1 Encoder-Decoder 中英文翻译实验

1.1 实验介绍

翻译任务在日常生活应用广泛，如手机中有各种翻译软件，可以满足人们交流、阅读的需求。本实验基于 seq2seq 编码器-解码器框架，结合 LSTM 单元实现英文转中文的翻译任务，框架示意图如下：

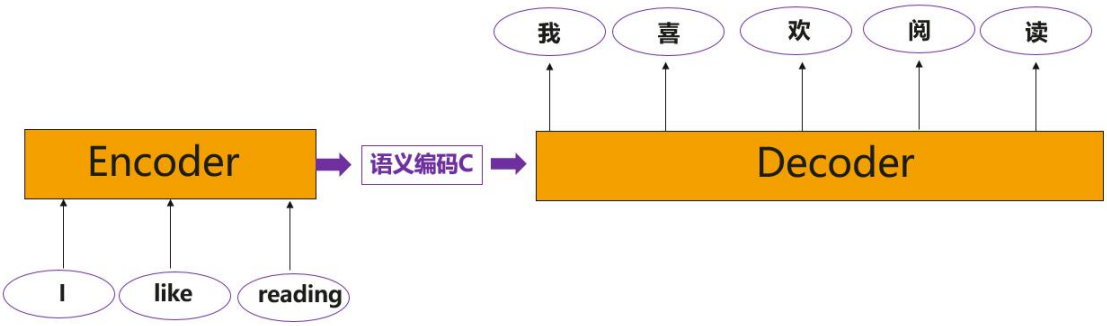

图 1 编码器-解码器框架示意图

1.2 实验预备知识

- 有文本预处理的基础。
- 有相应 Python 语言的编程基础。
- 有 Keras 的操作基础。
- 有神经网络 LSTM 的理论基础。

1.3 实验环境介绍

- ModelArts 平台：TensorFlow-1.13.1-Python3.6

1.4 实验总体设计

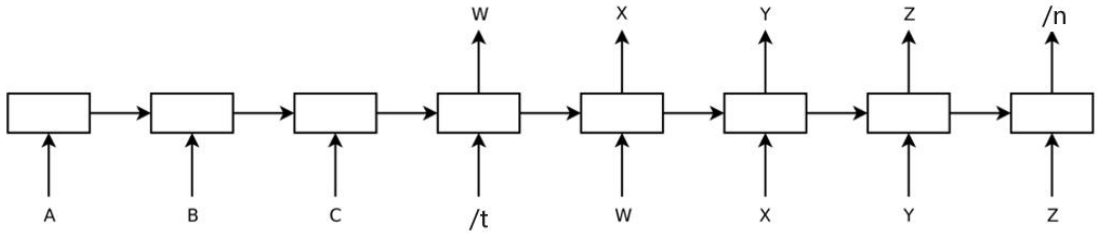

图 2 实验总体设计示意图

如图所示，encoder 接受输入 ABC，编码后将语义信息传递给 decoder，decoder 经过解码后获得 wxyz，/t 为起始符，/n 为结束符。

1.5 实验过程

本节将详细介绍实验的设计与实现。

1.5.1 导入实验环境

```
from tensorflow.keras.layers import Input,LSTM,Dense
from tensorflow.keras.models import Model,load_model
from tensorflow.keras.utils import plot_model
import pandas as pd
import numpy as np
```

1.5.2 配置参数

```
N_UNITS = 256        #LSTM 单元中隐藏层节点数
BATCH_SIZE = 64     #网络训练时每个批次的大小
EPOCH = 200           #训练的 epoch 数
NUM_SAMPLES = 1000   #训练数据数量，即样本条数
data_path = './cmn_zhsim.txt'     #训练数据路径(根据的路径设置)
```

1.5.3 读取数据并处理

读取数据并处理，载入英中翻译数据集，本数据集共有 20403（实际使用其中的 1000，数据太大要耗上五六个小时）条英中对应数据，并进行预处理。

```
df = pd.read_table(data_path,header=None).iloc[:NUM_SAMPLES,:]
df.columns=['inputs','targets']

#将每句中文句首加上'\t'作为起始标志，句末加上'\n'作为终止标志
df['targets'] = df['targets'].apply(lambda x: '\t'+x+'\n')
#英文句子列表
input_texts = df.inputs.values.tolist()
#中文句子列表
target_texts = df.targets.values.tolist()

#确定中英文各自包含的字符。df.unique()直接取 sum 可将 unique 数组中的各个句子拼接成一个长句子
input_characters = sorted(list(set(df.inputs.unique().sum())))
target_characters = sorted(list(set(df.targets.unique().sum())))

#输入数据的时刻 t 的长度，这里为最长的英文句子长度
INUPT_LENGTH = max([len(i) for i in input_texts])

#输出数据的时刻 t 的长度，这里为最长的中文句子长度
OUTPUT_LENGTH = max([len(i) for i in target_texts])

#每个时刻进入 encoder 的 lstm 单元的数据 xt 的维度，这里为英文中出现的字符数
INPUT_FEATURE_LENGTH = len(input_characters)

#每个时刻进入 decoder 的 lstm 单元的数据 xtxt 的维度，这里为中文中出现的字符数
OUTPUT_FEATURE_LENGTH = len(target_characters)
```

1.5.4 句子向量化

每条句子经过对字母转换成 one-hot 编码后，生成了 LSTM 需要的三维输入[n_samples, timestamp, one-hot feature]

```
encoder_input = np.zeros((NUM_SAMPLES,INUPT_LENGTH,INPUT_FEATURE_LENGTH))
```

```
decoder_input =
np.zeros((NUM_SAMPLES,OUTPUT_LENGTH,OUTPUT_FEATURE_LENGTH))
decoder_output =
np.zeros((NUM_SAMPLES,OUTPUT_LENGTH,OUTPUT_FEATURE_LENGTH))
#构建英文字符集的字典
input_dict = {char:index for index,char in enumerate(input_characters)}
#键值调换
input_dict_reverse = {index:char for index,char in enumerate(input_characters)}
#构建中文字符集的字典
target_dict = {char:index for index,char in enumerate(target_characters)}
#键值调换
target_dict_reverse = {index:char for index,char in enumerate(target_characters)}
#对句子进行字符级 one-hot 编码，将输入输出数据向量化
#encoder 的输入向量 one-hot
for seq_index,seq in enumerate(input_texts):
    for char_index, char in enumerate(seq):
        encoder_input[seq_index,char_index,input_dict[char]] = 1
#decoder 的输入输出向量 one-hot，训练模型时 decoder 的输入要比输出晚一个时间步，这样才能对输出监督
for seq_index,seq in enumerate(target_texts):
    for char_index,char in enumerate(seq):
        decoder_input[seq_index,char_index,target_dict[char]] = 1.0
        if char_index > 0:
            decoder_output[seq_index,char_index-1,target_dict[char]] = 1.0
```

1.5.5 创建模型

```
def create_model(n_input,n_output,n_units):
    #训练阶段
    #encoder
    #encoder 输入维度 n_input 为每个时间步的输入 xt 的维度，这里是用来 one-hot 的英文字符数
    encoder_input = Input(shape = (None, n_input))
```

```
    #n_units 为 LSTM 单元中每个门的神经元的个数, return_state 设为 True 时才会返回最后时刻的状态 h,c
     encoder = LSTM(n_units, return_state=True)
    #保留下来 encoder 的末状态作为 decoder 的初始状态
     _,encoder_h,encoder_c = encoder(encoder_input)
     encoder_state = [encoder_h,encoder_c]

    #decoder
    #decoder 的输入维度为中文字符数
    decoder_input = Input(shape = (None, n_output))

     #训练模型时需要 decoder 的输出序列来与结果对比优化，故 return_sequences 也要设为 True
     decoder = LSTM(n_units,return_sequences=True, return_state=True)
     #在训练阶段只需要用到 decoder 的输出序列，不需要用最终状态 h.c
     decoder_output, _, _ = decoder(decoder_input,initial_state=encoder_state)
     #输出序列经过全连接层得到结果
     decoder_dense = Dense(n_output,activation='softmax')
     decoder_output = decoder_dense(decoder_output)

    #生成的训练模型
    #第一个参数为训练模型的输入，包含了 encoder 和 decoder 的输入，第二个参数为模型的输出，包含了 decoder 的输出
     model = Model([encoder_input,decoder_input],decoder_output)

     #推理阶段，用于预测过程
    #推断模型—encoder，预测时对序列 predict，生成的 state 给 decoder
     encoder_infer = Model(encoder_input,encoder_state)

     #推断模型-decoder
     decoder_state_input_h = Input(shape=(n_units,))
     decoder_state_input_c = Input(shape=(n_units,))
```

```
    decoder_state_input = [decoder_state_input_h, decoder_state_input_c]#上个时刻的状态 h,c


    decoder_infer_output, decoder_infer_state_h, decoder_infer_state_c = decoder(decoder_input,initial_state=decoder_state_input)
    decoder_infer_state = [decoder_infer_state_h, decoder_infer_state_c]#当前时刻得到的状态
    decoder_infer_output = decoder_dense(decoder_infer_output)#当前时刻的输出
#参数输入和输出；输入：input+前一个状态的 state_input,输出：output+state_output
    decoder_infer = Model([decoder_input]+decoder_state_input,[decoder_infer_output]+decoder_infer_state)


    return model, encoder_infer, decoder_infer


model_train, encoder_infer, decoder_infer = create_model(INPUT_FEATURE_LENGTH, OUTPUT_FEATURE_LENGTH, N_UNITS)
```

思考题

结合 Seq2Seq 框架及内部使用单元思考，Seq2Seq 框架在编码-解码过程中是否存在信息丢失？具体表现在哪些方面？

1.5.6 编译模型

```
model_train.compile(optimizer='adam', loss='categorical_crossentropy')
```

训练的模型

```
#查看模型结构
model_train.summary()
```

结果：

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         (None, None, 73)          0
__________________________________________________________________________________________________
input_2 (InputLayer)         (None, None, 2128)        0
__________________________________________________________________________________________________
lstm_1 (LSTM)                [(None, 256), (None,      337920    input_1[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                [(None, None, 256),       2442240   input_2[0][0]
                                                                 lstm_1[0][1]
                                                                 lstm_1[0][2]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, None, 2128)        546896    lstm_2[0][0]
==================================================================================================
Total params: 3,327,056
Trainable params: 3,327,056
Non-trainable params: 0
__________________________________________________________________________________________________
```

图 3 训练模型的结构

从图中，我们可以看到，英文字符（含数字和符号）共有 73 个，中文中字符（含数字和符号）共有 2128 个，LSTM 中隐藏层节点数为 256 个。

编码器推理模型

```
encoder_infer.summary()
```

结果：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, None, 73)          0
_________________________________________________________________
lstm_1 (LSTM)                [(None, 256), (None, 256) 337920
=================================================================
Total params: 337,920
Trainable params: 337,920
Non-trainable params: 0
_________________________________________________________________
```

图 4 编码器推理模型的结构

解码器推理模型

```
decoder_infer.summary()
```

结果：

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_2 (InputLayer)         (None, None, 2128)        0
__________________________________________________________________________________________________
input_3 (InputLayer)         (None, 256)               0
__________________________________________________________________________________________________
input_4 (InputLayer)         (None, 256)               0
__________________________________________________________________________________________________
lstm_2 (LSTM)                [(None, None, 256),       2442240     input_2[0][0]
                                                                   input_3[0][0]
                                                                   input_4[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, None, 2128)        546896      lstm_2[1][0]
==================================================================================================
Total params: 2,989,136
Trainable params: 2,989,136
Non-trainable params: 0
__________________________________________________________________________________________________
```

图 5 解码器推理模型的结构

1.5.7 模型训练

```
model_train.fit([encoder_input,decoder_input],decoder_output,batch_size=BATCH_SIZE,
epochs=EPOCH,validation_split=0.2)
```

```
800/800 [==============================] - 5s 6ms/sample - loss: 0.0682 - val_loss: 2.6471
Epoch 198/200
800/800 [==============================] - 5s 6ms/sample - loss: 0.0666 - val_loss: 2.6236
Epoch 199/200
800/800 [==============================] - 5s 6ms/sample - loss: 0.0620 - val_loss: 2.6501
Epoch 200/200
800/800 [==============================] - 5s 6ms/sample - loss: 0.0583 - val_loss: 2.6291
```

图 6 模型训练

1.5.8 构建测试函数并测试

```
def predict_chinese(source,encoder_inference, decoder_inference, n_steps, features):
    #先通过推理 encoder 获得预测输入序列的隐状态
    state = encoder_inference.predict(source)
    #第一个字符'\t',为起始标志
    predict_seq = np.zeros((1,1,features))
    predict_seq[0,0,target_dict['\t']] = 1


    output = ''
    #开始对 encoder 获得的隐状态进行推理
    #每次循环用上次预测的字符作为输入来预测下一次的字符，直到预测出了终止符
    for i in range(n_steps):     #n_steps 为句子最大长度
```

```
            #给 decoder 输入上一个时刻的 h,c 隐状态，以及上一次的预测字符 predict_seq
            yhat,h,c = decoder_inference.predict([predict_seq]+state)
            #注意，这里的 yhat 为 Dense 之后输出的结果，因此与 h 不同
            #每次预测都是最后一个中文字符
            char_index = np.argmax(yhat[0,-1,:])
            char = target_dict_reverse[char_index]
            output += char
            #更新 state，本次状态做为下一次的初始状态继续传递
            state = [h,c]
            #前一状态的输出结果会作为下一状态的输入信息
            predict_seq = np.zeros((1,1,features))
            predict_seq[0,0,char_index] = 1
            if char == '\n':#预测到了终止符则停下来
                break
    return output
for i in range(100,200):
    test = encoder_input[i:i+1,:,:]  #i:i+1 保持数组是三维
    out =
predict_chinese(test,encoder_infer,decoder_infer,OUTPUT_LENGTH,OUTPUT_FEATURE
_LENGTH)
    print(input_texts[i])
    print(out)
```

结果：

```
Try some.
试试吧。

Who died?
谁死了？

Birds fly.
鸟类飞行。

Call home!
打电话回家！

Catch him.
抓住他。

Come home.
回家吧。
```

图 7 翻译结果

1.6 创新设计

阅读以下两篇文献，了解 dot-product attention 和 bahdanan attention 的工作原理，在以上基本的 seq2seq 模型中尝试添加 dot-product attention（不带学习参数的）和 bahdanan attention（带学习参数）并做前后对比。

论文链接：

[1]Luong M T, Pham H, Manning C D. Effective approaches to attention-based neural machine translation[J]. arXiv preprint arXiv:1508.04025, 2015.

[2]DZMITRY B, CHO K, YOSHUA B. Neural machine translation by joinly learning to align and translate.[J]. arXiv preprint arXiv:1409.0473, 2014.