Chatbot using Seq2Seq Model in Python using Tensorflow

IDS – 576|Spring-2018

Advanced Predictive Modelling

Instructor:

Theja Tulabandhula

ASHISH SHINDE (650541070)

KOFFI AGBAVO (667058211)

SHRINIVAS KALLOL (661625322)

TABLE OF CONTENTS

1- **Motivation, Business goal, application and significance**

In today's fast paced world, technology has revolutionized the way we do business. Chatbots appear to be a powerful means of automating customer interactions in enterprise. In today's economy, the high competitiveness among businesses requires a more effective ways to cut the costs and provide customers with greater experience.
A chatbot is a program designed to simulate a conversation with a human being over the internet. We can distinguish two main types of chatbots: one type functions based on a set of rules and the other one is more advanced and uses machine learning. Our project is on the one using machine learning to mimic real conversations.

An employee's job effectiveness depends on his or her knowledge of the job. Similarly, the effectiveness of a chatbot solution depends on its knowledge base and ability to learn. Knowing this allows us to make sure we fill the chatbot with our personality (in case of a personal assistant), our brand's identity and make it speak to our customers like we would, change its message depending on the input from the user. This changes the whole game of customer support and brings it to a more autonomous level.

According to Suhas Ulliyarı, oracle's VP of bots and AI, there is a rapid adoption of AI chatbot technology among enterprise customers for the following key factors:

- Chatbots provide strong ROI by automating some of the repetitive end-user interactions, leaving the customer service agents to focus on high-value and high priority customers.

- Chatbots are a great use case for adopting AI in the enterprise.

- There is little to no extra cost to support many chat and voice channels, so businesses can reach more users without extra cost.

A chatbot solution's implementation has a huge advantage especially in customer support by allowing labor efficiencies, enhanced customer experience, improved productivity and drives important cost saving. It's important to highlight the fact that a well-trained chatbot that answers all the questions it has been taught and which is always available 24/7 to respond to customer questions enables the business to provide higher quality support with less investment.

2- **Literature and Background**

Neural Machine Translation (NMT) was the very first testbed for seq2seq models. It all began after the birth of Neural Machine Translation in 2013, [1] Nal Kalchbrenner and Phil Blunsom proposed a new end-to-end encoder-decoder structure for machine translation. This model will encode a given source text into a continuous vector using Convolutional

Neural Network (CNN), and then use Recurrent Neural Network (RNN) as the decoder to transform the state vector into the target language. In the following year the first ever paper on seq2seq learning was published by scientists at Google [2]. The research paper talked about overcoming the limitations of a Deep neural networks, DNNs back then could only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality, since many important problems are best expressed with sequences whose lengths are not known a-priori this becomes a serious limitation for the DNN's. Sequence-to-sequence (seq2seq) models have enjoyed great success in a variety of tasks such as machine translation, speech recognition, and text summarization. In the recent years a lot of research has been concentrated in improving the attention mechanism and memory of the seq2seq models.

Between 2010 and 2016, we have seen the raise of virtual personal assistants such as Siri (Apple), Google Now, Alexa from Amazon, Cortana from Microsoft and Google assistants (2016)[3]. These assistants are connected to web services and use natural language processing to respond to questions. Early 2016, we have seen the introduction of the first wave of artificial intelligence technology in the form of chatbots. Most of those bots perform customer support or customer engagement functions answering questions and offering products suggestions for purchase.

Like we mentioned earlier that the effectiveness of a chatbot depends on its knowledge base and its ability to learn.

3- **Dataset Description**

For the project, we have used Cornell Movie-Dialogs Corpus to train our chatbot. This dataset contains 220,579 conversational exchanges between 10,292 pairs of movie characters [4]

- 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)

The 2 files used were:

1. ‘movie_lines.txt’ which had format

→ L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ they do not!

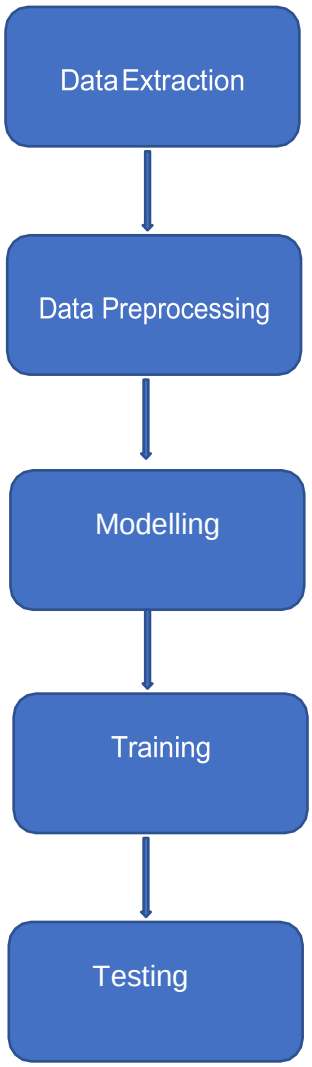
L1045 refers to the line id’s, u0 refers to the user number (movie character number), m0 refers to the movie number, ‘BIANCA’ is the name of the movie character and ‘They do not!’ is the line said by the movie character.

2. ‘movie_conversations.txt’ which had the format

→ u0 +++$+++ u2 +++$+++ m0 +++$+++ 'L194', 'L195', 'L196', 'L197'

Here the conversation is between characters u0 and u2, 'L194', 'L195', 'L196', and ‘L197’ refers to the lines in the conversation between characters and m0 refers to the movie. The actual lines are available in the movie_line.txt, for instance L1045 refers to the line ‘They do not!’

4- Methodology and Statistical Methods



4.1- Data Extraction

The Data was downloaded from the following link:

http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

The two files that were needed 'movie_lines.txt', 'movie_conversations.txt were downloaded and read in Python as text files.

4.2- Pre-Processing

Firstly, there was a mapping created between the movie lines id's and the actual line. The lines in the movie_lines.txt file were split on '+++$+++'. Then, a list was created which had all the conversation ids in a sub list from movie_convesations.txt file. Separating out the questions and answers from the conversation id list; the conversation id's list which looked like [['L194', 'L195', 'L196', 'L197'], ...] , the first element of the first sub list is the question 'L194' and the second element of the sub list ''L195'' is the answer or reply to that question, these are separated out in two different lists by questions vs answers function. Now, we have actual questions and answers in two lists. These lines were cleaned using re library, all the punctuation marks were taken off, the short hand for words like 'I've', 'won't' were replaced with 'I have' and 'would not' respectively. Questions and answers that were either too short (length less than 2) or that were too long (length greater than 25) were filtered out by encoder function. Then a dictionary was created that maps each word to its number of occurrences. Created two dictionaries that map the questions words and the answers words to a unique integer, adding the last tokens '<PAD>', '<EOS>', '<UNK>', '<GO>'to these two dictionaries by using vocab function. <PAD> was used for padding the questions and answers as all the questions and answers were of different lengths. Created the inverse dictionary of the dictionary that contained answer words and unique integers as key and value, added the End of String token to the end of every answer this was done by pad data function. <EOS> is used after end of each line to indicate that the line has ended, our model thus knows when a line has ended in this manner. All the words in the corpus which had an occurrence of less than 15 were replaced by the <UNK> token. Sorted questions and answers by the length of questions. The <GO> token was added at the beginning of every answer. Bucket data function was used to create buckets of our data where we put sentences into buckets of different sizes

4.3 - Modelling

<u>Seq2Seq Model and its components</u>:

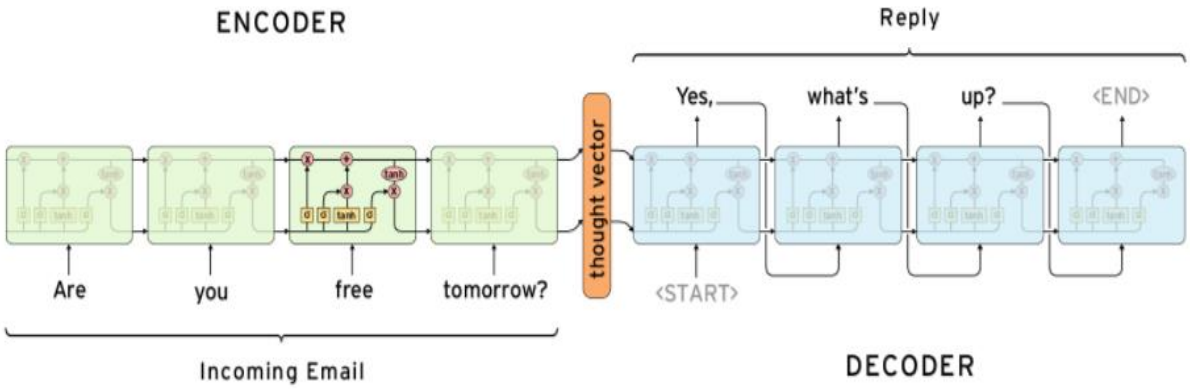
Seq2seq model has become the Go-To model for Dialogue Systems and Machine Translation. It consists for two RNN's an encoder and a decoder.

Encoder

The encoder takes a sequence (in our case a sentence) as input and processes one word at each time step. Its objective is to convert a sequence of symbols into a fixed size feature vector that encodes only the important information in the sequence while losing the unnecessary information. Each hidden state influences the next hidden state and the final hidden state becomes the summary of the sequence. This state is called the context or thought vector, as it represents the intention of the sequence.

Decoder

The decoder generates another sequence, one symbol (word) at a time from the thought vector. Here, at each time step, the decoder is influenced by the context and the previously generated symbols. Two RNNs are trained jointly to maximize the conditional probability of the target sequence given a source sequence.



However, there are a few challenges that come with using this model, the model cannot handle variable sequence length, and another one is the vocabulary size. The decoder must run softmax over a huge vocabulary of words and assign a probability value to each of those words, due to this the training process takes a lot of time.

Padding

It is a part of preprocessing wherein we convert the variable length of sequences (sentences) into a fixed length sequence. This is process is known as padding. Padding depends on the largest sentence in our corpus.

For example, we have the following question-answer pair:

Q: How are you?

A: I am fine.

If we have set our fixed length of Question and Answers to be 10, then this will be converted into

Q: [PAD, PAD, PAD, PAD, “PAD”, “PAD”, “?”, “you”, “are”, “How”]

A: [“GO”, “I”, “am”, “fine”, “.”, “EOS”, “PAD”, “PAD”, “PAD”, “PAD”]

Bucketing

Padding isn’t the most efficient way of handling variable length, for instance say that the length of the longest sentence in our corpus is 100, if we consider the question “how are you ? ” which is of length 4 there will be 96 symbols of PAD in encoded version of the sentence. This will overshadow the actual information in the sentence.

Bucketing kind of solves this problem, by putting sentences into buckets of different sizes. Considering 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 (as in the above example), 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. All these models share the same parameters and hence function the same way.

If we were using the bucket (5, 10), our sentences will be encoded to

Q: [PAD, “?”, “you”, “are”, “How”]

A: [GO, “I”, “am”, “fine”, “.”, EOS, PAD, PAD, PAD, PAD]

Word Embedding

Word Embedding is a technique for learning dense representation of words in a low dimensional vector space. Each word be 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.

Attention Mechanism

At each step the hidden state h is getting updated with the most recent information, and therefore h might be getting "diluted" in information as it processes each token. Even with a relatively short sequence, the last token will always get the last say (will be relatively more important) and therefore the thought vector will be somewhat biased/weighted towards that last word. To deal with the problem, we use an attention mechanism. The attention mechanism allows the decoder to put different amounts of attention on all the hidden states produced by the encoder and use whatever information it most needs for the prediction.

An attention vector is computed at each output time t over the input words. We compute a vector ut with length T (where there are T tokens in the input sequence) and the *i*-th entry contains a score telling how much attention should be put on the *i*-th hidden state of the encoder hi:
$u\ (i,\ t) = vT\ tanh(W1\ hi + W2\ dt)$
where v, $W1$, and $W2$ are learnable parameters of the network and dt is the (original) hidden state of the decoder at time t.

MODELLING STEPS

- The architecture of the seq2seq model was built and saved in models_utils.py where we had defined various functions which are later used in the class 'chatbot'. The 'graph input function' is where we created placeholders for the inputs, target, encoder sequence length, decoder sequence length, and keep probability (Dropout) because all the variables used are nothing but tensors; hence we define the tensorflow placeholders, placeholders are the initial step of building a computation graph. Creating these placeholders help us use these variables in the future for training.
- The 'cell' function helped us introduce the basic Lstm Cell and Dropout (That is deactivating a certain percentage of the neurons during the training iterations) for these LSTM cells.
- This architecture was twofold. We first created the encoder RNN layer because this is what comes first in the architecture of the seq2seq model and then we created the decoder RNN layer.
- In the 'encoder' function we created multi-RNN cell which had 3 lstm layers introduced in cell function and then created 'encoder embedding' variable which converts the words that are previous assigned unique integers to its embedding of given encoder embedding size. We then created dynamic RNN which returns encoder state and encoder outputs.
- The decoder only accepts a certain format of the targets. The neural network will not accept targets/answers one by one, they will only accept them in

batches/buckets. For example, say 10 answers at a time. The answers in the batch of targets must start with the 'GO' token, the function 'Decoder inputs preprocessing' carries out this step.

- In the 'decoder' function decoding takes place. At the beginning we created embeddings for decoders using embedding lookup (The class from tensorflow.nn). Then we connected Dense layer (Fully Connected) at the end of the Decoder which was used for generating probabilities for each word in the vocabulary. With variable scope we introduced 'Training helper' which is used only to read inputs in the training stage. We defined decoder using 'tf.seq2seq.dynamic_decoder' sub-class which returns train decoder outputs. We used 'REUSE' option in tf.variable scope as we wanted to get the same parameters learnt in the previous 'decoder' scope, later we got the vector of the '<GO>' tokens in the integer representation. We used 'basic greedy' technique (tf.contrib.seq2seq.greedyembeddinghelper) to get next word in the inference time (based only on probability). This 'basic greedy' method returns train decoder outputs and inference decoder outputs.

- In attention mechanism function we first defined a RNN cell, then using helper function from seq2seq sub lib for Bahdanau attention we created attention mechanism, then we ended the attention with the attention holder (Attention Wrapper). We used zero state of LSTM of the decoder cell, and fed the value of the last encoder state to it.

- We defined 'optimizer loss function', where we get the loss and training optimizer of sequence to sequence model. We applied gradient clipping to the optimizer to avoid exploding or vanishing gradient issue. This is a technique with operations that will cap the gradient in the graph between a minimum value and a maximum value. We defined a new scope which contains two elements that are used for the training, they were 'loss error' and 'the optimizer' with grading tipping. The loss error is going to be based on a weighted cross entropy loss error which is the most relevant loss to use when dealing with sequences, and in general when dealing with deep NLP. The optimizer that we used was 'adam optimizer' which is one of the best optimizers to use for stochastic gradient descent.

- Finally, we created a class called chatbot. Initially, we got all the variables to be used in training using 'graph inputs', then we got the outputs from the functions encoder, decoder inputs preprocessing, attention mechanism, decoder and optimizer loss.

Training - Config.py-setting the hyper parameters

- The number of epochs were 50, each epoch is forward propagated inside the encoder to get the encoder state, then forward propagating this encoder state with the target inside the decoder RNN. We then get the final output that is the final answer predicted by chatbot. Backpropagation of the loss generated by outputs and the targets are sent back into the neural network, where it updates the weights to better direction such that the chatbot speaks like human.

- Then we selected buckets of [(10, 15), (15, 25), (25, 45), (45, 60), (60,100)].

- The RNN size we choose was 512 that is there are 512 neurons in each LSTM layers.

- The number of LSTM layers in encoder and decoder had 3 layers each.

- The encoding embedding size and decoding embedding size are the number of columns in our embedding matrix. The number of columns we want to have for the words in the line were 512. In this matrix each line corresponds to each token in the whole corpus.

- We choose our learning rate to be 0.001 after trial and experimentation.

- We implemented time-based decay learning rate over the iterations of the training to reduce the learning rates; so that it can learn in depth the logic of human conversations. We choose a decay of 90 percent - 0.9.

- Drop out which is a simple way to prevent neural networks from overfitting. By referring to the paper published by Geoffrey Hinton [3] we choose dropout rate to be 0.5.

- We imported all the .py files, we obtained the cleaned questions, cleaned answers, vocab, word to id, id to word, encoded questions, encoded answers and bucketed data from these files.

- Using the Chatbot class and config file we obtained our model, then we initialized the session and ran it by initializing all the global variables.

- Then we trained our model for 50 epochs which had 5 buckets in each epoch. We saved the weights of our model after each epoch.

Testing:

- The weights were saved at the end of the training in '.ckpt' format, we first load the weights and run the session.
- We then convert the questions from strings to lists of encoding integers.
- Using a While loop we set up the chatbot, where there is an input prompt available for the user to type in the question. The chatbot replies with the answer in the form of text. If the user types in 'Goodbye', the while loop stops and breaks out of the question and answer schema.

What went wrong: We underestimated the amount of time required for the chatbot to train and perform like a real world chatbot, the corpus contained 221,616 lines which apparently was too much to process for our local machine. We installed NVDIA computing tools and used the 4 Gb GPU to train our bot, we also used google collab to train our model and save the weights of our model. However, the performance of the chatbot was poorer than expected.

What went right: We were successfully able to build seq2seq model with attention mechanism, we created class called 'chatbot', this allows us to deploy our chatbot into an app or a web browser easily. We were able to handle and resolve the degraded sub-classes in the tensorflow seq2seq class.

What can be improved: The chatbots response could be close to a real-world chatbot only if we train it for a long period of time (at least 100 epochs for our data), the chatbot can be converted into a speech-to-text bot using Google speech API or IBM Watson Watson API. The model can also be integrated with memory networks like end to end memory networks where the model learns to focus on certain words while generating answers. The RNN's could be replaced by GRU's to speed up the process of training without compromising much on the accuracy of the chatbot.

5- **Achievements and Results**

We initially trained the chatbot on the entire Dataset and set the epochs to be 20, the learning rate used was 0.0001. It took around 2 and half days to run on the local machine. The chatbot only responded back with the same answer to anything that was typed in as the question.

```
You: Hi
Chatbot: I dont know
You: how are you doing?
Chatbot: I dont know
You: Are you ok?
Chatbot: I don t know
You: you are crazy
Chatbot: I dont know
You: what is your name?
Chatbot: I don t know
You: Lets dance
Chatbot: I dont know
You: Goodbye
```

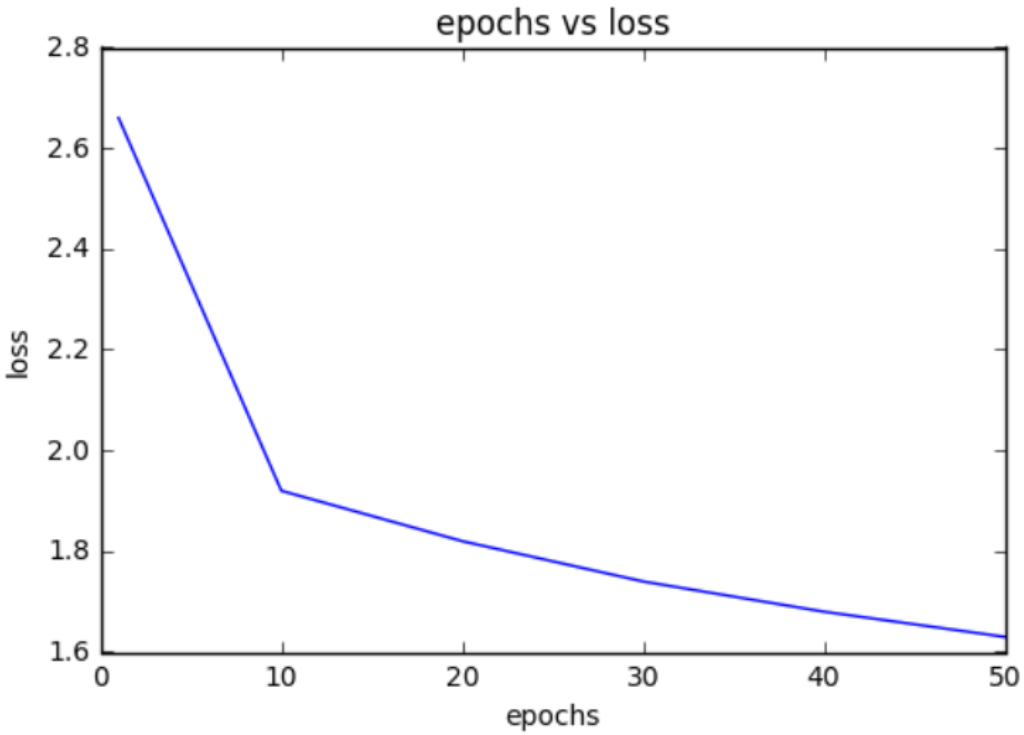
We trained on 20k lines out of the 220,579 lines (10% of the Data) for 50 epochs with 0.001 as the learning Rate. It took around 30 hours to run on the local machine and the chatbot only replied with gibberish to anything it was asked.

```
You: hi
Chatbot: four years ago I was working in a bakery four years ago I was working in a bakery four years ago I was working in a ba
kery
You: What is your name
Chatbot: four years ago I was working in a baker y four years ago I was working in a bakery  four years ago I was working in a
bakery
You: Who is the president of the United States?
Chatbot: four years ago I was working in a bakery four years ago I was working in a bakery  four years ago I was working in a b
akery
You: What is up with you?
Chatbot: four years ago I was working in a bakery  four years ago I was working in a bakery  four years ago I was working in a
bakery
You: What is your favorite color?
Chatbot: four years ago I was working in a bakeryfour years ago I was working in a bakery four years ago I was working in a bak
ery
You: Goodbye
```

We then tried to train the chatbot on the entire Dataset and set the epoch value to be 50 implemented time-based decay learning rate over the iterations of the training. The chatbot took nearly 50 hours to train, the result was not as good as we anticipated. As the chatbot was not able to interpret most of our questions.

```
You: Hi
Chatbot: The water is out  .
You: How are you doing?
Chatbot: I dont know .
You: Are you ok?
Chatbot: I dont know .
You: I need your help
Chatbot: okay
You: I need you to sing for me
Chatbot: I dont understand
You: Who is the president
Chatbot: Why you ask
You: Goodbye
```

The following is the Graph of Epoch vs Loss for the above Model:



Key Insights: We underestimated the amount of time it would take to train the chatbot, reducing the number of lines in the dataset did not help our cause. Using a higher learning rate to speed up the computations was no good either. We assume that only by training the Chatbot for a long period of time (100 epochs at least) we will be able to build a chatbot that can have a conversation like an actual Industry level chatbot. We also believe that GRU's would work best for the model as they train faster than the LSTM's, GRU's supposedly also generalize better than the LSTM's.

6. Creative contributions

- To improve the computational efficiency bucketing was used to handle the data in an efficient manner [(10,15), (15,25), (25,45), (45,60), (60,100)].
- The words that had the word count less than a threshold value (20 in our case) were replaced by the token <GO>. If we pick a number too high as the threshold value the model will lose its ability to interpret the sequence as it will have to deal with a lot of <GO>.

- We applied gradient clipping to the optimizer to avoid exploding and vanishing gradient issue.
- We used time-based decay learning rate of 0.9, to improve computational efficiency.
- We made use of TrainingHelper and GreedyEmbeddingHelper. TrainingHelper is used at the training time, when (one of the) inputs to your decoder RNN is the ground truth from the previous time step.

7. References

- http://www.aclweb.org/anthology/D13-1176
- http://emnlp2014.org/papers/pdf/EMNLP2014179.pdf
- https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf
- https://arxiv.org/pdf/1506.05869.pdf
- http://suriyadeepan.github.io/2016-06-28-easy-seq2seq/
- http://www.parlo.io/blog/whats-the-big-deal-with-ai-chatbots-and-intelligent-enterprise-automation
- http://letzgro.net/blog/9-reasons-to-build-a-chatbot-now/
- https://venturebeat.com/2016/08/15/a-short-history-of-chatbots-and-artificial-intelligence/
- [4] http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html
- https://www.infoworld.com/article/3230370/artificial-intelligence/before-you-build-your-bot-what-it-takes-to-make-a-successful-chatbot.html
- https://medium.com/swlh/what-is-a-chatbot-and-how-to-use-it-for-your-business-976ec2e0a99f