Human Language Technologies
Implementing NMT models

Dalla Noce Niko, Ristori Alessandro

Master Degree in Computer science.

Curriculum: Artificial Intelligence.

n.dallanoce@studenti.unipi.it, a.ristori5@studenti.unipi.it .

Human Language Technologies, Academic Year: 2020/2021

Date: 7/12/2021

`https://github.com/nikodallanoce/HLT`

Abstract

The report starts with a long, but needed introduction to the main concepts that we've seen during the course of our work and it shows what was our purpose and how we reached it. Then it shows the results obtained by the models we tried giving at the same time the reasons of their performances. The report concludes with our final considerations on what and how we could have done more.

Contents

List of Figures

1 Introduction

Before we dive into our work, we briefly talk about our task and introduce the key concepts and models that we used during our project.

1.1 Task

The purpose of our task was to understand and implement NMT models and do a comparison between them and state-of-art models, to achieve that we had to build an encoder-decoder from scratch and then we exploited the transformers available on huggingface.co [2021] to improve the performance of our model. The best one was then pushed even further by feeding it more records during training and letting it run for more epochs in hope to achieve a better revisited SacreBLEU score by Goyal et al. [2021]. As for the datasets we used the english-italian one from manythings.com [2021] and the en-it europarl Koehn et al. [2005], which, combined, resulted in more than 2 milion records, not enough to reach the perfomances of state-of-art models, but still enough to achieve nice results.

Over the next subsections, we'll talk about NMT and the models used for our work and their architecture, to give a better insight on their peculiarities and how we tried to exploit them.

1.2 Neural Machine Translation

Neural Machine Translation, or simply NMT, is an approach to translate sentences from one language to another one (e.g.: from english to italian) by using a single neural network whose architecture is based on the encoder-decoder paradigm. In its easiest form the network is composed by two RNNs (LSTMs or GRUs), one for the encoder and one for the decoder, that are trained jointly, in an end-to-end fashion, for maximizing the translation performance; such model composed of two RNNs has a very straight forward way of working:

- The **encoder** receives one sentence from the source language and produces an encoding from it, such encoding will provide the fist hidden state of the decoder;
- The **decoder** receives the encoding produced by the encoder and builds the sentence in the target language one word at time conditioned on the encoding.

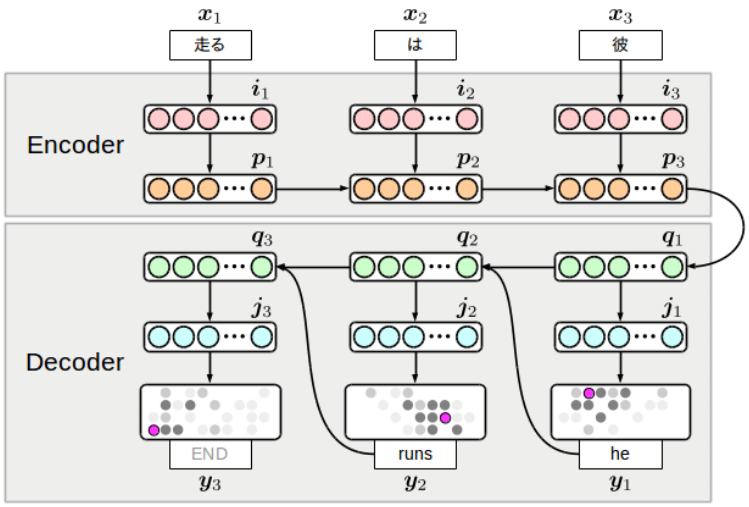


Figure 1.2.1: Standard NMT model

This extremely simple model beat all the previous paradigms by a large margin providing more fluent and precise context-wise translations and its training was even easier since both parts, encoder and decoder, are trained together giving less headaches on optmizing every single sub-module.

1.3 Transformer

In 1.2 we've talked about a very simple model composed by two RNNs, one for the encoder and one for the decoder, but a more powerful and complex architecture was introduced by Vaswani et al. [2017] called transformer that disposes of any recurrent or convolutional layer and relies solely on attention. The transformer still follows the encoder-decoder paradigm, but both parts are now composed by multiple layers and sub-layers:

- **Encoder**
  The encoder is composed of a stack of N identical layers and each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a position-wise fully connected feed-forward network. Each of the two sub-layers is followed by layer normalization. All sub-layers in the model, as well as the embedding layer, produce outputs of the same dimension.

- **Decoder**
  The decoder is also composed of a stack of N identical layers. The decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, each of the sub-layers is followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to

prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

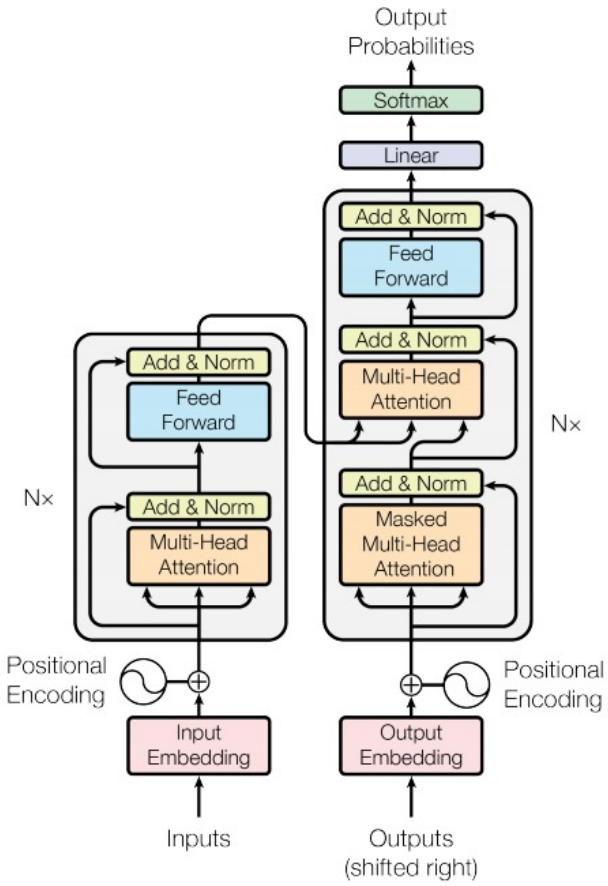


Figure 1.3.1: Transformer model

1.3.1 Positional encoding

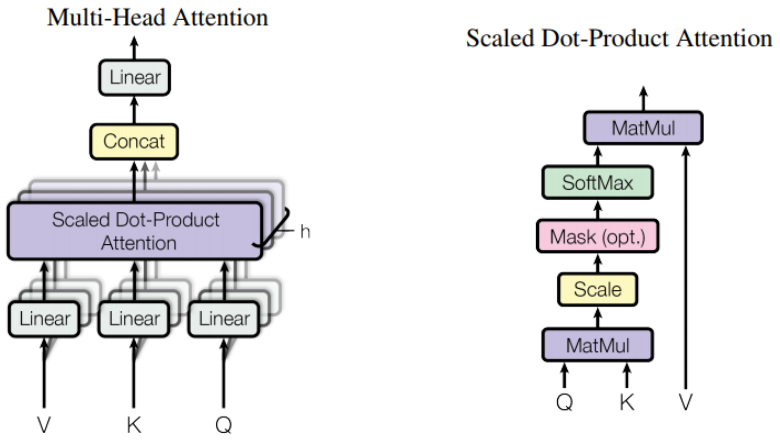
Attention layers see their input as a set of vectors, with no sequential order. This model also doesn't contain any recurrent or convolutional layers. Because of this a "positional encoding" is added to give the model some information about the relative position of the tokens in the sentence.
The positional encoding vector is added to the embedding vector. Embeddings represent a token in a d-dimensional space where tokens with similar meaning will be closer to each

other. But the embeddings do not encode the relative position of tokens in a sentence. So after adding the positional encoding, tokens will be closer to each other based on the similarity of their meaning and their position in the sentence, in the d-dimensional space. The formula for calculating the positional encoding is as follows:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}), \qquad PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

1.3.2 Scaled Dot-Product Attention and multi-head attention

The particular attention used by Vaswani et al. [2017] is called "Scaled Dot-Product Attention" (Figure 1.3.2 (b)). The input consists of queries and keys of dimension dk, and values of dimension dv. We compute the dot products of the query with all keys, divide each by $\sqrt{dk}$, and apply a softmax function to obtain the weights on the values.



(a) Multi-Head Attention (b) Scaled Dot-Product Attention

Figure 1.3.2: On the left the multi-head attention is shown composed by multiple heads, on the right the single Scaled Dot-Product Attention mechanism.

The attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{dk}})V$$

Instead of performing a single attention function with keys, values and queries whose dimension is based on the model dimension, it is beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively. On each of these projected versions of queries, keys and values

we then perform the attention function in parallel, yielding dv-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 1.3.2 (a). Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

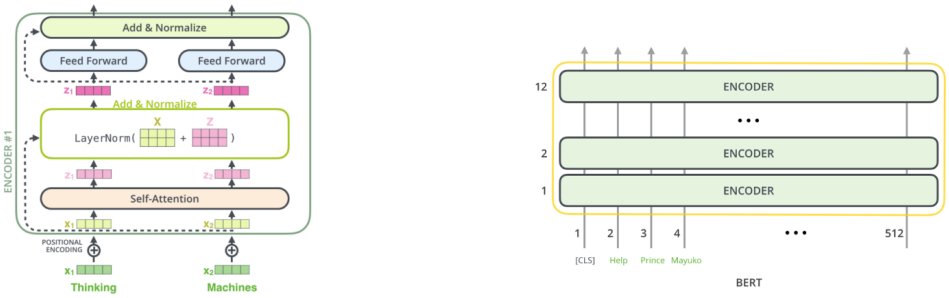
where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

If we call h as the number of parallel attention layers, or heads, for each of these we use $dk = dv = dmodel/h$ as dimension. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

1.4 BERT and DistilBERT

BERT (Bidirectional Encoder Representations from Transformers) is a multi-layer bidirectional Transformer encoder introduced by Devlin et al. [2018] and based on the original transformer implementation described in Vaswani et al. [2017].



(a) BERT single layer

(b) BERT stacked layers

Figure 1.4.1: BERT architecture

BERT is conceptually simple and empirically powerful and that's why we decided to use it in our project, even though it's not aimed at NMT (its purpose are masked language tasks) it still provides great performances for our work, but BERT is extremely big even in his base form (it contains twelve layers, as many as a standar transformer) so it isn't the most cost-efficient model for our task, so for this reason we opted to use its distilled[1] version, called DistilBERT developed by Sanh et al. [2019].

DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT. It has 40% less parameters than BERT, runs 60% faster while preserving over 95%

[1]Knowledge distillation is a compression technique in which a compact model - the student - is trained to reproduce the behaviour of a larger model - the teacher - or an ensemble of models.

of BERT's performances as measured on the GLUE language understanding benchmark from Wang et al. [2018]. For those reasons, DistilBERT was chosen instead of BERT since we didn't have very powerful machines in our hands.

1.5 RoBERTa

Roberta is a transformer encoder with the same architecture as BERT, described in 1.4, developed by Facebook AI. As written by Liu et al. [2019], the team found that BERT was significantly undertrained, and can match or exceed the performance of every model published after it. They have just modified BERT key hyperparameters, removing the next-sentence pre-training objective and training with much larger mini-batches and learning rates. In 2019, when Facebook AI wrote the article, their best model achieved state-of-the-art results on GLUE (Wang et al. [2018]), RACE and SQuAD.

1.6 T5v11

A Google research team published the paper by Raffel et al. [2019], introducing a novel "Text-to-Text Transfer Transformer" (T5) transformer model which can convert any language problem into a text-to-text format.

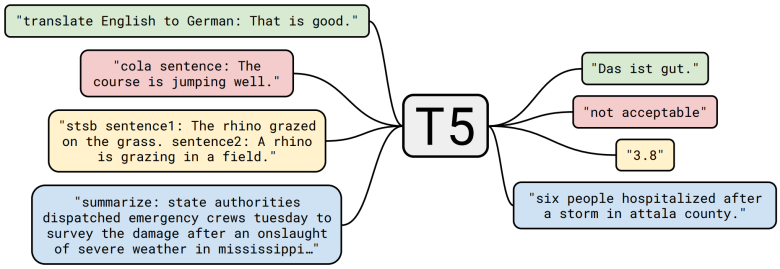


Figure 1.6.1: Google T5 model.

The idea behind T5 was to exploit the potential of transfer learning, where a model is first pre-trained on a data-rich task before being fine-tuned on a downstream task, emerging as a powerful technique in natural language processing (NLP). The proposed model is essentially a Encoder-Decoder Transformer with some architectural changes (like applying Layer Normalization before a sub block and then adding the initial input to the sub-block output; also known as pre-norm).

The model configuration is similar to BERT base and we decided to use the more polished version, T5v11, since it had some improvements:

- GEGLU activation in the feed-forward hidden layer, rather than ReLU.

- Dropout was turned off in pre-training (quality win). Dropout should be re-enabled during fine-tuning (which we did).
- Pre-trained on C4 only without mixing in the downstream tasks.
- No parameter sharing between the embedding and classifier layer.

Between all the transformers models we considered for the project, T5 was the only one aimed at an NMT task, so we expected it to give us better performances than the previous models we've shown.

2 Datasets and benchmark

Before talking about the workflow and the results, we thought it was necessary to explain the datasets we used and how we preprocessed them. At the beginning we first tried to use the europarl en-it corpus, however we've found ourselves overwhelmed by the dataset's huge size, moreover it was filled with extremely long sentences, so the training phase of our model was extremely time-consuming, even on Colab TPUs. As a consequence, we tried using only an half of the original dataset but the model did not translate well. To overcome those difficulties, we decided to adopt a second dataset, the en-it anki corpus which contains way less records than the first one (with more sentences used in a normal conversation), but enough to build a simple and well behaved NMT model.

2.1 ANKI en-it dataset

The anki corpus from manythings.com [2021] to work on was the english-italian one which contained 352040 records at the time of our download and then we preprocessed and divided it in our development (training and validation) and test sets.

2.1.1 Preprocessing the anki corpus

The corpus was, luckily, without any issue, we just had to take away the copyrights from each single sentence pair, which was done by a simple python method. Moreover we had two versions of the corpus, one with no copyright and one without, the method takes care of this case too. The method returns two lists, one for the english sentences and one for the italian ones, we then tokenized both the lists and we proceeded with the creation of our training, validation and test sets with the (0.8, 0.1, 0.1) split. At this point the training set is divided in batches and is ready to be fed to the model.

2.2 EuroParl en-it dataset

Since the anki dataset consists of very simple sentences, we thought that it would be better for our model to include some records from the europarl dataset.

2.2.1 Preprocessing the europarl corpus

As for the anki dataset, luckily for us the europarl corpus was already largely preprocessed, the main issue was the huge size which meant that the dataset couldn't fit inside the allocated memory on colab, that's why we decided to take only a fifth of the initial corpus (381823 records).

2.3 Merging the datasets

We merged the two datasets in hope to have a better performance on our models, so we have 733863 records to be splitted into training, validation and test sets.

```
def merge_datasets(first_dataset, second_dataset) -> (list, list):
    first_src, first_dst = first_dataset
    second_src, second_dst = second_dataset
    src_set = first_src + second_src
    dst_set = first_dst + second_dst
    return src_set, dst_set
```

The split was (0.8, 0.1, 0.1) resulting in the following size for our sets:

- Training set size: 587090.
- Validation set size: 73386.
- Test set size: 73387.

The sets are then splitted into batches to improve the training done by the keras fit method.

2.4 Benchmark

We found a recently developed benchmark based on SacreBLEU for our project which is the flores dataset built by Goyal et al. [2021], it was developed by Facebook and it's useful for those languages with not so many resources (italian doesn't have many evaluation datasets around) and for assessing not so huge models, which is perfect for our case.

The benchmark, as we said before, is based on SacreBLEU(which is an improved version of BLEU that expects detokenized sentence to evaluate a model performances developed by Post [2018]) and comes with a test set composed by 1012 sentences taken from the english wikipedia and translated by humans, so each of our models were evaluated on how they performed on that set.

3 Workflow

3.1 Setup

Hardware Models were trained in Google Colab using Google TPU v2-8 [2], it has 64 GB HBM memory and a compute capability of 180 teraflops. During our work we noticed that TPUs outperform NVidia GPUs available in Colab by running ten time faster and with a better power efficiency.

Software In order to train our transformer models, we used Tensorflow with Keras Chollet [2021] and the huggingface.co [2021] python library to retrieve pre-trained models that acted as encoders.

3.2 Dataset creation

As a starting comparison, we trained our models using the entire anki dataset and 20%of the Europarl bilingual dataset as we said in section 2. In order to train or infere on a machine translation model, the sentences inside the dataset that are written in natural language must be tokenized. Every encoder from Huggingface has its own tokenizer, that is a module that parse words into tokens (those are integers essentially).

Large dataset handling These two datasets don't fit together into the RAM available in Google Colab, so it was necessary to split them into smaller ones in order to apply the tokenization, then after we did this work on each part of the dataset, we merged the results using the Tensorflow Dataset API in order to build the entire tokenized dataset. This is a mandatory step that allows to train any model using very large datasets, in fact, Tensorflow Dataset swaps data into the secondary memory if the main memory is not enough to contain the entire data. At training time, Tensorflow Dataset loads data into the accelerator (GPU or TPU) memory in batches of equal dimension.

Dataset The dataset is made up of the tokenized sentence that is the language to translate from, the tokenized sentence of the target language and the same tokenized sentence but shifted to left (removing the start of sentence token), the latter one was inserted in the dataset only to speed up the training phase even if the dataset is larger. Anyway, our models were too large for the resources we had available, and the training phase was very time consuming.

[2] `https://cloud.google.com/tpu`

3.3 Tokenization

As announced earlier, any dataset made up of sentences written in natural language has to be tokenized before training or inference. The Huggingface library provides a Tokenizer class that can retrieve a pre-trained tokenizer from a wide database and tokenize any provided sentence. As shown in the example below in code 1, we created a dataset with a fixed length of tokens, padded with zeros whether the sentence was shorter than the maximum length or truncated in case it exceed the pre-defined size.

```
tok_trg = "dbmdz/bert-base-italian-cased"
tokenizer_target = BertTokenizerFast.from_pretrained(tok_trg)
tokens_source = tokenizer_source(source_set, truncation=True,
↪  padding="max_length", return_tensors="tf",
↪  max_length=sequence_length).data["input_ids"]
```

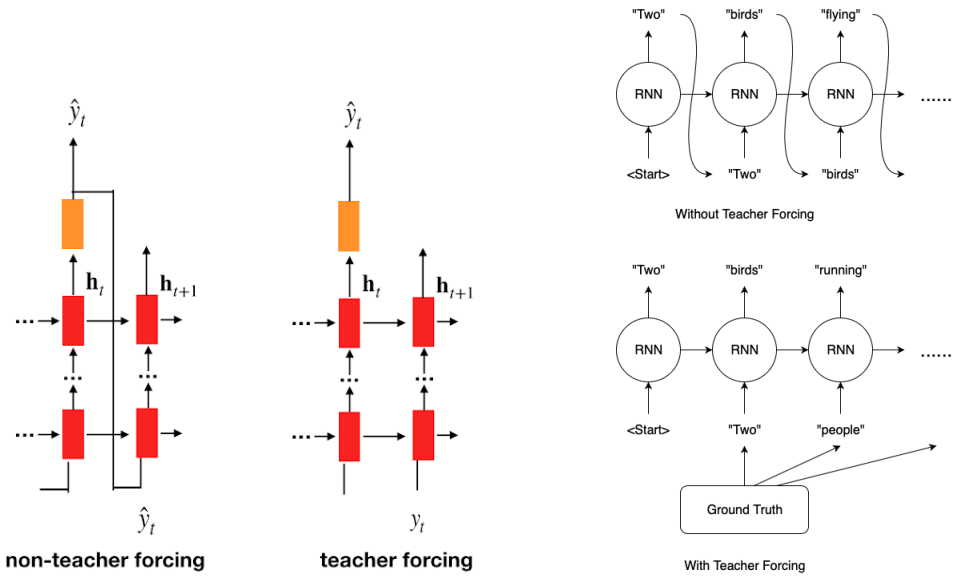
Listing 1: Example of code to tokenize a piece of dataset using Huggingface Bert Tokenizer.

Example 3.1. Bert tokenizer parse the following sentence: "that is progress of some sort, small though it may be." to the following vector of integers: [101 1337 1110 5070 1104 1199 3271 117 1353 1463 1122 1336 1129 119 102 0 0 0 0 0]. The '101' and '102' ids correspond to the start and the end of the sentence ("[CLS]" and "[SEP]"), respectively. Zeros at the ends of the array are the padding tokens added in order to reach the maximum length that, in this case, is equal to 20.

3.4 Model creation and training

Model creation During the model creation phase it's possible to create a vanilla transformer (Vaswani et al. [2017]), meaning that both the encoder and the decoder should be trained from scratch. In this case it's possible to set the number of the layers, the size of the feed forward network and the size of the latent space. It's also possible to build a transformer using a pre-trained encoder from a checkpoint by retrieving them from the huggingface.co [2021] library. In this case we can setup only those parameters that refer to the decoder part due to the fact that the encoder is pre-trained and, as a consequence, has the parameters already set.

Training In both cases, the transformer is trained using teacher forcing, in particular the decoder part. By exploiting this technique the decoder always receives the ground truth to predict the next token whereas, without teacher forcing, the decoder will predict the next token using the one that it predicted at the previous time step, the latter (called free running) is used during inference where no ground truth is known beforehand.



(a) Teacher forcing technique.

(b) Decoding comparison without and with teacher forcing, an example.

Figure 3.4.1: Teacher forcing technique, used only during the training phase.

We did fune-tuning on the encoders from Huggingface by training them together with the decoder, another solution that was tried by Imamura and Sumita [2019] suggested to freeze the encoder's weights and to train the decoder before a final training step where both layers would be trained at the same time, even though the paper stated that this would imply better performances from our models, we didn't find any major improvement on the SacreBLEU score.

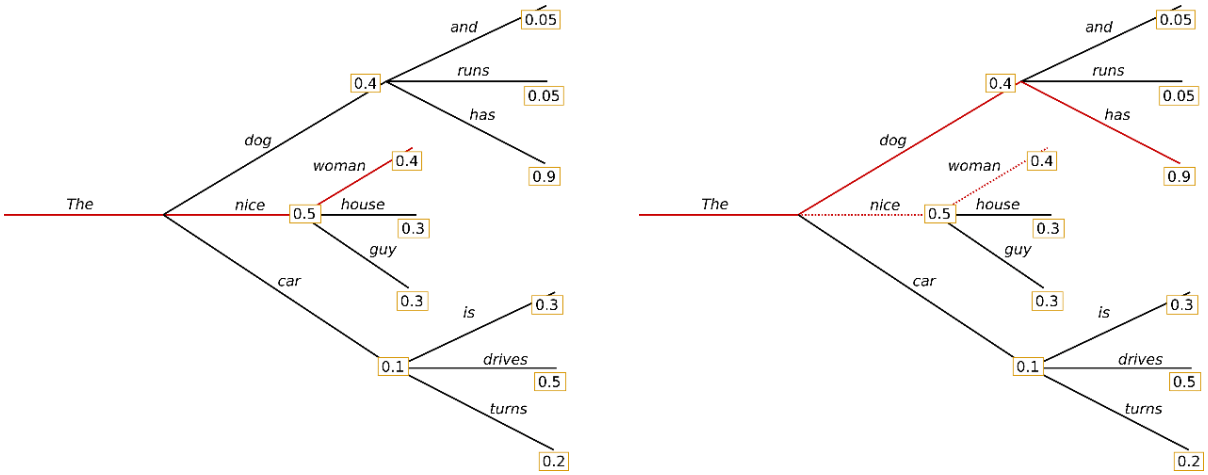
3.5 Translator

The translator class implements the method to translate a sentence, it takes as input a sentence to translate and returns it translated. The method works in this way: it tokenizes the sentence using the encoder tokenizer and gives it as input to the encoder. The decoder receives as input the start of sequence token ("[CLS]") then, at each time step t, it predicts the next token taking as input the one generated at the previous time step $(t-1)$. This procedure is the same as the one shown in figure 3.4.1, without taking into account the teacher forcing. We implemented several different ways to translate a sentence and the results are shown in section 4.

Greedy search Greedy search simply selects the word with the highest probability as its next word: $w_t = argmax_w P(w|w_{1:t-1})$ at each time step t. As sketched in figure

3.5.1 (a), starting from the word "The", the algorithm greedily chooses the next word of highest probability "nice" and so on, so that the final generated word sequence is ("The","nice","woman") having an overall probability of $0.5 \times 0.4 = 0.2$. This process can be visualized in figure 3.5.1 (a).

Beam search Beam search reduces the risk of missing hidden high probability word sequences by keeping the most likely number of beams of hypotheses at each time step and eventually choosing the hypothesis that has the overall highest probability. Let's illustrate with num_beams=2 in figure 3.5.1 (b).



(a) Greedy search

(b) Beam search

Figure 3.5.1: A comparison between greedy and beam search.

Top-K sampling Beam search can work very well in tasks where the length of the desired generation is more or less predictable as in machine translation or summarization (Yang et al. [2018]). As argued in Holtzman et al. [2019], high quality human language does not follow a distribution of high probability next words. In other words, as humans, we want generated text to surprise us and not to be boring/predictable. The authors show this nicely by plotting the probability that a model would give to human text vs. what beam search does.

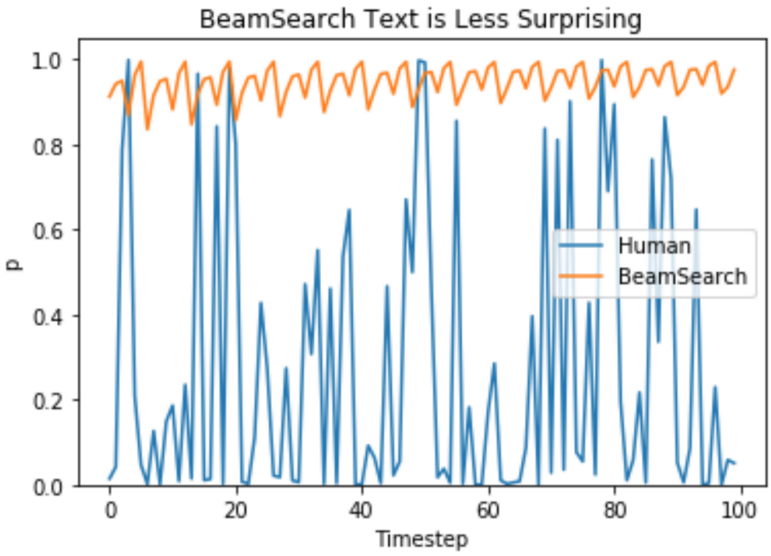


Figure 3.5.2: Beam search versus sampling approach.

Fan et al. [2018] introduced a simple, but very powerful sampling scheme, called Top-K sampling. In Top-K sampling, the K most likely next words are filtered and the probability mass is redistributed among only those K next words, this approach worked well in model for text generation like GPT2, and we know that is the field for which it was designed, anyway we wanted to test how this technique performs in machine translation. Whether a model is trained using a large dataset, we expect to have translations that differ each of other by some words, which are actually synonymous.

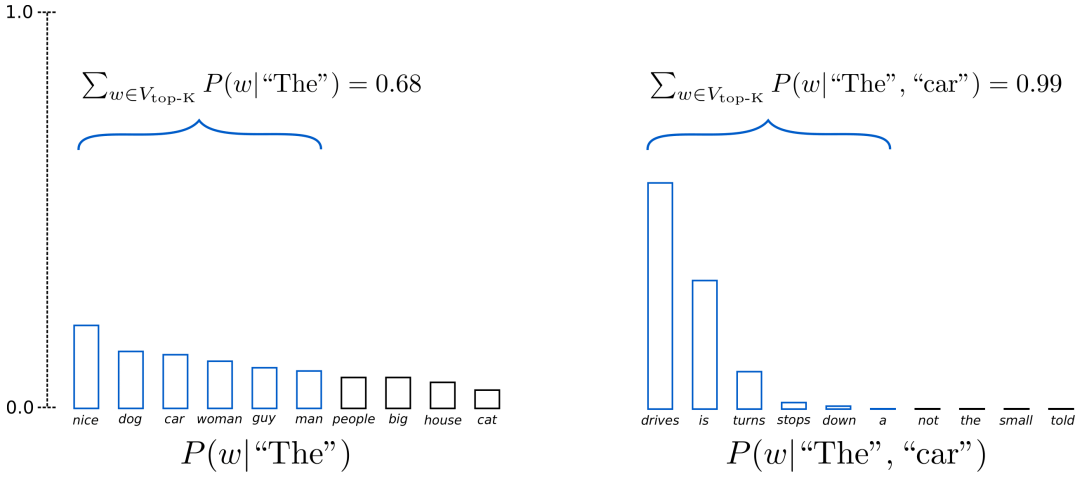


Figure 3.5.3: Top-k sampling approach.

3.6 Evaluation

As we said at the end of section 2, we used the Flores dataset by Goyal et al. [2021] provided by Facebook as benchmark for our models, to evaluate the performances we need to translate the 1012 sentences in the set with each one of our trained models. We saved the translations on files which are then fed to a command that computes the sacreBLEU score as the guide from the Flores github suggests.

```
# Setup the benchmark by cloning the repository
!git clone --single-branch --branch adding_spm_tokenized_bleu
↪  https://github.com/ngoyal2707/sacrebleu.git
!python setup.py install

# Evaluate the model's translations by computing the BLEU score
!cat translation.txt | sacrebleu -tok spm
↪  flores101_dataset/devtest/ita.devtest
```

We decided to evaluate our models on their translation with both top-k-sampling with k=1 (greedy search) and with k=5, even though the beam search works pretty well (there some examples in section 5) it's too time consuming as it depends heavily on a sentence length and since the Flores dataset contains sentences that are long (we're talking about an average of 40 words per sentence) the beam search can take more then twenty times with respect to the other two approaches, so we decided to use it only for single cases.

4 Results

4.1 Models architecture

The following table shows the different architectures we used in order to compare the performance of our models for the neural machine translation task. In most of the cases we had to reduce the number of parameters on the decoder side of the transformer due to memory and time limitation on the TPUs available on Google Colab and because the encoders (the ones from huggingface.co [2021]) weren't customizable. While the number of encoder and decoder layers could differ as shown by Ma et al. [2021] for the DeltaLM model in which they suggested a number of decoder layers that was $\frac{3}{4}$ of the number of encoder layers, moreover we noticed that a similar amount of attention heads in the decoder and in the encoder produced better results.

Architectures					
Model	**Encoder**	**Decoder**	**Heads (En/De)**	**Latent dim**	**FFNN**
Base	6 layers	6 layers	8 / 8	512	2048
DistilBERT	6 layers	4 layers	12 / 8	768	3072
RoBERTa	12 layers	3 layers	12 / 8	768	3072
T5 v1.1 small	8 layers	6 layers	6 / 8	512	1024

Table 4.1.1: An architectural comparison between the models we used for NMT.

4.2 Comparisons between models

We started comparing our models, which were trained on 20% of the dataset, between themselves using the Flores dataset as benchmark. Keep in mind that the training time is based on the TPUs performances.

Our Models						
Encoder	**Task**	**Param.**	**Tr. time**	**Acc. (tr, val)**	**k=1**	**k=5**
Base	NMT	223M	372 m	0.781, 0.796	11.2	8.6
DistilBERT	Masked LM	276M	396 m	0.809, 0.826	12.8	10.1
RoBERTa	Masked LM	276M	400 m	0.805, 0.823	12.8	10.0
T5 v1.1 small	NMT	180M	380 m	0.769, 0.787	15.2	12.4

Table 4.2.1: A quantitative comparison of the results obtained by the models we used for NMT.

As we expected, models aimed at masked language tasks performed worse than those aimed at NMT ones (moreover we can see that DistilBERT and RoBERTa gave more or less the same results). We can notice that an increase in the number of parameters

corresponds to an increase in accuracy and SacreBLEU score, but that's not the case for the model using T5 as an encoder which, even though was the smallest one, performed way better than the others.

Even though we expected similar results with the translations done with top-k sampling with k=5, that wasn't the case and the BLEU score wasn't at the same level with those translations done with k=1.

We were saddened by the fact that our models didn't have nice SacreBLEU scores, so we decided to push further our best model (the one with T5 v1.1 as an encoder) by adding more layers on the decoder side (8 instead of 6) and by using the 70% of the entire dataset and then the full one. As shown in Table 4.2.3 the size of the dataset really makes the difference. In fact, by comparing two identical models, one trained using 70% of the dataset and one trained on the full dataset, the second one outperforms the first one by 0.8 SacreBLEU score.

The training set, from the full dataset, now contains nearly two million records which is three times the amount on which we did the previous analysis, so we expected it to give us better results.

Our Final Models					
Encoder	**Decoder**	**Param.**	**Epochs**	**Acc. (tr, val)**	**SacreBLEU**
T5 v1.1 small	8 layers	218M	20	0.769, 0.787	16.9
T5 v1.1 small	8 layers	218M	18	0.735, 0.751	17.7

Table 4.2.2: Our final models built from the best model from the previous comparison.

We then compared our best models against some state-of-art models, that we've taken from the web, by using the Flores dataset as benchmark.

Models Comparison		
Model	**Corpus size**	**SacreBLEU**
DeltaLM	6.088 TB (multilang)	31.7
MarianMT	45 M (of sentence pair)	33.2
Google translate	It's Google!	38.9
Our T5 (70%)	460MB (70% of 658MB)	16.9
Our T5 (full)	658MB	17.7

Table 4.2.3: Comparison between our models and the most popular in the NMT field.

4.3 Final considerations on our work

The task was a battle against titans, we knew we couldn't stand a chance from the start since we don't have the same amount of parallel data on which the state-of-art models

were trained and we don't have powerful machines that can run the training flawlessly for many hours and days (the TPUs on Colab are way better than the GPUs on our machines). Moreover the benchmark dataset is based on sentences taken from Wikipedia so they aren't really that common and our models had some issues with entities' names (which led the translation process astray), but we've seen that our model performs pretty well on day-to-day sentences, so we're somewhat satisfied that it's doing its task somewhat correctly. We've tried to push the best model for more epochs but the improvements were not so great since the available corpus was still the same and, as we know from the course, we need more records to improve the performances of our model.

5 Appendix: Beam search translations

In this section we will show how our best model translates some sentences using the beam search technique. The beam size is set equal to 2 and the branches of the tree are pruned if their conditional cumulative probability is lower than an half of to the best one. The choice of the beam size is motivated by the fact that our dataset is limited and does not contain too much words having the same meaning. As a confirmation, we noticed that in almost all the cases the third word extracted had a probability near to zero.

In the following list, we put a tag "EN" to indicate the sentence to translate and the tag "IT" to indicate the best translation, according to the beam search. Moreover, inside "Best branches" we listed the top 3 most probable sentences and their final cumulative probability. In some cases, there could be less than 3 candidate sentences, due to the pruning strategy which cuts the branches having too low probability. We won't show the translations with the other methods we've talked about in subsection 3.5 since the results are pretty similar between them.

- **EN:** In many other cities of Italy and in the rest of the world, particularly in Spain, similar setups were made, which were viewed by a great number of people.

  Best branches:

  - In molte altre città italiane e nel resto del mondo , in particolare in Spagna , sono state fatte degli insediamenti simili , visti da molti cittadini ., **5.3550e-06**
  - In molte altre città italiane e nel resto del mondo , in particolare in Spagna , sono state fatte delle formazioni simili , viste da moltissime ., **2.8514e-07**
  - In molte altre città italiane e nel resto del mondo , in particolare in Spagna , sono state fatte delle strutture simili , viste da moltissime ., **1.2335e-07**

  IT: In molte altre città italiane e nel resto del mondo , in particolare in Spagna , sono state fatte degli insediamenti simili , visti da molti cittadini .

- **EN:** I don't think there is anyone in the world who has so near his heart the ideal of peace, as I have.

  Best branches:

  - Non penso che nessuno nel mondo abbia così vicino al suo cuore l ' ideale della pace come me ., **0.0171**
  - Io non penso che nessuno nel mondo abbia così vicino al suo cuore l ' ideale della pace come me ., **0.0131**
  - Io non penso che nessuno nel mondo abbia così vicino al suo cuore l ' ideale della pace , come ho ., **0.0027**

IT: Non penso che nessuno nel mondo abbia così vicino al suo cuore l ' ideale della pace come me .

- **EN:** The girl stepped to the mirror and stared, she was fascinated by her elegance.

  Best branches:

  - La ragazza si è rivolta allo specchio e ha guardato , era affascinata dalla sua eleganza .', **0.001102**
  - La ragazza si è rivolta allo specchio e ha guardato , è affascinata dalla sua eleganza .', **0.000262**
  - La ragazza si è mossa allo specchio e ha guardato , era affascinata dalla sua eleganza .', **0.000822**

  IT: La ragazza si è rivolta allo specchio e ha guardato , era affascinata dalla sua eleganza .

- **EN:** He wanted what he wanted and he wanted it beyond thought and beyond rationality.

  Best branches:

  - Voleva quello che voleva e che era oltre la logica ., **0.00402**
  - Voleva quello che voleva e che voleva oltre la logica ., **0.00350**
  - Voleva quello che voleva e che era al di là della logica ., **0.00448**

  IT: Voleva quello che voleva e che era al di là della logica .

- **EN:** Your muscles are amazingly strong and your reaction time is amazingly fast, you're one of the best players i've ever seen.

  Best branches:

  - Le sue muscoli sono incredibilmente forti e il suo tempo di reazione è incredibilmente veloce , sei uno dei giocatori migliori che abbia mai visto ., **0.002565**
  - Le sue muscoli sono incredibilmente forti e il suo tempo di reazione è incredibilmente veloce , siete uno dei giocatori migliori che abbia mai visto ., **0.001649**
  - Le vostre muscoli sono incredibilmente forti e il suo tempo di reazione è incredibilmente veloce , sei uno dei giocatori migliori che abbia mai visto ., **0.001826**

IT: Le sue muscoli sono incredibilmente forti e il suo tempo di reazione è incredibilmente veloce , sei uno dei giocatori migliori che abbia mai visto .

- **EN:** You will have the necessary computers and reference material.

  Best branches:

  - Avrai i computer necessari e il materiale di riferimento .,**0.1522**
  - Avrete i computer necessari e il materiale di riferimento ., **0.1659**

  IT: Avrete i computer necessari e il materiale di riferimento .

- **EN:** We've been doing this task for about three months and i'm very tired now, i think we both need some rest before starting another one.

  Best branches:

  - Noi facciamo questo compito da circa tre mesi e sono molto stanco ora , penso che entrambi abbiamo bisogno di riposarmi prima di iniziare un altro ., **0.0001522**
  - Noi facciamo questo compito da circa tre mesi e sono molto stanco ora , penso che entrambi abbiamo bisogno di riposarci prima di iniziare un altro ., **0.0001506**
  - Noi facciamo questo compito da circa tre mesi e sono molto stanco adesso , penso che entrambi abbiamo bisogno di riposarmi prima di iniziare un altro ., **0.0001502**

  IT: Noi facciamo questo compito da circa tre mesi e sono molto stanco ora , penso che entrambi abbiamo bisogno di riposarmi prima di iniziare un altro .

- **EN:** What else do we have to do for this project now? We need to think about what to do for the next one.

  Best branches:

  - Che altro dobbiamo fare per questo progetto ? Dobbiamo pensare a cosa fare per il prossimo ., **0.0178**
  - Che altro dobbiamo fare per questo progetto adesso ? Dobbiamo pensare a cosa fare per il prossimo ., **0.0447**
  - Che altro dobbiamo fare per questo progetto ? Ora dobbiamo pensare a cosa fare per il prossimo ., **0.0226**

IT: Che altro dobbiamo fare per questo progetto adesso ? Dobbiamo pensare a cosa fare per il prossimo .

- **EN:** We weren't sure about what to do at the start, but now we're sure we made the right choices, hopefully.

  Best branches:

  - Non eravamo sicuri di cosa fare all ' inizio , però ora siamo sicuri di aver fatto le scelte giuste , speriamo ., **0.0200**
  - Non eravamo sicuri di cosa fare all ' inizio , ma ora siamo sicuri di aver fatto le scelte giuste , speriamo ., **0.0196**

  IT: Non eravamo sicuri di cosa fare all ' inizio , però ora siamo sicuri di aver fatto le scelte giuste , speriamo .

References

Francois Chollet. Keras: Simple. flexible. powerful. `https://keras.io/`, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzman, and Angela Fan. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. *arXiv preprint arXiv:2106.03193*, 2021.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

huggingface.co. Huggingface, the ai community building the future. `https://huggingface.co/`, 2021.

Kenji Imamura and Eiichiro Sumita. Recycling a pre-trained BERT encoder for neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 23–31, Hong Kong, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5603. URL `https://aclanthology.org/D19-5603`.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P18-4020`.

Philipp Koehn et al. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86. Citeseer, 2005.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, Alexandre Muzio, Saksham Singhal, Hany Hassan Awadalla, Xia Song, and Furu Wei. Deltalm: Encoder-decoder

pre-training for language generation and translation by augmenting pretrained multilingual encoders. *arXiv preprint arXiv:2106.13736*, 2021.

manythings.com. Anki bilingual dataset, eng-ita. `http://www.manythings.org/bilingual/`, 2021.

Matt Post. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the beam search curse: A study of (re-) scoring methods and stopping criteria for neural machine translation. *arXiv preprint arXiv:1808.09582*, 2018.