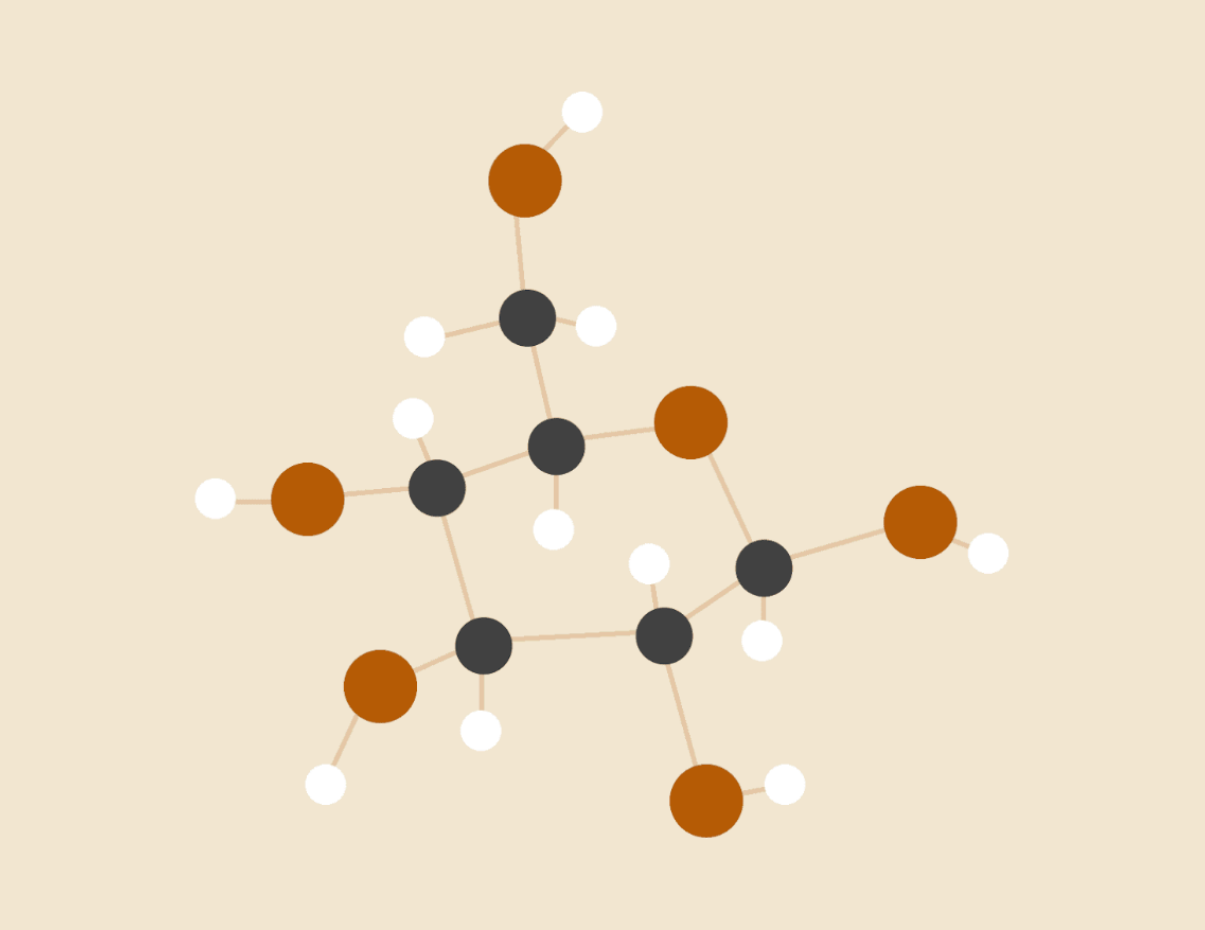
Luigi Faticoso
Antonio Lomuscio
Franci Rrapi

Prof. Aurelio Uncini
Danilo Cominiello

Titolo corso:
Neural Network

Data:
7 maggio 2019



END-TO-END SPEECH RECOGNITION USING QUATERNIAL CONVOLUTIONAL NEURAL NETWORKS

Abstract

In this report, we present an implementation of two different data preprocessing for phonetic classification, and one for dialect classification in Python.
We used the model described in the paper: "Quaternion Convolutional Neural Networks for End-to-End Automatic Speech Recognition". Therefore, the presented data of preprocessing files are very suitable for further experimentation. We compare the performance of our models for a number of different settings. We report for the phonetic classification 40.10% and 23.99% and for the dialect classification 16.57%.

Introduction

Following is the report made for the project of Neural Network that led us to build a model for speech recognition using the quaternion convolutional neural networks.
The work we have done is based on a paper[1] (published on 20 June 2018) and on a repository[2] that provides all the Keras classes for a simple implementation of the QCNN.
The work is about the quaternion representation of audio sources, divided in multiple time-frame frequencies encoded as imaginary parts of a hypercomplex number. The vectors of quaternions are embedded using operations defined by quaternion algebra to preserve distinction between features of each frequency separation.

Our Work

The objective of the work was to replicate the model proposed in the report and to test it on the TIMIT Corpus Dataset[3]. We have started by analyzing the problem to define the following steps:

1. Preprocess all the data before using it as input to our neural network
2. Convert the preprocessed data into quaternion numbers and split all data in three datasets organized as follows:
   - 70%: Training Set
   - 20%: Dev Set
   - 10%: Test Set
3. Concatenate to each row the associated class represented also in quaternion
4. Train the three different models on the Neural Network:
   - Two different phonetic classification
   - Dialect Classification

Materials

Data

To train a neural network for recognizing phones and dialects, it needs to be shown many examples of audio segments and the corresponding phonetic annotations. For each example, it will predict the phonetic contents of the audio. The true phonetic annotations will then be compared to these guesses in order to determine how well the network performs. These examples form a training set, and the final evaluation of a system is based on a held-out test set.

The TIMIT Corpus

TIMIT is a speech data set designed for developing speech recognition systems (Garofolo et al., 1993). It is recorded by Texas Instruments (TI) and transcribed at the Massachusetts Institute of Technology (MIT). The set contains English sentences spoken by 630 speakers in eight dialects of American English. Additionally, time-aligned word and phonetic transcriptions are provided. The phonetic transcriptions are used in this thesis as the target labels for classification. TIMIT is split in a suggested testing and training division. There are no speakers that occur in both the testing and training set, with the motivation that if a model trained on the training set performs well on new speakers, it generalizes well.
There are three collections of sentences: SA, SI and SX. The two SA sentences were read by all speakers in order to expose their dialects. As is consistent with other similar studies (Lee & Hon, 1989; Mohamed et al., 2012; Graves, Mohamed, & Hinton, 2013), these were not included in our experiments to reduce bias towards these sentences.

Data Preparation: "Speech Data”

The TIMIT set contains recorded wave audio files of many pronounced sentences, called utterances. These are not directly suitable as input for a neural network because they contain too much information, making it difficult for a speech recognition system to capture the important information. In essence, a speech recognition system would be confused by the sheer amount of data, most of which is not relevant for recognizing phones. The raw audio data is a waveform that is constantly changing. We can simplify this representation by assuming that, for a small frame of audio, the signal does not change significantly. To extract the most relevant features from this framed waveform, it is usually converted to a mel-

frequency cepstrum. This cepstrum represents the features of a speech spectrum that are the most relevant in human perception of speech. They were introduced by Davis and Mermelstein (1980) and have been used in state-of- the-art ASR systems ever since. We will not go into detail on how a mel-frequency cepstrum is generated; for more information see the work of Davis and Mermelstein. A mel-frequency cepstrum consists of a number of real-valued Mel-Frequency Cepstral Coefficients (MFCCs). We have used 40 MFCCs as input features that are generated using python_speech_features[4], very good tool for converting TIMIT's audio files to MFCCs.

PARAMETERS	DIALECT CLASSIFICATION	PHONEME CLASS. 1	PHONEME CLASS. 2
Number of quaternions (N_SPLIT)	100	100	150
Number of classes	8	61	61
MFCC (Numcep)	40	40	40
Number of filters (Numfilt)	40	40	40
Length of analysis window (Winlen)	25	25	25
Step between successive windows (Winstep)	0.01	0.01	0.01
Accuracy	17%	40%	24%

Approach

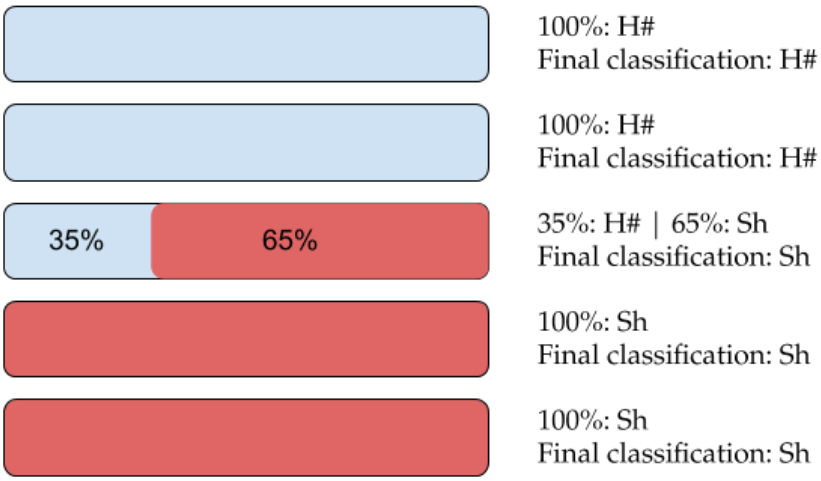
In this section we explain the three different approaches and explain the decisions made which allowed us to have the results presented later.

First Phonetic Classification

Before preprocessing all the audio files into quaternion, we have analyzed the TIMIT Corpus, acknowledging the average duration of the sentences in terms of time-frequency, the average duration of the phonemes and other data. The output is:

Number of single phonemes (without duplicates)	61
Average duration weighted based on the frequency of every phoneme	1289
Standard deviation	371
Average duration of the sentences	49074

Based on those results it's clear that a single phoneme takes around 2% of the sentence. Following there is the conversion of audio files in quaternions by running an analyser script similar to the one used for the TIMIT understanding that a split after 100 quaternions was the best choice. By doing so the algorithm could take in account every phoneme.

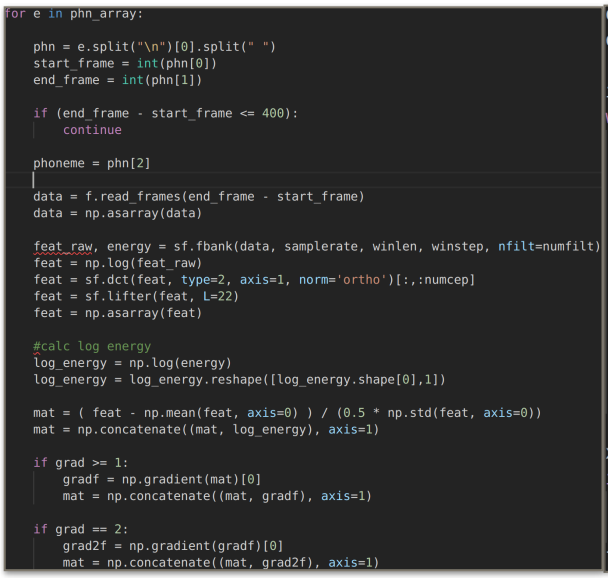


The figure shows how the logic behind process of classification was. At the loading, the sentence gets split into slots, every slot represents 2% of the duration of the sentence, in the figure every slot is represented by the rectangles. Following is the time-frame of the slot with the time-frame of the phoneme of the sentence.

In the case of the figure, the Phoneme H# Started at the beginning of the sentence and lasted for a bit more than 4% of the total duration of the sentence. At the end of the phoneme H# there is the phoneme Sh taking more than the half of the previous slot which means that in that slot there is more presence of the phoneme Sh, so in the end it will label that slot as Sh and so on for the others.
At the end of the preprocessing phase every sentences has been divided in slots composed by 100 quaternions and every slot had an associated phoneme.

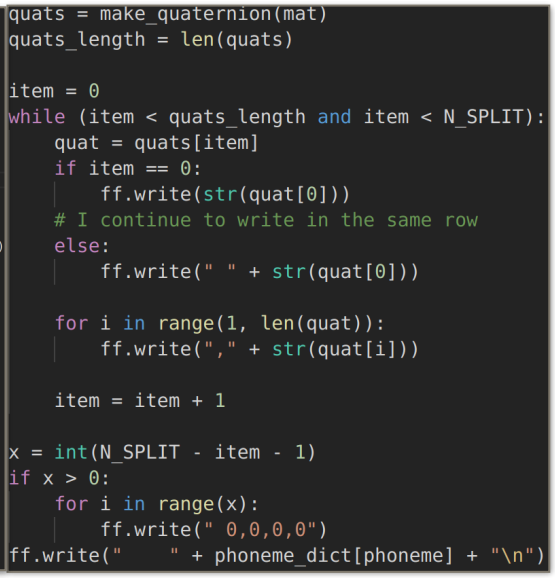
Second Phonetic Classification

In this approach, for the creation of the file to pass as input to the QCNN, for each row a variable number of frames are read every time and pre-processed, and this depends on the interval of frames in which each specific phoneme appears in the audio file (Figure 1). After being pre-processed, the data are converted into quaternion numbers and only a range of this numbers are taken (depends on how much we have set the limit N_SPLIT). In case the numbers generated are less than the limit for each data row (N_SPLIT), then quaternions with all zero values (0,0,0,0) are added (Figure 2). After this, at the end of each row, that regards a specific phoneme, the corresponding phoneme class defined in the PHN file associated is added.



```
for e in phn_array:

    phn = e.split("\n")[0].split(" ")
    start_frame = int(phn[0])
    end_frame = int(phn[1])

    if (end_frame - start_frame <= 400):
        continue

    phoneme = phn[2]

    data = f.read_frames(end_frame - start_frame)
    data = np.asarray(data)

    feat_raw, energy = sf.fbank(data, samplerate, winlen, winstep, nfilt=numfilt)
    feat = np.log(feat_raw)
    feat = sf.dct(feat, type=2, axis=1, norm='ortho')[:,:numcep]
    feat = sf.lifter(feat, L=22)
    feat = np.asarray(feat)

    #calc log energy
    log_energy = np.log(energy)
    log_energy = log_energy.reshape([log_energy.shape[0],1])

    mat = ( feat - np.mean(feat, axis=0) ) / (0.5 * np.std(feat, axis=0))
    mat = np.concatenate((mat, log_energy), axis=1)

    if grad >= 1:
        gradf = np.gradient(mat)[0]
        mat = np.concatenate((mat, gradf), axis=1)

    if grad == 2:
        grad2f = np.gradient(gradf)[0]
        mat = np.concatenate((mat, grad2f), axis=1)
```

Figure 1



```
quats = make_quaternion(mat)
quats_length = len(quats)

item = 0
while (item < quats_length and item < N_SPLIT):
    quat = quats[item]
    if item == 0:
        ff.write(str(quat[0]))
    # I continue to write in the same row
    else:
        ff.write(" " + str(quat[0]))

    for i in range(1, len(quat)):
        ff.write("," + str(quat[i]))

    item = item + 1

x = int(N_SPLIT - item - 1)
if x > 0:
    for i in range(x):
        ff.write(" 0,0,0,0")
ff.write("    " + phoneme_dict[phoneme] + "\n")
```

Figure 2

Bellow is possible to see as example, the begin of a PHN file. The first and second elements represent the interval of frames where a phoneme appear in the audio file while the last element represent the phoneme to consider.

0 2150 h#
2150 3690 p
3690 4428 y

And the phoneme classes are defined as follows:

H#	"1,1,1,1 0,0,0,0 …… 0,0,0,0"
DH	"0,0,0,0 1,1,1,1 …… 0,0,0,0"
…	
ENG	"0,0,0,0 0,0,0,0 …… 1,1,1,1"

Dialect Classification

Consist in the similar code of second phonetic classification but with different implementation.
We preprocess the ".wav" files in eight different folder of dialect with the script "organizer_files_timit_folder_dict.py".
After we used these folders for organize the different dialects in a dictionary (python)

DR1	"1,1,1,1 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0"
….	
DR8	"0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 0,0,0,0 1,1,1,1"

Then for each folder inside of dialects directory we compute:

1. First we scan all files inside the folder with a "cycle for", we split the name of file and we used only the ".wav audio"
2. Before we set the number of elements for each set of quaternions at **"N_SPLIT= 150"** because was the best solution for a good prediction. Then we compute the file in quaternion form (the same seen in phonetic classification).
3. For creation of different "preprocessing.data" (TRAIN, DEV and TEST) we impose a conditional "If" with a counter "count" (who count for all files of eight dialects)

4. In the end we append the right class of dialect in the end of “row file” inside the "while cycle"
5. This repeated for all dialect folders.

After we used these preprocessed data for take the results from Neural Network thanks to **Colab Platform** (Colab_neural_dialect.ipynb). We trained all different settings changing the number of split from 100 quaternion to 400, but the best result was with “150".

Results

The results obtained training the different models for speech classification are shown below.

First Phonetic Classification

```
135057/135057 [==============================] - 385s 3ms/step - loss: 2.4778 - acc: 0.3292 - val_loss: 2.2861 - val_acc: 0.3697
Epoch 2/15
135057/135057 [==============================] - 380s 3ms/step - loss: 2.1217 - acc: 0.4043 - val_loss: 2.1058 - val_acc: 0.4120
Epoch 14/15
135057/135057 [==============================] - 386s 3ms/step - loss: 1.5102 - acc: 0.5539 - val_loss: 2.4330 - val_acc: 0.3938
Epoch 15/15
135057/135057 [==============================] - 383s 3ms/step - loss: 1.4868 - acc: 0.5598 - val_loss: 2.5402 - val_acc: 0.3880
39312/39312 [==============================] - 4s 114us/step
Test Loss = 2.4574050720527882 | Test accuracy = 0.4010734635734636
```

Second Phonetic Classification

```
Epoch 1/15
117195/117195 [==============================] - 208s 2ms/step - loss: 2.9267 - acc: 0.2223 - val_loss: 2.7739 - val_acc: 0.2613
Epoch 2/15
117195/117195 [==============================] - 206s 2ms/step - loss: 2.6885 - acc: 0.2748 - val_loss: 2.6901 - val_acc: 0.2831
.
.
Epoch 14/15
117195/117195 [==============================] - 204s 2ms/step - loss: 1.9350 - acc: 0.4547 - val_loss: 3.3828 - val_acc: 0.2439
Epoch 15/15
117195/117195 [==============================] - 205s 2ms/step - loss: 1.9068 - acc: 0.4612 - val_loss: 3.4418 - val_acc: 0.2391

Evaluating test...
32313/32313 [==============================] - 3s 85us/step
Test Loss = 3.405372650396878 | Test accuracy = 0.23999628632231437
```

Dialect Classification

This is the results of different configuration:

N_SPLIT	batch_size	Epoch	Training acc. %	Test acc. %
150	3	15	0.1687	0.1657
250	3	20	0.1667	0.1657
100	3	15	0.1634	0.1619
250	15	20	0.8192	0.1494
400	15	15	0.9404	0.1464

This is the best results:

271056/271056 [==============================] - 528s 2ms/step - loss: 2.0065 - acc: 0.1639 - val_loss: 2.0052 - val_acc: 0.1610
Epoch 2/15
271056/271056 [==============================] - 513s 2ms/step - loss: 2.0047 - acc: 0.1648 - val_loss: 2.0045 - val_acc: 0.1601
Epoch 3/15
.
.
Epoch 13/15
271056/271056 [==============================] - 509s 2ms/step - loss: 2.0038 - acc: 0.1700 - val_loss: 2.0039 - val_acc: 0.1669
Epoch 14/15
271056/271056 [==============================] - 507s 2ms/step - loss: 2.0039 - acc: 0.1672 - val_loss: 2.0042 - val_acc: 0.1657
Epoch 15/15
271056/271056 [==============================] - 506s 2ms/step - loss: 2.0041 - acc: 0.1687

Evaluating test...
Test Loss = 2.004 | Test accuracy = 0.1657

Conclusion

From this paper was expected that the dialect classification was easier to predict respect to phoneme classification; but we're wrong. The dialects ".wav" audios are very closely to each other and the model can't predict them with good results.
On the other hand, the phoneme classification has overwhelmed our expectations. In first phoneme classification we get the best result thanks to reducing the noise caused by reduction of zeros (padding) making each quaternion slots more clear increasing the precision. For the second phoneme classification has a full knowledge of phonemes because it includes all the parts of a phoneme without cutting the beginning and the end.

REFERENCES

1. “Quaternion Convolutional Neural Networks for End-to-End Automatic Speech Recognition” of Titouan Parcollet, Ying Zhang, Mohamed Morchid, Chiheb Trabelsi, Georges Linarès, Renato De Mori and Yoshua Bengio
2. https://git.io/vx8so
3. https://figshare.com/articles/TIMIT_zip/5802597
4. https://github.com/jameslyons/python_speech_features
5. For the full code visit our repo: https://github.com/luigifaticoso/Quaternion-speech-recognition