به نام خدا

دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

مدل‌های مولد عمیق

تمرین شماره سه

نام و نام خانوادگی	فاطمه نادی
شماره دانشجویی	۸۱۰۱۰۱۲۸۵
تاریخ ارسال گزارش	۱۴۰۲/۱۱/۲۱

فهرست گزارش سوالات

فهرست شکل‌ها

فهرست جداول

پرسش سه

توضیحاتی پیرامون مدل

مدل SpeechT5 یک چارچوب یکپارچه کدگذار-کدگشا برای وظایف مختلف پردازش زبان سخن مانند شناسایی خودکار گفتار(ASR) ، تبدیل متن به گفتار(TTS) ، ترجمه گفتار(ST) ، تبدیل صدا(VC) ، بهبود گفتار(SE) ، و شناسایی سخن‌گو (SID) است.

ساختار مدل شامل یک شبکه مبنا کدگذار-کدگشا و ۶ شبکه‌های قبل و بعد از مدال مخصوص هر حالت است. شبکه‌های پیش‌مدال ورودی گفتار یا متن را به نمایش‌های مخفی تبدیل می‌کنند. کدگذار-کدگشا تبدیل دنباله به دنباله را مدل می‌کند. شبکه‌های پس‌مدال خروجی گفتار یا متن را تولید می‌کنند.
تکنیک joint-pre training زمانی استفاده می‌شود که مدالیتی‌های مختلف به طور جداگانه آموزش دیده اند و در اینجا ما نیازمند یک باز نمایش واحد از آن‌ها هستیم. برای پیش‌آموزش، SpeechT5 بر داده‌های بی‌برچسب گفتار (صوت‌های LibriSpeech) و متن (مدل زبانی متن LibriSpeech) با استفاده از اهداف توالی به توالی دنباله‌ای و مدل‌سازی زبان ماسک‌شده آموزش می‌بیند. یک رویکرد نوآورانه برای کوانتیزاسیون بردارهای متقابل معرفی می‌شود تا نمایش‌های گفتار و متن را در یک فضای معنایی مشترک در طول پیش‌آموزش هماهنگ کند. کدهای لاتین گسسته بین حالت‌ها به اشتراک گذاشته می‌شوند. برای ایجاد یک تطابق بین حالت‌های گفتار و متن، SpeechT5 از روش کوانتیزاسیون بردار متقابل در طول پیش‌آموزش استفاده می‌کند. کوانتیزاسیون بردار متقابل نمایش‌های پیوسته گفتار و متن را از کدگذار به نمایش‌های گسسته با استفاده از یک کتابخانه کد اشتراکی تبدیل می‌کند.

به طور خاص، خروجی‌های کدگذار به نزدیک‌ترین بردارهای همسایه در کتابخانه کد با استفاده از فاصله L2 تبدیل می‌شوند. این امکان را به وجود می‌آورد که کدهای لاتن گسسته را بین حالت‌ها به اشتراک بگذاریم. در طول پیش‌آموزش، بخشی از نمایش‌های کدگذار به صورت تصادفی با کدهای لاتن کوانتیزه در مراحل زمانی متناظر جایگزین می‌شوند. در این نمایش‌های ترکیبی که حاوی خروجی‌های کدگذار و کدهای کوانتیز شده هستند، محاسبه می‌شود. این باعث می‌شود کدهای کوانتیزه به یادگیری تطابق‌های متقاطعی میان گفتار و متن بپردازند، زیرا رمزگشایی‌کننده باید به هر دو نمایش گفتار و متن به طور همزمان توجه کند. به این ترتیب، فضای تعبیه کوانتیزه به اشتراک گذاری نمایش‌های گفتار و متن را هماهنگ می‌کند و اجازه می‌دهد مدل روابط متقاطع حالت را یاد بگیرد. این آموزش مشترک یک پیوند بین داده‌های گفتار و متن را فراهم می‌کند که برای وظایف پایین‌جریانی که نیازمند تطابق حالتی مانند شناسایی و سنتز گفتار هستند، مفید است. هدف آموزش مشترک شامل اتلاف‌های خودکدگذاری بر روی گفتار و متن با رویکرد کوانتیزاسیون بردار متقاطع است.

شبکه‌های پس‌مدال مورد استفاده در decoder شامل دو شبکه خطی هستند. یکی از این شبکه‌ها ورودی decoder را می‌گیرد و ویژگی‌های log-mel خروجی را ایجاد می‌کند. سپس پنج لایه از فیلترهای conv 1x1 بر روی این ویژگی‌ها اعمال می‌شود که وظیفه‌ی اصلی‌شان پاکسازی این ویژگی‌هاست. یک شبکه خطی دیگر هم برای پیش‌بینی توکن پایانی تعبیه شده است. در شبکه‌های پیش‌مدال و پس‌مدال مرتبط با متن، از یک لایه تعبیه مشترک استفاده می‌شود که توکن ورودی را به یک بردار تعبیه تبدیل می‌کند و پس‌مدال هم hidden state را با استفاده از یک softmax به احتمال تولید توکن بعدی تبدیل می‌کند. برای fine-tuning، کدگذار-کدگشا پیش‌آموزش‌دیده با شبکه‌های pre/post nets ، مناسب برای هر وظیفه downstream task، ترکیب می‌شود و با استفاده از task-specific losses. آموزش داده می‌شود.

آماده سازی دیتاست برای آموزش مدل

برای آموزش مدل از تمامی دادگان دیتاست آموزش برای آموزش مدل استفاده شد.
که تعداد آن‌ها ۲۸۰۲۴ ویس است.
برای داده‌های تست نیز به همین میزان داده وجود دارد که ده درصد آن‌ها را برای اعتبارسنجی استفاده شده است.

به وسیله تابع زیر می‌توان ویس‌های موجود را در محیط کولب خواند:

```
def play_sound(df, idx):
  data_point = df.iloc[idx]
  print(data_point["sentence"], "\n")
  path = "/content/cv-corpus-13.0-2023-03-09/fa/clips/" + data_point["path"]

  speech, sample_rate = torchaudio.load(path)
  speech = speech[0].numpy().squeeze()
  speech = librosa.resample(np.asarray(speech), orig_sr = sample_rate, target_sr = 16000)

  return ipd.Audio(data=np.asarray(speech), autoplay=True, rate=16000)

play_sound(df_train, idx = 1)
```

فایل صوتی با torchaudio لود می‌شود و به Numpy تبدیل می‌شود.
نرخ نمونه برداری را مطابق با شنوایی انسان ۱۶۰۰۰ تنظیم کرده‌ایم و این آرایه صوتی نهایی را با کتاخانه ipd پخش می‌کنیم.

سپس یک نمونه از دادگان را با این روش می‌خوانیم.
متن : "پیام رمزی آنها را دریافت کردم".
صورت در فولدر voice ضمیمه شده است با نام 1.wav.

سپس مدل speechT5 را لود کرده و به همراه آن یک tokenizer و یک feature extractor لود می‌کنیم.
متن صوت‌ها را به کمک tokenizer توکنایز کرده و ویژگی‌های صوت را به وسیله feature extractor استخراج می‌کنیم.
مدل speechT5 یک مدل مالتی‌مدال است بدین صورت که مدالیتی‌های مختلفی را می‌گیرد و به هر فرم در اینجا نسخه صوتی در می‌آورد در نتیجه چون ما می‌خواهیم این مدل را fine tune کرده و به یک مدل text to speech برسیم نیاز است تا مراحل زیر را طی کنیم:

در ابتدا tokenizer برای متن فارسی آموزش ندیده است یکی از راه‌کارها به جهت استفاده در متن فارسی اضافه کردن حروف به توکنایز و یا تبدیل متن فارسی به فرم فینگلیش و استفاده که توکنایزر اولیه.

هر دو روش پیاده‌سازی شد اما به جهت آن‌که روش دوم نتایج بهتری در نهایت داد از روش دوم استفاده نمودیم.

پس در نتیجه ابتدا متن فارسی را به فرم فینگلیش درآورده که این روش سبب می‌شود مدل مصوت‌ها را به خوبی ادا نکند اما به نسبت اضافه کردن توکن که سبب می‌شود توکنایزیشن و فاین‌تیون کردن مدل سخت‌تر شود بهتر عمل کرده است.
به کمک کلاس PersianTextPreprocessor این عمل صورت گرفت همچنین توکن‌هایی که در این عمل mapping وجود نداشت شناسایی و به جای آن کاراکتر مناسب جایگزین شد. همچنین برای نقطه گذاری نیز جایگزین معادل آن در انگلیسی انتخاب و جایگذاری شد.
در ادامه نگاهی مختصر به این کلاس خواهیم داشت:

```
class PersianTextPreprocessor:

  def __init__(self):
    self.persian_to_english_dict = {
      u"\u0627":"A", # الف
      u"\u0627":"A", # الف
      u"\u0675":"A", # الف کوچك
      u"\u0673":"A", # همزه بالا
      u"\u0630":"A", # ضبط الف
      u"\u0622":"AA", # آ
      u"\u0628":"B", # ب
      u"\u067E":"P", # پ
      u"\u062A":"T", # ت
```

```
    u"\u0637":"T", # ط
    u"\u0679":"T", # ة
    u"\u0633":"S", # س
    u"\u062B":"S", # ث
    u"\u0635":"S", # ص
    u"\u062C":"J", # ج
    u"\u0686":"CH", # چ
    u"\u062D":"H", # ح
    u"\u0647":"H", # ه
    u"\u0629":"H", # ة
    u"\u06DF":"H",# ۃ
    u"\u062E":"KH", # خ
    u"\u062F":"D", # د
    u"\u0688":"D", # ضبط د
    u"\u0630":"Z", # ذ
    u"\u0632":"Z", # ز
    u"\u0636":"Z", # ض
    u"\u0638":"Z", # ظ
    u"\u068E":"Z", # ضبط ز
    u"\u0631":"R", # ر
    u"\u0691":"R", # ڑ
    u"\u0698":"ZH", # ژ
    u"\u0634":"SH", # ش
    u"\u0639":"A", # ع
    u"\u063A":"GH", # غ
    u"\u0641":"F", # ف
    u"\u0642":"GH", # ق
    u"\u06A9":"K", # ک
    u"\u06AF":"G", # گ
    u"\u0644":"L", # ل
    u"\u0645":"M", # م
    u"\u0646":"N", # ن
    u"\u06BA":"N", # ں
    u"\u0648":"O", # و
    u"\u0649":"Y", # ى
    u"\u0626":"Y", # ي
    u"\u06CC":"Y", # ي
    u"\u06D2":"E", # ۂ
    u"\u06C1":"H", # ہ
    u"\u064A":"E", # ي
    u"\u06C2":"AH", # ے
    u"\u06BE":"H", # ێ
    u"\u0643":"K", # ك
    u"\u0621":"A", # ا
    u"\u0624":"O", # و
    u"\u0648":"U", # و
    u"\u0623":"A", # أ
    u"\uFBB5":"T", # ة
    u"\u2e2e":"?", # ؟
    u"\u201D":"\"", # علامت نقل قول
    u"\u201C":"\"", # علامت نقل قول
    u"\u0601":"?", # علامت سوال معکوس
    u"\u064E":"A", # فتحه
    u"\u0650":"E", # کسره
    u"\u064F":"OU", # ضمه
    "؛": ";", # علامت ویرگول
    "؟":"?", # علامت سوال
    u"\u060C":"" # علامت وصل
   }

  self.text_replacements = [
    (' ', ' '),
    ('«', ' '),
    ('»', ' '),
    ('ـ', ' '),
    ('ٓ', ' '),
    ('ً', ' '),
    ('ٔ', ' '),
    ('ۀ', 'E'),
    ('–', ' '),
```

```
            (' ', ' ')
        ]

    def transliterate(self, text):
        for persian, english in self.persian_to_english_dict.items():
            text = text.replace(persian, english)
        return text

    def clean_text(self, text):
        for src, dst in self.text_replacements:
            text = text.replace(src, dst)
        return text

    def preprocess(self, inputs):
        english_text = self.transliterate(inputs["sentence"])
        cleaned_text = self.clean_text(english_text)

        # Add punctuation
        if cleaned_text[-1] not in [".", "?", "!"]:
            cleaned_text += "."

        inputs["sentence"] = cleaned_text
        return inputs

preprocessor = PersianTextPreprocessor()

dataset_train = dataset_train.map(preprocessor.preprocess)
dataset_test = dataset_test.map(preprocessor.preprocess)
```

حال می‌توان از متن موجود در دیتاست برای توکنایز شدن استفاده نمود.

یک نمونه از mapping صورت گرفته:

پیام رمزی آنها را دریافت کردم.
PYAM RMZY AANHA RA DRYAFT KRDM

همچنین برای استخراج ویژگی از صوتی که به طور خام موجود است feature extractor صوت را می‌گیرد آن را نرمال می‌کند تا مدل بتواند از آن استفاده کند در نتیجه ورودی decoder موجود در speechT5 باید از خروجی feature extractor تغذیه کند. این مدل علاوه‌بر نرمالایز کردن log-mel را نیز استخراج می‌کند که در آموزش مدل و محاسبه loss مورد استفاده قرار می‌گیرد.

```
def prepare_dataset(data_point, dir = "/content/test_data/"):

    path = dir + data_point["path"]

    # read audio
    speech, sample_rate = torchaudio.load(path)
    speech = speech[0].numpy().squeeze()
    audio = librosa.resample(np.asarray(speech), orig_sr = sample_rate, target_sr = 16000)

    # speaker embedding
    speaker_embedding =  create_speaker_embedding(torch.Tensor(audio))

    # extract features
    audio_values = feature_extractor(audio_target = torch.Tensor(audio), sampling_rate=16000, return_attention_mask=True,
padding=True, return_tensors='pt')

    data_point["input_ids"] = tokenizer(data_point['sentence'])['input_ids']     #tokenizer
    data_point['attention_mask'] = tokenizer(data_point['sentence'])['attention_mask']
    data_point["labels"] = audio_values['input_values'].squeeze()
    data_point["speaker_embeddings"] = speaker_embedding

    return data_point
```

برای محاسبه تابع هزینه مدل speechT5 ابتدا متن توکنایزشده به انکودر داده می‌شود سپس ویژگی‌های صوت استخراج شده و خروجی دیکودر که یک spectrogram است با این ویژگی‌ها مقایسه شده و loss محاسبه می‌شود.

این تابع دیتاهای موجود در دیتاست را گرفته و متن ورودی را توکنایز می‌کند همچنین از speaker embedding استفاده می‌کند تا ویژگی‌های صورت را امبد کند و این امبدینگ را به گونه‌ای انجام می‌دهد که طول تمامی خروجی‌ها یکسان باشد (۵۱۲) و این‌کار را با پد کردن خروجی انجام می‌دهد. همچنین ویژگی‌های آدیو را استخراج کرده و موارد مورد نیاز برای آموزش مدل را فراهم می‌کند.

قبل از fine tune کردن مدل، متن ورودی را به شکل زیر می‌خواند:
در 2.wav ضمیمه شده است.
متن: .ANDAKHTN TSUYR RUY PRDH
انداختن تصویر روی پرده ☺

ساخت dataloader

ورودی‌های مدل‌های صوتی به جهت آن‌که قابل آموزش باشند باید در هر batch از داده طول یکسانی داشته باشند. می‌توان به طور passive تمامی دادگان را pad کرد به اندازه طول بزرگ‌ترین داده یا می‌توان به جهت استفاده بهینه از حافظه رم هر بچ از داده را به اندازه بلندترین داده آن pad کرد.
به همین منظور پدینگ مربوط به توکنایزر و ویژگی‌های استخراج شده به طور جداگانه اعمال می‌شود.
کلاس DataCollatorWithPadding در کتابخانه Hugging Face Transformers به تهیه دسته‌هایی از داده برای آموزش مدل‌های ترنسفورمر کمک می‌کند. به طور خاص، آن برای مواردی طراحی شده است که دنباله‌های ورودی اندازه‌های مختلفی دارند و با اضافه کردن پدینگ‌ها به آنها درون یک دسته، اندازه‌های یکسانی برای آنها فراهم می‌آید. در زمان آموزش یک مدل ترنسفورمر، معمول است که دنباله‌ها را به هم پیوسته کنیم تا پردازش بهینه‌تری داشته باشیم. اما، از آنجایی که دنباله‌ها ممکن است اندازه‌های مختلفی داشته باشند، باید آنها را درون هر دسته به یک اندازه مشترک پد کنیم. کلاس DataCollatorWithPadding این فرآیند را به صورت خودکار انجام می‌دهد. data_collator دنباله های دسته را به صورت پویا تا حداکثر طول آن دسته قرار می دهد.
در این مدل‌ها از عدد 100- برای پد کردن استفاده می‌شود. و ماتریس attention mask نیز تولید و برگردانده می‌شود. سپس از reduction factor در تنظیم مدل استفاده می‌شود بدین صورت که متناسب با هر spectrogram اعمال می‌شود در طول label. در مرحله بعد، طول ویژگی‌های خروجی با استفاده از reduction factor به مضربی از این فاکتور گرد می‌شود. در نتیجه باید spectrogram مضربی از reduction function باشد و اگر بزرگ‌تر از آن بود به اندازه اولین بزرگترین ضریب درمیاید.
در نهایت، داده‌های ورودی با استفاده از این کلاس آماده می‌شوند و به عنوان دسته‌های آموزشی به مدل تولید گفتار داده می‌شوند. در واقع به عنوان ورودی دیکودر داده می‌شود.

```
@dataclass
class TTSDataCollatorWithPadding:
    tokenizer: Any
    feature_extractor: Any

    def __call__(self, features):
        input_ids = [{"input_ids": feature["input_ids"]} for feature in features]
        label_features = [{"input_values": feature["labels"]} for feature in features]
        speaker_features = [feature["speaker_embeddings"] for feature in features]
        inputs = tokenizer.pad(input_ids, return_tensors="pt", return_attention_mask=True)

        feature_size_hack = feature_extractor.feature_size
        feature_extractor.feature_size = feature_extractor.num_mel_bins
        targets = feature_extractor.pad(label_features, padding=True, return_tensors="pt", return_attention_mask=True)
        feature_extractor.feature_size = feature_size_hack

        labels = targets["input_values"]
        decoder_attention_mask = targets.get("attention_mask")

        batch = {}
```

```
    batch['input_ids'] = inputs['input_ids']
    batch['attention_mask'] = inputs['attention_mask']
    batch['decoder_attention_mask'] = decoder_attention_mask
    batch['labels'] = labels
    batch['speaker_embeddings'] = torch.tensor(speaker_features)

    # # replace padding with -100 to ignore loss correctly
    batch["labels"] = batch["labels"].masked_fill(
        batch['decoder_attention_mask'].unsqueeze(-1).ne(1), -100
    )

    # round down target lengths to multiple of reduction factor
    if model.config.reduction_factor > 1:
        target_lengths = torch.tensor([
            len(feature["input_values"]) for feature in label_features
        ])
        target_lengths = target_lengths.new([
            length - length % model.config.reduction_factor for length in target_lengths
        ])
        max_length = max(target_lengths)
        batch["labels"] = batch["labels"][:, :max_length]

    return batch
```

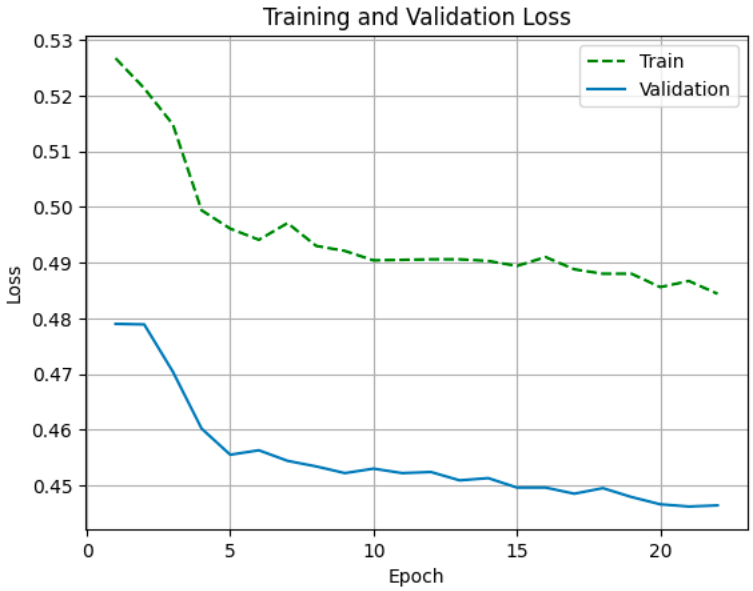
آموزش مدل

مدل speechT5 را بر روی دادگان آماده شده با استفاده از هایپر پارامترهای زیر آموزش می‌دهیم:

جدول 1هایپر پارامترهای مدل **speechT5**

پارامتر	مقدار
Batch size	**16**
optimizer	**ADAM**
Epoch#	**22**
Learning rate	**2e-5**
gamma	**0.7**
Weight decay	**1e-5**

با استفاده از تابع train مدل در ۲۲ ایپاک آموزش می‌بیند و fine tune می‌شود.
نمودار تابع هزینه بر روی داده‌های تست و ارزیاب به فرم زیر است:



شکل1 نمودار تابع هزینه بر روی آموزش مدل

وجود اختلاف نسبتا زیاد بین داده‌های ارزیاب و آموزش شاید به علت کم بودن تعداد داده‌های ارزیاب سبب شده مدل بهتر عمل کند. اگر تعداد دادگان را زیادتر کنیم این دو خطا بهم نزدیک می‌شوند. اما داده اورفیت نمی‌شود در نتیجه می‌توان آموزش را ادامه داد.

نتایج

نتایج مدل در فایل voice ضمیمه شده است که نتایج نشان می‌دهد مدل به درستی متن ورودی را می‌خواند.

خروجی مدل همراه با کمی نویز است که احتمالا اگر مدل در تعداد اپیاک بیشتر آموزش ببیند با توجه به اینکه تابع خطا روند کاهشی دارد این نویز نیز حذف می‌شود.
اگر در هنگام فینگلیش کردن متن ورودی اصوات را هم بتوان به گونه‌ی موثرتر صورت گیرد، می‌توان از مدل نتایج بهتری را تولید کرد.
از مقایسه ویس‌ها می‌توان دید که اگر نویز موجود را چشم‌پوشی کرد، می توان دید اندکی از لهجه لاتین همچنان باقی است اما تا حد خوبی مدل لحن فارسی را یادگرفته است.
اما موردی که بیشتر مشاهده می‌شود لحن گفتار گوینده به درستی در این مدل ادا نمی‌شود و گویی ویژگی‌های ساختاری گوینده به درستی یادگرفته نمی‌شود.

به بررسی چندی از نتایج می‌پردازیم:

RFTN BH MUZH.wav
متن: رفتن به موزه

همان‌طور که شنیده می‌شود مدل جنسیت صدا و تا حد خوبی آواها را منتقل کرده است اما صدای ضمیمه شده حاوی نویز است که با افزایش دوره آموزش تا حد خوبی برطرف می‌شود.

CHUB KHSHK BH AASANY MYSUZD.wav
متن: چوب خشک به آسانی می‌سوزد.

کلماتی مثل ؛چ؛ که کاراکتر آنها در زبان پایه انگلیسی نیست به خوبی یادگرفته نمی‌شود.

SPS TUFAN BH KHRUSH AAMD.wav
متن: سپس طوفان به خروش آمد.

در اینجا به نسبت لحن منتقل شده.
مشاهده می‌شود نسخه‌های اصلی نیز دارای نویز هستند.

مثلا خدا را کهدا می‌خواند یا انتخاب را اَنتخاب به این علت که هنگام منتقل کردن به انگلیسی صداهای ُ ً منتقل نمی‌شود.

در نهایت با توجه به اینکه فینگلیش کردم مصوت‌ها را منتقل نمی‌کند در نتیجه همان‌طور که پیش‌بینی می‌شود به خوبی آن متن بیان نمی‌شود.
اگر بتوان در هنگام جمع آوردی دیتا آن را به همان شکل فینگلیش نوشت یا تابع پیچیده‌تری برای تبدیل آن از فارسی به انگلیسی نوشت می‌توان به نتایج بهتری دست یافت در ادامه به چند نمونه از صداهای با کیفیت بالاتر آورده می‌شود.

AAB AZ GYSUANSH MYCHKYD.wav
AU ANTKHAB BDY KRD.wav

سوال سه

https://medium.com/@sujathamudadla1213/what-is-datacollatorwithpadding-in-hugging-face-transformers-12c2b3b2f612

https://huggingface.co/speechbrain/spkrec-ecapa-voxceleb
https://github.com/HamedHemati/Tacotron-2-Persian/blob/master/notebooks/test_phonemizer.ipynb

https://arxiv.org/abs/2110.07205