Professional E-mail Writing Assistant Using GRU and 1D Convolutions

MS Artificial Intelligence
Columbia University in the City of New York

Skand Upmanyu *su2236@columbia.edu*

Master of Science in Business Analytics *Industrial Engineering and Operations Research*

Abstract: Using appropriate professional language at workplace and universities is of utmost importance and is something often ignored while writing hasty e-mails. Moreover, companies need employees who can communicate with partners and clients all over the world but many fresh graduates often face difficulties in writing effective e-mails. A well-structured e-mail helps in making effective first impressions with the client and portrays professionalism. Google has recently come up with an Smart Compose [1] feature which tries to complete sentences to save typing effort and time. Inspired by this feature, we aim to develop an e-mail writing assistant which can help improve the effectiveness of the email by completing sentences in a professional manner. For this purpose, we create sequence models using Long Short-Term Memory (LSTM) cells, create character embeddings, and develop a deep learning architecture to create an e-mail assistant which can complete partial words and can also predict next words, given a sequence of words or a phrase. Finally, we use transfer learning to customize our assistant based on the emails sent by a given user making it adaptive to the unique writing style of a working professional. We expect this assistant to be helpful in writing effective e-mails and in saving typing effort and time.

I. Introduction

Almost everyone has experienced a predictive keyboard on our smartphone, which suggests upcoming words for super-fast typing. Before Google's own official keyboard app added prediction, companies like Swype and SwiftKey built keyboards that learn the words we use most often. Prediction can be hilariously bad at first, and takes time to train. Plus, we have to fork over a good bit of data about what you type in order for predictive keyboards to work well. In this study, we address the issues of initial bad predictions and will also curate the predictions for writing well-structured professional e-mails.

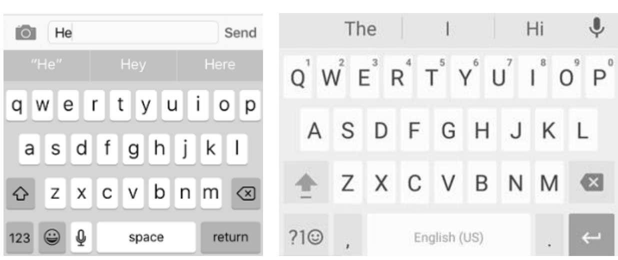

Figure 1.1 Predictive keyboard examples: keyboard in iOS (left), keyboard in Android (right)

Below are some characteristics of typical predictive keyboards:

- Often initialized with dictionary words and are adaptive to the typing style of user
- Can predict 3 possible ordered completions
- Can predict an incomplete word
- Can predict the next word

In this study, we will create an email writing assistant similar to Gmail's Smart Compose, but will curate it for predicting text in the context of professional emails. To build and improve upon the characteristics mentioned above in this context, we designed the following approach for this study:

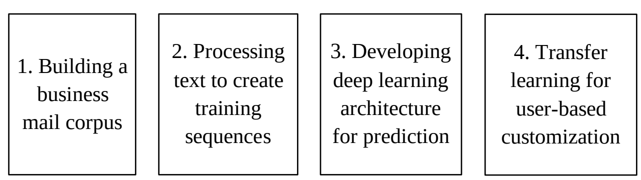

Figure 1.2 Process flow

[1] Google's Smart Compose: https://support.google.com/mail/answer/9116836?co=GENIE.Platform%3DDesktop&hl=en

II. Text Preprocessing

A. Business e-mail corpus

For creating a professional e-mail writing assistant, we used the Enron Email Dataset [2] which contains data from about 150 users, mostly senior management of Enron. Enron Corporation was an American energy, commodities, and services company based in Houston, Texas. From this dataset, we created a compiled all the bodies of the e-mails to create a business e-mail corpus. Only the sent e-mails were considered as we wanted to make sure that we consider only those e-mails which have been written by the user. In this corpus, we did not remove any punctuations or stop words and did not carry out stemming or lemmatization as the output of the model is expected to have all text attributes. The only text preprocessing carried out was to convert all e-mails to lower case characters. The final corpus can be summarized as follows:

Attribute	Value
#E-mails	65,631
#Words	2,172,779
#Unique words	84,361
#Characters	11,583,356
#Unique characters	68

Figure 2.1 Business e-mail corpus

This corpus was used as the starting point for the model i.e. when we do not have any information about the user key strokes, the predictive keyboard would use this corpus to calculate the model weights for making predictions.

B. User e-mail corpus

Getting data for user e-mails is difficult. Therefore, for this purpose, I used my own Gmail data export to create a corpus of user e-mails. The data was extracted using the “Download your data” [3] feature of Google. Only the sent e-mails were considered as we want to make sure that we consider only those e-mails which have been written by the user. The same text preprocessing was carried out to create a user e-mail corpus.

This corpus was used to create user-specific predictions for the input text and would be used to create personalized weights for each user’s predictive model.

The final corpus can be summarized as follows:

Attribute	Value
#E-mails	516
#Words	29,870
#Unique words	4,472
#Characters	164,941
#Unique characters	67

Figure 2.2 User e-mail corpus

C. Creating sequences

For each line in the corpus we created training sequences and target words using sliding windows to create our features and labels. The method adopted to create the sequence data is described below:

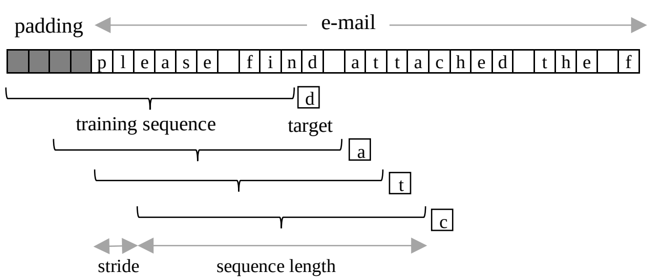

Figure 2.3 Creating sequences

The parameters chosen for generating these sequences were decided based on memory limits and context. Our model was able to do give good performance for a sliding window of length in range [60, 100]. The final parameters chosen are described below:

Parameter	Value
Sequence length	100
Stride	1
Min number of characters	10

Figure 2.4 Parameters for sequence generation

D. Train, validation, and test split

90% of the data was used for training set, 5% for validation set, and 5% for test set. The number of sequences in each of the following sets are as follows:

Set	Business	User
Training set	8,607,647	143,347
Validation set	478,203	7,964
Test set	478,203	7,964

Figure 2.5 No. of examples in each dataset

[2] Enron Email Dataset: https://www.cs.cmu.edu/~enron/

III. Model Architecture

A. Architecture

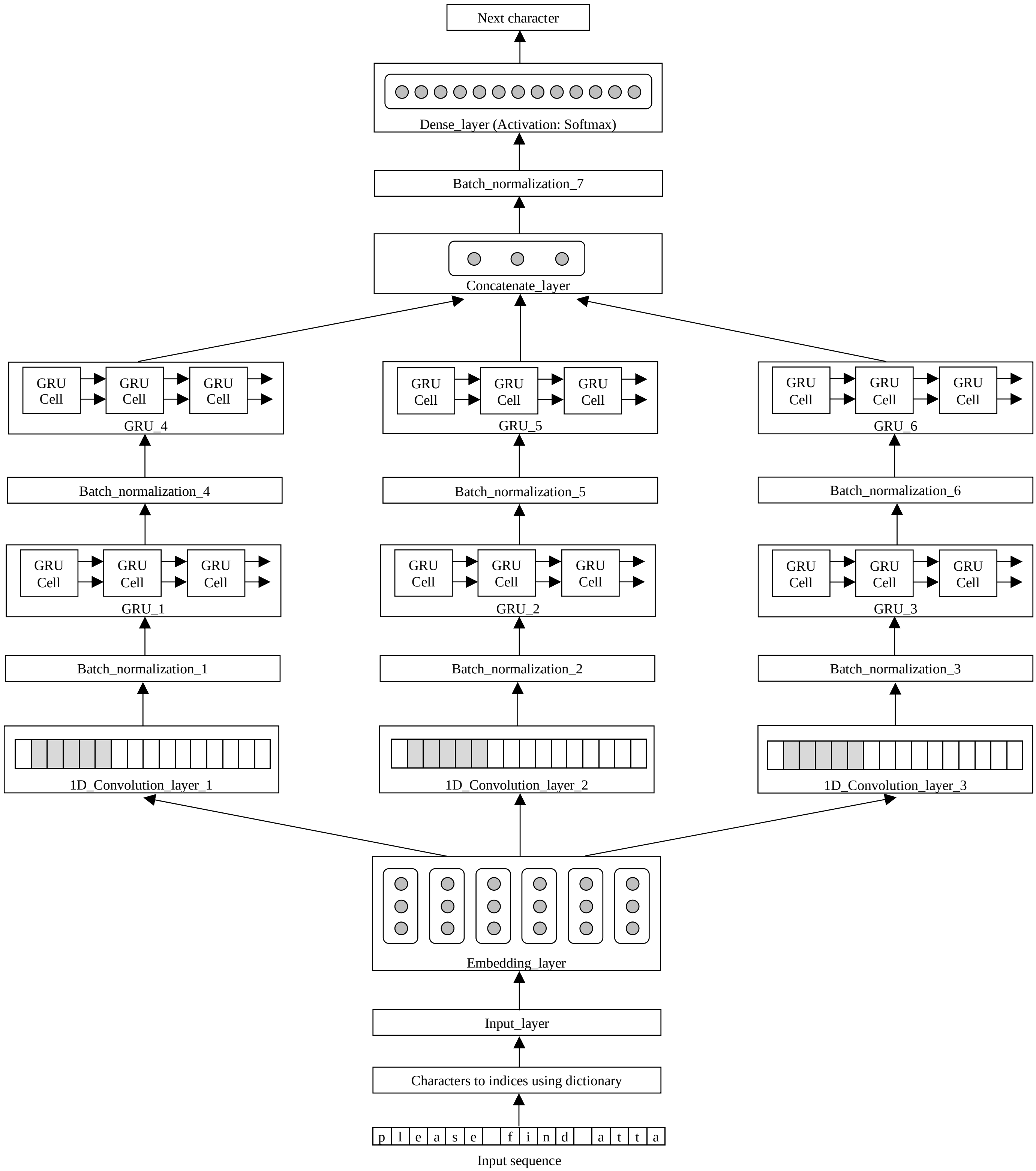

Figure 3.1 Model architecture

B. Layer properties

Layer (type)	**Output Shape**	**Param #**	**Connected to**
Input_layer (InputLayer)	[(None, 100)]	0	
Embedding_layer (Embedding)	(None, 100, 64)	4480	Input_layer[0][0]
1D_Convolution_layer_1 (Conv1D)	(None, 96, 50)	16050	Embedding_layer[0][0]
1D_Convolution_layer_2 (Conv1D)	(None, 96, 50)	16050	Embedding_layer[0][0]
1D_Convolution_layer_3 (Conv1D)	(None, 96, 50)	16050	Embedding_layer[0][0]
Batch_normalization_1 (BatchNorm)	(None, 96, 50)	200	1D_Convolution_layer_1[0][0]
Batch_normalization_2 (BatchNorm)	(None, 96, 50)	200	1D_Convolution_layer_2[0][0]
Batch_normalization_3 (BatchNorm)	(None, 96, 50)	200	1D_Convolution_layer_3[0][0]
GRU_1 (GRU)	(None, 96, 128)	69120	Batch_normalization_1[0][0]
GRU_2 (GRU)	(None, 96, 128)	69120	Batch_normalization_2[0][0]
GRU_3 (GRU)	(None, 96, 128)	69120	Batch_normalization_3[0][0]
Batch_normalization_4 (BatchNorm)	(None, 96, 128)	512	GRU_1[0][0]
Batch_normalization_5 (BatchNorm)	(None, 96, 128)	512	GRU_2[0][0]
Batch_normalization_6 (BatchNorm)	(None, 96, 128)	512	GRU_3[0][0]
GRU_4 (GRU)	(None, 64)	37248	Batch_normalization_4[0][0]
GRU_5 (GRU)	(None, 64)	37248	Batch_normalization_5[0][0]
GRU_6 (GRU)	(None, 64)	37248	Batch_normalization_6[0][0]
Concatenate_layer (Concatenate)	(None, 192)	0	GRU_4[0][0]
			GRU_5[0][0]
			GRU_6[0][0]
Batch_normalization_7 (BatchNorm)	(None, 192)	768	Concatenate_layer[0][0]
Dense_layer (Dense)	(None, 70)	13510	Batch_normalization_7[0][0]

Total params	388,148
Trainable params	386,696
Non-trainable params	1,452

Figure 3.2 Layer properties

IV. Results

A. Training results

The model was trained on the cloud (Google Colab) for 10 epochs. The number of epochs was identified based on the flattening of the Validation loss vs epochs curve. Each epoch took approximately 55 minutes to complete and the entire training process took approximately 9.5 hours. The training and validation loss vs. the number of epochs is show below. We can observe that the validation loss does not decrease significantly beyond 10 epochs

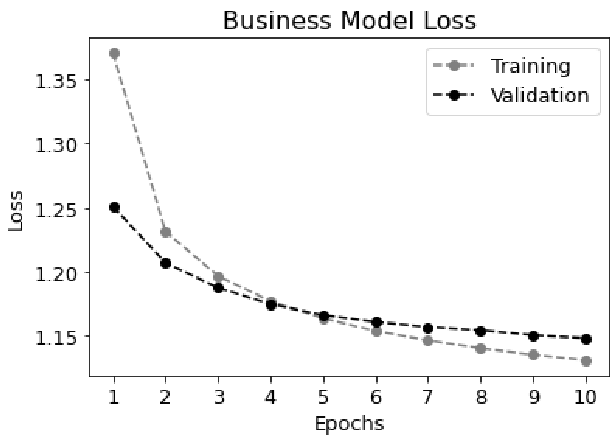

Figure 4.1 Business model loss Vs. Epochs

The model training and validation accuracy vs. the number of epochs is shown below. Similar to the loss, the model validation accuracy does not increase significantly after 10 epochs.

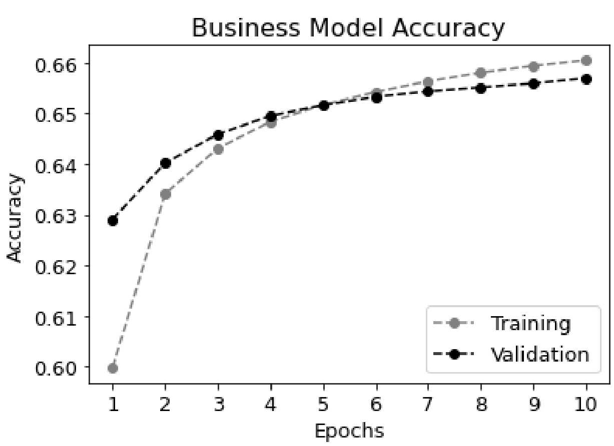

Figure 4.2 Business model accuracy Vs. Epochs

B. Model performance

The final performance of the model on the training, validation, and test data is summarized below:

Dataset	Loss	Accuracy
Training	1.1311	0.6605
Validation	1.1483	0.6570
Test	1.1481	0.6572

Figure 4.3 Business model performance

C. Predictions

For predictions, here is an example sentence which was not present in the training/validation/test set: *"Can you let me know if you have completed the project that we discussed".* The top 5 predictions for each of the inputs are shown below:

Input	Predictions
"Can you let "	'me ', 'him ', 'you ', 'them ', 'change '
"Can you let me "	'know ', 'see ', 'have ', 'the ', 'check '
"Can you let me know "	'if ', 'what ', 'how ', 'and ', 'that '
"Can you let me know if "	'you ', 'this ', 'i ', 'any ', 'we '
"Can you let me know if you "	'have ', 'need ', 'want ', 'can ', 'are '
"Can you let me know if you have "	'any ', 'questions ', 'some ', 'to ', 'not '
"Can you let me know if you have compl"	'eted ', 'icated ', 'y ', 'ained ', 'oyed '
"Can you let me know if you have completed "	'the ', 'and ', 'in ', 'on ', 'by '
"Can you let me know if you have completed the "	'contract ', 'process ', 'state ', 'following ', 'deal '
"Can you let me know if you have completed the proj"	ect ', 'ust ', 'ose ', 'icing ', 'dc '
"Can you let me know if you have completed the project "	'for ', 'in ', 'when ', 'and ', 'contract '
"Can you let me know if you have completed the project that "	'we ', 'the ', 'i ', 'you ', 'he '
"Can you let me know if you have completed the project that we "	'will ', 'have ', 'can ', 'are ', 'should '
"Can you let me know if you have completed the project that we disc"	'ussed ', 'overed ', 'repent ', 'less ', 'ase '

Figure 4.4 Sample predictions

The prediction logic is based on the occurrence of a space character. The model keeps on predicting the next character until it predicts a space character. This logic makes sure that the model completes either the current incomplete word or if the current word is already completed, it predicts the next word.

V. User-level customization

Using the weights of the business model as the initial weight, a new user-specific customized model was created using the user e-mail corpus data. This data would make the predictions based on the writing style of the user which would be unique to him/her.

A. Training results

The model was trained on the same environment for 5 epochs. The number of epochs was identified based on the increase of the Validation loss after 5 epochs. Each epoch took approximately 55 seconds to complete and the entire training process took approximately 5 minutes. The training and validation loss vs. the number of epochs is show below. We can observe that the validation loss flattens near 5 epochs and if we train longer, the validation loss starts to increase. In this situation, we have a small corpus for user specific data. Therefore, we need to limit our number of epochs to make sure that we do not overfit the model.

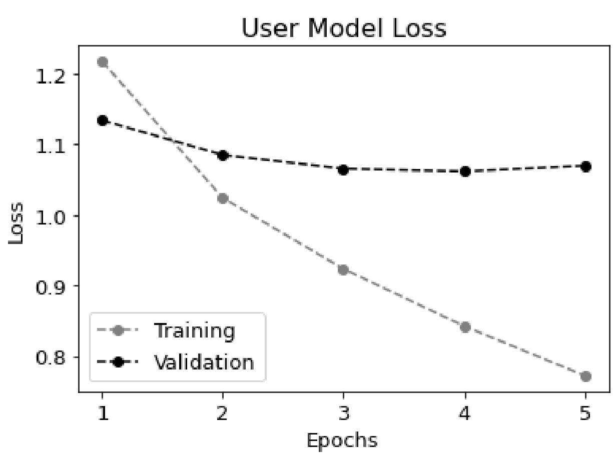

Figure 5.1 User model loss Vs. Epochs

The model training and validation accuracy vs. the number of epochs is shown below. Similar to the loss, the model validation accuracy does not increase significantly after 5 epochs.

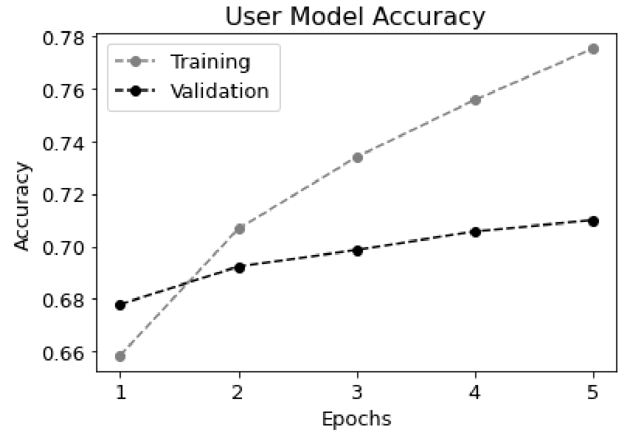

Figure 5.2 User model loss Vs. Epochs

B. Model performance

The final performance of the model on the training, validation, and test data is summarized below:

Dataset	**Loss**	**Accuracy**
Training	0.7724	0.7751
Validation	1.0697	0.7099
Test	1.0574	0.7064

Figure 5.3 User model performance

C. Predictions

For predictions, here is an example sentence which was not present in the training/validation/test set: *"Hi Professor, hope you are doing well. I wanted to thank you for taking the time for the meeting yesterday".* The top 5 predictions for each of the inputs are shown below:

Input	**Predictions**
"Hi Prof",	'essor, ', 'it, ', '. ', ' to ', 'ord '
"Hi Professor, ",	' i ', 'i ', 'and ', 'we ', 'so '
"Hi Professor, hope you ",	'are ', 'had ', 'want ', 'can ', 'got '
"Hi Professor, hope you are ",	'doing ', 'going ', 'already ', 'really ', 'still '
"Hi Professor, hope you are doing ",	'well. ', 'a ', 'the ', 'forward ', 'great '
"Hi Professor, hope you are doing well.",	' ', '..i ', 'come ', 'upmention ', ', '
"Hi Professor, hope you are doing well. I wanted to thank ",	'you ', 'to ', 'some ', 'please ', 'very '
"Hi Professor, hope you are doing well. I wanted to thank you ",	'for ', 'then ', 'so ', 'and ', 'might '

"Hi Professor, hope you are doing well. I wanted to thank you for ",	'taking ', 'your ', 'catch ', 'reaching ', 'sure '
"Hi Professor, hope you are doing well. I wanted to thank you for taking ",	'the ', 'my ', 'some ', 'it ', 'a '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the ",	'time ', 'mean ', 'projects ', 'distance ', 'students '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the time ",	'to ', 'for ', 'and ', 'with ', 'in '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the time for ",	'the ', 'you. ', 'a ', 'me. ', 'some '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the time for the ",	'interview. ', 'exam. ', 'meeting ', 'same ', 'time '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the time for the mee",	'ting ', 'p ', 'bert ', 'ming ', 'sing '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the time for the meeting ",	'with ', 'for ', 'and ', 'in ', 'on '
"Hi Professor, hope you are doing well. I wanted to thank you for taking the time for the meeting yeste"	'rday. ', 'e ', 'ad ', 'd ', 'ct '

Figure 5.4 Sample predictions

V. Discussions and Conclusions

A. *Discussion*

The professional e-mail writing assistant proposed in this study performs well but has certain limitations. In a production level e-mail writing assistant, we usually observe that the predictions are made only when the model is completely sure about the predictions. This would require using the output probabilities of the model to derive a confidence for the predictions. This confidence would indicate how likely the predictions are going to make sense in a given context. If the output probabilities are above a certain threshold then predictions would be shown to the user, otherwise the application would wait for more input. Before implementation, this would be a crucial analysis to derive this threshold and would be a key factor in determining the success of the application.

B. *Conclusions and future scope*

This study demonstrates the use of non-sequential models and the combination of 1D Convolutions and Stacked GRUs for time series predictions. The use of Convolutions makes sure that we are looking at di-grams, tri-grams, and so on and stacked GRUs keep the information about context. The embedding layers help in the use of vowels, numbers, and special characters and batch normalization helps in better training. Finally, the non-sequential model helps in creating an ensemble of 3 architectures and prevents overfitting. This architecture can be used in a similar fashion for other applications like music generation, stock prediction, and text generation.

VI. References

- Build a simple predictive keyboard using python and Keras: IJAS A H, Analytics Vidhya [3]
- How to Develop Word-Based Neural Language Models in Python with Keras: Jason Brownlee, Machine Learning Mastery [4]
- Making a Predictive Keyboard using Recurrent Neural Networks — TensorFlow for Hackers (Part V): Venelin Valkov, Medium [5]

VII. Resources

Github link (with code and data) [6]

[3] https://medium.com/analytics-vidhya/build-a-simple-predictive-keyboard-using-python-and-keras-b78d3c88cffb
[4] https://machinelearningmastery.com/develop-word-based-neural-language-models-python-keras/
[5] https://medium.com/@curiousily/making-a-predictive-keyboard-using-recurrent-neural-networks-tensorflow-for-hackers-part-v-3f238d824218
[6] https://github.com/skandupmanyu/Predictive_Keyboard