Exploring the Impact of Tokenization Strategies on Machine Translation using Transformer Seq2Seq Architecture - Research Design

Filippo Merlo[1]
[1] CIMEC, University of Trento
filippo.merlo@studenti.unitn.it

April 30, 2025

1 Intoduction

How does the choice of tokenization strategy affect the performance of a Transformer-based Seq2Seq architecture in machine translation tasks?

In the last years, the field of natural language processing (NLP) has undergone a big transformation, largely driven by the advent of Transformer architectures. Introduced in the seminal paper "Attention is All You Need," [8] Transformers have revolutionized NLP tasks by leveraging self-attention mechanisms that effectively capture long-range dependencies in text sequences. Unlike their predecessors, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), Transformers process sequences in parallel, resulting in significant improvements in both efficiency and performance.

One of the most critical applications of NLP is machine translation, where models convert text from one language to another. The introduction of sequence-to-sequence (Seq2Seq) models with encoder-decoder architectures, first popularized in the paper "Sequence to Sequence Learning with Neural Networks," [7] laid the groundwork for modern neural machine translation (NMT). While these early models used RNNs or LSTMs, their performance was limited by the sequential processing of tokens and the difficulty in handling long input sequences. The subsequent introduction of the attention mechanism, as described in "Neural Machine Translation by Joint Learning to Align and Translate," [2] marked a significant breakthrough, allowing models to dynamically focus on different parts of the input sentence and thereby improving translation accuracy.

Transformers, by eliminating the sequential processing constraints, have further advanced NMT, enabling more efficient training and the ability to scale to larger datasets. However, the performance of these models is highly sensitive to the choice of tokenization strategy, which breaks down text into manageable subunits (tokens). Tokenization directly affects the input representation that the model learns from, influencing the model's ability to capture linguistic nuances and, consequently, its translation quality.

This project hypothesizes that different tokenization strategies, such as word-level, subword-level (e.g., WordPiece Tokenization [9], Byte-Pair Encoding [6]), and character-level tokenization, will significantly impact the translation quality and overall performance of Transformer-based Seq2Seq models, measured with the Bilingual evaluation understudy (BLEU) score [4].

Specifically, subword-level tokenization is expected to offer a balanced approach, optimizing the trade-off between vocabulary size and model performance, and thereby yielding superior translation results.

2 Data

To test our hypothesis I chose to use the CCMatrix dataset [5]. This dataset has been obtained with a method for large-scale mining of parallel sentences from monolingual corpora using margin-based bitext mining in a multilingual sentence space. The authors apply this technique to a vast dataset of 32.7 billion unique sentences across 38 languages, using ten snapshots of a curated Common Crawl corpus. By embedding sentences in a shared multilingual space with the LASER toolkit (Language-Agnostic SEntence Representations) [1], the method identifies semantically similar sentences across languages. The margin-based criterion, which compares the similarity between sentence pairs against their nearest neighbours, allows for the effective mining of parallel sentences. The margin between two candidate sentences x and y is defined as the ratio between the cosine distance between the two sentence embeddings, and the average cosine similarity of its nearest neighbours in both directions.

I used 203,000 English-Italian sentence pairs from the CCMatrix dataset, splitted in 200000 for training and 3000 for testing the translation quality, to investigate three different tokenization methods: character-level, subword-level (using WordPiece tokenization [9]), and word-level.

Character-level tokenization consists of treating each character in the text as a separate token. So, for example, the word "hello" will be broken down into its individual letters: ['h', 'e', 'l', 'l', 'o']. This approach can capture detailed information and embed any words not in the vocabulary. The backlash is that it often leads to longer sequences, posing a challenge in training the model.

Word-level tokenization treats each word in a sentence as an individual token. For example, the sentence "I am programming" would be tokenized into ['I', 'am', 'programming']. While this method is straightforward to implement, it has some limitations. It struggles with out-of-vocabulary (OOV) words, leading to potential inaccuracies, and can create an excessively large vocabulary if the dataset used to build the tokenizer is extensive. This can make the model less efficient and harder to manage.

WordPiece is a commonly used subword-level tokenization method in models such as BERT [3]. The algorithm is designed to handle OOV words by breaking them down into smaller, known subwords or characters. This approach allows the model to understand and process words that were not seen during training, which is crucial for handling a wide range of vocabulary in real-world text data. Decomposing words into more fundamental and shared pieces allows for a decrease the vocabulary size. WordPiece begins with a small vocabulary that includes special tokens and an initial set of characters. Each word is initially split into subwords using a prefix (e.g., "##" in BERT), so "word" becomes "w ##o ##r ##d". The model then learns to merge subwords based on a calculated score, which prioritizes less frequent combinations over more frequent ones 1.

The score for a subword pair is calculated as follows:

$$\text{score} = \frac{\text{freq\_of\_pair}}{\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element}} \tag{1}$$

By dividing the frequency of the pair by the product of the frequencies of each of its parts, the algorithm prioritizes the merging of pairs where the individual parts are less frequent in the vocabulary. This process continues until the vocabulary reaches the desired size.

3 Model

The model I used is a standard seq2seq transformer architecture shown in 1. This setup comprises two neural networks: an encoder transformer for processing the tokens of the initial English sentence, and a decoder transformer for generating tokens of the Italian translated sentence. Both the encoder and decoder have 1 layer each, 8 attention heads per layer, an input size of $s = 200$ (the size of the embedding vector for each token in the sequence), and a hidden size of $d = 512$. ReLU activation functions are used in the linear layers.

The encoder (left of Fig.1) processes the input sequence tokens. Initially, each token is converted into one-hot encoding vectors of dimension $s \times$ vocab_size. Each of these vectors is then individually processed and compressed to embeddings of the size $s \times d$.

Once the words are embedded into vectors, positional encoding is introduced to specify their absolute and relative positions in the sequence. This step is crucial because the position in natural language can carry significant information. For instance, in the sentence "The cat watches the dog run fast," swapping "cat" and "dog" changes the subject and object, thereby altering the sentence's meaning. To encode this positional information, a method called Sinusoidal positional encoding is used. This method uses sine and cosine functions based on the token positions and adds the result to the input vectors.

Following positional encoding is the self-attention mechanism. This process involves comparing each token in the input sequence to every other token to compute attention weights. High attention weights between tokens suggest a syntactic or semantic relationship, indicating which tokens the model should focus on.

The attention mechanism in transformers operates by distinguishing between three types of input matrices: queries (Q), keys (K), and values (V). Each of these three matrices is $s \times d$ dimension matrix representing the processed tokens of the input. The mechanism works by comparing the queries (Q) with the keys (K) to evaluate their relevance or compatibility. This comparison generates attention weights, which are then used to select corresponding values (V) that have higher weights. The embedded Q and K tokens are multiplied using the dot product. The result of the multiplication is scaled by dividing by $\sqrt{d}$. In this case, scaling helps reduce the magnitude and ensure suitable inputs to the softmax function. Next, the scaled product is fed into a softmax function, which outputs the attention weights. The returned attention weights of all pairs of tokens are between 0 and 1 and sum up to 1. The computed attention weights are then multiplied with the values V. Token pairs where Q and K are less compatible have lower attention weights, closer to zero. Consequently, their values V are also reduced to vectors close to zero. The model attends less to these values than to others with high attention scores.

This model uses a multi-head attention mechanism. In this case, the model repeats the attention mechanism explained above multiple times (h times) in parallel. Before passing the input tokens of dimension d into these h attention blocks, they projected into smaller embeddings of size d/h by using small linear neural networks. Each of the h linear networks has different weights and leads to different projections. Consequently, each attention "head" can learn to focus on different aspects. The outputs of the h heads are then concatenated and passed through another linear neural network.

Finally, a standard feed-forward neural network processes the outputs of the multi-head attention block by applying a series of linear transformations followed by non-linear activation functions, such as the rectified linear unit (ReLU). This enhances the model's capacity to model non-linear relationships between words within the sequence. The encoded tokens are then passed to the decoder.

It's important to note that the outputs from both the attention block and the feed-forward

network are added to their original inputs and normalized through residual connections, essential for preserving gradient flow during backpropagation, preventing the loss of gradients due to their diminishing nature.

The decoder (right of Fig.1) in the architecture is similar to the encoder but operates on the output sequence. It takes both the encoded input tokens and the output tokens produced so far as input to generate new ones. Like the encoder, it embeds output tokens into vectors and uses positional encoding and self-attention to process them.

The big difference is that the decoder includes a cross-attention block that compares the output sequence tokens with the encoded input tokens. The decoder's cross-attention module uses the encoded input tokens as K and V and the produced output tokens as Q. This cross-attention is crucial for generating the next output token by considering both the input and the output sequence so far.

Finally, a linear layer and softmax activation produce probabilities for all possible tokens, selecting the one with the highest probability as the next token. The process continues until an end-of-sequence token is generated.

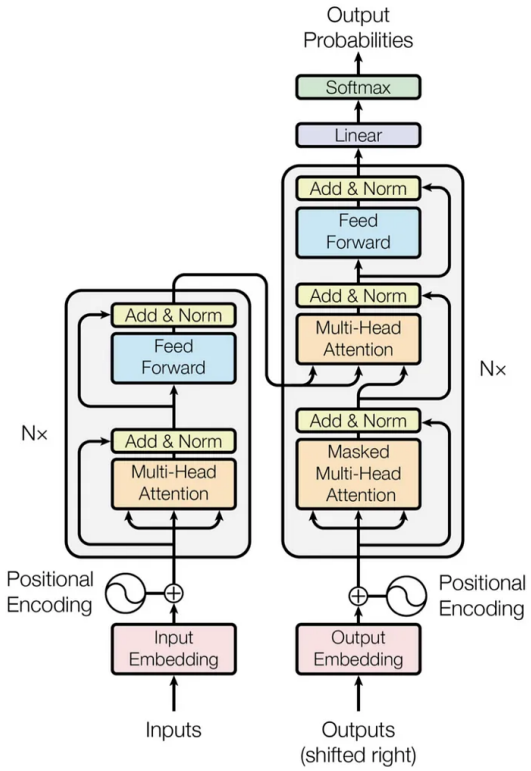

Figure 1: Transformer Architecture.

4 Training regime

The transformer is trained using the Adam optimizer to minimize the cross-entropy loss averaged over tokens, with a batch size of 30 data point (english-italian sentence pairs). The training last for 10 epochs, starting with a learning rate of 0.0001. A dropout rate of 0.1 is applied to the input embeddings and transformers. No hyperparameter tuning is applied since the aim of the study is to compare the change in performance among different tokenization strategies. All the parameters are kept constant in the three different trainings to allow a more fair comparison.

References

[1] Mikel Artetxe and Holger Schwenk. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. 7:597–610.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding.

[4] 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 on Association for Computational Linguistics - ACL '02, page 311. Association for Computational Linguistics.

[5] Holger Schwenk, Guillaume Wenzek, Sergey Edunov, Edouard Grave, and Armand Joulin. CCMatrix: Mining billions of high-quality parallel sentences on the WEB.

[6] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units.

[7] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need.

[9] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation.