CONVERSATION BOT

Semester Project

Guided by: Prof. Manish Srivastava

Submitted By:

Ayush Khandelwal-201505512

Rashi Chauhan-201506527

K Subash Chandra Reddy-201505544

CONTENTS

Introduction

Project aims to build a generic intelligent chatbot capable of giving relevant and accurate responses. Prime Target is to employ models that can capture context which was not possible accurately from our previous implementation that involves “k-means” cluster based transitions to get appropriate response for a query.

Datasets

An immediate challenge in the project is finding and deciding on an appropriate dataset. Our aim was to find a large enough dataset, with little noise but with enough diverse meaningful information enough to efficiently train a model.

1. Cornell Movie-Dialog Corpus

This corpus contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts:

- 220,579 conversational exchanges between 10,292 pairs of movie characters
- involves 9,035 characters from 617 movies
- in total 304,713 utterances
- movie metadata included:
    - genres
    - release year
    - IMDB rating
    - number of IMDB votes
    - IMDB rating
- character metadata included:
    - gender (for 3,774 characters)
    - position on movie credits (3,321 characters)

http://www.cs.cornell.edu/%7Ecristian/Cornell_Movie-Dialogs_Corpus.html

2. Twitter Chat Log

Twitter_en

Corpus scrap from twitter (7,00,000 tweets), where odd lines are tweet and even lines are corresponding responded tweets.

Twitter_en big

Twitter corpus with 25 lakh query-response pairs.

https://github.com/Marsan-Ma/chat_corpus

3. Reddit Comments

This corpus contains data in a JSON format.

Fields per reddit comment
{"author", "name", "body", "author_flair_text", "gilded", "score_hidden", "score", "link_id","retrieved_on", "author_flair_css_class", "subreddit", "edited", "ups", "downs", "controversiality", "created_utc", "parent_id", "archived", "subreddit_id", "id", "distinguished"}

“name” -> unique to each comment
“parent_id” -> name of message to which current message is reply to
“body” -> actual message body

Reddit Comments 2011-03

Reddit Data with 43,36,705 query-response pairs after appropriate data cleaning.

https://files.pushshift.io/reddit/comments/RC_2011-03.bz2

Reddit Comments 2009-03

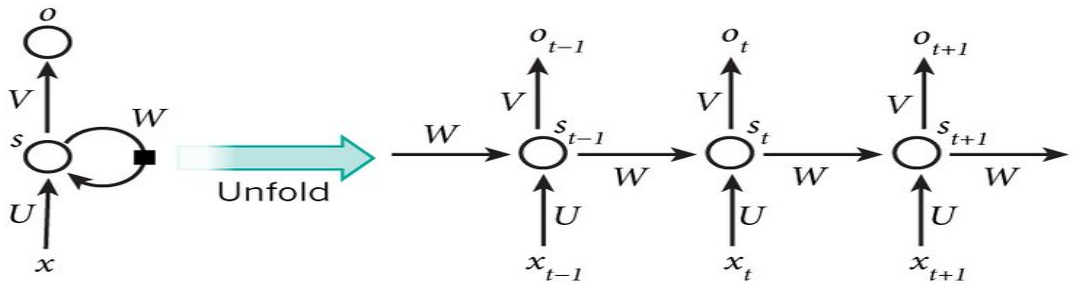
Reddit Data with 5,96,871 query-response pairs after appropriate data cleaning.

https://files.pushshift.io/reddit/comments/RC_2009-03.bz2

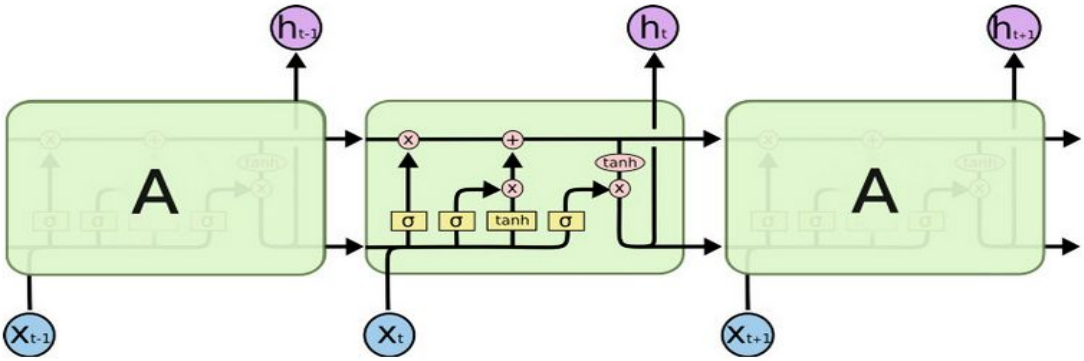
Conversational TensorFlow Model

Recurrent Neural Networks

Recurrent Neural Networks or simple RNNs, are a special kind of neural networks that are capable of dealing with sequential data, like videos(sequence of frames) and more commonly, text sequences or basically any sequence of symbols. The beauty of it is, the network doesn't need to know what the symbols mean. It will infer the meaning of symbols, by looking at the structure of the text and relative positions of symbols. To put it simply, an RNN, unlike an MLP or CNN, has an internal state as memory of the network. As the RNN devours a sequence (sentence), word by word, the essential information about the sentence is maintained in this memory unit (internal state), which is periodically updated in each timestep. The essence of a sentence of 7 words will be captured by an RNN, in 7 timesteps.

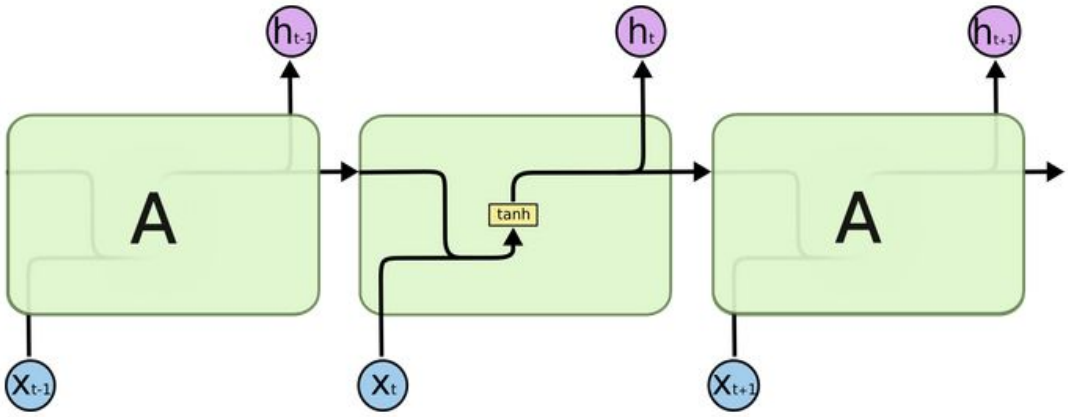


There are more complex versions of RNN, like LSTM (Long Short Term Memory) and GRU (Gated Recurrent Units) RNNs. An LSTM cell consists of multiple gates, for remembering useful information, forgetting unnecessary information and carefully exposing information at each time step.



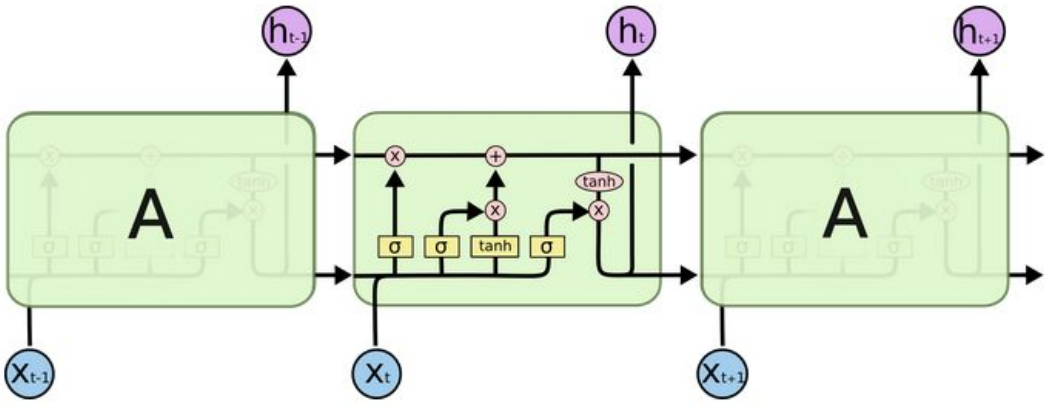
LSTM Networks

Long Short Term Memory networks called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

The Core Idea Behind LSTMs:

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

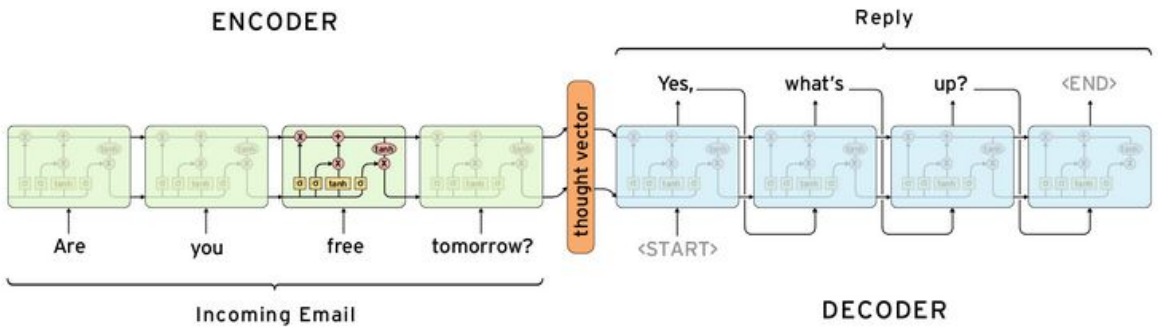
The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state.

Sequence to Sequence Learning

The Sequence to Sequence model (seq2seq) consists of two RNNs - an encoder and a decoder. The encoder reads the input sequence, word by word and emits a context (a function of final hidden state of encoder), which would ideally capture the essence (semantic summary) of the input sequence. Based on this context, the decoder generates the output sequence, one word at a time while looking at the context and the previous word during each timestep.



The context can be provided as the initial state of the decoder RNN or it can be connected to the hidden units at each time step.

There are a few challenges in using this model. The most disturbing one is that the model cannot handle variable length sequences. It is disturbing because almost all the sequence-to-sequence applications, involve variable length sequences. The next one is the vocabulary size. The decoder has to run softmax over a large vocabulary of say 20,000 words, for each word in the output. That is going to slow down the training process, even if your hardware is capable of handling it. Representation of words is of great importance. Use of one-hot vectors means we need to deal with large sparse vectors due to large vocabulary and there is no semantic meaning to words encoded into one-hot vectors.

Padding

Before training, we work on the dataset to convert the variable length sequences into fixed length sequences, by **padding**. We use a few special symbols to fill in the sequence.

1. **EOS** : End of sentence
2. **PAD** : Filler
3. **GO** : Start decoding
4. **UNK** : Unknown; word not in vocabulary

Bucketing

Padding did solve the problem of variable length sequences, but consider the case of large sentences. If the largest sentence in our dataset is of length 100, we need to encode all our sentences to be of length 100, in order to not lose any words.

Bucketing solves this problem, by putting sentences into buckets of different sizes. Consider this list of buckets : **[** (5,10), (10,15), (20,25), (40,50) **]**. If the length of a query is 4 and the length of its response is 4, we put this sentence in the bucket (5,10). The query will be padded to length 5 and the response will be padded to length 10. While running the model (training or predicting), we use a different model for each bucket, compatible with the lengths of query and response.

Word Embedding

Word Embedding is a technique for learning dense representation of words in a low dimensional vector space. Each word can be seen as a point in this space, represented by a fixed length vector. Semantic relations between words are captured by this technique.

Word Embedding is done in the first layer of the network : Embedding layer, that maps a word (index to word in vocabulary) from vocabulary to a dense vector of given size. In the seq2seq model, the weights of the embedding layer are jointly trained with the other parameters of the model.

TensorFlow

TensorFlow is an open-source machine learning library. It relies on the construction of dataflow graphs with nodes that represent mathematical operations (*ops*) and edges that represent tensors (multidimensional arrays represented internally as numpy ndarrays). The dataflow graph is a summary of all the computations that are executed asynchronously and in parallel within a TensorFlow session on a given device (CPUs or GPUs). The library boasts true portability between devices, meaning that the same code can be used to run models in CPU-only or heterogeneous GPU-accelerated environments.

TensorFlow relies on highly optimized C++ for computation and supports APIs in C and C++ in addition to the Python API used to build this ChatBot.

Implementation

Observations on Dataset Selection

Practical Observation suggested Cornell Movie-Dialog Corpus is neither large-enough and nor rich in information to provide relevant responses. While "twitter_en" gave very promising results, "twitter_en gib" and "reddit data" look promising after training with sufficient number of epochs and vocabulary size.

Data Preprocessing

While twitter data required little processing that includes handling smileys and other special characters, reddit data was in json like format to start with, therefore required tag based storage and other additional conditions to get data in query-response format.

* Further processing was done to get thread-format, if present, from the reddit dataset.

* Preprocessing scripts were added to git repository enclosed.

Choosing vocabulary Size

Vocabulary size was chosen to make sure to cover 93-95% of the most-frequent words are present for model training. 6000 for twitter_en was chosen that way.

Choosing Epoch Count

Is done more on trial-and-error basis until we got satisfactory results. Indicator, we observed, to insufficient epoch count is if you notice same repeated responses for different queries or unrelated responses to queries as simple as “hello” or “good morning”.

Build a wrapper for Seq2Seq

We created a class, Seq2Seq that provides high level abstractions for building the graph, training, evaluation, saving and restoring trained model. The constructor takes these parameters as input:

- xseq_len, yseq_len
- xvocab_size, yvocab_size
- emb_dim
- num_layers
- ckpt_path
- lr=0.0001
- epochs=100000

```
model = seq2seq_wrapper.Seq2Seq(xseq_len=xseq_len,
                                yseq_len=yseq_len,
                                xvocab_size=xvocab_size,
                                yvocab_size=yvocab_size,
                                ckpt_path='ckpt/twitter/',
                                emb_dim=emb_dim,
                                num_layers=3
                                )
```

Training

We created a method train, that runs the train op, for a given number of epochs. Evaluation is done periodically on the validation set. The model is saved after evaluation. A dropout of 0.5 is used during training. The dropout is disabled during evaluation. When restore_last_session is called, the last saved checkpoint is restored and returned. The predict function does a forward step and returns the indices of most probable words emitted by the model.

We trained the model using the processed dataset. The *load_data* function returns the dataset (x,y) and the metadata (index2word, word2index). We split the dataset into train, validation and test sets. We set the parameters of the model, like sequence length, vocabulary size and embedding dimensions. We then, create an instance of the model and train it, by passing data iterators to *model.train* method. If the training gets interrupted deliberately or otherwise, we can

continue training from the last saved checkpoint. This is done by getting the session from *model.restore_last_session* and then passing the session to *model.train* method.

Working model

A working model of a Twitter bot based on seq2seq model. Following are some of the decent responses spit out by the ChatBot. These are responses after running the model on twitter_en (3,50,000 tweets) restricting to 6000 vocabulary ran for 40,000 epochs.

Query	Reply
who won the first presidential debate	trump will be a better time for a man
donald trump won last nights presidential debate according to snap online polls	thought he was a joke
what about hillary	she is a disgrace
what do you think about trump	i dont know what you think about it
who is trump	trump is a lie
been drinking pumpkin spice protein shake every morning for a week and now i literally cannot even	me too i just got it
its a good day	it was a lot of fun
happy birthday	thank you
its my birthday	happy birthday
thank you	youre welcome
love you	love you too
you are cute	thank you i appreciate it
i am just kidding	i dont know what you think i was talking about it

The unk symbols in the conversations, refer to the words that aren't frequent enough (rare) to be put in the vocabulary.

We ran the model on twitter_en big (with 25 lakh query response pairs) restricting vocabulary to 40,000 for 55,000 epochs. Results were not satisfactory.

Probable Reasons:

1. Relative to our first model giving satisfactory results, number of tweets have increased 7-fold, so vocabulary was also linearly increased. This may result in loss of vocabulary as vocabulary selection is based on frequency and certain words which may be less frequent but may be crucial to answering majority of queries. So, increasing vocabulary to 80,000-1,00,000 seems like a better idea.

2. Matching number of tweet increase, it is logical to say number of epochs should be increased appropriately, along the lines of 2,00,000 to 3,00,000 epochs to get more related and accurate responses.

Future Work

1. Train the model on twitter_en big dataset for 3,00,000 epochs and 80,000 vocabulary size. (currently running on LTRC server)

2. Train the model on reddit_03-2011 dataset. Data is preprocessed to query-response format. (scripts included in repo)

Code Repository

https://github.com/ras1234/ChatBot

References

1. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation https://arxiv.org/abs/1406.1078
2. Sequence to Sequence Learning with Neural Networks https://arxiv.org/abs/1409.3215
3. A Neural Conversational Model https://arxiv.org/abs/1506.05869
4. http://www.cs.cornell.edu/%7Ecristian/Cornell_Movie-Dialogs_Corpus.html
5. https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/
6. https://www.tensorflow.org/versions/master/tutorials/seq2seq
7. http://colah.github.io/posts/2015-08-Understanding-LSTMs