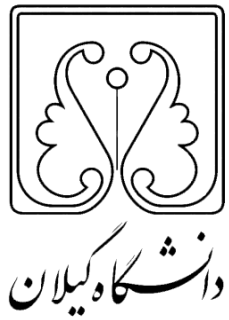


گزارش پروژه نهایی هوش محاسباتی

استاد: علی تورانی

عنوان:

Spoken Digits Classification

اعضای گروه:

آرش علی پور : ۹۷۱۲۲۶۸۱۰۰

فاطمه کمانی : ۹۷۰۱۲۲۶۸۰۰۳۳

امیر عباس نصیری : ۹۸۰۱۲۲۶۸۰۰۴۷

هدف پروژه:

توانایی تشخیص ارقام به صورت شنیداری

شرح پروژه:

در این پروژه سعی شد تا با استفاده از شبکه عصبی پرسپترون چند لایه دیتا های ورودی جدا سازی و دسته بندی شوند.

حاصل این کار دقت در حد ۹۰ درصد بود.

فولدر زیر مسیر دسترسی به فولدر Spoken_Digits_Classification روی درایو است:

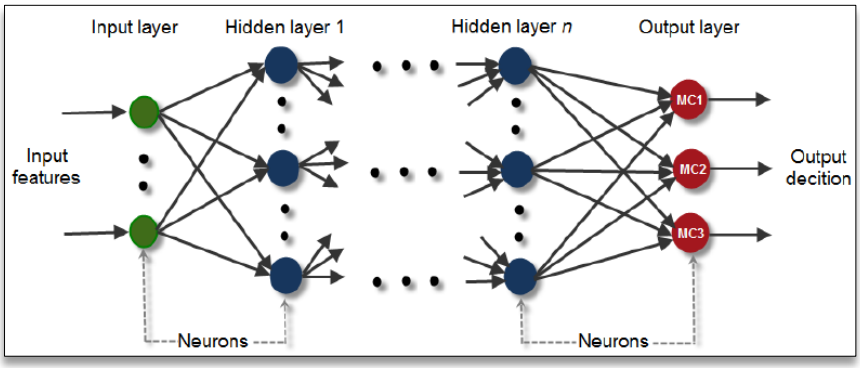
https://drive.google.com/drive/folders/1uMnuZp2s2cafBznaZ7HjbD-U72SlQ27G?usp=sharing

شبکه عصبی پرسپترون چند لایه Multilayer perceptron(MLP)

شبکه عصبی پرسپترون چند لایه، دسته ای از شبکه های عصبی مصنوعی Feed Forward محسوب می شوند. در یک شبکه عصبی پرسپترون چند لایه، حداقل سه لایه از نود ها وجود خواهند داشت:

- یک «لایه ورودی» (Input Layer)
- یک «لایه نهان» (Hidden Layer)
- یک «لایه خروجی» (Output layer)

نودهای شبکه عصبی که به آن ها «نرون» (Neuron) نیز گفته می شود، واحدهای محاسباتی در یک شبکه عصبی محسوب می شوند. در این شبکه عصبی، از خروجی های لایه اول (ورودی)، به عنوان ورودی های لایه بعدی (نهان) استفاده می شود؛ این کار به همین شکل ادامه پیدا می کند، تا زمانی که، پس از تعداد خاصی از لایه ها، خروجی های آخرین لایه نهان به عنوان ورودی های لایه خروجی مورد استفاده قرار می گیرد. به لایه هایی که بین لایه ورودی و لایه خروجی قرار می گیرند، «لایه های نهان» (Hidden Layers) گفته می شود. شبکه های پرسپترون چند لایه، مانند شبکه های عصبی پرسپترون تک لایه، حاوی مجموعه ای از وزن ها نیز هستند که باید برای آموزش و یادگیری شبکه عصبی تنظیم شوند.



دیتاست:

مجموعه داده ها به سه دسته تقسیم می شوند:

- مجموعه ی Training Data: برای هدایت پروسه ی و به روز کردن وزن های شبکه ی عصبی به هنگام آموزش، به کار گرفته می شوند.
- مجموعه ی Validation Data: برای نظارت کردن بر کیفیت مدل شبکه ی عصبی به هنگام فرآیند یادگیری و تعیین شرط توقف یادگیری برای پروسه ی یادگیری استفاده می شوند.
- مجموعه ی Test Data: برای تعیین کیفیت نهایی شبکه ی آموزش دیده شده استفاده می شود.

در این پروژه ما دیتاست مورد نظر را از سایت زیر دانلود کردیم.

https://github.com/Jakobovski/free-spoken-digit-dataset

این دیتاست شامل ۳۰۰۰ فایل صوتی بود که توسط ۶ نفر ضبط شده و ارقام ۰ تا ۹ را پوشش میداد. هر نفر هر رقم را ۵۰ بار اجرا کرده بود.

آماده سازی دیتاست:

مجموعه ی دیتاست دانلود شده شامل ۳۰۰۰ فایل صوتی (بعضا دارای قسمت silence) بود که با استفاده از کد زیر تا حد زیادی از این silence ها از فایل ها حذف شد و فایل های بدست آمده در پوشه ی

"Spoken_Digits_Classification /Audio/cut_silence/" قرار داده شد.

https://colab.research.google.com/drive/1tjRleaArahLjvxkBrWTHZy3mUgo5gpIl?usp=sharing

تابع کات باید دقیقا یک فایل خروجی بدهد اگر کمتر یا بیشتر بود خطایابی می شود.

پس از اجرای توابع زیر ۲۸۰۸ فایل پردازش شده در فولدر cut_silence ذخیره شد و ۱۹۲ فایل به دلیل صدای بسیار کم حذف شد.

```
In [ ]: #removing silence

from pydub import AudioSegment
from pydub.silence import split_on_silence
import os


def cut(
    file_name,
    folder_path,
    des_path,
    leng=300,
    ):

    sound_file = AudioSegment.from_wav(folder_path + file_name)
    audio_chunks = split_on_silence(sound_file, min_silence_len=leng,
                                    silence_thresh=-40)
    out_file = des_path + file_name
    for (i, chunk) in enumerate(audio_chunks):

        chunk.export(out_file, format='wav')

    # if code coudnt find a part -> try shortning the silence limit
    if len(audio_chunks) == 0 and leng > 50:
        cut(file_name, folder_path, des_path, leng - 50)
    # probably a silence file -> drop
    elif len(audio_chunks) == 0 and leng <= 50:
        print ('Silence', out_file)
    # more than one parted file
    elif len(audio_chunks) >1:
        print("!Error!", file_name)
        os.remove(out_file)
```

```
In [ ]: import os

# a function for collecting all files in a path and cut them to make no silence files
def cutting_files(folder_path, des_path):
    mylist = os.listdir(folder_path)
    mylist.sort()
    i = 0
    for file_name in mylist:
        cut(file_name, folder_path, des_path)
        if i % 50 == 0:
            print ('%.2f' % (i / 3000 * 100), '%')
        i += 1


cutting_files('/content/drive/MyDrive/CI_files/Spoken_Digits_Classifications/Audio/raw_recordings/'
      ,
      '/content/drive/MyDrive/CI_files/Spoken_Digits_Classifications/Audio/cut_silence/'
      )
```

Feature extraction:

با استفاده از کد زیر از هر فایل صوتی ۳۰ ردیف ویژگی استخراج میشود (به کمک دستور librosa.feature.mfcc)
Shape=(30,n)

سپس این ویژگی ها در ردیف خود با هم میانگین گرفته شده و داده ای یک بعدی با طول ۳۰ را می سازند. (با استفاده از np.mean) shape=(30,)

در انتها این ارایه به عنوان دیتای استخراج شده به مراحل بعدی می رود. لیبل ها نیز از روی کاراکتر اول اسم هر فایل بدست می آیند.

```
[126]  1 # importing audios and collecting labels
       2
       3 sounds_dir = dir + "Audio/cut_silence/"
       4 # integer of sampels per audio
       5 samples=30
       6 # list for data
       7 data_list = []
       8 # list for label
       9 data_label = []
      10 # importing the audio file names into "mylist" array
      11 mylist = os.listdir(sounds_dir)
      12 # shuffling audio names
      13 random.shuffle(mylist)
      14
      15 # using tqdm for showing a progress bar
      16 for i in tqdm(range(len(mylist)), position=0, leave=True):
      17     # making file path for each file
      18     path = sounds_dir + mylist[i]
      19     # using librosa to extract data and sampling_rate from audios
      20     data, sampling_rate = librosa.load(path)
      21     # extracting audio feature using mfcc number of rows = samples
      22     # according to file length we have feature number of columns
      23     # but using np.mean with axis 1 we are able to convert (samples,col) shape into (samples,) shape
      24     # this function calculates the average of each row
      25     mfccs = np.mean(librosa.feature.mfcc(y=data, sr=sampling_rate, n_mfcc=samples), axis=1)
      26     # appending mfccs to data_list
      27     data_list.append(mfccs)
      28     # appending labels collected from first char of the name of each audio file ex: 9_yweweler_48.wav -> label='9'
      29     data_label.append(int(mylist[i][0]))

     100%|██████████| 2808/2808 [00:54<00:00, 51.33it/s]
```

نمونه دیتای استخراج شده:

```
[45]  1 # length and example
      2
      3 print("number of data: ", len(data_list))
      4 print("number of label: ", len(data_label))
      5 print("data ex: ", data_list[10:11])
      6 print("label ex: ", data_label[10:11])

    number of data:  2808
    number of label:  2808
    data ex:  [array([-548.5122    ,  166.13968   , -66.13242   ,   44.67197   ,
             38.86034   ,  -47.4652    ,   -4.0596566 ,   -8.832043  ,
            -30.090097  ,    3.5134807 ,  -17.342863  ,  -21.36366   ,
             12.159886  ,   -4.4861965 ,   -2.267304  ,    9.768618  ,
            -15.463841  ,    0.7378615 ,   13.42748   ,  -18.990982  ,
             -7.781482  ,    4.8500385 ,  -19.259123  ,   -8.833177  ,
              2.3469172 ,   -8.304173  ,    2.036557  ,   -0.60702324,
            -10.933233  ,    1.5726228 ], dtype=float32)]
    label ex:  [6]
```

تقسیم دیتا ها:

ابتدا دیتا ها را به صورت np array تبدیل کرده و سپس با استفاده از تابع train_test_split:

- ۸۰ درصد به train set اختصاص میدهیم
- از ۲۰ درصد باقی مانده به تقارن بین validation set , test set دیتا ها را تقسیم میکنیم (هر کدام ۱۰ درصد)

```
[15]  1 # converting arrays to np_array
      2
      3 data_np = np.array(data_list)
      4 label_np = np.array(data_label)
```

```
[16]  1 """
      2 cutting the data_set into three portions: train, validation, test
      3 using train_test_split:
      4 first we chop 80% of data for train and put remaining 20% into tmp
      5 then we will chop 50% of tmp (10% of total) into test and the remaining data (10% of total) into valid
      6 """
      7 x_train, x_temp, y_train, y_temp = train_test_split(data_np, label_np, test_size=0.2, random_state=42)
      8 x_test, x_valid, y_test, y_valid = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)
      9
     10 print(len(x_train))
     11 print(len(x_valid))
     12 print(len(x_test))

     2246
     281
     281
```

یعنی:train_set=2246, validation_set=281, test_set=281 نمونه

این تابع در هر مرحله دو دسته خروجی خواهد داشت که دسته اول ارایه ویژگی ها (x) و دسته دوم ارایه لیبل ها(y) هستند.

ساخت مدل (شبکه عصبی)

طبق بحث مطرح شده شبکه عصبی مد نظر شبکه ی پرسپترون چند لایه خواهد بود:

- لایه ورودی: ۳۰ نود: از هر فایل صوت ۳۰ ویژگی استخراج شد که به این لایه داده می شود.
- لایه پنهان ۱: ۱۲۰ نود: لایه پنهان با تابع فعال ساز "relu"
- لایه پنهان۲: ۱۲۰ نود: لایه پنهان با تابع فعال ساز "relu"
- لایه خروجی: ۱۰ نود: اعداد ۰ تا ۹

. و خلاصه ی مدل نمایش داده شده است.

برای اینکه در هر بار اجرا به مدل لایه اضافی داده نشود در خط ۱۵ کد مدل ریست می شود.

```
"""
we will use keras to create our model:

input layer should be 30 nodes since we have 30 features extracted
first hidden layer is 120 nodes
second hidden layer is also 120 nodes
drop out
output layer should be 10 nodes since we have digits 0 to 9

source available in: https://keras.io/api/models/model/#model-class
"""

# clear the [previous] model
# available in: https://stackoverflow.com/questions/52133347/how-can-i-clear-a-model-created-with-keras-and-tensorflowas-backend
tf.keras.backend.clear_session()

# create model
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=(samples,)),
    tf.keras.layers.Dense(120, activation='relu'),
    tf.keras.layers.Dense(120, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)])

# summary method available in: https://keras.io/api/models/model/#summary-method
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 120)               3720
_________________________________________________________________
dense_1 (Dense)              (None, 120)               14520
_________________________________________________________________
dropout (Dropout)            (None, 120)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1210
=================================================================
Total params: 19,450
Trainable params: 19,450
Non-trainable params: 0
_________________________________________________________________
```

کامپایل کردن مدل:

با استفاده از آپتیمایزر adam به دلیل بهینه بودن و تابع SparseCategoricalCrossentropy به دلیل اینکه لیبل ها به مدل one_hot نیستند، مدل را کامپایل می کنیم.

```
# set compile options

"""
defining optimizer:
available in: https://keras.io/api/optimizers/

choosing adam optimizzer:
help: https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/


choosing SparseCategoricalCrossentropy loss function:
https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class
https://keras.io/api/losses/probabilistic_losses/#sparsecategoricalcrossentropy-class
from two fourms above we conclude that:
if we have "one_hot" label representation we should use : CategoricalCrossentropy
and if we have "Integer" labels we should use : SparseCategoricalCrossentropy
here we have Integers

defining accuracy metric
available in: https://keras.io/api/metrics/accuracy_metrics/
"""

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

آموزش مدل:

با epoch=25 و batch_size=3 مدل را اموزش می دهیم.

پارامتر اول داده های استخراج شده در هر ست(x) و پارامتر دوم لیبل های این داده ها(y) هستند.

در هر epoch مقادیر evaluation را نسبت به tran_set , valid_set بدست آورده و ذخیره میکنیم.

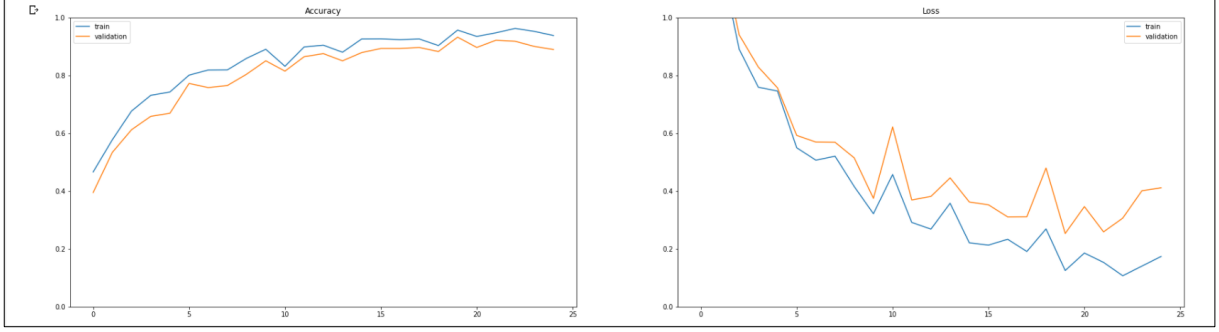
تا در سلول پایین تر بتوانیم نمودار loss , accuracy را رسم کنیم.

```python
# training the model and collection the loss and accuracy in each epoch

epochs = 25

# two list for collecting evaluation data
evals_vaild = [[], []]
evals_train = [[], []]

# traiing model in n epoches and batch_size=3
for i in range(epochs):
    print(i)
    model.fit(x_train, y_train, batch_size=3)
    # collecting evaluations
    evaluate_valid = model.evaluate(x_valid, y_valid)
    evaluate_train = model.evaluate(x_train, y_train)

    evals_vaild[0].append(evaluate_valid[0])
    evals_vaild[1].append(evaluate_valid[1])
    evals_train[0].append(evaluate_train[0])
    evals_train[1].append(evaluate_train[1])

```

نمودار های روند accuracy و loss در هر epoch

Plots

```python
# drawing the plot for accuacy and loss in train and valid

plt.figure(figsize=(30,8))
plt.subplot(1,2,1)
plt.ylim([0,1])
plt.plot(range(epochs),evals_train[1],label="train")
plt.plot(range(epochs),evals_vaild[1],label="validation")
plt.legend()
plt.title("Accuracy")
plt.subplot(1,2,2)
plt.ylim([0,1])
plt.plot(range(epochs),evals_train[0],label="train")
plt.plot(range(epochs),evals_vaild[0],label="validation")
plt.legend()
plt.title("Loss")
plt.show()
```



Evaluate model:

در این قسمت داده های تست روی مدل آزمایش می شوند و مشاهده می کنیم که :

Accuracy=0.903

Loss=0.298

در ادامه به عنوان نمونه یک داده از تست ست به مدل داده شده و با استفاده از تابع np.argmax() ایندکس بالاترین پیش بینی مدل از داده ورودی، ۸ بدست آمده که به درستی با لیبل این داده تست تطابق دارد.

TEST

```
[39]  1 # evaluating model
      2 # available in: https://keras.io/guides/training_with_built_in_methods/
      3
      4 model.evaluate(x_test,  y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 0.2989 - accuracy: 0.9039
[0.2989061772823334, 0.9039145708084106]
```

```
[40]  1 # testing some inputs to see the result
      2 # we can see that usualldy the label found in the second command has the largest value in the first array
      3 arr = model(x_test[40:41])
      4 print(arr)
      5 answer = np.argmax(arr)
      6 print(answer)
      7 print(y_test[40:41])
```

```
tf.Tensor(
[[-16.453619  -20.219076  -20.806492   -2.7242463 -15.747955  -13.049114
    7.0720077 -21.952307   14.995591  -15.626367 ]], shape=(1, 10), dtype=float32)
8
[8]
```

نحوه خواندن داده خروجی:

یک ارایه به طول ۱۰ از طرف مدل به ما بازگردانده می شود که ایندکس بزرگترین داده این ارایه همان پیش بینی شبکه عصبی است. این ایندکس همانطور که پیش تر اشاره شد با استفاده از تابع np.argmax() به ما بازگردانده می شود.

ذخیره کردن مدل شبکه:

با استفاده از سلول اول ما توانستیم شبکه را در مسیر dir و پوشه ی model ذخیره کنیم،

و با استفاده از سلول دوم بعد از اتصال به نوت بوک مدل را لود کنیم.

Saving and Loading model from files

```
[ ]   1 # saving model
      2 # available in: https://keras.io/api/models/model_saving_apis
      3
      4 tf.keras.models.save_model(
      5     model,
      6     filepath=dir + "model",
      7     overwrite=True,
      8     include_optimizer=True,
      9     save_format=None,
     10     signatures=None,
     11     options=None,
     12     save_traces=True,
     13 )
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/CI_files/Spoken_Digits_Recognition/model/assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/CI_files/Spoken_Digits_Recognition/model/assets
```

```
[ ]   1 # loading model
      2 # available in: https://keras.io/api/models/model_saving_apis
      3
      4 model = tf.keras.models.load_model(
      5     filepath=dir + "model", custom_objects=None, compile=True, options=None
      6 )
```