Homework 4: Sequence to Sequence Modeling (100 points)

Kathleen McKeown, Spring 2021
COMS W4705: Natural Language Processing

Due 04/14/21 at 11:59pm

The aim of this assignment is to provide you with an understanding of how encoder-decoder sequence to sequence models work, give you practical experience training an encoder-decoder model, and show you problems and open areas of research in this domain.

In the first part of the assignment, you will write code and train a encoder-decoder model on a restaurant description dataset. This will be done in a jupyter notebook, and the training *must* be done on a GPU. In the second part, you will do an error analysis and answer questions about open problems and how the model could be improved.

General instructions

Please post all clarification questions about this homework on EdStem under the "hw4" folder. You may post your question privately to the instructors if you wish. If your question includes a partial solution, please post it privately to the **instructors only.**

This assignment is done in pairs. You and your partner may discuss the assignment freely, but you should not discuss the answers with other students. All code and written answers must be entirely your own and your partner's. We will provide a sign-up sheet where you can either indicate who you would like to be paired with or ask us to pair you with someone else in the class. You must report how you divided up the work (see first question of the writeup section).

Late policy: You may use late days on this assignment. You can count late days from either partner (e.g., if you both want to use 1 late day, then you have 2 late days total). The assignment closes one week after the deadline. You cannot submit after that. However, you MUST include at the top of your submission how many late days you are using. If you don't tell us or have used all your late days, 10% per late day will be deducted from the homework grade.

If you have not already confirmed you can run code on the GPU, budget extra time for this. Getting set up on VMs, especially with GPUs, is notoriously

finicky. Give yourself extra time to work out any problems you may encounter. Additionally, be sure to **shut down your VM instance when not in use**, so you don't run out of credits.

1 Coding Portion

Please download the code from courseworks. This code includes one jupyter notebook, and a folder of data. The instructions for the coding portion are included in more detail in the jupyter notebook itself.

You will need to do some set-up to be able to use jupyter notebooks on the Google Cloud VM. This tutorial gives a good overview of how to do so. If you have set up a VM with GPUs for a past homework assignment, you should be able to use this machine. If not, you will first need to set up a new machine with GPUs. (Please follow the instructions provided in HW3 in order to set up your VM)

Google Cloud VM Settings

Once you have your VM, you will need to **change two settings**. The **first** is giving your machine a static external IP address. On the Google Cloud Platform, click on the menu icon in the top left, then 'VPC network', then 'External IP addresses'. For your machine, change the type from ephemeral to static. The **second** is to change the firewall settings. Similarly under 'VPC network' you can find 'Firewall rules'. Add a rule that allows the following specified protocols and ports: `tcp:5000`. This will allow you access a jupyter notebook on port 5000.

Set Up Jupyter

Install jupyter and check if you have a config file:

```
ls ~/.jupyter/jupyter_notebook_config.py
```

If not create a config file:

```
jupyter notebook --generate-config
```

Then, add the following lines to this config file:

```
c = get_config()
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 5000
```

Now you should be able to launch jupyter notebooks from the command line of your VM with this command:

```
jupyter-notebook --no-browser --port=5000
```

To access the notebook, go to the following URL:

```
http://<External Static IP Address>:5000
```

Finally–**don't forget to shut down your VM when not in use.**

1.1 Modify the code to run with the E2E dataset

As is, the code imports a number reversal dataset, which is a simple dataset of number sequences and those numbers in reverse. This is a dummy dataset designed to show that the model and training procedure work correctly. You should be able to run the entire notebook as is, and a model will learn to reverse short sequences of numbers.

Information about the E2E dataset is provided in the notebook. It is a dataset of restaurant meaning representations and associated sentences describing the restaurant. You should read about the dataset, and write code to load it in correctly.

1.2 Train a model on this dataset

Once you have imported the E2E dataset, you must train the model on a GPU. You will not need to make any changes to the model, but you are required to change the training procedure to work in batched mode. You will be required to submit a trained model (encoder and decoder). This model should pass an accuracy threshold, determined by a BLEU evaluation. This threshold should not be difficult to achieve; it simply checks that you imported and trained your model successfully.

The threshold is 0.4 **on the development data.** That is, if you calculate the BLEU score for every sentence in the development data (using greedy decoding), the average BLEU score should be above 0.4. This threshold should not be difficult to achieve. See the note below about how to correctly calculate BLEU scores.

1.3 Implement a beam search decoder

We provide an evaluation function that performs greedy decoding on any input sequence provided. You must write a new evaluation function that implements beam search for an arbitrary beam size. There are many resources available to learn more about beam search, we provide one to you here: (https://medium.com/@dhartidhami/beam-search-in-seq2seq-model-7606d55b21a5).

In greedy decoding, at each decoding step the most likely word is selected, resulting in one decoded sequence output. In beam search, at each decoding step the top k most likely sequences are selected. Each of these k sequences is then used to generate the next step; the top k next words per sequence are considered (for a total of $k * k$ sequences) and the top k sequences are selected to take to the next decoding step. At the end, you have k decoded sequence outputs.

1.4 Implement a nucleus sampling decoder

Rather than selecting the top k tokens at each decoding step, we could also sample from the token distribution at each step . This allows us to incorporate some randomness into the decoding process. Nucleus sampling [Holtzman et al., 2019] is an example of a sampling-based decoding strategy, where at each decoding step we only consider the smallest subset of the vocabulary tokens whose cumulative probability mass reaches a predefined threshold p (this is called the nucleus of the distribution). Note that the

probability mass of the truncated tokens is redistributed to the selected tokens. You are required to implement a nucleus sampling decoding function that returns k sampled sequences along with their associated probabilities.

1.5 Implement a BLEU evaluation

Although there exist python libraries that will calculate BLEU for you, you are required to implement your own. This will make you familiar with exactly how it works, which will be useful for understanding how well it measures performance.

BLEU scores require a list of candidate correct outputs. If you look at the development data, you will see that there are multiple data points with the same meaning representation (i.e. input) and different output sentences. When calculating your BLEU score, you must collect all the possible correct outputs.

To understand how BLEU is calculated, please rely on the original BLEU paper, BLEU: a Method for Automatic Evaluation of Machine Translation for how they calculate *sentence level* BLEU scores. The description in Natural Language Processing by Jacob Eisenstein may also be useful.

Additionally, we outline it for you here below. At a high level, BLEU is calculated as:

$$\text{BLEU-N} = \text{BrevityPenalty} * \exp(\sum_{n=1}^{N} w_n \log p_n)$$

BLEU-N is a weighted sum (w_n) of modified precision scores (p_n) for a set of n-grams (n). That is, BLEU-4 is the weighted sum of the modified precision scores for unigrams, bigrams, 3-grams, and 4-grams. A geometric mean is used; in the equation above this amounts to $w_n = 1/N$.

Additionally, a brevity penalty is applied. This brevity penalty is calculated using the length of the output (often called 'candidate') sentence and the length of the reference sentence that has the closest length to the candidate sentence. If c is the length of the candidate sentence and r is the length of the closest reference sentence, then:

$$\text{BrevityPenalty} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases}$$

So how do we calculate p_n, the modified precision? For each unique ngram in the candidate, count the maximum number of times it occurs in a reference sentence. Then clip this number by the maximum number of times it occurs in the candidate. That is, take the minimum of (max # occurrences in ref) and (# occurrences in candidate).

Sum these across all unique ngrams in the candidate, and divide by the total number of ngrams in the candidate. Sometimes p_n is 0 for higher order ngrams. In this case, back-off and use a lower N. If it is 0 for all ngrams, report BLEU as 0.

Please calculate BLEU-4 for this homework assignment.

2 Written Portion

2.1 Group work description

Briefly list the contributions of each group member for this assignment.

2.2 Error analysis

Describe **three** kinds of errors the system makes, with examples (input and output) from the development dataset. You may use greedy decoding for this question. Describe each error in at least 2 sentences.

2.3 Beam search analysis

(a) Show your BLEU scores on the development set for greedy decoding and beam search with different beam sizes (5, 10, 15, 20). For beam search, use the top scoring sequence (i.e. the one with the highest probability) to calculate the BLEU score for that datapoint. Discuss the differences in BLEU score for greedy and beam search decoding.

(b) Provide and discuss three examples of beam search. That is, for three input sequences, show the input sequence and multiple outputs from the beam search (use beam size=5).

   i. How do the multiple outputs differ?

   ii. Will beam search always include the greedily decoded output? (Why or why not?)

   iii. What is one advantage and one disadvantage of beam search, compared to greedy decoding?

2.4 Nucleus sampling analysis

(a) Show your BLEU score on the development set using nucleus sampling for decoding (use p=0.95), and compare it with greedy decoding and beam search (beam size=5). Discuss the differences.

(b) How does the average sequence probability compare to greedy decoding and beam search? Give an explanation for differences observed.

(c) Use nucleus sampling with different values of p (0.1, 0.5, 0.95) to generate 5 sequences per sample. You can randomly sample 5 examples from the development set for this comparison.

   i. How does the value of p affect the quality of the generated outputs?

   ii. How does the value of p affect the average number of tokens in the nucleus?

(d) Use nucleus sampling (p=0.95) to generate 5 sequences per sample and compare the output with beam search (beam size=5). You can randomly sample 5 examples from the development set for this comparison.

   i. How does the quality of the sequences generated using nucleus sampling compare to those generated by beam search?

   ii. What is one advantage and one disadvantage of nucleus sampling, compared to beam search?

   iii. When would you you prefer nucleus sampling? When would you prefer beam search?

2.5 Beam search with nucleus sampling

One natural extension would be to combine beam search and nucleus sampling, so that we can get the benefits of both approaches.

(a) Write pseudocode for a modified beam search where rather than considering the top k next words per sequence, you sample k words from the nucleus of the distribution, at each decoding step.

(b) What's wrong with combining them in this way?

(c) Now write a modified version of the pseudocode you wrote above, that fixes this issue. Feel free to cite existing work that may help you address the problem, but you may not copy the solution. You should be able to explain it in your own words, and write your own psuedocode.

2.6 What's wrong with BLEU?

Although several new metrics have been proposed, BLEU remains a common evaluation metric used to evaluate models.

(a) What are some problems with BLEU as a metric? List two and explain why these are a problem.

(b) What are the benefits of BLEU as a metric?

2.7 How could we deal with unseen restaurant names?

Try generating a sentence with a restaurant name that the model has never seen before. Show the result. How could you change the model to deal with this? Be specific enough that one could try to implement this. (You will not actually implement this, so it could be quite complicated. As long as it's within the realm of possibility, dream big.)

2.8 Paper analysis

You read two papers that discussed the problem of hallucination of neural models in abstractive summarization (https://arxiv.org/pdf/1509.00685.pdf) and for language generation (https://arxiv.org/abs/1911.03373). Compare the methods proposed in each paper to alleviate hallucination by: 1) providing a brief, paragraph-length description of the method, 2) discuss evidence on how well it works as indicated in the paper and 3) providing your opinion on whether it can improve your model's generations and why.

3 Submission Instructions & Grading

You must submit the following:

1. A compressed file named 'hw4_your-uni.zip' uploaded to Courseworks. This should contain one jupyter notebook, one encoder model, and one decoder model. The jupyter notebook must have the outputs as specified in the notebook. The models must be trained on a GPU. The names of these files must be:

   HW4.ipynb encoder.mdl decoder.mdl

2. A pdf with your answers to the written portion on Gradescope.

3.1 Coding Portion (44 points)

- Model that runs on GPU in batched mode and passes threshold (4 points)
- Beam search implementation (15 points)
- Nucleus sampling implementation (15 points)
- BLEU evaluator (10 points)

3.2 Written Portion (56 points)

- Q1: Group Work (1)
- Q2: Error analysis (6 points)
- Q3: Beam search analysis (10 points)
- Q4: Nucleus sampling analysis (10 points)
- Q5: Nucleus sampling with beam search (10 points)
- Q6: Problems with BLEU (4 points)
- Q7: Unseen restaurant names (5 points)
- Q8: Paper Analysis (10)

References

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *CoRR*, abs/1904.09751, 2019. URL http://arxiv.org/abs/1904.09751.