ATTENTION IS ALL YOU NEED

Deep Learning for NLP

Lucía Cuevas. 12338515
Lorenzo Ramírez. FILLTHIS

Importance of the paper

The paper "Attention is All You Need" by Vaswani et al. introduced the Transformer model. It revolutionized the field NLP by moving away from traditional recurrent and convolutional architectures with the use of self-attention mechanisms.

Before Transformers, RNNs (Recurrent Neural Networks) and CNNs (Convolutional Neural Networks) were predominant in NLP. Transformers have a more efficient and scalable architecture and significantly improved performance.

Self-Attention

Self-attention allows the model to weigh the importance of different words in a sentence, regardless of their position. This mechanism enables better understanding of context and relationships in the data.

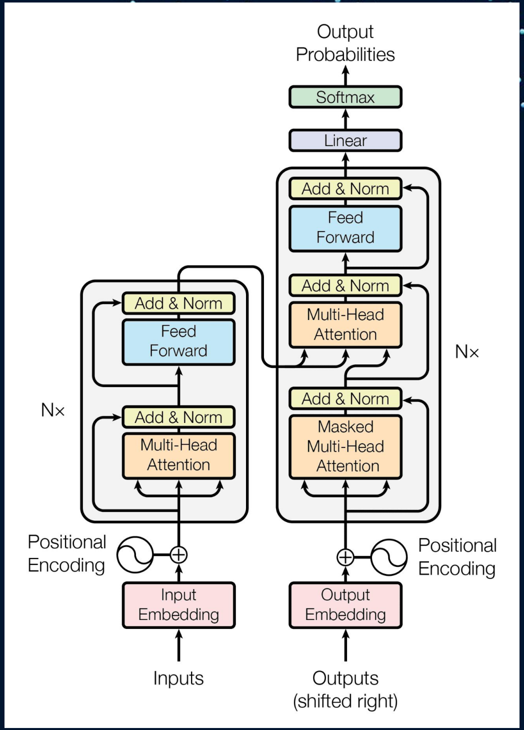
Transformer Architecture

The Transformer architecture consists of an encoder-decoder structure.

It uses self-attention and feed-forward neural networks. Unlike RNNs, it processes all tokens in parallel, making it faster and more efficient.

Transformers set new performance benchmarks across various NLP tasks. Their architecture is the foundation for many state-of-the-art models, such as BERT, GPT, and T5.

Key concepts: **Scaled dot-product** and **Multi-head attention**



Scaled Dot-Product

Every input vector is used as the Query, the Key and the Value. To obtain this roles, we need three weight matrices and compute three linear transformation for each xi.

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

These three matrices are K, Q and V, three learnable weight layers that are applied to the same encoded input, we can apply the attention mechanism of the input vector with itself, a “self-attention”. We used them (dot product of the query vector with the key vector of the respective word we’re scoring) to calculate the attention scores (how much focus to place on other words of the input sequence).

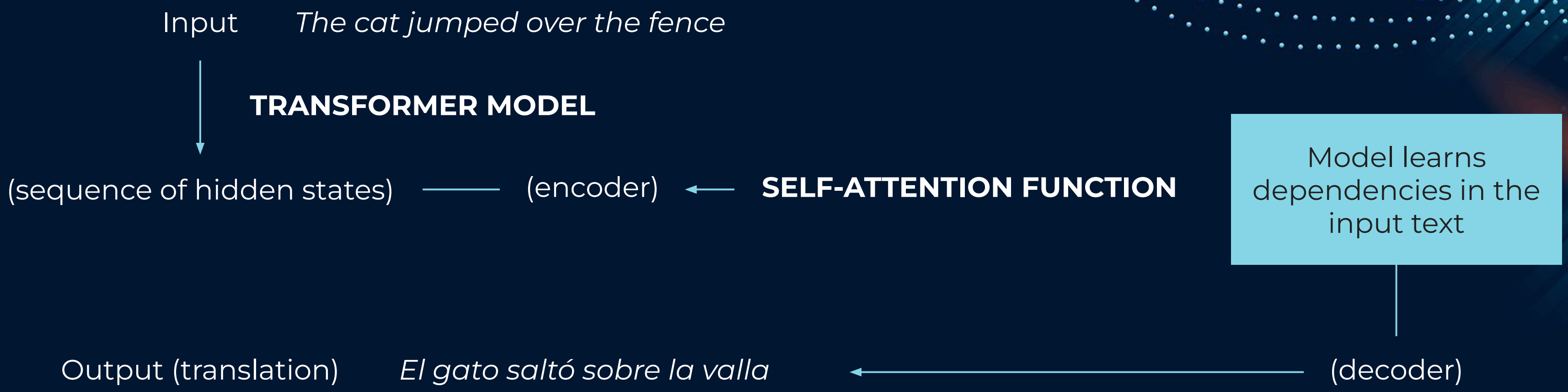
Next we apply the “scaled” factor to have more stable gradients. After “softmaxing” we multiply by the Value matrix to keep the values of the words we want to focus on.

Multi-head Attention

The attention scores are focused on the whole sentence at a time
→ same results with same words in a different order

Instead, we would like to attend to different segments of the words. We can combine several self attention heads, dividing the words vectors into a fixed number of chunks, and then self-attention is applied on the corresponding chunks, this produces h different output matrices of scores. After calculating the dot product of every head, we concatenate the output matrices and multiply them by an additional weights matrix.

Example


Input
The cat jumped over the fence
TRANSFORMER MODEL
(sequence of hidden states)
(encoder)
SELF-ATTENTION FUNCTION
Model learns dependencies in the input text
(decoder)
Output (translation)
El gato saltó sobre la valla

Reproducing the Paper

- Implementation in PyTorch
  - Failure to achieve good translations
  - Similar loss to next implementation
- Implementation inspired by:
  - Umar Jamil
  - https://github.com/hkproj/pytorch-transformer
  - https://www.youtube.com/watch?v=bCz4OMemCcA

Dataset

https://www.kaggle.com/code/sharanharsoor/spanish-to-english-translation

- English and spanish sentences
- ~140,000 pairs of sentences
- Seq length: 82

Hugging Face Text

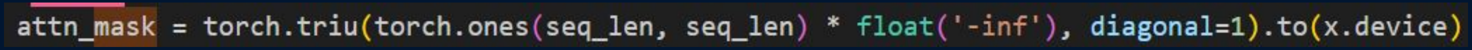
- Hugging ace Word Level Tokenizer due to relatively small dataset
- [SOS]: Start of sentence
- [EOS]: End of sentence
- [UNK]: Unknown token
- [PAD]: Padding token

Parameters

- D_model: 512 (embedding vector dimension)
- Dff: 2048 (projects embedding vector in feed forward step)
- Seq_length: 90 (length of all our sequences)
- Dropout: 0.1
- Heads: 8 (heads in multihead attention)
- Layers: 6 (how many encoder/decoder layers are chained)

Multihead Attention Implementation

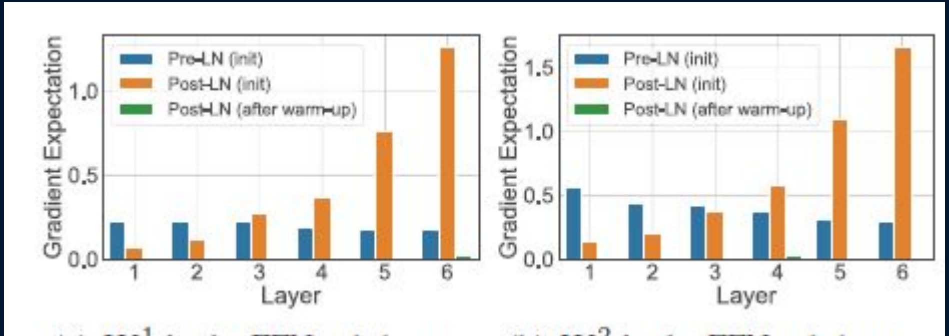
- torch.nn.MultiheadAttention attention is very general purpose
- More than likely reason why first implementation failed to learn properly
- Used once in the encoder and twice in the decoder
- Requires mask for both padding
- In the decoder a mask is required to prevent some words in the matrix to depend on each other



```
attn_mask = torch.triu(torch.ones(seq_len, seq_len) * float('-inf'), diagonal=1).to(x.device)
```

Modifications

- Experimental results show that layernorm should be done before each stage and not after



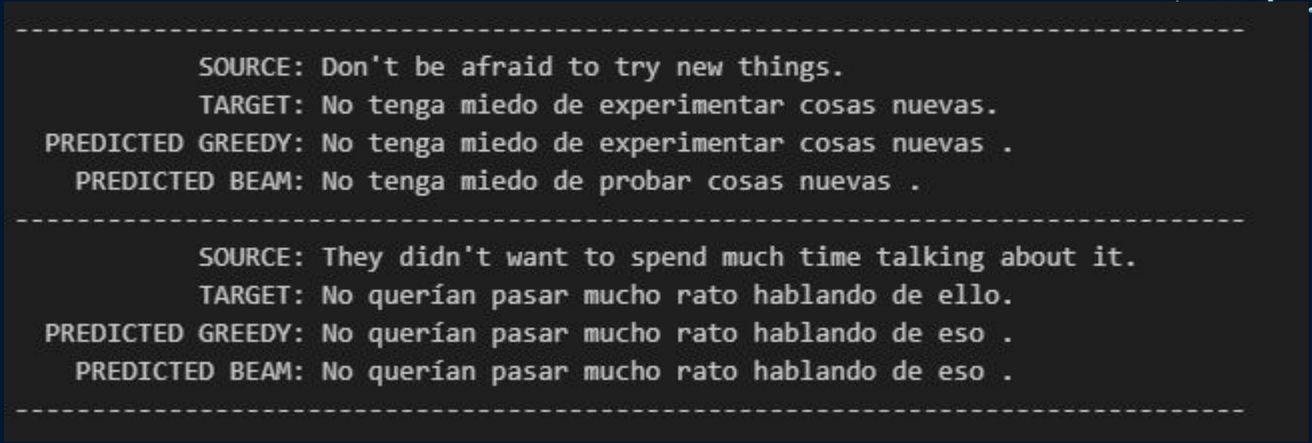
(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers

https://sh-tsang.medium.com/review-pre-ln-transformer-on-layer-normalization-in-the-transformer-architecture-b6c91a89e9ab

Benchmarks

- Ran on rtx 4070
- Average epoch length: 12min
- Epochs: 20

Example Translations



```
--------------------------------------------------------------------------------
          SOURCE: Don't be afraid to try new things.
          TARGET: No tenga miedo de experimentar cosas nuevas.
PREDICTED GREEDY: No tenga miedo de experimentar cosas nuevas .
  PREDICTED BEAM: No tenga miedo de probar cosas nuevas .
--------------------------------------------------------------------------------
          SOURCE: They didn't want to spend much time talking about it.
          TARGET: No querían pasar mucho rato hablando de ello.
PREDICTED GREEDY: No querían pasar mucho rato hablando de eso .
  PREDICTED BEAM: No querían pasar mucho rato hablando de eso .
--------------------------------------------------------------------------------
```

Challenges Encountered

- Tensor shapes due to batching
- Some details only mentioned briefly in paper
- Long training times even with small dataset