Generating personalized chat replies

Kristen Aw
jiayuaw@stanford.edu

1 Introduction

The goal of the project is to automate text messaging, a common but manual activity. NLP systems today are good at learning to talk conversationally, but need many days and much data, and have inconsistent personalities. There is no existing solution to cheaply and quickly automate the process of making input-relevant responses that take on the voice of individual personalities. A non-goal of this project is accounting for the user's objective in mind- so we do not take into account their relationship with the sender, personal history, interests or daily schedule.

Our solution takes as input text files where every line is one chat message to train on. Given a text file of new lines to evaluate, the models output predicted reply strings for each new input message. The two models evaluated were: firstly, one that is a combination of k-means clustering and Markov chains; secondly, one that does sequence-to-sequence (Seq2Seq) with attention, but trained mostly with a larger dataset from a different domain. Seq2Seq trained on both public and personal datasets did best.

2 Related Work

2.1 Hand-crafted chatbots

Traditionally, chatbots [12] have been hand-crafted using pattern matching rules and a biography. Chatbots [11] especially convince human judges of being actual humans by varying their replies and having strong personalities. Therefore, convincing chat replies should try to do both.

2.2 Word embeddings

Words can be vectorized in relation to one another [5]. Such embeddings (word2vec), are more powerful than straightforward bag-of-words vectors since they can encapsulate relative meanings: "king" - "man" + "woman" = "queen". It was found that this could be generalized from word-level to phrase-level and sentence-level [6].

2.3 Sequence to sequence neural networks

Sequence-to-sequence neural networks [1,16] are the state-of-the-art text generation models today. It is made up of two recurrent neural networks [4,16] that learn how an input sentence becomes an embedding, which represents the meaning of that sentence, and learning how to convert the embedding into another output sentence. It can then respond to fresh inputs, and create complete sentences instead of disjoint phrases, and so is more powerful than previous models.

3 Dataset and Features

3.1 Dataset details

The training/dev/test split was 0.8/0.1/0.1. Personal and public datasets were used. The personal dataset contained SMS (5642 conversations, 131 threads, 1 year) and Whatsapp messages (3612 messages, 5 threads, 6 months). Conversational English public datasets were used to approximate chat messages. Public dataset only training was done with the Cornell Movie Dialog Corpus [13] (220k messages) and Marsan Ma's Twitter Chat Corpus [14] (700k messages). Later a pretrained model on a Reddit dataset (3.4M, 1 month) [10] was used when training the personal dataset on top of a pretrained model. The data came in different formats (XML, CSV) and were preprocessed to plain text files and JSONs.

3.2 Features

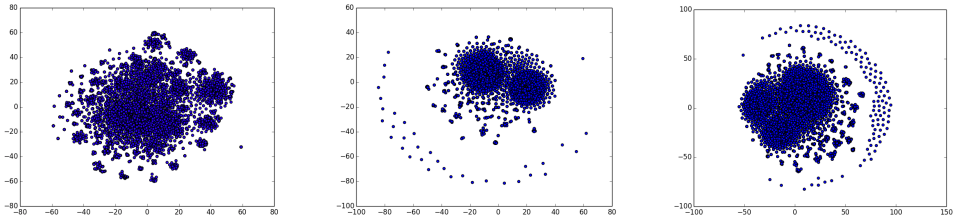

Figure 1. Left-to-right: visualized SMS, whatsapp, SMS+whatsapp messages

The messages were visualized with t-SNE [15], which like Principal Component Analysis reduces data from high dimensions into lower dimensions. From these visualizations it looked like the messages could be clustered. To do K-means, the sentences were converted into bag-of-word (BOW) vectors. Both plain counts and TD-IDF (which counts relative frequency) were tried. For the neural networks, the words were embedded using word2vec.

4 Methods

4.1 k-means only

In this baseline method, we converted every input message into a BOW vector, x, and clustered them using k-means. Here, we randomly intialize k centroids, μ_j, and classify each vectors into a centroid's cluster: $c^{(i)} := \text{argmin } ||x^{(i)} - \mu_j||^2$. Then we shift the centroid of each cluster to the average cluster vector: $\mu_j := \Sigma_{i=1} 1\{c^{(i)} = j\} x^{(i)} / \Sigma_{i=} 1\{c^{(i)} = j\}$, making it a "theme" vector that represents its cluster. Then we repeat until convergence. The clustered vectors each have a corresponding response message. For incoming new messages, we find the nearest trained centroid, pick a random message in that centroid's cluster, and return the response.

4.1 k-means + Markov chain

This is the same as the k-means only method, except instead of randomly returning a response, we trained a Markov chain for every cluster, and generate responses for new incoming message using the nearest trained centroid's cluster's chain. Markov chains assume that the probability of a word occuring is dependent on n={1, 2} preceding words: $p(x_i) = p(x_i|x_{i-1},...,x_{i-n})$. As a seed for each generated sentence, we picked one, or two consecutive, word(s) from the input message to start off the chain, and probabilistically picked the next word, untill some cutoff length.

4.2 Sequence-to-sequence

As mentioned before, sequence to sequence models are made up of two RNNs. One RNN acts as an encoder that transforms an input vector of a vocabulary-indexed sentence into an output embedding known as a "thought" vector which encapsulates the meaning of the sentence. Another RNN acts as a decoder that transforms the "thought" vector into an actual output sentence (a softmax probability of each word). The input and output sequences can have arbitrary lengths. An optional decoder layer, attention, can be applied. It learns weights as to what elements of the input sequence to focus on in generating each output word.

Instead of basic RNN cells, Seq2Seq cells are usually more complex. One special cell that the RNNs are made of is called the Long-Short-Term-Memory unit (LSTM), which has the capability to remember and forget inputs. There is also the more concise Gated Recurrent Unit (GRU), which roughly behaves the same but is cheaper to compute.

4.2.1 Markov Decision Processes to permute Seq2Seq generated words

We tried using reinforcement learning to permute words, since the Seq2Seq training words were spelled differently from the SMS words ("what" vs. "wat/whaatt/wud"). MDPs model problems as a tuple of states, actions, transition probabilities, rewards, and a discount factor. The goal is to find an optimal policy that suggests value-maximizing actions. Taking every character as a dimension would be too many dimensions and not work well with MDP, so character-level permutation was vastly simplified: In the MDP, each state was a "<before_character>, <after_character>" tuple (flattened in actual implementation for performance), and there were four actions "NOOP" (do nothing), "INSERT" (insert a new character after the chosen character), "DELETE" (the chosen character), "SUBSTITUTE" (the chosen character for a new character). The reward function was summing the XOR of the current word and the target word.

5 Experiments

5.1 k-means (K)

Suprisingly, clustering grouped the messages into themes instead of the original threads:

Themes	Examples
schedule	busy, will be there, am here, sorry will be late
activities	want to do X, where to do X, how I've been, why X is Y
sounds good	that's good, sounds good
answers	K, sure, can't, can, np, cool, will join, am free, reached, parking
identity	what person A has done/will do/wants to do/is doing
getting feedback	can we target this time, let me know, are you available, are you planning

Figure 2. Cluster themes observed in K-means (n=11) with clusters with <50 messages discarded

k=11 was found to do better than lower k's (2, 5, 7) in the two personal datasets (separate and together). k=11 is able to find meaningful themes, while lower k's had less interesting themes (e.g. haha vs everything else). We only trained on the personal dataset. It took <6.5s to train and save.

5.2 k-means with Markov chains (K+MC)

Markov chains came up with sentences that sounded like the user but were low quality, since they picked words by n-previous words instead of the whole sequence as Seq2Seq does. The output lengths were constrained by a cutoff length (the personal dataset's average text length, 8). Randomizing the cutoff did not improve this much since the chains do not learn when to naturally

end a sentence. Training the chains on the clusters resulted in lower quality sentences that were more input-relevant. The model took <2 min to train and save.

5.2 Seq2Seq with Attention (s2s+A)

The best configuration when running Tensorflow on CPU was found as: using (Luong) attention, using LSTM cells, with learning rate 0.01, 2 layers, 128 cells, dropoff 0.4. On GRU cells, the model trained faster and it also did better with less training data.

Seq2Seq loss in the personal dataset typically converged to ~10.5, and for the public datasets went to ~4.5. Perplexity typically stayed high (110+, 330+). The model took 5 days to train to 10000 steps, and should have been trained much longer as its sentences were less grammatical than those found in papers.

Our original idea was doing k-means and training Seq2Seq on each cluster (K+s2s), but this was discarded since the model was not able to train well on the unclustered personal dataset. Seq2Seq performed poorly regardless of toggling the following features: regularization, number of layers, number of cells, dropout rate, attention. The generated outputs had many repeating words or quickly resulted in all UNKs (unknown word tokens) after a few thousand steps.

5.2.1 Training personal data on top of a pretrained Seq2Seq model

Training on just a public dataset, or only on the personal dataset resulted in less plausible replies than k-means with Markov chain, which was undesirable since sentences from Markov chains are usually incomplete and jumbled. The Pender Seq2Seq pretrained model[10] had been trained for 4.6M steps (43 days). Sentences generated from this model were of high quality, although much more aggressive and offensive than previous datasets. We trained the personal dataset for 2 days on the model. The replies became slightly less offensive with more hedge phrases like "well", "i just wanted", "i guess" from the personal dataset. It appears that with a longer training period, the model might overfit itself to take on the personal dataset's personality.

5.3 MDP on character level

The model did not perform well (the characters may as well be randomly changed) as the reward function and the states were too simplistic. For example, "what you think" became "wyat you tink", when the expected was "whatt you think". So we quickly discarded this idea.

5.4 Comparison of Generated Replies

Model	Example (x:input, y:actual, h: generated reply)	Sounds like user	Input-relevant	Sentence quality	Average score
K	x: ok cool. hopping in singles y: i'm on the right hand side h: ohh why is there chocolate? thanks!	4.00	1.12	4.00	3.04
K+MC	x: how's the price? y: yay april then! h: 15-20%? i'm really it's etc://	2.85	2.87	1.50	2.41
s2s+A (w/ personal data)	x: eat first or swim? y: yep h: 20 if if if if korean korean cost	1.19	1.69	1.00	1.29
s2s+A (w/ public data)	x: busy at work? y: been playing com games ohgod h: i love you	1.82	3.01	3.05	2.63
s2s+A	x: but this is also his choice tho..	2.46	2.6	3.90	2.99

(pretrained on public data + personal data)	y: lol yeah h: do you know what happens if you don't want to?				

Figure 3. In the example replies, x is the input message, y is the actual reply, h is the predicted reply.

The metrics on the right four columns are human evaluated, since standard metrics like loss/BLEU/perplexity do not account for relevance or personality. Each predicted output was bucketed into one of four numbers, 1 being a low score, 4 being high, for each of the three categories, "sounds like user", "input-relevant", "sentence quality". Each model's scores were averaged for each character.

As can be seen, Seq2Seq trained by both public and personal data had high sentence quality, and comparable personality and relevance as k-means with Markov chain. k-means-only gets maximum scores for "sounds like user" and "sentence quality" since the output are actual replies, but in terms of input relevance, the Seq2Seq models scored better, and are therefore more suitable for generating SMS replies since they are more robust.

6 Conclusion

Seq2Seq trained on public data can adapt to talk more like an individual user with additional training. Since conversational English in the same region has the same basic grammar and social norms, a Seq2Seq model trained well on a large dataset acquires a basic foundation of knowing how to start and complete meaningful sentences that can then be overfitted to one user's style of chatting.

The pretraining step, which takes a long time, can be done beforehand. The additional training step takes much shorter, although the time taken in the experiments (2 days) is still unsable for a real world user setting up their mobile application. k-means with Markov chains was much cheaper to run than Seq2Seq, and had high scores on both input-relevance and personality, but the sentence quality was low.

Future work could include experimenting more with training personal data on top of a pretrained model, considering how a user responds to different people on different topics, and actually building a mobile application that real-world users can use to generate auto-replies to their own messages.

7 References

[1] Sutskever et al- Sequence to Sequence Learning with Neueral Networks
http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
[2] Vinyals, Le- A Neural Conversational Model, 2015 https://arxiv.org/pdf/1506.05869.pdf
[3] Sordoni, et al- A Neural Network Approach to Context-Sensitive Generation of Conversational Responses, 2015 https://arxiv.org/pdf/1506.06714.pdf
[4] Greff et al- LSTM: A Search Space Odyssey, 2017 https://arxiv.org/pdf/1503.04069.pdf
[5] Mikolov et al- Efficient Estimation of Word Representation in Vector Space, 2013 https://arxiv.org/pdf/1301.3781.pdf
[6] Mikolov et al- Distributed Representation of Words and Phrases and their Compositionality, 2013 https://arxiv.org/pdf/1310.4546.pdf
[7] Tensorflow (0.12, 1.0.0, 1.4, Nightly), Numpy, Python 2.7, 3.6, Scipy, Scikit-learn
[8] Google Tensorflow NMT tutorial https://github.com/tensorflow/nmt/tree/master/nmt

[9] Practical Seq2Seq tutorial https://suriyadeepan.github.io/2016-12-31-practical-seq2seq/
[10] Chatbot RNN tutorial https://github.com/pender/chatbot-rnn
[11] Wilcox et al- Making it Real: Loebner-winning Chatbot Design
http://arbor.revistas.csic.es/index.php/arbor/article/view/1888/2079
[12] Jurafsky- Speech and Language Processing Textbook 3rd Edition
https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf
[13] Cornell Movie Dialog Corpus https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html
[14] Marsan Ma's Twitter Chat Corpus https://github.com/Marsan-Ma/chat_corpus
[15] Maaten et al- Visualizing Data using t-SNE, 2008
http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
[16] Cho et al- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014 https://arxiv.org/abs/1406.1078
[17] IR-book- Markov chains, 2008 https://nlp.stanford.edu/IR-book/html/htmledition/markov-chains-1.html
[18] Reddit dataset (1 month)
https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/