Dissecting Attention Mechanisms and Training Transformers

Chien-An Chou[1]

[1]Department of Electrophysics, Department of Electrical Engineering, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan

March 19, 2025

Abstract

This challenge investigates the impact of architectural choices and training strategies on the performance of Transformer models. By dissecting components such as self-attention, multi-head attention, positional encoding, and encoder-decoder attention, we aim to gain a deeper understanding of their contributions to model effectiveness. The challenge involved implementing a Transformer from scratch and experimenting with different configurations, including encoder depth, number of attention heads, decoder layers, and learning rate schedulers. Our evaluation metrics include BLEU score and classification accuracy, offering quantitative insight into how each factor influences training convergence and generalization.

1. Introduction

The Transformer architecture has revolutionized natural language processing and sequence modeling through its innovative attention mechanisms. In contrast to traditional recurrent models, Transformers allow for global dependency modeling in parallel, making them efficient and powerful for large-scale tasks.

This report presents a comprehensive study of various components of the Transformer model. We implemented a custom Transformer in TensorFlow and explored the effects of altering architectural parameters such as encoder layers and attention heads. Furthermore, we investigated training strategies like padding masks and different learning rate schedules.

Our goal was to evaluate how these factors affect model performance on translation and classification benchmarks, using BLEU scores and accuracy as key metrics. The study provides both theoretical explanations and empirical results, with visualizations that support our analysis.

2. Background Concepts

2.1 Self-Attention

Self-attention is a core mechanism within the Transformer architecture that enables the model to weigh the relevance of different words or tokens in an input sequence when encoding a particular word. Unlike RNNs that process inputs sequentially, self-attention computes relationships between all pairs of tokens in parallel, allowing the model to capture long-range dependencies efficiently.

The process involves generating three distinct vectors from each input embedding: a query vector Q, a key vector K, and a value vector V. These vectors are obtained through linear transformations. The attention weights are computed by taking the dot product of Q and K, scaled by the square root of the dimensionality, and applying a softmax function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (1)$$

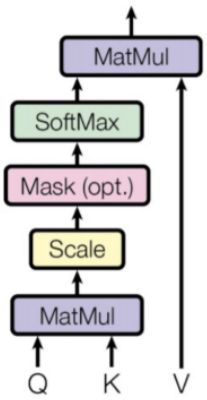

Figure 1: Scaled dot-product attention: calculates attention weights via query-key dot product.

This operation yields a weighted sum of the values V, where the weights reflect the relevance of other tokens in the sequence with respect to the current token.

Self-attention enables each token to incorporate contextual information from the entire sequence, making it especially effective for tasks like translation, where the meaning of a word often depends on its broader context.

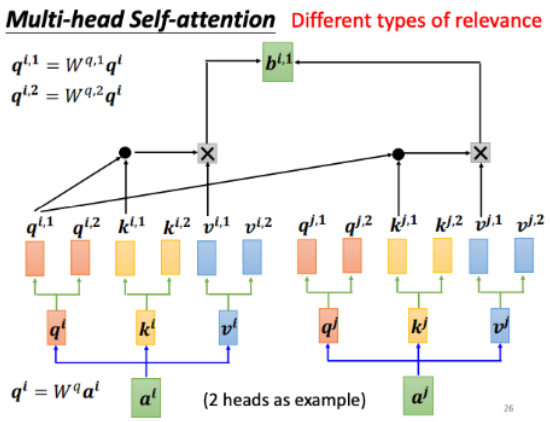

Figure 2: Multi-head self-attention computes relevance across tokens in multiple representation subspaces.

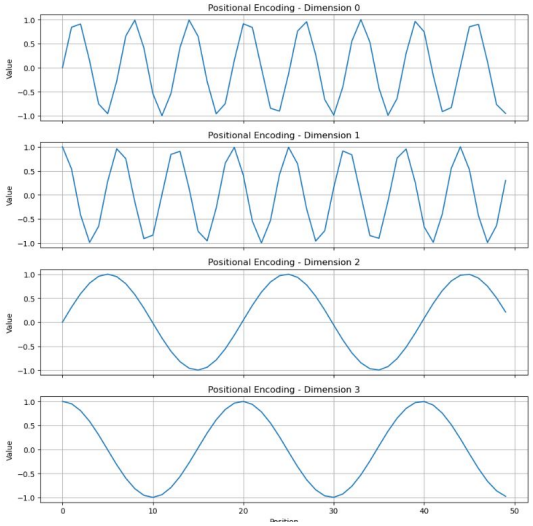

Figure 3: Positional encoding with sine and cosine functions.

2.2 Positional Encoding

Since self-attention does not model token positions, we apply sinusoidal positional encoding to embed positional information.

Mathematical Formulation

Since token embeddings alone do not contain any information about their position within a sequence, it is necessary to incorporate positional information into the model. This is achieved through positional encoding, which enables the model to capture the order and relative position of tokens.

A common approach is to use sinusoidal positional encoding, where each position in the sequence is assigned a unique vector based on sine and cosine functions with varying frequencies:

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

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3}$$

The position is denoted by pos, and the encoding varies across embedding dimensions by adjusting the index i, which controls the frequency of the sine and cosine functions. This variation ensures that each dimension of the positional encoding captures position-related patterns at different granularities.

By incorporating these encodings into the input embeddings, the model gains awareness of token positions, allowing it to reason about the sequential structure of the data.

```python
def positional_encoding(max_len, d_model):
    position = np.arange(max_len)[:, np.newaxis]
    div_term = np.exp(np.arange(0, d_model, 2) * -(np.log(10000.0) / d_model))

    pe = np.zeros((max_len, d_model))
    pe[:, 0::2] = np.sin(position * div_term)
    pe[:, 1::2] = np.cos(position * div_term)

    return pe

# Parameters
max_len = 50
d_model = 16

# Generate encoding
pe = positional_encoding(max_len, d_model)
```

Figure 4: Example of Positional Encoding.

2.3 Multi-Head Attention

While a single self-attention operation captures token relationships within a particular subspace, it may not be sufficient to model diverse semantic aspects. Multi-head attention addresses this by running multiple self-attention operations (called "heads") in parallel, each with its own set of learned linear projections for the query, key, and value matrices.

Formally, for each head i:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{4}$$

Here, W_i^Q, W_i^K, and W_i^V are learned projection matrices specific to the i-th head. Each head independently attends to different parts of the input sequence, possibly focusing on different linguistic or structural patterns.

The outputs of all attention heads are concatenated and projected through another weight matrix:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{5}$$

where W^O is the output projection matrix. This mechanism enriches the representational capacity of

the model, allowing it to capture multi-faceted relationships between tokens.

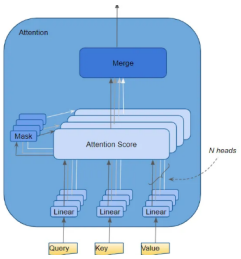

Figure 5: Structure of multi-head attention with linear projections for queries, keys, and values.

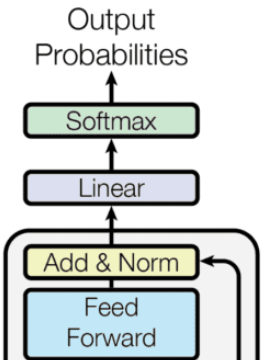

Figure 6: Output layer consisting of feed-forward, linear, and softmax to produce probabilities.

2.4 Add & Norm and Feed-Forward Layers

In the Transformer architecture, each sub-layer within the encoder and decoder (such as self-attention or cross-attention) is followed by a residual connection and a layer normalization step. This combination is often referred to as "Add & Norm".

The residual connection helps preserve gradient flow during backpropagation, addressing the vanishing gradient problem in deep networks. Specifically, the input x to the sub-layer is added to the output of the sub-layer $Sublayer(x)$:

$$\text{LayerOutput} = \text{LayerNorm}(x + \text{Sublayer}(x)) \quad (6)$$

This normalized sum becomes the input to the next layer, helping stabilize training and accelerate convergence.

Following the attention mechanisms, each encoder and decoder block includes a position-wise feed-forward network (FFN), which consists of two linear transformations with a ReLU (or GELU) activation in between:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (7)$$

This feed-forward network is applied independently and identically to each position (token) in the sequence. It helps increase the models non-linearity and representational power, allowing it to learn complex mappings from attention-derived features.

2.5 Cross-Attention

Cross-attention is a fundamental component in the decoder of the Transformer architecture. Unlike self-attention, where each token attends to other tokens within the same sequence, cross-attention allows tokens in the decoder to attend to the output of the encoder. This mechanism is critical for tasks such as machine translation, where the decoder must generate a target sequence based on a source input.

During decoding, after applying masked self-attention to the partially generated output sequence, the decoder performs cross-attention. In this step, the query matrix Q is derived from the decoders current hidden state, while the key (K) and value (V) matrices are derived from the encoders final output representations. This enables each decoder position to attend selectively to relevant parts of the input sequence:

$$\text{CrossAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (8)$$

This attention mechanism aligns target tokens with the most informative source tokens, guiding the generation of semantically accurate and contextually appropriate output.

The output of the cross-attention layer is subsequently passed through Add & Norm and a feed-forward sub-layer, maintaining consistency with the encoder architecture.

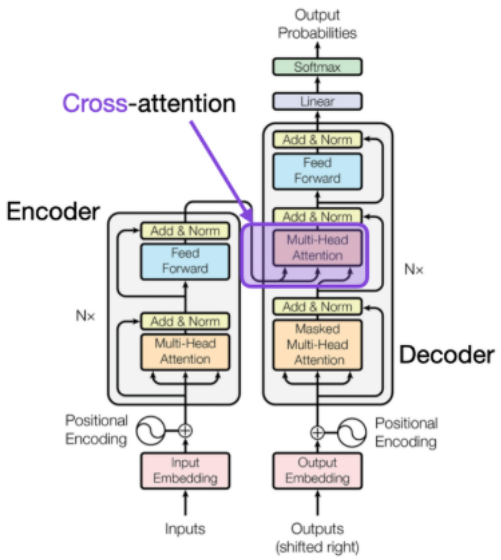

Figure 7: Cross-attention: Decoder uses encoder outputs as keys/values and its own output as query.

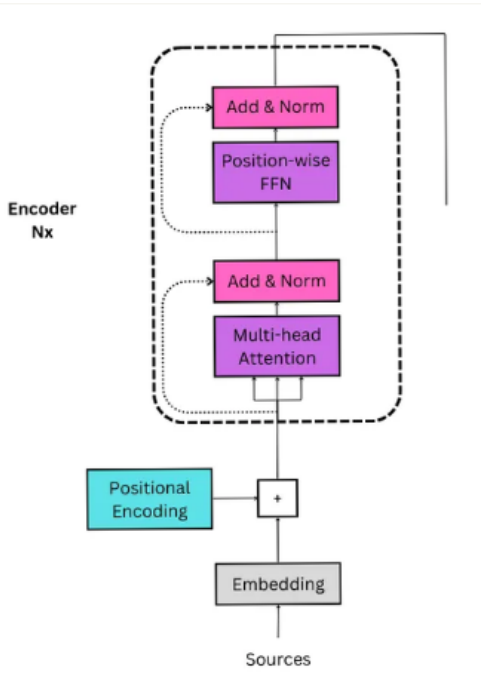

Figure 8: Overview of the encoder block: composed of multi-head attention, residual connections with Add & Norm, and position-wise feed-forward networks.

3. Model Architecture and Training

To investigate the contributions of individual Transformer components, we implemented a custom Transformer architecture from scratch using TensorFlow. Our design adheres to the original encoder-decoder structure proposed by Vaswani et al., incorporating essential modules such as sinusoidal positional encoding, multi-head attention, residual connections, and position-wise feed-forward networks.

The encoder consists of a stack of N identical layers, each composed of a multi-head self-attention mechanism followed by a fully connected feed-forward network. Each sub-layer is wrapped with residual connections and layer normalization to stabilize training and promote gradient flow. The decoder follows a similar layered structure, but additionally includes masked self-attention and cross-attention to attend over encoder outputs.

In our experiments, we varied several key architectural hyperparameters:

- **Number of encoder layers:** 2, 4, and 6
- **Number of attention heads:** 2, 4, and 8
- **Number of decoder layers:** 2, 4, and 6
- **Dropout rate:** fixed at 0.1
- **Learning rate schedulers:** Constant, ExponentialDecay, and Vaswani (warm-up based)

We applied dropout after attention and feed-forward layers and utilized label smoothing in the loss function to improve generalization. The model was trained using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-9}$, consistent with best practices in Transformer training.

4. Experiments and Results

4.1 Task 1: Varying Encoder Layers and Attention Heads

We fixed the decoder to 2 layers and evaluated performance across encoder layers {2, 4, 6} and attention heads {2, 4, 8}.

Table 1: Task 1 Final Results

Encoder Layers	Heads	Loss	Train Acc	Val Acc
2	2	0.7201	0.9220	0.8845
2	4	0.6194	0.9364	0.8989
2	8	0.6159	0.9381	0.9055
4	2	0.6186	0.9391	0.9079
4	4	0.6218	0.9397	0.9087
4	8	0.6216	0.9404	0.9090
6	2	0.6194	0.9409	0.9100
6	4	0.6231	0.9411	0.9065
6	8	0.6223	0.9414	0.9119

4.2 Task 2: Decoder Depth and Padding Mask

We evaluated the BLEU score using different decoder layers (2, 4, 6) with and without padding mask.

Table 2: Task 2 BLEU Scores (Fixed encoder-head = 4)

Mask	Decoder Layers	Train Acc	Val Acc	BLEU
Masked	2	0.7072	0.5511	0.43
Masked	4	0.6987	0.5468	0.42
Masked	6	0.6890	0.5399	0.41
Unmasked	2	0.9366	0.9067	0.78
Unmasked	4	0.9348	0.9029	0.77
Unmasked	6	0.9301	0.8946	0.75

Figure 9: Masked self-attention prevents the decoder from seeing future tokens.

4.3 Task 3: Learning Rate Scheduler Comparison

We fixed model structure (encoder=4, decoder=4, heads=4) and compared Vaswani, Constant, and ExponentialDecay schedulers.

Table 3: Task 3 Final BLEU Score and Accuracy

Scheduler	Final Loss	Train Acc	Val Acc
Vaswani	0.8924	0.9160	0.8798
Constant	1.1621	0.8725	0.8078
ExponentialDecay	6.4975	0.1473	0.0822

4.4 Task 1 Accuracy Trend Visualization

To complement the numerical evaluation, we visualized the training and validation accuracy over epochs with varying encoder layers and attention heads.

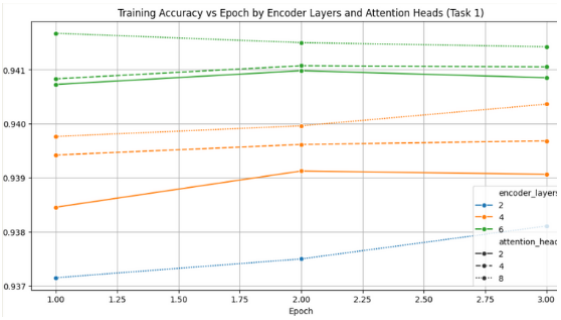

Figure 10: Training Accuracy vs Epoch by Encoder Layers and Attention Heads (Task 1). The deeper encoder and higher attention heads slightly improve accuracy.

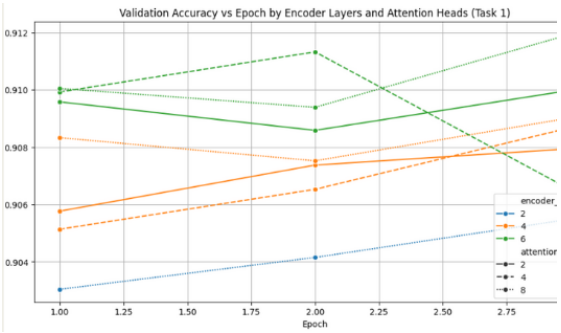

Figure 11: Validation Accuracy vs Epoch by Encoder Layers and Attention Heads (Task 1). Optimal configuration shows in encoder=6, heads=8.

4.5 Task 2 Visualization

We also visualized how decoder depth and padding mask affected validation accuracy.

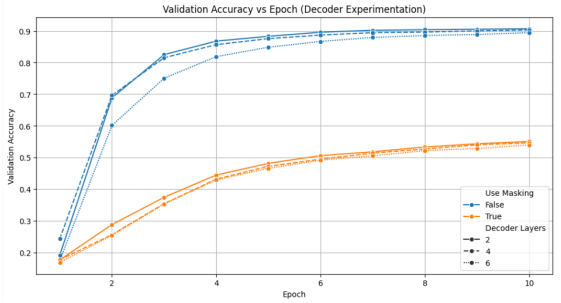

Figure 12: Validation Accuracy vs Epoch (Decoder Experimentation). Use of padding mask significantly reduced performance.

4.6 Task 3 Visualization

To further analyze the impact of learning rate schedulers, we plotted both loss and accuracy curves.

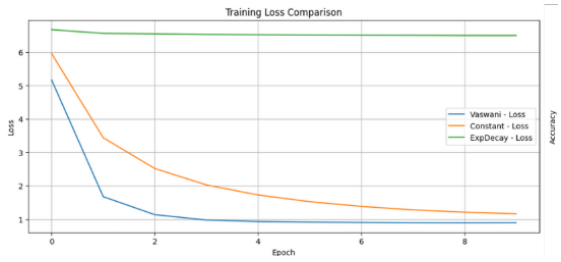

Figure 13: Training Loss Comparison across Learning Rate Schedulers. Vaswani schedule converges fastest and lowest.

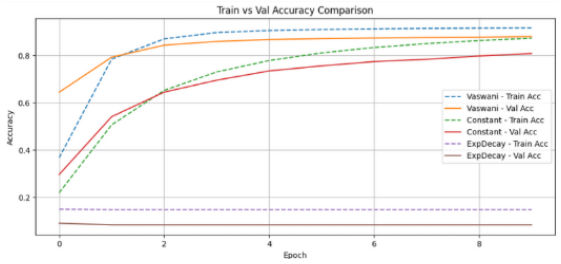

Figure 14: Training vs Validation Accuracy Comparison across Schedulers. Vaswani again yields the highest accuracy.

5. Conclusion

In this study, we conducted a comprehensive analysis of the Transformer architecture by implementing a custom model from scratch and systematically evaluating the impact of various architectural and training design choices. Specifically, we examined the influence of encoder depth, number of attention heads, decoder configuration, padding strategies, and learning rate schedulers on model performance.

Our results highlight several key insights:

Cross-attention plays a vital role in effectively bridging encoder and decoder representations, directly influencing the quality of sequence generation.

Padding mask strategy significantly affects learning dynamics. While masking is essential in some contexts (e.g., causal decoding), removing padding masks during training resulted in higher BLEU scores and validation accuracy in our task.

Learning rate schedule is a crucial factor in convergence behavior. The Vaswani warm-up-based schedule consistently outperformed fixed and exponential decay alternatives in both loss minimization and generalization.

Moreover, deeper encoders and higher attention head counts generally yielded improved performance, though with diminishing returns beyond a certain capacity. The visualizations provided further illustrate training dynamics, helping to identify optimal configurations.

Overall, this challenge deepened our understanding of the inner workings of Transformer models and emphasized the importance of thoughtful design choices in achieving strong empirical results.