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

September 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. It feeds on brainwave energy ... 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 and rich in categories with about 6000 to 7000 languages spoken worldwide.[1] 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 [38] from the original phiosophical ideas of universal languages in the seventeen century to the first practical instances of MT in the twentieth century, e.g., one proposal by Weaver [95]. Despite several excitement moments that led to hopes that MT will be solved "very soon", e.g., the 701 translator[2] developed by scientists at George Town and IBM in the 1950s or a simple vector-space transformation technique[3] proposed by Google researchers at the beginning of the twenty-first century, MT remains to be an extremely challenging problem.[4] To understand why MT is difficult, let us trace through one "evolution" path of MT which crosses

[1] http://www.linguisticsociety.org/content/how-many-languages-are-there-world

[2] http://www-03.ibm.com/ibm/history/exhibits/701/701_translator.html

[3] https://www.technologyreview.com/s/519581/how-google-converted-language-transl

[4] http://www.huffingtonpost.com/nataly-kelly/why-machines-alone-cannot-translati

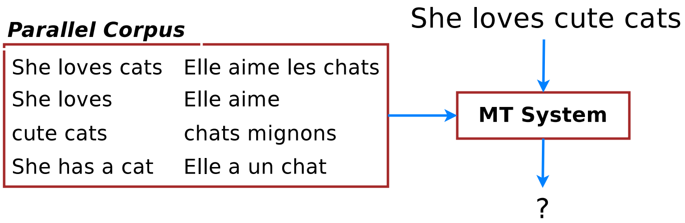


Figure 1.1: **Machine translation** (MT) – a general setup of MT. Systems build translation models from parallel corpora to translate new unseen sentences, e.g., "She loves cute cats".

through techniques that are used extensively in commercial MT systems.

1.1 Machine Translation Development

Modern statistical MT started out with a seminal work by IBM scientists [12]. The proposed technique requires minimal linguistic content and only needs a *parallel corpus*, i.e., a set of pairs of sentences that are translations of one another, to train machine learning algorithms to tackle the translation problem. Such a language-independent setup is illustrated in Figure 1.1 and remains to be the general approach for nowadays MT systems. For over twenty years since the IBM seminal paper, approaches in MT such as [14, 16, 22, 47, 48, 50, 72], are, by and large, similar according to the following two-stage process (see Figure 1.2). First, source sentences are broken into chunks which can be translated in isolation by looking up a "dictionary", or more formally a *translation model*. Translated target words and phrases are then put together to form coherent and natural-sounding sentences by consulting a *language model* (LM) on which sequences of words, i.e., n-*grams*, are likely to go with one another.

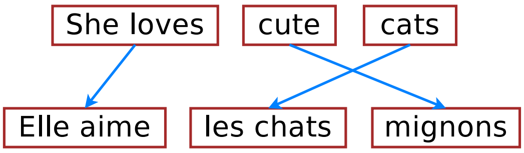


Figure 1.2: **Phrase-based machine translation** (MT) – example of how phrase-based MT systems translate a source sentence "She loves cute cats" into a target sentence "Elle aime les chats mignons": sentences are split into chunks and phrases are translated.

The aforementioned approach, while has been successfully deployed in many commercial systems, 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. Second, it is slightly “strange” that language models (LMs), despite being a key component in the MT pipeline, utilize context information that is both short, consisting of only a handful of previous words, and target-only, never looking at the source words. These shortcomings in LMs gives rise to a new wave of *hybrid* systems which aim to empower phrase-based MT with neural network components, most notably neural probabilistic language models (NPLMs).

NPLMs were first proposed by Bengio et al. [8] as a way to combat the “curse” of dimensionality suffered by traditional LMs. In traditional LMs, one has to explicitly store and handle all possible n-grams occurred in a training corpus, the number of which quickly becomes enormous. As a result, existing MT systems often limit themselves to use only short, e.g., 5-gram, LMs [35], which capture little context and cannot generalize well to unseen n-grams. NPLMs address these concerns by using distributed representations of words and not having to explicitly store all enumerations of words. As a result, many MT systems, [58, 82, 93], inter alia, start adopting NPLMs alongside with traditional LMs. To make NPLMs even more powerful, recent work [1, 19, 83, 86] propose to condition on source words beside the target context to lower uncertainty in predicting next words (see Figure 1.3).[5]

These hybrid MT systems with NPLM components, while having addressed shortcomings of traditional phrase-based MT, still translate locally and fail to capture long-range dependencies. For example, in Figure 1.3, the source-conditioned NPLM 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”. More problematically, the entire MT pipeline is already complex with different components needed to be tuned separatedly, e.g., translation models, language models, reordering models, etc.; now, it becomes even worse as different neural components are incorporated. Neural Machine Translation to the rescue!

[5] In [19], the authors have constructed a model that conditions on 3 target words and 11 source words, effectively building a 15-gram LM.

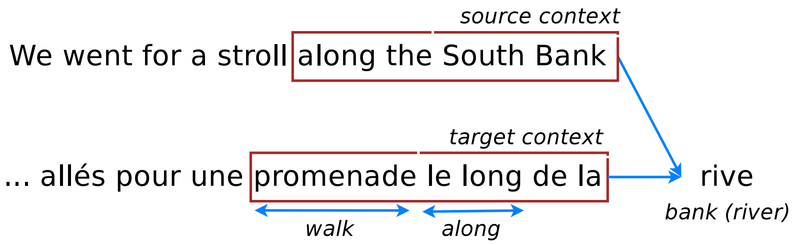

source context
We went for a stroll along the South Bank
target context
... allés pour une promenade le long de la
rive
bank (river)
walk
along

Figure 1.3: **Source-conditioned neural probabilistic language models** (NPLMs) – example of a source-conditioned NPLM proposed by Devlin et al. [19]. To evaluate a 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 NPLMs [8], but also utilizes source context "along the South Bank" to lower uncertainty in its prediction.

Neural Machine Translation (NMT) is a new approach to translating text from one language into another that captures long-range dependencies in sentences and generalizes better to unseen texts. The core of NMT is a single deep neural network with hundreds of millions of neurons that learn to directly map source sentences to target sentences [17, 44, 90]. This is often referred as the sequence-to-sequence or encoder-decoder approach.[6] NMT is appealing since it is conceptually simple and can be trained end-to-end. NMT translates as follows: an *encoder* reads through the given source words one by one until the end, and then, a *decoder* starts emitting one target word at a time until a special end-of-sentence symbol is produced. We illustrate this process in Figure 1.4.

Such simplicity leads to several advantages. NMT requires minimal domain knowledge: it only assumes access to sequences of source and target words as training data and learns to directly map one into another. NMT beam-search decoders that generate words from left to right can be easily implemented, unlike the highly intricate decoders in standard MT [47]. Lastly, the use of recurrent neural networks (RNNs) allow NMT to generalize well to very long word sequences while not having to explicitly store any gigantic phrase tables or language models as in the case of standard MT.

[6] Forcada and Neco [26] wrote the very first paper on sequence-to-sequence models for translation!

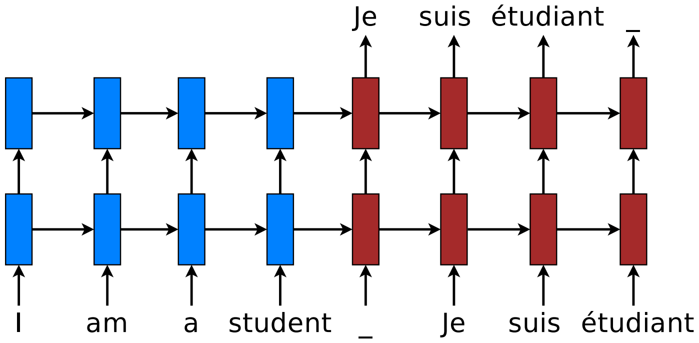


Figure 1.4: **Neural machine translation** – example of a deep recurrent architecture proposed by Sutskever et al. [90] for translating a source sentence "I am a student" into a target sentence "Je suis étudiant". Here, "_" marks the end of a sentence.

1.2 Thesis Outline

Despite all the aforementioned advantages and potentials, the early NMT architecture [17, 90] still has many drawbacks. In this thesis, I will highlight three problems pertaining to the existing NMT model, namely the *vocabulary size*, the *sentence length*, and the *language complexity* issues. Each chapter is devoted to solving each of these problems in which 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 [60], English-German [55, 59], and English-Czech [56]. 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 [61]; and (2) whether we can compress NMT for mobile devices [84]. 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 for NMT future 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 conventional 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, we propose simple and effective techniques to address this *vocabulary size* problem through teaching NMT to "copy" words from source to target. Specifically, we 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. Our 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, our NMT system is the first to surpass the best result achieved on a WMT'14 contest task.

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. [3] 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. We demonstrate the effectiveness of both approaches on the WMT translation tasks between English and German in both directions. With local attention, we achieve a significant gain of 5.0 BLEU points over non-attentional systems that already incorporate known techniques such as dropout. Our 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.

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. We build hybrid systems that translate mostly at the *word* level and consult the *character* components for rare words. Our 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. Our best system achieves a new state-of-the-art result with 20.7 BLEU score. We demonstrate that our character models can successfully learn to not only generate well-formed words for Czech, a highly-inflected language with a very complex vocabulary, but also build correct representations for English source words.

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.

For the first question, we 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. Our 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, we have established a new *state-of-the-art* result in constituent parsing with 93.0 F_1 by utilizing translation data. Lastly, we reveal interesting properties of the two unsupervised learning objectives, autoencoder and skip-thought, in the MTL context: autoencoder helps less in terms of perplexities but more on BLEU scores compared to skip-thought.

For the second question, we 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. We demonstrate the efficacy of weight pruning as a compression technique for a state-of-the-art NMT system. We show that an NMT model with over 200 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. Our main result is that with *retraining*, we can recover and even surpass the original performance with an 80%-pruned model.

Chapter 2

Background

For neural machine translation, it all started from language modeling.

Thang Luong.

Language modeling plays an indispensable role in ensuring that machine translation systems produce fluent target sentences and has always been an active area of research. Despite much effort in improving traditional n-gram language models [25, 35, 36, 75, 79, 87, 91], traditional LMs inherently can only handle short contexts of a few words. Approaches to building neural probabilistic language models (NPLMs) using feed-forward networks such as those initiated by Bengio et al. [8] and enhanced by others [6, 68, 69, 71] have addressed that drawback to model longer contexts. Still, NPLMs can only capture fixed-length contexts and is incapable of handling variable-length sequences, which is the case for sentences. Recurrent neural networks (RNNs) come in handy as a powerful and expressive architecture to handle sequential data and have successfully been applied to the language modeling task [64, 65, 66]. By viewing RNNs as generative models [89] that can produce texts and by pushing another step towards conditioning RNNs on source sentences, recent works [17, 44, 90] have started a new line of resesearch in machine translation, namely Neural Machine Translation (NMT). NMT is technically a source-conditioned NPLM that can be trained end-to-end.

In this chapter, we provide background knowledge on two main topics, RNN and NMT.

We first go through the basics of RNNs, explaining how they can be used to model sentences. Then, we delve into details of one particular type of RNNs, the Long Short-term Memory, that makes training RNNs easier. Given RNNs as a building block, we discuss NMT together with tips and tricks for better training and testing NMT.

2.1 Recurrent Neural Network

Recurrent Neural Network (RNNs) [24] are models that help understand the temporal aspect as well as build up representations for sequential data using a dynamic memory structure. At the surface form, 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 beauty of RNNs lies in the fact that it can capture the dynamics of an arbitrarily long sequence without having to increase its modeling capacity unlike the case of feedforward network which can only model relationship within a fixed-length sequence. 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:

$$\boldsymbol{h}_t = f\left(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\right) \tag{2.1}$$

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 $\tanh$.[1]

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}) \tag{2.2}$$

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 languages,

[1] There could also be an optional bias term in Eq. (2.2).

a probability distribution $\boldsymbol{p}$ over a set of output classes Y is derived as follows[2]:

$$\boldsymbol{s}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t \tag{2.3}$$

$$\boldsymbol{p}_t = \mathrm{softmax}(\boldsymbol{s}_t) \tag{2.4}$$

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. (2.3) 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 $\mathrm{softmax}$ 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.

$$\boldsymbol{p}_t(y) = \frac{\mathrm{e}^{\boldsymbol{s}_t(y)}}{\sum_{y' \in Y} \mathrm{e}^{\boldsymbol{s}_t(y')}} \tag{2.5}$$

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 2.1 Draw a picture on general RNNs, which enables RNNs to handle arbitrarily long sequences.

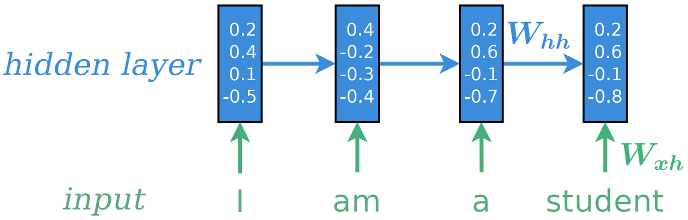


Figure 2.1: **Recurrent neural networks** – example of a recurrent neural network that processes a sequence of input words "I am a student" to build up hidden representations as input symbols are consumed. The recurrent $\boldsymbol{W_{hh}}$ and feed-forward $\boldsymbol{W_{xh}}$ weights are shared across timesteps.

[2] For the real-valued case, we refer readers to mixture density models [10] which have been applied to RNN training, e.g., for hand-writing synthesis [29].

Next, we discuss the training and testing phases of RNNs from a slightly more focused angle, the language learning aspect. For more details on RNNs, we refer readers to the following resources [45, 63, 88].

2.1.1 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 2.2 Draw an RNN with embedding.

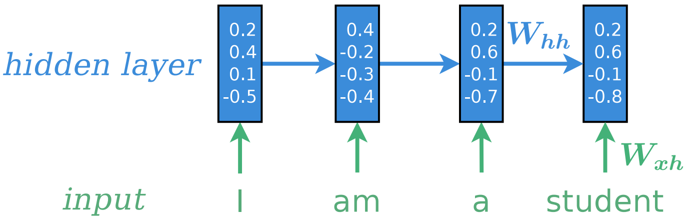


Figure 2.2: **Recurrent language models** – example of a recurrent neural network that processes a sequence of input words "I am a student" to build up hidden representations as input symbols are consumed. The recurrent $\boldsymbol{W_{hh}}$ and feed-forward $\boldsymbol{W_{xh}}$ weights are shared across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

In language modeling (LM), the task 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:

$$p(y) = \prod_{i=1}^{m} p(y_i | y_{<i}) \tag{2.6}$$

In the above formula, each of the individual term $p(y_i|y_{<i})$ is the conditional probability of the current word y_i given previous words $y_{<i}$, also referred as the *context* or the *history*. To model these conditional probabilities, traditional n-gram as well as feedforward-based neural language models have to resort to the Markovian assumption to model only a fixed window of context, i.e., $p(y_i|y_{i-n+1}, \ldots, y_{i-1})$. An RNN-based language model naturally lends itself to model the full history as we shall see now.

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 2.2.

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. (2.2) and Eq. (2.3) 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 [28] 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. (5.2), 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. (2.4) 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. (2.3), 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 $\boldsymbol{T}_{\text{rnn}} = [\boldsymbol{W_{xh}} \boldsymbol{W_{hh}}]$ and $\boldsymbol{z}_t = [\boldsymbol{x}_t; \boldsymbol{h}_{t-1}]$, so the RNN formulation in Eq. (2.2) 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.[3]

Backpropagation Through Time (BPTT) Having defined a single-timestep backpropagation procedure, we are now ready to go through the BPTT algorithm [80, 96]. Inspired by Sutskever [88], we summarize the BPTT algorithm for RNNs below with the following remarks: (a) Lines 3, 5, 6, 7 accumulate the gradients of RNN weights $\{\boldsymbol{W}_{hy}, \boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_e\}$

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

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

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

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

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

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 $\boldsymbol{W}_e$;[4] and (c) Line 4 accumulates gradients for the current hidden state $\boldsymbol{h}_t$ by considering two paths, a "vertical" one from the current loss at time t and a "recurrent" one from the timestep $t+1$ which was set in Line 8 earlier.

Algorithm 1: BPTT algorithm for "vanilla" RNNs

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

2.1.2 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. [7]. 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

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

rigorous and in-depth analyses in [7, 37, 62, 74]. 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. [74].

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 [63] 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. [74]. 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 [52, 94], hierarchical architectures [23], leaky integrators [39], second-order methods [62], and regularization [74], to name a few; also, see [9] for a comparison of some of these techniques. Among all, Long Short-term Memory (LSTM), invented by Hochreiter and Schmidhuber [37], 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 [29, 31, 32]. 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 [33, 42]. As such, in this section and through out this thesis, we will stick to the formulation described in [98].

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 [9, 88] 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. [33] 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 [98] in which σ is chosen to be $\tanh$:

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

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

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

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. 2.3-5.2. The training objective in Eq. (2.8) remains unchanged as well.

LSTM Backpropagation Since the prediction procedure is the same, LSTM backpropagation pipeline mimics that of RNNs up to Eq. (2.13) 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:

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

$$d\boldsymbol{c}_t = \tanh'(\boldsymbol{c_t}) \circ \boldsymbol{o_t} \circ d\boldsymbol{h}_t \tag{2.33}$$

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:

Algorithm 2: BPTT algorithm for LSTM

```
for $t = T \rightarrow 1$ do
    // Output backprop
    $d\boldsymbol{s}_t \leftarrow \mathbf{1}_{y_t} - \boldsymbol{p}_t$
    $d\boldsymbol{W}_{hy} \leftarrow d\boldsymbol{W}_{hy} + d\boldsymbol{s}_t \cdot \boldsymbol{h}_t^\top$
    $d\boldsymbol{h}_t \leftarrow d\boldsymbol{h}_t + \boldsymbol{W}_{hy}^\top \cdot d\boldsymbol{s}_t$
    // LSTM backprop
    $d\boldsymbol{o}_t \leftarrow \tanh(\boldsymbol{c}_t) \circ d\boldsymbol{h}_t$
    $d\boldsymbol{c}_t \leftarrow d\boldsymbol{c}_t + \tanh'(\boldsymbol{c_t}) \circ \boldsymbol{o_t} \circ d\boldsymbol{h}_t$ ;        // Already included $d\boldsymbol{c}_{t+1}$
    $d\boldsymbol{f}_t \leftarrow \boldsymbol{c}_{t-1} \circ d\boldsymbol{c}_t$
    $d\boldsymbol{i}_t \leftarrow \hat{\boldsymbol{h}}_{\boldsymbol{t}} \circ d\boldsymbol{c}_t$
    $d\hat{\boldsymbol{h}}_{\boldsymbol{t}} \leftarrow \boldsymbol{i}_t \circ d\boldsymbol{c}_t$
    $d\boldsymbol{c}_{t-1} \leftarrow \boldsymbol{f}_t \circ d\boldsymbol{c}_t$ ;        // Compute $d\boldsymbol{c}_{t-1}$
    $d\boldsymbol{u}_t = [d\boldsymbol{i}_t; d\boldsymbol{f}_t; d\boldsymbol{o}_t; d\hat{\boldsymbol{h}}_{\boldsymbol{t}}]$
    $d\boldsymbol{T}_{\text{x}} \leftarrow (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{x}_t^\top$
    $d\boldsymbol{T}_{\text{h}} \leftarrow (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t) \cdot \boldsymbol{h}_{t-1}^\top$
    // Input backprop
    $d\boldsymbol{x}_t \leftarrow \boldsymbol{T}_{\text{x}}^\top \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t)$
    $d\boldsymbol{h}_{t-1} \leftarrow \boldsymbol{T}_{\text{h}}^\top \cdot (g'(\boldsymbol{T}_{\text{lstm}}\boldsymbol{z}_t) \circ d\boldsymbol{u}_t)$
end
```

2.2 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 [17, 44, 90]. The *encoder* computes a representation $\boldsymbol{s}$ for each source sentence. Based on that source representation, the *decoder* generates a translation, one target word at a time, and hence, decomposes the log conditional probability as:

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

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 [37], or a gated recurrent unit (GRU) [17]. In general, for the encoder, almost any architecture can be used since we have fully observed the source sentence. For example, Kalchbrenner and Blunsom [44] 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 2.3 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 (2.1.1): 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.

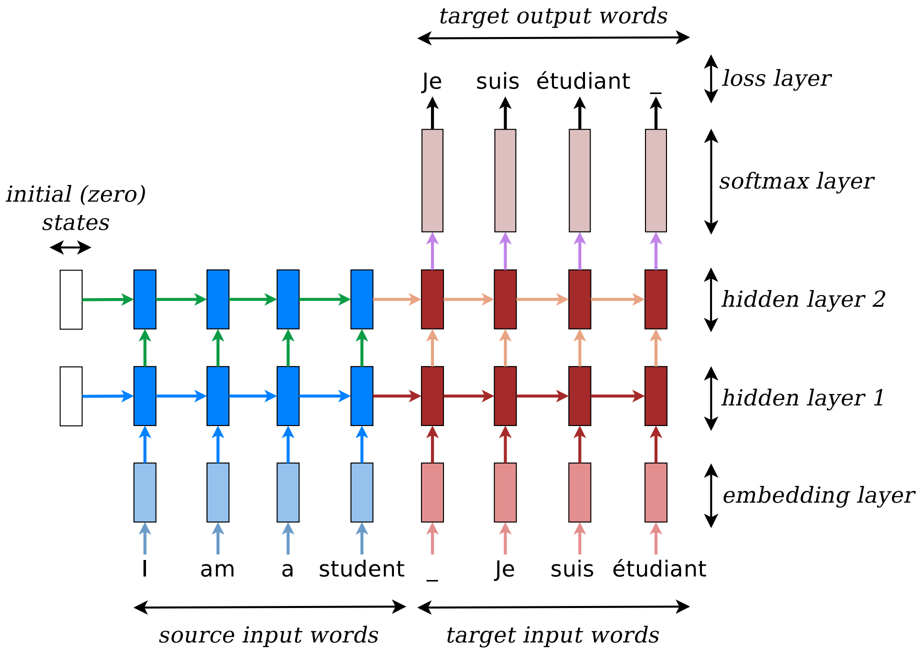


Figure 2.3: **Neural machine translation** – example of a deep recurrent architecture proposed by Sutskever et al. [90] for translating a source sentence “I am a student” into a target sentence “Je suis étudiant”. Here, “_” marks the end of a sentence.

In more detail, at the bottom layer, the encoder and decoder RNNs receive as *input* the following: first, the source sentence, then a boundary marker “_” which indicates the transition from the encoding to the decoding mode, and the target sentence. Given these discrete words, the model looks up the source and target embeddings to retrieve the corresponding word representations. For this *embedding layer* to work, a vocabulary is chosen for each language, and often the top V frequent words are selected. These embedding weights, one set per language, are learned during training. While one can choose to initialize embedding weights with pretrained word representations, such as word2vec [67] and Glove [76], 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.[5] In Figure 2.3, 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 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$\} \times \{1^{\text{st}}, 2^{\text{nd}}$ 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. (2.3) and Eq. (2.4).

Training Training neural machine translation is similar to training a recurrent language model that we have discussed in Section (2.1) 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,

[5] This is not the only way to initialize the decoder, e.g., Cho et al. [17] connect the last encoder state to every timesteps in the decoder.

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 2.3-2.4 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

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
        $d\boldsymbol{h}, d\boldsymbol{W} \leftarrow$ 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
        $d\boldsymbol{h}^{(l)}, d\boldsymbol{c}^{(l)}, d\boldsymbol{x}, d\boldsymbol{T} \leftarrow$ 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
    $d\boldsymbol{W}_e \leftarrow$ 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
```

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 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.

2.2.1 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 2.4. 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).

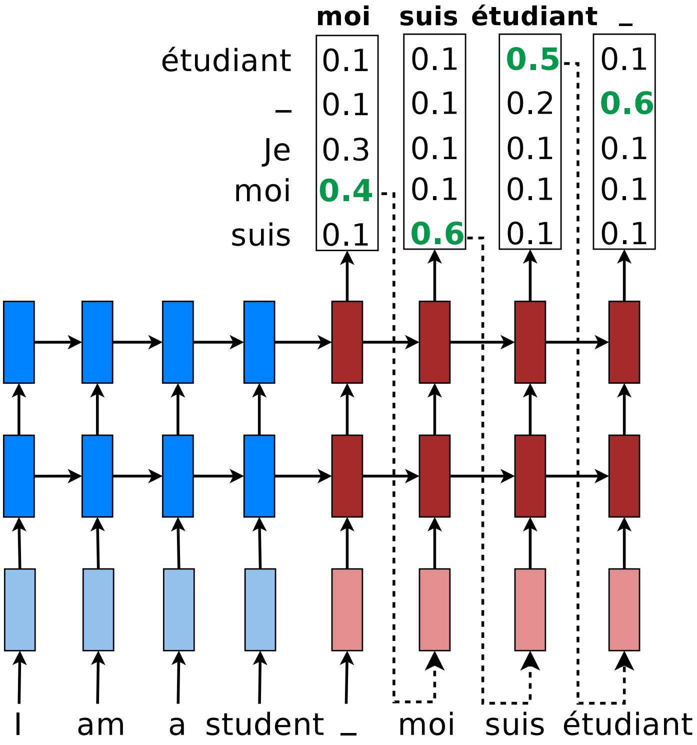


Figure 2.4: **Greedy Decoding** – example of how a trained NMT model produces a translation for a source sentence "I am a student" using greedy search.

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 $\alpha * m_x$.[6] 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 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 [90]. 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 [47]. However, unlike phrase-based SMT, NMT has a much

[6] We often set α to 1.5

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      $\boldsymbol{h}_t, \boldsymbol{c}_t \leftarrow \texttt{MultiLayerLSTM}\left(\boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, y_t, \boldsymbol{W}_e^{\text{decoder}}, \boldsymbol{T}_{\text{lstm}}^{\text{decoder}}\right)$ ;
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 $y_{2..t}$
```

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 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. [90] 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

Neural Machine Translation (NMT) is a novel approach to MT that has achieved promising results [3, 17, 40, 44, 90]. An NMT system is a conceptually simple large neural network that reads the entire source sentence and produces an output translation one word at a time. NMT systems are appealing because they use minimal domain knowledge which makes them well-suited to any problem that can be formulated as mapping an input sequence to an output sequence [90]. In addition, the natural ability of neural networks to generalize implies that NMT systems will also generalize to novel word phrases and sentences that do not occur in the training set. In addition, NMT systems potentially remove the need to store explicit phrase tables and language models which are used in conventional systems. Finally, the decoder of an NMT system is easy to implement, unlike the highly intricate decoders used by phrase-based systems [47].

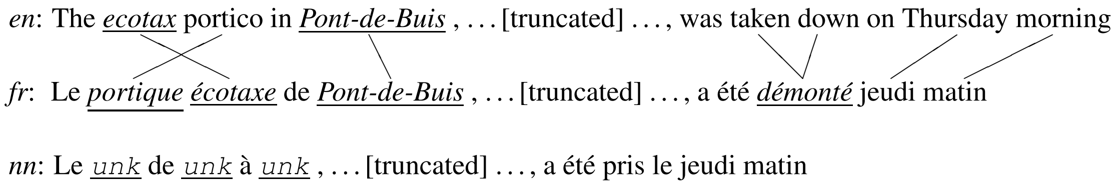
Despite these advantages, conventional NMT systems are incapable of translating rare words because they have a fixed modest-sized vocabulary[1] which forces them to use the *unk* symbol to represent the large number of out-of-vocabulary (OOV) words, as illustrated in Figure 3.1. Unsurprisingly, both Sutskever et al. [90] and Bahdanau et al. [3] have observed that sentences with many rare words tend to be translated much more poorly than sentences containing mainly frequent words. Standard phrase-based systems [14, 16, 22,

[1] Due to the computationally intensive nature of the softmax, NMT systems often limit their vocabularies to be the top 30K-80K most frequent words in each language. However, Jean et al. [40] has very recently proposed an efficient approximation to the softmax that allows for training NTMs with very large vocabularies. As discussed in Section 5.2, this technique is complementary to ours.

48], on the other hand, do not suffer from the rare word problem to the same extent because they can support a much larger vocabulary, and because their use of explicit alignments and phrase tables allows them to memorize the translations of even extremely rare words.

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

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



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

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

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

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

3.1 Neural Machine Translation

A neural machine translation system is any neural network that maps a source sentence, $s_1, \ldots, s_n$, to a target sentence, $t_1, \ldots, t_m$, where all sentences are assumed to terminate with a special "end-of-sentence" token $<$eos$>$. More concretely, an NMT system uses a neural network to parameterize the conditional distributions

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

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

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

There are many ways to parameterize these conditional distributions. For example, Kalchbrenner and Blunsom [44] used a combination of a convolutional neural network and a recurrent neural network, Sutskever et al. [90] used a deep Long Short-Term Memory (LSTM) model, Cho et al. [17] used an architecture similar to the LSTM, and Bahdanau et al. [3] used a more elaborate neural network architecture that uses an attentional mechanism over the input sequence, similar to Graves [29] and Graves et al. [30].

In this work, we use the model of Sutskever et al. [90], which uses a deep LSTM to encode the input sequence and a separate deep LSTM to output the translation. The encoder reads the source sentence, one word at a time, and produces a large vector that represents the entire source sentence. The decoder is initialized with this vector and generates a translation, one word at a time, until it emits the end-of-sentence symbol <eos>.

None the early work in neural machine translation systems has addressed the rare word problem, but the recent work of Jean et al. [40] has tackled it with an efficient approximation to the softmax to accommodate for a very large vocabulary (500K words). However, even with a large vocabulary, the problem with rare words, e.g., names, numbers, etc., still persists, and Jean et al. [40] found that using techniques similar to ours are beneficial and complementary to their approach.

3.2 Rare Word Models

Despite the relatively large amount of work done on pure neural machine translation systems, there has been no work addressing the OOV problem in NMT systems, with the en: The $\underline{\texttt{unk}_1}$ portico in $\underline{\texttt{unk}_2}$ ...

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

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

notable exception of Jean et al. [40]'s work mentioned earlier.

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

We present three annotation strategies that can easily be applied to any NMT system [17, 44, 90]. We treat the NMT system as a black box and train it on a corpus annotated by one of the models below. First, the alignments are produced with an unsupervised aligner. Next, we use the alignment links to construct a word dictionary that will be used for the word translations in the post-processing step.[2] If a word does not appear in our dictionary, then we apply the identity translation.

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

3.2.1 Copyable Model

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

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

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

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

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

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

3.2.2 Positional All Model (PosAll)

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

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

3.2.3 Positional Unknown Model (PosUnk)

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

en: The $\underline{\texttt{unk}}$ portico in $\underline{\texttt{unk}}$ ...
fr: Le $\underline{\texttt{unkpos}}_1$ $\underline{\texttt{unkpos}}_{-1}$ de $\underline{\texttt{unkpos}}_1$ ...

Figure 3.4: **Positional Unknown Model** – an example of the PosUnk model: only aligned unknown words are annotated with the $\texttt{unkpos}_d$ tokens.

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

3.3 Experiments

We evaluate the effectiveness of our OOV models on the WMT'14 English-to-French translation task. Translation quality is measured with the BLEU metric [73] on the newstest2014 test set (which has 3003 sentences).

3.3.1 Training Data

To be comparable with the results reported by previous work on neural machine translation systems [3, 17, 90], we train our models on the same training data of 12M parallel sentences

(348M French and 304M English words), obtained from [81]. The 12M subset was selected from the full WMT'14 parallel corpora using the method proposed in Axelrod et al. [2].

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

We annotate our training data using the three schemes described in the previous section. The alignment is computed with the Berkeley aligner [51] using its default settings. We discard sentence pairs in which the source or the target sentence exceed 100 tokens.

3.3.2 Training Details

Our training procedure and hyperparameter choices are similar to those used by Sutskever et al. [90]. In more details, we train multi-layer deep LSTMs, each of which has 1000 cells, with 1000 dimensional embeddings. Like Sutskever et al. [90], we reverse the words in the source sentences which has been shown to improve LSTM memory utilization and results in better translations of long sentences. Our hyperparameters can be summarized as follows: (a) the parameters are initialized uniformly in [-0.08, 0.08] for 4-layer models and [-0.06, 0.06] for 6-layer models, (b) SGD has a fixed learning rate of 0.7, (c) we train for 8 epochs (after 5 epochs, we begin to halve the learning rate every 0.5 epoch), (d) the size of the mini-batch is 128, and (e) we rescale the normalized gradient to ensure that its norm does not exceed 5 [74].

We also follow the GPU parallelization scheme proposed in [90], allowing us to reach a training speed of 5.4K words per second to train a depth-6 model with 200K source and 80K target vocabularies ; whereas Sutskever et al. [90] achieved 6.3K words per second for a depth-4 models with 80K source and target vocabularies. Training takes about 10-14 days on an 8-GPU machine.

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

3.3.3 A note on BLEU scores

We report BLEU scores based on both: (a) *detokenized* translations, i.e., WMT'14 style, to be comparable with results reported on the WMT website[4] and (b) *tokenized translations*, so as to be consistent with previous work [3, 17, 40, 81, 90].[5]

The existing WMT'14 state-of-the-art system [21] achieves a detokenized BLEU score of 35.8 on the newstest2014 test set for English to French language pair (see Table 3.1). In terms of the tokenized BLEU, its performance is 37.0 points (see Table 3.2).

System	**BLEU**
Existing SOTA [21]	35.8
Ensemble of 8 LSTMs + PosUnk	**36.6**

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

3.3.4 Main Results

We compare our systems to others, including the current state-of-the-art MT system [21], recent end-to-end neural systems, as well as phrase-based baselines with neural components.

The results shown in Table 3.2 demonstrate that our unknown word translation technique (in particular, the PosUnk model) significantly improves the translation quality for both the individual (non-ensemble) LSTM models and the ensemble models.[6] For 40K-word vocabularies, the performance gains are in the range of 2.3-2.8 BLEU points. With larger vocabularies (80K), the performance gains are diminished, but our technique can still provide a nontrivial gains of 1.6-1.9 BLEU points.

It is interesting to observe that our approach is more useful for ensemble models as compared to the individual ones. This is because the usefulness of the PosUnk model

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

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

[6] For the 40K-vocabulary ensemble, we combine 5 models with 4 layers and 3 models with 6 layers. For the 80K-vocabulary ensemble, we combine 3 models with 4 layers and 5 models with 6 layers. Two of the depth-6 models are regularized with dropout, similar to Zaremba et al. [98] with the dropout probability set to 0.2.

System	Vocab	Corpus	BLEU
State of the art in WMT'14 [21]	All	36M	**37.0**
Standard MT + neural components			
Schwenk [81] – neural language model	All	12M	33.3
Cho et al. [17]– phrase table neural features	All	12M	34.5
Sutskever et al. [90] – 5 LSTMs, reranking 1000-best lists	All	12M	36.5
Existing end-to-end NMT systems			
Bahdanau et al. [3] – single gated RNN with search	30K	12M	28.5
Sutskever et al. [90] – 5 LSTMs	80K	12M	34.8
Jean et al. [40] – 8 gated RNNs with search + UNK replacement	500K	12M	37.2
Our end-to-end NMT systems			
Single LSTM with 4 layers	40K	12M	29.5
Single LSTM with 4 layers + PosUnk	40K	12M	31.8 (+2.3)
Single LSTM with 6 layers	40K	12M	30.4
Single LSTM with 6 layers + PosUnk	40K	12M	32.7 (+2.3)
Ensemble of 8 LSTMs	40K	12M	34.1
Ensemble of 8 LSTMs + PosUnk	40K	12M	36.9 (+2.8)
Single LSTM with 6 layers	80K	36M	31.5
Single LSTM with 6 layers + PosUnk	80K	36M	33.1 (+1.6)
Ensemble of 8 LSTMs	80K	36M	35.6
Ensemble of 8 LSTMs + PosUnk	80K	36M	**37.5 (+1.9)**

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

directly depends on the ability of the NMT to correctly locate, for a given OOV target word, its corresponding word in the source sentence. An ensemble of large models identifies these source words with greater accuracy. This is why for the same vocabulary size, better models obtain a greater performance gain our post-processing step. Except for the very recent work of Jean et al. [40] that employs a similar unknown treatment strategy[7] as ours, our best result

[7] Their unknown replacement method and ours both track the locations of target unknown words and use a word dictionary to post-process the translation. However, the mechanism used to achieve the "tracking" behavior is different. Jean et al. [40]'s uses the attentional mechanism to track the origins of all target words, not just the unknown ones. In contrast, we only focus on tracking unknown words using unsupervised alignments. Our method can be easily applied to any sequence-to-sequence models since we treat any model

of 37.5 BLEU outperforms all other NMT systems by a arge margin, and more importanly, our system has established a new record on the WMT'14 English to French translation.

3.4 Analysis

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

3.4.1 Rare Word Analysis

To analyze the effect of rare words on translation quality, we follow Sutskever et al. [90] and sort sentences in newstest2014 by the average inverse frequency of their words. We split the test sentences into groups where the sentences within each group have a comparable number of rare words and evaluate each group independently. We evaluate our systems before and after translating the OOV words and compare with the standard MT systems – we use the best system from the WMT'14 contest [21], and neural MT systems – we use the ensemble systems described in [90] and Section 5.4.

Rare word translation is challenging for neural machine translation systems as shown in Figure 3.5. Specifically, the translation quality of our model before applying the post-processing step is shown by the green curve, and the current best NMT system [90] is the purple curve. While [90] produces better translations for sentences with frequent words (the left part of the graph), they are worse than best system (red curve) on sentences with many rare words (the right side of the graph). When applying our unknown word translation technique (purple curve), we significantly improve the translation quality of our NMT: for the last group of 500 sentences which have the greatest proportion of OOV words in the test set, we increase the BLEU score of our system by 4.8 BLEU points. Overall, our rare

as a blackbox and manipulate only at the input and output levels.

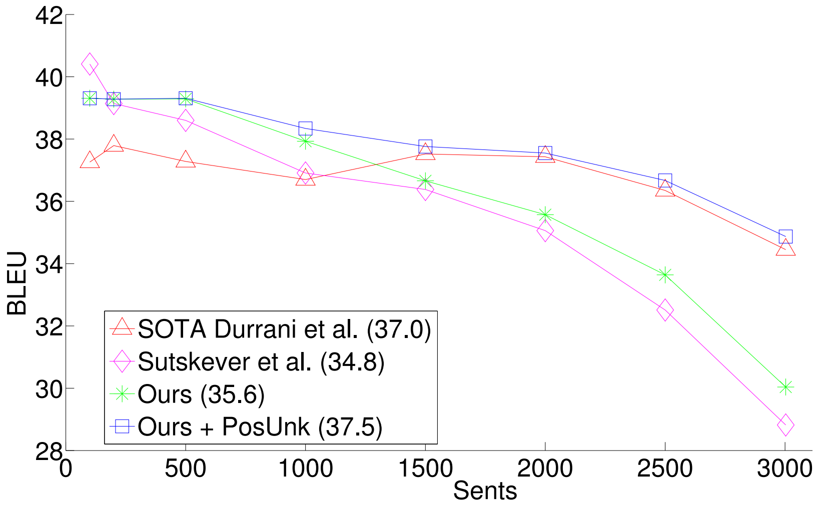


Figure 3.5: **Rare word translation** – On the x-axis, we order newstest2014 sentences by their *average frequency rank* and divide the sentences into groups of sentences with a comparable prevalence of rare words. We compute the BLEU score of each group independently.

word translation model interpolates between the SOTA system and the system of Sutskever et al. [90], which allows us to outperform the winning entry of WMT'14 on sentences that consist predominantly of frequent words and approach its performance on sentences with many OOV words.

3.4.2 Rare Word Models

We examine the effect of the different rare word models presented in Section 3.2, namely: (a) *Copyable* – which aligns the unknown words on both the input and the target side by learning to copy indices, (b) the Positional All (*PosAll*) – which predicts the aligned source positions for every target word, and (c) the Positional Unknown (*PosUnk*) – which predicts the aligned source positions for only the unknown target words.[8] It is also interesting to

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

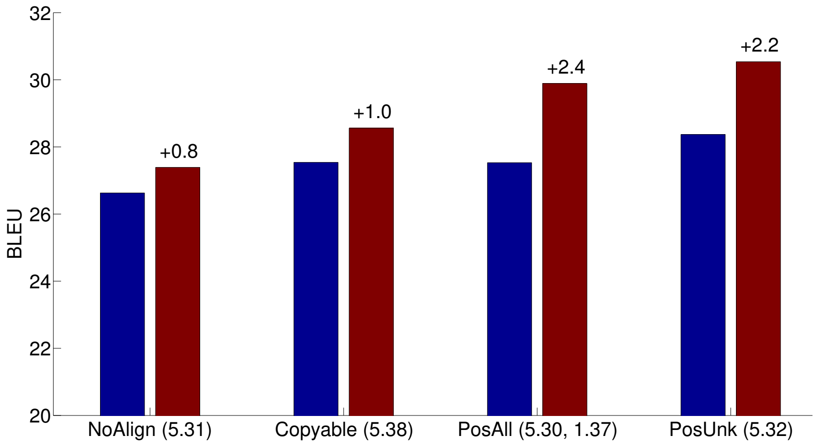


Figure 3.6: **Rare word models** – translation performance of 6-layer LSTMs: a model that uses no alignment (*NoAlign*) and the other rare word models (*Copyable, PosAll, PosUnk*). For each model, we show results before (*left*) and after (*right*) the rare word translation as well as the perplexity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

measure the improvement obtained when no alignment information is used during training. As such, we include a baseline model with no alignment knowledge (*NoAlign*) in which we simply assume that the i^{th} unknown word on the target sentence is aligned to the i^{th} unknown word in the source sentence.

From the results in Figure 3.6, a simple monotone alignment assumption for the *NoAlign* model yields a modest gain of 0.8 BLEU points. If we train the model to predict the alignment, then the *Copyable* model offers a slightly better gain of 1.0 BLEU. Note, however, that English and French have similar word order structure, so it would be interesting to experiment with other language pairs, such as English and Chinese, in which the word order is not as monotonic. These harder language pairs potentially imply a smaller gain for the NoAlign model and a larger gain for the Copyable model. We leave it for future work.

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

3.4.3 Other Effects

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

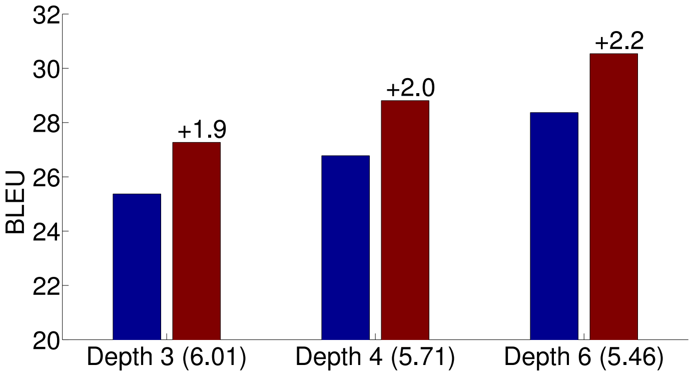


Figure 3.7: **Effect of depths** – BLEU scores achieved by *PosUnk* models of various depths (3, 4, and 6) before and after the rare word translation. Notice that the PosUnk model is more useful on more accurate models.

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

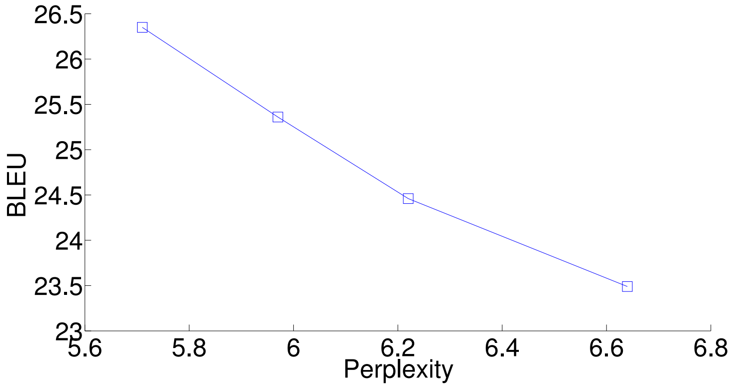


Figure 3.8: **Perplexity vs. BLEU** – we show the correlation by evaluating an LSTM model with 4 layers at various stages of training.

3.4.4 Sample Translations

We present three sample translations of our best system (with 37.5 BLEU) in Table 5.4. In our first example, the model translates all the unknown words correctly: *2600*, *orthopédiques*, and *cataracte*. It is interesting to observe that the model can accurately predict an alignment of distances of 5 and 6 words. The second example highlights the fact that our model can translate long sentences reasonably well and that it was able to correctly translate the unknown word for *JPMorgan* at the very far end of the source sentence. Lastly, our examples also reveal several penalties incurred by our model: (a) incorrect entries in the word dictionary, as with *négociateur* vs. *trader* in the second example, and (b) incorrect alignment prediction, such as when $\mathtt{unkpos}_3$ is incorrectly aligned with the source word *was* and not with *abandoning*, which resulted in an incorrect translation in the third sentence.

3.5 Conclusion

We have shown that a simple alignment-based technique can mitigate and even overcome one of the main weaknesses of current NMT systems, which is their inability to translate words that are not in their vocabulary. A key advantage of our technique is the fact that it is applicable to any NMT system and not only to the deep LSTM model of Sutskever et al.

	Sentences
src	An additional *2600* operations including *orthopedic* and *cataract* surgery will help clear a backlog .
trans	En outre , $\underline{\texttt{unkpos}_{1}}$ opérations supplémentaires , dont la chirurgie $\underline{\texttt{unkpos}_{5}}$ et la $\underline{\texttt{unkpos}_{6}}$, permettront de résorber l' arriéré .
+unk	En outre , *2600* opérations supplémentaires , dont la chirurgie *orthopédiques* et la *cataracte* , permettront de résorber l' arriéré .
tgt	2600 opérations supplémentaires , notamment dans le domaine de la chirurgie orthopédique et de la cataracte , aideront à rattraper le retard .
src	This *trader* , Richard *Usher* , left RBS in *2010* and is understand to have be given leave from his current position as European head of forex spot trading at *JPMorgan* .
trans	Ce $\underline{\texttt{unkpos}_{0}}$, Richard $\underline{\texttt{unkpos}_{0}}$, a quitté $\underline{\texttt{unkpos}_{1}}$ en 2010 et a compris qu' il est autorisé à quitter son poste actuel en tant que leader européen du marché des points de vente au $\underline{\texttt{unkpos}_{5}}$.
+unk	Ce *négociateur* , Richard *Usher* , a quitté RBS en *2010* et a compris qu' il est autorisé à quitter son poste actuel en tant que leader européen du marché des points de vente au *JPMorgan* .
tgt	Ce trader , Richard Usher , a quitté RBS en 2010 et aurait été mis suspendu de son poste de responsable européen du trading au comptant pour les devises chez JPMorgan
src	But concerns have grown after Mr *Mazanga* was quoted as saying *Renamo was* abandoning the 1992 peace accord .
trans	Mais les inquiétudes se sont accrues après que M. $\underline{\texttt{unkpos}_{3}}$ a déclaré que la $\underline{\texttt{unkpos}_{3}}$ $\underline{\texttt{unkpos}_{3}}$ l' accord de paix de 1992 .
+unk	Mais les inquiétudes se sont accrues après que M. *Mazanga* a déclaré que la *Renamo était* l' accord de paix de 1992 .
tgt	Mais l' inquiétude a grandi après que M. Mazanga a déclaré que la Renamo abandonnait l' accord de paix de 1992 .

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

[90]. A technique like ours is likely necessary if an NMT system is to achieve state-of-the-art performance on machine translation.

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

Chapter 4

Attention Mechanisms

Neural Machine Translation (NMT) achieved state-of-the-art performances in large-scale translation tasks such as from English to French [60] and English to German [40]. NMT is appealing since it requires minimal domain knowledge and is conceptually simple. The model by Luong et al. [60] reads through all the source words until the end-of-sentence symbol <`eos`> is reached. It then starts emitting one target word at a time, as illustrated in Figure 4.1. NMT is often a large neural network that is trained in an end-to-end fashion and has the ability to generalize well to very long word sequences. This means the model does not have to explicitly store gigantic phrase tables and language models as in the case of standard MT; hence, NMT has a small memory footprint. Lastly, implementing NMT decoders is easy unlike the highly intricate decoders in standard MT [47].

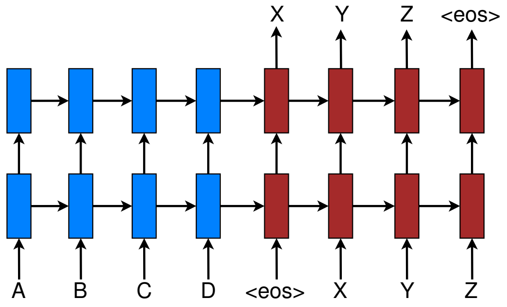


Figure 4.1: **Neural machine translation** – a stacking recurrent architecture for translating a source sequence `A B C D` into a target sequence `X Y Z`. Here, <`eos`> marks the end of a sentence.

In parallel, the concept of "attention" has gained popularity recently in training neural networks, allowing models to learn alignments between different modalities, e.g., between image objects and agent actions in the dynamic control problem [70], between speech frames and text in the speech recognition task [18], or between visual features of a picture and its text description in the image caption generation task [97]. In the context of NMT, Bahdanau et al. [3] has successfully applied such attentional mechanism to jointly translate and align words. To the best of our knowledge, there has not been any other work exploring the use of attention-based architectures for NMT.

In this work, we design, with simplicity and effectiveness in mind, two novel types of attention-based models: a *global* approach in which all source words are attended and a *local* one whereby only a subset of source words are considered at a time. The former approach resembles the model of [3] but is simpler architecturally. The latter can be viewed as an interesting blend between the *hard* and *soft* attention models proposed in [97]: it is computationally less expensive than the global model or the soft attention; at the same time, unlike the hard attention, the local attention is differentiable almost everywhere, making it easier to implement and train.[1] Besides, we also examine various alignment functions for our attention-based models.

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

[1] There is a recent work by Gregor et al. [34], which is very similar to our local attention and applied to the image generation task. However, as we detail later, our model is much simpler and can achieve good performance for NMT.

4.1 Neural Machine Translation

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

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

A natural choice to model such a decomposition in the decoder is to use a recurrent neural network (RNN) architecture, which most of the recent NMT work such as [3, 17, 40, 44, 60, 90] have in common. They, however, differ in terms of which RNN architectures are used for the decoder and how the encoder computes the source sentence representation $\boldsymbol{s}$.

Kalchbrenner and Blunsom [44] used an RNN with the standard hidden unit for the decoder and a convolutional neural network for encoding the source sentence representation. On the other hand, both Sutskever et al. [90] and Luong et al. [60] stacked multiple layers of an RNN with a Long Short-Term Memory (LSTM) hidden unit for both the encoder and the decoder. Cho et al. [17], Bahdanau et al. [3], and Jean et al. [40] all adopted a different version of the RNN with an LSTM-inspired hidden unit, the gated recurrent unit (GRU), for both components.[3]

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

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

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

[2] All sentences are assumed to terminate with a special "end-of-sentence" token <eos>.

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

[4] One can provide g with other inputs such as the currently predicted word y_j as in [3].

the RNN hidden unit, abstractly computed as:

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

where f computes the current hidden state given the previous hidden state and can be either a vanilla RNN unit, a GRU, or an LSTM unit. In [17, 44, 60, 90], the source representation $\boldsymbol{s}$ is only used once to initialize the decoder hidden state. On the other hand, in [3, 40] and this work, $\boldsymbol{s}$, in fact, implies a set of source hidden states which are consulted throughout the entire course of the translation process. Such an approach is referred to as an attention mechanism, which we will discuss next.

In this work, following [60, 90], we use the stacking LSTM architecture for our NMT systems, as illustrated in Figure 4.1. We use the LSTM unit defined in [**?**]. Our training objective is formulated as follows:

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

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

4.2 Attention-based Models

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

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

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

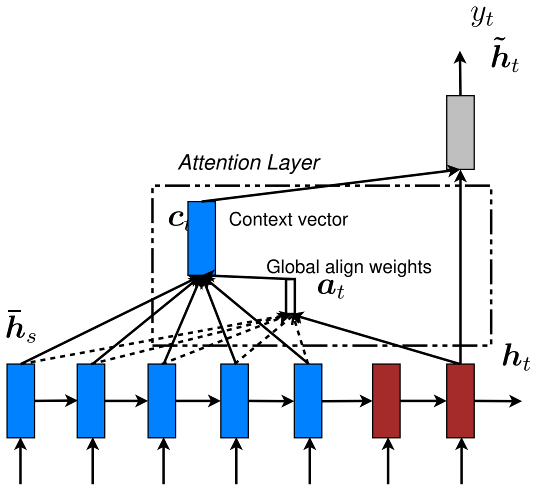


Figure 4.2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed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

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

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

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

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

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

4.2.1 Global Attention

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

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

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

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

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

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

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

Comparison to [3] – While our global attention approach is similar in spirit to the model proposed by Bahdanau et al. [3], there are several key differences which reflect how we have both simplified and generalized from the original model. First, we simply use hidden states at the top LSTM layers in both the encoder and decoder as illustrated in Figure 4.2. Bahdanau et al. [3], on the other hand, use the concatenation of the forward and backward source hidden states in the bi-directional encoder and target hidden states in their non-stacking uni-directional decoder. Second, our computation path is simpler; we go from $\boldsymbol{h}_t \rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \tilde{\boldsymbol{h}}_t$ then make a prediction as detailed in Eq. (5.4), Eq. (4.6), and Figure 4.2. On the other hand, at any time t, Bahdanau et al. [3] build from the previous hidden state **t-1** $\rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \boldsymbol{h}_t$, which, in turn, goes through a deep-output and a maxout layer before making predictions.[6] Lastly, Bahdanau et al. [3] only experimented with one

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

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

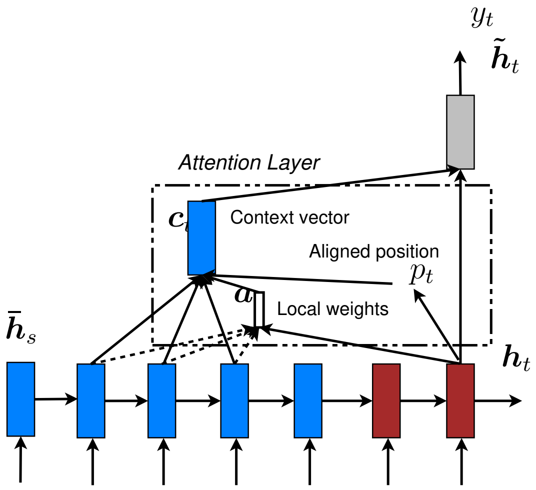


Figure 4.3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights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

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

4.2.2 Local Attention

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

This model takes inspiration from the tradeoff between the *soft* and *hard* attentional models proposed by Xu et al. [97] to tackle the image caption generation task. In their work, soft attention refers to the global attention approach in which weights are placed "softly" over all patches in the source image. The hard attention, on the other hand, selects one patch of the image to attend to at a time. While less expensive at inference time, the hard attention model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train.

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

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

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

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

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

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

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

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

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

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

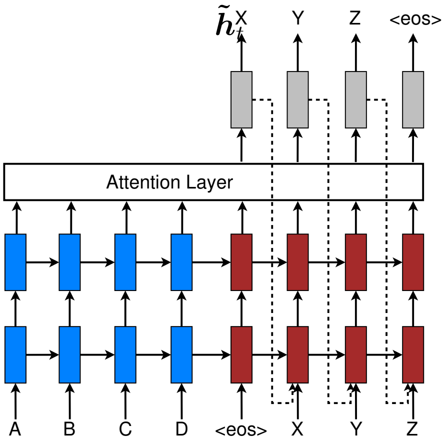


Figure 4.4: **Input-feeding approach** – Attentional vectors $\tilde{\boldsymbol{h}}_t$ are fed as inputs to the next time steps to inform the model about past alignment decisions.

Comparison to [34] – have proposed a *selective attention* mechanism, very similar to our local attention, for the image generation task. Their approach allows the model to select an image patch of varying location and zoom. We, instead, use the same "zoom" for all target positions, which greatly simplifies the formulation and still achieves good performance.

4.2.3 Input-feeding Approach

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

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

Comparison to other work – Bahdanau et al. [3] use context vectors, similar to our $\boldsymbol{c}_t$, in building subsequent hidden states, which can also achieve the "coverage" effect. However, there has not been any analysis of whether such connections are useful as done in this work. Also, our approach is more general; as illustrated in Figure 4.4, it can be applied to general stacking recurrent architectures, including non-attentional models.

Xu et al. [97] propose a *doubly attentional* approach with an additional constraint added to the training objective to make sure the model pays equal attention to all parts of the image during the caption generation process. Such a constraint can also be useful to capture the coverage set effect in NMT that we mentioned earlier. However, we chose to use the input-feeding approach since it provides flexibility for the model to decide on any attentional constraints it deems suitable.

4.3 Experiments

We evaluate the effectiveness of our models on the WMT translation tasks between English and German in both directions. newstest2013 (3000 sentences) is used as a development set to select our hyperparameters. Translation performances are reported in case-sensitive BLEU [73] on newstest2014 (2737 sentences) and newstest2015 (2169 sentences). Following [60], we report translation quality using two types of BLEU: (a) *tokenized*[11] BLEU to be comparable with existing NMT work and (b) *NIST*[12] BLEU to be comparable with WMT results.

4.3.1 Training Details

All our models are trained on the WMT'14 training data consisting of 4.5M sentences pairs (116M English words, 110M German words). Similar to [40], we limit our vocabularies to be the top 50K most frequent words for both languages. Words not in these shortlisted vocabularies are converted into a universal token `<unk>`.

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

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

System	**Ppl**	**BLEU**
Winning WMT'14 system – *phrase-based* + *large LM* [13]		20.7
Existing NMT systems		
RNNsearch [40]		16.5
RNNsearch + unk replace [40]		19.0
RNNsearch + unk replace + large vocab + *ensemble* 8 models [40]		**21.6**
Our NMT systems		
Base	10.6	11.3
Base + reverse	9.9	12.6 (+*1.3*)
Base + reverse + dropout	8.1	14.0 (+*1.4*)
Base + reverse + dropout + global attention (*location*)	7.3	16.8 (+*2.8*)
Base + reverse + dropout + global attention (*location*) + feed input	6.4	18.1 (+*1.3*)
Base + reverse + dropout + local-p attention (*general*) + feed input	5.9	19.0 (+*0.9*)
Base + reverse + dropout + local-p attention (*general*) + feed input + unk replace		20.9 (+*1.9*)
Ensemble 8 models + unk replace		**23.0** (+*2.1*)

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

When training our NMT systems, following [3, 40], we filter out sentence pairs whose lengths exceed 50 words and shuffle mini-batches as we proceed. Our stacking LSTM models have 4 layers, each with 1000 cells, and 1000-dimensional embeddings. We follow [60, 90] in training NMT with similar settings: (a) our parameters are uniformly initialized in $[-0.1, 0.1]$, (b) we train for 10 epochs using plain SGD, (c) a simple learning rate schedule is employed – we start with a learning rate of 1; after 5 epochs, we begin to halve the learning rate every epoch, (d) our mini-batch size is 128, and (e) the normalized gradient is rescaled whenever its norm exceeds 5. Additionally, we also use dropout with probability 0.2 for our LSTMs as suggested by [98]. For dropout models, we train for 12 epochs and start halving the learning rate after 8 epochs. For local attention models, we empirically set the window size $D = 10$.

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

4.3.2 English-German Results

We compare our NMT systems in the English-German task with various other systems. These include the winning system in WMT'14 [13], a phrase-based system whose language models were trained on a huge monolingual text, the Common Crawl corpus. For end-to-end NMT systems, to the best of our knowledge, [40] is the only work experimenting with this language pair and currently the SOTA system. We only present results for some of our attention models and will later analyze the rest in Section 5.5.

As shown in Table 4.1, we achieve progressive improvements when (a) reversing the source sentence, +1.3 BLEU, as proposed in [90] and (b) using dropout, +1.4 BLEU. On top of that, (c) the global attention approach gives a significant boost of +2.8 BLEU, making our model slightly better than the base attentional system of Bahdanau et al. [3] (row *RNNSearch*). When (d) using the *input-feeding* approach, we seize another notable gain of +1.3 BLEU and outperform their system. The local attention model with predictive alignments (row *local-p*) proves to be even better, giving us a further improvement of +0.9 BLEU on top of the global attention model. It is interesting to observe the trend previously reported in [60] that perplexity strongly correlates with translation quality. In total, we achieve a significant gain of 5.0 BLEU points over the non-attentional baseline, which already includes known techniques such as source reversing and dropout.

The unknown replacement technique proposed in [40, 60] yields another nice gain of +1.9 BLEU, demonstrating that our attentional models do learn useful alignments for unknown works. Finally, by ensembling 8 different models of various settings, e.g., using different attention approaches, with and without dropout etc., we were able to achieve a *new SOTA* result of 23.0 BLEU, outperforming the existing best system [40] by +1.4 BLEU.

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

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

Latest results in WMT'15 – despite the fact that our models were trained on WMT'14

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

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

with slightly less data, we test them on newstest2015 to demonstrate that they can generalize well to different test sets. As shown in Table 4.2, our best system establishes a *new SOTA* performance of 25.9 BLEU, outperforming the existing best system backed by NMT and a 5-gram LM reranker by +1.0 BLEU.

4.3.3 German-English Results

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

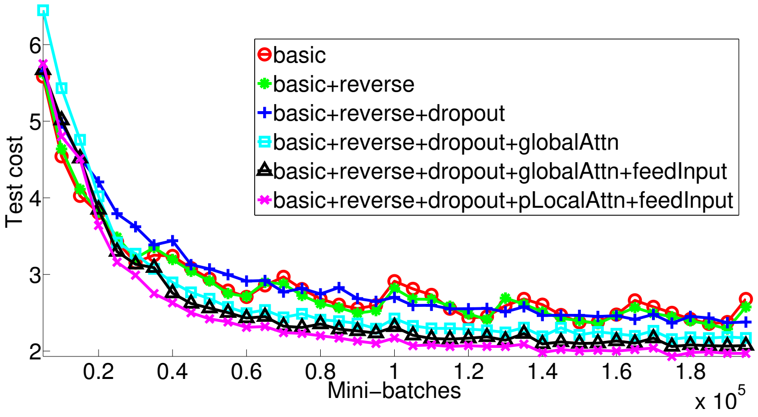


Figure 4.5: **Learning curves** – test cost ($\ln$ perplexity) on newstest2014 for English-German NMTs as training progresses.

4.4 Analysis

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

4.4.1 Learning curves

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

4.4.2 Effects of Translating Long Sentences

We follow [3] to group sentences of similar lengths together and compute a BLEU score per group. Figure 4.6 shows that our attentional models are more effective than the non-attentional one in handling long sentences: the quality does not degrade as sentences become longer. Our best model (the blue + curve) outperforms all other systems in all length buckets.

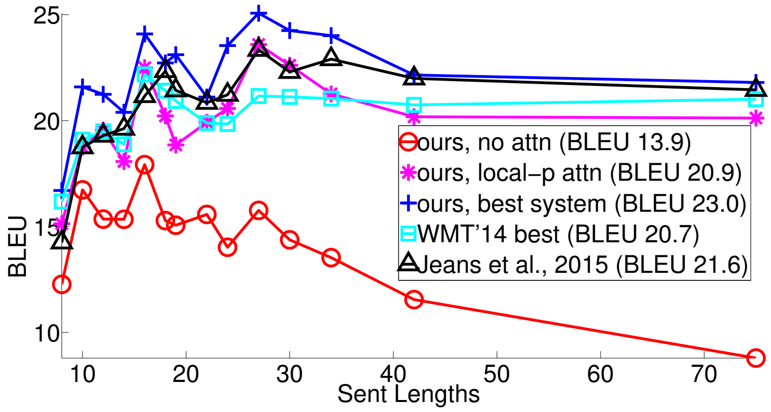


Figure 4.6: **Length Analysis** – translation qualities of different systems as sentences become longer.

4.4.3 Choices of Attentional Architectures

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

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

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

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

analysis should be done to understand the reason.[14] It is interesting to observe that *dot* works well for the global attention and *general* is better for the local attention. Among the different models, the local attention model with predictive alignments (*local-p*) is best, both in terms of perplexities and BLEU.

4.4.4 Alignment Quality

A by-product of attentional models are word alignments. While [3] visualized alignments for some sample sentences and observed gains in translation quality as an indication of a working attention model, no work has assessed the alignments learned as a whole. In contrast, we set out to evaluate the alignment quality using the alignment error rate (AER) metric.

Given the gold alignment data provided by RWTH for 508 English-German Europarl sentences, we “force” decode our attentional models to produce translations that match the references. We extract only one-to-one alignments by selecting the source word with the highest alignment weight per target word. Nevertheless, as shown in Table 4.5, we were able to achieve AER scores comparable to the one-to-many alignments obtained by the Berkeley aligner [51].[15]

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

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

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

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

We also found that the alignments produced by local attention models achieve lower AERs than those of the global one. The AER obtained by the ensemble, while good, is not better than the local-m AER, suggesting the well-known observation that AER and translation scores are not well correlated [27]. Due to space constraint, we can only show alignment visualizations in the arXiv version of our paper.[16]

4.4.5 Sample Translations

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

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

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

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

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

4.5 Conclusion

In this paper, we propose two simple and effective attentional mechanisms for neural machine translation: the *global* approach which always looks at all source positions and the *local* one that only attends to a subset of source positions at a time. We test the effectiveness of our models in the WMT translation tasks between English and German in both directions. Our local attention yields large gains of up to 5.0 BLEU over non-attentional models that already incorporate known techniques such as dropout. For the English to German translation direction, our ensemble model has established new state-of-the-art results for both WMT'14 and WMT'15.

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

Chapter 5

Hybrid Models

Neural Machine Translation (NMT) is a simple new architecture for getting machines to translate. At its core, NMT is a single deep neural network that is trained end-to-end with several advantages such as simplicity and generalization. Despite being relatively new, NMT has already achieved state-of-the-art translation results for several language pairs such as English-French [60], English-German [40, 55, 59], and English-Czech [41].

While NMT offers many advantages over traditional phrase-based approaches, such as small memory footprint and simple decoder implementation, nearly all previous work in NMT has used quite restricted vocabularies, crudely treating all other words the same with an <unk> symbol. Sometimes, a post-processing step that patches in unknown words is introduced to alleviate this problem. Luong et al. [60] propose to annotate occurrences of target <unk> with positional information to track their alignments, after which simple word dictionary lookup or identity copy can be performed to replace <unk> in the translation. Jean et al. [40] approach the problem similarly but obtain the alignments for unknown words from the attention mechanism. We refer to these as the *unk replacement* technique.

Though simple, these approaches ignore several important properties of languages. First, *monolingually*, words are morphologically related; however, they are currently treated as independent entities. This is problematic as pointed out by Luong et al. [57]: neural networks can learn good representations for frequent words such as "distinct", but fail for rare-but-related words like "distinctiveness". Second, *crosslingually*, languages have different alphabets, so one cannot naïvely memorize all possible surface word translations

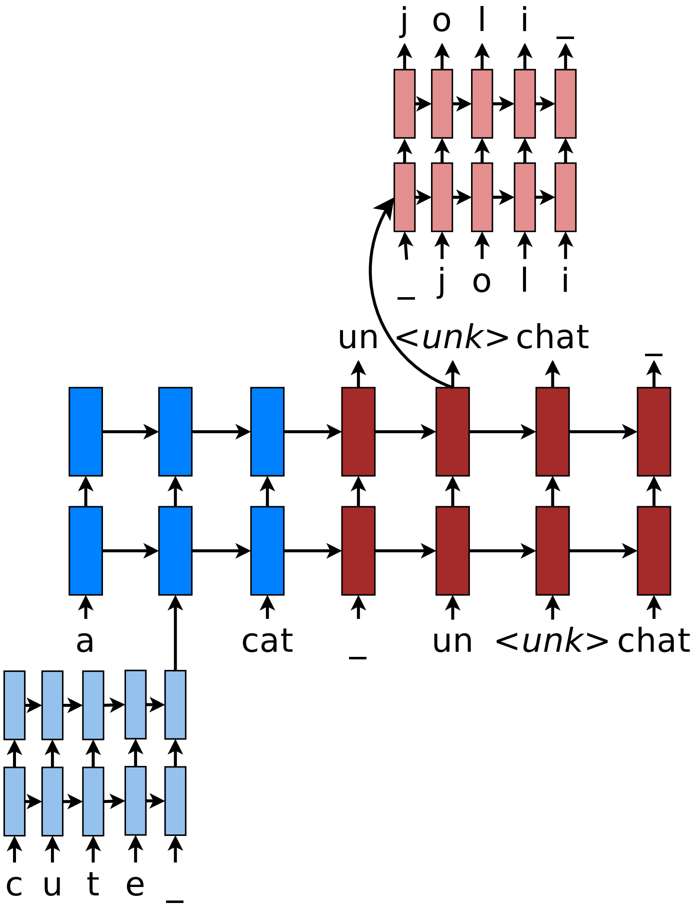


Figure 5.1: **Hybrid NMT** – example of a word-character model for translating "a cute cat" into "un joli chat". Hybrid NMT translates at the word level. For rare tokens, the character-level components build source representations and recover target <unk>. "_" marks sequence boundaries.

such as name transliteration between “Christopher” (English) and “Kryštof” (Czech). See more on this problem in [85].

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

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

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

5.1 Related Work

There has been a recent line of work on end-to-end character-based neural models which achieve good results for part-of-speech tagging [20, 53], dependency parsing [5], text classification [99], speech recognition [4, 15], and language modeling [43, 46]. However, success has not been shown for cross-lingual tasks such as machine translation.[1] Sennrich

[1] Recently, Ling et al. [54] attempt character-level NMT; however, the experimental evidence is weak. The authors demonstrate only small improvements over word-level baselines and acknowledge that there are no

et al. [85] propose to segment words into smaller units and translate just like at the word level, which does not learn to understand relationships among words.

Our work takes inspiration from [57] and [49]. Similar to the former, we build representations for rare words on-the-fly from subword units. However, we utilize recurrent neural networks with characters as the basic units; whereas Luong et al. [57] use recursive neural networks with morphemes as units, which requires existence of a morphological analyzer. In comparison with [49], our hybrid architecture is also a hierarchical sequence-to-sequence model, but operates at a different granularity level, word-character. In contrast, Li et al. [49] build hierarchical models at the sentence-word level for paragraphs and documents.

5.2 Background & Our Models

Neural machine translation aims to directly model the conditional probability $p(y|x)$ of translating a source sentence, $x_1, \ldots, x_n$, to a target sentence, $y_1, \ldots, y_m$. It accomplishes this goal through an *encoder-decoder* framework [17, 44, 90]. The *encoder* computes a representation $\boldsymbol{s}$ for each source sentence. Based on that source representation, the *decoder* generates a translation, one target word at a time, and hence, decomposes the log conditional probability as:

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

A natural model for sequential data is the recurrent neural network (RNN), used by most of the recent NMT work. Papers, however, differ in terms of: (a) architecture – from unidirectional, to bidirectional, and deep multi-layer RNNs; and (b) RNN type – which are long short-term memory (LSTM) [37] and the gated recurrent unit [17]. All our models utilize the *deep multi-layer* architecture with *LSTM* as the recurrent unit; detailed formulations are in [98].

Considering the top recurrent layer in a deep LSTM, with $\boldsymbol{h}_t$ being the current target

differences of significance. Furthermore, only small datasets were used without comparable results from past NMT work.

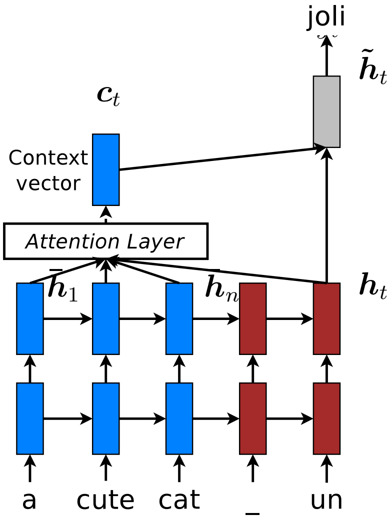


Figure 5.2: **Attention mechanism**.

hidden state as in Figure 5.2, one can compute the probability of decoding each target word y_t as:

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

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

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

Attention Mechanism – The early NMT approaches [17, 90], which we have described above, use only the last encoder state to initialize the decoder, i.e., setting the input representation $\boldsymbol{s}$ in Eq. (5.1) to $[\bar{\boldsymbol{h}}_n]$. Recently, Bahdanau et al. [3] propose an *attention mechanism*, a form of random access memory for NMT to cope with long input sequences. Luong et al. [59] further extend the attention mechanism to different scoring functions, used to compare source and target hidden states, as well as different strategies to place the attention. In all our models, we utilize the *global* attention mechanism and the *bilinear form* for the attention scoring function similar to [59].

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

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

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

5.3 Hybrid Neural Machine Translation

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

5.3.1 Word-based Translation as a Backbone

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

A nice property of our hybrid approach is that by varying the vocabulary size, one can control how much to blend the word- and character-based models; hence, taking the best of both worlds.

5.3.2 Source Character-based Representation

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

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

5.3.3 Target Character-level Generation

General word-based NMT allows generation of <unk> in the target output. Afterwards, there is usually a post-processing step that handles these unknown tokens by utilizing the alignment information derived from the attention mechanism and then performing simple word dictionary lookup or identity copy [40, 59]. While this approach works, it suffers from various problems such as alphabet mismatches between the source and target vocabularies and multi-word alignments. Our goal is to address all these issues and create a coherent framework that handles an unlimited output vocabulary.

Our solution is to have a separate deep LSTM that "translates" at the character level given the current word-level state. We train our system such that whenever the word-level NMT produces an <unk>, we can consult this character-level decoder to recover the correct surface form of the unknown target word. This is illustrated in Figure 5.1. The

[2] While Ling et al. [54] found that it is slow and difficult to train source character-level models and had to resort to pretraining, we demonstrate later that we can train our deep character-level LSTM perfectly fine in an end-to-end fashion.

training objective in Eq. (5.3) now becomes:

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

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

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

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

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

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

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

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

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

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

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

5.4 Experiments

We evaluate the effectiveness of our models on the publicly available WMT'15 translation task from English into Czech with *newstest2013* (3000 sentences) as a development set and *newstest2015* (2656 sentences) as a test set. Two metrics are used: case-sensitive NIST BLEU [73] and chrF$_3$ [78].[3] The latter measures the amounts of overlapping character n-grams and has been argued to be a better metric for translation tasks out of English.

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

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

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

5.4.1 Data

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

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

5.4.2 Training Details

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

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

	System	**Vocab**	**Perplexity**		**BLEU**	**chrF$_3$**
			w	c		
(a)	Best WMT'15, big data [11]	-	-	-	i18.8	-
	Existing NMT					
(b)	RNNsearch + unk replace [41]	200K	-	-	15.7	-
(c)	iEnsemble 4 models + unk replace [41]	200K	-	-	18.3	-
	Our *word-based* NMT					
(d)	Base + attention + unk replace	50K	5.9	-	17.5	42.4
(e)	iEnsemble 4 models + unk replace	50K	-	-	18.4	43.9
	Our *character-based* NMT					
(f)	Base-512 (600-step backprop)	200	-	2.4	3.8	25.9
(g)	Base-512 + attention (600-step backprop)	200	-	1.6	17.5	i46.6
(h)	Base-1024 + attention (300-step backprop)	200	-	1.9	15.7	41.1
	Our *hybrid* NMT					
(i)	Base + attention + same-path	10K	4.9	1.7	14.1	37.2
(j)	Base + attention + separate-path	10K	4.9	1.7	15.6	39.6
(k)	Base + attention + separate-path + 2-layer char	10K	4.7	1.6	i17.7	44.1
(l)	Base + attention + separate-path + 2-layer char	50K	5.7	1.6	19.6	46.5
(m)	iEnsemble 4 models	50K	-	-	**20.7**	**47.5**

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

In training word-based NMT, we follow Luong et al. [59] to use the global attention mechanism together with similar hyperparameters: (a) deep LSTM models, 4 layers, 1024 cells, and 1024-dimensional embeddings, (b) uniform initialization of parameters in $[-0.1, 0.1]$, (c) 6-epoch training with plain SGD and a simple learning rate schedule – start with a learning rate of 1.0; after 4 epochs, halve the learning rate every 0.5 epoch, (d) mini-batches are of size 128 and shuffled, (e) the gradient is rescaled whenever its norm exceeds 5, and (f) dropout is used with probability 0.2 according to [77]. We now detail differences across the three architectures.

Word-based NMT – We constrain our source and target sequences to have a maximum length of 50 each; words that go past the boundary are ignored. The vocabularies are limited to the top $|V|$ most frequent words in both languages. Words not in these vocabularies are converted into <unk>. After translating, we will perform dictionary[5] lookup or identity copy for <unk> using the alignment information from the attention models. Such procedure is referred as the *unk replace* technique [40, 60].

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

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

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

[5] Obtained from the alignment links produced by the Berkeley aligner [51] over the training corpus.

5.4.3 Results

We compare our models with several strong systems. These include the winning entry in WMT'15, which was trained on a much larger amount of data, 52.6M parallel and 393.0M monolingual sentences [11].[6] In contrast, we merely use the provided parallel corpus of 15.8M sentences. For NMT, to the best of our knowledge, [41] has the best published performance on English-Czech.

As shown in Table 5.2, for a purely *word-based* approach, our single NMT model outperforms the best single model in [41] by +1.8 points despite using a smaller vocabulary of only 50K words versus 200K words. Our ensemble system *(e)* slightly outperforms the best previous NMT system with 18.4 BLEU.

To our surprise, purely *character-based* models, though extremely slow to train and test, perform quite well. The 512-dimensional attention-based model *(g)* is best, surpassing the single word-based model in [41] despite having much fewer parameters. It even outperforms most NMT systems on chrF$_3$ with 46.6 points. This indicates that this model translate words that closely but not exactly match the reference ones as evidenced in Section 5.5.3. We notice two interesting observations. First, attention is critical for character-based models to work as is obvious from the poor performance of the non-attentional model; this has also been shown in speech recognition [15]. Second, long time-step backpropagation is more important as reflected by the fact that the larger 1024-dimensional model *(h)* with shorter backprogration is inferior to *(g)*.

Our *hybrid* models achieve the best results. At 10K words, we demonstrate that our *separate-path* strategy for the character-level target generation (§5.3.3) is effective, yielding an improvement of +1.5 BLEU points when comparing systems *(j)* vs. *(i)*. A *deeper* character-level architecture of 2 LSTM layers provides another significant boost of +2.1 BLEU. With 17.7 BLEU points, our hybrid system *(k)* has surpassed word-level NMT models.

When extending to 50K words, we further improve the translation quality. Our best single model, system *(l)* with 19.6 BLEU, is already better than all existing systems. Our

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

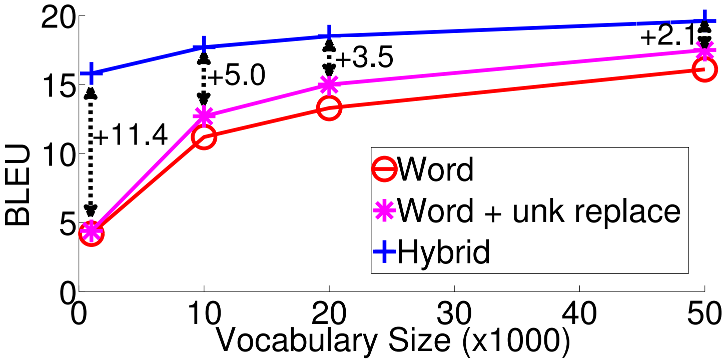


Figure 5.3: **Vocabulary size effect** – shown are the performances of different systems as we vary their vocabulary sizes. We highlight the improvements obtained by our hybrid models over word-based systems which already handle unknown words.

ensemble model *(m)* further advances the SOTA result to **2**0.7 BLEU, outperforming the winning entry in the WMT'15 English-Czech translation task by a large margin of +1.9 points. Our ensemble model is also best in terms of chrF$_3$ with **4**7.5 points.

5.5 Analysis

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

5.5.1 Effects of Vocabulary Sizes

As shown in Figure 5.3, our hybrid models offer large gains of +2.1-11.4 BLEU points over strong word-based systems which already handle unknown words. With only a small vocabulary, e.g., 1000 words, our hybrid approach can produce systems that are better than word-based models that possess much larger vocabularies. While it appears from the plot that gains diminish as we increase the vocabulary size, we argue that our hybrid models are still preferable since they understand word structures and can handle new complex words at test time as illustrated in Section 5.5.3.

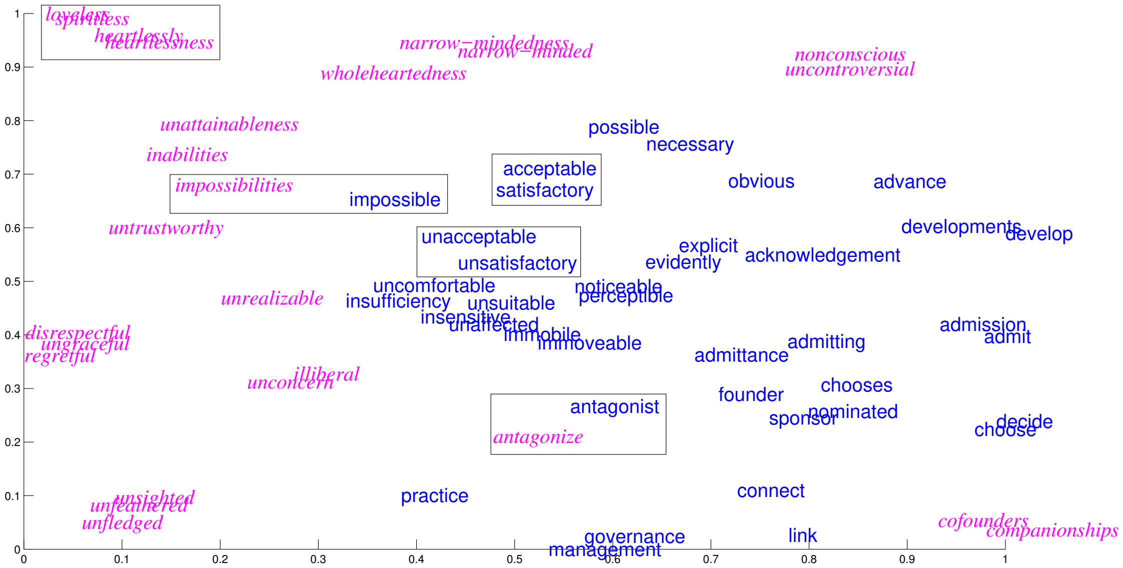


Figure 5.4: **Barnes-Hut-SNE visualization of source word representations** – shown are sample words from the *Rare Word* dataset. We differentiate two types of embeddings: frequent words in which encoder embeddings are looked up directly and *rare* words where we build representations from characters.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

5.5.2 Rare Word Embeddings

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

Quantitatively, we follow Luong et al. [57] in using the word similarity task, specifically on the *Rare Word* dataset, to judge the learned representations for complex words. The evaluation metric is the Spearman's correlation ρ between similarity scores assigned by a model and by human annotators. From the results in Table 5.3, we can see that source representations produced by our hybrid[7] models are significantly better than those of the word-based one. It is noteworthy that our deep recurrent character-level models can outperform the model of [57], which uses recursive neural networks and requires a complex morphological analyzer, by a large margin. Our performance is also competitive to the best Glove embeddings [76] which were trained on a much larger dataset.

System		Size	$\lvert V\rvert$	ρ
[57]		1B	138K	34.4
Glove [76]		6B	400K	38.1
		42B	400K	**47.8**
Our NMT models				
(d)	Word-based	0.3B	50K	20.4
(k)	Hybrid	0.3B	10K	42.4
(l)	Hybrid	0.3B	50K	i47.1

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

Qualitatively, we visualize embeddings produced by the hybrid model *(l)* for selected words in the Rare Word dataset. Figure 5.4 shows the two-dimensional representations of words computed by the Barnes-Hut-SNE algorithm [92].[8] It is extremely interesting to observe that words are clustered together not only by the word structures but also by the meanings. For example, in the top-left box, the *character*-based representations for "loveless", "spiritless", "heartlessly", and "heartlessness" are nearby, but clearly separated into

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

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

1	source human	The author iStephen Jay Gould died 20 years after idiagnosis . Autor *Stephen Jay Gould* zemřel 20 let po *diagnóze* .
	word	Autor Stephen Jay <unk> zemřel 20 let po <unk> . Autor *Stephen Jay Gould* zemřel 20 let po **po** .
	char	Autor **Stephen Stephen** zemřel 20 let po *diagnóze* .
	hybrid	Autor <unk> <unk> <unk> zemřel 20 let po <unk>. Autor *Stephen Jay Gould* zemřel 20 let po *diagnóze* .
2	source human	As the Reverend iMartin Luther King Jr. said ififty years ago : Jak *před padesáti lety* řekl reverend *Martin Luther King Jr* . :
	word	Jak řekl reverend Martin <unk> King <unk> před padesáti lety : Jak řekl reverend *Martin Luther King* **řekl** před padesáti lety :
	char	Jako reverend *Martin Luther* **král říkal** před padesáti lety :
	hybrid	Jak před <unk> lety řekl <unk> Martin <unk> <unk> <unk> : Jak *před padesáti lety* řekl reverend *Martin Luther King* Jr. :
3	source human	Her i11-year-old daughter , iShani Bart , said it felt a " little bit iweird " [..] back to school . Její *jedenáctiletá* dcera *Shani Bartová* prozradila , že " je to trochu *zvláštní* " [..] znova do školy .
	word	Její <unk> dcera <unk> <unk> řekla , že je to " trochu divné " , [..] vrací do školy . Její **11-year-old** dcera *Shani* **,** řekla , že je to " trochu divné " , [..] vrací do školy .
	char	Její *jedenáctiletá* dcera , *Shani Bartová* , říkala , že cítí trochu divně , [..] vrátila do školy .
	hybrid	Její <unk> dcera , <unk> <unk> , řekla , že cítí " trochu <unk> " , [..] vrátila do školy . Její *jedenáctiletá* dcera , **Graham** Bart , řekla , že cítí " trochu divný " , [..] vrátila do školy .

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

two groups. Similarly, in the center boxes, *word*-based embeddings of "acceptable", "satisfactory", "unacceptable", and "unsatisfactory", are close by but separated by meanings. Lastly, the remaining boxes demonstrate that our character-level models are able to build representations comparable to the word-based ones, e.g., "impossibilities" vs. "impossible" and "antagonize" vs. "antagonist". All of this evidence strongly supports that the source character-level models are useful and effective.

5.5.3 Sample Translations

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

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

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

5.6 Conclusion

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

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

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

Chapter 6

NMT Future

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:

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

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

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

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

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

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

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

[6] Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Trans. Neural Networks*, 19(4):713–722, 2008.

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

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

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

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

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

[12] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 06 1993.

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

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

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

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

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

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

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

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

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

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

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

[24] Jeffrey L. Elman. Finding structure in time. In *Cognitive Science*, 1990.

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

[26] Mikel L. Forcada and Ramón Neco. Recursive hetero-associative memories for translation. *Biological and Artificial Computation: From Neuroscience to Technology*, pages 453–462, 1997.

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

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

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

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

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

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

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

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

[35] Kenneth Heafield. KenLM: faster and smaller language model queries. In *WMT*, 2011.

[36] Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable modified Kneser-Ney language model estimation. In *ACL*, 2013.

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

[38] W. John Hutchins. Machine translation: A concise history, 2007.

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

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

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

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

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

[44] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013.

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

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

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

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

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

[50] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *ACL*, 2006.

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

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

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

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

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

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

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

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

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

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

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

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

[63] Tomáš Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

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

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

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

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

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

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

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

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

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

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

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

[75] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *ACL*, 2011.

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

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

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

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

[80] David E. Rumelhart and James L. McClelland. 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*, pages 216–271. MIT Press, 1986.

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

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

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

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

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

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

[87] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *ICSLP*, 2002.

[88] Ilya Sutskever. *Training Recurrent Neural Networks*. PhD thesis, University of Toronto, 2012.

[89] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

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

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

[92] Laurens van der Maaten. Barnes-Hut-SNE. In *ICLR*, 2013.

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

[94] Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Readings in speech recognition. chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404. 1990. ISBN 1-55860-124-4.

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

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

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

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

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