به نام خدا

موضوع: دسته بندی ژانر موسیقی

نام اعضای گروه:

فاطمه حسینی

حسین هدایتی

پرهام شیروانیون

نام استاد:

علی تورانی

توضیحات کلی:

برای پیاده سازی این پروژه ابتدا دیتاست مربوط به فایل های مربوط به موسیقی را از سایت زیر دریافت کردیم.

https://colab.research.google.com/drive/1k_Q6b7KBVEbybW-Tl_k295hvAUKIJArs?usp=sharing

پس از دیدن آموزش هایی در این زمینه شروع به پیاده سازی توابع و استفاده از دیتاست مربوطه برای ساخت شبکه عصبی کردیم.

در ابتدای کار کتابخانه های مورد نیاز را به پروژه اضافه کردیم.

باتوجه به اینکه نیاز به پردازش فایل های صوتی داریم از کتابخانه librosa برای کار با فایل های صوتی استفاده میکنیم.

در قسمت بعدی مسیر هایی را که نیاز داریم مثل دیتاست و فایل json را تعیین میکنیم.

سپس توابع مورد نیاز در برنامه را داریم که به صورت زیر می باشند:

```
mfccToJson
load_data
prepare_datasets
plot_history
predict
build_model
```

: تئوری کلی

شبکه عصبی مصنوعی در واقع سیستمی است که از سلول های عصبی (نورون ها) الهام گرفته است به طوری که با انجام دادن پردازش ها و عملیات های کوچک منطقی و انتقال داده ها در شبکه می تواند فرایند یادگیری را شبیه سازی بکند و شبکه عصبی می تواند آموزش ببیند .

هر شبکه عصبی دارای لایه های مختلفی از نورون هاست که شامل لایه ورودی ، لایه خروجی و لایه های میانی خواهد شد .

فورووارد پروپگیشن :

در فرایند فورووارد پرو پگیشن : برای به دست آوردن مقدار خروجی هر کدام از لایه ها نیاز است که ایتدا تابع نت هر نورون را محاسبه کرده و سپس با قرار دادن مقدار تابع نت در تابع فعالساز یک مقدر مناسبی را به عنوان خروجی نورون دریافت بکنیم .

تابع فعالساز سیگموِئید یکی از توابع کاربردی است که به هر مقدار حقیقی عددی بین 0 و 1 را نظیر می نماید .

برای انجام عملیات به دست اوردن تابع نت می توان با استفاده از ماتریس ورودی ها (خروجی توابع فعالساز لایه پیشین) و ماتریس وزن ها و به وسیله ضرب آنها در هم مقدار ماتریس نت را به دست آورد که هر عنصر ماتریس مربوط به یکی از نورون هاست .
سپس با قرار دادن هر کدام از نت نورون ها در تابع سیگموئید مقدار خروجی را حساب کرد .

بک پروپگیشن : با طی شدن مرحله فورووارد پرو پگیشن و طی ایپاک های مختلف مقدار نهایی شبکه عصبی با مقدار هدف تفاوت خواهد داشت که می توان با استفاده از تابعی به نام تابع ارور این مقدار را فرموله کرد .

با طی هر ایپاک هدف به روز رسانی وزن هاست به طوری پس از طی چندین مرحله به مقدار مطلوب برسیم ، اگر بتوانیم نرخ رشد تابع ارور را به دست آوریم میتوانیم با حرکت در خلاف مسیر آن بهترین وزن ها را انتخاب کنیمم و زود تر به نتیجه برسیم ، مفهوم شیب همان مفهوم مشتق است کافیست تا از تابع ارور نسبت به وزن ها مشتق بگیریم آنگاه توابعی به دست خواهد آمد که راهنمای ما در بروز رسانی وزن ها و آپدیت شبکه عصبی و در نهایت راهنمای شبکه عصبی در انجام فرایند یادگیری خواهند بود .

کد پروژه:

```
#BCI PROJECT MUSIC GENRE CLASSIFICATION

#importing libraries
import json
import os
import math
import librosa
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras



DATASET_PATH = "dataset"
DATA_PATH = "json/file"
JSON_PATH = "data.json"

#########GETING AND EXTRACTING OUR DATASET########

#function to save all data of audios to json
def mfccToJson(dataPath, jsonPath, nMfcc=13, n_fft=2048, hop_length=51
2, nSegments=5):
    samples_per_segment = int( 22050 * 30 / nSegments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / hop
_length)
    #data is our dictionary to save information of audio like genre, l
abels and mfcc
    data = { "genre": [], "labels": [], "mfcc": [] }
    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataPat
h)):
        #if we are in genre's folder
        if dirpath is not dataPath:
            data["genre"].append(dirpath.split("/")[-
1])  #add the name of genre to our data
            # processing music with librosa library
            for music in filenames:
                musicPath = os.path.join(dirpath, music)  #getting the
 path of each music
                signal, sample_rate = librosa.load(musicPath, sr=22050
)   #load the music
                # process all segments of audio file
```

```
                for d in range(nSegments):
                    mfcc = (librosa.feature.mfcc(signal[(samples_per_s
egment * d) : (samples_per_segment * d + samples_per_segment)], sample
_rate, n_mfcc=nMfcc, n_fft=n_fft, hop_length=hop_length)).T
                    if len(mfcc) == num_mfcc_vectors_per_segment:
                        data["labels"].append(i-1)
                        data["mfcc"].append(mfcc.tolist())

    # save all to our json file
    with open(jsonPath, "w") as fp:
        json.dump(data, fp, indent=4)



def load_data(data_path):
    with open(data_path, "r") as fp:
        data = json.load(fp)
    return  np.array(data["mfcc"]), np.array(data["labels"])

def prepare_datasets(test_size, validation_size):
    X, y = load_data(DATA_PATH)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
e=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_
train, y_train, test_size=validation_size)
    return X_train, X_validation, X_test, y_train, y_validation, y_tes
t



def plot_history(history):
    fig, axs = plt.subplots(2)
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy"
)
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")
    plt.show()
```

```
def predict(model, X, y):
    X = X[np.newaxis, ...]
    prediction = model.predict(X)
    predicted_index = np.argmax(prediction, axis=1)
    print("Target: {}, Predicted label: {}".format(y, predicted_index)
)



def build_model(input_shape):

    model = keras.Sequential()
    model.add(keras.layers.LSTM(64, input_shape=input_shape, return_se
quences=True))
    model.add(keras.layers.LSTM(64))
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.3))
    model.add(keras.layers.Dense(10, activation='softmax'))
    return model
```