Understanding MiniLM for Query Routing: A Mathematical and Intuitive Guide

Saurav Chowdhury
sauravchowdhury16.sc@gmail.com

1 Introduction

In this article, we explore how MiniLM can be used for query classification—specifically, to route user queries to either a lightweight Retrieval-Augmentative Generation (RAG) system or a deeper Large Language Model (LLM) for complex reasoning. We focus on both the mathematical foundations and the intuition behind each component, based on hands-on discussion and analysis.

2 Problem Statement

Given a user query, we aim to classify it into one of two categories:

- Class 0: Answerable via LLM 1
- Class 1: Answerable by LLM 2

We use MiniLM as a lightweight encoder, paired with a simple classifier (Logistic Regression model) trained on labeled query-answer pairs. Figure 1 shows a comprehensive architecture.

3 Step-by-Step Breakdown

3.1 Tokenization and Embedding Lookup

Each user input query is broken into tokens using a tokenizer. For example, consider the query:

"Why is electricity bill high in summer?"

This is tokenized into, say, 10 subword tokens:

$$\text{Tokens} = [\text{“[CLS]”}, \text{“why”}, \text{“is”}, \text{“electricity”}, \text{“bill”}, \text{“high”}, \text{“in”}, \text{“summer”}, \text{“?”}, \text{“[SEP]”}]$$

Each token is mapped to a vector using a pre-trained vocabulary of size 30,000 (MiniLM's vocabulary). MiniLM is a pretrained BERT model with 30,000 words, where each word is represented by a vector of dimension 384. This forms the embedding matrix:

$$[\text{vocab\_size}, \text{hidden\_size}] = [30,000, 384]$$

The embedding lookup is performed as:

$$\text{input\_embeddings} = \text{embedding\_matrix}[\text{token\_ids}]$$

This yields a shape of:

$$[\text{sequence\_length}, \text{hidden\_size}] = [10, 384]$$

Only the tokens present in the query are looked up, so there is no need to process the entire 30,000-word vocabulary during inference. The resulting embeddings are:

Token	Vector (dim=384)
[CLS]	[0.12, -0.38, …, 0.04]
why	[0.09, 0.18, …, -0.02]
is	[…]
⋮	⋮
?	[…]

3.2 Positional Embeddings

To maintain word order, a positional embedding is added to each token embedding:

$$x_i = \text{embedding}_i + \text{position\_embedding}_i$$

This results in a sequence of vectors that encode both the meaning of the tokens and their positions in the sequence.

3.3 Transformer Encoding (MiniLM)

Each vector is processed through multiple Transformer layers, which include:

- Multi-head self-attention
- Feedforward layers

The output is a contextualized representation for each token:

$$[h_1, h_2, \ldots, h_{10}] \quad \text{with shape: } (10, 384)$$

3.4 Pooling: From Token Vectors to One Vector

Since the classifier requires a single vector per query, we pool the token vectors. The options are:

- **[CLS] token pooling**: Take the first token's vector.
- **Max pooling**: Take the maximum value over each of the 384 dimensions across all tokens.
- **Mean pooling**: Average the vectors over all tokens.

For example, max pooling transforms the shape from $(10, 384)$ to $(1, 384)$ by selecting the maximum value in each dimension. This pooled vector captures the entire query.

3.5 Classifier

The pooled vector is passed to a linear classifier followed by a sigmoid activation:

$$z \to \text{sigmoid}(Wz + b)$$

This outputs a probability between 0 and 1:

- Output close to 0: Route to RAG
- Output close to 1: Route to LLM

3.6 Training

We train the classifier using binary cross-entropy loss:

$$L = -\left[y \log(p) + (1 - y) \log(1 - p)\right]$$

where:

- y is the true label (0 or 1)

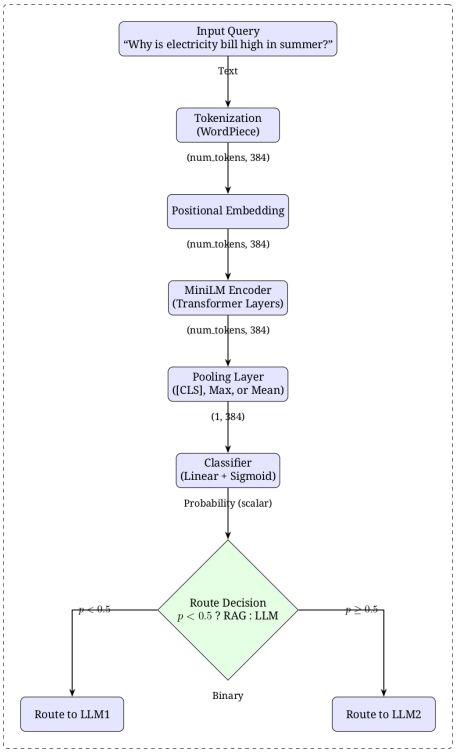

Figure 1: Architecture

- p is the predicted probability

Only the embeddings for tokens in the training queries are used, so there is no need to load the full vocabulary.

4 Final Intuition

- MiniLM generates a vector per token, but the classifier requires a single vector. Pooling bridges this gap.
- The [CLS] token is specially trained to represent the entire sentence, making it suitable for classification tasks.
- Pooling is a crucial step to convert multiple token vectors into a single sentence vector.
- The classifier learns to distinguish between classes based on the pooled embeddings.
- The class label is not part of the input—it is the target during training.