# Final Report CIS-5930 - **Detailed Comparative study and implementation of the paper: Attention Is All You Need, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.**

Shaurya Tiwari[1] and Aditya Sugandhi[2]

[1]Graduate, Department of Computer Science, FSU

Keywords:
Transformer architecture
Learning rates
Hyperparameters
Perplexity score
Validation loss
Text completion
Model optimization

Abstract In this research paper, we detail our development of a text completion model utilizing the Transformer architecture. We focused on the effects of varying learning rates by maintaining a consistent 50,000 epochs for each of our 15 training iterations, using different hyper-parameters. We analyzed training and validation loss trends over these epochs and assessed performance through the perplexity score, a key metric for language models. The model was trained on four NVIDIA A100 GPUs with the same specifications. The training time per epoch was approximately 0.09 second. By leveraging parallel processing on multiple GPUs using the PyTorch and Tensorflow library in Python, we were able to accelerate the training process and achieve faster convergence. Our results showed that **optimal learning [specified in conclusion] rates and data splits** significantly minimized validation loss and perplexity score, marking them as the most effective configurations for producing coherent and fluent text. This study underscores the critical role of **precise hyper-parameter** adjustments in enhancing Transformer-based models, offering profound insights for ongoing research in this field.

 Submitted: April 30, 2024 as the Final Report Project in Data Science CIS-5930.

## 1. Introduction

Transformers have emerged as a revolutionary deep learning architecture that has transformed the landscape of artificial intelligence (AI) research and applications. Introduced in 2017, transformers have demonstrated remarkable success in various domains, including natural language processing (NLP), computer vision, and even audio processing.

At the core of a transformer lies the attention mechanism, which allows the model to efficiently capture long-range dependencies and contextual information. Unlike traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), transformers do not rely on sequential processing, making them more parallelizable and faster to train.

The transformer architecture consists of two main components: the **encoder and the decoder**. The encoder receives an input sequence, processes it, and generates a representation that captures the essential features of the input. The decoder, on the other hand, uses this representation along with the input to generate an output sequence.

Transformers have found widespread applications in various domains. In natural language processing, they have revolutionized tasks such as machine translation, text summarization, and language generation. Transformer-based models like GPT (Generative Pretrained Transformer) and BERT (Bidirectional Encoder Representations from Transformers) have achieved state-of-the-art results in numerous NLP benchmarks.

Beyond NLP, transformers have also made significant strides in computer vision, where they have been used for tasks like image classification, object detection, and segmentation. The ability of transformers to capture long-range dependencies has proven particularly useful in these domains.

## 2. Problem Statement

The optimization of hyperparameters, particularly learning rates, within Transformer-based text completion models, is critical for achieving high performance in natural language processing tasks. Despite the capabilities of these models, identifying the most effective learning rates and data split configurations that can consistently minimize validation loss and perplexity remains a challenge. This research aims to systematically explore the impact of varying learning rates and training epochs on the performance of Transformer models, targeting the reduction of validation loss and improvement of text coherence and fluency. The study seeks to establish optimal hyperparameter settings that maximize model efficiency and effectiveness, providing a framework for enhancing model training processes and outcomes in real-world applications.

## 3. Related Terms/Equations

1. **Multi-Head Self-Attention Mechanism: Key, Query, and Value Computation:**

$$K = W^K X, \quad Q = W^Q X, \quad V = W^V X$$

Where $K, Q, V$ represent the matrices for keys, queries, and values respectively, $X$ is the input matrix, and $W^K, W^Q, W^V$ are the parameter matrices in the linear transformations within each head.

2. **Scaled Dot-Product Attention:**

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

Here, $\sqrt{d_k}$ is the scaling factor for normalization, with $d_k$ being the dimension of the keys and queries. This scaling helps in stabilizing the gradients during training.

3. **Masking and Dropout:** Before applying the softmax function, a triangular mask is applied to the matrix to ensure that the prediction for a position does not depend on the positions that come after it. After softmax, dropout is applied as a form of regularization.

4. **Concatenation and Linear Projection:** The outputs of individual heads are concatenated and then linearly transformed:

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

Where $h$ is the number of heads and $W^O$ is the parameter matrix for the output linear transformation.

5. **Feed-Forward Network**
Each attention output is passed through a position-wise feed-forward network which is the same for different positions, but different across layers:

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

Where $W_1, W_2$ are the weights and $b_1, b_2$ are the biases for the two linear transformations, and ReLU is used as the activation function.

6. **Add and Normalize**
After both the self-attention and feed-forward networks, a residual connection followed by layer normalization is applied:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

Where $\text{Sublayer}(x)$ is the operation performed by either the multi-head attention or the feed-forward network. The addition operation is the residual connection that helps in preventing the vanishing gradient problem in deep networks.

7. **Output Linear Layer and Softmax:** Finally, the output of the last layer is passed through a linear transformation followed by a softmax to predict the next token probabilities:

$$\text{logits} = xW + b$$

$$\text{probabilities} = \text{softmax}(\text{logits})$$

Where $W$ and $b$ are the weights and biases of the linear layer mapping the transformer outputs to the vocabulary size.

8. **Loss Calculation:** If training, the cross-entropy loss between the predicted probabilities and the actual next tokens is calculated to guide the training process.

These mathematical formulations encapsulate the core operations within the Transformer architecture implemented in the provided code. Each operation contributes to the overall capability of the model to understand and generate language based on the input tokens.

## 4. Experiments

### 4.1 Tools and Resources:

**Software:**

1. **Programming Language**: Python 3.10
2. **Deep Learning Framework**: PyTorch or TensorFlow
3. **Tokenizer**: Hugging Face Tokenizer or NLTK
4. **Libraries**:
   - NumPy for numerical operations
   - Matplotlib or Seaborn for data visualization and plotting
   - Scikit-learn for evaluation metrics

**Hardware**

1. **GPUs** with at 64GB of memory, *NVIDIA A100*
2. **Integrated Development Environment (IDE)**: Jupyter Notebook, Visual Studio Code for writing and executing code.
3. **Operating System**: MacOS
4. **Distributed Training**: PyTorch's DistributedDataParallel or TensorFlow's tf.distribute.Strategy for parallel training on multiple GPUs.
5. **Logging and Monitoring**: TensorBoard for tracking experiment metrics and visualizing training progress.
6. **Version Control**: Git for tracking code changes and collaborating with team members.

**Data Sets:**

1. **The Stock Exchange by Charles Duguid**: Plain Text

### 4.2 Experimental Setup:

A total of 7 experiments [7 combinations of Learning Rate and Data Split] are performed, on one text file.

- . **LR:** The current learning rate.
  **Epoch:** Indicating which epoch out of the total (e.g., 25,000 out of 50,000) the log corresponds to.
  **Trn Loss:**Training Loss: loss calculated after training on the batch or epoch.
  **Val loss:** Validation Loss: loss calculated after running the validation dataset.
  **Perp.1:** Current perplexity score on the training dataset.
  **Perp.2:** Current perplexity score on the validation dataset.
  **Split:** Training/Validation Split: percentage split between the training and validation datasets.
  **Time/Epoch:** How long each epoch is taking to run
  **Total:** The total time for Epoch that is shown in the table.
  **GPU Util.:** GPU Utilization: the percentage of time the GPU is actively engaged in processing data.

$$GPU\ Utilization(\%) = \frac{Time/Epoch \times Epoch}{Total\ time} \times 100$$

### 4.3 Performance Metrics

1. **Perplexity Score**: Evaluate the model's performance on the validation set for different hyperparameter configurations and dataset splits [70-30, 80-20]. Compare the perplexity scores to determine the optimal setup.
2. **Validation Loss**: When the loss reaches a certain epoch, it is generally the loss at convergence. It is used to evaluate the model's performance on the validation set.
3. **Training Loss**: When the loss reaches a certain epoch, it is generally the loss at convergence. It is used to evaluate the model's performance on the training set.

### 4.4 Reported Results

All the plots and a table are at the end of the report.

## 5. Conclusion

In this research paper, we present our work on building a text completion model from scratch, implementing the Transformer architecture. We conducted comparative studies of training the model on different hyperparameters, focusing on learning rates. To ensure a fair comparison, we kept the number of epochs constant at 50,000 for each training run. Our study involved training the model 7 times, varying the learning rate (1e-6,1e-5,1e-4,2e-4,3e-4) and the training-validation data split (70% - 30%, 80% - 20% respectively). We plotted the training and validation loss against the number of epochs to analyze the model's performance. Additionally, we evaluated the perplexity score, a widely used metric for assessing language models, to further compare the model's performance across different configurations. Our findings indicate that when the learning rate was set to 3e-4 and the training-validation data split was 80% - 20%, the validation loss reached a minimum of 0.083, which was the lowest among all the configurations tested within the given training time period. The perplexity score on the validation set is 1.086, which was also the lowest among all the experiments. Furthermore, this configuration also achieved the lowest perplexity score, demonstrating its superior performance in generating coherent and fluent text.

**Our results indicate that the most efficient training occurred with a learning rate of 2e-4 and a data split of 70-30 between training and validation datasets. This particular learning rate facilitated faster convergence compared to other rates tested, underscoring its effectiveness for this model configuration. Higher learning rates often led to divergence or oscillatory behavior around minima, which can introduce instability in the training process. Conversely, a learning rate set too low may result in the model failing to escape local minima effectively, hindering its ability to reach the global optimum. Thus, the selected rate of 2e-4 strikes an optimal balance, promoting steady and rapid convergence without sacrificing the stability of the learning process.**

Our research highlights the importance of hyperparameter tuning, particularly the learning rate, in optimizing the performance of text completion models based on Transformer architectures. By conducting a systematic comparison and visualizing the training dynamics, we provide valuable insights for researchers and practitioners working on similar tasks. The inclusion of perplexity scores further strengthens our findings and underscores the effectiveness of our approach in developing high-performing text completion models.

## 6. References


- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
- Bahdanau, D., Cho, K., Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
- Sutskever, I., Vinyals, O., Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27.


## 7. Source code

Here is a link to our GitHub repository: https://github.com/Shazam6565/Shazam-GPT

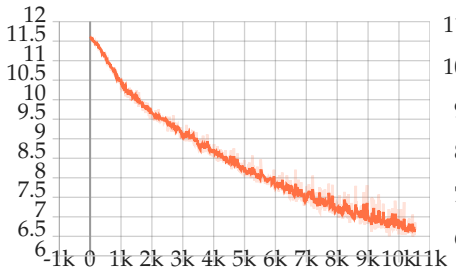


FIGURE 1. Training loss.

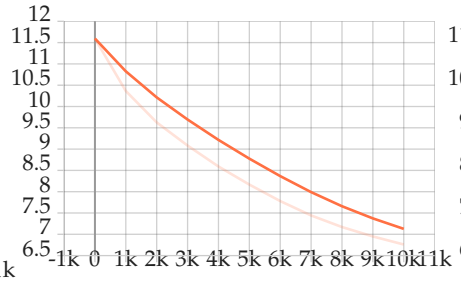


FIGURE 2. step: 10000, train loss: 6.760, lr:1e-6, split: 70-30

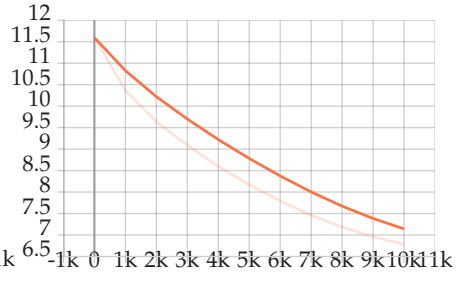


FIGURE 3. step: 10000, val loss: 6.783, lr:1e-6, split: 70-30

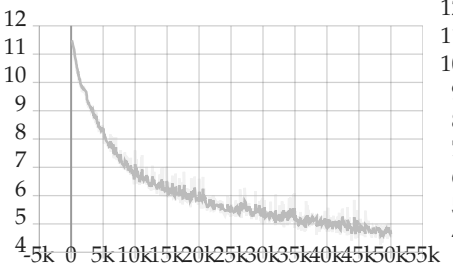


FIGURE 4. Training loss.

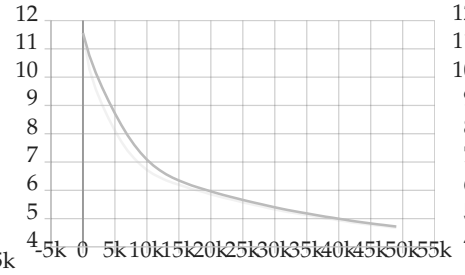


FIGURE 5. step: 50000, train loss: 5.196, lr:1e-5, split: 80-20

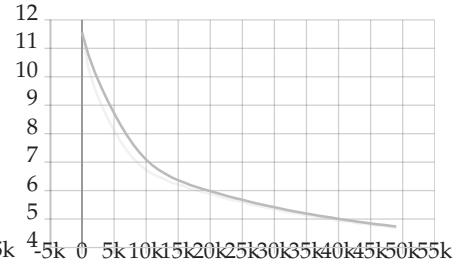


FIGURE 6. step: 50000, val loss: 5.233, lr:1e-5, split: 80-20

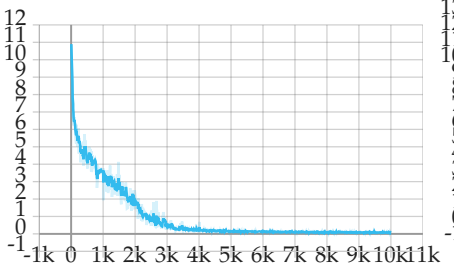


FIGURE 7. Training loss.

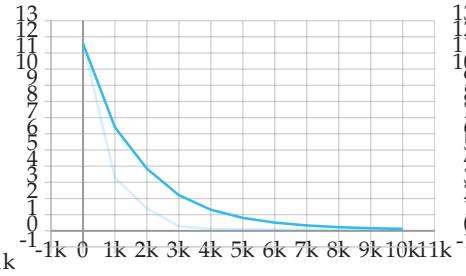


FIGURE 8. step: 10000, train loss:0.060, lr:2e-4, split: 70-30

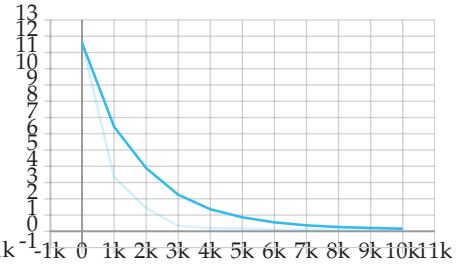


FIGURE 9. step: 10000, val loss: 0.092, lr:2e-4, split: 70-30

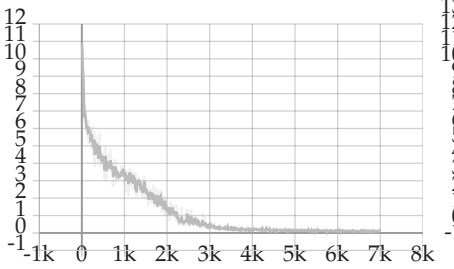


FIGURE 10. Training loss.

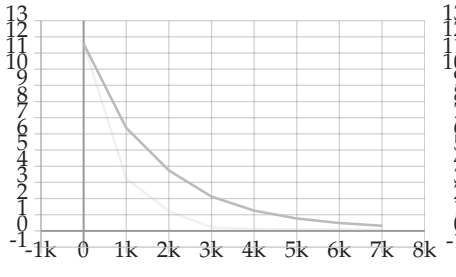


FIGURE 11. step: 7000, train loss:0.066, lr:3e-4, split: 80-20

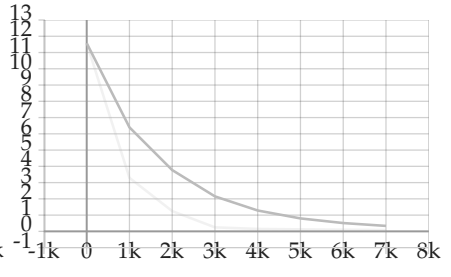


FIGURE 12. step: 7000, val loss: 0.083, lr:3e-4, split: 80-20

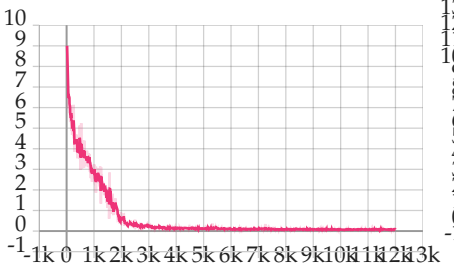

FIGURE 13. Training loss.

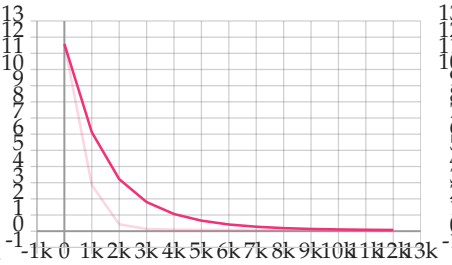

FIGURE 14. step: 12000, train loss:0.058, lr:3e-4, split: 70-30

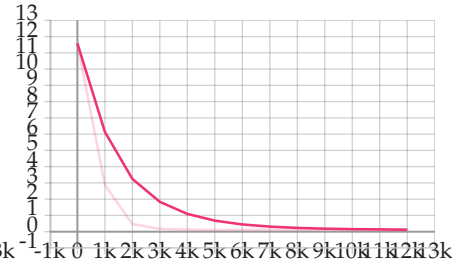

FIGURE 15. step: 12000, val loss: 0.094, lr:3e-4, split: 70-30

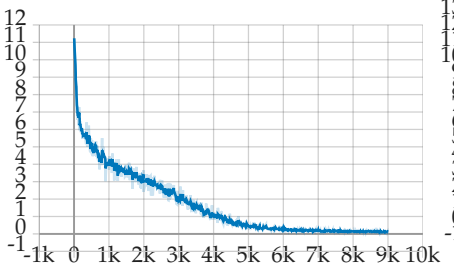

FIGURE 16. Training loss.

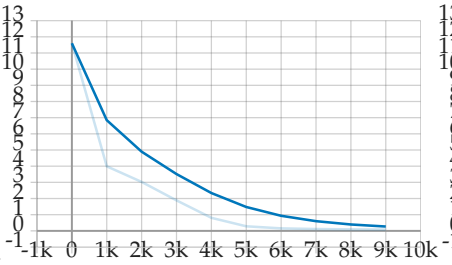

FIGURE 17. step: 9000, train loss:0.058, lr:1e-4, split: 80-20

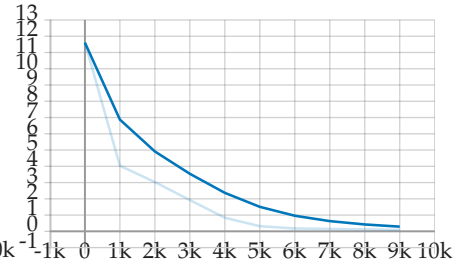

FIGURE 18. step: 9000, val loss: 0.094, lr:1e-4, split: 80-20

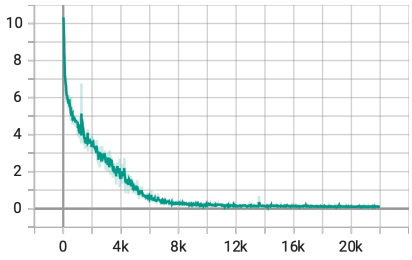

FIGURE 19. Training loss.

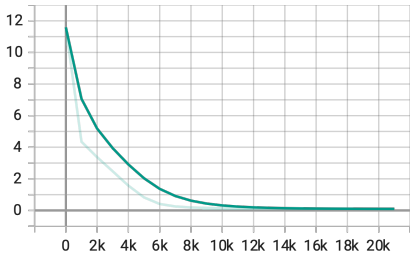

FIGURE 20. step: 22000, train loss:0.096, lr:1e-4, split: 70-30

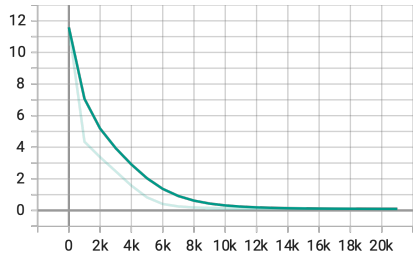

FIGURE 21. step: 22000, val loss: 0.124, lr:1e-4, split: 70-30

TABLE 1. Experiment Log for Various Learning Rates

| LR | Epoch | Trn Loss | Val Loss | Perp.1 | Perp.2 | Split | Time/Epoch | Total | GPU Util. |
|---|---|---|---|---|---|---|---|---|---|
| 1e-6 | 10000 | 6.760 | 6.783 | 862.901 | 883.040 | 70%/30% | 0.09 secs | 26.22 mins | 57.2% |
| 1e-5 | 50000 | 5.196 | 5.233 | 870.217 | 900.262 | 80%/20% | 0.09 secs | 128.4 mins | 58.4% |
| 1e-4 | 22000 | 0.094 | 0.124 | 1.101 | 1.132 | 70%/30% | 0.09 secs | 58.40 mins | 56.5% |
| 1e-4 | 9000 | 0.058 | 0.094 | 1.899 | 1.986 | 80%/20% | 0.09 secs | 23.17 mins | 58.3% |
| 2e-4 | 10000 | 0.060 | 0.092 | 1.062 | 1.097 | 70%/30% | 0.09 secs | 26.22 mins | 57.2% |
| 3e-4 | 7000 | 0.066 | 0.083 | 1.068 | 1.086 | 80%/20% | 0.09 secs | 19.44 mins | 54.0% |
| 3e-4 | 12000 | 0.058 | 0.094 | 1.268 | 1.386 | 70%/30% | 0.09 secs | 38.38 mins | 46.9% |