***Generative Chatbot Description***

* A generative Chatbot based on Sequential Neural Network and Deep Learning which can be trained on any desired dataset for specific purposes. Instead of ordinary Chatbots which are based on hard-coded responses, it can understand context and respond accordingly.

***Dataset Description***

* ***A) Brief description:*** (Cornell Movie Dialogue Corpus)

This corpus is a 40MB contains a 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

***Code Workflow “Architecture”:***

***Generative models (harder and smarter)*** don’t rely on pre-defined responses. They generate new responses from scratch. Generative models are typically based on Machine Translation techniques, but instead of translating from one language to another, we “translate” from an input to an output (response). They can refer to entities in the input and give the impression that you’re talking to a human. However, these models are hard to train, are quite likely to make grammatical mistakes (especially on longer sentences), and typically require huge amounts of training data.

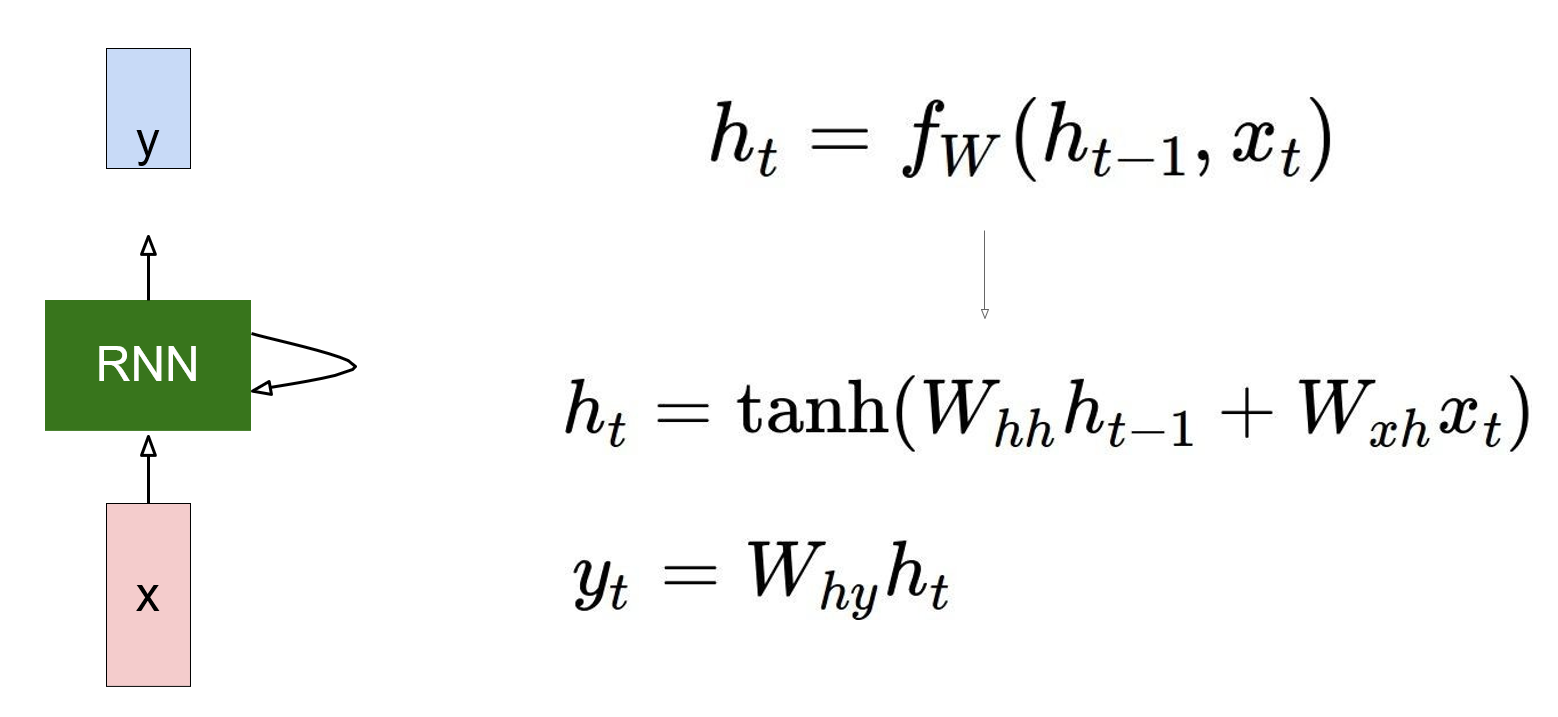
***" Generative model* "** represents one of the several possibilities that are possible by training a sequential neural network. Going into the specifications, it is constructed of three layers of the sequential neural network each containing 128 neurons. The encoder and decoder have a vocabulary size of 20,000 each. The corpus used for training is a collection of dialogues from 617 movies containing about 220,000 conversations (Cornell Movie Dialogue Corpus).

***The Recurrent Neural Network (RNN)*** is a natural generalization of feedforward neural networks to sequences. Given a sequence of inputs, a standard RNN computes a sequence of outputs, by iterating a certain equation, The RNN can easily map sequences to sequences whenever the alignment between the inputs the outputs is known ahead of time. However, it is not clear how to apply an RNN to problems whose input and the output sequences have different lengths with complicated and non-monotonic relationships. The simplest strategy for general sequence learning is to map the input sequence to a fixed-sized vector using one RNN, and then to map the vector to the target sequence with another RNN. While it could work in principle since the RNN is provided with all the relevant information, it would be difficult to train the RNNs due to the resulting long term dependencies However, the Long Short-Term Memory (LSTM) is known to learn problems with long range temporal dependencies, so an LSTM may succeed in this setting. The goal of the LSTM is to estimate the conditional probability p (y1, . . ., yT′ |x1, . . ., xT) where (x1, . . ., xT) is an input sequence and y1, . . ., yT′ is its corresponding output sequence whose length T ′ may differ from T. Note that we require that each sentence ends with a special end-of-sentence symbol “”, which enables the model to define a distribution over sequences of all possible lengths. The overall scheme is outlined in the figure. Our actual models differ from the above description in three important ways. First, we used two different LSTMs: one for the input sequence and another for the output sequence, because doing so increases the number model parameters at negligible computational cost and makes it natural to train the LSTM on multiple language pairs simultaneously Second, we found that deep LSTMs significantly outperformed shallow LSTMs, so we chose an LSTM with Three layers. Third, we found it extremely valuable to reverse the order of the words of the input sentence. So, for example, instead of mapping the sentence a, b, c to the sentence α, β, γ, the LSTM is asked to map c, b, a to α, β, γ, where α, β, γ is the translation of a, b, c. This way, a is near α, b is fairly close to β, and so on, a fact that makes it easy for SGD to “establish communication” between the input and the output. We found this simple data transformation to greatly improve the performance of the LSTM.

We found that the LSTM models are fairly easy to train, we used deep LSTMs with 3 layers, with 128 cells at each layer, the encoder and decoder have a vocabulary size of 20,000 each.

A screenshot of a cell phone

Description automatically generated



A screenshot of a cell phone

Description automatically generated

A picture containing text

Description automatically generated

***Tools and Libraries used***

* TensorFlow
* Tkinter
* Anaconda
* Spyder

***Instructions to run***

There are two steps involved in running the chatbot:

To train the bot

* a. Open the file "seq2seq.ini"
* b. Set 'mode = train'
* c. Run the file execute.py using the code "python execute.py"

To test the bot after training

* a. Open the file "seq2seq.ini"
* b. Set 'mode = test'
* c. Run the file execute.py using the code "python execute.py"

***Retrieval Chatbot Description***

**Retrieval-based models (easier)** use a repository of predefined responses and some kind of heuristic to pick an appropriate response based on the input and context. The heuristic could be as simple as a rule-based expression match, or as complex as an ensemble of Machine Learning classifiers. These systems don’t generate any new text, they just pick a response from a fixed set. retrieval-based methods don’t make grammatical mistakes. However, they may be unable to handle unseen cases for which no appropriate predefined response exists. For the same reasons, these models can’t refer back to contextual entity information like names mentioned earlier in the conversation

A close up of a map

Description automatically generated

***Dataset Description “The Movies Dataset***

These files contain metadata for all 45,000 movies listed in the Full Movie Lens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official Group Lens website.

***Text Pre- Processing***

The main issue with text data is that it is all in text format (strings). However, the Machine learning algorithms need some sort of numerical feature vector in order to perform the task. So, before we start with any NLP project, we need to pre-process it to make it ideal for working. Basic text pre-processing includes

-Converting the entire text into uppercase or lowercase, so that the algorithm does not treat the same words in different cases as different

-Tokenization: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e. words that we want. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings

-Removing Noise i.e. everything that isn’t in a standard number or letter.

* -Removing Stop words**.**Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called *stop words*

**-**Stemming: Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form — generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stigmatization”, the result would be a single word “stem”.

**-**Lemmatization: A slight variant of stemming is lemmatization. The major difference between these is, that, stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of Lemmatization are that “run” is a base form for words like “running” or “ran” or that the word “better” and “good” are in the same lemma, so they are considered the same.

***TF-IDF Approach***

A problem with the Bag of Words approach is that highly frequent words start to dominate in the document (e.g. larger score) but may not contain as much “informational content”. Also, it will give more weight to longer documents than shorter documents

One approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach to scoring is called Term Frequency-Inverse Document Frequency, or TF-IDF for short, where: Term Frequency: is a scoring of the frequency of the word in the current document*.*

***Inverse Document Frequency***: is a scoring of how rare the word is across documents

***Tf-idf weight*** is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus

***Cosine Similarity*** TF-IDF is a transformation applied to texts to get two real-valued vectors in vector space. We can then obtain the Cosine similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors. Cosine similarity is a measure of similarity between two non-zero vectors. Using this formula, we can find out the similarity between any two documents d1 and d2.