实验复现报告

XIA

2024 年 12 月 9 日

https://github.com/Game-learning/recurrent

目录

1 图像增强

(a)

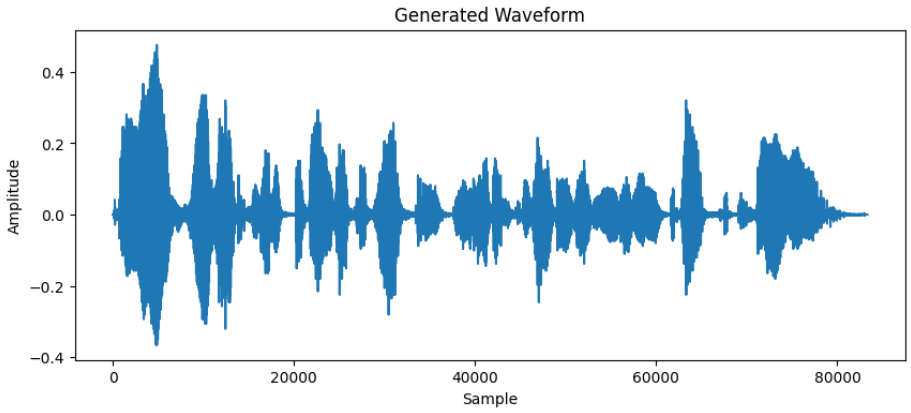
(b)

可以将一张模糊的照片增强为一张清晰的照片，提高图像的质量。

2 文本到语音转换

输入文本:

text = ”Hello, this is a test of speech synthesis using Tacotron2.”



可以将一段英文读成一个句子的语音。将中文文本转化为语音方面有待学习与探索。

3 文本到图片转换

输入文本:

Astronaut riding a horse.

可以根据英文提示词，生成一张对应的图片。

4 光学字符识别

(a)

(b)

图片 2a描述: a man and woman standing on a cliff overlooking a lake
图片 2b描述: a wolf is walking through a hole in the snow
可以识别图片，生成对图片的描述。

5 文本生成

当文本只有 20 句子时，学出来的结果可能是：
输入：
input_str = ”Python is a good”
生成的文本: Python is a good Python Python.. code topic science code data Python industry. industry. partsMachine partsMachine. partsMachine partsMachine partsMachine.
当你的文本只有很大时，已经能够学习出句子的格式：
epoch = 10 , Loss: 3.2292 , 输入：
input_str = ”I find it necessary”
生成的文本: I find it necessary to make a few changes to the code.
The first thing to do is to change the code to:
#include <iostream> #include <iostream> #include <iostream>
epochs = 20, Loss: 2.3805, 输入：
input_str = ”I find it necessary”
生成的文本: I find it necessary to make a few changes to the code.

The first is to make the code a bit more readable.

The second is to make the code a bit more readable.

The third is to make the code a bit more readable

6 内容审核

输入文本:

”Those people are all garbage and don’t deserve to live,You idiot, you can’t do anything well.”

类别	toxic	severe_toxic	obscene	threat	insult	identity_hate
是否存在	True	False	False	False	True	False

表 1: 内容判定情况

可以根据文本内容，判断文本情感。

7 Python 代码

######推荐使用虚拟环境，安装所需的包，避免出错。

```
#图像增强(cmd)————————————————————————————————————————————————————————————
git clone https://github.com/xinntao/Real-ESRGAN.git
cd Real-ESRGAN
# 安装 basicsr - https://github.com/xinntao/BasicSR
# 我们使用BasicSR来训练以及推断
pip install basicsr
# facexlib和gfpgan是用来增强人脸的
pip install facexlib
pip install gfpgan
pip install -r requirements.txt
python setup.py develop

wget https://github.com/xinntao/Real-ESRGAN/releases/download/v0.1.0/RealESR

#python inference_realesrgan.py -n RealESRGAN_x4plus -i inputs --face_enhanc

python inference_realesrgan.py -n RealESRGAN_x4plus -i photo1.jpg --face_enh



#文本到语音转换————————————————————————————————————————————————————————————
import torch
import torchaudio
import matplotlib.pyplot as plt
from torchaudio.models import Tacotron2
from torchaudio.pipelines import TACOTRON2_WAVERNN_PHONE_LJSPEECH

# 加载 Tacotron2 模型和 WaveRNN 声码器
bundle = TACOTRON2_WAVERNN_PHONE_LJSPEECH
```

```
processor = bundle.get_text_processor()
tacotron2 = bundle.get_tacotron2()
vocoder = bundle.get_vocoder()

# 将模型设置为评估模式
tacotron2.eval()
vocoder.eval()

# 输入文本
text = "Hello, this is a test of speech synthesis using Tacotron2."

# 处理文本
with torch.inference_mode():
    processed, lengths = processor(text)

    # 生成梅尔频谱
    mel_spec, mel_lengths, _ = tacotron2.infer(processed, lengths)

    # 使用声码器生成波形
    waveforms, lengths = vocoder(mel_spec, mel_lengths)

# 可视化波形
plt.figure(figsize=(10, 4))
plt.plot(waveforms[0].cpu().numpy())
plt.title("Generated Waveform")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.show()

# 保存音频文件
output_path = "output.wav"
torchaudio.save(output_path, waveforms.cpu(), sample_rate=22050)
print(f"Audio has been saved to {output_path}")
```

```
#文本到图片转换 ————————————————————————————————————————————————————————

import requests
import io
from PIL import Image

#API_URL = ""  # 你的API地址
#headers = {}  # 你的请求头
def query(payload):
    response = requests.post(API_URL, headers=headers, json=payload)
    return response.content

image_bytes = query({
    "inputs": "Astronaut␣riding␣a␣horse",      # 输入英文提示词
})

image = Image.open(io.BytesIO(image_bytes))



#光学字符识别 ————————————————————————————————————————————————————————

from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import requests

# 加载BLIP模型和处理器
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-
```

```
# 读取图片
image_path = 'cc2.jpg'  # 替换为你的图片路径
image = Image.open(image_path)

# 预处理图片
inputs = processor(images=image, return_tensors="pt")

# 生成描述
out = model.generate(**inputs)

# 解码并输出描述
description = processor.decode(out[0], skip_special_tokens=True)
print("图片描述:", description)


#文本生成 ----------------------------------------------------------------------

import requests
url = "https://www.gutenberg.org/cache/epub/74840/pg74840.txt"
response = requests.get(url)
# 检查请求是否成功
if response.status_code == 200:
    # 读取文本内容
    text_data = response.text

    # 将文本分割成行
    lines = text_data.split('\n')

    # 示例: 打印前几行
    for i, line in enumerate(lines[:10]):
        print(f"Line {i+1}: {line}")
else:
    print("Failed to download the file.")
```

```
    import torch
import torch.nn as nn
import torch.optim as optim
from transformers import GPT2LMHeadModel, GPT2Tokenizer


# 检查设备
device = "cuda" if torch.cuda.is_available() else "cpu"

# 使用 GPT-2 模型（带语言建模头）和对应的分词器
model = GPT2LMHeadModel.from_pretrained("gpt2").to(device)
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
vocab_size = tokenizer.vocab_size

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
epochs = 10



# 数据处理类
class Corpus:
    def __init__(self, data, tokenizer):
        self.data = data
        self.tokenizer = tokenizer

    def make_batch(self, batch_size):
        # 使用 GPT-2 分词器编码数据
        tokenized_data = [self.tokenizer.encode(text, return_tensors="pt").s
        # 拼接所有数据
```

```
        inputs = torch.cat(tokenized_data[:-1])[:batch_size].to(device)
        targets = torch.cat(tokenized_data[1:])[:batch_size].to(device)
        return inputs, targets

# 示例数据
data = text_data

corpus = Corpus(data, tokenizer)
batch_size =256

# 训练模型
for epoch in range(epochs):
    optimizer.zero_grad()
    inputs, targets = corpus.make_batch(batch_size)
    outputs = model(inputs.unsqueeze(0)).logits  # GPT2LMHeadModel 的 logits
    loss = criterion(outputs.view(-1, vocab_size), targets.view(-1))
# 计算损失
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 2 == 0:
        print(f"Epoch {epoch + 1}, Loss: {loss.item():.4f}")




# 文本生成函数
def generate_text(model, input_str, max_len=50):
    model.eval()
    input_ids = tokenizer.encode(input_str, return_tensors="pt").to(device)
    output_ids = input_ids.tolist()[0]

    with torch.no_grad():
        for _ in range(max_len):
```

```
            # 获取 logits 并预测下一个 token
            outputs = model(input_ids).logits
            next_token_id = torch.argmax(outputs[:, -1, :], dim=-1).item()
            if next_token_id == tokenizer.eos_token_id:
                break
            output_ids.append(next_token_id)
            input_ids = torch.tensor([output_ids], dtype=torch.long).to(devi

    return tokenizer.decode(output_ids, skip_special_tokens=True)


# 测试文本生成
input_str = "Kike arrived at"
generated_text = generate_text(model, input_str)
print("生成的文本:", generated_text)


# 测试文本生成
input_str = "I find it necessary"
generated_text = generate_text(model, input_str)
print("生成的文本:", generated_text)


#内容审核 ----------------------------------------------------------------------
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import classification_report

# 加载数据集
train_df = pd.read_csv('train.csv')
```

```
test_df = pd.read_csv('test.csv')
test_labels_df = pd.read_csv('test_labels.csv')
sample_submission_df = pd.read_csv('sample_submission.csv')

# 随机提取100000个样本
train_sample = train_df.sample(n=100000, random_state=12)
test_sample = test_df.sample(n=100000, random_state=12)

# 注意，test_labels中的ID需要与test_sample中的ID匹配
test_labels_sample = test_labels_df[test_labels_df['id'].isin(test_sample['i

# sample_submission中的ID也需要与test_sample中的ID匹配
sample_submission_sample = sample_submission_df[sample_submission_df['id'].i

# 保存样本数据
train_sample.to_csv('train_sample.csv', index=False)
test_sample.to_csv('test_sample.csv', index=False)
test_labels_sample.to_csv('test_labels_sample.csv', index=False)
sample_submission_sample.to_csv('sample_submission_sample.csv', index=False)
```

```
# 加载样本数据集
train_sample = pd.read_csv('train_sample.csv')

# 提取特征和标签
X = train_sample['comment_text']
y = train_sample[['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'i

# 分割数据集
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando
```

```
# 特征提取
vectorizer = TfidfVectorizer(max_features=10000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)

# 训练多标签分类模型
model = MultiOutputClassifier(LogisticRegression(max_iter=1000))
model.fit(X_train_tfidf, y_train)

# 预测和评估
y_pred = model.predict(X_val_tfidf)
print(classification_report(y_val, y_pred, target_names=y.columns))

def predict_labels(comment):
    comment_tfidf = vectorizer.transform([comment])
    prediction = model.predict(comment_tfidf)
    labels = y.columns
    return {label: bool(pred) for label, pred in zip(labels, prediction[0])}

    # 示例句子
    comment = "Those people are all garbage and don't deserve to live"
    prediction = predict_labels(comment)
    print(prediction)
    # 示例句子
    comment = "You idiot , you can't do anything well."
    prediction = predict_labels(comment)
    print(prediction)
```