



دانشگاه تهران

دانشکده علوم و فنون نوین

تمرین پنج

نام و نام خانوادگی	فاطمه چیت ساز
شماره دانشجویی	830402092
تاریخ ارسال گزارش	17 دی 1402

Contents

1	تمرین یک: تخمین بیشینه شباهت
3	تمرین دو: روش گشتاور ها
3	تمرین سه: روش گشتاور ها
5	تمرین چهار: مدل تشخیص گفتار با استفاده از پرسپترون چند لایه
17	تمرین پنج: خوشه بندی
19	تمرین شش: k means

تمرین یک: تخمین بیشینه شباهت

قسمت اول:

$$P(x) = \frac{\theta b^\theta}{x^{\theta+1}}$$

$$L(\theta | x_1, \dots, x_n) = \prod_{i=1}^n P(x_i) = \frac{\prod_{i=1}^n \theta b^\theta}{\prod_{i=1}^n x_i^{\theta+1}} \xrightarrow{\text{تابع likelihood}} \frac{\theta^n b^{n\theta}}{\prod_{i=1}^n x_i^{\theta+1}}$$

$$\log L = n \log \theta + \theta n \log b - \sum_{i=1}^n (\theta + 1) \log x_i$$

$$= \frac{n}{\theta} + n \log b - \sum_{i=1}^n \log x_i$$

$$= 0 \rightarrow \theta = \frac{n}{\sum_{i=1}^n \log x_i - n \log b}$$

CS Scanned with CamScanner

قسمت دو و سه:

$P(\lambda|\theta) \propto P(\theta|\lambda)P(\lambda)$

$P(\theta|\lambda) = L(\alpha, \beta|\lambda) = \prod_{i=1}^n (\alpha, \beta|\lambda) = C^n \lambda^{\alpha-1} e^{-\lambda\beta}$

$P(\lambda|\theta) = P(\theta|\lambda)P(\lambda) = (C^n \lambda^{\alpha-1} e^{-\lambda\beta}) C \lambda^{\alpha-1} e^{-\lambda\beta}$

سمت سوم: برای اینکه احتمال پیشین و likelihood مرتبط باشند

باید پیشین و likelihood با هم رابطه داشته باشند و تابع هم نوع باشند

پس λ ← λ (likelihood) و λ ← λ (prior)

سمت سوم: اگر خواستیم MAP را پیدا کنیم ← باید تابع هم نوع تابع پیشین را برای بازسازی داشته باشیم

این انتخاب منطقی است؟ جایی که برابر تابع هم نوع است

$\lambda = \frac{\alpha-1}{\beta} \leftarrow \theta = \frac{\alpha-1}{\beta}$

سمت چهارم:

MAP

www.melabank.ir

Scanned with CamScanner

قسمت چهار:

اگر پارامترهای توزیع پیشین به بی نهایت بزرگ شوند (به سوی بی نهایت)، تخمین گر MAP به تخمین گر ML میل می کند. دلیل این امر این است که تأثیر توزیع پیشین با تعداد نمونه ها متناسب است و با افزایش تعداد نمونه ها، تأثیر توزیع پیشین اهمیت کمتری پیدا می کند و تخمین گر MAP به سمت تخمین گر ML همگرا می شود.

تمرین دو : روش گشتاور ها

تمرین سه : روش گشتاور ها

قسمت اول :

$$\begin{aligned}
 P(x|\theta) &= \frac{1}{\theta} x^{\frac{1-\theta}{\theta}} \\
 \hookrightarrow \prod_{i=1}^n \frac{1}{\theta} x^{\frac{1-\theta}{\theta}} &\rightarrow \frac{1}{\theta^n} \prod_{i=1}^n x^{\frac{1-\theta}{\theta}} \\
 \hookrightarrow \log \frac{1}{\theta^n} \prod_{i=1}^n x^{\frac{1-\theta}{\theta}} &\rightarrow \\
 \log \frac{1}{\theta^n} + (\log \prod_{i=1}^n x_i) \frac{1-\theta}{\theta} &\rightarrow n \log \frac{1}{\theta} + \frac{1-\theta}{\theta} \log \prod_{i=1}^n x_i \\
 \xrightarrow{\text{مشتق}} -\frac{n}{\theta} + \frac{1}{\theta^2} \log \prod_{i=1}^n x_i &= 0 \\
 -\frac{n}{\theta} + \frac{1}{\theta^2} \log \prod_{i=1}^n x_i &= 0 \\
 \theta n = -\log \prod_{i=1}^n x_i \\
 \boxed{\theta = \frac{-\log \prod_{i=1}^n x_i}{n}}
 \end{aligned}$$

قسمت دوم :

$$P(x|\theta) = \frac{1}{\theta} x^{\frac{1-\theta}{\theta}}$$

می دانیم

$$E[x^n] = \int_0^{\infty} x^n p(x|\theta) dx$$

$$E[x^n] = \int_0^{\infty} x^n \frac{1}{\theta} x^{\frac{1-\theta}{\theta}} dx$$

$$= \frac{1}{\theta} \int_0^{\infty} x^{n + \frac{1-\theta}{\theta}} dx \rightarrow \frac{1}{\theta} x^{\frac{1}{n + \frac{1-\theta}{\theta} + 1}} x^{n + \frac{1-\theta}{\theta} + 1} \Big|_0^{\infty}$$

$$= \frac{1}{\theta(n+1)} \rightarrow E[x] = \frac{1}{2\theta}$$

برای محاسبه صابین

تمرین چهار: مدل تشخیص گفتار با استفاده از پرسپترون چند لایه

هدف چیست؟

در این کد، یک مدل شبکه عصبی برای تشخیص گفتار سطح فریم (Frame-Level Speech Recognition) آموزش داده می‌شود. ورودی‌های این مدل اطلاعات MFCC (Mel-Frequency Cepstral Coefficients) برای هر فریم می‌باشند و هدف آن تشخیص فونم‌های حاضر در هر فریم است.

مراحل اجرا:

1. پیش‌پردازش داده: داده‌های آموزش و اعتبارسنجی از دو دسته مجزا از دایرکتوری‌های مختلف برای MFCC و ترنسکریپت‌ها بارگیری می‌شوند. سپس یک مرحله پیش‌پردازش شامل نرمالیزیشن اسپترال بر روی داده‌های MFCC صورت می‌گیرد.

2. ساختار مدل: یک مدل MLP ساده با دو لایه خطی و یک لایه ReLU برای تصمیم‌گیری از معماری استفاده می‌کند.

3. تعریف توابع آموزش و اعتبارسنجی: ** توابع جداگانه برای مراحل آموزش و اعتبارسنجی تعریف شده‌اند. این توابع شامل مراحل انتقال داده به دستگاه GPU (اگر ممکن باشد)، محاسبه خطا، انجام مراحل بهینه‌سازی، و ثبت نتایج هر اپوک می‌شوند.

4. آموزش مدل: مدل با تعداد اپوک‌های مشخص شده آموزش داده می‌شود و نتایج مربوط به هر اپوک چاپ می‌شوند.

5. گزارش و نتیجه‌گیری: گزارش کاملی از نتایج آموزش و اعتبارسنجی ارائه شده و به توضیحات و جزئیات هر بخش پرداخته می‌شود.

هدف پروژه: هدف اصلی این پروژه بهینه‌سازی یک مدل برای تشخیص گفتار بر اساس اطلاعات MFCC و ترنسکریپت‌هاست تا مدل بتواند بهترین عملکرد را در تعیین فونم‌های هر فریم از ورودی ارائه دهد.

خب حال میتونیم به توضیح کد بپردازیم :

ابتدا کتابخانه‌های لازم نصب و وارد می‌شوند.

```

import torch
import numpy as np
from torchsummaryX import summary
import sklearn
import gc
import zipfile
import pandas as pd
from tqdm.auto import tqdm
import os
import datetime
import wandb

```

البته لازم به ذکره چندتا از کتابخونه ها را با pip نصب میکنیم

یک لیست از فونمها تعریف می شود

```

### PHONEME LIST
PHONEMES = [
    '[SIL]', 'AA', 'AE', 'AH', 'AO', 'AW', 'AY',
    'B', 'CH', 'D', 'DH', 'EH', 'ER', 'EY',
    'F', 'G', 'HH', 'IH', 'IY', 'JH', 'K',
    'L', 'M', 'N', 'NG', 'OW', 'OY', 'P',
    'R', 'S', 'SH', 'T', 'TH', 'UH', 'UW',
    'V', 'W', 'Y', 'Z', 'ZH', '[SOS]', '[EOS]'
]

```

به گوگل درایو وصل میشیم برای دیتاست بزرگ عزیزرز

```

# If you are using colab, you can import google drive to save model checkpoints in a folder
from google.colab import drive
drive.mount('/content/drive')

```

Mounted at /content/drive

دیتاست بزرگوار رو دانلود میکنیم

```
!gdown 1gNVILsmbYZbYJQk2-NN2q7VG2RKJp-7x
```

Downloading...

From: <https://drive.google.com/uc?id=1gNVILsmbYZbYJQk2-NN2q7VG2RKJp-7x>

To: /content/Data.zip

100% 4.28G/4.28G [00:34<00:00, 123MB/s]

میداریمش تو پوشه content/data و ان زیپ میفرماییم


```
!unzip -qo /content/Data.zip -d '/content/data'
```

در این بخش کلاس مجموعه داده برای داده‌های گفتار تعریف می‌شود

خب بیاید ببینیم این لودرمون چه ها می‌کنه

مسئولیت اول: تعیین دو دایرکتوری برای داده‌ها. یکی برای MFCC و دیگری برای ترنسکرپت. این دو

دایرکتوری با استفاده از root و partition مشخص می‌شن. مثل اینکه که می‌گیم "داده‌ها اینجا هستن!"

مسئولیت دوم: لیستی از نام فایل‌ها رو به ترتیب می‌گیریم. اول از همه، نام فایل‌های MFCC و ترنسکرپت رو مرتب می‌کنیم.

مسئولیت سوم: محاسبه میانگین و انحراف معیار برای هر MFCC. این یه جورایی یه کار تنظیمیه که داده‌ها رو به یه حالت خوب برسونه.

مسئولیت چهارم: تعداد کل فریم‌ها محاسبه میشه و مطمئن میشیم که تعداد فایل‌ها یکسانه.

مسئولیت پنجم: یک ماتریس برای MFCC و یک آرایه برای ترنسکرپت ایجاد میشه. اینجا داده‌ها به ماتریس و آرایه مربوطه اضافه می‌شن.

مسئولیت ششم: تابع getitem تعریف می‌کنیم که یک بخش از داده رو به عنوان ورودی برای مدل میاره. اینجا با مراعات فریم‌های اطراف و با توجه به شماره فریم، یک زوج از داده‌های MFCC و ترنسکرپت برمی‌گردونه.

```
# Dataset class to load train and validation data
class AudioDataset(torch.utils.data.Dataset):

    def __init__(self, root, phonemes = PHONEMES, context=0, partition= "train-clean-100"): # Feel free to add more arguments

        self.context = context
        self.phonemes = phonemes

        # TODO: MFCC directory - use partition to acces train/dev directories from kaggle data using root
        self.mfcc_dir = root + '/' + partition + '/mfcc/'
        # TODO: Transcripts directory - use partition to acces train/dev directories from kaggle data using root
        self.transcript_dir = root + '/' + partition + '/transcript/'

        # TODO: List files in self.mfcc_dir using os.listdir in sorted order
        mfcc_names = sorted(os.listdir(self.mfcc_dir))
        # TODO: List files in self.transcript_dir using os.listdir in sorted order
        transcript_names = sorted(os.listdir(self.transcript_dir))

        # Making sure that we have the same no. of mfcc and transcripts
        total_timestamps = 0
        assert len(mfcc_names) == len(transcript_names)
        for mfcc in mfcc_names:
            total_timestamps += len(np.load(self.mfcc_dir + mfcc))
        self.length = total_timestamps
        print(total_timestamps)

        print("HERE")
        self.mfccs, self.transcripts = np.zeros((2*context+total_timestamps, 28), dtype=np.float32), np.zeros((total_timestamps), dtype=np.uint8)
        #self.mfccs, self.transcripts = [], []
        # TODO: Iterate through mfccs and transcripts
        current_index = context
        for i in range(len(mfcc_names)):
            # Load a single mfcc
            mfcc = np.load(self.mfcc_dir + mfcc_names[i])
```

خب مرحله بعد ست کردن تنظیماته :

این تیکه از کد یک فایل تنظیمات به نام config ایجاد می‌کنه که تمام پارامترها و تنظیمات مربوط به آموزش یک مدل صوتی رو در خودش نگه می‌داره. مثلاً تعداد اپوک‌ها (تعداد دوره‌های آموزش)، اندازه دسته (تعداد نمونه‌ها در هر مرحله)، محدوده متن (تعداد فریم‌های اطراف هر فریم) و نرخ یادگیری اولیه از جمله پارامترهای مهم هستند.

در واقع مثل اینه یک دفترچه یادداشت داشته باشیم که تمام جزئیات لازم برای هر آزمایش رو در اختیار داریم.

Parameters Configuration

Storing your parameters and hyperparameters in a single configuration dictionary makes it easier to keep track of them during each experiment. It can also be used to log your parameters for each experiment and keep track of them across multiple experiments.

```
config = {
    'epochs': 5,
    'batch_size': 1024,
    'context': 20,
    'init_lr': 1e-3,
    'architecture': 'very-low-cutoff'
    # Add more as you need them - e.g dropout values, weight decay, scheduler parameters
}
```

و بالاخره درست کردن دیتاها

1. ساخت آبجکت دیتاست:

این خط یک آبجکت از کلاس AudioDataset با نام train_data ایجاد می‌کند.

پارامتر اول آدرس دایرکتوری حاوی داده‌های آموزشی است.

پارامتر دوم مقدار context را از دیکشنری config دریافت می‌کند. context مشخص می‌کند که چند فریم قبل و بعد از هر فریم هدف باید برای آموزش مدل در نظر گرفته شود.

```
#TODO: Create a dataset object using the AudioDataset class for the training data
train_data = AudioDataset('/content/data/11-785-f23-hw1p2', context = config['context'])
```

2. ساخت آبجکت دیتاست برای اعتبارسنجی:

مشابه train_data، اما از پارتیشن dev-clean برای بارگیری داده‌های اعتبارسنجی استفاده می‌کند.

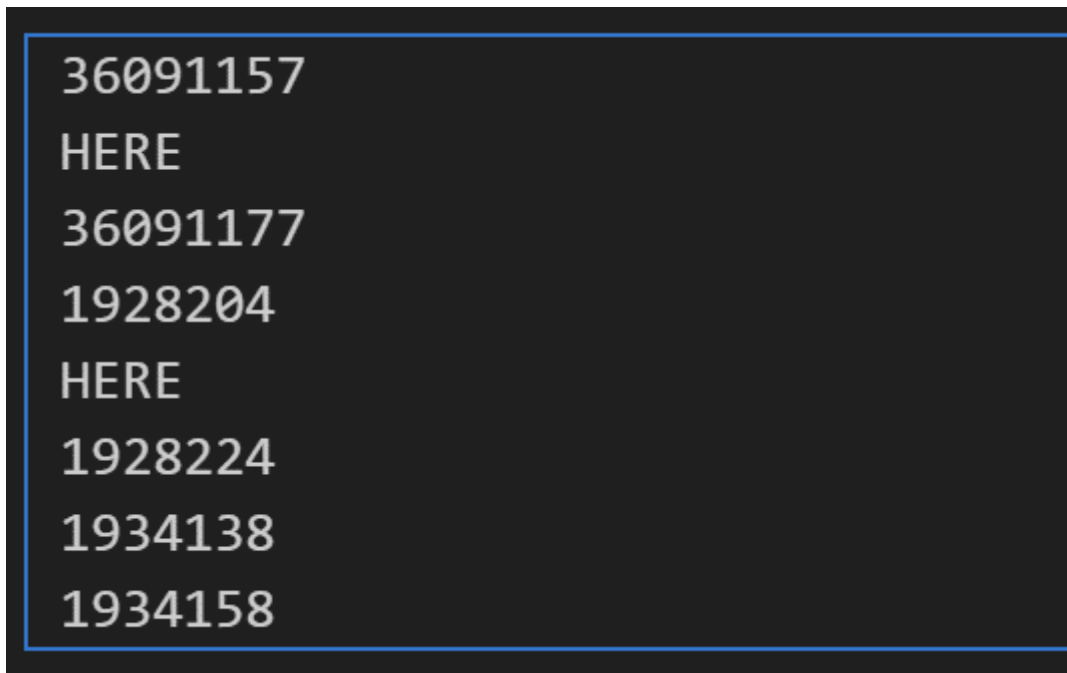
```
# TODO: Create a dataset object using the AudioDataset class for the validation data
val_data = AudioDataset('/content/data/11-785-f23-hw1p2', context = config['context'], partition='dev-clean')
```

3. ساخت آبجکت دیتاست برای تست:

در اینجا یک شی از کلاس AudioTestDataset برای داده‌های تست ایجاد شده است. همچنین از مسیر دیتاست و مقدار context از فایل تنظیمات config استفاده شده‌اند. این دیتاست مربوط به قسمت تست (partition='test-clean') است.

```
# TODO: Create a dataset object using the AudioTestDataset class for the test data
test_data = AudioTestDataset('/content/data/11-785-f23-hw1p2', context = config['context'], partition='test-clean')
```

خروجی :



```
36091157
HERE
36091177
1928204
HERE
1928224
1934138
1934158
```

اینجا دوتا چیز اول برای ترین بعد validation و بعد تست چاپ میشه
:current_index

این متغیر نشان‌دهنده‌ی اندیس فریم فعلی در آرایه‌ی self.mfccs است
در ابتدای حلقه‌ی for مقدار آن context است

با پیمایش حلقه و اضافه کردن فریم‌های جدید به آرایه، مقدار آن افزایش می‌یابه
در نهایت، مقدار آن برابر با اندیس آخرین فریم به همراه فریم‌های زمینه‌ای هس
:total_timestamps

این متغیر نشان‌دهنده‌ی تعداد کل فریم‌های صوتی در دیتاست است
با جمع کردن طول هر فایل MFCC به دست میاد

خب حالا می‌خوایم داده‌ها رو دسته‌بندی کنیم تا مدل یادگیری ماشینی مون بتونه راحت‌تر باهاشون کار کنه.

این کدها یه چیزی درست می‌کنن به اسم DataLoader که وظیفه‌ش اینه که داده‌های آموزشی، اعتبارسنجی و تست رو برامون دسته‌بندی کنه.

چیکار می‌کنه:

برای هر کدوم از سه تا دیتاست، یه DataLoader می‌سازه:

یکی برای داده‌های آموزشی: train_loader

یکی برای داده‌های اعتبارسنجی: val_loader

یکی برای داده‌های تست: test_loader

به DataLoader می‌گه که:

از کدوم دیتاست استفاده کنه

چند تا کارگر بذاره برای بارگذاری داده‌ها

هر دسته چقدر نمونه داشته باشه

داده‌های آموزشی رو قاطی کنه

بعدش یه سری اطلاعات چاپ می‌کنه:

اندازه‌ی دسته‌ها

تعداد فریم‌های قبلی و بعدی که مدل برای پیش‌بینی هر فریم در نظر می‌گیره (مثل اینکه بهش بگی قبل

از اینکه پیش‌بینی کنی، به چند تا فریم قبلی و بعدی نگاه کن)

اندازه‌ی ورودی مدل (تعداد کل ویژگی‌هایی که مدل باید یاد بگیره)

تعداد نمادهای خروجی

تعداد نمونه‌ها و دسته‌ها توی هر دیتاست

```

train_loader = torch.utils.data.DataLoader(
    dataset      = train_data,
    num_workers  = 4,
    batch_size   = config['batch_size'],
    pin_memory   = True,
    shuffle      = True
)

val_loader = torch.utils.data.DataLoader(
    dataset      = val_data,
    num_workers  = 2,
    batch_size   = config['batch_size'],
    pin_memory   = True,
    shuffle      = False
)

test_loader = torch.utils.data.DataLoader(
    dataset      = test_data,
    num_workers  = 2,
    batch_size   = config['batch_size'],
    pin_memory   = True,
    shuffle      = False
)

print("Batch size      : ", config['batch_size'])
print("Context         : ", config['context'])
print("Input size       : ", (2*config['context']+1)*28)
print("Output symbols  : ", len(PHONEMES))

print("Train dataset samples = {}, batches = {}".format(train_data.__len__(), len(train_loader)))
print("Validation dataset samples = {}, batches = {}".format(val_data.__len__(), len(val_loader)))
print("Test dataset samples = {}, batches = {}".format(test_data.__len__(), len(test_loader)))

```

خروجی :

```

Batch size      : 1024
Context         : 20
Input size      : 1148
Output symbols  : 42
Train dataset samples = 36091157, batches = 35246
Validation dataset samples = 1928204, batches = 1884
Test dataset samples = 1934138, batches = 1889

```

حالا تست کوچیک انجام می‌دیم تا مطمئن شیم که DataLoaderها درست کار می‌کنه.

کار کد اینه که:

یه دور می‌زنه روی اولین دسته از داده‌های آموزشی:

```

for i, data in enumerate(train_loader):

```

داده‌های هر دسته رو باز می‌کنه:

```

frames, phoneme = data

```

شکل داده‌های صدا و برچسب‌ها رو چاپ می‌کنه:

```
print(frames.shape, phoneme.shape)
```

بعد از اولین دسته دیگه ادامه نمی‌ده:

```
break
```

خروجی :

```
torch.Size([1024, 1148]) torch.Size([1024])
```

خب حال شبکه عصبی بزرگوار رو تعریف کنیم :

این شبکه یه ورودی می‌گیره (یه سری عدد که ویژگی‌های یه صدا رو نشون می‌دن)

از یه سری لایه‌های خطی و ReLU ردش می‌کنه (مثل اینکه یه سری فیلتر و پردازش روش انجام می‌ده)

یه خروجی تولید می‌کنه

```
# This architecture will make you cross the very low cutoff
# However, you need to run a lot of experiments to cross the medium or high cutoff
class Network(torch.nn.Module):
    def __init__(self, input_size, output_size):
        super(Network, self).__init__()

        self.model = torch.nn.Sequential(
            torch.nn.Linear(input_size, 512),
            torch.nn.ReLU(),
            torch.nn.Linear(512, output_size)
        )

    def forward(self, x):
        out = self.model(x)

        return out
```

حالا چندتا کار جالب کنیم قبل ترین کردن :

اندازه‌ی ورودی رو مشخص می‌کنن:

$INPUT_SIZE = (2 * config['context'] + 1) * 28$

اینجوری حسابش می‌کنن:

هر فریم صوتی 28 تا ویژگی داره (مثلاً 28 تا عدد).

می‌خوان برای پیش‌بینی هر فریم، علاوه بر خودش، $config['context']$ فریم قبلی و $config['context']$ فریم بعدی رو هم در نظر بگیرن.

پس در کل $(config['context'] + 1) * 2$ فریم رو با هم ترکیب می‌کنن.

به همین خاطر، اندازه‌ی ورودی میشه حاصل ضرب تعداد فریم‌ها $(config['context'] + 1) * 2$ در تعداد ویژگی‌های هر فریم (28).

```
INPUT_SIZE = (2*config['context'] + 1) * 28 # Why is this the case?
```

مدل رو می‌سازن و می‌فرستن روی کارت گرافیک (اگه موجود باشه):

مدل رو با اندازه‌ی ورودی مشخص‌شده و تعداد نمادهای خروجی می‌سازن.

بعدش هم می‌فرستنش روی کارت گرافیک تا سریع‌تر کار کنه.

```
model = Network(INPUT_SIZE, len(train_data.phonemes)).to(device)
```

خلاصه‌ای از مدل رو نشون می‌دن:

```
summary(model, frames.to(device))
```

این تابع ساختار مدل و تعداد پارامترهایش رو نشون می‌ده.

یه نمونه از داده‌ها رو هم بهش می‌دن تا بتونه محاسباتش رو انجام بده.

خروجی:

```
=====
              Kernel Shape Output Shape    Params Mult-Adds
Layer
0_model.Linear_0 [1148, 512] [1024, 512]  588.288k  587.776k
1_model.ReLU_1   -          [1024, 512]    -         -
2_model.Linear_2 [512, 42] [1024, 42]   21.546k   21.504k
-----
Totals
Total params      609.834k
Trainable params  609.834k
Non-trainable params  0.0
Mult-Adds         609.28k
```


تعریف تابع خطا:

که صد البته میزان اختلاف بین پیش‌بینی مدل و جواب درست رو اندازه می‌گیره.

```
criterion = torch.nn.CrossEntropyLoss() # Defining Loss function.  
# We use CE because the task is multi-class classification
```

تعریف بهینه‌ساز:

بهینه‌ساز به مدل کمک می‌کنه که با توجه به تابع خطا، پارامتراش رو تنظیم کنه و یاد بگیره.

اینجا از بهینه‌ساز Adam استفاده می‌کنن که معمولاً عملکرد خوبی داره.

lr هم نرخ یادگیری که سرعت یادگیری مدل رو کنترل می‌کنه.

```
optimizer = torch.optim.Adam(model.parameters(), lr= config['init_lr']) #Defining Optimizer  
# Recommended : Define Scheduler for Learning Rate,  
# including but not limited to StepLR, MultiStepLR, CosineAnnealingLR, ReduceLROnPlateau, etc.  
# You can refer to Pytorch documentation for more information on how to use them.  
  
# Is your training time very high?  
# Look into mixed precision training if your GPU (Tesla T4, V100, etc) can make use of it  
# Refer - https://pytorch.org/docs/stable/notes/amp\_examples.html
```

چندتا بهینه سازی و مراقبت

```
torch.cuda.empty_cache()
```

حافظه‌ی کش کارت گرافیک رو خالی می‌کنه.

این کار باعث می‌شه که فضا برای محاسبات جدید باز بشه و عملکرد کارت گرافیک بهتر بشه.

```
gc.collect()
```

زباله‌های حافظه‌ی اصلی رو جمع‌آوری می‌کنه.

این کار باعث می‌شه که حافظه‌ی اصلی خالی بشه و از کمبود حافظه جلوگیری بشه.

حالا میریم سراغ ترین و این داستانامون :

تابع train:

مدل در حالت آموزش قرار می گیرد.

برای هر دسته از داده های آموزشی:

گرادیان ها صفر می شوند.

داده ها به کارت گرافیک منتقل می شوند (در صورت وجود).

پیش بینی انجام می شود.

خطا محاسبه می شود.

گرادیان ها محاسبه می شوند.

پارامترهای مدل به روزرسانی می شوند.

خطا و دقت محاسبه و نمایش داده می شوند.

حافظه آزاد می شود.

```
def train(model, dataloader, optimizer, criterion):

    model.train()
    tloss, tacc = 0, 0 # Monitoring loss and accuracy
    batch_bar = tqdm(total=len(train_loader), dynamic_ncols=True, leave=False, position=0, desc='Train')

    for i, (frames, phonemes) in enumerate(dataloader):

        ### Initialize Gradients
        optimizer.zero_grad()

        ### Move Data to Device (Ideally GPU)
        frames = frames.to(device)
        phonemes = phonemes.to(device)

        ### Forward Propagation
        logits = model(frames)

        ### Loss Calculation
        loss = criterion(logits, phonemes)

        ### Backward Propagation
        loss.backward()

        ### Gradient Descent
        optimizer.step()

        tloss += loss.item()
        tacc += torch.sum(torch.argmax(logits, dim=1) == phonemes).item()/logits.shape[0]

        batch_bar.set_postfix(loss="{:.04f}".format(float(tloss / (i + 1))),
                               acc="{:.04f}%".format(float(tacc*100 / (i + 1))))
        batch_bar.update()
```

تابع eval:

مدل در حالت ارزیابی قرار می گیرد.

برای هر دسته از داده‌های اعتبارسنجی:

داده‌ها به کارت گرافیک منتقل می‌شوند.

پیش‌بینی انجام می‌شود (بدون محاسبه گرادیان).

خطا محاسبه می‌شود.

خطا و دقت محاسبه و نمایش داده می‌شوند.

حافظه آزاد می‌شود.

```
def eval(model, dataloader):
    model.eval() # set model in evaluation mode
    vloss, vacc = 0, 0 # Monitoring loss and accuracy
    batch_bar = tqdm(total=len(val_loader), dynamic_ncols=True, position=0, leave=False, desc='Val')

    for i, (frames, phonemes) in enumerate(dataloader):
        ### Move data to device (ideally GPU)
        frames = frames.to(device)
        phonemes = phonemes.to(device)

        # makes sure that there are no gradients computed as we are not training the model now
        with torch.inference_mode():
            ### Forward Propagation
            logits = model(frames)
            ### Loss Calculation
            loss = criterion(logits, phonemes)

            vloss += loss.item()
            vacc += torch.sum(torch.argmax(logits, dim=1) == phonemes).item() / logits.shape[0]

        # Do you think we need loss.backward() and optimizer.step() here?

        batch_bar.set_postfix(loss="{:.04f}".format(float(vloss / (i + 1))),
                               acc="{:.04f}%".format(float(vacc*100 / (i + 1))))
        batch_bar.update()

        ### Release memory
        del frames, phonemes, logits
        torch.cuda.empty_cache()

    batch_bar.close()
    vloss /= len(val_loader)
```

حلقه‌ی اصلی آموزش:

برای تعداد مشخص‌شده‌ی اپاکا:

نرخ یادگیری فعلی نمایش داده می‌شود.

مدل روی داده‌های آموزشی آموزش داده می‌شود.

مدل روی داده‌های اعتبارسنجی ارزیابی می‌شود.

```
# Iterate over number of epochs to train and evaluate your model
torch.cuda.empty_cache()
gc.collect()

# wandb.watch(model, log="all")

for epoch in range(config['epochs']):

    print("\nEpoch {}/{}".format(epoch+1, config['epochs']))

    curr_lr = float(optimizer.param_groups[0]['lr'])
    train_loss, train_acc = train(model, train_loader, optimizer, criterion)
    val_loss, val_acc = eval(model, val_loader)

    print("\tTrain Acc {:.04f}%\tTrain Loss {:.04f}\t Learning Rate {:.07f}".format(train_acc*100, train_loss, curr_lr))
    print("\tVal Acc {:.04f}%\tVal Loss {:.04f}".format(val_acc*100, val_loss))

    ### Log metrics at each epoch in your run
    # Optionally, you can log at each batch inside train/eval functions
    # (explore wandb documentation/wandb recitation)
    # wandb.log({'train_acc': train_acc*100, 'train_loss': train_loss,
    #           'val_acc': val_acc*100, 'valid_loss': val_loss, 'lr': curr_lr})

    ### Highly Recommended: Save checkpoint in drive and/or wandb if accuracy is better than your current best

### Finish your wandb run
#run.finish()
```

خروجی :

```
Epoch 1/5

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"version_major":2,"version_minor":0,"model_id":"c9851a6ab0934d37a087745cba8ed836"}

/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 work
warnings.warn(_create_warning_msg(

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"version_major":2,"version_minor":0,"model_id":"f4d47cda451b41e8ba97d05206feaae7"}

Train Acc 69.7034%    Train Loss 0.9805    Learning Rate 0.0010000
Val Acc 68.8041%    Val Loss 1.0077

Epoch 2/5

Could not render content for 'application/vnd.jupyter.widget-view+json'
{"version_major":2,"version_minor":0,"model_id":"9b486d512d154d098532065ccb79d8b6"}

```

تمرین پنج: خوشه بندی

قسمت اول :

افزایش ϵ :

با افزایش اندازه شعاع همسایگی، نقاط بیشتری در همسایگی یک نقطه قرار می گیرند. این می تواند منجر به افزایش تعداد نقاط در یک خوشه شود و در نتیجه کاهش تعداد خوشه ها. همچنین، با افزایش ϵ ، تعداد نقاطی که به عنوان نویز شناخته می شوند کاهش می یابد، زیرا نقاط بیشتری قادر به رسیدن به حداقل تعداد نقاط مورد نیاز (MinPts) در شعاع ϵ خود هستند.

کاهش ϵ : با کاهش اندازه شعاع همسایگی، کمتر نقطه در همسایگی یک نقطه قرار می‌گیرد. این می‌تواند منجر به کاهش تعداد نقاط در یک خوشه شود و در نتیجه افزایش تعداد خوشه‌ها. همچنین، با کاهش ϵ ، تعداد نقاطی که به عنوان نویز شناسایی می‌شوند افزایش می‌یابد، زیرا کمتر نقطه قادر به رسیدن به حداقل تعداد نقاط مورد نیاز (MinPts) در شعاع ϵ خود هستند.

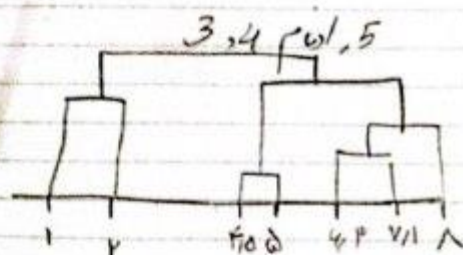
قسمت دوم :



X
 5
 2
 4, 1, 2
 8
 4, 5
 1
 7, 1

تاریخ:
 موضوع:

1. هر داده را به 5، 4، 2، 1 و 7 دسته بندی می‌کنیم
 2. هر دسته را به 5، 4، 2، 1 و 7 دسته بندی می‌کنیم
 3. هر دسته را به 5، 4، 2، 1 و 7 دسته بندی می‌کنیم



$$a = 1 = 2 - 1$$

$$b = 1 = 2 - 1$$

$$b = 1, 1$$

$$S = \frac{b - a}{\max(a, b)} = \frac{1 - 1}{1, 1} = 0, 0$$

تمرین شش: k means

قسمت یک:

اثبات همگرایی:

برای اثبات همگرایی الگوریتم K-means، باید نشان دهیم که تابع هزینه الگوریتم در هر تکرار کاهش می‌یابد. تابع هزینه الگوریتم K-means مجموع مربعات فاصله هر نقطه داده از مرکز خوشه آن است.

مرحله 1:

در مرحله انتساب، هر نقطه داده به نزدیک‌ترین مرکز خوشه اختصاص داده می‌شود. این امر باعث می‌شود که تابع هزینه کاهش یابد، زیرا هر نقطه داده به خوشه‌ای اختصاص داده می‌شود که مرکز آن به آن نقطه نزدیک‌تر است.

مرحله 2:

در مرحله به‌روزرسانی، مرکز هر خوشه به عنوان میانگین نقاط داده‌ای که به آن خوشه اختصاص داده شده‌اند، محاسبه می‌شود. این امر نیز باعث می‌شود که تابع هزینه کاهش یابد، زیرا مرکز جدید خوشه به طور متوسط به تمام نقاط داده‌ای که به آن خوشه اختصاص داده شده‌اند، نزدیک‌تر است.

نتیجه:

از آنجایی که تابع هزینه در هر تکرار الگوریتم K-means کاهش می‌یابد، و تابع هزینه یک تابع کراندار است، الگوریتم K-means باید همگرا شود.

قسمت دو :

به دلیل ماهیت ناحیه‌های متمرکز و ناحیه‌های با چگالی کمتر، احتمالاً در نهایت مراکز خوشه‌ها به صورت یکنواخت بین دو ناحیه توزیع نمی‌شوند. اگر دو ناحیه جدا از یکدیگر هستند و داده‌ها در هر ناحیه به صورت تقریباً یکنواخت توزیع شده‌اند، مراکز خوشه‌ها نیز به یکنواختی در امتداد هر ناحیه توزیع خواهند شد

اگر k کمتر از تعداد واقعی خوشه‌ها باشد، مراکز خوشه‌ها به طور مساوی بین دو ناحیه توزیع نمی‌شوند. به احتمال زیاد مراکز بیشتری در ناحیه متمرکز با چگالی داده بیشتر وجود خواهد داشت.

اگر k بیشتر از تعداد واقعی خوشه‌ها باشد، ممکن است مراکز خوشه‌ها در هر دو ناحیه به طور مساوی توزیع شوند.

ی سری چیز مهمه :

معیار فاصله:

انتخاب معیار فاصله مناسب می تواند بر توزیع مراکز خوشه ها تأثیر بگذارد.

پارامترهای دیگر:

پارامترهای دیگر الگوریتم K-means مانند epsilon (معیار توقف) نیز می توانند بر توزیع مراکز خوشه ها تأثیر بگذارند.

در نهایت، بدون اطلاعات بیشتر در مورد مجموعه داده خاصمون و پارامترهای انتخابی الگوریتم K-means، نمی توان با قاطعیت گفت که مراکز خوشه ها در نهایت چگونه توزیع می شوند.

قسمت سه :

این فرمول به این معنی است که نقاط داده ای که از مراکز موجود دورتر هستند، احتمال بیشتری برای انتخاب شدن به عنوان مرکز بعدی دارند.

این روش مشابه الگوریتم ++K-means است که هدف آن ارائه یک نقطه شروع بهتر برای K-means است تا منجر به خوشه بندی بهینه تری شود.

هنگام به روزرسانی مراکز (در مراحل بعدی K-means)، این روش دوباره مراکز جدید را بر اساس همان فرمول احتمال انتخاب می کند. این یک تفاوت با K-means استاندارد است که در آن مراکز جدید معمولاً میانگین نقاط داده در خوشه انتخاب می شود. این نوع الگوریتم به نظر می رسد هنگام به روزرسانی مراکز، وزن بیشتری به نقاط داده ای می دهد که از مرکز فعلی دورتر هستند.

الگوریتم ++K-means به طور معمول بهبودهایی در سرعت همگرایی به نسبت K-means ساده دارد. این بهبود اصلی به دلیل انتخاب هوشمندانه تر مراکز اولیه است.

در ++K-means، انتخاب مراکز بر اساس احتمالات صورت می گیرد تا از ایجاد خوشه های شروع با فاصله بسیار بزرگ یا بسیار کوچک جلوگیری شود. این انتخاب هوشمندانه مراکز، معمولاً منجر به کاهش تعداد مراکز به روز شده و در نتیجه به سرعت بهبود می بخشد.

در K-means ساده، انتخاب تصادفی مراکز اولیه ممکن است باعث ایجاد شروع نامناسبی برای خوشه بندی شود، که موجب نیاز به تعداد بیشتری تکرار (iteration) برای همگرایی مطلوب می شود.

بنابراین، کلیتاً می توان گفت که ++K-means ممکن است به سرعت بهتری در همگرایی نسبت به K-means ساده داشته باشد، به خصوص در مواقعی که داده ها حجم بالایی دارند.

قسمت چهار:

عیار فاصله کوسینوسی:

معیار فاصله کوسینوسی شباهت بین دو بردار را بر اساس زاویه بین آنها اندازه گیری می کند.

این معیار به مقیاس داده ها حساس نیست، که می تواند یک مزیت در برخی موارد باشد.

مناسب بودن برای داده های سری زمانی:

معیار فاصله کوسینوسی برای داده های سری زمانی که ترتیب زمانی داده ها مهم است، مناسب نیست.

دلیل این امر این است که معیار فاصله کوسینوسی به ترتیب زمانی داده ها توجهی نمی کند.

معیارهای مناسب تر:

معیار فاصله اقلیدسی: این معیار فاصله بین دو نقطه را به عنوان طول کوتاهترین خط مستقیم بین آنها

اندازه گیری می کند.

معیار فاصله دینامیک زمانی (DTW): این معیار فاصله بین دو سری زمانی را با در نظر گرفتن ترتیب زمانی داده ها اندازه گیری می کند.

انتخاب معیار مناسب:

انتخاب معیار مناسب به نوع داده ها و هدف از خوشه بندی بستگی دارد.

برای داده های سری زمانی، معیارهای فاصله اقلیدسی یا DTW مناسب تر از معیار فاصله کوسینوسی هستند.

مثال:

فرض کنید دو سری زمانی داریم که هر کدام 10 نقطه داده دارند.

اگر داده ها به طور تصادفی مرتب شوند، معیار فاصله کوسینوسی ممکن است شباهت زیادی بین دو سری زمانی گزارش دهد.

با این حال، اگر داده ها بر اساس ترتیب زمانی مرتب شوند، معیار فاصله اقلیدسی یا DTW ممکن است شباهت کمتری بین دو سری زمانی گزارش دهد.

نتیجه:

استفاده از معیار فاصله کوسینوسی برای داده های سری زمانی مناسب نیست.
معیارهای فاصله اقلیدسی یا DTW برای داده های سری زمانی مناسب تر هستند.