Natural Language Processing
Project 2: Dialog System

Problem Description

Dialog systems have been all the rage. Everyone seems to want a chatbot. You are given the starter code of a basic chatbot that uses a sequence to sequence model with an attention decoder. Your job is to improve on the chatbot to make it sound as human as possible.

Traditionally, chatbots have been rule-based. As recent as 2014, Siri and Google Now still relied on handcrafted rules to find the most relevant answers. But deep learning techniques and powerful computers have opened up new possibilities that we are still exploring.

The model

The chatbot is based on the translate model on the TensorFlow (1.14) repository, with some modification to make it work for a chatbot. It's a sequence to sequence model with attention decoder. If you don't know what a sequence to sequence model is, please see the lecture 10 on Canvas related to Dialog systems. The encoder is a single utterance, and the decoder is the response to that utterance. An utterance could be a sentence, more than a sentence, or even less than a sentence, anything people say in a conversation!

The chatbot is built using a wrapper function for the sequence to sequence model with bucketing. The loss function we use is sampled_softmax.

```
self.outputs, self.losses = tf.contrib.legacy_seq2seq.model_with_buckets(
                                        self.encoder_inputs,
                                        self.decoder_inputs,
                                        self.targets,
                                        self.decoder_masks,
                                        config.BUCKETS,
                                        lambda x, y: _seq2seq_f(x, y, True),
                                        softmax_loss_function=self.softmax_loss_function)

                                        lambda x, y: _seq2seq_f(x, y, True),
                                        softmax_loss_function=self.softmax_loss_function)
```

The _seq2seq_f is defined as:

```
def _seq2seq_f(encoder_inputs, decoder_inputs, do_decode):
            return tf.nn.seq2seq.embedding_attention_seq2seq(
                    encoder_inputs, decoder_inputs, self.cell,
                    num_encoder_symbols=config.ENC_VOCAB,
                    num_decoder_symbols=config.DEC_VOCAB,
                    embedding_size=config.HIDDEN_SIZE,
                    output_projection=self.output_projection,
```

```
                    feed_previous=do_decode)
```

By default, do_decode is set to be True, which means that during training, we'll feed in the previously predicted token to help predicting the next token in the decoder even if the token was the wrong prediction. This helps approximate the training to be closer to the real environment when the chatbot has to make the prediction for the entire decoder from solely the encoder inputs.

And the softmax_loss_function is the sampled softmax to approximate the softmax.

```
def sampled_loss(inputs, labels):
            labels = tf.reshape(labels, [-1, 1])
            return tf.nn.sampled_softmax_loss(tf.transpose(w), b, inputs, labels,
                                              config.NUM_SAMPLES, config.DEC_VOCAB)
self.softmax_loss_function = sampled_loss
```

The outputs object returned by seq2seq.model_with_buckets or any pre-built seq2seq functions in TensorFlow is a list of decoder_size tensors, each of dimension 1 x decoder_vocab_size corresponding to the (more or less) probability distribution of the token at the decoder time step. I said more or less because it's not a real distribution -- the values in the tensor aren't limited to be between 0 and 1, and don't necessarily sum up to 1. However, the highest value still means the most likely token. For example, if your decoder size is 3 (which means the model should construct a decoder of 3 tokens), and your decoder vocabulary has a size of 4 corresponding to 10 tokens (a, b, c, d), then the outputs will be something like this:

```
self.outputs = [[2.3, 4.2, 3.0, 1.9], [-1.2, 0.1, 0.3, 2.0], [1.6, -1.8, 0.4, 0.5]]
```

To construct the response from an input, the starter code uses the greedy approach, which means it takes the most likely token at each time step. For example, given the outputs above, we'll get the b as the first token (corresponding to the value 4.2), d as the second token, and a as the third token. So the response will be 'b d a'.

This greedy approach works poorly, and restricts the chatbot to give one fixed answer to a input. You can improve this -- see **Your improvement** section.

Dataset

The bot comes with the script to do the pre-processing for the Cornell Movie-Dialogs Corpus, created by Cristian Danescu-Niculescu-Mizil and Lillian Lee at Cornell University. This is an extremely well-formatted dataset of dialogues from movies. It has 220,579 conversational exchanges between 10,292 pairs of movie characters, involving 9,035 characters from 617 movies with 304,713 total utterances.

The corpus comes together with the paper "Chameleons in Imagined Conversations: A new Approach to Understanding Coordination of Linguistic Style in Dialogs", which was featured on Nature.com. It is a fascinating paper that highlights several cognitive biases in conversations that will help you make your chatbot more realistic. I highly recommend that you read it.

The preprocessing is pretty basic.

- consider most of the punctuations as separate tokens.

- normalize all digits to '#'.
- lowercase everything.
- The dialogs have a lot of <u> and </u>, as well as [and], so you can get rid of those.

You're welcome to experiment with other ways to pre-process your data.

Sample conversations

The bot comes with the code that writes down all the conversations the bot has on the output_convo.txt in the processed folder. Some of the conversations are sassy, but some are also pretty creepy. Some make absolutely no sense at all.

Starter code

The starter code can be found on Canvas and the class GitHub repository.

In the folder **chatbot**, there are 4 main files:
model.py is where you specify the model and build the graph for your model.
data.py is the script to do all the data-related tasks, from separating the data into test set and train set, preprocessing the data, to making it ready to be fed to the model.
config.py contains configuration hyperparameters for the model.
chatbot.py is the main file that you'll run to train or to chat with your chatbot.

Please see README.md for instruction on how to run the starter code.

If you have problems in using the starter code you can use your own baseline but you need to implement the following improvements in order to complete the assignment. There is no mark on running or using a new baseline.

Your improvement

The starter bot's conversational ability is far from being satisfactory, and there are many ways you can improve the bot. You have the free range to use anything you want, even if you want to construct an entirely new architecture. Below are some of the improvements you can make.

To pass the class, **you will have implement the following improvements:**

1. Train on multiple datasets (20%)

Bots are only as good as their data. If you play around with the starter bot, you'll see that the chatbot can't really hold normal conversations such as "how are you?", "what do you want to for lunch?", or "bye", and it's prone to saying dramatic things like "what about the gun?", "you're in trouble", "you're in love". The bot also tends to answer with questions. This makes sense, since Hollywood screenwriters need dramatic details and questions to advance the plot. However, training on movie dialogues makes your bot sound like a dumb version of the Terminator.

To make the bot more realistic, you can try training your bot on other datasets. Here are some of the possible datasets:

Twitter chat log (courtesy of Marsan Ma)
More movie substitles (less clean)
Every publicly available Reddit comments (1TB of data!)
Your own conversations (chat logs, text messages, emails)

You'll have to do the pre-processing yourself. Once you've had the train.dec, train.enc, test.dec, and test.enc, you can just plug the current code in to make the data ready for the model. Please see the data.py file to have a better understanding of how this is done.

2. Make your chatbot remember information from the previous conversation (20%)

Right now, if I tell the bot my name and ask what my name is right after, the bot will be unable to answer. This makes sense since we only use the last previous utterance as the input to predict the response without incorporating any previous information, however, this is unacceptable in real life conversation.

```
> hi
hi . what ' s your name ?
> my name is chip
nice to meet you .
> what ' s my name ?
let ' s talk about something else .
```

What you can do is to save the previous conversations you have with that user and refer to them to extract information relevant to the current conversation. **This is not an easy task, but it's an exciting one.**

3. Create a chatbot with personality (30%)

Right now, the chatbot is trained on the responses from thousands of characters, so you can expect the responses are rather erratic. It also can't answer to simple questions about personal information like "what's your name?" or "where are you from?" because those

tokens are mostly unknown tokens due to the pre-processing phase that gets rid of rare words.

You can change this by using one of the two approaches (or another, this is a very open field).

Approach 1: At the decoder phase, inject consistent information about the bot such as name, age, hometown, current location, job.

Approach 2: Use the decoder inputs from one character only. For example: your own Sheldon Cooper bot!

There are also some [pretty good Quora answers to this.]

Michael (The Ofifce) Rude, abrupt Thinks he's the best boss 	Barney (HIMYM) Very bro-y Likes to hit on women Drinks a lot 	Sheldon (BBT) Socially awkward Thinks other people are stupid 	Joey (Friends) Not very smart but kind A player 
> hi **4_michael** have a seat . > who are you? **4_michael** i ' m just a friend . > what's your name? **4_michael** michael scott. > do you have a girlfriend? **4_michael** i have a lot . > tell me a joke **4_michael** no , i ' m not gonna tell you . > you are boring. **4_michael** you are . > where do you work? **4_michael** right here . > tell me michael. why do you work here? **4_michael** i thought i had to do something better . > bye bye michael	> who are you? **2_barney** i am the master of the most barney . > hi barney! **2_barney** hi , have a drink . > oh thanks. what drinks do you have? **2_barney** whatever what do you do ? > i'm a student. **2_barney** a . . . buddy . > yeah? **2_barney** sleep with her mom ? > lol no **2_barney** because she ' s not happy . > ha ha okay. how's work? **2_barney** okay . > just okay? **2_barney** oh , okay . > what do you think of that girl over there? **2_barney** do not even	> hi **1_sheldon** hello . > who are you? **1_sheldon** i ' m sheldon . > what's up? **1_sheldon** i see how i could do . > what can you do? **1_sheldon** i have to tell you a secret . > tell me. **1_sheldon** no . > what do you like to do for fun? **1_sheldon** i want to leave . > to go home? **1_sheldon** no . > how are you doing? **1_sheldon** i just wanted to sleep . > sleep! **1_sheldon** penny , i am getting too much for you .	> hey what's up **4_joey** hey ! > how are you? **4_joey** oh , oh , i ' m fine ! > what's your name **4_joey** joey , joey . > who are you? **4_joey** oh , joey ' s joey ! joey ? > joey joey joey **4_joey** what ? > tell me a joke. **4_joey** a woman . > a woman is not a joke! **4_joey** oh , i am very sorry .

4. Create a feedback loop that allows users to train your chatbot (30%)

That's right, you can create a feedback loop so that users can help the bot learn the right response -- treat the bot like a baby. So when the bot says something incorrect, users can say: "That's wrong. You should have said xyz" and the bot will correct its response to xyz.

5. Use character-level sequence to sequence model for the chatbot (optional)

Some of you built in project 1 (OCR Error Correction) a character-level language model and it seems to be working pretty well, so is there any chance a character-level sequence to sequence model will work?

An obvious advantage of this model is that it uses a much smaller vocabulary so we can use full softmax instead of sampled softmax, and there will be no unknown tokens! An obvious disadvantage is that the sequence will be much longer -- it'll be approximately 4 times longer than the token-level one.

6. An improvement of your choice (optional)

There is still a lot of room for improvement. Be creative!

Tips

1. Know the data

You should know your dataset very well so that you can do the suitable data preprocessing and to see the characteristics you can expect from this dataset.

2. Let your friends try the bot

You can learn a lot about how humans interact with bots when you let your friends try your bot, and you can use that information to make your bot more human-like.

3. Don't be afraid of handcrafted rules

Sometimes, you'll have to resort to handcrafted rules. For example, if the generated response is just empty, then instead of having the bot saying nothing, you can say something like: "I don't know what to say." or "I don't understand what you just said." or "Tell me about something else." This will make the conversation flows a lot more naturally.

4. Processing time!

It'll take a long time to train. For a batch of 64, it takes 1.2 - 2.2s/step on a GPU, and on a CPU it's about 4x slower with 3.8 - 7.5s/step. On a GPU, it'd take an hour to train an epoch for a train set of 100,000 samples, and you'd need to train for at least 3-4 epochs before your bot starts to make sense. Plan your time accordingly.

Submission

Expected Deliverables

You need to submit the following:
1. Your code and instructions on how to run it
2. Specify what dataset you use
3. output_convo.txt file **(not your training Data)**
4. Detailed description of what improvement you did for the bot

Zip everything and upload it on Canvas.
It is your responsibility to make sure you upload the correct file.

All files can be submitted with Canvas before **the 23rd of December 2020**. Please ensure to include **your name and student number** on the **Jupyter notebook document**.

Code

All code should be completed using Python as the programming language.
Your code should have a logical structure and a high level of readability and clarity. Please comment your code and put all code into functions. You code should be efficient and should avoid duplication.

Late submissions

If you don't get the assignments done to your satisfaction and don't meet the minimum requirements by the deadline, you have the option (as with any assignment at CIT) of submitting up to 1 week late for a penalty of 10%.

This penalty is subtractive. Work that would have earned 55% if on time, would get 45% (not 49.5%) if late.

The penalty is applied weekly. So, 1 day late costs the same 6. If you want to take that option please let me know. Otherwise, I will just correct whatever I have.

If you have a specific reason for submitting a late assignment (sickness, etc) please contact me directly or submit a medical certificate in the department secretary.

Plagiarism

Please read and strictly adhere to the CIT Honesty, Plagiarism and Infringements Policy Related to Examinations and Assessments. Note that reports are **checked** against each other and against external web sources for plagiarism. Any suspected plagiarism will be treated seriously and may result in penalties and a zero grade.

Grading

The assignment is worth 50% of the overall mark for the module. Marks will be awarded based on the quality of the code and the results. In particular, I will be checking to see if you are handling and preprocessing data correctly, carrying out exploratory analysis to gain insights, correctly performing model implementation, and critically, documenting everything in a clear and concise way. The submitted code will also be checked to ensure that the work is your own.