CSOC'25 IG Assignment-4

Natural Language Processing

Presented By- Juhi Saxena
Roll No: 24095044

Summary of the Task

1. The task assigned revolves around the deep analysis and scrutiny of sequence to sequence modelling and elucidating upon the performance of different architectures employed in sequence to sequence modelling in a machine translation task using primarily three architectures- A simple Encoder-Decoder architecture, Encoder-Decoder+Luong Attention and ultimately finetuning a pre-trained transformer architecture to our dataset

2. The performance metric utilized in this task is namely,The Bleu Score(Bilingual Evaluation Understudy) which can judge the quality of a translated sentence based on one or more reference sentences by calculating n-gram precision, avergaing that and ultimately exponentiating it along with a Brevity Penality(Applied to very short machine translations)

Approach Implemented

Choice of the Language Pair Dataset-"agentlans/en-fr"

The dataset chosen is the, "agentlans/en-fr"dataset which is offered by Hugging Face under the translation datasets. Its a standard hugging face dataset consisting of around 3 million samples of English and French Sentence pairs along with other attributes like readability,translation quality etc.

Data Sampling-Catering to Memory and Time limit constraints

The dataset is pretty large and thus utilizing the "CUDA"memory effectively to train the model requires a small sample set for training. (Many platforms were explored-Kaggle,Colab and Lightning.AI.Lightning.AI does provide a point based access to a number of GPU models, but connection to the HuggingFace dataset was an issue despite multiple attempts.Hence finally Colab was employed since installing the fsspec library fixed all path errors and dataset was easily accessible)

Data Sampling

Data Sampling

Out of all the available 3 million sentence pairs, 14,000 are used for training and 4000 for testing-which is further divided into 2000 test samples and 2000 validation samples-(To train a model from scratch its understandable that these many samples are pretty less- though adhering to the colab time constraints, with GPU training approximately took 4 hours(for 10 epochs), this time could be unimaginably high on an ordinary cpu-7 to 8 hours for a single epoch!!)

Pre-processing Pipeline

Pre-processing,Tokenization and Building the Vocab

Spacy has been utilized for tokenization, pre-processing and building the vocab(token-based encoding)-this approach was implmented after testing two major approaches-firstly pretrained BERT Tokenizer was being utilized for the English sentences and Camembert Tokenizer for the French sentences, however this led to an OOM error, training sample size couldnt have been reduced rather neither the max length of the sentence(potential context loss).Second approach tried was simple lowercasing and splitting using ".split()"method-but this led to an absolute 0 Bleu score, thus finally spacy was used for pre-tokenization and vocab building.

Tokenization was not done on the fly-rather finsihed beforehand to reduce overhead and cut down on the computaion expenses-also trainable 768 dimensional embeddings have been utilized in both the encoder and decoder stacks.

Simple Encoder and Decoder Architecture

The simple encoder-decoder architecture is primarily made up of three layers-The Encoder, A bridging layer and the Decoder.

The Encoder Architecture

- Begins with a trainable embedding layer that maps input token IDs to 768-dimensional vectors.
- Uses a 2-layer bidirectional LSTM with hidden_dim units per direction to model contextual informat and Operates in batch-first mode, with input/output shaped as (batch_size, seq_len, feature_dim)
- After LSTM processing, extracts the final forward and backward hidden states from the topmost layer and Concatenates these two states to form a single context vector of size $2 \times$ hidden_dim per input sequence.

Simple Encoder-Decoder Architecture-Continued

Briding Layer

- The encoder output is used to initialize the LSTM based Decoder's hidden state and cell state. To effectively project the context vector into these two portions-this layer has been added
- Outputs are unsqueezed and repeated across num_layers to match the decoder's LSTM layer depth(2 in our case)

The Decoder Architecture

- Uses a trainable embedding layer to map token IDs to 768-dimensional vectors.
- Employs a multi-layer(double layered) unidirectional LSTM to capture left-to-right dependencies in the output sequence, which is essential for autoregressive decoding.
- Takes initial hidden and cell states from the BridgeLayer, ensuring that decoding is contextually grounded in the encoded input representation.

Simple Encoder-Decoder Architecture-Continued

- Applies a linear layer after the LSTM to project hidden states into the vocabulary space, enabling token-wise prediction at each time step.
- Includes a decoding method to iteratively generate output tokens, providing a simple yet effective strategy for sequence generation during inference.

The above architecture is trained using Pytorch Dataset and Dataloader classes using a batch size of 4(otherwise OOM error) for 10 epochs using the Adam optimizer and a learning rate of 3e-4.

Results Obtained-*Simple Encoder Decoder* Architecture

- Bleu score obtained on the validation set-0.0135
- Bleu score obtained on the test set-0.0147

Encoder-Decoder+Attention Architecture I

Luong attnetion is added to the previous simple Encoder-Decoder architecture and a few changes were made.

The Encoder Architecture

- Everything in the architecture remains intact, but instead of returning the final hidden states across all LSTM layers, the encoder returns the complete sequence of LSTM outputs. This provides contextual representations for all time steps, which is essential for integrating with attention-based decoder mechanisms.

Bridging Layer

Encoder-Decoder+Attention Architecture II

- As with the non-attention architecture, the BridgeLayer is used to initialize the decoder's hidden and cell states. However, instead of using the final hidden states directly from the encoder, the full sequence output is first mean-pooled across time steps. This pooled representation serves as a summarized context vector, which is then linearly projected to generate the initial states for the decoder. This design aligns with attention-based encoder-decoder setups, where the full encoder output is preserved for attention, and a pooled version is used solely for decoder state initialization

The Decoder Architecture

- Added a Luong-style attention mechanism, which computes alignment scores between the current decoder hidden state and encoder outputs using a dot-product-like formulation.
- Introduced a new layer: self.attn, a linear transformation to project decoder hidden states before attention scoring.

Encoder-Decoder+Attention Architecture III

- Incorporated a context vector, computed as a weighted sum over encoder outputs based on attention scores.
- Concatenated the context vector with the decoder hidden state, and passed it through a self.context_combine layer to fuse contextual and local information.
- Modified both forward() and generate() to include attention computation at every time step, allowing the decoder to dynamically attend to different source positions during generation.

The rest of the training pipeline remains the same with same hyperparamters to maintain consistency while comparing

Results Obtained-*Attention Based Encoder-Decoder Architecture*

- Bleu score obtained on the validation set-0.0209
- Bleu score obtained on the test set-0.0201

Fine-tuning Transformer

- T5-small decoder only model has been used for this purpose
- In the pre-processing pipeline, the only change made is using T5's pretrained tokenizer for tokenization purposes.Also DataCollatorForSeq2Seq from hugging face has been utilized as the collate function to generate batches of data for training.
- Batch size 8 has been used and training is done for 3 epochs-learning rate is same as the previous two approaches(AdamW optimizer has been used).

Results Obtained-*Pre-trained Transformer Architecture*

- Bleu score obtained on the validation set-37.69116
- Bleu score obtained on the test set-37.0824

Comparing the Three Architectures

- The test and validation Bleu Scores are pretty close across all three architectures-overfitting is not a problem.
- There is a highly significant difference between a pre-trained transformer architecture and simple encoder-decoder or attention based encoder-decoder architecture.Attention based encoder-decoder architecture is slightly better than simple encoder-decoder architecture.
- This vast difference in results can be attributed to the tradeoff between high quality translation and computational resources.A transformer architecture not only uses mechanisms like self attention, masked attention, cross attention, multiple layers are used within the architecture and even the sub-layers employ multi-head attention + positional encoding too, layer normalization+residual connection is too applied.

Why such a huge difference?

- Nothing of this sort is used in simple architectures, to even train a simple 2 layered LSTM architecture requires an ordinary T4 GPU to run for 4 hours for 10 epochs and for a cpu based architecture its nearly 7-8 hours per epoch. Not only this, lack of training data is too a factor.We have sampled data- but for a task as intensive as translation, based on research papers models are often trained for 12 hours or so on 8 GPUs using millions of examples to achieve state of the art accuracy.
- These could have been the potential reasons for such significant performance drop.Also Bleu score is pretty strict in penalizing taking into consideration 4-grams or higher n-grams for analysis.

-Resources Utilized- Resources on COPS IG(GitHub Repo)
-Kaggle
-ChatGPT for grammatical errors and theoretical questions.
Thank You!