Advanced Natural Language Processing

TRANSFORMERS FOR MACHINE TRANSLATIONS

Hardik Sharma

Introduction

In this assignment we implement the Transformer architecture proposed in the 2017 paper **Attention Is All You Need**. We take a deep dive into the architecture, explore different hyperparameters and train it from scratch for the task of English to French Translation.

Theory Questions

Question 1

What is the purpose of self-attention, and how does it facilitate capturing dependencies in sequences?

Self-Attention Mechanism in Transformers:

The purpose of self-attention in neural networks, particularly in transformers, is to enable the model to dynamically focus on different parts of an input sequence when generating the output, allowing it to capture relationships between words regardless of their positions in the sequence.

How Self-Attention Works

1. **Contextual Focus:** Each word (or token) in a sentence can attend to all other words in the sequence, learning which words are important for its own understanding. For example, in the sentence "The cat sat on the mat," the word *sat* can attend to the words *cat* and *mat* to understand who is performing the action and where the action is taking place.

2. **Weighting Important Words:** Self-attention assigns a score (or weight) to each word relative to the target word. Words that are more relevant to the target word receive higher attention scores. This mechanism allows the model to prioritize important information in the sequence.

3. **Capturing Dependencies:** Self-attention helps the model capture both *short-range dependencies* (like nearby words) and *long-range dependencies* (words farther apart in the sequence). For example, in the sentence "The boy who went to the store bought milk," self-attention allows the model to recognize that *boy* is the subject of *bought*, even though several words separate them.

Question 2

Why do transformers use positional encodings in addition to word embed- dings? Explain how positional encodings are incorporated into the trans- former architecture. Briefly describe recent advances in various types of po- sitional encodings used for transformers and how they differ from traditional sinusoidal positional encodings.

Transformers use **positional encodings** in addition to word embeddings to maintain the sequential information of input data. Unlike recurrent neural networks (RNNs), which inherently process data in sequence, transformers operate on the entire input simultaneously. This parallel processing can lead to a loss of information regarding the order of words. Positional encodings address this issue by providing a unique representation for each position in the input sequence, allowing the model to learn relationships based on word order.

Incorporation of Positional Encodings into Transformer Architecture

Positional encodings are incorporated into the transformer architecture by adding them directly to the input embeddings. Specifically, for each word embedding $\mathbf{E}_i$ at position i, the positional encoding $\mathbf{PE}_i$ is computed and added as follows:

$$\mathbf{Z}_i = \mathbf{E}_i + \mathbf{PE}_i$$

where $\mathbf{Z}_i$ is the resultant embedding that includes positional information. The positional encoding can be generated using various methods, with the original transformer model proposing sinusoidal functions defined as:

$$\mathbf{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$\mathbf{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Here, pos is the position and i is the dimension index, while d_{model} is the dimensionality of the embeddings.

Recent Advances in Positional Encodings

Recent research has explored various types of positional encodings beyond traditional sinusoidal methods. Some notable advancements include:

- **Absolute Position Embedding (APE)**: This method assigns a unique vector to each position based on its absolute index within the sequence.
- **Relative Positional Encoding (RPE)**: Instead of encoding absolute positions, RPE captures the relative distances between tokens, allowing for more flexible handling of input sequences. This approach has shown benefits in tasks where understanding relationships between tokens matters more than their absolute positions.
- **ALiBi (Attention with Linear Biases)**: A recent method that modifies attention scores based on relative distances without requiring additional parameters, thus improving efficiency.
- **Rotary Positional Embedding**: This technique incorporates rotary transformations into positional encodings, allowing for better handling of longer sequences and enhancing model performance.
- **No Positional Encoding (NoPE)**: Some studies suggest that models can perform well without explicit positional encodings by learning to capture positional information through training dynamics alone. NoPE has been shown to outperform traditional methods in certain tasks by effectively representing both absolute and relative positions through learned attention patterns.

These advancements differ from traditional sinusoidal positional encodings primarily in their focus on either relative positioning or efficiency in computation, addressing limitations observed in earlier models and adapting to specific tasks and data types.

Implementation and Training

The model is implemented in python and the code is open source. Pytorch has been used as the computational framework/backend for this implementation. The model, being trained on very less data than the original paper, does not come close to replicating the performance of the original paper, but nonetheless provides deep insight into the working of the transformer.

The model parameters are as follows :

1. Embedding Size : 256
2. Encoder Layer :
    - 6 layers
    - 8 Attention Heads
    - 10% dropout
    - Projection to $2x$ the embedding size

- Single Perceptron at each block

3. Decoder Layer :
    - 6 layers
    - 8 Attention Heads
    - 10% dropout
    - Projection to $2x$ the embedding size
    - Single Perceptron at each block
4. Final projection to vocab size

Each epoch in the model takes approximately 4 minutes to run, with a batch size of 4, using $5800MB$ of VRAM at around 88% average GPU utilization for a laptop grade **RTX** 3060.