به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

مدل‌های مولد عمیق

تمرین شماره سوم

نام و نام خانوادگی	کیانا هوشانفر
شماره دانشجویی	۸۱۰۱۰۱۳۶۱
تاریخ ارسال گزارش	

فهرست گزارش سوالات

سوال ۳ – Speech synthesis

ابتدا کتابخانه های لازم را دانلود می کنیم:

```
!pip install --q datasets soundfile speechbrain
!pip install --q sentencepiece
!pip install --q datasets
!pip install --upgrade accelerate --q
```

در مرحله ی بعد برای دسترسی به مدل ها و دیتاست باید به سایت huggning face دسترسی پیدا کنیم:

```
from huggingface_hub import notebook_login

notebook_login()
```

در گام بعدی دیتاست و مدل را به شکل زیر لود میکنیم:

```
from transformers import SpeechT5Processor, SpeechT5ForTextToSpeech

processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")
```

این کد پایتون از کتابخانه Hugging Face Transformers برای بارگذاری مدل های از پیش آموزش دیده برای کارهای تبدیل متن به گفتار از SpeechT5 استفاده می کند. SpeechT5Processor برای پردازش داده های ورودی استفاده می شود، در حالی که مدل SpeechT5ForTextToSpeech برای انجام تبدیل متن به گفتار بارگیری می شود. روش from_pretrained برای آوردن وزن‌ها و پیکربندی‌های از پیش آموزش‌دیده مرتبط با مدل microsoft/speecht5_tts استفاده می‌شود. «processor» مراحل پیش پردازش لازم را برای مدل مدیریت می کند، و «model» مدل واقعی تبدیل متن به گفتار را نشان می دهد.

```
from datasets import load_dataset, DatasetDict, Audio

common_voice = DatasetDict()
common_voice["train"] = load_dataset("mozilla-foundation/common_voice_13_0",
"fa", split="train", use_auth_token=True)
common_voice["test"] = load_dataset("mozilla-foundation/common_voice_13_0", "fa",
split="test", use_auth_token=True)
```

برای دانلود دیتاست از دیتایی که در huggingface بود استفاده می کنیم و به شکل بالا آن را لود میکنیم و دیتاهای ترین و تست را از آن میگیریم.

این کد پایتون از کتابخانه Hugging Face Datasets برای بارگذاری مجموعه داده های Common Voice برای زبان فارسی fa استفاده می کند. یک DatasetDict به نام common_voice ایجاد می‌کند تا تقسیم‌های مجموعه داده را سازماندهی کند. مجموعه داده به دو تقسیم، یعنی train و test با استفاده از تابع load_dataset بارگذاری می شود. مجموعه داده از mozilla-foundation/common_voice_13_0 ، مجموعه داده Common Voice گرفته شده است. پارامتر use_auth_token=True نشان می دهد که از نشانه های احراز هویت برای دسترسی به مجموعه داده استفاده می شود.

مشاهده میکنیم که دیتاست ما به شکل زیر خواهد بود:

```
Dataset({
    features: ['client_id', 'path', 'audio', 'sentence', 'up_votes', 'down_votes', 'age', 'gender', 'accent',
'locale', 'segment', 'variant'],
    num_rows: 28024
})
```

در قدم بعدی ستون های اضافی را حذف میکنیم:

```
# Remove unnecessary columns
common_voice = common_voice.remove_columns(['path', 'up_votes', 'down_votes',
'age', 'gender', 'accent', 'locale', 'segment', 'variant'])
common_voice = common_voice.cast_column("audio", Audio(sampling_rate=16000))
```

و در آخر به دیتاهای زیر خواهیم رسید:

```
DatasetDict({
    train: Dataset({
        features: ['client_id', 'audio', 'sentence'],
        num_rows: 28024
    })
    test: Dataset({
        features: ['client_id', 'audio', 'sentence'],
        num_rows: 10440
    })
})
```

مجموعه داده ممکن است شامل کاراکترهایی باشد که در و SpeechT5 tokenizer نباشند. آن ها به نشانه های نامشخص تبدیل می شوند. به این دلیل کد های زیر را برای این منظور ران میکنیم.

ابتدا باید بفهمیم توکن های پشتیبانی نشده چی هستن. SpeechT5Tokenizer با کاراکترها به عنوان token کار می کند، بنابراین باید همه کاراکترهای متمایز مجموعه داده را استخراج کرده و ببینیم چه کاراکترهایی در آن نیستند.

```
def update_tokenizer_with_dataset_vocab(processor, common_voice):
    tokenizer = processor.tokenizer

    # Concatenate all sentences in the training dataset
    all_text = " ".join(common_voice["train"]["sentence"])

    # Extract unique characters to create vocabulary
    vocab = list(set(all_text))

    # Create a dataset containing the vocabulary and all concatenated text
    vocabs = common_voice["train"].map(
        lambda batch: {"vocab": [vocab], "all_text": [all_text]},
        batched=True,
        batch_size=-1,
        keep_in_memory=True,
        remove_columns=dataset.column_names,
    )

    # Extract the vocabulary from the created dataset
    dataset_vocab = set(vocabs["vocab"][0])

    # Extract the vocabulary from the tokenizer
    tokenizer_vocab = {k for k, _ in tokenizer.get_vocab().items()}

    # Find new characters in the dataset vocab that are not in the tokenizer
vocab
    new_chars = dataset_vocab - tokenizer_vocab

    # Add new tokens to the tokenizer
    new_tokens = processor.tokenizer.add_tokens(list(new_chars))

    # Get the updated vocabulary
    updated_vocab = processor.tokenizer.get_vocab()

    return updated_vocab

# Assuming 'processor' and 'common_voice' are already defined
updated_vocab = update_tokenizer_with_dataset_vocab(processor, common_voice)
```

این قطعه کد استخراج واژگان را برای مجموعه داده متنی با استفاده از کتابخانه Hugging Face Datasets انجام می دهد. با الحاق تمام جملات در تقسیم آموزشی مجموعه داده برای تجزیه و تحلیل بیشتر شروع می شود. سپس کاراکترهای منحصربه‌فرد در متن به هم پیوسته استخراج می‌شوند تا فهرستی از واژگان تشکیل شود. متعاقباً، یک مجموعه داده با استفاده از تابع map ایجاد می‌شود، که در آن هر دسته توسط یک تابع لامبدا پردازش می‌شود تا هم واژگان و هم کل متن پیوسته را شامل شود. این ساختار مجموعه داده تجزیه و تحلیل و مقایسه بعدی را تسهیل می کند.

سپس واژگان مجموعه داده به دست آمده استخراج می شود و واژگان tokenizer با گشتن از شناسه های نشانه موجود در tokenizer به دست می آید. واژگان مجموعه داده‌ها بینش‌هایی را درباره شخصیت‌های منحصربه‌فرد موجود در داده‌های آموزشی ارائه می‌کند، در حالی که واژگان نشانه‌ساز مجموعه نشانه‌هایی را نشان می‌دهد که نشانه‌گذار از آنها آگاه است. این اطلاعات برای کارهایی مانند پردازش متن، توکن‌سازی و مدل‌سازی زبان، کمک به درک تنوع کاراکترها در مجموعه داده‌ها و نحوه توکن‌سازی متن توسط نشانه‌گذار بسیار مهم است.

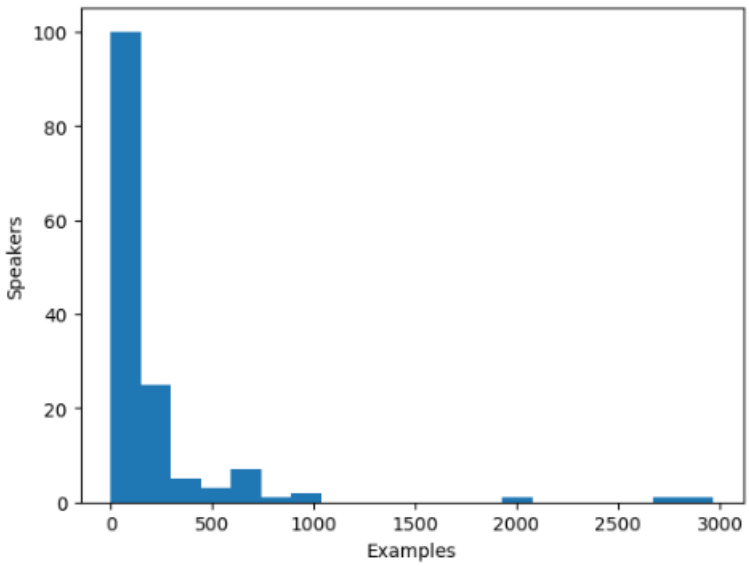
```
new_char = dataset_vocab - tokenizer_vocab
new_tokens = processor.tokenizer.add_tokens(list(new_char))
updated_vocab = processor.tokenizer.get_vocab()
```

با کد بالا توکن هایی که نبودند را به tokenizer اضافه میکنیم.

```
tokenizer_vocab = {k for k, _ in tokenizer.get_vocab().items()}
dataset_vocab - tokenizer_vocab
```

خروجی کد بالا set() است که نشان میدهد که tokenizer تمام حروف را دارد.

این دیتاست همانطوری که در توضیحات آمده است دارای چند speaker است. با کشیدن هیستوگرام زیر متوجه میشویم چه مقدار داده برای هر speaker وجود دارد.



شکل ۱ – تعداد دیتاها برای **speaker**

مشاهده میکنیم که مقدار زیادی از speakerها کمتر از ۵۰۰ مثال داره اند. برای سرعت بخشیدن به آموزش، ما speakerهایی را درنظر میگیریم که بین ۰ تا ۱۰۰۰ نمونه دارند و میبینیک که حدود ۱۴۲ speaker برای ما مانده است.

برای اینکه مدل TTS بتواند بین چند speaker تمایز قائل شود، باید برای هر نمونه یک speaker embedding ایجاد کنیم.

```
import os
import torch
from speechbrain.pretrained import EncoderClassifier

custom_spk_model_name = "speechbrain/spkrec-xvect-voxceleb"

custom_device = "cuda" if torch.cuda.is_available() else "cpu"

custom_speaker_model = EncoderClassifier.from_hparams(
    source=custom_spk_model_name,
    run_opts={"device": custom_device},
    savedir=os.path.join("/tmp", custom_spk_model_name)
)

def generate_custom_speaker_embedding(input_waveform):
    with torch.no_grad():
        custom_speaker_embeddings =
custom_speaker_model.encode_batch(torch.tensor(input_waveform))
```

```
        custom_speaker_embeddings =
torch.nn.functional.normalize(custom_speaker_embeddings, dim=2)
        custom_speaker_embeddings =
custom_speaker_embeddings.squeeze().cpu().numpy()
    return custom_speaker_embeddings
```

این کد از کتابخانه SpeechBrain برای بارگذاری یک مدل تشخیص بلندگو از پیش آموزش‌دیده بر اساس معماری x-vector برای مجموعه داده VoxCeleb استفاده می‌کند. مدل ابتدا از ماژول از پیش آموزش‌دیده SpeechBrain بارگیری می‌شود و نام مدل speechbrain/spkrec-xvect-voxceleb. سپس مدل بارگذاری شده برای ایجاد یک تابع generate_custom_speaker_embedding استفاده می‌شود که شکل موج ورودی (یک سیگنال صوتی) را می‌گیرد و speaker embeddings مربوطه را برمی‌گرداند.

تابع generate_custom_speaker_embedding شکل موج ورودی را با کدگذاری آن از طریق مدل تشخیص speaker از پیش آموزش‌دیده پردازش می‌کند. تعبیه‌های به‌دست‌آمده در امتداد بعد زمانی عادی شده و به آرایه‌های NumPy تبدیل می‌شوند. این تعبیه‌ها اساساً نمایشی فشرده و معنادار از ویژگی‌های بلندگو در سیگنال صوتی داده شده را نشان می‌دهند. این کد در سناریوهایی که به تشخیص speaker نیاز است. مدل از پیش آموزش‌دیده، بر اساس معماری x-vector، قادر به یادگیری ویژگی‌های speaker متمایز است و آن را برای شناسایی و تمایز speakerها در داده‌های صوتی مناسب می‌سازد.

```
def prepare_dataset(example):
    # Load the audio data; if necessary, this resamples the audio to 16kHz
    audio = example["audio"]
    # Feature extraction and tokenization using the processor
    example = processor(
        text=example["sentence"],
        audio_target=audio["array"],
        sampling_rate=audio["sampling_rate"],
        return_attention_mask=False,)
    # Strip off the batch dimension from the labels
    example["labels"] = example["labels"][0]

    # Use SpeechBrain to obtain x-vector speaker embeddings
    example["speaker_embeddings"] =
generate_custom_speaker_embedding(audio["array"])
    return example

dataset = dataset.map(
    prepare_dataset, remove_columns=dataset.column_names,
)
```

کد تابعی به نام prepare_dataset را تعریف می کند که برای پیش پردازش نمونه ای از مجموعه داده حاوی اطلاعات صوتی و متنی، با تمرکز ویژه بر وظایف پردازش گفتار. این تابع یک مثال را به عنوان ورودی می گیرد، که در آن مثال فرض می شود یک فیلد "صوتی" حاوی داده های صوتی و یک قسمت "جمله" حاوی اطلاعات متنی مربوطه دارد.

ابتدا، داده های صوتی از قسمت صوتی مثال بارگیری می شود. این مرحله شامل نمونه‌برداری مجدد بالقوه صدا به نرخ نمونه‌گیری استاندارد ۱۶ کیلوهرتز است که از ثبات داده‌ها اطمینان می‌دهد.

در مرحله بعد، این تابع از یک پردازنده استفاده می‌کند که برای پردازش گفتار و صدا طراحی شده است تا استخراج ویژگی و tokenization را انجام دهد. این شامل تبدیل اطلاعات متنی و داده های صوتی به فرمت مناسب برای پردازش بیشتر است. پردازنده با پارامترهایی مانند متن ورودی، داده های صوتی و سرعت نمونه برداری پیکربندی شده است و گزینه ای برای حذف تولید ماسک های توجه وجود دارد.

پس از مرحله پردازش، تابع با حذف ابعاد دسته ای از قسمت برچسب ها، مثال را اصلاح می کند. این معمولاً برای رسیدگی به ابعاد تانسور و اطمینان از سازگاری با وظایف پایین دستی مانند آموزش یک مدل انجام می شود.

در نهایت، این عملکرد شامل تولید جاسازی speaker با استفاده از SpeechBrain، یک جعبه ابزار برای پردازش گفتار و صدا است. از تابعی به نام generate_custom_speaker_embedding برای به دست آوردن نمایش های بردار x برای speaker مرتبط با صدای داده شده استفاده می کند. X-vectors نوعی جاسازی speaker مبتنی بر شبکه عصبی عمیق هستند که در وظایف تشخیص speaker استفاده می‌شوند.

برای training ابتدا دیتاست را به دو دسته تربن و تست تقسیم میکنیم، برای اینکه training راحتتر انجام شود train dataset زا به دو دسته train-test تقسیم میکنیم.

```
dataset = dataset.train_test_split(test_size=0.1)
```

بعد از اولین بار ترین کردن به این مشکل برمیخوردم که طول توالی Token indices از حداکثر طول توالی مشخص شده برای این مدل بیشتر میبود و ارور میداد. برای حل این مشکل باید آن‌هایی که طولانی تر بودند را از مجموعه داده حذف کنیم. در واقع، هر چیزی بیش از ۵۰۰ توکن را حذف می کنیم.

در قدم بعدی نیاز داریم که ما باید یک TTSDataCollatorWithPadding تعریف کنیم تا چندین نمونه را در یک دسته ترکیب کنیم. با این کار دنباله‌های کوتاه‌تری با نشانه‌های padding تکمیل می‌شود.

```python
from dataclasses import dataclass
from typing import Any, Dict, List, Union

@dataclass
class TTSDataCollatorWithPadding:
    processor: Any

    def __call__(self, features: List[Dict[str, Union[List[int], torch.Tensor]]]) -> Dict[str, torch.Tensor]:
        # Extract input_ids, label_features, and speaker_features from the provided 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]

        # Collate the inputs and targets into a batch using the processor's pad method
        batch = self.processor.pad(
            input_ids=input_ids,
            labels=label_features,
            return_tensors="pt",
        )
        # Replace padding with -100 to ignore loss correctly
        batch["labels"] = batch["labels"].masked_fill(
            batch.decoder_attention_mask.unsqueeze(-1).ne(1), -100
        )
        # Not used during fine-tuning, so it's removed from the batch
        del batch["decoder_attention_mask"]
        # Round down target lengths to a multiple of the reduction factor if applicable
        if self.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 % self.model.config.reduction_factor for length in target_lengths
            ])
            max_length = max(target_lengths)
            batch["labels"] = batch["labels"][:, :max_length]

        # Add speaker embeddings to the batch
        batch["speaker_embeddings"] = torch.tensor(speaker_features)

        return batch
```

کد یک کلاس به نام TTSDataCollatorWithPadding را تعریف می‌کند که برای وظایف تبدیل متن به گفتار TTS طراحی شده است. جمع‌آوری داده یک جزء حیاتی در آموزش مدل‌های یادگیری ماشینی است که مسئول دسته‌بندی و پردازش کارآمد ویژگی‌ها و اهداف ورودی در طول آموزش است. این جمع‌آوری داده خاص برای وظایف TTS طراحی شده است و در ارتباط با یک پردازنده، که یک processor خاص TTS است، کار می‌کند.

کلاس TTSDataCollatorWithPadding با یک پارامتر processor مقداردهی اولیه می شود، که نمونه ای از یک TTS processor است. این کلاس شامل یک متد __call__ است که در طول هر تکرار آموزش برای جمع‌آوری و پیش پردازش ویژگی‌ها و اهداف ورودی فراخوانی می‌شود.

در این روش، ویژگی‌های ورودی، از جمله input_ids نماینده دنباله‌های ورودی نشانه‌گذاری شده، label_features شامل مقادیر_ورودی که دنباله‌های صوتی هدف را نشان می‌دهند ، و Speaker_features شامل speaker embeddings، از لیست ویژگی‌های ارائه شده استخراج می‌شوند. سپس فرآیند جمع‌بندی با استفاده از روش processor pad انجام می‌شود، و اطمینان حاصل می‌شود که input_ids و label_features به‌طور مناسب برای تشکیل دسته‌ای با ابعاد یکنواخت قرار گرفته‌اند.

برای رسیدگی به اهداف ماسک‌دار در زمینه TTS، لایه‌بندی برچسب‌ها با -۱۰۰ جایگزین می‌شود تا از محاسبه صحیح تلفات اطمینان حاصل شود، و decoder_attention_mask از دسته حذف می‌شود، زیرا در هنگام تنظیم دقیق استفاده نمی‌شود. بعلاوه، اگر یک ضریب کاهش در پیکربندی مدل مشخص شده باشد (که نشان دهنده کاهش نمونه در مدل TTS است)، طول هدف به گونه ای تنظیم می شود که چند برابر ضریب کاهش باشد، که تراز بین توالی های ورودی و هدف را تسهیل می کند.

در نهایت، دسته جمع‌آوری‌شده با speaker embeddings تقویت می‌شود و یک تنسور ورودی کامل آماده برای آموزش مدل TTS را تشکیل می‌دهد. این جمع‌آوری داده برای حصول اطمینان از اینکه ویژگی‌های ورودی به‌طور کارآمد دسته‌بندی و پیش‌پردازش به شیوه‌ای سازگار با معماری‌های مدل TTS هستند، حیاتی است و به کارایی و اثربخشی کلی آموزش در برنامه‌های TTS کمک می‌کند.

```
data_collator = TTSDataCollatorWithPadding(processor=processor)
```

کد یک TTSDataCollatorWithPadding را با ارائه یک TTS processor به عنوان جمع کننده_داده نشان می دهد. این جمع‌آوری داده به‌طور خاص برای کارهای تبدیل متن به گفتار طراحی شده است. در طول هر تکرار آموزشی، ویژگی‌های ورودی، از جمله توالی‌های متنی توکن‌سازی‌شده و اهداف صوتی مربوطه

را به‌طور مؤثر جمع‌بندی و پیش پردازش می‌کند. همکاری با یک TTS processor تضمین می‌کند که مراحل پیش‌پردازش با الزامات مدل‌های TTS هماهنگ می‌شود، فرآیند آموزش را ساده می‌کند و مدیریت مؤثر ویژگی‌های ورودی متنوع را تسهیل می‌کند. نمونه‌سازی این جمع‌آوری داده، سادگی، کارایی و سازگاری آموزش مدل TTS را با خودکار کردن وظایف پیش‌پردازش ضروری افزایش می‌دهد.

در ادامه پارامترهایی که مدل را میخواهیم با آن ترین کنیم را مشخص میکنیم.

```
model.config.use_cache = False

from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="./speecht5_tts_fa_kh",
    per_device_train_batch_size=16,
    gradient_accumulation_steps=2,
    learning_rate=1e-4,
    warmup_steps=500,
    max_steps=6000,
    gradient_checkpointing=True,
    fp16=True,
    evaluation_strategy="steps",
    per_device_eval_batch_size=8,
    save_steps=1000,
    eval_steps=500,
    logging_steps=25,
    load_best_model_at_end=True,
    greater_is_better=False,
    label_names=["labels"],
    push_to_hub=False,
)
```

در ادامه، آرگومان های آموزشی مدل Seq2Seq را با استفاده از کلاس Seq2SeqTrainingArguments از کتابخانه ترانسفورماتور Hugging Face پیکربندی می کنیم. فراپارامترهای مختلفی در اینجا تنظیم شده است، از جمله فهرست خروجی برای ذخیره checkpoint مدل، اندازه دسته برای آموزش و ارزیابی، نرخ یادگیری، مراحل انباشت گرادیان، و موارد دیگر. نکته قابل توجه این است که نقطه کنترل گرادیان را فعال می کند، یک تکنیک ذخیره سازی حافظه برای آموزش مدل

های بزرگ، و تمرین با دقت ترکیبی fp16 را فعال می کند تا با استفاده از محاسبات ممیز شناور نیمه دقیق، در صورت امکان، سرعت تمرین را افزایش دهد.

در نهایت، قسمت آخر پیکربندی های آموزشی اضافی مانند استراتژی ارزیابی، ورود به سیستم و تنظیمات ذخیره مدل را مشخص می کند. استراتژی ارزیابی تنظیم شده است تا ارزیابی را در فواصل زمانی منظمی که توسط eval_steps تعریف شده است، با اندازه دسته ارزیابی به طور جداگانه مشخص شود، آغاز کند. نقاط checkpoint مدل در فواصل زمانی تعیین شده با ذخیره_گام‌ها ذخیره می‌شوند و مدل با بهترین عملکرد در پایان آموزش بارگذاری می‌شود. Logging برای گزارش معیارها به TensorBoard برای اهداف تجسم و نظارت تنظیم شده است. این پیکربندی جامع، آموزش کارآمد مدل Seq2Seq را برای سنتز گفتار تضمین می‌کند و در عین حال قابلیت‌های نظارت و ثبت را برای ردیابی و تحلیل عملکرد ارائه می‌دهد.

در ادامه مدل را با ران کردن خط بعد ترین میکنیم:

```
trainer.train()
```

Step	Training Loss	Validation Loss
500	0.635500	0.600808
1000	0.580200	0.570030
1500	0.570300	0.532141
2000	0.553400	0.531975
2500	0.544100	0.519837
3000	0.538000	0.502330
3500	0.514000	0.491730
4000	0.519100	0.489570
4500	0.500200	0.481429
5000	0.489400	0.474633
5500	0.486500	0.468066
6000	0.484400	0.463868

شکل ۲ - مقادیر **loss** در حین آموزش

با استفاده از شکل ۲ متوجه می شویم که مدل در حال ترین شدن هست و مقادیر لاس ما در train- val نزولی هستند.

بعد از اتمام فرایند training مدل finetune شده را روی صفحه huggingface بارگذاری میکنیم که در ادامه راحتتر بتوانیم خروجی را تولید کنیم.

```
kwargs = {
    "dataset_tags": "facebook/voxpopuli",
    "dataset": "VoxPopuli",
    "dataset_args": "config: nl, split: train",
    "language": "nl",
    "model_name": "SpeechT5 TTS Farsi",
    "finetuned_from": "microsoft/speecht5_tts",
    "tasks": "text-to-speech",
    "tags": "",
}
trainer.push_to_hub('KHooshanfar/SpeechT5_TTS_fa')
```

برای ارائه گفتار تولید شده توسط مدل با کیفیت مناسب مراحل زیر را طی میکنیم:

```
from transformers import SpeechT5Processor, SpeechT5ForTextToSpeech

model = SpeechT5ForTextToSpeech.from_pretrained("KHooshanfar/speecht5_tts_fa_kh")

example = common_voice["test"][300]
speaker_embeddings = torch.tensor(example["speaker_embeddings"]).unsqueeze(0)
text = " خروجی است"
tokenizer = processor.tokenizer

  ت س ا   ی ج و ر خ </s>

tokenizer.decode(tokenizer(text)["input_ids"])

inputs = processor(text=text, return_tensors="pt")

spectrogram = model.generate_speech(inputs["input_ids"], speaker_embeddings)

from transformers import SpeechT5HifiGan
vocoder = SpeechT5HifiGan.from_pretrained("microsoft/speecht5_hifigan")

with torch.no_grad():
    speech = vocoder(spectrogram)

from IPython.display import Audio
Audio(speech.numpy(), rate=16000)
```

این کد نحوه استفاده از کتابخانه Hugging Face Transformers را برای انجام سنتز متن به گفتار با استفاده از مدل SpeechT5 از پیش آموزش‌دیده که برای زبان‌های فارسی (فارسی) تنظیم شده است، نشان می‌دهد.

ابتدا ماژول های لازم از کتابخانه Transformers وارد می شوند. شامل SpeechT5Processor ، مسئول پیش پردازش ورودی‌های متن، و SpeechT5ForTextToSpeech است. مدل پیش‌آموزش‌شده با استفاده از روش «از_پیش‌آموزش‌شده» با شناسه مشخص شده Khooshanfar/speecht5_tts_fa_kh بارگذاری می‌شود که نشان‌دهنده مدل تنظیم‌شده برای TTS فارسی است.

سپس، یک نمونه ورودی متن از مجموعه داده common_voice برای اهداف آزمایشی انتخاب می‌شود. جاسازی های speaker برای مثال استخراج شده و برای ورودی به مدل آماده می شوند. متنی که باید ترکیب شود به عنوان "خروجی است" مشخص می شود که به "خروجی است" ترجمه می شود. متن با استفاده از SpeechT5Processor tokenizer نشانه‌گذاری می‌شود و آن را به شناسه‌های ورودی مناسب برای مدل تبدیل می‌کند.

پس از توکن‌سازی، متن ورودی با استفاده از نمونه processorپردازش می‌شود. ورودی های پیش پردازش شده در قالب تانسور مناسب به مدل به دست می آیند. سپس مدل با استفاده از روش تولید_گفتار spectrogram مربوط به گفتار سنتز شده را تولید می‌کند که شناسه‌های ورودی و speakerهای از پیش پردازش‌شده را به عنوان ورودی می‌گیرد.

علاوه بر این، یک مدل رمزگذار صوتی جداگانه SpeechT5HifiGan بارگذاری می شود. این مدل صداگذار وظیفه تبدیل طیف نگار تولید شده به شکل موج گفتار صوتی خام را بر عهده دارد. Vocoder شکل موج گفتار را از طیف نگار تولید می کند.

در نهایت، شکل موج گفتار سنتز شده با استفاده از کلاس Audio از ماژول نمایش IPython پخش می‌شود. گفتار سنتز شده به‌عنوان یک آرایه نمایش داده می‌شود، و با نرخ نمونه‌برداری ۱۶ کیلوهرتز پخش می‌شود و نمایشی شنیداری از گفتار تولید شده ارائه می‌دهد.

فایل تولید شده نهایی نیز پیوست شده است.

منابع استفاده شده:

https://huggingface.co/learn/audio-course/en/chapter6/fine-tuning

https://medium.com/nlplanet/a-full-guide-to-finetuning-t5-for-text2text-and-building-a-demo-with-streamlit-c72009631887

https://colab.research.google.com/drive/1i7I5pzBcU3WDFarDnzweIj4-sVVoIUFJ

https://huggingface.co/microsoft/speecht5_tts