Polynomial Expansion using Seq2Seq Transformer Encoder-Decoder Network

Debayan Ghosh
Carnegie Mellon University

September 25, 2022

1 Abstract

This repository implements a Sequence-to-Sequence Transformer model for Polynomial Expansion. The repository implements Training, Data Preparation, and Evaluation code for the Polynomial Expansion Task

2 Model Architecture

The nature of the problem (Polynomial Expansion) is synonymous with Seq2Seq tasks such as Automatic Speech Recognition (ASR), Machine Translation, etc. Traditionally LSTM/RNN recurrence models were employed for such tasks, where the Encoder Hidden States were used to capture Input information, and an Attention-Based Auto-regrssive Decoder was used as a Language model to generate new text.

However, more recent work on Transformers built on Self-Attention Layers showed an improvement in Seq2Seq tasks when compared to traditional recurrent networks. Motivated by this, the Transformer model was chosen for the Polynomial Expansion task.

The model layer configuration chosen for this task are the following:

- Number of Encoder Layers: 4
- Number of Decoder Layers: 4
- Number of Attention Heads: 8
- Hidden Dimension: 256

  The number of Trainable Parameters in the model: 4,249,636.

  The model was trained for 30 epochs.

3 Hyperparameter Choices

- Optimizer: We used the Adam optimizer, with:
  - initial Learning Rate = 0.0001.
  - The value of β_1 (Running $\frac{\partial L}{\partial W}$ factor) = 0.9
  - The value of β_2 (Running $\frac{\partial L}{\partial W}^2$ factor) = 0.98

We did hypermarameter tuning to arrive at these parameters. Adam was chosen as the optimizer, as it combines Momentum and RMSProp optimization methods, i.e.

  - Mometum Property: The derivative update at step t is a running average of the current derivative, and past derivative values (Momentum). This helps us take steps on an average of current and past gradients, and helps in smoother convergence
  - RMSProp Property: RMSProp estimates the Hessian, and this enables each dimension to have an independent learning rate.

- Scheduler: We employed the ReduceLR on Plateau scheduler.
  - The Scheduler reduces the LR based on the Validation Loss.
  - The Patience Value was set to 2. This means that if there is no improvement in Validation Loss for 2 epochs, we reduce the LR.
  - The threshold value was set to 0.001.

4 Data Preparation

The full dataset can be found at dataset_files/dataset.txt , which has 1,000,000 samples

- Train Split = 0.93 : Train (train.txt) with 930,000 samples
- Validation Split = 0.01 : Val (val.txt) with 10,000 samples
- Test Split = 0.06 : Test (test.txt) with 60,000 samples

The reasoning for the split is as follows:

- We wanted a significant amount of Test Samples (60,000) to prove our model actually works.
- For Validation, 10,000 samples was sufficient to track if the model was learning appropriately (i.e track over-fitting, etc.)
- For Training, 930,000 samples were chosen which was sufficient for training the model.

NOTE: Training with 980,000 samples resulted in significant improvement in Test Accuracy (On 10,000 test samples). However, I felt having a slightly lower accuracy (4%) on 60,000 test samples was a better indication of model performance, than having a higher accuracy with 10,000 test samples. We will report the accuracy for both these type of Train/Test/Val Splits(0.93/0.06/0.01 and 0.98/0.01/0.01), but encourage the reader to consider the Train/Test/Val Split of 0.93/0.06/0.01 as the reference result. Note that reference .pth file is provided only for 0.93/0.06/0.01 split

5 Tokens

The following was the vocabulary for our model. This was obtained by analysing the Train dataset.

sin, cos, tan, $numeric\_digit(0-9)$, $alphabet$, (,), $+$, $-$, $**$, $*$.

The total Vocabulary size was 36 for our model. The Token-Index mapping can be found here: vocab.json

Train/Test/Val Split	Test Accuracy
0.93/ 0.06/ 0.01	92.71 %
0.98/ 0.01/ 0.01	96.51 %

Table 1: Accuracy on Test Dataset

6 Results

The Test Accuracy results on the different Train/Test/Val split is reported in Table 1

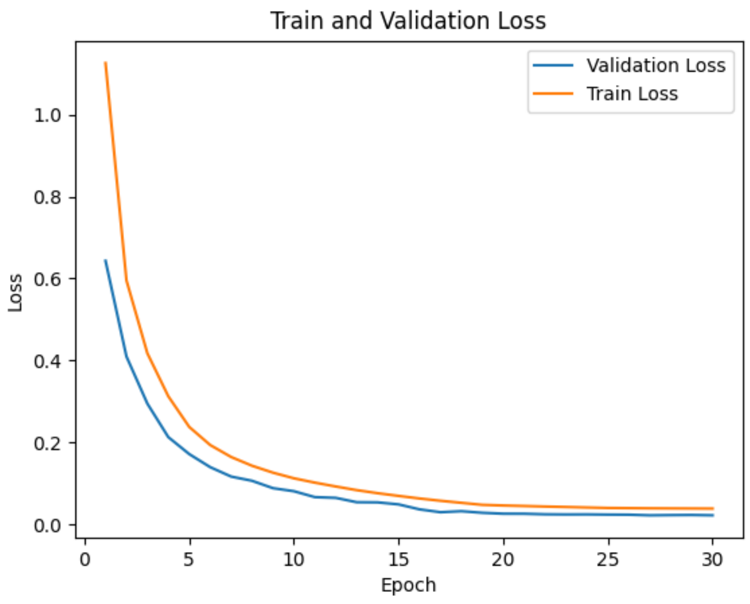
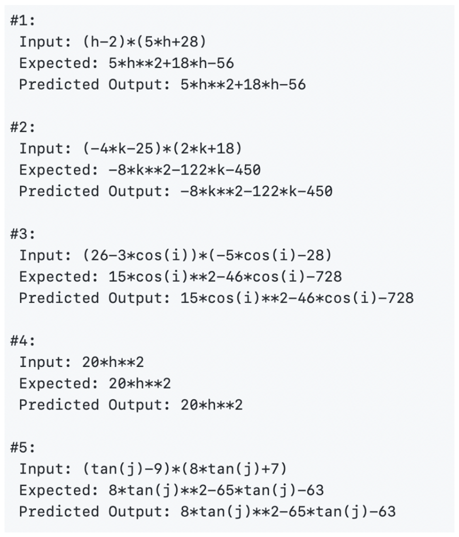


Figure 1: Train and Validation Loss Curve



```
#1:
 Input: (h-2)*(5*h+28)
 Expected: 5*h**2+18*h-56
 Predicted Output: 5*h**2+18*h-56

#2:
 Input: (-4*k-25)*(2*k+18)
 Expected: -8*k**2-122*k-450
 Predicted Output: -8*k**2-122*k-450

#3:
 Input: (26-3*cos(i))*(-5*cos(i)-28)
 Expected: 15*cos(i)**2-46*cos(i)-728
 Predicted Output: 15*cos(i)**2-46*cos(i)-728

#4:
 Input: 20*h**2
 Expected: 20*h**2
 Predicted Output: 20*h**2

#5:
 Input: (tan(j)-9)*(8*tan(j)+7)
 Expected: 8*tan(j)**2-65*tan(j)-63
 Predicted Output: 8*tan(j)**2-65*tan(j)-63
```

Figure 2: Example Predicted Outputs of our model