Voice Patch

Carl Yu, Jerry Chen, Xinhai Wei
{Qkyu, X67wei, Zj7chen}@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Many schools, companies and organizations have opted to use online conference tools such as Zoom and Skype due to the serious COVID-19 pandemic. However, due to the unstable connections, many users are frustrated by the quality of online meetings, especially when some fragments are missed in the audio. We aim to create a tool that first transcribes the audio, then identifies the missing fragments (if any), and finally fills in the fragment based on the surrounding context. For this paper, we have created a tool named Voice Patch that uses DeepSpeech to transcribe the audio and uses RNN models to complete the fragment based on the assumption that the audio is one sentence and the missing fragment is exactly one word. We were able to achieve a training accuracy of **16.954 %** and validation accuracy of **36.25%**. We believe that this tool can be really helpful in improving the user experience in using online meeting tools in the future.

Introduction

- As COVID-19 struck the globe, many schools, companies and organizations have been using video conferencing software such as Zoom, Skype, and Microsoft Teams for meetings, exams and classes. However, due to either loss of packet or unstable internet connection, it is possible that parts of the sentences in audio conversations are filled with noise, or seriously missing. As Garel wrote in the HuffPost, there are many people like Christen Shepherd whose voice "dipped in and out of clarity, the sound glitching and going dead every few words over the course of the 20-minute conversation, because of poor cellular service" (Garel 2020). Moreover, on August 13 2020, the Education Minister issued a Policy/Program Memorandum with new minimum requirements of 225 minutes of "Zoom-style" classes starting in September. For Christen's children, it would be a miserable 225 minutes of school as they struggle to participate in class activities.

On the business side, many business professionals also value video/audio calls when choosing their video conferencing software. OWLLab surveyed 1017 business professionals in 2019, 51% of the executives chose their video conferencing software to "provide high-quality video calls for their team" (owl 2019). Lifesize's survey also reports that "poor audio quality/echo" is one of the top detractors to call experience. (Lifesize 2019).
If the audio conversations are automatically transcribed and potential missing text fragments are auto-completed with high accuracy, the users will be less likely to be bothered with missing parts, and have a better understanding of how conversations are ought to be like.

As described in the scenarios in the introduction section, incomplete audio fragments potentially cause misunderstanding and frustration during the usage of online meeting technologies. However, to our best knowledge, there does not exist a mature tool which can efficiently solve the problem we described above. To tackle this problem, our group will focus on a compact plan towards solving this problem. In particular, this report will describe a tool that will transcribe the audio message with high accuracy and attempt to fill in missing components (if the sentence is incomplete), so that the transcript generated from the tool may be used to clarify misunderstandings when the audio is unclear or incomplete.

This is a particularly interesting topic since many of the prior works are either concentrated in speech translation or sentence (word) completion. Thus, by connecting those two technologies, identifying parts that need to be auto-completed, and finish the bad sentences with high accuracy will be our task to tackle. In our project, we decided to specifically focus on a practical tool such can amend one sentence from an audio-file input.That is, if the sentence is faulty, our tool will attempt to

correct the sentence by considering one word only. Otherwise, our tool will transcribe the sentence as close as possible to the audio file provided. Our output is a text-based sentence which the model believes is complete.

We divide our methodology into two stages. The first stage is to use DeepSpeech to translate audio messages to text messages. We will use DeepSpeech, a Speech-To-Text engine which takes audios over a time interval as input and generates a text transcript as output. The transcript marks down the words recognized by the engine, and their duration in the time interval. The engine is based on a Recurrent Neural Network (RNN) which will be discussing in the later Methodology part.

The second stage is to gain input from our previous stage and feed the data received in the first part into both the blend of LSA, the Good-Turing language model, and the Recurrent Neural Network model(RNN) and tune the hyper-parameters. The model will finally return a text output after trial completion of the sentence. In our project, we create a model named Word-Prediction to predict the missing word in a sentence. During the training stage, we will use the word embedding generated from word2vec, which is implemented in gensim.(Gensim is an open-source library for unsupervised topic modeling and NLP by statistical machine learning.) Testing data sets will be based on Mozilla Common Voice, and we generate sentences from the data-set with exactly one word missing. Briefly, the method is to break sentence into three word embedding(vectors). Our idea to generate the missing word based on calculation on these vectors. More details will be revealed in Methodology part of this report.

We will measure the performance for the hyper-parameters by measuring the exact match rate between the generalized text sentence with a transcript that we expect the model to output. The total accuracy is a number-based measurement which calculated by dividing the number of times that the output of our model exactly matches the transcript with the total number of testing cases. Through the supervised learning process, we setup two milestones in our plan. The first milestone is to convert audio input, which is one sentence of human voice, into text. Secondly, we conduct trials in all positions to insert a word from vocabulary. For vocabulary we currently using "GoogleNews-vectors-negative300.bin". In case the text sentence is already finished, we then also allow the inserted word to be empty (ϵ), to ensure that our model will not change good sentence to bad one. After trials in all positions, we select the best plan following our algorithm from them and output the sentence.

For our main results, we attempted to use existing technologies to tackle the problem we described above. Surprisingly, we succeed in using DeepSpeech and 1 layer of LSTM in filling the missing part of one sentence with a word from the vocabulary set. During our research, we modify the number of Epochs, number of Layers, and Dropout rate to get the highest accuracy, and achieved the an training accuracy of **16.954%** and validation accuracy of **36.25%** at 5 Epochs, 1 Layer and 0 Dropout rate. This is a fairly high accuracy comparing to randomly filling the blank from the vocabulary set. With intuitive interpretation, our model isolate the choices of the missing part from thousands to three.

Note that this project shows the following:

- Through our implementation, one can fill in the missing word with an accuracy of 0.3625.
- This project can be further developed so that the model can allow more missing words, or even missing sentences.
- This project shows that using data generated from well-established model will still achieve a reasonable result.
- We created a fast tool MuteOne which can be used to create the training and validation data that we need.
- We analyze the influence of the number of Epochs, number of Layers, and Dropout rate in our RNN training model.

Related Work

To tackle the problem of auto-completion, a relatively straight forward approach is to use the voice sharpening techniques described in the paper “Audio Super-Resolution Using Neural Works” (Kuleshov, Enam, and Ermon 2017) to upgrade the quality of the voice. However, it is entirely possible that parts of the sentences are completely missing. So this approach certainly does not apply to all of the cases. There are many results have been produced in the field of speech recognition. For example, the Deep Speech described in “Deep Speech: Scaling up end-to-end speech recognition” (Hannun et al. 2014) is an excellent example of voice-to-text recognition. Lastly, various scholars have proposed models regarding sentence completion from “Computational Approaches to Sentence Completion” (Zweig et al. 2012). Deep learning is also broadly used in this task in “Deep Learning for Audio Signal Processing” (Purwins et al. 2019) and “Knowledge-Powered Deep

Learning for Word Embedding" (Bian, Gao, and Liu 2014). However, we have not discovered any tools that solve our problem directly. Currently, we found Word2Vec, which is a useful tool that maps every word to a specific vector in the vocabulary. This tool may help us in some area of finding a word embedding of a word and further discover the relations between words in the a sentence. We will keep searching for potentially feasible solutions, and update them here.

A fun story to know about is that Geoffrey Zweig along with other authors have used a simple linear combination of the outputs of the other models, as well as the new model to build a k4 validation. They then used the k4 cross validation to validate on several language models and found that there are combinations of n-gram, and the local n-gram results got significantly better.

Methodology

Audio to text

We will use DeepSpeech, an open-source Speech-To-Text engine that was described in Deep Speech: Scaling up end-to-end speech recognition (Hannun et al. 2014) in 2014. An implementation of this engine is maintained by Mozilla on GitHub. We will first briefly introduce the model.

The DeepSpeech model will take speech spectrograms over a time interval T as input and generate a transcript. (Speech sepctrograms are audio features that describes the frequency for a time interval. Call this input x) The model first uses a Recurrent Neural Network (RNN) to attempt to construct a a transcript y such that $\hat{y}_t = P(c_t|x)$ for time slice $t \in 1, 2, \cdots, T$.

The RNN described in the paper uses 5 layers of hidden units, and the first 3 layers are non-recurrent and uses the clipped rectified linear activation function $g(z) = \min\{max0, z, 20\}$. Thus, the first 3 layers are computed as follows:

$$h_t^{(\ell)} = g(W^{(\ell)} h_t^{(\ell-1)} + b^{(\ell)})$$

where ℓ denotes the current layer number, and W, b are the weight and bias function respectively. t represents the time interval described in the previous paragraph. The fourth layer is a bi-directional layer with a forward recurrence and a backward recurrence as follows:

$$h_t^{(f)} = g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_{t-1}^{(f)} + b^{(4)})$$

$$h_t^{(b)} = g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_{t+1}^{(b)} + b^{(4)})$$

and $h_t^{(4)} = h_t^{(f)} + h_t^{(b)}$.

The fifth layer is also non-recurrent and have the same output as layer 1,2 and 3. The output layer is a standard softmax function that computes the probability for each time slice t and each letter in the alphabet (including space, apostrophe and blank). The error is measured by the Connectionist Temporal Classification (CTC) loss function, which is typically used in speech recognition. Finally, the model is trained via back-propagation.

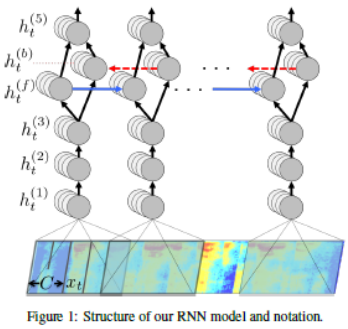

Figure 1: Structure of our RNN model and notation.

Figure 1: The layout of DeepSpeech. Adapted from Awni Hannun et al's Deep Speech: Scaling up end-to-end speech recognition (Hannun et al. 2014)

In order to reduce the variance, a dropout rate of 5% to 10 % is applied on layer 1,2 and 3 during the training process.

Note that the RNN does not guarantee 100% accuracy in the transcript. Hence, DeepSpeech used a N-gram model,which can be easily trained with large text corpus, in attempt to correct the output some of the utterance. In general, DeepSpeech aims to find a sequence of characters that maximizes the following objective function:

$$\log(P(c|x)) + \alpha \log(P_{lm}(c)) + \beta wordcount(c)$$

where α and β are parameters that tuned by cross-validation. P_{lm} denotes the probability of c based on the N-gram model. DeepSpeech uses a highly optimized beam search algorithm with beam size between 1000-8000 to maximize this objective.

There are several optimization techniques that DeepSpeech uses, but it is beyond the scope of this paper.

We will use DeepSpeech because the recent releases of pre-trained model are trained with the CommonVoice Dataset (an open source dataset of voices that anyone can use to train speech-enabled applications) maintained by Mozilla. The data set consists of audio files as well as transcript of a single sentence, which aligns perfectly with our project's assumption. Also, the model supports fine-tuning,

which implies that we can also use the dataset described above to fine-tune our model.

Fill in the missing word

A model called Word-Prediction is being used in this case to predict the missing word in a sentence. The testing data set comes from the transcript of Mozilla Common Voice, composed of sentences with exactly one word missing.

In 2013, word2vec was introduced by Tomas Mikolov et al in the paper "Efficient Estimation of Word Representations in Vector Space" (Mikolov et al. 2013). The paper contained 2 models, and one of them is the Skip-gram model, which attempts to predict the center word given the context around it, and it is also what we are using in this model to generate word embedding

During the training stage, we will use an implementation of word2vec from gensim and the training data to train the word2vec of every single word. Gensim is an open-source library implemented in Python, and is used for unsupervised topic modeling and natural language processing, using modern statistical machine learning. The sentence will be broken down into three individual word vectors, and these vectors will contain the part of the sentence before the missing word, the part after, and all of the words in the vocabulary. The idea is to add the first vector with the vector of all words in the vocabulary, and compare the cosine similarity of the two with the second vector to see which one has a better match. The reason we do this is because, according to the paper that introduces word2vec, we know that the word vectors are capable of doing addition. The vector that we get after adding two other vectors is the meaning of those 2 words combined together. For example:

$$embedding(king) - embedding(man) + embedding(woman) = embedding(queen).$$

The model takes all of the potential choices for the missing word and form a vector out of it. By filtering out the least frequently occurred words will make the result better, because there might be misspelling in those words. In the training process of word vector, the misspelled words will be embedded to somewhere else along with the correct spelling words. The next step is to obtain a validation set of size $batch\_size * 10$ from training set. If the output from RNN has a cosine similarity $>= 0.4$ with the expected vector, then we treat it as the correct result. We also define a baseline model. We set the baseline model randomly pick a choice from the potential choices for the missing word to against our model. We will compare the accuracy using the comparative test on these two models.

In the testing phase, suppose a sentence has x number of words, since we don't know which word is missing from the sentence, so we will predict what the sentence is like each time by removing a word from the original sentence. That will generate $x - 1$ number of multiple choice questions. Every multiple choice question has v choices, where v is the vocabulary number trained from the corpus, including empty string, and eventually we will pick a sentence that makes the most sense out of it. The way we do this is to put every single word into the space, which is the third vector, and add it to the first part of the phrase to see which one has the greatest Cosine Similarity with the second part of the phrase, and that word is likely the missing one. There will be a difference of approximately 2%.

Results

In order to validate our model, we use the model that was trained using CommonVoice DataSet. Since this model is pre-trained and open-source, there is no need to split the test data here. As for Word-Prediction, we will use the transcripts from CommonVoice Dataset transcript and perform a 10-fold validation with the following parameter attempt to fill the missing spots (if any) in the sentence. For testing purpose, we use the following parameter:

learning_rate = 0.000018
epoches = 2
batch_size = 256
display_step = 100
validation_size = batch_size $*10$
RNN model: 1 layer LSTM (hidden layer size = 800)
no dropout
word vector size: 300

We adapted those parameters from (https://github.com/pochih/Sentence-Completion/blob/master/introduction.pdf). The GitHub project used those parameters for filling blanks for SAT-like questions, which is a subset of our problem. Hence, this is a good starting point. we will use the audio from a selected set of sentences from Tatoeba, which is a large database of sentences and translations, where its contents come from contributions of thousands of peoples as the input, and the output is compared against the existing transcript from Tatoeba to yield the Word Error Rate for each sentence that is calculated as

follows:

$$WER = \frac{C}{N}$$

where C is the number of correct words in the same position as the transcript and N is the number of words in the sentence. Using our performance metric, we will use standard error of our metric, to expose the precision of the means or in comparing and testing differences between means.

This experiment will provide us with a numerical index about the accuracy of our trained model. We choose Common Voice as the trained set since it contains numerous audio files in English comes from different countries and regions.Based on its official website, it provides a dataset with 1469 validated hours. Also, given the Common Voice data-set provides a corresponding transcript for each of audio file, it is convenient for us obtain the transcript necessary for validation purpose, as described in our Methodologies part. For testing, we choose Tatoeba since it's a large database of sentences and translations, which comes from contributions of thousands of peoples from diverse places. Compare to Common Voice, Tatoeba provides better resources about the sentences, which is vital in the process of testing the correctness of our model. As our final output is a text message of full sentences, a mature and abundant test data-set of sentences like Tatoeba would be preferred. Our negative samples will simply be the original complete audio files and transcript from this data-set. As our performance metrics, WER(Word error rates) is a commonly used index which has a high correlation to perplexity that defines how well a model predicts a word (Klakow and Peters 2002). In our performance metric, we modify the classical measurement by only consider the proportion that the predicted word exactly matches the transcript we provided. We choose WER not only because of its high usage in the field of sentence completion, also its low computation cost. For each sentence, our method only needs comparison between two words, which is the predicted word and the expected word in the transcripts.

Implementation

Manipulating Datasets To derive our dataset, we first use Deepspeech with pre-trained model to convert the audio files from Common Voice into transcripts. According to the Deepspeech Github, the pre-trained model achieves an 7.06% word error rate on the LibriSpeech clean test corpus (ReubenMorais 2020). Hence, 92.94% of the transcript is expected to be correct. This is an acceptable loss as the Common voice dataset provides "Validated" audios, by which refer to audio files that have been validated matching their transcripts. However, we need one more step before we directly use the file as the input: as our setup, we need to ensure that the file should miss at most one word. To create training data, we developed a simple useful tool, MuteOne, which has the functionality that randomly remove only one word from a one-sentence audio file. After receiving the audio file, MuteOne would use Deepspeech to generate the transcript for that sentence. Referring the transcript, MuteOne randomly selects one word to be removed and uses ffmpeg (ffmpeg developers 2020), which is a opens source powerful command line audio manufacturing tool, to silence the time interval of that word in the audio following the transcript which provides the approximate time interval for each word. To achieve randomness, we use the random function to select a word and silence it using our implementation of tool. MuteOne may also receive multiple audio files, randomly generated positions of words in sentences, and mute these words in the audio file, and output multiple files. After obtaining the modified file, we used Deepspeech we mentioned above to generate a new transcript with each one.

For example, given the sentence "I am a god." in WAV format, which is a sound-only audio format. The following transcript, which is a JSON file, will be produced when the audio is fed into DeepSpeech.

Suppose the word "am" was chosen by MuteOne and supposed to be removed, then MuteOne would refer to the transcript above, set the volume of intervals containing the word "am" in the file to 0, and generate a new WAV file. Once again, we use Deepspeech to produce the following transcript, a JSON file containing timestamps of words, to show that MuteOne successfully removed the word.

```
{
  "transcripts": [
    {
      "confidence": -11.989204406738281,
      "words": [
        {
          "word": "i",
          "start_time": 1.32,
          "duration": 0.02
        },
        {
          "word": "am",
          "start_time": 2.4,
          "duration": 0.38
        },
        {
          "word": "a",
          "start_time": 3.86,
          "duration": 0.16
        },
        {
          "word": "god",
          "start_time": 4.98,
          "duration": 0.26
        }
      ]
    }
  ]
}
```


Figure 2: JSON File 1

As shown in the JSON file above, the word "am" which appeared in the first transcript disappears in the second transcript.

Word Prediction As for the stage WordPrediction, since we are unsure about the configuration of the best-performing model, we decided to leave some of the hyper-parameters as global parameters for fine-tuning. A list of those parameters include the **number of epochs**, and the **dropout rate**, the **learning rate**,etc. Among them, the number of epochs represents number of passes of the whole training dataset in which the algorithm has finished. Dropout rate represents the proportion that we does not used in the training set to avoid over-fitting of the training data-set. Learn rate is the size of step in Gradient descent. By controlling these hyper-paremters, we want reduce the total error of our model. To ensure that our model still has high accuracy in extremely long sentences, which may contain more than 10 words, we decided on using LSTM cell over GRU and


```
{"transcripts": [
        {
      "confidence": -16.411922454833984,
      "words": [
        {
          "word": "i",
          "start_time": 1.32,
          "duration": 0.02
        },
        {
          "word": "a",
          "start_time": 3.82,
          "duration": 1.14
        },
        {
          "word": "god",
          "start_time": 5.1,
          "duration": 0.16
        }
      ]
    }
}
```


Figure 3: JSON File 2

linear cell which are capable of remembering long sequences. Although LSTM takes more time to train, the potential of higher accuracy is preferred over GRU and linear units. We also choose an optimizer to reduce the time required. we decided to use the Gradient Descent optimizer, which is straight forward to implement and with high computation efficiency, to mediate the vanishing Gradient problem.

Performance

Since DeepSpeech has a high accuracy, we mainly focused on training WordPrediction. Initially, we ran the RNN model with 2 epochs,1 layer of LSTM with a dropout rate of 0.1, and the result is as follows:

Note that the training accuracy increases from epoch 1 to epoch 2. We deduce that increasing the number of epochs will increase the training accuracy of the model. By running the model with 5 epochs, the results are as follows:

From the figure, we observe that the training accuracy did not improve after the third epoch. However, the validation accuracy increased slightly. Note that the validation accuracy is calculated for the whole model instead of only WordPredict. We show the Validation accuracy versus Epoch in Figure 6.

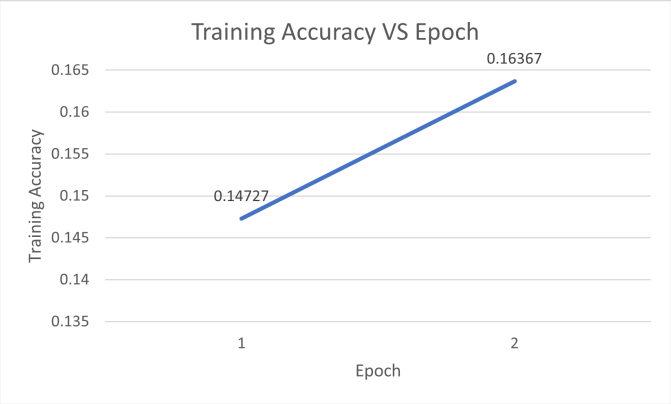

Figure 4: Trained with 2 epochs

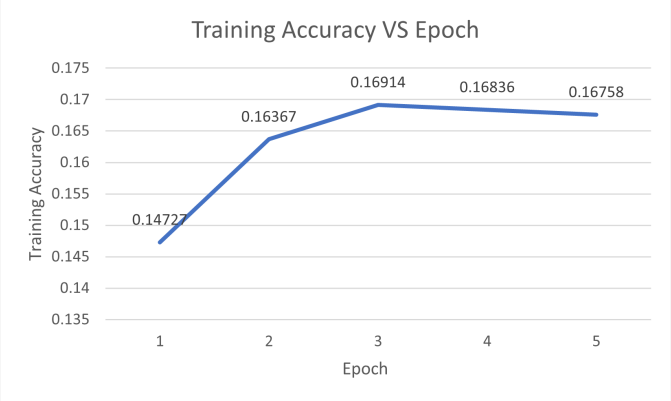

Figure 5: Trained with 5 epochs. Training accuracy did not increase after the third epoch

Although the same trend did not appear in validation accuracy, the change of slope also occurred on epoch 3. Hence, we decided to use epoch 3 as the optimal parameter.

We then decided to increase the number of layers of LSTM cell from 1 to 2, running with 3 epochs and a dropout rate of 0.1. Since increasing the number of layers of LSTM cell should allow more complex relationships, both accuracies should increase. However, the problem of WordPredict is simply predicting vectors based on two other vectors, the increase should be insignificant. We show the trends of training accuracy in Figure 7 and show the validation accuracy versus the number of layers in Figure 8.

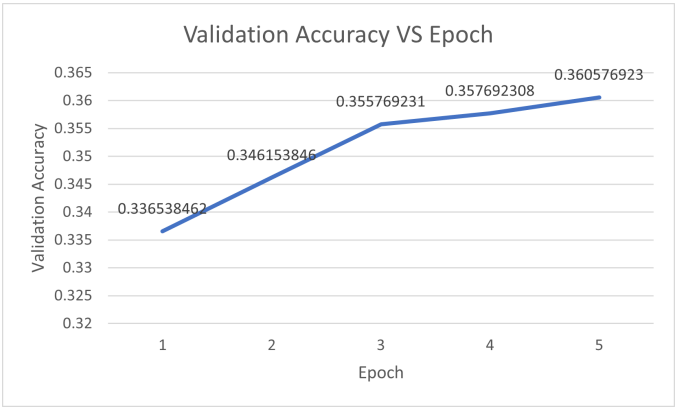

Figure 6: Trained with 5 epochs. Accuracy did increase.

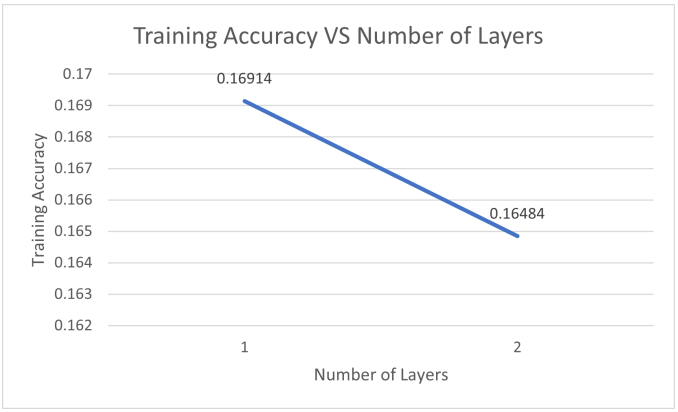

Figure 7: Accuracy decreased as number of layer increased

To our surprise, the training accuracy did not increase, either. However, the validation accuracy did increase, but only by 0.006. Since 2 layers of LSTM cell are harder to train, we decided to stick with 1 layer of LSTM cell.

Lastly, we decided to increase the dropout rate. Since increasing the dropout rate should make the model more robust, a decrease in the training accuracy should be observed. Indeed, the following trend is observed when the dropout rate increase from 0.0 to 0.1. We show the Traning accuracy versus Dropout rate in Figure 9 and versus validation accuracy in Figure 10.

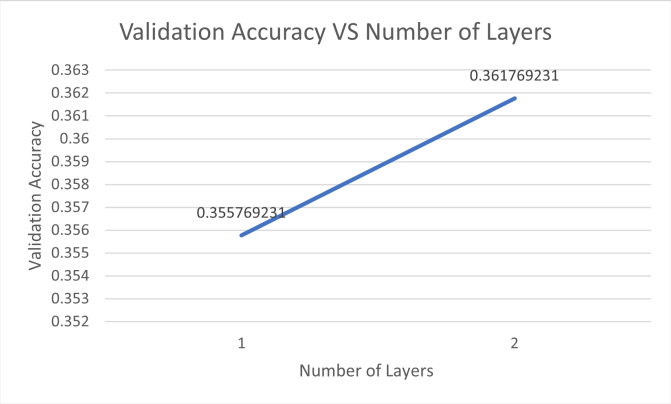

Figure 8: Accuracy increased as number of layer increased

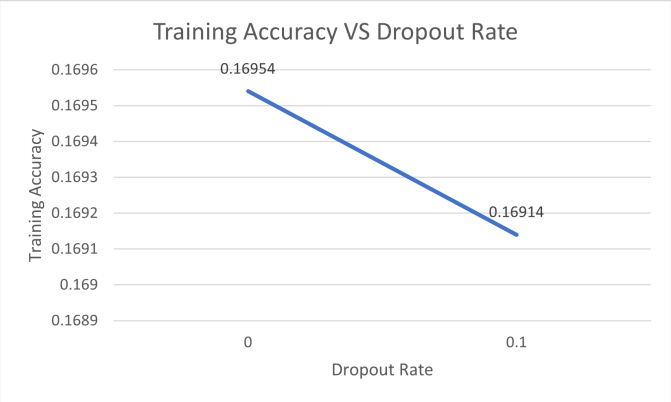

Figure 9: Trained with 5 epochs. Accuracy did increase.

As expected, both training and validation accuracy decreased as dropout rate increased from 0.0 to 0.1. Hence, we decided to use no dropout as the accuracy is higher.
As a result, we arrived at the following configuration that achieves a training accuracy of 0.16954. and validation accuracy of 0.3625:

epoches = 3
1 layer LSTM (hidden layer size = 800)
no dropout

Lesson Learned

On the path of manipulating the data set, we settled several targets together, and accomplishment of these tasks are proved to make our design of implementation easier and increase the performance of the model. The first one is to ensure that dataset we choose must fit our requirements at the maximum scale. We initially tried some open-source datasets like Google Audioset and VoxCeleb, but later

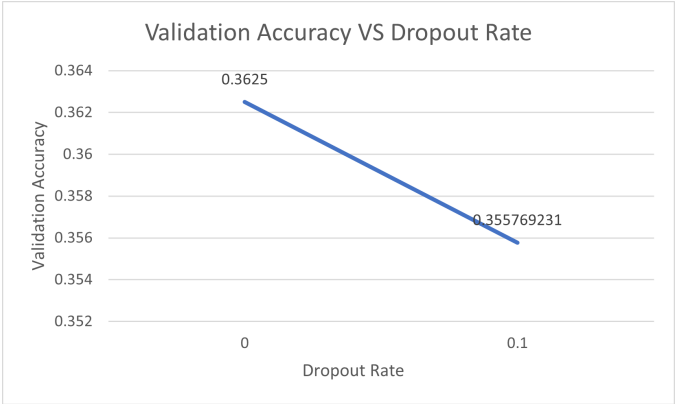

Figure 10: Trained with 5 epochs. Accuracy did increase.

we tuned to the Common Voice since that dataset provides more than 1000 hours of full sentences with transcripts, and all files really come from their user's contributions. It also turns out Common voice dataset was really helpful in our process of training models. The second target ensure that our modification of the data set does not destroy some hidden features. Initially we use trials to remove a silence a word in the audio file, but we found that DeepSpeech, the tool we plan to use in the later stage, can play a role in the modification stage. Following the highly accurate transcript generated by Deepspeech, we may locate the position of word we want to delete more precisely, and ensure that our data set does not contain ambitious cases(ambiguous means the case may not have a solution, and the model would just randomly approach to one.), where ambiguous ones would reduce the performance of our model.
Moreover, throughout this research activity, we improved our understanding of RNN by reading about it in the DeepSpeech paper and implementing one (WordPrediction) using TensorFlow. In the DeepSpeech paper, a 5-layer RNN is described. Each non-recurrent layer (except for the output layer) uses a clipped rectified linear activation function, which is helpful in reducing the vanishing gradient problem, while the output layer uses a softmax function which ensures that the resulting vector can be interpreted as a probability distribution. As for WordPrediction, we have settled on a single layer RNN that uses LSTM cell with 800 hidden features. The optimizer used is the Gradient Descent Optimizer. Also, we had a valuable experience in fine-tuning the model in attempt to improve the model's performance. By increasing the number of epochs and dropout rate, we have seen an increase in model's performance even with just a single layer RNN. However, increasing

epochs will have a significant increase in the training time because it requires the model to go through the entire training set multiple times, while increasing the dropout rate will decrease model's performance and may even lead to underfitting if overtuned.

We also gained some experiences about editing audio files. To our best knowledge, since there is no open source data set completely fulfill our requirements, we were in the situation of manipulating data sets since the very beginning. However, a tool in the field of editing is not familiar to any of us, since we all had no knowledge about it. We tried a lot of software and applications, but most of them only provides the functionality of cutting and merging files. In D1 and D2, our solution was manually determining and slicing the specific part of a word from the audio file based on the time interval that we can hear the word, and then replace that slice with the same length silenced recording file. This method did work but lack of efficiency and quality, since we may need to create dozens of training data. In D3, with effort, we discovered a fantastic command line tool, ffmpeg, which support various ways of modifying audio files and accept many formats of files. Based on ffmpeg, we created the tool of MuteOne and it contributes a lot of work to our project.

Discussion

Note that by using existing technologies and well-established models, we were able to fill one missing word in the sentence (which is transcribed from an audio) with accuracy that is slightly higher than randomly guessing among 3 choices of words, instead of choosing from a whole huge set of vocabularies. We believe that after good implementation on its wrapper and visualization, our project would be a good support, or additional helper, during chat in some online conference software. There are still directions that can be proceeded in the future. One may use more layers of LSTM and see if a better result can be achieved. Since our project assumes that there is at most one missing word in the sentence while the problem to our motivation does not have any constraint related to the neither number of missing words nor the number of sentences, this project only tackles a fraction of the whole. However, being able to fill only one missing word is a start to solve. Another limitation is that the training time maybe too long for fine-tuning the hyper-parameters. By giving more time and more chances to fine-tune, the model may achieve a better result. Finally, note that this project shows that it is possible to fill a missing word in the sentence without giving any word choices (although the predication process is slow due to the large vocabulary).

Conclusion

As COVID-19 stucks the globe, online conference meeting software have a spontaneous increase in usage. However, due to unstable internet and other causes, some of the audio may be incomplete and the users are often frustrated. We aim to solve the problem of incomplete audio fragments by converting them into text messages and filling the missing fragments (if any). This project tackles the problem above with the assumption that there is only one missing word if the sentence is incomplete. By using Deepspeech to transcribe the audio message and then use a 1 layer LSTM to process the transcribed message, we were able to achieve a validation accuracy of 0.3625.
For future directions, note that one may attempt to improve this accuracy by using more layers or more complex models than the RNN model described in this project. In our project, we only increase the layer from 1 to 2. However, it is possible that more layers may increase some favorable properties of the model. Besides, using different training model is also a possible path. For instance, one may attempt to use a different word embedding such as BERT instead of the current word embedding we use to see if the model achieves a higher accuracy. In future, there may be more kinds of embedding that can improve the performance of our project in different variations.

Finally, note that this model can be fine-tuned to complete sentences for only one specific person because everyone has an unique way to speak and has a tendency to use similar sentence structures and vocabulary. Thus, the model may achieve a higher accuracy in completing sentences for one specific person when trained against only his/her audio data. In this case, it may be possible to relax the one-missing-word limitation because everyone may have their own mantras/favorite phrases that can be easily predicted by the model.

References

Bian, J.; Gao, B.; and Liu, T.-Y. 2014. Knowledge-powered deep learning for word embedding. In *Joint European conference on machine learning and knowledge discovery in databases*, 132–148. Springer.

ffmpeg developers. 2020. ffmpeg documentation.

Garel, C. 2020. Bad internet in rural canada is making online school a nightmare.

Hannun, A.; Case, C.; Casper, J.; Catanzaro, B.; Diamos, G.; Elsen, E.; Prenger, R.; Satheesh, S.; Sengupta, S.; Coates, A.; et al. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Klakow, D., and Peters, J. 2002. Testing the correlation of word error rate and perplexity. *Speech Communication* 38(1-2):19–28.

Kuleshov, V.; Enam, S. Z.; and Ermon, S. 2017. Audio super resolution using neural networks. *arXiv preprint arXiv:1708.00853*.

Lifesize. 2019. Video conferencing statistics for 2019.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

2019. 2019 state of video conferencing report.

Purwins, H.; Li, B.; Virtanen, T.; Schlüter, J.; Chang, S.-Y.; and Sainath, T. 2019. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing* 13(2):206–219.

ReubenMorais. 2020. Release deepspech 0.9.1 · mozilla/deepspeech.

Zweig, G.; Platt, J. C.; Meek, C.; Burges, C. J.; Yessenalina, A.; and Liu, Q. 2012. Computational approaches to sentence completion. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 601–610.