TÉCNICO LISBOA

Factored Models for Neural Machine Translation

Pedro Dias Coelho

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. André Filipe Torres Martins
Eng. Christine Anne Maroti

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. André Filipe Torres Martins
Member of the Committee: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur

January 2021

Acknowledgments

First and foremost I must thank my parents for always providing me all the needed resources to conclude my academic path, and for always supporting and encouraging my choices. A special thank you to my sister, for her advice and patience throughout the time I was writing this document.

Secondly, to my supervisor André Martins for all his valuable inputs and discussions, and for allowing me to develop a Thesis that joining my two most keen interests, software engineering, and AI. Also to my co-advisor Christine Maroti for her valuable help and continuous guidance.

Thirdly, to the NMT team from Unbabel for all the technical support and shared knowledge, with a special mention to António Lopes and Austin Matthews. Doing this work with Unbabel was a challenging, rewarding, and positive experience at all levels. Thank you for providing me all the necessary resources to develop this work.

Fourthly, to my colleagues of the Driverless Team of FST Lisboa, for making my last year in IST such an incredible experience and for the understanding that they always had when I had to focus more on the development of this work.

Last but for sure not least to all my close friends. You were undoubtedly one of the main pillars in this 5-year journey in IST and it would be impossible to finish it without your help. To all of you, my deepest thank you.

Resumo

Com a globalização, tornou-se cada vez mais importante traduzir texto com elevada qualidade. Ultimamente, a Tradução Automática Neuronal (TAN) tem sido a principal solução escolhida para suprir esta necessidade. Dentro da vasta pesquisa feita nesta área, formas inovadoras de representar os dados têm sido estudadas, numa tentativa de enriquecer a informação contida na representação de cada palavra. Uma delas são os fatores, um mecanismo através do qual as palavras, ao invés de serem representadas apenas por elas próprias, são definidas por um conjunto de características.

No presente trabalho, testamos e avaliamos a usabilidade do código referente aos fatores no Marian, uma ferramenta *open-source* para TAN. Mostramos o impacto que usar fatores tem na qualidade das traduções e no desempenho desta ferramenta em termos de velocidade de treino e de inferência. Para além disso, contribuímos para o código base desta ferramenta através da implementação da concatenação como um possível método para combinar as representações (*embeddings*) das palavras e dos fatores.

Realizamos três experiências nas quais usamos fatores para três diferentes aplicações, e mostramos como isso melhorou a qualidade das traduções dos sistemas de TAN. Usamo-los para injectar terminologia em tempo de execução, e mostramos como combinar as representações das palavras e fatores através da concatenação é a melhor opção. Usamo-los para representar divisões em subpalavras, comparando dois métodos previamente propostos para o efeito. Finalmente, usamo-los para codificar informação sobre morfologia, numa tentativa de melhorar a qualidade da tradução de linguagens morfologicamente ricas, obtendo resultados promissores para o par de linguagens Inglês-Romeno.

Palavras-chave: Aprendizagem profunda, processamento de linguagem natural, tradução automática neural, tradução automática neural fatorizada.

Abstract

Globalization has increased the importance of having high quality, quickly generated, and easily accessible translations. For the past few years, neural machine translation (NMT) has been the major chosen solution to fulfill this need. Among the vast research done in the field, novel ways of representing input data have been studied in an attempt to enrich the information contained in each word representation. One of them is factors, a mechanism where each word, instead of being represented by only the word itself, is defined by a bundle of features.

In our work, we test and evaluate the usability of the factors' code in Marian, an open-source NMT toolkit. We show how using factors impacts the quality of translations and the performance of this framework in terms of inference and training speed. We also contribute to its codebase by implementing concatenation as a method of combining the embeddings of the word and the factors.

We conduct three experiments, where we use factors for three different use cases and show how they improve the quality of the NMT systems' translations. We use them to inject terminology at run time, and show how combining word and factor embeddings by concatenating them is the most valuable option. We use them to represent subword splits, comparing two different methods previously proposed to do so. Finally, we use them to encode morphological information in an attempt to improve the translation quality of morphologically rich languages, showing promising results for the English-Romanian language pair.

Keywords: Deep learning, natural language processing, neural machine translation, factored neural machine translation

Contents

List of Tables

List of Figures

Acronyms

BPE Byte Pair Encoding

CEF Connecting Europe Facility

FNMT Factored Neural Machine Translation

GRU Gated Recurrent Unit

LSTM Long Short-Term Memory

NLP Natural Language Processing

NMT Neural Machine Translation

RNN Recurrent Neural Network

SMT Statistical Machine Translation

Chapter 1

Introduction

1.1 Motivation

Globalization brings the need for communicating with people from all around the world, in any language, enhancing the importance to have high quality, quickly generated, and easy accessible translations. While human translators can produce high quality translations, they can be cost and time consuming. Machine Translation (MT) seeks to close this gap, by producing more cost-effective translations, in a more scalable manner.

One of the earliest breakthroughs in machine translation was Statistical Machine Translation (SMT) (Brown et al. [1]). As the name suggests, it uses statistical analysis and predictive algorithms to translate from a certain language to another. Brown et al. [2] introduced what became known as the "five IBM models", a series of five statistical models that defined the concept of word-by-word alignment between a pair of bilingual sentences, which consisted in mapping directly each word in a source sentence to a word in a target sentence. These word-based models were later replaced by phrased-based systems (Marcu and Wong [3], Koehn et al. [4]), which map longer sequences between language pairs.

The increasing availability of large amounts of data, and the advances done in the scope of Deep Learning (Goodfellow et al. [5]), led to a different approach to MT, Neural Machine Translation (NMT) (Kalchbrenner and Blunsom [6], Cho et al. [7]), based on neural networks. The earlier NMT architectures included Recurrent Neural Networks (RNNs) (Elman [8]), Long Short-Term Memory (LSTMs) cells, (Hochreiter and Schmidhuber [9], Sutskever et al. [10]), Gated Recurrent Units (GRU) (Cho et al. [7]) and Convolutional Neural Networks (CNNs) (Gehring et al. [11]). A stronger take to neural machine translation was later took by the introduction of the attention mechanism (Bahdanau et al. [12], Luong et al. [13]), which allows the model to learn which words from the source side to attend when generating a certain word of the target side. Consequently, the transformer, a new architecture solely based on attention mechanisms discarding completely recurrence and convolution, was introduced by Vaswani et al. [14], which produced new benchmarks for NMT systems.

Parallel to all these developments in neural machine translation architectures, novel ways of representing input data were also studied. Motivated by the desire to enrich the information contained in each

word representation, Bilmes and Kirchhoff [15] introduced the concept of **factors**, where each word, instead of being represented by only the word itself, is defined as a bundle of different features. In machine translation, factors were first incorporated in SMT (Yang and Kirchhoff [16], Koehn and Hoang [17]), being later extend to NMT by Sennrich and Haddow [18], who applied factors to the source side, and by García-Martínez et al. [19], who used them for target data. Given the versatility of factors, these works used them to encode many kinds of different information, namely part-of-speech tags, morphological information, capitalization, as well as carrying information about how a word is split into its subword units (Sennrich et al. [20]). The usage of factors proved to be beneficial to improve different aspects of NMT systems, such as the generation of unknown words, the grammatical coherence of sentences, the disambiguation of homonyms, and decreasing decoding time, by reducing the size of the vocabulary for example. More recently, Dinu et al. [21] and Exel et al. [22] proposed factors as a solution to tackle the problem of domain adaptation by injecting custom terminology into neural machine translation at run time.

Accompanying all these developments, a variety of different NMT open-source toolkits have been developed. Among them, the Marian toolkit (Junczys-Dowmunt et al. [23]) stood out from the remaining ones, given its pure implementation in C++, its minimal dependencies, and its competitive benchmarks related to training and inference speed, as well as its cost-effectiveness (Junczys-Dowmunt et al. [24]). However, some toolkits like Nematus (Sennrich et al. [25]), Sockeye (Hieber et al. [26]) or Open-NMT (Klein et al. [27]) have already released software versions with the implementation of source and target factors which has not happened with Marian until this date. Motivated by the reported positive results of using factors, the Connecting Europe Facility (CEF) decided to incorporate the inclusion of user-supplied factors as one of the milestones of their "User-Focused Marian" action, focused on improving Marian to address the needs of CEF eTranslation[1] and to broaden its user base. It is in the scope of that project that this thesis has been developed.

Regarding the aeronautics industry, it has been reported that language barriers affect performance negatively, specially when in comes to maintenance (Drury and Ma [28], Drury et al. [29]), as most of the maintenance personnel does not have English as native language. Misunderstandings can lead to prolongations of the time it takes to restore an asset to operational readiness. Allowing maintenance technicians to communicate in their native language would aid in the technicians' training processes, as often training in English takes a part of it. Therefore, the new advances in machine translation, which have increased the speed and accuracy of automatic translation, while lowered the cost of having access to them, can help mitigate the reported problems caused by faulty communication.

Furthermore, factors in particular can have a key role in enriching the quality of these translations, not only by improving the performance of the system as a whole, but mainly due to their use for domain adaptation, allowing the injection of specific terminology to the generated translation, useful for the aeronautics industry as it is a field which has a considerable term base. A graphical representation of a translation process where factors are used for this use case, can be seen in Figure 1.1.

[1] eTranslation is an automated translation tool from the EU: `https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eTranslation`

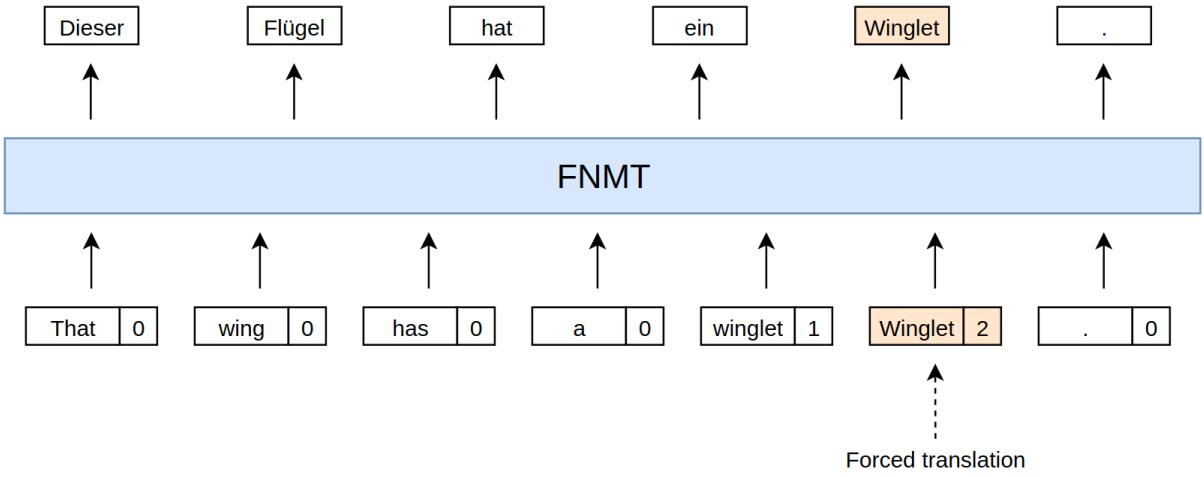


Figure 1.1: Example of a FNMT architecture were factors are used to inject terminology. In this example, when translating the English sentence “That wing has a winglet.” into German, we want to force the translation of the term “winglet” to be “Winglet”. The wanted term translation is inserted into the source sentence as an inline annotation and factors are used to signal the switch between words of the original source sentence and the added target term. The factor “0”, is used for source words, the factor “1” for the source term, and the factor “2” for the target term. After training, the model learns a “copy-behaviour” and translates terms marked with “1”, by copying the terms marked with “2”.

1.2 Contributions

The main contributions of this thesis are the following:

- We extend the work from Dinu et al. [21], by comparing different factor embedding options. We show that concatenation is the best option, and we also extended the experiment to other three language pairs;

- We compare two different approaches to represent the subword splits with factors (Sennrich and Haddow [18] and Wilken and Matusov [30]). Also, we show that using factors for this use-case, despite resulting in positive results for English-German as reported in the literature, when extending it to other language pairs, the same positive behavior is not noticed;

- We used factors to improve the translation of morphologically rich languages, showing promising results for the English-Romanian language pair;

- We tested and evaluated the usability of the factors’ code in the Marian toolkit. Furthermore, we contributed to the open source Marian’s code base by means of the CEF’s Marian action,[2] with the implementation of concatenation as an option to combine word and factor embeddings, as the original code only had sum as an available option.

[2]Code submission available in: `https://github.com/marian-cef/marian-dev`

1.3 Thesis Outline

This thesis starts by introducing the needed background to understand the proposed work. This is done in Chapter 2, where we introduce neural machine translation by explaining the main neural architectures behind it (Sections 2.1.1, 2.1.2 and 2.1.3) and also the concept of vocabulary (Section 2.1.4) and the concept of word embeddings (Section 2.1.5).

Chapter 3 starts by covering the work done regarding factors (Section 3.1), both for the source (Section 3.1.1) and target (Section 3.1.2) side, also detailing the changes that need to be done in an NMT architecture to incorporate them. Then, we explain how factors are implemented in Marian (Section 3.2), starting as well by the source side (Section 3.2.1), where our implementation of concatenation to combine word and factor embeddings is also explained, moving then to the target side (Section 3.2.2). Furthermore, we show three different applications for factors (Section 3.3), the same three application for which we used them in our experiments. We look into the work already done for each use case and introduce how we extended that same work.

The experimental analysis is done in Chapter 4. After detailing the datasets used (Section 4.1), and the baselines training setup (Section 4.2), we report and analyze the results of three experiments. Each of the experiments shows the applicability of factors for different goals, the first one for domain adaptation and the injection of terminology (Section 4.3), the second one for representing the splits into subword units (Section 4.4) and the third one for carrying morphological information (Section 4.5).

Finally, Chapter 5 concludes this thesis, and we do an overview of the main achievements obtained with this work, leaving also some suggestions for future improvements.

Chapter 2

Background

The present chapter provides the key theoretical concepts that will be used throughout this thesis. It starts by introducing neural machine translation (Section 2.1), describing the neural architectures associated to it, namely RNNs (Section 2.1.1), the encoder-decoder approach (Section 2.1.2) and the transformer (Section 2.1.3). Furthermore, an explanation of the key concepts regarding word representations is given, namely the concept of the vocabulary (Section 2.1.4) and word embeddings (Section 2.1.5). Then, two evaluation metrics used in MT are presented (Section 2.2). Finally, the Marian toolkit, on top of which the implementation work of this thesis has been developed, is described (Section 2.3), and its benefits are highlighted.

2.1 Neural Machine Translation

According to Bahdanau et al. [12], neural machine translation is an approach to machine translation that consists of training a single, large neural network (NN) end-to-end. This NN takes a sentence as input and outputs an automatic translation. The NN is trained with parallel sentences, which are sentences in two languages with the same meaning. This technique relies on different concepts of Deep Learning, and this section serves the purpose of detailing them.

2.1.1 Recurrent Neural Network

A Recurrent Neural Network (RNN) (Elman [8]), is a special type of Neural Network, which, as the name suggests, has a recurrent nature. These networks perform the same computation for every element of a given sequence, making the output dependent on previous computations. We can think about this type of NN as if it had a "memory" that retains information about what has been seen and computed so far. Therefore, this kind of architecture is suitable for dealing with sequential data, where the different elements from the input (and/or output) data are dependent on each other, which is the case for translating a sentence from one language into another.

Figure 2.1 shows how a forward pass works in a RNN. For each time step t, we provide the input $x^{(t)}$ to the network in order to compute the hidden state $h^{(t)}$. This hidden state is the key to propagate the

information of previous states and therefore keep track of the context. $h^{(t)}$ is calculated with equation 2.1,

$$h^{(t)} = g\left(b + Wh^{(t-1)} + Ux^{(t)}\right), \tag{2.1}$$

where W is a weight matrix for hidden-to-hidden connections, and U and b are a matrix and a bias for input-to-hidden connections, respectively. $g()$ is a generic activation function, like $\tanh$ for example. In order to obtain the prediction $\hat{y}^{(t)}$, we use equation 2.2,

$$\hat{y}^{(t)} = \text{softmax}(c + Vh^{(t)}), \tag{2.2}$$

where V and c are respectively a weight matrix and a bias for hidden-to-output connections.

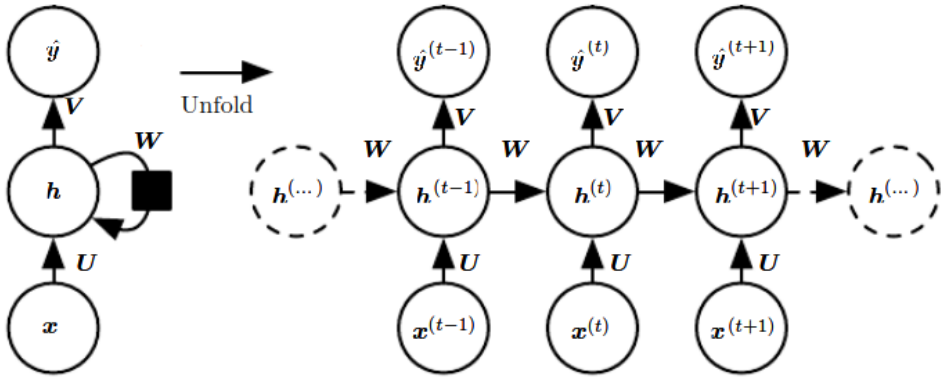


Figure 2.1: RNN schematic graph. Adapted from Goodfellow et al. [5].

In this kind of NN, during training, we need to backpropagate the error from the different time steps. This is done with an algorithm called backpropagation through time (BPTT) (Werbos [31]).

Although RNNs can solve some problems regarding NMT, this architecture has some disadvantages. The main shortcoming is the vanishing gradient problem (Bengio et al. [32]), which could be described as the difficulty that the network has in remembering long-range dependencies in long sequences of inputs. When we are backpropagating the errors throughout the network, the values of the gradient become smaller at each time step, leading the predictions to be biased towards recent inputs, receiving little influence from the inputs far back in the sequence. In an attempt to solve this problem, LSTMs (Long Short Term Memory) were introduced (Hochreiter and Schmidhuber [9]).

Long Short Term Memory

Long Short Term Memory (LSTM) networks (Hochreiter and Schmidhuber [9]) are a special type of RNN and were developed to avoid the vanishing gradient problem. What differentiates them from the already seen RNNs is the structure of the repeating module that predicts the hidden state. This is replaced by an LSTM cell unit, which propagates through each time step not only the hidden state $h^{(t)}$ but also a cell state $C^{(t)}$. The LSTM has the capability of adding and removing information from the cell state, regulated by three structures called gates, which calculate which information to let through based on the current input $x^{(t)}$ and the previous hidden state $h^{(t-1)}$. The gates are the following: the forget gate $f^{(t)}$,

that decides which information to get rid of from the previous cell state $C^{(t-1)}$; the input gate $i^{(t)}$, which controls which part of the input is important to keep and pass to the cell state $C^{(t)}$; and the output gate $o^{(t)}$, which decides what should be present in the output of the hidden state $h^{(t)}$ of a given LSTM cell. These gates can be described by equations 2.3, 2.4 and 2.5 respectively,

$$f^{(t)} = \sigma\left(V_f x^{(t)} + W_f h^{(t-1)} + b_f\right) \tag{2.3}$$

$$i^{(t)} = \sigma\left(V_i x^{(t)} + W_i h^{(t-1)} + b_i\right) \tag{2.4}$$

$$o^{(t)} = \sigma\left(V_o x^{(t)} + W_o h^{(t-1)} + b_o\right) \tag{2.5}$$

where V_f, W_f, V_i, W_i, V_o and W_o are weight matrices and b_f, b_i, and b_o are biases.

Both the cell state $C^{(t)}$ and the hidden state $h^{(t)}$, are obtained with equations 2.7 and 2.8 respectively,

$$\widetilde{C^{(t)}} = \tanh(V_C x^{(t)} + W_C h^{(t-1)} + b_C) \tag{2.6}$$

$$C^{(t)} = f^{(t)} \odot C^{(t-1)} + i^{(t)} \odot \widetilde{C^{(t)}} \tag{2.7}$$

$$h^{(t)} = o^{(t)} \odot \tanh(C^{(t)}) \tag{2.8}$$

where V_C and W_C are weight matrices and b_C is a bias. $\widetilde{C^{(t)}}$ represents the candidate values to update the cell state $C^{(t)}$, with its usage controlled by the gate $i^{(t)}$. $\odot$ represents the *Hadamard product*, *i.e.*, the element-wise product. A prediction $\hat{y}^{(t)}$ for the time stamp t can later be done using equation 2.2.

A graphical representation of a LSTM cell is shown in Figure 2.2.

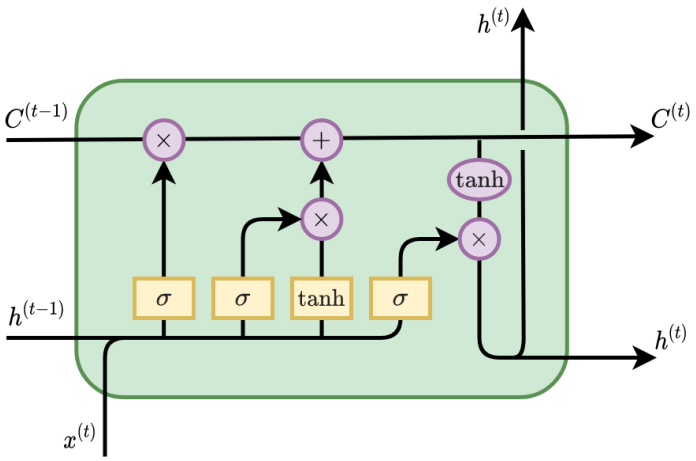


Figure 2.2: LSTM schematic graph.

Even though the LSTM networks mitigate the vanishing gradient problem (Bengio et al. [32]), they are slower to train than RNNs. This happens in part due to the nature of these networks, that forces us to pass input data sequentially, as we need inputs from previous states to make any operations in the current state. This sequential flow prevents us from using parallelization, which impacts speed. Transformers (Section 2.1.3) aim to solve this and other problems with this kind of architecture.

2.1.2 Encoder-Decoder Approach

The main objective of Neural Machine Translation is to transform a sequence of input data, a sentence in a source language, into another sequence of data, a sentence in a target language. When we use the vanilla RNNs and LSTMs (Section 2.1.1) we are limited to sequences of fixed and equal size. Also, it is beneficial to have information about the entire input sequence, in order to start generating the target sequence.

This motivated the introduction of the Encoder-Decoder architectures (Cho et al. [7], Sutskever et al. [10]). These models can be split in two separate parts: the **encoder**, which after processing the input sequence, outputs an encoder vector that aims to encapsulate the information of all input elements; and the **decoder**, which uses the encoder vector as its initial hidden state and, conditioned on the source sentence, generates the target sequence one word at a time. Both the encoder and decoder are a stack of recurrent units as the ones presented in section 2.1.1.

This approach, however, has some drawbacks. The encoder vector needs to encode all the information from the input sentence, which leads to a bottleneck problem. The fact that this is a fixed-length vector is problematic for long sentences, where the size of this encoder vector may be too small to encode everything we should know from the source sentence. Besides this, the words that should be translated into each other might require reordering, which creates long distance dependencies that the model struggles to capture. Bidirectional networks (Schuster and Paliwal [33]), were used to try to overcome this barrier, but the introduction of the **attention mechanism** (Bahdanau et al. [12]) is what made a bigger difference in the success of this task.

Attention

The attention mechanism introduced by Bahdanau et al. [12] and later refined by Luong et al. [13] allows the decoder to make its predictions not only based on its hidden state but also on a representation of every encoder step, essentially a weighted combination of all the hidden states from the encoder. More importantly, it learns what to pay attention to, focusing at each time step only on the relevant parts of the source sequence when generating each output element.

2.1.3 Transformer

The Transformer was a novel neural network architecture introduced by Vaswani et al. [14], mostly based on attention mechanisms. Not only did this new model improve the quality of the output of NMT systems, especially when dealing with longer sentences, but it also boosted the training speed of NMT models, favoring parallelization and therefore making use of modern GPU parallel computation power.

In this subsection the details of the transformer are described, covering the three different sections of the model, the Embedding Layer, the Encoder, and the Decoder. Later, the new attention mechanisms introduced with this architecture, scaled-dot-product attention and multi-headed attention, are analyzed in detail. A graphical representation of the model architecture is presented in Figure 2.3.

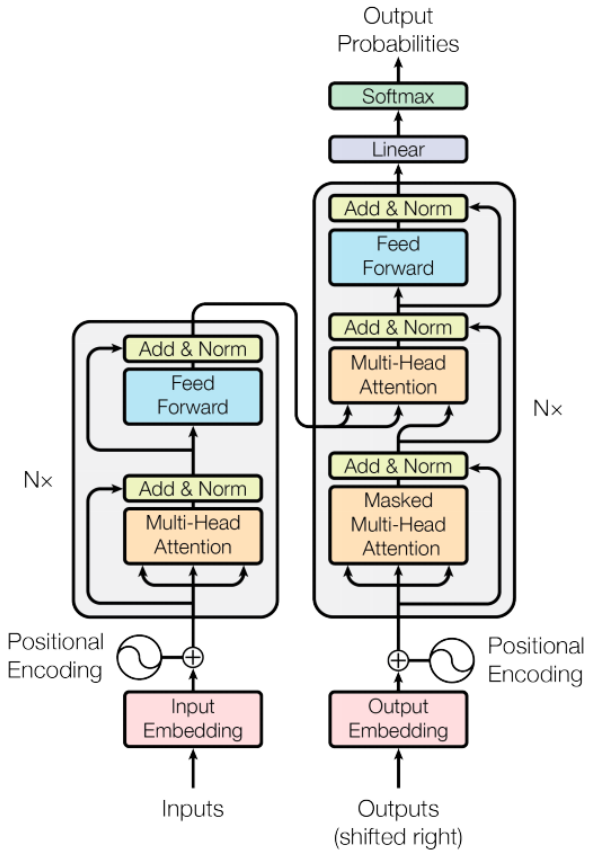


Figure 2.3: The Transformer - model architecture, from [14].

Embedding Layer

The first step of the Transformer, as in any NLP (Natural Language Processing) task, is the embedding layer, where word embedding is applied to the words of a sentence. There are two embedding layers in the transformer, one at the beginning of the encoder and another at the beginning of the decoder. Word embeddings are discussed in more detail in section 2.1.5. For now, the important information to retain is that word embeddings are learned vector representations of a particular word in a sentence, and that both the input and output embedding layers map words to vectors. Since this architecture does not have recurrence like RNNs (Section 2.1.1), there is no information regarding the absolute or relative position of a certain word in a sentence. To this end, positional encoding is added to the input embeddings in an attempt to inject information regarding the order of the sequence. The authors of [14] tested both a trainable positional encoder (Gehring et al. [34]), and a fixed one (equations 2.9a and 2.9b),

$$PE_{(pos,2i)} = sin\left(pos/10000^{2i/d_{model}}\right) \tag{2.9a}$$

$$PE_{(pos,2i+1)} = cos\left(pos/10000^{2i/d_{model}}\right) \tag{2.9b}$$

where pos stands for position, i for dimension, and d_{model} the dimension of the input embedding. The two choices led to similar results.

Encoder

The transformer's encoder is in fact a stack of six encoders (the original paper suggests six but other arrangements were attempted in later works), all identical in structure and without shared weights. This is done in order to learn different input representations, boosting the predictive power of the network. Each encoder layer is broken down into two main sub-layers, the self-attention sub-layer and a feed-forward neural network. The input of each encoder layer is the output of the previous one in the stack, except the first one, which receives the word embeddings.

The self-attention sub-layer allows the model to quantify how relevant each word in the input is to the other words of the input sequence. To accomplish this, for every word, an attention vector that captures the contextual relationships between words in the same sentence is computed. The details of how these attention vectors are calculated will be described in the end of this subsection.

After the self-attention sub-layer, the vectors are passed through a fully-connected feed-forward neural network (Goodfellow et al. [5]) that consists of two linear transformations with a ReLU activation (Glorot et al. [35]) in between. This is done to project the attention outputs potentially giving them a richer representation. Each vector is fed to the same feed forward network (FFN) independently, allowing these computations to be done in parallel.

By analyzing Figure 2.3, we can see that in the encoder, after each one of the two sub-layers mentioned, there is an "Add & Norm" layer. Here, both residual connections (He et al. [36]) and layer-normalization (Ba et al. [37]) are applied. The former consists of adding the input of each sub-layer to its output, which helps the network to train by allowing gradients to flow through the network directly. The latter is done to stabilize the network, reducing the training time.

Decoder

The decoder has a similar structure to the encoder. It is also a stack of decoder layers (six also, as suggested by the paper), with the same structure between them and without shared weights. Besides the sub-layers already mentioned in the encoder (the self-attention sub-layer and the FFN) the decoder has another attention sub-layer between these two that performs attention over the output of the encoder. As in the encoder, each of these sub-layers is followed by a sub-layer where residual connection and layer-normalization happen.

The decoder is auto-regressive, which means that at each step it predicts a token that is consumed in the following iteration as additional input until a special end of sequence token is predicted. Given this auto-regressive property, a change needs to be done in the self-attention sub-layer to ensure that the predictions for a given position only depend on the known outputs from previous positions. To accomplish this, we mask out the positions that correspond to future predictions and so, each position in a sequence only attends to the positions in the sequence up to and including that same position.

The output of the masked self-attention sub-layer is fed to the second attention sub-layer of the decoder, that also takes as input what is outputted by the encoder. Here, every position in the decoder is allowed to attend over all positions in the source input sequence. The output of this "encoder-decoder

attention" sub-layer goes through a FFN with the same characteristics as the ones mentioned for the encoder.

Finally, the output of the FFN is passed to a linear layer that is essentially a fully connected neural network that acts as a classifier. Then this is fed to a softmax layer that transforms the scores given by the linear layer into probabilities, the cell with the highest probability is chosen, and the word associated with that index is the predicted word.

Scaled Dot-Product and Multi-Headed Attention

We now take a deeper look into how the attention is computed inside the transformer, and what are the main innovations brought by the model.

In every attention sub-layer, we start by converting the inputs into three different vectors called Query (Q), Key (K) and Value (V). These are obtained by feeding the inputs of the attention layer into three different fully connected layers that output the three mentioned vectors. If we are dealing with a self-attention layer, we only have one input so the Q, K and V are all computed from the same vector. If it is a encoder-decoder attention layer, the Q is generated from the output of the previous decoder sub-layer, and the K and V vectors are computed from the output of the encoder.

Then, a dot product matrix multiplication is applied to Q and K to produce a score matrix that determines how much focus a word should put on the other words. Then, we do the scaling step by dividing the obtained score matrix by the square root of the dimension of the keys vector (d_k), which allows for more stable gradients, as multiplying values can have exploding effects. Then, these are passed through a softmax function. If we are dealing with a decoder self-attention layer, the masking of the positions that correspond to future predictions is applied before the softmax function. The result of this is multiplied by the values vector and an output vector is obtained. This is shown schematically in Figure 2.4 and is summarized as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.10}$$

(a) Scaled Dot-Product Attention

(b) Multi-Head Attention

Figure 2.4: Transformer attention mechanisms, from [14].

Besides scaling the dot-product attention, the transformer also introduced the concept of **multi-headed attention**. Basically, we do the same scaled dot-product attention shown above, h times, with different weight matrices, where h is the number of attention heads. We split Q, K and V into h different projections that perform the attention function in parallel, resulting in h different outputs that are concatenated and re-projected by a linear layer into the expected dimension from the following sub-layer. Vaswani et al. [14] suggest the use of eight different attention heads ($h = 8$). This mechanism is graphically represented in Figure 2.4, and summarized as follows:

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O \tag{2.11}$$

where $\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. $\mathrm{Attention}$ is equation 2.10 and W_i^Q, W_i^K, W_i^V and W^O are projection matrices.

2.1.4 Vocabulary

When we are processing a sentence to be fed into a NMT model, we first need to break it into more basic and separate units which are called tokens. A vocabulary is basically a set of this tokens. The vocabulary size has a direct impact on the number of trainable parameters in a NMT system. Considering this, we need to find a balance between the coverage of the sub units that might occur for a certain language to avoid having out-of-vocabulary tokens (commonly replaced by an *unknown symbol*, denoted as "<unk>"), and its computational cost. Subword-level vocabularies (Sennrich et al. [20]) are a common choice to have well defined vocabularies that are not too large, and Byte Pair Encoding (BPE) is one of the most common mechanism to obtain these subwords.

Byte Pair Encoding

In Sennrich et al. [20] the idea that different words share common smaller units, the so-called subwords, is explored. The subwords can be separated, and later concatenated to its original form. The authors highlight three groups of words for which the separation into subword units might be more plausible to occur: named entities, cognates and loanwords, and morphologically complex words.

To obtain these subword representations, the authors of the paper adapted the byte pair encoding algorithm (Gage [38]), and instead of merging frequent pairs of bytes, characters or character sequences are merged. When a certain word is split into its subwords, the suffix "@@" is appended to them so that it is easily recognizable where concatenations need to be performed to restore a split word. An example of a sentence after applying BPE can be seen in Table 2.1.

Tokenized sentence	It is a despicable practice .
Tokenized sentence after applying BPE	it is a desp@@ ic@@ able practice .

Table 2.1: Example of the application of BPE to a sentence.

This approach has two main benefits. First, the correct translation of rare and unseen words is

increased. Second, it is reported that the reduced vocabulary size improves the performance of the model in terms of memory usage, and also improves the quality of the translations.

2.1.5 Word Embeddings

If we look at NMT as a black box, it transforms a sequence of words into another sequence of words. However, the neural networks of which these systems are composed deal with numerical data. Word embeddings are the mechanisms that represent words as numerical vectors. To do so, there are two possible approaches, one-hot encoding and dense feature embeddings.

To encode a word using one-hot encoding, a binary vector is produced, where each position corresponds to a word in the vocabulary. The vector is full of zeros with the exception of the index that corresponds to the word being encoded which is a one. This approach has two major problems. First, the vector ends up having the same dimension has the vocabulary, which is too large when compared to the common dimension of the input used in the most common NMT architectures. Second, with this encoding technique no information regarding similarities between the words is captured and provided to the model.

Dense feature embeddings attempt to overcome this problem by projecting the words into an embedding space with dimension d, where this dimension is usually much smaller than the vocabulary. In each d-dimensional space, each word has a representation as a vector (the word embedding) of dimension d, and semantically similar words receive similar representations. Word embeddings are a field of research inside NLP, and different approaches to this problem have been proposed, although the most simple solution is to learn a simple embedding matrix. This matrix has its first dimension with the same size of the vocabulary, and the second with the same dimension of the embedding space d. This matrix is trained alongside the NMT model and is expected to learn how to represent each word, presumably sharing similar features between synonyms. This matrix can be seen as a simple lookup table. To encode a sentence, we generate a vector with the indices of the words inside the vocabulary, and since the ith row of the embedding matrix has the word embedding vector of the ith index in the vocabulary, we just need to select the correct rows of the embedding matrix to form the embedded input of the model.

Tied Embeddings

In a typical NMT model, there are two different locations where word embeddings occur, as we do not only have to embed the input sentence but also the target sentence when we are training the model. If we use the simple embedding matrix method described above, there are in fact two embedding matrices in the system that convert words into vector representations.

In the final stages of a generic NMT system there is another linear layer that performs the opposite task (like in the transformer as we saw in section 2.1.3), which converts the output of the model into a vector of scores. This conversion is done with a matrix that is very similar to the target embedding matrix, as they share the same size (one dimension is equal to the target vocabulary size and the other one, the output dimension of the model, which is usually the same as the input dimension). This matrix

can be referred to as the output embedding matrix, and it is also expected that after training, rows corresponding to similar words share some similarities between them.

Given this similarity between the target input and output embedding matrices (U_{target} and V respectively), Press and Wolf [39] showed that we can improve the performance of NMT models by tying these two matrices, *i.e*, forcing $U_{target} = V$. Actually, results show that, surprisingly, the embedding matrix of the tied model evolves in a more similar way to the output embedding than to the input embedding of the untied model. Also when the preprocessing of the dataset is done using BPE (see section 2.1.4), many of the subwords appear in both languages of the language pair if they share many characters among them. Consequently, Press and Wolf [39] also propose to use a shared single vocabulary and also tie the input embedding matrix (U_{source}), forcing $U_{source} = U_{target} = V$. Another great improvement of this approach is the fact that the number of trainable parameters decreases, which contributes to both a decrease in memory usage and training time, while also improving the quality of the translations.

2.2 Evaluation Metrics

In order to evaluate the quality of the outputs of machine translation systems, and to overcome the need to rely on human evaluation, which is time consuming, different automatic evaluation metrics have been proposed, attempting to maintain a good correlation with human judgment. We will take a deeper look into two evaluation metrics, BLEU (Papineni et al. [40]) and COMET (Rei et al. [41]).

2.2.1 BLEU

BLEU was proposed by Papineni et al. [40] and stands for Bilingual Evaluation Understudy. This metric is widely used in NMT due to its simplicity, efficiency, language independence, and fair correlation with human evaluation. BLEU is reported at corpus-level and its output scores lie between 0 and 1 (usually presented as percentages). The higher the score value, the closer the candidate translation is to the reference and consequently the better it is. In order to understand how BLEU is computed, two concepts must be introduced, brevity penalty and n-gram modified precision, being a n-gram a contiguous sequence of n items from a given sample of text.

Brevity Penalty (BP) attempts to penalize sentences that are shorter than the reference, and is calculated with,

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-\frac{r}{c})} & \text{if } c \leq r \end{cases} \tag{2.12}$$

where c and r are the length (number of words) of the candidate and reference translations, respectively. Regarding the n-gram modified precision, this differs from a normal precision calculation because the counts of each translated n-gram are clipped to the maximum number of times they appear in a reference translation.

The BLEU score is then calculated with,

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n log(p_n)\right) \tag{2.13}$$

where BP is the brevity penalty (equation 2.12), p_n is the n-gram modified precision value, w_n is the uniform weight and is given by $w_n = \frac{1}{N}$, and finally N is the maximum length of the n-grams used. The authors of the paper suggest $N = 4$.

BLEU has several advantages, although it has some limitations as well. There are several elements that influence the final BLEU score, that are not usually reported along side the scores, which might compromise the fairness of comparison between MT systems. The most relevant one is the fact that different preprocessing steps (specially different tokenization methods) are applied to the test sets. In an attempt to standardize the way BLEU score is computed and reported among different MT systems evaluations, sacreBLEU[1] was introduced (Post [42]), which has its own preprocessing steps built in.

2.2.2 COMET

COMET[2] was proposed by Rei et al. [41] and stands for Crosslingual Optimized Metric for Evaluation of Translation. The authors claim to have obtained new state-of-the-art levels of correlation with human judgments. This automatic evaluation metric is neural based, and has the property of using the source as an additional input to perform the evaluation per sentence, alongside the reference sentence and the system hypothesis. To accomplish this, a multilingual embedding space is used, which allows to leverage information from all three inputs. Even though it is possible to train a custom evaluation model with this framework, a recommend pretrained model is available to use. This evaluation metric is segment-level based, therefore the corpus-level score is an arithmetic mean between the segment-level scores. Its scores are not bounded like BLEU for example, and the higher the COMET score the better the system is. Also, when compared to BLEU, COMET has the advantage of capturing semantic similarities.

2.3 Marian

Marian[3] (Junczys-Dowmunt et al. [23]) is an efficient neural machine translation framework written in pure C++ with minimal dependencies (at the moment only CUDA is needed).

When compared to other available frameworks, Marian is specially competitive when it comes to efficiency, speed, and cost-effectiveness. In Junczys-Dowmunt et al. [23] we can see a comparison with the Nematus toolkit (Sennrich et al. [25]), where Marian outperformed it in terms of speed by being four times faster during training with a single GPU, increasing this advantage if multi-GPU training is used. In Junczys-Dowmunt et al. [24], we have a comparison in terms of cost-effectiveness, where Marian outperformed the current baseline produced by the Sockeye toolkit (Hieber et al. [26]), with

[1] Code available in: `https://github.com/mjpost/sacrebleu`
[2] Code available in: `https://github.com/Unbabel/COMET`
[3] Code available in: `https://github.com/marian-nmt/marian`

substantially faster translations both on GPU and CPU, with minimal impact on quality. This was even further optimized in Kim et al. [43]. In Junczys-Dowmunt et al. [23], it is reported that when training a Transformer model, Marian can reach 9100 words per second on an NVIDIA Titan X Pascal, reaching a maximum of 54900 words per second if 8 GPUs are used.

One of the main aspects that contributes to Marian's boost in performance is the fact that it is self-contained with its own back end, which provides reverse-mode automatic differentiation based on dynamic graphs. Even though this could be used to other tasks rather than MT, it was optimized specifically for this and similar use cases. This includes, for instance, various fused RNN cells, attention mechanisms or an atomic layer-normalization.

On top of the auto-diff engine, many efficient meta-algorithms were implemented in Marian. These include multi-device (GPU or CPU) training, scoring and batched beam search, implementation of heterogeneous models like deep RNN models (Miceli Barone et al. [44]), the Transformer (Vaswani et al. [14]) or language models, and multi-node training, to name a few.

Besides all the already mentioned features and optimizations available in Marian, also the extensible Encoder-Decoder framework should be highlighted, which eases the process of implementing new architectures. With top level classes that can be inherited from when new configurations are attempted, one can easily combine different encoder and decoder types (e.g. RNN-based encoder with a Transformer decoder), or to create new ones by only implementing the new inference steps, and then training, scoring and translating with the new model becomes possible.

2.4 Summary

This chapter introduced the necessary background to better understand the work done in this thesis.

Firstly, we introduced the concept of neural machine translation (Section 2.1), giving an overview throughout the different concepts of Deep Learning from which NMT relies on. We explained what are RNNs and LSTMs (Section 2.1.1) and what is an encoder-decoder architecture, highlighting the concept of attention (Section 2.1.2), which lead to the development of the transformer architecture (Section 2.1.3). We gave a more in dept overview of the latter, as it was the architecture chosen in our experiments (Chapter 4). Furthermore, we introduced the concept of vocabulary in Section 2.1.4, where we also explained what is BPE, a mechanism used to separate words into its subwords units, which takes an important part in one of our experiments (Section 4.4). Word embeddings are introduced in Section 2.1.5, as well as the concept of tying the embeddings.

Then, we explained which are the automatic evaluation metrics used to evaluate the performance of our NMT systems, namely BLEU (Section 2.2.1), and COMET (Section 2.2.2).

Finally we did an overview about Marian (Section 2.3), the neural machine translation framework used in this thesis. We highlighted its main benefits presenting some benchmarks and how it compares to other frameworks available.

Chapter 3

Factored Neural Machine Translation

The present chapter introduces the concept of Factored Neural Machine Translation (FNMT).

It starts by presenting the definition of factors (Section 3.1), followed by a historical overview of the several developments done regarding their usage until they were incorporated into NMT, giving origin to FNMT. Subsequently, an overview of the state of the art of this subject is done. We divide this analysis between source factors (Section 3.1.1) and target factors (Section 3.1.2), looking over the changes that need to be done to typical NMT architectures to incorporate them.

In Section 3.2, we detail how factors are implemented in Marian, once again dividing the analysis between source factors (Section 3.2.1), and target factors (Section 3.2.2). Furthermore, alongside the analysis of the original implementation of source factors in Marian (Section 3.2.1), we detail the changes that we performed in our implementation of source factor embeddings.

Finally, in Section 3.3 we show for what applications factors can be used, exploring three different use cases, overviewing the work already done for each one of them, and introducing how we later extended that work.

3.1 Factors

Factors can be defined as a mechanism from which we can represent words in a sentence, not only by the words themselves but as a bundle of features. With this, we can associate to each word several types of attributes like part-of-speech tags, morphological information, or capitalization, to name a few, as a way of enriching the information that each word contains when it's provided to the system. When representing a word with a set of F different features, the first feature ($k = 1$), is called the lemma and it is either the original word or a simplified version of it, depending on what is represented in the remaining features $k \in \{2, ..., F\}$, which we call factors. For example, if we use a factor to represent the capitalization of a word (e.g. "c0" for a lowercase word, and "c1" for a word with the first letter capitalized), the representation of the word "Portugal", would be done by the lemma "portugal" and the additional factor "c1".

The development of factors has been formerly done in the field of language modeling. After statistical

language modeling (Bilmes and Kirchhoff [15]), they were introduced to neural language models by Alexandrescu and Kirchhoff [45], and later extended to recurrent neural language models by Wu et al. [46]. Regarding machine translation, a first attempt of improving the performance of statistical machine translation systems by using factors was done by Yang and Kirchhoff [16], and later by Koehn and Hoang [17], and Bojar [47] all focusing on improving the translation of morphologically rich languages.

All these works inspired later the usage of factors in neural machine translation. As we have seen throughout Chapter 2, NMT transforms a sequence of data (source sentence) into another sequence of data (target sentence). Given this, there are two scenarios regarding the way factors can be used. Information can be appended to the words of the source sentence, resulting in **source factors** (also called input factors), or they can be used for the target sentence, resulting in **target factors** (also called output factors). The introduction of factors in NMT, which gave origin to factored neural machine translation (FNMT), was done by two breakthrough papers, Sennrich and Haddow [18] for source factors, and García-Martínez et al. [19] for target factors. Each approach requires different changes in the NMT architectures detailed in Section 2.1, and those changes will be described in this section.

3.1.1 Source Factors

In an attempt to improve neural machine translation quality by enriching the input data with external linguistic information, Sennrich and Haddow [18] showed how source factors containing various levels of linguistic annotations can be incorporated into a NMT architecture, proving their worthiness.

To do so, an architecture similar to Bahdanau et al. [12] (Section 2.1.2) was used, and the main innovation over it was introducing the representation of the encoder's input as a combination of features, inspired by Bilmes and Kirchhoff [15]. When we are dealing with source factors, the main change to the neural architecture is applied to the embedding layer, keeping the rest of the model unchanged. In the original architecture, the word embeddings E_i of a word x_i in a sentence $x = (x_1, ..., x_N)$ is obtained following the method detailed in Section 2.1.5 using an embedding matrix U. On the other hand, in the factored architecture, each word is represented by a total of F different features, which requires a distinct approach.

In order to embed the lemma and the factors, separate embedding vectors are computed for each feature, which are then concatenated. Summarily:

$$E_i = \Big\Vert_{k=1}^{F} U_k^T x_{ik} \tag{3.1}$$

where $\|$ denotes vector concatenation, $U_k \in \mathbb{R}^{K_k \times m_k}$ is the kth feature embedding matrix, K_k is the vocabulary size of the kth feature, m_k is the embedding size of the kth feature, and $x_{ik} \in \mathbb{R}^{K_k}$ is a one-hot encoding vector indicating the value of the kth feature.

The authors of Sennrich and Haddow [18], experimented with adding to each word four different factors, namely: the morphological lemma of the word; subword tags information (more details about it can be seen in Section 3.3.2); part-of-speech tags and dependency labels, and finally morphological

features. They reported an increase in BLEU score when adding all the features as factors, for both translations directions of the English-German language pair. Tests to evaluate the impact of each factor alone were also performed, concluding that each factor by itself improved the results, even though not as much as using all of them at the same time. Furthermore, they concluded that the gain from using different factors individually was not fully cumulative, which suggested that information encoded in different factors overlaps. Tests were also performed for a lower resource language pair (English-Romanian), also reporting positive results when using all the proposed factors.

3.1.2 Target Factors

García-Martínez et al. [19] showed how target factors can be incorporated inside a NMT architecture.

To accomplish this, an architecture similar to Bahdanau et al. [12] (Section 2.1.2) was used. To incorporate the target factor into it, changes in two different locations had to be performed. The encoder remained unchanged from the original architecture, as both changes were applied to the decoder side. First of all, the decoder embedding layer needed a factored embedding layer, similar to the one seen in Section 3.1.1, so that the model was able to embed both the lemmas and its factors when processing the target corpus of the parallel data during training. Furthermore, at the end of the decoder, the architecture was changed in a way that instead of producing only one output (the predicted word), it predicted both the lemma and its factors in each time step. Following the concept of target embeddings matrix introduced in Section 2.1.5, in the unaltered architecture, after obtaining the last hidden state $h^{(t)}$, a predicted word $\hat{y}^{(t)}$ can be obtained using equation 2.2. If instead of a single word one wants to output a bundle of F features, then F different embedding matrices are needed, therefore predicting each feature $\hat{y}_k^{(t)}$ with a generalization of equation 2.2,

$$\hat{y}_k^{(t)} = \text{softmax}(V_k^T h^{(t)} + c_k), \tag{3.2}$$

where $c_k \in \mathbb{R}^{K_k} \in$ and $V_k \in \mathbb{R}^{m \times K_k}$ are the kth feature bias and output embedding matrix respectively, with K_k the size of the vocabulary of the kth feature, and m the size of the last hidden state (in this case the same as the embedding size, m).

In their experiment, García-Martínez et al. [19] deconstructed each word into its morphological lemma, and the respective morphological tags (see Section 3.3.3). The morphological tags were all concatenated into a string that was later treated as a single factor. Therefore, each word was represented by two features ($F = 2$): the lemma, and the morphological tags factor.

It should be noted that the decoder of the FNMT architecture, predicts lemmas and factors in a synchronous stream, but in separate outputs. Therefore, it's the lemma sequence that is responsible for concluding the decoding predictions of a given sentence, by producing the end-of-sequence token. This is motivated by the fact that the lemmas carry most of the "meaning" of the word.

Due to the new configuration of the decoder, which produced two outputs instead of one, the feedback rule (*i.e.* the previous generated symbol reembedding) also needed to be revised. The authors of [19] tried different configurations for feedback, such as: using only the lemma embedding; using only the linguistic information factor embedding; using a summation of both embeddings, and finally whether

or not the network was able to learn a better combination of the lemmas and factors embeddings, by applying to the summation and to the concatenation of both embeddings a linear and a non linear operation. The results showed that when comparing by BLEU score, the feedback option that produced the best results was the one with the non linear operation ($\tanh$) applied to the concatenation of both the embeddings of lemma and factors. However, even the best system was slightly under-performing in terms of BLEU score when compared to an unfactored NMT system, trained on the same data with the same hyper-parameters.

Given the results, the authors of [19] introduced a lemma dependency to the prediction of the factors, in an attempt of improving the factor predictions and therefore improving the system results. The dependency was implemented by concatenating the predicted lemma embeddings to the hidden layer used to generate the factors. Two different approaches were taken. Either contextualizing a certain factor output with the corresponding lemma being generated at the same time step or using the previously generated lemma to do so. Respectively, by extending equation 3.2, the prediction of a factor with lemma dependency is done with equation 3.3 or 3.4:

$$\hat{y}_k^{(t)} = \text{softmax}\left(V_k^T\left(\left(U_1^T\hat{y}_1^{(t)}\right)\middle\|h^{(t)}\right) + c_k\right), \quad 2 \leq k \leq F \tag{3.3}$$

$$\hat{y}_k^{(t)} = \text{softmax}\left(V_k^T\left(\left(U_1^T\hat{y}_1^{(t-1)}\right)\middle\|h^{(t)}\right) + c_k\right), \quad 2 \leq k \leq F. \tag{3.4}$$

A schematic representation of this decoding layer with lemma dependency based on the lemma predicted at that same time step as the factor (equation 3.3), can be seen in Figure 3.1. Furthermore, the introduction of a dependency of the previously generated factor was also experimented, essentially a custom feedback specific to the layer responsible to output the factors predictions. The prediction of a factor with factors dependency is done with:

$$\text{softmax}\left(V_k^T\left(\left(U_k^T\hat{y}_k^{(t)}\right)\middle\|h^{(t)}\right) + c_k\right), \quad 2 \leq k \leq F. \tag{3.5}$$

From these three different architectures the one that showed the best results was the one that included the previous factor dependency, resulting in a slight BLEU score increase when compared to the baseline, even though the lemma dependency also generated really close BLEU scores to the same baseline.

3.2 Factors in Marian

In this section, an explanation will be made regarding how both the embeddings of source factors and the prediction mechanism of target factors are implemented in Marian. In the subsection related to source factors, we will explain our implementation of concatenation to combine the embeddings of both lemma and factors.

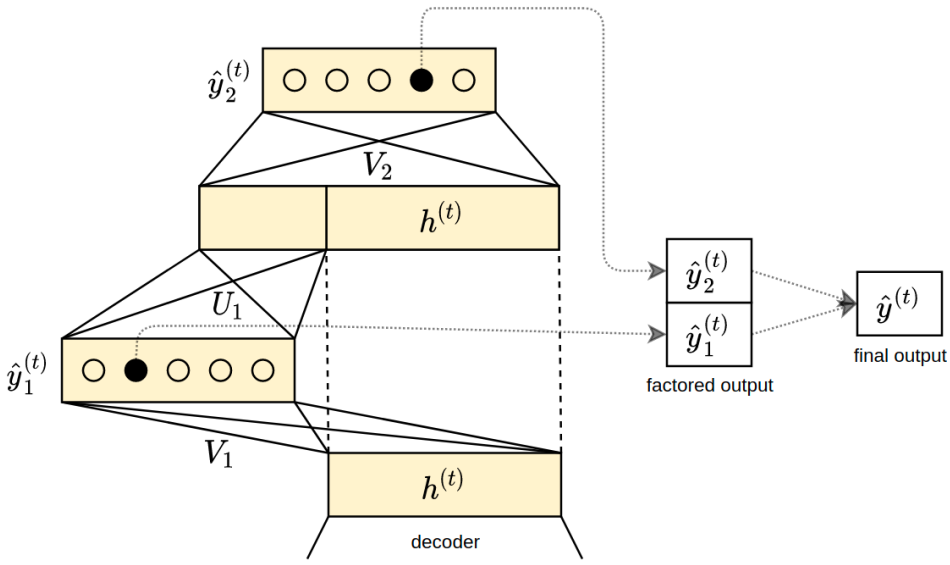


Figure 3.1: Structure of the factored output layer with lemma dependency. Crossed areas represent matrix products. Bias terms were omitted for simplicity. In this example a word was represented by a lemma and only one factor also for simplicity. $\hat{y}^{(t)}$ is obtained by combining $\hat{y}_1^{(t)}$ and $\hat{y}_2^{(t)}$ in post processing. Adapted from [30].

3.2.1 Source Factors

The implementation of source factors in Marian has some differences when compared to the one analyzed in section 3.1.1. Instead of having several embedding matrices for each one of the different features used to represent a word, we have only one large embedding matrix where all this information is encoded. Also, instead of concatenating the lemma and the factor embeddings, these are summed. Let's see how this is accomplished.

To embed a sentence x with N words, $x = (x_1, ..., x_N)$, where each word is represented by F different features, we start by creating a large embedding matrix $U \in \mathbb{R}^{K \times m}$, where K is the vocabulary size of all the features vocabularies K_k combined, $\sum_{k=1}^{F} K_k = K$, and m is the embedding size. In fact, this matrix could be seen as the concatenation (by the rows) of the different factors embedding matrices $U_k \in \mathbb{R}^{K_k \times m_k}$ seen in section 3.1.1, with the constraint that they all have the same embedding dimension $m_k = m$.

When we process the input sentence, we start by constructing a sparse binary matrix $M \in \mathbb{R}^{N \times K}$. Each row of this matrix has a multi-hot encoded vector that encodes the information represented in each word of the sentence x. These vectors differ from the one-hot encoded vector mentioned in Section 2.1.5, because, as the name suggests, instead of having one non-zero value, more than one non-zero value can occur. When we do not have factors, we use one-hot encoding, as each word is represented by a single entry in a certain vocabulary. Marian concatenates all factor vocabularies into one, and consequently, we are able to encode in a single vector which are the lemma and factors represented in a given word, by means of multi-hot encoding.

This matrix M is then multiplied with the embedding matrix U, resulting in the word embeddings $E \in \mathbb{R}^{N \times m}$. Taking into account that each row of the sparse matrix M, has the value 1, in the ith indexes

correspondent to the lemma and factors used in a given token when we multiply it by the embedding matrix that contains in its ith rows the embeddings corresponding to that same lemma and factors, we end up summing the embeddings of the lemma and factors. A schematic representation of this process for the particular situation of representing words with lemma and two factors ($F = 3$), can be seen in Figure 3.2.

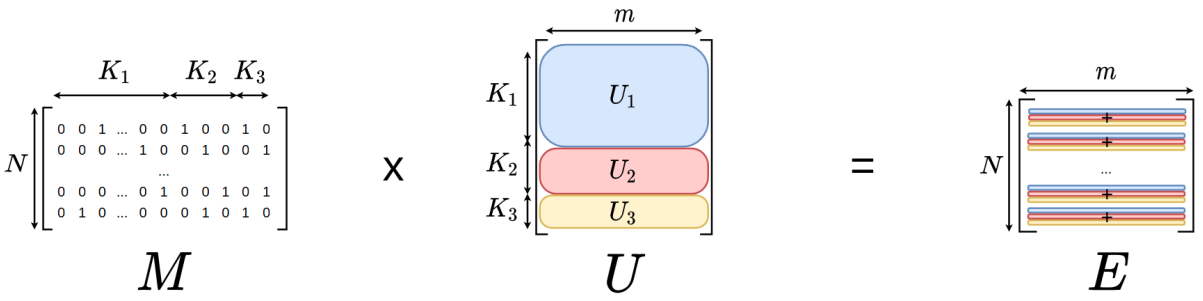


Figure 3.2: Representation of original lemma and factor embedding process of a sentence in Marian. Combining them with sum. For simplicity in this example a word is represented by a lemma and two factors ($F = 3$). N is the sentence length, m the embedding size, K_1, the vocabulary size of the lemma vocabulary, and K_2 and K_3 the size of the vocabulary of each one of the factors. M is a sparse matrix with multi hot encoding vectors in its rows, U is the embedding matrix and E the resulting embeddings. U_k are the sub-embeddings matrices that form the larger embedding matrix U. U_1 embeds the lemmas and U_2 and U_3 the factors.

Even though this approach is beneficial from a computational point of view, as it limits the embedding process to a single matrix product, this approach has some drawbacks. Firstly, it forces that all the factors must have the same embedding dimension as the lemma, not allowing the choice of customized embeddings for the different factors. Secondly, this approach forces that the factor embeddings must be summed with the lemma embeddings, not allowing other options such as concatenation. Thirdly, it does not favor tied embeddings (Section 2.1.5) when only source or target factors are used, due to the fact that the same embedding matrix has to encode the lemma and the remaining factors, and so, if one side encodes both lemma and factors, while the other side only encodes lemmas, the size of the source and target matrices would not match and therefore they would not be eligible to be tied. We next describe the solution we proposed that addresses these issues.

Implementation of Concatenation of the Embeddings

In an attempt of overcoming the problems mentioned above, we decided to implement a new method of combining lemma and factor word embeddings, more similar to what we saw in section 3.1.1.

To do so, we split the embedding process of the lemmas from the embedding process of the remaining factors. We created two embedding matrices $U_{lemma} \in \mathbb{R}^{K_1 \times m}$ and $U_{factors} \in \mathbb{R}^{(K-K_1) \times m_f}$, being m_f the embedding size of the factors, and K_1 the vocabulary size of the lemmas. To embed the lemmas, we can make use of the embedding matrix as a lookup table as explained in Section 2.1.5, and therefore avoid the matrix dot product, thus obtaining the lemma embeddings $E_{lemma} \in \mathbb{R}^{N \times m}$. Then for the remaining factors we create a multi-hot embedding matrix $M_{factors} \in \mathbb{R}^{N \times (K-K_1)}$ that is then multiplied

with the factors embedding matrix $U_{factors}$, thus obtaining the factor embeddings $E_{factors} \in \mathbb{R}^{N \times m_f}$. Finally, the lemmas and factors embeddings are concatenated, in order to obtain the final embeddings $E = E_{lemma} \| E_{factors}$.

This approach favors the tying of the lemma embedding matrix of the source and target side even if only one of them uses factors. Furthermore, with this we are able to control the embedding size of the factors. A schematic representation of this process for the particular situation of representing words with lemma and two factors ($F = 3$), can be seen in Figure 3.3.

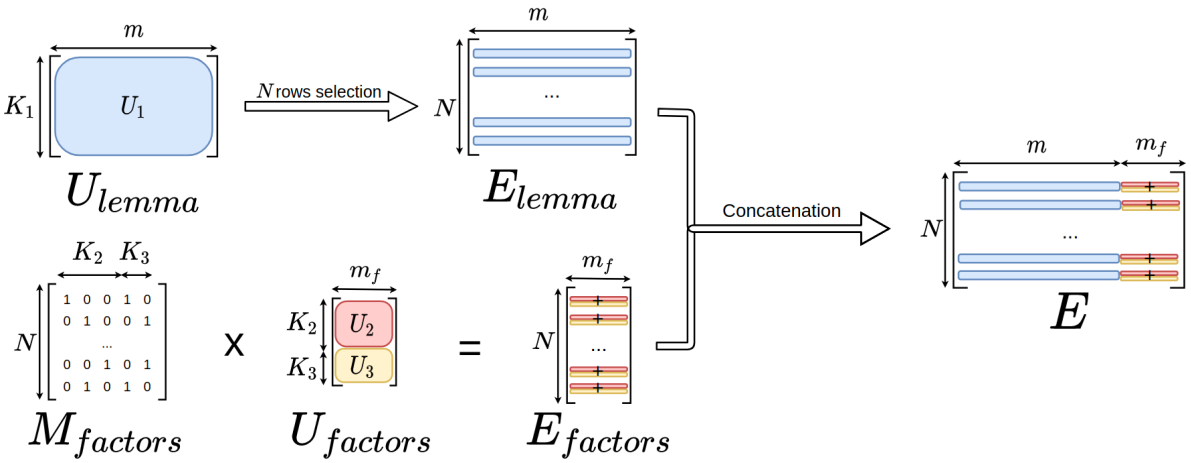


Figure 3.3: Representation of implemented lemma and factor embedding process of a sentence in Marian. Combining them with concatenation. For simplicity in this example a word is represented by a lemma and two factors ($F = 3$). N is the sentence length, m the embedding size of the lemma, m_f, the embedding size of the factors, K_1, the vocabulary size of the lemma vocabulary, and K_2 and K_3 the size of the vocabulary of each one of the factors. $M_{factors}$ is a sparse matrix with multi hot encoding vectors in its rows, $U_{factors}$ is the embedding matrix for the factors and $E_{factors}$ the resultant factor embeddings. Similarly, U_{lemma} is the embedding matrix for the lemmas and E_{lemma} the resultant lemma embeddings. Both $E_{factors}$ and E_{lemma} are then concatenated resulting in the final embeddings E. U_k are the sub-embeddings matrices that form the larger embedding matrices U_{lemma}, and $U_{factors}$. U_1 embeds the lemmas and U_2 and U_3 the factors.

3.2.2 Target Factors

Regarding the target side, the implementation of the algorithm that predicts the different factors has also a different approach than the one explained in Section 3.1.2. There are in fact four different factor prediction options in Marian, that replace the last typical linear layer of the decoder architectures. All four are going to be explained in this subsection. All these different methods start by obtaining the lemma prediction $\hat{y}_1^{(t)}$ by simply applying equation 3.2. This equation uses the output embedding matrix $V_1 \in \mathbb{R}^{m \times K_1}$, which is the output embedding matrix for feature $k = 1$, the lemmas. In Marian, as we did not have for the input embedding layer, we do not have several separate output embedding matrices V_k, but only one large output embedding matrix V. Although, as we mentioned in Section 3.2.1, and as it can also be seen schematically in Figure 3.2, the input embedding matrix U is in fact a concatenation by the rows of the sub embedding matrices U_k. Therefore, following the same pattern, in order to obtain V_1, we just need to select the rows from V correspondent to the first feature, and therefore, we obtain

V_1, allowing us to use equation 3.2.

Soft Transformer Layer

The name of this factor prediction algorithm comes from the fact that the prediction of each factor is done with a condition mechanism that mimics a transformer layer (Section 2.1.3), as all the typical elements of one (attention layer, feed forward layer, and "Add & Norm" layer) are present in it.

After obtaining the lemma prediction $\hat{y}_1^{(t)}$, to predict each factor using the novel condition mechanism that mimics a transformer layer, a multi-headed scaled dot product attention layer (Section 2.1.3) is applied, receiving as inputs both the embedding of the predicted lemma and the outputted last hidden state by the decoder. Subsequently, as in a typical transformer layer, this is passed through a FFN, applying an "Add & Norm" layer after both the attention layer and the FFN.

To the output of this condition mechanism ($h_k^{(t)}$) is then applied equation 3.2, and the factor prediction ($\hat{y}_k^{(t)}$) is obtained. This is then repeated to obtain the remaining factors.

Hard Transformer Layer

This factor prediction method is similar in every way to the soft transformer layer method, with the exception that, when applying equation 3.2, to predict the lemma ($\hat{y}_1^{(t)}$), $\mathrm{hardmax}$ is used instead of $\mathrm{softmax}$.

Lemma Custom Projection

This method for including a lemma dependency in the prediction of the different factors is the most similar to the one seen in section 3.1.2. After obtaining the lemma prediction $\hat{y}_1^{(t)}$, the factors are obtained with a variation of equation 3.3. Firstly, instead of embedding the predicted lemma with the input embedding matrix, the lemma prediction $\hat{y}_1^{(t)}$, is projected to a new dimension m', with a new trainable parameter $D_1 \in \mathbb{R}^{K_1 \times m'}$, being reprojected again to dimension m by $D_2 \in \mathbb{R}^{m' \times m}$. Secondly, the lemma dependency is summed to the outputted hidden state instead of concatenated. Equation 3.3 becomes:

$$\hat{y}_k^{(t)} = \mathrm{softmax}\left(V_k^T \left(\left(D_1 D_2\right)^T y_1^{(t)} + h^{(t)}\right) + c_k\right), \quad 2 \leq k \leq F. \tag{3.6}$$

Also, if $m' = m$, only D_1 is used.

Lemma Dependent Bias

To predict each factor with this method, a lemma dependent bias is computed for each factor, which is later added to the factor logits vector. A logits vector is the name given to a vector before the application of the $\mathrm{softmax}$ function. Therefore, the factor logits of the kth feature, is obtained with equation 3.2, but without the application of the $\mathrm{softmax}$ function. The lemma dependent bias b_k is obtained by multiplying

the predicted lemma logits and a trainable parameter $B \in \mathbb{R}^{K_1 \times K_k}$. The factors are then obtained with,

$$\hat{y}_k^{(t)} = \text{softmax}(V_k^T h^{(t)} + c_k + b_k), \quad 2 \leq k \leq F. \tag{3.7}$$

3.3 Factors Applications

In this section, we present three different possible applications for factors, which correspond to the three different use cases for which we used them in our experiments (Chapter 4). For each one of them, we overview the work done previously in the literature, explaining their experiments and showing their results, finally pointing to how we extended their work with our own experiments.

3.3.1 Factors To Apply Terminology Constraints

Dinu et al. [21] proposed a novel method to inject custom terminology into neural machine translation at run time. The idea was to train the model in a way that it learned a "copying behavior", by providing the target term in the source side, enforcing the translation to produce that term. These target terms were integrated into the source sentences by appending them to their source version. The factors are used to signal this "code-switching" in the source sentence. Three factors were used (the numbers 0, 1, and 2), one to indicate source terms (1), another for the added target terms (2), and finally another for the words that were already part of the source sentence (0). The authors of [21] also tried to add the target term into the source sentence by replacing the source term with the respective translation. An example sentence with terminology injection using both methods can be seen in Table 3.1. In Figure 1.1 we can see a graphical representation of a FNMT system that uses factors for this application.

Source (en)	All **alternates** shall be elected .
Add by appending (en)	All$_0$ **alternates**$_1$ **Stellvertreter**$_2$ shall$_0$ be$_0$ elected$_0$.$_0$
Add by replacing (en)	All$_0$ **Stellvertreter**$_2$ shall$_0$ be$_0$ elected$_0$.$_0$
Reference (de)	Alle **Stellvertreter** werden für eine Amtszeit gewählt.

Table 3.1: Example input for the term injection experiment done by Dinu et al. [21]. Two alternatives ways used to add target terms to the source side can be seen. The terminology entry pair is (alternates, Stellvertreter). Adapted from [21].

The lemma and the factor were embedded with a strategy similar to what we saw in Section 3.1.1, where lemma and factors have separate embedding matrices, and the resultant embeddings are concatenated. Using this factored approach proved to be better when compared to the baseline, increasing the percentage of correctly translated terminology by around 15%, and increasing slightly the BLEU score as well, even though no major increase was expected in the later as terminology affects only a small part of the sentence. Comparing the two strategies for adding the target terms to the source side, using the replacing method actually produced a slighter higher rate (2%) of corrected translated terms, although on the other hand appending the target terms to their source version resulted in better BLEU scores. Given this, train by appending seems a better solution, as one of the goals of this work was

to be able to inject custom terminology without changing the quality of the regular translations. When compared to the best reported model in the space of constrained decoding algorithms for NMT at the time, (Post and Vilar [48]), even though the correct terminology translated was a little underperforming (less 5%), the decoding time was reduced by a factor of four, producing this way a solution with a much higher speed vs performance trade-off.

We extended this work by evaluating the impact of using factors to inject terminology at run time in other language pairs, and also compared different methods to combine the embeddings of the lemma and factors. This experiment is reported in Section 4.3.

3.3.2 Factors To Replace Subword Joining Markers

Both Sennrich and Haddow [18] and Wilken and Matusov [30], used factors to encode subword tags (Section 2.1.4) information, following two distinct approaches:

- Sennrich and Haddow [18] used an annotation similar to the IOB format, having a factor that indicated if a certain subword unit was part of the beginning (B) of a word, the inside of a word (I), or the end of a word (E). (O) was used if a symbol corresponded to a full word, *i.e*, a word not split after applying BPE. A graphical representation of a FNMT system that uses this approach can be seen in Figure 3.4 a).

- Wilken and Matusov [30] used an annotation method that indicated when two consecutive subwords had to be merged. They used the factor "T" to indicate if a subword unit should be concatenated to the previous subword unit in a sentence, and used the factor "F" otherwise. A graphical representation of a FNMT system that uses this approach can be seen in Figure 3.4 b).

Despite the method used, the goal of replacing the subword joining markers with factors was mainly done to allow that in a compound word, for example, a subword unit that could also appear as a single word in the corpus could have the same representation in both situations apart from the factor, therefore sharing the same embedding. Furthermore, this approach reduces the vocabulary size.

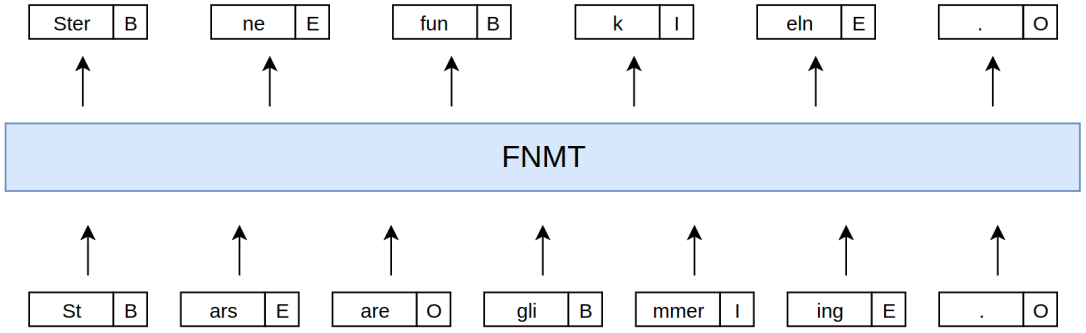
Both Sennrich and Haddow [18] and Wilken and Matusov [30], tested this for the English-German language pair reporting slight increases in BLEU score when comparing their systems to baselines.

In Section 4.4, we show how we extended their works, both by comparing directly these two encoding methods, but also by experimenting it in other language pairs rather than English-German, to evaluate if using factors to encode subwords splits information also had a positive impact on them.

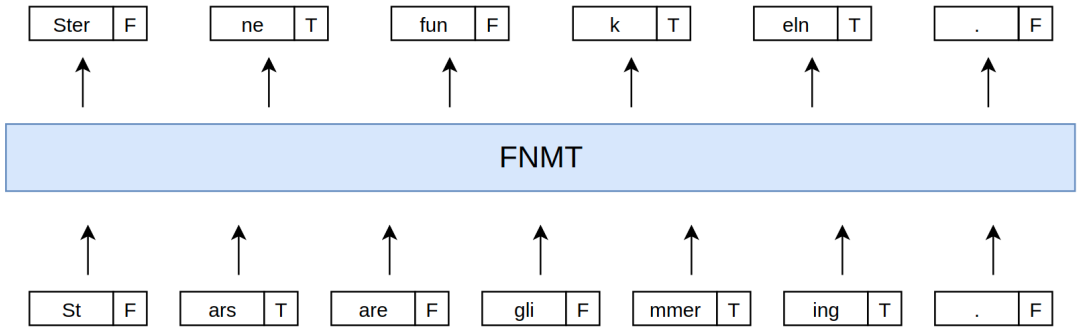
3.3.3 Factors to Encode Morphological Information

García-Martínez et al. [19] used factors to encode morphological information for the target side. Their main motivation was increasing vocabulary coverage and decreasing the number of unknown tokens produced.

In their experiment, each word was deconstructed into its morphological lemma, and the respective morphological tags, both predicted by a morphological analyzer. The morphological tags (POS tags,



(a) Factors used to replace subword joining markers with the method proposed by [18].



(b) Factors used to replace subword joining markers with the method proposed by [30].

Figure 3.4: Example of a FNMT system where factors are used to replace subword joining markers. In this example, the English sentence “Stars are glimmering .”, that after applying BPE becomes “St@@ ars are gli@@ mmer@@ ing .”, is translated to the German sentence “Sterne funkeln .”, that after applying BPE becomes “Ster@@ ne fun@@ k@@ eln .” We can see two different methods of encoding the BPE splits information using factors.

tense, gender, number, and person) were all concatenated into a string that was later treated as a single factor. Therefore, each word was represented by two features ($F = 2$), the lemma, and the morphological tags factor. Moreover, the target vocabulary size was reduced, due to all the derived forms of verbs, nouns, and adjectives not being kept.

After obtaining the predicted sequence of both the lemma and factors, for each pair of predictions, in post-processing, a morphological inflector is responsible for combining the lemma and its morphological information to generate the final word. New words not initially in the vocabulary, like a specific inflection of a certain word, could thus be generated. The authors reported slight increases in BLEU score for the English-German language pair when using the FNMT architecture that produced the highest score (Section 3.1.1). Furthermore, the number of unknown words was reduced to more than half.

A graphical representation of a FNMT system that uses factors to encode morphological information can be seen in Figure 3.5. To enrich the visual representation we represented factors both on source and target sides, therefore the figure does not reflect exactly the work done by García-Martínez et al. [19] as they only used factors for the target side. On the bottom side of Figure 3.5, we can see how the morphological analyzer deconstructs each word into its morphological lemma and the respective morphological tags. On the upper side, after the FNMT model predicts both the lemma and the respective morphological tags, the inflector is the mechanism that generates the correct inflection, generating the

final form of the predicted words.

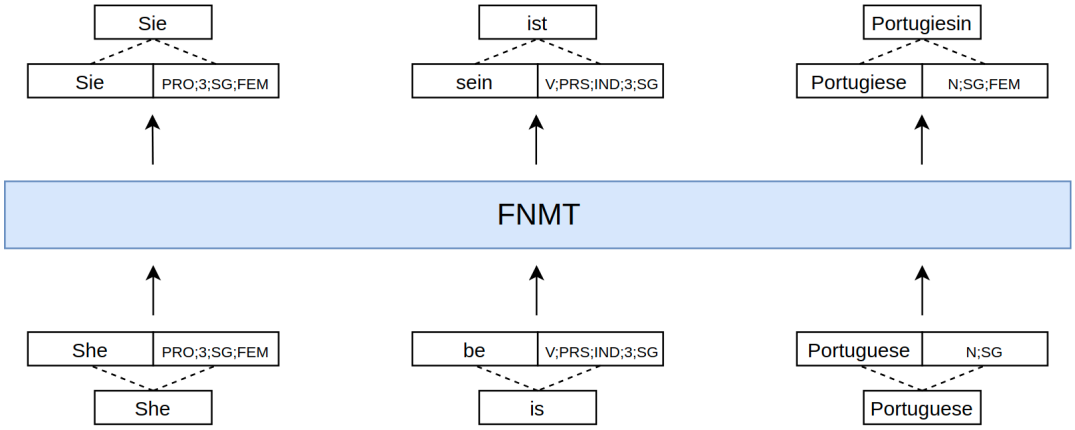


Figure 3.5: Example of a FNMT architecture where factors are used to encode morphological information. Each morphological tag has the following meaning: PRO - pronoun; 3 - third person; SG - singular; FEM - feminine; V - verb; PRS - present; IND - indicative; N - noun;

Motivated by the work done by García-Martínez et al. [19], we followed their approach of encoding morphological information into factors and tried to improve the translation of morphologically rich languages. These experiments are reported in Section 4.5.

3.4 Summary

This chapter presented the concept of factored neural machine translation which are NMT architectures that use factors.

We started by explaining that factors (Section 3.1) are a mechanism from which we can represent words in a sentence, not only by the words themselves but as a bundle of features. This is useful to enrich the information contained in a word representation when it is provided to the NMT system. We then gave an overview of the developments done throughout time regarding factors, until they were incorporated into NMT.

After explaining what are factors, we focused on explaining how factors are used. We divided our explanation by looking first at factors when used on the source side, called source factors (Section 3.1.1), and then to factors used on the target side, called target factors (Section 3.1.2). For both cases, we explained what are the changes that need to be done in a NMT architecture to use them. For source factors, the change must be applied to the embedding layer, where the different features must be embedded and joined in a single final word embedding that is provided to the system. On the other hand, for target factors, besides changing the embedding layer, also a change must be done to the output layer as the model must not only predict a word at each time step, but also the factors associated with it.

We then focused on the Marian toolkit, explaining how factors are implemented in it (Section 3.2). Regarding source factors (Section 3.2.1), we showed that the only option available to combine the embeddings of the lemma and the factors was by summing them. We showed that there are some

drawbacks associated with this solution which motivated us to implement concatenation as an option to combine the embeddings. A comparison between our approach and the original implementation will be done in Section 4.3. Regarding the target side (Section 3.2.2), we explored the different factor prediction mechanisms that are available in the Marian toolkit. All these prediction options will be compared in Section 4.4.

Finally, in Section 3.3 we showed three different possible applications for factors, which correspond to the three use cases for which we used them in our experiments in Chapter 4. For all of them, we review the work done in the literature and introduced how we extended their work. We first gave an overview regarding the work done by Dinu et al. [21] to inject terminology into NMT using factors in Section 3.3.1. The extension that we did to his work can be seen in Section 4.3. In Section 3.3.2 we showed that factors can be used to encode BPE splits information and that it was done in the past with two different approaches, which we compare in Section 4.4. Finally, in Section 3.3.3, we showed that factors can be used to encode morphological information, which motivated us to attempt to improve the quality of the translations of morphologically rich languages in Section 4.5.

Chapter 4

Experimental Analysis

In this chapter we describe the experiments performed, and subsequently, analyze their results.

We start by presenting the datasets used for all our experiments (Section 4.1). Then, we describe all the preprocessing steps applied to the baselines (Section 4.2.1), and the hyperparameters setup used when training them (Section 4.2.2).

Afterwords, we analyze the experiments done using factors in Marian. The experiments are divided into three major groups taking into account the use case for which factors are used. In the first one (Section 4.3), we use factors to apply terminology constraints. In the second one, we use factors to encode subword units (Section 4.4), and finally, we use factors to encode morphological information (Section 4.5). All those sections share the same structure. We start by detailing the experimental setups, where the experiment is described, its goal is presented, and it is detailed the preprocessing steps done and the decisions that were taken. This is followed by a subsection where the results are shown, and subsequently by a subsection where those results are analyzed and conclusions are inferred.

4.1 Datasets

In our experiments, we used four different language pairs. All of them have English (en) as one of their languages, while the other four chosen were: German (de), Latvian (lv), Norwegian (no), and Romanian (ro). The choice of English-German was made due to being the language pair used in the work done by Dinu et al. [21], and extending their research (Section 4.3), was our starting point. The remaining were chosen due to being part of the CEF's Marian agreement, under which this thesis was developed.

For English-German (en-de), we followed the data set up from Dinu et al. [21]. Therefore, we used the data from the WMT 2018 English-German news translation shared task[1] (Bojar et al. [49]). We gathered both the Europarl corpora (Koehn [50]) and the news commentary data corpora to form the training set. For the development (dev) and test sets, we used the 2013 and 2017 WMT English-German news translation shared task[2][3] (Bojar et al. [51], Bojar et al. [52]) test sets respectively. The dimensions

[1] Available in: `http://www.statmt.org/wmt18/translation-task.html`
[2] Available in: `http://www.statmt.org/wmt13/translation-task.html`
[3] Available in: `http://www.statmt.org/wmt17/translation-task.html`

of each dataset can be seen in Table 4.1.

For the English-Latvian (en-lv) and English-Romanian (en-ro) language pairs, we joined the Europarl (Koehn [50]), DGT (Steinberger et al. [53]), JRC-Acquis (Steinberger et al. [54]), and EMEA corpora, all available through the OPUS data base[4] (Tiedemann [55]), to create our datasets. For English-Norwegian (en-no), we used the Tilde-Model[5] (Rozis and Skadiņš [56]), data from the National Library of Norway,[6][7] and finally data from the ELRC-SHARE repository.[8] After collecting the data from these three language pairs, all the corpora was cleaned, in order to remove empty and duplicate lines, as well as lines that were equal for the source and target side. After shuffling, we randomly selected 2.2 million sentences for the training data and 3000 for the dev and test set. Due to the smaller dimension of the English-Norwegian corpora, no random selection was needed, only the division between training, dev, and test sets was done. Also, so that the results could be more comparable to other researches, two commonly used test sets in literature were also used to evaluate our experiments. These are the 2017 WMT English-Latvian news translation shared task[9] (Bojar et al. [52]) test set, and the 2016 WMT English-Romanian news translation shared task[10] (Bojar et al. [57]) test set. For English-Norwegian, no extra test set was used given that it was never a language pair used for a WMT news translation shared task. All the datasets lengths can be seen in Table 4.1.

	Training set	Dev set	Test set	Test set (newstest)
en-de	2204455	3000	-	3004
en-lv	2200000	3000	3000	2001
en-no	1485400	3000	3000	-
en-ro	2200000	3000	3000	1999

Table 4.1: Training, development and test set dimensions (number of lines).

4.2 Baselines

In order to evaluate the performance of the FNMT systems, we trained NMT baselines to compare them to. This section describes the preprocessing steps applied to the data sets (Section 4.2.1), as well as the hyperparameters settings used for training (Section 4.2.2).

4.2.1 Preprocessing Steps

After gathering the corpora for each language pair, we applied the same preprocessing sequence to all datasets. Firstly, we started by normalizing the punctuation and tokenizing the data. After tokenization, we cleaned sentences that had a high length source-target or target-source ratio (higher than 9). We

[4]Available in: `http://opus.nlpl.eu/`
[5]Available in: `https://tilde-model.s3-eu-west-1.amazonaws.com/Tilde_MODEL_Corpus.html`
[6]Available in: `https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-45/`
[7]Available in: `https://www.nb.no/sprakbanken/en/resource-catalogue/oai-nb-no-sbr-46/`
[8]Available in: `https://elrc-share.eu/repository/search/?q=norwegian%20english&sort=resourcename_asc&page=1`
[9]Available in: `http://www.statmt.org/wmt17/translation-task.html`
[10]Available in: `http://www.statmt.org/wmt16/translation-task.html`

subsequently applied truecasing, and finally, we applied BPE[11] (Sennrich et al. [24]) (Section 2.1.4) to segment sentences into subword symbols, using 32000 merge operations, and the vocabulary threshold option set to 50. The BPE learned the merges over a concatenated corpus of the two languages in the language pair. Finally, we built shared vocabularies, in order to later tie the embeddings (Section 2.1.5). All the scripts used for the stages of the preprocessing pipeline prior to BPE were done with scripts from Moses[12] (Koehn et al. [58]).

4.2.2 Hyperparameters

All the baselines were trained under the same hyperparameter setup. The version of Marian used was 1.9.0. We trained a Transformer model (Vaswani et al. [14]) (Section 2.1.3), with six encoder and decoder layers. The model dimension and consequently the embedding size was set to 512, the FFN size was set to 2048 with a depth value of 2, and we used 8 attention heads for the multi-head attention layers. Regarding batches, we used the Marian option of "batch-fitting", which determines the batch size automatically based on sentence-length to fit reserved memory (8500 Mb). We used dropout (Srivastava et al. [59]) of 0.1. Regarding optimization we used the Adam optimizer (Kingma and Ba [60]) ($\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$). Also, label-smoothing of 0.1 (Szegedy et al. [61]), a gradient clipping with a maximum gradient norm of 5 and exponential smoothing were applied. We limited the length of source sentences to 100 tokens. For decoding, we used beam-search, with a beam size of 12. We used early-stopping so that training would stop when the model did not improve for 5 epochs. Finally, the source, target, and output embeddings matrices were tied (Section 2.1.5).

The setup for all the experiments throughout sections 4.3, 4.4, and 4.5, even for the FNMT architectures, were the same as described in this subsection, except where noticed.

4.3 Factors To Apply Terminology Constraints

This experiment is an extension of the work done by Dinu et al. [21], analyzed in section 3.3.1. In this work, lemma and factors were embedded with separate matrices which were later concatenated. The goal of this experiment is to compare this method of combining lemma and factors embeddings, using our implementation of concatenation (Section 3.2.1), to the default Marian option to do so, which is combining them with sum (Section 3.2.1). We evaluate their impact on both the quality of the translations and the percentage of correctly translated terms. Only source factors are used.

4.3.1 Experimental Setup

As seen in Section 3.3.1, Dinu et al. [21] added target terms to the source sentences, so that the model would learn to bias the translation to contain the provided terms. The target terms of the terminology were added to the input as inline annotations with two methods, either by appending the target term to its

[11] Code available in: `https://github.com/rsennrich/subword-nmt`
[12] Code available in: `https://github.com/moses-smt/mosesdecoder`

source version or by directly replacing the original term with the target one. We use the former approach as it was the one that provided the best balance between a good correctly translated terminology rate, and the general quality of the translations.

We also use a factor group with three possible values (“p0”, “p1”, and “p2”) to indicate if a word is either part of a source term (“p1”), part of a target term (“p2”), or if it is not part of a terminology pair (“p0”). The choice of this particular vocabulary for this factor group was done to mimic the choice of Dinu et al. [21] (where the numbers 0, 1, and 2 were used as factors), with the letter ”p” being used due to the fact that in Marian all the factors of the same factor group must share the same prefix. The same preprocessing pipeline described in Section 4.2.1 was applied, adding the factors to the source text before the truecasing step. After applying BPE (Section 2.1.4), if a word that was part of a term was split, the factors were shared among its subword units. In Marian, factors are placed directly in the corpus, by separating the lemma and the factors with a pipe character (“|”). An example of a sentence used in this experiment after being preprocessed can be seen in Table 4.2.

Source	They need clear **guidance**.
Factored Source	they|p0 need|p0 clear|p0 **guidance|p1 An@@|p2 leitung|p2** .|p0
Reference	Sie brauchen eine klare **Anleitung**.

Table 4.2: Example input for the experiment where factors are used for terminology injection, for en-de. The terminology entry is (guidance, Anleitung).

Following the choices from [21], if a certain sentence has multiple matches from a term base, we keep the longest match. Furthermore, when checking for matches of a term inside a source sentence, we apply an “approximate matching”, using a simple character sequence match, allowing a word in the text to be considered a match even if, for example, it happens to be inflected. Moreover, we also limit the amount of data added to 10% of the corpus as we want the model to work equally well when there are no terms provided as input.

Regarding the term bases used, for English-German, English-Latvian, and English-Romanian we used the European Union IATE term base,[13] and for English-Norwegian we used two glossaries from eurotermbank.[14][15] We then filtered out entries occurring in the top 500 most frequent English words as well as single character entries. For the IATE glossaries, we also made some filtering regarding the quality scores given to the terminology pairs, only keeping the ones with the highest scores. We also started by adding the terminology to the dev and test sets, so that those terms were subsequently removed from the glossary used for the training sets, and therefore the tests were performed with unseen terminology.

As explained in Section 3.2.1, when we choose the option of combining the lemmas and factor embeddings with sum, their embedding size must match, and therefore, they were embedded with the model size 512. For concatenation, we followed Dinu et al. [21] and embedded the factors with a size of 16. All the other hyperparameters were kept in accordance with Section 4.2.2. For concatenation, we

[13] Available in: `https://iate.europa.eu`
[14] Available in: `https://www.eurotermbank.com/collections/49`
[15] Available in: `https://www.eurotermbank.com/collections/622`

tied the source, target, and output lemma embedding matrices. As explained in section 3.1.1, tying the embeddings of the source and target side, when only one of them uses factors and when combining the lemma and factor embeddings by summing is, at first sight, difficult to do. Although, we can work around it by creating "dummy" factors for the target side that are actually never added to the target data, only to make the lemma and factor embeddings large matrix match in size. As we will see in section 4.3.3, tying the embeddings has also a considerable impact on the performance of the factors.

4.3.2 Results

The results of this experiment can be seen in Table 4.3. For the language pairs for which we had two test sets (en-lv and en-ro), we evaluated the model on the WMT news task test sets (Section 4.1).

		Term%	BLEU	COMET
en-de	Baseline	78.0	24.75	0.3918
	Factored model - sum	85.8	24.78	0.3863
	Factored model - concatenation	**86.3**	**24.86**	**0.4009**
en-lv	Baseline	56.8	15.90	0.3769
	Factored model - sum	71.2	**15.96**	0.3698
	Factored model - concatenation	**73.9**	15.67	**0.3830**
en-no	Baseline	83.1	22.04	**-0.4013**
	Factored model - sum	91.7	**22.17**	-0.4067
	Factored model - concatenation	**93.2**	22.00	-0.4145
en-ro	Baseline	81.8	24.13	0.4048
	Factored model - sum	92.9	**24.60**	**0.4065**
	Factored model - concatenation	**94.9**	24.29	0.3865

Table 4.3: Results of the experiment where factors are used for terminology injection.

4.3.3 Results Analysis

By analyzing Table 4.3, starting by looking at the column that contains the results of the terminology percentage that was correctly translated, we can realize that the factored model outperforms the baseline for all language pairs. Furthermore, when comparing the two methods of combine lemma and factor embeddings, we also see a pattern among all the language pairs, as concatenation produces a slight increase (+0.5% to +2%) over summation in the correctly translated terms percentage.

Regarding the general quality of the translations, we can not take the same conclusion that concatenation outperforms summing. By analyzing the columns of the BLEU and COMET scores, we notice that for all the language pairs at least one of the two factored models outperformed the baseline. Although, the factored model that leads to the better results is not consistent for all the language pairs.

It is also noticeable that the COMET values for en-no differ a lot from the range of values for the other language pairs. This happens because the models for en-no generated some hallucinations in some of its translations. As COMET is not bounded, unlike BLEU, these sentences are scored with

greatly negative values, decreasing the average score of the corpus. As BLEU is a bounded metric, with its lowest possible score being zero, this is not reflected in its scores. Although, by looking at the column of terminology percentage, we see that independently of that, the results for correctly translated terminology were positive.

In Table 4.4 we can see two examples where combining the embeddings of lemma and factors by concatenating them produces better results than summing them. In the first example (for en-de), we wanted to force the translation of the word "homicide", to be "Tötungsdelikt". We can see that both the baseline and the factored model with sum embeddings chose "Mord", a synonym of "Tötungsdelikt", while the factored model with concatenation chose the correct term translation. In the second example, we injected "sviets" as the wanted translation of the word "butter", for the en-lv language pair. Both the baseline and the factored model with sum embeddings chose the correct base word, although they mis-inflected it, while the factored model with concatenation did not produce the same error.

Example 1: en-de	
Source	"It is a **homicide**".
Reference	"Es handelt sich um einen **Tötungsdelikt**".
Baseline	"Es ist ein **Mord**".
Factored Model - Sum	"Es ist ein **Mord**".
Factored Model - Concat	"Es ist ein **Tötungsdelikt**".
Example 2: en-lv	
Source	Peanut **butter** has nothing to say to a baked onion.
Reference	Zemesriekstu **sviests** un cepts sīpols nav saderīgi.
Baseline	Zemesriekstu **sviestam** nav nekā sakāma ar ceptu sīpolu.
Factored Model - Sum	Zemesriekstu **sviestam** nav sakāms par izceptu sīpolu.
Factored Model - Concat	Zemesriekstu **sviests** nav nekas sakāms par ceptu sīpolu.

Table 4.4: Two examples in which joining the embeddings of the factors and the lemmas by summing them produces worse results than by concatenating them. For Example 1 (en-de), the terminology entry is (homicide, Tötungsdelikt). For the second example (en-lv), the terminology entry is (butter, sviests). The color red indicates the bad translation choice of the injected term.

During this experiment, an interesting find regarding tying the lemma embeddings was made. As explained in Section 4.3.1, when combining the lemma and factor embeddings by summing them only on the source side, we managed to find a way to tie the source and target embeddings matrices. Before realizing how to tie the lemma embeddings with the workaround explained in Section 4.3.1, we tried using lemma and factor embeddings combined with sum, without tying the source and target embeddings matrices. The results can be seen in Table 4.5.

		Term%	**BLEU**	**COMET**
en-de	Baseline	78.0	24.75	**0.3918**
	Untied factored model - sum	81.0	21.65	0.2395
	Tied factored model - sum	**85.8**	**24.78**	0.3863

Table 4.5: Results comparing tying and not tying the lemma embeddings.

By analyzing it, we see that the quality of the translations drops considerably (-3.13 BLEU and -0.1468 COMET) when comparing the untied and the tied model, what was expected. Regarding the percentage of correctly translated injected terminology, the untied model still slightly outperforms the baseline (+2%), although it underperforms (-5.8%) when compared to the tied model. With this, we can infer that tying the lemma embeddings is also important to guarantee a better performance of the information carried within the factors.

4.4 Factors To Replace Subword Joining Markers

In this experiment, we use factors to replace the subword (Section 2.1.4) joining marker ("@@"). Contrary to what we saw in Section 4.3, where we used only source factors, in this experiment we use factors both on the source and the target side. This experiment has the following objectives:

- Firstly, compare the four different decoding options present in Marian (Section 3.2.2), in terms of performance and speed.
- Secondly, compare two approaches taken in the literature (Sennrich and Haddow [18], Wilken and Matusov [30]) to encode subword splits into factors.
- Thirdly, evaluate if replacing BPE joint markers with factors improves the quality of the translations, for the four language pairs explored in this Thesis.

4.4.1 Experimental Setup

In section 3.3.2, we saw two different methods to encode subword splits into factors, proposed by Sennrich and Haddow [18] and Wilken and Matusov [30].

Following Sennrich and Haddow [18] we also used a factor group with 4 possible values ("bb", "bm", "bf", and "bn"). The factor "bb" indicates if a certain unit in the text forms the beginning of a word, "bm" indicates the middle of a word, "bf" indicates the end of a word, and finally "bn" indicates that a certain unit is a full word.

For the other method, following Wilken and Matusov [30], we use a factor group with 2 possible values ("jt", "jn"). These two factors indicate either if a certain token in a sentence should be joined with the previous token ("jt"), or if it should not be joined ("jn").

The same preprocessing steps described in Section 4.2.1, are applied to the corpus in this experiment, adding the factor information after applying BPE. An example of a sentence used in this experiment after being preprocessed can be seen in Table 4.6.

After decoding, in postprocessing, the predicted tokens with the factors "bm", and "bf" are joined with the previous token in the sequence. In the other factored approach the tokens with the factor "jt", are joined with the previous token in the sentence.

Before comparing the two approaches, we compared the different factor prediction options available in Marian and detailed in section 3.2.2. From the four options available, we compared three of them:

Source	The final stop is Lerik.
Reference	Endstation ist Lerik.
Reference with BPE	**End@@ station** ist **L@@ eri@@ k** .
Factored Reference based on [18]	**End**|**bb station**|**bf** ist|bn **L**|**bb eri**|**bm k**|**bf** .|bn
Factored Reference based on [30]	**End**|**jn station**|**jt** ist|jn **L**|**jn eri**|**jt k**|**jt** .|jn

Table 4.6: Example input for the experiment where factors are used to encode BPE splits, for the en-de language pair.

the soft transformer layer, the lemma custom projection, and finally the lemma dependent bias. The hard transformer layer was not used, as the only difference to the softmax transformer layer is the usage of hardmax instead of softmax, which is mainly used for debugging purposes, and so, having also this option in the comparison would not bring any value. Moreover, for the lemma custom projection, we tried three different lemma projection sizes: 16, 100, and 512. To compare this, we used the second method of encoding the BPE splits information, the one that uses the factors "jt" and "jn" (the one based on [30]).

We later used the softmax transformer layer for comparing the two approaches to encode BPE splits information into factors, and to extend the experiment to the other language pairs. As concatenation was not implemented for the target side, the regular Marian factors embedding was used (Section 3.2.1), and all the embedding matrices were tied.

4.4.2 Results

The results on the comparison of the different factor prediction methods available in Marian are reported in Table 4.7, where we can see a comparison regarding the quality of the translations generated, by looking at the BLEU and COMET columns. Furthermore, we also have a comparison regarding training speed in the "Words/s" column, where the values reported are an average of the words processed per second during training. The rightmost column reports the time in seconds that it took to translate the en-de newstest set (3004 sentences) with a batch size of 16. This was trained and evaluated in a GeForce RTX 2080 Ti GPU.

		BLEU	**COMET**	**Words/s (Training)**	**Dec. Time [s]**
en-de	Baseline	24.75	0.3918	**17197**	**50.35**
	Soft transformer layer	**24.89**	0.3919	13336	62.13
	Lemma Custom Projection (16)	24.59	0.3772	15797	56.94
	Lemma Custom Projection (100)	24.70	0.3924	15165	57.27
	Lemma Custom Projection (512)	24.78	**0.3981**	13530	61.23
	Lemma Dependent Bias	24.78	0.3821	15191	56.30

Table 4.7: Results on the comparison of the different factor prediction options available in Marian. The BPE splits factor encoding method is the one based on [30].

Then, in Table 4.8, we can see the results of using factors to encode BPE splits for various language pairs. We can also see a comparison of the two methods to represent the subword splits with factors. For the language pairs for which two test sets were available (en-lv and en-ro) (Section 4.1), the WMT

news task test sets were used.

		BLEU	COMET
en-de	Baseline	24.75	0.3918
	Factored model (BPE encoding based on [18])	24.85	0.3719
	Factored model (BPE encoding based on [30])	**24.89**	**0.3919**
en-lv	Baseline	15.90	**0.3769**
	Factored model (BPE encoding based on [18])	**15.97**	0.3442
	Factored model (BPE encoding based on [30])	15.50	0.3707
en-no	Baseline	**22.04**	**-0.4013**
	Factored model (BPE encoding based on [18])	21.10	-0.4451
	Factored model (BPE encoding based on [30])	21.11	-0.4184
en-ro	Baseline	**24.13**	**0.4048**
	Factored model (BPE encoding based on [18])	24.06	0.3531
	Factored model (BPE encoding based on [30])	23.76	0.3960

Table 4.8: Results on encoding BPE splits with factors with two different methods, for different language pairs.

4.4.3 Results Analysis

By analyzing Table 4.7, where we have the comparison of the different Marian factor prediction options, we can see that if evaluating based on BLEU, the model that generated the best translations was the one that uses the soft transformer layer to predict the factors, while the model that had the highest COMET score was the model that reprojects the lemma embeddings to a layer with size 512. We can also see that the only model that underperformed the baseline in both BLEU and COMET is the one that predicts factors with the lemma custom projection of size 16, probably due to not capturing enough features given the short dimension of the lemma reembedding size. In terms of speed, we can see that using target factors decreases the speed of both training and inference. All the methods to predict factors were slower than the baseline, which means that, even though the vocabulary size is reduced when replacing the BPE joint markers "@@" by factors (the vocabulary size for the baseline was 30915 tokens while the vocabulary size in the factored models was 28653), it does not balance the increase in the number of trainable parameters and in the complexity of the model, thus not allowing for a decrease in training and translation times. Between the different methods, the soft transformer layer and the lemma custom projection with a size of 512 are the slowest as they are the ones that introduce the highest number of trainable parameters. Although, this also turns to be the case that generates the better translations, so these should be the choice when time is not an issue. Furthermore, the lemma dependent bias and the lemma custom projection with a medium size (100), appear to be the methods with the best balance between speed and quality of the translations.

Regarding the comparison of the two methods to encode BPE splits as factors when looking at Table 4.8, we can see that if compared with BLEU, the method proposed by Sennrich and Haddow [18] seems to be more effective for en-lv and en-ro that the method proposed by Wilken and Matusov [30]. However,

when looking at the COMET column the same conclusion cannot be taken, as the method proposed by Wilken and Matusov [30] outperforms the method proposed by Sennrich and Haddow [18] for all the language pairs with a considerable margin (between +0.02 and +0.0429). Besides this, when comparing the highest scored factored model with the baseline for each language pair, only for the en-de language pair we see a positive result for which the best scoring factored model (the one with the method proposed by [30]), outperformed the baseline on BLEU and on COMET. Even though the increase on the latter is too small (+0.0001) at least we can conclude that the performance has not been compromised. For the remaining language pairs, only for the language pair en-lv we see a factored model outperform the baseline on BLEU (+0.07), although the same comparison by COMET (-0.0327) does not report the same positive result.

We can conclude that even though encoding BPE splits with factors seems to have a positive impact for English-German, as also reported in the literature (Sennrich and Haddow [18], Wilken and Matusov [30]), when extending it to other languages where this was never tested, it does not lead to the same performance improvements. Nevertheless, the method proposed by Wilken and Matusov [30] proved to be more effective than the method proposed by Sennrich and Haddow [18].

4.5 Factors to Encode Morphological Information

The goal of this experiment is to evaluate if representing words by their morphological lemmas and respective morphological information encoded in factors has a positive impact on the performance of NMT models. We experimented it for both directions of each language pair, which means, translating from English and to English. The morphological factors were added always to the non-English language in the language pair, as English is not particularly morphologically rich, and so, having its morphological information encoded into factors did not seem relevant. Therefore, we used factors either on the source or the target if the translation generated was to English or from English, respectively.

4.5.1 Experimental Setup

In order to encode morphological information into factors, there are a total of three systems that should be used. Firstly a morphological tagger to predict the morphological tags of a certain word. Secondly, a lemmatizer which reduces each word to their morphological lemma. And thirdly a morphological inflector, that after the translation is predicted by the NMT model combines the lemma and the respective morphological tags and generates the final form of a certain word with the correct inflection.

The morphological tagger and lemmatizer system[16] used (Malaviya et al. [62]) form a sequence of two neural models, where the first one, the morphological tagger, predicts a sequence of morphological tags of a certain word, and subsequently, the lemmatizer uses the predicted tags and the word to predict the morphological lemma. One of the positive aspects of this architecture is that it was developed to work for several different languages, therefore being possible to use it for all the languages used in this

[16]Code available in: `https://github.com/sigmorphon/contextual-analysis-baseline`

thesis. The models were trained under the Universal Dependencies (UD) Treebanks (Nivre et al. [63]), although the UD schema was converted to the UniMorph schema (Sylak-Glassman [64]) instead.[17] This schema attempts to standardize the tags used to represent morphological features. As an example, the word “does” would be tagged as being a verb, finite, with the indicative mood, in the present tense, third person, and singular which in the UniMorph schema would be “V;FIN;IND;PRS;3;SG”.

The morphological inflector[18] used (Peters and Martins [65]), can also be trained for all the languages used in this thesis, and it was trained in the exact same data as the tagger and the lemmatizer. The accuracy obtained after training the morphological tagger, lemmatizer and morphological inflector can be seen in Table 4.9.

	Tagger	**Lemmatizer**	**Inflector**
German	82.34	96.17	91.93
Latvian	88.92	93.26	88.69
Norwegian	93.76	97.33	93.12
Romanian	94.84	96.68	91.62

Table 4.9: Accuracy obtained for the morphological tagger, lemmatizer and morphological inflector.

Regarding the factors used, we combined all the morphological tags into a single string so that they would be treated as a single factor. Furthermore, the word is replaced by its morphological lemma. This way, using the previous example of the word ”does”, this word would be represented as, ”do|mV;FIN;IND;PRS;3;SG”. As explained before, the letter “m” is only used because it is mandatory that in Marian all the factors from the same factor group share the same prefix. Furthermore, if a certain token does not carry morphological information, as punctuation characters for example, the factor “mNOFACT” was used. The size of each factored vocabulary is language dependent, as the number of morphological tags from the UniMorph schema that each language has differs. Some statistics regarding the number of factors used, the number of different morphological tags that combined make that factors, and the number of UniMorph schema morphological groups from which they belong can be seen in Table 4.10.

	Factors	**Morphological Tags**	**Morphological Groups**
German	1081	31	12
Latvian	1232	50	15
Norwegian	265	42	14
Romanian	550	48	14

Table 4.10: Statistics regarding the number of factors, morphological tags, and the number of different groups from which the morphological tags belong, used for the experiment where factors encode morphology.

The same preprocessing pipeline described in Section 4.2.1 was applied in this experiment. The factors with the morphological tags and the replacement of the words with their morphological lemmas

[17] Data available in: `https://github.com/sigmorphon/2019/tree/master/task2`
[18] Code available in: `https://github.com/deep-spin/sigmorphon-seq2seq`

was done before the truecasing step, for the non-English language of a given language pair. After applying BPE (Section 2.1.4), the factors information was shared among the subword units of a split word. An example of a sentence used in this experiment after being preprocessed can be seen in Table 4.11.

Source	Sounds tricky?
Reference	Klingt schwierig?
Factored Reference	klingen|mV;FIN;IND;PRS;3;SG; schwierig|mADJ ?|mNOFACT

Table 4.11: Example input for the experiment where factors are used for encoding morphological information. Example for de en-de language pair.

If factors are used for the target side, after predicting the lemma and the factors with the FNMT model, the inflector is used to combine that information and generate the final word in postprocessing. If a certain token is predicted with the "mNOFACT" factor, the predicted lemma is kept unchanged.

Given the neural nature of the tagger and the lemmatizer, the time needed to preprocess the data increased dramatically. In order to decrease the preprocessing time, we reduced the training sets to half the size reported in Table 4.1, therefore retraining also the baselines with the shorter datasets so that the comparison between the NMT model and FNMT model could be fair. The same hyperparameters described in 4.2.2 were used.

The lemma and factor embeddings were combined by summing them (Section 3.2.1), and when using them on the target side, the method used to predict the factors was the softmax transformer layer (Section 3.2.2). The source and target embedding matrices were tied using the workaround described in Section 4.3.1.

4.5.2 Results

The results for the translations with English in the target side and therefore the language with the morphological information encoded in factors in the source can be seen in Table 4.12. For the language pairs for which two test sets were available (en-lv, and en-ro) (Section 4.1), the WMT news task test sets were used.

Regarding translating from English to the languages with the morphological information encoded as factors, the results can be seen in Table 4.13. As we can see in that table we did two different experiments for this translation direction. While in one we factored all the words in the corpus, in the other only the words of certain part-of-speech groups (nouns, verbs, adverbs, and adjectives) had their representation changed to lemma and respective morphological tags. Furthermore, on the first of these two experiments an evaluation solely based on the lemmas was also done. An analysis of these results and the motivation to do these two extra evaluations will be made in Section 4.5.3.

Finally, for the English-Romanian and English-Latvian language pairs, since we had another extra test set with data more similar to the training data, as explained in Section 4.1, we decided to also reevaluate this experiment with those test sets and therefore, see the variations that evaluating in a more in-domain data could bring. Those results can be seen in Table 4.14.

		BLEU	COMET
de-en	Baseline	**27.66**	**0.3125**
	Factored model	27.19	0.2908
lv-en	Baseline	**12.87**	-0.1521
	Factored model	12.43	**-0.1492**
no-en	Baseline	**23.73**	**-0.5505**
	Factored model	23.00	-0.5562
ro-en	Baseline	23.77	0.0942
	Factored model	**26.65**	**0.2468**

Table 4.12: Results on using factors to encode morphological information for the source language when translating to English.

		All words w/ factors			Some POS w/ factors	
		BLEU	COMET	BLEU (lemmas)	BLEU	COMET
en-de	Baseline	**24.36**	**0.3605**	**27.52**	**24.36**	**0.3605**
	Factored model	21.18	0.1527	26.15	22.40	0.2420
en-lv	Baseline	**11.82**	**-0.0296**	**14.24**	**11.82**	**-0.0296**
	Factored model	10.01	-0.2727	13.70	10.78	-0.2045
en-no	Baseline	**20.98**	**-0.4464**	**22.10**	**20.98**	**-0.4464**
	Factored model	16.97	-0.6205	20.63	18.21	-0.5462
en-ro	Baseline	19.15	**0.1858**	23.40	19.15	**0.1858**
	Factored model	**19.20**	-0.1093	**25.16**	**19.26**	-0.0568

Table 4.13: Results on using factors to encode morphological information for the target language when translating from English. The baseline results were duplicated between the “All words w/ factors”, and “Some POS w/ factors” columns, to ease the comparison between them and the results of the factored models.

		BLEU	COMET
lv-en	Baseline	**60.57**	**0.6761**
	Factored model	58.01	0.6475
en-lv	Baseline	**53.64**	**0.9485**
	Factored model	41.84	0.6677
ro-en	Baseline	37.85	-0.1214
	Factored model	**49.68**	**0.1078**
en-ro	Baseline	31.51	**-0.0622**
	Factored model	**32.00**	-0.1712

Table 4.14: Results on using factors to encode morphological information reevaluated with a more in-domain test set.

4.5.3 Results Analysis

Starting by analyzing the results of translating to English (x-en), reported in Table 4.12, we can see that the factored model only resulted in an improvement for the ro-en language pair, for which we can see

an increase in both BLEU and COMET scores. For the remaining language pairs, even though no other surpassed the baseline, they were very close, especially when comparing with COMET.

Regarding the translations from English to the language with the morphology encoded in the factors (en-x), if we look at the first two columns of Table 4.13, we see that all the factored systems underperformed when compared to the baseline. Only for en-ro and when evaluated with BLEU, we see a slight increase, although that improvement is not reflected in the respective COMET score.

As explained in Section 4.5.1, when splitting the words into morphological lemma and respective morphological tags for the target side, once both the lemma and the factor are predicted, we must use the inflector to combine them and generate the final word. In order to evaluate if the lacking in performance was due to an underperforming inflector, we computed BLEU solely based on lemmas. To do so, we lemmatized the reference and removed all the factors from the FNMT hypothesis. In order to have something to compare this to, we lemmatized all the baseline hypothesis as well. As the sentences lose their linguistic meaning due to all the words not being inflected, we did not use COMET for this analysis. This evaluation is also useful to understand if the FNMT model is at least choosing the correct words. These results can be seen in the third column of Table 4.13. Firstly all the BLEU scores are higher when compared to the inflected corpus (first column) which is expected, as mis-inflections are not taken into account in this scenario. When comparing the factored model with the baselines we see that except for en-ro once again, the factored models are still underperforming, suggesting that despite the errors injected by the inflector and the missed predicted factors, the factored model is struggling to choose the correct words when compared to the baseline.

When looking for answers for these unsatisfying results, by analyzing the hypothesis outputs we saw that the tagger and subsequently the lemmatizer were having some unexpected behaviors, as some words were being wrongly tagged when they were not supposed to generate tags (like punctuation tokens or numbers), as well as words were being wrongly lemmatized, noticing some errors especially for pronouns and determiners. Given this, the FNMT models were retrained with data that only had words represented by lemma and morphological tags if they were nouns, verbs, adjectives, or adverbs, as that group of words is the one were inflections are most common and relevant. If we look at the two rightmost columns of Table 4.13, we can see that even though the factored models got closer to the baselines, they were still underperforming, with again the exception of en-ro. This suggests that limiting the prediction of lemmas and morphological tags to that restricted group of POS words, stabilized the translations, even though not enough to say that using factors to encode morphology brings benefits when translating from English, with the exception of the translation to Romanian.

We also decided to use the two extra test sets that we had for en-ro and en-lv, which contained data more similar to the training data, and reevaluated the experiments in both directions of each language pair. The goal with this was to understand if evaluating with more in domain data would close the gap between baseline and underperforming factored models and increase it for cases where the factored models outperformed the baselines, indicating that the factored models would benefit from being evaluated in more in-domain data. When comparing the results from Table 4.14, with the previous ones seen in Tables 4.12 and 4.13, the conclusion that we can take is that evaluating with more in-domain data

amplifies the gaps between baseline and the factored models either if the latter underperformed (lv-en, en-lv) or outperformed (ro-en, en-ro) the baseline in the previously done evaluations.

Finally, comparing the overall results of the translations in both directions of all the language pairs used, it was already expected that we got higher results when translating to English than from English, as usually, systems that translate to English tend to score higher due to the low morphological richness of this language. Furthermore, the higher scores in the factor models that translate to English and consequentially only use factors on the source side could be explained with the fact that there's no need to use the inflector, therefore having one fewer step in the translation pipeline and consequently one system fewer introducing possible failures. Moreover, the fact that all the models used to predict morphology related features (tagger, lemmatizer, and inflector), have in average a 10% margin for error (Table 4.9), could lead to the injection of noise into the corpus and harm the quality of the translations of the factored models.

We conclude that with the current setup, representing words by their morphological lemmas and respective morphological information encoded in factors only brings benefits to translate between English and Romanian.

4.6 Summary

Throughout this chapter we presented the results from the three experiments that we performed, where we evaluated the impact of using factors for three different use cases, namely, the application of terminology constraints, encoding subwords splits, and encoding morphological information. Parallel to this we also used the experiments to take considerations regarding the implementation of factors in Marian.

In the first experiment (Section 4.3), where we used factors to apply terminology constraints, we showed that our implementation of combining lemma and factor embeddings by concatenating them outperformed the method originally implemented in Marian, which was summing. Furthermore, we showed that factors are a valuable solution to inject custom terminology at run time.

In the second experiment (Section 4.4), where we used factors to encode subwords splits after applying BPE, we compared two different approaches to do so. Furthermore, we saw that for this use case, factors are especially beneficial for the English-German language pair. We also compared the different methods available in Marian to predict factors in terms of performance and speed of both training and inference.

In the third experiment (Section 4.5), where we used factors to encode morphological information, we saw that for this use case, factors can lead to considerable improvements in the quality of translation in both directions of the English-Romanian language pair.

Chapter 5

Conclusions

In this chapter, we summarize the main contributions and achievements of this thesis, followed by a suggestion of possible directions for future work.

5.1 Achievements

The main goal of this thesis was to test and evaluate the usability of the factors' code in the Marian toolkit, fulfilling the first milestone of the CEF's "User-Focused Marian" agreement. We contributed to the open source Marian's codebase with the implementation of concatenation as an option to combine lemmas and factor embeddings. We analyzed the performance of factored neural machine translation models by conducting three experiments, that use factors for three different use cases, evaluating them for the English-German, English-Latvian, English-Norwegian, and English-Romanian language pairs.

In the first experiment, we used factors to inject custom terminology into NMT at run time. We saw that our implementation to combine lemmas and factor embeddings by concatenating them outperformed in terms of correctly translated terminology the method originally implemented in Marian, which is combining them with sum. We extended the work done by Dinu et al. [21], by comparing these two embedding options and also by extending their research to three other language pairs, showing positive results for all of them. We also showed the importance of tying the lemmas embeddings for improving the performance of the usage of factors.

The second experiment used factors to replace the subword joining markers. We compared two methods of representing this subword splits proposed by Sennrich and Haddow [18], and Wilken and Matusov [30], and concluded that the latter is the one that leads to better results. Furthermore, we showed that even though improving the translation quality in the English-German language pair, this usage of factors does not have a positive impact on the remaining tested language pairs. This experiment was also used to do a detailed analysis regarding the different factor prediction methods available in Marian, evaluating them in terms of quality of the translations generated, and speed of both training and inference.

Finally, in the third experiment, we used factors to encode morphological information, aiming to

improve the translation quality of morphologically rich languages. Considerable improvements were obtained for both translations directions of the English-Romanian language pair. For the remaining language pairs, the results were underwhelming, in particular when translating from English to the remaining languages.

5.2 Future Work

From the three use cases for which factors were used in this thesis, using them to encode morphological information and therefore attempt to improve the translation quality of morphologically rich languages is where there is more space for future development. Extending our research by using morphological taggers, lemmatizers, and morphological inflectors with a higher rate of accuracy could lead to better results. Instead of using models thought in order to be used for several languages, using implementations more specific for a certain language could lead to improvements. Furthermore, instead of gathering all the morphological information into one single factor, one could try to split the different morphological groups among different factors.

Finally, the implementation of concatenation as a method to combine the lemma and factor embeddings in Marian was only done for the source side. The factor decoding mechanisms already available in the codebase could be adapted to incorporate this lemma and factor combination method that showed promising results when compared to the original available embedding method.

Bibliography

[1] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2): 79–85, 1990. URL https://www.aclweb.org/anthology/J90-2002.

[2] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993. URL https://www.aclweb.org/anthology/J93-2003.

[3] D. Marcu and D. Wong. A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 133–139. Association for Computational Linguistics, July 2002. doi: 10.3115/1118693.1118711. URL https://www.aclweb.org/anthology/W02-1018.

[4] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003. URL https://www.aclweb.org/anthology/N03-1017.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

[6] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA, Oct. 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1176.

[7] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

[8] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL http://www.sciencedirect.com/science/article/pii/036402139090002E.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[10] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

[11] J. Gehring, M. Auli, D. Grangier, and Y. Dauphin. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1012. URL https://www.aclweb.org/anthology/P17-1012.

[12] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 2015 International Conference on Learning Representations*, 2015. URL https://arxiv.org/pdf/1409.0473.pdf.

[13] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[15] J. A. Bilmes and K. Kirchhoff. Factored language models and generalized parallel backoff. In *Companion Volume of the Proceedings of HLT-NAACL 2003 - Short Papers*, pages 4–6, 2003. URL https://www.aclweb.org/anthology/N03-2002.

[16] M. Yang and K. Kirchhoff. Phrase-based backoff models for machine translation of highly inflected languages. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy, Apr. 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E06-1006.

[17] P. Koehn and H. Hoang. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 868–876, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D07-1091.

[18] R. Sennrich and B. Haddow. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers*, pages 83–91, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-2209. URL https://www.aclweb.org/anthology/W16-2209.

[19] M. García-Martínez, L. Barrault, and F. Bougares. Factored neural machine translation architectures. In *Proceedings of the International Workshop on Spoken Language Translation. IWSLT'16*, Seattle, USA, 2016. URL https://workshop2016.iwslt.org/downloads/IWSLT_2016_paper_2.pdf.

[20] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://www.aclweb.org/anthology/P16-1162.

[21] G. Dinu, P. Mathur, M. Federico, and Y. Al-Onaizan. Training neural machine translation to apply terminology constraints. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3063–3068, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1294. URL https://www.aclweb.org/anthology/P19-1294.

[22] M. Exel, B. Buschbeck, L. Brandt, and S. Doneva. Terminology-constrained neural machine translation at SAP. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 271–280, Lisboa, Portugal, Nov. 2020. European Association for Machine Translation. URL https://www.aclweb.org/anthology/2020.eamt-1.29.

[23] M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. Fikri Aji, N. Bogoychev, A. F. T. Martins, and A. Birch. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P18-4020.

[24] M. Junczys-Dowmunt, K. Heafield, H. Hoang, R. Grundkiewicz, and A. Aue. Marian: Cost-effective high-quality neural machine translation in C++. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 129–135, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2716. URL https://www.aclweb.org/anthology/W18-2716.

[25] R. Sennrich, O. Firat, K. Cho, A. Birch, B. Haddow, J. Hitschler, M. Junczys-Dowmunt, S. Läubli, A. V. Miceli Barone, J. Mokry, and M. Nadejde. Nematus: a toolkit for neural machine translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 65–68, Valencia, Spain, April 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/E17-3017.

[26] F. Hieber, T. Domhan, M. Denkowski, D. Vilar, A. Sokolov, A. Clifton, and M. Post. Sockeye: A toolkit for neural machine translation, 2017. URL http://arxiv.org/abs/1712.05690.

[27] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Van-

couver, Canada, July 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P17-4012.

[28] C. G. Drury and J. Ma. Do language barriers result in aviation maintenance errors? In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 47, pages 46–50, Denver, Colorado, October 2003. doi: doi.org/10.1177/154193120304700110.

[29] C. G. Drury, J. Ma, and C. V. Marin. Language error in aviation maintenance. Technical report, University of Buffalo, August 2005.

[30] P. Wilken and E. Matusov. Novel applications of factored neural machine translation, 2019. URL http://arxiv.org/abs/1910.03912.

[31] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

[32] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

[33] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi: 10.1109/78.650093.

[34] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/gehring17a/gehring17a.pdf.

[35] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings. URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[37] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL https://arxiv.org/pdf/1607.06450.pdf.

[38] P. Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, Feb. 1994. ISSN 0898-9788.

[39] O. Press and L. Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics:*

Volume 2, Short Papers, pages 157–163, Valencia, Spain, Apr. 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E17-2025.

[40] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://www.aclweb.org/anthology/P02-1040.

[41] R. Rei, C. Stewart, A. C. Farinha, and A. Lavie. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online, Nov. 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-main.213.

[42] M. Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, Oct. 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-6319.

[43] Y. J. Kim, M. Junczys-Dowmunt, H. Hassan, A. Fikri Aji, K. Heafield, R. Grundkiewicz, and N. Bogoychev. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5632. URL https://www.aclweb.org/anthology/D19-5632.

[44] A. V. Miceli Barone, J. Helcl, R. Sennrich, B. Haddow, and A. Birch. Deep architectures for neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 99–107, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4710. URL https://www.aclweb.org/anthology/W17-4710.

[45] A. Alexandrescu and K. Kirchhoff. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 1–4, New York City, USA, June 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N06-2001.

[46] Y. Wu, H. Yamamoto, X. Lu, S. Matsuda, C. Hori, and H. Kashioka. Factored recurrent neural network language model in TED lecture transcription. In *2012 International Workshop on Spoken Language Translation, IWSLT 2012, Hong Kong, December 6-7, 2012*, pages 222–228. ISCA, 2012. URL http://www.isca-speech.org/archive/iwslt_12/sltc_222.html.

[47] O. Bojar. English-to-Czech factored machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 232–239, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W07-0735.

[48] M. Post and D. Vilar. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of*

the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 1314–1324, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1119. URL https://www.aclweb.org/anthology/N18-1119.

[49] O. Bojar, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, P. Koehn, and C. Monz. Findings of the 2018 conference on machine translation (wmt18). In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 272–307, Belgium, Brussels, October 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W18-6401.

[50] P. Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand, 2005. AAMT. URL http://mt-archive.info/MTS-2005-Koehn.pdf.

[51] O. Bojar, C. Buck, C. Callison-Burch, C. Federmann, B. Haddow, P. Koehn, C. Monz, M. Post, R. Soricut, and L. Specia. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 1–44, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-2201.

[52] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, S. Huang, M. Huck, P. Koehn, Q. Liu, V. Logacheva, C. Monz, M. Negri, M. Post, R. Rubino, L. Specia, and M. Turchi. Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 169–214, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W17-4717.

[53] R. Steinberger, A. Eisele, S. Klocek, S. Pilos, and P. Schlüter. DGT-TM: A freely available translation memory in 22 languages. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 454–459, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/814_Paper.pdf.

[54] R. Steinberger, B. Pouliquen, A. Widiger, C. Ignat, T. Erjavec, D. Tufiş, and D. Varga. The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2006/pdf/340_pdf.pdf.

[55] J. Tiedemann. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

[56] R. Rozis and R. Skadiņš. Tilde MODEL - multilingual open data for EU languages. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 263–265, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W17-0235.

[57] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, M. Huck, A. Jimeno Yepes, P. Koehn, V. Logacheva, C. Monz, M. Negri, A. Neveol, M. Neves, M. Popel, M. Post, R. Rubino, C. Scarton, L. Specia, M. Turchi, K. Verspoor, and M. Zampieri. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W16/W16-2301.

[58] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P07-2045.

[59] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[60] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/pdf/1412.6980.pdf.

[61] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

[62] C. Malaviya, S. Wu, and R. Cotterell. A simple joint model for improved contextual neural lemmatization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1517–1528, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1155. URL https://www.aclweb.org/anthology/N19-1155.

[63] J. Nivre, Ž. Agić, L. Ahrenberg, M. J. Aranzabe, M. Asahara, A. Atutxa, M. Ballesteros, J. Bauer, K. Bengoetxea, R. A. Bhat, E. Bick, C. Bosco, G. Bouma, S. Bowman, M. Candito, G. Cebiroğlu Eryiğit, G. G. A. Celano, F. Chalub, J. Choi, Ç. Çöltekin, M. Connor, E. Davidson, M.-C. de Marneffe, V. de Paiva, A. Diaz de Ilarraza, K. Dobrovoljc, T. Dozat, K. Droganova, P. Dwivedi, M. Eli, T. Erjavec, R. Farkas, J. Foster, C. Freitas, K. Gajdošová, D. Galbraith, M. Garcia, F. Ginter, I. Goenaga, K. Gojenola, M. Gökırmak, Y. Goldberg, X. Gómez Guinovart, B. Gonzáles Saavedra, M. Grioni, N. Grūzītis, B. Guillaume, N. Habash, J. Hajič, L. Hà Mỹ, D. Haug, B. Hladká,

P. Hohle, R. Ion, E. Irimia, A. Johannsen, F. Jørgensen, H. Kaşıkara, H. Kanayama, J. Kanerva, N. Kotsyba, S. Krek, V. Laippala, P. Lê H`ông, A. Lenci, N. Ljubešić, O. Lyashevskaya, T. Lynn, A. Makazhanov, C. Manning, C. Mărănduc, D. Mareček, H. Martínez Alonso, A. Martins, J. Mašek, Y. Matsumoto, R. McDonald, A. Missilä, V. Mititelu, Y. Miyao, S. Montemagni, A. More, S. Mori, B. Moskalevskyi, K. Muischnek, N. Mustafina, K. Müürisep, L. Nguy˜ên Thị, H. Nguy˜ên Thị Minh, V. Nikolaev, H. Nurmi, S. Ojala, P. Osenova, L. Øvrelid, E. Pascual, M. Passarotti, C.-A. Perez, G. Perrier, S. Petrov, J. Piitulainen, B. Plank, M. Popel, L. Pretkalniņa, P. Prokopidis, T. Puolakainen, S. Pyysalo, A. Rademaker, L. Ramasamy, L. Real, L. Rituma, R. Rosa, S. Saleh, M. Sanguinetti, B. Saulīte, S. Schuster, D. Seddah, W. Seeker, M. Seraji, L. Shakurova, M. Shen, D. Sichinava, N. Silveira, M. Simi, R. Simionescu, K. Simkó, M. Šimková, K. Simov, A. Smith, A. Suhr, U. Sulubacak, Z. Szántó, D. Taji, T. Tanaka, R. Tsarfaty, F. Tyers, S. Uematsu, L. Uria, G. van Noord, V. Varga, V. Vincze, J. N. Washington, Z. Žabokrtský, A. Zeldes, D. Zeman, and H. Zhu. Universal dependencies 2.0, 2017. URL `http://hdl.handle.net/11234/1-1983`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[64] J. Sylak-Glassman. The composition and use of the universal morphological feature schema (unimorph schema). *Johns Hopkins University*, 2016. URL `https://unimorph.github.io/doc/unimorph-schema.pdf`.

[65] B. Peters and A. F. T. Martins. IT–IST at the SIGMORPHON 2019 shared task: Sparse two-headed models for inflection. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 50–56, Florence, Italy, Aug. 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4207. URL `https://www.aclweb.org/anthology/W19-4207`.