# Comparison of Rhymer Chatbots Based on HMM and RNN

**Anastasiya Chernova**
Innopolis University
a.chernova@innopolis.ru

**Lidiya Edilgiriyeva**
Innopolis University
l.edilgiriyeva@innopolis.ru

**Abstract**

In this paper we present two Telegram chatbots which create meaningful rhymes to the user inputs. Two approaches were used to generate sentences: Hidden Markov Model (HMM) and RNN Neural Network. Then we show a comparison of answers of bots on the same words or phrases.

## 1 Introduction

In the first part of Methodology section we describe the rules by which bots generate sentences. Then, we discuss details of our Hidden Markov Model, approaches to create sentences and description of data preprocessing. We present a general diagram of the HMM workflow. Second part of the Methodology section describes architecture of Recurrent Neural Network. We describe two neural networks which were created: char-based and word-based NN. Then we discuss the results and make conclusions of the work.

## 2 Methodology

### 2.1 Rhyme rules

Since a chatbot generates a rhyme to an input sentence it should cover following properties:

1. Amount of vowel sounds of an input.

2. The last words should make a rhyme

Input sentence is processed with pronouncing library[3] which gives an opportunity to read a transcription of words. For example transcription of *salary: Transcript ['S AE1 L ER0 IY0'].*

As it can be seen vowel sounds are pointed with digits - used as stress markers. From transcription we are able to extract information about amount of sounds in a sentence - so that generated and input texts would have harmonic soundness.

Beside, pronouncing library gives opportunity to find rhymes to a word. Example rhymes to awake word: *'ache', 'ake', 'bake', 'betake', 'blake', 'brake', 'break', 'cake', 'drake', 'fake' etc.*

Unfortunately the library is not full and contains small set of words. We used CMU pronouncing dictionary to solve this problem. NLTK[4] contains cmudict with information about over 134000 words and their pronunciations. Method nltk.corpus.cmudict.entries() returns a tuple with a word and its transcription. This method is used to find most similar words to the target one. Our function takes a target word and level of similar sound (such as last N letters should sound similarly) as arguments. This method recognizes whether the endings of words transcriptions are similar (the sounds are the same) and returns the set of words which rhyme with the target word.

### 2.2 Hidden Markov Model

#### 2.2.1 Markovify Library

This section describes sentences generation using HMM. First, the library for building models - markovify[1] was found. We chose this library for the following reasons:

1. Extensible Markov Model Library

2. Well documented code

3. Models are serializable

4. A good thing to start with

#### 2.2.2 Model modifications

Markovify gives opportunity to modify its methods and classes. Following classes were modified: Text, NewlineText and Chain.

Chain class has methods to serialize a model as JSON file, walk through the model moving from state to state. Model is represented as a dictionary of dictionaries where the keys of the outer dict represent all possible states, and point to the inner dictionaries. Also, it contains the number which indicates amount of integer values which represent each state. Empirical results have shown that three states has best results in our case. We override the gen() method and put a restriction on amount of generated words, so that output sentence should not be much longer than input.

Beside, there are two classes in the library which are used to create a model which have similar functionality with minor modification. Text class generates sentences using Chain class functions, processes input and output sentences. It takes sentences separated by dots - main restriction of the library is that a dataset which forms a model should be well-punctuated. In contrast, NewlineText class has the same functionality except for this class accepts text file with every sentences separated by a new line.

In this classes we modify whole generation of sentences. We create a function which accepts a set with possible last words which are sent by a rhyme() method. Then, program generates sentences from the end to a beginning. This means, that our model is reversed - building phrases from the end to the start. Sentence generation creates lots of sentences which are processed by our function sentence_sounds which returns amount of vowels which represent a sound in a sentence. If it generates the sentence with the same sound number - then it chooses it, instead, it picks output randomly.

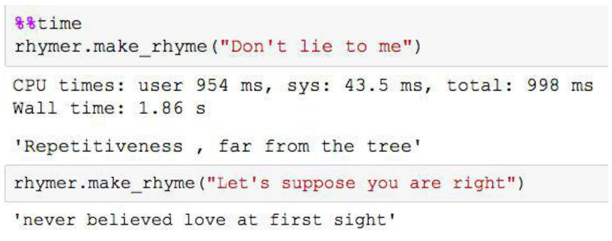
#### 2.2.3 Data preparation

Any input data is preprocessed by converting it to lowercase letters, reverse order so that our model is able to generate sentences from the last word, as well as punctuation was removed.

We tried three different datasets for sentences generation. First one is movie reviews data from nltk.corpus library. Generated sentences were grammatically correct but some of them did not have any meaning. There are two examples of first attempts:

But anyway there were interesting results:

Then, we parsed a movies dialogs dataset[5]. A .tsv was preprocessed to get only texts of dialogs. It turned out that messages became less attractive



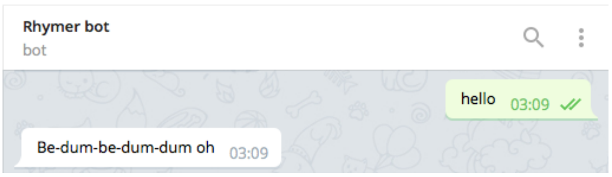
```
%%time
rhymer.make_rhyme("Don't lie to me")

CPU times: user 954 ms, sys: 43.5 ms, total: 998 ms
Wall time: 1.86 s

'Repetitiveness , far from the tree'

rhymer.make_rhyme("Let's suppose you are right")

'never believed love at first sight'
```

Figure 1: The first results

and grammatically inaccurate, that is why it was decided to use another dataset.

Finally, it was decided to find a dataset with small sentences and some meaning. Dataset with most popular songs was chosen to create a model[6]. There was a csv file with information about signer, date, title of a song, and text. We took only text data and preprocessed it, so that every sentence would be in a separate line.

One of the examples of the last version:



Rhymer bot
bot

hello 03:09

Be-dum-be-dum-dum oh 03:09

Figure 2: A chat with the telegram bot

This diagram shows common workflow of our model:

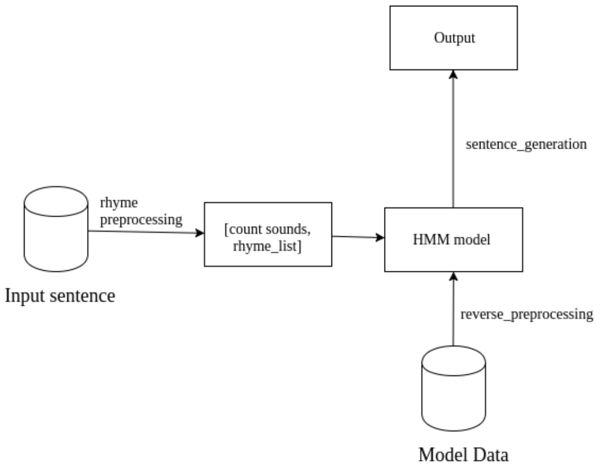


Figure 3: HMM work-flow

### 2.3 Recurrent Neural Networks (RNN)

For our goal we need not only to find rhyme for the last word, but also generate a sentence that contains this word. RNN is common way to generate sequences. We use LSTM neural networks to create a sequence of words or chars. Both char and word-based model were tested. They were built using keras library with theano backend. The neural networks use for training movie reviews data from nltk library.

#### 2.3.1 Word-based Neural Network

Gutenberg corpus data was preprocessed for this kind of RNN. Each sentence was tokenized and every token was replaced with its own unique index using prepared dictionary. Then pairs that contain train sample and target word index were constructed from each sentence. A train sample is sequence of L word indices and target word index is L+1 word index in the sentence. Hot-encoding is applied to target word indexes. In other words NN learns to predict L+1 word index using previous L ones. Due to sparsity of train matrix word2vec method was applied to input matrix using Embedding layer for NN.

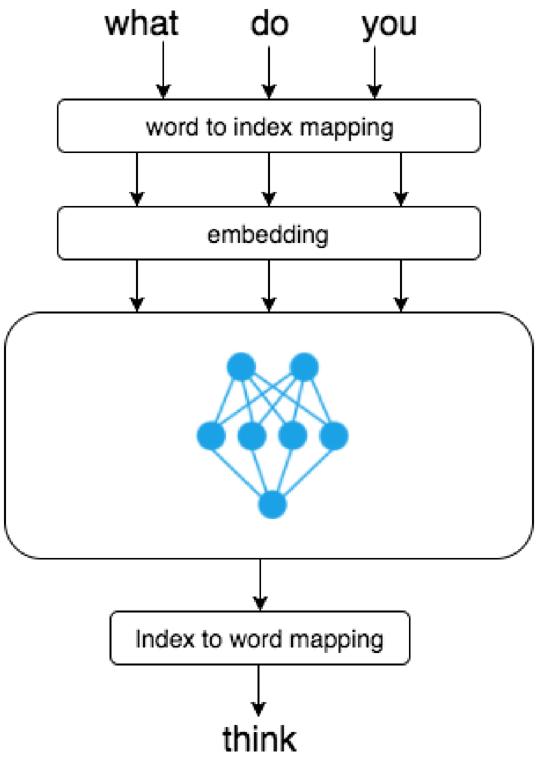


Figure 4: Word-based model work-flow

The one significant modification is made: in order to generate sentence from the last word, LSTM learns to predict previous word for given one. That is why the input text was reversed before preprocessing.

The result of this model varies depending on the $L$ parameter. If $L$ is small there are loops in generated sentence. However, it allows to train RNN for acceptable time and, consequently, to conduct more experiments with the model. If $L$ is large it requires a lot of time for learning and there is a small number of experiments with parameter that are conducted.

The result of this model varies depending on the $L$ parameter. If $L$ is small there are loops in generated sentence. However, it allows to train RNN for acceptable time and, consequently, to conduct more experiments with the model. If $L$ is large to be continued (experiments will be conducted soon)

#### 2.3.2 Char-based Neural Network

The major advantage of char-based RNN for our goal is that we can use only several last symbols to generate a sentence. These last symbols can be chosen to make a rhyme to an input sentence. Input data this RNN was prepared differently. We used also sequence of $L$ word indexes and input train sample. However, as target sample, sequence of $L$ indices shifted by 1 position were used.

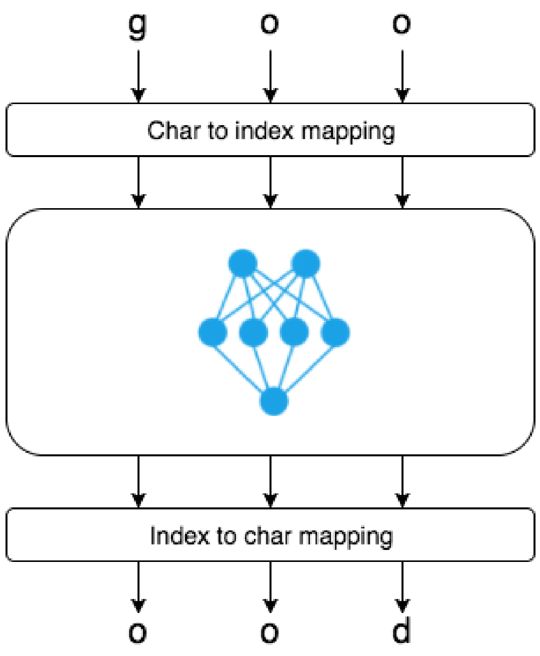


Figure 5: Char-based model work-flow

The result sentence also depends on $L$ parameter and often contains loops. However, it is typical behavior for RNN with large $L$ that is about 40 symbols. Nevertheless, model with small $L$ parameter is prone to creating new nonexistent words.

### 2.3.3 Rhyme making

A rhymed sentence is generated using RNN by accepting it start-point parameter. In case of word-based LSTM it is a word that is rhyme for last input word. In case of char-based it is several last chars that are needed to make rhyme. There are several restriction for generated rhymes, as an example, the generated sentence should not be too long.

## 2.4 Telegram Bot

Since this is a chatbot, it was decided to create a bot for Telegram. TeleBot API was used to connect models to a bot. This API gives opportunity to process user inputs and send messages back. The alias in telegram is: *@rhyme_maker_bot.*

The bot uses two models to make rhymes for user messages: Hidden Markov model and word-based LSTM Neural Network. A user can choose needed model using special commands: */hmm* for Markov model and */word_rnn* for RNN model. A user can compare both ways for sentence generating.

## 3 Results and Discussion

We studied how to generate text using Hidden Markov Models and RNN. As a result of out work we have a telegram bot which generates rhymes according to the last word of user input. The bot prints meaningful sentences as well as texts with grammar mistakes. The reasons why sometimes output of the program is not linguistically correct can be different: the dataset should contain short phrases, more information about words pronunciations should be used.

## 4 Conclusions and Future work

Two approaches were used to create rhymer bots: using RNN and HMM. Further we would like to enhance our rhyme rules, as well as improve our bot so that it could generate sentences using context of conversation.

| **Last word** | **RNN** | **HMM** |
|---|---|---|
| Hello | But i of it the glow | Future is casting a shadow no |
| Yes | If it was in this film of this film and for a suppress | - |
| Bad | The ad | Guy closing your front door had |
| Bed | They read | Andante tread |
| Book | And look | Means something special to me look |
| Mood | That it would be reviewed | Maybe you'd see me, baby you'd |
| True | And that it is not woo | And the pain will end you |
| Go | The from is below | Get love in return so |

Figure 6: Experiments

## 5 References

1. https://github.com/jsvine/markovify
2. https://pypi.python.org/pypi/pyTelegramBotAPI
3. https://pypi.python.org/pypi/pronouncing
4. http://www.nltk.org/
5. https://www.kaggle.com/Cornell-University/movie-dialog-corpus
6. https://www.kaggle.com/mousehead/songlyrics/data