# SPEECH-TRANSFORMER: A NO-RECURRENCE SEQUENCE-TO-SEQUENCE MODEL FOR SPEECH RECOGNITION

**Guilherme Augusto Rocha de Figueiredo**
Undergraduate Student of Computer Science
Universidade Federal de Viçosa
guilherme.figueiredo@ufv.br

## Abstract

This project involves the implementation of the Transformer architecture as presented in the paper titled "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition." The model is used for automatic speech recognition (ASR). An implementation of this Transformer using the TensorFlow library in Python was found on GitHub. This project utilizes that implementation as the base code while translating the architecture into the PyTorch library. Unlike the original paper, this project employs the LibriSpeech dataset for training and evaluation.

## 1 Introduction

The Transformer architecture, first introduced by Vaswani et al. in 2017 [1], revolutionized the field of machine translation by achieving state-of-the-art performance. Since then, Transformers have gained significant popularity and have been successfully applied to a wide range of tasks across various domains.

In the context of speech recognition, Dong et al. [2] were the first to employ the Transformer model, marking a shift from traditional recurrent neural networks to self-attentive architectures for Automatic Speech Recognition (ASR) tasks. This project aims to replicate their work by implementing the Speech-Transformer model using the PyTorch library. One of the key innovations in the original paper is the introduction of a 2D-Attention mechanism, which simultaneously attends to both the time and frequency dimensions of the 2-dimensional speech input, thereby generating more expressive representations. This approach enabled the model to achieve competitive Word Error Rate (WER) metrics while running significantly faster than the published results of recurrent sequence-to-sequence models at the time.

Although the authors of the Speech-Transformer did not provide a direct implementation, this project utilizes an open-source implementation from Xingchen Song, available on GitHub[1]. This implementation, originally developed in Python using the TensorFlow library, serves as the foundation of the work and has been translated to PyTorch for this project. Our PyTorch-based implementation is also available on GitHub[2].

In the original study, the model was evaluated using the Wall Street Journal (WSJ) speech recognition dataset, which contains less than 200 hours of training data. For this project, the LibriSpeech dataset[3] is used instead. The LibriSpeech dataset, introduced by Panayotov et al. [3], consists of approximately

[1] https://github.com/xingchensong/Speech-Transformer-tf2.0
[2] https://github.com/GuiFigueired0/SpeechTransformer-Pytorch
[3] https://www.openslr.org/12

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

1,000 hours of read English speech derived from audiobooks in the LibriVox project. It has been shown to provide superior performance on WSJ test sets compared to acoustic models trained directly on WSJ itself.

## 2 Implementation Details

The project is organized into a modular structure with multiple files and folders to ensure maintainability and clarity. In the root folder, there are three Python scripts, one main **model** folder, and three auxiliary folders. The purpose of each script and auxiliary folder is as follows:

- **/LibriSpeech**: This folder houses the dataset files and two scripts. One script is responsible for converting the dataset from .flac to .wav, while the other traverses all subfolders of the dataset, gathering the transcriptions into simple .txt files paired with the audio IDs to assist the `DataFeeder`. These files are stored in a **data** folder for easy access.
- **/model_weights**: This folder stores the Transformer model weights saved at the end of each training epoch, enabling resumption of training or evaluations with previously trained models.
- **/extra**: Contains supplementary resources, including code for plotting evaluation metrics and a model summary file that provides the total parameter count of the architecture.
- **datafeeder.py**: Implements the `DataFeeder` class, which is responsible for loading, batching, and feeding data to the model during training and evaluation.
- **train_transformer.py**: Implements the training loop for the Speech-Transformer. It also handles saving the model weights and metrics obtained during training.
- **evaluate_transformer.py**: Evaluates the model's performance on test datasets, leveraging the `beam_search_decoding` function and calculating CER and WER.

### 2.1 Model Folder

The **model** folder contains the core implementation of the Speech Transformer. This architecture is primarily divided into three parts: **Pre_Net**, **Encoder**, and **Decoder**. PyTorch was utilized to provide the basic layers of these components, such as `Linear`, `Conv2d`, and `LayerNorm`. Each component is implemented in its own file, along with a corresponding class for its major structure: **TwoD_Attention_layer**, **EncoderLayer**, and **DecoderLayer**, respectively. Additional auxiliary layers and functions are organized into separate files, including `attention.py`, `feed_forward.py`, `input_masks.py`, `loss.py`, `optimizer.py`, and `position_encoding.py`.

A `speech_transformer.py` file is responsible for defining a **Transformer** class, which connects the encoder and decoder using the previously described classes, and a **SpeechTransformer** class, which integrates the Pre_Net with the Transformer. This class also implements the `beam_search_decoding` function, which uses the Beam Search algorithm for evaluation.

Lastly, the `evaluation_metrics.py` file provides a function for evaluating model predictions. This function calculates the Character Error Rate (CER), using Levenshtein distance, and the Word Error Rate (WER), which is already implemented in the Python library `jiwer`.

### 2.2 Differences from the Base Code

Key differences between this implementation and the base code include the following:

- **Framework**: The original implementation used TensorFlow, while this project uses PyTorch.
- **Evaluation Metrics**: Additional metrics, such as WER and CER, were included alongside accuracy.
- **Dataset Organization**: The dataset folder and scripts were reorganized for clarity and ease of use.
- **Beam Search**: A Beam Search algorithm was added to the evaluation section, which was not present in the base code.

- **Padding Masks**: The base code did not use padding masks. While padding masks for text were successfully implemented in the loss function, implementing padding masks for audio data proved challenging due to the numerous convolutions and transformations applied to the audio data. Future work could address this by generating audio masks after the convolutions in the Pre_Net.

## 3 Experimental Setup

The model was trained for ten epochs using the train-clean-360 subset of the LibriSpeech dataset. The entire LibriSpeech dataset was not utilized due to time and memory constraints; however, this subset is still larger than the Wall Street Journal training set used in the original paper. The training process spanned approximately 40 hours on a single NVIDIA GeForce RTX 3050 Laptop GPU with 4GB of memory. Due to the limited GPU memory, the batch size was reduced to 8, resulting in 13,001 batches per epoch.

The hyperparameters for the model were as follows: `num_layers_enc=8`, `num_layers_dec=4`, `d_model=256`, `num_heads=4`, `dff=1024` and `dropout=0.1`. The Adam optimizer was configured with the same parameters as described in the original paper: `Beta1=0.9`, `Beta2=0.98` and `Epsilon=1e-9`. Additionally, a custom learning rate scheduler was implemented in the **model** folder, consistent with the learning rate warm-up approach proposed in the paper.

The differences between this setup and the original paper's implementation are primarily related to dataset size and hardware limitations. The original paper batched samples by approximate feature length and applied a larger penalty on distant position-pairs in the attention mechanism. In contrast, this implementation used simple shuffling for the samples and a basic implementation of the attention mechanism.

## 4 Results

The evolution of the learning curve is illustrated in Figure 1. Due to changes in how the metrics were stored during training, the figure can be divided into two parts. In the first half, metrics were recorded every 100 batches, causing significant variations due to the small batch size. In the second half, the average of these 100-batch metrics was computed, resulting in more stable values.

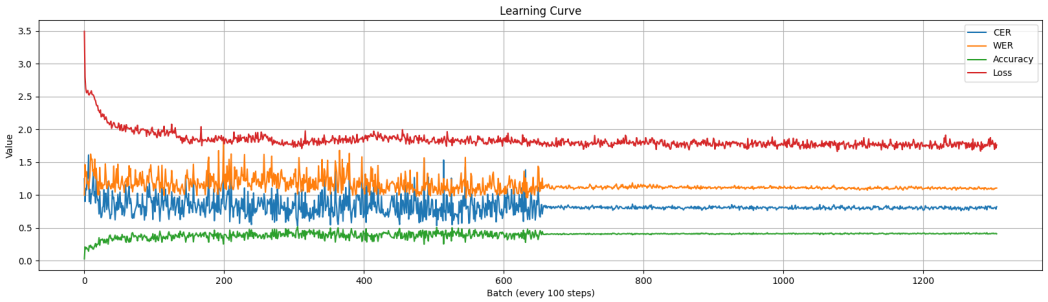


Figure 1: Learning curve evolution across epochs.

Unfortunately, the model's performance did not improve significantly after the second epoch. This stagnation might be due to errors in the implementation. However, the lack of an evaluation module in the original code makes it unclear whether the issues stem from translating the code from TensorFlow to PyTorch or pre-existing flaws in the original implementation.

Given the limited time for debugging and rerunning the experiments before the project deadline, this issue remains unresolved. A PyTorch implementation of an improved version of the Speech-Transformer has been identified[4], which may assist in debugging and refining this implementation in the future.

[4] `https://github.com/kaituoxu/Speech-Transformer/tree/master?tab=readme-ov-file`

Initially, I had some interesting ideas on how to evaluate the model and show its evolution over time. However, since the model did not learn effectively, the evaluation was simplified. Figure 2 presents the different predictions for a sample from the test set at the end of epochs 1, 5, and 10. As shown, the model was able to learn individual words, which indicates that the decoder worked to some extent. Since the model predicts at a character level, this demonstrates functionality within the decoder. However, the predicted words have no correlation with the audio sample.

| Expected transcription | | | |
|---|---|---|---|
| he_hoped_there_would_be_stew_for_dinner_turnips_and_carrots_and_bruised_potatoes_and_fat_mutton_pieces_to_be_ladled_out_in_thick_peppered_flour_fattened_sauce | | | |
| **Epoch** | **CER** | **WER** | **Transcription predicted** |
| 1 | 0.7215 | 1.0357 | which_he_had_been_the_said_and_the_she_was_the_said_and_the_she_was_the_said_that_the_principle_of_the_principlation_of_the_princes_of_the_princes_of_ |
| 5 | 0.7722 | 1.0714 | there_was_not_one_of_the_principle_of_the_princess_of_the_country_of_the_country_of_the_country_of_the_country_of_the_country_of_the_country_that_the_ |
| 10 | 0.7405 | 1.1429 | and_when_they_were_she_said_that_they_were_all_that_they_had_not_been_in_the_country_of_the_country_of_the_country_of_the_country_of_the_country_of_th |

Figure 2: Different predicted values for a sample from the test set at the end of epochs 1, 5, and 10.

This behavior likely suggests an issue in the encoder, particularly in the Pre-Net and its use of the 2D attention mechanism and how it connects to the encoder. This flaw might explain the stagnation of the learning curve; without meaningful input from the audio, the decoder cannot progress beyond predicting random words.

## 5 Discussion

Reproducing a Transformer model poses significant challenges, primarily due to its high level of complexity. The sheer scale of Transformer models demands substantial training time and computational resources. This makes mistakes particularly expensive, as each run requires significant time, making debugging and experimentation both time-intensive and resource-heavy.

Despite these challenges, this project has been a valuable learning experience. Transformers represent a fascinating and influential advancement in machine learning. Reproducing this model provided an opportunity for an in-depth exploration of its components and offered insight into adapting an implementation from another framework. Additionally, it highlighted the critical importance of rigorous debugging and efficient resource management in deep learning research.

## References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998-6008.

[2] L. Dong, S. Xu and B. Xu, "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition," 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 2018, pp. 5884-5888, doi: 10.1109/ICASSP.2018.8462506.

[3] V. Panayotov, G. Chen, D. Povey and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 2015, pp. 5206-5210, doi: 10.1109/ICASSP.2015.7178964.