Design Details

The first step in any deep learning project is acquiring a dataset. Once the dataset was downloaded it had to go through the pre-processing steps. The pre-processing includes removing abbreviations, unnecessary symbols, so all in all, it involves all the steps to make the data more readable and less noisy.

Between the pre-processing and training is the very important step of making the data possible to be read by the machine. Deep learning models do not understand plain English, they can only understand numbers. Therefore the questions have to be converted primarily to sequence of integers, where one word corresponds to only one number and vice-versa.

This project uses deep learning methods to try to implement a chatbot. More specifically it relies on Recurrent Neural Networks for the implementation, from now on referred to as RNN. More specifically this project implements a sequence-to-sequence (abbr: seq2seq) to manage the conversion from input to output of the data.

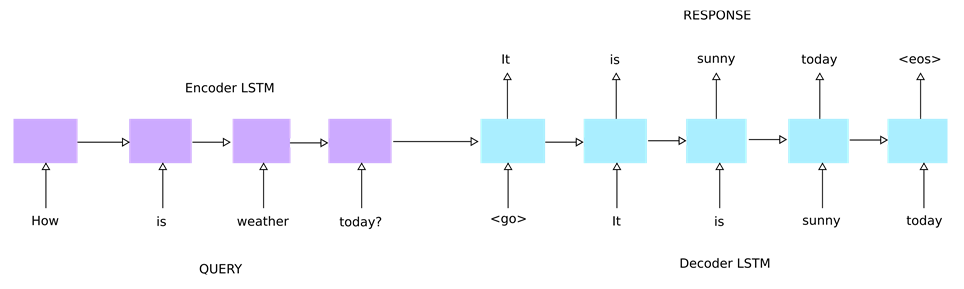


Fig. General Structure of a Seq2Seq model

A seq2seq model takes in an sequence of integers and converts it to another sequence. Since a chatbot takes a question and produces an answer based on it needs to implement an automatic encoder-decoder translator. The encoder takes a pre-processed question as input, this input is converted to a vector state and is read by the decoder which finds the appropriate response to every word that the encoder has fed through the machine. Since RNN’s suffer from the vanishing gradient problem, a modern version of RNN is used: Long Short Term Memory RNN (abbr: LSTM). The encoder and the decoder are both LSTM layers. In a stacked LSTM model, the output of the first layer will be passed to the second one. This makes the layers aware of the layers before and after them. The output of the final layer is passed into a dense layer, which server as regulator to change the shape of the output. However this dense layer is applied to every time step of the unrolling process through time distribution. An important factor to be discussed is dropout. In order to maintain the model from over-fitting and just memorizing the answers a randomizing layer is applied. Dropout intends to alternate how the inputs are propagated through the network. As final steps optimizers are added to the model and the model is compiled. Through training the achieved accuracy and the loss are seen as outputs at the end of each epoch. Keeping track of this data is important to determine how the chatbot is operating.

Due to the data being large in size and taking up too much RAM a mechanism needs to implemented to reduce the amount of data that is processed per unit of time. This mechanism is in the form of a generator. Generators are python functions that take a large amount of data in split into subsection. Each of these section is then fed to the model for training. The weights of the model are updated after every epoch. This way the model can keep learning while taking up more processing power but less RAM.

Once the data has been trained and the chatbot has produced the weights an inference model has to be built. The purpose of the inference model is to go one time step into the future to produce the final output. The inference layer uses the states from the main model and has the same shape as it. By going one time step ahead this layer removes the ‘start of sentence’ token that was introduced earlier to help with the training process.

The final step will be to produce the answer to the question asked to the chatbot. Every final token produced by the decoder will be added to the answer step by step. This process will happen until the decoder meets an ‘end of sentence’ token or it reaches the word limit set for the answer. In which case the decoder stops and the final output is produced.

Implementation details

This model was developed entirely using Tensorflow and Keras classes. It was mostly run using google colabs due to stronger processing power, but some of the tests were completed in our personal computers.

Dataset

This is the first step of the chatbot production. The dataset we used was a dataset retrieve from Yahoo which contained answer and questions on various subjects. These conversations were produced by the public n discussions with each other on Yahoo forums. The dataset contained more than 1 million questions on different topics such as: medicine, everyday life, history, geography etc. However since our project was banking and financing focused we only took the questions on economy. Our dataset to work with had a size of about 140000 questions and their respective answers.

Pre-processing

Once the dataset is saved the pre-processing part begins. This part of the project achieved the followings.

* Answers and questions were taken from the dataset and passed into 2 lists while maintaining their order intact.
* Since our maximum length of question and the maximum length of answer to be provided by the model was 20, only sentences shorter than this were take, and only the first 20 words from other sentences replace the rest.
* Considering how the conversations were between general public members the data was very noisy. Hence, a lot had to be removed including here: abbreviations, informalities, links, extra spaces and special characters.

Once the data is cleaned from all the noises, the second step is to create a vocabulary. Through the use of dictionaries every word in the questions and answers were added to a vocabulary and a special index was assigned to it. This index is basically just count of the word. So in example if word ‘how’ appears first on the dataset it is assigned index 0 and all other same words after that are not counted. The second word to appear is assigned index 2 and so on.

After every word is mapped to an integer, a new dictionary is created to map every integer to its corresponding word. This is done because the output of the model will be a list of integers, and for them to make sense to humans this list will be converted to words using this dictionary.

Now that every word corresponds to an integer the sentences have to be converted to the sequences that will be used for the model. For this process we loop through every question or answer in the questions/answers list, and every word in these lists is mapped to its corresponding integer value. At the end two lists of same lengths as answers/questions lists are created but instead of the English words they have the equivalent integers of these words.

The last step is to save the integer version of the question and answers, the vocabulary and the reversed vocabulary dictionaries. These files will be used later during the training and that concludes pre-processing.

Pre-model definition

Before the model is built and the data is trained there are some intermediate steps to be followed. The first step is to increment the index of the answers, questions, and vocabularies. Later before the training the data will be padded so that every sentence has the same length. The padding process is done by adding zeros to the end of every sentence. However at this point we have some words that are mapped to zeros in our lists and dictionaries, and by incrementing the indexes of each word by 1 this problem is solved.

In this protect we chose GloVe embeddings in order to create the vectors for each word. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. GloVe by Stanford contains a dictionary of words and is used to create a vector representation of each word. Every word in our dictionary will be converted to a vector using this algorithm. We chose the GloVe embeddings with a dimension of 50.

After the GloVe embedding list is uploaded every word is converted to an embedding matrix. An embedding matrix is a linear mapping from the original space (one-of-k) to a real-valued space where entities can have meaningful relationships.

Building the Model

A model begins with an embedding layer. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. This layer is offered by Keras and it includes the following arguments: input\_dim: This is the size of the vocabulary in the text data. For example, if the data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words. In our case this is the vocabulary size. output\_dim: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. Which is the expected output from the GloVe vectors=50. The third parameter is trainable=True to allow the weights to be updated during the training. Before the data can be passed into the model it has to be in the right shape and right type. This is achieved through the input layers for both the encoder and the decoder. The input layer is the entrance point to the model. The data going through this layer is then passed into the embedding layer defined earlier and is then fed to the encoder.

We decided on an LSTM encoder with size 300, since any size to small would not learn enough and layers with more neurons than 300 would just memorize everything. Return state and return sequences are two important factors for all LSTM layers. Each unit or cell within the layer has an internal cell state, often abbreviated as “c“, and outputs a hidden state, often abbreviated as “h“. Each LSTM cell will return these two states. Since our model is stacked then we need to initialize the following layer with the states of the lower layers. Return sequence just returns a sequence from each cell that can be used in the following cell. Three encoders mean three layers and for each encoder there is a decoder. The decoder takes as input the states from its corresponding encoder and the sequence from the decoder below it. The first decoder takes as input the initial sequence that was embedded in the same way that the encoder input was. However this initial sequence is the padded version of the answers unlike the input for the encoder which was the padded questions.

After the data has been processed through the three layers it goes through a dense layer that will transform the vectors. An important element of the dense layer is the activation. We have chosen for activation ‘softmax’ since it assigns probabilities to all the entries and it makes it the best solution for such models. Time distribution is a wrapper that will apply the dense layer to every time step after the unrolling of the LSTM decoder. This layer adds the time dimension to the dense layer that we have already applied.

The final step is to apply the dropout. The dropout is the randomness that the network will have as it propagates through the layers. Suggested values for dropout range between 0.2 and 0.5. When testing with datasets we noticed that a dropout of 0.3-0.4 (depending on the dataset size) would give the best results.

The mode is compiled with optimizer adam and loss as ‘categorical\_crossentropy’. The reason for adam is one of the most used optimizers in deep learning applications. The main idea behind adam is that loss will change with respect to results and time passed.

Once the data has gone through all these processes it is the model is instantiated. The model takes as input 2 arguments: the input data which in our case are the input layers of input for the encoder and input for the decoder, and the output which is the output produced by the model itself.

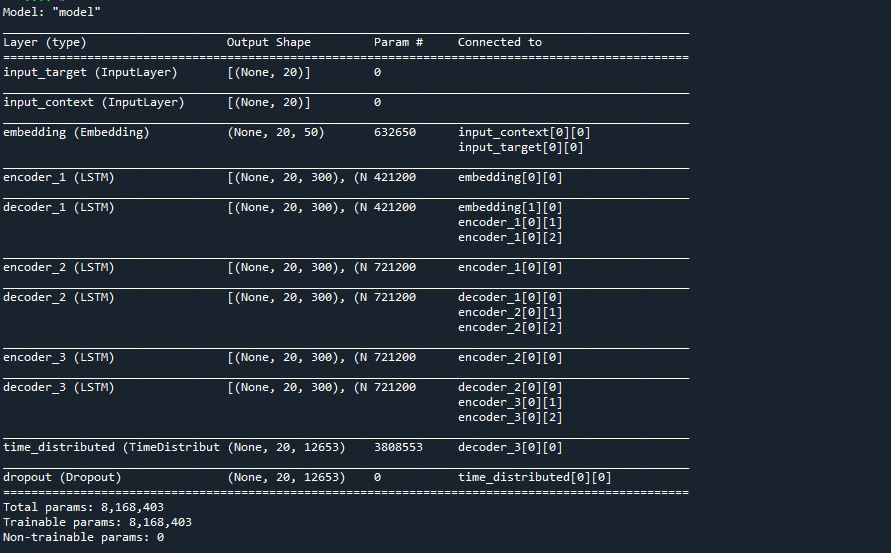


Fig. Full representation of what the model includes

Training

Among the issues that such implementations have is the size issue. At many points during the process the generated data would be too much for the memory to handle. To solve this issue we implemented generators. The generator takes as input the sequence of questions and answers (as numbers) and a batch size which will be the batch size on which the model will train. This generator completes the final steps on the data before it is fed into the model. This algorithm selects an amount of lines from the dataset equal in size to the batch size declared earlier. Answers and questions are padded during this process, these answers and questions will later go to the model through the embedded input layers as discussed earlier. The answers will also be moved one time step later and will be converted to categorical data and the outcomes of this will go into the output parameter of the model.

The model is finally trained with the output from the generator, the batch size as 64/128, since these two batch sizes gave similar results with 128 performing slightly better on larger datasets. An early stopping parameter with patience 5 is added so that in case that the model is not improving for the last 15 epochs the training will assume that this is its limit and exit despite how man epochs are left. Validation data is also added to the training parameter to keep track of how the training is working. Validation data is generated in a similar way to the training data. When validation data is made available the training produces extra outputs which are val\_loss and val\_accuracy. val\_loss is the value of cost function for your cross-validation data and loss is the value of cost function for your training data, same logic applies to val\_accuracy and accuracy.



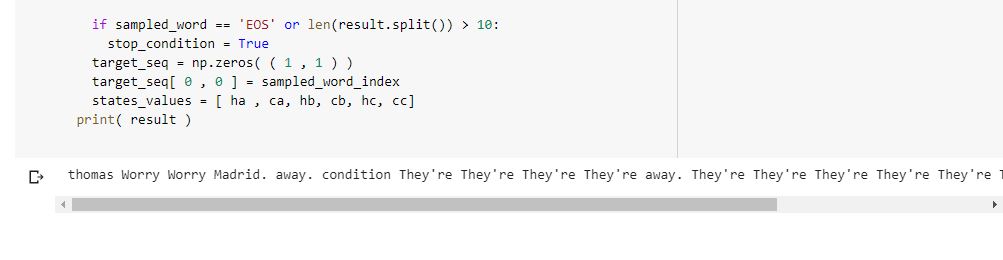
Fig, snapshot of training process

Prediction and results

After training has finishes the final weights of the network have been determined the next step is prediction. The first thing we did was create an inference model which will be used to for taking smaller batches of real-world data and quickly coming back with the same correct answer (really a prediction that something is correct). This prediction is based on the states, weights and layers that the model was trained on. The inference model needs to have same sizes and layers of the main model. This model is comprised from two models, one of which includes the encoders from the main model. Finally a question is passed to the inference model. This question needs to be clean, that means it should contain no capital letters or symbols, however pre-processing can be applied to it. The question is converted to a sequence and is passed to the inference model which will make its prediction based on what it can see from the weights and will produce an output which is the final answer.

2.8 Concluding remarks

The main issue with deep learning in general and LSTM specifically is that the topics are relatively new and hence there isn’t much certainty around them. While working on this project we ran more into questions than answers. Usually the answers were to try with values and run several tests and then can the most optimum layout for a model be found. For this we ran multiple tests and settled on the model that gave the best results. What we can take from this work is that there are a lot of factors that can affect results in an area such as deep learning. Tuning the parameters and settling for the correct amount of layers and factors was our main focus during this project. Another hindrance was the dataset issue. In our efforts to build a chatbot concerned about economics, banking and finance we faced the issue of finding a proper dataset. The dataset we settled on had a lot of noise and hence did not help us produce the results we were aiming for. The process of building this project ran into several problems, the biggest of which was how to deal with processing and training large amounts of data. However given the resources we had, it is our belief that we developed a decent model. Following all the rules and techniques required to develop such a model with better equipment our model would deliver better performance.



Fig, Results produced after testing chatbot

3.0

3.1

3.2

The required files include: source code, dataset, GloVe embeddings file. After the paths extracting the dataset, and the GloVe embeddings file have been edited and the files have been imported properly. The code can be run all at the same time. It is however suggested to run the code according to the cells in the notebook.

Notebook:

<https://colab.research.google.com/drive/1xmpQlSuZyTCOkAOW4RMMv5tDrmoeql36#scrollTo=0B0qeCQ2WjMf>

GloVe file:

<https://drive.google.com/file/d/1a4KY1HLwrIigntTDRNCoUho92-BuuMPN/view?usp=sharing>

Dataset:

<https://drive.google.com/file/d/1BXFGVgeLXZ4GWha4BfOcRgecCgEon9K2/view?usp=sharing>