Jarvis: Seq2seq based Chatbot for Customer Support

Awadh Kejriwal Devansh Rajgorhia Utkarsh Patel Abhisek Mohanty

Abstract

Conversational models are a hotly debated issue in artificial intelligence research. Chatbots can be found in an assortment of settings, including client assistance applications and online helpdesks. These bots are frequently powered by retrieval-based models, which yield predefined reactions to inquiries of specific structures. In an exceptionally limited space like a company's IT helpdesk, these models might be adequate, nonetheless, they are not robust enough for broader use-cases. Teaching a machine to complete a significant discussion with a human in numerous areas is a research question that is a long way from settled. In this paper, we will implement this kind of model in PyTorch.

1 Introduction

A chatbot is a man-made reasoning (AI) programming that can reenact a discussion (or a visit) with a client in normal language through informing applications, sites, portable applications or through the phone. Why are chatbots important? A chatbot is regularly depicted as quite possibly the most exceptional and promising articulations of connection among people and machine. In any case, according to an innovative perspective, a chatbot just addresses the normal development of a Question Answering framework utilizing Natural Language Processing (NLP). Defining reactions to inquiries in normal language is one of the most regular Examples of Natural Language Processing applied in different ventures end-use applications.

No place is this extreme change to the client experience as evident as in the new influx of chatbots. Artificial intelligence controlled chatbots have opened up and allowed you to do what was once incomprehensible: help clients all day, every day, naturally resolve inquiries with practically no human intercession, and offer help to numerous clients without a moment's delay. We are profoundly accepting mechanization and bots to assist businesses with drastically upgrading their client experience, make more conversational connections, and accomplish quicker development. That is the reason we assembled Jarvis, an insightful chatbot that consequently and immediately settles clients most normal inquiries, at scale.

It's very obvious that chatbots help your client care specialists take care of their responsibilities better. However, perhaps the greatest advantage comes as time saved: since chatbots resolve straightforward inquiries rapidly, your group have more opportunity to handle more perplexing queries.AI chatbots aren't a substitution for genuine human communications. Bots are better at expanding these communications and are best used to improve on undertakings and eliminate reiteration from work processes.

Human specialists should deal with discussions where somebody is exploring a mind-boggling buy or is feeling disappointed or befuddled. As cutting edge as regular language handling has become, it can never truly offer a veritable "I'm sorry" or make things right like a human can.

For continuous communications, support bots can truly have some significant advantages both to your group and your clients. Simply recall that AI is a remote helper, it is there to assist your human specialists with taking care of their responsibilities better it can never supplant them completely.

2 Dataset

Customer Support on Twitter: The dataset contains almost 3 million tweets by biggest brands in the world. It is a large, modern corpus of tweets and replies to aid innovation in natural language understanding and conversational models, and for study of modern customer support practices and impact. Natural language

stays the densest encoding of human experience we have, and development in NLP has sped up to control comprehension of that information, yet the datasets driving this advancement don't coordinate with the genuine language being used today.

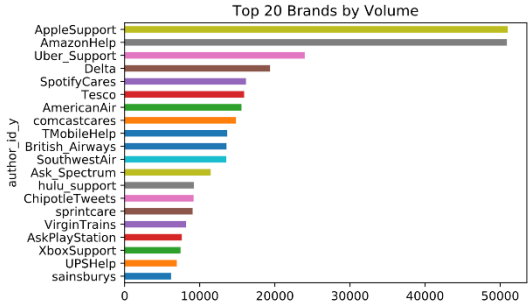

Figure 1: Top 20 brands by number of tweets in the dataset

The Customer Support on Twitter dataset offers a huge corpus of present-day English (for the most part) discussions among purchasers and client care specialists on Twitter, and enjoys three significant upper hands over other conversational message datasets:

- **Centered** - Consumers contact client care to have a particular issue tackled, and the complex of issues to be examined is generally little, particularly contrasted with unconstrained conversational datasets like the Reddit Corpus.
- **Natural** - Consumers in this dataset come from a lot more extensive section than those in the Ubuntu Dialog Corpus and have considerably more normal and late utilization of composed text than the Cornell Movie Dialogs Corpus.
- **Brief** - Twitter's quickness causes additional normal reactions from help specialists (rather than prearranged), and to-the-point portrayals of issues and arrangements. Additionally, its advantageous in considering a moderately low message limit size for intermittent nets.

The dataset is a CSV, where each line is a tweet. The various sections are portrayed underneath. Each discussion included has no less than one solicitation from a customer and somewhere around one reaction from an organization. Which client IDs are organization client IDs can be determined utilizing the inbound field.

Attributes	Description
tweet_id	Anonymized ID for the tweet.
author_id	Anonymized client ID
response_tweet_id	ID of tweets that are reaction to this tweet (comma-separated)
in_response_to_tweet_id	ID of the tweet this tweet is response of
text	Tweet content
created_at	Date-time when the tweet was sent
inbound	Whether the tweet is ‘inbound’ to an organization doing client care on Twitter

Table 1: Attributes contained in the Twitter Customer Support Corpus

	tweet_id	author_id	inbound	created_at	text	response_tweet_id	in_response_to_tweet_id
0	1	sprintcare	False	Tue Oct 31 22:10:47 +0000 2017	@115712 I understand. I would like to assist y...	2	3.0
1	2	115712	True	Tue Oct 31 22:11:45 +0000 2017	@sprintcare and how do you propose we do that	NaN	1.0
2	3	115712	True	Tue Oct 31 22:08:27 +0000 2017	@sprintcare I have sent several private messag...	1	4.0
3	4	sprintcare	False	Tue Oct 31 21:54:49 +0000 2017	@115712 Please send us a Private Message so th...	3	5.0
4	5	115712	True	Tue Oct 31 21:49:35 +0000 2017	@sprintcare I did.	4	6.0
...	...	...	...	...	...	...	...
2811769	2987947	sprintcare	False	Wed Nov 22 08:43:51 +0000 2017	@823869 Hey, we'd be happy to look into this f...	NaN	2987948.0
2811770	2987948	823869	True	Wed Nov 22 08:35:16 +0000 2017	@115714 wtf!? I've been having really shitty s...	2987947	NaN
2811771	2812240	121673	True	Thu Nov 23 04:13:07 +0000 2017	@143549 @sprintcare You have to go to https://...	NaN	2812239.0
2811772	2987949	AldiUK	False	Wed Nov 22 08:31:24 +0000 2017	@823870 Sounds delicious, Sarah! 😋 https://t.c...	NaN	2987950.0
2811773	2987950	823870	True	Tue Nov 21 22:01:04 +0000 2017	@AldiUK warm sloe gin mince pies with ice cre...	2987951,2987949	NaN

Figure 2: Heads and tails for the Twitter Customer Support Corpus

3 Preprocessing dataset

We created 10k (input, response) pairs from the Twitter Custom Support Corpus by choosing the first response tweet for a given tweet, if it exists. We apply standard preprocessing techniques like switching to lower case, cleaning HTML tags, removing URLs and username (typically @user on Twitter) and non-alphanumeric characters.

4 Loading dataset

4.1 Creating vocabulary

We create a vocabulary object to store count and ID for each word present in the dataset. This object helps us to trim out the words for which count falls below a specified threshold. The vocabulary also contains three special words/tokens:

- `PAD_token` to represent padding for short sentences
- `SOS_token` to represent start of the sentence
- `EOS_token` to represent end of the sentence

4.2 Preparing data for models

We define the maximum sequence length of the models. The pairs for which either the input or response exceed this limit are removed from the dataset. However, the texts, whose lengths are less than the defined maximum length, are padded using `PAD_token`. The seq-to-seq models use these padded texts along with the padding mask.

5 Seq2Seq models

Sequence-to-sequence learning (Seq2Seq) is about training models to convert sequences from one domain (e.g. sentences in English) to sequences in another domain (e.g. the same sentences translated to French). This can be used for machine translation or for free-from question answering (generating a natural language answer given a natural language question) -- in general, it is applicable any time you need to generate text.

Formally, in the machine translation task, we have an input sequence $x_1, x_2, \dots, x_n$ and an output sequence $y_1, y_2, \dots, y_m$ (note that their lengths can be different). Translation can be thought of as finding the target sequence that is the most probable given the input; formally, the target sequence that maximizes the conditional probability:

$$p(y|x): y^* = \underset{y}{\operatorname{argmax}}\, p(y|x)$$

If you are bilingual and can translate between languages easily, you have an intuitive feeling of $p(y|x)$ and can say something like "...well, this translation is kind of more natural for this sentence". But in machine translation, we learn a function $p(y|x,\theta)$ with some parameters θ, and then find its argmax for a given input:

$$y' = \underset{y}{\operatorname{argmax}}\, p(y|x,\theta)$$

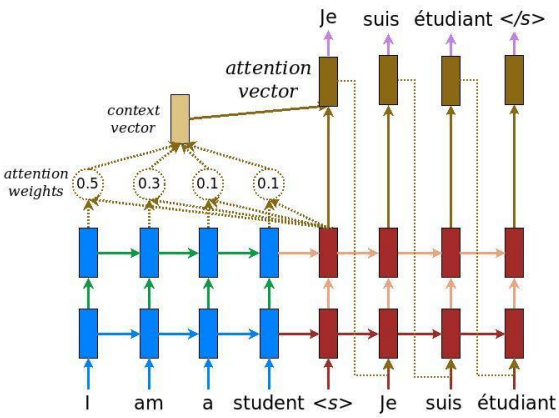

Figure 3: Seq2Seq models in machine translation

5.1 Encoder Architecture

The encoder RNN iterates through the input sentence one token at a time, at each time step outputting an “output” vector and a “hidden state” vector. The hidden state vector is then passed to the next time step, while the output vector is recorded. The encoder transforms the context it saw at each point in the sequence into a set of points in a high-dimensional space, which the decoder will use to generate a meaningful output for the given task.

At the heart of the encoder is a multi-layered Gated Recurrent Unit, invented by Cho et al.

(2014). We will use a bidirectional variant of the GRU, meaning that there are essentially two independent RNNs: one that is fed the input sequence in normal sequential order, and one that is fed the input sequence in reverse order. The outputs of each network are summed at each time step. Using a bidirectional GRU will give the advantage of encoding both past and future contexts.

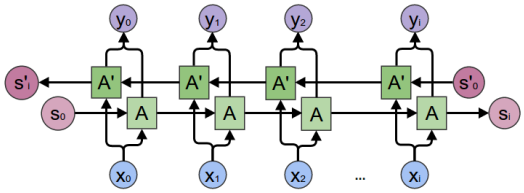

Figure 4: Bidirectional RNN

5.2 Decoder Architecture

The decoder RNN generates the response sentence in a token-by-token fashion. It uses the encoder's context vectors, and internal hidden states to generate the next word in the sequence. It continues generating words until it outputs an `EOS_token`, representing the end of the sentence. A common problem with a vanilla seq2seq decoder is that if we rely solely on the context vector to encode the entire input sequence's meaning, it is likely that we will have information loss. This is especially the case when dealing with long input sequences, greatly limiting the capability of the decoder.

To combat this, Bahdanau et al. created an "attention mechanism" that allows the decoder to pay attention to certain parts of the input sequence, rather than using the entire fixed context at every step.

At a high level, attention is calculated using the decoder's current hidden state and the encoder's outputs. The output attention weights have the same shape as the input sequence, allowing us to multiply them by the encoder outputs, giving us a weighted sum which indicates the parts of encoder output to pay attention to. Luong et al. improved upon Bahdanau et al.'s groundwork by creating "Global attention". The key difference is that with "Global attention", we consider all of the encoder's hidden states, as opposed to Bahdanau et al.'s "Local attention", which only considers the encoder's hidden state from the current time step.

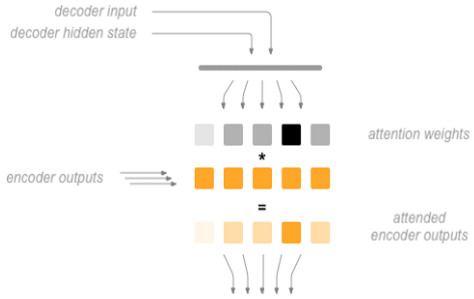

Figure 5: Attention mechanism in Bahdanau et al.

Another difference is that with "Global attention", we calculate attention weights, or energies, using the hidden state of the decoder from the current time step only. Bahdanau et al.'s attention calculation requires knowledge of the decoder's state from the previous time step. Also, Luong et al. provides various methods to calculate the attention energies between the encoder output and decoder output which are called "score functions":

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a [\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

where h_t = current target decoder state and h_s = all encoder states.

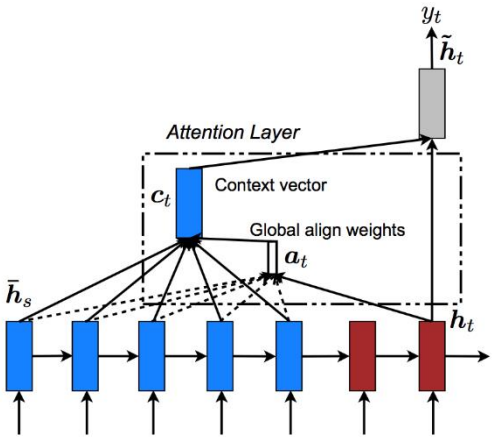

Figure 6: Attention mechanism in Luong et al.

6 Training Procedure

6.1 Masked loss

Since we are dealing with batches of padded sequences, we cannot simply consider all elements of the tensor when calculating loss. We define `maskNLLLoss` to calculate our loss based on our decoder's output tensor, the target tensor, and a binary mask tensor describing the padding of the target tensor. This loss function calculates the average negative log likelihood of the elements that correspond to a 1 in the mask tensor.

6.2 Single training iteration

The train function contains the algorithm for a single training iteration (a single batch of inputs). We will use a couple of optimizations to aid in the convergence:

- The first trick is using teacher forcing. This means that at some probability, set by `teacher_forcing_ratio`, we use the current target word as the decoder's next input rather than using the decoder's current guess. This technique acts as training wheels for the decoder, aiding in more efficient training. However, teacher forcing can lead to model instability during inference, as the decoder may not have a sufficient chance to truly craft its own output sequences during training. Thus, we must be mindful of how we are setting the `teacher_forcing_ratio`, and not be fooled by fast convergence.
- The second trick that we implement is gradient clipping. This is a commonly used technique for countering the "exploding gradient" problem. In essence, by clipping or thresholding gradients to a maximum value, we prevent the gradients from growing exponentially and either overflow (NaN), or overshoot steep cliffs in the cost function.

Sequence of operations:

- Forward pass entire input batch through encoder.
- Initialize decoder inputs as SOS token, and hidden state as the encoder's final hidden state.
- Forward input batch sequence through decoder one time step at a time.
- If teacher forcing: set next decoder input as the current target; else: set next decoder input as current decoder output.
- Calculate and accumulate loss.
- Perform backpropagation.
- Clip gradients.
- Update encoder and decoder model parameters.

PyTorch's RNN modules (RNN, LSTM, GRU) can be used like any other non-recurrent layers by simply passing them the entire input sequence (or batch of sequences). We use the GRU layer like this in the encoder. The reality is that under the hood, there is an iterative process looping over each time step calculating hidden states. Alternatively, you can run these modules one time-step at a time. In this case, we manually loop over the sequences during the training process like we must do for the decoder model. As long as you maintain the correct conceptual model of these modules, implementing sequential models can be very straightforward.

6.3 Training iterations

It is finally time to tie the full training procedure together with the data. The `trainIters` function is responsible for running `n_iterations` of training given the passed models, optimizers, data, etc. One thing to note
is that when we save our model, we save a `tarball` containing the encoder and decoder `state_dicts` (parameters), the optimizers' `state_dicts`, the loss, the iteration, etc. Saving the model in this way will give us the ultimate flexibility with the checkpoint. After loading a checkpoint, we will be able to use the model parameters to run inference, or we can continue training right where we left off.

7 Evaluation

After training a model, we want to be able to talk to the bot ourselves. First, we must define how we want the model to decode the encoded input.

7.1 Greedy decoding

Greedy decoding is the decoding method that we use during training when we are NOT using teacher forcing. In other words, for each time step, we simply choose the word from `decoder_output` with the highest softmax value. This decoding method is optimal on a single time-step level.

To facilitate the greedy decoding operation, we define a `GreedySearchDecoder` class. When run, an object of this class takes an input sequence (`input_seq`) of shape (`input_seq length`, 1), a scalar input length (`input_length`) tensor, and a `max_length` to bound the response sentence length. The input sentence is evaluated using the following computational graph:

Computation Graph:

- Forward input through encoder model.
- Prepare encoder's final hidden layer to be first hidden input to the decoder.
- Initialize decoder's first input as `SOS` token.
- Initialize tensors to append decoded words to.
- Iteratively decode one word token at a time:
    - Forward pass through decoder.
    - Obtain most likely word token and its softmax score.
    - Record token and score.
    - Prepare current token to be next decoder input.
- Return collections of word tokens and scores.

7.2 Evaluating given text

The evaluate function manages the low-level process of handling the input sentence. We first format the sentence as an input batch of word indexes with `batch_size` of 1. We do this by converting the words of the sentence to their corresponding indexes, and transposing the dimensions to prepare the tensor for our models. We also create a lengths tensor which contains the length of our input sentence. In this case, `lengths` is scalar because we are only evaluating one sentence at a time (`batch_size`=1). Next, we obtain the decoded response sentence tensor using our `GreedySearchDecoder` object (`searcher`). Finally, we convert the response's indexes to words and return the list of decoded words.

`evaluateInput` acts as the user interface for our chatbot. When called, an input text field will spawn in which we can enter our query sentence. After typing our input sentence and pressing Enter, our text is normalized in the same way as our training data, and is ultimately fed to the evaluate function to obtain a decoded output sentence. We loop this process, so we can keep chatting with our bot until we enter either "q" or "quit". Finally, if a sentence is entered that contains a word that is not in the vocabulary, we handle this by printing an error message and prompting the user to enter another sentence.

8 Steps to execute the code

To reproduce the results, run the `notebook.ipynb` in CUDA-based environment. To download the dataset, you will need to generate access API via Kaggle. This file should be named as `kaggle.json`. Visit this page for more detail on how to create access API: https://github.com/Kaggle/kaggle-api

9 Observations

From Table 2, it can be observed that the responses we get from this seq2seq based chatbot are exceptionally well written and are relevant to the queries. This feat can be attributed to the attention mechanism used in Luong et al.

10 Results

Query	Response
thank you I updated my phone and now it is even slower and barely works	wed like to take a closer look into whats happening with the iphone please let us know if you have any questions mm well be happy to look into this for you
i have iphone 6s plus and just did the most recent update	to make sure is ios installed on it currently also any steps tried so far dm us here well get started
battery on iphone is very bad	wed love to look into this for you now you are still experiencing down services let us know if you need further assistance nr for your file a claim for your shipping info for your file a claim
i have messaged you resolve fast	if you have any questions in the future please feel free to reach back out if we can help with any assistance
delivery guy threw my parcel	sorry of the frustration who was the carrier of the package you can find that info here ad well know in touch they will be in touch in the store sophie they will be able to help
my uber driver charged me extra i want a refund	were sorry to hear this was your experience send us a note via so we can assist in the link for your account so we can assist
im trying my fire stick help please	please reach out to us for realtime troubleshooting here wt order details here well be happy to look into this for you
very bad internet provider slow internet	hello would you mind following us and sending a dm when you are so we can gather more info please, for your account if we can help with any connectivity issues
xfinity tv service issues	hey whats up as your service device and see what we can do to help please dont hesitate
thanks for your service really fast	hello thanks for reaching out about your feedback and we will follow up with you soon

Table 2: Conversations with Jarvis

11 Individual Contribution

Member	Contribution
Utkarsh Patel	Data extraction, data analysis, data pre-processing and report making
Devansh Rajgarhia	Model training and report making
Awadh Kejriwal	Model evaluation and testing
Abhisek Mohanty	Report making

Table 3: Contribution by members