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
October 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$^{\text{th}}$ 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.

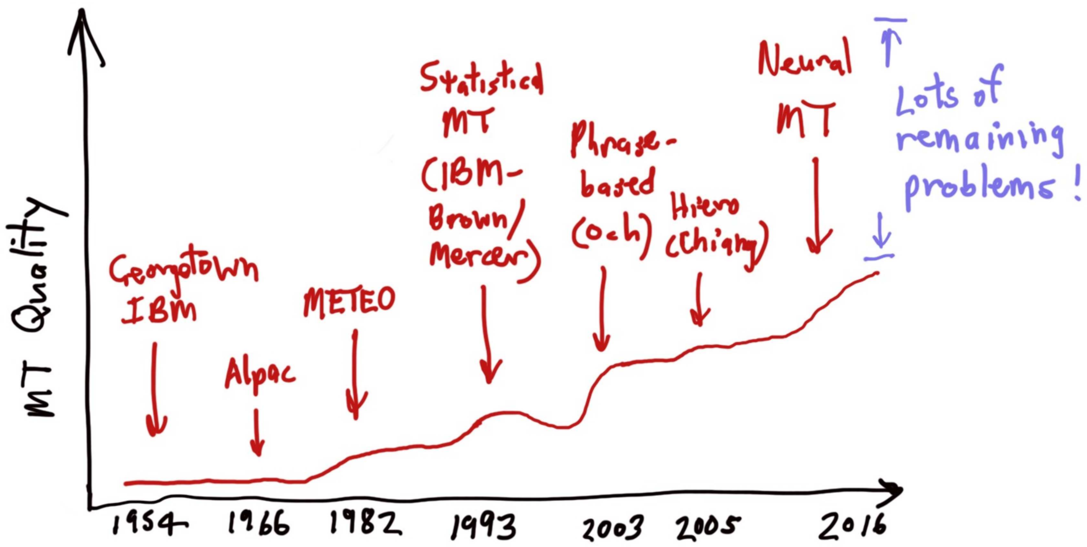

Figure 1.1: **Machine translation progress** – from the 1950s, the starting of modern MT research, until the time of this thesis, 2016, in which neural MT becomes a dominant approach. Image courtesy of Christopher D. Manning.

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).

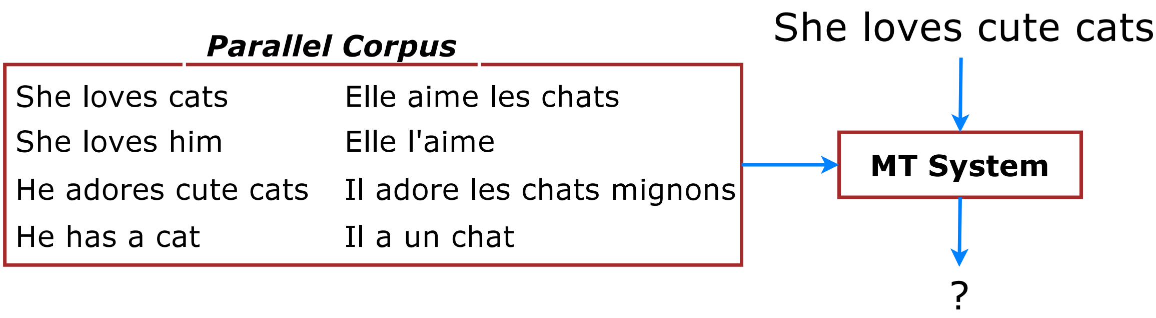

Figure 1.2: **Corpus-based approaches to machine translation** – a general setup in which MT systems are built from parallel corpora of sentence pairs having the same meaning. Once built, systems are used to translate new unseen sentences, e.g., "She loves cute cats".

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.

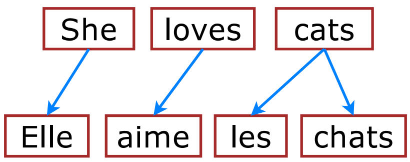

Figure 1.3: **Word-based alignment** – example of an alignment between source and target words. In IBM alignment models, each target word is aligned to at most one source word.

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.

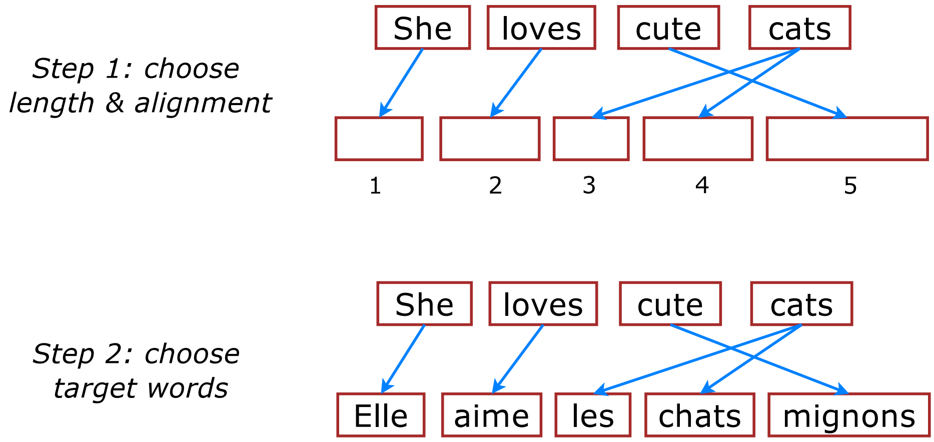

Figure 1.4: **A simple translation story** – example of the generative story in IBM Model 1 to produce a target translation given a source sentence and a learned translation model.

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:

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

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.

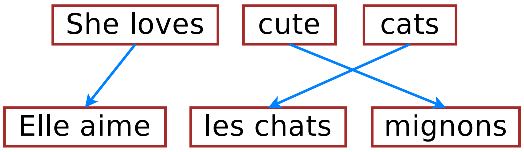

Figure 1.5: **Phrase-based machine translation** (MT) – example of how phrase-based MT systems translate a source sentence "She loves cute cats" into a target sentence "Elle aime les chats mignons": sentences are split into chunks and phrases are translated.

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.

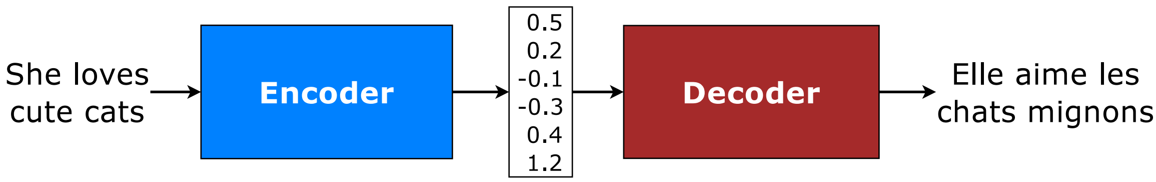

Figure 1.6: **Encoder-decoder architecture** – example of the general approach for NMT. An *encoder* converts a source sentence into a meaning vector which is passed through a *decoder* to produce a translation.

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.

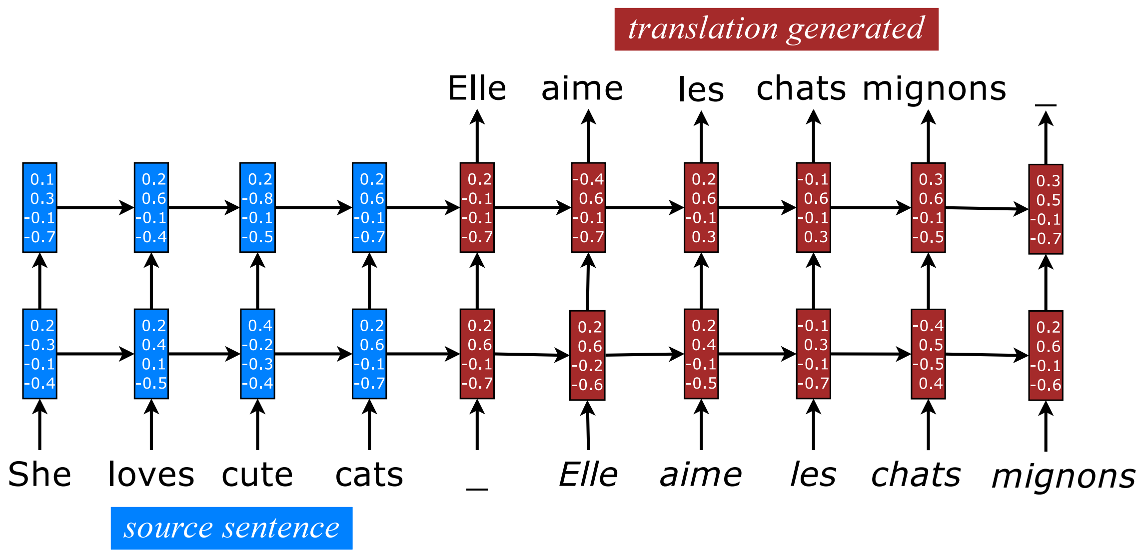

Figure 1.7: **Sequence Models for NMT** – example of a deep recurrent architecture for translating a source sentence "She loves cute cats" into a target sentence "Elle aime les chats mignons". On the decoder side, *words* generated from previous timesteps are used as inputs for the next ones. Here, "_"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

Related publications

(Luong et al., 2013)

(Luong et al., 2015a)

[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".

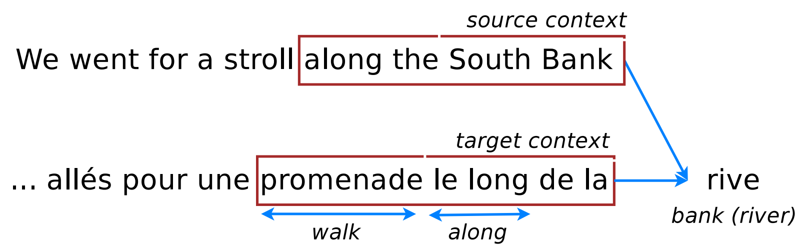

Figure 2.1: **Source-conditioned neural language models** (NLMs) – example of a source-conditioned NLM proposed by Devlin et al. (2014). To evaluate how likely a next word "rive" is, the model not only relies on previous target words (context) "promenade le long de la" as in traditional NLMs (Bengio 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, e.g., translation models, language models, reordering models, etc.; now, it becomes even worse as different neural components are incorporated. This motivates neural machine translation to redesign the entire MT pipeline completely. Before we go into details of NMT, we will first learn about recurrent neural networks which is a building block for NMT and a key component that will help address the aforementioned problem of translating locally in statistical MT systems including hybrid models.

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 Draw a picture on general RNNs. 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.

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).

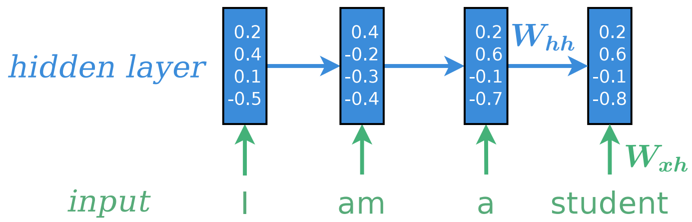

Figure 2.2: **Recurrent neural networks** – example of a recurrent neural network that processes a sequence of input words "I am a student" to build up hidden representations as input symbols are consumed. The recurrent $\boldsymbol{W}_{hh}$ and feed-forward $\boldsymbol{W}_{xh}$ weights are shared across timesteps.

Recurrent Language Models To apply RNNs to sentences in languages, or generally sequences of discrete symbols, one can consider one-hot representations $\boldsymbol{x}_i \in \mathbb{R}^{|V|}$, 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 *word 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 simple RNN applied to language modeling will generally have $\theta = \{\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_{hy}, \boldsymbol{W}_e\}$ as its weights as illustrated in Figure 2.3 Draw an RNN with embedding.

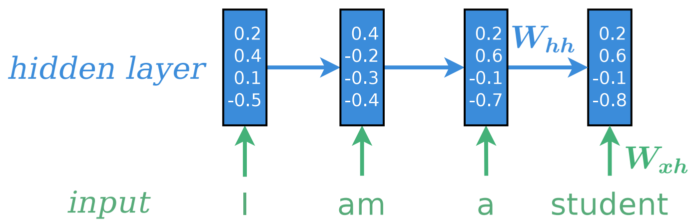

Figure 2.3: **Recurrent language models** – example of a recurrent neural network that processes a sequence of input words “I am a student” to build up hidden representations as input symbols are consumed. The recurrent $\boldsymbol{W}_{hh}$ and feed-forward $\boldsymbol{W}_{xh}$ weights are shared across timesteps.

An RNN-based language model (RNNLM) is a special case of RNNs in which: (a) the input and output are sequences of discrete words, (b) the output sequence ends with a special symbol <eos> that marks the boundary, e.g., y = { “I”, “am”, “a”, “student”, <eos>}, and (c) the input sequence is a shift-by-1 version of the output sequence with <sos> as a starting symbol, e.g., x = { <sos>, “I”, “am”, “a”, “student”}. We illustrate this in Figure 2.3.

Training 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:

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

Single-timestep 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 $\{\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\}$. 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. 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: $l_t \rightarrow \boldsymbol{s}_t \rightarrow \{\boldsymbol{h}_t, \boldsymbol{W}_{hy}\} \rightarrow \{\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}\}$. To simplify the math, we will utilize several lemmas and corollaries provided in Appendix A.

First, from Eq. (2.6), we have:

$$d\boldsymbol{s}_t = \frac{\partial l_t}{\partial \boldsymbol{s}_t} = \frac{\partial}{\partial \boldsymbol{s}_t} \left(\boldsymbol{s}_t(y_t) - \log \sum_{y'} e^{\boldsymbol{s}_t(y')} \right) \tag{2.10}$$

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

$$\frac{\partial}{\partial \boldsymbol{s}_t(y)} \left(\boldsymbol{s}_t(y_t) - \log \sum_{y'} e^{\boldsymbol{s}_t(y')} \right) = \begin{cases} 1 - \boldsymbol{p}_t(y_t) & y = y_t \\ -\boldsymbol{p}_t(y) & y \neq y_t \end{cases} \tag{2.11}$$

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

$$d\boldsymbol{s}_t = \mathbf{1}_{y_t} - \boldsymbol{p}_t \tag{2.12}$$

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:

$$d\boldsymbol{h}_t = \boldsymbol{W}_{hy}^{\top} \cdot d\boldsymbol{s}_t \tag{2.13}$$

$$d\boldsymbol{W}_{hy} = d\boldsymbol{s}_t \cdot \boldsymbol{h}_t^{\top} \tag{2.14}$$

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:

$$\boldsymbol{h_t} = \sigma\left(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t\right) \tag{2.15}$$

Applying Lemma 2, we have:

$$d\boldsymbol{z}_t = \boldsymbol{T}_{\text{rnn}}^{\top} \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \tag{2.16}$$

$$d\boldsymbol{T}_{\text{rnn}} = (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{z}_t^{\top} \tag{2.17}$$

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. (2.16) and Eq. (2.17), 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     $d\boldsymbol{s}_t \leftarrow \boldsymbol{1}_{y_t} - \boldsymbol{p}_t$
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:

$$d\boldsymbol{x}_t = \boldsymbol{W}_{xh}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \tag{2.18}$$

$$d\boldsymbol{h}_{t-1} = \boldsymbol{W}_{hh}^\top \cdot (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \tag{2.19}$$

$$d\boldsymbol{W}_{xh} = (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{x}_t^\top \tag{2.20}$$

$$d\boldsymbol{W}_{hh} = (\sigma'(\boldsymbol{T}_{\text{rnn}}\boldsymbol{z}_t) \circ d\boldsymbol{h}_t) \cdot \boldsymbol{h}_{t-1}^\top \tag{2.21}$$

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

2.2.1 Better Training RNNs

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 Lemma 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 in terms of the largest eigen value of $\boldsymbol{W}_{hh}$ as well as the norms of these signals. Roughly speaking, if the largest eigen value is large enough, exploding gradients will be likely to happen. On the contrary, if the largest eigen value 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 has been 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), 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 through out 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:

$$\boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \sigma\left(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}\right)) \tag{2.22}$$

$$\boldsymbol{h}_t = \boldsymbol{c}_t \tag{2.23}$$

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 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:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \sigma\left(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}\right) \tag{2.24}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \sigma\left(\boldsymbol{c}_t\right)) \tag{2.25}$$

We note that, in Eq. (2.25), 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 an evidence, Greff et al. (2015) have shown that such a 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. All of these desiderata lead to the below LSTM formulation described in (Zaremba et al., 2014) in which σ is chosen to be $\tanh$:

$$\begin{pmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \hat{\boldsymbol{h}}_t \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \\ \mathrm{sigm} \\ \tanh \end{pmatrix} \begin{bmatrix} \boldsymbol{W}_{xi}\boldsymbol{W}_{hi} \\ \boldsymbol{W}_{xf}\boldsymbol{W}_{hf} \\ \boldsymbol{W}_{xo}\boldsymbol{W}_{ho} \\ \boldsymbol{W}_{xh}\boldsymbol{W}_{hh} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} \tag{2.26}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \hat{\boldsymbol{h}}_t \tag{2.27}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) \tag{2.28}$$

Following the same spirit as Eq. (2.15), we can be GPU-efficient with Eq. (2.26) since the 8 different submatrices is 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. (2.26) as:

$$\boldsymbol{u}_t = g(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \tag{2.29}$$

$$= g(\boldsymbol{T}_{\text{x}}\boldsymbol{x}_t + \boldsymbol{T}_{\text{h}}\boldsymbol{h}_{t-1}) \tag{2.30}$$

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:

$$(\boldsymbol{h}_t, \boldsymbol{c}_t) = f\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}\right) \tag{2.31}$$

In our case, the abstract function f is implemented by Eq. 2.26-2.28. Once $\boldsymbol{h}_t$ is computed, the prediction process is the same as that of RNNs which is given by Eq. 2.4-2.6. 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. (2.13) and Eq. (2.14), which computes $d\boldsymbol{h}_t$ and $d\boldsymbol{W}_{hy}$ respectively.

Given $d\boldsymbol{h}_t$, we now work backwark to derive other gradients. First, starting from Eq. (2.28) and by applying Lemma 3, we have:

$$d\boldsymbol{o}_t = \tanh(\boldsymbol{c}_t) \circ d\boldsymbol{h}_t \tag{2.32}$$

$$d\boldsymbol{c}_t = \tanh'(\boldsymbol{c}_{\boldsymbol{t}}) \circ \boldsymbol{o}_{\boldsymbol{t}} \circ d\boldsymbol{h}_t \tag{2.33}$$

Before backpropagating Eq. (2.27), 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 Corollary 2 to derive:

$$d\boldsymbol{f}_t = \boldsymbol{c}_{t-1} \circ d\boldsymbol{c}_t \tag{2.34}$$

$$d\boldsymbol{c}_{t-1} = \boldsymbol{f}_t \circ d\boldsymbol{c}_t \tag{2.35}$$

$$d\boldsymbol{i}_t = \hat{\boldsymbol{h}}_{\boldsymbol{t}} \circ d\boldsymbol{c}_t \tag{2.36}$$

$$d\hat{\boldsymbol{h}}_{\boldsymbol{t}} = \boldsymbol{i}_t \circ d\boldsymbol{c}_t \tag{2.37}$$

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. (2.30). In a similar manner as RNNs, Eq. 2.18-2.21, we arrive at:

$$d\boldsymbol{x}_t = \boldsymbol{T}_{\text{x}}^{\top} \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \tag{2.38}$$

$$d\boldsymbol{h}_{t-1} = \boldsymbol{T}_{\text{h}}^{\top} \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \tag{2.39}$$

$$d\boldsymbol{T}_{\text{x}} = (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{x}_t^{\top} \tag{2.40}$$

$$d\boldsymbol{T}_{\text{h}} = (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{h}_{t-1}^{\top} \tag{2.41}$$

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* or *sequence-to-sequence* 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

Algorithm 2: BPTT algorithm for LSTM

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

  `// Output backprop`

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

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

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

  `// 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

translation, one target word at a time, and hence, decomposes the log conditional probability as:

$$\log p(y|x) = \sum_{t=1}^{m} \log p\left(y_t | y_{<t}, \boldsymbol{s}\right) \tag{2.42}$$

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. 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, 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 LSTM as the recurrent unit. We show an example of such model in Figure 2.4 though it should be easy to extend to other RNN architectures. 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 language models as described in Section **??**: 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

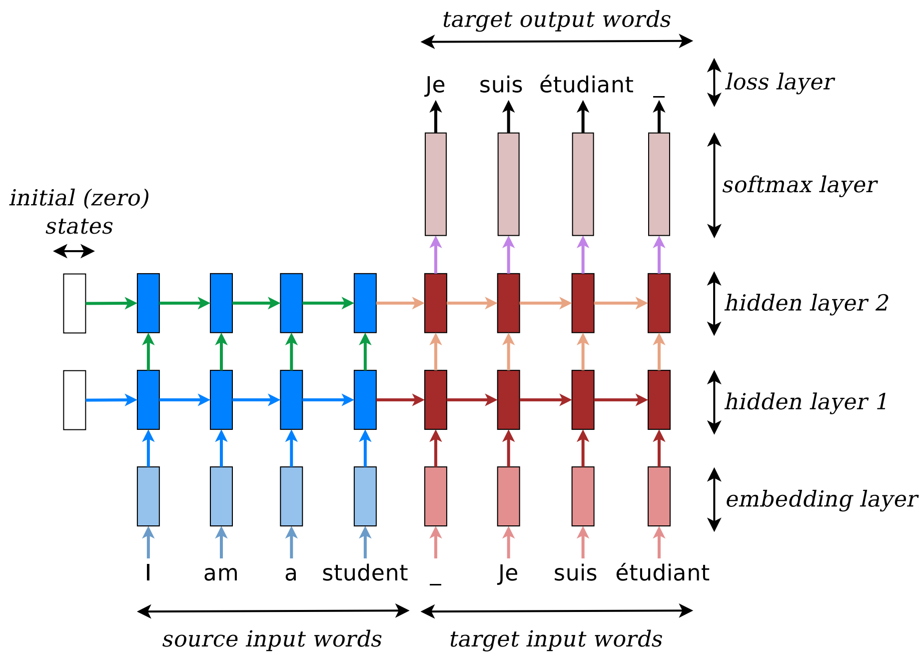

Figure 2.4: **Neural machine translation** – example of a deep recurrent architecture proposed by Sutskever et al. (2014) for translating a source sentence "I am a student" into a target sentence "Je suis étudiant". Here, "_" marks the end of a sentence.

weights with pretrained word representations, such as word2vec (Mikolov et al., 2013) and 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. 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 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

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

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. (2.29), over {encoder, decoder} $\times$ {1^{st}, 2^{nd} 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).

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

$$J = \sum\nolimits_{(x,y)\in\mathbb{D}} -\log p(y|x) \tag{2.43}$$

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 backpropating 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 in 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 both the source sentence x (length m_x) and the target sentence y (length m_y) together to form an input sequence s as shown in Line 1, which also includes the end-of-sentence marker "_". We first start with the encoder weights and initial states set to zero (Line 2-3). The algorithm switches to the decoder mode at time $m_x + 1$ (Line 5). The same LSTM codebase (Line 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 2.26-2.28, 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.

1 $s \leftarrow [x, \_, y, \_]$; `// Length of` s `is` $m_x + 1 + m_y + 1$
2 $\boldsymbol{W}_e, \boldsymbol{T}^{(1..L)}_{\text{lstm}} \leftarrow \boldsymbol{W}^{\text{encoder}}_e, \boldsymbol{T}^{\text{encoder}}_{\text{lstm}}$; `// Encoder weights`
3 $\boldsymbol{h}^{(1..L)}_0, \boldsymbol{c}^{(1..L)}_0 \leftarrow \boldsymbol{0}$; `// Zero init`
4 **for** $t = 1 \rightarrow (m_x + 1 + m_y)$ **do**
    `// Decoder transition`
5   **if** $t == (m_x + 1)$ **then**
6         $\boldsymbol{W}_e, \boldsymbol{T}^{(1..L)}_{\text{lstm}} \leftarrow \boldsymbol{W}^{\text{decoder}}_e, \boldsymbol{T}^{\text{decoder}}_{\text{lstm}}$;
7   **end**
    `// Multi-layer LSTM`
8     $\boldsymbol{h}^{(0)}_t \leftarrow$ `Emb_LookUp`$(s_t, \boldsymbol{W}_e)$;
9   **for** $l = 1 \rightarrow L$ **do**
10         $\boldsymbol{h}^{(l)}_t, \boldsymbol{c}^{(l)}_t \leftarrow$ `LSTM` $\left(\boldsymbol{h}^{(l)}_{t-1}, \boldsymbol{c}^{(l)}_{t-1}, \boldsymbol{h}^{(l-1)}_t, \boldsymbol{T}^{(l)}_{\text{lstm}}\right)$; `// LSTM hidden unit`
11   **end**
    `// Target-side prediction`
12   **if** $t \geq (m_x + 1)$ **then**
13         $l_t, \boldsymbol{p}_t \leftarrow$ `Predict`$(s_{t+1}, \boldsymbol{h}^{(L)}_t, \boldsymbol{W}_{hy})$;
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 start with the decoder weights and initialize all gradients to zero (Line 1-2). At time m_x, we switch to the encoder mode while saving the currently accumulated LSTM and embedding gradients for the decoder (Line 5-7). Thanks to the backpropagation procedure presented earlier for LSTM, we can simplify the core NMT gradient computation (Line 9-19) by making the following two referents: (a) `Predict_grad` (Line 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` (Line 5-15 of Algorithm 2) which computes gradients for inputs to

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

$\boldsymbol{W}_e, \boldsymbol{T}^{(1..L)}_{\text{lstm}} \leftarrow \boldsymbol{W}^{\text{decoder}}_e, \boldsymbol{T}^{\text{decoder}}_{\text{lstm}}$; `// Decoder weights`
$d\boldsymbol{h}^{(1..L)}, d\boldsymbol{c}^{(1..L)}, d\boldsymbol{T}^{(1..L)}_{\text{lstm}}, d\boldsymbol{W}_e, d\boldsymbol{W}_{hy} \leftarrow \boldsymbol{0}$; `// Zero init`
for $t = (m_x + 1 + m_y) \rightarrow 1$ **do**
    `// Encoder transition`
    if $t == m_x$ **then**
        $\boldsymbol{W}_e, \boldsymbol{T}^{(1..L)}_{\text{lstm}} \leftarrow \boldsymbol{W}^{\text{encoder}}_e, \boldsymbol{T}^{\text{encoder}}_{\text{lstm}}$;
        $d\boldsymbol{W}^{\text{decoder}}_e, d\boldsymbol{T}^{\text{decoder}}_{\text{lstm}} \leftarrow d\boldsymbol{W}_e, d\boldsymbol{T}^{(1..L)}_{\text{lstm}}$; `// Save decoder gradients`
        $d\boldsymbol{T}^{(1..L)}_{\text{lstm}}, d\boldsymbol{W}_e \leftarrow \boldsymbol{0}$;
    end
    `// Target-side prediction`
    if $t \geq (m_x + 1)$ **then**
        $d\boldsymbol{h}, d\boldsymbol{W} \leftarrow \texttt{Predict\_grad}(s_{t+1}, \boldsymbol{p}_t, \boldsymbol{h}^{(L)}_t, \boldsymbol{W}_{hy})$;
        $d\boldsymbol{h}^{(L)} \leftarrow d\boldsymbol{h}^{(L)} + d\boldsymbol{h}$;
        $d\boldsymbol{W}_{hy} \leftarrow d\boldsymbol{W}_{hy} + d\boldsymbol{W}$;
    end
    `// Multi-layer LSTM`
    for $l = L \rightarrow 1$ **do**
        $d\boldsymbol{h}^{(l)}, d\boldsymbol{c}^{(l)}, d\boldsymbol{x}, d\boldsymbol{T} \leftarrow \texttt{LSTM\_grad}\left(d\boldsymbol{h}^{(l)}, d\boldsymbol{c}^{(l)}, \boldsymbol{h}^{(l)}_{t-1}, \boldsymbol{c}^{(l)}_{t-1}, \boldsymbol{h}^{(l-1)}_t, \boldsymbol{T}^{(l)}_{\text{lstm}}\right)$;
        $d\boldsymbol{h}^{(l-1)} \leftarrow d\boldsymbol{h}^{(l-1)} + d\boldsymbol{x}$;
        $d\boldsymbol{T}^{(l)}_{\text{lstm}} \leftarrow d\boldsymbol{T}^{(l)}_{\text{lstm}} + d\boldsymbol{T}$;
    end
    $d\boldsymbol{W}_e \leftarrow \texttt{Emb\_grad\_update}(s_t, d\boldsymbol{h}^{(0)}, d\boldsymbol{W}_e)$;
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`

LSTM and the LSTM weights per layer $\boldsymbol{T}^{(l)}_{\text{lstm}}$. It is important to note that in Lines 11 and 16 of Algorithm 4, we add the gradients (flowed 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. Lastly, in Line 19, we perform sparse updates on the corresponding embedding matrix for participating words only.

2.3.1 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.

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 Line 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

[7]We often set α to 1.5

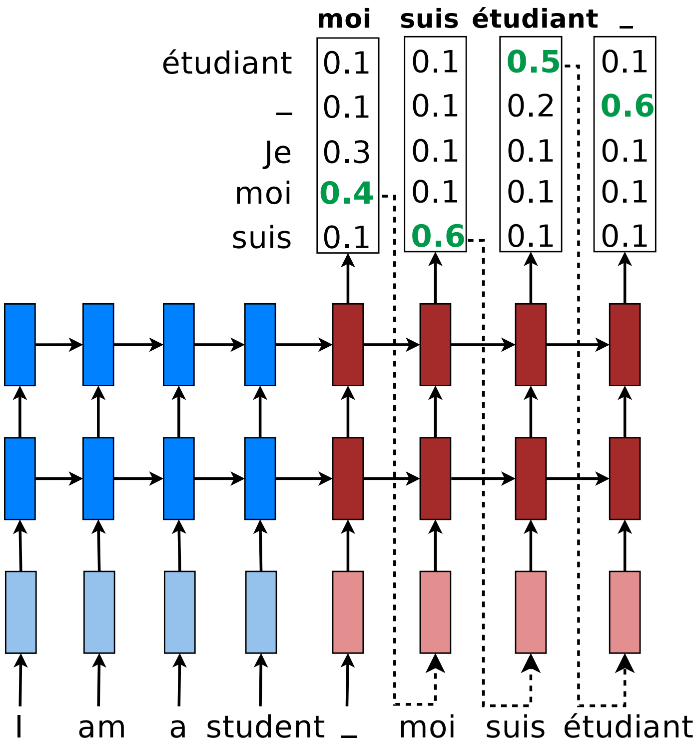

Figure 2.5: **Greedy Decoding** – example of how a trained NMT model produces a translation for a source sentence "I am a student" using greedy search.

to produce the index of the translation word at that timestep (Line 7). The process ends when we have produced the marker "_" as a translation word or when the translation length exceeds the length threshold.

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 better result, a more popular strategy is to use *beam-search* decoding algorithm which has been the core of phrase-based statistical machine translation for years (Koehn et al., 2003). However, unlike phrase-based SMT, NMT has a much simpler beam-search decoding algorithm since it generates translations word-by-word from left to right. 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

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     $\boldsymbol{h}_t, \boldsymbol{c}_t \leftarrow \texttt{MultiLayerLSTM}\big(\boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, y_t, \boldsymbol{W}_e^{\text{decoder}}, \boldsymbol{T}_{\text{lstm}}^{\text{decoder}}\big)$;
6     $\boldsymbol{p}_t \leftarrow \texttt{Softmax}(\boldsymbol{h}_t^{(L)}, \boldsymbol{W}_{hy})$;
7     $y_{t+1} \leftarrow \text{argmax}_i \, \boldsymbol{p}_t(i)$; // ***Greedy*** choice
8   **if** $y_{t+1} == \text{Index}(\_)$ **then** // Ending condition
9   break;
10   **end**
11     $t \leftarrow t + 1$
12 **end**
13 **return** $y_{2..t}$

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 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 > 1000.

Lastly, 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 be used for both greedy and beam-search decoders as if we decode from a single model.

Chapter 3

Copy Mechanisms

Chapter 4

Attention Mechanisms

Chapter 5

Hybrid Models

Chapter 6

NMT Future

Chapter 7

Conclusion

Appendix A

Miscellaneous

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

$$diag(\boldsymbol{u}) \cdot \boldsymbol{v} = \boldsymbol{u} \circ \boldsymbol{v} \tag{A.1}$$

Lemma 2. *Let l be a loss value that in which we already knew 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:*

$$d\boldsymbol{h} = \boldsymbol{W}^\top \cdot (f'(\boldsymbol{W}\boldsymbol{h}) \circ d\boldsymbol{v}) \tag{A.2}$$

$$d\boldsymbol{W} = (f'(\boldsymbol{W}\boldsymbol{h}) \circ d\boldsymbol{v}) \cdot \boldsymbol{h}^\top \tag{A.3}$$

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

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

Let $\boldsymbol{w}_i^\top$ be the i$^{\text{th}}$ row vector of matrix $\boldsymbol{W}$ and v_i, z_i be the i$^{\text{th}}$ 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 2, when f is an identity function, i.e., $\boldsymbol{v} = \boldsymbol{W}\boldsymbol{h}$, we have:*

$$d\boldsymbol{h} = \boldsymbol{W}^\top \cdot d\boldsymbol{v} \tag{A.4}$$

$$d\boldsymbol{W} = d\boldsymbol{v} \cdot \boldsymbol{h}^\top \tag{A.5}$$

Lemma 3. *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:*

$$d\boldsymbol{u} = f(\boldsymbol{v}) \circ d\boldsymbol{s} \tag{A.6}$$

$$d\boldsymbol{v} = f'(\boldsymbol{v}) \circ \boldsymbol{u} \circ d\boldsymbol{s} \tag{A.7}$$

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

$$d\boldsymbol{u} = \boldsymbol{v} \circ d\boldsymbol{s} \tag{A.8}$$

$$d\boldsymbol{v} = \boldsymbol{u} \circ d\boldsymbol{s} \tag{A.9}$$

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.

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

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

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.

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.

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*.

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*.

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

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

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*.

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*.

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

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

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*.

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* .

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

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

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.

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.

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.

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*.

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.

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*.

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*.

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

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*.

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.

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*.

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*.

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*.

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

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.

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.