Everything I know, I learned From the Movies

A Chatbot Trained on Movie Dialogue

Jeff Sheldon jeffsh@uw.edu

One of the exciting prospects of working in natural language processing is the opportunity to build applications that others may interact with. People have become accustomed to interacting with NLP-powered intelligences. Examples include asking their cable remote to put on a show and expecting Google to provide a better than literal word translation to a foreign language.

The use of chatbots is another opportunity for users to interact with an intelligent system. If done well and used in the correct circumstances, chatbots may provide tangible benefits to both end users, as well as the organizations utilizing them. Chatbots may aid consumers when they encounter problems with a product, providing immediate support if human support is not currently available.

This exercise does not attempt to produce a state-of-the-art chatbot. The research and experimentation done here has not produced a chatbot capable of fooling anyone into believing that they are interacting with a human. It does provide a model by which a chatbot may be built and trained, without the aid of high-powered infrastructure. The chatbots analyzed by this paper are far from perfect, but they do provide surprising and interesting results and interactions.

Related Work

The work in this paper is inspired and heavily influenced by *A Neural Conversational Model (Vinyals, Le; 2015)*[1]. That paper proposed the application of Seq2Seq neural networks to the problem of chatbots. The model proposed by that paper was built entirely of LSTM cells. Vinyals and Le trained their model on two datasets. One dataset was pulled from IT help logs, thus lending itself to the purpose of providing an automated IT chatbot. The second dataset was pulled from movie captions. In the analysis section of this paper, the responses generated by this experiment's chatbot will be compared to those generated by Vinyals and Le.

A prior paper, *Sequence to Sequence Learning with Neural Networks (Sutskever, Vinyals, Le; 2014)*[2] provides the basis for the model. In that paper, LSTM RNNs are looked at more deeply and applied to the problem of machine translation.

Seq2Seq RNNs are well documented and heavily implemented. Pytorch offers a thorough example of implementing such an architecture.[3] The information and code contained within that example have been re-used heavily here. The Pytorch example implements the Seq2Seq RNN using GRU cells and Greedy decoding.

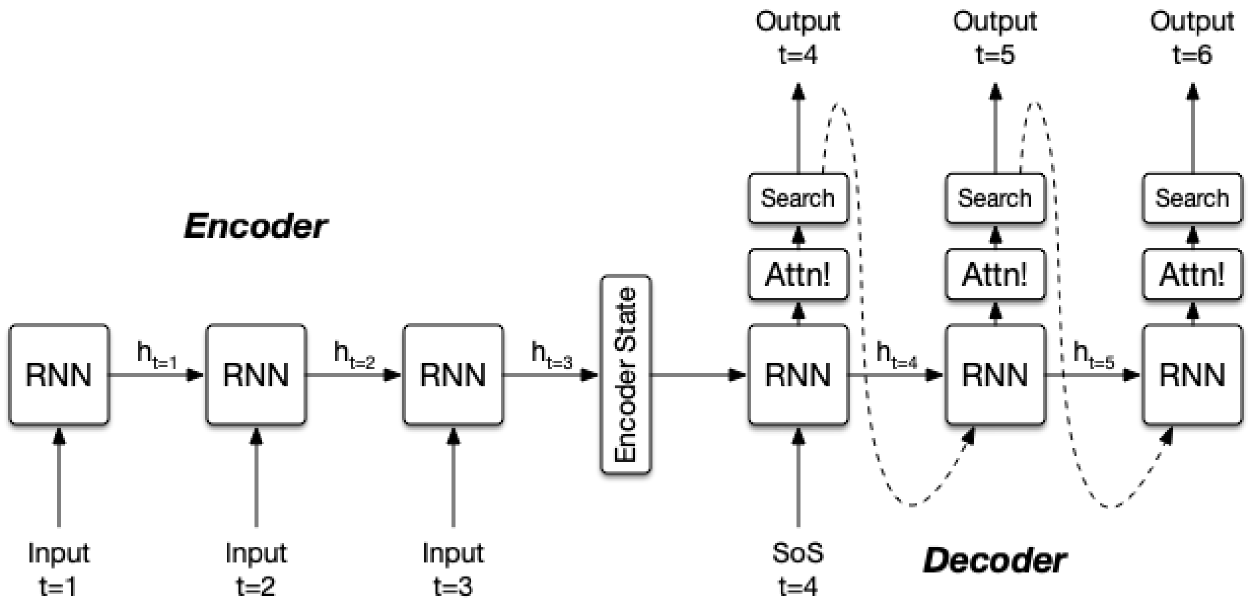
[1] https://arxiv.org/pdf/1506.05869.pdf
[2] https://arxiv.org/pdf/1409.3215.pdf
[3] https://pytorch.org/tutorials/beginner/chatbot_tutorial.html

Model Architecture

For this experiment, a Seq2Seq Encoder-Decoder recurrent neural network architecture was utilized. Such an architecture is no longer the optimal solution for problems such as chatbots. Rather, a transformer-based approach, such as GPT2, far outperforms Seq2Seq RNNs. The goal of this work is not to recreate a state-of-the-art chatbot. Instead, this work was used as an opportunity to go through the motions of building a neural language chatbot from the ground up.

Recurrent neural networks provide value to a problem such as generating dialogue in response to a query, as they maintain information about earlier events, over time. At a given time T, a RNN cell will have access to not only information about the data input into the network, but about cell's prior state as well. This information is carried from time T=1 through to the current time T. This allows the network to maintain knowledge of prior inputs and outputs.

The Sequence-to-Sequence architecture provides two RNN blocks. The first is the encoder block, a bidirectional RNN, which takes as input a user generated statement. At the final timestep of input, the state of the encoder, which represents the entirety of the user input, is passed to the decoder block. The decoder will generate output, one word at a time. The output generated will become a part of the decoder state, to be used in the generation of the subsequent word. At each time step, the decoder output will be passed to an attention layer. The attention layer will have access to all prior RNN output. This allows attention to produce an output based directly on prior outputs, eliminating issues related to the degradation of data in state as it is passed through the RNN. Attention's output is then passed through a softmax function to generate a series of probabilistic outputs, which are then fed through a search decoder to generate an output word.



The Pytorch GRU Greedy implementation was expanded upon heavily. Support for LSTMs has been added to both the encoder and decoder blocks. Models built may interchangeably swap out either GRU or LSTM units. It should be noted that the choice of GRU or LSTM must define both the encoder and decoder architecture blocks. This is done to support the transfer of cell state, in addition to hidden state, from the encoder block to the decoder block, when LSTM units are to be used.

The base output decoder implemented only a greedy search algorithm. Given a choice of words to utter, from the softmax layer of the decoder block, a greedy algorithm will always select the most likely word. Syntactically and contextually, such a method may produce the most probable response, given the following definition of statement probability:

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} p(x_i | x_1 \dots x_{i-1})$$

Such responses are shown to come across to as robotic.[4] To attempt to provide a more human output, two additional search decoders were added: TopK and TopP. As is the case with RNN method, search decoders may be swapped out of models interchangeably.

TopP reduces the available vocabulary until a set probability threshold is met. Iterating over the available probabilities, from greatest to lowest. The TopP vocabulary is defined as:

$$\sum_{i\ s.t.\ t_i \in V_p} p[i] \geq p$$

Given the reduced vocabulary, the distribution of probabilities must then be renormalized, according to those words which are a part of the TopP vocabulary.

$$p'[i] = \frac{p[i]}{\sum_{i\ s.t.\ t_i \in V_p} p[i]}\ if\ t_i \in V_p, else\ 0$$

TopK redefines the vocabulary by selecting the highest probability choices available, until K choices have been made. The set of probabilities is then renormalized, according to the redefined vocabulary. Renormalization of TopK is identical to TopP renormalization.

Dataset

The model has been trained on the Cornell Movie Dialogue Corpus.[5] This dataset comprises several raw files, which define lines, the conversations the lines belong to, the character who spoke the line, and the movie that the conversation belongs to. The raw data provides 304,713 lines, making up 83,097 conversations.

From the lines and conversations, lines and response pairs can be identified. This provides the basis for the training data: the initial line is fed as input to the encoder and the decoder output is compared to the response to measure loss.

Given that the raw data is pulled from multiple files, an initial data normalization step assembles the line pairs into a formatted file. A maximum length hyperparameter defines that maximum number of words that can appear within a line. Any line or response that exceeds that length will be ignored. This is done to ease the burden of training. As such, the number of lines and responses to train on varies according to

[4] https://courses.cs.washington.edu/courses/csep517/21wi/slides/NeuralGeneration.pdf slide 30
[5] https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

the configured maximum length. With a maximum length of 20 words, 221,061 line and response pairs are available for training.

A configurable amount of training data is set aside to be used as holdout test data. For this experiment, one of every 1000 pairs was held out of the training corpus and assigned instead to the test corpus. The test corpus was only used for the evaluation of completed models.

Training Process

An iterative approach is taken to train a model. Rather than seeking convergence, each model is preconfigured to be trained for a set number of iterations. The number of training iterations has been altered, per model. The results of those differences will be analyzed in the subsequent section.

For each training iteration, a batch of 64 sample inputs are fed to the model. Teacher forcing is used during decoding for 95% of the iterations trained. Using teacher forcing, the expected prior response is fed to the decoder at each time step, rather than the response generated by the decoder at the prior time step. The Adam optimization algorithm is used to apply weight adjustments at the conclusion of each iteration.

Experiment Analysis

The flexibility of the model was used to test a variety of component combinations. A standard set of hyperparameters were assigned to each model tested and never altered.

- Minimum count of a word to be included in the vocabulary: 3
- Dropout layer ratio (encoder and decoder blocks): .1
- Batch size, per iteration: 64
- Gradient clipping for weight normalization: 50
- Teacher forcing ratio: .95
- Dot-product Attention

Models were built, each utilizing a unique combination of RNN cell types, search decoders, neural network sizes, training data max length, and training iterations. Each model was then analyzed on the holdout test dataset, using both Meteor and Bleu performance metrics.

Model Analysis on Holdout Test Data

MODEL	HYPERPARAMETERS	ITERATIONS	METEOR	BLEU
GRU GREEDY 2-2-500	GRU + Greedy Size: 2, 2 @ 500 10 Max Words	4000	0.07267	0.00844
GRU TOPP 2-2-500	GRU + TopP=.4 Size: 2, 2 @ 500 10 Max Words	4000	0.07654	0.01836
LSTM GREEDY 2-2-500	LSTM + Greedy Size: 2, 2 @ 500 10 Max Words	4000	0.07009	0.00468
LSTM TOPP 2-2-500	LSTM + TopP=.4 Size: 2, 2 @ 500 10 Max Words	4000	0.06158	0.00411
LSTM TOPP 2-2-1000	LSTM + TopP=.4 Size: 2, 2 @ 1000 10 Max Words	5000	0.0688	0.00977

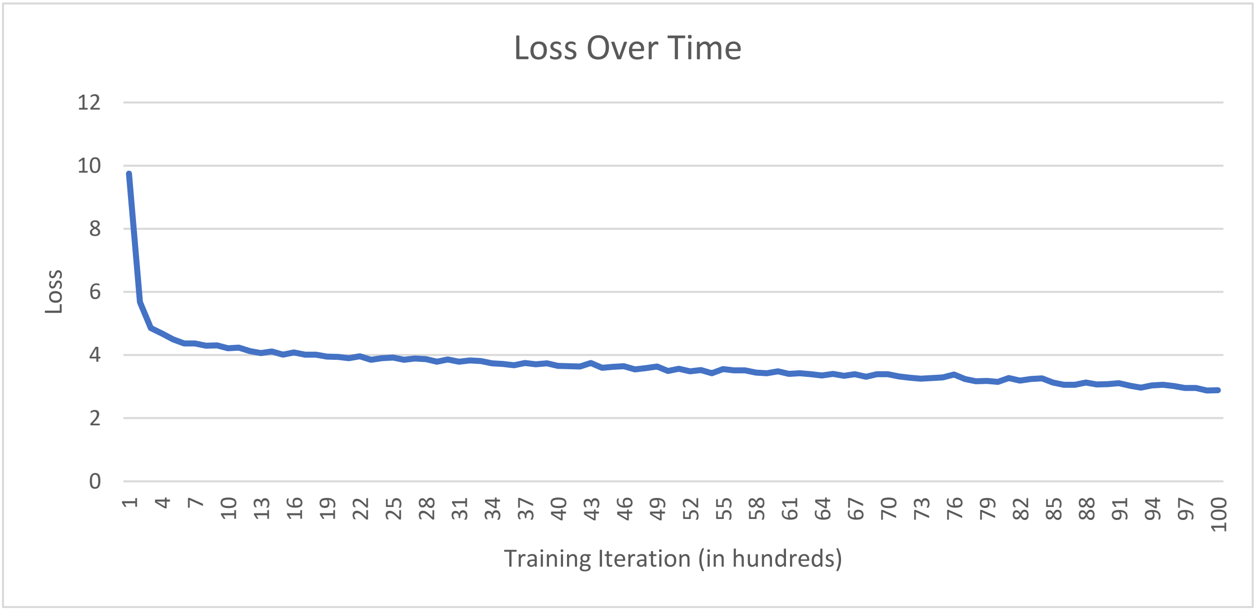
LSTM TOPP 4-4-1000	LSTM + TopP=.4 Size: 4, 4 @ 1000 10 Max Words	16,000	0.06068	0.00471
LSTM TOPP 2-2-1000 M20	LSTM + TopP=.4 Size: 2, 2 @ 1000 20 Max Words	10,000	0.08138	0.01466

Note: Size: #,# @ # is read as Size: <num encoder layers>, <num decoder layers> @ <neurons per layer>

Meteor scores indicate that the most complex model, *LSTM TopP 2-2-1000 M20*, is the most performant. The highest Bleu score is awarded to one of the simplest models, *GRU TopP 2-2-500*. The differences in scores are quite minimal. Such differences may be attributed to random factors, such as the initialization of weights within the model when training is initiated.

Additionally, those models using TopP as their search decoder will exhibit a degree of randomness in the words they select to output. Such models have set their probability threshold for the algorithm to .4. This setting allows for few possibilities if the model feels strongly about certain options. If the model's conviction is weak, TopP will have many low-probability options to choose from. While this will still prevent pure sampling of options, weak convictions will still result in very low probability options being made available to the algorithm.

LSTM TopP 2-2-1000 M20 Loss Curve



The loss curve of the most complex mode; representative of all models trained.

Scripted Analysis

The best models, complex model *LSTM TopP 2-2-1000 M20,* and simple model *GRU TopP 2-2-500*, were selected for scripted analysis. This provides a comparison between the model configuration approaches.

Script 1 – Basic Interaction

The first script will attempt to analyze basic behavior of the models. It will begin with basic questions and will ask related questions, intended to elicit common answers. Both models show some ability to produce responses that fit the context of the query. For example, asking the model what its name is elicits a proper response. At other times, the models become confused. When asked about their mom, neither model gives convincing answers.

Query: Hello.
Complex: Hello.
Simple: Hello.
Query: My name is Jeff.
Complex: I don't want to hear you do this.
Simple: Oh.
Query: What is your name?
Complex: Belle.
Simple: Earl.
Query: What is your first name?
Complex: Belle.
Simple: A righteous man.
Query: Building you was a lot of work.
Complex: I was the one of the best.
Simple: No.
Query: How is the weather?
Complex: Beautiful.
Simple: I don't know.
Query: What is your favorite food?
Complex: No I'm not hungry.
Simple: You're not a doctor.
Query: What time is it?
Complex: Twelve thirty.
Simple: Thirty ten thirty.
Query: Do you go to school?
Complex: No.
Simple: No.
Query: What's my name?
Complex: Dr. Feur.
Simple: A priest.
Query: Do you like cats?
Complex: Yes.
Simple: Yeah.
Query: Tell me about your mom.
Complex: She's my sister.
Simple: She'll be dead.
Query: What about your dad?
Complex: He's dead.
Simple: I'm a dancer.

Script 2 – Vinyals/Le General Knowledge and Philosophical Q&A

The second script is drawn directly from *A Neural Conversational Model*[6] and provides a comparison between the models configured here and the model trained by Vinyals and Le. The models are asked questions that demand specific answers. The complex model outperforms the simple model here. For example, the complex model accurately answers that a cat has four legs. The simple model occasionally

[6] https://arxiv.org/pdf/1506.05869.pdf, pg5, Conversations 3&4

runs into vocabulary issues. This is due to a combination of a smaller vocabulary and an implementation shortcoming.

Query: Who is Skywalker?
Complex: You know who I am.
Simple: *<Error Missing Vocab>*
Query: Is sky blue or black?
Complex: You think the American guy is a little weird?
Simple: No.
Query: Does a cat have a tail?
Complex: Yes.
Simple: I have a headache.
Query: Does a cat have a wing?
Complex: No.
Simple: I don t know.
Query: Can a cat fly?
Complex: Sure.
Simple: No.
Query: How many legs does a cat have?
Complex: Four.
Simple: Three.
Query: How many legs does a spider have?
Complex: Six.
Simple: Ten nine.
Query: What is the color of the sky?
Complex: It's a cobra.
Simple: The marriage of the nine.
Query: What is the color of water?
Complex: It's . . . . . . The winter.
Simple: The plan.
Query: What is the color of blood?
Complex: You...
Simple: It's a complicated of fact.
Query: What is the color of a yellow car?
Complex: I don't know . . . I think I've got a car.
Simple: The car.
Query: How much is two plus two?
Complex: Two.
Simple: Nine. Two hundred.
Query: How much is ten minus two?
Complex: Two million.
Simple: Four hundred.
Query: What is the purpose of life?
Complex: A of course.
Simple: The reward.
Query: What is the purpose of living?
Complex: I have to kill you.
Simple: The children.
Query: What is the purpose of existence?
Complex: I'm not sure.
Simple: It's me.
Query: Where are you now?
Complex: I'm here to find out.

Simple: Out.
Query: What is the purpose of dying?
Complex: ...
Simple: What's the point?
Query: What is the purpose of being intelligent?
Complex: A big loan of the power of the galaxy.
Simple: We're looking for a five thousand dollars.
Query: What is the purpose of emotions?
Complex: I should be married.
Simple: *<Error Missing Vocab>*

Conclusion

While the models produced using the Seq2Seq RNN architecture can provide interesting and occasionally relevant responses, none of the models trained for this experiment are capable of consistently performing at an acceptable level. At times, models directly answer the query the given. They may even provide answers that appear to draw on insight not provided in the original query. At other times, they give nonsensical answers, failing completely to respond to the context of the original query.

Both the Bleu and Meteor scores indicate poor performance for all the models. Chatbots are put into a scenario that is similarly challenging to neural generation of story endings. That is, they are given the flexibility to produce creative answers to the original query. Because of this, the opportunity to find overlap between the answer produced and the ground-truth is limited, thus resulting in lower than usual metrics scores.

Vinyals and Le show in their work that there is promise in the capabilities of Seq2Seq RNNs for chatbot applications. Their chatbot trained on a movie corpus was significantly more powerful than any chatbot produced for this experiment, featuring two-layer LSTMs with 4096 cells. They make no mention of corpus limiting, as was done for this experiment.

Future work on increasing the complexity of the model could yield interesting results: more hidden layers and neurons per layer. Given more complexity, removing the maximum sentence length cap would allow the model to train on more complex data. The training process can also be enhanced. Rather than training for a set number of iterations, model loss can be measured and used to define convergence criteria.

The current state of the art neural language generation is in transformer architectures, not recurrent neural networks. Models such as GPT2 will outperform a Seq2Seq RNN. This was understood at the outset of this experiment. However, the goal of this project was not to build the most performant model. In building and running this model architecture, knowledge has been gained in the use of neural language tools and how to apply them to real problems.

References

Vinyals, O., Le, Q. A Neural Conversational Model
https://arxiv.org/abs/1506.05869v3
Sutskever, I., Vinyals, O., Le, Q. Sequence to Sequence Learning with Neural Networks
https://arxiv.org/abs/1409.3215v3
Inkawhich, M. Chatbot Tutorial
https://pytorch.org/tutorials/beginner/chatbot_tutorial.html