Universitat Politècnica de Catalunya
BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

Final degree project

Bachelor's degree in Informatics Engineering
(Computer Science Specialization)
Grau en Enginyeria Informàtica (Especialitat en Computació)

Neural Machine Translation and Linked Data

Author:
Jordi Armengol Estapé

Supervisor (Director):
Marta Ruiz Costa-Jussà
Signal Theory and Communications (TSC) Department

Tutor:
Lluís Padró Cirera
Computer Science (CS) Departament

3 July 2019

Abstract

Machine translation is the task of automatically translating from one language into another. By linked data, we mean structured, interlinked data. Currently, machine translation is addressed by means of deep learning techniques. These algorithms have the big drawback of requiring large quantities of data to be trained. BabelNet (an example of linked data) consists of concepts and named entities connected in 271 languages. This project, based on previous emergent works, wants to exploit BabelNet in the latest Neural Machine Translation systems. In particular, this work proposes extracting concepts from BabelNet and using them alongside words in order to improve machine translation, particularly in low-resource settings. Classical linguistic features will be tested as well.

Resum

La traducció automàtica és la tasca de traduir automàticament d'un idioma a un altre. Per dades enllaçades entenem aquelles produïdes de manera estructurada i entrellaçada. Actualment, la traducció automàtica es porta a terme mitjançant tècniques d'aprenentatge profund. Aquests algorismes tenen el desavantatge de requerir grans quantitats de dades per entrenar-se. BabelNet (un exemple de dades enllaçades) consisteix en conceptes i entitats connectades en 271 idiomes. Aquest projecte, basat en publicacions recents, vol explotar BabelNet en els últims sistemes de traducció automàtica neuronal. En particular, aquest treball proposa d'extreure conceptes de BabelNet i d'utilitzar-los juntament amb les paraules amb l'objectiu de millorar la traducció, particularment en casos de tasques amb pocs recursos. També es provaran característiques lingüístiques clàssiques.

Resumen

La traducción automática es la tarea de traducir automáticamente de un idioma a otro. Por datos enlazados entendemos aquellos producidos de manera estructurada y entrelazada. Actualmente, la traducción automática se lleva a cabo mediante técnicas de aprendizaje profundo. Estos algoritmos tienen la desventaja de requerir grandes cantidades de datos para ser entrenados. BabelNet (un ejemplo de datos enlazados) consiste en conceptos y entidades conectadas en 271 idiomas. Este proyecto, basado en publicaciones recientes, quiere explotar BabelNet en los últimos sistemas de traducción automática neuronal. En particular, este trabajo propone extraer conceptos de BabelNet y utilizarlos junto con las palabras con el objetivo de mejorar la traducción, particularmente en casos de tareas con pocos recursos. También se probarán características lingüísticas clásicas.

Acknowledgements

I would like to thank Marta Ruiz Costa-Jussà, a top deep learning researcher and the supervisor of this project, for giving me the opportunity of doing a research project in a field that interesting as neural machine translation, particularly with state-of-the-art techniques. I am very thankful to Lluís Padró Cirera for being the tutor of my project. I would like to thank Carlos Escolano Peinado as well, for suggesting many ideas, helping me with debugging the code and attending to the meetings with Marta in order to give me feedback. I am grateful for the suggestions that Mercedes García Martínez and Noe Casas did to me as well. Presenting the preliminary results of this work to the UPC machine translation group for receiving feedback was a unique opportunity.

The experiments of this project have been executed in the UPC-TSC department clusters. Without them, it would not have been possible to develop this work.

Finally but not least, I dedicate this thesis to my parents and my brother for their unconditional support.

List of Figures

List of Tables

Contents

Part I

Introduction and thesis management

This project consists in building a machine translation system by combining two different technologies, neural networks and linked data, and investigating whether this proposal effectively improves the current approaches and, if so, to what extent.

In Part I, we will start by introducing the topics of machine translation and linked data and describing the state of the art as well as specifying the stakeholders of the project. We will be particularly keen on the Transformer, the neural architecture we will base our work on, and the existing proposals on how to incorporate certain features alongside words in order to improve the translations. We will see that although other architectures have successfully taken advantage of linguistic features, they are just starting to be used with the Transformer. On the other hand, semantic features extracted from BabelNet, a linked data database, have only been used in machine translation in one recent work.

Later on, we will more precisely formulate the problem and outline elements related to the management of this thesis, particularly the temporal and economic planning of the different stages of this project. We will see that the problem is relevant because machine translation is daily used by millions of people, while data for building these systems is scarce and our proposal could improve translations, particularly in low-resource settings. Finally, we will outline the environmental aspects of this project and the highlights of the follow-up meeting, in which we essentially analyzed the state of the project at that point, including the deviations from the original planning and the considered alternatives.

Once the project will have been fully introduced, in Part II we will see our contributions for trying to solve the problem, the results obtained and our conclusions.

1 Context

This section introduces the topic by giving the required context to understand it, fully specifying the stakeholders of the project and describing the state of the art in machine translation.

1.1 Introduction

1.1.1 Machine learning and Natural Language Processing

Machine learning [1] is a sub-field of artificial intelligence that consists in letting the algorithm learn a function from data instead of manually defining rules for the goal we want to accomplish. In *supervised learning* we have annotated data (for each data point we have the desired output), while in *unsupervised learning* we do not. The dataset is usually *re-sampled* in two independent subsets:

- Learn set: It is used for *training* (that is, learning the parameters) of our *models* and choosing the best one. It is further divided into training and validation. The former is

used for adjusting the model, while the latter is used for choosing the best model, that is to say, selecting the best *hyperparameters* and architectures among the ones which have been tried. By hyperparameter, we mean a parameter which cannot be learned by the machine learning algorithm because it works by assuming a constant value for it.

- Test set: It is used for giving an honest estimation of the generalization error, which is the error that commits the model with new data (data that have not been seen during the learning phase). The test set should be locked until the final model is chosen. Otherwise, the estimation of the generalization error would not be fair.

The *loss* or error function will depend on the task. Since the error we care about is the generalization error, the concepts of *underfitting* and *overfitting* are central to the machine learning practitioner. On the one hand, a model is said to underfit if it is too simple to grasp the complexity of the patterns of the real distribution of the data. On the other hand, a model is said to overfit if it is detecting random patterns present in the data as if they were patterns of the real distribution of the data. Simplicity is a desirable feature of a machine learning model.

Usually, before applying a machine learning algorithm, the data must be preprocessed for imputing missing values, data cleaning, feature selection, and scaling or normalization, among other procedures.

As far as Natural Language Processing (NLP) [2] is concerned, dealing with natural text is a huge challenge because we are keen on semantic features and the human languages are complex. Although the first NLP practitioners (and some of the current ones) tried to explicitly design rules for their tasks, nowadays the dominant approach is applying machine learning techniques, while statistical methods are still in use in some cases. Preprocessing is particularly important. Assuming a clean and well-formatted dataset, the text must be split into sentences and *tokenized* (a token can be thought as a word or a punctuation mark, as seen in figure 1). Words must be encoded in such a way that they can be treated by a machine learning algorithm, which usually will involve numeric vectors. In NLP, datasets are referred as corpora.

1.1.2 Deep learning and Machine Translation

Machine Translation (MT) [3] is the task of automatically translating from the *source* language to the *target* language, so if we refer to natural languages it is a sub-field of NLP.

Early MT systems involved the use of handcrafted, deterministic rules (for instance, with the means of formal grammars), which required domain experts (translators, linguists). The main problem of this approach is that it is difficult to scale because all the possible cases have to be explicitly taken into account explicitly. Later on, statistical translation dominated the field. In recent years, the field mainly pivoted to machine learning. One important characteristic of MT that makes it a more difficult challenge compared to other machine learning tasks is the fact that the function which we want to learn does not have a one-to-one nor many-to-one mapping (e.g. classification), but one-to-many, since one sentence in the source language could be translated into the target language by using many different combinations of words, which leads to two relevant implications:

- The translation function we want to learn is modeled as a *conditional probability* $p(y|x)$, not as a deterministic function.

- It is not a good idea to score the translation using the plain error with respect to the *ground truth* (the real translation), because two completely different sentences could be both correct. More sophisticated scoring metrics, such as the Bilingual Evaluation Understudy Score (BLEU) [4] are used instead. Although BLEU is based on precision, a well-known metric (the number of true positives divided by the sum of true positives and false positives), it can take into account more than one translation as a reference. BLEU computes the precision of the N-grams (a sequence of size N, in this case for sizes 1 to 4) overlap between the reference and the output of the MT system, and it has a brevity penalty for avoiding short translations. It is defined follows:

$$\text{BLEU} = \min(1, \frac{\text{output length}}{\text{reference length}}) \times \prod_{i=1}^{4} \text{precision}_i{}^{\frac{1}{4}}$$

In order to train a statistical MT system, a dataset $\mathbf{D}$ with N pairs $(\mathbf{x_i}, \mathbf{y_i})$ of sentences in both source and target languages is required. Otherwise, it is difficult to train the system, although some recent works investigate unsupervised methods (that is to say, methods which do not use the said pairs) for MT. In frequentist statistics, *log-likelihood* is used to describe the plausibility of the parameters of the statistical model given our data with better mathematical conditioning. Statistical MT makes use of this concept to score a machine translation model:

$$\ell(\boldsymbol{\theta}, \mathbf{D}) = \sum_{(\mathbf{x_i}, \mathbf{y_i} \in \mathbf{D})} \ln p(\mathbf{y_i} | \mathbf{x_i}, \boldsymbol{\theta})$$

Our purpose is to perform a maximum likelihood estimation, which is a usual procedure in machine learning. Nevertheless, the problem of modeling $p(\mathbf{y_i}|\mathbf{x_i}, \boldsymbol{\theta})$ remains. The Watson Research Center suggested approximating the logarithm of the true probability distribution with a linear combination of many features [5]. Although more advanced models have been proposed, they all required manual feature engineering, which has proven to be a difficult task. Artificial neural networks, a very popular machine learning algorithm, are useful for letting the machine automatically figure out the relevant features for a dataset and can be used as feature extractors by feeding the statistical MT with the output of the network. However, the dominant approach (and the one taken in this project), named Neural Machine Translation (NMT) [6], is to get rid of the statistical model and train the neural network to perform an end-to-end translation, that is, to let the network model the probability distribution on its own.

Neural networks [7] consist of neurons, which are based on the *perceptron* algorithm. A neuron receives a set of values and outputs a linear combination of them. The parameters for each value, known as *weights*, and the independent term, known as *bias*, can be learned, as in a classical linear regression. If we stack multiple layers of neurons such that the outputs of the neurons of one layer are input to the neurons of the next layer and make use of non-linear functions (known as *activation* functions) at the end of each layer, the model is no longer linear and can express complex functions. This model is known as the MultiLayer Perceptron (MLP). In a feed-forward network the *computational graph*[1] of the network goes forward (i.e. it has no *cycles*). Algorithms for efficiently training large MLPs have been developed. The main challenge was solving the *credit assignment problem*: if the output is wrong, how can we know which parameters are to blame and correct them accordingly? Since the *loss* function is differentiable,

[1] A computational graph is a graph such that nodes indicate variables or results from other computations and edges indicate operations and it is usually used to formalize neural architectures [7].

the gradient can be computed. Once we know which parameters of the last layer would have been optimal by comparing the ground truth with the output of the network and making use of the derivative, we can apply the chain rule of derivation backwards. This technique is known as *backpropagation*. With the gradient we can update the parameters accordingly. Although there are many different optimizers available, most of them are based on a first-order optimization technique known as *gradient descent*. In Stochastic Gradient Descent (SGD), a *minibatch* of samples is used to descend the gradient. By learning rate we mean how large is the step in each iteration of the gradient descent. We want the optimization to be fast, but at the same time we do not want to make a step that big that a local minimum is missed. Non-convex optimization is a tough problem since we do not have any convergence guarantees. Nevertheless, at least in the case of deep learning, the empirical evidence suggests that the local minima found by the optimizers tend to be good enough. In NLP, the literature agrees that the best optimizer tends to be Adam [8], an SGD-based algorithm with an adaptive learning rate.

Neural networks are prone to overfitting because of their expressive power. Regularization techniques such as *weight decay* (a penalty on the magnitude of the weights) or *dropout* (the random deactivation of a number of neurons in each training iteration) are used to counter this effect.

Deep learning means that we have many *hidden* or intermediate layers in the MLP. Deep learning systems need large quantities of data and the computational costs of training are high, which are major concerns. However, in recent years we have seen the increase in computing resources with GPU-based parallelization, the availability of large amounts of data and the development of algorithms for efficiently training large networks and for adapting them to non-numeric domains.

In NMT, the conditional probability of the target text given the source text is directly learned by the network. Nevertheless, for doing so, a particular kind of neural networks known as Recurrent Neural Networks (RNNs) seems to be needed [9] [10]. In MT, we require context, since in order to have a coherent sentence in the target language the whole sentence in the source language must be taken into account, and sentences do not have a fixed length. Feed-forward networks in principle do not have this capability. RNNs attempt to compress the input sequence into a vector of fixed dimensions by making use of recursion, that is to say, cycles in the computational graph of the network. At each time step t, we have a vector $\mathbf{h}_{t-1}$ which represents the history of the preceding inputs and can be used to model the probabilities of a sequence. The new value for $\mathbf{h}$, $\mathbf{h}_t$, is defined as:

$$\mathbf{h}_t = \phi_{\boldsymbol{\theta}}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

where $\phi_{\boldsymbol{\theta}}$ is an activation function parameterized by $\boldsymbol{\theta}$. It is more complex than the usual activation functions since recurrent weights must be taken into account. However, vanilla RNNs are outperformed by *gated recurrent units*, which have even more sophisticated activation functions:

- Long Short Term Memory units (LSTMs) [11]: LSTMs have a *gate* for deciding what part of the past must be forgotten (*forget* gate), a gate for deciding which new information should be added (*input* gate), a mechanism for updating the state (*memory*) and an *output* gate. LSTMs can be made to be bi-directional (Bi-LSTMS) for taking into account the context in both directions of the sequence.
- Gated Recurrent Units (GRUs) [9]: they are a simplified version of LSTMs that achieves similar results. In GRUs, the forget and the input gates are combined, and the same happens for the memory and output.

With LSTMs and GRUs, *encoder-decoder* architectures, known as Sequence2Sequence architectures [12], can be built, as seen in figure 2. The encoder *projects* the sentences of the source language into an *embedding space* (of a lower dimensionality) and a decoder generates sentences for the target sequence taking the encoder output as input. In other words, we have one RNN for the source language with the goal of understanding the sentences, and another RNN for the target language with the goal of generating the new sentences. Usually, the output layer of the decoder returns the likelihood of each word (provided that it is present in the vocabulary) at each step in the sequence. The *softmax* function[2] is applied to normalize the output into a vector of probabilities, and a search algorithm is needed in order to select the best words given the probability distribution. Beam search [13] outperforms greedy approaches (i.e. selecting the best word at each step) and it is the prevalent algorithm for searching in the decoder. The vanilla decoder can be improved by extracting the n-best translations from an *ensemble* of LSTMs [14].

Nonetheless, both LSTMs and GRUs exhibit similar problems. Letting apart the problem of vanishing gradients (gradients so small that training is difficult), which can be solved by regularizing or using the REctified Linear Unit (RELU) as the activation function, they collapse all the sense of a sentence into a single vector, which may miss the whole meaning of the sentence if the sequence is long. Also, they are not particularly efficient to train, since the recursion involved in RNNs cannot be easily parallelized. The current state of the art methods successfully address these problems, the solutions of which will be described in the corresponding section.

Apart from the need of taking context into account, another point that is needed for making neural networks work with natural next is defining a procedure for transforming text into numeric vectors (an *encoding*), since neural networks can only be fed with the latter. This procedure is also relevant for achieving state of the art results. A very naive manner of doing so is using a *one-hot encoding* (or *one-of-K*) vector, which consists in having a vector with as many elements as possible words, where each element is 1 if it is the one corresponding to the particular word we want to identify and 0 otherwise. Unknown words, or words not frequent enough, can be mapped to a generic unknown token. This approach is not sufficiently convenient for two reasons:

- The representations are that sparse and the number of possible words is that big that the resulting vectors are huge and therefore computational and memory costs are high.
- All words have the same distance between them.

Word *embeddings*, like Word2Vec [15], involve learning a lower dimensional representation, possibly in an unsupervised way. Not only achieve they much smaller vectors improving the efficiency, but remarkably they allow to use distance as a semantic similarity measure, that is, close tokens have similar meanings. A number of new word embeddings algorithms featuring both compactness and semantic distance have been proposed. Word embeddings can be improved taking characters into account when determining the embedding in addition to the tokens themselves. In NMT, in contrast to other areas of NLP, token embeddings are typically learned from scratch instead of pre-loaded, since empirically this approach tends to give better results, although recently some proposals involving pre-trained embeddings in NMT have given good results. A few breakthroughs in word representations have improved the existing machine translation systems, which will be detailed in the state of the art section. The position of word embeddings in an NMT pipeline can be appreciated in figure 1.

[2] $\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$

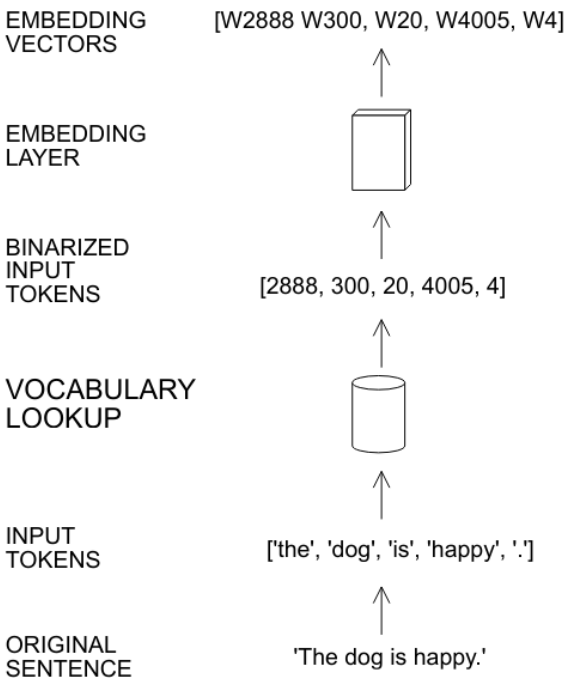

Figure 1: Tokenization and word embeddings in NMT: MT can be modeled as discrete sequence learning. Sentences are parsed into tokens that can correspond to words, usually normalized (eg. without capital letters), and punctuation marks, among others. Each token is assigned a unique number that corresponds to a one-hot encoding vector. In the embedding layer, which is trained alongside the rest of the network and can be seen as a lookup table, each one-hot encoding vector, is projected into a dense, lower-dimensional vector. In algorithms for explicitly learning word embeddings, the embedding layer is used to predict a fragment of the original input. This way, their outputs become vectors such that low distance (eg. Euclidean) means being semantically close. In NMT, although pre-trained embeddings can be used, most systems are capable of implicitly learning efficient representations.

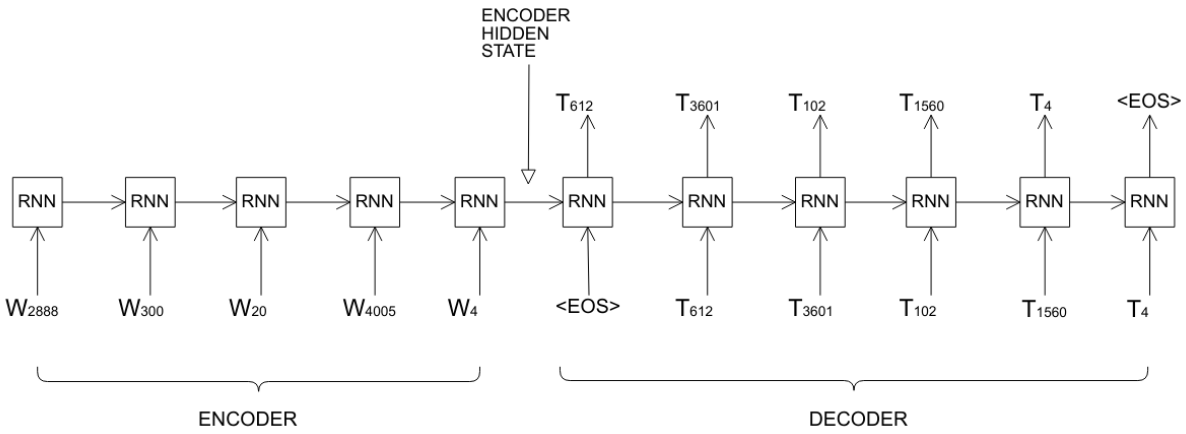

Figure 2: Sequence2Sequence architecture: We have two gated RNNS, usually LSTMs, although GRUs can be used as well, one for the source sequence (encoder) and another one for the target sequence (decoder). The input word embeddings from figure 1 are sequentially input to the encoder. At each time step, the LSTM takes the previous hidden state and the current word embedding and outputs the next hidden state. Hidden states can be seen as historical data compression. Note that this process cannot be run on parallel. The encoder stops predicting when it encounters the `<EOS>` (end of sentence) marker. The decoder takes the encoder output as an input, and starts predicting the target tokens (in the figure, represented with `T`). At each time step, it takes the previous hidden state and the previous predicted token as an input, until it outputs `<EOS>`. Notice that although in the figure we see the two RNNs *unrolled* for each time step, they are actually the same RNNs, respectively.
While the encoder has an input dimensionality equal to the size of the word embeddings, the decoder output dimension is equal to the number of tokens in the target vocabulary. At each time step, the probability for each token is output. An additional algorithm must be executed for converting the probabilities into the actual target sequence. A greedy approach would consist on just selecting the token that has the highest probability at each step, but in practice it is better to run more complex search procedures (because selecting a particular token in one time step may alter the probability distributions for the following steps), typically beam search.
Source: Adapted from [12].

1.1.3 Linked Data

The inventor of the World Wide Web, Tim Berners-Lee, coined the term *linked data* [16], which refers to structured data that is properly linked, ie. forming a *graph*. The idea behind this concept is that computers can easily take advantage of the relations between different documents by making semantic queries, provided they are properly linked. For instance, if instead of having a plain-text document detailing medical concepts we had an open database of interlinked diseases with unique identifiers and a common interface, computers could explore the graph and find new relations between diseases. Linked data can be thought as an *ontology*, which in *knowledge engineering* is a method for representing knowledge as a graph.

The idea of the classical AI researchers in the 80s of building *expert systems* by expanding ontologies did not succeed because it did not scale properly. Berners-Lee's dream of a World Wide Web made of open and linked data did not come true for a variety of reasons. Nevertheless, high quality linked data can be found for different domains. In the case of the general linguistic domain, BabelNet[3] [17] consists of concepts and name entities connected in 271 languages and provides an Application Programming Interface (API).

1.2 Stakeholders

Once the project has been fully contextualized, we identify different groups of stakeholders involved in the project.

1.2.1 Target audience

Since the aim of this project is not to build a product, but to research a new neural architecture, the main target audience of this project is the MT research community.

1.2.2 Users

As stated before, the aim of this project is not to build a product for the market. For instance, it will not have a graphical interface. The intended users are other MT researchers or machine learning engineers.

1.2.3 Beneficiaries

In the short and medium term, the main beneficiaries will be other machine translation researchers. If the results of the new architecture do not outperform the existing ones, researchers will be more informed about whether or not this path is worth pursuing, and why it has not worked as expected. If they do, they will have a program that they will be able to use as the new baseline for doing more research. For instance, the TALP-UPC NLP research group could be benefited from this work.

[3]BabelNet: `https://babelnet.org/about`.

Nevertheless, we can identify other potential beneficiaries in the long term. If a research proposal in deep learning gives good results, at some point it makes its path to the market, so if the new architecture were to be useful, both directly or indirectly, the fragment of the general population which daily uses translation systems would be benefited from this project too. Companies or governments using or building translation systems are benefited from research in this area as well. In particular, speakers of minority languages or speakers with languages which do not have enough NLP resources could be benefited the most, since one of the intended advantages of the new architecture is the promise that it would need less data and perform better *few-shot learning* (learning to translate two language pairs which have almost not been seen in training, having the system been pre-trained with other pairs), although this would be in a long-term scenario and in this project our goal is to start exploring this path.

In addition, there are some relevant actors who must be mentioned and could be academically benefited, particularly the student (or developer), who is the person in charge of this work. The supervisor and tutor are the professors in charge of supervising it, and the project is developed within the scope of the UPC university.

1.3 State of the art

The current state of the art results in MT can be achieved starting from the encoder-decoder architectures we described in the end of the introduction, but some key changes are required for a better performance.

1.3.1 Attention mechanisms

As stated before, the problem of the vanilla Sequence2Sequence architectures we described is that the meaning of the sentences of different sizes is collapsed into a single vector of fixed length, so important information can be missed. Due to the sequential nature of RNNs, the early input symbols can be easily forgotten and long-range dependencies cannot be easily learned. Attention mechanisms can solve this issue, as described in one of the sources used to write this section [18]. Instead of outputting only one vector, now in the encoder we are going to output multiple vectors, depending on the length of the input. The hidden state of the gated unit (or the two hidden states, if we are using a Bi-LSTM) has compressed the history of the sentence up to the word we are located in, so it has information on the context. This hidden state of the encoder, together with the hidden state of the decoder (which contains information on what has already been translated) is input to an auxiliary feed-forward network (the *soft* attention mechanism), which is trained to predict the importance of each word in the source language for translating the next word. The model learns a set of attention weights to estimate the relevance of each word in the source language at each time step. Then it outputs what is known as the *context vector*, $\mathbf{c}$, based on hidden states $\mathbf{s}_1, ..., \mathbf{s}_m$ such that:

$$\mathbf{c}_i = \sum_{j=1}^{m} \mathbf{a}_{ij} \mathbf{s}_j$$

$$\mathbf{a}_i = \text{softmax}(f_{att}(\mathbf{h}_i, \mathbf{s}_j))$$

where the attention function $f_{att}(\mathbf{h}_i)$ computes an alignment score between the current hidden state $\mathbf{h}_i$ and the previous hidden state $\mathbf{s}_j$. In other words, the attention mechanism allows the

decoder to look at the entire sentence and extract the information that is needed at each step. The decoder has access to all the hidden states, and attention gives weights to these states.

There are different attention variants, although the reader should notice that this is no a disjunctive classification:

- Additive attention: The original attention mechanism [19] used a feed-forward network of only one hidden layer to compute attention alignment.

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j) = \mathbf{v}_a^T \tanh(\mathbf{W}_a[\mathbf{h}_i; \mathbf{s}_j])$$

- Multiplicative attention [20]: It is a simplification of the additive attention such that not only is it slightly faster and more space-efficient than the original one, but it allows to obtain a distribution of attention over other features, such as word embeddings. Remarkably this means that multiplicative attention can be used with vanilla feed-forward networks and Convolutional Neural Networks [21] (CNNs, a kind of feed-forward network used mainly for computer vision tasks). That is to say, theoretically now it would be possible to treat sequences without the recursion (cycles) involved in RNNs.

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j) = \mathbf{h}_i^T \mathbf{W}_a \mathbf{s}_j$$

- Self-attention [22]: Without any additional information from the decoder, regarding what we have already translated, attention mechanisms can still extract relevant information by just looking at the hidden states themselves.

$$f_{att}(\mathbf{h}_i) = \mathbf{h}_i^T \mathbf{W}_a \mathbf{s}_j$$

  In fact, self-attention can be either additive or multiplicative.

- Key-value attention [23]: This recent variant uses *query vector* and *key vector* pairs to compute a similarity measure (dot product). Each hidden vector $\mathbf{h}_i$ is split into a key $\mathbf{k}_i$ and a value $\mathbf{v}_i$. Key-value attention can only be multiplicative.

Self-attention and key-value attention are the ones we are concerned the most with and they can be used with feed-forward networks too. Not only have attention mechanisms enabled the improvement of the existing LSTM-based architectures, but they have allowed the use of CNNs, which have less complexity because of the lack of recursion, to deal with sequences [24].

1.3.2 The Transformer

The best existing MT systems are based on the Transformer [25][4]. The Transformer algorithm was proposed in 2017 and apart from completely getting rid of RNNs, it does not even make use of CNNs; it essentially consists of feed-forward layers and attention modules. The researchers found that apart from improving the best results in standard MT datasets, it dramatically improved the computational efficiency of the existing models, thanks to not having the recursion involved in RNNs.

[4] A comprehensive explanation of the paper, used together with the paper itself for writing this section, can be found on [26].

This architecture is based on the encoder-decoder architecture we previously saw. The tokens are not input sequentially as in RNN-based architectures, but all at once. Firstly, the source and the target words are input to the respective embedding layers of the encoder and the decoder, as in the Sequence2Sequence architecture, but with one important difference. The Transformer needs a particular kind of embeddings, named *positional* embeddings, which are summed to the input word embeddings. Since the network is not recurrent, it cannot naturally deal with the ordering of a sentence. Therefore, for each input token, its position must be encoded in the embedding itself. The authors of the Transformer original paper experimented with both learned (ie. with trainable parameters) positional embeddings and sinusoidal positional embeddings, but the former worked better:

$$PE[\text{pos}, 2i] = sin(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}})$$

Both the encoder and the decoder are composed of a stack of N identical layers. In the case of the encoder, each layer has two sub-layers, a *multi-head* self-attention mechanism and a standard fully-connected layer. By multi-head attention we mean that the attention mechanisms of the Transformer use multiple attention distributions and outputs for a single input, which are concatenated, as in figure 3. This allows to capture complex aspects of the source sentence. For instance, if a verb refers to the subject of the sentence, the multi-head attention allows to attend both to the subject and to a pronoun referring to that subject when working on this verb.

The decoder is built in a similar manner, but it has 3 sub-layers. The additional sub-layer is a multi-head attention for the output of the encoder. This attention mechanism is not self-attention, but encoder-decoder attention as in the Sequence2Sequence architecture with attention, for attending to the source with respect to the target. Another difference is that the attention in the decoder is prevented from attending positions where we have not arrived yet. This fact, together with the offset by one position in the target tokens, guarantees that the prediction for one position can only depend on the previous positions. If this procedure was not applied, the Transformer would be useless because the network would simply learn to repeat the target sentence. At the end of the decoder, softmax and a search procedure are applied as in the case of the Sequence2Sequence architecture. For an overview of the Transformer, see figure 4.

Both multi-head and feed-forward sub-layers have residual connections [27], that is to say, connections to skip to the following layer, which in this case allow to retain the information related to the position across the different layers of the network. Both sub-layers are followed by an *add and norm* block that performs layer normalization [28], which consists in normalizing the outputs of the corresponding layer with respect to the mean and variance from the summed inputs to the layer from a single training case.

As we said, vanilla RNNs have problems modeling long term dependencies. Whilst the attention mechanisms before the Transformer solved the dependency between inputs and outputs (encoder and decoder), the architecture introduced by the Transformer extended this idea to the modeling of dependencies between the different parts of the source language sentences and the different parts of the target language sentences. Whereas RNNs *read* from left to right (and from right to left if we use Bi-LSTMs), the Transformer allows the encoder and the decoder to see the entire input sequence all at once. The Transformer uses a key-value attention, but obviously not for the hidden states (since a feed-forward network does not have the LSTM hidden states). While in the encoder the embeddings of the tokens in the source language are used for the keys, values and queries, in the decoder the outputs of the encoder are used for keys and values and the

embeddings of the tokens in the target sentence are used for the queries. A dot product between the query and the key is computed. Everything can be expressed just as matrix multiplications, in contrast to RNNs.

The researchers in [25] chose Adam as the optimization algorithm, with $\beta_1 = 0.9$ and $\beta_2 = 0.98$, and set dropout to 0.1 for regularization. The Transformer obtained a BLEU score of 28.4 on the WMT 2014 English-to-German translation task, improving the existing best results by over 2 BLEU points and by using a fraction of the computational resources used by other systems.

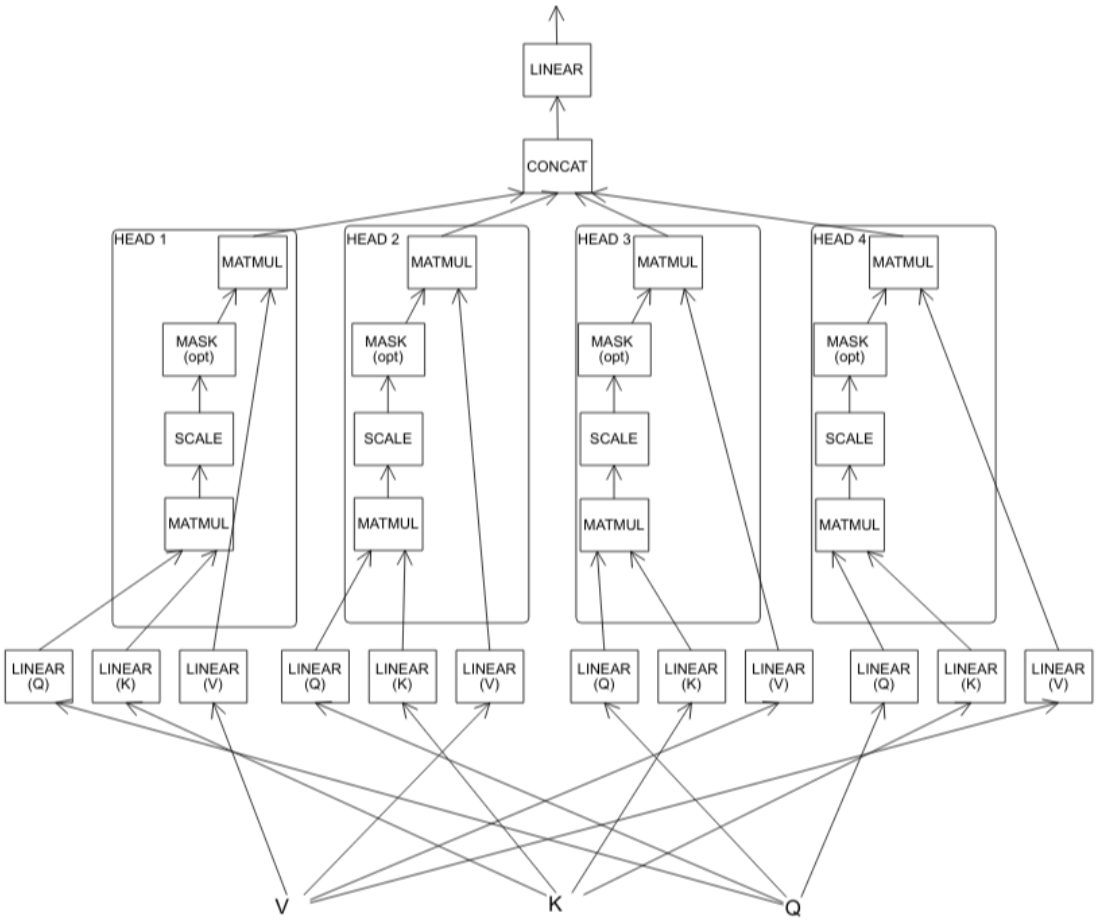

Figure 3: Multi-head attention: Scaled dot-product attention is computed from **V** (*values*), **K** (*keys*) and **Q** (*queries*). In this example, we have a multi-head attention mechanism with four heads. Each head is a scaled dot-product attention mechanism, and they all run in parallel. The motivation for having multiple heads is that each one can specialize to focus on different fragments from sequences, so the Transformer can attend to multiple aspects simultaneously. Source: Adapted from figure 2 in [25].

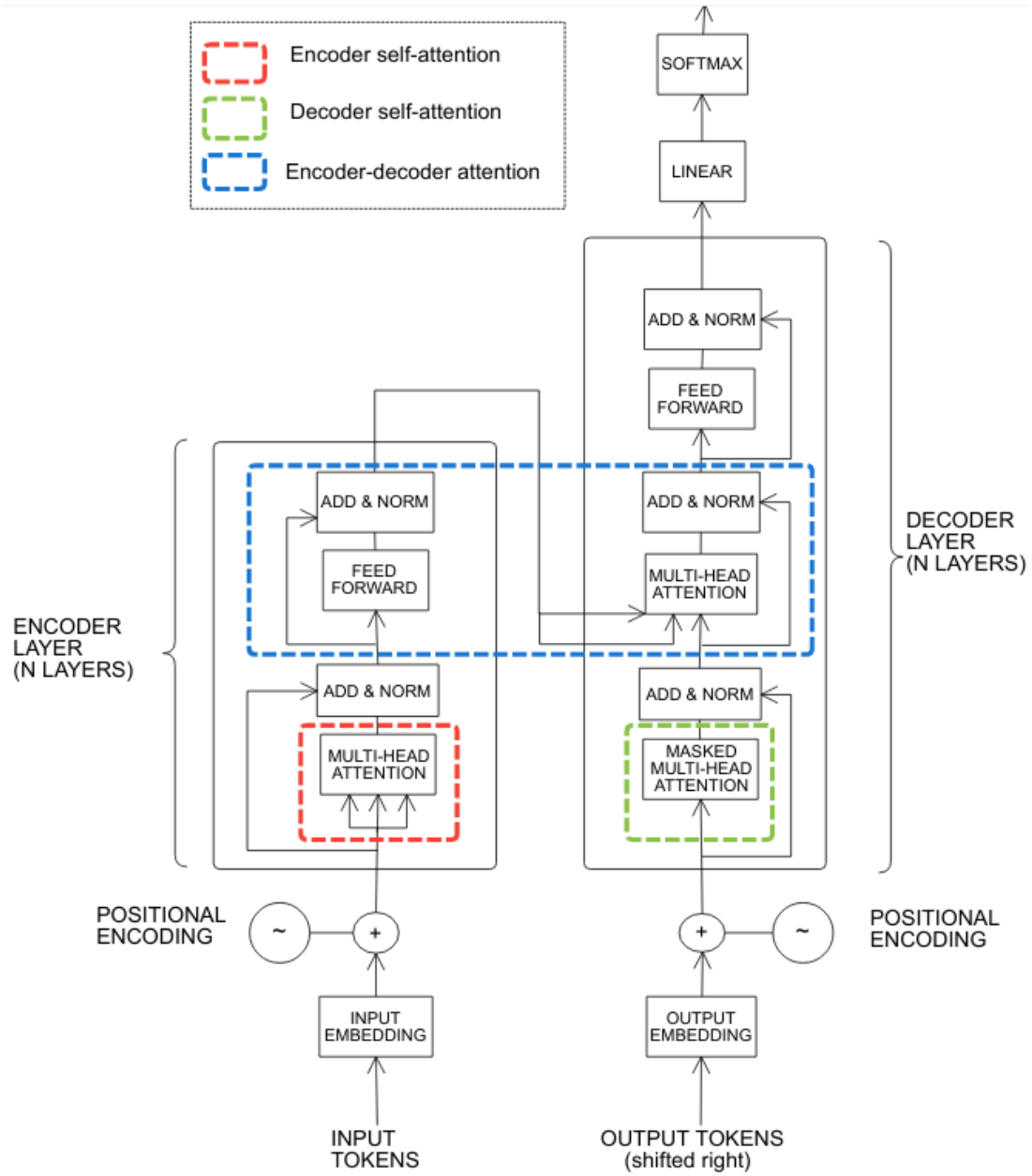

Figure 4: Transformer architecture: Both input and right-shifted target tokens are input to their respective embedding layers. Both embeddings are summed to the positional encoding in order to allow the Transformer to be aware of the positions of the tokens in the sentence. The encoder consists of a stack of N layers with two sub-layers each, namely a multi-head self-attention mechanism and a feed-forward network, being both layer-normalized and having residual connections (connections skipping to the next layers). The decoder is similar, but the multi-head attention mechanism is masked, which together with the right-shift of the output tokens prevents the decoder from looking at future positions (which would cause the decoder to learn to repeat the sequence). There is an additional encoder-decoder attention for attending to the target with respect to the source. Source: Based on figure 1 in [25].

1.3.3 Subwords and BPE

It has been empirically shown that NMT with *subwords* outperforms NMT with word-level tokenization [29]. By subwords, we mean the splitting of words in different segments as a pre-processing step, which can be done in an unsupervised manner by detecting co-occurrences and counting frequencies. Intuitively, linguists know that words actually consist of different parts (like the root, the prefix or the suffix) and neural networks can take advantage of a more granular splitting. Another problem addressed by subword based representations is the fact that most MT systems use a fixed vocabulary, but MT is an open-vocabulary task, that is to say, words unseen (or seen only a few times) in training can appear when translating new sentences. In addition, there are rare words whose translations tend to be transparent, like named entities (eg. a surname) and morphologically complex words. By transparent, we mean that the translation is simple or even leaves the word unchanged.

The particular technique we are keen on is Byte Pair Encoding (BPE) [30], a compression technique that when adapted to MT enhances the translation of rare or unknown words by making use of subword units [29]. The original BPE technique is a data compression algorithm that works by replacing the most frequent pair of bytes in a sequence with an unused byte. In MT, instead of merging frequent bytes, characters or character sequences are merged. For example, the word `'football'` could be encoded as `'foot@@ ball'`, (where `'@@'` is the BPE symbol), because depending on the context the word `'football'` is relatively uncommon while both `'foot'` and `'ball'` can appear more frequently.

BPE is learned from the training data, and then the same model is applied to both validation and test sets. When generating new translations, it is trivial to recover the actual words because it is a simple, deterministic step consisting in replacing `'@@ '` by `''`.

1.3.4 Factored Neural Machine Translation

By Factored Neural Machine Translation, we mean NMT systems that instead of just using the plain words as input are capable of working with factored representations of them, which should not to be confused with subwords (subwords are actual *sub-strings* of a word, whereas factors are data about the word). For instance, one may feed a neural network with both the words and the Part-of-Speech (PoS) tags (eg. `'verb'`) of each word. It has been shown that this additional information can ease the task of the network, although it might be not always the case.

The work which coined the term [31] used lemmas and PoS tags, tense, person, gender and number, which can be obtained by running a morphological analyser on the corpora. They showed that factored NMT allowed to deal with larger vocabularies, to decrease the Out-Of-Vocabulary (OOV) rates, to produce words not present in the vocabulary and to decrease training time, although performance did only increase up to 2% BLEU points in experiments with English to French pairs. The work extended the standard Sequence2Sequence with LSTMs and attention to allow generating multiple output symbols (lemmas and other factors) at the same time, while feeding the RNN with only the lemmas. The beam search procedure had to be adapted. In this case, the decoder was the focus of the work.

On other hand, another work showed that linguistic features (the same ones used by the previous work, in addition to *dependency labels*) improve NMT, without modifying the decoder [32], but the encoder. In this case, each feature had its own embedding layer, and the embedding vectors of

words themselves and their respective features were concatenated. BPE preprocessing was used, and apart from linguistic features, an additional feature, *subword tags*, was introduced to encode the position of each subword (and the features of their respective word), while linguistic features were repeated for each subword. They achieved improvements between 0.6 and 1.5 BLEU points depending on the particular dataset they used. In their ablation study, it was shown that lemmas were the most useful feature. Interestingly, they motivated working on word features by stating that although the improvement of NMT techniques might make these linguistic features less useful, using features can still be key for low-resource tasks and languages with high morphological variations. It is worth noting that this work was based on the Sequence2Sequence architecture as well. In section 8.1, we will see linguistic features and ways to make them compatible with BPE with more detail.

Some NMT frameworks, such as OpenNMT[5] or Nematus [34] already provide some implementations. In both cases, features are separated with `'|'` in the source language file, and each feature has its own embedding layer. In section 7 we will revisit this matter with more detail.

1.3.5 Linked Data

A recent work used both PoS tags and the identifiers of the BabelNet *synsets* (concepts) as factors, which enabled moderate improvements for languages seen in training and increased more than 3 points the BLEU scores for languages not seen in training [35]. In this case they also used the standard Sequence2Sequence with LSTMs and attention as well as subwords. They present two ways of encoding the synsets:

- Adding synsets as a factor, which enables moderate improvements for known languages and increases more than 3 points the BLEU scores for languages not seen in training. This approach is the one we are more keen on for this work.
- Ignoring the original words and replacing them by the synsets and PoS tags. This approach would be more interesting for monolingual training, which is out of the scope. However, we could use features such as PoS tags when synsets are not defined (see section 8.2).

1.3.6 Conclusions on the state of the art

Attention mechanisms have been a fundamental breakthrough in MT. In particular, the Transformer, which is a neural architecture composed of feed-forward layers and attention modules (without recursion), seems to be the best available architecture, outperforming LSTMs and CNNs in both results and efficiency. Using subwords tends to improve the results as well. Using features as a source of additional information should ease the task of the network and even decrease the amount of required data. However, as far as we know, factors are just starting to be used with the Transformer[6], and using BabelNet as a source of additional information in factored NMT has only been proposed in the recent work we cited. In addition, instead of just concatenating the outputs of the embedding layers, we will explore alternative architectures, and we will be focused on low-resource settings.

[5] OpenNMT: `http://opennmt.net`. See [33].

[6] We are aware of some groups working on introducing factors in the Transformer, but as far as we are concerned there are no publications available.

2 Project scope

2.1 Formulation of the problem, justification and goals

The problem we are formulating is the following: **can we improve the existing state of the art NMT systems by implementing factors on the top of the Transformer architecture and then using the concepts of BabelNet's linked data as factors?** The justification for selecting this problem is that it follows from the state of the art that this is the logical next step for factored NMT, and a promising one. MT is used on a daily basis by millions of people, companies and governments, and training time and the difficulty of obtaining data are major concerns, so the problem is relevant. Based on the formulation of the problem, the objectives for this project are clear:

- To take the Transformer (ideally using BPE as subword preprocessing) as the baseline and to reproduce the results.
- To implement factors in the Transformer architecture.
- To develop a preprocessing script to add the BabelNet synsets as factors. Classical linguistic features should be tested as well.
- To compare fairly the baseline with the proposed architectures by conducting the necessary experiments to objectively and rigorously determine whether the new architecture outperforms the existing system, and if so to what extent and in which cases.
- To properly document the work.

2.2 Scope definition

The field of MT is huge, and so are the possibilities for each planned project. Therefore, it is particularly important in this case to define the scope of the project. The project will be based on the problem formulation and objectives stated before.

For *debugging* the proposed architecture and comparing it with the baseline a single, well-known, relatively small dataset will be used, and for only one language pair (the details will be listed later on). For extensively testing the new architecture it would be interesting to try different languages pairs and different datasets, but training a deep learning model with only one dataset can already be too time-consuming for a bachelor project. The fact that the dataset is small will allow faster development cycles. Since it was used in a workshop, the reference scores are known. In addition, from the state of the art section we got the intuition that factored NMT should be particularly useful in low-resource settings. Monolingual (unsupervised) or multimodal translation (e.g. using both text and images) will not be explored.

In machine learning we usually search the best hyperparameters. In this case, we will not, since the default ones coming from the original Transformer paper and the published works on the dataset of choice are considered to be good enough and again, apart from being extremely time-consuming and computing-intensive, it is not related with the goals of this project. In addition, the mentioned baseline configuration will be taken as a whole and we will not try

different configurations for it. Recent variants to the vanilla Transformer, like the Universal Transformer [36], will not be considered.

As stated before, this is a research project, not a product intended for end users, so usability will not be a concern. The program will probably require some sort of programming and computer science knowledge to be used. Making the code as readable and documented as possible could be considered to be within the scope of the project, since we would like this architecture to be used by other researchers or engineers, but at the same time we must take into account that the code is intended for fast prototyping of new neural architectures and configurations and we will be constantly iterating over different ideas. Moreover, it will not be considered to be ready for production.

2.2.1 Possible obstacles

Different difficulties can arise with the development of this project and it is adequate to try to enumerate them in advance if we want to be able to overcome them. On the one hand, we can identify the following possible generic obstacles:

- Personal problems: Only one person is in charge of the project. If the developer of this project were to have a certain illness or another personal problem, the deadlines of the different deliveries of the project would be compromised.
- Scheduling: Some tasks can be more difficult and take more time than expected.
- Some APIs, libraries or research papers might not be well documented: When trying to program a particular feature using a particular API or trying to reproduce the steps followed in a research paper, the documentation is not extensive enough for our needs or similar problems.
- Hardware, local software or online software service failure: The PC falls apart, the installed Linux system breaks, an online service in which we rely on, such as Github, has a shortage or similar problems.
- Bugs: The code has errors and related problems. This problem can get worse if the bugs cannot be detected early enough.

On the other hand, we can identify some possible obstacles specifically related with the field of MT and deep learning:

- Not enough computational power: Although we have access to the GPU clusters of the TSC-UPC, it could happen that it was not enough for our needs, or that the waiting time for entering the execution queues was too long. In that case the development cycle would be slow (it would take too much time to get feedback) or some experiments would not be finished in time.
- Incompatible technologies: The proposed combination of technologies might end up not being compatible enough, or it might be very tough to make them work together.

2.3 Methodology and rigor

Since the schedule for this project is tight and the goals are ambitious, an agile methodology seems to be the best approach. Nevertheless, the aspects of agile methodologies that are related to team work do not apply, since this is an individual project. In particular, the development cycles will be short in order to get early feedback. Instead of following the traditional software engineering practices, the code will be developed iteratively. Moreover, an intensive client (in this case, the supervisor of the project) feedback will be key for ensuring that the project is going in the right direction.

The particular experimental settings will be detailed in the corresponding section, but it is important to notice that the best practices of rigorous research in machine learning will be followed. In order to have a fair comparison of the different architectures, they will have access to exactly the same learning set, and the test set will not be used for building the model.

2.3.1 Tools

Development tools and resources

The programming language of choice is Python 3, since apart from being open source and having and extensive support for many libraries, it is particularly well suited for deep learning and it is the most widely used language in NLP as well. The project will use PyTorch [37], an open-source deep learning library for Python developed by Facebook, and Fairseq [38], a PyTorch module for NLP. PyTorch leverages CUDA, a GPU-based computing platform developed by NVIDIA. In addition, Git, one of the most popular version-control systems, will be used for tracking changes in the source code. The git repository wil be stored in Github. The Unix environment and command line utilities and text editors will be extensively used. Additional tools will be detailed in the planning section.

With regard to resources, for running the models we will have access to the GPU clusters of the TSC-UPC department (with the Slurm Workload Manager), so we should have enough computational power. The dataset of choice is the IWSLT 2014 corpus (English to German) [39]. The linked data that will be used is BabelNet.

Monitoring tools

The Git repository will be hosted on Github and shared with the supervisor of the project. Thus, the development will be easily monitored thanks to the different tools offered by Github. Apart from that, daily emails and weekly or biweekly meetings will be held with the supervisor.

2.3.2 Validation

During the development of the project, the different milestones will be validated by making use of the monitoring tools detailed before. In the final stages, the project will be validated by comparing the results with other research projects and reference results, since the task and the dataset, as well of the results of introducing factors (in other architectures), are well-known in the MT community.

3 Planning

Planning is vital in order to guarantee that the project is successfully and timely delivered.

3.1 Estimated project duration and considerations

The project officially begins on Monday 12^{th} February, 2019, when the academic term starts, and the deadline is set as Monday 24^{th} June, 2019, a week before the presentation, since the thesis have to be submitted in advance. Thus, the duration is estimated as approximately 4 months. In case of a severe eventuality, it would be rescheduled and it would end in October, which would almost double the duration. The developer of this project will be focused on the project but it will not imply a full-time dedication, because of the enrollment to two other courses.

3.2 Resources

Although some of the resources have already been mentioned, it is still worth explicitly enumerating them in order to plan accordingly.

3.2.1 Human resources

The supervisor, the tutor and the GEP professor will play an essential role as far as guidance is concerned, but they are not considered as job posts. Although this is an individual project, the following positions are considered:

- **Project manager**: The person in charge of coordinating the project development.
- **Software developer**: The person in charge of coding and testing the software.
- **Deep learning researcher and data scientist**: The person in charge of designing the new neural architecture, processing the data and conducting the experiments.

3.2.2 Hardware resources

The following computers will be used:

- Personal laptop: ASUS Zenbook UX305 i7-6500U 8GB RAM 256 SSD (2017). It will be mainly used for programming and writing the documentation.
- *Calcula* cluster: For training the network. The VEU research group has access to 1 node, 40 CPUs (Intel Xeon X5660), 8 NVIDIA GPUs, and 256 GB RAM.

3.2.3 Software resources and data resources

It is worth noting that almost all the software resources are open source:

- Ubuntu 18.04 and Unix utilities and editors: Installed in both the development laptop and the cluster.
- Python 3, PyTorch 0.6 and Fairseq: The programming language, the deep learning library and its module for Sequence2Sequence systems, respectively.
- TreeTagger[7] and Subword-NMT[8]: A linguistic tagger for different languages and an implementation of the BPE pre-preprocessing by the original author of the paper we previously cited, respectively.
- Slurm 18.08, Git and Github: The Workload Manager installed in the cluster, the software versioning tool and an online service to store the Git repositories, respectively.
- Overleaf[9] and GanntProject[10]: An online Latex editor and a tool for building Gannt charts and managing projects, respectively.
- EN-DE IWSLT 2014 corpus and BabelNet: The dataset for debugging and experimenting and a linked data database, respectively.

3.3 Main tasks

The following tasks can be identified. Since an agile approach is taken, the planned tasks could be modified or rescheduled depending on the feedback.

3.3.1 Previous tasks: learning

Some of the required tasks were performed before the project officially started (therefore, they will not appear in the Gannt chart and the table of hours per task), during the previous academic term. Once the project was assigned, the developer of this project was asked to read about MT and to assist to the Introduction to Deep Learning course[11] as audit. The developer familiarised with the PyTorch library and BabelNet. The used resources were the cited references and the laptop.

3.3.2 Initial development

The initial development will begin on Monday 12^{th} February and it will end by Friday 15^{th} March. The developer will start by familiarising himself with the cluster, installing the required dependencies and running the baseline architecture in order to reproduce the results. At the end of this phase, the design of the first version of the new architecture should have been fully specified, and some early components and pre-processing steps should have been coded and tested. In particular, a procedure for generating the PoS tags such that they are in the right format for the neural network should be developed, since it is required for testing the new architecture in the next phase. With regard to resources, the laptop and the cluster will be used.

[7] TreeTagger: `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger`.
[8] Subword-NMT: `https://github.com/rsennrich/subword-nmt`.
[9] Overleaf: `https://www.overleaf.com`.
[10] GanntProject: `https://www.ganttproject.biz`.
[11] Winter School on Introduction to Deep Learning ETSETB `https://telecombcn-dl.github.io/2019-idl`.

3.3.3 Thesis Management Course (GEP)

Mostly in parallel with the previous one, the task of completing the GEP course is scheduled from Monday 18^{th} February to Monday 1^{st} April, since the oral presentation of the final delivery of GEP is due on this date. The main objectives of GEP are both learning how to manage a project and writing the initial sections of the thesis. The used resources will be the course material provided by the professors, the development laptop, Overleaf and GanttProject. Professor Marcos Eguiguren will be providing me with the necessary feedback.

3.3.4 Architecture development

This task will involve the main development of the new architecture. It is scheduled from mid March to Tuesday 23^{th} April. Using PoS tags as factors is not the main goal of this project (instead, it is using linked data), but since adding them is a well known practice, they will be useful for checking whether the code is working as intended during this phase. The design of the architecture will evolve depending on the feedback. The GPU clusters will be heavily used during this phase.

3.3.5 BabelNet integration and final development

In this phase both the integration of BabelNet and further development of the architecture will be performed, because it is foreseeable that some bugs could still be detected. The task should start by Wednesday 24^{th} April and end by Friday 10^{th} May. Once the new architecture will have been extensively tested with the use of PoS tags, we will be able to explore the use of BabelNet.

3.3.6 Experiments and documentation

The experiments will be designed and conducted during the second half of May and the first week of June. Since the execution of them can take a considerable amount of time, at this point the new MT system should be fully ready to be executed. Although the documentation of the project will be constantly updated during the development of all the previous tasks, in the final phase writing the thesis document will be a major focus, specially when waiting for the experiments results. At this point, the role of researcher and data scientist will be more relevant.

3.4 Gannt chart

The Degree Final Project consists of 18 ECTS and 3 of these belong to GEP. At UPC, 1 ECTS corresponds to 25 hours of learning activities (up to 30 hours in the case of the final project), so we can estimate the total number of hours that should be devoted to the project, distribute them with a Gannt chart (figure 5) and break them down as follows (see table 9):

$$18 \text{ ECTS} \times \frac{25 \text{ hours}}{1 \text{ ECTS}} = 450 \text{ hours}$$

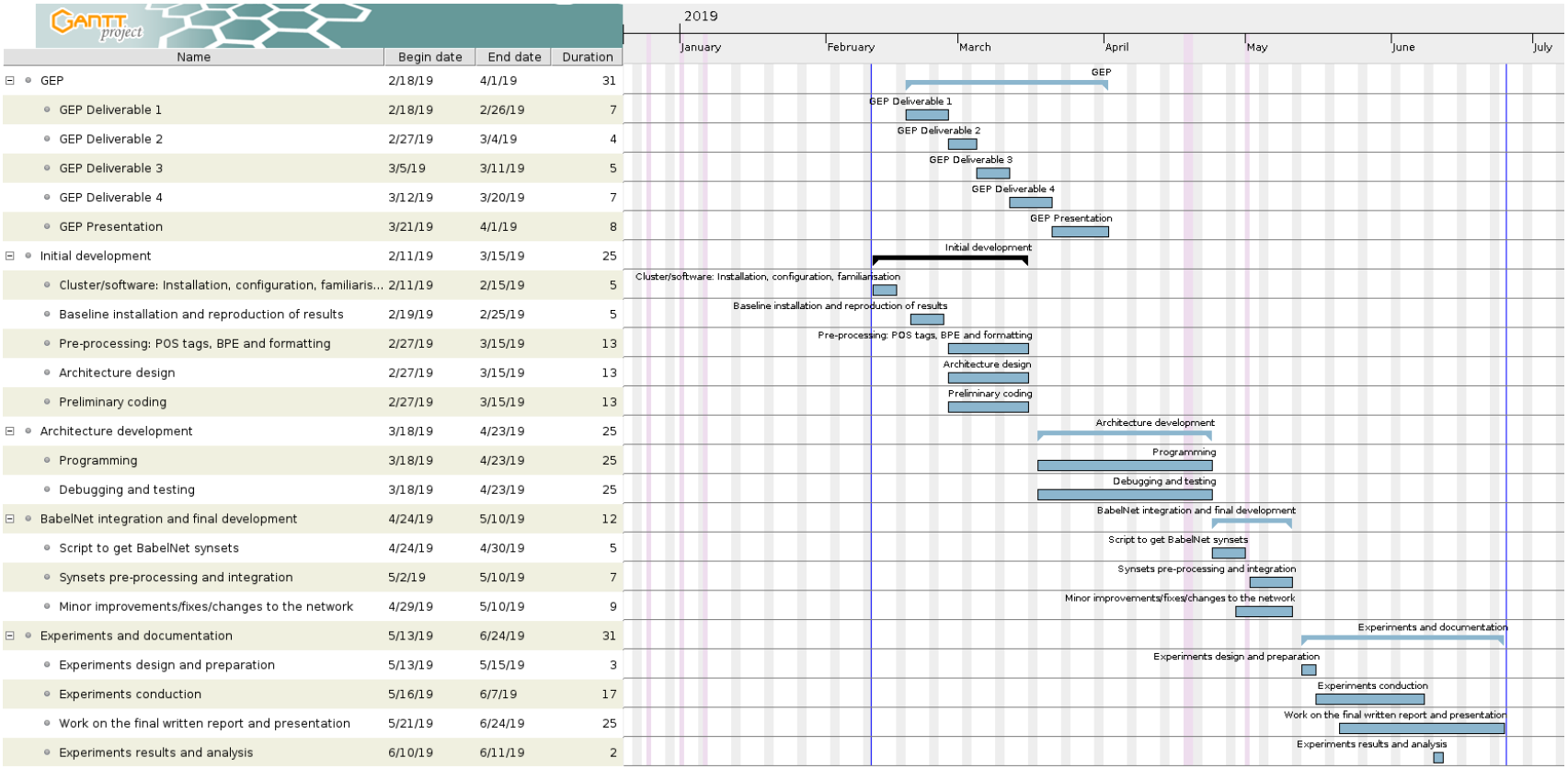

Figure 5: Gannt chart.

Task	Hours
Initial development	75 hours
GEP	75 hours
Architecture development	150 hours
BabelNet integration and final development	50 hours
Experiments and documentation	100 hours
Total	450 hours

Table 1: Estimated time of each task.

3.5 Alternatives and action plan

Even though the initial planning is feasible, different scenarios should be considered. Possible obstacles were already outlined in the scope section, but potential deviations from the scheduling and an action plan to deal with them accordingly will be explicitly detailed in this section. The action plan ensures that the project will be finished within the given time, or at least within the next term with the additional enrolment option provided by the university (in the case of a severe eventuality). The consequences for the duration of the project will be stated as well.

- Personal problems: If serious personal problems (eg. an illness) arose, the project would be rescheduled and the thesis would be delivered in the next term (15th October, one week before the presentations start), which would almost double the duration of the project (8 months).
- Scheduling: In general, if some tasks take more time than expected and the project is behind schedule, the first alternative that will be considered is using the additional hours not taken into account so far (weekends, holidays and extra hours per day). These hours have been intentionally reserved for countering eventualities and they give a huge margin. The duration in terms of days would remain the same, but the number of working hours per week would be increased (up to 90 hours), as detailed in the economic management section. Provided the project was severely behind schedule, the project would be rescheduled and the submission would take place in October. In that case, the duration of the project would take almost 4 additional months. In addition, in the next bullet point we anticipate task-specific solutions for a series of scenarios.
- Task-specific problems:
  - BPE and factors (like PoS tags or synsets) are not compatible or it is too difficult to use both of them simultaneously: In that case, BPE would be discarded. The fact that generating the PoS tags in order to debug the new architecture is planned to be one of the first tasks will allow to quickly decide whether it is feasible.
  - Integrating factors on the top of the Transformer architecture is not feasible or it is too time-consuming for a bachelor project: The first option will be exploring whether it would be easier in other libraries other than Fairseq. Otherwise, the Transformer architecture would be discarded for this project, and RNN-based or CNN-based architectures would be explored. Factors have already been implemented in RNNs, so

this existing works could be taken as a baseline and we could try to improve them by further exploring the features provided by BabelNet or by modifying the architecture.

  - Computing power: If the computing power provided by the cluster is not enough for our needs or the waiting time is too long, the following alternatives will be considered:
    * Selecting a subset of the training data for debugging faster.
    * Decreasing the number of experiments.
    * Renting a server with GPUs for some hours, if needed, but it would imply a major increase in terms of resource consumption.

  These alternatives for task-specific problems could increase the duration of the project in terms of total hours, but the project would still be delivered in June. Up to 30 additional hours can be anticipated for these task-specific problems.

- Lack of proper documentation: If the APIs, libraries or research papers we base our work on are not properly documented for our needs, we could open a Github *issue* to ask about doubts or consider using alternative libraries or works. The duration would remain the same in terms of total days. However, the number of working hours per week could be increased by using the reserved hours as stated before. For problems related to the lack of proper documentation, we can anticipate an increase of up to 30 additional hours of the total duration.

- Hardware, local software or online software service failure: If the hardware or software of the development laptop fails and we cannot easily fix it, we will work with the computers of the university, which will increase the resource consumption. We will always be able to work locally (eg. Latex can be used locally). All the work will be backed up in both Github and the cluster. This redundancy should be enough to prevent any major loss of information and to avoid any increase of the project duration.

- Bugs: Bugs should be avoided by extensively testing the program. If an error arises, there is no alternative to fixing it. If a minor bug was detected too late to repeat the experiments, it should be stated in a note when reporting the results of the experiments. Otherwise, if the bug was critical, the project delivery would have to be rescheduled. Again, delivering the project in October would imply an additional duration of approximately 4 months.

The maximum number of additional hours caused by deviations of the original plan is set to 90 hours. In the case that the project was rescheduled, since the duration in terms of months would almost double, the increase of the duration in terms of hours would be estimated as 450 additional hours, the same number of hours original planned for the first 4 months. Notice that while the originally planned duration of the project is actually 13 days longer than 4 months and the extra time provided by the additional enrolment is actually 9 days shorter than 4 months, we can safely approximate both of them to 4 months, since during the additional enrollment no any other courses would be enrolled.

4 Economic management

In order to budget the project, we will identify and estimate both direct and indirect costs and reserve a contingency fund. All relevant costs will be listed and linked to the tasks appearing in the Gannt chart, when possible. In addition, we will propose monitoring mechanisms.

4.1 Direct costs

We identify the following direct costs:

- **Software and data resources:** All the programs used in this project are open source. The only exceptions are Github and Overleaf, but since we can sign up to their free versions, no costs have to be assigned to software. The same applies to data resources, since both the dataset of choice and BabelNet have an open licence. Hence, the total cost of the software and data resources is 0€.
- **Hardware resources:** As far as personal hardware is concerned, the cost of the laptop, which was bought in April 2017, can be amortized as follows:

$$\frac{\text{Price}}{\text{Lifespan} \times \text{Working days per year} \times \text{Working hours per day}} \times \text{Hours of use for the project}$$

$$\frac{800\ €}{4\ \text{years} \times 220\ \text{days/year} \times 8\ \text{hours/day}} \times 450\ \text{hours} = 51.14\ €$$

Notice that we are counting all the assigned hours, because the laptop will be always in use when working on the project. That cost can be distributed among the different tasks proportionally to the number of hours, as done later on in order to build 5.

Product	Price	Units	Lifespan	Hours of use	Amortization
Laptop	800€	1	4 years	450 hours	51.14€

Table 2: Estimated cost of personal hardware resources.

The cluster will be considered as a service, because clusters are not bought by individuals, but institutions. In order to approximate its cost, we will take as a reference the market price of renting a roughly equivalent server with GPUs[12]. Since only hourly rates are offered and the cluster will not be used in all the activities, the costs will be reported for each task. Notice that the required compute hours do not necessarily match the working hours assigned to the task. The cluster may keep running the experiments during the night, or a particular task may require less computing hours or do not require them at all.

[12]It is difficult to find an exact match, but we are going to take an NVIDIA K80 instance offered by Google Cloud as a reference: `https://cloud.google.com/gpu`. In particular, we are keen on the on-demand hourly prices for Europe/Asia customers, $0.49. At the moment of writing this document, this quantity is equivalent to 0.44€.

Activity	Total hours	Required compute hours	Hourly price	Cost
Initial dev.	75 hours	50 hours	0.44€	22€
GEP	75 hours	0 hours	0.44€	0 €
Architecture dev.	150 hours	125 hours	0.44€	55€
BabelNet and final dev.	100 hours	75 hours	0.44€	33€
Experiments/Doc.	100 hours	90 hours	0.44€	39.60€
Total	450 hours	340 hours	0.44€	**149.60€**

Table 3: Estimated cost of the usage of the cluster.

- **Human resources:** In the case of human resources, the hourly prices of the project manager, software developer and deep learning researcher and data scientist have been estimated from Spanish (if possible, from Barcelona) salaries of the required positions[13].

Activity	Total	Project manager	Software developer	Data scientist
Initial dev.	75 hours	25 hours	25 hours	25 hours
GEP	75 hours	50 hours	0 hours	25 hours
Architecture dev.	150 hours	25 hours	60 hours	65 hours
BabelNet and final dev.	50 hours	10 hours	20 hours	20 hours
Experiments/Doc.	100 hours	10 hours	0 hours	90 hours
Total	450 hours	120 hours	105 hours	225 hours

Position	Hourly rate	Total hours	Total cost
Project manager	21€/h	120 hours	2520€
Software developer	12€/h	105 hours	1260€
Data scientist / deep learning researcher	22€/h	225 hours	4950€
Total	-	450 hours	**8730€**

Table 4: Estimated cost of the human resources.

By distributing the laptop amortization and the human resources costs among the tasks, the total direct cost of each task can be summarized with the following table:

[13]Data scientist (it is difficult to find estimates for deep learning salaries in Spain): 44,000€/y, approximately 22€ per hour (`http://www.expansion.com/expansion-empleo/empleo/2018/05/10/5af434a6e5fdea603f8b45b7.html`). Software developer: 25,000€/y, approximately 12€ per hour (`https://www.indeed.es/salaries/Desarrollador/a-de-software-Salaries,-Barcelona-CT`). Project manager: 42,000€/year, approximately 21€ per hour (`https://www.glassdoor.com/Salaries/madrid-project-manager-salary-SRCH_IL.0,6_IM1030_KO7,22.htm`). Other considerations, like the cost of social security or the fact that data scientists or project managers willing to accept a part-time job may be difficult to find, are not taken into account for this approximation.

Task	Software/data res.	Hardware res.	Human res.	Total
Initial dev.	0€	30.52€	1375€	1405.52
GEP	0€	8.52€	1600€	1608.52€
Architecture dev.	0€	72.05€	2675€	2747.05
BabelNet and final dev.	0€	38.68€	890€	928.68€
Experiments/Doc.	0€	50.90€	2190€	2150€
Total	0€	200.74€	8730€	**9155.74€**

Table 5: Direct costs of each task.

4.2 Indirect costs

As indirect costs, electricity, Internet connection and office supplies will be considered. The laptop is considerably energy-efficient and its power consumption can be estimated to be about 22 W. Electricity can be estimated to cost 0.12€/kWh. The cluster energy consumption will not be considered because it has already been factored in when detailing the costs of using the cluster. With regard to the Internet connection, a monthly price of 30€ can be taken as a reference. Office supplies and other minor costs could be approximated to be about 20€/month. All of them will be uniformly used during the execution of the project. In the case of the Internet connection and the office supplies cost, it is more difficult to estimate the cost for each task, since they are billed monthly and there is no exact task-month correspondence (some tasks even overlap), but we can approximate it.

$$\text{Total power consumption} = 450 \text{ hours} \times \frac{3600s}{1hour} \times \frac{22J}{1 \text{ s}} \times \frac{1kWh}{3.6 \times 10^6 J} = 9.9kWh$$

$$\text{Electricity cost} = 9.9kWh \times \frac{0.12 \text{ €}}{1kWh} \approx 1.19 \text{ €}$$

$$\text{Internet cost} = 4 \text{ months} \times \frac{30€}{1 \text{ month}} = 120 \text{ €}$$

Task	Hours	Electricity cost	Internet cost	Office/other	Total
Initial dev.	75 hours	0.20€	15€	10€	25.20€
GEP	75 hours	0.20€	15€	10€	25.20€
Architecture dev.	150 hours	0.40€	45€	30€	75.40€
BabelNet and final dev.	50 hours	0.13€	15€	10€	25.13€
Experiments/Doc.	100 hours	0.26€	30€	20€	50.26€
Total	450 hours	1.19€	120€	80€	201.19€

Table 6: Indirect costs of each task.

Resource	Total cost
Electricity	1.19€
Internet connection	120€
Office supplies and others	80€
Total	**201.19€**

Table 7: Estimated amount of indirect costs.

4.3 Cost contingency

Some of the obstacles and possible deviations from the planning do not involve new costs. For instance, if the laptop fails, it will be possible to switch to a computer from the university, and the costs should be approximately the same to the ones associated to the use of the laptop. Nevertheless, some deviations do involve potential extra costs:

- Not enough computational power: If an extra supply of computational power is needed, even after taking actions such as discarding some experiments, as stated in the action plan a server with GPUs shall be rented. The same Google Cloud pricing taken as a reference for the price of the cluster applies: Up to 100 additional hours $\times$ 0.44€/hour = 44€.
- Behind schedule (within term): If some tasks take more time than than expected or personal problems arise but the project can still be timely delivered, up to 90 additional hours may be factored in and the following extra costs can be foreseen:
  - Laptop amortization: + 10.3€.
  - Cluster: In principle the cluster usage would remain constant (we have already factored it in in the previous extra cost).
  - Human resources (assuming a proportional increase for the three positions): + 1650€.
  - Indirect costs: The Internet and office supplies costs would remain constant. Electricity cost could have an increase of up to 0.24€.
- Behind schedule (next term): 4 extra months should be considered with regard to the indirect costs, and up to 450 additional hours should be factored in with regard to laptop amortization and human resources. 200 additional compute hours should be factored in as far as the cluster is concerned:
  - Laptop amortization: + 51.14€.
  - Cluster: + 88€.
  - Human resources: The human resources cost would be doubled (+ 8730€).
  - Indirect costs: The Internet and office supplies costs would double because of the 4 additional months. Electricity cost could have an increase of up to 1.19€.

Risk	Probability (%)	Estimated cost	Risk exposure
Not enough computational power	20%	44€	8.80€
Behind schedule (within term)	50%	1660.54	830.27€
Behind schedule (next term)	10%	9070.33€	907.03 €
Total	-	-	**1746.10€**

Table 8: Contingency.

In addition, a 10% of the total budget will be reserved for unexpected costs.

4.4 Total costs

The total costs of the project can be summarized as follows:

Factor	Cost
Direct costs	9155.74€
Indirect costs	201.19€
Contingency for planned deviations	1746.10 €
Additional contingency	1110.30 €
Total	**12213.33€**

Table 9: Total costs.

4.5 Control management and budget monitoring

Since the main source of deviations in an IT project tends to be the duration and most of the costs of this project depend on the total number of hours, real hours will be the main control indicator. Time will be registered daily with a spreadsheet. The cluster usage will be monitored as well. At the end of each task, the tracked hours will be compared to the budgeted hours. A significant increase in the rates of electricity, Internet connection or the extra server (if used) would be factored in as well if that was the case. Provided that a task exceeds its planned cost, the additional contingency reserve will be used for re-balancing the budget. The following formulae will be used in case of an increase of the number of hours and the cost per hour, respectively:

$$\text{Cost increase} = (\text{Real hours} - \text{Expected hours}) \times \text{Estimated cost per hour}$$

$$\text{Cost increase} = (\text{Real cost per hour} - \text{Estimated cost per hour}) \times \text{Estimated hours}$$

In particular, we are planning the following key monitoring milestones corresponding to the possible deviations detailed in the action plan:

- Personal problems, severe delay in the scheduling or critical bugs found too late: In the case of a serious personal problem, a severe delay in the scheduling due to the underestimation of the difficulty of some tasks or a critical bug found too late to be fixed in time, an extraordinary meeting would be held with the supervisor and tutor, and the project

would be rescheduled. In this case, the whole contingency reserve should have to be used. However, it is a very unlikely scenario.

- Delays due to lack of proper documentation or minor bugs: These two scenarios could happen in the tasks of the initial development, architecture development or BabelNet integration. The progress will be shared with the director by sending daily emails and having weekly meetings. At the end of each week, if we consider that the project is behind schedule, additional hours will be added for the next week (notice that we have intentionally reserved time, like weekends). The contingency will be used accordingly, basically for human resources and laptop usage.

- Task-specific problems: In the case of delays because of the difficulty or inability to use BPE and factors simultaneously or the infeasibility of integrating factors on the top of the Transformer architecture, the same approach as before would be followed. In the case of lack of computing power, if renting a server was necessary, the planned contingency costs for this scenario would be allocated. Up to 100 additional hours have been considered in the contingency, which should be enough.

5 Sustainability report

The three dimensions of the sustainability matrix will be analyzed in terms of project put into production (PPP), exploitation and risks. As a summary, the quality of each cell is evaluated with a number between 0 and 10, the greater, the better (e.g. a high mark in risk means that there are only minor risks).

	PPP	**Exploitation**	**Risks**
Environmental	Consumption of the design: 2	Ecological footprint: 6	Environmental: 7
Economic	Invoice: 7	Viability plan 6	Economic risk: 6
Social	Personal Impact: 8	Social impact: 7	Social risk: 8

Table 10: Sustainability matrix.

5.1 Economic dimension

The economic dimension revolves around the budget of the project. We have tried to keep the cost estimation realistic and cautious. Nevertheless, as stated before, some considerations like the cost of social security or the fact that data scientists or project managers willing to accept a part-time job may be difficult to find (and therefore the actual cost per hour could be higher), have not been taken into account. The approximation of the cost of the cluster and the electricity might not be precise enough, but it was difficult to be more accurate. Hence, although the economic viability should be guaranteed, there are some caveats.

With regard to the PPP, the cost per training iteration may be higher than in other institutions because of economies of scale. However, the cost of this project is dominated by salaries, which are way lower in Barcelona than in California or some European countries, where most of the research in deep learning is carried out. Thus, this project is competitive in comparison to other deep learning research projects.

This work is a non-profit project, but regarding the useful life it can induce economic benefits, since our proposal may need less training iterations and data than existing state of the art approaches. Both training and obtaining data for NMT systems are considerably expensive. No economic risk can be identified apart from the failure of the project, which could imply a sub-optimal use of the involved resources, even though negative results in research are sometimes useful for future works.

5.2 Environmental dimension

As far as the consumption of the design is concerned, the PPP will involve a considerable energy consumption. Training neural networks has high computational costs. High performance GPUs spend a significantly greater amount of energy than CPUs. In comparison, the energy cost of the laptop is negligible. It is difficult to approximate the carbon footprint of the cluster, but considering a carbon footprint of 0.3 kg CO_2/kWh and a power consumption of 0.3 kW of the NVIDIA GPU:

$$0.3kW \times 340 \text{ hours} \times \frac{0.3KgCO_2}{1kWh} = 30.6KgCO_2$$

Thus, in order to minimize this impact the usage of the cluster is intended to be as low as possible, but in this case reusing resources is not feasible. Moreover, producing electronic components has a high environmental cost that should be taken into account even if the components are not new.

With regard to the exploitation of the project, the ecological footprint of the new architecture should decrease the one of the current state of the art models, because one of the intended benefits of combining NMT with linked data is that it would require less data (and therefore less training). Furthermore, the baseline architecture, the Transformer, is more efficient than the existing factored NMT systems, since they are based on recurrent neural networks. Nevertheless, it could happen that modifying the baseline architecture in order to work with factors increased its execution time more than it helped to decrease the need of data, so efficiency is an aspect that will have to be taken into account.

Since it is a research software project, the environmental risks regarding the construction, the useful life and the dismantling of the project are mostly negligible.

5.3 Social dimension

Regarding PPP, in terms of personal growth, apart from learning how to manage a project, there will be the satisfaction of delivering a project that could have benefits for the society.

With regard to useful life, the proposed architecture could improve the existing systems, and MT is used daily by millions of people. One of the promises of factored NMT with linked data, apart from requiring less data and training, is that it would perform better in few-shot learning, unlike existing systems, which do not always perform well on languages with less or no data at all. Since the concepts of linked data are universal, training with a particular language pair could transfer better to unseen languages. Hence, the new architecture could benefit the speakers of minority languages or languages with less NLP resources. Nevertheless, this possible benefit would mostly hold in the long-term, since in this project we will only start exploring the proposed path. The usefulness of the project is clear, provided it is successful.

6 Follow-up

In early May, we had the follow-up meeting for considering deviations. We can find its outline in this section.

6.1 Deviations and alternatives

In fact, with regard to timing, the initial work plan has been followed almost without deviations:

- The initial development and GEP exactly met the deadlines.
- The last bug in one of the proposed architectures was discovered in 23th April and the corresponding experiment ended the day after. The architecture development stage was planned to end the 23th.
- The BabelNet integration phase task ended a few days before its deadline (10th May).

However, in terms of resource consumption, it is worth noting that the project has been more costly than expected. In particular, apart from implying more working hours than planned, a considerable amount of extra computing hours have been spent. For instance, reproducing the baseline of IWSLT 14 implied a number of experiments until finding the exact configuration needed in order to get to 34 BLEU points and a half. In addition, several configurations of the factored architecture have been tried.

Fortunately, both additional working and computing hours were covered by the action plan and contingency. In order to keep to the schedule, we had reserved 90 hours (from weekends, for instance). On the other hand, we had foreseen a potential use of 100 extra computing hours.

Apart from using additional working and computing hours, no other alternatives from the action plan have had to be taken. Seen under the perspective allowed by time, the probability of needing extra computing hours was underestimated and the probability of being behind schedule was overestimated. 1746.10€ were reserved for the cost contingency, and a considerable amount of this quantity has had to be used (if we factor in the cost of the cluster and human resources).

The possibility of using additional datasets for evaluation has been considered, but it would be difficult to accommodate it to the time restrains.

On the other hand, since up to this point BabelNet synsets had not improved the baseline, we considered processing them in a different way and trying taking advantage of the developed architectures by using classical linguistic features.

In general, most changes can be considered to be minor and will not affect the implementation of the project, although it could happen that the best model was not based on BabelNet. With regard to the objectives, it is still unclear whether this project is going to be able to significantly improve the baseline, but if that was not the case it could still be considered a result.

6.2 Tasks done

Specifically, at the point of the follow-up meeting the following tasks had been finished:

- Baseline reproduction.
- Script to apply PoS tagging and align the PoS tags with BPE subwords. Different strategies for combining BPE with word features have been considered.
- Development of the different architectures that we will propose in section 7.
- Conduction of a number of experiments: As many as 15 without including the ones required to reproduce the baseline and the failed ones required to debug the new architectures (when they were still not working properly).
- Script to get disambiguated BabelNet synsets..
- Script to assign synsets. Different strategies for handling conflicts between synsets have been considered.

6.3 Status

At the moment of the follow-up meeting, the new neural architectures had been developed and tested and both PoS tags, subword tags and BabelNet synsets had been assigned and aligned. Several configurations had been tried in IWSLT14, but for the moment they did not clearly outperform the baseline. The following tasks remained to be done:

- Keep exploring possibilities in order to find a situation such that the new architectures clearly outperform the baseline.
- Thesis writing: thanks to GEP, sections such as state of the art and methodology are already written. At this point the focus was still on experimenting. With regard to the rest of the final document, a table of contents was proposed in the meeting.

6.4 Technical competences and integration of knowledge

The project, developed within the framework of the Computer Science specialization, involves the following technical competences:

- Evaluation of computational complexity of a problem [A little bit]: Computational complexity understanding is not directly required in order to develop this project. Nevertheless, it can be both useful and interesting to take it into account, since one of the main drawbacks of deep learning is its computational cost. One of the main reasons to chose the Transformer over other architectures is that it has less computational costs compared to other architectures. In addition, when developing preprocessing scripts it is important to try to make them linear if possible. Otherwise it could be the case that running them would not be feasible, since in IWLST 14 there are millions of tokens.
- Understanding of intelligent systems [In depth]: Deep learning and NMT in particular are techniques within the field of artificial intelligence. Thus, in order to successfully carry out this project we have to be aware of the paradigms and techniques of intelligent systems. For instance, it is important to take into account that these systems tend to be way more difficult to debug than other software components.

- Knowledge acquisition, representation and formalization, specially in the case of intelligent systems [Enough]: In the case of MT, the encoding of text (by the means of word embeddings) is crucial. In this project, aspects such as the embedding size or the way of combining different embeddings are relevant. In addition, this competence is particularly important because the project consists in using linked data (BabelNet) to enhance MT.
- Computational learning [In depth]: The project consists in developing a new deep learning architecture, therefore this competence is extensively covered. Fundamental concepts of machine learning, such as overfitting, are very important for this project, too.

6.5 Identification of applicable laws and regulations

The licenses of Fairseq (BSD license[14]), Babelfy[15] and the IWSLT dataset (Creative Commons license[16]) must be respected. In general, software and data used for this project are either completely open or at least permissive for research purposes. Other than that, legislation issues do not seem to be particularly relevant in this project.

[14] Fairseq BSD license: `https://github.com/pytorch/fairseq/blob/master/LICENSE`

[15] Babelfy, a private API that we will use for this project as detailed later on, has different usage limits depending on whether the project is for research or for commercial profit-seeking: `http://babelfy.org/guide`.

[16] See: `http://workshop2014.iwslt.org/downloads/IWSLT_Evaluation.pdf`.

Part II
Contributions

In Part II, we will detail the work that has been done during this project. First of all, we will propose a set of modifications to the vanilla Transformer architecture such that we will be able to input multiple aligned sequences (factors) instead of only one (the source sentence itself). The motivation is being able to input features referring to words alongside words themselves, instead of just the original words as in the vanilla Transformer.

Once we will have detailed the required architectural modifications in order to work with factors, we will go through the particular features that we will test. We will start by revisiting the classical linguistic features. Later on, we will introduce the linked data we are going to use, BabelNet, and how we are going to extract and process its information into word features that can be input to our architectures. We we will discuss the implications of introducing features if words are split into subword units as in BPE preprocessing. Before outlining the results, we will introduce the experimental framework where the experiments were conducted. Finally, we will discuss the process and results of the project and suggest future research lines.

7 Factored Transformer

In this section, we are going to go through the proposed architectures for trying to improve the vanilla Transformer by introducing extra information apart from the source sentences. The two proposed architectures and their respective variants will have, apart from other components, N neural modules, one for each feature and one for the words themselves, or more generally, *factors*. The motivation of these architectures is clear: being able to input features alongside words in a state of the art model, the Transformer, as was already proposed in other works but with RNN-based architectures.

7.1 Considerations

In the state of the art section, we saw different works that incorporated linguistic features. In all of them, the architectures were RNN-based, usually involving a Bi-LSTM. Recall that in the case of the work which coined the term of factored NMT [31] the modifications were mostly applied in the decoder, which involved modifying the beam search procedure. On the other hand, in [32], the modifications were applied in the encoder side, by adding an embedding layer for each feature and concatenating their outputs with the output of the word embeddings.

From now on, we are going to refer to the system of [32] as *RNN-EncoderFeat*. For this work, we are going to take the RNN-EncoderFeat approach as a reference, in the sense that we are going to introduce features on the source side. Modifying the decoder is considered to be out of the scope of this project. Nevertheless, one of the main novelties of this project is that instead of being based on a LSTM, we are going to base our work on the Transformer architecture, which is considered to be state of the art by the literature.

Using the Transformer instead of an RNN-based architecture presents some challenges:

- The Transformer can be more sensitive to hyperparameters than an LSTM [40]. However, optimizing hyperparameters for each model is not a feasible option.
- There is a lack of reference implementations.
- The embedding sizes will have some restrictions, as we will see later on.

Still, we consider that it is worth betting on the Transformer for this project, for both its efficiency and results. Apart from just implementing the RNN-EncoderFeat model but with the Transformer, we are going to explore alternative approaches. The general idea will be that we will have N neural modules, M_i, one for each feature and one for the words themselves (factors), and their outputs will be combined by concatenating, as in RNN-EncoderFeat, or in some alternative ways. In addition, these modules could be either embedding layers, as in RNN-EncoderFeat, or full-blown encoders, as we will propose as well.

7.2 Single encoder architecture

As in RNN-EncoderFeat, this architecture will have only one encoder, but multiple embedding layers (one for each feature and one for the words themselves). The embedding vectors are looked up for each factor separately. Then, the respective outputs are combined. Later on, we will discuss the different combination strategies. Finally, positional embeddings are summed to the combined embedding vector, and the first attention/feed-forward layer of the Transformer's encoder is input this vector as in the vanilla model. Positional embeddings are summed to the final vector and not to the individual embeddings because we are modifying the individual token embeddings, not the length of the sequence, which remains unchanged, so the positions in the sequence are the same. We can think of these embedding layers as a *factored embedding layer*. Apart from the embedding layer of the encoder, the rest of the Transformer remains unchanged.

7.3 Multiple-encoder architecture

Using multiple encoders for factored NMT, one for each factor, is not a common approach in the literature. In fact, we are not aware of any other work that does so. Instead, it has been used for multi-source MT, which is a task that consists in translating from multiple sources (ie. languages) referring to the same sentence into a single target sequence, always with RNN-based models as far as we know, as in [41]. It is possible that the architecture that we are introducing could be used for multi-source MT, but it should be investigated and it is not the goal of this project. Nevertheless, we had the intuition that having one encoder for each factor could improve the results, particularly for features that have a vocabulary size comparable to the ones of the words (specially, linked data features, as we will see later on).

In this architecture, we have N encoders, one for each factor. These encoders are not different from the one of the vanilla Transformer. Instead of combining the outputs of the embedding layers of one encoder as in the single-encoder architecture, in this architecture the outputs of the multiple encoders are combined (*factored encoder*). The rest of the model remains unchanged.

In the following pages, in figures 6 and 7 we can find graphical descriptions of the single-encoder and multiple-encoder architectures, respectively. In the next section, we will see in which particular ways the outputs of the N embedding layers or the N encoders could be combined.

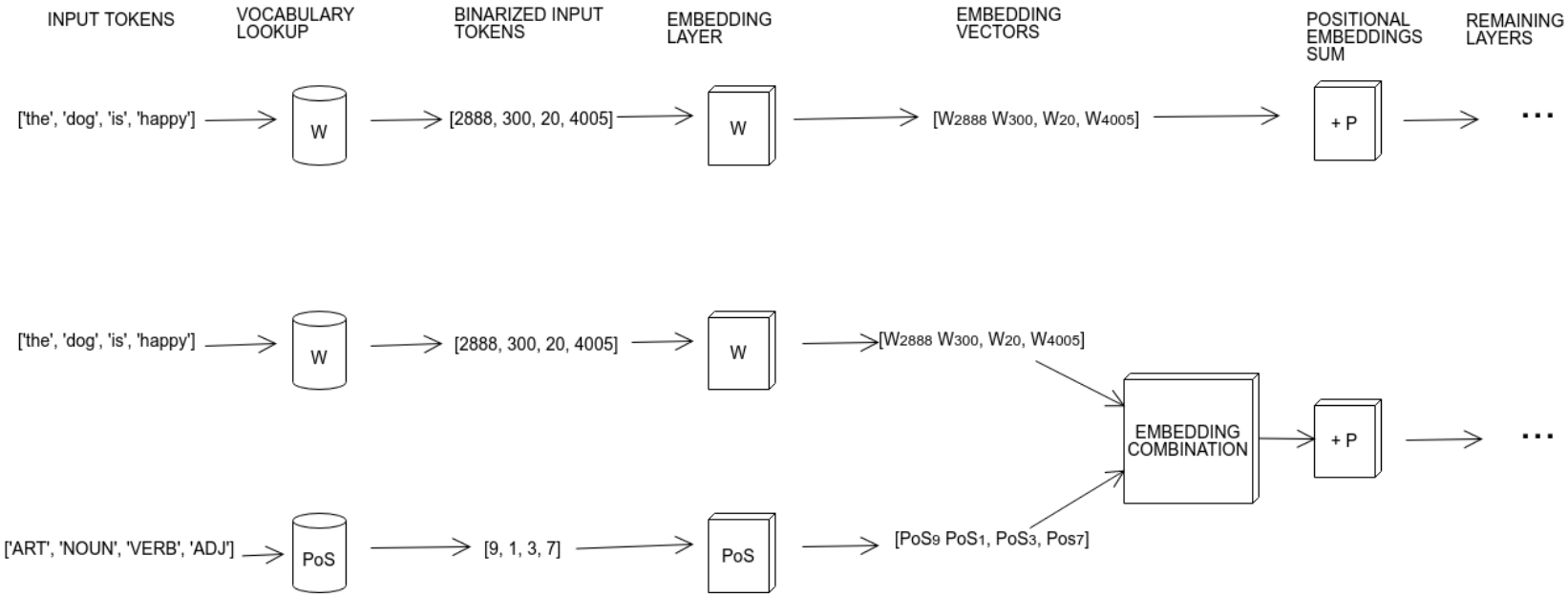

Figure 6: The vanilla Transformer (above) and the single-encoder architecture (below): In the single-encoder architecture, we modify the encoder such that each factor, including the words, W, has its own embedding layer.
In this example, a tokenized sentence in English and the corresponding Part-of-Speech tags, PoS, are input to the encoder. The tokens are replaced by their unique numeric identifiers. The embedding layers work as lookup tables that transform the numeric identifiers of the tokens by dense, lower-dimensional vectors. The respective embedding outputs are combined, which can be done in different ways. Then, positional embeddings, which in the Transformer are used for encoding the position of each token in the sequence, are summed, and these vectors are input to the remaining layers of the encoder. The rest of the model remains unchanged. Notice that positional embeddings are summed to the combined embeddings and not to the individual embeddings separately because we are modifying the individual token embeddings, not the length of the sequence, which remains unchanged, so the positions do not change.

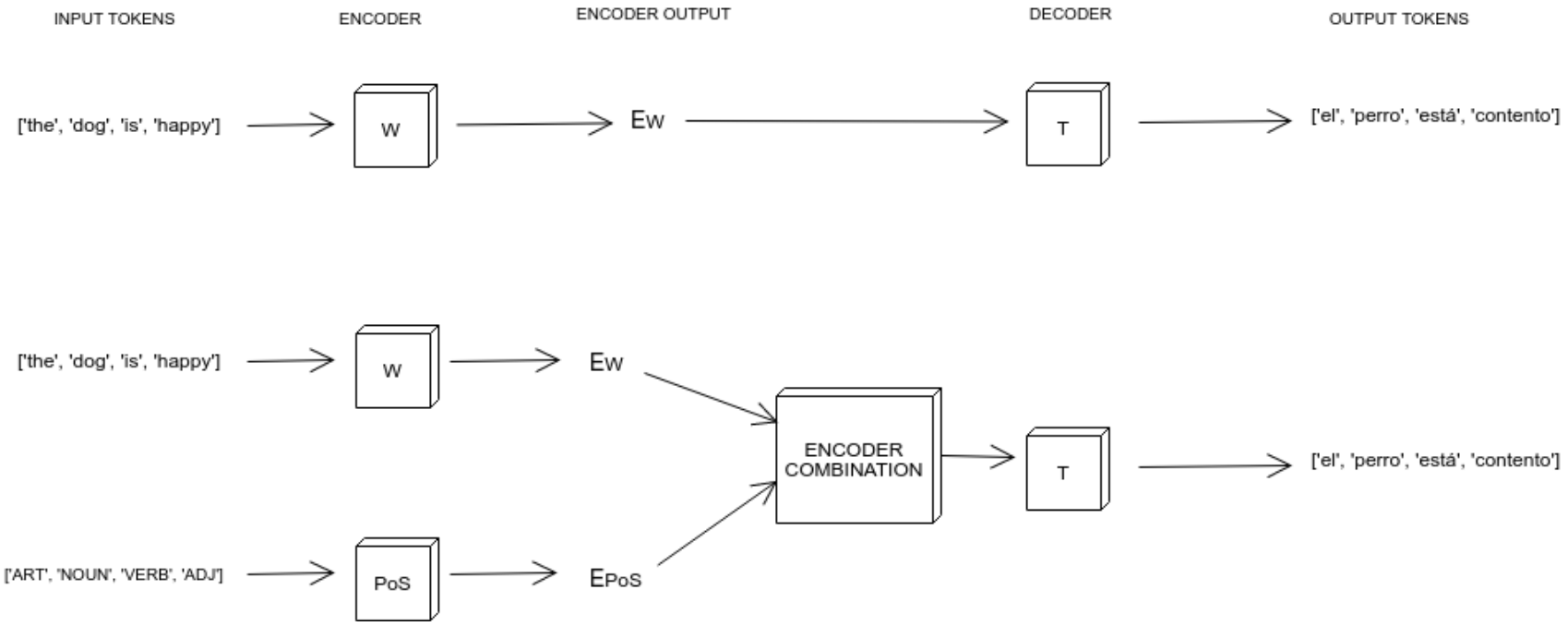

Figure 7: The vanilla Transformer (above) and the multiple-encoder architecture (below): In the multiple-encoder architecture, we modify Transformer such that it has N encoders, one for each factor, including words, W.
In this example, a tokenized sentence in English and the corresponding Part-of-Speech tags, PoS, are input to their respective encoders. The encoders remain unchanged with respect to the vanilla Transformer. Each encoder has its own embedding layer and positional embeddings. The encoder outputs are combined, which can be done in different ways. The rest of the model remains unchanged. The decoder for the target sequence, T, generates the sentence from the combined encoder outputs in the target language, in this example, Spanish.

7.4 Combination strategies

In RNN-EncoderFeat, features and words embeddings were combined by the means of concatenating. However, it is not the only possible combination strategy. OpenNMT allows both summation and concatenation. In the multi-source MT work we mentioned, the authors tried both concatenation and a tangent activation (as in LSTMs) and a full-blown LSTM for combining the outputs of the two encoders.

If we have N neural modules, M_i with $1 \leq i \leq N$, one for each feature and one for the words themselves, an input to each of them of dimensions `[# tokens in sentence]`[17] and an output from each of them, $\mathbf{F_i}$ with $1 \leq i \leq N$, of dimensions `[# tokens in sentence, M_i embedding_size]`, we can consider the following combination strategies:

- Concatenation: The respective outputs of each M_i, $\mathbf{F_i}$, are concatenated, such that the resulting dimension is `[# tokens in sentence,` $\sum_{i=1}^{N}$ embedding size of M_i`]`, and input to the next layer. Obviously, all M_i share the number of tokens (and the number of sentences if we were considering the full batch), because they are referring to the same sentences, but notice that is not necessary for the embedding sizes to be equal. The operation can be represented as:

  $$[\mathbf{F_1}; \mathbf{F_2}; \ldots; \mathbf{F_N}]$$

  where ';' is the tensor[18] concatenation along the corresponding axis.

- Summation: The respective outputs of each M_i are summed and input to the next layer, instead of concatenated. Thus, the embedding size of each factor must be equal, and the resulting dimension is `[# tokens in sentence,` embedding size of each M_i`]`:

  $$\mathbf{F_1} + \cdots + \mathbf{F_N} = \sum_{i=1}^{N} \mathbf{F_i}$$

- Hybrid combination: Another option would be to sum some $\mathbf{F_i}$ and then concatenate the resulting vector to the remaining $\mathbf{F_i}$, to concatenate some $\mathbf{F_i}$ separately and then sum the resulting vectors, in a tree fashion, or other alternatives consisting in combining the first two combination strategies we saw. For each operation, the dimensions must agree as we said before.

- Fully-connected layer: The outputs are concatenated as in the first combination strategy that we saw, but before being input to the next layer, a fully-connected layer (as in multilayer perceptron) performs dimensionality reduction to the desired size:

  $$\text{ReLu}(\mathbf{W_f}[\mathbf{F_1}; \mathbf{F_2}; \ldots; \mathbf{F_N}])$$

  where W_f are the weights associated to this fully-connected layer.

[17] Here we are considering the inputs to the neural modules themselves. Usually, inputs are batched into a number of sentences, such that we have dimensions `[# sentences, # tokens per sentence]`, but this fact does not affect our observations in this section. Recall that in the encoder input tokens are the natural numbers that have been assigned in the preprocessing, so they are scalars, even if they can be thought as representing a one-hot encoding vector.

[18] Tensors are a generalization of vectors and matrices used in deep learning (among other uses) [7]. However, for this section and in our work, we are not exploiting the full mathematical properties of tensors and we can consider them just as n-dimensional matrices, even if that is an oversimplification in mathematical terms.

For the implementation and experiments, we are going to consider the first two strategies.

7.5 Embedding sizes

The bigger the embedding size, the more vocabulary can be learnt. However, at some point the embedding layer can start to overfit. In addition, we do not want the embedding size to be that big that it makes the model too computationally expensive.

In OpenNMT, the embedding size cannot be specified for each feature. Instead, either all of them have the same size, or the embedding size is computed from this formula[19]:

$$\text{embedding size}_i = \text{vocabulary size}_i^{\text{feature exponent}} \forall i \in \text{Features}$$

With feature exponent = 0.7 by default. This formula tries to capture the principle that the embedding size should grow with the vocabulary size. In practice, it can result in large, infeasible embeddings for features with big vocabulary sizes, and it is not flexible enough to let introduce a different exponent for each feature.

In the RNN-EncoderFeat proposal, the total size of the embedding layer was keep to 500. Features with a few dozens of possible values were assigned an embedding size of 10, for instance, while words had an embedding size of 350. However, in practice the Transformer is known to need bigger embeddings [40]. A word embedding size smaller than 512 (without counting the features) could not be enough for learning the word vocabulary.

Needless to say, the total embedding size of the encoder should match the embedding size of the decoder, because otherwise there would be a dimension mismatching.

Another issue to consider is the fact that while RNN-based embeddings can be of arbitrary dimensions, in the Transformer they must be a multiple of the number of attention heads. Recall from previous sections that the multi-head attention mechanism applies multiple blocks in parallel (one for each head), concatenates their outputs and finally applies a linear transformation. Notice that as many outputs as the number of heads will be concatenated. Thus, the embedding dimension must be divisible by the number of heads. Otherwise, there would be a dimension mismatching.

The reader will notice that the last restriction affects the total embedding size of the encoder, but not the individual embedding sizes. That is to say, in the multiple-encoder architecture, each factor should verify the restriction. In contrast, this does not need to be the case for each factor in the single-encoder architecture (the total sum must be a multiple of the number of heads, but not each embedding individually). Ideally, the embedding size of each factor should be optimized by some search procedure. In practice, we will have computational limitations for doing so.

One alternative for avoiding overfitting would be to freeze the weights (ie. stop updating weights) at an early epoch, but only for the features with small vocabulary size, and keep training and updating the rest of the model. We implemented this feature as well.

[19] OpenNMT training options: `http://opennmt.net/OpenNMT-py/options/train.html`.

7.6 Complexity

Although performing a full complexity analysis of our models is out of the scope of this project, we can make some relevant observations.

We said that the architecture we are basing our modifications on, the Transformer, has less computational complexity and it is more efficient. In the Transformer's original publication [25] we can find a complexity analysis, summarized below. In table 11, we can see how self-attention layers (the ones used in the Transformer) compare to other alternatives:

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \times d)$	$O(1)$	$O(1)$
Recurrent	$O(n \times d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \times n \times d^2)$	$O(1)$	$O(\log_k(n))$

Table 11: Complexity of Self-Attention layers and other alternatives. n is the length of the sequence, d is the representation dimension, and k is the kernel size of convolutions in the case of CNNs. Source: Adapted from [25].

Apart from having constant sequential operations (which means that they can efficiently run in parallel) and constant maximum path length, in self-attention layers the complexity is quadratic with respect to the sequence length (ie. the number of tokens in a sentence), not the representation space (tokens). This a key difference, because it means that self-attention layers will be faster than their recurrent counterparts whenever the sentence length, n, is smaller than the representation dimensionality, d, which will be often the case, specially when using BPE.

In the case of the single-encoder architecture, notice that we are increasing d, not n (the sequence remains equal; we do not modify the number of words), which is convenient for the Transformer for the reason we stated before. In the case of the multi-encoder architecture, even if the number of factors, N, can be considered as a constant, having N full encoders (each of them with multiple self-attention layers) could considerably affect the efficiency of the model. With regard to the scalability, the N neural modules could run in parallel, because they do not depend on each other.

In practice, the memory necessary for loading the different embedding layers could be a concern. However, as we will see later, we have been able to execute the models without any problems.

8 Linguistic features and linked data

In the previous section, we detailed the neural architectures that we are proposing in this work. Their key addition is that they allow to introduce multiple sequences referring to the same source sentence, which we referred as *factors*. Thus, we are going to be able to input both the words (or subwords) themselves, the first factor, and a set of features associated to them, the other factors.

In this section, we are going to go through the features that we are going to consider to use alongside words in order to try to improve MT. Traditionally, linguistic features have been used in some NLP applications, even with non-neural approaches [2]. Intuitively, linguistic information should be useful for getting a better representation of the text.

However, one of the main novelties presented in this work is the use of linked data, which has barely been exploited in NMT [35]. In particular, we are going to extract semantic and language-agnostic information.

Finally, we are going to go through the implications of using features alongside subwords (instead of whole words).

Notice that our intention is to use features in the source sequence, not in the target sequence, but from the point of view of the features themselves it would not make any difference.

8.1 Linguistic features

In RNN-EncoderFeat, the following features were used:

- Lemmas: The same base word can take different forms. For instance, in the case of verbs, both 'are' and 'is' derive from 'be'. These variations increase the vocabulary size and can confuse the MT system. By *stemming*m we mean the heuristic-based procedure of taking the beginning of each word in order to get something similar to the *stem* or root of the word. Instead, by *lemmatization*, we refer to a more complex procedure that factors in both morphological analysis and a pre-defined vocabulary. Such procedure is usually able to retrieve the *lemma* or dictionary form of a word. In figure 8, we provide an example of lemmatization.

  ```
  und|und was|was menschliche|menschliche gesundheit|gesundheit ist|sein ,|,
  kann|können auch|auch ziemlich|ziemlich kompliziert|komplizieren sein|mein
  .|.
  ```

  Figure 8: Lemmatization of a tokenized German sentence from the training set of the IWSLT 14 German-English dataset. Features are separated by '|'. Notice how the verb form 'ist' ('is') is lemmatized as 'sein' ('to be').

- Part-Of-Speech: Starting from the three basic lexical categories (nouns, verbs and adjectives), linguistic models introduce additional categories such as prepositions and determinants, apart from the category of punctuation marks. The vocabulary size of this feature

is tiny (a few dozens of unique tags, depending on the tagger). While the point of lemmatization was to expose the base form of the different words in order to abstract away from the particular derivations or inflections, the promise of PoS tagging is letting the MT system access the underlying structure of the text. In figure 9 we provide an example of PoS tagging.

```
und|CONJ was|PRON menschliche|ADJ gesundheit|ADJ ist|AUX ,|PUNCT kann|VERB
auch|ADV ziemlich|ADV kompliziert|ADJ sein|AUX .|PUNCT
```

Figure 9: Part-Of-Speech tagging of the same sentence as before. For instance, `kann` (can) is tagged as `VERB` because it is the main verbal form of the sentence.

- Word dependencies[20]: By dependency parsing, we mean profound syntactic analysis such that dependencies between words are detected in a tree fashion. For instance, in a passive sentence, key components such as a the passive subject or the agent would be detected, although most dependency parsers offer a more granular analysis. In figure 10, we provide an example of dependency parsing.

```
kennen|ROOT wir|sb das|oa nicht|ng alle|oa ?|punct das|oa haben|ROOT wir|sb
alle|oa schon|mo erlebt|oc .|punct
```

Figure 10: Dependency parsing of another sequence in German from the same corpus as before. Notice how `wir` (we) is detected as the subject (`sb`) of both sentences. `ROOT` means that the word has no syntactic head.

- Morphological features: Morphological features are less context-dependent. Depending on the tagger, they can be considered a fine-grained version of PoS. The parser can annotate words with features denoted by their morphemes, such as the number (singular or plural) in the case of nouns or the tense (past, present or future) in the case of verbs. In figure 11, we provide an example of morphological tagging.

```
und|KON was|PWS menschliche|ADJA gesundheit|ADJD ist|VAFIN ,|$, kann|VMFIN
auch|ADV ziemlich|ADV kompliziert|ADJD sein|VAINF .|$.
```

Figure 11: Morphological features of a German sentence. Notice how `kann` is annotated as `VMFIN` (finite verb, modal) instead of `VERB` as in the Part-Of-Speech example in figure 9.

For a complete reference of classical linguistic features, see [42] and [2], from where we extracted information for describing the features.

The vocabulary size of each feature, which as we have said in the previous section is very relevant to the size of their respective embeddings, is very small in most of the cases, except in the case of

[20]For more details and examples on word dependencies, see Stanford Dependencies: https://nlp.stanford.edu/software/stanford-dependencies.html

lemmas. Lemmas have a vocabulary size of the same order of magnitude than the original words (without subword splitting), but smaller, because all the different inflections and derivations of a word will collapse into the same base form.

There are different taggers available and nowadays some of them are based on deep learning. In RNN-EncoderFeat, the authors used Stanford's CoreNLP tagger [43]. For this work, we started with TreeTagger, a Part-Of-Speech tagger that also offers lemmatization, but later on we pivoted to Stanford models, through a Spacy Python wrapper[21].

As a matter of fact, linguistic taggers are not infallible. Even though they are supposed to help the NLP system by providing additional information, at least in some cases they could be providing wrong annotations. In addition, tagging can be a resource-intensive task, and the corresponding neural network will need more parameters at least for dealing with the feature embeddings.

8.2 Linked data

As we have said, by linked data we mean structured data that forms a graph, that is, an ontology-based database. In the context of NLP, linked data is often used to describe a dataset such that thanks to its interlinked nature, semantic information can be extracted.

There are many different linked data databases that we could use with our proposed architectures, without any modifications, provided that words could be tagged by discrete identifiers. For instance, domain-specific linked data could be used for domain adaption of an NMT system, by inputting domain features, as we will further suggest as future work, in section 11. Nevertheless, for this work we will explore linked data for the general domain.

8.2.1 BabelNet

In particular, we are keen on BabelNet [17], which is the linked data used by [35]. BabelNet is a very large multilingual encyclopedic dictionary and semantic network.

BabelNet is automatically built from the combination of many sources[22], such as Wikipedia, WordNet (a computational lexicon of different languages), Wikidata (a machine-readable knowledge base), Wiktionary (Wikimedia's multilingual dictionary), and ImageNet (an image database organized according to the WordNet hierarchy).

Apart from the large amount of data, its potential comes from the way the different sources are integrated. For instance, as we have said, ImageNet is organized according to WordNet's hierarchy, while Wikipedia entries are linked to Wikidata and Wiktionary terms. In figure 12, we provide an example of the kind of semantinc information that can be extracted from WordNet, one of the sources of BabelNet.

Thanks to this deep integration, BabelNet makes available a large semantic network. In particular, there are approximately 16 million entries, called synsets. Synsets are language-agnostic,

[21] spaCy + StanfordNLP: `https://github.com/explosion/spacy-stanfordnlp`.

[22] See Wikipedia: `https://www.wikipedia.org/`, WordNet: `https://wordnet.princeton.edu/`, Wikidata: `https://www.wikidata.org/wiki/Wikidata:Main_Page`, Wiktionary: `https://www.wiktionary.org/` and ImageNet: `http://www.image-net.org/`, among others.

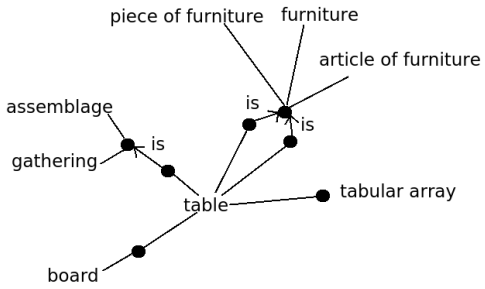

Figure 12: WordNet visualization: Some of the semantic relationships for 'table'. We can observe some instances of the 'is' relation. Source: Adapted from WordVis (`http://wordvis.com`), a visualization tool for WordNet.

that is to say, the same concept will have the same synset assigned in all the 271 languages in which BabelNet is available. Perhaps even more importantly, synsets are shared for synonyms as well.

The reader will notice that BabelNet has a great potential for both entity recognition (identifying target entities in text, eg. identifying countries) and word sense disambiguation (determining to which particular meaning a polysemous word or expression may be referring to), which intuitively should be useful for improving MT, particularly the latter. In addition, unlike linguistic taggers, BabelNet is uniformly available for many languages.

Nevertheless, BabelNet itself is just a large and interlinked database with REST and Java APIs to extract word-level information. Plain BabelNet will retrieve all the possible synsets that a particular word may have, and links to possibly linked concepts. It is up to the developer to actually perform the sense disambiguation by building a rule based system or a MT application.

In the work that we cited that used BabelNet [35], the authors used plain BabelNet id's, without sense disambiguation. Instead, for this work we are going to use Babelfy.

8.2.2 Babelfy

Babelfy[23] [44] is a service based on BabelNet that performs both entity recognition and word sense disambiguation. Both its Java and REST APIs retrieve only one synset for each word given its context (that is to say, its sentence), instead of returning the list of all the possible synsets.

Unlike BabelNet, Babelfy is a private API. We requested the maximum allowed limit for research purposes that is still free of pay, which is 5000 calls per day. For optimizing the usage allowed by this limit, we split the sentences of the corpora into chunks of about 3500 characters (without surpassing this limit and without dividing the same sentence into 2 groups), which is the character limit per call. In particular, we are going to use its REST API by developing a script for retrieving synsets by making HTTP calls to the said API. The process is depicted in figure 13. Apart from the detected synsets, Babelfy returns other information, and in the query we can specify different configurations. However, having analyzed the different options[24], the configurations

[23]Babelfy: `http://babelfy.org/about`.

[24]Babelfy's API guide: `http://babelfy.org/guide`.

and parameters we are keen on are the following:

- Configuration: In general, all the configurations will be left as default, which should work for most of the cases. However, it is important to limit matches to exact matches and deactivate the multilingual option. Otherwise, for instance, random German words can be identified as some American locations. In our experience, the partial matching mechanism is not very reliable, because having almost 300 languages and that many concepts it is very easy for Babelfy to misclassify some entities.
- Offsets: Babelfy returns both the token offset and the character offset. We are going to discard the former and keep the latter, because Babelfy does not return the results of its tokenization process. The n-th token detected by Babelfy may not be n-th in the user's tokenization process. Instead, the character offset is shared both by Babelfy and the user, because it is the exact start and end of synset in terms of characters in the original text.
- Candidates: We are only keen on the top synset candidate.
- Scoring: Babelfy returns the confidence for each synset. We will see more details on this matter later on.

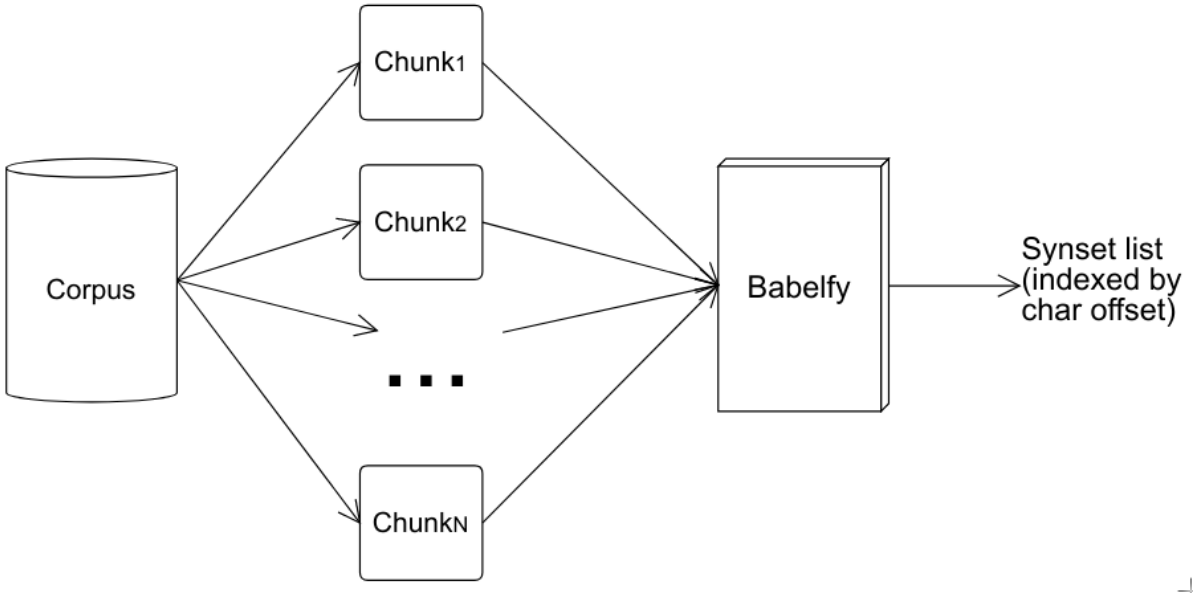

Figure 13: Synset retrieval process: 1-The corpus (all the subsets) is split into chunks of the largest number of characters such that it is less or equal than 3500, Babelfy's character limit per call, and such that no sentences are split (ie. the whole sentence must be input in the same chunk, otherwise Babelfy misses the context provided by the sentence). 2-Each chunk is sent to Babelfy via an HTTP call. The number of calls per day cannot be greater than the number allowed by the API limits, which happens to be 5000 in this case. Thus, the process has to wait until the following day if the corpus has more than 5000 chunks of 3500 characters, which was the case in this work. 3-The list of synsets, indexed by character offset, is retrieved by Babelfy.

The retrieved synsets must be post-processed. In particular, apart from parsing the results, we must assign a synset to each word by taking into account the offsets returned by Babelfy.

It is worth noting that even by specifying that we are only keen on the top candidate for each word in the query, Babelfy will return more than one synset in the case of multi-word synsets. That is, if an individual word has been assigned to a synset, and this word belongs to a set of words that have been assigned to a collective synset (shared by all the fragment of the sentence), Babelfy will return both. For instance, in the case of "semantic network", Babelfy will return three synsets: one for "semantic", another one for "network", and a third one for "semantic network". Different strategies can be considered:

- Prioritizing individual synsets.
- Prioritizing the *longest* synsets (ie. multi-word synsets with the largest number of words).
- Deciding depending on the scores returned by Babelfy.

The third option would be the best one because the synsets with the most confidence would be chosen. Babelfy returns three different scores, namely, the score, the global score, the coherence score. However, apart from the lack of documentation on this matter, the three scores do not seem to be particularly coherent, specially in the case of multi-word synsets. For instance, in the case of "encyclopedic dictionary", the multi-word synset of "encyclopedic dictionary" has a higher score than the single-word one of "encyclopedic", but the synset of "dictionary" has a higher score than the one of "encyclopedic dictionary". In this case, the score-based decision would be incoherent, because if we chose a multi-word synset for one particular token, all the words corresponding to that synset should be assigned the same synset. In addition, the three kinds of score do not seem to be particularly coherent in our experience.

In the course of our work, we have found that in the corpus that we explored (in particular, IWSLT 14 DE-EN) about 70% of the tokens do not get any synset, which might be problematic, since a vector composed of 70% of unknown tokens might not be that informative to the neural network. Instead, we are going to assign an alternative feature for tokens without any assigned synsets. In particular, we are going to assign morphological features, since many words do not have a synset because they are not supposed to (eg. punctuation marks, determinants...).

There is one last aspect that should be taken into account with regard to Babelfy post-processing, which is the vocabulary size. In the course of our work, we have found the vocabulary size of the synsets to be quite large and therefore difficult to learn. Again, there are different strategies for fixing this issue:

- Thresholding: Setting a minimum number of occurrences in the training set and pruning all the synsets that are below the threshold.
- Rule-based choice.

The second option would be the best option provided some fancy heuristic or even rule-based domain knowledge was available. However, in the general case, it is not practical to come up with useful rules. For instance, we tried discarding the redundant synsets. By redundant synsets, we mean the cases in which a token is always assigned the same synset in the whole training set. However, this rule affected a very low percentage of the tokens. In IWSLT 14 German-English, only 2% of the tokens had the same synset regardless of the context, including the cases in which the token sometimes has a synset and sometimes it does not have any synset. We will give more

details on this dataset in section 9. In figure 14 we provide an example of a sentence tagged with synsets, while in the following page, in figure 15, we summarize the process of aligning and assigning synsets that we have described.

```
in|APPR der|ART zwischenzeit|NN ,|$, wurde|VAFIN die|ART alte|bn:00107787a
mentalität|bn:00054380n ,|$, mit|APPR grossem|ADJA a|FM ,|$, von|APPR
seinem|PPOSAT vater|bn:00009616n repräsentiert|bn:00090234v ,|$, dem|ART
vorsitzenden|ADJA der|ART nigerianischen|bn:00107300a bank|bn:00008364n ,|$,
der|PRELS die|ART cia|bn:00017182n vor|APPR seinem|PPOSAT sohn|bn:03854737n
warnte|bn:00095642v ,|$, der|PRELS kurz|bn:00098854a vor|APPR dem|ART
angriff|bn:00006994n stand|bn:00021644n ,|$, und|KON diese|PDAT
warnung|bn:17105905n traf|bn:00085339v auf|APPR taube|bn:00062350n ohren|NN
.|$.
```

Figure 14: Synsets of a German sentence from the same corpus as before. Notice that prepositions, articles and punctuation signs, among others, have been assigned their respective morphological features, since they did not have a defined synset. Features starting with `bn:` are BabelNet synsets. Most importantly, we can see that the word `bank`, which in German is polysemous as in English (the financial institution or the bench for sitting), is mapped to the synset `bn:00008364n`. If we search this synset in BabelNet[25], we can see that it is the one corresponding to financial institutions, as it should, since the sentence is talking about the chairman of a Nigerian bank.

[25]BabelNet: bn:00008364n: `https://babelnet.org/synset?word=bn:00008364n`

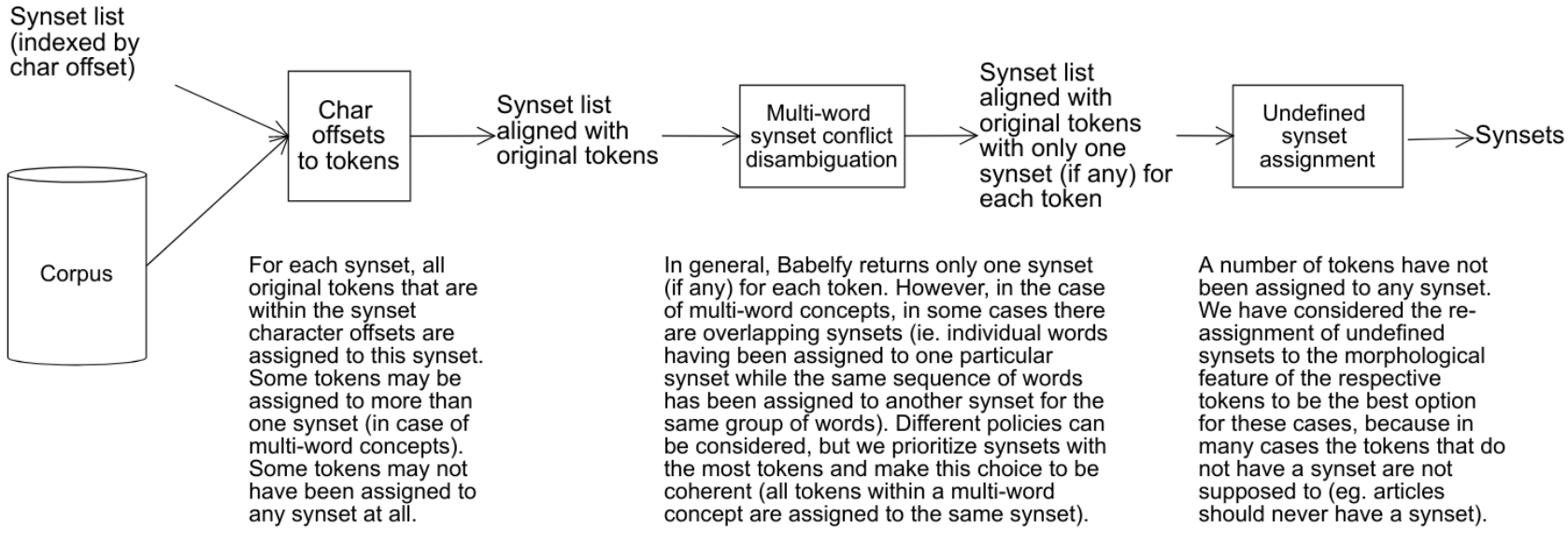

Figure 15: Synset alignment and assignment process.

8.3 Features and subword encoding

As we have seen, some form of subword encoding is required for achieving state of the art results in most MT tasks. However, linguistic features are usually word-level features. The same holds for BabelNet synsets, which in fact can even refer to a set of words instead of only one. In addition, if the number of word tokens and the number of token features did not agree, in most configurations we would probably end up with a dimension disagreement and we would not even be able to input that information to the neural network.

Therefore, we must align the tokens corresponding to the features to the tokens corresponding to subwords. That is to say, we must have the same number of subword tokens than word features, and the n-th feature must refer to the corresponding word of the n-subword. Also, we can take into account that the fact that one word feature belongs to, say, an intermediate subword, can have a slightly different meaning than the same feature belonging to the final subword. That is to say, the same way BPE adds the tag '@@' to the subword tokens in order to encode where the original word started, we could find an equivalent way of encoding this information in the feature side. We can consider different strategies for doing so or at least tackling this difficulty:

- Not using BPE: We discarded this option because the baseline would be weaker. We provide an example of this approach in figure 16.

```
word1 word2 word3... wordN
feat1 feat2 feat3... featN

und|CONJ diese|DET einfachen|ADJ themen|VERB sind|AUX eigentlich|ADV
keine|DET komplexen|ADJ wissenschaftlichen|ADJ zusammenhänge|ADJ ,|PUNCT
sondern|CONJ tatsachen|VERB ,|PUNCT die|PRON wir|PRON alle|PRON gut|ADJ
kennen|VERB .|PUNCT
```

  Figure 16: Feature alignment of a sentence without BPE (ie. without splitting words into subwords). For instance, a tokenized German sentence and the corresponding Part-of-Speech tags from the training set of the IWSLT 14 corpus, tagged by the Stanford NLP model.

- Just repeating the word features for each subword: This approach has the benefit of being simple, trivially compatible with the proposed architectures and not increasing the vocabulary size of the features. However, it does not encode the fact that consecutive tags are part of the same word. For an example, see figure 17.

```
subword1.1 subword1.2 ... subword1.L1 subword2.1 ... subwordN.LN
feat1 feat1 ... feat1 feat2 ... feat2 ... featN

und|CONJ diese|DET einfachen|ADJ themen|VERB sind|AUX eigentlich|ADV
keine|DET komplexen|ADJ wissenschaftlichen|ADJ zusammen@@|ADJ hän@@|ADJ
ge|ADJ ,|PUNCT sondern|CONJ tat@@|VERB sachen|VERB ,|PUNCT die|PRON
wir|PRON alle|PRON gut|ADJ kennen|VERB .|PUNCT
```

Figure 17: Feature alignment with BPE. Each feature is repeated for each subword of its corresponding word. For instance, the same sentence as figure 16. Notice that not all words are split into subwords, but `zusammenhänge` becomes `zusammen@@ hän@@ ge`, and therefore its Part-of-Speech, `ADJ` (adjective), is repeated for the three subwords.

- Using '@@' in word features, the same was that this tag is used in BPE: Again, this approach has the benefit of being trivially compatible with architectures such as the multi-encoder architecture and the single-encoder architecture with summation instead of concatenation, and not requiring an additional feature, but it has the drawback that it approximately doubles the vocabulary. The latter happens because the same feature can appear in different positions of the word. Therefore, there will be one token with '@@' and one token without this tag. For an example of this approach, see figure 18.

```
subword1.1 subword1.2 ... subword1.L1 subword2.1 ... subwordN.LN
feat1@@ feat1@@ ... feat1 feat2@@ ... feat2 ... featN

und|CONJ diese|DET einfachen|ADJ themen|VERB sind|AUX eigentlich|ADV
keine|DET komplexen|ADJ wissenschaftlichen|ADJ zusammen@@|ADJ@@ hän@@|ADJ@@
ge|ADJ ,|PUNCT sondern|CONJ tat@@|VERB@@ sachen|VERB ,|PUNCT die|PRON wir|PRON
alle|PRON gut|ADJ kennen|VERB .|PUNCT
```

Figure 18: Feature alignment with BPE and '@@' tags in the feature side as well. Each feature, apart from being repeated for each subword of its corresponding word, is concatenated with '@@'. In the example, *zusammenhänge* becomes `zusammen@@ hän@@ ge`, and therefore its Part-of-Speech, `ADJ` (adjective), becomes `ADJ@@ ADJ@@ ADJ`.

- RNN-EncoderFeat approach: Repeating the word features for each subword and introducing a new factor, subword tags, to encode the position of the subword in the original word. The 4 possible tags are:
  - B: Beginning of word.
  - I: Intermediate subword.
  - E: End of word.
  - O: Whole word (not split into subwords).

  This approach has the benefit of both encoding the position of the word which the subword belongs to and not increasing the vocabulary of the features. However, it is not compatible with the multi-encoder architecture, at least not trivially, because it would not make sense to have a specific encoder just for learning the 4 tokens of the vocabulary of subword tags.

However, it could be used with a modified multi-encoder architecture by the means of an hybrid approach (each feature encoder having its own subword tags as an additional feature). We provide an example of this encoding in figure 19.

```
subword1.1 subword1.2 ... subword1.L1 subword2.1 ... subwordN.LN
feat1 feat1 ... feat1 feat2 ... feat2 ... featN
B I ... E B ... E ... E

und|CONJ|O diese|DET|O einfachen|ADJ|O themen|VERB|O sind|AUX|O
eigentlich|ADV|O keine|DET|O komplexen|ADJ|O wissenschaftlichen|ADJ|O
zusammen@@|ADJ|B hän@@|ADJ|I ge|ADJ|E ,|PUNCT|O sondern|CONJ|O
tat@@|VERB@@|B sachen|VERB|E ,|PUNCT|O die|PRON|O wir|PRON|O alle|PRON|O
gut|ADJ|O kennen|VERB|O .|PUNCT|O
```

Figure 19: Feature alignment with BPE and subword tags (the third feature). Apart from repeating each subword. In the example, *zusammenhänge* becomes `zusammen@@ hän@@ ge`, and therefore its Part-of-Speech, `ADJ` (adjective), becomes `ADJ ADJ ADJ`, while the corresponding subword tags are `B I E`, for denoting the beggining, the middle and the end of the word, respectively. Most words are marked by 'O', which means that they were not split into subwords.

Each approach presents its own advantages and disadvantages and it is unclear which one should be followed. The procedure of choice should be the most practical one (eg. the one that it is compatible with our architecture, or the simplest one) or the one that performs better experimentally. In the following page, in figure 20, we can see the simplest case of the algorithm for aligning a text processed with BPE with the respective features.

```
def align_bpe(text_bpe, tags):
    aligned_tags = ''
    subword_tags = ''
    for (index_line, (line_bpe, line_tags)) in enumerate(zip( \
    text_bpe.splitlines(), tags.splitlines())):
        bpe_tokens = line_bpe.split()
        tag_tokens = line_tags.split()
        bpe_index = 0
        tag_index = 0
        aligned_tags_line = ''
        subword_tags_line = ''
        while bpe_index < len(bpe_tokens):
            if '@@' in bpe_tokens[bpe_index]:
                first_subword = True
                while '@@' in bpe_tokens[bpe_index]:
                    aligned_tags_line += tag_tokens[tag_index] + ' '
                    if first_subword:
                        subword_tags_line += 'B '
                        first_subword = False
                    else:
                        subword_tags_line += 'I '
                    bpe_index += 1
                    if '@@' not in bpe_tokens[bpe_index]:
                        aligned_tags_line += tag_tokens[tag_index] + ' '
                        subword_tags_line += 'E '
                        bpe_index += 1
                        tag_index += 1
                        break
            else:
                aligned_tags_line += tag_tokens[tag_index] + ' '
                subword_tags_line += 'O '
                bpe_index += 1
                tag_index += 1
        aligned_tags += aligned_tags_line + '\n'
        subword_tags += subword_tags_line + '\n'
    return aligned_tags, subword_tags
```

Figure 20: Algorithm for feature alignment with BPE (in Python): `text_bpe` is the BPE text and `tags` is the feature text (the one we want to align). '@@' is the BPE symbol. Notice that in this case we are not adding '@@' to the aligned features, but emitting subwords tags. We are assuming that features are already aligned with the original text, which is not always the case (eg. in the case of BabelNet synsets, we must align synsets with the original words by grouping by the character offsets). For the full implementation of this script and the rest of the code, see the repository that is attached to the delivery of this document.

9 Experimental framework

In this section, we will outline the particular details of the experiments needed to conduct in order to investigate whether the proposed architectures outperform the baseline and, if so, to what extent and in which circumstances, particularly when using BabelNet synsets.

9.1 Methodological considerations

In the first part, we already said that we would be following classical machine learning research practices. However, we must remark some additional considerations. Recall that it is considered to be out of the scope of this project to optimize the hyperparameters. Instead, we will base all our models in the configuration of the baseline taken as reference. For cleaning and splitting the datasets, we will use exactly the same scripts as the authors of the baseline reference, in order to make the comparison fair.

Instead of having the traditional train, validation and test split, we will have an additional set, the *devtest*, which can be considered as a second validation set that we will use for selecting our best models for each dataset. In particular, we will select the one with highest BLEU score in the devtest set. In case of not significant difference[26], we will select the simplest model (ie. the one with less parameters, which could be the baseline). The validation set itself will be mainly used for monitoring the training procedure and selecting the best epoch. In the case of the dataset that we will use, the split in three subsets is done by the script provided by the authors of the baseline in a reproducible manner, and for building an additional, untouched set for testing, we will take the test set of the evaluation campaign of the same task but from another year.

For reporting a honest generalization error of the selected model of each dataset, we will use the test set, which will not have been used at any point for building or selecting the best model.

For all the experiments we will be using the default random seed[27], so the results will be reproducible.

9.2 Dataset: IWSLT 14 DE-EN

The corpus we are keen on for our experiments should fulfill the following requirements:

- Low-resource tasks: It should be considered a low-resource dataset, at least as far as parallel data are concerned, for two reasons. The main reason is that, following the state of the art works we saw, we have the intuition that our architectures and BabelNet synsets should be particularly useful for low-resource tasks. Moreover, the dataset will have to be tagged for different features, Babelfy has hard limits on its usage, and many executions will have

[26] In NMT, most papers do not provide a statistical analysis. Instead, improvements in BLEU, even if in the first decimal (eg. 0.5), are reported. In our case, for assuring that the model BLEU with the highest BLEU score is actually better than the baseline, we will perform a statistical analysis for the selected model against the baseline. In particular, we will use a statistical test for MT based on BLEU and paired bootstrapping (in statistics, by bootstrapping we mean random sampling with replacement) as proposed in [45] and implemented in `https://github.com/pytorch/translate/blob/master/pytorch_translate/bleu_significance.py`.

[27] In Fairseq, the default seed is `1`.

to be executed for both debugging and evaluation, while this project has a tight schedule and the computational resources are not infinite.

- Reproducibility: The baseline should be easy to reproduce and to compare with, ideally with Fairseq, the framework we are basing our implementation on.

For these reasons, IWSLT 14 German-English, a well-known corpus, was used for both development (and preliminary experimentation) and evaluation. The International Workshop on Spoken Language Translation (IWSLT) is a yearly workshop with different open tasks related to MT that is very popular in the NLP community[28]. The IWSLT 14 German-English dataset[29] consists of translated TED talks[30] subtitles. As data samples, consider revisiting the example sentences used in the previous section, as in figure 8.

The script suggested by the authors of the baseline builds the sets as follows:

- Train: $\frac{22}{23}$ of the sentences from the train data provided for IWSLT 14.
- Validation: $\frac{1}{23}$ of the sentences from the train data provided for IWSLT 14.
- Devtest (referred as test in other publications): Built by merging the development and test sets of 2010, 2011 and 2012.

For building the test set, we took the 2013 test set from the corpus released for IWSLT 16[31]. In table 12, we provide the number of sentences for each set.

Set	Sentences
Training	160,239
Validation	7,283
Devtest	6,750
Test	993

Table 12: Subsets and number of sentences in IWSLT 14 German-English. The devtest set is referred as the test set in other publications, but our test itself will be the forth subset.

Notice that even though the number of sentences is small, the task is considered to be easy, because German and English are relatively close and all the sentences belong to the same domain of TED talks. Therefore, the baseline is strong and difficult to beat, so we do not expect our proposed architectures to obtain much higher scores. Instead, we are expecting a minor but still noticeable improvement.

[28] International Workshop on Spoken Language Translation: `iwslt.org`.

[29] International Workshop on Spoken Language Translation: Proceedings: `http://workshop2014.iwslt.org/downloads/proceeding.pdf`.

[30] TED talks: `https://www.ted.com/`.

[31] In particular, we used `IWSLT16.TED.tst2013.de-en`, which can be downloaded from `https://wit3.fbk.eu/archive/2016-01/texts/de/en/de-en.tgz`.

9.3 Preprocessing and tagging

We are using the well-known data cleaning and preparation (including tokenization and reproducible splitting) script for this dataset[32]. Then, a joined BPE (ie. German and English share subwords) of 32,000 operations is learned from the training data, with a threshold of 50 occurrences for the vocabulary.

As far as the features are concerned, the corpus was tagged with features and BabelNet synsets as detailed in section 8. Once they had been aligned as described before, no additional considerations had to be taken into account, except for the vocabulary size. Lemmas were pruned to a vocabulary size of 10k, because their original vocabulary size was too big, while for synsets we experimented both with and without pruning. In the latter case, we used threshold of 30 occurrences, because apart from having a big vocabulary, there were many synsets that did not appear frequently enough). This threshold of 30 allowed for a 95% coverage of in the case of IWSLT 14. The rest of the vocabularies were not pruned, because the vocabulary sizes of features such as PoS was already too small and were always defined for all the cases.

9.4 Baseline configuration

We will use the vanilla Transformer as the baseline architecture. For choosing its hyperparameters, we will take the ones used in the reference that we take as a baseline.

Although the original evaluation campaign of the task was due in 2014, better models have been released ever since. We used the Transformer architecture [25] with the hyperparameters proposed by the Fairseq authors[33]. In particular, 6 layers in encoder and decoder, 4 attention heads, embedding size of 512 and 1024 for the feedforward expansion size, dropout of 0.3 and a total batch size of 4000 tokens, using label smoothing[34] of 0.1.

9.5 Implementation details

As we have said, the architectures have been implemented on the top of Fairseq. Fairseq is the framework used by the MT group at the TALP Research Center. It was chosen because, thanks to being backed by an AI powerhouse such as Facebook, it is frequently updated. It is very flexible and has many state of the art neural components already implemented as well. The authors of RNN-EncoderFeat implemented their proposal in their own toolkit, Nematus, which we mentioned in 1.3.4, while the OpenNMT implementation allows specifying a similar configuration. However, Fairseq did not support factored NMT up to our work, and as we have said, we used the Transformer instead of an RNN-based architecture and we considered using multiple encoders as well.

Apart from that, different tagging and aligning scripts were necessary. For an implementation

[32] `prepare-iwslt14.sh`: `https://github.com/NVIDIA/DeepLearningExamples/blob/master/PyTorch/Translation/Transformer/examples/translation/prepare-iwslt14.sh`.

[33] `https://github.com/pytorch/fairseq/tree/master/examples/translation`

[34] Label smoothing consists in decreasing the probabilities of the ground truth (eg. instead of having a probability of 1, we can make the ground truth to have a probability of 0.9) and it tends to work very well in practice as a regularizer. For a recent study on this matter, see [46].

reference, as well as references to the particular scripts needed to run the experiments, see appendix A. Nevertheless, we will highlight some matters related to the implementation. The code repository is available in the same delivery as this document, and will be open-sourced in Github[35].

One of the most time-consuming aspects of this project was developing pre-processing scripts for tagging and aligning the features. With regard to tagging, even if the tagger is already built, it is required to interface with it. Aligning is a crucial aspect and even a small, unnoticed dis-alignment of the features can make the model to perform very poorly, as it happened with the first experiments with lemmas, so had we to repeat them. Aligning to BPE can be mostly trivial, but the problem is that the tokenization of the taggers may not be exactly the same than the one of the words. For instance, Babelfy performs its own tokenization without giving much details about it, so we had to align synsets by taking characters into account, as detailed before.

On the other hand, as far as the multiple-encoder architecture is concerned, we developed a generic factored encoder, which could be used with non-Transformer architectures as well (although we did not).

In general, we tried to make the code as configurable and parameterized as possible. However, since this is a research project and we had to iterate fast for experimenting with different approaches, the code should not be considered ready for production.

The Fairseq source code sections that were involved in our modifications were the following:

- Parsing, datasets and tasks: Instead of loading the language pairs (source and target), N source datasets had to be loaded. Fortunately, Fairseq already had a multi-language task, but it was intended for alternatively training different language pairs, not for simultaneously using multiple source sequences. It had to be adapted but it was a solid starting point. Instead of separating features by `'|'` in the same file as in Nematus or OpenNMT, we load the features from different files.
- Forward pass: The forward pass of the models had to be modified in order to compute and combine the different outputs of the factored encoder or embedding layer.
- Training and validation steps: The batches coming from the different source sequences had to be combined into a single batch, and then input to the forward pass of the models.
- Sequence generation and generation steps: The same as before had to be done for the generation (ie. testing).
- Parameter storing: At each checkpoint (eg. at each epoch), Fairseq stores the weights of the model in disk. When evaluating the model, the weights are loaded from the corresponding checkpoint. However, Fairseq did not saved by default the extra parameters added by some of our models, and we had to explicitly store them.

9.6 Planned experiments

We are going to conduct experiments with different combinations:

[35]The repository will be released in the coming weeks in `https://github.com/jordiae/fairseq-baseline-factored`

- Architectures: Both the single-encoder and multiple-encoder architectures.
- Combination strategies: Both summation and concatenation.
- Different embedding sizes.
- Features: Synsets and the linguistic features, having in mind that in the RNN-EncoderFeat approach lemmas were the most useful feature.

Unfortunately, for computational and time restrains, it will not be possible to experiment with all the combinations. Instead, in some cases we will have to cut down the number of experiments by heuristically discarding some configurations.

10 Results

In this section, we will outline the results of evaluating the proposed architectures with different features, specially BabelNet synsets.

10.1 Baseline reproduction

We started by reproducing the baseline described in 9.4 as shown in table 13.

Set	Loss
Training	3.29
Validation	3.84

Set	BLEU	W.A. BLEU
Devtest	34.08	34.43

Table 13: Results of the baseline architecture in IWSLT 14 German-English, where W.A. means weight averaging of the 10 latest epochs.

It trained for 43 epochs achieving the lowest validation set in the 40th epoch, which suggests that the training converged. The evolution of the training and validation losses are depicted in figure 21. The best validation perplexity[36] was as low as 5.02, which is a good sign as well.

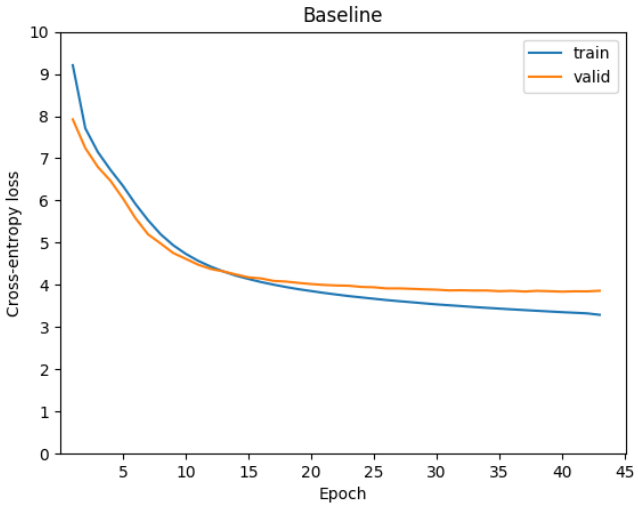

Figure 21: Baseline training: The label-smoothed cross-entropy loss of the validation set showed that at epoch 43, when the training stopped, the model had already arrived to an optima. The training loss was still improving, but by overfitting.

[36] In information theory, perplexity is a measure related to entropy [47]: $\texttt{perplexity} = 2^H$. In NLP it is used to evaluate language models (see `http://www1.icsi.berkeley.edu/Speech/docs/HTKBook3.2/node188_mn.html`). In NMT, although there are other metrics (like BLEU), a low perplexity is a good sign.

10.2 Preliminary experiments with PoS

We used PoS for preliminary experiments. The first experiments of the new architectures consisted in running the factored architectures (both signle-encoder and multiple-encoder) with concatenation and PoS, with the default hyperparameters (the ones used in the baseline). The results obtained are summarized in table 14.

Embeddings		Arch.	Comb.	Feat. BPE	W.F.	Val. loss	Devtest W.A. BLEU
Word	PoS						
512	512	Mult.	Concat	No	No	5.33	12.60
512	32	Mult.	Concat	Yes	No	3.86	34.38
512	32	Mult.	Concat	Yes	PoS (7)	3.86	34.26
512	32	Sing.	Concat	Yes	No	3.87	**34.48**

Table 14: Experiments with PoS in IWSLT 14 German-English, where *Arch.* means architecture, *Comb.* means combination strategy, *Feat. BPE* means whether the BPE tag was added to mark subwords in the features and *W.F.* means whether some weights were frozen early in the training procedure in order to counter overfitting, and if so at which epoch. In this case, the BLEU score is after performing weight averaging.

All of the models seemed to have converged by looking at the evolution of the losses. Although it was not computationally feasible to perform a full ablation study, these preliminary experiments were useful for getting some insights. The first execution in table 14 clearly performed poorly compared to the baseline, most probably due to overfitting of the PoS encoder (figure 22). For countering this problem, in the following 2 experiments, we tried both decreasing the embedding size and freezing the PoS weights at an early epoch in training (figures 23 and 24). For deciding at which epoch should we freeze the weights of the PoS encoder, we observed at which epoch the new architecture started to diverge from the baseline loss. Decreasing the embedding size clearly improved the results, while weight freezing proved to be useless. Nevertheless, the results did not outperform the baseline (actually, the BLEU scores were slightly worse).

Later on, we tested a similar configuration but with the single-architecture, and we achieved a slight improvement with respect to the baseline (figure 25). However, a BLEU improvement of 0.05 could well be due to random aspects such as weight initialization. The neural network could have just mostly ignored the PoS tags.

As far as using the BPE tag in features is concerned, as detailed before, the motivation was to encode the position of the subwords in the feature side. In addition, in the case of a feature with low vocabulary size as PoS, increasing the vocabulary size could help to prevent overfitting. However, training with a PoS embedding dimension of 32 and without the BPE tags in the features showed validation losses very similar to the ones presented in the table, so we did not consider to be worth evaluating the devtest set (which is resource-intensive as well) with this configuration, because it did not seem to be a key factor. The PoS vocabulary sizes with and without the BPE tags were as low as 64 and 96, respectively.

Instead, the key difference was the embedding dimension of PoS. The multi-encoder architecture did not seem to be worth it, because its results were slightly worse than the single-encoder model, while having far more parameters.

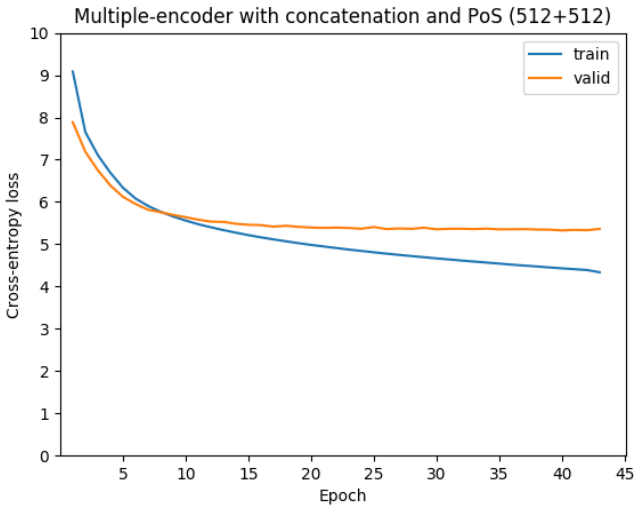

Figure 22: Multiple-encoder with concatenation and PoS (512 + 512) training: There is a clear overfitting pattern. At an early epoch (8), the training loss starts to markedly diverge from the validation loss, and validation loss gets stuck at a much higher value than the baseline. Although in this case losses are not directly comparable, it is a very bad sign. Having an encoder with an embedding dimension of 512 for a feature with a vocabulary of a few dozens of tokens, like PoS, and having a decoder with embedding size of 1024 (the one required if we concatenate two encoders with an embedding size of 512) leads to overfitting and poor performance.

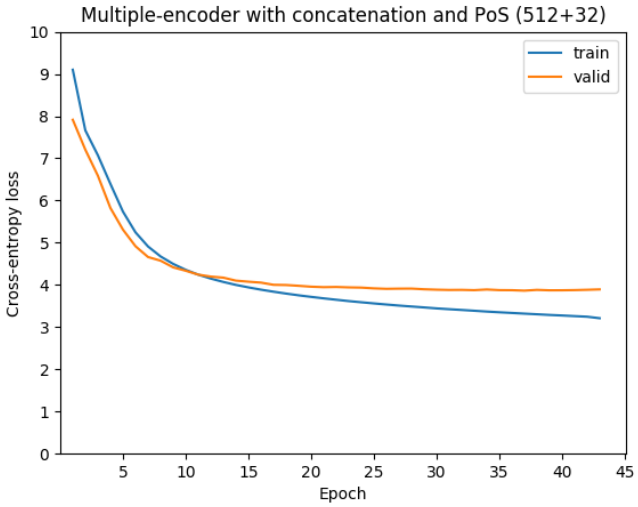

Figure 23: Multiple-encoder with concatenation and PoS (512 + 32) training: Decreasing the embedding size of PoS clearly improves the results seen in figure 22 by preventing overfitting, but the resulting BLEU is slightly worse than the baseline and it could be the case that the model just learnt to ignore PoS tags.

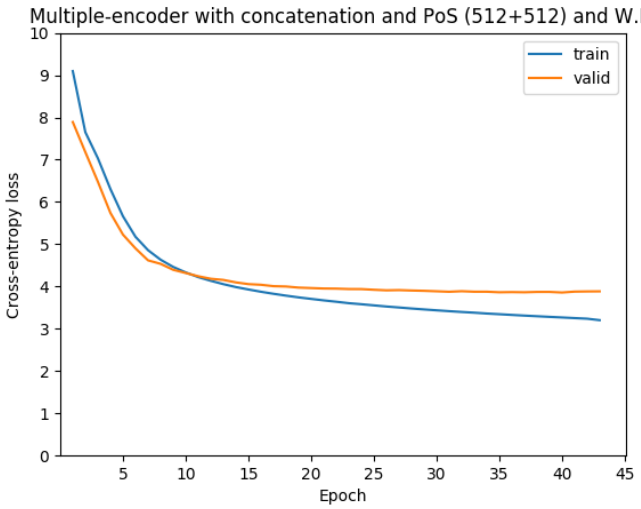

Figure 24: Multiple-encoder with concatenation and PoS (512 + 32) and weight freezing training: In addition to decrease the PoS embedding size as in figure 23, we experimented with freezing the weights of the PoS encoder at an early epoch (7, when the baseline seemed to start outperforming) while still letting the word encoder and the decoder to continue updating its parameters, with the hope of further preventing overfitting. However, it almost did not have any visible effect. Actually, the resulting BLEU score was lower than in the previous experiment.

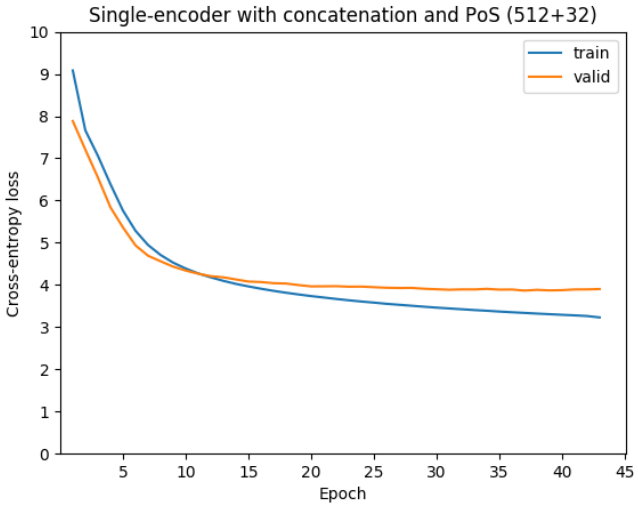

Figure 25: Single-encoder with concatenation and PoS (512 + 32): Finally, we switched to the single-encoder architecture. The losses looked similar to the ones with two encoders as in figure 24, while the model was much simpler. In addition, in this case the model outperformed the BLEU score of the baseline by 0.05 points, which is marginal and probably due to weight initialization. However, it was the first time that am architecture of ours was not surpassed by the baseline, although it could still be the case that the model was learning to ignore PoS.

10.3 Synsets

Since BabelNet synsets are one of the main novelties in this project, we devoted more experiments to testing configurations with these features, partially inspired by the results obtained with PoS.

Notice that while for PoS we computed the BLEU score after performing weight averaging, in this case we did not do so, because storing the checkpoints for the last 10 epochs would have had high storage costs. Instead, from now on, since we have the BLEU score of the baseline both with and without weight averaging and weight averaging is expected to show a slight improvement, the idea is to first compare the results without weight averaging, and then, if a model seems promising, re-train it storing the last 10 epochs and then perform weight averaging for the final model.

Moreover, although the loss and the BLEU score of a model are related, the former is much more reliable for assessing the models. Nevertheless, for computational restrains, in this case, if we observed that the validation loss was too far from the baseline or the other ones, we did not evaluate and compute the BLEU for saving resources. In particular, we did not do so if the validation loss was greater than 3.95. Again, we considered the hyperparameters used in the baseline as the default configuration. The results are summarized in table 15.

Embeddings		Arch.	Comb.	Vocabulary	Valid loss	Devtest BLEU
Word	Synsets					
512	512	Multiple	Concat	BPE tag: 62k	4.3	-
512	512	Single	Concat	BPE tag: 62k	6.7	-
512	128	Multiple	Concat	BPE tag: 62k	3.95	33.25
512	128	Single	Concat	BPE tag: 62k	3.86	33.86
512	512	Multiple	Sum	BPE tag: 62k	4.51	-
512	512	Single	Sum	BPE tag: 62k	3.91	33.66
512	128 + sub (4)	Single	Concat	NO BPE tag: 32k	3.90	**34.13**
512	128 + sub (4)	Single	Concat	Freq. thres.: 6.6k	3.89	34.07
512	256 + sub (4)	Single	Concat	NO BPE tag: 32k	3.98	-

Table 15: Experiments with synsets in IWSLT 14 German-English, where *Arch.* means architecture, *Comb.* means combination strategy, and *BPE tag* means whether the BPE tag was added to denote subwords in synsets. By *sub* we mean whether we used an additional factor, subword tags, and if so with which embedding size, while by *freq thres.* we mean whether we applied a vocabulary frequency threshold (ie. synsets with less than the threshold of occurrences were pruned), in this case 30 occurrences. In order to save resources we did not evaluate the devtest set if the validation loss was greater than 3.95.

All models seemed to have converged, except the one with the multiple-encoder architecture and summation, because at the last epoch it was still improving the validation loss. For this reason, we allowed 10k additional weight updates for this model.

For brevity, we have omitted some of the executions that were too far from the baseline. The configuration indicated in bold in table 15 outperformed the baseline (recall that in this case we are comparing with the baseline result without weight averaging, which obtained 34.08). However, as in the case of PoS, an increase of 0.05 BLEU could be due to the stochastic nature

of neural networks training procedures. Moreover, synsets have a much larger vocabulary size and imply more parameters (larger encoders and embedding layers), so in this case a 0.05 increase is even less meaningful than in the case of PoS.

Since we did not observe gains, while synsets where adding a considerable amount of complexity into the model, we conduct an additional experiment for checking the actual usefulness of the synsets. In particular, we tested the baseline configuration, but instead of inputting the original tokens, we used synsets. Instead of morphological features, this time we assigned the original tokens in case that no synsets were available, with the hope that this acted like a word sense disambiguation pre-processing step. The vocabulary consisted of about 36k tokens. However, we obtained a validation loss of 4.13 and a devtest BLEU of 30.83, so we did not further explore this option. At least with this dataset, the configurations we tried and the preprocessing steps we performed, BabelNet synsets did not prove to be that useful as we expected.

As far as the architectures themselves are concerned, having one specific encoder for the synsets did not prove to be worth the additional complexity, as in the case of PoS. Not only did they have more parameters, but they also obtained worse results in general. Using subword tags seems to be better than introducing the BPE tag in the features (because otherwise the vocabulary size is too big). However, as we explained in section 8 it does not make sense to use this encoding with architectures other than the singe-encoder architecture with concatenation. Furthermore, in the case of the model that obtained the increase, since it was one of the configurations that implied smaller embeddings, it could have happened that the neural network learned to ignore synsets or that it just used the morphological features that were used when synsets were undefined.

In the last section we will revisit and discuss this matter. In the following page, in figure 26 we can observe the losses obtained in the trained procedures with synsets.

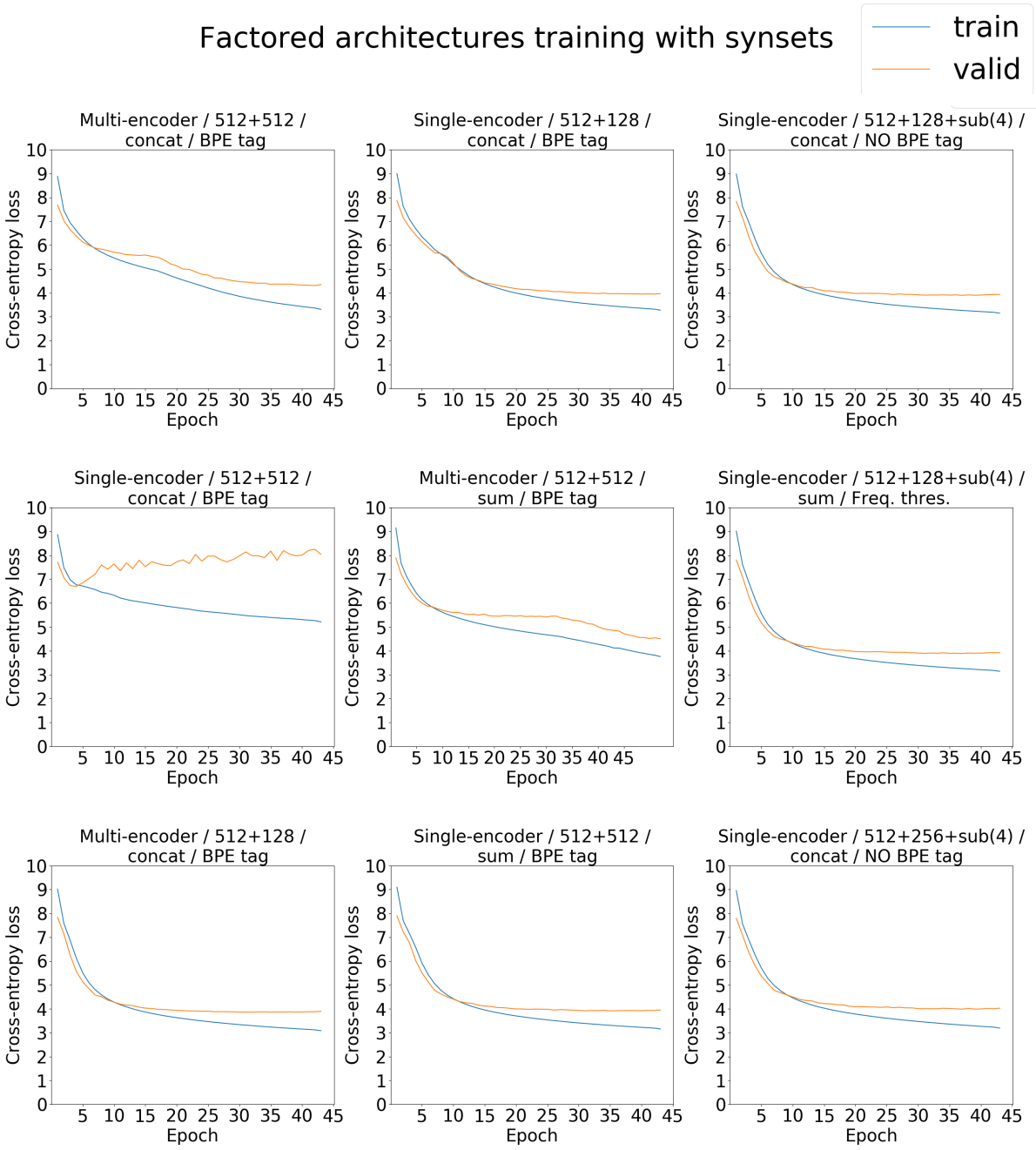

Figure 26: Experiments with the factored architectures with synsets: This time, apart from overfitting, we observe some irregular validation curves in some cases. Some architectures trained well but anyway only one architecture (marginally) outperformed the baseline.

10.4 Classical linguistic features in the Transformer

At this point, since we had not been able to improve the baseline (at least significantly), we considered to reproduce the RNN-EncoderFeat configuration but with one important modification: using the Transformer instead of an RNN-based architecture. This way, we will see if there is any inherent characteristic of the Transformer that makes word features useless, or whether the baseline was too strong to be beaten. That is to say, we run the single-encoder architecture with concatenation and lemmas, PoS, dependency tags and morphological features with the same embedding dimensions as in the RNN-EncoderFeat work, except that for the Transformer's restrains, the total embedding size was 512, not 500, assigning the reaming 12 dimensions to the word embeddings (not the features). Thus, the embedding sizes will be the ones shown in 16.

Factor	Embedding dimension
Words	362
Part-Of-Speech	10
Lemmas	115
Word dependencies	10
Morphological features	10
Subword tags	5
Total	512

Table 16: Embedding dimensions of the RNN-EncoderFeat approach adapted to the Transformer: The only difference is that in this case the word embeddings have a dimension of 362 instead of 350, in order to have the same total embedding size as the baseline.

Again, the baseline parameters will be left as the default ones. It is important to recall from previous sections that the Transformer is thought to need larger embedding sizes to learn appropriately, and embeddings larger than 512 are not usually considered for words. However, it was the most equivalent configuration we could find for trying the RNN-EncoderFeat approach but with the Transformer, and we considered it worth a try. Notice that some of the feature embeddings might be too small for the Transformer as well.

This time, in order to align the features, taken from the Stanford tagger, it was more efficient to use its own tokenizer, which was not exactly the one used in the baseline. For assuring that this would not be the cause of the improvement (or the worsening), we rerun the BPE preprocessing and the baseline configuration with the new tokenization and obtained a devtest BLEU score (without weight averaging) of 34.05, virtually identical to the original baseline (34.08).

The results are summarized in table 17.

Set	Loss
Training	3.24
Validation	3.89

Set	BLEU
Devtest	**34.21**

Table 17: Results of the RNN-EncoderFeat approach but with the Transformer in IWSLT 14 German-English.

As we can see, for the first time we obtained a noticeable, yet very modest, improvement, of 0.16 BLEU points, even if the embedding size for words was smaller than the baseline, which seems to be evidence that at least one of the features is contributing significantly. In addition, it seems to prove that at least the single-architecture with concatenation is working as expected, because otherwise the baseline would not have been outperformed with a word embedding size 200 dimensions smaller than the one from the baseline.

10.5 Lemmas

Finally, we run both architectures with both concatenation and summation and lemmas as the only feature. We had two reasons for doing so:

- In RNN-EncoderFeat, the features that gave the largest improvement, by far, were the lemmas.
- As we have said, the Transformer is thought to need larger embedding sizes, and in this case we are sacrificing embedding dimensions of the supposedly most important factors, namely words and lemmas, for other features that might be less useful.

We used an embedding size of 512 for each feature. In the case of the single-encoder architecture, we used subword tags (4) as well. We summarize the obtained results in table 18.

Architecture	Combination	Valid loss	Devtest BLEU
Single	Concat	4.38	27.10
Single	Sum	3.89	**34.35**
Multiple	Concat	4.00	33.58
Multiple	Sum	5.45	9.71

Table 18: Results of the different factored architectures with lemmas in IWSLT 14 German-English. This time, with the insights from previous experiments with other features, we did not experiment with the embedding sizes or different ways to encode the position of subwords with regard to the features.

With the single-encoder architecture and summation we obtained an improvement of about 0.3 points, which is close to a 1% increase. This configuration had less features than in the previous

experiment. In addition, the single-encoder architectures are simpler, and the summation implies less parameters in the decoder (if using the same embedding sizes as in the baseline), so in this case the simplest architecture (apart from the baseline) obtained the best results as well.

From these experiments, we can make some additional observations. First of all, the architectures seem to be working, because the results are coherent. However, the multiple-encoder architecture does not seem to be a good idea, at least for dealing with features.

The multiple-encoder architecture seems to work better with concatenation, while the single-encoder architecture performs better with summation. The reason why this might be happening is that having a different encoder for each feature might mean that a completely disentangled representation for each feature is being learned. The outputs from both encoders are in very different spaces, and when combined with summation, it is very difficult for the decoder to interpret the vectors. Instead, if their outputs are concatenated, at least it can learn to ignore half of the concatenated vector, which is probably what is happening. In the case of the single-encoder architecture, the summation gives a much more compact representation. Summing lemmas implies a simple, linear transformation that allows the decoder layers to have a dimension of 512 (instead of doubling that, which is probably resulting in overfitting).

10.6 Final model

We performed weight averaging (again, of the 10 last epochs) of our candidate (single-encoder with summation and lemmas) and evaluated the resulting model against the devtest set, obtaining BLEU = **34.71**, still higher (and with a very similar proportion) than its baseline counterpart (recall the baseline model with weight averaging obtained **34.43**), so our model was almost 0.3 points better.

For assuring that the difference was statistically significant, we performed a statistical test (the paired bootstrapping test for MT we saw in section 9.1) against its baseline counterpart, still with the devtest set. The output of the script is provided in figure 27.

```
Baseline BLEU: 33.27
New BLEU: 33.75
BLEU delta: 0.48
Baseline better confidence: 0.00%
New better confidence: 100.00%
```

Figure 27: BLEU significance output for IWSLT 14 DE-EN.

The statistical test confirmed that the new model is better than the baseline. As we can see, at the beginning it prints the BLEU scores of both models. They are slightly lower than the actual BLEU computed by Fairseq because in this script they are computed from a bootstrapped sample. Although the confidence might seem too high at a first glance, it is worth noting two considerations:

- Even if training neural networks is a highly stochastic process, at least in this task and the Transformer we used we did not observe high variability due to weight initialization. For instance, repeating the baseline with a slightly different tokenization gave a difference of only 0.03 BLEU points.
- Although the improvement is minor, it seems to be the case that it is consistent across the set. That is, the difference does not come from only a tiny group of sentences in which the new model outperformed the baseline, but from minor improvements along the whole set, because otherwise a test based on bootstrapping would not have obtained high confidence.

Thus, our view is that this model performs better. Another issue is whether it will be worth using it, taking into account that lemmatization is required, and if so to what extent, which we will discuss later on.

Having selected this model as the final one, we evaluated the test set (untouched up to this point) for giving a final estimation of the generalization error of our model, and we obtained BLEU = **37.46**. Since this set is smaller than the other ones and the purpose of this set is not comparing with other models, we cannot extract many conclusions from it, but at least we think that this result is strong evidence that our model did not overfit to the devtest set.

10.7 Performance

In section 7.6, we observed some aspects to take into account with regard of the complexity of both the vanilla Transformer and our modified versions. This time, we will outline some of the results with regard to the actual efficiency of the models. Although our intention is not to perform a full analysis and execution times are not directly comparable because some of the nodes of the cluster are faster than the other ones, we will be able to extract some relevant insights.

Model	Execution time (in seconds)		Parameters
	Training	Inference	
Baseline	9,397.8	1,250.0	39,833,600
Selected model (single-enc/sum/lemmas)	9,431.4	1,249.0	45,277,184
Classical linguistic features	21,914.1	11,418.9 (CPU)	40,380,984
Multiple-enc/concat/lemmas	21,583.7	16,723.9 (CPU)	120,016,896
Multiple-enc/sum/lemmas	14,809.7	14,366.6 (CPU)	57,893,888
Single-enc/concat/lemmas	17,104.0	1,835.6	146,210,944

Table 19: Performance (execution time and number of parameters of some of the models): Although the execution times are not directly comparable because not all GPUs had the same exact capabilities, we still can extract some insights. The number of parameters can help to get an idea of the complexity of the models. Training refers to training time with the train set and inference refers to the evaluation time with the devtest set. With regard to inference, we were not always able to execute the models with GPU because of the high demand of the cluster.

Both the baseline and the selected model achieved an inference performance of about 5.5 sentences/s and 120 tokens/s, although we have to take into account that in IWSLT 14 sentences

are not particularly long (because it consists of a set of TED talks) and the Transformer precisely can be slower than RNN when sentences are too long, as we saw in 7.6.

The number of parameters of the selected model is not much higher than in the case of the baseline architecture because it just needs an additional embedding layer for the lemmas. Since words and lemmas are combined by concatenating, the embedding size of the decoder can still be 512 (instead of being doubled to 1024). It seems to be the case that the selected model is at least close to the efficiency of the original architecture. However, tagging and feature alignment have a non-negligible computational cost. If tagging can be considered a single-time preprocessing step, this fact could be mostly ignored, but if we were to use this model in production, for each call to the model we should have to tag the text, which could compromise real-time capabilities.

For illustrating the performance of other models, in table 19 we included some of the other configurations as well. Although the model with all the classical linguistic features has slightly more parameters than the baseline, it seems to be considerably slower. We suspect that the embedding layers are not being parallelized, contrary to our expectations. In general, the multiple-encoder architecture seems to be slower than the single-encoder architecture (in the case of the RNN-EncoderFeat approach adapted to the Transformer, it is not the case because as we said having many embedding layers seems to penalize the performance). Summing is far more efficient than concatenating, not because of the operation itself, but because summing allows not to double the embedding size of the decoder. One thing that might seem incoherent is the fact that the single-encoder architecture with concatenation has more parameters than its equivalent multiple-encoder version. However, this is explained because in the case of the single-encoder architecture we used subwords as well, which added an embedding layer and slightly increased the embedding dimension of both the encoder and the decoder.

11 Conclusions

General discussion We started from recent works that showed that using word features could improve NMT. Most of them used classical linguistic features, while one work suggested using concepts from a linked data database, BabelNet. BabelNet concept identifiers, known as synsets, offered the promise of giving language-agnostic, semantic information that could lead to better translations. We believed that this would particularly be the case for low-resource datasets, although we had the intuition that a baseline based on the Transformer architecture would be difficult to beat and we knew that the previous works using features had not achieved great improvements, even though they were still noticeable.

The goal of this project was to improve state of the art NMT by using synsets. In particular, for dealing with multiple sequences referring to the same source sentence, namely factors (the original words and a set of features referring to them), we have developed two architectures (each one with two combination variants, namely concatenation and summation) based on the Transformer, a non-recurrent architecture for dealing with sequences that apart from being generally more efficient than RNN-based systems in most of the cases achieves better translations as well.

The first one, the single-encoder architecture, is a more conventional approach and can be considered as the adaption of the already existing RNN-based architectures that modified the encoder in order to concatenate multiple embedding layers (one for each factor) to the Transformer. The second one, the multiple-encoder architecture, is a less conventional approach motivated by our intuition that features with large vocabulary sizes, as in the case of synsets, would benefit from having a specific encoder.

On the other hand, we have tagged the German words of IWSLT 14 DE-EN with both synsets and classical linguistic features. We expected the synsets to be more useful than classical features, but well-established features such as PoS have been useful for preliminary experiments and comparing their results with features obtained from BabelNet.

The process of tagging resulted more challenging than expected for a variety of reasons. Even though aligning features with the subwords of their respective words is relatively straightforward provided words (without having been split into subwords yet) and features are already aligned, this is not always the case if the tokenizer used by the tagger is not the same used for tokenizing words. Most taggers do not allow the user to specify whether to respect the original tokenization. Even a small error when aligning features with words could make the neural network unable to train properly, and this could be easily unnoticed, so it is very important to make sanity checks.

In the case of synsets, instead of using plain BabelNet id's, as in [35], we have used Babelfy, a word sense disambiguation API based on BabelNet that generally returns a single synset (if any) based on the context, instead of retrieving the list of all the possible synsets, which do not seem to be particularly useful. This way we have been able to obtain disambiguated synsets for polysemous words. We had to deal with Babelfy's usage limits, since it is a private API, and the fact that its tokenization is not necessarily compatible with ours and it does not return their tokens either. Instead, we have aligned synsets by making use of the character offsets. Although we specified that we were only keen on the top candidate synset for each word, we have found that Babelfy retrieves more than one synset per token in the case of multi-word synsets. In cases like "semantic network", Babelfy returns one synset for each word, individually, and then a third synset for the collective meaning. We have decided to prioritize multi-word concepts since it is

the most practical option.

Nevertheless, we have found that in the case of IWSLT 14 DE-EN train set, about 70% of the tokens do not have an assigned synset. This fact, particularly when combined with the large vocabulary size of synsets, is problematic, because the factor of synsets is a vector with many undefined values and the ones that are defined are still difficult to be learned because many synsets appear with low frequency. In order to counter this fact, and after having inspected the problem and considered different options, we have decided to assign morphological features whenever synsets were unavailable, because many words do not have a synset because they are never supposed to (eg. articles do not have associated concepts in BabelNet). In addition, the feature we have added to synsets has a small vocabulary size. We have experimented with pruning the synsets that are too infrequent as well.

When we formulated the problem, we asked ourselves whether synsets could improve a NMT system based on the Transformer. The experiments have shown that synsets are not more useful than classical linguistic features in IWSLT 14 DE-EN, and we are unsure whether they are useful at all in this dataset with the Transformer, because in the best case they achieve a marginal improvement that could well be due to the stochastic nature of the training procedure. Thus, we can answer that at least with our approaches we have not been able to improve the existing NMT system by using synsets.

Since synsets have not improved the results, we have decided to try the system proposed in [32] but with our factored Transformer (single-encoder architecture). By applying the single-encoder architecture with the same features as in this RNN-based work we have obtained an increase of 0.16 BLEU. An increase of 0.16 BLEU might seem to be possible due to the stochastic nature of neural networks training, but in our experience we have not observed high variability (eg. repeating the baseline with a slightly different tokenization obtained virtually the same BLEU score, a difference of just 0.03 points). Most importantly, in this case the word embedding size was as small as 362 (a low value, particularly for the Transformer) so the required information must be coming from the features. The baseline had a word embedding size of 512. For this reason, we believe that this result is very relevant, because it shows that the Transformer can take advantage of word features and that our implementation is working. Up to this point, as far as we know, the existing (at least, published) factored architectures were all based on RNNs like LSTMs.

Provided that in [32] most of the improvement obtained by adding linguistic features came from lemmas, we had the intuition that it would make sense to discard the rest of the features and devote bigger embedding sizes to both words and lemmas. We have experimented again with both the single-encoder and multiple-encoder architectures and with both summation and concatenation. We have found the single-architecture with summation to improve both the baseline and the previous experiment with all the linguistic features.

The final model, based on the single-architecture with summation and lemmas, has obtained a modest, yet statistically significant improvement, of about 0.3 in the devtest set, which is an increase slightly below 1%. The fact that the statistical test, based on bootstrapping, has showed that our model is better is evidence that the improvements are consistent along the text, not due to some great improvements in only a small subset of sentences.

The improvements obtained by our approach are generally smaller compared to those obtained by the ones cited in the previous work. The best result obtained by the work that used synsets

with RNN-based architectures (in [35]) was an increase slightly above 1%, while in one case the addition of synsets degraded the baseline. Increases obtained by the use of classical linguistic features in [31] and [29] were greater than ours, but not bigger than 1.5 BLEU in the best case. We believe that the fundamental reason for our improvement being smaller is the fact that our baseline was stronger because of the Transformer architecture, and therefore difficult to beat.

When evaluating the test set, which has been taken from the test set of the same task but another year and has remained untouched up to this point (ie. it was not used for selecting the best model), we have obtained a BLEU score about 3 points higher than the one obtained by the same model in the devtest set. We are cautious with this result because it is a small set. However, at least we believe that it is strong evidence that our selected model has not overfitted to the devtest set.

Crucially, the selected model happens to have a similar efficiency to the one of the baseline. It only requires and extra embedding layer for lemmas, and thanks to the fact that we are summing and not concatenating, the decoder can have the same embedding dimensions as the baseline. The execution time for both training and inference and the number of parameters are close to the ones of the baseline, although the execution times are not directly comparable because not all the GPUs of the cluster have the exact same capabilities.

Nevertheless, our model requires lemmatization and aligning lemmas to subwords, which would have to be integrated into the data pipeline of the model. Provided we factor in this time, if we were to deploy this model to production, perhaps the slight improvement would not be worth it, particularly depending on whether the system required real-time capabilities. Apart from that, the code is not meant for production, but for research and prototyping.

With regard to the proposed architectures, in general, the results show that the multi-encoder architecture was not a good idea, at least the way we have implemented and used it, because the obtained translations are worse than the ones obtained with the single-encoder architecture (and even the baseline), while being more complex. It seems to be more prone to overfitting because features, which are not providing that much information, instead of having just an embedding layer have multiple self-attention layers. Moreover, we suspect that having a different encoder for each factor seems to produce disentangled representations and the decoder struggles at combining the different sources. This seems to be the reason why concatenation performs better than summation in the case of the multiple-encoder architecture, because with the former at least the decoder can learn to ignore or treat differently different parts of the factored sequence. In the case of the single-encoder architecture, summing seems to work better because it produces a more compact factored embedding and it allows the decoder embedding size not to be doubled (if there are two source factors), preventing overfitting.

We do not have evidence to confirm our initial intuition that factored NMT, particularly with linked data, would be more useful with low-resource tasks, because we have obtained a slighter improvement than the works we cited, being IWSLT 14 a low-resource task while the other works used larger datasets, but at the same time their respective baselines were weaker.

We consider that this project was ambitious, since it had the goal of outperforming a state of the art baseline. Instead of doing one incremental modification, we have tried to tackle different sides. We have both developed new architectures and tried to use a feature that has not been extensively used in NMT (in fact, to the best of our knowledge, only the work we cited had worked with BabelNet synsets in NMT), particularly with BPE. We have not limited ourselves

to adapting an existing factored architecture to the Transformer, but we have tried different variants and developed a non-conventional approach like the multi-encoder architecture as well. Moreover, the baseline we chose was strong and difficult to beat. We have conducted many experiments in order to try different configurations. These challenges have been rewarding, but as we already anticipated when stating the scope of this project, this work consisted in exploring this research direction. The topics of developing new architectures for dealing with factors and the topic of using BabelNet as a source of semantic features for NMT could be whole research lines in their own. With regard to the limitations, our best model has obtained a modest increase and it requires lemmatization. On the other hand, we have not been able to significantly improve the baseline by using linked data.

Personal conclusions Personally, I have learned and applied both generic and technical skills. I learned how to manage a project, particularly a research one, and the importance of planning and being able to find alternatives. On the technical side, I realized that deep learning can be as easy as using already implemented libraries like Fairseq or PyTorch if the task that we have in mind fits in the schema and it works. However, when we have to do some modifications that do not fit in the schema of the library and the architecture does not work as expected, it can be extremely hard to debug, because the errors could be caused by the data, the design or the particular implementation, or it could be that the original idea did not work, at least with the dataset of choice.

Future work There are few works exploring the use of linked data for improving NMT, while classical features are just starting to being used with the Trasformer. We propose research lines related to this work that we consider to be worth exploring.

With regard to factored NMT architectures, we suggest:

- Exploring alternative combination strategies: In section 7.4 we already described four alternative combination strategies, but we did not implement two of them, namely a concatenation-summation hybrid and a fully-connected layer. An obvious extension to this work would be repeating some of the experiments with these two combination strategies.
- Modifying the decoder: As we have seen, most prior work on adding features in order to improve MT consisted in modifying the encoder, as we have done. Modifying the decoder was considered to be out of the scope of the project from the beginning. In fact, the only work that we are aware of that modified the decoder for introducing features was [31], and it required a full PhD program. An equivalent work that modified the beam search procedure as well but using the Transformer architecture instead of LSTMs would be worth exploring. Apart from the source words, the target sequences would have to be tagged as well. Since BabelNet synsets are universal among languages (unlike classical linguistic features), perhaps this approach would take more advantage from synsets. As future work it could even be considered to use the LSTM-based system of [31], but the vanilla Transformer, without any features, would probably outperform it.
- Masked Sequence to Sequence Pre-training for Language Generation (MASS) [48]: A work published very close to the final stage of this project proposed randomly *masking* a fragment (a number of consecutive tokens) of the input sentence, that is to say, setting a random

fragment of the source sequence to a special token. They improved language generation by forcing the neural network to predict the unknown fragments. Inspired by this proposal, we propose testing a similar mechanism, but adapted to the case of factored NMT. In particular, since words already provide a great deal of information, factored architectures can have a tendency to learn to ignore the features, which can decrease performance when evaluating. When training, a random token from the source word sequence (eg. chosen from a uniform distribution) would be masked. Features would never be masked, in order to force the neural network to take advantage of them. In inference time, no words would be masked. In fact, we started working on a prototype of this proposal but for time constraints it was not feasible to perform the full implementation and experimentation.

- Exploring unsupervised learning and transfer learning: A great deal of the interest of NLP community has recently switched to unsupervised learning techniques and improving transfer learning for NLP. In future works, we could come up with unsupervised techniques particularly adapted to the case of factors in NMT, like pre-training feature embeddings with monolingual (or even masked) corpora. Those pre-trained feature embeddings could transfer among tasks, although it should be investigated empirically.

As far as linked data and features are concerned, we propose:

- Using our architectures for domain adaptation, particularly exploring other linked data databases: We have tested our new architectures for general domain tasks, in which words can be enough for getting a great deal of information. Nevertheless, the different factored Transformer configurations could be more useful for adapting an existing NMT system to a particular domain by combining general domain embeddings with domain-specific embeddings, or by introducing domain-specific features built from a domain-specific linked data database. For instance, for translating medical texts, words referring to diseases or drugs could be annotated with their respective Unified Medical Language System (UMLS)[37] id.

- Synsets preprocessing and rules: In section 8.2.2 we outlined different options for assigning synsets and described the reasons for our choices. However, perhaps it would be worth exploring fancier heuristics (or even expert knowledge from linguists, for example) for discarding synsets, choosing between single-word and multi-word synsets, the frequency threshold or the rules for dealing with words without assigned synsets.

- Using lemmas instead of morphological features for undefined synsets: Regarding the last point, we propose a particular approach that would be worth exploring, assigning words with undefined synsets to lemmas, instead of other features. In fact, we considered this approach during the development of this project, but we did not follow it for a variety of reasons. First of all, the big vocabulary size of both lemmas and synsets could have made the vocabulary too big, even more than the one that synsets already had with synsets. On the other hand, pruning by frequency could have lead to sequences mostly dominated by lemmas. Overcoming these challenges could be an interesting research line.

- Exploring alternative usage for BabelNet synsets: Instead of using synsets as a feature, perhaps they could be used to perform data augmentation for low-resource languages by

[37] UMLS is an ontology of medical terms: `https://www.nlm.nih.gov/research/umls/`.

training a NMT system to translate from the synsets of monolingual text in the high-resource language into the words of the low-resource language. This way the amount of parallel sentences would increase.

- Building a deep learning architecture for extracting information from linked data: For instance, by building an alternative to Babelfy. In this work, we extracted BabelNet synsets from Babelfy, a private API based on BabelNet that performs word sense disambiguation. Perhaps it would be worth developing an open-source alternative based on state of the art techniques. Since linked data are graph-based, recent works on applying deep learning to graphs (like [49]) could be a starting point.

Finally, with regard to the experimental settings, we suggest:

- Optimizing hyperparameters and further exploring the different configurations: The most obvious follow-up that comes to our mind would be to search different hyparameters and further explore some of the configurations we tried.

- Evaluating our model and the baseline in out-of-domain data: We have the intuition that it would be the case that our model generalized better to non-TED texts, but it should be investigated.

- Facebook Low Resource (FLoRes) MT Benchmark: This year Facebook Research released both a benchmark for MT and the respective baselines for each dataset [50]. There are two datasets: Nepali–English and Sinhala–English, and baselines for both directions. The datasets are already split into train, validation, devtest and test. Although there are more sentences than in IWSLT 14 DE-EN (for instance, about 600k sentences for English-Nepali), the task is still considered to be low-resource and in fact it is much more difficult than IWSLT 14 DE-EN, so the baselines are weaker and the architectures and features we used could be more useful in this case. For instance, the baseline for parallel data only for English-Nepali, is as low as BLEU = 4.3. In fact, when this work was close to being delivered, we finished the baseline reproduction and feature tagging (including both classical linguistic features and BabelNet synsets) for English-Nepali, and we expect to continue working on this dataset for the coming months as a follow-up of this work.

- Performing a systematic study of the effectiveness of word features in RNN-based (or even convolutional) architectures in comparison to the Transformer: Although in one experiment we followed the same approach as [32] but with the Trasformer, the results are not directly comparable because it was not the same dataset.

- Performing a systematic study of the effectiveness of word features depending on the size of the dataset: This way, we could confirm or deny our intuition that factored NMT is more useful in low-resource tasks.

- Finding alternative uses for the multi-encoder architecture: Perhaps it could be adapted to multi-source NMT (translating from different sources simultaneously) or using the multi-encoder as an ensemble of encoders from the same source.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[2] N. Indurkhya and F. J. Damerau, *Handbook of Natural Language Processing*, 2nd ed. Chapman & Hall/CRC, 2010.

[3] K. Cho, "Introduction to neural machine translation with gpus (parts 1, 2 and 3)." [Online]. Available: https://devblogs.nvidia.com/introduction-neural-machine-translation-with-gpus/

[4] 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 on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: https://doi.org/10.3115/1073083.1073135

[5] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer, "The mathematics of statistical machine translation: Parameter estimation," *Comput. Linguist.*, vol. 19, no. 2, pp. 263–311, Jun. 1993. [Online]. Available: http://dl.acm.org/citation.cfm?id=972470.972474

[6] M. R. Costa-jussà, "From feature to paradigm: Deep learning in machine translation," *J. Artif. Int. Res.*, vol. 61, no. 1, pp. 947–974, Jan. 2018. [Online]. Available: http://dl.acm.org/citation.cfm?id=3241691.3241715

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

[8] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, p. arXiv:1412.6980, Dec 2014.

[9] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv e-prints*, p. arXiv:1406.1078, Jun 2014.

[10] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2013, pp. 1700–1709. [Online]. Available: http://aclweb.org/anthology/D13-1176

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *arXiv e-prints*, p. arXiv:1409.3215, Sep 2014.

[13] C. Tillmann and H. Ney, "Word reordering and a dynamic programming beam search algorithm for statistical machine translation," *Comput. Linguist.*, vol. 29, no. 1, pp. 97–133, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1162/089120103321337458

[14] M. Freitag and Y. Al-Onaizan, "Beam Search Strategies for Neural Machine Translation," *arXiv e-prints*, p. arXiv:1702.01806, Feb 2017.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv e-prints*, p. arXiv:1301.3781, Jan 2013.

[16] T. Berners-Lee, "Berners-lee: Linked data." [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[17] R. Navigli and S. P. Ponzetto, "BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network," *Artificial Intelligence*, vol. 193, pp. 217–250, 2012.

[18] S. Ruder, "Deep learning for nlp best practices: Attention." [Online]. Available: http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

[19] D. Boehning and S. H. Snyder, "Novel neural modulators," *Annual Review of Neuroscience*, vol. 26, no. 1, pp. 105–131, 2003, pMID: 14527267. [Online]. Available: https://doi.org/10.1146/annurev.neuro.26.041002.131047

[20] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *arXiv e-prints*, p. arXiv:1508.04025, Aug 2015.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[22] Z. Lin, M. Feng, C. Nogueira dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A Structured Self-attentive Sentence Embedding," *arXiv e-prints*, p. arXiv:1703.03130, Mar 2017.

[23] M. Daniluk, T. Rocktäschel, J. Welbl, and S. Riedel, "Frustratingly Short Attention Spans in Neural Language Modeling," *arXiv e-prints*, p. arXiv:1702.04521, Feb 2017.

[24] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional Sequence to Sequence Learning," *arXiv e-prints*, p. arXiv:1705.03122, May 2017.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv e-prints*, p. arXiv:1706.03762, Jun 2017.

[26] K. Kurita, "Paper dissected: "attention is all you need" explained." [Online]. Available: http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[28] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv e-prints*, p. arXiv:1607.06450, Jul 2016.

[29] R. Sennrich, B. Haddow, and A. Birch, "Neural Machine Translation of Rare Words with Subword Units," *arXiv e-prints*, p. arXiv:1508.07909, Aug 2015.

[30] P. Gage, "A new algorithm for data compression," *C Users J.*, vol. 12, no. 2, pp. 23–38, Feb. 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=177910.177914

[31] M. García-Martínez, L. Barrault, and F. Bougares, "Factored Neural Machine Translation," *arXiv e-prints*, p. arXiv:1609.04621, Sep 2016.

[32] R. Sennrich and B. Haddow, "Linguistic input features improve neural machine translation," *CoRR*, vol. abs/1606.02892, 2016. [Online]. Available: http://arxiv.org/abs/1606.02892

[33] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "OpenNMT: Open-Source Toolkit for Neural Machine Translation," *ArXiv e-prints*.

[34] R. Sennrich, O. Firat, K. Cho, A. Birch, B. Haddow, J. Hitschler, M. Junczys-Dowmunt, S. Läubli, A. V. M. Barone, J. Mokry, and M. Nadejde, "Nematus: a toolkit for neural machine translation," *CoRR*, vol. abs/1703.04357, 2017. [Online]. Available: http://arxiv.org/abs/1703.04357

[35] C. E. i Bonet and J. van Genabith, "Multilingual semantic networks for data-driven interlingua seq2seq systems," in *Proceedings of the LREC 2018 Workshop "MLP-MomenT"*, J. Du, M. Arcan, Q. Liu, and H. Isahara, Eds. o.A., 2018, pp. 8–13.

[36] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Universal Transformers," *arXiv e-prints*, p. arXiv:1807.03819, Jul 2018.

[37] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[38] M. O. Sergey Edunov and S. Gross, "Fairseq: Facebook ai research sequence-to-sequence toolkit written in python." [Online]. Available: https://github.com/pytorch/fairseq

[39] S. S. L. B. M. F. Mauro Cettolo, Jan Niehues, "Report on the iwslt 2014 evaluation campaign." [Online]. Available: http://workshop2014.iwslt.org/downloads/IWSLT_Evaluation.pdf

[40] M. Popel and O. Bojar, "Training tips for the transformer model," *CoRR*, vol. abs/1804.00247, 2018. [Online]. Available: http://arxiv.org/abs/1804.00247

[41] B. Zoph and K. Knight, "Multi-source neural translation," *CoRR*, vol. abs/1601.00710, 2016. [Online]. Available: http://arxiv.org/abs/1601.00710

[42] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[43] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[44] A. Moro, A. Raganato, and R. Navigli, "Entity Linking meets Word Sense Disambiguation: a Unified Approach," *Transactions of the Association for Computational Linguistics (TACL)*, vol. 2, pp. 231–244, 2014.

[45] P. Koehn, "Statistical significance tests for machine translation evaluation," in *Proceedings of EMNLP 2004*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 388–395. [Online]. Available: https://www.aclweb.org/anthology/W04-3250

[46] R. Müller, S. Kornblith, and G. Hinton, "When Does Label Smoothing Help?" *arXiv e-prints*, p. arXiv:1906.02629, Jun 2019.

[47] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 7 1948. [Online]. Available: https://ieeexplore.ieee.org/document/6773024/

[48] K. Song, X. Tan, T. Qin, J. Lu, and T. Liu, "MASS: masked sequence to sequence pre-training for language generation," *CoRR*, vol. abs/1905.02450, 2019. [Online]. Available: http://arxiv.org/abs/1905.02450

[49] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "Pytorch-biggraph: A large-scale graph embedding system," *CoRR*, vol. abs/1903.12287, 2019. [Online]. Available: http://arxiv.org/abs/1903.12287

[50] F. Guzmán, P.-J. Chen, M. Ott, J. Pino, G. Lample, P. Koehn, V. Chaudhary, and M. Ranzato, "Two New Evaluation Datasets for Low-Resource Machine Translation: Nepali-English and Sinhala-English," *arXiv e-prints*, p. arXiv:1902.01382, Feb 2019.

A Implementation reference

In this appendix, we will provide an overview of the implementation and some usage examples.

A.1 Overview

It is highly recommended to read Fairseq's documentation[38] and take a look at its Github repository[39] in order to have a better understanding of our implementation. Notice that the scripts are mostly intended for running in a high performance cluster with GPUs.

In our source code repository, the relevant directories and files are:

- `fairseq/data/factored_language_pair_dataset.py`: Based on `language_pair_dataset`. In this file we can find the implementation for parsing and loading a dataset such that there are N source files (one for each factor) instead of only one. Unlike `language_pair_dataset`, `factored_language_pair_dataset.py` assures that batches are loaded such that all the source sentences refer to the same target sentence (ie. the features refer to the word sequence that has been loaded).
- `fairseq/models`: In this directory we can find the different variants of the proposed architectures, as well as a required module for the multi-encoder architecture.
  - `factored_composite_encoder.py`: A module for having multiple encoders (not specific to the Transformer) and concatenating their outputs. It is based on `composite_encoder.py`, which is intended for having multiple encoders for the same source sequence (not factors referring to the same sequence) without combining their outputs.
  - `factored_composite_encoder_sum.py`: The same as before, but summing instead of concatenating.
  - `factored_transformer.py`: The implementation of the multi-encoder architecture with concatenation. It leverages the `factored_composite_encoder.py` module.
  - `factored_transformer_sum.py`: The same as before, but summing instead of concatenating.
  - `factored_transformer_one_encoder.py`: The implementation of the single-encoder architecture with concatenation.
  - `factored_transformer_one_encoder_sum.py`: The same as before but summing instead of concatenating.
- `fairseq/tasks/factored_translation.py`: The Fairseq task for executing the configuration specified in the parameters. It leverages `language_pair_dataset.py` in order to parse and load the required dataset, and it calls the specified model. It is based on `multilingual_translation.py`, with the key difference that instead of alternatively training different pairs it must use the different sources simultaneously, so the batches must be combined. Both the training step and the validation step are modified in order to account for this fact.

[38] Fairseq documentation: `https://fairseq.readthedocs.io/en/latest/`.
[39] Fairseq repository: `https://github.com/pytorch/fairseq`.

- `fairseq/sequence_generator`: The code for loading the sequences and generating the translations during inference had to be modified similarly to `factored_translation.py`.
- `train.py`: The generic trainer was modified to allow freezing the weights of a certain component at the specified epoch.
- `preprocessing`: In this directory we can find the scripts for downloading, cleaning, splitting and tagging the dataset:
  - `babelfy`: Scripts for retrieving synsets from Babelfy, assigning and aligning them.
  - `iwslt14`: Scripts for getting and preparing IWSLT 14 DE-EN. The script `get-train-valid-test-iwslt14-de-en.sh` is taken from the authors of the baseline[40]. In this directory there is the script for getting the test set from another year as well.
  - `postagger`: Scripts for PoS tagging and aligning with the first tagger that we used, TreeTagger.
  - `stanford`: Scripts for tagging and aligning classical linguistic features with Stanford models and Spacy.
- `preprocess_tags.sh`: Script for Fairseq's preprocessing of features. It builds the dictionaries and binarizes the data.
- `train_tags.sh`: Script for running the training of the model. In this example, for the multiple-encoder architecture.
- `generate_tags.sh`: Script for generating and evaluating translations with features.
- `iwslt14_pos_synsets`: Scripts for running the experiments with PoS and synsets.
- `new_iwslt14_scripts`: Scripts for running the experiments with all the linguistic features and lemmas.

A.2 Usage

For downloading, cleaning and tokenizing the train, validation and devtest sets:

```
sbatch preprocessing/iwslt14/get-newtest-iwslt14-de-en.sh
```

In order to do the same but for the test set, we have to run `get-newtest-2013.sh` the same way as before. The scripts `run_feature_tagger_spaces.sh` and `run_align_tokensS.sh`, in `preprocessing/stanford /feature_tagger_iwslt14/`, will tag the text with classical linguistic features and align them with BPE, respectively. `run_feature_tagger_spaces.sh` downloads the Stanford model required for tagging as well.

On the other hand, for retrieving synsets from Babelfy:

```
sbatch preprocessing/babelfy/run_get_synsets.sh
```

[40] `https://github.com/pytorch/fairseq/blob/master/examples/translation/prepare-iwslt14.sh`

In the case that Slurm was not installed, it could be run with plain Bash. The scripts for getting the data or tagging and aligning the features are prepared to run without modifying any option except the absolute path in some cases. However, in the case of the script for retrieving scripts, the API key must be changed. By default, the script has `KEY = 'KEY'`, which is the default one and it is allowed to do a few calls per day.

For running the factored architectures, the Python dependencies of Fairseq must be satisfied (see `requirements.txt`).

As an example of training and inference procedures, we will see the case of the single-architecture with concatenation and all the classical linguistic features, although the three required steps are analogous to those of the other configurations. Firstly, Fairseq preprocessing must be applied in order to build the dictionaries and binarize the data. We have to run the corresponding preprocessing scripts. In this case, `preprocess_stanfordS.sh` and `preprocess_tags_tokensS.sh` for words and features, respectively, in the `new_iwslt14_scripts/` directory.

In the model code (in this case, `factored_transformer_one_encoder.py`), we have to add the desired architecture configuration:

```
@register_model_architecture('factored_transformer_one_encoder',
'factored_transformer_one_encoder_sennrichS')
    def factored_one_encoder_iwslt_de_en(args):
        args.encoder_embed_dim = getattr(args, 'encoder_embed_dim', 512)
        args.encoder_ffn_embed_dim = getattr(args, 'encoder_ffn_embed_dim', 1024)
        args.encoder_attention_heads = getattr(args, 'encoder_attention_heads', 4)
        args.encoder_layers = getattr(args, 'encoder_layers', 6)
        args.encoder_embed_dim_sizes = {'de_tokensS': 362, 'de_tokensS_lemmas': 115,
        'de_tokensS_pos': 10, 'de_tokensS_deps': 10, 'de_tokensS_tags': 10,
        'de_tokensS_subword_tags': 5}
        args.decoder_embed_dim = getattr(args, 'decoder_embed_dim', 512)
        args.decoder_ffn_embed_dim = getattr(args, 'decoder_ffn_embed_dim', 1024)
        args.decoder_attention_heads = getattr(args, 'decoder_attention_heads', 4)
        args.decoder_layers = getattr(args, 'decoder_layers', 6)
        factored_one_encoder_base_architecture(args)
```

For executing the training procedure:

```
python train.py <WORKING_DIR> \
 --task factored_translation --arch factored_transformer_one_encoder_sennrichS \
 --optimizer adam --adam-betas '(0.9, 0.98)' \
 --lr-scheduler inverse_sqrt --warmup-init-lr 1e-07 --warmup-updates 4000 \
 --lr 0.0005 --min-lr 1e-09 --dropout 0.3 --weight-decay 0.0001 --criterion \
 label_smoothed_cross_entropy --label-smoothing 0.1 --max-tokens 4000 \
 -save-dir <MODEL_DIR> --lang-pairs de_tokensS-en,de_tokensS_lemmas-en, \
 de_tokensS_pos-en,de_tokensS_deps-en, \
 de_tokensS_tags-en,de_tokensS_subword_tags-en \
 --max-update 50000 --multiple-encoders False
```

For generating and evaluating translations with the trained model with features:

```
python generate.py <DESTINATION_DIR> --path <MODEL_DIR>/model.pt \
        --beam 5 --batch-size 1 --lang-pairs \
        de_tokensS-en,de_tokensS_lemmas-en,de_tokensS_pos-en,de_tokensS_deps-en,\
        de_tokensS_tags-en,de_tokensS_subword_tags-en \
        --task factored_translation --remove-bpe --target-lang en
```