" Natural Language Processing with Attentions Models"

NLP 5th Lab

MEKKI Meriem

ESI SBA - May 2024

Summary

1. Introduction

The advent of attention mechanisms and transformer architectures has revolutionized the field of Natural Language Processing. These innovations have significantly improved the performance of various NLP tasks, including automatic translation, by enabling models to capture complex dependencies and contextual information more effectively.

Automatic translation, or machine translation, is the process of converting text from one language to another. This task is inherently challenging due to the complexities and nuances of human languages. Traditional translation models often struggled with accuracy and fluency. However, transformer models, introduced by Vaswani et al. in "Attention is All You Need," have set new standards in translation quality.

In this Lab, we focus on building and training a transformer-based model for translating sentences from French to English. The objective is to understand how transformers can be effectively applied to automatic translation and to evaluate their performance in this specific NLP task. This investigation aims to deepen our understanding of modern translation techniques and their practical applications.

2. Dataset

For this lab, we utilized the wmt14_translate dataset, a widely recognized benchmark for machine translation tasks. The WMT14 dataset is part of the TensorFlow Datasets (TFDS) API, which provides a convenient and standardized way to access and preprocess this data.

This dataset consists of parallel sentences in English and French, making it suitable for training and evaluating translation models. Specifically, the dataset includes the following:

- Training Samples: 40,836,876
- Validation Samples: 3,000
- Test samples: 3,003

For training the model in this lab we utilized a subset of the training set that consists of 60,000 samples, as training on the entire original training set is computationaly expensive due to its size

3. Preprocessing

3.1. Tokenization

For Tokenization we utilized a subword tokenizer. We have first generated a subword vocabulary from our chosen dataset, and then we used it to build a text.BertTokenizer from the vocabulary.

The main advantage of a subword tokenizer is that it interpolates between word-based and character-based tokenization. Common words get a slot in the vocabulary, but the tokenizer can fall back to word pieces and individual characters for unknown words. Below are the steps for building this Tokenizer:

- Loading the dataset.
- Generating a wordpiece vocabulary from the dataset.
- Saving the generated vocabulary for both english and french (each in a file).
- Building the Tokenizer using the previously generated vocabulary.
- Save and Export the Tokenizer for later usage.

4. Setting up the data pipeline

In this section, we establish a robust data pipeline using TensorFlow's tf.data API to preprocess text data for training our model. We begin by implementing a function that efficiently tokenizes input text into ragged batches, ensuring they adhere to a maximum token length. This function further divides target tokens (English) into inputs and labels, pivotal for aligning inputs with their subsequent tokens. The subsequent conversion of RaggedTensors to padded dense Tensors ensures compatibility with training. Following this, we detail a process to convert the dataset of text examples into batches suitable for training. This involves tokenization, filtering of excessively long sequences, shuffling, and batch assembly. Leveraging caching optimizes efficiency by executing operations just once, while prefetching enables parallel data processing with model training. Consequently, the resulting tf.data.Dataset objects, structured as (inputs, labels) pairs, seamlessly integrate with Keras Model.fit training, facilitating the efficient and effective training of machine translation models.

5. Defining the components of the architecture

The transformer architecture in NLP is a groundbreaking model that relies on self-attention mechanisms to process input sequences. Unlike traditional models, it operates in parallel and efficiently captures long-range dependencies and contextual information. The architecture consists of an encoder-decoder framework, where the encoder transforms input sequences into contextualized representations, and the decoder generates output sequences based on these representations. This parallelizable and scalable nature has made transformers the cornerstone of modern NLP applications, excelling in tasks such as machine translation, text generation, and sentiment analysis.

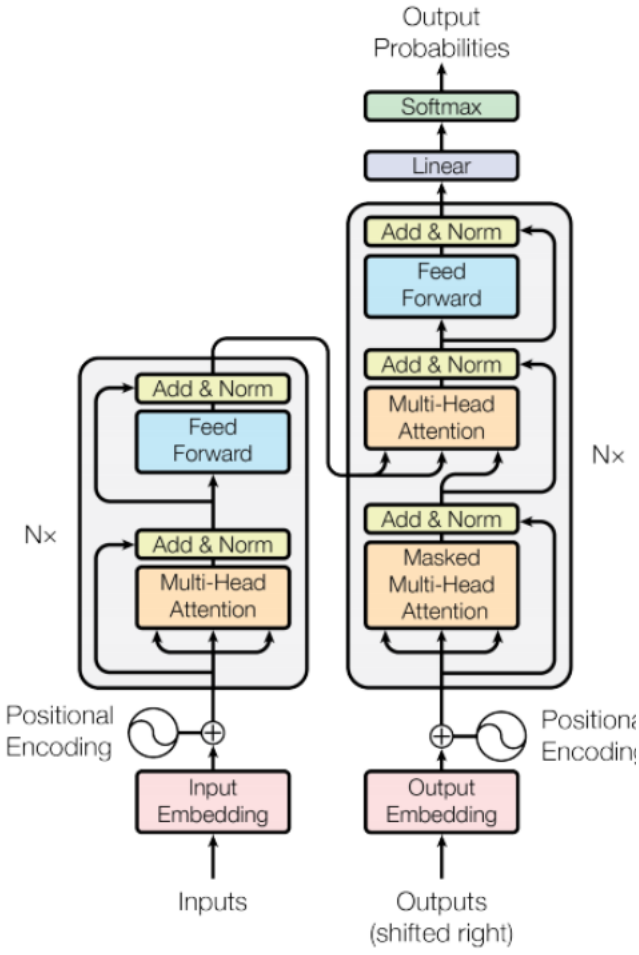

Figure 1: Transformer architecture

5.1. The embedding and positional encoding layer

Given a sequence of tokens, both the input tokens (French) and target tokens (English) have to be converted to vectors using a tf.keras.layers.Embedding layer. A Transformer adds a

"Positional Encoding" to the embedding vectors. It uses a set of sines and cosines at different frequencies (across the sequence). By definition nearby elements will have similar position encodings.

5.2. Add and normalize

The residual "Add & Norm" blocks are included so that training is efficient. The residual connection provides a direct path for the gradient (and ensures that vectors are updated by the attention layers instead of replaced), while the normalization maintains a reasonable scale for the outputs.

5.3. The base attention layer

Attention layers are used throughout the model. These are all identical except for how the attention is configured. Each one contains a layers.MultiHeadAttention, a layers.LayerNormalization and a layers.Add. To implement these attention layers, start with a simple base class that just contains the component layers. Each use-case will be implemented as a subclass.

5.4. The cross attention layer

At the literal center of the Transformer is the cross-attention layer. This layer connects the encoder and decoder. This layer is the most straight-forward use of attention in the model, it performs the same task as the attention block in the NMT with attention tutorial.

5.5. The global self attention layer

This layer is responsible for processing the context sequence, and propagating information along its length.

5.6. Global self attention layer

The global self attention layer on the other hand lets every sequence element directly access every other sequence element, with only a few operations, and all the outputs can be computed in parallel.

5.7. The causal self attention layer

This layer does a similar job as the global self attention layer, for the output sequence

5.8. The feed forward network

The transformer also includes this point-wise feed-forward network in both the encoder and decoder. The network consists of two linear layers (tf.keras.layers.Dense) with a ReLU activation in-between, and a dropout layer.

5.9. The encoder

The encoder consists of a PositionalEmbedding layer at the input and a stack of EncoderLayer layers.

5.10. The decoder

Similar to the Encoder, the Decoder consists of a PositionalEmbedding, and a stack of DecoderLayer

5.11. Dense layer

It converts the resulting vector at each location into output token probabilities. The output of the decoder is the input to this final linear layer.

6. Model summary

Here is the summary used for training in this lab:

Model: "transformer"

Layer (type)	Output Shape	Param #
encoder_1 (Encoder)	?	3,620,480
decoder_1 (Decoder)	?	5,768,064
dense_38 (Dense)	?	1,026,195

Total params: 10,414,739 (39.73 MB)
Trainable params: 10,414,739 (39.73 MB)
Non-trainable params: 0 (0.00 B)

Figure 2: Model summary

7. Training

To train the transformer model, we use the Adam optimizer combined with a custom learning rate scheduler, following the formula from the original Transformer paper. This scheduler adjusts the learning rate dynamically to improve training stability and performance. Given that the target sequences include padding, we apply a padding mask during loss calculation to ensure accurate training. The loss function used is sparse categorical cross-entropy (tf.keras.losses.SparseCategoricalCrossentropy), which is well-suited for handling the tokenized output sequences by computing the loss based only on non-padded tokens.

We trained the model for 20 epochs, each epoch took approximately more than one hour.

8. Conclusion

In this lab, we implemented and trained a transformer model for automatic translation from French to English using the WMT14 dataset. The model demonstrated accurate predictions on two examples from the training set, indicating its ability to learn and generalize from the provided data. However, it struggled with predicting random words not present in the training data.

This discrepancy suggests several potential areas for improvement:

- Tokenizer Quality: The tokenizer might not be capturing the nuances of the language adequately, particularly for out-of-vocabulary words or uncommon phrases. Improving the tokenizer or using a more advanced tokenization method could enhance the model's performance.
- Training Duration: The number of training epochs might have been insufficient for the model to fully learn the intricate patterns of the language. Extending the training period could allow the model to achieve better accuracy and generalization.
- Model Architecture: While the transformer is a powerful architecture, specific modifications or enhancements might be necessary to improve its performance on challenging inputs. Experimenting with different architectures or adding more layers could potentially yield better results.

Overall, while the initial results are promising, further refinement in tokenizer quality, training duration, and model architecture could lead to significant improvements in the model's translation capabilities.