به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

مدل‌های مولد عمیق

تمرین شماره 3-part2

نام و نام خانوادگی	مهسا ندافی قهنویه
شماره دانشجویی	۸۱۰۱۰۰۴۹۰
تاریخ ارسال گزارش	۱۴۰۲/۱۱/۱۷

فهرست گزارش سوالات

سوال ۳ – Speech synthesis ۳

سوال ۱ – Speech synthesis

برای دانلود دیتاست به کار رفته در این سوال از مجوعه دیتاست های سایت huggingface استفاده شده :

```
persian_train_dataset = load_dataset("mozilla-foundation/common_voice_13_0", "fa", split="train", use_auth_token=True)
persian_test_dataset = load_dataset("mozilla-foundation/common_voice_13_0", "fa", split="test", use_auth_token=True)
```

برای آموزش کدل از ۱۵۰۰۰ نمونه از دیتاست آموزش معادل ۱۰۰۰ ساعت استفاده شده است و برای تست مدل بصورت رندوم در هر ایپاک از ۳ دیتای تست استفاده شده است.یک نمونه از دیتاست آموزش خام در زیر نشان داده شده است:

{'client_id': 'f07716fdbad07f44e58a9597e164df11e5d8bbe99a8f3ec384f6654d0f91aa55f0603e14b2c3c7f20f37da41784b20587a08f20cb6c434efffff9904a5977e72f',

'path': '/root/.cache/huggingface/datasets/downloads/extracted/85450b3d404f174436bdb1a6045423e59f6032eb7b4209ebcd88e5d67d4e7c54/fa_train_0/common_voice_fa_30206283.mp3',

'audio':{'path': '/root/.cache/huggingface/datasets/downloads/extracted/85450b3d404f174436bdb1a6045423e59f6032eb7b4209ebcd88e5d67d4e7c54/fa_train_0/common_voice_fa_30206283.mp3',

'array': array([1.77635684e-14, -2.48689958e-14, 6.75015599e-14, ..., 2.67317664e-05, 3.16987680e-05, 2.79086235e-05]),

'sampling_rate': 16000},

'sentence': 'او خود را نزد خویشاوندان پولدار خود خوار و خفیف کرد.',

'up_votes': 2,

'down_votes': 0,

'age': '',

'gender': '',

'accent': '',

'locale': 'fa',

'segment': '',

'variant': ''}

- پیش شبکه و پس شبکه

در ابتدا مجموعه کاراکتر های فارسی به توکن های توکنایزر مدل اضافه شده است.با استفاده از توکنایزر کاراکترهای جملات به input id تبدیل میشوند تا به مدل داده شوند.

```
#add persian charecter to tokenizer
persian_characters = ['ع', 'ف', 'ط', 'ر', 'ل', 'پ', '«', ' ', 'ض', 'ظ',
'ج', 'خ', 'چ', 'ؤ', 'ـ', 'د', 'ً', 'ی', '', 'ئ', 'ٔ', 'ے', 'ة', '“', 'غ',
'ء', 'أ', 'م', '”', 'ش', 'ق', 'ُ', 'و', 'ت', 'ه', 'ِ', 'ب', '؟', 'ذ', 'ي',
'ک', 'ح', 'ص', 'ك', 'آ', '»', 'ز', '،', 'ا', 'ن', '؛', '=', '-', 'س', 'گ',
'ث', 'ی', '', 'ژ']
tokenizer.add_tokens(list(persian_characters))
model.resize_token_embeddings(len(tokenizer))
```

یک نمونه id برای یک جمله از دیتاست:

```
Sample : اشاره کرد که دنبالش بروم.

Ids: [131, 111, 131, 84, 116, 88, 123, 84, 96, 88, 123,
116, 88, 96, 132, 119, 131, 85, 111, 88, 119, 84, 114,
109, 4, 26, 2]
```

قسمت speaker embedding به صورت زیر پیاده شده است در واقع خروجی این تابع بردارد تعبیه گوینده برای یادگیری بازنمایی از هویت و ویژگی های گوینده است که مدل توانایی یادگیری آن را دارد:

```
def speaker_embedding_generator(waveform):
    with torch.no_grad():
        speaker_embeddings =
speaker_model.encode_batch(torch.tensor(waveform))
        speaker_embeddings =
torch.nn.functional.normalize(speaker_embeddings, dim=2)
        speaker_embeddings = speaker_embeddings.squeeze()
    return speaker_embeddings
```

دیتاست آموزش با استفاده از مراحل زیر تغییر یافته است:

باید دقت شود که نرخ نمونه برداری ۱۶۰۰۰ می باشد.همچنین input features شامل نمودار اسپکتوگرامی می باشد که در فرایند آموزش مدل نیاز داریم و از سیگنال صوتی هر نمونه توسط feature extractor استخراج میشود.

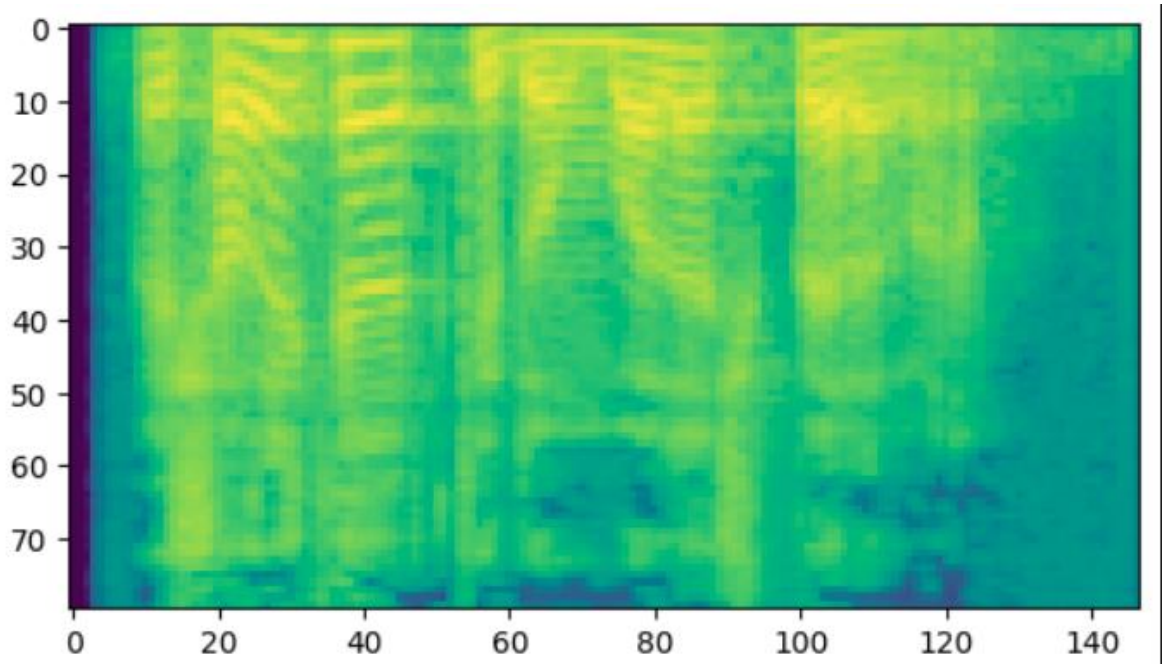
```
def prepare_data(row_data):
    audio = row_data["audio"]
    result = {}
    with torch.no_grad():
```

```
        input_features =
feature_extractor(audio_target=audio["array"],
sampling_rate=16000,
return_attention_mask=False,  return_tensors="pt")
        input_features['input_values'] =
input_features['input_values'][0]
        speaker_embeddings =
speaker_embedding_generator(audio["array"])

        input_ids = tokenizer(text=row_data["sentence"],
return_attention_mask=False,  return_tensors="pt",
padding='longest').input_ids[0]
    result["input_ids"] = input_ids
    result["input_features"] = input_features
    result["speaker_embeddings"] = speaker_embeddings.squeeze()

    return result
```

یک نمونه از Mel spectrogram :



شکل ۱ : **Mel spectrogram** از دیتای شماره ۲ آموزش

هر صوتی که بخواهیم تولید کنیم متناظر با آن یک فایل صوتی در دیتاست آموزش وجود دارد از این فایل صوتی باید speaker embedding بگیریم و به همراه متنی که قرار است Mel spectrogram آن generate شود و ما آن را با استفاده از tokenizer به input ids تبدیل کرده ایم به مدل بدهیم و مدل باید یاد بگیرد که Mel spectrogram شبیه به Mel spectrogram ، Speech اصلی تولید کند. برای این کار ما فقط قسمت decoder/encoder را fine-tune میکنیم.

همچنین با استفاده ازHiFi-GAN به عنوان Vocoder، Mel spectrogram را به صدا تبدیل میکنیم.بدین منظور تابع زیر نوشته شده است که برای تست مدل از آن استفاده میشود:

```
def speech_generator(model, row_data):
    model.eval()
    speeches = []
    with torch.no_grad():
        input_ids = tokenizer(text=row_data["sentence"],
return_tensors="pt", padding='longest').input_ids.to(device)
        embeddings = torch.zeros((1, 512)).to(device)
        speech = model.generate_speech(input_ids=input_ids,
speaker_embeddings=embeddings, vocoder=vocoder)
        speeches.append(speech.cpu().numpy())
    return speeches
```

- آموزش مدل

پیاده سازی نهایی برای آموزش مدل:

```
from IPython.display import Audio, display
import random

num_epochs = 30
batch_size = 15
learning_rate = 1e-5
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate,
weight_decay=0.0)
losses=[]

train_data = dataset.select(range(15000))

batch_loop = tqdm(range(0, len(train_data), batch_size),
total=len(train_data)//batch_size, leave=False)
scheduler = torch.optim.lr_scheduler.LinearLR(optimizer,
total_iters=(len(train_data) // batch_size) * num_epochs)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    for batch_number in batch_loop:
        inputs = train_data[batch_number:batch_number+batch_size]

        f_size = feature_extractor.feature_size
        feature_extractor.feature_size = feature_extractor.num_mel_bins
        input_features = feature_extractor.pad([{'input_values':
input_feature['input_values']} for input_feature in
inputs['input_features']], return_tensors='pt')
```

```
        feature_extractor.feature_size = f_size
        input_values =
input_features['input_values'].masked_fill(input_features['attention_mask']
.unsqueeze(-1).ne(1), -100)
        input_ids = tokenizer.pad([{'input_ids': v} for v in
inputs['input_ids']], return_tensors='pt')

        if model.config.reduction_factor > 1:
            target_lengths = torch.tensor([len(feature["input_values"]) for
feature in inputs['input_features']])
            target_lengths = target_lengths.new([length - length %
model.config.reduction_factor for length in target_lengths])
            max_length = max(target_lengths)
            input_values = input_values[:, :max_length]

        inputs = {
            'labels': input_values.to(device),
            'input_ids': input_ids['input_ids'].to(device),
            'attention_mask': input_ids['attention_mask'].to(device),
            'speaker_embeddings':
torch.tensor(inputs['speaker_embeddings']).to(device)
        }

        optimizer.zero_grad()
        outputs = model(**inputs)
        total_loss += outputs.loss
        outputs.loss.backward()
        optimizer.step()
        scheduler.step()
        batch_loop.set_postfix(loss=outputs.loss.item())

        if batch_number % 10 == 0:
                torch.cuda.empty_cache()
    average_loss = total_loss / (len(train_data) // batch_size)
    losses.append(average_loss.cpu().detach().numpy())

    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {average_loss}')

    for i in range(3):
      test = persian_test_dataset[random.randint(1, 10440)]
      generated_speeches = speech_generator(model, test)
      print(f"test sample{i} : {test['sentence']}")
      display(Audio(generated_speeches, rate=16000))


    torch.save(model.state_dict(), f'speech_t5{epoch}.pt')
    torch.cuda.empty_cache()
```

همانطور که مشاهده میشود برای مدل ۳۰ ایپاک با batch-size=15 به علت محدودیت gpu گوگل در نظر گرفته شده و از ۱۵۰۰۰ نمونه آموزش استفاده شده است.همچنین از optimizer adam برای آموزش با نرخ یادگیری 1e-5 استفاده شده است.

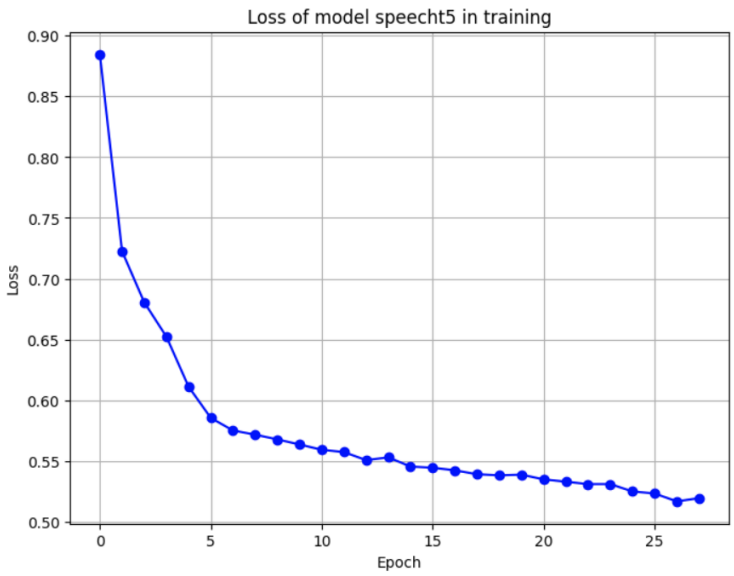
برای پیاده سازی فرایند آموزش که توسط trainer انجام میشود در ۳۰ ایپاک برای هر batch یک لوپ نوشته میشود که برای پیاده سازی processor با استفاده از توابع گفته شده و استفاده از تغییرات سایز input feature به طور موقت اندازه ویژگی را به تعداد Mel bin ها تغییر می دهیم و تمام آنها را در هر batch ، pad کرده و سپس به مقدار اصلی خود برمیگردانیم.عملیات pad کردن را برای input ids نیز انجام میدهیم همچنین از Attention mask برای ورودی مدل استفاده میکنیم.

در نهایت طول های ورودی را بر اساس reduction factor مشخص شده در پیکربندی مدل تنظیم می کنیم. این تضمین می کند که طول های ورودی بر reduction factor تقسیم می شوند و آنها را به حداکثر طول محاسبه شده برای ثبات در پردازش هر batch کاهش می دهد.

در نهایت به آموزش مدل و گرفتن لاس میپردازیم برای ارزیابی مدل با استفاده از تابعی که قبلا به آن اشاره شد از دسته داده تست استفاده میکنیم و آموزش را تا جایی ادامه میدهیم که یک نمونه تست قابل استناد تولید شود.

- نتایج

در فرایند آموزش در ۲ ایپاک از تمام دیتاست استفاده شد و سپس کد بالا ران شده است که ۲۳ ایپاک آن اجرا شده است به همین جهت نمودار خطا در فرایند آموزش به صورت زیر کاهش یافته است.



شکل ۲-نمودار لاس در فرایند آموزش و ارزیابی برای ۲۵ ایپاک

دلیل توقف آموزش نمونه تقریبا خوبی بوده است همچنین به علت محدودیت gpu آموزش بیشتر مدل قابل انجام نبود و این ۲۵ ایپاک در ۴ ساعت آموزش به نتیجه زیر منتهی شد:

test sample: می خواهم بگویم چقدر قدردان شما هستم.

final sample.wav

یک نمونه تولید شده در ایپاک ۱۲:

test sample1 : خورش کدو

download.wav

همانطور که در صورت سوال اشاره شده ارزیابی این مدل به شکل تولید ۳ نمونه تست در هر ایپاک صورت گرفته است که در نوت بوک آپلود شده قابل بررسی است.

دو فایل مورد نظر نیز در فایل اصلی تمرین آپلود شده است.

- Joint pre-training

یک تکنیک قدرتمند است که در مدل‌های پردازش زبان طبیعی (NLP) برای یادگیری بازنمایی‌های مشترک از روش‌های مختلف (مانند متن و گفتار) بر روی یک مدل واحد استفاده می‌شود. هدف ایجاد یک فضای بازنمایی یکپارچه است که در آن بتوان هر دو روش را به طور موثر مدلسازی کرد.از آنجا که در مدل speecht5 میتوان هر دو وروردی متن و گفتار را برای پردازش داشته باشیم استفاده از این رویکرد ضروری می باشد

روش‌های pretraining فقط می‌توانند از داده‌های گفتاری یا متنی برای مدل‌سازی اطلاعات آکوستیک یا زبان به صورت جداگانه استفاده کنند. برای ایجاد یک نقشه برداری متقاطع بین گفتار و متن، در این مدل یک روش کوانتیزاسیون برداری متقاطع را برای یادگیری بازنمایی هایی پیشنهاد می شود که اطلاعات modality-invariant را ضبط می کنند.در speecht5 با به اشتراک گذاشتن tokenهای مجزا در بین modalities ، رویکرد joint pretraining پل‌هایی را بین گفتار و متن ایجاد می‌کند. حالت های پنهان و واحدهای latent مخلوط شده و به عنوان ورودی ماژول cross-attention در decoder استفاده می شود.