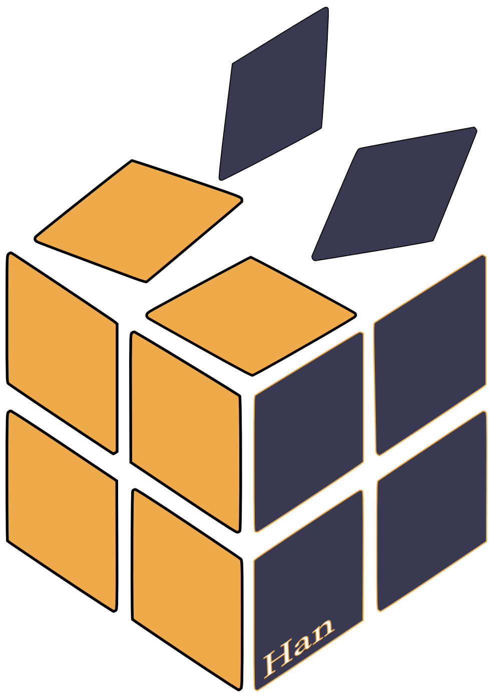
Deep Learning System Design



Engineering and Service Architectures

Han Cheol Moon
School of Computer Science and Engineering
Nanyang Technological University

Singapore
hancheol001@e.ntu.edu.sg

February 2, 2025

Contents

Part I

Introduction

Chapter 1

Introduction

1.1 Complexity of Matrix Multiplication

Matrix multiplication is a fundamental operation in many computational tasks, including neural networks. The complexity of multiplying two matrices depends on their dimensions. Let's dive into the specifics.

- Let A be a matrix of size $m \times k$.
- Let B be a matrix of size $k \times n$.
- The result C will be a matrix of size $m \times n$.

Standard Matrix Multiplication: For each element c_{ij} in the resulting matrix C:

$$c_{ij} = \sum_{l=1}^{k} a_{il} \cdot b_{lj}$$

This involves:

- Multiplications: k multiplications for each element c_{ij}.
- Additions: $k - 1$ additions for each element c_{ij}.

Complexity

- The total number of elements in C is $m \times n$.
- Therefore, the total number of multiplications is $m \times n \times k$.
- The total number of additions is $m \times n \times (k - 1)$.

Thus, the total complexity is $O(m \times n \times k)$.

Even though there are several advanced methods, the standard $O(m \times n \times k)$ complexity is often used in practice, due to the simplicity and efficiency of implementation on modern hardware. Optimized libraries (like BLAS, cuBLAS for GPUs) leverage hardware-specific optimizations to improve practical performance.

1.1.1 Complexity in Neural Networks

In the context of neural networks:

- Input Matrices: Weight matrices and input feature vectors.
- Typical Sizes:
  - Weight matrix: $d \times d_{in}$ for RNNs, $d \times d$ for Transformers.
  - Input/Output vectors: Usually batch-processed, leading to sizes like $batch\_size \times sequence\_length \times feature\_size$.

Part II

Transformers

Chapter 2

Attention

2.1 Multi-Head Attention

2.2 KV Cashing

2.3 Flash Attention

Chapter 3

Positional Embeddings

Rather than focusing on a token's absolute position in a sentence, relative positional embeddings concentrate on the distances between pairs of tokens. This method doesn't add a position vector to the word vector directly. Instead, it alters the attention mechanism to incorporate relative positional information.

3.0.1 Rotary Positional Embeddings

RoPE represents a novel approach in encoding positional information. Traditional methods, either absolute or relative, come with their limitations. Absolute positional embeddings assign a unique vector to each position, which though straightforward, doesn't scale well and fails to capture relative positions effectively. Relative embeddings, on the other hand, focus on the distance between tokens, enhancing the model's understanding of token relationships but complicating the model architecture.

RoPE ingeniously combines the strengths of both. It encodes positional information in a way that allows the model to understand both the absolute position of tokens and their relative distances. This is achieved through a rotational mechanism, where each position in the sequence is represented by a rotation in the embedding space. The elegance of RoPE lies in its simplicity and efficiency, enabling models to better grasp the nuances of language syntax and semantics.

RoPE introduces a novel concept. Instead of adding a positional vector, it applies a rotation to the word vector. Imagine a two-dimensional word vector for "dog." To encode its position in a sentence, RoPE rotates this vector. The angle of rotation (θ) is proportional to the word's position in the sentence. For instance, the vector is rotated by θ for the first position, 2θ for the second, and so on. This approach has several benefits:

Chapter 4

Tokenization

Chapter 5

Model Compression