Homework 4: Open Domain Dialogue Generation

Due: 24th May 2017 at 11:59 PM PST

Overview

The goal of this homework is to give you a basic sense of how recent open-domain dialogue generation algorithms work. You will experiment with different response generation algorithms, including an IR (information retrieval) based extractive generation model and an abstractive response generation model using neural networks. You will also get to work on designing metrics for automatic dialogue evaluation.

We encourage you to start early, and come to the TA office hours for help if needed. Have fun!

Starter Code To obtain the starter code, run

```
scp -r <SUNet ID>@corn.stanford.edu:/afs/.ir/class/cs224s/hw/hw4 .
or if you're on corn: cp -r /afs/.ir/class/cs224s/hw/hw4 .
```

Problem Sketch

The problem of open-domain dialogue generation can be formalized as follows: given dialogue history[1] as input, a system needs to needs to produce a sentence that is coherent and relevant to that input. For example, given the dialogue history *How old are you?*, an appropriate response is *"I am 16."*.

We will use BLEU scores [1] to evaluate the performance of our system. To simplify the calculation of BLEU scores, you can run the following script:

```
python bleu.py reference_file < candidate_file
```

where each line in candidate_file consists of a source input (dialogue history) and the response generated by your system, separated by a |, and each line in reference_file consists **the same** source input and the corresponding response generated by humans. A sample line for both files is:

```
how old are you | I don't know .
```

Note that lines in reference_file and candidate_file with the same line number should have the **same** input.

Run the script using the given files:

```
python bleu.py data/dev.txt < data/example.txt
```

After running the script, you should be able to get a BLEU score of 0.99.

[1] In this assignment, dialogue history, source, input, source input, and message all refer to the dialogue history input to your system. That is, they are all synonymous. Also, output, response and target are all used interchangeably. Moreover, for this assignment, we consider a simple case where we approximate dialogue history using only the latest dialogue utterance.

1 The IR based model

You are given a big pool of source-target pairs $P = \{s_i, t_i\}$ stored in the file *data/pool.txt*. Given a new input s, your goal is to select a source-target pair (s', t') from P, and directly copy t' as a response to the input s. The task is to design an algorithm to best select a source-target pair from the pool P.

You will find two files in the folder: *dev.txt* (development set) and *test.txt* (test set), each line of which consists of a source (dialogue history) and a target (human-generated response) separated by a |. For each dialogue history in *test.txt*, pick a response from *pool.txt* and store your input-output pair in *test_decode.txt*. Then run the evaluation code:

python bleu.py data/test.txt < data/test_decode.txt

You can use dev.txt to tune your model. But you **should not** tune your model on test.txt. You are permitted to use any algorithm of your choice.

Hint on Model Design: The basic idea is to find the most relevant source in P to your new input s, and then you use the response to s', which is t', as your response to s. There are many ways to compute the relevance between two natural language utterances: the simplest is counting the number of overlapping words and normalizing by sentence length; more sophisticated method is using tf-idf[2]; or using cosine similarity between two sentence vector representations as a proxy.[3]

We have provided starter code in *q1.py*. You should only need to change function *SelectOutput*, which takes as input a new-input, source_list and target_list in *pool.txt*, and returns a selected output. Write a report on how your algorithm works, and report the BLEU scores on both the development set and the test set. You should get a BLEU score higher than 0.52 on the dev set.

2 Neural Generation Models for Dialogue Generation

Neural sequence-to-sequence models (seq2seq) [3] have been widely used in dialogue generation. Please refer to slides from CS224n `http://web.stanford.edu/class/cs224n/lectures/cs224n-2017-lecture10.pdf` for more details about seq2seq models. At train time, a model is trained on a large corpus of source-target pairs by sequentially predicting each token given the previous tokens. During test time, given a source message input s, the model generates (decodes) the most probable response t using the trained model. Decoding can be treated as a combinatorial search problem where standard heuristic search algorithms can be used, an example being beam search. Because training sequence-to-sequence models is very time-intensive (it typically takes weeks to train), you will not be asked to train the models in this problem; rather, you will focus on developing the decoding step.

Your instructions are as follows. For each input s, you are given a big list of response candidates generated by a pre-trained sequence-to-sequence model. This list is known as an *N-best* list. Each candidate is associated with a few features, for example, candidate length, log p(t|s) (the log probability of a target t given the source s), log p(s|t) (the log probability of a source s given the target t), and you are encouraged to design additional features that you think are useful. Your job is to design a score function that scores each candidate using these features.

[2] `https://en.wikipedia.org/wiki/Tf-idf`. You can treat each sentence as a document when computing idf.

[3] the representation of a sentence can be obtained by averaging the representations of its constituent words. You can download Glove [2] word representations here; `http://cs.stanford.edu/~bdlijiwei/word_vector.tar`.

Given an input, you can pick an utterance from the N-best list and use it as the response using this score function. This process is known as *reranking*.

The N-best list files are stored in *data/dev_N_best* and *data/test_N_best*. Each line in the file is as follows:

source index $ candidate length $ logp(t|s)$ logp(s|t) $ source $ candidate

Use the code in *q2.py*. You should only need to modify the function *computeScore*. While a simple algorithm could return just log p(t|s), your task is to design a more meaningful score function than leads to better BLEU scores. You should at least get a BLEU score of 1.10 on the dev set. Again, you **should not** tune your model on test.txt.

Write a short report on how you design your function, weight values associated with different features, and explain why they lead to better BLEU scores.

3 Adversarial Evaluation

Background: So far, we have been using BLEU for evaluation. BLEU computes the amount of word-overlap between the machine-generated response and the ground-truth response. This word-overlapping can to some extent measures the quality of the proposed response, but an observant researcher can easily spot the fundamental flaw with BLEU: there are way more than a single way to respond to an input, and computing the word-overlapping between **one** proposed response and the the ground-truth response is both not enough and unreliable.

Many alternative evaluation metrics have been proposed, one of which is *adversarial evaluation*. The basic idea of the *adversarial evaluation* draws intuitions from the Turing Test. In the Turing test, a machine is said to be able pass the test if it can fool a human evaluator into believing that it (the machine) is a human, based on the conversational responses generated from the machine. Drawing intuitions from the Turing test, we can say that a machine-generated response is of higher quality if it is able to fooling an evaluator into believing that it is generated from a human, or in other words, a machine-generated response is indistinguishable from a human generated response.

Everything sounds good so far, but another issue emerges: who is the evaluator? One direct choice is to ask humans (e.g., Turkers) to do the evaluations, but that might be too costly, time-consuming, and hard to scale up. Another option is, how about training a machine to be an evaluator, where we train a machine-learning classifier to distinguish between machine-generated responses and human-generated responses. We call such a strategy **adversarial evaluation**. Such a strategy also relates to the idea of adversarial training in image generation [4].

Adversarial evaluation involves both training and testing. At training time, the evaluator is trained to label dialogues as machine-generated (negative) or human-generated (positive). **The evaluator is thus a binary classifier**. At test time, the trained evaluator is evaluated on a heldout dataset. If the human-generated dialogues and machine-generated ones are indistinguishable, the model will achieve 50 percent accuracy at test time.

We define **Adversarial Success** (AdverSuc for short) to be the fraction of instances in which a model is capable of fooling the evaluator. Thus, AdverSuc = 1 - Evaluator Accuracy. Higher values of AdverSuc for a dialogue generation model on the test set are generally better, however, we will explore this in more detail through the question.

Process: You are given three files, *data/adver.train*, *data/adver.dev* and *data/adver.test*, which respectively correspond to the training, development and test file. Each line in these files take the following forms:

label | source | target

where label denotes whether the target is generated by a human or a machine (1 indicating being generated by a human and -1 by a machine).

You can either use the code provided in *q3.py*, or rewrite your own code to train your evaluator. If you plan to use *q3.py*, you need to implement a feature extractor in function *feature_extractor*. Your extracted features need to be outputted to the file *feature_train.txt*, *feature_dev.txt* and *feature_test.txt*. Each line outputted to *feature_train.txt* should be of the following form:

label feature1:value feature2:value feature3:value ...

For example,

```
1 1:3  5:1.5  7:3.13
-1 1:2  4:1.1  8:5
```

this means you have two examples, the first instance is a human-generated response (since label is 1). The first instance consists of three features, with index 1, 5, 7, the values of which are respectively 3, 1.5, 3.13. Similarly, the second instance is a machine generated response, and consists of three features with index 1, 4, 8 with value of 2, 1.1, 5. **You can extract whichever feature you want, for example, unigrams within responses**. Within each line, the indexes of features should be in ascending order. Therefore, you will encounter an error if your features are as follows: 1 5 : 3 1 : 1.5 since your feature indexes are not in ascending order. **Feature index starts from 1, not 0**. File *data/feature_example.txt* provides an example of what your feature file should be like.

When you are done with feature extracting, you can train your classifier using the svm-light package. Navigate to the directory svm_light and run the following script:

```
cd svm_light
./svm_learn ../data/feature_train.txt model
```

After you are done with training, test your model on your development set and test set in the following way:

```
./svm_classify ../data/feature_dev.txt model output
```

Report Adversarial Success on both dev and test sets.

You are also encouraged to design your own models, using whichever architecture you like (e.g., neural net models), in which case you don't need to use the svm package. If that is the case, please submit your code for submission.

Apart from the AdverSuc results on the dev and test sets, answer the following questions:

1. Other than the fact that your response generation system is able to produce responses that are indistinguishable from human generated responses, what is the other reason that might lead to a high AdverSuc (hint, what if your evaluator is bad)? What is a simple way to get a high AdverSuc score on the test set ? For a balanced dataset (equal number of 1s and -1s), what would you expect the result to be?

2. If you get a low AdverSuc score on the test set, what does that say about your dialogue system? Give a 2-sentence explanation.

3. Write a few sentences about the potential drawbacks of *adversarial evaluation*.

References

[1] Papineni et al., BLEU: a method for automatic evaluation of machine translation. *ACL* 2002.

[2] Pennington et al., Pennington, Jeffrey and Socher, Richard and Manning, Christopher D. *EMNLP*, 2014.

[3] Sutskever et al., Sequence to Sequence Learning with Neural Networks. *NIPS*, 2014.

[4] Goodfellow et al., Generative adversarial nets. *NIPS*, 2014.