Implementing Transformers

Winter semester 2024/25

Professorship: Dialog Systems and Machine Learning

by Timon Ludwigs
February 2, 2025

Supervisor: Dr. Carel van Niekerk

Contents

1 Introduction

The Transformer model, introduced by Vaswani et al. [7], marked a turning point in natural language processing by showcasing how self-attention mechanisms can efficiently capture dependencies in sequential data. This report details the practical implementation of Transformer architectures, exploring both theoretical underpinnings and experimental insights. Key features such as Multi-Head Attention, scalability, and replicability are emphasized.

2 Methodology

2.1 Understanding Attention and Multi-Head Attention Mechanism

The attention mechanism is the cornerstone of the Transformer architecture, enabling it to dynamically focus on relevant parts of the input sequence. It calculates a weighted sum of values (v) based on their relevance to a query (q) and keys (k), where Q represents the query matrix, K represents the key matrix, and V represents the value matrix. The formula for scaled dot-product attention can be expressed as follows:

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

The softmax operation ensures that the attention scores sum up to 1, transforming them into a probability distribution over the input elements. Scaling the dot-product by $\sqrt{d_k}$ prevents the gradients from becoming excessively large, stabilizing the training.

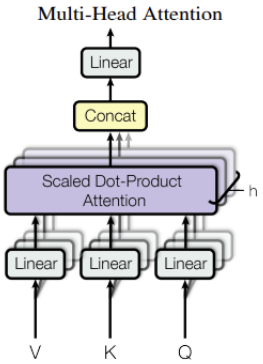

Figure 1: Multi-Head Attention mechanism enabling the model to focus on different parts of the input simultaneously[7].

In practice, a single attention mechanism might not capture all complex relationships between different elements of the input. To address this, the Transformer model uses multi-head atten-

tion (MHA), which computes multiple attention scores in parallel, each with different sets of learned query, key, and value weights as in Figure 1. The results of these parallel heads are then concatenated and linearly transformed to produce the final output. MHA allows the model to focus on different aspects of the input simultaneously, capturing richer contextual information.

2.2 Role of Positional Encoding

Since attention mechanisms lack inherent sequence information, positional encoding introduces a structured representation of token positions using sinusoidal functions. This encoding enables the model to differentiate tokens based on order while enhancing the interpretability of long-range dependencies. The positional encoding vector for a position p and a dimension i is computed using sine and cosine functions of different wavelengths. The formula for each position and dimension is:

$$\text{PE}(p, 2i) = sin\left(\frac{p}{10000^{\frac{2i}{d}}}\right) \text{ and } \text{PE}(p, 2i+1) = cos\left(\frac{p}{10000^{\frac{2i}{d}}}\right)$$

where d is the is the dimensionality of the positional encoding. Although sinusoidal encoding is the standard, other methods of positional encoding have been explored. Instead of using pre-defined sinusoidal functions, some implementations use learned embeddings for each position, which are optimized during training. These learned embeddings can potentially capture more complex relationships between token positions [5] [6]. The authors justified sinusoidal positional embeddings to enable generalization to sequence lengths beyond those seen during training, as the fixed mathematical structure supports extrapolation and by the following properties:

1. For a fixed offset k, the positional encodings PE_{p+k} can be represented as a linear function of PE_p.

Proof: The positional encodings are defined as:

$$\text{PE}(p, 2i+1) = cos\left(\omega_i p\right) \text{ and } \text{PE}(p, 2i) = sin\left(\omega_i p\right), \text{ with } \omega_i = \frac{1}{10000^{\frac{2i}{d}}}$$

Similarly for PE_{p+k}:

$$\text{PE}(p+k, 2i+1) = cos\left(\omega_i (p+k)\right) \text{ and } \text{PE}(p+k, 2i) = sin\left(\omega_i (p+k)\right)$$

Using the trigonometric sum identities $PE(p+k)$ is expanded to:

$$PE(p+k, 2i) = \sin(\omega_i p)\cos(\omega_i k) + \cos(\omega_i p)\sin(\omega_i k),$$

$$PE(p+k, 2i+1) = \cos(\omega_i p)\cos(\omega_i k) - \sin(\omega_i p)\sin(\omega_i k).$$

This can be written in matrix form:

$$\begin{bmatrix} PE(p+k, 2i) \\ PE(p+k, 2i+1) \end{bmatrix} = \begin{bmatrix} \cos(\omega_i k) & \sin(\omega_i k) \\ -\sin(\omega_i k) & \cos(\omega_i k) \end{bmatrix} \begin{bmatrix} PE(p, 2i) \\ PE(p, 2i+1) \end{bmatrix}.$$

The matrix is a rotation matrix, with the rotation angle determined by k and the frequency ω_i. Therefore, PE_{p+k} is a linear transformation of PE_p.

2. The wavelengths of the sinusoidal functions form a geometric progression from 2π to $10000 \cdot 2\pi$.

Proof: The frequency of the sinusoidal functions is given by $\omega_i = \frac{1}{10000^{\frac{2i}{d}}}$ and the corresponding wavelength λ_i is:

$$\lambda_i = \frac{2\pi}{\omega_i}.$$

Substituting ω_i:

$$\lambda_i = 2\pi \cdot 10000^{\frac{2i}{d}}.$$

The ratio of successive wavelengths is:

$$\frac{\lambda_{i+1}}{\lambda_i} = \frac{2\pi \cdot 10000^{\frac{2(i+1)}{d}}}{2\pi \cdot 10000^{\frac{2i}{d}}} = 10000^{\frac{2}{d}}.$$

Since the ratio is constant, the wavelengths form a geometric progression with a common ratio of $10000^{\frac{2}{d}}$. The range of the progression is:

- For $i = 0$: $\lambda_0 = 2\pi \cdot 10000^0 = 2\pi$,
- For $i = \frac{d}{2} - 1$: $\lambda_{d/2-1} = 2\pi \cdot 10000^{\frac{2(\frac{d}{2}-1)}{d}} = 10000 \cdot 2\pi$.

Thus, the wavelengths span a geometric progression from 2π to $10000 \cdot 2\pi$ [7].

The positional encoding is typically added to the word embeddings before being fed into the transformer model. This combined embedding vector allows the self-attention mechanism to attend not only to the content of each token but also to its position in the sequence. In practice, the positional encoding is simply element-wise added to the token embeddings:

$$\text{Input}_i = \text{Embedding}_i + \text{PE}(p)$$

Where Embedding_i is the token embedding for the i-th token and $\text{PE}(p)$ the positional encoding for the position of the token.

2.3 Importance of Residual Connections

Residual connections allow the model to retain input information while enabling deeper architectures to train effectively. They work by adding the input of a sub-layer to its output, ensuring better gradient flow and mitigating issues like vanishing gradients. This mechanism preserves critical information across layers, stabilizes training, and facilitates efficient representation learning by encouraging the model to focus on incremental refinements rather than learning transformations from scratch.

2.4 Layer Normalization Techniques

Layer normalization stabilizes the training dynamics by normalizing inputs at the layer level. Unlike batch normalization, which operates across the batch dimension, layer normalization normalizes the features of each input independently for a given token. This independence from batch size makes it particularly well suited for sequence-based models such as transformers. Layer normalization is applied to the input x of a layer by normalizing its features across the hidden dimension. For an input vector $x = (x_1, x_2, ..., x_d)$ with d features, the normalized output $\hat{x}$ is computed as:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Here, $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$ and $\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$ are the mean and variance of the input features. ϵ is a small constant added to prevent division by zero. To allow the model to learn optimal transformations, trainable parameters γ (scaling factor) and β (offset) are introduced. The final output of LayerNorm is given by:

$$\text{LayerNorm}(x_i) = \gamma \odot \hat{x}_i + \beta$$

where $\odot$ is the element-wise multiplication [1].

2.5 Position-wise Feed-Forward Networks

The position-wise feed-forward network (FFN) is a key component of the transformer architecture, applied independently to each token position in the sequence. It consists of two fully connected layers with ReLU activation in between:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

Here, W_1, W_2 are learned weight matrices and b_1, b_2 are biases. The FFN is applied identically to all positions, making it a position-wise operation that does not share information between tokens. This design allows the network to perform non-linear transformations on the token representations, enhancing their expressiveness while maintaining independence across positions. Combined with residual connections and layer normalization, the FFN contributes to the overall depth and capacity of the transformer model.

2.6 Comprehensive Architectural Overview

The transformer architecture is composed of two main parts: the encoder and the decoder, which are connected through the attention mechanism, as shown in Figure 2.

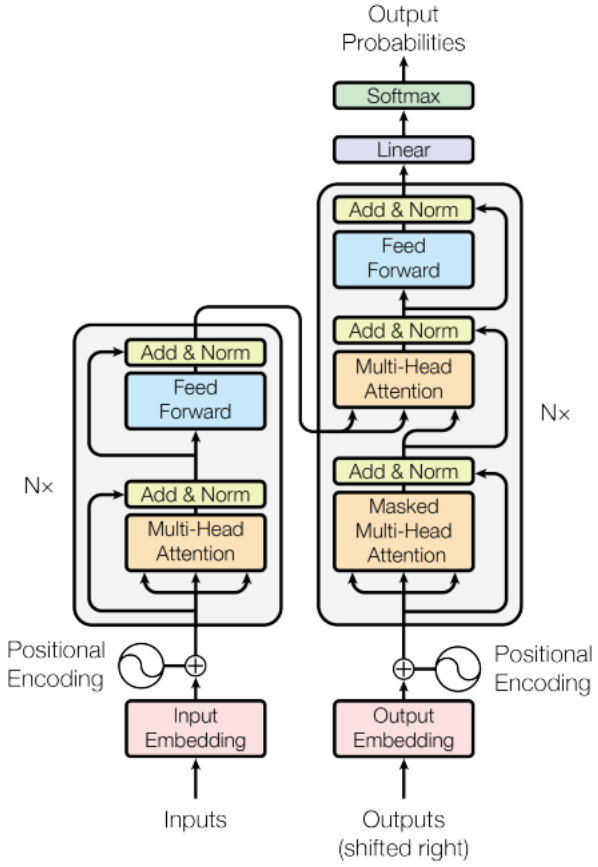

Figure 2: Complete Transformer architecture from original paper [7].

The encoder is responsible for processing the input sequence and generating a rich, contextual representation of it. It consists of multiple identical layers, each of which includes the subcomponents described before. The encoder outputs a sequence of contextual embeddings for the input tokens, which are passed to the decoder.

The decoder generates the target sequence token-by-token, leveraging the encoder's output. Like the encoder, the decoder is composed of multiple identical layers, with the major difference of masked multi-head attention. Masked multi-head attention is a specialized form of the multi-head attention mechanism, used in the decoder of the Transformer model to ensure the autoregressive nature of sequence generation. Unlike standard attention, which allows each position to attend to all other positions, masked multi-head attention restricts the attention scope so that a token at position p can only attend to tokens at positions $\leq p$. This is achieved by applying a mask to the attention scores, setting the values corresponding to future positions to $-\infty$ before applying the softmax function. This prevents the model from peeking at future tokens during training or inference, ensuring that predictions for a given position depend only on the preceding tokens.

3 Training

For training, the base model variation referenced in [7] was used. The WMT 2017 English-German dataset [3], containing roughly 6 million sentence pairs, served as the training data. Tokenization was performed using Byte Pair Encoding [2], producing a shared vocabulary of approximately 50,000 tokens for both the source and target languages. To standardize input processing, sentence pairs were limited to a maximum sequence length of 64 tokens.

3.1 Adam Optimizer in Transformer Models

The Adam optimizer is central to training transformer models due to its adaptive learning rates and efficient gradient updates. Vaswani et al. [7] initially employed Adam with L_2 regularization. However, AdamW [4], which introduces decoupled weight decay, has since become the preferred choice due to its improved performance.

Adam corrects biases in the first and second moment estimates (m_t and v_t) caused by their initialization at zero (line 2 of the Adam algorithm in Figure 3). Without correction, these estimates are biased toward zero in the early stages of training. Bias correction scales the estimates with a decaying factor over time, ensuring unbiased moments and improving optimization during early training.

In standard Adam, L_2 regularization interacts with the adaptive learning rate, as the weight de-

cay term influences the moving averages (m_t, v_t). This can lead to uneven regularization, where weights with large gradients are penalized less, reducing generalization. AdamW addresses this by decoupling weight decay from gradient updates, applying it directly to the weights after step size adjustments. This approach maintains the adaptive properties of the optimizer while ensuring consistent regularization.

To counteract gradient instability introduced by Post-Norm layers [9], AdamW is often paired with a warm-up and decay learning rate schedule. This strategy stabilizes training and facilitates efficient convergence, particularly in large-scale transformer models.

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Figure 3: Comparison of Adam and AdamW algorithms, showing the position of the weight decay term [4].

4 Results

To evaluate the flexibility of both positional encoding approaches a comprehensive robustness analysis framework that subjects the models to various input perturbations was implemented. This analysis aims to assess how well each encoding strategy maintains performance when faced with imperfect or degraded input sequences. Some of the following ideas were already tested using BERT [8] and tried out for this transformer implementation.

4.1 Methodology

The robustness testing framework implements three distinct perturbation types, each designed to simulate different kinds of real-world sequence degradation:

1. Middle Sequence Shuffling: This perturbation randomly reorders tokens in the middle section of the input sequence, affecting 50% of the sequence length centered around the midpoint. This test evaluates the models reliance on local word order and their ability to maintain contextual understanding despite position disruption.

2. Random Token Dropping: This test randomly removes 20% of input tokens, replacing them with padding tokens. It simulates scenarios with missing or corrupted information and tests the models ability to maintain consistency with incomplete sequences.

3. Token Repetition: The framework randomly duplicates 20% of tokens by copying them to adjacent positions. This perturbation evaluates how the models handle redundant information and maintain position awareness despite sequence expansion.

Each perturbation is applied independently during validation, and the models performance is measured using the same loss function used during training. A control test with unmodified sequences serves as a baseline for comparison.

4.2 Robustness Analysis Results

While both approaches show similar robustness patterns under various perturbations, several concerning patterns emerge. The learning curves indicate significant overfitting, with training loss continuing to decrease while validation loss plateaus early (Figure 4). The position-wise accuracy hovers around just 15%, suggesting poor model performance. The robustness tests show high overall loss values (> 7.0) across all conditions, with minimal variation between perturbation types (< 0.03 difference), indicating the model might not be effectively learning position-dependent features.

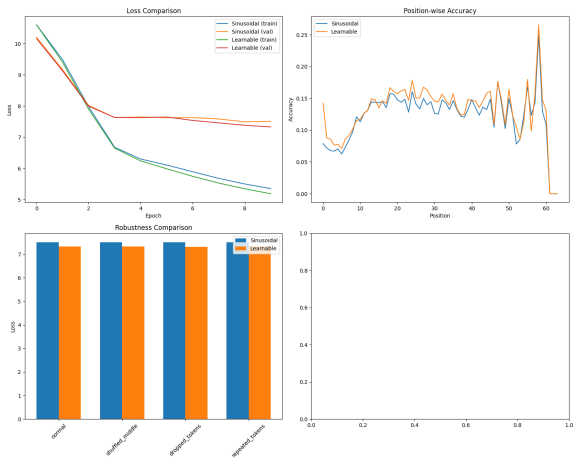

Figure 4: Analysis of robustness in sinusoidal and learnable positional encoding.

Word count: 1672

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450.

[2] Denny Britz et al. *Massive Exploration of Neural Machine Translation Architectures*. 2017. arXiv: 1703.03906 [cs.CL]. URL: https://arxiv.org/abs/1703.03906.

[3] Hugging Face. *WMT 2017 English-German Dataset Viewer*. Accessed: 2025-01-26. n.d. URL: https://huggingface.co/datasets/wmt/wmt17/viewer/de-en.

[4] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101.

[5] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. *Self-Attention with Relative Position Representations*. 2018. arXiv: 1803.02155 [cs.CL]. URL: https://arxiv.org/abs/1803.02155.

[6] Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[7] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[8] Benyou Wang et al. "On position embeddings in bert". In: *International Conference on Learning Representations*. 2020.

[9] Ruibin Xiong et al. *On Layer Normalization in the Transformer Architecture*. 2020. arXiv: 2002.04745 [cs.LG]. URL: https://arxiv.org/abs/2002.04745.