HW4 - Summarization

COMS W4705 Natural Language Processing - Due Date: December 7, noon

Introduction

In this assignment, we'll be diving deeper into more advanced architectures of neural networks, which have helped in the field of summarization. There are two types of summarization: extractive and abstractive. Extractive summarization entails concatenating extracts from a sentence/corpus to form a summary, whereas abstractive summarization involves paraphrasing from the sentences/corpus. This assignment focuses on **abstractive summarization**.

The task of summarization is a problem in a space of much more general problems called **sequence to sequence** problems. In a sequence to sequence problem, we take in an input sequence $x_1, x_2, x_3, \ldots x_n$ and produce an output sequence $y_1, y_2, y_3, \ldots y_m$ (as opposed to, for example, producing a single label as in sentiment analysis).

For summarization, $x_1, x_2, x_3, \ldots x_n$ are the words of the article and $y_1, y_2, y_3, \ldots y_m$ are words of summary. (Typically m is much less n in this case.) Another instance of a sequence to sequence problem is neural machine translation, in which $x_1, x_2, x_3, \ldots x_n$ are words in the input language and $y_1, y_2, y_3, \ldots y_m$ are words in the output language.

A special framework of Deep Learning models called **encoder-decoder networks** is used for tackling these problems. An encoder converts the input sentence into an "encoded state"; the decoder then takes as input the encoded information and, at each time step, picks the word y which maximizes $argmax(P(y|x,h))$ where x is the sequence seen so far and h is the encoded state. We highly recommend reading this paper to understand a general encoder-decoder framework.

The implementation part of the assignment includes (All coding-specific instructions are present in the Jupyter Notebook which you will find under Assignments on courseworks):

1. Developing a unidirectional LSTM based encoder-decoder network for title generation corresponding to each input sentence.
2. Improvement over the previous architecture by using attention

Before looking into the coding specifics, it is essential to understand the architecture of LSTMs. Over the years, there have been various excellent blog posts/primers which give an in-depth overview of LSTMs and their usefulness. We recommend taking a look at the following:

1. Understanding LSTM Networks by Christopher Olah
2. Exploring LSTMs by Edwin Chen
3. Beginner's guide to Recurrent Networks and LSTMs
4. Neural Network Models for NLP by Yaov Goldberg

Tips:

Training will take quite a long time (i.e., hours), and then you may need to debug and re-train. Thus, it would make sense to train on a small set of examples first to do your debugging. For instance, if your system is working, you should be able to perfectly fit your training data if you use only a dozen examples (i.e., accuracy on the training set should be 100%), which should run fairly quickly. We recommend using batch sizes of around 500 - 600 when you train on the whole dataset to ensure that training time is not very long.

Also, we **STRONGLY** recommend that you start early. You have three weeks for the project. Think about coding up your model in the first week, debugging in the second week, and analyzing your output and writing up your ideas for how to address problems in the third week.

Implementation:

1. Unidirectional LSTM Encoder-Decoder

The first model which you have to implement is a unidirectional LSTM encoder-decoder. Step by step instructions on vectorizing the input are provided in the Jupyter notebook. Follow the instructions to prepare the input, and then create the model. Once the model has been created and trained, use it to predict titles on the test data. Evaluate the predicted titles using the ROUGE score and report it. For specifics on ROUGE grading, please refer to the "Evaluation" section below.

2. Tensorboard visualizations

Tensorflow provides information on validation accuracy, loss, etc., using a tool called tensorboard. You can use tensorboard by simply typing the command "`tensorboard`" on the command line. Attach the plots of loss and accuracy from the tensorboard display in your notebook.

3. Unidirectional LSTM Encoder Decoder with Attention

In the encoder decoder approach, the output is only learned from the final state of the encoder. However, we can improve accuracy by giving a weighted sum of all the states to the decoder. This approach is called **attention**. You can read more about it here. An excellent visualization of attention is available at this blog post. You need to implement an attention-based approach for the encoder-decoder model and report ROUGE score improvements. You are provided a custom keras layer for attention and you should read about it in the Jupyter notebook. For specifics on ROUGE grading, please refer to the "Evaluation" section below.

4. Unidirectional LSTM Encoder-Decoder with Attention and Beam Search (30 points Extra Credit)

So far, these models use a Greedy approach to predict the output. A Greedy approach, however, is not optimal. The Greedy way of computing the maximum likely sequence is computationally expensive (This is similar to the Viterbi Algorithm). We can improve the speed and score by implementing beam search with a finite beam width and try to get a better estimate of the output sequence. For extra credit, implement beam search on the decoder (you choose how it is implemented, what the beam size is, etc.).

Dataset

The data for this assignment is available on CourseWorks under Assignments. Note that this data is only for use by the class. It should not be shared. Your dataset consists of the following files:

Training data:

train_article.txt - This file contains train data sentences, where each sentence is the first line of an article.
train_title.txt - This file contains the train title/summary corresponding to the article sentences in above file.

Test data:

test_article.txt - This file contains test data sentences, where each sentence is the first line of an article.
test_title.txt - This file contains the test title/summary corresponding to the article sentences in above file.

Evaluation:

You will be evaluating your summaries using their ROUGE scores. The grading will be relative based on the highest Rouge-1 and Rouge-2 scores achieved by your classmates. For each model, you will receive 15 points for implementing a working model and will be allocated 10 points on your Rouge score relative to your classmates. Therefore, we recommend you try to tune the model hyperparameters, use cross-validation etc., to achieve a higher score; you may experiment with things like bidirectional LSTMs as well. You should only submit one model without attention and one with attention. Your scores are evaluated based on these models, so submit only your best.

Note that your model will not actually be optimizing ROUGE, but will be optimizing a loss function like cross entropy. ROUGE is an evaluation applied to the finished model. When training, therefore, make sure to track your cross entropy and see that it is decreasing over time.

Analysis:

Select 15 article/summary pairs from your final attention model for an error analysis. Choose 5 that you think are good and 5 that you think are bad, with the remaining 5 selected randomly. Create a table where you show system output for each of the systems you developed (i.e., the plain LSTM, LSTM with attention, and also beam search if you performed it) along with the input and the gold summary in the test data. You may model this on the table used in the paper https://arxiv.org/pdf/1509.00685.pdf in Section 8, Results and shown on the last page before references.

For the same 15 examples, visualize the attention weights and include the diagrams in your output; an example diagram can be found on the first page of the above paper. Discuss how the attention weights show whether the system is working for these examples (4-5 sentences, not including images or captions).

Identify the problems that your system has the most trouble with. These may include repetition of words, ungrammaticality, syntactic problems such as attachment which make the resulting summary syntactically fluent but semantically incorrect, reference issues (e.g., using pronouns that are not resolved or NP references that are incorrect), wrong phrases dropped from the sentence, improper handling of proper nouns, etc. Also note how much abstraction vs. extraction your system does. Is your system making lexical substitutions, generalizing, compressing? Compare your different system outputs to reveal the pros and cons of each.

Select one of the problems that you identified and provide a proposal for what you would do to correct the problem. Suppose you were to either augment this model or re-do this in a traditional supervised approach perhaps using parses of the input or pos tagging. For example, your augmentation may modify the output of the LSTM to remove additional clauses. How would you approach this and what features would you use? Be creative. **Provide a 1-3 page description**.

Grading scheme (100 points, 30 extra credit points):

Preparing Input	5
Unidirectional LSTM encoder decoder	10
Tensorboard Visualization	5
Unidirectional LSTM Encoder Decoder With Attention	15
Perplexity reasoning (see Jupyter notebook)	5
Relative grade for rouge results (10 for without attention and 10 for decoder with attention)	20
Analysis	40 total
Selection of sentences and analysis of errors, comparisons (discuss all 15 sentences)	15
Visualizations (15 sentences) and description	10
Proposal for addressing error	15
Beam Search (Extra credit)	30