

**National University of Singapore
School of Computing
IT5005 Artificial Intelligence**

Transformers (GPT)

This tutorial focuses on implementing a decoder-only transformer architecture, specifically a Generative Pre-trained Transformer (GPT). The core transformer implementation is provided in the *transformer.py* file, with most components already implemented except for the weight initialization methods.

Dataset and Training Setup: The GPT model will be trained on the Penn Treebank (PTB) dataset. Helper functions for preparing GPT-compatible training data are provided in the *utils_gpt.py* file. The complete training pipeline, including model training and weight saving procedures, is implemented in the *GPT_trainer.ipynb* notebook.

Model Deployment and Inference After training: The learned parameters will be saved and subsequently loaded into a separate GPT instance for inference. The code for creating this inference model and loading the pre-trained weights is contained within the *GPT_generator.ipynb* file.

Tasks

This tutorial involves two main components:

Implementation Task: Write the weight initialization code for the transformer architecture

Conceptual Understanding: Answer questions to deepen your understanding of the model training and inference processes.

The list of tasks to be done is presented below.

1. Define weight matrices in MHA and FFN blocks (fill the missing blocks in 'transformer.py'). Note that the implementation of transformer block follows the original implementation in "Attention is all you need."
2. Train the above model and use the pretrained model to generate text. Use *GPT_trainer.ipynb* file for training. The pretrained model and vocabulary would be stored in the same folder. Then use *GPT_generator.ipynb* to generate text for a given prompt.
3. **LayerNorm in a Transformer Feed-Forward Block**

Let y be the output of Post-LN and it can be written as

$$y = \text{LN}(x + \text{FFN}(x)). \quad (1)$$

Assume three input tokens with the corresponding embedding vectors $\mathbf{x}_i, i \in \{1, 2, 3\}$:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} 5 \\ 2 \\ 6 \end{bmatrix} \quad (2)$$

FFN is a two-layer MLP with ReLU:

$$\text{FFN}(x) = W_2^T \text{ReLU}(W_1^T x + b_1) + b_2, \quad (3)$$

with

$$W_1 = I_3, \quad b_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0 \end{bmatrix}, \quad W_2 = I_3, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \quad (4)$$

LayerNorm parameters (per feature):

$$\gamma = \begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix}, \quad \beta = \begin{bmatrix} 0.5 \\ -0.5 \\ 1.0 \end{bmatrix} \quad (5)$$

Compute the output of Layer Normalization during training and inference.

4. Why is LayerNorm applied before attention/FFN in modern transformers (Pre-LN) vs after (Post-LN)? How does this affect gradient flow?