**Introduction**

In this exercise we develop and run a chatbot. A chatbot is a piece of software with artificial intelligence on a device (Siri, Alexa, Google Assistant, etc.), application, website or other networks that try to measure consumer needs and then help them execute a particular work such as a commercial transaction, hotel bookings, bookings, etc. Today, almost every company has a chatbot developed to work with users. Some of the ways companies use chatbots are [1]:

* To deliver flight information
* to connect customers and their finances
* As customer support
* The possibilities are (almost) limitless.

**How do Chatbots work?**

There are generally two variations of chatbots: Rule-based and self-taught.

1. In a rules-based approach, a bot answers questions based on specific rules in which it is trained. The standards set can be very simple to very complex. Bots can handle simple queries but fail to manage complexly.

2. Self-learning bots are those that use some machine learning-based approaches and are more effective than rule-based bots. These bots can be of two additional types: Retrieval Based or Generative

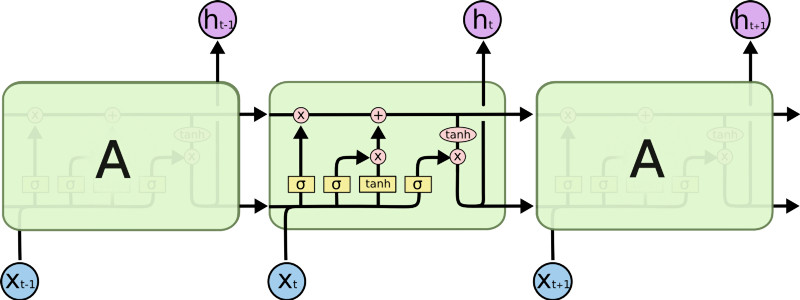
* In recovery-based models, a chatbot uses a heuristic to select an answer from a library of predefined answers. The chatbot uses the message and chat box to choose the best response from a predefined bot-message list. The table may include a current location in the dialog tree, all previous messages in the conversation, variables previously-stored (eg, username). The heuristics for selecting a response can be constructed in many different ways, from rule-based conditional logic to mechanical classifiers.
* Boot generations can generate the answers and not always the answers with one of the solutions from a set of answers. This makes them smarter as they take the word for word from a question and generate the responses [1].

**Recurrent Neural Networks**

In this exercise we use RNN. Repeated neural networks or simple RNNs are a special kind of neural network capable of dealing with sequential data, such as video (frame sequence) and, more often, text sequences or any sequence of symbols. To put it, an RNN, as opposed to an MLP or CNN, has an internal state. As the RNN devours a sequence (word for word), essential information about the sentence is retained in this memory (internal state), which is periodically updated at any time. An RNN will record the essence of a 7-word sentence seven times. This is just a "story" about the RNN, which aims to provide a high-level understanding of the RNN. To understand the exact mechanism of RNN, read Denny Britz's series of articles on RNN - 1, 2, 3, 4, and then proceed to Karpathy's The Unreasonable Effectiveness of RNNs [2].

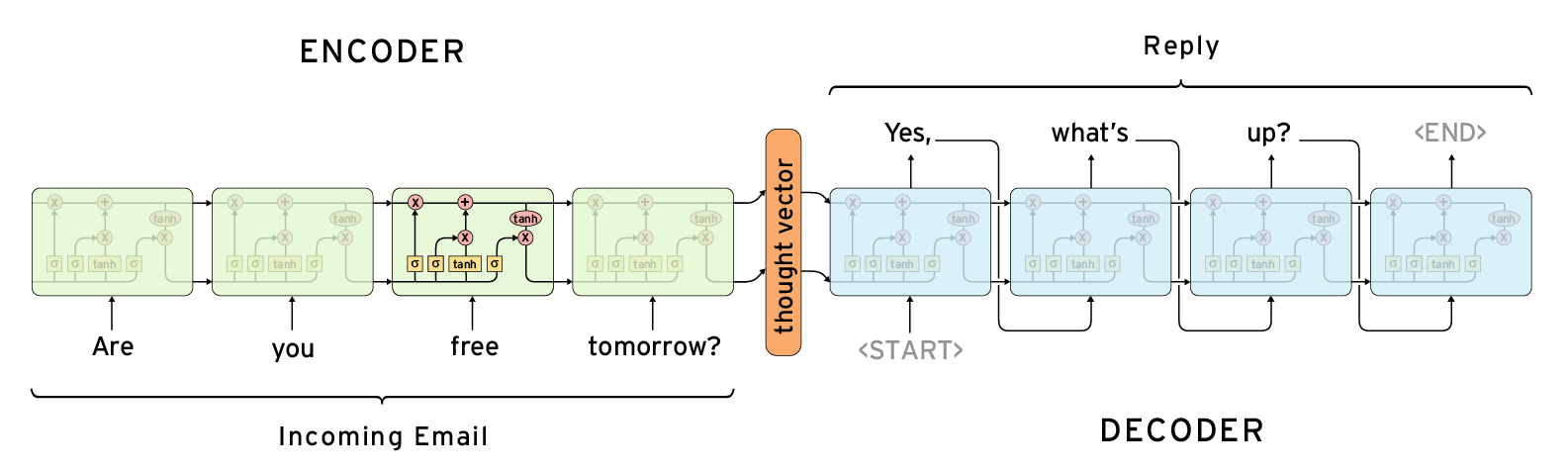


The naive version of RNN is usually called Vanilla RNN, which is sad enough to remember the long sequences. There are more sophisticated versions of RNNs, such as RNNs LSTM (Long Short Term Memory) and GRU (Gated Recurrent Units). The only difference between the Vanilla RNN and LSTM / GRU networks is the memory module architecture. An LSTM cell consists of multiple gateways to remember useful information, forgetting unnecessary information, and carefully exposing information at every step [2].



**Sequence to Sequence Learning**

The Sequence Sequence Model (seq2seq) consists of two RNNs - one encoder and one decoder. The encoder reads the input sequence word for word and transmits a frame (a function of the encoder's final hidden state), which would ideally capture the essence (semantic summary) of the input sequence. Based on this frame, the decoder produces the output sequence, one word at a time, while looking at the structure and the previous word during each timing. The goal of the exercise is to maximize together with the probability of recording the output sequence dependent on the input sequence [2].



**The Model**

Here is a working model of a Cornell bot based on the seq2seq model.

| **Query** | **Reply** |
| --- | --- |
| you hate me dont you | i dont really think you warrant that strong an emotion |
| then say youll spend dollar night at the track with me | and why would i do that |
| come on the ponies the flat beer you with money in your eyes me with my hand on your ass | you covered in my vomit |
| are you following me | i was in the laundromat i saw your car thought id say hi |

**The Code**

Below we are able to see the step by step the bot. Any project in Machine Learning follows this pattern [2].

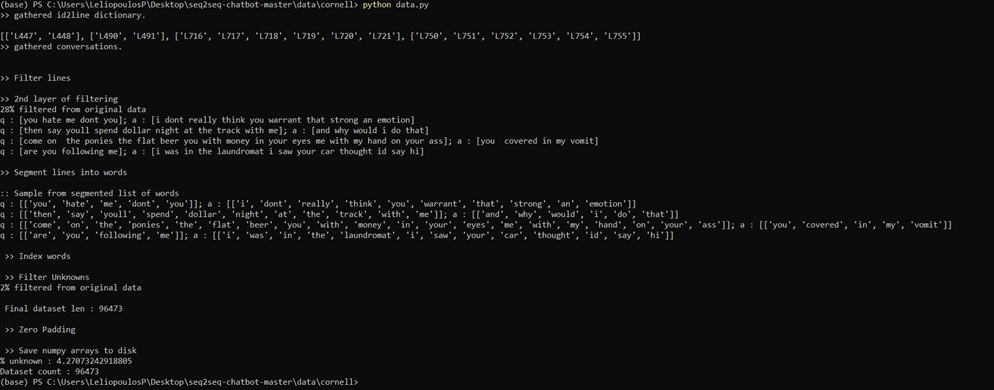
* Study and analyze the data.
* Preprocess the data to make it compatible with the model.
* Split it into a train.
* Valid, and test sets.
* Create a model.
* Feed the training data.
* Let it overfit.
* Reduce the depth/width of the model and train again (which I usually skip).
* Evaluate the model periodically to look for overfitting/underfitting.

**Data Preprocessing**

First we make data preprocessing from the Cornell dataset. The dataset including metadata from movie characters and movie titles. The data including conversations from movies, the movie lines and raw scripts urls. This corpus contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts [3]:

* 220,579 conversational exchanges between 10,292 pairs of movie characters
* involves 9,035 characters from 617 movies
* in total 304,713 utterances

We start the preprocessing running the prepare\_data.py script and as we can see in the below figure starting the preprocessing of the raw data files in 3 new files. The file Idx\_a.npy is about the questions, the idx\_q.npy is about the answers and the file metadata.pkl including the metadata of the dataset.



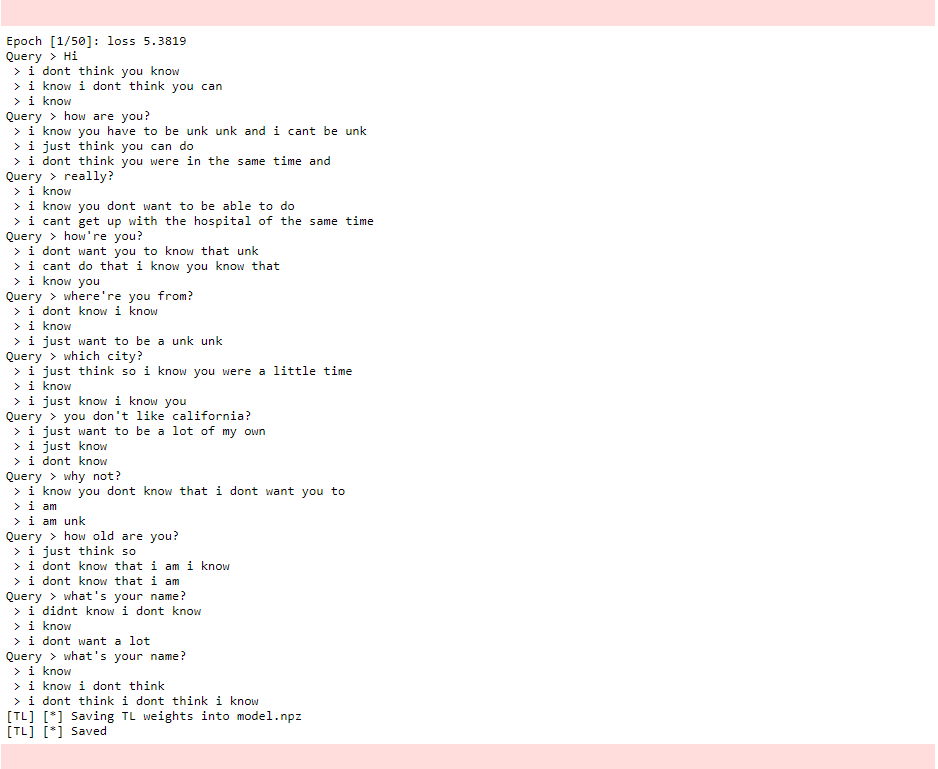
**The requirements of the experiment**

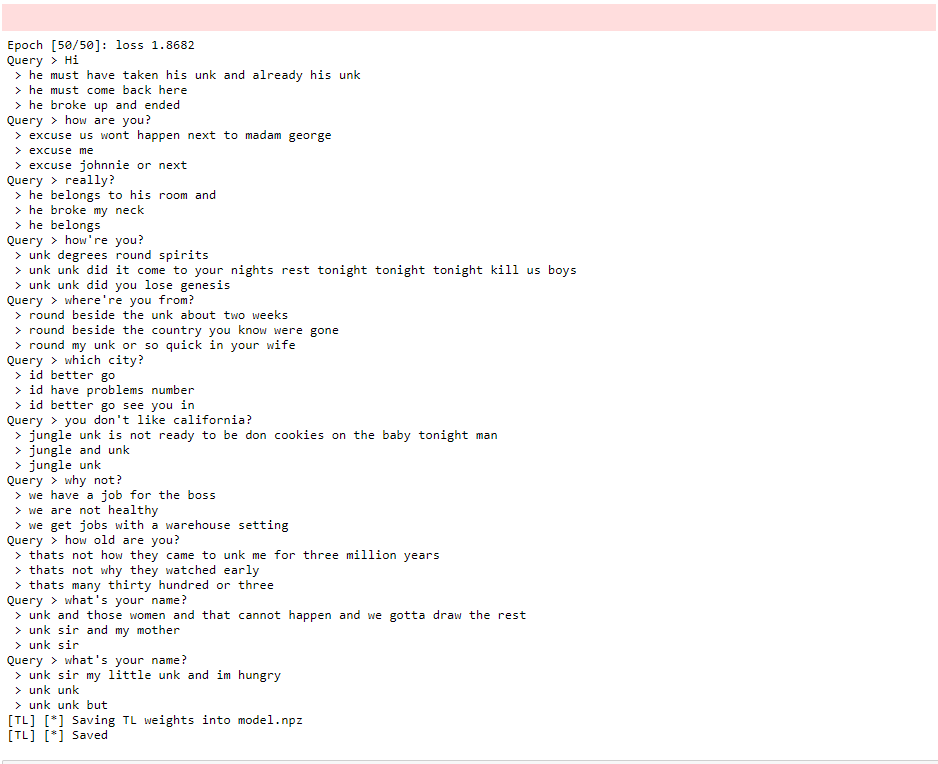
For our results we meet the below requirements.

* For our experiments we use the software, Python 3.6, TensorFlow 2.1 and TensorLayer 2.0.
* The hardware we use is a PC with 16core CPU and 32GB Ram.
* The whole processing for parameters as input: n\_layer=3, n\_units=256, num\_epochs=50, learning\_rate=0.001.
* The processing time was approximately 15 hours.

**The Results**

As we can see from the above figures we have a lot of improvement in loss from 5.3819 for the epoch 1 to 1.8682 for the epoch 50. Also as we can mention from the given answers there is a lot of improvement from the bot. Also from the given answers we choose to take the three best answers.





**Conclusion**

So as we can see from the below figure we from the epochs 1 to 5, we have a significantly lower loss. After that, we can see as we increase the number of epochs, we can receive lower losses. But after several times, the loss is getting to be stable.

**References**

[1] P. Pandey, “Building a Simple Chatbot from Scratch in Python (using NLTK),” *Medium*, 09-Aug-2019. [Online]. Available: https://medium.com/analytics-vidhya/building-a-simple-chatbot-in-python-using-nltk-7c8c8215ac6e. [Accessed: 30-Jan-2020].

[2] S. Ramamoorthy, “Practical seq2seq.” [Online]. Available: http://suriyadeepan.github.io/2016-12-31-practical-seq2seq//. [Accessed: 28-Jan-2020].

[3] “suriyadeepan/datasets,” *GitHub*. [Online]. Available: https://github.com/suriyadeepan/datasets. [Accessed: 28-Jan-2020].