

## Contents

<b>1</b>	<b>Setting the environment up</b>	<b>1</b>
<b>2</b>	<b>ESPNet in Bash</b>	<b>3</b>
2.1	Directory tree and KALDI recipes . . . . .	3
2.2	Data . . . . .	4
2.3	Training the neural networks. . . . .	5
2.4	Recognition and evaluation . . . . .	5
2.5	Checking the evaluation results . . . . .	6

## Introduction

In this assignment, we will present the Automatic Speech Recognition (ASR) task. The main goal of this task is to assign a sequence of words, letters or phonemes to a given input, which is usually audio features. In particular, we will use an already **built** recognition system based in End-to-end deep learning. We will use *ESPNet (End-to-end Speech Processing Toolkit)*. The code that we will use was provided by [Beltrán Labrador](#) and [Doroteo Torre](#), and can be found in this [Google Colab notebook](#).

## Setting the environment up

To test ESPNet, we will use the dataset TIMIT, which contains 6300 phrases, divided in 10 phrases of each of the 630 speakers from the 8 main different US dialect regions. The phrases are also divided in male and female speakers. It is also important to remark that the division in train/test splits is done in a way that:

- A speaker appears rather in the train or in the test split separately, but not in both at the same time.
- The phrases of the test set are **not** in the train set.

The first task will simply consists on looking at the content of the files that the TIMIT corpus provides and understanding these files.

### Questions.

- Check the content of the *.txt*, *.wrđ* and *.phn* files listening to the audio (*.wav* file) with a few examples. Comment how it was done and the results obtained.

To check the content of the files, we can use `bash` commands. After loading the dataset in Google Colab, we store it in the `content/timit/TIMIT` folder, in which we can find the `TRAIN/TEST` folders. Having a look at the [official documentation](#) and to the assignment documentation, we observe that we have the folders have the following structure:

```
<DIALECT>/<SEX><SPEAKER_ID>/<SENTENCE_ID>.<FILE_TYPE>
```

We can list the different users of any of the regions and select one of them to observe his or her data. For convenience, we firstly use the code provided to examine the data of the user `DR1/FCJF0`, meaning that it is an user from the first dialect region, and it is a female. We select the audio `SX397` and then we show the *spelling* of the phrase.

```
!cat /content/timit/TIMIT/TRAIN/DR1/FCJF0/SX397.TXT
```

```
0 39220 Tim takes Sheila to see movies twice a week.
```

We appreciate that this audio consist uniquely on one phrase. If we use

```
!python.display.audio('path_to_the_audio')
```

(where the path in this case is `/content/timit/TIMIT/TRAIN/DR1/FCJFo/SX397.WAV`), we listen to the audio and verify that the spelling is correct. We can also check the *aligned word-level transcription*:

```
!cat /content/timit/TIMIT/TRAIN/DR1/FCJFo/SX397.WRD

2240 5540 tim
5540 8610 takes
8610 14707 sheila
14707 15791 to
15791 19735 see
19735 26402 movies
26402 31210 twice
31210 31791 a
31791 37180 week
```

Which indicates exactly the steps of time where the words happen, and we can lastly observe the phonetic transcription:

```
!cat /content/timit/TIMIT/TRAIN/DR1/FCJFo/SX397.PHN

0 2240 h#
2240 2940 t
2940 4469 ih
4469 5540 m
5540 5860 tcl
5860 6570 t
6570 8211 ey
8211 8610 kcl
8610 11112 sh
...
```

Since this is the provided example, we repeat the process with a different example to check that everything is correct in another case. We use the same region and female speaker. We select the sentence id: SX307. We show the spelling transcription and the word-level transcription:

```
!cat /content/timit/TIMIT/TRAIN/DR1/FCJFo/SX307.TXT
!echo "-----"
!cat /content/timit/TIMIT/TRAIN/DR1/FCJFo/SX307.WRD

0 23143 The meeting is now adjourned.
-----
1960 2616 the
2616 8293 meeting
8293 10160 is
10960 13707 now
13707 20887 adjourned
```

Which shows the content of the phrase. Lastly, we have played the audio as we have explained before to check that the phrase is correct. We detect a little bit of noise on the speaker, but the words can be understood correctly.

- Explore the documentation of the TIMIT corpus and find information associated to the speakers. Can you think of any other applications of this corpus further from the training and evaluation of Speech Recognition systems?

We consider the folder structure mentioned before to give an answer to this. we find that we have audios classified by sex and *speaker*. Thanks to this, we could definitely use this corpus to build a *male/female* recognition system or even a **speaker recognition system**. However, the available data for this kind of tasks would probably too low since this dataset is not focused on this task.

Also, we know that we have both the **aligned word transcription** and the **aligned phonetic transcription** available, which could help us to create a **event detection system** with the capability of detecting the

event: *Speech*. However, it would be this unique event since it is the only one labeled in this dataset, so it would not be very useful, although this dataset could be joined with other events datasets to improve the event detection system.

Lastly, in [this file of the documentation](#), we observe that we can find more information about the speakers such as the race, birth date or education level. All this information could be use to perform different classification tasks, as we mentioned before with the sex. The birth date could be used for a regression problem using audio.

## ESPNet in Bash

In this section, we will use ESPNet in Bash to evaluate a STT system using end-to-end neural networks. We will use the previously presented corpus. The structure of stages that will have to be followed in order to train the system is presented in Figure 1

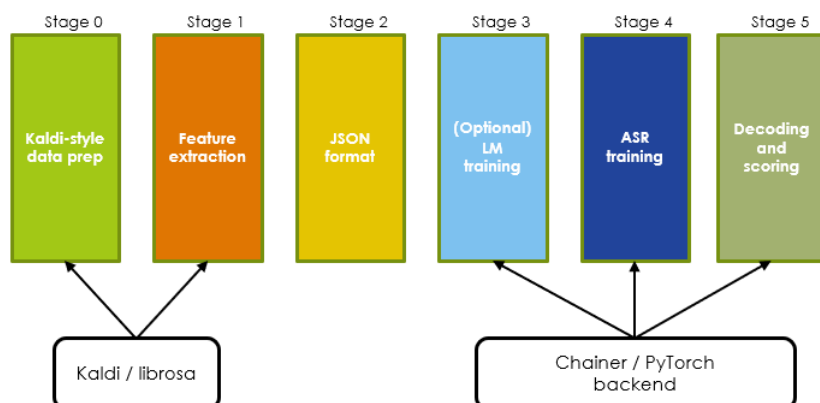


Figure 1: Stages of the complete recipe.

### 2.1 Directory tree and KALDI recipes

- Which file(s) should be modified to change...:

- The DNN backbone: The DNN backbone is specified in the file:

```
espnet/egs/timit/asr1/conf/train.yaml
```

If we explore this file, we can see that we can specify the *encoder* architecture, the *decoder* architecture and also the attention related layers and hybrid CTC/attention parameters.

- The Seq-to-seq mapping used: In the scrip

```
espnet/egs/timit/asr1/run.sh
```

We can find the parameter `trans_type` which indicates if we want to do the Seq-to-seq mapping in the character level or in the phoneme level. This is the parameter that we should modify.

- Comparing the obtained results in phoneme recognition and character recognition in the TIMIT corpus, which one do you consider more precise?

The results are shown in a Markdown file. In this file, the number of files, words, the percentage of correct tokens, percentage of substitutions/deletions/insertions and the error rate are shown.

For each of the cases (phoneme/character) a few different configurations of encoder, decoder and attention type are displayed. We resume the most most relevant information in Table 1.

Transcription type	Encoder	Attention type	Decoder	Accuracy on tokens	Error on tokens
Phonemes	vggbgrup	coverage_location	gru	82.1	21.4
	<b>bgrup</b>	<b>coverage_location</b>	<b>lstm</b>	<b>82.7</b>	<b>20.2</b>
	bgrup	location	gru	82.5	20.5
Characters	vggbgrup	location	lstm	69.6	38.2
	vggbgrup	location2d	gru	72.0	34.0
	bgrup	coverage_location	lstm	72.8	33.7

Table 1: Results on Phonemes and Characters recognition.

The results show that, in general, this system is **more capable of recognizing the phonemes** than the characters itself. There is a difference of about 10% in accuracy and more than 15% in the error rates. Recall that the error rates do not have to sum 100%, since the error percentage can be higher than 100%. In particular, in the shown cases, we obtain that the best performing system for phoneme recognition uses **bgrup** as the encoder, an **lstm** as decoder, and uses **coverage\_location** as the attention type, obtaining an accuracy of 82.7% on tokens and an error of 20%.

## 2.2 Data

In this section we will prepare the data for the training of the system.

### Questions.

- What are the names of the three subsets of *TIMIT* that are used? How many phrases does each of them have?

The subsets and the number of phrases are summarized in Table 2.

Subset name	Number of phrases	Mean duration	Min Duration	Max duration
train_wav	3696	3.063	0.915	7.788
dev_wav	400	3.08	1.09	7.43
test_wav	192	3.03	1.30	6.21

Table 2: Information about the used TIMIT subsets.

- What are the "FBANK" features?

**FBANK** (Mel-filter bank coefficients) features are extracted from the raw audios. They can be compared with the known MFCCs (Mel Frequency Cepstral Coefficients), which are obtained using the FBANKS and the DCT. This is shown in [1]. The difference in the process can be observed in Figure 2.

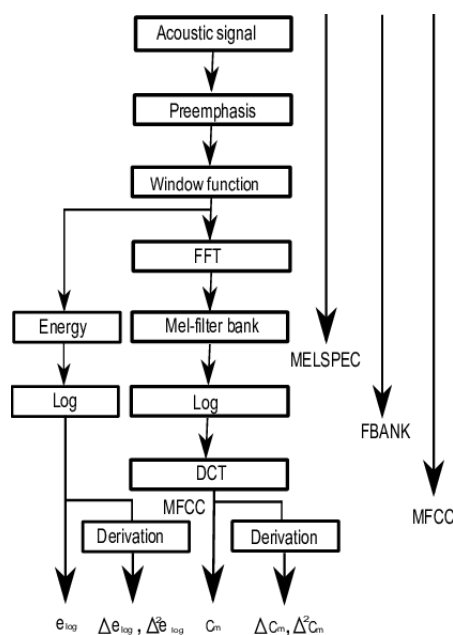


Figure 2: Obtention of FBANK and MFCCs.

- What does the acronym "CMVN" shown in the computational logs mean?  
We have found in [Wikipedia](#) that this acronym stands for **Cepstral Mean and Variance Normalization**, which is a computationally efficient normalization used in robust speech recognition. This technique minimizes distortion by noise contamination, leading to **robust feature extraction**, linearly transforming the Cepstral coefficients.

## 2.3 Training the neural networks.

It is time to train the model and check its performance.

### Questions.

Examine the file `train.yalm` and try to identify which parameter should be modified in order to change:

- Number of encoder layers: This is specified in the variable `elayers`.
- Encoder type: This parameter is stored in the variable `etype`
- Number of units per encoder layer: It can be found in the variable `eunits`.
- Encoder attention type: The variable `atype` indicates this.
- Number of units in encoder: This is stored in the variable `dunits`.

The variable names are self-explanatory and the file `train.yalm` is properly commented so finding the answers was easy. Usually, variables related to the encoder start with `e`, the attention variables start with `a` and the decoder variables begin with `d`, making the script intuitive.

**Tensorboard** can be used to graphically analyze the evolution of the training.

### Question.

- Can you identify the parameters that correspond to the CTC (Connectionist temporal classification) and attention?
- Check the evolution of the different parameters and the evolution in train and validation. Do you consider that the training has worked correctly?

ESPNet also generates charts about the training process.

### Questions.

- Can you observe any difference between the evolutions associated to CTC and attention? Why?
- What is the final approximated PER in train and validation? Why are the results different?

## 2.4 Recognition and evaluation

In this section, we will examine the file `decode.yalm` and answer some questions about it.

### Questions.

- What does the parameter `beam-size` do?

The parameter `beam-size` controls the number of hypotheses kept during the Beam Search (found in the [documentation](#) of the `beam search` implementation). **Beam search** is a heuristic search algorithm that explores a graph exploring the most promising node in a limited set. In this sense, it is a *greedy* algorithm, reducing the memory requirements. It uses breadth-first search to build its search tree.

- What does the parameter `ctc-weight` do?

Lastly, we can execute the recognition script.

**Question:** Why do two different results appear? What do they mean? Do they behave as expected?

If we have a look at the output of the command that executes the ASR decoding, we find two different lines right before the results:

```
write a CER (or TER) result in exp/train_nodev_pytorch_train/decode_test_decode/result.txt
---
write a CER (or TER) result in exp/train_nodev_pytorch_train/decode_train_dev_decode/result.txt
```

As it is indicated, the first one is writing on the file the error in the test partition, while the second one is writing the result on train and dev partitions. We can say that the results behave **as expected**, since the error in the test partition is a little bit higher than the error in the train set, as it usually happens.

## 2.5 Checking the evaluation results

Lastly we show that the results can be shown in detail. Let us look at the output of an specific example (the result is tiny, it can be zoomed):

```
!tail -n 7 espnet/egs/timit/asr1/exp/train_nodev_pytorch_train/decode_test_decode/result.txt
---
id: (mvev0-mvev0_sx371)
Scores: (#C #S #D #I) 32 8 0 0
REF:  <space> r AY <space> n AW m EY n AA <space> b iy DH ih <space> b EH s <space> t ay M f er <space> b ih z n ih s <space> m er <space> jh ER z <space>
HYP:  <space> r EH <space> n AH m IH n AH <space> b iy DX ih <space> b UW s <space> t ay AE f er <space> b ih z n ih s <space> m er <space> jh UW z <space>
Eval:      S      S      S      S      S      S      S      S
```

As we can see, the Substitutions/Deletions and Insertions are shown in detail.

**Question.** Can you manually compute the PER for the previous alignment? What is the PER for this file?

Using the expression of the PER, we obtain that:

$$PER = \frac{S + D + I}{C} \cdot 100\% = \frac{8}{32} \cdot 100\% = 25\%.$$

## References

- [1] Eva Vozáriková et al. “Comparison of Different Feature Types for Acoustic Event Detection System”. In: vol. 368. June 2013, pp. 288–297. ISBN: 978-3-642-38558-2 (Print) 978-3-642-38559-9 (Online). DOI: [10.1007/978-3-642-38559-9\\_25](https://doi.org/10.1007/978-3-642-38559-9_25).