دانشگاه تهران

دانشکده علوم و فنون نوین

تمرین شماره سه پردازش گفتار

نام و نام خانوادگی	فاطمه چیت ساز
شماره دانشجویی	830402092
تاریخ ارسال گزارش	19 اردیبهشت 1403

سوال یک :

WaveNet :

WaveNet یک مدل شبکه عصبی حلقه‌ای است که برای تولید گفتار و صدا استفاده می‌شود. این مدل به روش حلقه‌ای (autoregressive) عمل می‌کند و با استفاده از یک شبکه عصبی عمیق، نسبت به دیگر مدل‌های تولید گفتار، کیفیت بالاتری دارد. در این مدل، هر نمونه از گفتار یا صدا به عنوان یک دنباله از نمونه‌های اولیه (sample) در نزدیک‌ترین حالت ممکن تولید می‌شود.

WaveNet از یک ساختار شبکه عصبی حلقه‌ای تشکیل شده است که از لایه‌های حلقه‌ای که به آن dilated causal convolutional layers گفته می‌شود، تشکیل شده است. هر لایه از این لایه‌ها یک عملکرد تبدیل گاوسی (Gaussian transformation) را انجام می‌دهد و نمونه‌های اولیه را به یک فضای بالاترین سطح تبدیل می‌کند. سپس این لایه‌ها با استفاده از یک عملکرد ترکیبی (gating) به هم متصل می‌شوند و یک نمونه اولیه جدید تولید می‌کنند.

در WaveNet، هر لایه با استفاده از یک عملکرد دیلیشن به طوری طراحی شده است که دسترسی به نمونه‌های قبلی بیشتری را داشته باشد. این عملکرد باعث می‌شود تا شبکه عصبی توانایی داشته باشد تا وابستگی‌های بین نمونه‌های گفتار یا صدا را به طور بهتر یادگیری کند.

منابع:

https://deepmind.google/discover/blog/wavenet-a-generative-model-for-raw-audio/

Music Transformer :

Music Transformer یک مدل شبکه عصبی است که برای تولید موسیقی استفاده می‌شود. این مدل به روش تبدیلگر (Transformer) که در مدل‌های پردازش زبان طراحی شده است، عمل می‌کند و با استفاده از یک شبکه عصبی عمیق، موسیقی را به صورت یک دنباله از نوت‌ها تولید می‌کند.

Music Transformer از یک ساختار شبکه عصبی تشکیل شده است که از لایه‌های تبدیلگر (Transformer layers) تشکیل شده است. هر لایه از این لایه‌ها شامل دو بلوک اصلی است: یک بلوک توجه (attention) و یک بلوک حلقه‌ای (feed-forward).

بلوک توجه برای یافتن وابستگی‌های بین نوت‌های موسیقی استفاده می‌شود. این بلوک با استفاده از یک مکانیسم توجه چندروی (multi-head attention) عمل می‌کند و به شبکه عصبی امکان می‌دهد تا بیش از یک نوت را در هر زمان در نظر گرفته و وابستگی‌های بین آن‌ها را یادگیری کند.

بلوک حلقه‌ای برای تبدیل نوت‌های موسیقی به یک فضای بالاترین سطح استفاده می‌شود. این بلوک شامل دو لایه حلقه‌ای است که هر کدام با استفاده از یک عملکرد تناسب (ReLU) و یک لایه نرمالیزه (Layer Normalization) عمل می‌کنند.

منابع :

https://magenta.tensorflow.org/music-transformer

سوال دو :

سوال ۲-۱ ۸ الف)

$$y(t) = a_0 w(t) + a_1 w(t-1) + \cdots + a_N w(t-N)$$

$$y(t+1) = a_0 w(t+1) + a_1 w(t) + \cdots + a_N w(t-N+1)$$

$$\hat{y} \Rightarrow E(y(t+1) \mid y(t)) =$$

$y(t)$ با $a_0 w(t+1)$ باتوجه به اینکه توزیع ما نرمال است و استقلال

$$0 = E(a_0 w(t+1) \mid y(t)]$$ صحیح است بنابراین داریم

$$\# \hat{y}(t+1) = a_1 w(t) + a_2 w(t-1) + \cdots + a_N w(t+1-N)$$

$$MMSE = E(y(t+1) - \hat{y}(t+1)] = E(a_0 w(t+1)^2) = a_0^2$$

$\text{var} = 1$

روز هنرهای نمایشی

۹ فروردین ۱۳۹۷

پنجشنبه Thursday
29 March 2018
۱۱ رجب ۱۴۳۹

سوال ۲-۱۸ ب)

$y(t) = a_0 w(t) + a_1 w(t-1) + \dots + a_N w(t-N)$

$y(t-1) = a_0 w(t-1) + a_1 w(t-2) + \dots + a_N w(t-N-1)$

$y(t+1) = a_0 w(t+1) + a_1 w(t) + \dots + a_N w(t-N+1)$

$y(t) = E(a_0 w(t-1) + \dots) = 0$ ← چون w نویز سفید است

$Cov[Y(t), Y(t-1)] = E[Y(t)Y(t-1)] - \underbrace{E(Y)E(Y(t-1))}_{0}$

$= E(Y(t)Y(t-1))$

$MMSE(Y(t+1)) = Cov[y(t), y(t-1)]Y(t-1) + Cov(y(t), Y(t+1))$

$\times Y(t+1) \longrightarrow a_1 y(t-1) + a_2 y(t+1)$

۱۰ جمعه Friday
30 March 2018
۱۲ رجب ۱۴۳۹

سوال ۲_۲

$$P_x(k) = e^{-\lambda}\frac{\lambda^k}{k!}$$

MLE ⟵ $\prod_{j=1}^{n} e^{-\lambda}\frac{1}{k_j!}\lambda^{k_j}$ ⟵ Liklihood Function

$$\Longrightarrow \log \text{Likelihood} \quad \sum \ln\left(e^{-\lambda}\frac{1}{k_j!}\lambda^{k_j}\right) = \sum_{j=1}^{n}\left(-\lambda - \ln(k_j!) + \ln(\lambda^{k_j})\right] \longrightarrow \sum\left[-\lambda - \ln(k_j!) + k_j \ln(\lambda)\right]$$

$$\Longrightarrow -n\lambda - \sum_{j=1}^{n}\ln(k_j!) + \ln(\lambda)\sum_{j=1}^{n}k_j$$

حال برای دست آوردن λ باید از عبارت بالا نسبت به λ مشتق گرفته و مساوی صفر بگذاریم

$$\frac{d}{d\lambda}\left(-n\lambda - \sum\ln(k_j!) + \ln(\lambda)\sum k_j\right) =$$

$$-n + \frac{1}{\lambda}\sum k_j = 0 \longrightarrow \lambda = \frac{1}{n}\sum_{j=1}^{n}k_j$$

سوال سوم :

در این سوال میخوایم تاثیر پنجره گذاری های مختلف را ببینیم

اولین کار اینه تابع پنجره گذاریمون رو تعریف کنیم

این تابع چهار ورودی دریافت می‌کنه:

1. `data`: آرایه یک بعدی حاوی داده‌های سیگنال
2. `frame_length`: طول هر پنجره یا فریم (تعداد نقاط در هر پنجره)
3. `hop_size`: اندازه گام بین پنجره‌های متوالی (میزان جابجایی پنجره در هر گام)
4. `windowing_function`: نوع تابع پنجره که برای اعمال بر روی هر فریم استفاده می‌شود (rect, hann, cosine, hamming)

خروجی این تابع یک ماتریس دو بعدی است که در آن هر ستون نمایانگر یک فریم پنجره‌بندی شده از سیگنال ورودی است.

در ابتدا، طول داده (`data_length`) و تعداد کل فریم‌های لازم (`number_of_frames`) بر اساس طول داده، طول فریم و اندازه گام محاسبه می‌شود. سپس یک ماتریس خالی با ابعاد `frame_length` در `number_of_frames` ایجاد می‌شود.

سپس در یک حلقه، برای هر فریم، شروع و پایان فریم در سیگنال داده شده مشخص می‌شود. بخش مربوطه از سیگنال انتخاب و با تابع پنجره مورد نظر ضرب می‌شود. این فریم پنجره‌بندی شده در ستون مربوطه در ماتریس `windowed_frames` قرار می‌گیرد.

در نهایت، ماتریس `windowed_frames` که حاوی تمام فریم‌های پنجره‌بندی شده است، بازگردانده می‌شود.

```python
import os
import numpy as np




def ex3_windowing(data, frame_length, hop_size, windowing_function):
    data_length = len(data)
    number_of_frames = 1 + ((data_length - frame_length) // hop_size)
    windowed_frames = np.zeros((frame_length, number_of_frames))

    for i in range(number_of_frames):
        start_index = i * hop_size
        end_index = start_index + frame_length

        frame = data[start_index:end_index]
        if windowing_function == 'rect':
            window= np.ones(frame_length)
        elif windowing_function == 'hann':
            window= np.hanning(frame_length)
        elif windowing_function == 'cosine':
            window= np.cos(np.linspace(0, np.pi, frame_length))
        elif windowing_function == 'hamming':
            window= np.hamming(frame_length)
        else:
            raise ValueError("Invalid windowing function specified")
        windowed_frame = frame * window
        windowed_frames[:, i] = windowed_frame

    return windowed_frames
```

حالا اگ بخوایم از تابعی ک نوشتیم استفاده کنیم

اول فایل را میخوانیم بعد به ازای هر window function مان میایم تابعی که نوشتیم را فراخوانی می کنیم برای محاسبه ی frame length باید سمپل ریتمون که 16000 سمپل در یک ثانیه است را در طول فریم که 24 میلی ثانیه است ضرب کنیم

برای یافتن hope size باید آن بخش overlap را از frame length کم کرده و بقیه را به تابع پاس دهیم که در اینجا پنجاه درصد length ما overlap است

```python
# Read the audio file SX83.WAV and sampling rate
file_path = path.join('.', 'Sounds')
sound_file = path.join(file_path, 'SX83.wav')
Fs, in_sig = wav.read(sound_file)

# Make sure the sampling rate is 16kHz, resample if necessary
Fs_target = 16000
if not (Fs == Fs_target):
    in_sig = sig.resample_poly(in_sig, Fs_target, Fs)
    Fs = Fs_target

# Parameters for windowing
frame_duration_ms = 25  # milliseconds
overlap_percent = 50  # 50% overlap
window_functions = ['hamming','rect', 'hann', 'cosine' ]
#
# Calculate frame length and hop size
frame_length = int((frame_duration_ms / 1000) * Fs)  # Convert milliseconds to samples
overlap_size = int(frame_length * overlap_percent / 100)
hop_size = frame_length - overlap_size

for windowing_function in window_functions:
    # Obtain windowed frames using the windowing_3ex function
    windowed_data = win.ex3_windowing(in_sig, frame_length, hop_size, windowing_function)
```

در نمودار اول باید نمودار زمان به دامنه را بکشیم برای محاسبه دامنه ابتدا میابیم طول سیگنال ما چقدر است و آن را بر sample rate خود تقسیم میکنیم تا تایم را به ثانیه بیابیم

```
# Original audio signal
# Define time axis for the original audio signal
time_orig = np.arange(len(in_sig)) / Fs
axs[0].plot(time_orig, in_sig, color='b')
axs[0].set_title('Original Audio Signal')
axs[0].set_xlabel('Time (s)')
axs[0].set_ylabel('Amplitude')
```

در پلات بعدی باید یک فریم صدا دار پیدا کنیم و نمودار آن را بکشیم برای این کار من فریمی که بیشترین انرژی را دارد انتخاب کردم و نمایش دادم

```
# Calculate the energy of each frame
frame_energies = np.sum(windowed_data ** 2, axis=0)

# Find the frame with the maximum energy (most voiced frame)
voiced_frame_idx = np.argmax(frame_energies)
voiced_frame = windowed_data[:, voiced_frame_idx]

# Subplot 2: Plot the voiced frame
start_time = voiced_frame_idx * hop_size / Fs_target
time_axis_frame = np.arange(len(voiced_frame)) / Fs_target + start_time
axs[1].plot(time_axis_frame * 1000, voiced_frame)
axs[1].set_title(f'Most Voiced Frame ({windowing_function.capitalize()} Window)')
axs[1].set_xlabel('Time (ms)')
axs[1].set_ylabel('Amplitude')
```

برای محاسبه تایم با توجه به اینکه این فریم صدا دار چه موقعیتی دارد میایم و مدت زمانش را پیدا میکنیم و نمایش میدهیم (تایم به میلی ثانیه)

در قسمت بعدی باید نمودار فرکانس به دامنه همین فریم خاص را بکشیم که برای این کار fft این فریم را میابیم و نمایش میدهیم

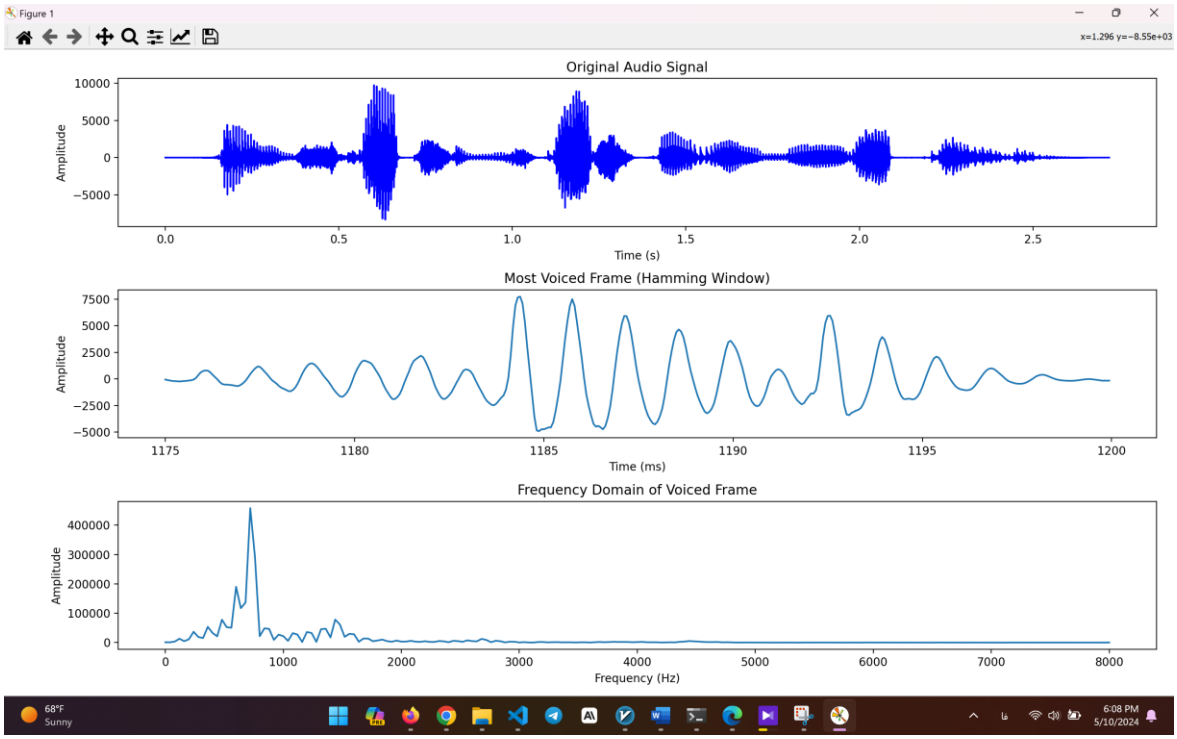
```
# Plot frequency domain of the voiced frame
freq_axis = np.fft.rfftfreq(len(voiced_frame), 1 / Fs_target)
voiced_frame_fft = np.abs(np.fft.rfft(voiced_frame))
axs[2].plot(freq_axis, voiced_frame_fft)
axs[2].set_xlabel('Frequency (Hz)')
axs[2].set_ylabel('Amplitude')
axs[2].set_title('Frequency Domain of Voiced Frame')
fig.tight_layout()
```

voiced_frame_fft = np.abs(np.fft.rfft(voiced_frame)):

np.fft.rfft: این تابع یک تبدیل فوریه سریع را انجام می‌دهد. ورودی آن فریم صوتیه

np.abs: این تابع برای محاسبه مقدار مطلق (جهت به دست آوردن مقدار) از اعداد مختلط استفاده می‌شه

خروجی برای window function های مختلف:



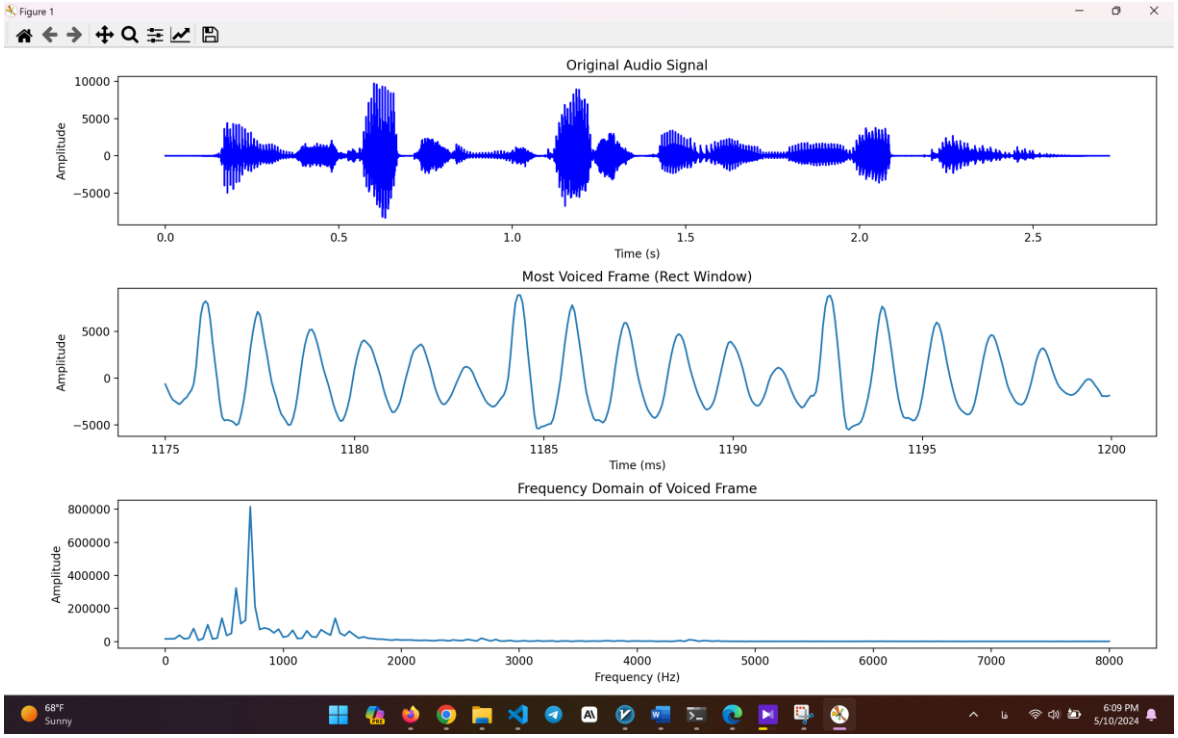


Figure 1
Original Audio Signal
Amplitude
Time (s)
Most Voiced Frame (Rect Window)
Amplitude
Time (ms)
Frequency Domain of Voiced Frame
Amplitude
Frequency (Hz)
68°F
Sunny
6:09 PM
5/10/2024

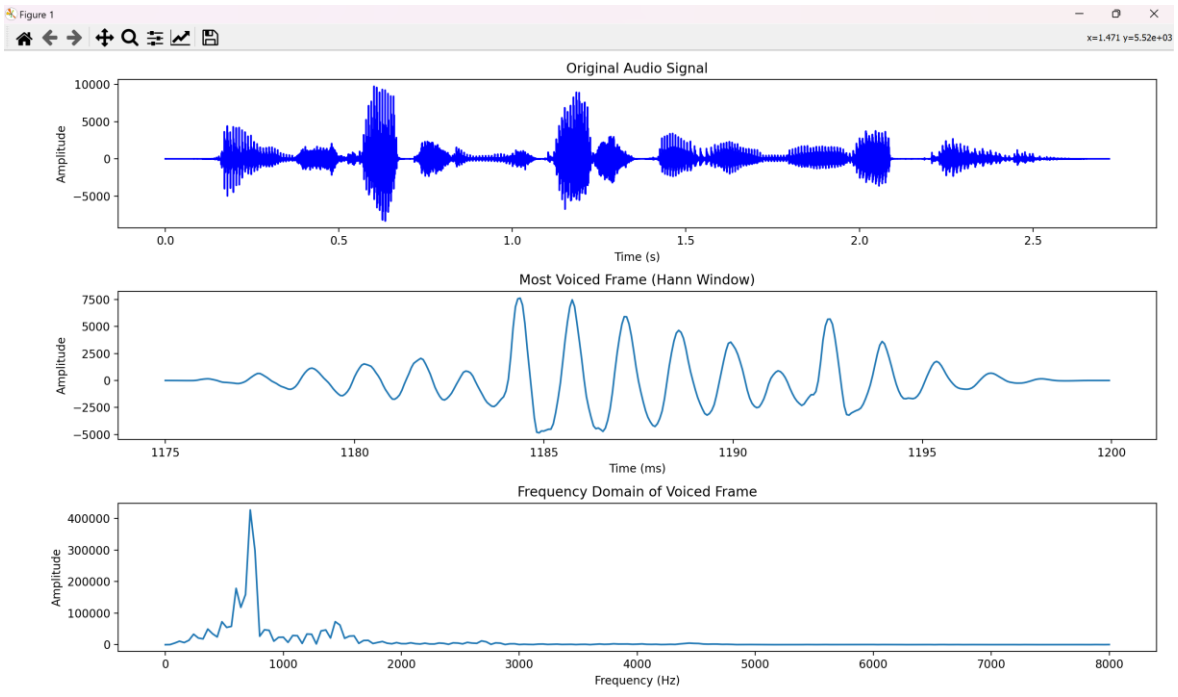
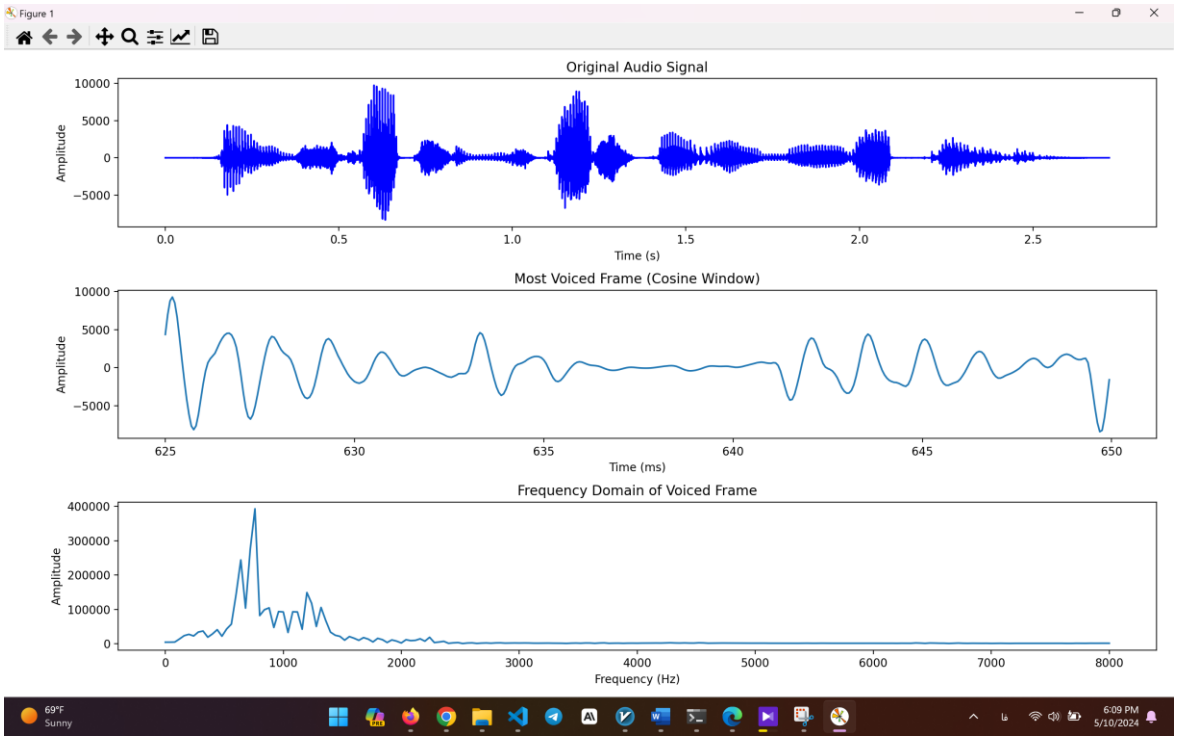


Figure 1
x=1.471 y=5.52e+03
Original Audio Signal
Amplitude
Time (s)
Most Voiced Frame (Hann Window)
Amplitude
Time (ms)
Frequency Domain of Voiced Frame
Amplitude
Frequency (Hz)



در قسمت بعدی باید Magnitude Spectrum برای نصفه اول فریم ها بکشیم

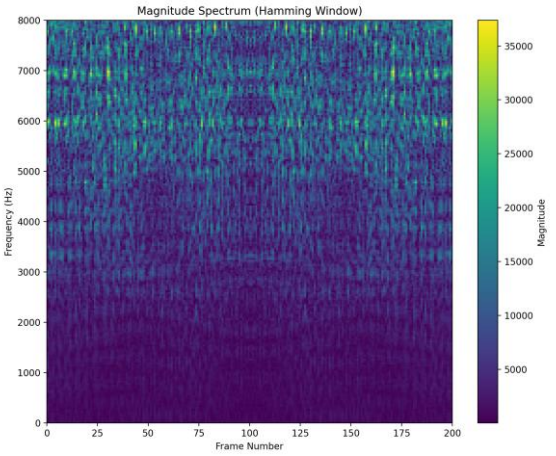
ابتدا نصفه اول فریم ها را جدا کرده و از آنها fft میگیریم و سپس abs آن را بدست آورده و با imshow نمایش میدهیم

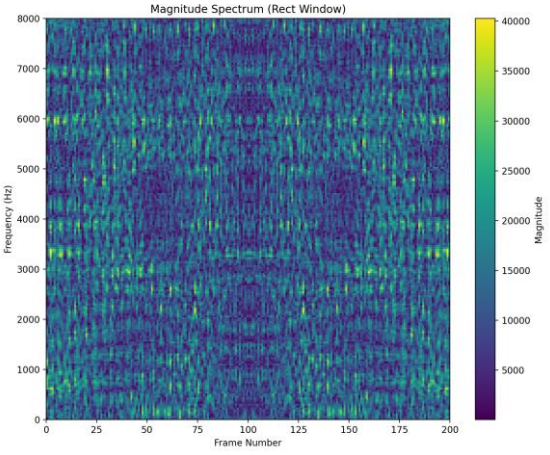
```
frequencies = np.fft.rfftfreq(len(windowed_data[:windowed_data.shape[0]//2,:]), 1 / Fs_target)
```

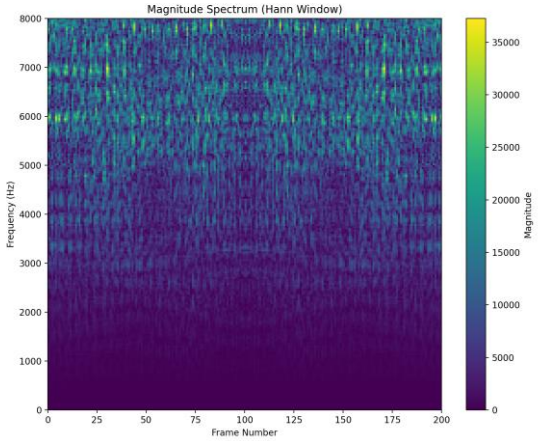
این خط کد برای محاسبه فرکانس‌های متناظر با بن‌های فرکانس استفاده می‌شود. این فرکانس‌ها با استفاده از تابع np.fft.rfftfreq محاسبه می‌شوند. و با این کار طیف فرکانس هایمان را پیدا میکنیم

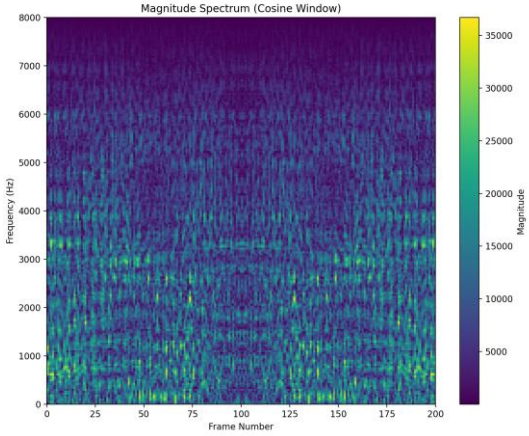
```python
magnitude_spectrums = np.abs(np.fft.fft(windowed_data[:windowed_data.shape[0]//2,:], axis=1))

# Get the number of frames and number of frequency bins
number_of_frames, number_of_freq_bins = magnitude_spectrums.shape

frequencies =np.fft.rfftfreq(len(windowed_data[:windowed_data.shape[0]//2,:]), 1 / Fs_target)

plt.subplots(1, 1, figsize=(10, 8))
# Plot the magnitude spectrum
plt.imshow(magnitude_spectrums, aspect='auto', origin='lower', extent=[0, number_of_frames, 0, max(frequenci
plt.colorbar(label='Magnitude')
plt.xlabel('Frame Number')
plt.ylabel('Frequency (Hz)')
plt.title(f'Magnitude Spectrum ({windowing_function.capitalize()} Window)')
plt.show()

plt.show()
```

خروجی ها :









سوال چهار :

این کد یک سیستم طبقه‌بندی صوتی را پیاده‌سازی می‌کند با استفاده از ویژگی‌های MFCC و الگوریتم‌های مختلف یادگیری ماشینی مانند K-Nearest Neighbors (KNN) و Support Vector Machine (SVM).

توضیح کد:

1. `load_audio_with_padding`: این تابع یک فایل صوتی را بارگیری می‌کند و در صورتی که طول آن کمتر از مقدار مشخصی (که به عنوان `target_duration` ورودی داده می‌شود) باشد، با استفاده از پدینگ، آن را تا مقدار مشخصی (که به طول `target_duration` باشد) پر می‌کند.

```python
def load_audio_with_padding(file_path, target_duration):
    audio, sr = librosa.load(file_path, sr=None,)
    if len(audio) < target_duration:
        pad_width = int(target_duration - len(audio))
        audio = np.pad(audio, pad_width=((0, pad_width)), mode='constant')
    return audio, sr
```

2. `extract_mfcc`: این تابع یک فایل صوتی را بارگیری می‌کند و ویژگی‌های MFCC آن را استخراج می‌کند. این تابع همچنین از تابع `load_audio_with_padding` برای بارگیری فایل و پدینگ آن استفاده می‌کند.

```python
# Function to extract MFCC features from audio file
def extract_mfcc(file_path, num_mfcc=12,num_mel_filters=24,frame_length = 0.020 ,target_duration=None ):
    audio, sr = load_audio_with_padding(file_path, target_duration)
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=num_mfcc,n_mels=num_mel_filters,n_fft=int(sr * frame_length))

    mfcc_delta = librosa.feature.delta(mfccs)

    mfcc_delta2 = librosa.feature.delta(mfccs, order=2)

    return np.hstack((np.mean(mfccs,axis=1), np.mean(mfcc_delta,axis=1), np.mean(mfcc_delta2,axis=1)))
```

✓ 0.0s

3. `load_data`: این تابع وظیفه بارگیری داده‌های آموزشی یا آزمایشی را دارد. این تابع برای هر فایل صوتی در دایرکتوری مورد نظر، ویژگی‌های MFCC را استخراج می‌کند و برچسب متناظر با آن فایل را نیز ذخیره می‌کند.

```python
# Function to load data and extract features
def load_data(data_dir, num_mfcc=12):
    labels = []
    features = []

    for root, dirs, files in os.walk(data_dir):
        for file in files:
            if file.endswith('.wav'):
                file_path = os.path.join(root, file)
                label = int(os.path.basename(root))
                labels.append(label)
                mfcc = extract_mfcc(file_path, num_mfcc=num_mfcc,target_duration=target_duration)
                features.append(mfcc)




    return features, labels
```

✓ 0.0s

برای استفاده از توابع بالا ابتدا یک حلقه میزنیم روی داده هامون تا max طول را پیدا کنیم تا پدینگ مناسب را اضافه کنیم

```
train_data_dir = "TrainSet"
test_data_dir = "TestSet"

train_durations = []
for root, dirs, files in os.walk(train_data_dir):
    for file in files:
        if file.endswith('.wav'):
            file_path = os.path.join(root, file)
            audio, sr = librosa.load(file_path, sr=None)
            train_durations.append(len(audio))

test_durations = []
for root, dirs, files in os.walk(test_data_dir):
    for file in files:
        if file.endswith('.wav'):
            file_path = os.path.join(root, file)
            audio, sr = librosa.load(file_path, sr=None)
            test_durations.append(len(audio))

target_duration = max(max(train_durations), max(test_durations))
```

بعد یافتن ماکزیمم طول زمان load کردن داده ترین و تست ماست که در تابع load data ما پدینگ مناسب اضافه میشود و سپس فیچر mfcc یافت میشود و به عنوان فیچر برگردادنده میشود با توجه به اینکه mfcc میاید و برای هر فریم آن تعداد ویژگی که میدهیم ویژگی برمیدارد بهتر است میانگین فریم ها را بگیریم پس دوازده ویژگی برای mfcc داریم دوازده تا برای مشتق اول و دوازده تا برای مشتق دوم

داده ها ک اماده شد یک تبدیل به np هم میکنیم که کار باهاشون راحت تر بشه

```
# Load training data
train_data_dir = "TrainSet"
X_train, y_train = load_data(train_data_dir)

# Load test data
test_data_dir = "TestSet"
X_test, y_test = load_data(test_data_dir)
```

✓ 9.9s

```
X_train =  np.array(X_train)

X_test = np.array(X_test)
```

✓ 0.0s

```
print(X_test.shape,X_train.shape)
```

✓ 0.0s

```
(296, 36) (1900, 36)
```

سپس مدل KNN با انتخاب مقادیر مختلف برای پارامتر K آموزش داده می‌شود و دقت آن بر روی داده‌های آزمایشی محاسبه می‌شود.

```python
# Define K values for KNN
k_values = [7, 11, 15, 20]

# Iterate over K values
for k in k_values:
    # Train KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict labels for test data
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for K={k}: {accuracy}")
```

✓ 0.0s

```
Accuracy for K=7: 0.46283783783783783
Accuracy for K=11: 0.4831081081081081
Accuracy for K=15: 0.46959459459459457
Accuracy for K=20: 0.46959459459459457
```

دقت داده ما برای مقادیر مختلف k قابل نمایش است

پس از آن، دو مدل SVM با هسته خطی و چند جمله‌ای نیز با استفاده از ویژگی‌های MFCC آموزش داده می‌شود و دقت آن‌ها نسبت به داده‌های آزمایشی محاسبه می‌شود.

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# SVM with linear kernel using MFCC features
svm_linear_mfcc = SVC(kernel='linear')
svm_linear_mfcc.fit(X_train, y_train)
y_pred_linear_mfcc = svm_linear_mfcc.predict(X_test)
accuracy_linear_mfcc = accuracy_score(y_test, y_pred_linear_mfcc)
print("Accuracy of SVM with linear kernel using MFCC features:", accuracy_linear_mfcc)

# SVM with polynomial kernel using MFCC features
svm_poly_mfcc = SVC(kernel='poly')
svm_poly_mfcc.fit(X_train, y_train)
y_pred_poly_mfcc = svm_poly_mfcc.predict(X_test)
accuracy_poly_mfcc = accuracy_score(y_test, y_pred_poly_mfcc)
print("Accuracy of SVM with polynomial kernel using MFCC features:", accuracy_poly_mfcc)
```

✓ 2.3s

```
Accuracy of SVM with linear kernel using MFCC features: 0.7297297297297297
Accuracy of SVM with polynomial kernel using MFCC features: 0.20945945945945946
```

+ Code + Markdown

دقت برای حالت خطی و پلی محاسبه شده و به شرح بالاست

کد های این فایل در پوشه Q4 در فایل Q4-a موجود میباشد

برای قسمت بعدی به جای استفاده از mfcc باید از lpc استفاده کنیم

```python
# Function to extract MFCC features from audio file

def extract_lpc(file_path, num_lpc=14):
    # Load audio file
    y, sr = load_audio_with_padding(file_path)

    # Extract LPC features
    lpc = librosa.core.lpc(y, order=num_lpc)

    return lpc
```

✓ 0.0s

نتایج lpc با knn:

```python
# Define K values for KNN
k_values = [7, 11, 15, 20]

# Iterate over K values
for k in k_values:
    # Train KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict labels for test data
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for K={k}: {accuracy}")
```

✓ 0.1s

```
Accuracy for K=7: 0.5405405405405406
Accuracy for K=11: 0.5405405405405406
Accuracy for K=15: 0.5337837837837838
Accuracy for K=20: 0.4966216216216216
```

نتایج lpc با svm :

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# SVM with linear kernel using LPC features
svm_linear_lpc = SVC(kernel='linear')
svm_linear_lpc.fit(X_train, y_train)
y_pred_linear_lpc = svm_linear_lpc.predict(X_test)
accuracy_linear_lpc = accuracy_score(y_test, y_pred_linear_lpc)
print("Accuracy of SVM with linear kernel using LPC features:", accuracy_linear_lpc)

# SVM with polynomial kernel using LPC features
svm_poly_lpc = SVC(kernel='poly')
svm_poly_lpc.fit(X_train, y_train)
y_pred_poly_lpc = svm_poly_lpc.predict(X_test)
accuracy_poly_lpc = accuracy_score(y_test, y_pred_poly_lpc)
print("Accuracy of SVM with polynomial kernel using LPC features:", accuracy_poly_lpc)
```

✓ 0.5s

```
Accuracy of SVM with linear kernel using LPC features: 0.6756756756756757
Accuracy of SVM with polynomial kernel using LPC features: 0.4222972972972973
```

توضیح تابع mfcc و lpc:

تابع lpc ما فایل audio ما که همان y است و order که همان تعداد ویژگی هایی که میخوایم است را میگیرد

تابع mfcc :

y: این ورودی نمایانگر سیگنال صوتی است که از فایل صوتی بارگیری شده است.

sr: این ورودی نشان دهنده نرخ نمونه‌برداری (Sampling Rate) سیگنال صوتی است، یعنی تعداد نمونه‌های صوتی در هر ثانیه. این ورودی مشخص می‌کند که هر ثانیه از سیگنال صوتی چند نقطه داده دارد.

n_mfcc: این پارامتر تعداد ضرایب Cepstral Coefficients (MFCC) را که برای هر فریم محاسبه می‌شود، مشخص می‌کند. این ضرایب نمایانگر ویژگی‌های اصلی صوتی هستند.

n_mels: این پارامتر تعداد فیلترهای Mel-frequency را که برای تبدیل طیف فرکانسی صوت به مقیاس فرکانسی مل استفاده می‌شود، مشخص می‌کند.

n_fft: این پارامتر طول پنجره فوریه (FFT window) را که برای محاسبه تبدیل فوریه سریع استفاده می‌شود، مشخص می‌کند

که ما این طول پنجره با با استفاده از طول فریم به ثانیه در فریم ریت میتوانیم محاسبه کنیم

کد های این فایل در پوشه Q4 در فایل Q4-b موجود میباشد

برای قسمت بعدی باید هم از lpc و هم از mfcc استفاده کنیم پس تابع load ما بدین صورت میشود :

```python
# Function to extract MFCC features from audio file
def extract_feature(file_path, num_mfcc=12,num_mel_filters=24,frame_length = 0.020  ):
    audio, sr = librosa.load(file_path, sr=None)
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=num_mfcc,n_mels=num_mel_filters,n_fft=int(sr * frame_length))


    lpc=extract_lpc(audio)

    return np.concatenate((np.mean(mfccs,axis=1),lpc.flatten()))
```

نتایج :

استفاده از knn:

```python
# Define K values for KNN
k_values = [7, 11, 15, 20]

# Iterate over K values
for k in k_values:
    # Train KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict labels for test data
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for K={k}: {accuracy}")
```

✓ 0.3s

```
Accuracy for K=7: 0.4864864864864865
Accuracy for K=11: 0.4560810810810811
Accuracy for K=15: 0.4831081081081081
Accuracy for K=20: 0.46959459459459457
```

استفاده از svm :

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# SVM with linear kernel using MFCC features
svm_linear_mfcc = SVC(kernel='linear')
svm_linear_mfcc.fit(X_train, y_train)
y_pred_linear_mfcc = svm_linear_mfcc.predict(X_test)
accuracy_linear_mfcc = accuracy_score(y_test, y_pred_linear_mfcc)
print("Accuracy of SVM with linear kernel using MFCC and lpc features:", accuracy_linear_mfcc)

# SVM with polynomial kernel using MFCC features
svm_poly_mfcc = SVC(kernel='poly')
svm_poly_mfcc.fit(X_train, y_train)
y_pred_poly_mfcc = svm_poly_mfcc.predict(X_test)
accuracy_poly_mfcc = accuracy_score(y_test, y_pred_poly_mfcc)
print("Accuracy of SVM with polynomial kernel using MFCC and lpc features:", accuracy_poly_mfcc)
```

✓ 2.1s

```
Accuracy of SVM with linear kernel using MFCC and lpc features: 0.7871621621621622
Accuracy of SVM with polynomial kernel using MFCC and lpc features: 0.21621621621621623
```

کد های این فایل در پوشه Q4 در فایل Q4-c موجود میباشد

در قسمت اخر باید هم از ویژگی lpc و هم mfcc و zero crossing استفاده کنیم

بنابرین تابع load ما بدین صورت میشود :

```python
# Function to extract MFCC features from audio file
def extract_feature(file_path, num_mfcc=12,num_mel_filters=24,frame_length = 0.020  ):
    audio, sr = load_audio_with_padding(file_path)
    mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=num_mfcc,n_mels=num_mel_filters,n_fft=int(sr * frame_length))

    # Calculate Zero Crossing Rate
    zcr = librosa.feature.zero_crossing_rate(audio).sum()
    lpc=extract_lpc(audio)

    return np.concatenate((np.mean(mfccs,axis=1),lpc.flatten(),[zcr]))
```

تابع feature.zero_crossing_rate میزان تغییر علامت (تعداد بارهایی که سیگنال از مثبت به منفی یا برعکس تغییر می‌کند) را برای سیگنال ورودی محاسبه می‌کند. سپس با استفاده از متد .sum()، این تعداد تغییرات علامت برای تمام نقاط سیگنال جمع‌آوری می‌شود و به عنوان میزان تغییرات علامت کلی در سیگنال برگردانده می‌شود.

نتایج برای knn :

```python
# Define K values for KNN
k_values = [7, 11, 15, 20]

# Iterate over K values
for k in k_values:
    # Train KNN model
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Predict labels for test data
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for K={k}: {accuracy}")
```

✓ 0.3s

```
Accuracy for K=7: 0.49324324324324326
Accuracy for K=11: 0.4864864864864865
Accuracy for K=15: 0.4831081081081081
Accuracy for K=20: 0.4864864864864865
```

نتایج برای : svm

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# SVM with linear kernel using MFCC features
svm_linear_mfcc = SVC(kernel='linear')
svm_linear_mfcc.fit(X_train, y_train)
y_pred_linear_mfcc = svm_linear_mfcc.predict(X_test)
accuracy_linear_mfcc = accuracy_score(y_test, y_pred_linear_mfcc)
print("Accuracy of SVM with linear kernel using MFCC features:", accuracy_linear_mfcc)

# SVM with polynomial kernel using MFCC features
svm_poly_mfcc = SVC(kernel='poly')
svm_poly_mfcc.fit(X_train, y_train)
y_pred_poly_mfcc = svm_poly_mfcc.predict(X_test)
accuracy_poly_mfcc = accuracy_score(y_test, y_pred_poly_mfcc)
print("Accuracy of SVM with polynomial kernel using MFCC features:", accuracy_poly_mfcc)
```

✓ 2.0s

```
Accuracy of SVM with linear kernel using MFCC features: 0.8175675675675675
Accuracy of SVM with polynomial kernel using MFCC features: 0.21621621621621623
```

کد های این فایل در پوشه Q4 در فایل Q4-d موجود میباشد

جدول :

Svm-Poly	Svm-linear	knn=k=7	
۰٫۲۰	۰٫۷۲	۰٫۴۴	MFCC
۰٫۴۲	۰٫۴۷	۰٫۵۴	LPC
۰٫۲۱	۰٫۷۸	۰٫۴۸	MFCC-LPC
۰٫۲۱	۰٫۸۱	۰٫۴۴	MFCC-LPC-ZCR

همانطور که مشاهده میشود وقتی تعداد ویژگی ها بیشتر باشد svm به مراتبط بهتر کار میکند و هر چه تعداد ویژگی ها را بالا ببریم یعنی به جای lpc از mfcc استفاده کنیم یا مجموع mfcc و lpc را استفاده کنیم دقت از استفاده تنها آنها بهتر میشود البته svm در حالت poly برای lpc بهتر کار میکند

برای knn وقتی تعداد ویژگی ها بیشتر میشود دقت بالا میرود اما lpc باز هم در اینجا استثنا است و بالاترین دقت را در جدول دارد

بنابراین به طور کلی بالاترین دقت مربوط به svm در حالت linear به همراه تمام سه ویژگی است که این نشان دهنده این موضوع است که وقتی تعداد ویژگی ها موثر بیشتر میشود ترین بهتری خواهیم داشت و به دقت بهتری در داده ازمون میرسیم در مورد اینکه mfcc بهتر از یا lpc به طور قطع در همه الگوریتم های ml نمیتوان نظر داد اما دقت بهترین الگوریتم در mfcc بیشتر است mfcc فیچر های بهتری را از صدا بیرون میکشد همچنین میتوان از مشتق اول و دوم آن نیز استفاده کرد