Natural Language Processing (CS60075)

Assignment 2: Seq2seq Language Models

Deadline: 6th Apr 2025 11:59 pm IST

General Guidelines

- This assignment is to be done individually by each student. You should not copy any code from one another, or from any web source. We will use standard plagiarism detection tools on the submissions. Plagiarized codes will be awarded zero for the assignment, and we will not differentiate between who copied from whom.
- In this assignment, you are allowed to use Python libraries like pytorch, transformers, spacy, NLTK, numpy, pandas and matplotlib for performing different sub-tasks. However, your code should follow an object-oriented approach as described in the tasks.
- **You can utilize free GPU services to run your experiments, e.g. Google Colab or Kaggle (limited free use).**
- **Follow the submission instructions given at the end. Solutions should be uploaded via the CSE Moodle website (see course website for details).**
- You should organize your code into different functions and add explanatory comments. Make sure your code can be easily read. Codes that cannot be understood will be penalized.

Overview

In this assignment, we will use Deep Neural Network (DNN) based language models to generate titles for new Wikipedia articles, given the body text. Specifically, we shall train a Sequence to Sequence (Seq2seq) model that encodes an input sequence of tokens and then generates a desired sequence of tokens, using Recurrent Neural Networks (RNN) and Transformers. Such kinds of seq2seq models are useful for tasks like translation and summarization, and the "title-generation" task of this assignment can be considered as a form of extreme document compression. Finally, we shall try to improve different parts of the model for better performance.

A. Dataset and Preprocessing

The dataset contains a random selection of Wikipedia articles, in two standard CSV files. Each row contains the title of the article along with the text body. For this assignment use the text field as input and the title as the target output.

- The dataset is already divided into train and test sets
- The 'test.csv' file contains 100 articles.
- The 'train.csv' file contains ~14k articles.

[TASKS]

Download the dataset from this link.

1. Extract 500 articles from the training set and keep them as the validation/development set.
2. Remove punctuations and non-ASCII characters (if any)
3. Stopword removal
4. Stemming/Lemmatization
5. Any other pre-processing steps you feel will improve efficiency and make the data more suitable for this type of language modeling.

B1. Setting up a basic RNN Seq2seq model

EncoderRNN takes an input token sequence as input, and maps them into some vector embeddings which are then fed to a bidirectional Gated Recurrent Unit (GRU) capturing the meaning of the entire sequence into some hidden embeddings.

DecoderRNN takes as input one token as input along with the previous hidden state, encodes the token into vector embedding and processes it through a GRU (uni-directional) to get the current hidden state and outputs. These outputs are fed through a linear Feed Forward Network (FFN) to map it to the vocabulary dimension, obtaining a set of logits. Perform softmax on the logits and return this probability distribution of tokens along with the current hidden state.

Seq2seqRNN consists of an EncoderRNN which "encodes" the input sequence, and these hidden representations, along with the "<bos>" (begin of sequence) token as input, are fed into a DecoderRNN to generate the first output token probabilities, along with the updated hidden states. From the probabilities, select the corresponding token with the highest value as the output token.
This process of feeding DecoderRNN with the last generated token as input along with the previous hidden states is to be repeated for a maximum of "max_new_tokens" times, or till the output token is the "<eos>" (end of sequence) token.
Note that during training, however, the input token fed to the DecoderRNN is the actual target token in the ground truth, and not the token generated in the previous time step. This process is also known as teacher-forcing.

You can use this PyTorch tutorial for reference.

[TASKS]

Train and evaluate Seq2seqRNN

1. Create the 3 defined classes EncoderRNN, DecoderRNN and Seq2seqRNN extending torch.nn.module, implementing the corresponding constructors (**__init__**) and the **forward** functions. You can add other relevant functions as required. Add dropout/ ReLU where you see fit. All parameters can be randomly initialized. While creating the vocabulary, consider only tokens that appear in at least **1%** of the training set. You can use a hidden dimension of **300**.

2. Train the model to predict titles from the article bodies from the training data. During training select appropriate hyperparameters like max_new_tokens, batch size, number of epochs, learning rate. You may extract a validation set from the training data to help you decide optimal hyperparameters.

3. Use the trained model to generate titles for articles in the test set, and calculate the ROUGE-1, ROUGE-2 and ROUGE-L f1 scores. [link to resource]

B2. Improving the RNN model

GloVe (Global Vectors for Word Representation) embeddings are designed to capture the semantic meaning and relationships between words by analyzing the co-occurrence statistics of words in a large text corpus. Download and use the Wikipedia 6B vectors from this link. You may use the entire 400k vocabulary, or a subset based on the previously calculated vocabulary.

HierEncoderRNN is a hierarchical encoder which processes the input at two levels - word and sentence level. The first word-level GRU processes the input token embeddings as earlier, and the output states are averaged for tokens in each sentence. These averaged embeddings for each sentence are encoded by a second sent-level GRU, which also utilizes the final hidden states from the word-level GRU. The final hidden state from this sent-level GRU is returned, to be fed into the decoder. You can split sentences by punctuation (e.g. period, question mark, exclamation) or using a sentence tokenizer (e.g. from spacy/nltk libraries).

Decoder2RNN is a decoder with 2 GRUs, with the second one processing the outputs given by the first one, and having the same initial hidden states from the encoder. The output logits from this second GRU is then processed as earlier.

Beam Search algorithm can be used during inference to find an optimal sequence of output tokens instead of greedily selecting the best token at each time step. The algorithm expands all possible next steps or tokens creating a tree of possible sequences. The top k sequences with highest scores are retained after each iteration, The best one is returned after the stopping criterion is reached for all the beams. [link to resource]

[TASKS]

4. Import the described GloVe embeddings into the RNN model using a setter function **load_embeddings** in the EncoderRNN class. Have a parameter in Seq2seqRNN constructor which will contain if embeddings are to be loaded, and the path from where to load.

5. Write a class HierEncoderRNN as described above. Have a parameter in Seq2seqRNN constructor to load either EncoderRNN or HierEncoderRNN.

6. Write a class Decoder2RNN as described above. Have a parameter in Seq2seqRNN constructor to load either DecoderRNN or Decoder2RNN.

7. Incorporate beam search in Seq2seqRNN. Have a parameter in the forward function of Seq2seqRNN to select if greedy or beam search is to be used during decoding.

C1. Attention is all you need!

Transformer models leverage attention mechanisms offering different benefits over RNNs, and have been widely adopted in all NLP tasks. Huggingface offers a set of very useful libraries for using Transformers, data processing, evaluation and many others. They also provide extensive documentation and informative articles on common tasks. In this part, you have to load a pre-trained Seq2seq Transformer model, fine-tune it on the given dataset, and generate and evaluate new titles for the test set articles. T5 models have a max sequence length of 512 tokens, For articles exceeding this, simply truncate the remaining text.

[TASKS]

Use the raw text (with punctuations, stopwords etc.) for this next part

1. Load the pretrained `google-t5/t5-small` model using the AutoModelForSeq2SeqLM class.

2. Train the model using the Seq2SeqTrainer on the training and validation data, while trying different hyperparameters in Seq2seqTrainingArguments

3. Generate predictions on the test set [link to huggingface article] and calculate the ROUGE metrics as in Part B, greedily and by beam search

C2. Prompt engineering

Modern Transformer models have already been "Instruction-tuned" on a wide range of tasks, which improve their generalizability. This allows the model to perform well on new unseen tasks/data without additional finetuning. Google's Flan-T5 model family is a well-known Instruction-tuned Seq2seq LM, excelling in various NLP tasks. You simply give an instruction describing the task to be performed, or a "prompt"; followed by the actual data, as input text to the model. [Link to huggingface article]

[TASKS]

Use the raw text (with punctuations, stopwords etc.) for this next part

4. Load the pretrained `google/flan-t5-base` and `google/flan-t5-large` models using the AutoModelForSeq2SeqLM class.
5. Try giving each of the models at least 2 variations of prompts to **generate** the titles from the article bodies. Evaluate the generated outputs as before.

Submission Instructions

Submit one .zip file containing the following. Name the compressed file the same as your roll number. Example: "25CS60R00.zip"''

1. **Codes:** Submit 3 files taskA.py, taskB.py, taskC.py (or corresponding .ipynb or .sh scripts) from where execution will be started. You are free to create other files as per convenience for legibility.
2. Do **NOT** include the **dataset folder** or any of the **saved models**.
3. **A report (as a doc or pdf)** containing:
    - Report the time taken (in seconds) for each part of the code to run.
    - ROUGE-1,2,L F1 scores in the form of tables and/or plots: for the basic RNN model, RNN with each of the 4 individual improvements and with all the improvements, and the Transformer models, finetuned and zero-shot prompting.
    - Some intuitions/discussions, based on the performance of different model variants.
    - The assignment requires many design decisions to be taken. Explain the different choices you've made and the reasons behind them. Marks will be given based on the novelty and ingenuity of the solutions.

Queries, if any, should be addressed to TAs Shounak Paul and Soham Poddar (email IDs given on course website).