Implementing Transformers

Report

by

Florian Kark

born in

Düsseldorf

submitted to

Professorship for Dialog Systems and Machine Learning
Prof. Dr. Milica Gašić
Heinrich-Heine-University Düsseldorf

March 2024

Supervisor:
Carel Niekerk, PhD

Contents

List of Figures

Chapter 1

Introduction

The Transformer model, introduced in the seminal paper Vaswani et al. [14], revolutionized the field of natural language processing by demonstrating the effectiveness of self-attention mechanisms for capturing long-range dependencies in sequential data through leveraging so called Multi-Head Attention. The paper focuses on an efficient implementation, scalability, and replicability, laying the groundwork for subsequent successful Large Language Models that continue to shape NLP research today.

This report mirrors the process of the practical, starting with the attention mechanism and resulting in model assembly, training, and evaluation. To maintain the report's length, we omit trivial architectural details and instead highlight tasks from the practical, and address challenging or ambiguous aspects of the paper. Our aim is to offer fresh perspectives and insights, complementing the original paper with mathematically meticulous considerations and novel research findings.

Chapter 2

Methodology

First, we start by explaining the attention mechanism and from there motivate the other parts of the Transformer. We derive, proof and motivate the other parts of the architecture as encountered in the practicals and beyond. Moreover, we examine them closely if they are vaguely justified in the paper.

2.1 Attention

We start by revisiting the scaled dot product attention formulation as in Vaswani et al. [14]. We simplify the formulation by considering individual representations for query (q), key (k), and value (v) without the batch representation, leading to the following expression:

$$X^{\text{next layer}} = Attention(Q,K,V) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_\text{k}}}\right)\mathbf{V}$$
$$\Leftrightarrow x^{\text{next layer}} = \sum_j \text{softmax}\left(\frac{\mathbf{q}^\top\mathbf{k}_j}{\sqrt{d_\text{k}}}\right)\mathbf{v}_j$$

Now, the reason for the name "scaled dot product" becomes more clear. We proceed by expanding the softmax function. Since the softmax function serves as normalization, we simplify the denominator. Finally, we rewrite the q,k,v as $W_q x, W_k x, W_v x$ to receive the desired

representation of the formula:

$$
\begin{aligned}
x^{\text{next layer}} &= \sum_j \text{softmax}\left(\frac{\mathbf{q}^\top \mathbf{k}_j}{\sqrt{d_\text{k}}}\right)\mathbf{v}_j \\
&= \sum_j \frac{\exp(\mathbf{q}^\top \mathbf{k}_j/\sqrt{d_\text{k}})}{\sum_{z=1}\exp(\mathbf{q}^\top \mathbf{k}_z/\sqrt{d_\text{k}})}\mathbf{v}_j \\
&= \sum_j \frac{\exp(\mathbf{q}^\top \mathbf{k}_j/\sqrt{d_\text{k}})}{\text{normalization}}\mathbf{v}_j \\
\Leftrightarrow x_i^{\text{next layer}} &= \sum_j \frac{e^{x_i^T W x_j/\sqrt{d_\text{k}}}}{\text{normalization}} W_v x_j
\end{aligned}
$$

Here, $W = W_q^T W_k$ and x_i represents the vector for the ith word token. Given or word embedding space, we assume if vectors have the same meaning they align/point in the same direction. Thus, the term $e^{x_i^T W x_j/\sqrt{d_\text{k}}}$ serves as a filter for relevant words, emphasizing their importance based on their similarity to the current word. We filter v_j, which is in many code implementations equal to k_j, thus a self filtering mechanism.

Subsequently, the weighted sum over all relevant words contributes to the representation of the current word in the next layer. This mechanism facilitates the flow of contextual information, allowing words to influence each other's representations based on their semantic relationships.

A pivotal aspect lies in the learned transformation matrix W, which aligns words in semantic space. Consisting of transformations[1] that include rotation, stretching, and rotation again, W ensures that semantically related words point in similar directions.

The decision to divide attention scores by $\sqrt{d_\text{k}}$ is a normalization technique aimed at stabilizing gradient flow during training, thereby mitigating issues like exploding or vanishing gradients. While the authors provide insight into why dot products tend to become large, they omit the rationale behind the specific choice of $\sqrt{d_\text{k}}$.

Derivation: They assume that q and k are independent random variables with mean 0 and

[1] `https://en.wikipedia.org/wiki/Singular_value_decomposition`

variance 1. This can be assumed because we apply Layer normalization. In addition, we use Xavier initialization and so the computed keys, queries, and values will also have unit std.

$$
\begin{aligned}
qk &= \sum_{i=1}^{d_\mathrm{k}} q_i k_i \\
\Rightarrow Var(qk) &= Var(\sum_{i=1}^{d_\mathrm{k}} q_i k_i) \\
&= \sum_{i=1}^{d_\mathrm{k}} Var(q_i k_i), \text{ with } q, k \text{ independent} \\
&= \sum_{i=1}^{d_\mathrm{k}} 1 = d_\mathrm{k}, \text{ with } Var(q_i k_i) = 1
\end{aligned}
$$

Thus, to ensure stable gradient flow, the authors normalize by dividing by the standard deviation, which is $\mathrm{std}(qk) = \sqrt{Var(qk)} = \sqrt{d_\mathrm{k}}$.

By having established the scaled dot product attention mechanism, the subsequent components of the Transformer architecture will unfold naturally, with careful consideration of the requirements posed by the attention mechanism.

2.2 Positional Encoding

Upon tokenization of a text sequence, the inherent order is lost, restricting the attention mechanism to a bag-of-words representation. However, in human language, the meaning of a word is not only influenced by its neighboring words but also by their positional relationships. To address this, the authors introduce a positional encoding after the creation of the embeddings. They choose an absolute positional encoding based on a sinusoidal pattern, defined as follows:

$$\mathrm{PE}(pos, 2i) = \sin\left(pos/10000^{\frac{2i}{d_{\text{model}}}}\right)$$
$$\mathrm{PE}(pos, 2i+1) = \cos\left(pos/10000^{\frac{2i}{d_{\text{model}}}}\right)$$

The addition of a positional encoding to the word embeddings allows the attention mechanism to learn positional information alongside semantic relationships. Considering the denormalized filter term from the attention formula $x_i^\top W x_j$, where W determines the attention to be allocated to x_i given x_j, the positional encoding enhances this by incorporating positional information:

$$\begin{aligned}
&\Rightarrow (x_i + \mathrm{PE}_i)^\top W_q^\top W_k (x_j + \mathrm{PE}_j) \\
&= (W_q x_i + W_q \mathrm{PE}_i)^\top (W_k x_j + W_k \mathrm{PE}_j) \\
&= (W_q x_i)^\top W_k x_j + (W_q x_i)^\top W_k \mathrm{PE}_j + (W_q \mathrm{PE}_i)^\top W_k x_j + (W_q \mathrm{PE}_i)^\top W_k \mathrm{PE}_j \\
&= x_i^\top W x_j + x_i^\top W \mathrm{PE}_j + \mathrm{PE}_i^\top W x_j + \mathrm{PE}_i^\top W \mathrm{PE}_j
\end{aligned}$$

By adding positional encodings, the attention mechanism learns to attend not only to the content of words but also to their positions relative to each other, facilitating the understanding of word order within the sequence.

While various visualizations[2] illustrate the sequential patterns created by positional encoding based on sequence length and model dimension, understanding the distribution of distances provides deeper insights. Figure 2.1 makes the authors' choice of dividing the wavelength by 10000 apparent. It aims to ensure that the vector representations of positions maintain a distinctiveness proportional to their distances. This behavior, more prominent in longer sequences, is beneficial to tasks involving shorter sequences, such as sentence translation.

[2] `https://jalammar.github.io/images/t/attention-is-all-you-need-positional-encoding.png`

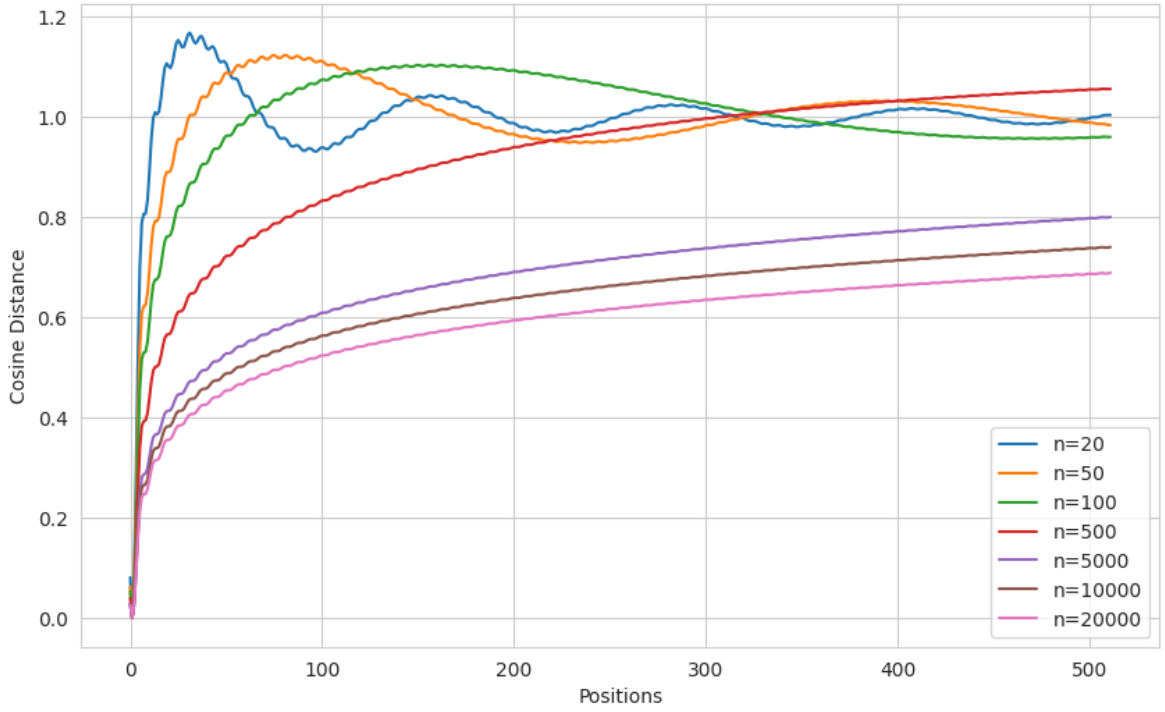

Figure 2.1: Distribution of cosine distances of the Positional Encoding for various n. Here, n is the base of the exponent in the denominator.

The authors justify the use of sinusoidal embeddings based on three key properties:

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

Proof: We utilize the angle sum identity from Trigonometry $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$ on $\text{PE}(pos+k, 2i)$ which yields:

$$\begin{aligned}
\text{PE}(pos+k,2i) &= \text{PE}(pos,2i)\cos\left(k/10000^{\frac{2i}{d_{\text{model}}}}\right) + \text{PE}(pos,2i+1)\sin\left(k/10000^{\frac{2i}{d_{\text{model}}}}\right) \\
&= \text{PE}(pos,2i)\cos\left(k/c^i\right) + \text{PE}(pos,2i+1)\sin\left(k/c^i\right), \text{ with } c = 10000^{\frac{2}{d_{\text{model}}}} \\
&= (\text{PE}(pos,2i), \text{PE}(pos,2i+1))(\cos\left(k/c^i\right), \sin\left(k/c^i\right))
\end{aligned}$$

which is a linear function of PE_{pos}.

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

Proof: Sinusoids, existing in both position and time, possess a spatial variable x representing the position on the dimension where the wave propagates, and a wave number (or angular wave number) k, which signifies the proportionality between the angular frequency ω and the linear speed v. The wave number is mathematically related to the angular frequency as follows:

$$k = \frac{\omega}{v} = \frac{2\pi f}{v} = \frac{2\pi}{\lambda}$$

where λ is the wavelength.

Solving for λ, we derive:

$$\begin{aligned} \lambda &= \frac{2\pi}{k}, \text{ here } k = \frac{1}{10000^{2i/d_{\text{model}}}} \\ \Rightarrow \lambda_i &= \frac{2\pi}{10000^{-2i/d_{\text{model}}}} \\ &= 2\pi \cdot 10000^{2i/d_{\text{model}}} \end{aligned}$$

Substituting $i = 0$ yields 2π, and for $i = d_{\text{model}}/2$, we obtain $2\pi \cdot 10000$.

To demonstrate that the term $\lambda_i = 2\pi \cdot 10000^{2i/d_{\text{model}}}$ forms a geometric progression, we need to establish that the ratio between consecutive terms remains constant.

Let $\lambda_i = 2\pi \cdot 10000^{2i/d_{\text{model}}}$ represent the ith term and $\lambda_{i+1} = 2\pi \cdot 10000^{2(i+1)/d_{\text{model}}}$ denote the $(i+1)$th term.

The ratio of consecutive terms is computed as follows:

$$\begin{aligned} \frac{\lambda_{i+1}}{\lambda_i} &= \frac{2\pi \cdot 10000^{2(i+1)/d_{\text{model}}}}{2\pi \cdot 10000^{2i/d_{\text{model}}}} \\ &= \frac{10000^{2(i+1)/d_{\text{model}}}}{10000^{2i/d_{\text{model}}}} \\ &= 10000^{2(i+1)/d_{\text{model}} - 2i/d_{\text{model}}} \\ &= 10000^{2(i+1-i)/d_{\text{model}}} \\ &= 10000^{2/d_{\text{model}}} \end{aligned}$$

As $2/d_{\text{model}}$ is a constant, the ratio $\frac{\lambda_{i+1}}{\lambda_i}$ remains constant $\forall i$.

Thus, the wavelengths λ_i indeed form a geometric progression from 2π to $2\pi \cdot 10000$.

3. The sinusoidal encoding is superior over learned embeddings, as it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

Initially, the superiority of sinusoidal encoding over learned embeddings seems intuitive because learned embeddings are fixed at the trained sequence length, while sinusoidal positional encodings offer a degree of flexibility, potentially enabling the model to predict positions beyond the maximum trained length. However, Press, Smith, and Lewis [10] empirically demonstrate that this intuition does not hold, as performance, measured in perplexity, deteriorates.

Additionally, the question arises: if sinusoidal encodings yield similar performance compared to learned embeddings, which is a feed-forward network (FFN), could other FFN network components, like position-wise feed-forward layers, additionally serve this function?

Interestingly, Haviv et al. [6] demonstrate that positional embeddings, regardless of type, are not required. Their experiments unveil that models develop an implicit grasp of absolute positions across the network, even in the absence of explicit positional information. Through exploratory experiments across diverse datasets, model sizes, and sequence lengths, the robustness of this phenomenon is affirmed. The authors posit that causal attention mechanisms empower models to infer the number of preceding tokens each token can attend to, effectively approximating its absolute position. Thus, positional awareness may also emerge from the implications of causal masks.

2.3 Residual connections

To address the potential overwrite of word representations by semantically related words in subsequent layers, caused by the attention mechanism, residual connections are employed. They preserve the identity of the word and its positional encoding, while also preventing rank collapse [4].

2.4 Layer normalization

Thirdly, Layer Normalization (LN) [1], originally developed for RNNs, is applied to both the input x and the output of the preceding sublayer $l_{-1}(x)$. It is defined as:

$$\text{LayerNorm}(x + l_{-1}(x))$$

where $\text{LayerNorm}(x)$ is given by:

$$\text{LayerNorm}(x) = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \varepsilon}} \odot \gamma + \beta$$

Here, $\odot$ represents element-wise multiplication, and ε is a small positive constant, typically $1e^{-5}$, to prevent division by zero.

LN shares similarities with Batch Normalization (BN) [7], except that it operates on a single observation at a time, reducing training time and stabilizing training by mitigating internal covariate shift and facilitating faster convergence. LN also exhibits scale independence, which prevents divergence, as evidenced by $\text{LayerNorm}(x) \approx \text{LayerNorm}(\alpha x)\ \forall \alpha \neq 0$, with equality as $|\alpha| \to \infty$.

While Vaswani et al. do not elaborate extensively on the rationale behind LN's specific usage, its preference in NLP tasks is attributed to empirical observations suggesting significant performance degradation with a naive use of BN. However, a comprehensive understanding of the underlying reasons remains elusive [12].

There is ongoing debate regarding the optimal placement of LN within the transformer architecture. Traditionally depicted post-sublayer in the original transformer figure, the updated code implementation[3] features LN pre-sublayer, known as Pre-LN ($x^{\text{next layer}} = x + \text{LayerNorm}(l_{-1}(x))$). Studies by Xiong et al. [16] advocate for Pre-LN, demonstrating improved performance and addressing vanishing gradient issues without the need for a warm-up stage, consequently reducing training time and hyperparameter tuning. However, this approach may lead to representation collapse [8]. Recent proposals by Xie et al. [15] ad-

[3] `https://github.com/tensorflow/tensor2tensor`

vocate for a fusion of Post-LN and Pre-LN connections, aiming to leverage their respective advantages while avoiding their limitations.

2.5 Position-wise Feed-Forward Network

Lastly, the position-wise feed-forward network, operating uniformly across sequence positions, transforms representations using a shared multi-layer perceptron:

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

This layer, applied independently to each position, encompasses two thirds of the transformer's parameter space. It introduces non-linearity and captures intricate feature interactions, crucial for mapping each position's representation to a higher-dimensional space. By maintaining isotropy within layers, the feed-forward network prevents convergence to a singular embedding, preserving individual token information [13, 4]. Additionally, it is speculated that the FFN stores factual knowledge within its weights [3], functioning similar to hierarchical hash maps [5].

2.6 Architecture

Combining the components discussed earlier, the transformer architecture employs a multi-head attention (MHA) block, comprising parallel attention layers, as in figure 2.2, to enhance representational capacity, learn long-range dependencies, and improve computational efficiency.

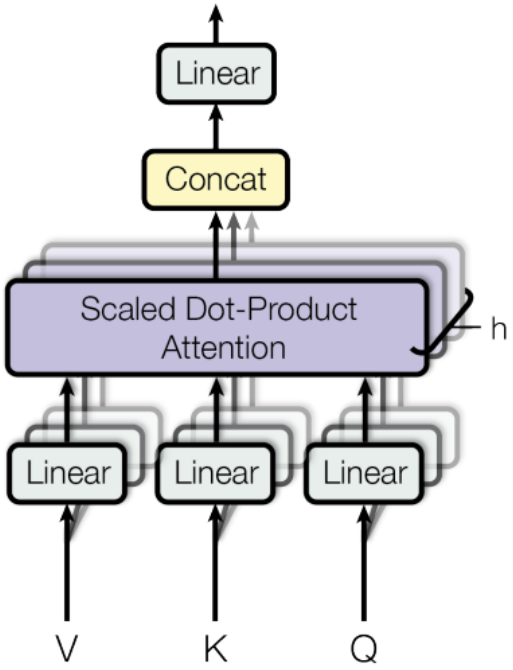

Figure 2.2: Depiction of the MHA block directing how to apply the earlier introduced Scaled Dot-Product Attention. Taken from the original paper [14].

The Transformer architecture, as can be seen in Figure 2.3, employs an encoder-decoder architecture, necessary for translation tasks. While we can directly adopt the previously discussed parts to create the encoder, we have to make some additional changes to build the decoder.

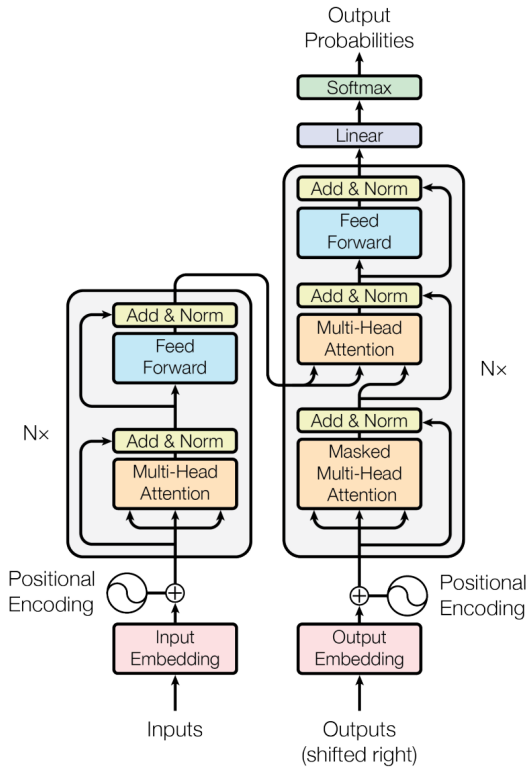

Figure 2.3: The overall Transformer architecture after assembling all previous building blocks. Taken from the original paper [14].

The first MHA block in the decoder masks the output sequence, preventing the model from peeking ahead while training on the entire sequence concurrently. This additional masking, along with padding masking, ensures the model avoids learning unnecessary padding and maintains generalization. The second MHA block computes attention based on the source sentence from the encoder, aiding in learning language semantics and grammar necessary for successful translation.

Despite the figure suggesting q, k are from the encoder and v from the decoder, in practice, q originates from the decoder, while k and v are sourced from the encoder, aligning with the attention mechanism explained earlier in section 2.1.

Lastly, an important implementation detail not illustrated in the figure refers to weight sharing between the two embedding layers and the linear output layer. Supported by findings from Press and Wolf [11], this strategy reduces parameters by 52% without notable perfor-

mance loss. However, it introduces a challenge: positional embeddings, initialized within the range of $[-1, 1]$, may overshadow signals from word embeddings. To mitigate this issue, the authors scale word embeddings by $\sqrt{d_{\text{model}}}$ (11.3 for 128, 22.6 for 512, and 32 for 1024), enabling them to be shared with the output embedding.

Chapter 3

Training

This chapter covers the training specifics. The code used to train and evaluate the model is provided at the following link `https://git.hhu.de/flkar101/transformer_project`

3.1 Model variation and Training Data

The model chosen for training is the base model variation from [14]. The model is trained on the standard WMT 2017 English-German dataset[1] consisting of about 6 million sentence pairs. Sentences were encoded using BPE [2], which has a shared source-target vocabulary of about 50000 tokens. The Sentence pairs were truncated to a maximum sequence length of 64. Each training batch had a size of 512 sentences.

3.2 Hardware and Schedule

Training was done on one NVIDIA A100 GPU. Each training step took about 0.57 seconds. The model trained for a total of 100,000 steps or 16 hours. f16 precision was utilized to gain a speed-up in training by 200%.

[1] `https://huggingface.co/datasets/wmt17/viewer/de-en`

3.3 Optimizer and Learning Rate Scheduler

While Vaswani et al. [14] uses Adam with L_2 regularization, in the practicals AdamW [9] is proposed, which was published afterward.

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

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

Figure 3.1: The algorithm of AdamW and Adam. Both have the weight decay term at a different position, highlighted accordingly by color. Taken from the original paper [9].

The practical asks to explain the reasoning behind the bias correction, which both optimizers have, and decoupled weight decay, which is introduced with AdamW, as illustrated by figure 3.1. Therefore, we make a case distinction on the Adam equation. For explaining Decoupled Weight Decay, we simplify the equation by not accounting for η_t and inserting line $6, 7, 8$ in 12:

For small t: The estimates of the first and second moments (m_t,v_t) are biased towards zero due to their initialization at zero (line 2). Bias correction alleviates this initialization bias by scaling the moments with a factor that decays exponentially over time. This ensures that the estimates become unbiased as the optimization progresses.

For $t \rightarrow \infty$:

$$\begin{aligned}\theta_t &= \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \\ &\rightarrow \theta_{t-1} - \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)(\nabla f_t + \lambda \theta_{t-1})}{\sqrt{v_t} + \varepsilon}\end{aligned}$$

Now, the L2 regularization from Adam (violet term in line 6) appears in the counter of the update equation and the moving averages of the gradient and its square (m,v) keep track not only of the gradients of the loss function but also of the regularization term. This leads to that the regularization term is normalized by $\sqrt{v}$ as well. Thus, if the gradient of a certain weight is large, the corresponding v is large too and the weight is regularized less than weights with small and slowly changing gradients. This in return reduces generalization capabilities.

The authors, therefore, suggest performing weight decay only after controlling the parameter-wise step size (line 12). The weight decay or regularization term does not end up in the moving averages, and is thus only proportional to the weight itself.

We couple the optimizer with the proposed learning rate scheduling to address the gradient instability introduced through Post-Norm [16], resulting in a learning rate as shown in figure 3.2

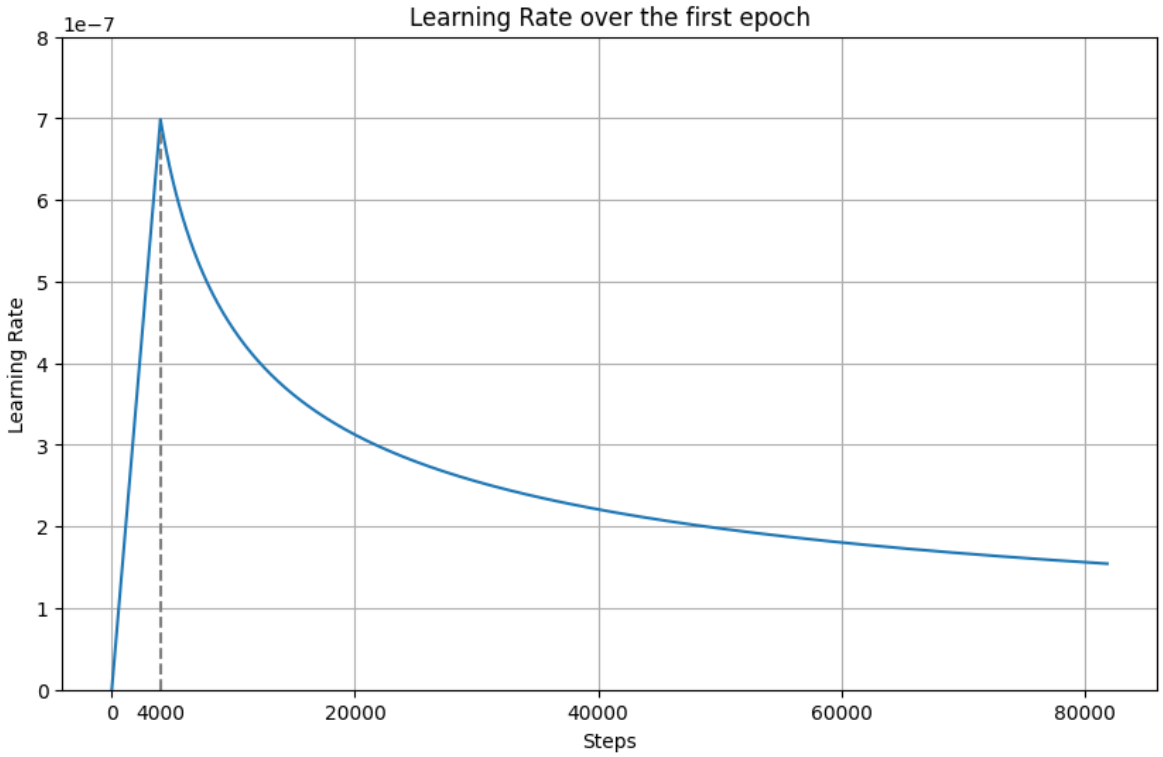

Figure 3.2: Visualization of the proposed learning rate schedule by [14] for the first training epoch and AdamW with a base learning rate of 0.001 and d_{model} of 512.

Chapter 4

Results

Unfortunately, the model was not able to achieve a lower train loss than 4.0. Even exhaustive hyperparameter tuning and architecture optimization did not yield a successful training run until the deadline. Thus, the generated results are very poor. Even exhaustive hyperparameter tests and architecture changes did not yield a successful training run until the deadline.

We will make an additional effort and aspire to demonstrate the outcomes in the presentation.

4.1 Translations

-

4.2 Score

-

Word count: 2715

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[2] Denny Britz et al. “Massive exploration of neural machine translation architectures”. In: *arXiv preprint arXiv:1703.03906* (2017).

[3] Damai Dai et al. “Knowledge neurons in pretrained transformers”. In: *arXiv preprint arXiv:2104.08696* (2021).

[4] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. “Attention is not all you need: Pure attention loses rank doubly exponentially with depth”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2793–2803.

[5] Mor Geva et al. “Transformer feed-forward layers are key-value memories”. In: *arXiv preprint arXiv:2012.14913* (2020).

[6] Adi Haviv et al. “Transformer language models without positional encodings still learn positional information”. In: *arXiv preprint arXiv:2203.16634* (2022).

[7] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[8] Liyuan Liu et al. “Understanding the difficulty of training transformers”. In: *arXiv preprint arXiv:2004.08249* (2020).

[9] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[10] Ofir Press, Noah A Smith, and Mike Lewis. “Train short, test long: Attention with linear biases enables input length extrapolation”. In: *arXiv preprint arXiv:2108.12409* (2021).

[11] Ofir Press and Lior Wolf. "Using the output embedding to improve language models". In: *arXiv preprint arXiv:1608.05859* (2016).

[12] Sheng Shen et al. "Powernorm: Rethinking batch normalization in transformers". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8741–8751.

[13] Shashank Sonkar and Richard G Baraniuk. "Investigating the Role of Feed-Forward Networks in Transformers Using Parallel Attention and Feed-Forward Net Design." In: *CoRR* (2023).

[14] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[15] Shufang Xie et al. "ResiDual: Transformer with Dual Residual Connections". In: *arXiv preprint arXiv:2304.14802* (2023).

[16] Ruibin Xiong et al. "On layer normalization in the transformer architecture". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10524–10533.