Assignment 2: Transformers from Scratch

Course: Advanced Natural Language Processing

Deadline: 30 September | 23:59

General Instructions

1. The assignment must be implemented in Python.

2. The assignment must be done using PyTorch. Usage of other frameworks will not be accepted.

3. Submitted assignment must be your original work. Please do not copy any part from any source including your friends, seniors, and/or the internet. If any such attempt is caught, then serious actions including an F grade in the course is possible.

4. A single .zip file needs to be uploaded to the Moodle Course Portal.

5. Your grade will depend on the correctness of answers and output. In addition, due consideration will be given to the clarity and details of your answers and the legibility and structure of your code.

6. Please start early since no extension to the announced deadline would be possible.

7. **Do not use the readily available torch modules for positional encoding, encoder stack, decoder stack, multi-head attention or any other part of the transformer architecture. You are expected to implement them from scratch**

1 Transformers for Machine Translation

Machine translation, the task of automatically translating text from one language to another, has significantly evolved with the advent of neural networkbased approaches. Traditional machine translation systems, such as rule-based or statistical approaches, faced challenges in capturing complex linguistic patterns and contextual dependencies. The introduction of neural networks revolutionized this field by leveraging their ability to learn intricate relationships directly from data. Sequence to sequence models, initially introduced for various natural language processing (NLP) tasks, were adapted for machine translation due to their inherent suitability for handling sequential data.

The transformer architecture, introduced in the paper ”Attention is All You Need” by Vaswani et al. in 2017, emerged as a groundbreaking innovation in sequence to sequence modelling. Transformers rely heavily on self-attention mechanisms, enabling them to weigh the importance of each word in the input sequence dynamically. This attention mechanism is what allows the model to capture long-range dependencies and contextual information effectively. The attention mechanism aims to assign weights to different positions in the input sequence, and this allows the model to focus on relevant parts during the encoding and decoding processes. By attending to specific words or tokens based on their importance scores, the model creates a context-aware representation of the input sequence, which is crucial for high-quality translations.

Transformers are trained using parallel datasets, where source sentences and their corresponding target translations are aligned. The model learns to minimize a loss function, often based on maximum likelihood estimation or other variants, to optimize the translation performance.

In this assignment, we will delve into the practical implementation of a simple transformer for machine translation, understanding its components and applying the theoretical concepts discussed above.

2 Theory Questions

2.1 Question 1

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

2.2 Question 2

Why do transformers use positional encodings in addition to word embeddings? Explain how positional encodings are incorporated into the transformer architecture. Briefly describe recent advances in various types of positional encodings used for transformers and how they differ from traditional sinusoidal positional encodings.

3 Implementation and training

3.1 Model Architecture

Design and present the architecture of the transformer model from scratch for the task of machine translation. This should include all components of the original transformer architecture. You are not allowed to use any readily available Torch modules related to transformers, such as the decoder module, encoder module, etc.

3.2 Model Training

You must train your transformer for the machine translation task. You are provided with a parallel corpus of English and French sentences for training on this task. The corpus includes train, dev, and test splits that should be used in a similar manner.

3.3 Hyperparameter Tuning

Test the performance of the model by varying these hyperparameters:

- Number of layers in the encoder/decoder stack
- Number of attention heads

- The dimensions of the word embeddings
- Dropout

Report the model's performance on the translation task for at least three different combinations of these hyperparameters. Include loss graphs, translation-related metrics (e.g., BLEU, ROUGE), and any other relevant performance metrics.

3.4 Analysis

Write up a thorough analysis of the performance of your transformer model. Evaluate the quality of the model's translations using the BLEU metric (you are also required to submit the BLEU scores for all sentences in the test set). Describe the hyperparameters you chose for the model and their significance. Provide explanations for the performance differences across different hyperparameter configurations. Additionally, plot the loss curves to give insights into the training process

4 Corpus

The data can be downloaded from here. The dataset is a subset of the dataset for the English-French translation task in IWSLT 2016. It contains the files for train, dev, and test splits as follows:

1. train.`[en-fr]`: 30,000 lines each
2. dev.`[en-fr]`: 887 lines each
3. test.`[en-fr]`: 1,305 lines each

Submission Format

Zip the following files into one archive and submit it through the Moodle course portal. The filename should be `<roll number>_assignment2.zip`, for example, `2021114005_assignment2.zip`.

- **Source Code**

    - `train.py`: Main code to train the transformer for machine translation. Save the trained model to `transformer.pt`.
    - `test.py`: Code to test the pre-trained model on the test set. Load the model saved during training.
    - `encoder.py`: Implement the encoder class and relevant classes/functions.
    - `decoder.py`: Implement the decoder class and relevant classes/functions.
    - `utils.py`: Include any other helper functions or classes.
  - **Pretrained Models**
    - `transformer.pt`: Saved transformer model.
  - **Text Files**
    - `testbleu.txt`: Scores for all sentences in the test set, printed in the format `<sentence> <score>`.
  - **Report (PDF)**
    - Answers to the theory questions
    - Hyperparameters used to train the model(s).
    - Corresponding graphs and evaluation metrics.
    - Your analysis of the results.
  - **README**
    - Instructions on how to execute the files, load the pretrained model, implementation assumptions, etc.
    - Feel free to save the pre-trained transformer model to onedrive. Make sure you include the link here

Grading

Evaluation will be individual and based on your viva, report, and code review. During your evaluation, you will be expected to walk us through your code and explain your results. You will be graded based on the correctness of your code, accuracy of your results, and the quality of the code.

Theory Questions: 10 marks

Transformer Architecture: 30 marks

Transformer Training: 20 marks

Hyperparameter Tuning: 15 marks

Analysis: 10 marks

Viva during Evaluation: 15 marks

Resources

1. Attention is All You Need
2. The Illustrated Transformer
3. You can also refer to other resources, including lecture slides!