***Project 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.

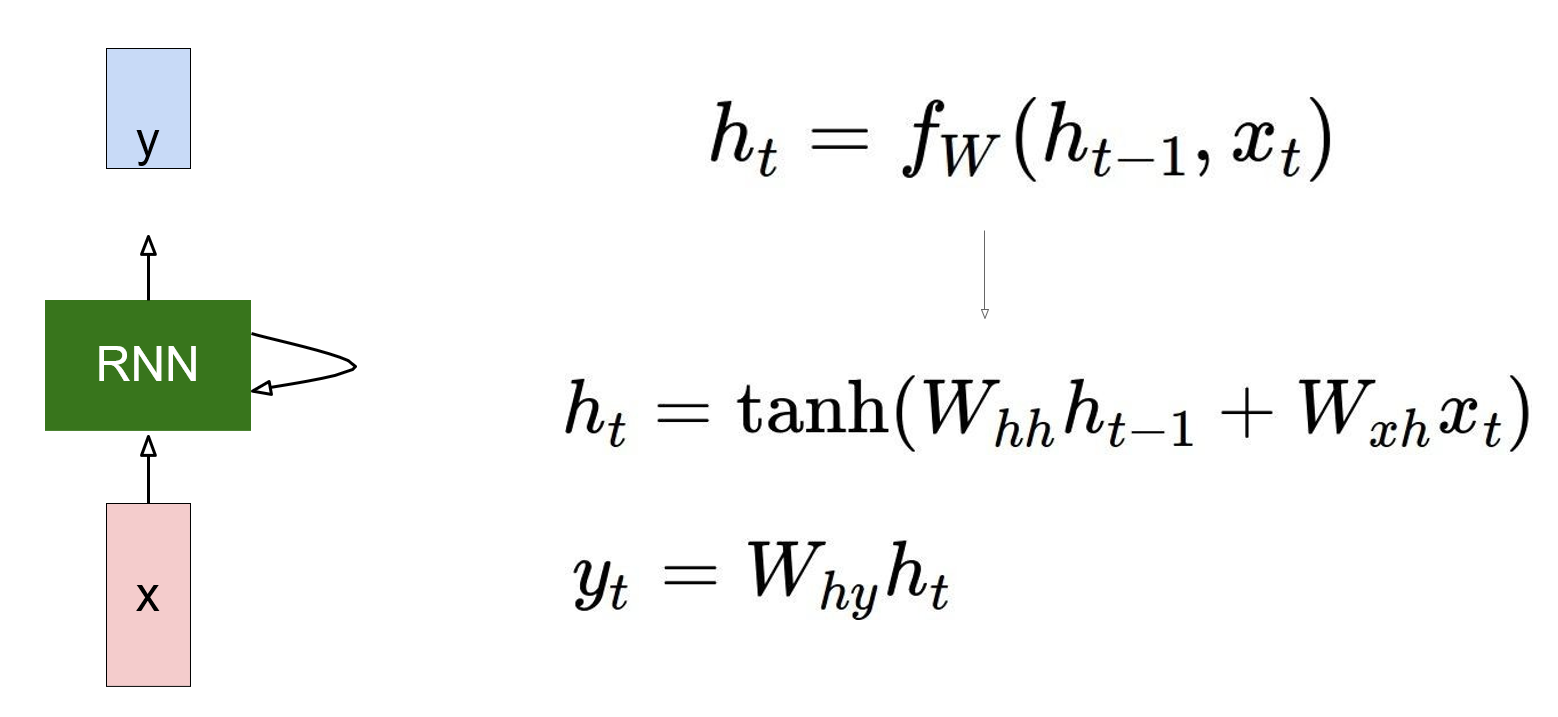
***" Generative model* "** represents one of the several possibilities that are possible by training a sequential neural network. Going into the specifications, it is constructed of three layers of the sequential neural network each containing 128 neurons. The encoder and decoder have a vocabulary size 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