VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

B.Tech. Winter Semester 2024-25
School Of Computer Science and Engineering
(SCOPE)

Review of Generative Techniques in Deep Learning

Cryptography and Network Security Lab

Apurva Mishra: 22BCE2791
Date: 3 March, 2025

Contents

1 Intro

Large Language Models have been instrumental in recent advances in the field of deep learning. This is in large part fueled through the introduction of new transformer architecture for sequence modelling [1].

Using this transformer architecture, two major paradigms have become popular: Auto-regressive and diffusion based.

2 Transformer

2.1 Vanilla Transformer

The vanilla transformer [1] is a sequence to sequence model, composed of a **encoder**, **stack of layers** and **decoder**.

Encoder: The encoder is responsible for converting the input sequence into representation in a continuous latent space. It contains multi-head self attention mechanism and position-wise feed-forward network (FFN). Finally each block is normalized [2].

Algorithm: Token Embedding
Input: $v \in \cong [N_V]$, a token ID.
Output: $e \in R^{d_e}$, the vector representation of the token.
Parameters: $W_e \in R^{d_e * N_V}$, the token embedding matrix.
return $e = W_e[:, v]$

Table 1: Algorithm: Represetnting input tokens into latent space

Decoder: The decoder is responsible for converting the new transformation of the input sequence in the latent space through the stack of layers back into text. The decoder is similar to encoder in design. However it also includes a additional multi-head attention layer between the self attention layer and the feed-forward network (FFN) layer. Again this is also normalized.

Algorithm: Un-Embedding
Input: $e \in R^{d_e}$, a token encoding.
Output: $p \in \Delta(V)$, a probability distribution over the vocabulary
Parameters: $W_e \in R^{d_e * N_V}$, the unembedding matrix.
return $\text{softmax}(W_u e)$

Table 2: Algorithm: Converting vector from latent space to vocablury

Layer Normalization For all: encoder, sub-layers and decoder layer normalization is performed for the final output. This is essential as transformer have shown to become unstable due to changes in activations from variying dataset. [3] Here layer normalization ensure that each output is normalized to a consistent distribution. This ensures a consistent distribution.

Algorithm: Layer Normalization
Input: $e \in R^{d_e}$, neural network activations.
Output: $e \in R^{d_e}$, normalized activations.
Parameters: $y, \beta R^{d_e}$, element-wise scale and offset
1. $m \leftarrow \sum_{i=1}^{d_e} \frac{e[i]}{d_e}$
2. $v \leftarrow \frac{\sum_{i=1}^{d_e} \left(\frac{e[i]}{d_e} - m\right)^2}{d_e}$
return $e = \frac{e-m}{\sqrt{v}} \odot y + \beta$, where $\odot$ denotes element-wise multiplication.

Table 3: Algorithm: Algorithm for layer normalization

2.2 Attention

The state of the art performance of vanilla transformer is obtained using the attention mechanism. It works based on three inputs: *Query-Key-Value (QKV)*. The function can be represented as:

$$\text{Attention}(QKV) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V \tag{1}$$

Algorithm: Attention

Input: $v \in \approx [N_V]$, a token ID.
Output: $e \in R^{d_e}$, the vector representation of the token.
Parameters: $W_e \in R^{d_e * N_V}$, the token embedding matrix.

return $e = W_e[:, v]$

Input: $X \in R^{d_x * l_x}, Z \in R^{d_z * l_z}$, vector representation of primary and context sequence
Output: $V \in R^{d_{\text{out}} * l_x}$,, updatead represetations of tokens in X, folding in information from token **Z**

Parameter: W_{qkv} consist of
$W_q \in R^{d_{\text{attn}} * d_x}, b_q \in R^{d_{\text{attn}}}$
$W_k \in R^{d_{\text{attn}} * d_x}, b_k \in R^{d_{\text{attn}}}$
$W_v \in R^{d_{\text{attn}} * d_x}, b_v \in R^{d_{\text{attn}}}$
Hyperparameters: Mask $\in \{0, 1\}^{l_z * l_x}$
1. $Q \leftarrow W_q X + b_q$
2. $K \leftarrow W_k X + b_k$
3. $Q \leftarrow W_v X + b_v$
4. $S \leftarrow K^T Q$

return $V \cdot \text{softmax}\left(\frac{S}{\sqrt{d_{\text{attn}}}}\right)$

Table 4: Algorithm: Attention

2.3 Autoregressive

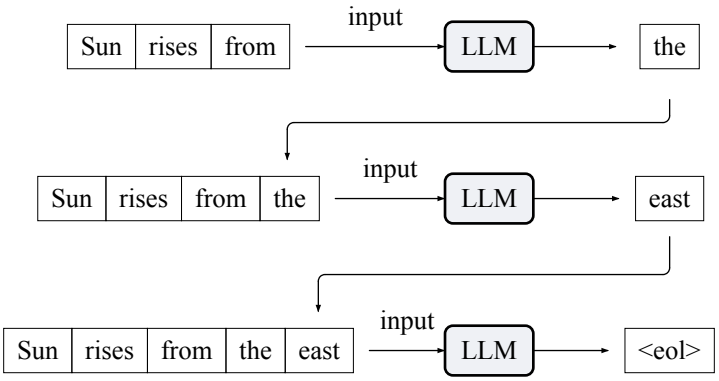

Figure 1: Example for an autogressive LLM policy. The LLM is trained to predict the next most probably word. Then this is appended to the original string and the process continues until the end of line special token is received.

For an input string $\{s_1..s_n\}$ and output string $\{s_{n+1}..s_{n+k+1}\}$, a model works on the policy of:

$$\begin{gathered} p(s_{n+1}..s_{n+k+1}|s_1..s_n) \\ p(s_{n+1}|\ s_1..s_n)p(s_{n+2}|\ s_1..s_n+1)..p(s_{n+k}|\ s_1..s_{n+k-1}) \end{gathered} \tag{2}$$

Thus each successive string is sampled based all the previous set of strings in sequence. This policy can be generalizsed to simply predict the next word in the string using the chain rule and then repeated.

2.4 Context Length

The total length of the input tokens is called the context length. The context length can be considered as the short term memory of the llm and the conditional tokens against which the llm predicts the next most likely word.

Therefore a high context length is topic of much discussion. Theoretically the length of input string can be infinite, however in practice its limited due to several factors. For example LLama 3.3 [4] family of models have an impressive context length of $128k$ tokens.

However the effective context length is often much shorter at just

TODO

Find the effective context length.

2.4.1 Size of training dataset

The documents in training dataset have a finite length. Most documents in popular datasets often do not exceed 10k tokens. LLM(s) trained on this context length often do not generalize over longer contexts. Increasing the context length above this during inference causes the performance of the model to degrade rapidly. [5], [6]

Dataset Name	Average Length	Data Access
SumSurvey	>12,000 tokens	SumSurvey Dataset
NarrativeXL	>50,000 words	NarrativeXL Dataset
LongBench	10,000 tokens	LongBench Dataset
HiPool	4,034 tokens	HiPool Benchmark
Databricks DocsQA	2,856 tokens	Databricks Blog

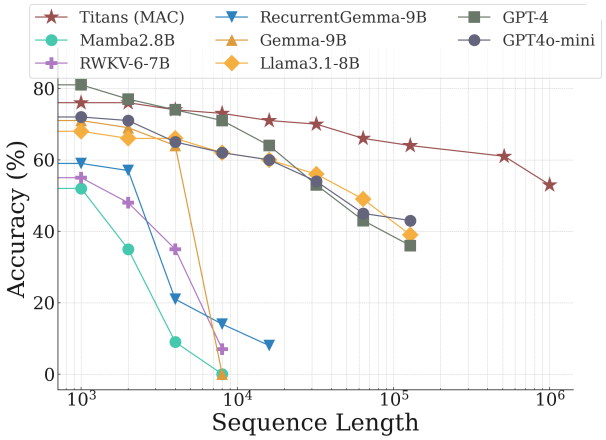

Table 5: Left: Average sizes of documents in few popular LLM training datasets
Right: Accuracy of popular LLM with increasing context length. Signifying effective context length. [7]

2.4.2 Quadratic complexity

Transformer based attention architecture requires quadratic time complexity over the context length. Specifically QK^T multiplication requires $O(n^2)$ computation and memory. This is the vanilla attention and here the output token can attend to all the input tokens. This can be visualized using the attention mask.[8]

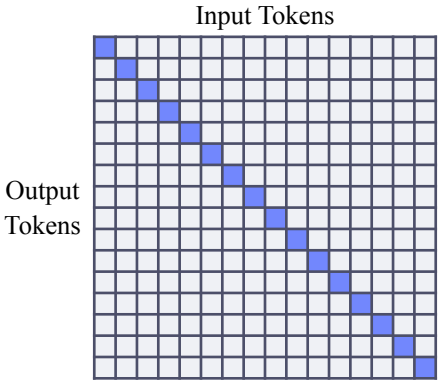

Figure 2: Attention Mask of Vanilla Transformer

2.5 Representation of Data

Let the vocablury of Model be: V. The input text/data needs to be represented in this vocablury on which the model would be trained. Empirical experiments have shown that correct scaling of vocablury size in relation to parameter count of the model is very important for optimal performance. [9]

The process of representing input data as a sequence of vocablury elements is called **tokenization**. Tokenization can be achieved in several manner. For a piece of text: "Sun rises from the east", it can be tokenized as:

1. **Character-level Tokenization**:

$$V \in [a, .., z] \tag{3}$$

   Here each vocablury token corresponds to a alphabet: ['S', 'u', ..'s', 't']. These generalize well, however these are too small representation and very expensive for large context lengths due to quadratic complexity. In addition a big vocablury causes the embedding matrix to huge causing increasing memory complexity.

2. **Word-level Tokenization**:

$$V \in \text{Words} \tag{4}$$

Here each vocablury token corresponds to a word: [‘Sun’, ‘rises’, ‘from’, ‘the’, ‘east’]. These require very large vocablury and do not generalize well.

3. **Subword Tokenization**:

$$V \in \text{Commonly Occurring Sub Words} \tag{5}$$

Here each vocablury token corresponds to a commonly occurring word segments: [‘ris’, ses’, ‘the’ ..] These offer the best middle ground between vocablury size and generalization. Notably common words like *‘the’* are tokenized as it is.

4. **Byte-Pair Encoding (BPE)**: This is the most popular type of sub-word tokenization scheme. It was introduced in the paper: Neural Machine Translation of Rare Words with Subword Units [10]

3 Diffusion

Diffusion models are based on non-equilibrium thermodynamics. Here we iteratively add noise to the original data (forward diffusion process) and then learn to iteratively get the data back from noise (backward diffusion process).

3.1 Forward Diffusion Process

3.2 Backward Diffusion Process

Bibliography

[1] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[3] L. Berlyand, P.-E. Jabin, and C. A. Safsten, “Stability for the training of deep neural networks and other classifiers,” *Mathematical Models and Methods in Applied Sciences*, vol. 31, no. 11, pp. 2345–2390, 2021.

[4] A. Dubey *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.

[5] C. An *et al.*, “Why Does the Effective Context Length of LLMs Fall Short?,” *arXiv preprint arXiv:2410.18745*, 2024.

[6] M. Ulčar and M. Robnik-Šikonja, “Training dataset and dictionary sizes matter in Bert models: the case of Baltic languages,” in *International Conference on Analysis of Images, Social Networks and Texts*, 2021, pp. 162–172.

[7] A. Behrouz, P. Zhong, and V. Mirrokni, “Titans: Learning to memorize at test time,” *arXiv preprint arXiv:2501.00663*, 2024.

[8] Q. Fournier, G. M. Caron, and D. Aloise, “A practical survey on faster and lighter transformers,” *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–40, 2023.

[9] S. Takase, R. Ri, S. Kiyono, and T. Kato, “Large Vocabulary Size Improves Large Language Models,” *arXiv preprint arXiv:2406.16508*, 2024.

[10] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *arXiv preprint arXiv:1508.07909*, 2015.

[11] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning,” *arXiv preprint arXiv:2307.08691*, 2023.

[12] T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen, “Long-context llms struggle with long in-context learning,” *arXiv preprint arXiv:2404.02060*, 2024.