk> <unk> zemřel 20 let po <unk>. Autor *Stephen Jay Gould* zemřel 20 let po *diagnóze* .
2	source	As the Reverend ***Martin Luther King Jr.*** said ***fifty years ago*** :
	human	Jak *před padesáti lety* řekl reverend *Martin Luther King Jr* . :
	word	Jak řekl reverend Martin <unk> King <unk> před padesáti lety : Jak řekl reverend *Martin Luther King* **řekl** před padesáti lety :
	char	Jako reverend *Martin Luther* **král říkal** před padesáti lety :
	hybrid	Jak před <unk> lety řekl <unk> Martin <unk> <unk> <unk> : Jak *před padesáti lety* řekl reverend *Martin Luther King* Jr. :
3	source	Her ***11-year-old*** daughter , ***Shani Bart*** , said it felt a " little bit ***weird*** " [..] back to school .
	human	Její *jedenáctiletá* dcera *Shani Bartová* prozradila , že " je to trochu *zvláštní* " [..] znova do školy .
	word	Její <unk> dcera <unk> <unk> řekla , že je to " trochu divné " , [..] vrací do školy . Její **11-year-old** dcera *Shani* **,** řekla , že je to " trochu divné " , [..] vrací do školy .
	char	Její *jedenáctiletá* dcera , *Shani Bartová* , říkala , že cítí trochu divně , [..] vrátila do školy .
	hybrid	Její <unk> dcera , <unk> <unk> , řekla , že cítí " trochu <unk> " , [..] vrátila do školy . Její *jedenáctiletá* dcera , **Graham** Bart , řekla , že cítí " trochu divný " , [..] vrátila do školy .

Table 5.4: **Sample translations on newstest2015** – for each example, we show the *source*, *human* translation, and translations of the following NMT systems: *word* model *(d)*, *char* model *(g)*, and *hybrid* model *(k)*. We show the translations before replacing <unk> tokens (if any) for the word-based and hybrid models. The following formats are used to highlight *correct*, **wrong**, and close translation segments.

5.6 Conclusion

We have proposed a novel *hybrid* architecture that combines the strength of both word- and character-based models. Word-level models are fast to train and offer high-quality translation; whereas, character-level models help achieve the goal of open vocabulary NMT. We have demonstrated these two aspects through our experimental results and translation examples.

Our best hybrid model has surpassed the performance of both the best word-based NMT system and the best non-neural model to establish a new state-of-the-art result for English-Czech translation in WMT'15 with 20.7 BLEU. Moreover, we have succeeded in replacing the standard unk replacement technique in NMT with our character-level components, yielding an improvement of $+2.1-11.4$ BLEU points. Our analysis has shown that our model has the ability to not only generate well-formed words for Czech, a highly inflected language with an enormous and complex vocabulary, but also build accurate representations for English source words.

Additionally, we have demonstrated the potential of purely character-based models in producing good translations; they have outperformed past word-level NMT models. For future work, we hope to be able to improve the memory usage and speed of purely character-based models.

Chapter 6

NMT Future

In previous chapters, my effort to improving neural machine translation has been centered around enhancing the model architecture to address different needs such as translating long sentences or coping with complex vocabularies. In this chapter, I switch gear to examine more "external" aspects of NMT, which is also a way for me to take a quick peek into the future of NMT. Specifically, I first examine in Section 6.1 how NMT can be improved by utilizing data from not only the translation but also other tasks such as parsing, image caption generation, and unsupervised learning. This is framed under the multi-task setting which I believe is important for NMT future given a humongous amount of data available in the world growing at an exponentially fast pace. The second aspect that I examine is on making NMT models smaller, a topic of increasing importance as mobile devices become dominant. Specifically, in Section 6.2, I cast such desiderata as a model compression problem in which I answer how much we can reduce the sizes of NMT models without sacrifice in performance and reveal interesting patterns in the parameter space of NMT. Lastly, in Section 6.3, I highlight other future trends and potential research directions for NMT.

6.1 Multi-task Sequence to Sequence Learning

Multi-task learning (MTL) is an important machine learning paradigm that aims at improving the generalization performance of a task using other related tasks. Such framework has been widely studied by Thrun (1996); Caruana (1997); Evgeniou and Pontil (2004);

Ando and Zhang (2005); Argyriou et al. (2007); Kumar and III (2012), among many others. In the context of deep neural networks, MTL has been applied successfully to various problems ranging from language (Liu et al., 2015), to vision (Donahue et al., 2014), and speech (Heigold et al., 2013; Huang et al., 2013).

Recently, sequence to sequence (*seq2seq*) learning, proposed by Kalchbrenner and Blunsom (2013), Sutskever et al. (2014), and Cho et al. (2014), emerges as an effective paradigm for dealing with variable-length inputs and outputs. *seq2seq* learning, at its core, uses recurrent neural networks to map variable-length input sequences to variable-length output sequences. While relatively new, the *seq2seq* approach has achieved state-of-the-art results in not only its original application – machine translation – (Luong et al., 2015c; Jean et al., 2015a; Luong et al., 2015b; Jean et al., 2015b; Luong and Manning, 2015), but also image caption generation (Vinyals et al., 2015b), and constituency parsing (Vinyals et al., 2015a).

Despite the popularity of multi-task learning and sequence to sequence learning, there has been little work in combining MTL with *seq2seq* learning. To the best of our knowledge, there is only one recent publication by Dong et al. (2015) which applies a *seq2seq* models for machine translation, where the goal is to translate from one language to multiple languages. In this work, we propose three MTL approaches that complement one another: (a) the *one-to-many* approach – for tasks that can have an encoder in common, such as translation and parsing; this applies to the multi-target translation setting in (Dong et al., 2015) as well, (b) the *many-to-one* approach – useful for multi-source translation or tasks in which only the decoder can be easily shared, such as translation and image captioning, and lastly, (c) the *many-to-many* approach – which share multiple encoders and decoders through which we study the effect of unsupervised learning in translation. We show that syntactic parsing and image caption generation improves the translation quality between English and German by up to +1.5 BLEU points over strong single-task baselines on the WMT benchmarks. Furthermore, we have established a new *state-of-the-art* result in constituent parsing with 93.0 F_1. We also explore two unsupervised learning objectives, sequence autoencoders (Dai and Le, 2015) and skip-thought vectors (Kiros et al., 2015), and reveal their interesting properties in the MTL setting: autoencoder helps less in terms of perplexities but more on BLEU scores compared to skip-thought.

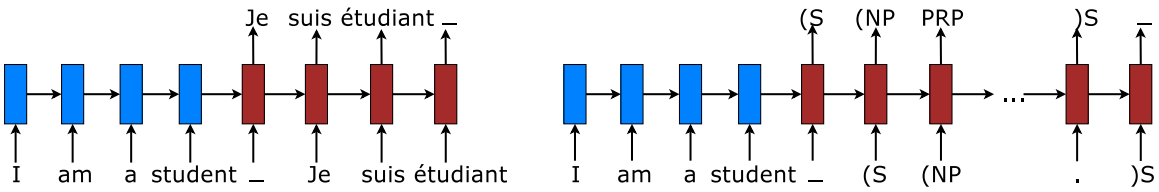

Figure 6.1: **Sequence to sequence learning examples** – (left) machine translation (Sutskever et al., 2014) and (*right*) constituent parsing (Vinyals et al., 2015a).

6.1.1 Multi-task Sequence-to-Sequence Learning

We generalize the work of Dong et al. (2015) to the multi-task sequence-to-sequence learning setting that includes the tasks of machine translation (MT), constituency parsing, and image caption generation. Depending which tasks involved, we propose to categorize multi-task *seq2seq* learning into three general settings. In addition, we will discuss the unsupervised learning tasks considered as well as the learning process.

One-to-Many Setting This scheme involves *one encoder* and *multiple decoders* for tasks in which the encoder can be shared, as illustrated in Figure 6.2. The input to each task is a sequence of English words. A separate decoder is used to generate each sequence of output units which can be either (a) a sequence of tags for constituency parsing as used in (Vinyals et al., 2015a), (b) a sequence of German words for machine translation (Luong et al., 2015b), and (c) the same sequence of English words for autoencoders or a related sequence of English words for the skip-thought objective (Kiros et al., 2015).

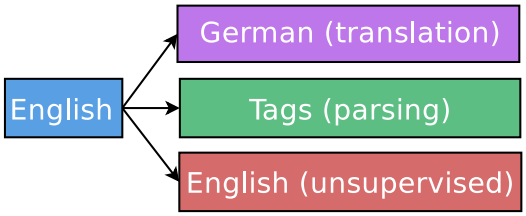

Figure 6.2: **One-to-many Setting** – one encoder, multiple decoders. This scheme is useful for either multi-target translation as in (Dong et al., 2015) or between different tasks. Here, English and German imply sequences of words in the respective languages.

Many-to-One Setting This scheme is the opposite of the *one-to-many* setting. As illustrated in Figure 6.3, it consists of *multiple encoders* and *one decoder*. This is useful for tasks in which only the decoder can be shared, for example, when our tasks include machine translation and image caption generation (Vinyals et al., 2015b). In addition, from a machine translation perspective, this setting can benefit from a large amount of monolingual data on the target side, which is a standard practice in machine translation system and has also been explored for neural MT by (Gulcehre et al., 2015).

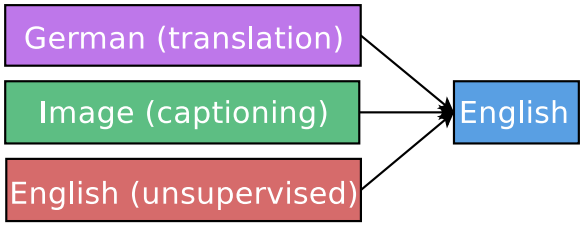

Figure 6.3: **Many-to-one setting** – multiple encoders, one decoder. This scheme is handy for tasks in which only the decoders can be shared.

Many-to-Many Setting Lastly, as the name describes, this category is the most general one, consisting of multiple encoders and multiple decoders. We will explore this scheme in a translation setting that involves sharing multiple encoders and multiple decoders. In addition to the machine translation task, we will include two unsupervised objectives over the source and target languages as illustrated in Figure 6.4.

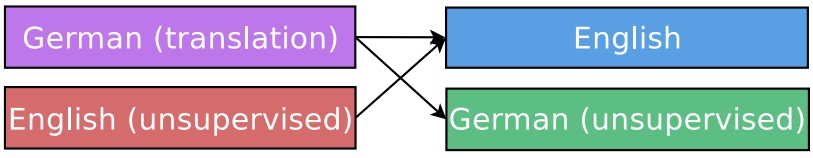

Figure 6.4: **Many-to-many setting** – multiple encoders, multiple decoders. We consider this scheme in a limited context of machine translation to utilize the large monolingual corpora in both the source and the target languages. Here, we consider a single translation task and two unsupervised autoencoder tasks.

Unsupervised Learning Tasks Our very first unsupervised learning task involves learning *autoencoders* from monolingual corpora, which has recently been applied to sequence

to sequence learning (Dai and Le, 2015). However, in Dai and Le (2015)'s work, the authors only experiment with pretraining and then finetuning, but not joint training which can be viewed as a form of multi-task learning (MTL). As such, we are very interested in knowing whether the same trend extends to our MTL settings.

Additionally, we investigate the use of the *skip-thought* vectors (Kiros et al., 2015) in the context of our MTL framework. Skip-thought vectors are trained by training sequence to sequence models on pairs of consecutive sentences, which makes the skip-thought objective a natural *seq2seq* learning candidate. A minor technical difficulty with skip-thought objective is that the training data must consist of ordered sentences, e.g., paragraphs. Unfortunately, in many applications that include machine translation, we only have sentence-level data where the sentences are unordered. To address that, we split each sentence into two halves; we then use one half to predict the other half.

Learning (Dong et al., 2015) adopted an *alternating* training approach, where they optimize each task for a fixed number of parameter updates (or mini-batches) before switching to the next task (which is a different language pair). In our setting, our tasks are more diverse and contain different amounts of training data. As a result, we allocate different numbers of parameter updates for each task, which are expressed with the *mixing* ratio values α_i (for each task i). Each parameter update consists of training data from one task only. When switching between tasks, we select randomly a new task i with probability $\frac{\alpha_i}{\sum_j \alpha_j}$.

Our convention is that the first task is the *reference* task with $\alpha_1 = 1.0$ and the number of training parameter updates for that task is prespecified to be N. A typical task i will then be trained for $\frac{\alpha_i}{\alpha_1} \cdot N$ parameter updates. Such convention makes it easier for us to fairly compare the same reference task in a single-task setting which has also been trained for exactly N parameter updates.

When sharing an encoder or a decoder, we share both the recurrent connections and the corresponding embeddings.

6.1.2 Experiments

We evaluate the multi-task learning setup on a wide variety of sequence-to-sequence tasks: constituency parsing, image caption generation, machine translation, and a number of unsupervised learning as summarized in Table 6.1.

Data Our experiments are centered around the *translation* task, where we aim to determine whether other tasks can improve translation and vice versa. We use the WMT'15 data (Bojar et al., 2015) for the English⇆German translation problem. Following Luong et al. (2015b), we use the 50K most frequent words for each language from the training corpus.[1] These vocabularies are then shared with other tasks, except for parsing in which the target "language" has a vocabulary of 104 tags. We use newstest2013 (3000 sentences) as a validation set to select our hyperparameters, e.g., mixing coefficients. For testing, to be comparable with existing results in (Luong et al., 2015b), we use the filtered newstest2014 (2737 sentences)[2] for the English→German translation task and newstest2015 (2169 sentences)[3] for the German→English task. See the summary in Table 6.1.

For the *unsupervised* tasks, we use the English and German monolingual corpora from WMT'15.[4] Since in our experiments, unsupervised tasks are always coupled with translation tasks, we use the same validation and test sets as the accompanied translation tasks.

For *constituency parsing*, we experiment with two types of corpora:

1. a small corpus – the widely used Penn Tree Bank (PTB) dataset (Marcus et al., 1993) and,

2. a large corpus – the high-confidence (HC) parse trees provided by Vinyals et al. (2015a).

The two parsing tasks, however, are evaluated on the same validation (section 22) and test (section 23) sets from the PTB data. Note also that the parse trees have been linearized

[1]The corpus has already been tokenized using the default tokenizer from Moses. Words not in these vocabularies are represented by the token <unk>.

[2]http://statmt.org/wmt14/test-filtered.tgz

[3]http://statmt.org/wmt15/test.tgz

[4]The training sizes reported for the unsupervised tasks are only 10% of the original WMT'15 monolingual corpora which we randomly sample from. Such reduced sizes are for faster training time and already about three times larger than that of the parallel data. We consider using all the monolingual data in future work.

Task	Train Size	Valid Size	Test Size	Vocab Size		Train Epoch	Finetune	
				Source	Target		Start	Cycle
English→German Translation	4.5M	3000	3003	50K	50K	12	8	1
German→English Translation	4.5M	3000	2169	50K	50K	12	8	1
English unsupervised	12.1M	Details in text		50K	50K	6	4	0.5
German unsupervised	13.8M			50K	50K	6	4	0.5
Penn Tree Bank Parsing	40K	1700	2416	50K	104	40	20	4
High-Confidence Corpus Parsing	11.0M	1700	2416	50K	104	6	4	0.5
Image Captioning	596K	4115	-	-	50K	10	5	1

Table 6.1: **Data & Training Details** – Information about the different datasets used in this work. For each task, we display the following statistics: (a) the number of training examples, (b) the sizes of the vocabulary, (c) the number of training epochs, and (d) details on when and how frequent we halve the learning rates (*finetuning*).

following Vinyals et al. (2015a). Lastly, for *image caption generation*, we use a dataset of image and caption pairs provided by Vinyals et al. (2015b).

Training Details In all experiments, following Sutskever et al. (2014) and Luong et al. (2015c), we train deep LSTM models as follows: (a) we use 4 LSTM layers each of which has 1000-dimensional cells and embeddings,[5] (b) parameters are uniformly initialized in [-0.06, 0.06], (c) we use a mini-batch size of 128, (d) dropout is applied with probability of 0.2 over vertical connections (Pham et al., 2014), (e) we use SGD with a fixed learning rate of 0.7, (f) input sequences are reversed, and lastly, (g) we use a simple finetuning schedule – after x epochs, we halve the learning rate every y epochs. The values x and y are referred as *finetune start* and *finetune cycle* in Table 6.1 together with the number of training epochs per task.

As described in Section 6.1.1, for each multi-task experiment, we need to choose one task to be the *reference task* (which corresponds to $\alpha_1 = 1$). The choice of the reference task helps specify the number of training epochs and the finetune start/cycle values which we also when training that reference task alone for fair comparison. To make sure our findings are reliable, we run each experimental configuration twice and report the average performance in the format *mean (stddev)*.

[5]For image caption generation, we use 1024 dimensions, which is also the size of the image embeddings.

Results

We explore several multi-task learning scenarios by combining a *large* task (machine translation) with: (a) a *small* task – Penn Tree Bank (PTB) parsing, (b) a *medium-sized* task – image caption generation, (c) another *large* task – parsing on the high-confidence (HC) corpus, and (d) lastly, *unsupervised tasks*, such as autoencoders and skip-thought vectors. In terms of evaluation metrics, we report both validation and test perplexities for all tasks. Additionally, we also compute test BLEU scores (Papineni et al., 2002) for the translation task.

Large Tasks with Small Tasks In this setting, we want to understand if a small task such as *PTB parsing* can help improve the performance of a large task such as translation. Since the parsing task maps from a sequence of English words to a sequence of parsing tags (Vinyals et al., 2015a), only the encoder can be shared with an English→German translation task. As a result, this is a *one-to-many* MTL scenario (§6.1.1).

To our surprise, the results in Table 6.2 suggest that by adding a very small number of parsing mini-batches (with mixing ratio 0.01, i.e., one parsing mini-batch per 100 translation mini-batches), we can improve the translation quality substantially. More concretely, our best multi-task model yields a gain of +1.5 BLEU points over the single-task baseline. It is worth pointing out that as shown in Table 6.2, our single-task baseline is very strong, even better than the equivalent non-attention model reported in (Luong et al., 2015b). Larger mixing coefficients, however, overfit the small PTB corpus; hence, achieve smaller gains in translation quality.

For parsing, as Vinyals et al. (2015a) have shown that attention is crucial to achieve good parsing performance when training on the small PTB corpus, we do not set a high bar for our attention-free systems in this setup (better performances are reported in Section 6.1.2). Nevertheless, the parsing results in Table 6.2 indicate that MTL is also beneficial for parsing, yielding an improvement of up to +8.9 F_1 points over the baseline.[6] It would be interesting to study how MTL can be useful with the presence of the *attention*

[6]While perplexities correlate well with BLEU scores as shown in (Luong et al., 2015c), we observe empirically in Section 6.1.2 that parsing perplexities are only reliable if it is less than 1.3. Hence, we omit parsing perplexities in Table 6.2 for clarity. The parsing test perplexities (averaged over two runs) for the last four rows in Table 6.2 are 1.95, 3.05, 2.14, and 1.66. Valid perplexities are similar.

mechanism, which we leave for future work.

Task	Translation			Parsing
	Valid ppl	Test ppl	Test BLEU	Test F_1
(Luong et al., 2015b)	-	8.1	14.0	-
Our single-task systems				
Translation	8.8 (0.3)	8.3 (0.2)	14.3 (0.3)	-
PTB Parsing	-	-	-	43.3 (1.7)
Our multi-task systems				
Translation + PTB Parsing (1x)	8.5 (0.0)	8.2 (0.0)	14.7 (0.1)	54.5 (0.4)
Translation + PTB Parsing (0.1x)	8.3 (0.1)	7.9 (0.0)	15.1 (0.0)	**55.2 (0.0)**
Translation + PTB Parsing (0.01x)	**8.2** (0.2)	**7.7** (0.2)	**15.8** (0.4)	39.8 (2.7)

Table 6.2: **English→German WMT'14 translation & Penn Tree Bank parsing results** – shown are perplexities (ppl), BLEU scores, and parsing F_1 for various systems. For muli-task models, *reference* tasks are in italic with the mixing ratio in parentheses. Our results are averaged over two runs in the format *mean (stddev)*. Best results are highlighted in boldface.

Large Tasks With Medium Tasks We investigate whether the same pattern carries over to a medium task such as *image caption generation*. Since the image caption generation task maps images to a sequence of English words (Vinyals et al., 2015b; Xu et al., 2015), only the decoder can be shared with a German→English translation task. Hence, this setting falls under the *many-to-one* MTL setting (§6.1.1).

The results in Table 6.3 show the same trend we observed before, that is, by training on another task for a very small fraction of time, the model improves its performance on its main task. Specifically, with 5 parameter updates for image caption generation per 100 updates for translation (so the mixing ratio of 0.05), we obtain a gain of +0.7 BLEU scores over a strong single-task baseline. Our baseline is almost a BLEU point better than the equivalent non-attention model reported in (Luong et al., 2015b).

Large Tasks with Large Tasks Our first set of experiments is almost the same as the one-to-many big-vs-small-task setting which combines *translation*, as the reference task, with parsing. The only difference is in terms of parsing data. Instead of using the small Penn Tree Bank corpus, we consider a large parsing resource, the high-confidence (HC)

Task	Translation			Captioning
	Valid ppl	Test ppl	Test BLEU	Valid ppl
(Luong et al., 2015b)	-	14.3	16.9	-
Our single-task systems				
Translation	11.0 (0.0)	12.5 (0.2)	17.8 (0.1)	-
Captioning	-	-	-	30.8 (1.3)
Our multi-task systems				
Translation + Captioning (1x)	11.9	14.0	16.7	43.3
Translation + Captioning (0.1x)	10.5 (0.4)	12.1 (0.4)	18.0 (0.6)	**28.4** (0.3)
Translation + Captioning (0.05x)	**10.3** (0.1)	**11.8** (0.0)	**18.5** (0.0)	30.1 (0.3)
Translation + Captioning (0.01x)	10.6 (0.0)	12.3 (0.1)	18.1 (0.4)	35.2 (1.4)

Table 6.3: **German→English WMT'15 translation & captioning results** – shown are perplexities (ppl) and BLEU scores for various tasks with similar format as in Table 6.2. *Reference* tasks are in italic with mixing ratios in parentheses. The average results of 2 runs are in *mean (stddev)* format.

corpus, which is provided by Vinyals et al. (2015a). As highlighted in Table 6.4, the trend is consistent; MTL helps boost translation quality by up to +0.9 BLEU points.

Task	Translation		
	Valid ppl	Test ppl	Test BLEU
(Luong et al., 2015b)	-	8.1	14.0
Our systems			
Translation	8.8 (0.3)	8.3 (0.2)	14.3 (0.3)
Translation + HC Parsing (1x)	8.5 (0.0)	8.1 (0.1)	15.0 (0.6)
Translation + HC Parsing (0.1x)	**8.2** (0.3)	**7.7** (0.2)	**15.2** (0.6)
Translation + HC Parsing (0.05x)	8.4 (0.0)	8.0 (0.1)	14.8 (0.2)

Table 6.4: **English→German WMT'14 translation** – shown are perplexities (ppl) and BLEU scores of various translation models. Our multi-task systems combine translation and parsing on the high-confidence corpus together. Mixing ratios are in parentheses and the average results over 2 runs are in *mean (stddev)* format. Best results are bolded.

The second set of experiments shifts the attention to *parsing* by having it as the reference task. We show in Table 6.5 results that combine parsing with either (a) the English autoencoder task or (b) the English→German translation task. Our models are compared against the best attention-based systems in (Vinyals et al., 2015a), including the state-of-the-art result of 92.8 F_1.

Task	Parsing	
	Valid ppl	Test F_1
LSTM+A (Vinyals et al., 2015a)	-	92.5
LSTM+A+E (Vinyals et al., 2015a)	-	**92.8**
Our systems		
HC Parsing	1.12/1.12	92.2 (0.1)
HC Parsing + Autoencoder (1x)	1.12/1.12	92.1 (0.1)
HC Parsing + Autoencoder (0.1x)	1.12/1.12	92.1 (0.1)
HC Parsing + Autoencoder (0.01x)	1.12/1.13	92.0 (0.1)
HC Parsing + Translation (1x)	1.12/1.13	91.5 (0.2)
HC Parsing + Translation (0.1x)	1.13/1.13	92.0 (0.2)
HC Parsing + Translation (0.05x)	**1.11/1.12**	**92.4 (0.1)**
HC Parsing + Translation (0.01x)	1.12/1.12	92.2 (0.0)
Ensemble of 6 multi-task systems	-	**93.0**

Table 6.5: **Large-Corpus parsing results** – shown are perplexities (ppl) and F_1 scores for various parsing models. Mixing ratios are in parentheses and the average results over 2 runs are in *mean (stddev)* format. We show the individual perplexities for all runs due to small differences among them. For Vinyals et al. (2015a)'s parsing results, LSTM+A represents a single LSTM with attention, whereas LSTM+A+E indicates an ensemble of 5 systems. Important results are bolded.

Before discussing the multi-task results, we note a few interesting observations. First, very small parsing perplexities, close to 1.1, can be achieved with large training data.[7] Second, our baseline system can obtain a very competitive F_1 score of 92.2, rivaling Vinyals et al. (2015a)'s systems. This is rather surprising since our models do not use any attention mechanism. A closer look into these models reveal that there seems to be an architectural difference: Vinyals et al. (2015a) use 3-layer LSTM with 256 cells and 512-dimensional embeddings; whereas our models use 4-layer LSTM with 1000 cells and 1000-dimensional embeddings. This further supports findings in (Jozefowicz et al., 2016) that larger networks matter for sequence models.

For the multi-task results, while autoencoder does not seem to help parsing, translation

[7]Training solely on the small Penn Tree Bank corpus can only reduce the perplexity to at most 1.6, as evidenced by poor parsing results in Table 6.2. At the same time, these parsing perplexities are much smaller than what can be achieved by a translation task. This is because parsing only has 104 tags in the target vocabulary compared to 50K words in the translation case. Note that 1.0 is the theoretical lower bound.

does. At the mixing ratio of 0.05, we obtain a non-negligible boost of 0.2 F_1 over the baseline and with 92.4 F_1, our multi-task system is on par with the best single system reported in (Vinyals et al., 2015a). Furthermore, by ensembling 6 different multi-task models (trained with the translation task at mixing ratios of 0.1, 0.05, and 0.01), we are able to establish a new *state-of-the-art* result in English constituent parsing with **93.0** F_1 score.

Multi-tasks and Unsupervised Learning Our main focus in this section is to determine whether unsupervised learning can help improve translation. Specifically, we follow the *many-to-many* approach described in Section 6.1.1 to couple the German→English translation task with two unsupervised learning tasks on monolingual corpora, one per language. The results in Tables 6.6 show a similar trend as before, a small amount of other tasks, in this case the *autoencoder* objective with mixing coefficient 0.05, improves the translation quality by +0.5 BLEU scores. However, as we train more on the autoencoder task, i.e. with larger mixing ratios, the translation performance gets worse.

Task	**Translation**			**German**	**English**
	Valid ppl	Test ppl	Test BLEU	Test ppl	Test ppl
(Luong et al., 2015b)	-	14.3	16.9	-	-
Our single-task systems					
Translation	11.0 (0.0)	12.5 (0.2)	17.8 (0.1)	-	-
Our multi-task systems with Autoencoders					
Translation + autoencoders (1.0x)	12.3	13.9	16.0	**1.01**	2.10
Translation + autoencoders (0.1x)	11.4	12.7	17.7	1.13	**1.44**
Translation + autoencoders (0.05x)	**10.9** (0.1)	**12.0** (0.0)	**18.3** (0.4)	1.40 (0.01)	2.38 (0.39)
Our multi-task systems with Skip-thought Vectors					
Translation + skip-thought (1x)	**10.4** (0.1)	**10.8** (0.1)	17.3 (0.2)	**36.9** (0.1)	**31.5** (0.4)
Translation + skip-thought (0.1x)	10.7 (0.0)	11.4 (0.2)	17.8 (0.4)	52.8 (0.3)	53.7 (0.4)
Translation + skip-thought (0.01x)	11.0 (0.1)	12.2 (0.0)	**17.8** (0.3)	76.3 (0.8)	142.4 (2.7)

Table 6.6: **German→English WMT'15 translation & unsupervised learning results** – shown are perplexities for translation and unsupervised learning tasks. We experiment with both *autoencoders* and *skip-thought vectors* for the unsupervised objectives. Numbers in *mean (stddev)* format are the average results of 2 runs; others are for 1 run only.

Skip-thought objectives, on the other hand, behave differently. If we merely look at the perplexity metric, the results are very encouraging: with more skip-thought data, we perform better consistently across both the translation and the unsupervised tasks. However, when computing the BLEU scores, the translation quality degrades as we increase the

mixing coefficients. We anticipate that this is due to the fact that the skip-thought objective changes the nature of the translation task when using one half of a sentence to predict the other half. It is not a problem for the autoencoder objectives, however, since one can think of autoencoding a sentence as translating into the same language.

We believe these findings pose interesting challenges in the quest towards better unsupervised objectives, which should satisfy the following criteria: (a) a desirable objective should be compatible with the supervised task in focus, e.g., autoencoders can be viewed as a special case of translation, and (b) with more unsupervised data, both intrinsic and extrinsic metrics should be improved; skip-thought objectives satisfy this criterion in terms of the intrinsic metric but not the extrinsic one.

6.1.3 Conclusion

In this paper, we showed that multi-task learning (MTL) can improve the performance of the attention-free sequence to sequence model of (Sutskever et al., 2014). We found it surprising that training on syntactic parsing and image caption data improved our translation performance, given that these datasets are orders of magnitude smaller than typical translation datasets. Furthermore, we have established a new *state-of-the-art* result in constituent parsing with an ensemble of multi-task models. We also show that the two unsupervised learning objectives, autoencoder and skip-thought, behave differently in the MTL context involving translation. We hope that these interesting findings will motivate future work in utilizing unsupervised data for sequence to sequence learning. A criticism of our work is that our sequence to sequence models do not employ the attention mechanism (Bahdanau et al., 2015). We leave the exploration of MTL with attention for future work.

6.2 Compression of NMT Models via Pruning

While NMT has a significantly smaller memory footprint than traditional phrase-based approaches (which need to store gigantic phrase-tables and language models), the model size of NMT is still prohibitively large for mobile devices. For example, a recent state-of-the-art NMT system requires over 200 million parameters, resulting in a storage size

of hundreds of megabytes (Luong et al., 2015b). Though the trend for bigger and deeper neural networks has brought great progress, it has also introduced over-parameterization, resulting in long running times, overfitting, and the storage size issue discussed above. A solution to the over-parameterization problem could potentially aid all three issues, though the first (long running times) is outside the scope of this paper.

Our contribution. We investigate the efficacy of weight pruning for NMT as a means of compression. We show that despite its simplicity, magnitude-based pruning with retraining is highly effective, and we compare three magnitude-based pruning schemes — *class-blind*, *class-uniform* and *class-distribution*. Though recent work has chosen to use the latter two, we find the first and simplest scheme — *class-blind* — the most successful. We are able to prune 40% of the weights of a state-of-the-art NMT system with negligible performance loss, and by adding a retraining phase after pruning, we can prune 80% with no performance loss. Our pruning experiments also reveal some patterns in the distribution of redundancy in NMT. In particular, we find that higher layers, attention and softmax weights are the most important, while lower layers and the embedding weights hold a lot of redundancy. For the Long Short-Term Memory (LSTM) architecture, we find that at lower layers the parameters for the input are most crucial, but at higher layers the parameters for the gates also become important.

6.2.1 Related Work

Pruning the parameters from a neural network, referred to as *weight pruning* or *network pruning*, is a well-established idea though it can be implemented in many ways. Among the most popular are the Optimal Brain Damage (OBD) (Le Cun et al., 1989) and Optimal Brain Surgeon (OBS) (Hassibi and Stork, 1993) techniques, which involve computing the Hessian matrix of the loss function with respect to the parameters, in order to assess the *saliency* of each parameter. Parameters with low saliency are then pruned from the network and the remaining sparse network is retrained. Both OBD and OBS were shown to perform better than the so-called 'naive magnitude-based approach', which prunes parameters according to their magnitude (deleting parameters close to zero). However, the high computational complexity of OBD and OBS compare unfavorably to the

computational simplicity of the magnitude-based approach, especially for large networks (Augasta and Kathirvalavakumar, 2013).

In recent years, the deep learning renaissance has prompted a re-investigation of network pruning for modern models and tasks. Magnitude-based pruning with iterative retraining has yielded strong results for Convolutional Neural Networks (CNN) performing visual tasks. (Collins and Kohli, 2014) prune 75% of AlexNet parameters with small accuracy loss on the ImageNet task, while (Han et al., 2015b) prune 89% of AlexNet parameters with no accuracy loss on the ImageNet task.

Other approaches focus on pruning neurons rather than parameters, via sparsity-inducing regularizers (Murray and Chiang, 2015) or 'wiring together' pairs of neurons with similar input weights (Srinivas and Babu, 2015). These approaches are much more constrained than weight-pruning schemes; they necessitate finding entire zero rows of weight matrices, or near-identical pairs of rows, in order to prune a single neuron. By contrast weight-pruning approaches allow weights to be pruned freely and independently of each other. The neuron-pruning approach of (Srinivas and Babu, 2015) was shown to perform poorly (it suffered performance loss after removing only 35% of AlexNet parameters) compared to the weight-pruning approach of (Han et al., 2015b). Though (Murray and Chiang, 2015) demonstrates neuron-pruning for language modeling as part of a (non-neural) Machine Translation pipeline, their approach is more geared towards architecture selection than compression.

There are many other compression techniques for neural networks, including approaches based on on low-rank approximations for weight matrices (Jaderberg et al., 2014; Denton et al., 2014), or weight sharing via hash functions (Chen et al., 2015). Several methods involve reducing the precision of the weights or activations (Courbariaux et al., 2015), sometimes in conjunction with specialized hardware (Gupta et al., 2015), or even using binary weights (Lin et al., 2016). The 'knowledge distillation' technique of (Hinton et al., 2015) involves training a small 'student' network on the soft outputs of a large 'teacher' network. Some approaches use a sophisticated pipeline of several techniques to achieve impressive feats of compression (Han et al., 2015a; Iandola et al., 2016).

Most of the above work has focused on compressing CNNs for vision tasks. We extend the magnitude-based pruning approach of (Han et al., 2015b) to recurrent neural networks

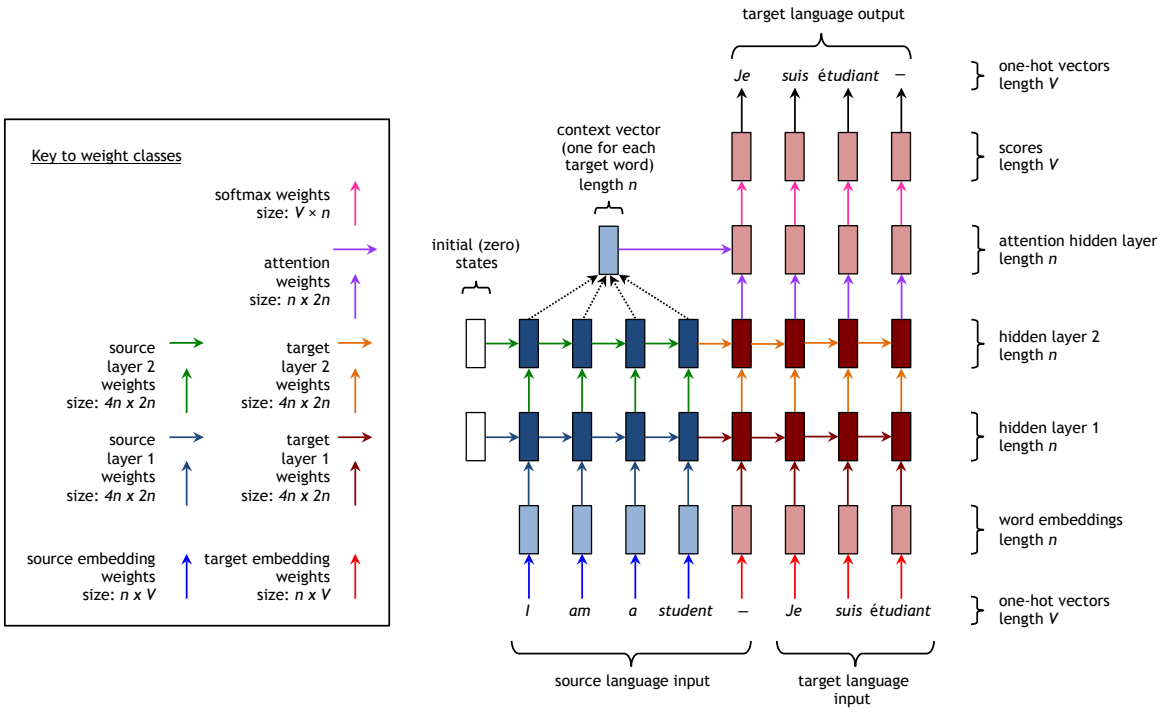

Figure 6.5: NMT architecture. This example has two layers, but our system has four. The different weight classes are indicated by arrows of different color (the black arrows in the top right represent simply choosing the highest-scoring word, and thus require no parameters). Best viewed in color.

(RNN), in particular LSTM architectures for NMT, and to our knowledge we are the first to do so. There has been some recent work on compression for RNNs (Lu et al., 2016; Prabhavalkar et al., 2016), but it focuses on other, non-pruning compression techniques. Nonetheless, our general observations on the distribution of redundancy in a LSTM, detailed in Section 6.2.3, are corroborated by (Lu et al., 2016).

6.2.2 Our Approach

Understanding NMT Weights

In this work, we specifically consider the *deep multi-layer recurrent* architecture with *LSTM* as the hidden unit type. Figure 6.5 shows the system in detail, highlighting the different types of parameters, or weights, in the model. We will go through the architecture

from bottom to top. First, a vocabulary is chosen for each language, assuming that the top V frequent words are selected. Thus, every word in the source or target vocabulary can be represented by a one-hot vector of length V. The source input sentence and target input sentence, represented as a sequence of one-hot vectors, are transformed into a sequence of word embeddings by the *embedding* weights. These embedding weights, which are learned during training, are different for the source words and the target words. The word embeddings and all hidden layers are vectors of length n (a chosen hyperparameter).

The word embeddings are then fed as input into the main network, which consists of two multi-layer RNNs 'stuck together' — an encoder for the source language and a decoder for the target language, each with their own weights. The *feed-forward* (vertical) weights connect the hidden unit from the layer below to the upper RNN block, and the *recurrent* (horizontal) weights connect the hidden unit from the previous time-step RNN block to the current time-step RNN block.

The hidden state at the top layer of the decoder is fed through an *attention* layer, which guides the translation by 'paying attention' to relevant parts of the source sentence; for more information see (Bahdanau et al., 2015) or Section 3 of (Luong et al., 2015b). Finally, for each target word, the top layer hidden unit is transformed by the *softmax* weights into a score vector of length V. The target word with the highest score is selected as the output translation.

Weight Subgroups in LSTM – For the aforementioned RNN block, we choose to use LSTM as the hidden unit type. To facilitate our later discussion on the different subgroups of weights within LSTM, we first review the details of LSTM as formulated by Zaremba et al. (2014) as follows:

$$\begin{pmatrix} i \\ f \\ o \\ \hat{h} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{4n,2n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix} \tag{6.1}$$

$$c_t^l = f \circ c_{t-1}^l + i \circ \hat{h} \tag{6.2}$$

$$h_t^l = o \circ \tanh(c_t^l) \tag{6.3}$$

Here, each LSTM block at time t and layer l computes as output a pair of hidden and memory vectors (h_t^l, c_t^l) given the previous pair (h_{t-1}^l, c_{t-1}^l) and an input vector h_t^{l-1} (either from the LSTM block below or the embedding weights if $l = 1$). All of these vectors have length n. The core of a LSTM block is the weight matrix $T_{4n,2n}$ of size $4n \times 2n$. This matrix can be decomposed into 8 subgroups that are responsible for the interactions between $\{$input gate i, forget gate f, output gate o, input signal $\hat{h}\} \times \{$feed-forward input h_t^{l-1}, recurrent input $h_{t-1}^l\}$.

Pruning Schemes

We follow the general magnitude-based approach of (Han et al., 2015b), which consists of pruning weights with smallest absolute value. However, we question the authors' pruning scheme with respect to the different weight classes, and experiment with three pruning schemes. Suppose we wish to prune $x\%$ of the total parameters in the model. How do we distribute the pruning over the different weight classes (illustrated in Figure 6.5) of our model? We propose to examine three different pruning schemes:

1. *Class-blind*: Take all parameters, sort them by magnitude and prune the $x\%$ with smallest magnitude, regardless of weight class. (So some classes are pruned proportionally more than others).

2. *Class-uniform*: Within each class, sort the weights by magnitude and prune the $x\%$ with smallest magnitude. (So all classes have exactly $x\%$ of their parameters pruned).

3. *Class-distribution*: For each class c, weights with magnitude less than $\lambda\sigma_c$ are pruned. Here, σ_c is the standard deviation of that class and λ is a universal parameter chosen such that in total, $x\%$ of all parameters are pruned. This is used by (Han et al., 2015b).

All these schemes have their seeming advantages. Class-blind pruning is the simplest and adheres to the principle that pruning weights (or equivalently, setting them to zero) is least damaging when those weights are small, regardless of their locations in the architecture. Class-uniform pruning and class-distribution pruning both seek to prune proportionally

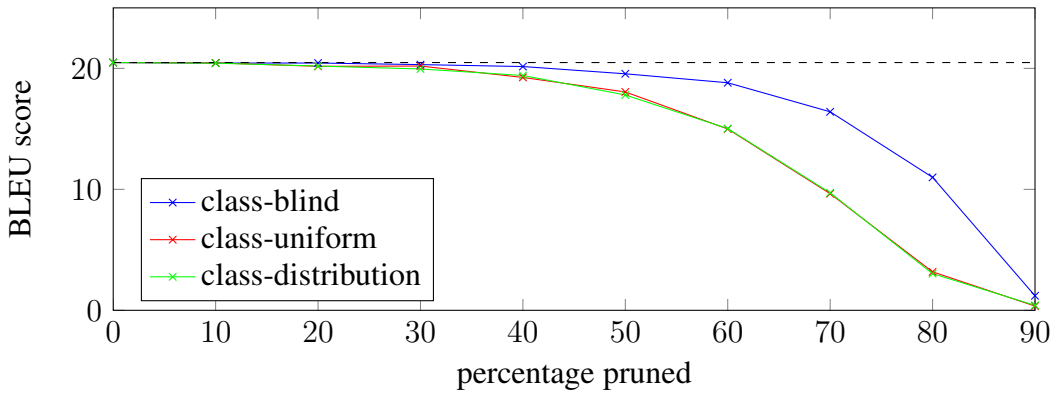

Figure 6.6: Effects of different pruning schemes.

within each weight class, either absolutely, or relative to the standard deviation of that class. We find that class-blind pruning outperforms both other schemes (see Section 6.2.3).

Retraining

In order to prune NMT models aggressively without performance loss, we retrain our pruned networks. That is, we continue to train the remaining weights, but maintain the sparse structure introduced by pruning. In our implementation, pruned weights are represented by zeros in the weight matrices, and we use binary 'mask' matrices, which represent the sparse structure of a network, to ignore updates to weights at pruned locations. This implementation has the advantage of simplicity as it requires minimal changes to the training and deployment code, but we note that a more complex implementation utilizing sparse matrices and sparse matrix multiplication could potentially yield speed improvements. However, such an implementation is beyond the scope of this paper.

6.2.3 Experiments

We evaluate the effectiveness of our pruning approaches on a state-of-the-art NMT model.[8] Specifically, an attention-based English-German NMT system from (Luong et al., 2015b) is considered. Training data was obtained from WMT'14 consisting of 4.5M sentence pairs

[8]We thank the authors of (Luong et al., 2015b) for providing their trained models and assistance in using the codebase at `https://github.com/lmthang/nmt.matlab`.

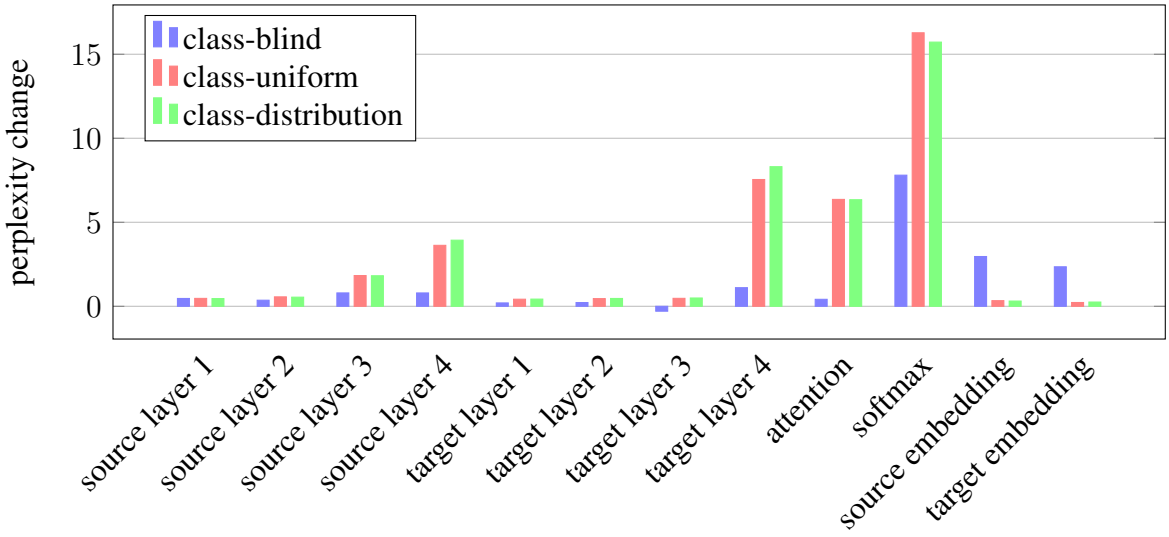

Figure 6.7: 'Breakdown' of performance loss (i.e., perplexity increase) by weight class, when pruning 90% of weights using each of the three pruning schemes. Each of the first eight classes have 8 million weights, attention has 2 million, and the last three have 50 million weights each.

(116M English words, 110M German words). For more details on training hyperparameters, we refer readers to Section 4.1 of (Luong et al., 2015b). All models are tested on newstest2014 (2737 sentences). The model achieves a perplexity of 6.1 and a BLEU score of 20.5 (after unknown word replacement).[9]

When *retraining* pruned NMT systems, we use the following settings: (a) we start with a smaller learning rate of 0.5 (the original model uses a learning rate of 1.0), (b) we train for fewer epochs, 4 instead of 12, using plain SGD, (c) a simple learning rate schedule is employed; after 2 epochs, we begin to halve the learning rate every half an epoch, and (d) all other hyperparameters are the same, such as mini-batch size 128, maximum gradient norm 5, and dropout with probability 0.2.

[9]The performance of this model is reported under row *global (dot)* in Table 4 of (Luong et al., 2015b).

Comparing pruning schemes

Despite its simplicity, we observe in Figure 6.6 that *class-blind* pruning outperforms both other schemes in terms of translation quality at all pruning percentages. In order to understand this result, for each of the three pruning schemes, we pruned each class separately and recorded the effect on performance (as measured by perplexity). Figure 6.7 shows that with class-uniform pruning, the overall performance loss is caused disproportionately by a few classes: target layer 4, attention and softmax weights. Looking at Figure 6.8, we see that the most damaging classes to prune also tend to be those with weights of greater magnitude — these classes have much larger weights than others at the same percentile, so pruning them under the class-uniform pruning scheme is more damaging. The situation is similar for class-distribution pruning.

By contrast, Figure 6.7 shows that under class-blind pruning, the damage caused by pruning softmax, attention and target layer 4 weights is greatly decreased, and the contribution of each class towards the performance loss is overall more uniform. In fact, the distribution begins to reflect the number of parameters in each class — for example, the source and target embedding classes have larger contributions because they have more weights. We use only class-blind pruning for the rest of the experiments.

Figure 6.7 also reveals some interesting information about the distribution of redundancy in NMT architectures — namely it seems that higher layers are more important than lower layers, and that attention and softmax weights are crucial. We will explore the distribution of redundancy further in Section 6.2.3.

Pruning and retraining

Pruning has an immediate negative impact on performance (as measured by BLEU) that is exponential in pruning percentage; this is demonstrated by the blue line in Figure 6.9. However we find that up to about 40% pruning, performance is mostly unaffected, indicating a large amount of redundancy and over-parameterization in NMT.

We now consider the effect of retraining pruned models. The orange line in Figure 6.9 shows that after retraining the pruned models, baseline performance (20.48 BLEU) is both recovered and improved upon, up to 80% pruning (20.91 BLEU), with only a small

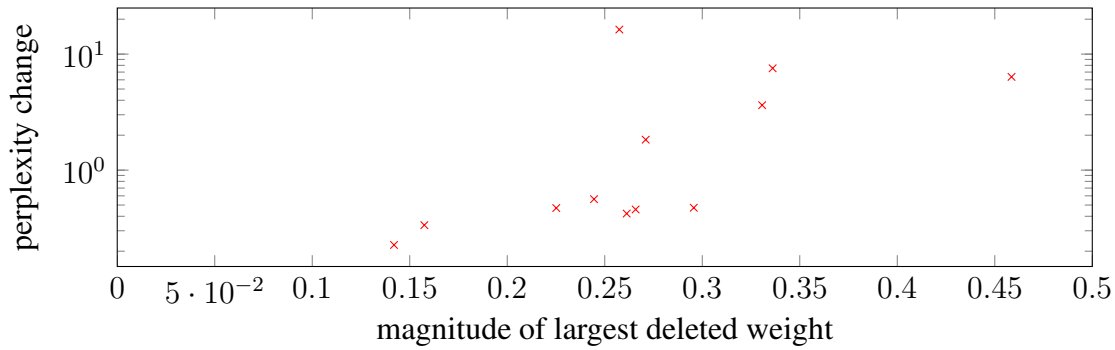

Figure 6.8: Magnitude of largest deleted weight vs. perplexity change, for the 12 different weight classes when pruning 90% of parameters by class-uniform pruning.

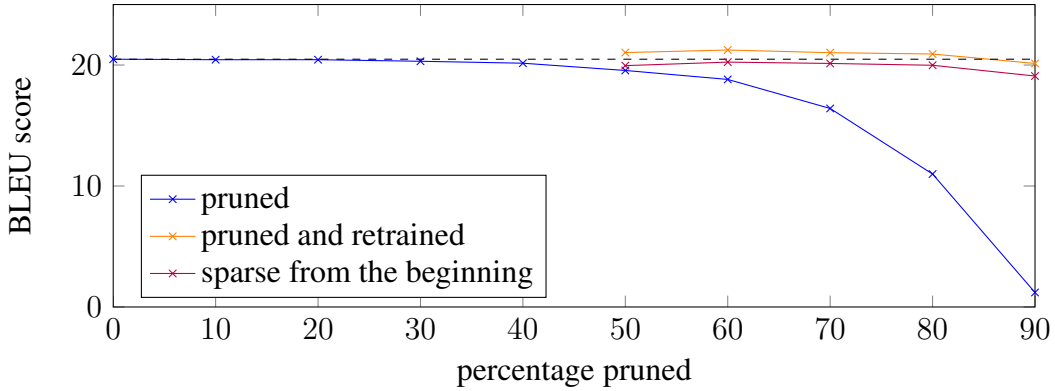

Figure 6.9: Performance of pruned models (a) after pruning, (b) after pruning and retraining, and (c) when trained with sparsity structure from the outset (see Section 6.2.3).

performance loss at 90% pruning (20.13 BLEU). This may seem surprising, as we might not expect a sparse model to significantly out-perform a model with five times as many parameters. There are several possible explanations, two of which are given below.

Firstly, we found that the less-pruned models perform better on the training set than the validation set, whereas the more-pruned models have closer performance on the two sets. This indicates that pruning has a regularizing effect on the retraining phase, though clearly more is not always better, as the 50% pruned and retrained model has better validation set performance than the 90% pruned and retrained model. Nonetheless, this regularization effect may explain why the pruned and retrained models outperform the baseline.

Alternatively, pruning may serve as a means to escape a local optimum. Figure 6.10

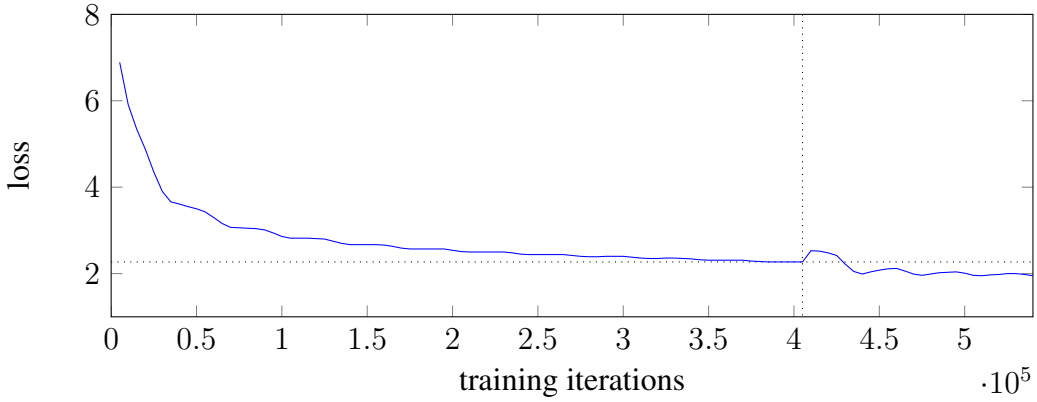

Figure 6.10: The validation set loss during training, pruning and retraining. The vertical dotted line marks the point when 80% of the parameters are pruned. The horizontal dotted line marks the best performance of the unpruned baseline.

shows the loss function over time during the training, pruning and retraining process. During the original training process, the loss curve flattens out and seems to converge (note that we use early stopping to obtain our baseline model, so the original model was trained for longer than shown in Figure 6.10). Pruning causes an immediate increase in the loss function, but enables further gradient descent, allowing the retraining process to find a new, better local optimum. It seems that the disruption caused by pruning is beneficial in the long-run.

Starting with sparse models

The favorable performance of the pruned and retrained models raises the question: can we get a shortcut to this performance by *starting* with sparse models? That is, rather than train, prune, and retrain, what if we simply prune then train? To test this, we took the sparsity structure of our 50%–90% pruned models, and trained completely new models with the same sparsity structure. The purple line in Figure 6.9 shows that the 'sparse from the beginning' models do not perform as well as the pruned and retrained models, but they do come close to the baseline performance. This shows that while the sparsity structure alone contains useful information about redundancy and can therefore produce a competitive compressed model, it is important to interleave pruning with training.

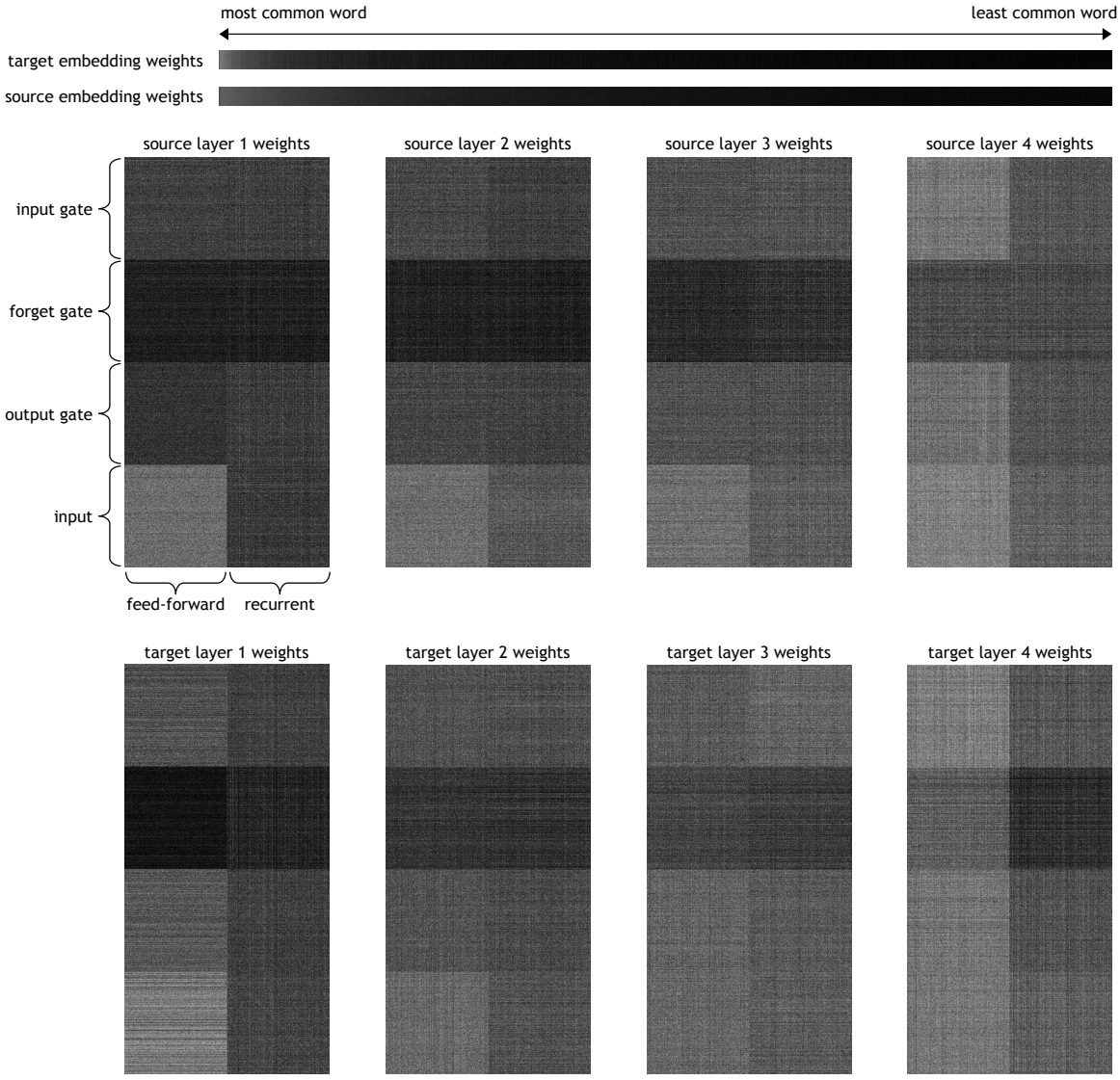

Figure 6.11: Graphical representation of the location of small weights in various parts of the model. Black pixels represent weights with absolute size in the bottom 80%; white pixels represent those with absolute size in the top 20%. Equivalently, these pictures illustrate which parameters remain after pruning 80% using our class-blind pruning scheme.

Though our method involves just one pruning stage, other pruning methods interleave pruning with training more closely by including several iterations (Collins and Kohli, 2014; Han et al., 2015b). We expect that implementing this for NMT would likely result in further compression and performance improvements.

Storage size

The original unpruned model (a MATLAB file) has size 782MB. The 80% pruned and retrained model is 272MB, which is a 65.2% reduction. In this work we focus on compression in terms of number of parameters rather than storage size, because it is invariant across implementations.

Distribution of redundancy in NMT

We visualize in Figure 6.11 the redundancy structore of our NMT baseline model. *Black* pixels represent weights near to zero (those that can be pruned); *white* pixels represent larger ones. First we consider the embedding weight matrices, whose columns correspond to words in the vocabulary. Unsurprisingly, in Figure 6.11, we see that the parameters corresponding to the less common words are more dispensable. In fact, at the 80% pruning rate, for 100 uncommon source words and 1194 uncommon target words, we delete *all* parameters corresponding to that word. This is not quite the same as removing the word from the vocabulary — true out-of-vocabulary words are mapped to the embedding for the 'unknown word' symbol, whereas these 'pruned-out' words are mapped to a zero embedding. However in the original unpruned model these uncommon words already had near-zero embeddings, indicating that the model was unable to learn sufficiently distinctive representations.

Returning to Figure 6.11, now look at the eight weight matrices for the source and target connections at each of the four layers. Each matrix corresponds to the $4n \times 2n$ matrix $T_{4n,2n}$ in Equation (6.1). In all eight matrices, we observe — as does (Lu et al., 2016) — that the weights connecting to the input $\hat{h}$ are most crucial, followed by the input gate i, then the output gate o, then the forget gate f. This is particularly true of the lower layers, which focus primarily on the input $\hat{h}$. However for higher layers, especially on the target side,

weights connecting to the gates are as important as those connecting to the input $\hat{h}$. The gates represent the LSTM's ability to add to, delete from or retrieve information from the memory cell. Figure 6.11 therefore shows that these sophisticated memory cell abilities are most important at the *end* of the NMT pipeline (the top layer of the decoder). This is reasonable, as we expect higher-level features to be learned later in a deep learning pipeline.

We also observe that for lower layers, the feed-forward input is much more important than the recurrent input, whereas for higher layers the recurrent input becomes more important. This makes sense: lower layers concentrate on the low-level information from the current word embedding (the feed-forward input), whereas higher layers make use of the higher-level representation of the sentence so far (the recurrent input).

Lastly, on close inspection, we notice several white diagonals emerging within some subsquares of the matrices in Figure 6.11, indicating that even without initializing the weights to identity matrices (as is sometimes done (**?**)), an identity-like weight matrix is learned. At higher pruning percentages, these diagonals become more pronounced.

6.2.4 Generalizability of our results

To test the generalizability of our results, we also test our pruning approach on a smaller, non-state-of-the-art NMT model trained on the WIT3 Vietnamese-English dataset (**?**), which consists of 133,000 sentence pairs. This model is effectively a scaled-down version of the state-of-the-art model in (Luong et al., 2015b), with fewer layers, smaller vocabulary size, smaller hidden layer size, no attention mechanism, and about 11% as many parameters in total. It achieves a BLEU score of 9.61 on the validation set.

Although this model and its training set are on a different scale to our main model, and the language pair is different, we found very similar results. For this model, it is possible to prune 60% of parameters with no immediate performance loss, and with retraining it is possible to prune 90%, and regain original performance. Our main observations from Section 6.2.3 are also replicated; in particular, class-blind pruning is most successful, 'sparse from the beginning' models are less successful than pruned and retrained models, and we observe the same patterns as seen in Figure 6.11.

6.2.5 Conclusion

We have shown that weight pruning with retraining is a highly effective method of compression and regularization on a state-of-the-art NMT system, compressing the model to 20% of its size with no loss of performance. Though we are the first to apply compression techniques to NMT, we obtain a similar degree of compression to other current work on compressing state-of-the-art deep neural networks, with an approach that is simpler than most. We have found that the absolute size of parameters is of primary importance when choosing which to prune, leading to an approach that is extremely simple to implement, and can be applied to any neural network. Lastly, we have gained insight into the distribution of redundancy in the NMT architecture.

In terms of future work, including *several* iterations of pruning and retraining would likely improve the compression and performance of our pruning method. If possible it would be highly valuable to exploit the sparsity of the pruned models to speed up training and runtime, perhaps through sparse matrix representations and multiplications (see Section 6.2.2). Though we have found magnitude-based pruning to perform very well, it would be instructive to revisit the original claim that other pruning methods (for example Optimal Brain Damage and Optimal Brain Surgery) are more principled, and perform a comparative study.

6.3 Future Outlook

Chapter 7

Conclusion

Bibliography

Robert B. Allen. 1987. Several studies on natural language and back-propagation. In *ICNN*.

Stephen R. Anderson. 2010. How many languages are there in the world? `http://www.linguisticsociety.org/content/how-many-languages-are-there-world`. Accessed: 2016-09-10.

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR* 6:1817–1853.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. 2007. Multi-task feature learning. In *NIPS*.

M Gethsiyal Augasta and T Kathirvalavakumar. 2013. Pruning algorithms of neural networks - a comparative study. *Central European Journal of Computer Science* 3(3):105–115.

Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint language and translation modeling with recurrent neural networks. In *ACL*.

Amittai Axelrod, Xiaodong He, and Jianfeng Gao. 2011. Domain adaptation via pseudo in-domain data selection. In *EMNLP*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. 2016. End-to-end attention-based large vocabulary speech recognition. In *ICASSP*.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *EMNLP*.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. 2013. Advances in optimizing recurrent networks. In *ICASSP*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *JMLR* 3:1137–1155.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2):157–166.

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* 22(1):39–71.

Christopher M. Bishop. 1994. Mixture density networks. Technical report, Aston University.

Ondřej Bojar and Aleš Tamchyna. 2015. CUNI in WMT15: Chimera Strikes Again. In *WMT*.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. 2015. Findings of the 2015 workshop on statistical machine translation. In *WMT*.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *EMNLP*.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics* 19(2):263–311.

Christian Buck, Kenneth Heafield, and Bas van Ooyen. 2014. N-gram counts and language models from the common crawl. In *LREC*.

Rich Caruana. 1997. Multitask learning. *Machine Learning* 28(1):41–75.

Daniel Cer, Michel Galley, Daniel Jurafsky, and Christopher D. Manning. 2010. Phrasal: A statistical machine translation toolkit for exploring new model features. In *ACL, Demonstration Session*.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. 2016. Listen, attend and spell. In *ICASSP*.

Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *ICML*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *ACL*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics* 33(2):201–228.

David Chiang, Kevin Knight, and Wei Wang. 2009. 11,001 new features for statistical machine translation. In *NAACL*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.

Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. End-to-end continuous speech recognition using attention-based recurrent NN: first results. *CoRR* abs/1412.1602.

Lonnie Chrisman. 1991. Learning recursive distributed representations for holistic computation. *Connection Science* 3(4):345–366.

Maxwell D Collins and Pushmeet Kohli. 2014. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442* .

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Low precision arithmetic for deep learning. In *ICLR workshop*.

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In *NIPS*.

Ernest David. 2016. Winograd schemas and machine translation. *arXiv preprint arXiv:1608.01884* .

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *ACL*.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. DeCAF: A deep convolutional activation feature for generic visual recognition.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. 2015. Multi-task learning for multiple language translation. In *ACL*.

Cícero Nogueira dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *ICML*.

Nadir Durrani, Barry Haddow, Philipp Koehn, and Kenneth Heafield. 2014. Edinburgh's phrase-based machine translation systems for WMT-14. In *WMT*.

Chris Dyer, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *ACL, Demonstration Session*.

Salah El Hihi and Yoshua Bengio. 1996. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*.

Jeffrey L. Elman. 1990. Finding structure in time. In *Cognitive Science*.

Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized multi–task learning. In *SIGKDD*.

Marcello Federico, Nicola Bertoldi, and Mauro Cettolo. 2008. IRSTLM: an open source toolkit for handling large scale language models. In *Interspeech*.

Alexander Fraser and Daniel Marcu. 2007. Measuring word alignment quality for statistical machine translation. *Computational Linguistics* 33(3):293–303.

Michel Galley and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *EMNLP*.

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. 2000. Learning to forget: Continual prediction with lstm. *Neural Computation* 12(10):2451–2471.

C. Goller and A. Kchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. *IEEE Transactions on Neural Networks* 1:347–352.

A. Graves. 2013. Generating sequences with recurrent neural networks. In *Arxiv preprint arXiv:1308.0850*.

A. Graves, G. Wayne, and I. Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* .

Alex Graves and Juergen Schmidhuber. 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18(5-6):602–610.

Spence Green, Sida Wang, Daniel Cer, and Christopher D. Manning. 2013. Fast and adaptive online training of feature-rich translation models. In *ACL*.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A search space odyssey. *arXiv preprint arXiv:1503.04069* .

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. 2015. DRAW: A recurrent neural network for image generation. In *ICML*.

Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loic Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2015. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535* .

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *ICML*.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. In *NIPS*.

Babak Hassibi and David G Stork. 1993. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *WMT*.

Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, Marc'Aurelio Ranzato, Matthieu Devin, and Jeffrey Dean. 2013. Multilingual acoustic models using distributed deep neural networks. In *ICASSP*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. 2013. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *ICASSP*.

W. John Hutchins. 2000. Warren Weaver and the launching of MT: brief biographical note. In *Early Years in Machine Translation: Memoirs and Biographies of Pioneers*, John Benjamins, pages 17–20.

W. John Hutchins. 2007. Machine translation: A concise history. In Chan Sin Wai, editor, *Computer Aided Translation: Theory and Practice*, Chinese University of Hong Kong.

Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 1mb model size. *arXiv preprint arXiv:1602.07360* .

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *NIPS*.

Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. 2007. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks* 20(3):335–352.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015a. On using very large target vocabulary for neural machine translation. In *ACL*.

Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015b. Montreal neural machine translation systems for WMT'15. In *WMT*.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* .

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *ICML*.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *EMNLP*.

Andrej Karpathy. 2015. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Accessed: 2016-07-05.

Slava Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35(3):400–401.

Nataly Kelly. 2014. Why machines alone cannot solve the worlds translation problem. `http://www.huffingtonpost.com/nataly-kelly/why-machines-alone-cannot-translation_b_4570018.html`. Accessed: 2016-09-10.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *AAAI*.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-thought vectors. In *NIPS*.

Philipp Koehn. 2010. *Statistical Machine Translation*. Cambridge University Press, 1st edition.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL, Demonstration Session*.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *NAACL*.

Abhishek Kumar and Hal Daumé III. 2012. Learning task grouping and overlap in multi-task learning. In *ICML*.

Yann Le Cun, John S Denker, and Sara A Solla. 1989. Optimal brain damage. In *NIPS*.

Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. 2015. A hierarchical neural autoencoder for paragraphs and documents. In *ACL*.

Percy Liang, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *NAACL*.

Tsungnan Lin, Bill G. Horne, Peter Tiňo, and C. Lee Giles. 1996. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks* 7(6):1329–1338.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2016. Neural networks with few multiplications. In *ICLR*.

Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. 2015a. Finding function in form: Compositional character models for open vocabulary word representation. In *EMNLP*.

Wang Ling, Isabel Trancoso, Chris Dyer, and Alan Black. 2015b. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586* .

Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *NAACL*.

Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. 2016. Learning compact recurrent neural networks. In *ICASSP*.

Minh-Thang Luong, Michael Kayser, and Christopher D. Manning. 2015a. Deep neural language models for machine translation. In *CoNLL*.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task sequence to sequence learning. In *ICLR*.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domain. In *IWSLT*.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *ACL*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015b. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Minh-Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *CoNLL*.

Minh-Thang Luong, Ilya Sutskever, Quoc V. Le, Oriol Vinyals, and Wojciech Zaremba. 2015c. Addressing the rare word problem in neural machine translation. In *ACL*.

Daniel Marcu and William Wong. 2002. A phrase-based, joint probability model for statistical machine translation. In *EMNLP*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics* 19(2):313–330.

James Martens and Ilya Sutskever. 2011. Learning recurrent neural networks with Hessian-free optimization. In *ICML*.

Tomáš Mikolov. 2012. *Statistical Language Models Based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

Tomáš Mikolov, Martin Karafit, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*.

Tomáš Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *ICASSP*.

Tomáš Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Tomáš Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. In *SLT*.

Andriy Mnih and Geoffrey Hinton. 2009. A scalable hierarchical distributed language model. In *NIPS*.

Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. In *ICML*.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *NIPS*.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS*.

Kenton Murray and David Chiang. 2015. Auto-sizing neural networks: With applications to n-gram language models. In *EMNLP*.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL*.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *ACL*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics* 29(1):19–51.

Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics* 30(4):417–449.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*.

Razvan Pascanu, Tomáš Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *ICML*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2014. Dropout improves recurrent neural networks for handwriting recognition. In *ICFHR*.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *WMT*.

Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Ian McGraw. 2016. On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition. In *ICASSP*.

Ronald Rosenfeld. 2000. Two decades of statistical language modeling: Where do we go from here? In *IEEE*. volume 88, pages 1270–1278.

David E. Rumelhart and James L. McClelland. 1986. On learning the past tenses of English verbs. In J. L. McClelland, D. E. Rumelhart, and PDP Research Group, editors, *Parallel Distributed Processing. Volume 2: Psychological and Biological Models*, MIT Press, pages 216–271.

H. Schwenk. 2014. University le mans. `http://www-lium.univ-lemans.fr/~schwenk/cslm_joint_paper/`. [Online; accessed 03-September-2014].

Holger Schwenk. 2007. Continuous space language models. *Computer Speech and Languages* 21(3):492–518.

Holger Schwenk. 2012. Continuous space translation models for phrase-based statistical machine translation. In *COLING*.

Abigail See, Minh-Thang Luong, and Christopher D. Manning. 2016. Compression of neural machine translation models via pruning. In *CoNLL*.

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

Peter Sheridan. 1955. Research in language translation on the IBM type 701. In *IBM Technical Newsletter*, 9.

Le Hai Son, Alexandre Allauzen, and Franois Yvon. 2012. Continuous space translation models with neural networks. In *NAACL-HLT*.

Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. In *BMVC*.

Andreas Stolcke. 2002. SRILM – an extensible language modeling toolkit. In *ICSLP*.

Ilya Sutskever. 2012. *Training Recurrent Neural Networks*. Ph.D. thesis, University of Toronto.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Yee Whye Teh. 2006. A hierarchical Bayesian language model based on Pitman-Yor processes. In *ACL*.

Sebastian Thrun. 1996. Is learning the n-th thing any easier than learning the first? In *NIPS*.

Laurens van der Maaten. 2013. Barnes-Hut-SNE. In *ICLR*.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *EMNLP*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015a. Grammar as a foreign language. In *NIPS*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015b. Show and tell: A neural image caption generator. In *CVPR*.

Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. 1990. Readings in speech recognition. chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404.

Warren Weaver. 1949. Translation. In William N. Locke and A. Donald Boothe, editors, *Machine Translation of Languages*, MIT Press, Cambridge, MA, pages 15–23. Reprinted from a memorandum written by Weaver in 1949.

Paul J. Werbos. 1990. Back propagation through time: What it does and how to do it. In *Proceedings of the IEEE*. volume 78, pages 1550–1560.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics* 23(3):377–403.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *ACL*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* .

Richard Zens, Franz Josef Och, and Hermann Ney. 2002. *Phrase-Based Statistical Machine Translation*, Springer Berlin Heidelberg, pages 18–32.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.