BACHELOR'S THESIS

Degree in Mathematics

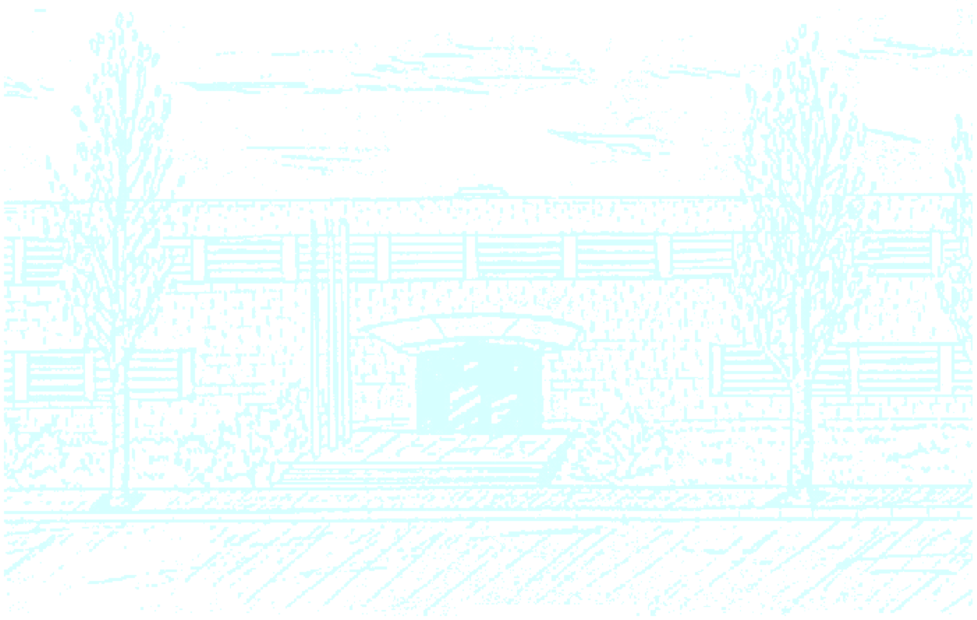
Title: Efficient Transformers for Direct Speech Translation

Author: Belén Alastruey Lasheras

Advisor: Marta R. Costa-Jussà, Gerard I. Gállego

Department: Computer Science

Academic year: 2020-2021



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Efficient Transformers for Direct Speech Translation

A Bachelor's Degree Thesis submitted to the

Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

by

Belén Alastruey Lasheras

In partial fulfilment of the requirements for the

Mathematics Bachelor's Degree

Supervised by:

Gerard I. Gállego

Marta R. Costa-Jussà

Barcelona, June 2021

Abstract

The advent of Transformer-based language models has revolutionized text translation. Because of its quadratic complexity, the Transformer is suitable for short sentences, but not for long text translation tasks. To overcome this problem many efficient Transformer variants have been proposed, with linear complexity instead of quadratic. But in the last few years, the Transformer has surpassed the barriers of text. In Speech-to-Text context, a standard approach is working with previously extracted audio features. Therefore, input sequence lengths of speech tasks are approximately an order of magnitude longer than usual text sequence lengths. Hence, working with such large speech sequences with the original Transformer has a dramatic impact. To bypass this problem, we usually use strided convolutions, to reduce the sequence length.

In this study, we propose a new approach for Speech-to-Text translation, where thanks to an efficient Transformer we can work with a spectrogram without having to use convolutional layers before the Transformer. This allows the encoder to learn directly from the spectrogram and no information is lost, which we believe could be profitable. We have created an encoder-decoder model, where the encoder is an efficient Transformer -the Longformer- and the decoder is a traditional Transformer decoder. Firstly we trained our model for an Automatic Speech Recognition (ASR) task, and then for Speech Translation using the ASR pre-trained encoder. Our results are close to the ones obtained with convolutional layers and a regular Transformer, showing less than a 10% relative reduction of the performance, meaning that this is a great starting point for a promising research path.

Al Sergi, per haver-me recolzat en tot el camí fins aquí.

A mi familia, por su apoyo incondicional.

Acknowledgements

I would like to thank Marta R. Costa-Jussà and Gerard I. Gállego for giving me the chance to work on this project and for guiding me, but above all for showing me how interesting research can be, for helping me regain motivation to continue studying, for all the opportunities they have given me, and for making me feel backed and valued during all these months.

Table of contents

List of Figures

List of Tables

Nomenclature

SMT	Statistical Machine Translation
NMT	Neural Machine Translation
Seq2Seq	Sequence to Sequence
End2End	End to End
ASR	Automatic Speech Recognition
ST	Speech Translation
MT	Machine Translation
RNN	Recurrent Neural Network
S2T	Speech-to-Text

Chapter 1

Introduction

Speech is the natural way of human communication. As humans, we learn how to write and read on the first school years, but before that, we already know how to communicate with others; since the early stages of life we know how to talk. However, communication is not always simple, since the different languages around the world can be a hindrance. For this reason, the study of both voice processing and translation have always been appealing research fields. In Artificial Intelligence context, a relevant task concerning these concepts is Speech Translation, and in particular, Speech-to-Text translation.

The first Speech-to-Text translation method, currently known as cascade [Ney, 1999b], consists in the concatenation of two independent models. The first one, an Automatic Speech Recognition model, writes a transcription of the spoken sentence, and the second one, a Machine Translation model, translates the transcription to another language. But in the last few years, new models based on end-to-end architectures have emerged. These models are capable of translating from audio to text, without the need to go through the intermediate step of transcription. These models have rapidly evolved and, nowadays, they can reach the same state-of-the-art results than cascade models [Ansari et al., 2020]. Nevertheless, results provided by both cascade and end-to-end architectures are far from optimal, and therefore these research fields are still under development.

Recently, the growing popularity of *Transformer*-based [Vaswani et al., 2017] models for text translation has broken the barriers of its main application, and it is currently being used to process all kinds of data, such as image [Parmar et al., 2018] or audio [Gangi et al., 2019] [Cross Vila et al., 2018]. When dealing with speech, we usually work with extracted audio features, like mel-spectrograms. But the sequence length of this kind of data is longer than the usual *Transformer*'s text input, what makes them difficult to process with the original *Transformer*. To overcome this problem, a usual approach is shortening the input's sequence length by adding strided convolutional layers.

This thesis objective is contributing to Speech-to-Text translation research, proving the feasibility of a new variation of the *Transformer* that makes it suitable for audio inputs, and therefore for speech tasks, without the need of adding convolutional layers. To do this, we want to use an *Efficient Transformer*, an adaptation of the *Transformer* suitable to process long sequences, initially created for long document tasks. In particular, we will use the *Longformer* [Beltagy et al., 2020], with a self-attention based on a sliding window, which we believe could be profitable for audio processing. We want to take advantage of the lower complexity of this model and create a Speech-to-Text *Transformer* for ST tasks (that will also work for ASR), where the *Longformer* will deal with the audio input. We believe the training could benefit from this approach, since it lets the model learn directly from the spectrogram and no information is lost in the convolutional layers. On the negative side, possible complications derived of this system are the reported *Longformer* instability [Beltagy et al., 2020], and that cross-attention between the encoder and the decoder could be hindered by a mismatching between the input and the output sequence lengths.

By following this approach, the results of this thesis can be used in the *LUNAR* project[1], to expand the current objective of creating a universal language representation for text translation, also for Speech-to-Text translation.

Finally, to close the introduction, we will describe this thesis structure:

Chapter 2. We first show the fundamental concepts concerning audio processing, and we define the basics about Automatic Speech Recognition and Machine Translation tasks for both text and speech translation. We also explain the main algorithms that have been used in these tasks during the last years, from statistical models to the *Transformer*. Additionally, we talk about word segmentation and evaluation scores.

Chapter 3. Once we have explained what we consider a preliminary background, in this chapter we study the state-of-the-art algorithms that concern directly our work. Specifically, we see the Speech-to-Text *Transformer* [Wang et al., 2020a] and some *Efficient Transformers* for text tasks.

Chapter 4. In this chapter we explain the body of our research, including our objective, our model and the methodology followed during the development of this thesis.

Chapter 5. This chapter includes all the information about the performed experiments. We start with an overview of the dataset and the experimental framework that we have used, followed by a detailed description of our model's parameters and training specifications. Finally, we show the results of the experiments.

[1]Project funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 947657)(`lunar.cs.upc.edu`)

Chapter 6. In this chapter we explain the conclusions drawn after performing the experiments, and we discuss future work that could be done to improve the current results.

Chapter 2

Background

For a better understanding of this thesis motivation, we must first contextualize the speech recognition and translation (specifically the speech-to-text translation) scenarios. In this section, we describe the pre-processing needed to work with audio and the algorithms that have led to the current state-of-the-art in Automatic Speech Recognition (ASR), Machine Translation (MT) and Speech Translation (ST). We will as well see other concepts that are relevant to the development of the thesis, such as word segmentation and evaluation scores.

2.1. Audio pre-processing

In order to process data, computers need it to be discrete and numerical, and this is also the case of ASR or ST algorithms. As a consequence, we need to transform sound waves into discrete numerical sequences.

Sound travels in time in one-dimensional waves, continuous functions in $\mathbb{R}$, that have a single amplitude value for every instant. The first step to transform an audio wave into discrete numerical data is known as sampling, i.e. saving the amplitude of the sound wave at some equally spaced time steps. However, we must use a sampling rate that allows us to recover all the information, ensuring no data is lost in the sampling. To determine the appropriate sampling rate, we use the following signal processing theorem:

Theorem 1 (Nyquist-Shannon Theorem). [Shannon, 1949] Given a function $f(t)$, that contains no frequencies higher than Ω_B Hz, we can reconstruct $f(t)$ perfectly from it is ordinates at a series of equally spaced points with a frequency Ω_S , if $\Omega_S > 2 \cdot \Omega_B$.
$\Omega_N = 2 \cdot \Omega_B$ is called Nyquist frequency. Therefore, any sampling rate larger than Ω_N is a sufficient. Equivalently, for a given sample rate Ω_S, perfect reconstruction is guaranteed for sounds with frequency $\Omega_B < \frac{\Omega_S}{2}$.

Music or industrial machines sounds can reach high frequency levels and are usually sampled at 44.1kHz or 48kHz. The human voice, on the other hand, has a much more limited frequency range that hardly exceeds 8kHz, and therefore is usually sampled at 16kHz.

The numerical sequence obtained after the audio wave sampling is usually too long and contains redundancies. Traditionally, to overcome this problem, the sequence is transformed to an spectrogram using Fourier Transform, that transforms signal from time domain to frequency domain. In this step, the sampled sound wave is grouped in overlapping windows, that are usually 25-millisecond-long and are placed every 10 milliseconds. Then each window is decomposed into the frequencies that form it using Fourier Transform. The result is a discrete function that contains information about the relative weight of every frequency range in that audio window, and that can be represented in a vector by fixing the top and low bounds of frequency. This vector is usually drawn as a chart, where colors show the intensity of every frequency. By doing this with every window in our audio and then placing all the charts together, we get a spectrogram, a chart where x axis shows time, y axis shows frequency and colors shows the weight of every frequency at every instant.

When working in speech tasks, the standard approach is using the mel-spectrogram, a transformation of a regular spectrogram using the mel scale [Stevens et al., 1937]. This scale is the result of a logarithmic transformation of the regular frequency scale, so that sounds of equal distance from each other on the mel scale are also perceived by humans as they are equal in distance. A mel-spectrogram is a spectrogram with the mel scale as its y axis (the frequency axis).

2.2. Automatic Speech Recognition, Machine Translation and Speech-to-Text Translation

In this section we will study the first steps and important concepts of ASR, MT and ST tasks.

2.2.1. Automatic Speech Recognition

ASR is the task of obtaining a transcription from a spoken sentence. This task can be complex, since in addition to the difficulty of the task itself, there are other complications derived of working with speech such as background noise, echo, different accents and pronunciations, and the similarity between different sounds.

Interest in ASR arose in the 1980s, starting with simple goals such as transcribing the digits of a number. As the years went by, the complexity of the task increased, first to the transcription of names or specific commands, then to short sentences, and up to nowadays, where all types of sentences can be handled.

The first ASR algorithms were statistical models, that were the predominant systems until the 2010s. In these models, given a spectrogram S, the transcription of the spoken sentence contained in S was defined as:

$$W^* = argmax_W P(S|Q) \cdot P(Q|W) \cdot P(W)$$

where W is any possible sentence, S is the spectrogram, and Q are the phonemes of the sentence. Therefore, $P(W)$ is a language model, that calculates the probability of the sentence W to be grammatically correct, $P(Q|W)$ is a pronunciation model, that calculates the probability that a sequence of phonemes Q is the one belonging to the sentence W, and $P(S|Q)$ is an acoustic model, that calculates the probability that a spectrogram S is the one where the sequence of phonemes Q is represented.

In the 2010s, the first neural models replaced the acoustic and the pronunciation models with a Recurrent Neural Network. RNNs have been widely used in problems where there exists a recurrence relation between the elements of the input and output sequences: the input at a time t is influenced by x_{t-1}, that is influenced by x_{t-2} and so on. This can be useful in a task as ASR.

In this model, known as Connectionist Temporal Classification (CTC) [Graves and Jaitly, 2014], a sequence of spectrogram slices (one every 10 milliseconds) is fed into the RNN. For every spectrogram slice, the RNN predicts a vector of size $|V|$ (where V is the vocabulary, usually the alphabet and the special *blank* and *space* tokens). This vector contains in each of its elements the probability of that character to be the one said in the spectrogram slice. It is important to note that the sound of a character usually lasts more than one spectrogram slice and for this reason, predictions usually contain every letter more than once. To solve this problem, a processing that deletes repetitions of letters is applied to the predicted strings. But this model can lead to predictions of nonexistent words or sentences that don't make sense. To overcome this problem and improve the performance of the model, it is usually followed by a language model. This language model takes the top most probable predictions from the probabilities vectors and decides which is more likely to be grammatically and syntactically correct. The result obtained after the language model is chosen as the transcription of the initial spoken sentence.

2.2.2. Machine Translation

MT is the task of using computers to translate from one language to another. Due to the complexity of languages this task can become very complicated. This happens as a consequence of noun declensions, verb conjugations or grammar, that work differently in every language. Additionally, there are problems as meanings that can be expressed with different words (this is the case of synonyms), while the same word can have different meanings (depending on the context in which it is found). This is why translation, and in particular MT, cannot be limited to translating words one by one, but must come to understand the structure of the sentences and the context of the words. One of the fundamental concepts in MT is derived from this idea: alignment (figure 2.2.1), which consists of the association of words of the original sentence with those of the translated sentence according to their meaning. To show this concept it is common to use matrices, where the alignment between the words of the two sentences can be visually represented:

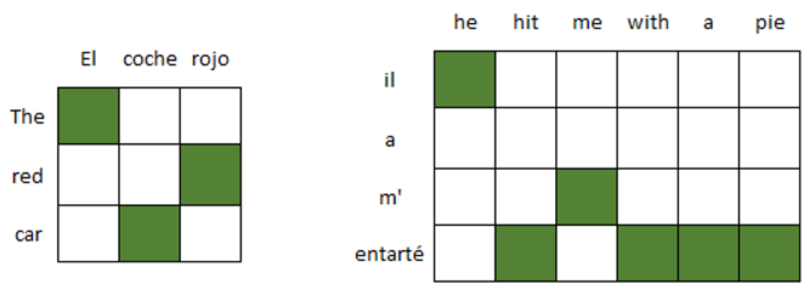


Figure 2.2.1: Depending on the sentence, alignment can be easy or complex.

The first MT implementations appeared in the early 1950s, when during the Cold War between the US and Russia, the need arose for the Americans to quickly and efficiently translate messages from Russian into English [Hutchins et al., 1955]. The systems used at that time were very simple, in fact they were rule-based systems, which consisted of using a bilingual dictionary and a few rules to translate each Russian word into the corresponding English word.

Years later, in the 1990s, the first statistical MT methods appeared, which would become the standard until the 2010s. A statistical MT model is based on the idea of representing the language as a probabilistic distribution. Thus, we can define an optimal translation as: given a sentence x, in language X, and a different language Y, we say that the translation of x into language Y is:

$$y = argmax_{y'} P(y'|x)$$

where y' is any sentence in language Y. Then, applying Bayes Rule, the formula can be decomposed in:

$$y = argmax_{y'} P(x|y') \cdot P(y')$$

where $P(x|y')$ works as a translation model, that models how to translate, and $P(y')$ as a language model, that models how to write correctly in language Y.

It was finally in 2014 when the first neural MT models appeared, and by 2016 they had outperformed SMT models [Ansari et al., 2020].

2.2.3. Speech-to-Text Translation

ST is the task in which a spoken phrase is translated into another language. Depending on whether the output of the translation is text or a spoken sentence, we can talk about Speech-to-Text or Speech-to-Speech translation respectively. If the task of translating text is already hard, trying to translate audio is even more complicated, due to problems related to audio processing, such as the ones seen in section 2.2.1.. Furthermore,in the case of ST, there are also problems related to speech and text mismatches, such as hesitations, repetitions, lack of punctuation or discourse makers.

In the 1990s that the first ASR model was created [Ney, 1999a]. Since then and until 2016 the models used were those currently known as cascade models (figure 2.2.2). Cascade models consist of a pipeline that contains two independent models with isolated objectives: the first one is in charge of performing an ASR task, turning a spoken sentence into its transcription, and then a second phase of text MT, which translates the transcription to another language. These models have a main advantage: by the nature of their construction it is possible to train separately the ASR model and the MT model, what allows using different training datasets for each one. This is the reason why the amount of data available for the training of these models is very large compared to the data that would be available if working on a model that does both the ASR and the ST at the same time. But cascade models do not perform the translation process optimally, since in addition to the inevitable errors that can occur in each of the models that are part of the process, an error in the ASR phase would automatically propagate to the MT phase, resulting in even more errors and a worse result of the full ST process [Weiss et al., 2017]. To avoid this kind of errors, it is common to find some layers, after the ASR and before the ST, in charge of converting a literal transcription of the sentence into a sentence that would make sense in written form, for example by eliminating repetitions or interjections, common in spoken language but not in written language.

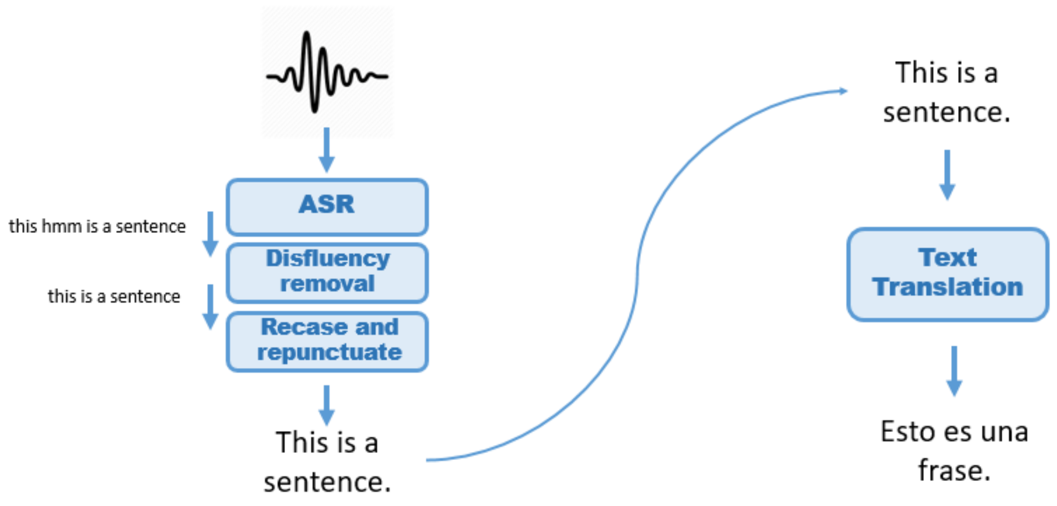


Figure 2.2.2: Cascade approach for speech-to-text translation, with disfluency removal, repunctuation and recasing layers.

But even with these adjustments, propagation errors are difficult to avoid. This is why since 2016, research in Speech-to-Text translation models has focused on models known as End-to-End (End2End), capable of translating directly from a spoken sentence to a written sentence in another language, without the need to go through the intermediate step of obtaining the initial sentence transcription. This kind of models needs to be trained End2End, and as commented above, this can lead to possible lack of training data. In addition to this problem, it is also important to note the difficulty of the alignment of a spoken sentence (a spectrogram) and a written sentence, due to the fact that their lengths are very different. Still, End2End models are currently a big research field [IWS, 2019] [Ansari et al., 2020], because of the belief that avoiding error propagation and having access to the audio during the translation should help to improve results.

	Cascade	End2End
Less complex tasks	x	
Lots of data available	x	
Access to all audio during translation		x
No error propagation		x
Easier managment		x

Table 2.2.1: Cascade vs End2End

Nowadays, both kinds of models can reach results of the same quality [Ansari et al., 2020], and there is still debate about which approach is better (see comparison in table 2.2.1). In particular, ideas such as how End2End models reduce errors by avoiding error propagation, or how does accessing the audio during translation help, still need to be thoroughly investigated.

2.3. Sequence-to-Sequence models (Seq2Seq)

Unlike in other tasks carried out by Machine Learning algorithms, when working with text it is very important to understand the context of each word. This is why it is not possible to apply a word-by-word algorithm for tasks such as translation, it is better to work at sentence-level.

MT is a Seq2Seq task, i.e. the input and the output of the model are sequences, which are usually time-series, but in this case are sentences (sequences of characters or words). This kind of models has an encoder-decoder based architecture. The encoder is in charge of processing the input sequence information and storing it. Then the decoder uses the encoder's extracted information to generate the output sequence. The first neural translation approaches consisted of a Seq2Seq model where the encoder and decoder were Recurrent Neural Networks (RNN) [Sutskever et al., 2014]. This kind of networks are used when there exists a recurrence relation between the elements of the input and output sequences. This is the case of translation, otherwise we would be translating word-by-word. But using RNNs would have a problem: since all the information obtained by the encoder has to be stored in only one vector (the hidden state between the encoder and the decoder), in many occasions the details belonging to the beginning of the sentence would be lost. Consequently they would not used by the decoder, causing bad results because of lack of parts of the information when doing the predictions. To try to solve this problem, the basic RNN layers were replaced by Long Short-Term Memory (LSTM) layers, a type of RNNs capable of detecting which information is more important and saving it as a priority. That is, instead of forgetting the beginning of the sentence, it is possible to identify which parts are important and which are not, and thus forget the unimportant, without being influenced by the position they occupy in the sentence. The resultant model is a Seq2Seq model with and encoder and a decoder that are made of RNNs (in particular LSTMs), and has the structure shown in figure 2.3.1 [Sutskever et al., 2014].

In this model, words are fed one-by-one to the encoder. Then all the extracted information is stored in the encoder's last hidden state and it is sent to the decoder, which uses it to predict the output in an autoregressive way, i.e. it predicts the words one at a time using the information from previous steps in each prediction.

In Seq2Seq models, training and testing processes are slightly different, with the objective of avoiding error propagation while training. During testing, a token of the input sentence is fed to the encoder in every step. Then, the decoder takes the *start of sentence (SOS)* token, and predicts the next one. Every time a token is predicted, the token is fed into the decoder until the *end of sentence (EOS)* token is predicted, indicating the end of the process. During

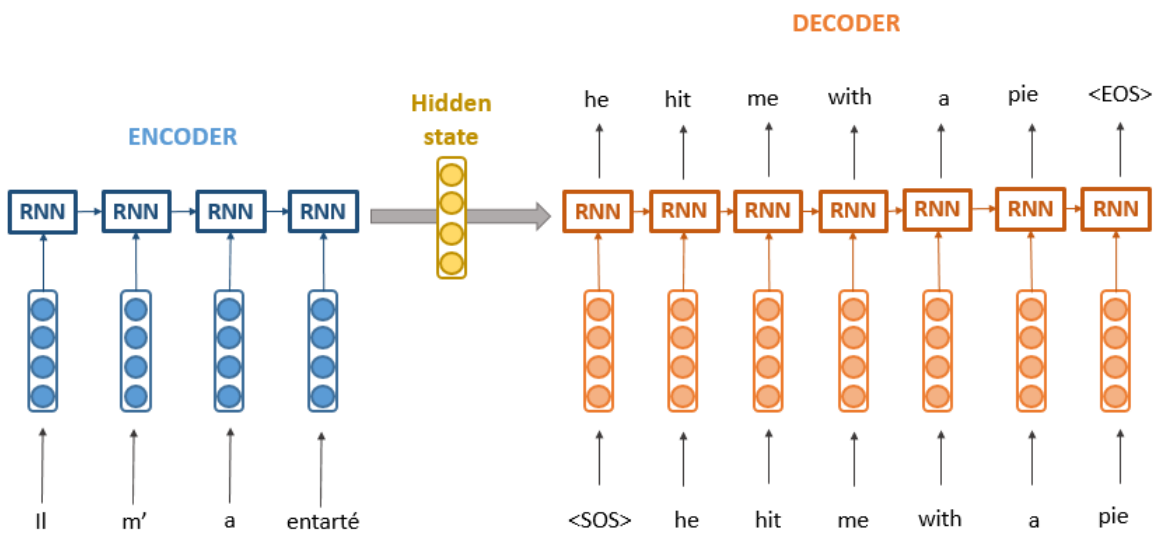


Figure 2.3.1: First NMT model [Sutskever et al., 2014], a Seq2Seq model with an encoder and a decoder that are made of RNNs, during training.

training this process is slightly different, while the encoder works in the same way during training and testing, the decoder doesn't. During training, each predicted token is saved but it is not used to generate the next token, instead the correct token is fed into the decoder. This process, known as *Teacher Forcing*, avoids errors concatenation thus ensuring a more effective training. The saved predicted token is later used to calculate the loss and update the neural network weights.

Using this algorithm results under-performed SMT, since all the information obtained by the encoder had to be stored in only one vector (the hidden state between the encoder and the decoder). And this was not enough to have an algorithm that worked optimally.

2.3.1. Attention

Even LSTM models, that are capable of keeping relevant information, cannot avoid the problem of having to store all the information in a single hidden state vector. In order to solve the bottleneck that this entailed, the attention mechanism was proposed [Bahdanau et al., 2015] (figure 2.3.2). The idea behind this method is allowing the decoder to have access to every hidden state in the encoder, instead of just to the last one. Using encoder's hidden states, a context (or attention) vector is calculated and then used to predict the next word. The steps to do a prediction using the attention mechanism are the following:

1 Calculate an attention score between the decoder hidden state and every encoder hidden state: Let h_i be the decoder's $i-th$ layer hidden state. Let $s_1, ..., s_m$ be encoder's hidden

states. The attention scores of h_i are

$$a_{ij} = f_{att}(h_i, s_j) \qquad j = 1, ..., m$$

Where f_{att} is usually a dot-product.

2 Apply the softmax function on the scalars obtained in the previous step to get the weights:

$$w_{ij} = softmax(a_{ij})$$

3 All the hidden states of the encoder are summed, each one multiplied by its corresponding weight:

$$c_i = \sum w_{ij} \cdot s_j$$

The attention vector c_i is obtained.

4 Concatenate the decoder hidden state (h_i) with the attention vector (c_i) and use it to make the prediction. Additionally, it is also possible to concatenate the encoding of the next token with the attention vector to calculate the next hidden state.

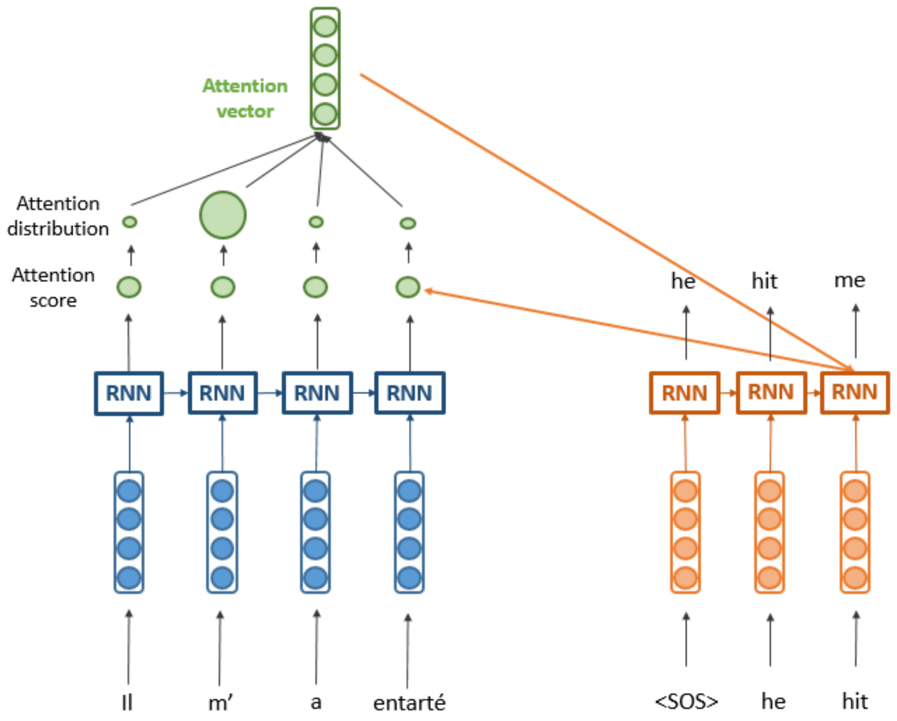


Figure 2.3.2: Attention Mechanism in a seq2seq model with an encoder and a decoder that are RNNs.

After the growth of Seq2Seq and attention-based models for translation tasks, some researchers decided to try this strategy for ASR. Following this idea, the Listen, Attend and Spell (LAS) [Chan et al., 2016] model appeared. This is a Seq2Seq model with attention, like the ones used for text translation tasks. However, models like this were not originally designed for audio processing. This kind of data, which has a longer sequence length than text, can lead to problems with complexity and redundancy of information (these problems will be further commented in chapter 3). For this reason, LAS has a pyramidal RNN in its encoder, that collapses neighbour input tokens into one.

2.3.2. The *Transformer*

The *Transformer* [Vaswani et al., 2017] model was proposed for text translation in 2017. It is a Seq2Seq model, like those described above. While previous models were based on RNNs or LSTMs and used attention as a mechanism to avoid the bottleneck between the encoder and decoder, the *Transformer* uses a different approach: the attention mechanism evolves from being an enhancement of the algorithm to being its basis. If so far the models used encoder-decoder attention, the *Transformer* keeps it but also replaces the RNN with a new self-attention mechanism inside both the encoder and decoder, which allows the analysis of the interaction between tokens of the same input sentence.

Self-Attention

Self attention is a mechanism used to determine how much is a token related to the rest of the tokens in the same sentence. The input of a self attention layer is a matrix containing the embeddings of every input token (numerical representations of words). Assuming there are n input tokens and that the embedding's space is d-dimensional, the input is $X = (x_1, ..., x_n) \in \mathbb{R}^{n \times d}$ where $x_i \in \mathbb{R}^d$ are the embeddings for each of the n input tokens (placed as rows of X). Inside a self-attention layer, three linear projections are applied over X:

$$Q = X \cdot W_Q \qquad K = X \cdot W_K \qquad V = X \cdot W_V$$

where Q, K and V stand for *queries*, *keys* and *values*, and W_Q, $W_K \in \mathbb{R}^{d \times d_k}$, $W_V \in \mathbb{R}^{d \times d_v}$ are matrices learned during the training process.[1] Using Q, K and V the attention matrix is defined as the $softmax$ of the product between Q and K (n^2 dot products between every *query* and *key*) divided by the square root of d_k (scaled):

$$A = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n \times n}$$

The resultant matrix A can be interpreted as the values between 0 and 1, where A_{ij} is the weight that measures how related the tokens i and j are. The larger the weight, the higher the relation between tokens. It is also worth noting that this is the step that gives scaled dot

[1]Note that $d_v = d$ when using single-headed attention.

product self attention its name. The final step is another product:

$$Z = A \cdot V$$

where Z is the matrix of context vectors. This means that $Z = (z_1, ..., z_n) \in \mathbb{R}^{n \times d_v}$ where $z_i \in \mathbb{R}^{d_v}$ is the context vector associated with the $i-th$ token x_i (placed as rows of Z).

The paper improves this attention mechanism by adding multiple attention heads. Using h different W_Q, W_K and W_V matrices the embeddings are projected to $3 \cdot h$ subspaces. This allows attention to focus on different parts of the sentence. The updated process is

$$Q^i = X \cdot W_Q^i \qquad K = X \cdot W_K^i \qquad V^i = X \cdot W_V^i \qquad\qquad i \in \{0, ..., h-1\}$$

$$A^i = softmax\Big(\frac{Q^i \cdot (K^i)^T}{\sqrt{d_k}}\Big)$$

$$Z^i = A^i \cdot V^i$$

To sum up the information in the different Z^i all the matrices are concatenated into a single matrix $(Z^0, ..., Z^{h-1}) \in \mathbb{R}^{n \times h \cdot d_v}$ and then projected using $W_O \in \mathbb{R}^{h \cdot d_v \times d}$ matrix.

$$Z = (Z^0, ..., Z^{h-1}) \cdot W_O$$

Where, again, $Z = (z_1, ..., z_n) \in \mathbb{R}^{n \times d}$ is the matrix of context vectors.

Model Structure

The *Transformer*'s structure (figure 2.3.3) has three main components: the positional encoding, the encoder and the decoder.

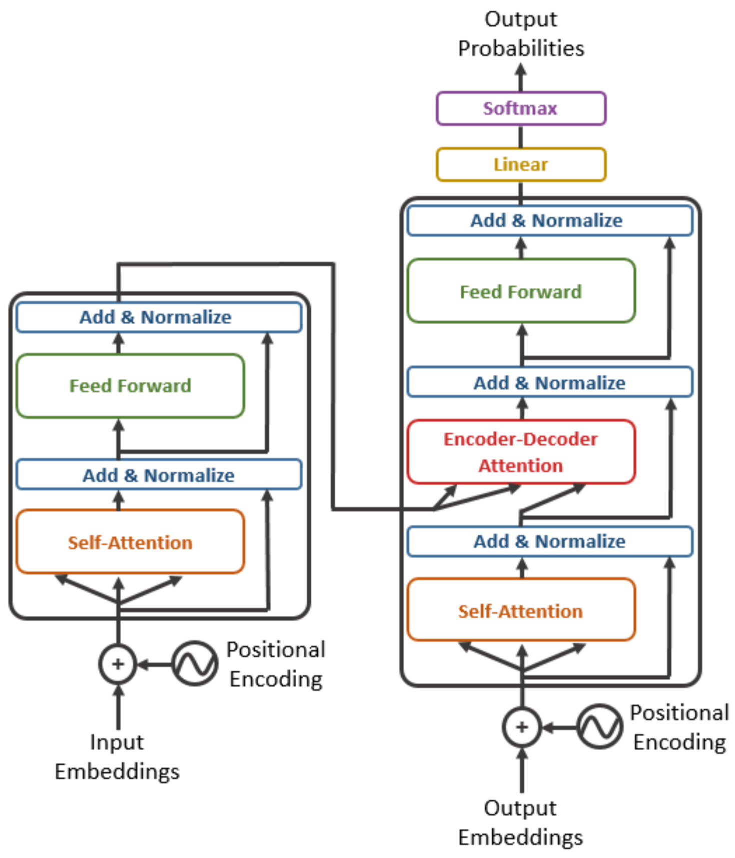


Figure 2.3.3: The *Transformer* architecture, with one encoder and one decoder blocks.

Positional Encoding

As explained before, the *Transformer* model has two main modules: the encoder and the decoder. But unlike the previous models, the encoder and decoder are not based on RNNs. In a RNN, a new token is received at each step, which allows the model to understand the order of the tokens in the sequence. In the *Transformer*, by substituting RNNs with self-attention this positional information is lost. This is why it is necessary to add a positional encoding layer to the input embeddings, preceding both the encoder and the decoder. This layer is in charge of adding information about the position of a token in the sequence. The positional encoding is a vector of length d as the embeddings, so that they can be summed. To create this positional encodings we use the *sine* and *cosine* functions of different frequencies:

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{\frac{2i}{d}}})$$

$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{\frac{2i}{d}}})$$

where *pos* is the position and $i = 1, .., d$. The positional encoding obtained is a sinusoid in each of its dimensions.

The Encoder

The *Transformer*'s encoder is in fact made of stacked encoder modules. Each module has a self-attention layer and a feed forward layer, both followed by an addition and normalization layer. The input of the first encoder block is the sequence of embedded tokens after the addition of the positional encodings, and the following blocks' input is the output of the previous encoder block.

The Decoder

The decoder, as the encoder, is made of stacked decoder blocks. Unlike the encoder, each decoder block consists of three layers instead of two. The first and last layers are the self-attention layer and the feed forward layer, just like in the encoder, but between them there is a new layer, the encoder-decoder attention layer. This layer handles the interaction between the encoder and decoder information using the same algorithm as the self-attention, but V and K are obtained from the output of the last encoder layer and Q is obtained from the output of the previous decoder self-attention layer. As in the encoder, each of the three layers is followed by a normalization and addition layer. In addition, during the training of the model, the self-attention layer uses masked attention, a system that hides those tokens subsequent to the one to be predicted. This ensures a prediction under the same conditions as the one that will be made during testing and avoids misleading results.

2.4. Word segmentation

In both ASR and MT tasks, predictions of tokens are done using a probabilities vector, that gives a weight to each possible token in the vocabulary. But these output tokens are not the same in ASR and MT tasks.

In the case of ASR, prediction is usually done character-by-character. This allows the vocabulary to be small and complete, since it must only be formed by every possible letter (26 in the English alphabet) and two special tokens, the *space* and the *blank*. So a vocabulary of size 28 is enough to predict every possible desired output.

When working with translation tasks, in both text or speech contexts, building a vocabulary is not that easy. In the case of translation, prediction is not made character-by-character. In this task alignment must be done between words according to their meanings, and for this reason, translation models predict word-by-word. This is a problem when trying to build a vocabulary, because in order to have one as complete as the one available for ASR we would have to collect every single possible word in a language. This would result in a reduction of efficiency and confusion when doing predictions, since there would be too many options and some of them would be very similar (for example a word and its plural). On the other hand, we could limit the number of words (for example, using only primitive words and not its derived words) and add an $< Unknown >$ token for every word that does not appear in the vocabulary. But we would end up obtaining too many $< Unknown >$ predictions.

To overcome these problems, the usual approach is using sub-words i.e. parts of words [Sennrich et al., 2016]. For example, these could include primitve words and all possible prefixes and suffixes. But this is not done manually, in order to find the best sub-words possible, the BPE algorithm finds the most common ones. This method reduces the number of $< Unknown >$ predictions while maintaining a vocabulary of an acceptable size.

2.5. Evaluation scores

In this section we will study the evaluation scores used to measure the performance of ASR and ST models.

2.5.1. Word Error Rate

Word Error Rate (WER) is the most common evaluation measure used in speech recognition. It measures the differences between the expected and the obtained output. WER considers three possible errors when doing ASR:

- Deletion: when a word should have been predicted but instead nothing is predicted.
- Insertion: when nothing should have been predicted but instead a word is predicted.
- Substitution: when a wrong word is predicted in place of a correct word.

Word Error Rate is calculated as the proportion of errors per total number of words in the expected output. it is usually represented as a percentage.

$$\text{WER} = 100 \cdot \frac{\text{Deletions} + \text{Insertions} + \text{Substitutions}}{\text{Total of words}}$$

2.5.2. Bilingual Evaluation Understudy score

The Bilingual Evaluation Understudy (BLEU) score [Papineni et al., 2002] is the most common evaluation measure used in translation tasks. Unlike in the case of ASR, where only one correct answer is possible (and therefore it is easy to evaluate the error), in the case of translation there can be different correct outputs. In order to overcome this problem, it is common to compare the output of the model with more than one possible human-made translations. For each comparison the ratio of 1, 2, 3 and 4-grams is calculated and then the results are averaged. Additionally, to avoid short sentences, which obtain better results in the n-gram comparison, a penalty is added if the sentence length obtained is too short. However, BLEU is useful but imperfect; there are many valid ways to translate a sentence and it would be impossible to list them all in the corpus, therefore a prediction can get a low BLEU score and still be correct.

Chapter 3

State of the art

In this chapter we will explain the state of the art of End2End Speech-to-Text algorithms and the new *Efficient Transformers* for long text tasks. These models will be of great importance for the development of the thesis and the quality evaluation of the results.

3.1. Speech-to-Text *Transformer*

The *Transformer* is currently the State of the Art for text translation tasks. After the good results in text and the growing interest in End2End algorithms for Speech-to-Text translation, it seems natural to consider using the *Transformer* also for speech tasks [Cross Vila et al., 2018], but when trying it we must face a main problem: the *Transformer*'s complexity. The cost of calculating the attention matrix is $O(n^2)$, where n is the sequence length. This sequence length is approximately an order of magnitude longer for a spoken input than for an usual text input, therefore the cost in time and complexity of the model can rise to high levels.

To overcome this problem, extensions of the *Transformer* for Speech-to-Text (S2T) tasks, such as End2End ASR and ST, have been proposed. A usual approach is adding convolutional layers before the *Transformer* encoder, as done by Gangi et al. [2019]. Recently, Wang et al. [2020a] introduced a new *S2T Transformer*, based on the addition of 1D convolutional layers before the *Transformer*. These layers reduce the sequence length of the input data by a factor of 4, shortening the audio data, so it is suitable for a regular *Transformer*.

3.2. *Efficient Transformers*

As we have previously seen, to get the attention weights of a sentence with n tokens it is necessary to calculate $Q \cdot K^T$, which has a complexity of $O(n^2)$. This cost in memory and time may be acceptable when working with short sentences but becomes excessively expensive when working with longer texts, and consequently forces to make partitions or to shorten the long texts into smaller sequences. For this reason, research is currently being made on modifications of this algorithm, in aim to reduce its complexity and make the model suitable for longer inputs. These new models are called *Efficient Transformers*.

The *Linformer*

The *Linformer* [Wang et al., 2020b] introduces a variation of the *Transformer* self-attention that reduces the complexity from quadratic to linear. To achieve this reduction, the *Linformer* proves that the attention matrix is low-rank and takes advantage of this finding to propose a new self-attention mechanism with a complexity of $O(n)$. Let P be the attention matrix after applying the *softmax* function over the weights. By applying singular value decomposition into P it can be proven that most of the information in P is concentrated in the few largest singular values, meaning that P is low-rank (for details of the demonstration, see [Wang et al., 2020b]). The straightforward idea to benefit from this result would be to use SVD to approximate P with a new matrix P_{low}, but this requires performing SVD in each attention matrix, which would add complexity to the algorithm. To avoid adding complexity, another method is proposed. The model adds two linear projection matrices ($E^i, F^i \in \mathbb{R}^{n \times k}$) when computing key and value. The resultant model works as follows:

$$Q^i = X \cdot W_Q^i \qquad K = X \cdot W_K^i \qquad V^i = X \cdot W_V^i \qquad\qquad i \in \{0, ..., h-1\}$$

$$A^i = softmax\Big(\frac{Q^i \cdot (E^i \cdot K^i)^T}{\sqrt{d_k}}\Big) \in \mathbb{R}^{n \times k}$$

$$Z^i = A^i \cdot F^i \cdot V^i$$

Concerning k, as proven in the paper, k is independent of the sequence length, and therefore, can be a constant.

The *Longformer*

The *Longformer* [Beltagy et al., 2020], introduces a variation on the original *Transformer*, which achieves a reduction in the cost of the attention calculation operation from quadratic to linear. To achieve this improvement, the *Longformer* algorithm defines a pattern in the attention matrix, specifying for each token those that should be attended to and those that should not. By eliminating some of the attentions between tokens, the number of operations is reduced and an algorithm that scales linearly with the input sequence length is obtained. *Longformer*'s attention pattern consists of different components:

- **Sliding Window (figure 3.2.2):** It is the main component of the attention pattern, and it is based on the idea of the importance of local context. With this component, an attention window of fixed size is placed around each token. Given a fixed window size w each token will attend to the $\frac{1}{2}w$ tokens on each of its sides. In addition to the local context, adding several stacked attention layers achieves an effect similar to the one used in CNNs, that allows the last layers to receive information from the whole sentence and not only from the tokens inside its window. Stacking l attention layers would provide a receptive field of size $l \times w$ at the top layer.
- **Dilated Sliding Window:** In order to increase each token's attention field without increasing the complexity, a variant of the sliding window was proposed. For every token, the dilated sliding window also attends to $\frac{1}{2}w$ tokens on each side but leaves gaps of size d. Consequently, the receptive field in the last layer has a size of $l \times d \times w$.
- **Global Attention (figure 3.2.3):** For some NLP tasks there exist special tokens, such as [CLS] in the case of BERT, that would not be attended enough just by using the sliding window. In these cases, global attention is added to pre-selected input tokens. This global attention is symmetric: the selected token will attend to every other token and vice versa.

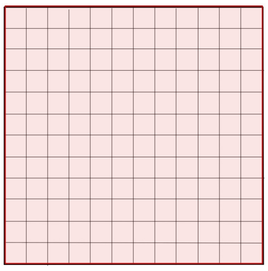


Figure 3.2.1: Full Attention

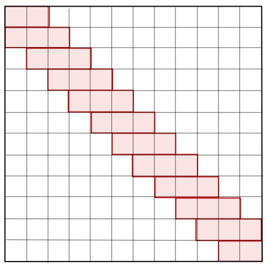


Figure 3.2.2: Sliding Window

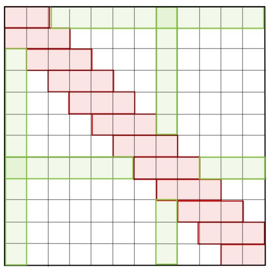


Figure 3.2.3: Sliding Window and Global Attention

The *Big Bird*

The *Big Bird* [Zaheer et al., 2020] introduces a variation on the *Longformer*, that adds more attention and improves the performance while keeping linear complexity. The *Big Bird* proposes a new component in the *Longformer* attention pattern:

- **Random Blocks (figures 3.2.4 and 3.2.5):** Just by using a sliding window and global tokens we might be losing important contextual information. To try to recover part of this information some random queries and keys are chosen to calculate their attention weight.

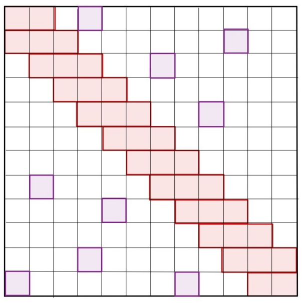


Figure 3.2.4: Sliding Window and Random Attention

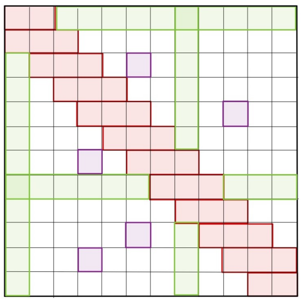


Figure 3.2.5: Sliding Window, Global and Random Attention

The *Reformer*

The *Reformer* [Kitaev et al., 2020] introduces a modification that reduces the complexity from $O(n)$ to $O(n \cdot \log n)$ by substituting regular self-attention with one that uses locality-sensitive hashing. To use LSH attention we impose $Q = K$. The *Reformer* notices that the costly operation of attention is QK^T, but we only use the result of this product after applying the *softmax* function over it. When applying the *softmax* the largest elements remain but the low ones get close to 0. This method proposes to calculate the product between a q and a k only when they are similar and avoids doing it when they are very different, since in this second case the result of calculation would vanish when applying the *softmax*. To find which q and k are similar the *Reformer* proposes to use locality-sensitive hashing [Andoni et al., 2015], a hashing method that assigns nearby vectors to the same hash with high probability. In order to do this, vectors are projected to a sphere that is divided into buckets. The hashing algorithm applies random rotations to the projected points and assigns them to the same hash if after every random rotation they are positioned in the same bucket.

The *Routing Transformer*

The *Routing Transformer* [Roy et al., 2021] is a *Transformer* modification that reduces complexity to $O(n^{\frac{3}{2}})$. This *Efficient Transformer* is based on the same idea as the *Reformer*, this is avoiding calculating the attention weights that will tend to 0 when applying the *softmax* function. The *Routing Transformer* uses the classical clustering method *k-means* to assign every query and key to a cluster. The initial centroids for this clustering are parameters and are learned during the training. After the clustering, attention is only calculated between queries and keys that belong to the same cluster.

The *Synthesizer*

The *Synthesizer* [Tay et al., 2020] proposes a new approach to the *Transformer*, proving that the self-attention might not be as important as it seems. In this paper, researchers find that random alignment works competitively and therefore that weights calculated from queries and keys are useful but not that important. Two different *Synthesizer* models are proposed:

- **Dense *Synthesizer*:** This *Synthesizer* is conditioned on each input, but not on the rest of the sentence. Every token predicts a vector of length equal to the sequence length, that contains a weight for every token in the sequence. These weights are only influenced by the current token and, depending on what this token is, a weight is decided for every position of the sentence.

- **Random *Synthesizer*:** This *Synthesizer* is not conditioned by input tokens, and the attention weights are fully random. These values can be either fixed or trainable parameters.

Comparison of *Efficient Transformers*

To close this section, we will consider the table 3.2.1, with other information about this *Efficient Transformers* regarding other features that are not the algorithm. In this table we compare their complexity, the architecture they are used in and the difficulty of their implementation[1].

	Complexity	Architecture	Implementation
Linformer	$O(n)$	encoder	medium
Longformer	$O(n)$	encoder	easy
Big Bird	$O(n)$	encoder	easy
Reformer	$O(n \cdot \log n)$	encoder-decoder	medium
Routing T.	$O(n^{\frac{3}{2}})$	encoder	difficult
Synthesizer	$O(1)$	encoder-decoder	medium

Table 3.2.1: *Efficient Transformers*

[1]An algorithm is classified as 'easy' if its implementation can be found in *Hugging Face*, 'medium' if its in another *git* repository or 'difficult' if there is no implementation available.

Chapter 4

Methodology

Once we have seen the required background, and the current state-of-the-art we will go through the specifics of our proposal. In this thesis, we present a new method for Speech-to-Text translation tasks, that can also work for ASR. We propose a new modified *Transformer*, where the encoder is an *Efficient Transformer* and the decoder is a traditional *Transformer* decoder. By using an *Efficient Transformer* in the encoder the model can handle long inputs, and therefore we can delete the convolutional layers used in the implementation by Wang et al. [2020a]. This way, the encoder has access to the full spectrogram and no information is lost.

4.1. Overview

To create and test our algorithm we will follow the steps below:

- ***Efficient* Transformer Selection:** Compare and choose one of the *Efficient Transformers* that have been analyzed in chapter 3.
- **Building the model:** Build the new model with the chosen *Efficient Transformer* encoder and a regular *Transformer* decoder.
- **Training and Evaluation:** Once the model is built, we plan the experiments that will be trained. After the training we will evaluate the results.

4.2. *Efficient Transformer* Selection

Although the objective of this project is not to translate long texts, we believe that *Efficient Transformers* could be useful to deal with spoken sentences, because they could help to address the problem of large sequence lengths.

We have studied different *Efficient Transformer* models. Since we are interested in modifying only the encoder, we would prefer to choose one of the models that have already been tried this way. Therefore we will not use the *Reformer* or the *Synthesizer*. Both the *Longformer* and the *Big Bird* share a similar attention pattern, that is based in a sliding window. We believe this feature could be profitable for a sound task, since it could help in the sound processing, and could also be helpful during training since these are models that allow modifications and adjustments in their parameters. On the negative side, the sliding window could be a constraint because it might focus on redundancies instead of on the sentence content.

Moreover, we would like to choose a model with the lowest complexity possible, since we will be dealing with high sequence lengths, and both the *Longformer* and the *Big Bird* have a linear complexity. To sum up, because of their attention pattern, the fact that they are encoder-based models, and their linear complexity we believe that the best possible choices are the *Longformer* and the *Big Bird*. Additionally, these two models allow an easy implementation, since they are both available in *Hugging Face* [1].

As we have seen, the *Longformer* and the *Big Bird* provide a very similar attention model. In the case of the *Longformer*, the attention pattern is based in a sliding window, that can be dilated or not, and it is in charge of the local attention. Both models use global attention too, that gives the appropriate importance to special tokens that can be needed depending on the task the model is being used for. Additionally, the *Big Bird* adds attention weights between some randomly chosen queries and keys. Since our task doesn't include any special token, we are not interested in adding global attention in neither the *Longformer* nor the *Big Bird*. Our initial objective was to try both models with the same attention window, in aim to see if adding random attention weights was useful or not when processing speech. We also wanted to try different attention window sizes and a dilated window, to see how adding or reducing local attention could affect the results.

After studying the implementations of the *Longformer* and the *Big Bird* available in *Hugging Face* we faced a main problem: The *Big Bird* model was restrictive with its parameters, and it was compulsory to add global attention to the tokens at the beginning of the sentence. This could complicate the comparison between the models, and most importantly, could lead to bad results, since the first milliseconds of an audio are usually silent, and this implementation would give them a too much importance. For this reason, we decided to choose the *Longformer* for our main experiments. Additionally we performed some experiments with the *Big Bird*, and they can be found in Appendix A.

[1]Hugging Face is an open-source library for natural language processing algorithms (`huggingface.co`)

4.3. Building the Model

Once the *Efficient Transformer* is chosen, we can start to build our model. To do it we will mainly use two libraries:

- *Fairseq*, a sequence modeling toolkit by *Facebook Research AI* written in *PyTorch* that allows researchers and developers to train models for text generation tasks such as translation, summarization, language modeling.
- *Hugging Face*, an open-source library for natural language processing algorithms.

Our model has three main modules, which we have to choose: the positional encoding, the encoder and the decoder.

Positional Encoding

Fairseq has two available positional encodings:

- Learned: In this positional encoding parameters are learned during the training of the model.
- Fixed: It is a sinusoidal encoding like the one used before the original *Transformer*, described in chapter 2.

For our work, we decide to use the sinusoidal encoding, since we want our model to be as similar as possible to the original *Transformer*, so it is considered a variation of it but for speech tasks. Additionally, by choosing this fixed encoding we avoid an increase of the trainable parameters of the model, making the training faster.

Encoder

We will use the encoder of the *Longformer* model available in *Hugging Face*. This encoder has the same layers as a *Transformer* encoder, but the self-attention will be the *Efficient* one. We will use a regular sliding window, since the dilated one is not implemented, and we won't use global attention, since it is not suitable for the tasks we are working on. We will try different window sizes.

Decoder

We will use a regular *Transformer* decoder, available at *Fairseq*.

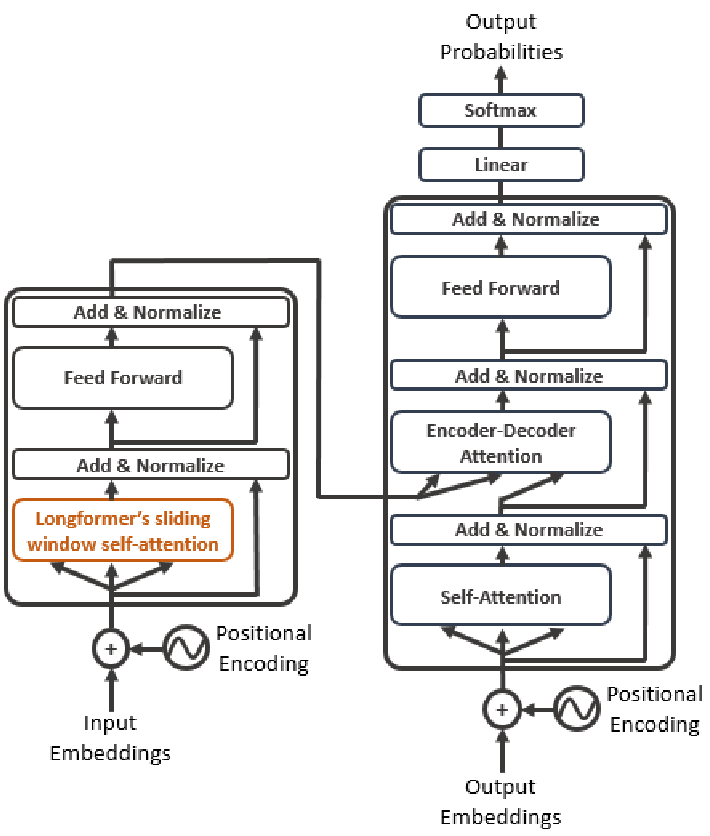


Figure 4.3.1: We propose a model with *Longformer*'s sliding window in the encoder's self-attention.

However, this model could have an inconvenience. In speech-to-text translation, as in every translation task, when calculating the encoder-decoder attention, it is important to align the input with the output. But in this case, while the input is a spectrogram, the output is text, and this sequence length mismatch can be a complication when doing a correct alignment. This was not a problem in *Fairseq*'s *S2T Transformer*, since they reduce the sequence length before using the *Transformer*, and therefore there is no sequence length difference between the input and the output.

In our proposed model the encoder is replaced with an *Efficient Transformer*, which is suitable to work with high sequence lengths without causing a sharp increase in complexity. However, this model does not solve the possible problem of alignment. This happens because, in contrast as with the *S2T Transformer*, in our case the input is still a spectrogram and the output has to be text, therefore, when calculating the encoder-decoder attention, the alignment will have to be done between these two sequences of very different lengths.

To solve this problem we propose to add a convolutional layer after the last stacked encoder. This will reduce the encoder output length by half, making it more similar in size to the predicted text sentence, eventually solving the lengths discrepancy problem. This approach is close to the one used in the *S2T Transformer*, but in our case there is only one convolutional layer and it is placed after the encoder, instead of before. We believe this could be profitable since we let the encoder learn directly from the spectrogram and no information is lost, but at the same time we reduce the complexity of the model and address the alignment problem of mismatching sequence lengths.

4.4. Training and Evaluation

We propose to train models with different attention window sizes, and with or without a convolutional layer after the encoder. We train our models for an ASR task, and then for ST using the ASR pre-trained encoder [Bérard et al., 2018]. This is done because in a ST task the encoder must learn two very different jobs at the same time: how to analyze audio features and how to understand the semantics of a sentence. The decoder must learn two tasks too, translating and language modeling, but these are strongly related, unlike the ones carried out by the encoder. Instead, when doing ASR, since alignment is monotonic and semantics are not the main issue, the encoder can focus on learning audio modeling. Additionally, the amount of data available for ASR training is higher than for ST, what can improve the results. By doing the ASR pre-training, our model already knows how to model audio before the ST training, and therefore the algorithm is able to focus on learning semantic and translation tasks.

Once the models are trained, we will test them on new data. We will calculate the WER for the ASR task and the BLEU score for the ST task.

Chapter 5

Experiments and Results

In this section we describe the different experiments that have been carried out, their parameters and results, and the settings. To evaluate the quality of our results, we will compare them with a baseline model that will be the *S2T Transformer* available in *Fairseq* and described in chapter 3. In particular, we will use the *s2t_transformer_s*, the one used by Wang et al. [2020a]. To simplify the notation we will denote this model as *s2t_transformer*. In the case of our models, we will call them *s2t_longformer* and *s2t_longformer_2x*, being the first one the base model, and the second one the variation with a convolutional layer after the encoder.

5.1. Settings

Before discussing the conducted experiments and the results obtained with each of them, we will describe some of the conditions that will be decisive, such as the data, the experimental framework, the details of the model implementation and the training parameters.

5.1.1. Dataset

Regarding the data, we will apply our Speech-to-Text algorithm to the *MuST-C* dataset [Cattoni et al., 2021]. This dataset was created after the popularity growth of End2End ST, with the aim to confront the scarcity of public data that researchers were facing when training these new models. Cascade solutions have many and varied data to train each of their modules, but End2End ST datasets were small and of limited language coverage. *MuST-C* dataset is a Multilingual ST Corpus (*MuST-C*) built using TED talks in English. The dataset includes a corpus for translation from English into 14 different languages, that belong to different families, including Dutch, French, German, Italian, Portuguese, Romanian, Russian, or Spanish. It contains at least 237 hours of transcribed recordings (430 on average) for each of the available languages. Furthermore, the data is free and of good quality, and includes a variety of topics and speakers.

For our work we will be using the English-German section included in the *MuST-C* dataset, which contains data for English ASR and for English-German ST.

5.1.2. Training Environment

For our trainings, we will use UPC's *Calcula* server, powered by NVIDIA GeForce RTX 2080 Ti GPUs. We will use one GPU for each training.

5.1.3. Speech-to-Text *Longformer* Implementation

To get the most realistic comparison possible between our results and the ones obtained with the *s2t_transformer* by Wang et al. [2020a], we try to create a model as similar as possible to theirs. We build our model with 6 encoder layers and 12 decoder layers. We apply a normalization layer before each decoder layer. In both the encoder and the decoder, we use 4 attention heads, the embedding dimension is 256, and in the FFNN layers it is 2048. The decoder output dimension is 256, the same as the decoder embedding dimension. We use a dropout probability of 0.1 in both the attention weights and in the FFNN activations. We use ReLU as the activation function for the FFNNs.

Additionally, we add some extra parameters, that are specific of our model, regarding the size of the attention window and the convolutional layers. For the convolutional layers of *s2t_longformer_2x* we use a kernel of size 5. The attention window size will be defined specifically for each experiment.

5.1.4. Training parameters

To ensure a reliable comparison, all ASR and ST experiments have been respectively performed under the same conditions and parameters. Specifically, we have tried to use the same parameters as in the implementation by Wang et al. [2020a], when possible. In ASR trainings we use 4 CPUs and 2 workers to load the data. We fixed a maximum of 20000 tokens per batch. We used Adam optimizer and a learning rate of $1 \cdot 10^{-3}$ with an inverse square root scheduler. We applied a warm-up for the first 10000 updates. We clipped the gradient to 10 to avoid exploding gradients. We used label smoothed Cross-entropy as a loss function, with a smoothing factor of 0.1. We used an update frequency of 16, simulating the use of 16 GPUs. We fix a maximum of 100000 updates for every training. In ST trainings we use use the same parameters as for ASR, but for the learning rate, that will be $1 \cdot 10^{-3}$, as done by Wang et al. [2020a].

5.2. Experiments description

As discussed in chapter 4, we will train our models for an ASR task and then for a ST task, using the ASR pre-trained encoder.

5.2.1. Automatic Speech Recognition

We will try the following experiments for an ASR task:

- Attention Window = 512: *s2t_longformer*, *s2t_longformer_2x*
- Attention Window = 76: *s2t_longformer*, *s2t_longformer_2x*
- Attention Window = 60: *s2t_longformer*, *s2t_longformer_2x*
- Attention Window = 48: *s2t_longformer*, *s2t_longformer_2x*

We begin our experiments choosing an attention window of size equal to 512 tokens, since this is the default value given in *Hugging Face* implementation. After 80 epochs the model doesn't converge, what shows that 512 is not an appropriate size for our task. A possible explanation for this result is that this model has been created for text (long documents), not for speech, and therefore the optimal parameter is not necessarily the same for both tasks. We believe that this attention window could be too big for a speech task, since we are attending to an interval of approximately 5 seconds, which is too wide for audio features extraction. To address this problem, we decide to reduce the attention window size to a length of 48 tokens (approximately 0.5 seconds). This modification indeed solves the convergence problem and reaches a result that is close to the baseline (**??**). Nevertheless, we believe that this window size could be too small to work well in a ST task, since it does not allow the model to see much context. This is not very relevant in ASR tasks, because alignment is monotonic, but can be a problem in future ST trainings where alignment with the rest of the words in the sentence is fundamental. For this reason, we decide to try a window of size 76. This experiment works well with the *s2t_longformer_2x*, but is unstable in the *s2t_longformer* training. Again, we believe this could be as a consequence of a too big window, and therefore we decide to try another model, with an attention window of size 60. In contrast with the previous experiment, this one works well with the *s2t_longformer*, but is unstable with the *s2t_longformer_2x*, what shows that the window might be too wide too. Therefore we decide to stop the ASR experiments and keep the ones that already work.

All these experiments have been trained using *MuST-C* dataset during 80 epochs, in a time of approximately 4 days. In table 5.2.1 we find the best WER obtained in each experiment and in figure 5.2.1 we can see the evolution of the WER along every training epoch.

	Attention Window Size	WER
s2t_transformer	-	13.31
s2t_longformer	512	does not converge
s2t_longformer_2x	512	does not converge
s2t_longformer	76	unstable
s2t_longformer_2x	76	15.00
s2t_longformer	60	14.99
s2t_longformer_2x	60	unstable
s2t_longformer	48	15.31
s2t_longformer_2x	48	15.06

Table 5.2.1: ASR results after the performed experiments

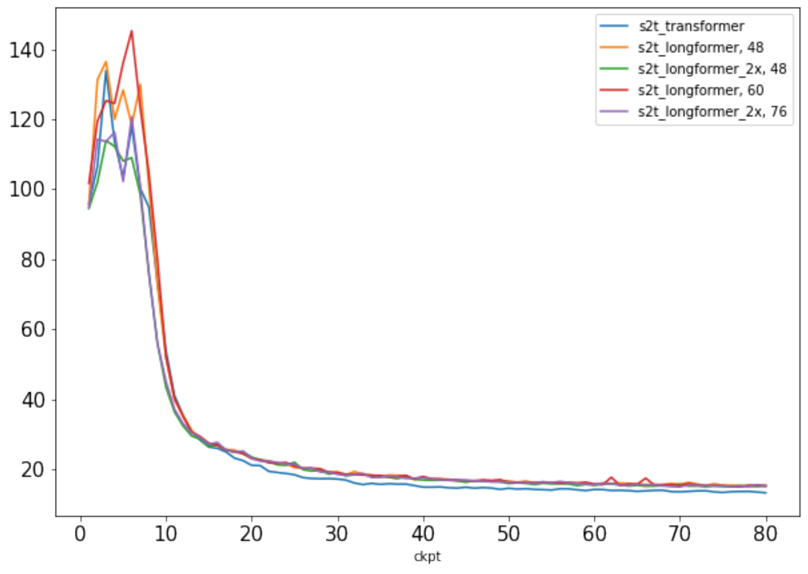


Figure 5.2.1: ASR over epochs.

5.2.2. Speech Translation

In this section, we describe the ST experiments that have been carried out. In order to train a model for ST we needed the pre-trained encoder obtained after the ASR training. For this reason we will only try models that worked well for ASR. It is also worth noting that since ST is a harder task than ASR (because of alignment), a model that has not worked well for ASR is highly unlikely to work for ST.

For these reasons, the only models that were trained for ST are:

- Attention Window = 76: *s2t_longformer_2x*

- Attention Window = 60: *s2t_longformer*
- Attention Window = 48: *s2t_longformer*, *s2t_longformer_2x*

We perform these experiments for a ST task. We have done a training using *MuST-C* dataset during 160 epochs, in a time of approximately 7 days and 19 hours for each experiment. In table 5.2.2 we find the best result of each experiment, and in figure 5.2.2 we find the BLEU score evolution in every epoch.

	Attention Window Size	BLEU
s2t_transformer	-	22.41
s2t_longformer_2x	76	20.64
s2t_longformer	60	20.34
s2t_longformer	48	20.49
s2t_longformer_2x	48	20.45

Table 5.2.2: BLEU results after the performed experiments

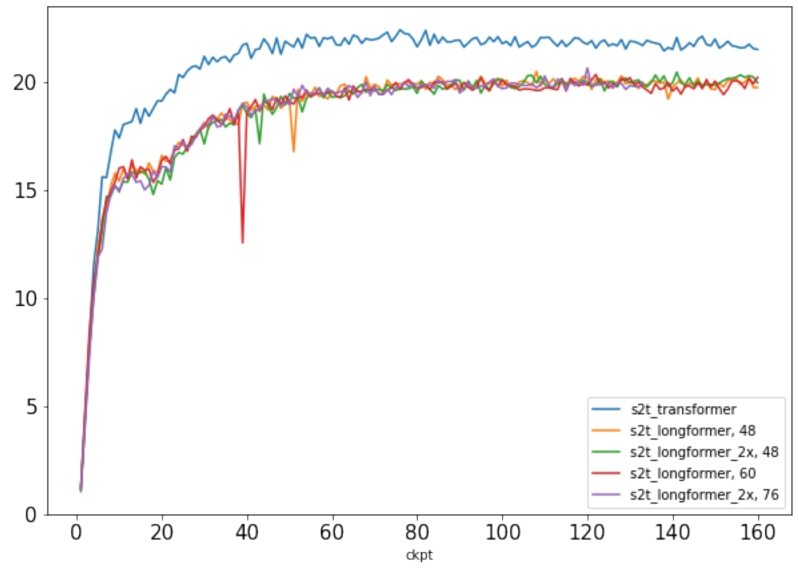


Figure 5.2.2: BLEU over epochs.

5.3. Results Analysis

During ASR trainings, we have seen that large attention windows can be harmful for audio processing since they do not let the encoder focus properly on the audio feature extraction.

We have also seen that the *Longformer* instabilities reported by Beltagy et al. [2020] have been an obstacle for the training process, especially aggravated with long window sizes that make the model more complex. Even so, we have been able to train 4 different models that work and that offer competitive results in ASR. We have used the pre-trained encoders of these 4 models for new trainings on ST. As in the case of ASR, the models have not been able to reach the baseline results but still offer promising results. Additionally, we have seen that adding a convolutional layer is not especially useful, in contrast to what we had hypothesized.

Chapter 6

Conclusions and future work

In this thesis, we have first studied the background and history of ASR, MT, and ST tasks, and the current approaches.

We have seen that Direct Speech-to-Text translation is a research field in development, since current results are poor, and therefore there is still work to be done before these algorithms can be used in real-life applications. We have contributed to the field with our proposed model, which shows an innovative approach.

Our model, consists of a variation of the original *Transformer* that makes it suitable for ST tasks. It replaces the encoder's self-attention with the attention pattern proposed by the *Longformer*, based on a sliding window that can be useful for audio processing. This allows the model to work directly with a spectrogram without losing any information in the convolutional layers before the *Transformer*, as happens in the models proposed by Gangi et al. [2019] and Wang et al. [2020a].

Unlike what we thought at the beginning of the thesis, alignment between speech and text has not been a problem for the performance of the model. Instead, we had to face other complications that we did not expect: *Longformer*'s natural instability aggravated by large attention window sizes caused a hindrance in the model's training.

The main goal of this thesis was to assess whether this model could give competent results. Finally, our model did not reach the baseline results but got a close performance: a WER of 14.99 (compared to 13.31 from the baseline system) and a BLEU of 20.64 (compared to 22.41 from the baseline system), which we consider a great starting point for a promising research path.

After these results, we believe it would be appropriate to study possible modifications in the training parameters, to get a more stable training even for wider attention windows. It would also be appropriate to try a different approach when defining the attention window size: using one of variable length that could be smaller in the first encoder layers, to help with the audio features extraction, and wider in the last layers, to obtain more information about the context for the ST task. Additionally, another future work could be modifying *Big Bird*'s *Hugging Face* implementation to deactivate global attention and to have a less restrictive model, so we could use it to study the effect of adding random attention to the *Longformer* approach. Finally, we could try other *Efficient Transformers*, such as the *Linformer*, that has been tried in encoders and therefore could be suitable for a model like ours.

Bibliography

The IWSLT 2019 Evaluation Campaign, November 2019. Zenodo. doi: 10.5281/zenodo.3525578. URL https://doi.org/10.5281/zenodo.3525578.

Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. *arXiv preprint arXiv:1509.02897*, 2015.

Ebrahim Ansari, Amittai Axelrod, Nguyen Bach, Ondřej Bojar, Roldano Cattoni, Fahim Dalvi, Nadir Durrani, Marcello Federico, Christian Federmann, Jiatao Gu, Fei Huang, Kevin Knight, Xutai Ma, Ajay Nagesh, Matteo Negri, Jan Niehues, Juan Pino, Elizabeth Salesky, Xing Shi, Sebastian Stüker, Marco Turchi, Alexander Waibel, and Changhan Wang. FINDINGS OF THE IWSLT 2020 EVALUATION CAMPAIGN. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 1–34, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.iwslt-1.1. URL https://www.aclweb.org/anthology/2020.iwslt-1.1.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Alexandre Bérard, Laurent Besacier, Ali Can Kocabiyikoglu, and Olivier Pietquin. End-to-end automatic speech translation of audiobooks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6224–6228, 2018. doi: 10.1109/ICASSP.2018.8461690.

Roldano Cattoni, Mattia Antonino Di Gangi, Luisa Bentivogli, Matteo Negri, and Marco Turchi. Must-c: A multilingual corpus for end-to-end speech translation. *Computer Speech Language*, 66:101155, 2021. ISSN 0885-2308. doi: https://doi.org/10.

1016/j.csl.2020.101155. URL https://www.sciencedirect.com/science/article/pii/S0885230820300887.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

Laura Cross Vila, Carlos Escolano, José A. R. Fonollosa, and Marta R. Costa-Jussà. End-to-End Speech Translation with the Transformer. In *Proc. IberSPEECH 2018*, pages 60–63, 2018. doi: 10.21437/IberSPEECH.2018-13. URL http://dx.doi.org/10.21437/IberSPEECH.2018-13.

Mattia A. Di Gangi, Matteo Negri, and Marco Turchi. Adapting Transformer to End-to-End Spoken Language Translation. In *Proc. Interspeech 2019*, pages 1133–1137, 2019. doi: 10.21437/Interspeech.2019-3045. URL http://dx.doi.org/10.21437/Interspeech.2019-3045.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1764–II–1772. JMLR.org, 2014.

W. John Hutchins, Leon Dostert, and Paul Garvin. The georgetown-i.b.m. experiment. In *In*, pages 124–135. John Wiley Sons, 1955.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

H. Ney. Speech translation: coupling of recognition and translation. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 1, pages 517–520 vol.1, 1999a. doi: 10.1109/ICASSP.1999.758176.

H. Ney. Speech translation: coupling of recognition and translation. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 1, pages 517–520 vol.1, 1999b. doi: 10.1109/ICASSP.1999.758176.

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

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4055–4064. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/parmar18a.html.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

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

C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, jan 1949. doi: 10.1109/jrproc.1949.232969. URL https://doi.org/10.1109/jrproc.1949.232969.

SS Stevens, J Volkmann, and EB Newman. The mel scale equates the magnitude of perceived differences in pitch at different frequencies. *Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Changhan Wang, Yun Tang, Xutai Ma, Anne Wu, Dmytro Okhonko, and Juan Pino. fairseq s2t: Fast speech-to-text modeling with fairseq. *arXiv preprint arXiv:2010.05171*, 2020a.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020b.

Ron J. Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. Sequence-to-sequence models can directly translate foreign speech. In *Proc. Interspeech 2017*, pages 2625–2629, 2017. doi: 10.21437/Interspeech.2017-503. URL `http://dx.doi.org/10.21437/Interspeech.2017-503`.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf`.

Appendix A

The *Big Bird*

In this appendix we show the results obtained using The *Big Bird* [Zaheer et al., 2020].

A.1. Methodology

As discussed in chapter 4, the *Big Bird* is a model that provides the same advantages as the *Longformer*: it has linear complexity, it has been used in encoders, and it has an attention pattern that could be useful for audio processing. This attention patter, that was seen in detail in chapter 3, is similar to the one used in the *Longformer*, but adds attention between randomly chosen queries and keys. This is particularly interesting because we would like to study the effect of adding random attention for a speech task by comparing the results of both the *Longformer* and the *Big Bird*.

To build our model we follow the same approach we used with the *Longformer*. We build a variation of the *Transformer* that will replace encoder's self-attention layer with one using *Big Bird*'s attention pattern. We will use *sinusoidal* positional encodings and a regular *Transformer* decoder.

A.2. Implementation

To build our model we will use an implementation available in *Hugging Face*. This implementation is based in a modification of the original *Big Bird* model that uses blocks in the attention pattern, in aim to make the complexity of the model lower. This blockified structure was initially proposed in the original *Big Bird* paper Zaheer et al. [2020] and packs queries and keys in blocks of size $b \times b$. Then uses this blocks to define the attention pattern, instead of doing it over individual queries and keys.

But *Hugging Face* adds some restrictive conditions to this attention pattern. It forces the model to use global attention in the first $2 \cdot b$ tokens, and forces the attention window to be of size $3 \cdot b$ (see this pattern in figure A.2.1) . This can be a hitch for our model, since we can not create a pattern exactly as the one we created with the *Longformer* but with random tokens, and also forces us to use global attention in the first milliseconds of our audio, which are usually silent. This means that results will probably be poor because of the global attention. But even if they are good, we will not be able to compare properly both the *Longformer* and the *Big Bird*, because there will be many differences between the attention patterns, not only the random tokens, as we wanted.

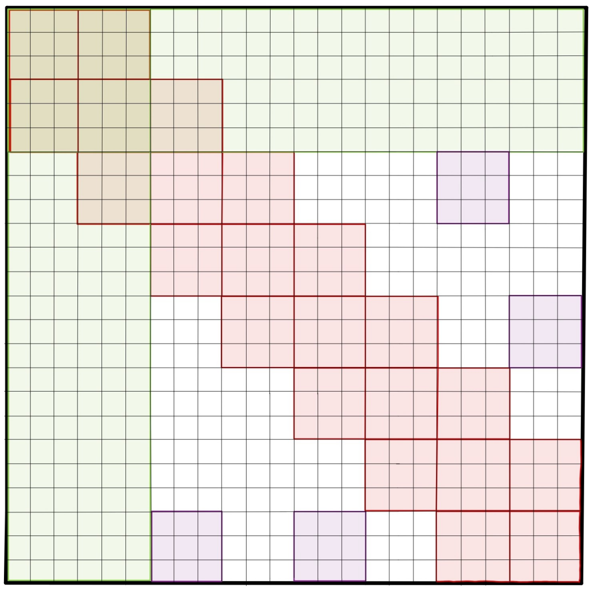


Figure A.2.1: Big Bird attention pattern with blocks (*block_size*=3, *random_blocks*=4)

A.3. Experiments and Results

For our experiments we will use the same dataset, experimental framework, model structure, and training parameters [1] as the ones used in chapter 5 for the experiments performed with the *Longformer*. In the case of the *Big Bird* we have to define two extra parameters, the block size and the number of random blocks. As said above, we don't need to define the attention window size, because it is set to $3 \cdot block\_size$, and neither the global attention, that is set to the first $2 \cdot block\_size$ tokens.

[1]The training parameters are the same but for the update frequency, that is 8, and the label smoothing, that is set to 0.

A.3.1. Automatic Speech Recognition

As we did with the *Longformer*, we will start training out model for an ASR task, and then for ST using the pre-trained encoder.

For our first experiment we try using the default parameters (*block_size* = 32, *random_blocks* = 3). As happened with the *Longormer*, the results with this parameters are poor, probably because they are optimized to work well in a text translation task, not for speech. We decide to reduce the block size, as we did to improve *Longofmer*'s results too. We decide to use a block size of 16, because this will result in a total attention window of size $16 \times 3 = 48$, and 48 was a one of the windows that worked better with the *Longformer*. Again, the *Big Bird* performs badly with this window, in contrast with the *Longformer*. This leads us to believe that our hypothesis about global attention, and that it could be a problem because it forced the model to give importance to silent tokens, was right. Therefore we decide to stop the experiments.

In the following figure (A.3.1) we sum up the results of the diferent experiments done using the *Big Bird*, and the results of the baseline (*s2t_transformer*).

	Block Size	WER
s2t_transformer	-	13.31
s2t_longformer	32	53.88
s2t_longformer_2x	32	35.27
s2t_longformer	16	54.65
s2t_longformer_2x	16	34.96

Table A.3.1: ASR results after the performed experiments.

Since the *Big Bird* does not work well in ASR, we do not have pre-trained encoder and therefore we can not train the model for ST.

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Universitat Politècnica de Catalunya

Barcelona, 2021