Assignment 3: TensorBro - A TensorFlow chatbot
CS20 SI: TensorFlow for Deep Learning Research (cs20si.stanford.edu)
Due 3/17 at 3:00pm
Prepared by Chip Huyen (huyenn@stanford.edu)

If you've read any sort of tech-related magazines, you're probably aware of the chatbot craze recently. It's the new, hip, unsolved challenge. Traditionally, chatbots have been rule-based -- even chatbots like Siri, up until 2014, were still built on handcrafted rules. But deep learning and powerful computers have opened a new direction with a lot of potential that people are still exploring. In this assignment, you'll be building a chatbot that can (probably) compete with many state-of-the-art chatbots.

You're given the starter code of a basic chatbot that is built using the sequence to sequence model with an attention decoder. Your job is to improve on the chatbot to make it sound as human as possible. You're encouraged to work in pairs.

Note*: If you read this handout and find the starter code useful, please head over to Anonymous chatlog donation to help us create the first realistic dataset to train chatbots on.*

The model

The chatbot is based on the translate model on the TensorFlow 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 read the lecture notes on the course website. 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.

```
self.outputs, self.losses = tf.nn.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)
```

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 tf.nn.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 chatbot's response from the outputs, 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.

For more detailed explanation of the sequence to sequence model, attention, sampled softmax, and bucketing, please read the lecture slides for lecture 13. I'll post the lecture note on the class website this Sunday (March 05). Sorry for the tardiness.

The dataset
The bot comes with the script to do the pre-processing for the Cornell Movie-Dialogs Corpus, created by the wonderful Cristian Danescu-Niculescu-Mizil and Lillian Lee at, guess where, 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 is distributed 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 pre-processing is pretty basic. I consider most of the punctuations as separate tokens. I normalize all digits to '#'. I lowercase everything. I noticed that the dialogs have a lot of <u> and </u>, as well as [and], so I just 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.

Code description
The starter code can be found on the class GitHub repository.

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, pre-processing 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.

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 at least one of them and make it work decently. For information on how I evaluate "decently", please read the evaluation section below.

1. Train on multiple datasets

Bots can only talk as well as the data they are trained on, so datasets are pretty key. If you play around with the starter chatbot, you'll realize 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 dummy 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)

I'm also embarking on an ambitious project to build the first realistic dialog dataset. Please head over to Anonymous Chatlog Donation to see how you can help advance research in this field!

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. Use more than just one utterance as the encoder

For the chatbot, the encoder is the last singular utterance, and the decoder is the response to that. You can see that this is problematic because conversations often go on for more than 2 utterances and you have to rely on the previous utterances to construct an appropriate response.

You can modify the model to be able to use more than one utterance as the encoder input. This will make your model more like a summarization model in which your encoder is longer than your decoder.

To do this, you'll have to modify the bucket lengths and some of the data processing code.

3. Make your chatbot remember information from the previous conversation

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.

4. Create a chatbot with personality

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. I'm super excited about this direction and I'm working on a project dealing with this, so if you're interested in talking more about this, hit me up!

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

We've built 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. Construct the response in a non-greedy way

This greedy approach works poorly, and restricts the chatbot to give one fixed answer to a input. For example, if the user says "hi", the bot will always "hi" back, while in real life, people can vary their responses to "hey", "how are you?", or "hi. what's up?"

You can try to use viterbi or beam search to construct the most probable response.

7. Create a feedback loop that allows users to train your chatbot
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.

It can be dangerous because users are mean and can turn your chatbot into something utterly racist and sexist. Microsoft did that for their chatbot Tay and see what happened!

8. An improvement of your choice
There is still a lot of room for improvement. Be creative!

Evaluation

The problem is that there isn't any scientific method to measure the human-like quality of speech. The matter is made even more complicated when we have humans that talk like bots.

The loss we report is the approximate softmax loss, and it means absolutely nothing in term of conversations. For example, if you convert every token to <unk> and always construct response as a series of <unk> tokens, then your loss would be 0.

So we'll try something fun with this assignment. For the last day of class, Friday March 17, we will have a demonstration of chatbots. Each person/team will have 5 minutes, of which 2 minutes to introduce themselves and demo their chatbots, then the rest of the class can try play with their chatbots. The class will vote for their favorite chatbot!

Tips
1. Know thy data
You should know your dataset very well so that you can do the suitable data pre-processing and to see the characteristics you can expect from this dataset.

2. Adjust the learning rate
You should pay attention to the reported loss and adjust the learning rate accordingly. Please read the CS231N note on how to read your learning rate.

Keep in mind that each bucket has its own optimizer, so you can have different learning rates for different buckets. For example, buckets with a larger size might need a slightly larger learning rate.

You should feel free to experiment with other optimizers other than SGD.

3. 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.

4. 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.

5. Have fun!
This assignment is supposed to be fun. Don't get disheartened if your bot seems to just talk gibberish -- even famous bots made by companies with vast resources like Apple or Google give nonsensical responses most of the 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
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
4. Detailed description of what improvement you did for the bot
5. Demo at the class on Friday (March 17). It's going to be super fun!

Zip everything and send it to my email huyenn@stanford.edu.