Attention is all you need - Literature review

Fegyó Benedek

September 2025

1 Introduction

The following report will contain a literature review of the Attention is all you need (Vaswani et al., 2017) paper concentrating on the historical motivation and the architectural design of the transformer attention mechanism.

2 What were the historical developments which motivated this paper

Before the introduction of the transformer architecture, recurrent models were considered being the state-of-the-art. This means, that the hidden state could be calculated with the following equation:

$$h_t = \sigma(U * x_t + W * h_{t-1}) \tag{1}$$

where σ is some form of non-linearity and U and W are weight matrices. The problem with this approach is that it cannot be parallelized across sequence positions, since the computation of h_t depends directly on h_{t-1}. This inherently sequential nature leads to inefficient training and slow inference, especially for long sequences.

The vanishing gradient problem also needed to be addressed as in order to efficiently convey context, long term dependencies are needed. For this LSTMs (long- short-term memory) and GRUs (gated recurrent unit) were introduced, that could propagate information. However this did not solve the problem as the gradient of the first input token still fades throughout the iteration even though its context might be important later.

The scalability issue could be alleviated by using hierarchical softmax to handle large vocabularies more effectively and conditional computation, in which only parts of the network are activated depending on the input, thus reducing overall computation. (Grave et al. 2016) (Bengio et al. 2015)

These improvements, even by contributing to scalability, could not solve the bottleneck arising from the sequential nature of the state-of-the-art models. Therefore the problem of long-range dependency handling while exploiting parallelism needed to be solved.

3 What were the key additions which this paper proposed to overcome earlier limitations

3.1 Scaled Dot Product Attention

The keys and queries are of dimension d_k and the values of dimension d_v. There are two attention mechanisms discussed, additive and dot-product. The latter was chosen, due to it being faster to calculate and more space efficient. However, since in that mechanism the values are multiplied and then fed into a softmax function, for large enough d_ks the additive attention outperforms the dot-product. To battle this effect, they introduced a scaling factor of $\frac{1}{\sqrt{d_k}}$. This creates the final attention formula of:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

where

- Q - Query matrix
- K - Key matrix
- V - Value matrix
- d_k - Dimensionality of the keys and queries

In the "encoder-decoder" attention layers, the keys and values are taken from the encoder's output, while the queries originate from the previous layer of the decoder. This addresses one issue, since each decoder layer can attend to every position in the encoder's output.

3.2 Multi-Head Attention

Instead of using a single attention head, they opted for a multi headed attention approach, with lower dimensionality to have an approximately equivalent computational cost. The benefit of this is that queries keys and values can be linearly projected h times with different, learned, linear projections. Then the attention function can be performed on each of these in parallel, yielding d_v dimensional output values. Concatenating and projecting these yields a final value, which allows the model to jointly attend to information from different representation subspaces at different positions.

3.3 Position-wise Feed-Forward Networks

Since the attention mechanism is a linear transformation, some form of non-linearity was needed in order to enable the model to learn more complex mappings and capture richer features. The same FFN is applied to independently to each token embedding, which is important as this ensures that each toke can be transformed individually into a higher-level representation.

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b2 \tag{3}$$

This helps the model transform the representation of the token after considering the context.

3.4 Positional Encoding

The model contains no recurrence or convolution, hence without an extra mechanism it would not be able to decide the difference between *"A likes B"* and *"B likes A"*, however different they might be. To overcome this challenge the relative or absolute position of the tokens in the sequence must be injected in some way. The proposed solution for this is adding a positional encoding "layer" at the bottom of the encoder and decoder stack.

$$PE_{pos,2i} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \tag{4}$$

$$PE_{pos,2i+1} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \tag{5}$$

where

- pos - position
- i - dimension

This approach gives each token a unique position signal, enabling the model to capture relative positions and generalise to longer sequences than it might have seen during training.

4 For each component of the transformer describe the key architectural design elements

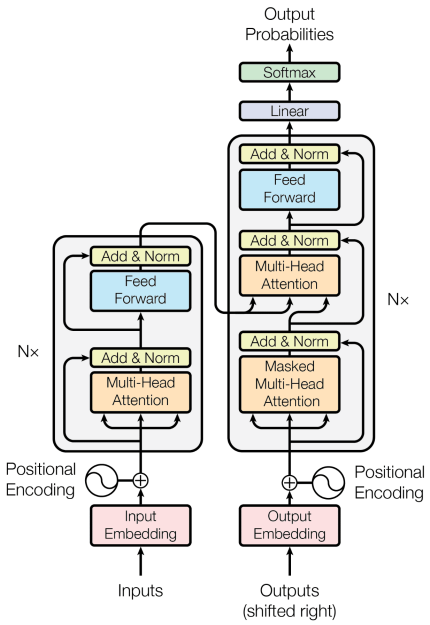

Figure 1: The proposed transformer architecture

- Encoder
  - Input Embeddings - Word to vector mapping
  - Positional Encoding - Information about the position in the sequence
  - Multi-Head Self-Attention - Context aware attention
  - Feed-Forward Network - Nonlienar transformation
  - Residual Connections and Layer Normalization - Stabilises deep training
- Decoder
  - Output Embeddings - Token to vector mapping
  - Positional Encoding - Information about the order of the sequence
  - Masked Multi-Head Self-Attention - Prevent future token leaks
  - Encoder-Decoder Attention - Focuses on encoder outputs
  - Feed-Forward Network - Nonlinear transformation

  - Residual Connections and Layer Normalization Stabilises deep training

- Output Layer
  - Linear Projection - Hidden layer to vocabularity mapping
  - Softmax Layer - Probability distribution output