Exercise 4 - 046747

Deep Learning of Speech Signals

January 2025

Guidelines

1. Submit your files using the Moodle system.
2. You are allowed to submit in **pairs**. If you choose to do so, **both students must submit**.
3. In order to submit your solution please submit the following files:
   - `ex_4_part1.py` - Python 3.9+ code file with your implementation for part 1.
   - `ex_4_part3.ipynb` - Python 3.9+ code file with your implementation for part 3. The notebook should include runtime results (graphs, classification results, etc.).
   - `output.txt` - your model's predictions on the given test set (see the instructions below).
   - `ex_4_report_id1_id2.pdf` - A pdf file of your answers for part 2 & 3 (replace ids with submitters ids).
4. For any questions, open a thread on the Q&A forum on the course's Moodle site.

1 Part 1

1.1 DTW

In this exercise, you will implement your first word recognizer using the Dynamic Time Warping (DTW) algorithm. Your algorithm will classify the digits 1, 2, 3, 4, 5. For that, you will implement the DTW algorithm as described below, where the distance metric (d) should be the Euclidean distance.

$$DTW[0,0] = d(0,0)$$
$$DTW[i,j] = d(i,j) + \min\{DTW[i-1,j-1], DTW[i,j-1], DTW[i-1,j]\}$$

Using a built-in DTW implementation is prohibited! You must implement DTW by yourself.

1.2 Dataset

You are provided with five labeled examples for each class [`'one'`, `'two'`, `'three'`, `'four'`, `'five'`] to use as your training set. For the test set you are provided with 253 unlabeled examples. Each file is exactly 1 second long.

1.3 Code

In practice, DTW can be applied to varying length sequences. However, to better understand the advantages of the DTW algorithm, you will compare its performance to a standard Euclidian distance (recall each file is exactly 1 second long).

To sum up, you need to run a 1 nearest neighbor classifier using both Euclidian distance and DTW distance. You need to compute both distance metrics (Euclidian and DTW) for each file in the test set with all training examples. Then, classify each test file as the label of the file with the minimal distance.

You should generate a file named: `output.txt`, with the predictions for each test file using both Euclidian distance and DTW distance. The output file should be constructed as follows:

```
<filename> - <prediction using euclidean distance> - <prediction using DTW distance>
```

For example,

```
sample1.wav - 4 - 4
sample2.wav - 1 - 1
sample3.wav - 3 - 3
...
```

Note: DTW and Euclidean distances can yield different predictions.

1.4 Features

In class, we learned how to transform a time domain waveform to the frequency domain using the Fourier Transform. However, in many applications we would like to use more compressed representation. One representation like that is the Mel Frequency Cepstrum Coefficients (MFCCs).

In order to extract these features and load the wave files, you will use a python package called `librosa`, using the following lines of code:

```
import librosa
y, sr = librosa.load(f_path, sr=None)
mfcc = librosa.feature.mfcc(y=y, sr=sr)
```

The dimensions of the MFCC object should be (20, 32), meaning 20 MFCC features over 32 time steps. A more detailed explanation of the MFCC can be found **here**, and an intuitive explanation can be found **here**.

1.5 Sanity-check

You are provided with a python file called `sanity_check.py`. Before submitting, please run it in the same directory as your `output.txt` file to make sure the format is correct.

2 Part 2

Nowadays, neural nets are used for this task. An example is shown here - **adiyoss-GCommandsPytorch**.
You can be impressed by the performance and the code, There is no need to run it.
Answer the following questions (**up to 1 page**):

1. GCommands is a dataset composed of 30 different words, where each word is pronounced by 1000 speakers. Compare CNN trained on this dataset to the DTW algorithm in terms of training time, inference time, and memory.
2. Given a dataset A consists of 50 samples and dataset B consists of 1,000,000 samples, which approach, CNN or DTW, would you choose for each dataset? Explain **shortly**.
3. Assume that the input consists of speech signals of 7 concatenated digits (e.g., zip codes). Suggest an adaptation for the CNN model to support the prediction of multiple digits at once (the entire zip code).

3 Part 3

On this section, you will experiment with a speech representation in order to try and classify speech utterances.

3.1 X-vectors

The goal of this part is to get familiar with and show how to use pre-trained speaker embeddings like x-vectors for speech classification tasks.

There exist plenty of resources and pre-trained models that can be useful for our task. In particular, x-vectors are the current state of the art approach to obtain speech embeddings that characterize very efficiently different properties in speech, such as speaker characteristics and other features of the audio signal in a way that is compact and discriminative. Once trained, they can be used to obtain a single embedding vector of fixed dimension for each audio input. This vector corresponds to the activations of one of the layers after the pooling layer.

On this part you will work with the `speechbrain` module, which includes few x-vectors embedding models. Make sure to install this module.

Example of an available checkpoint under `speechbrain` module:

- `speechbrain/spkrec-ecapa-voxceleb`: trained using a large speaker corpus: **url**

To extract an embedding from a given `.wav` file you can use this example:

```python
import torch
# import speech brain encoder module
from speechbrain.inference.classifiers import EncoderClassifier

device = "cuda" is torch.cuda.is_available() else "cpu"

# load embedding model to selected device
xvector_model = EncoderClassifier.from_hparams(source="speechbrain/spkrec-ecapa-voxceleb",
                                               savedir="<dir-path-here>")
xvector_model.device = device
xvector_model.to(device)
xvector_model.eval()

# load signal
# unsqueeze is required to add artificial batch dimension for encode_batch.
signal = xvector_model.load_audio('<path-to-wav-file>')
signal = signal.unsqueeze(0).to(device)

# extract embedding, remove artifical batch dim
emb = xvector_model.encode_batch(signal).squeeze(0)
```

These embedding vectors can be used to train models for several speech classification tasks, achieving excellent results.

During this part you will work using a `.ipynb` (jupyter notebook) file, on Google-Colab environment, to reduce local environment setup overhead.

Link to the notebook: `ex4_part3_046747.ipynb`.

3.2 Speaker Classification Using X-vectors

In this section, you will examine the x-vector model as a feature extractor for speaker classification. You will work with Google's `speech_commands` (Gcommands) dataset. Follow the instructions on the notebook, and then answer the following questions in your pdf report:

a) Add a scatter plot of the t-SNE projection of the embeddings you've produced. Make sure that the plot has an appropriate title, the axes are labeled as well, and each speaker is marked with a different color. What can you learn from this plot?

b) Add a table of the results for all three classifiers you chose. Display the accuracy of the classifiers using 5-Fold Cross-Validation (CV).

c) Conclude whether x-vectors are a good representation for speaker classification task. Explain your conclusions.

3.3 Command Classification Using X-vectors

In this section, you will examine the x-vector model as a feature extractor for commands classification. You will work with Google's `speech_commands` (Gcommands) dataset. Follow the instructions on the notebook, and then answer the following questions in your pdf report:

a) Add a scatter plot of the t-SNE projection of the embeddings you've produced. Make sure that the plot has an appropriate title, the axes are labeled as well, and each command is marked with a different color. What can you learn from this plot?

b) Add a table of the results for all three classifiers you chose. Display the accuracy of the classifiers using 5-Fold Cross-Validation (CV).

c) Conclude whether x-vectors are a good representation for command classification task. Explain your conclusions.

3.4 Conclusions

Before jumping to conclusions, it is recommended to read the paper where x-vectors were first presented, you can access it **here**.

After reading the paper, explain your results, and discuss how could you improve some of them.

Good Luck!