Projet en traitement avancé du son - M2 ISI 2020/2021

Groupe 10, sujet 4.2 : Reconnaissance de la Parole

WANG Zhongyi - XUE Chaoyi - ZHANG Yuancheng

1 Presentation and Objective

This project aims to realize an End-to-End Speech Recognition task, which means to build a system that automatically convert audio data files to comprehensible text files. Our work is mainly based on previous research [1]. Instead of using traditional sequence-to-sequence translation, in this project, we prefer to realize a character by character based recognition. More precisely, We use the spectrum of the audio signals as features extracted. These features are then passed to a bidirectional LSTM network to train the network with CTC (Connexionist Temporal Classification) loss. In this report we will explain how it is done, including the principle and the implementation of the Recurrent Neural Network with loss CTC.

The database used during this project is the CSTR VCTK Corpus version 0.92 provided by the University of Edinburgh [2]. The speech data included in this database is in English and is recorded by 110 native speakers who come from different English-speaking countries with various accents. Along with all speech data in wave format, each sentence has a corresponding text file as its label. This database containing a large amount of data is well organized and has helped us a lot during the research process.

In the following sections, the presentation of this project will be divided into three parts. The first part is the pre-processing of data (please refer to the section 2). This part includes the calculation of spectrum of audio signals, the encoding of the text files, the adjustment of dimensions and other related operations. The second part focuses on building the architecture of the neural network (please refer to the section 3). For this part, we will explain the principle and the usage of an implemented CTC model to build our neural network. The third part is training and evaluation of the model built (please refer to the section 4). Finally, we will analyse and summarize the results obtained in these parts, before a brief conclusion.

2 First Step : Data Pre-Processing

2.1 Description and Objectives

Data pre-processing is usually a mandatory step when using machine learning to deal with real-world problems.

There are two types of data in our task, the audio files from the 110 English speakers and the text files of the corresponding sentences spoken. The main purpose of the pre-processing is to generate input and output matrix based on these raw data to meet the requirements of the architecture of our neural network model.

2.2 Experimental Configuration

In our project, we use the module of the spectrum instead of directly using the audio signals in the time domain as the input of our neural network. Hence, the task of pre-processing of these audio signals is the calculation of spectrums with the help of short-time Fourier transform. Besides, the text files are encoded letter by letter to facilitate the calculation of the loss function.

2.2.1 Input Generation

As mentioned in the previous section, in order to make the input data meets the needs of our model, we need to pre-process the raw data to calculate the spectrums of each audio signal and then save all the

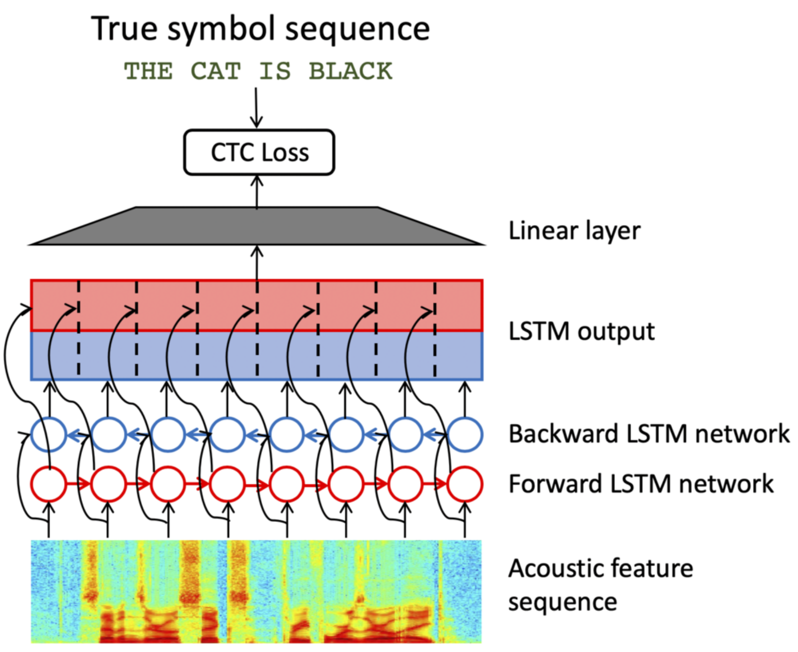


FIGURE 1 – Bidirectional LSTM CTC that unrolls over the whole speech utterance [3].

spectrum in one .npy file which could be loaded easily. The first thing we observe before the pre-processing is that in the database VCTK, the sentences spoken by different speakers were saved in different folders and each sentence was recorded by two microphones (left channel and right channel). For example, the first folder of the database VCTK contains 231 different sentences and 462 wave files spoken by a single person which means each sentence was recorded twice, probably with two microphones. To simplify our task, we decide to always use one of the two audio files for each sentence to calculate the spectrum. In this case, for the first folder, we have 231 spectrum data files and each of them corresponds to one audio signal. For each signal, we use the function melspectrogram from the module librosa to calculate the spectrum with 40 frequency bands. An example of spectrum is presented below.

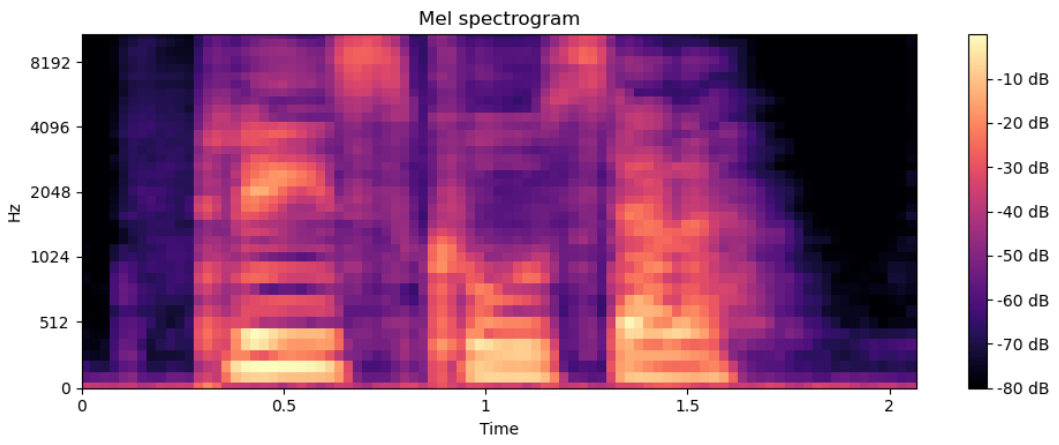


FIGURE 2 – Spectrogram for the "p225_001_mic1.flac" audio file in VCTK database, which corresponds to the sentence : “Please call Stella.” spoken by speaker NO.225 and recorded by the first microphone.

The length of the spectrum for each signal is not the same. Hence, we calculate first of all the spectrum of each signal and save them in a list. Then, we traverse the list to get the length of the spectrum corresponding to each signal and the maximum length of all the spectrums. Finally, padding all the spectrums to the same length, we reallocate a zero matrix with dimensions [number of sentences * 40 * the maximum length] and fill it with the previously calculated spectrums. After calculating the matrix of spectrums for all the folders and saving them as .npy files, we then write a function to re-joint the files needed to construct the training set and the testing set of our model.

2.2.2 Output Labels Generation

After pre-processing the input of our neural network, this sub-section is going to present the generation of the corresponding output.

Since each audio file contains a sentence recorded by one of the speakers, and has a corresponding text file giving the answer to what the speaker says. Thus, the goal is to encode the text to numerical values, which is going to be the labels of our model.

The idea is to firstly create a dictionary of all the characters, and then use this dictionary to encode each text. The dictionary contains one "space" character and all the 26 letters of alphabet (ignoring the capital or small letter issue) :

```
dictionary = {
               ' ': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4,'e': 5, 'f': 6, 'g': 7,
               'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14,
               'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21,
               'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26
             }
```

As shown above, the characters are encoded into 27 integers, from "0" to "26".

Thus, for example, before encoding, the text might be like :

```
Please call Stella.
```

And, after encoding, a row vector is generated, with each element correspond to a character in the text in order. The encoded version of the above text becomes a row vector :

```
[16  12   5   1  19   5   0   3   1  12  12   0  19  20   5  12  12   1]
  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
  P   l   e   a   s   e       c   a   l   l       S   t   e   l   l   a.
```

You may notice that there's a dot at the end of the sentence, but it doesn't appear afterwards. It is because that we have set in the algorithm to remove these punctuation marks like dot, comma, quotation mark, question mark, etc. when reading the text files in our program. This aims to reduce the influence of these special characters in the model. However, it can be one of our future steps after this project, to do more research on the speech recognition with automatic punctuation.

So far, we illustrate the principle of converting a single text to a row vector. Next, we do the same thing to all the text files in one folder and add the generated vectors into a matrix. Here comes a problem, the "matrix" here is not yet a real matrix, it is simply a stack of vectors with different length due to the texts in different size. To deal with this issue, we traverse every row of the "matrix" and find the row vector that has the maximum length. Then traverse every row again and pad the current row vector with zeros if the length of it is smaller than the maximum value. An example will be presented (please refer to 2.3) to help explain this.

2.3 Results Obtained

For the input generation :

Following the steps of the input pre-processing, including reading all the audio files in different folders and transcripting them into matrix of spectrums which are padded to the same length on second dimension. Finally, by combining all these matrix together, we obtain a data-set that can be directly loaded to a deep learning model.

For the output labels generation :

After the pre-processing described previously, we successfully read all the text files in one folder and convert them into one matrix with each row representing the encoded numerical values of corresponding text. To help explain what we've done here, an example is presented below.

Example :

Consider one folder contains only three text files and the texts included are :

```
Please call Stella.
He is in the queue now.
How are you, sir?
```

Then, after our pre-processing algorithms, we obtain a matrix representing all contents included in this folder :

```
[[16 12  5  1 19  5  0  3  1 12 12  0 19 20  5 12 12  1  0  0  0  0  0]
 [19  8  5  0  9 19  0  9 14  0 20  8  5  0 17 21  5 21  5  0 14 15 23]
 [ 8 15 23  0  1 18  5  0 25 15 21  0 19  9 18  0  0  0  0  0  0  0  0]]
```

As you can see, the second sentence is the longest, so it results in other row vectors with several zeroes at the end. This helps the generated matrix be homogeneous on the second dimension to meet the requirement of a keras model.

2.4 Discussion

For the output labels generation :

During the process of encoding the texts to vectors of indexes, as illustrated previously, we use a one-dimensional encoder, with indexes vary from "0" to "26" representing the characters encoded. There exists other encoding method like one-hot-encoder, this method has its own pros and cons, such as, it eliminates the hierarchy/order issues but does have the downside of adding more columns to the data set. Considering the amount of data waiting for processing, and the implemented CTC Model (which will be presented in next section) does not fit well with one-hot encoder, we then choose the previous approach, but the one-hot-encoder is also worth a try in the future.

Furthermore, we ignore the punctuation marks here in this project. However, it could be studied in our future work since using punctuation in speech recognition will improve the readability and usability of transcripts. Some punctuation marks like comma, question mark and exclamation mark are related to pauses and mood of the speech and speaker, according to [4]. More research on these factors will be needed afterwards.

For the whole advance preparation :

After the prepossessing, we finally generate four .npy files to be used after when training our model. The first .npy file named such as X_train.npy contains the spectrums of our audio signals. This file has three dimensions : the number of examples * the number of frequency bands(40 in our case) * the maximum length of all the spectrums. The second npy file named such as Y_train.npy contains the numeric labels corresponding to each sentence. This file has two dimensions : the number of examples * the maximum length of all the sentences. The third npy file named such as length_list_spectrum.npy contains the length of spectrum of each sentence in the database with one dimension : the number of examples. The last file named such as length_labels.npy contains the length of text of each sentence with the same dimension of the third one. These four files will then be used in our training process which will be explained in the following sections.

3 Second Step : Principle and Usage of the CTC Model

Considering speech recognition, We have a data set of recorded audio files and corresponding text transcripts. The difficulty is, we cannot align the characters in the transcript to the audio. This makes training a speech recognizer harder than it might at first seem. Thus, if we ignore the alignment issue and proceed to training directly, the model will hardly converge due to the different speech rate of people.

Connectionist Temporal Classification (CTC) is an approach to avoid the manual alignment between the input and the output. As we'll see, this algorithm is especially well suited to applications handwriting recognition and speech recognition.

More specifically in our project, we use an open-source CTC Model implemented by [5]. CTC Model makes the training of a RNN with the Connectionnist Temporal Classification approach completely transparent, according to [5].

3.1 Principle of the CTC model

This subsection aims to give a brief presentation of the CTC model. For more details please refer to [6], which provides a deeper explanation on each steps of the model.

3.1.1 Alignment

First thing to notice is that the CTC algorithm doesn't require an alignment between the input and the output. However, in order to compute the loss, we need to know the correspondence between the output of X and the label Y.

For example, in an OCR - Optical Character Recognition task, the input X is an image including word "cat", and the label Y is [c, a, t]. We divide X into several time-slices or time-steps, and we obtain an output (a character) from each time-slice, as shown in figure 3.

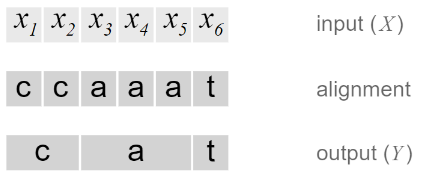


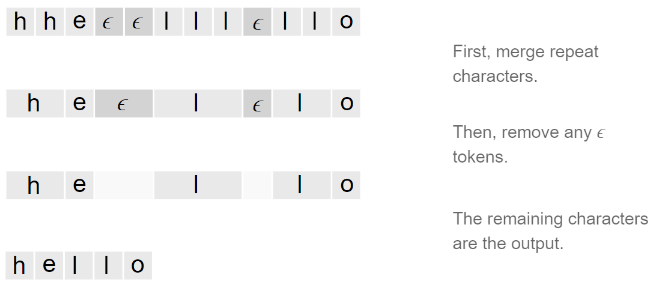
FIGURE 3 – A naive approach to align input to output [6].

This approach has two problems. First of all, it does not make sense to force every input step to align to some output. In speech recognition, for example, the input can have stretches of silence with no corresponding output.

Secondly, we have no way to produce outputs with multiple characters in a row. Consider the alignment [h, h, e, l, l, l, o]. Collapsing repeats will produce "helo" instead of "hello".

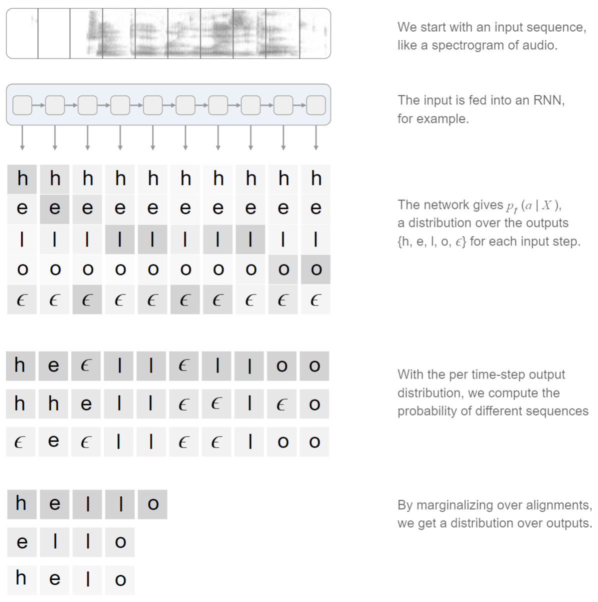
To deal with these problems, CTC introduces a new token to the set of allowed outputs. This new token is sometimes called the blank token. We'll refer to it here as ϵ The ϵ token doesn't correspond to anything and is simply removed from the output.

The alignments allowed by CTC are the same length as the input. We allow any alignment which maps to YY after merging repeats and removing ϵ tokens :



h h e ϵ ϵ l l l ϵ l l o
First, merge repeat characters.
h e ϵ l ϵ l o
Then, remove any ϵ tokens.
h e l l o
The remaining characters are the output.
h e l l o

FIGURE 4 – Improved alignment method used in CTC algorithm [6].



We start with an input sequence, like a spectrogram of audio.
The input is fed into an RNN, for example.
The network gives $p_t(a \mid X)$, a distribution over the outputs {h, e, l, o, ϵ} for each input step.
With the per time-step output distribution, we compute the probability of different sequences
By marginalizing over alignments, we get a distribution over outputs.

FIGURE 5 – Visualization of the general process of CTC Model [6].

3.1.2 Loss function

The mapping relation between each time-step and an output sequence is shown in the figure 5.

For label Y, the posterior probability given X can be expressed by the sum of all "paths" leading to Y in mapping. And thus, our goal is to maximize the probability P(Y|X). Supposing the outputs of each time-step are independent to each other, then the wanted probability can be expressed by the sum of all probabilities of each time-step. Please refer to the explanation below.

$$p(Y \mid X) \quad = \quad \sum_{A \in \mathcal{A}_{X,Y}} \quad \prod_{t=1}^{T} p_t(a_t \mid X)$$

The CTC conditional **probability**

marginalizes over the set of valid alignments

computing the **probability** for a single alignment step-by-step.

3.1.3 Inference

After training the model, the final step is to use it to find a likely output for a given input, which is called prediction. More precisely, we need to solve :

$$Y^* \quad = \quad \underset{Y}{\operatorname{argmax}} \; p(Y \mid X)$$

Greedy Search :

According to [6], one heuristic is to take the most likely output at each time-step. This gives us the alignment with the highest probability :

$$A^* \quad = \quad \underset{A}{\operatorname{argmax}} \; \prod_{t=1}^{T} p_t(a_t \mid X)$$

The drawback of this method is that it ignores one possible situation : a single output can have many alignments. We can use a modified beam search to solve this problem.

Beam Search :

Given limited computation, the modified beam search won't necessarily find the most likely Y.Y. It does, at least, have the nice property that we can trade-off more computation (a larger beam-size) for an asymptotically better solution.

A regular beam search computes a new set of hypotheses at each input step. The new set of hypotheses is generated from the previous set by extending each hypothesis with all possible output characters and keeping only the top candidates.

3.2 Usage of an implemented CTC model

Before the implementation of our own model, we first test an example code of the CTCModel [5]. The data set is composed of sequences of digits. The images and the digits corresponding to them from the MNIST data set have been concatenated to get observation sequences and label sequences. The example then create two lists per data set containing the length of each sequence, one list for the observations and one for the labels. Then data are padded in order to provide inputs of fixed-size to the Keras methods. As mentioned before, the example code used digits in the database MNIST as a sequence of labels. After ten epochs of training, the results are as follows :

```
[ 3.  3.  1.  4. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  -1.]
Prediction : [3.0, 3.0, 1.0, 4.0] -- Label :  [3. 3. 1. 4.]

[ 1.  1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  -1.]
Prediction : [1.0, 1.0] -- Label :  [1. 1.]
```

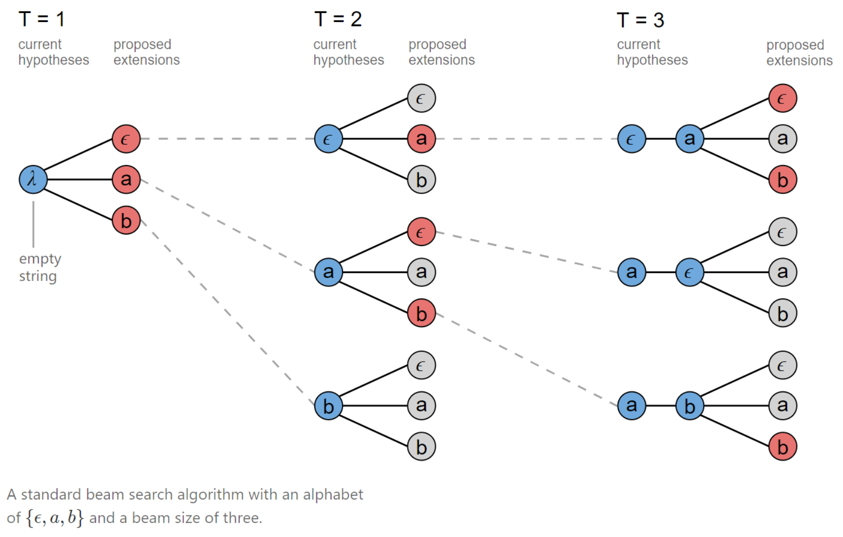


FIGURE 6 – Visualization of a standard beam search algorithm [6]

In the example code, The prediction given by the model of one sequence composed of several images which correspond to a sequence of label of digits is perfect. Hence, in our case of development, we have reason to implement a model which have a sequence of features of audio signals as input and a sequence of characters as output.

4 Third Step : Training and Results

4.1 Description and objectives

Once we have prepared the data to be used and chosen the models, we pass to the training and the evaluation process of the models.

4.2 Experimental Configuration

Compared to other deep learning models based on Keras, the fit function of the open source CTC model is also overridden to have its specific input X. In this case, the input X is not only the features which often used as input tensor but contains four parts : the input observation, the labels, the lengths of the input sequences and the lengths of the label sequences. These four parts correspond exactly to the four npy file generated during the preparation. Besides, the tensor y is also not used in a standard way and must be defined for Keras methods as the labels or an empty structure of length equal to the length of labels.

4.3 Results Obtained

During our research, we mainly train two architectures of models and try to compare their performances.

The first model is the one proposed in the article [1] which has five levels of bidirectional LSTM hidden layers, with 500 cells in each layer, giving a total of 26.5M weights. We train this model by using stochastic gradient descent with one weight update per utterance, a learning rate of 10^{-4} and a momentum of 0.9. The first model is trained on a amount of data close to 10000 sentences. The performance of the first model is shown in the following figure.

Obviously, the validation loss drops quickly after a few epochs and already converges towards 110 after 10 epochs so we could say that the model is under-fitting and we could not have better performance even after more epochs of training. We try several methods for example adjusting manually the optimizer of the model and the learning rate but they don't make the performance of this model better. From our perspective, the problem may be caused by the lack of data compared to the large amount of parameters, which makes the convergence of the model becomes more difficult. Hence, we decide to simplify our model to reduce the amount of parameters to be trained and increase the amount of data.

To solve the problem, we propose a second model simplified which contains three levels of bidirectional LSTM hidden layers, with 128 cells in each layer, giving a total of 968K parameters trainable, for each

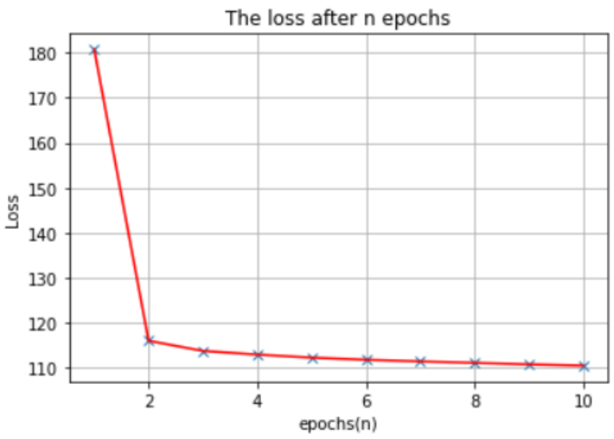


FIGURE 7 – Visualization of the validation loss with respect to the epochs of training .

layer a drop out of 0.1 is added. It is trained on 10000 sentences and 20000 sentences using Adam optimizer with a learning rate of 10^{-4}. The summary of this model is shown in the following figure :

```
________________________________________________________________________________________________________
Layer (type)                       Output Shape          Param #      Connected to
========================================================================================================
input (InputLayer)                 [(None, 609, 40)]     0
________________________________________________________________________________________________________
masking (Masking)                  (None, 609, 40)       0            input[0][0]
________________________________________________________________________________________________________
bidirectional (Bidirectional)      (None, 609, 256)      173056       masking[0][0]
________________________________________________________________________________________________________
bidirectional_1 (Bidirectional)    (None, 609, 256)      394240       bidirectional[0][0]
________________________________________________________________________________________________________
bidirectional_2 (Bidirectional)    (None, 609, 256)      394240       bidirectional_1[0][0]
________________________________________________________________________________________________________
time_distributed (TimeDistribut    (None, 609, 28)       7196         bidirectional_2[0][0]
________________________________________________________________________________________________________
softmax (Activation)               (None, 609, 28)       0            time_distributed[0][0]
________________________________________________________________________________________________________
labels (InputLayer)                [(None, None)]        0
________________________________________________________________________________________________________
input_length (InputLayer)          [(None, 1)]           0
________________________________________________________________________________________________________
label_length (InputLayer)          [(None, 1)]           0
________________________________________________________________________________________________________
CTCloss (Lambda)                   (None, 1)             0            softmax[0][0]
                                                                      labels[0][0]
                                                                      input_length[0][0]
                                                                      label_length[0][0]
========================================================================================================
Total params: 968,732
Trainable params: 968,732
Non-trainable params: 0
________________________________________________________________________________________________________
```

We train this model using 10000 sentences and 20000 sentences respectively for 300 epochs with a validation split of 0.1 and monitor the validation loss after each epoch. Compared to the previous five hidden layer model, the validation loss decreases this time to 59 with 10000 sentences trained and decreases to 47 with 20000 sentences trained. Hence, we decide to use 20000 sentences. The validation loss after each epoch of the model trained on 20000 sentences is shown in the following figure.

We then use the trained model to predict the test set, below is an example of the prediction. As shown, the accuracy is not that ideal since we are doing speech recognition task character by character using the CTC model instead of sequence to sequence, it means that a single mistake in the prediction could result in the incomprehension of a word or even the whole sentence for human being.

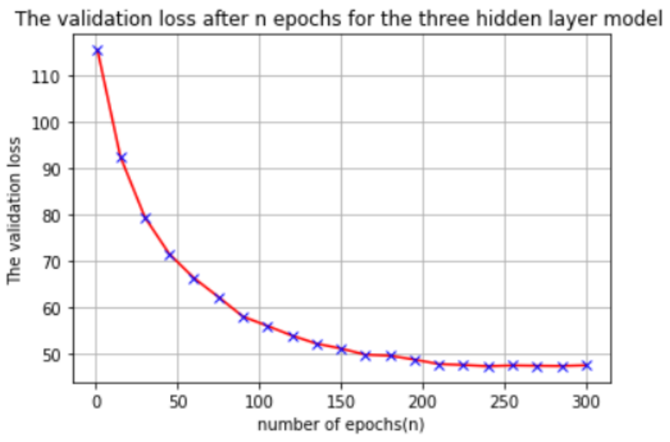


FIGURE 8 – Visualization of the validation loss with respect to the epochs of training for the model with three hidden layers.

```
Prediction : [23.  5.  0. 16. 18. 15.  4.  0.  1.  0.  2.  9. 20.
               5.  0.  9. 14.  0.  1. 20.  0. 14.  9.  7.  8. 20.]
——— Label : [23.  5.  0. 16. 21. 20.  0. 15. 21. 18.  0.  2.  9.
               4.  0.  9. 14.  0. 12.  1. 19. 20.  0. 14.  9.  7.
               8. 20.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ...]
Decoded prediction text: "we prod a bite in at night".
——— Decoded label text: "we put our bid in last night".
```

Another example of prediction :

```
Decoded prediction text: "our has was very smo".
——— Decoded label text: "our house was very small".
```

5 Evaluation of Results

If we look at the predictions made by our trained model. Although the result presented seems not good enough, there's still something interesting that draws our attention. In the prediction, our model successfully transcripts the words "we", "in" and "night". Other words that are wrongly predicted are "prod" against "put", "a" against "our", "bite" against "bid" and "at" against "last". By summarizing these mistakes, as well as in other examples, it's not hard to find out that the pronunciation of these wrong words are pretty close to the supposed ones. Thus, the result might be reasonable considering that we only use limited data for training. We can expect a better result by increasing the size of data to further reduce the validation loss.

6 Conclusions and Future Work

In this project, we realize an End-to-End Speech Recognition task through building and training a Recurrent Neural Network with the help of an implemented CTC Model. From the results obtained, our task can be considered as partially or mostly successful. Since the mission itself is different from other classification or recognition tasks, its nature decides that the speech recognition and text transcription require a higher level of accuracy on the results.

Table 1 concludes all the network models that we have tried throughout the realization of our project and their corresponding information such as the loss on validation set after 300 epochs.

Limited by the time consummation on the training of our model, and the upper limit of allocation of the data to our GPU, we are not able to use more examples for training. However, by comparing the computed loss in the table, we can infer that a better performance could be expected by increasing the amount of data.

On the other side, we also identify other possibilities to go deeper as an extension to this project. Such as the influence of different encoders. In this project, when dealing with the labels, a one-dimensional encoder is used because it's the most intuitive way to encode a text. Although involving one-hot encoder will add more columns to the data set, it is still a more reasonable way since it brings equality to the hierarchy and order of each character. Thus, we might try one-hot-encoder and see it's influence on the

The model used	the optimizer	the size of the training data	loss on validation set
model with five hidden Bidirectional LSTM layers	SGD lr : 1e-4	10000 examples	110
model with three hidden Bidirectional LSTM layers	Adam lr : 1e-4	10000 examples	59.18
model with three hidden Bidirectional LSTM layers	Adam lr : 1e-4	20000 examples	47.38

TABLE 1 – Conclusion of all training models used

results in the future. Moreover, finding a way to generate or predict the punctuation marks in a text is also an interesting extension, and this may be connected with with the recognition of speaker's mood.

Annexes

Example of code

Here is an abstract of the code used for this project. Please refer to our github page for more detail.

```
# Define a recurrent neural network using CTCModel
# Define the network architecture
input_shape = x_train[0].shape
input_data = Input(name='input', shape=(input_shape))
masking = Masking(mask_value=padding_value)(input_data)
blstm = Bidirectional(LSTM(128, return_sequences=True, dropout=0.1))(masking)
blstm = Bidirectional(LSTM(128, return_sequences=True, dropout=0.1))(blstm)
blstm = Bidirectional(LSTM(128, return_sequences=True, dropout=0.1))(blstm)
dense = TimeDistributed(Dense(nb_labels+1, name="dense"))(blstm)
outrnn = Activation('softmax', name='softmax')(dense)
network = CTCModel([input_data], [outrnn])

network.compile(Adam(lr=0.0001, decay=1e-6))
network.summary()

history = network.fit(x=[x_train, y_train, x_train_len, y_train_len], y=np.zeros(nb_train),
          batch_size=batch_size, epochs=nb_epochs, validation_split=0.1)

# Evaluation: loss, label error rate and sequence error rate are requested
eval = network.evaluate(x=[x_test, y_test, x_test_len, y_test_len], batch_size=
       batch_size, metrics=['loss', 'ler', 'ser'])

# Predict label sequences
pred = network.predict([x_test, x_test_len], batch_size=batch_size, max_value=
       padding_value)

# Print the 10 first predictions
for i in range(10):
   print("Prediction :", [j for j in pred[i] if j != -1], " -- Label : ", y_test[i])
```

Références

[1] Ying Zhang, Mohammad Pezeshki, Philémon Brakel, Saizheng Zhang, Cesar Laurent Yoshua Bengio, and Aaron Courville. Towards end-to-end speech recognition with deep convolutional neural networks. *arXiv preprint arXiv :1701.02720*, 2017.

[2] Christophe Veaux, Junichi Yamagishi, Kirsten MacDonald, et al. Superseded-cstr vctk corpus : English multi-speaker corpus for cstr voice cloning toolkit. 2016.

[3] Kurata Gakuto, Audhkhasi Kartik, and Kingsbury Brian. Ibm research advances in end-to-end speech recognition at interspeech. 2019.

[4] Piotr Żelasko, Piotr Szymański, Jan Mizgajski, Adrian Szymczak, Yishay Carmiel, and Najim Dehak. Punctuation prediction model for conversational speech. *arXiv preprint arXiv :1807.00543*, 2018.

[5] Yann Soullard, Cyprien Ruffino, and Thierry Paquet. CTCModel : Connectionist Temporal Classification in Keras, 2018.

[6] Awni Hannun. Sequence modeling with ctc. *Distill*, 2017. https ://distill.pub/2017/ctc.