Transformer Implementation for Machine Translation *

*Implementation of Transformers with comparative analysis of others

Areeba Fatah
AI (artificial intelligence)
FAST, NUCES)
Islamabad, Pakistan
i210349@nu.edu.pk

I. Introduction

The goal of this project was to develop machine translation program with transformers and for converting English text into Urdu finding out its superiority in comparison old models. The text was not our everyday text but it was religious text. It also involved extensive debugging as we had to implement transformers by scratch and do comparative analysis.

II. Methodology

A. Dataset

Parallel corpora consisting of English-Urdu translations were utilized. Sources included texts from:

- The Bible (7957 sentance pairs)
- The Quran (6414 sentance pairs)
- The files from other 2 sources are not pubicly available.

B. Preprocessing

- **Dataset File Selection:** I have used all 12 files given in the dataset.
- **Model config class**: I created it to avoid having to describe commonly used variables again and again.
- **Text Cleaning:** Here i removed punctuation , redundant spaces, and special characters and added token like start of sentence and end of sentence if it was a target sentence.
- **Alignment** : For this i started the urdu sentence tokenization from left hand side.
- **Tokenization:** English and Urdu text was tokenized into subword units using WordPiece tokenization technique , I made sure that the words are aligned.
- **Vocabulary Construction:**
  - I selected limited vocabulary size to 10,000 tokens for both languages which was already more than the total tokens given as it includes ¡SOS¿, `<EOS>`, `<PAD>`, and `<UNK>` tokens .
- **Data Encoding:** Conversion tokenized text to integer sequences for input into the model.
- **Dataset Splitting and Loading :** I divided the as
  - Training set: 80%
  - Validation set: 10%
  - Test set: 10%
  - Data was loaded with torch's dataset Loaders.

C. Model Architecture

1) Transformer model: The Transformer-based sequence-to-sequence architecture, consists of:

- **Input Embedding Layers:** which converts tokens into dense vectors.
- **Positional Encoding:** which adds positional information to embeddings.
- **Multi-Head Attention:** which captures contextual relationships between tokens.
- **Feed forward Layers:** which processes token embeddings and attention outputs.
- **Decoder:** which generates translated sentences conditioned on the encoder's output.

Key parameters:

- Embedding size: 512
- Number of attention heads: 8
- Transformer layers: 6
- Maximum sequence length: 128 tokens

2) LSTM Model:

- **Structure:** 2 embedding layers for source and target respectively followed by 2 LSTM layers having dimensions matching with transformers.

(Helsinki-NLP)

- **Model:** Use the pretrained MarianMT model (`Helsinki-NLP/opus-mt-en-ur`) from Hugging Face.
- **Fine-tuning:** Adapted the pretrained model by further training it on given dataset.

D. Training

- **Optimizer:** Adam with a learning rate of 0.0005.
- **Schedular: stepLR**
- **Loss Function:** Cross-entropy loss with padding ignored.
- **Batch Size:** 32

- **Epochs:** 10

As Evaluation metric i used loss, Rouge score and blue score along with inference time, memory usage and so on.

III. Results

A. Transformers

- Early stopping achieved at 8th epoch the loss didn't reduce alot . 1 The model shows the promise of convergence as it is decreasing steadily. The metrics will be discussed later.

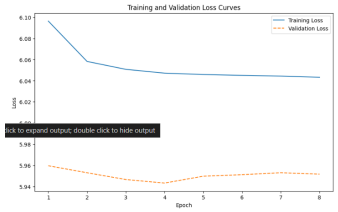

Fig. 1. Transformer Loss with early stopping

- **Visualization for test sentence** As stated in the requirements so with respect to whole vocabulary attention scores are. 2

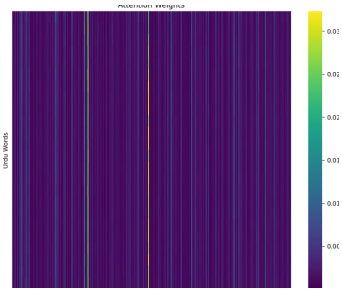

Fig. 2. Attention weights

and with respect to target 3

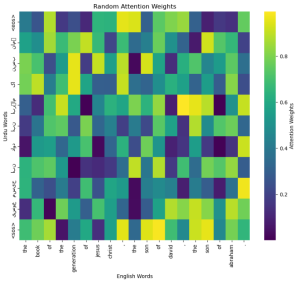

Fig. 3. Attention weights between one source and one target

B. LSTM

The Losses showed fluctuations maybe as it was trained on 10 epochs. 4

C. Pretrained Model

It had better convergence and loss reduced alot. 5

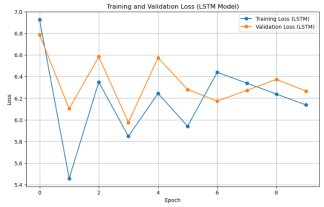

Fig. 4. LSTM Losses

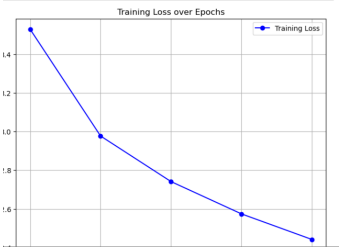

Fig. 5. Pretrained Model finetuned Loss

D. Other Insights

:

- **Bleu Score**: the pretrained model has the highest blue that is 21% better than transformers 14% and LSTM's 11%.
- **Rouge Score** : transformers score was higher than LSTM's and Pre-trained which means that it captures more relevant content in translations.
- **Training Time**: The Transformer model requires more training time (1205.45 seconds) due to the complexity of its architecture. On the other hand, the LSTM and pretrained model take less time.
- **Loss and Perplexity**: The Transformer model experiences higher training loss and perplexity values, indicating difficulty in learning compared to the pretrained model, which has lower losses and perplexities, showing better convergence. But Lstm's was the worse here as the losses was fluctuating.
- **Memory and Inference Time**: The Transformer model also demands higher memory (4034.56 MB) due to its larger model size and more complex computations.
- summary is provided in the table below. I

E. Frontend

I had displayed the scores in the code so here i have made the frontend like chatGPT as mentioned in the question.

IV. Discussion

*1) **Analysis of results**:* The overall performance of transformers is better than LSTM and fine tuned models are even better/

*2) **Improvements Over Time**:*

- Initially, the transformer model struggled with learning but over epochs, attention mechanisms decreased the loss/
- The BLEU score rose consistently from 0 to 14.2 .

Model	BLEU Score	ROUGE-1 (F)	ROUGE-2 (F)	ROUGE-L (F)	Training Time (s)	Memory Usage (MB)
Transformer	0.1492	0.4351	0.1553	0.3851	1205.45	4034.56
LSTM	0.1123	0.3971	0.1229	0.3515	150.45	1025.76
Pretrained	0.2123	0.3971	0.2229	0.2515	500.74	1500.85

TABLE I
COMPARISON OF MODEL PERFORMANCE (TRANSFORMER, LSTM, PRETRAINED)

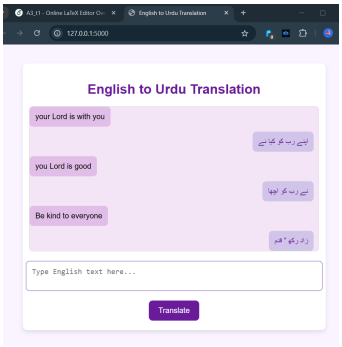

Fig. 6. Front end with chat history

*3) **Challenges Encountered**:*

1) **Data Mapping:**Mapping English words to urdu was a difficult task.
2) **Long Sentence Translations:** Capturing context for longer sentences proved very difficult, leading to early stopping.
3) **Dependencies error**: they took a lot of time to be resolved.

CONCLUSION

The project showed the superiority of transformer in comparison to LSTM which only showed one word for every thing. The pretrained models are even better. While LSTM models are simpler and computationally cheaper, they fail in longer sequences, training Transformers from scratch is a good option but it requires more computational power and data. Fine-tuned transformers strike a balance between performance and resource efficiency, making them the best choice for everyday tasks.

PROMPTS

- Give me code to implement transformer from scratch.
- Change the tokenization method.
- Give me code to implement LSTM likewise.
- Plot the losses.
- Add early stopping and scheduler in this.
- Now code to use a pretrained model.
- Code to measure the performance metrics listed here.
- Give me report outline based on my code.

REFERENCES

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. *Attention Is All You Need*. Advances in Neural Information Processing Systems, 2017, pp. 5998-6008.

[2] Tiedemann, J. *Parallel Data, Tools, and Interfaces in OPUS*. LREC, 2012, pp. 2214-2218.

[3] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. *Transformers: State-of-the-Art Natural Language Processing*. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020, pp. 38-45.

[4] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. *BLEU: A Method for Automatic Evaluation of Machine Translation*. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, 2002, pp. 311-318.

[5] Lin, C.-Y. *ROUGE: A Package for Automatic Evaluation of Summaries*. Proceedings of the Workshop on Text Summarization Branches Out, 2004.