Author – Abhinand Jha

PROBLEM STATEMENT

Implement a deep neural network model that learns to expand single variable polynomials. Model input is factorized sequence and output is predicted expanded sequence.

SOLUTION

The task of expanding factored sequences to an expanded sequence can be thought to be like a **Sequence-to-Sequence** translation problem.

Consider the task of neural machine translation, in which the goal is to convert the text from one language to another (e.g., English to French). The given problem of polynomial expansion is very similar to the neural machine translation task (albeit a lot easier than translation) and thus, this problem can be solved by using deep neural network architectures designed for the task of sequence-to-sequence conversions (seq2seq models) like RNNs, GRUs, LSTMs and Transformers.

To solve this problem, first I tried a traditional LSTM based Neural Network architecture but it was not able to attain an accuracy of more than 90% while training, and the training required more time to converge to an optimal solution. The introduction of Transformers and the attention mechanism revolutionized the way NLP tasks are solved currently (especially natural language generation and machine translation).

With this motivation, I designed a **compact attention-based architecture** to solve this problem with **4 million parameters only.** This model achieves an **accuracy of 98.63%** on the test data with, **with only 45 minutes of training** on a single NVIDIA GeForce RTX 3090 GPU. The code for training, testing and data processing is implemented using sacred python library that allows structured command line execution of python scripts. Exact commands to train and test the code is provided in the README.md file.

APPROACH AND MODEL ARCHITECTURE

Exploratory data analysis

- The very first step in solving any ML problem is to analyze the data, as it helps immensely in deciding the type of architecture to use for the problem
- Using regex parsing, I analyzed the data in various ways to extract the number of unique tokens, all the mathematical expressions, types of parenthesis and mathematical functions used in the training data
- The EDA helped me in narrowing down the tokens into Digits(0-9), Lowercase alphabets(a-z), Parenthesis(“(” , “)”) and Trigonometry functions (sin,cos,tan)

Data-preprocessing

- The entire dataset.txt file is randomly split into three files – train.txt, val.txt and test.txt in an 80:10:10 ratio (This ratio is parametrized and can be changed if required before training the data)
- The model parameters and hyperparameters are tuned on the training and validation datasets. The model testing is carried out on the test dataset
- Training/validation data is read, tokenized, and stored as source and target pairs in memory
- The training and validation data is split into mini-batches for training. The batch_size is a hyperparameter which was selected based on the validation loss.

Model Architecture

- The model I used is inspired by a transformer based encoder-decoder architecture as shown below

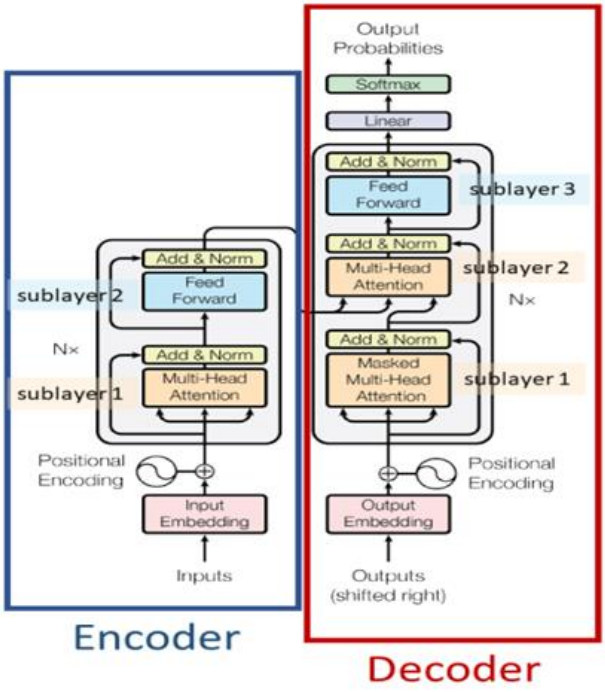

Figure 1 – Transformer architecture

- As one of the requirements of the problem was that the model should have less than 5 Million trainable parameters, I designed a very compact transformer architecture
- In my proposed architecture, I am using 256 dimensional embedding vectors that serve as the input features
- Both the encoder and decoder have only 3 LSTM layers each
- For the attention mechanism, I am using only 8 attention heads for both the encoder and decoder layers to keep the number of trainable parameters less
- Lastly, to help prevent overfitting on the training data, I am using a dropout of 10% in both the encoder and decoder layers
- The attention mechanism used in the model is the dot-product attention as described below

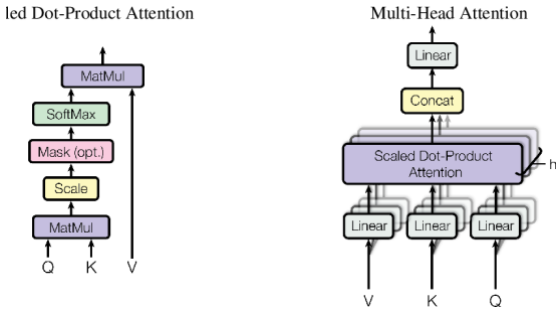

Figure 2 – Attention mechanism

Training and hyperparameter selection

- The model was trained for 20 epochs with a batch size of 512 and 800k training data points
- Training for more than 20 epochs did not provide any substantial accuracy improvement and it could make the model prone to overfitting. Therefore, the model parameters that minimized the validation loss during 20 epochs was selected and saved as the best model (stored in ./model/best_model.pt)
- The learning rate and batch size was selected using a grid search over different ranges and their effect on the validation loss
- The best results were obtained using a batch size of 512 and learning rate of 0.0005

Evaluation and results

- I implemented a **batch prediction** logic that decreased the time taken for the model to give out predictions by more than 70%
- The trained model is evaluated on the test dataset and it's accuracy is reported
- The accuracy of the model on test data is calculated using a **strict equality** in which the prediction from the model and expected results are compared character by character. The prediction is considered correct if and only if the predicted sequence exactly matches the actual sequence. The accuracy is calculated using the formula

  Accuracy = (No. of correct predictions / Total predictions)*100
- The training and validation loss obtained is shown below

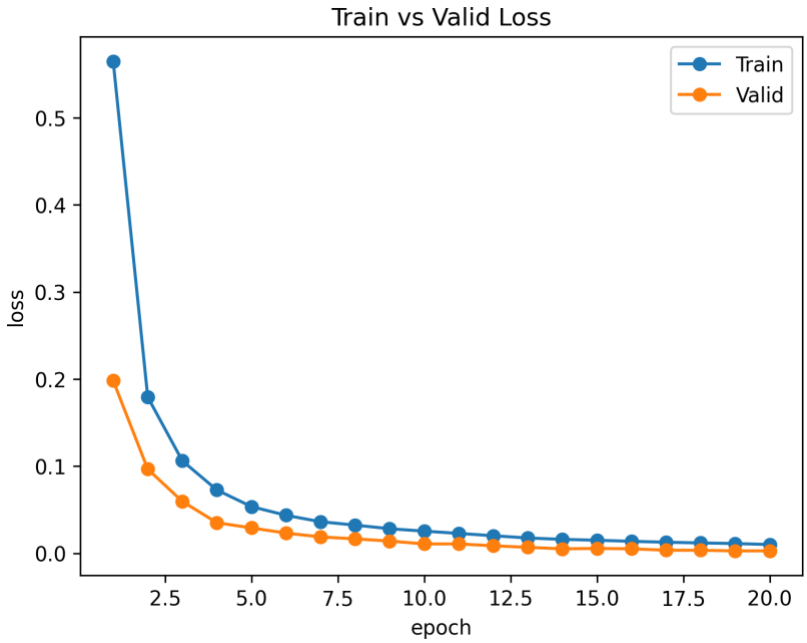

Figure 3 – Training and validation loss v/s epochs

- All predicted outputs from the model are stored in the output directory
- Some example predictions from the model –

```
--- example 1 ---
factored:   3*x*(x-29)
expanded:   3*x**2-87*x
prediction: 3*x**2-87*x
equal:      True
```

```
--- example 2 ---
factored:   7*s*(-7*s-8)
expanded:   -49*s**2-56*s
prediction: -49*s**2-56*s
equal:      True
```

REFERENCES

[1] Attention Is All You Need (https://arxiv.org/abs/1706.03762)

[2] https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

[3] Sequence to Sequence Learning with Neural Networks (https://arxiv.org/abs/1409.3215)

[4] Deep Learning for Symbolic Mathematics (https://arxiv.org/abs/1912.01412)

[5] https://sacred.readthedocs.io/en/stable/quickstart.html

[6] https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html