Charlie Anderson, Leila Awad, Ammar Babar, Teddy Willard
CS 322
Professor Blake Howald

Research Question

We began this project interested in the creation of companionship, an artificial bond between an AI agent and a user. We conceived the idea of a chatbot designed to act flirtatious with the user, mimicking a classic bar conversation. Our main version of the chatbot builds semantic analysis into a narrative template, allowing the chatbot to flirt with the user in relatively human-like way. After analysing the successes and failures of this version, we attempted a second version that involves natural language generation. We named our chatbot 'Baebot' as a play on words with the term *bae* which is used romantically to indicate that one's partner comes *before anyone else*.

Background Info and Related Work

As part of our early research, we began looking for flirty bots that are available on the internet. All of them are highly limited, having poor understanding of user input and lackluster response ability. We were curious why exactly these were so poor when generic chatbots exist that exhibit higher ability and potential. We received our answer later on as we delved deeper into semantic analysis technology: there exists no public annotated corpus that specifically covers the language of flirting or seduction. The closest thing we could find was a speed dating corpus, owned by a professor at Stanford who was unable to share his work due to privacy agreements. Thus, we were forced to create a "flirty corpus" and a "neutral corpus" ourselves. Due to time constraints, both corpuses ended up very limited, which affected our bot.

Technologies that we looked into during the creation of Baebot include OpenDial, Tensorflow, Vader Sentiment analysis, and NLTK. We decided against using OpenDial because the framework was beyond the scope of our project. We started by using Vader and NLTK Sentiment Analysis tools, then explored Tensorflow capabilities near the very end of the project.

Methods

1. Template Methodology

Our methodology for the first iteration of BaeBot was to create a custom tree structure where each node of the tree is a template response from the bot. The root node of the tree contains the start of the narrative. The system prints the template response, accept the ensuing user input, and then runs two different sentiment analyses on that input. The first analysis interprets whether the user input is positive or negative using Vader. Based on the interpretation, the bot chooses the next narrative template in the progression (from the two child nodes in the narrative tree). The second analysis interprets whether or not the user input is flirty. This is calculated in binary fashion (flirty or not flirty) and the result is factored into to the 'flirty

weight'. The flirty weight is a score starting at zero to which .1 is added for every user input deemed "flirty" and -.1 is added for every user input deemed "not flirty." We determined flirty polarity by training the NLTK sentiment analyzer on two corpuses: a flirty corpus and a neutral corpus. Both were created by the authors of this paper[1]. At the end of the narrative, the system chooses a final output based on the overall flirtiness of the conversation. In theory, Baebot should choose whether the user 'succeeded' in winning it over with flirtiness or 'failed,' in which case the user is rejected.

2. Sequence to Sequence Methodology

The sequence to sequence learning model is an algorithm for supervised machine learning. We chose this algorithm because of its suitability for NLP related applications since most state-of-art translation, summarization and conversation softwares implement seq2seq learning. See Appendix I for more details on seq2seq models.

To set up our model, we used the following parameters:

1. Learning rate: 0.5
2. LSTM layers: 3
3. Vocab size: 20000
4. Layer size: 256 cells
5. Batch size: 64
6. Steps per checkpoint: 300
7. Gradient clip point: 5.0
8. Learning rate decay factor: 0.99

The built-in seq2seq model also had an attention mechanism, and used a greedy decoder and a softmax loss function. The corpus we trained the model on only had 101 lines for the encoder and the decoder each, and we used a 90/10 split for train and test sets. For each line in the encoder, there was a corresponding line in the decoder that served as its response. Our assumption when making these sets was that a flirty input would be followed by a flirty response.

Results

1. Template Results

Qualitatively, the majority of user input (tested by the authors of this paper and their friends) was deemed not flirty, leading to the bot to almost universally end the conversation with rejection. The positive and negative analysis appeared to function as intended. Our narrative

[1] See the 'Research Ethics' section of our Ethical Statement for more details on the creation of these corpuses.

template, however, was not adaptive enough handle unwritten shifts in conversation. No users were convinced that they could be talking to a human, as Baebot was unable to respond in a flexible manner. Our information capturing system (e.g. parsing a sentence and extracting a name) also showed poor results, often failing to capture any information, leading to sentences with illogical gaps.

2. Sequence to Sequence Results

For our seq2seq model, during our training, a steady decrease in perplexity was noted and the model was trained until the perplexity of 1.01 was achieved:

global step 7800 learning rate 0.5000 learning rate 0.5000 step-time 4.88 perplexity 2.38
  eval: bucket 0 perplexity 1485.79
  eval: bucket 1 perplexity 39780.77
  eval: bucket 2 perplexity 495801.86
  eval: bucket 3 perplexity 145694.74

global step 8100 learning rate 0.5000 step-time 4.52 perplexity 1.02
  eval: bucket 0 perplexity 2945.93
  eval: bucket 1 perplexity 209147.37
  eval: bucket 2 perplexity 4179311.04
  eval: bucket 3 perplexity 606397.75

global step 8400 learning rate 0.5000 step-time 4.48 perplexity 1.01

Despite the steady improvement in training perplexity, the perplexity of the development set[2] was seen to remain extremely high.

When we tested our model, the seq2seq model for our chatbot did not do so well. A sample conversation was:

[2] Note that we do not have our own dev set but are using a standard set that we acquired with the chatbot code.

```
> hello
_UNK a the Bond Bond Bond minutes .
> what?
_UNK like , well .
> well, that doesn't make any sense.
Why ? : )
> Finally!
what ?
> I think you're very pretty...in a boxy, kind of way
This a secret .
> Alright, I'm going to bed. night night
Why do you get to go ?
> Because I want to
_UNK hanging me me .
>
```

Note that although we do not have a scientific metric for what qualifies as a flirtatious encounter, even a 'computer scientist' can conclude that the above conversation is a pretty bad attempt at courtship.

Analysis

1. Template Analysis

Much of Baebot's limited interpretation abilities can be attributed to the fact that we created the training corpuses ourselves. The size and robustness of our corpus is limited, leading to some odd variances in the interpretation. For example, we realized part way through that any sentence with an exclamation mark is considered flirty because the neutral corpus does not contain any such punctuation. Additionally, our bot almost always assumes inputs are not flirty unless they are tailored to be very similar to our flirty corpus. A more robust corpus would allow us to correctly analyze a wider range of sentences.

Our narrative template also has some issues, inherent to the nature of a template structure. During system tests, many users would write something that sounded like natural conversation but was not a complete or direct response to the system output, leading Baebot to respond in a completely unaware fashion. Simply put, our template was too stiff and too binary. Since Baebot leads the conversation and responds only to positive or negative input, any deviation outside that leads to unnatural interaction. We attempted to add in some more casual speech idiosynchracies by catching and responding to cases where the user might swear at the bot or say something inappropriate. We could take it a lot farther, though, simply by adding more edge cases and allow for further variation by just adding more template options (or increasing the possible response options from just positive or negative).

Our information collection had problems because we collected information by splitting the sentence and tagging the individual parts. Then, we recorded the expected user input. For example, we ask for the location where the user bought his/her shirt. The problem is that if the user types 'american eagle', rather than 'American Eagle', our system tagger interprets 'american eagle' as an adjective and a noun rather than as one proper noun. Similarly, if the user

writes ‘I got it at American Eagle or Old Navy’, our bot will record ‘American Eagle Old Navy’ because it concatenates all proper nouns. There are some immediate simple fixes, however. While we might miss some data, we could always just have an alternative template response that contains no information from the user if none is found. This would simply preserve the illusion of the narrative, though, than actually solve our issues. We also have no easy fix for the issue of multiple proper nouns; the best fix would be to have access to a semantic web so we could see what combination of proper nouns likely exists or not.

2. Sequence to Sequence Analysis

The flirt bot did not do well under our second method because our seq2seq model did not have enough data to train on. This was the reason why the bucket perplexity was high and our flirt bot gave nonsense responses. Usually, seq2seq models require to be trained on large corpuses to be able to generate comprehensible responses. For example, an English to German translator would require anywhere between 3 and 5 million lines, so naturally, our 90 lines of training data were quite insignificant for the task.

Looking past the scarcity of the corpus, this model can be improved in other ways. Instead of a greedy decoder, we could implement a beam searching decoder that picks three tokens with the highest probabilities at each step, and once all the beams have been calculated, picks the beam with the highest probability. Punctuation, and Out Of Vocabulary (OOV) words stored as _UNK tokens, could be split up into individual characters and the model could be trained on them. There’s work being done at Stanford that shows great promise using the aforementioned technique. Furthermore, noise from previous outputs of LSTM/RNN cell could be fed into the next cell to help prevent the model from derailing i.e. improving on bad output instances. There are many other optimizations that we can employ but the ones mentioned above would be a good start once we have a larger corpus to work with.

Conclusion

We are happy with our Baebot and its possible applications; however, both our narrative template version and our true generation version suffer massively from a lack of good data for flirty interaction versus neutral interaction. Thus, our sentiment analysis is limited in the first version, and our generated responses are incomprehensible in the second. As this type of technology becomes more refined, we hope that good data becomes more available. The potential for our project is quite high, but there is much work left to be done.

References:

1. https://github.com/llSourcell/tensorflow_chatbot
2. http://suriyadeepan.github.io/2016-06-28-easy-seq2seq/
3. https://www.youtube.com/watch?v=G5RY_SUJih4

Appendix I

A sequence to sequence model has an encoding(input) and a decoding(output) layer implemented using recurrent neural nets (RNNs) where each RNN cell feeds into the next one[3]. This way, we have a chain of RNNs, and training the model results in adjustments of weights and biases across each cell s.t. the loss function is minimized. The seq2seq algorithm is quite difficult to implement and optimize. Therefore, we decided to use the seq2seq model implemented in the TensorFlow library published by Google. Furthermore, we found existing software infrastructure for a chatbot implemented in TensorFlow that we used for our project purposes.

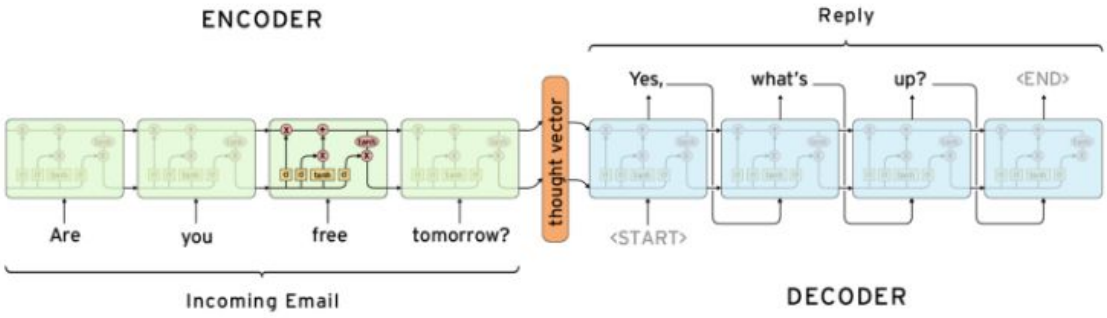

Image borrowed from Deep Learning for Chatbots : Part 1

[3] A seq2seq model can have more than 2 layers. In fact, a number of engineers are finding great success with up to 6 layers. Also, the model can use Long Short Term Memory (LSTM) network instead of an RNN which tensorflow actually does.