Chatbot

Chenwei Cao

November 25, 2020

* Introduction

A chatbot is a software that provides a real conversational experience to the user, A well-optimized chatbot can converse as well as humans can: asking and answering a wide range of questions, displaying knowledge, and being empathetic, personable, engaging, serious, or fun.

Here for our project, I use generation strategy to create an open domain chatbot which can generate a response as the name implies. It can be a great addition to a company’s business because it can help to boost conversation rates.

* Summary

I use Tensorflow and Keras to build LSTM Seq2Seq model, recurrent neural network to identify the context the user is asking and then provide it with the relevant answer. Since the dataset is relevant small and the limitation of computation, for preprocessing step, I separately use two main vector representation in Natural Language Processing(NLP) which are one-hot encoding and embedding to convert human conversation to vectors. For optimization performance process, I change the optimizer hyperparameter.

* Tools and Datasets Overview

I used Python and Keras API, NLP techniques and google colab to build our deep learning models. The datasets I used are collected from Kaggle which consists of 2363 entries records conversations between a human and other human acting as a companion bot.

* Data Pre-processing

I used Natural language Processing(NLP) to do text corpus preprocessing. I remove punctuation and space with the help of regular expression. After that, I grouped human response with robot response as pairs of sentences since we need to use human response as input sequence and robot response as target sequence :

Text

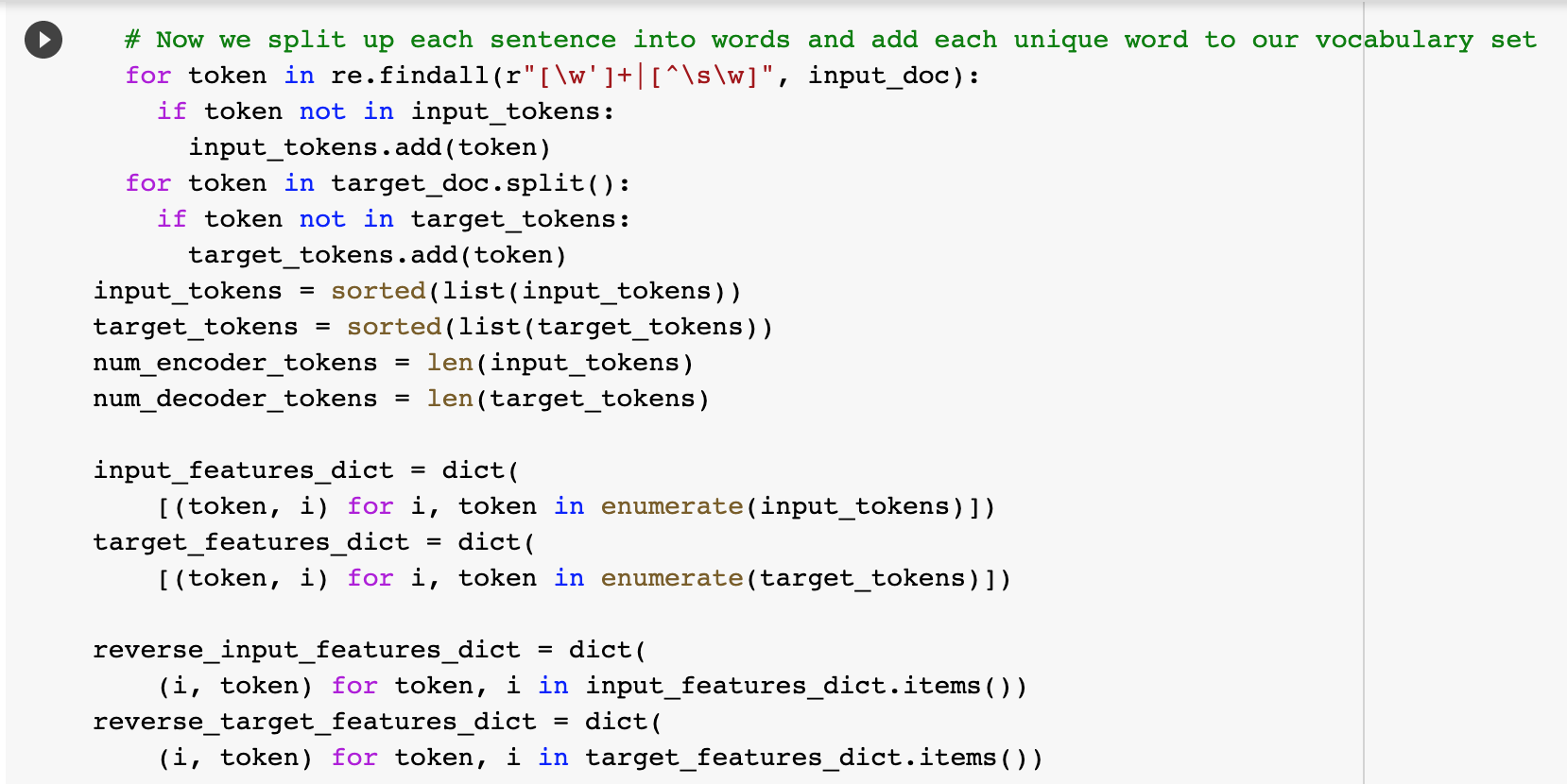
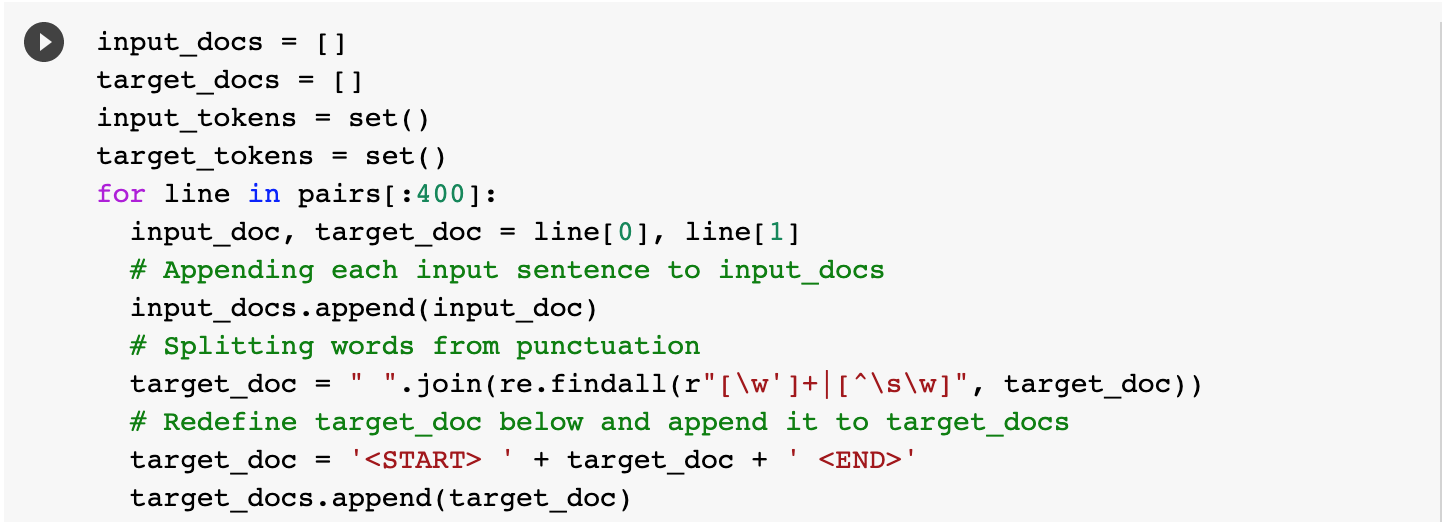
Description automatically generated

Here is output for first 10 conversation pairs

Text, letter

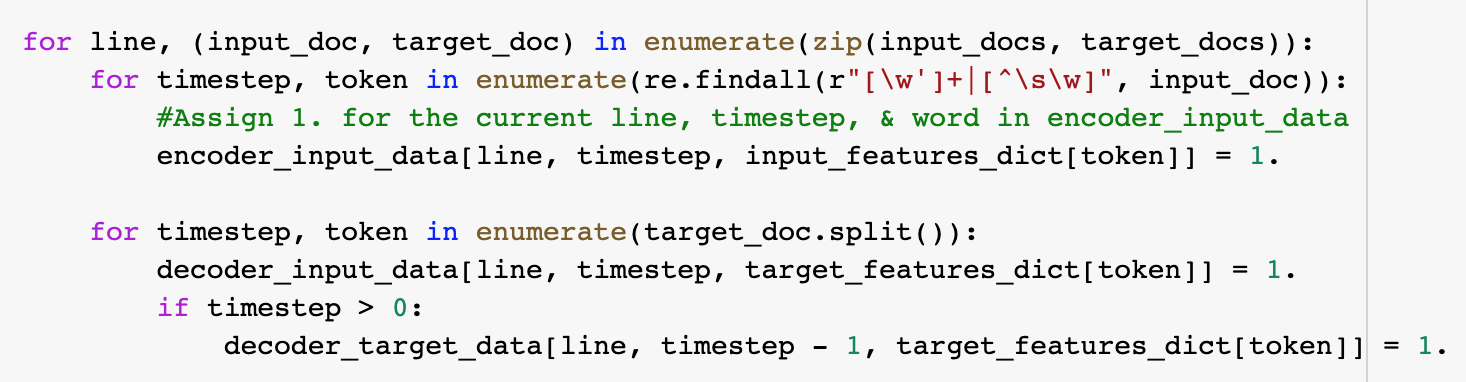
Description automatically generated

For target sequences, I have to add <START> at the beginning of the target sentence and <END> at the end of the target sentence. Then we need to do tokenization, I created separate lists for unique input tokens and unique target tokens. After that, I created input features dictionary and target features dictionary to store tokens as key-value pairs.

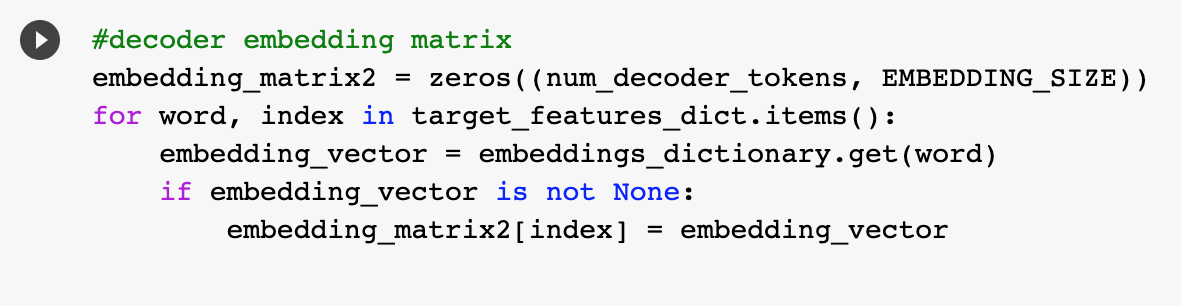
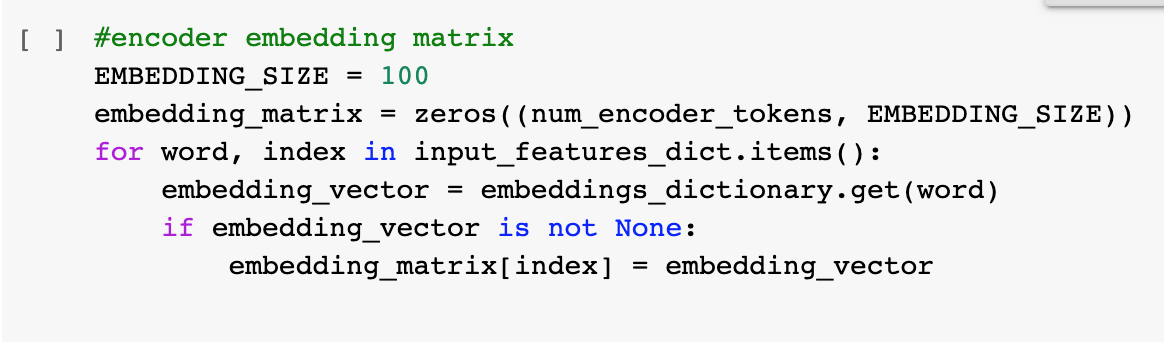
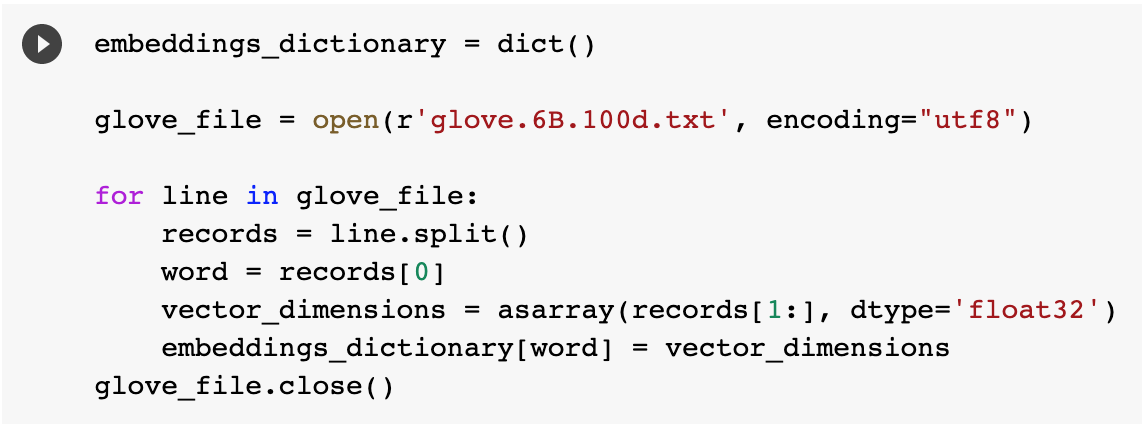


For our model to understand human and robot language, I need to convert our words into their corresponding numeric vector representations. There are two main methods to do this.For one-hot encoding, every word has its own value in a vector. It is easy to implement and can work really fast, but in this process, it loses the inner meaning of the word in a sentence. On the other hand, word embedding takes context into account and gives word with similar meaning or influence in a sentence similar value for a specific feature. But sometimes, word embedding does not work exceptionally well and it is also computationally more expensive than One Hot vector. So I decide to test both methods and hope I can get better results after applying either one.

Firstly, considering the limitation of computation, I use one-hot vectors for encoder input, decoder input and decoder output.



However, considering one-hot encoding will ignore the context of the sentence, so next, I use word embedding method for encoder input and decoder input to see if it can help improving model performance. Here, I use GloVe word embedding.

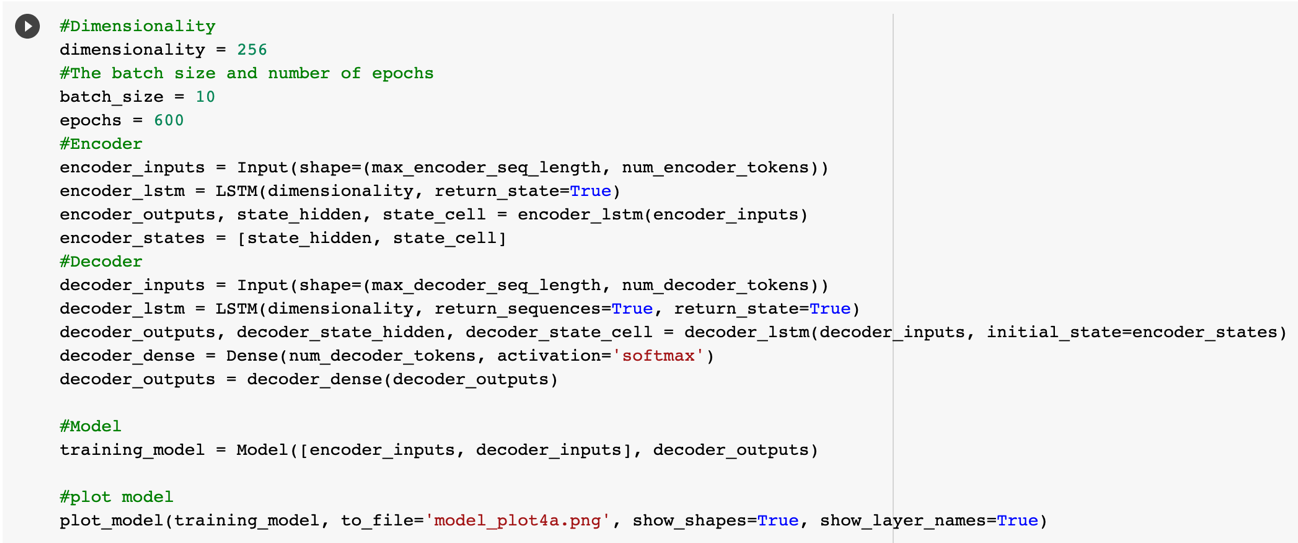


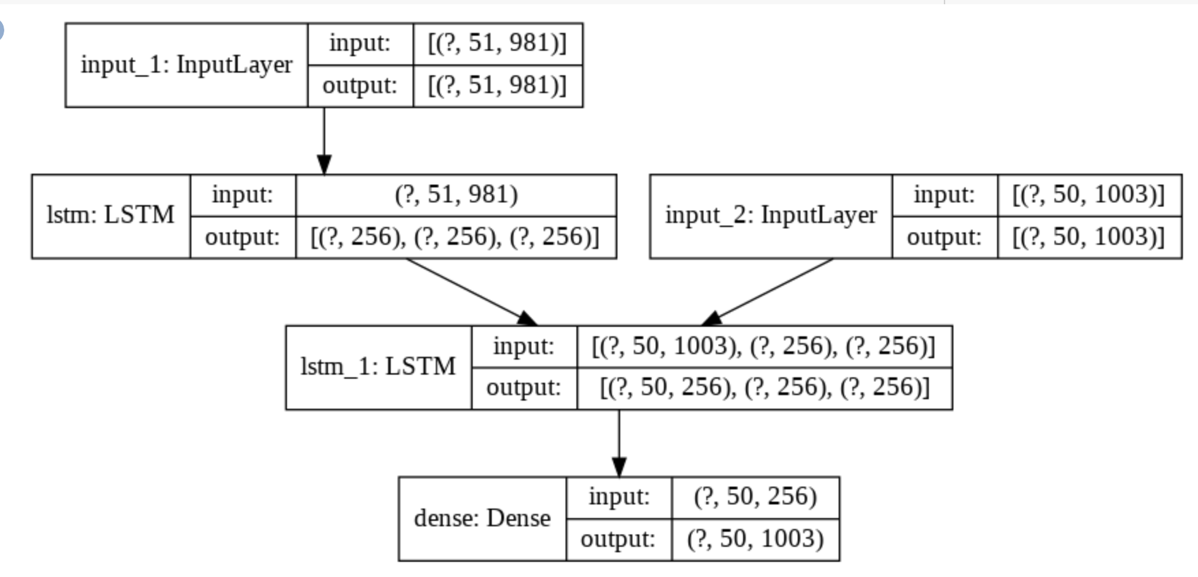
* Model Selection

I selected LSTM seq2seq model as our training model. We use Keras Functional API as my model structure.

The seq2seq model also called encoder-decoder model which use long-short term memory for text generation from the training sets. It predicts a word given in the user input and then each of the next words is predicted using the probability of likelihood of that word to occur.

This is the #1 model with one-hot method:

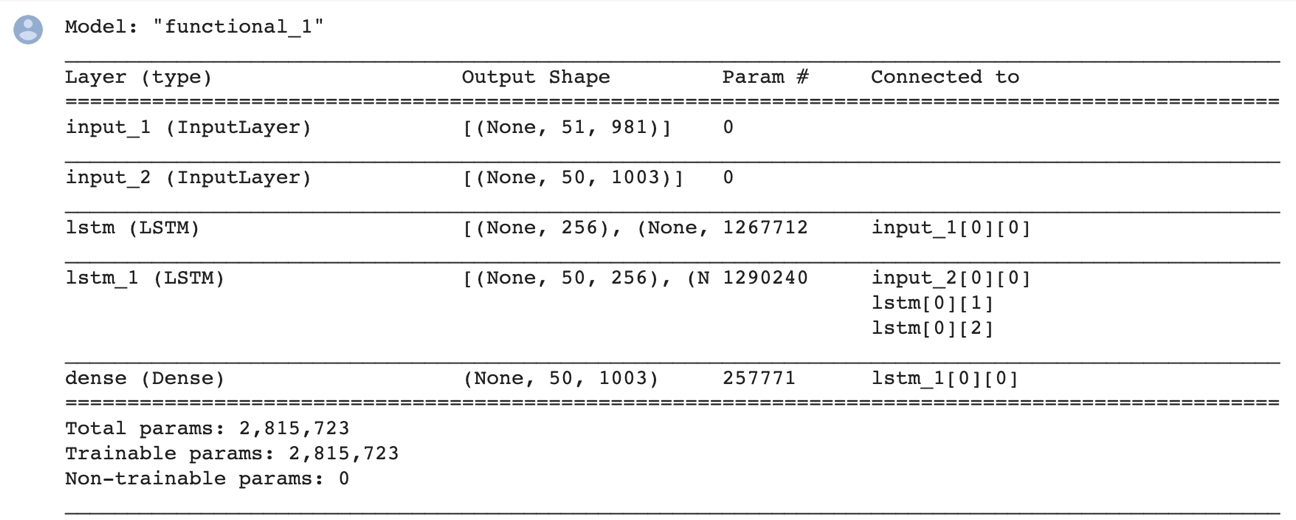




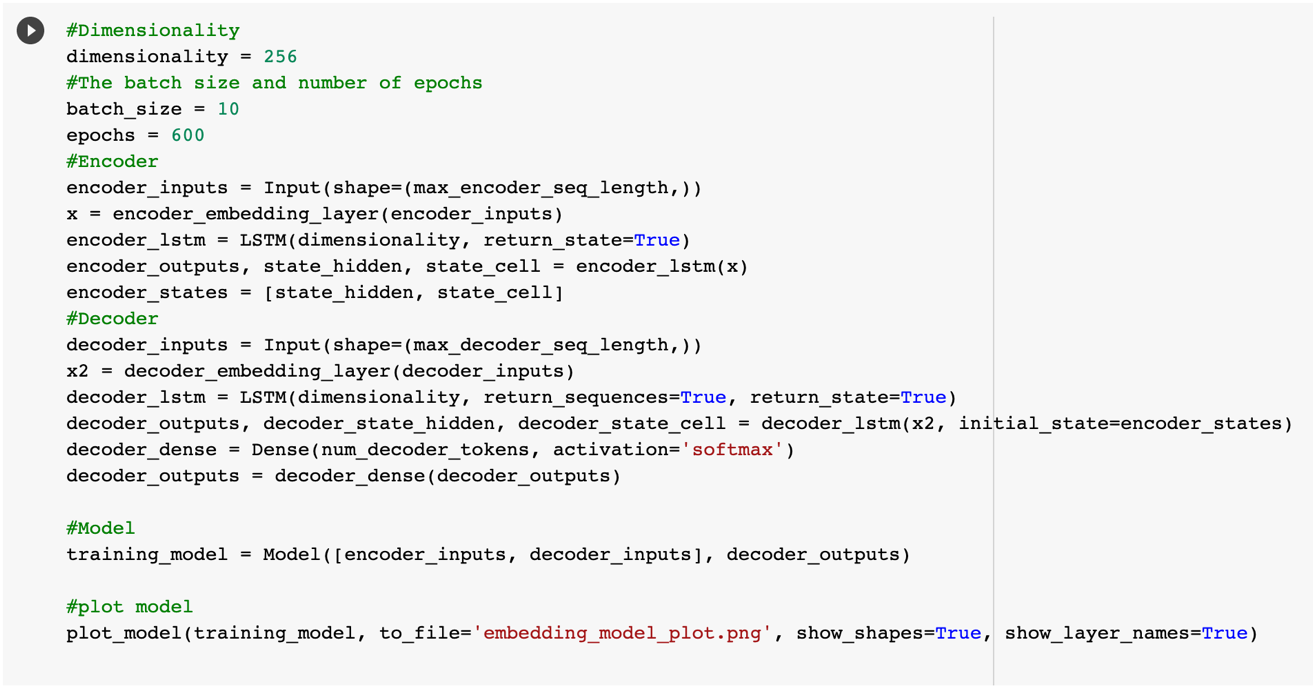
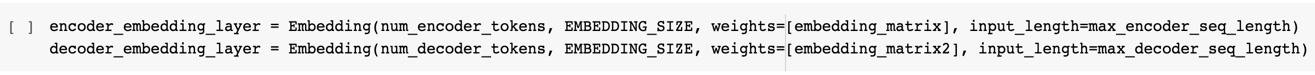
For encoder, the input layer is a matrix for holding the one-hot vectors. Since I have 981 unique encoder input words and the max length of these input sentences is 51, so the dimension of input matrix is 51\*981. Then I stack a LSTM layer, I set ‘return\_state’ to True to return cell state.

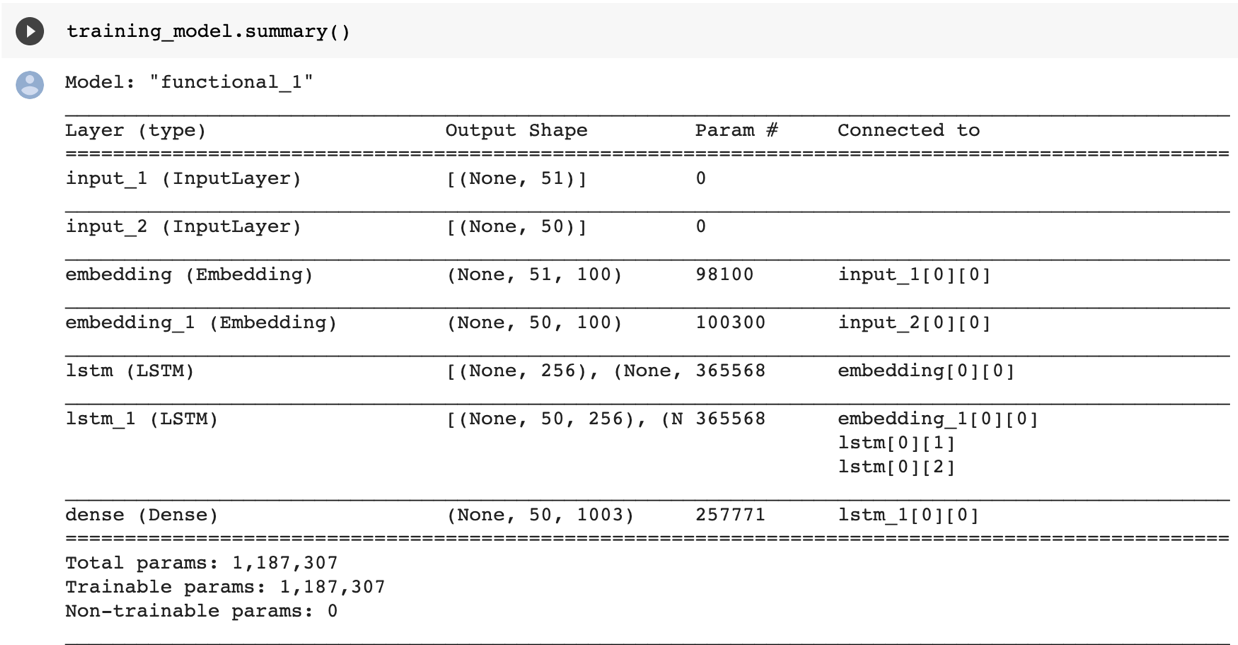
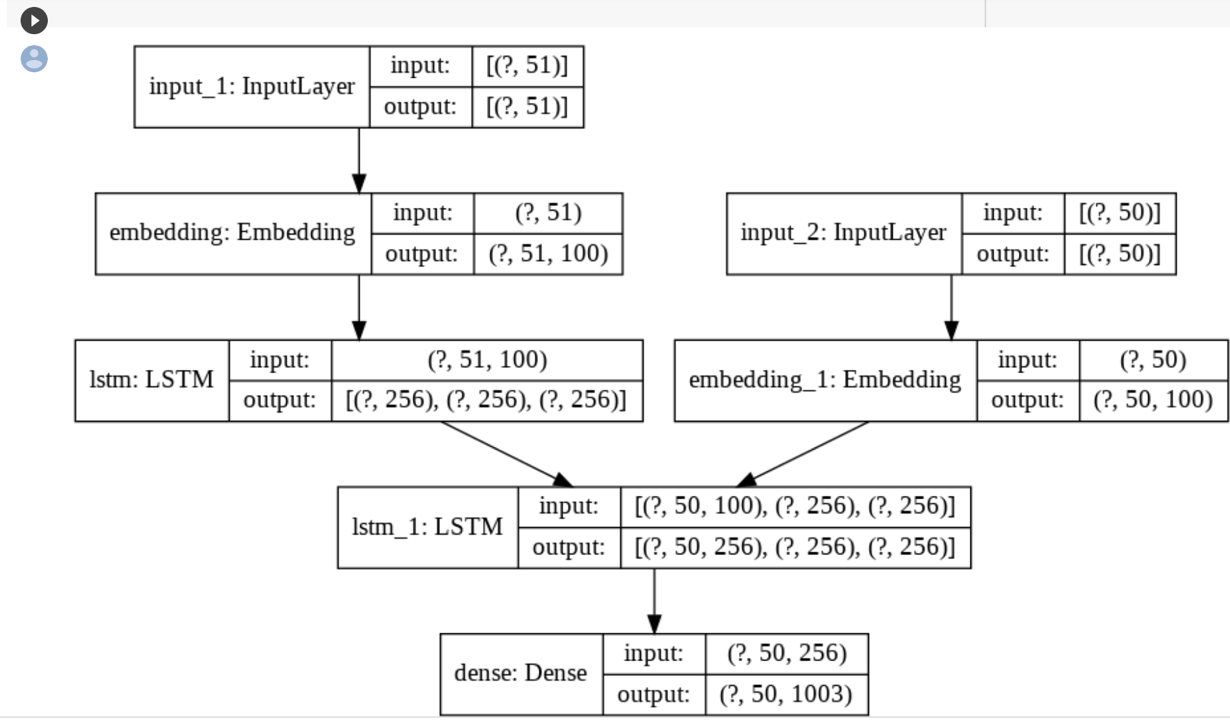
For decoder, I have 1003 unique decoder input words and the max length of these input sentences is 50, so the dimension of input sentences is 50\*1003.Then I stack a LSTM layer, I use encoder output hidden state and cell state with this decoder input matrix as decoder input, and set ‘return\_sequence’ to True to output hidden state for each input time step. Then we stack a dense ouput layer with ‘softmax’ as acitivation function.

This is the summary for this model:



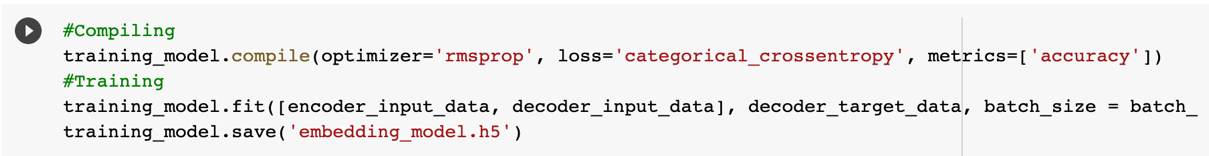
This is the #2 model with embedding layer





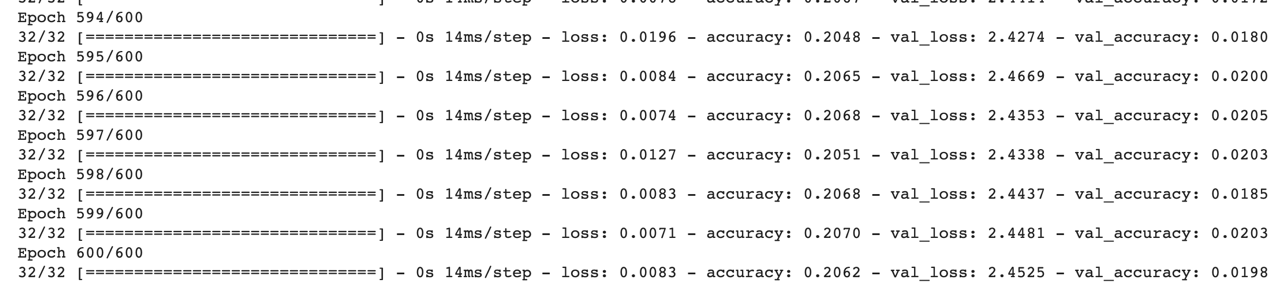
The difference for this one is that we add embedding layers. For encoder, I add and embedding layer between input layer and LSTM layer. Since embedding size is 100, the input matrix feed to LSTM will became 51\*100. For decoder, the input matrix feed to LSTM will became 50\*100.

Then I use the same parameters to train two models both with ‘rmsprop’ as optimizer, ‘categorical crossentropy’ as loss function, and ‘accuracy’ as metrics.

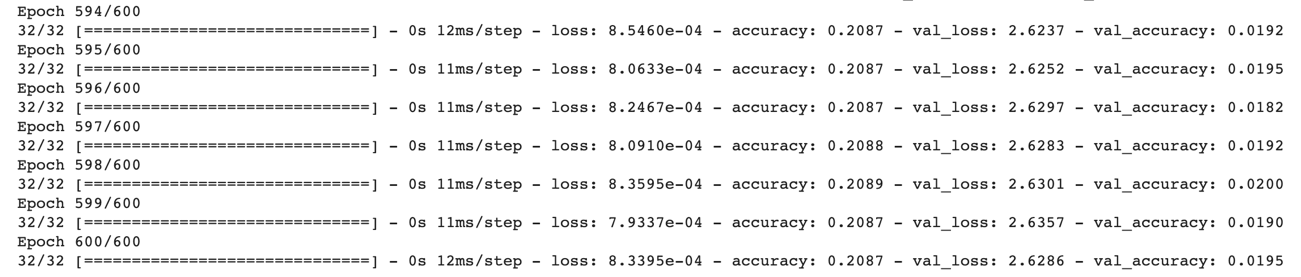


* Model Evaluation

One-hot encoding method training process:



Word embedding method training process:



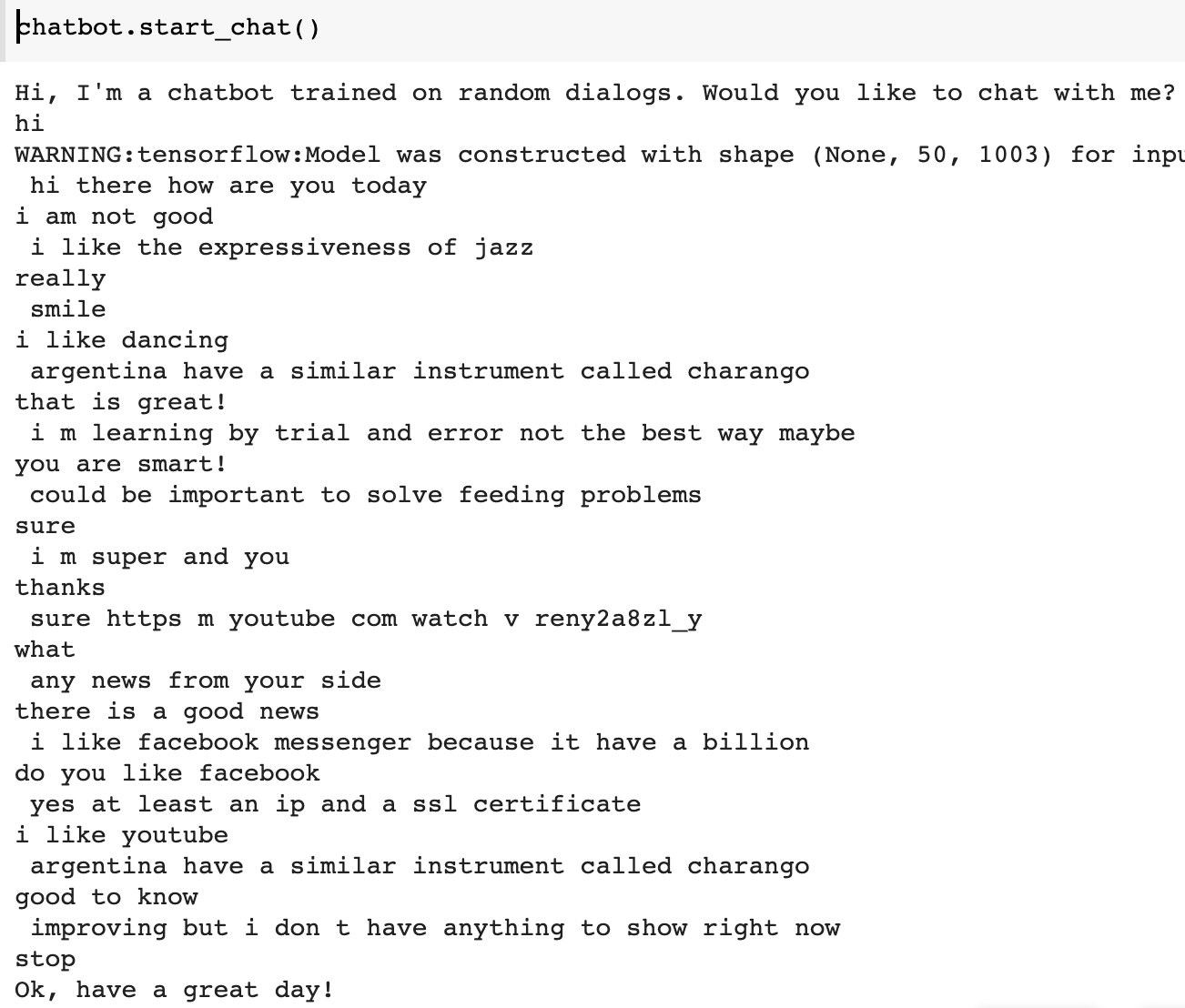
With same epochs of 600, batch size 10, I find results for two methods are similar and both have overfitting problem. Then I tried to change hyperparameter to make a comparison. I change optimizer to ‘Adam’ to see what the results will be.

Word embedding method with ‘Adam’ optimizer:

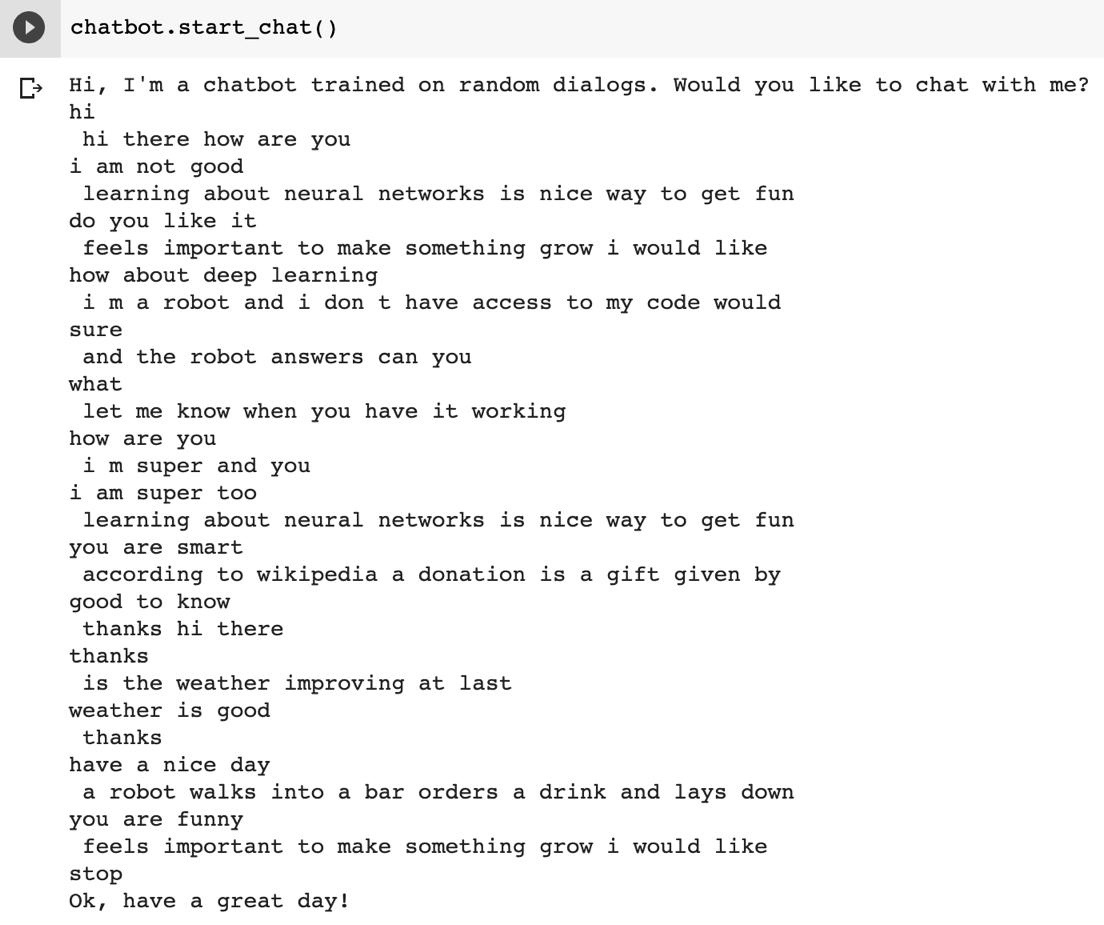
A picture containing table

Description automatically generated

For this one, the validation accuracy has a little bit improvement but also did not perform so well, since I consider more about how our chatbot can talk with us, let’s take a look at the interesting conversations:



This is my conversations using one hot methods chatbot.



This is my conversations using word embedding methods chatbot.

We can find sometimes both chatbot will lose their mind and tell something meaningless, but at least we develop our chatbot successfully. Even the word embedding chatbot told me that ‘learning about neural networks is a nice way to get fun’!

* Conclusion

Obviously, my chatbot is not so smart and it is not so human because conversation is an art. Even Facebook says its ‘Blender’ chatbot is the most humanlike ever with 9.4 billion parameters, it still would lose control sometimes. I also tried to change to different optimizer, different batch size and runs more or less epochs, but all of the changes did not give us really good results. Considering the limitation of dataset and computational expense, I suggest to using transfer learning instead of developing from scratch or we can add with attention if we would explore and improve our chatbot furthermore.

* References

Datasets link: https://www.kaggle.com/eibriel/rdany-conversations