《自然语言处理》

-机器翻译实验



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验总览 2](#_Toc97907797)

[1.1 实验背景 2](#_Toc97907798)

[1.2 实验目的 2](#_Toc97907799)

[1.3 实验清单 2](#_Toc97907800)

[1.4 实验开发环境 2](#_Toc97907801)

[2 Seq2Seq中英文翻译实验 3](#_Toc97907802)

[2.1 实验简介 3](#_Toc97907803)

[2.2 实验环境 3](#_Toc97907804)

[2.3 实验步骤 3](#_Toc97907805)

[2.3.1 实验准备 3](#_Toc97907806)

[2.3.2 实验过程 5](#_Toc97907807)

[2.4 实验小结 16](#_Toc97907808)

[3 Transformer中英文翻译实验 17](#_Toc97907809)

[3.1 实验简介 17](#_Toc97907810)

[3.2 实验环境 17](#_Toc97907811)

[3.3 预备知识 17](#_Toc97907812)

[3.4 实验步骤 19](#_Toc97907813)

[3.4.1 实验准备 19](#_Toc97907814)

[3.4.2 实验过程 21](#_Toc97907815)

[3.5 实验小结 32](#_Toc97907816)

[4 附录：ModelArts开发环境搭建 33](#_Toc97907817)

# 实验总览

## 实验背景

在实际生活中，机器翻译是人工智能技术比较广泛的一个应用，在之前的深度学习过程中，我们知道，循环神经网络能将输入序列映射成等长的输出序列，但在机器翻译应用中，输入序列与输出序列的长度通常不一样。序列到序列（Sequence to Sequene, Seq2Seq）的映射框架，就是用来解决这一问题，它能将一个可变长序列映射到另一个可变长序列。本章实验将探索Seq2Seq基础模型在机器翻译中的应用，以及Attention注意力机制、Transformer模型对基础Seq2Seq模型的改进。

## 实验目的

本章实验的主要目的是通过中英翻译实验，学员能深入地了解神经网络在NLP机器翻译领域的应用。通过案例代码的学习，重点理解seq2seq解码器-编码器框架、attention注意力机制、transformer架构。

## 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| Seq2Seq中英文翻译 | 基于seq2seq架构实现中英文的翻译 | 高级 | Python3.7.5、MindSpore1.5 | ModelArts Ascend Notebook |
| Transformer中英文翻译 | 基于Transformer架构实现中英文翻译 | 高级 | Python3.7.5、MindSpore1.5 | ModelArts Ascend Notebook |

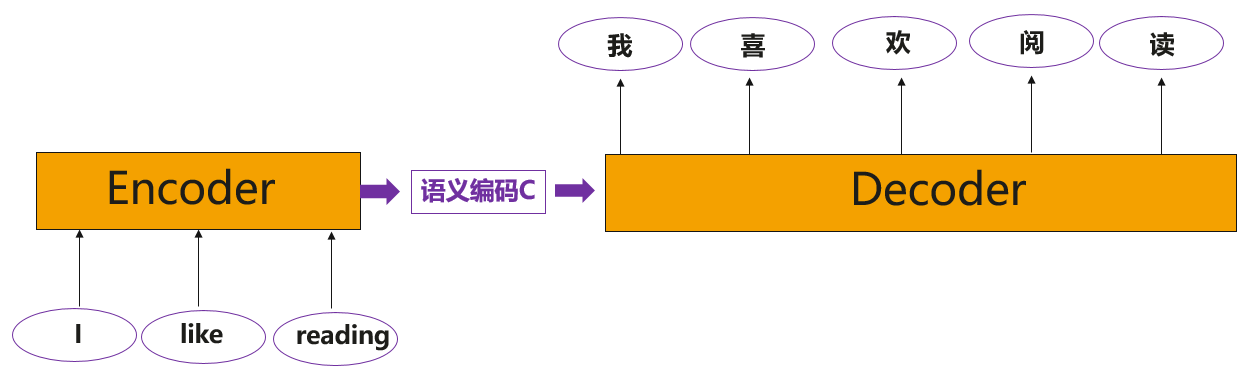
## 实验开发环境

Mindspore-1.5

若选择在华为云ModelArts上快速搭建开发环境，可参考文末附录：ModelArts开发环境搭建。

# Seq2Seq中英文翻译实验

## 实验简介

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GRU单元实现英文转中文的翻译任务，框架示意图如下： 

GRU（门递归单元）是一种递归神经网络算法，就像LSTM（长短期存储器）一样。它是由Kyunghyun Cho、Bart van Merrienboer等在2014年的文章“使用RNN编码器-解码器学习短语表示用于统计机器翻译”中提出的。本文提出了一种新的神经网络模型RNN Encoder-Decoder，该模型由两个递归神经网络（RNN）组成，为了提高翻译任务的效果，我们还参考了“神经网络的序列到序列学习”和“联合学习对齐和翻译的神经机器翻译”。

## 实验环境

华为云ModelArts Ascend Notebook环境，创建方式如下方2.3.1所示。

## 实验步骤

### 实验准备

进入ModelArts开发环境

参考文末附录，创建ModelArts上的开发环境Notebook并进入。

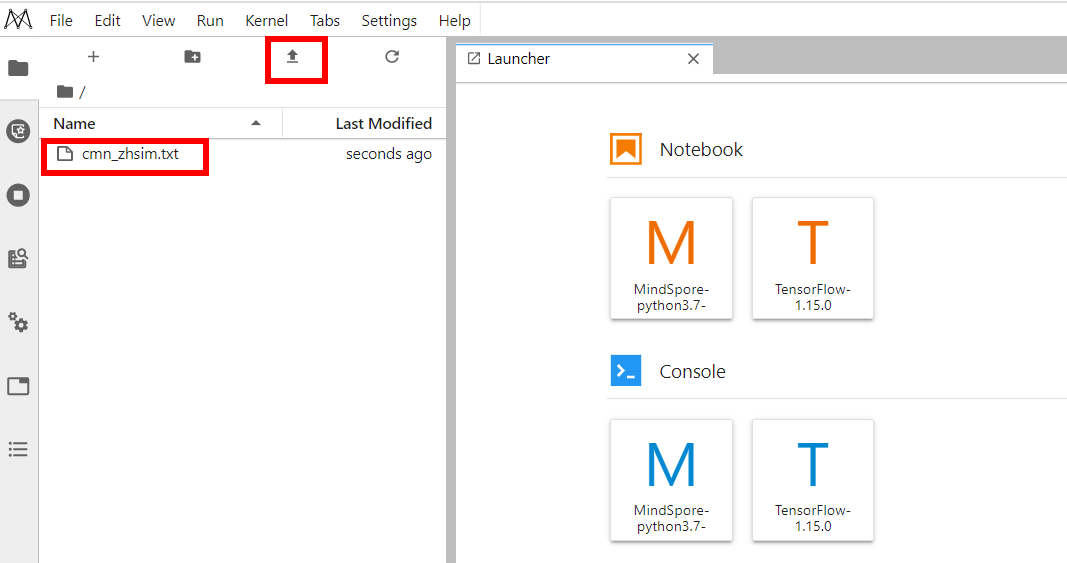
下载自然语言处理包

下载链接：<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/NLP/NLP.zip>

并在《自然语言处理》中的“实验指导书”-“机器翻译”模块下，获取Seq2Seq的cmn\_zhsim.txt文件。

上传实验数据

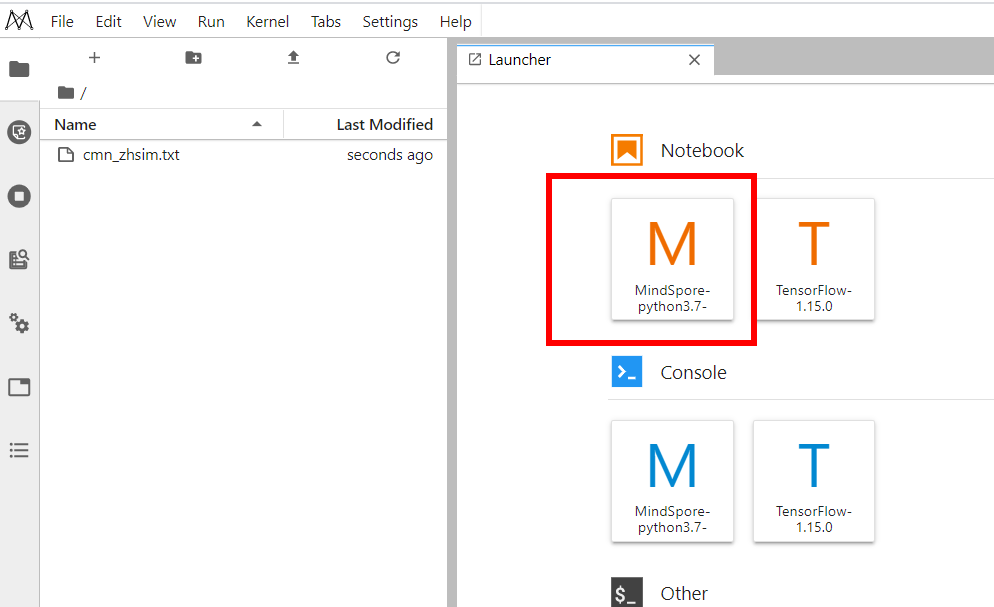
当Notebook状态变为“运行中”时，点击右侧“打开”按钮打开Notebook。点击如下图所示的上传按钮，选择本地的“cmn\_zhsim.txt”文件至服务器中。



上传实验数据

进入Notebook实验环境

选择右侧“MindSpore-python3.7-aarch64”按钮，进入Notebook实验环境。



进入Notebook实验环境

### 实验过程

导入实验所需的第三方模块

输入：

import os

import numpy as np

import re

import sys

import random

import unicodedata

import math

from mindspore import Tensor, nn, Model, context

from mindspore.train.serialization import load\_param\_into\_net, load\_checkpoint

from mindspore.train.callback import LossMonitor, CheckpointConfig, ModelCheckpoint, TimeMonitor

from mindspore import dataset as ds

from mindspore.mindrecord import FileWriter

from mindspore import Parameter

from mindspore.nn.loss.loss import \_Loss

from mindspore.ops import functional as F

from mindspore.ops import operations as P

from mindspore.common import dtype as mstype

设置运行环境

输入：

context.set\_context(mode=context.GRAPH\_MODE, save\_graphs=False, device\_target='Ascend')

查看数据内容

输入：

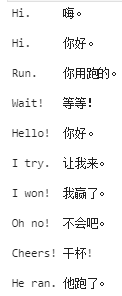
#查看训练数据内容前10行内容

with open("cmn\_zhsim.txt", 'r', encoding='utf-8') as f:

for i in range(10):

print(f.readline())

输出：



训练数据前10行

定义数据预处理函数

输入：

EOS = "<eos>"

SOS = "<sos>"

MAX\_SEQ\_LEN=10

#我们需要将字符转化为ASCII编码

#并全部转化为小写字母，并修剪大部分标点符号

#除了(a-z, A-Z, ".", "?", "!", ",")这些字符外，全替换成空格

def unicodeToAscii(s):

return ''.join(

c for c in unicodedata.normalize('NFD', s)

if unicodedata.category(c) != 'Mn'

)

def normalizeString(s):

s = s.lower().strip()

s = unicodeToAscii(s)

s = re.sub(r"([.!?])", r" \1", s)

s = re.sub(r"[^a-zA-Z.!?]+", r" ", s)

return s

def prepare\_data(data\_path, vocab\_save\_path, max\_seq\_len):

with open(data\_path, 'r', encoding='utf-8') as f:

data = f.read()

# 读取文本文件，按行分割，再将每行分割成语句对

data = data.split('\n')

# 截取前2000行数据进行训练

data = data[:2000]

# 分割每行中的中英文

en\_data = [normalizeString(line.split('\t')[0]) for line in data]

ch\_data = [line.split('\t')[1] for line in data]

# 利用集合，获得中英文词汇表

en\_vocab = set(' '.join(en\_data).split(' '))

id2en = [EOS] + [SOS] + list(en\_vocab)

en2id = {c:i for i,c in enumerate(id2en)}

en\_vocab\_size = len(id2en)

# np.savetxt(os.path.join(vocab\_save\_path, 'en\_vocab.txt'), np.array(id2en), fmt='%s')

ch\_vocab = set(''.join(ch\_data))

id2ch = [EOS] + [SOS] + list(ch\_vocab)

ch2id = {c:i for i,c in enumerate(id2ch)}

ch\_vocab\_size = len(id2ch)

# np.savetxt(os.path.join(vocab\_save\_path, 'ch\_vocab.txt'), np.array(id2ch), fmt='%s')

# 将句子用词汇表id表示

en\_num\_data = np.array([[1] + [int(en2id[en]) for en in line.split(' ')] + [0] for line in en\_data])

ch\_num\_data = np.array([[1] + [int(ch2id[ch]) for ch in line] + [0] for line in ch\_data])

#将短句子扩充到统一的长度

for i in range(len(en\_num\_data)):

num = max\_seq\_len + 1 - len(en\_num\_data[i])

if(num >= 0):

en\_num\_data[i] += [0]\*num

else:

en\_num\_data[i] = en\_num\_data[i][:max\_seq\_len] + [0]

for i in range(len(ch\_num\_data)):

num = max\_seq\_len + 1 - len(ch\_num\_data[i])

if(num >= 0):

ch\_num\_data[i] += [0]\*num

else:

ch\_num\_data[i] = ch\_num\_data[i][:max\_seq\_len] + [0]

np.savetxt(os.path.join(vocab\_save\_path, 'en\_vocab.txt'), np.array(id2en), fmt='%s')

np.savetxt(os.path.join(vocab\_save\_path, 'ch\_vocab.txt'), np.array(id2ch), fmt='%s')

return en\_num\_data, ch\_num\_data, en\_vocab\_size, ch\_vocab\_size

获得mindrecord文件

输入：

#将处理后的数据保存为mindrecord文件，方便后续训练

def convert\_to\_mindrecord(data\_path, mindrecord\_save\_path, max\_seq\_len):

en\_num\_data, ch\_num\_data, en\_vocab\_size, ch\_vocab\_size = prepare\_data(data\_path, mindrecord\_save\_path, max\_seq\_len)

# 输出前十行英文句子对应的数据

for i in range(10):

print(en\_num\_data[i])

data\_list\_train = []

for en, ch in zip(en\_num\_data, ch\_num\_data):

en = np.array(en).astype(np.int32)

ch = np.array(ch).astype(np.int32)

data\_json = {"encoder\_data": en.reshape(-1),

"decoder\_data": ch.reshape(-1)}

data\_list\_train.append(data\_json)

data\_list\_eval = random.sample(data\_list\_train, 20)

data\_dir = os.path.join(mindrecord\_save\_path, "gru\_train.mindrecord")

writer = FileWriter(data\_dir)

schema\_json = {"encoder\_data": {"type": "int32", "shape": [-1]},

"decoder\_data": {"type": "int32", "shape": [-1]}}

writer.add\_schema(schema\_json, "gru\_schema")

writer.write\_raw\_data(data\_list\_train)

writer.commit()

data\_dir = os.path.join(mindrecord\_save\_path, "gru\_eval.mindrecord")

writer = FileWriter(data\_dir)

writer.add\_schema(schema\_json, "gru\_schema")

writer.write\_raw\_data(data\_list\_eval)

writer.commit()

print("en\_vocab\_size: ", en\_vocab\_size)

print("ch\_vocab\_size: ", ch\_vocab\_size)

return en\_vocab\_size, ch\_vocab\_size

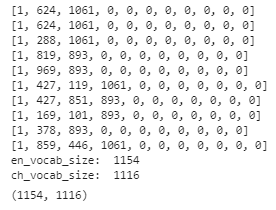
输入：

if not os.path.exists("./preprocess"):

os.mkdir('./preprocess')

convert\_to\_mindrecord("cmn\_zhsim.txt", './preprocess', MAX\_SEQ\_LEN)

输出：



处理后的数据信息

超参数设置

输入：

from easydict import EasyDict as edict

# CONFIG

cfg = edict({

'en\_vocab\_size': 1154,

'ch\_vocab\_size': 1116,

'max\_seq\_length': 10,

'hidden\_size': 1024,

'batch\_size': 16,

'eval\_batch\_size': 1,

'learning\_rate': 0.001,

'momentum': 0.9,

'num\_epochs': 15,

'save\_checkpoint\_steps': 125,

'keep\_checkpoint\_max': 10,

'dataset\_path':'./preprocess',

'ckpt\_save\_path':'./ckpt',

'checkpoint\_path':'./ckpt/gru-15\_125.ckpt'

})

定义读取mindrecord函数

输入：

def target\_operation(encoder\_data, decoder\_data):

encoder\_data = encoder\_data[1:]

target\_data = decoder\_data[1:]

decoder\_data = decoder\_data[:-1]

return encoder\_data, decoder\_data, target\_data

def eval\_operation(encoder\_data, decoder\_data):

encoder\_data = encoder\_data[1:]

decoder\_data = decoder\_data[:-1]

return encoder\_data, decoder\_data

def create\_dataset(data\_home, batch\_size, repeat\_num=1, is\_training=True, device\_num=1, rank=0):

if is\_training:

data\_dir = os.path.join(data\_home, "gru\_train.mindrecord")

else:

data\_dir = os.path.join(data\_home, "gru\_eval.mindrecord")

data\_set = ds.MindDataset(data\_dir, columns\_list=["encoder\_data","decoder\_data"],

num\_parallel\_workers=4,

num\_shards=device\_num, shard\_id=rank)

if is\_training:

operations = target\_operation

data\_set = data\_set.map(operations=operations,

input\_columns=["encoder\_data","decoder\_data"],

output\_columns=["encoder\_data","decoder\_data","target\_data"],

column\_order=["encoder\_data","decoder\_data","target\_data"])

else:

operations = eval\_operation

data\_set = data\_set.map(operations=operations,

input\_columns=["encoder\_data","decoder\_data"],

output\_columns=["encoder\_data","decoder\_data"],

column\_order=["encoder\_data","decoder\_data"])

data\_set = data\_set.shuffle(buffer\_size=data\_set.get\_dataset\_size())

data\_set = data\_set.batch(batch\_size=batch\_size, drop\_remainder=True)

data\_set = data\_set.repeat(count=repeat\_num)

return data\_set

输入：

ds\_train = create\_dataset(cfg.dataset\_path, cfg.batch\_size)

定义GRU单元

输入：

def gru\_default\_state(batch\_size, input\_size, hidden\_size, num\_layers=1, bidirectional=False):

'''Weight init for gru cell'''

stdv = 1 / math.sqrt(hidden\_size)

weight\_i = Parameter(Tensor(

np.random.uniform(-stdv, stdv, (input\_size, 3\*hidden\_size)).astype(np.float32)),

name='weight\_i')

weight\_h = Parameter(Tensor(

np.random.uniform(-stdv, stdv, (hidden\_size, 3\*hidden\_size)).astype(np.float32)),

name='weight\_h')

bias\_i = Parameter(Tensor(

np.random.uniform(-stdv, stdv, (3\*hidden\_size)).astype(np.float32)), name='bias\_i')

bias\_h = Parameter(Tensor(

np.random.uniform(-stdv, stdv, (3\*hidden\_size)).astype(np.float32)), name='bias\_h')

return weight\_i, weight\_h, bias\_i, bias\_h

class GRU(nn.Cell):

def \_\_init\_\_(self, config, is\_training=True):

super(GRU, self).\_\_init\_\_()

if is\_training:

self.batch\_size = config.batch\_size

else:

self.batch\_size = config.eval\_batch\_size

self.hidden\_size = config.hidden\_size

self.weight\_i, self.weight\_h, self.bias\_i, self.bias\_h = \

gru\_default\_state(self.batch\_size, self.hidden\_size, self.hidden\_size)

self.rnn = P.DynamicGRUV2()

self.cast = P.Cast()

def construct(self, x, hidden):

x = self.cast(x, mstype.float16)

y1, h1, \_, \_, \_, \_ = self.rnn(x, self.weight\_i, self.weight\_h, self.bias\_i, self.bias\_h, None, hidden)

return y1, h1

定义Encoder

输入：

class Encoder(nn.Cell):

def \_\_init\_\_(self, config, is\_training=True):

super(Encoder, self).\_\_init\_\_()

self.vocab\_size = config.en\_vocab\_size

self.hidden\_size = config.hidden\_size

if is\_training:

self.batch\_size = config.batch\_size

else:

self.batch\_size = config.eval\_batch\_size

self.trans = P.Transpose()

self.perm = (1, 0, 2)

self.embedding = nn.Embedding(self.vocab\_size, self.hidden\_size)

self.gru = GRU(config, is\_training=is\_training).to\_float(mstype.float16)

self.h = Tensor(np.zeros((self.batch\_size, self.hidden\_size)).astype(np.float16))

def construct(self, encoder\_input):

embeddings = self.embedding(encoder\_input)

embeddings = self.trans(embeddings, self.perm)

output, hidden = self.gru(embeddings, self.h)

return output, hidden

定义Decoder

输入：

class Decoder(nn.Cell):

def \_\_init\_\_(self, config, is\_training=True, dropout=0.1):

super(Decoder, self).\_\_init\_\_()

self.vocab\_size = config.ch\_vocab\_size

self.hidden\_size = config.hidden\_size

self.max\_len = config.max\_seq\_length

self.trans = P.Transpose()

self.perm = (1, 0, 2)

self.embedding = nn.Embedding(self.vocab\_size, self.hidden\_size)

self.dropout = nn.Dropout(1-dropout)

self.attn = nn.Dense(self.hidden\_size, self.max\_len)

self.softmax = nn.Softmax(axis=2)

self.bmm = P.BatchMatMul()

self.concat = P.Concat(axis=2)

self.attn\_combine = nn.Dense(self.hidden\_size \* 2, self.hidden\_size)

self.gru = GRU(config, is\_training=is\_training).to\_float(mstype.float16)

self.out = nn.Dense(self.hidden\_size, self.vocab\_size)

self.logsoftmax = nn.LogSoftmax(axis=2)

self.cast = P.Cast()

def construct(self, decoder\_input, hidden, encoder\_output):

embeddings = self.embedding(decoder\_input)

embeddings = self.dropout(embeddings)

# calculate attn

attn\_weights = self.softmax(self.attn(embeddings)) # [1,1,10]

encoder\_output = self.trans(encoder\_output, self.perm)

attn\_applied = self.bmm(attn\_weights, self.cast(encoder\_output,mstype.float32))

output = self.concat((embeddings, attn\_applied))

output = self.attn\_combine(output)

embeddings = self.trans(embeddings, self.perm)

output, hidden = self.gru(embeddings, hidden)

output = self.cast(output, mstype.float32)

output = self.out(output)

output = self.logsoftmax(output)

return output, hidden, attn\_weights

定义Seq2Seq整体结构

输入：

class Seq2Seq(nn.Cell):

def \_\_init\_\_(self, config, is\_train=True):

super(Seq2Seq, self).\_\_init\_\_()

self.max\_len = config.max\_seq\_length

self.is\_train = is\_train

self.encoder = Encoder(config, is\_train)

self.decoder = Decoder(config, is\_train)

self.expanddims = P.ExpandDims()

self.squeeze = P.Squeeze(axis=0)

self.argmax = P.ArgMaxWithValue(axis=int(2), keep\_dims=True)

self.concat = P.Concat(axis=1)

self.concat2 = P.Concat(axis=0)

self.select = P.Select()

def construct(self, src, dst):

encoder\_output, hidden = self.encoder(src)

decoder\_hidden = self.squeeze(encoder\_output[self.max\_len-2:self.max\_len-1:1, ::, ::])

if self.is\_train:

outputs, \_ = self.decoder(dst, decoder\_hidden, encoder\_output)

else:

decoder\_input = dst[::,0:1:1]

decoder\_outputs = ()

for i in range(0, self.max\_len):

decoder\_output, decoder\_hidden, \_ = self.decoder(decoder\_input,

decoder\_hidden, encoder\_output)

decoder\_hidden = self.squeeze(decoder\_hidden)

decoder\_output, \_ = self.argmax(decoder\_output)

decoder\_output = self.squeeze(decoder\_output)

decoder\_outputs += (decoder\_output,)

decoder\_input = decoder\_output

outputs = self.concat(decoder\_outputs)

return outputs

定义损失函数

输入：

class NLLLoss(\_Loss):

'''

NLLLoss function

'''

def \_\_init\_\_(self, reduction='mean'):

super(NLLLoss, self).\_\_init\_\_(reduction)

self.one\_hot = P.OneHot()

self.reduce\_sum = P.ReduceSum()

def construct(self, logits, label):

label\_one\_hot = self.one\_hot(label, F.shape(logits)[-1], F.scalar\_to\_array(1.0),

F.scalar\_to\_array(0.0))

#print('NLLLoss label\_one\_hot:',label\_one\_hot, label\_one\_hot.shape)

#print('NLLLoss logits:',logits, logits.shape)

#print('xxx:', logits \* label\_one\_hot)

loss = self.reduce\_sum(-1.0 \* logits \* label\_one\_hot, (1,))

return self.get\_loss(loss)

class WithLossCell(nn.Cell):

def \_\_init\_\_(self, backbone, config):

super(WithLossCell, self).\_\_init\_\_(auto\_prefix=False)

self.\_backbone = backbone

self.batch\_size = config.batch\_size

self.onehot = nn.OneHot(depth=config.ch\_vocab\_size)

self.\_loss\_fn = NLLLoss()

self.max\_len = config.max\_seq\_length

self.squeeze = P.Squeeze()

self.cast = P.Cast()

self.argmax = P.ArgMaxWithValue(axis=1, keep\_dims=True)

self.print = P.Print()

def construct(self, src, dst, label):

out = self.\_backbone(src, dst)

loss\_total = 0

for i in range(self.batch\_size):

loss = self.\_loss\_fn(self.squeeze(out[::,i:i+1:1,::]),

self.squeeze(label[i:i+1:1, ::]))

loss\_total += loss

loss = loss\_total / self.batch\_size

return loss

定义训练网络与优化器

输入：

network = Seq2Seq(cfg)

network = WithLossCell(network, cfg)

optimizer = nn.Adam(network.trainable\_params(), learning\_rate=cfg.learning\_rate, beta1=0.9, beta2=0.98)

model = Model(network, optimizer=optimizer)

定义回调函数、构建模型

输入：

loss\_cb = LossMonitor()

config\_ck = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps, keep\_checkpoint\_max=cfg.keep\_checkpoint\_max)

ckpoint\_cb = ModelCheckpoint(prefix="gru", directory=cfg.ckpt\_save\_path, config=config\_ck)

time\_cb = TimeMonitor(data\_size=ds\_train.get\_dataset\_size())

callbacks = [time\_cb, ckpoint\_cb, loss\_cb]

启动训练

输入：

model.train(cfg.num\_epochs, ds\_train, callbacks=callbacks, dataset\_sink\_mode=True)

定义推理网络

输入：

class InferCell(nn.Cell):

def \_\_init\_\_(self, network, config):

super(InferCell, self).\_\_init\_\_(auto\_prefix=False)

self.expanddims = P.ExpandDims()

self.network = network

def construct(self, src, dst):

out = self.network(src, dst)

return out

输入：

network = Seq2Seq(cfg,is\_train=False)

network = InferCell(network, cfg)

network.set\_train(False)

parameter\_dict = load\_checkpoint(cfg.checkpoint\_path)

load\_param\_into\_net(network, parameter\_dict)

model = Model(network)

定义翻译测试函数

输入：

#打开词汇表

with open(os.path.join(cfg.dataset\_path,"en\_vocab.txt"), 'r', encoding='utf-8') as f:

data = f.read()

en\_vocab = list(data.split('\n'))

with open(os.path.join(cfg.dataset\_path,"ch\_vocab.txt"), 'r', encoding='utf-8') as f:

data = f.read()

ch\_vocab = list(data.split('\n'))

输入：

def translate(str\_en):

max\_seq\_len = 10

str\_vocab = normalizeString(str\_en).split(' ')

print("English",str(str\_vocab))

str\_id = [1]

for i in str\_vocab:

str\_id += [en\_vocab.index(i)]

num = max\_seq\_len + 1 - len(str\_id)

if(num >= 0):

str\_id += [0]\*num

else:

str\_id = str\_id[:max\_seq\_len] + [0]

str\_id = Tensor(np.array([str\_id[1:]]).astype(np.int32))

out\_id = [1]+[0]\*10

out\_id = Tensor(np.array([out\_id[:-1]]).astype(np.int32))

output = network(str\_id, out\_id)

out= ''

for x in output[0].asnumpy():

if x == 0:

break

out += ch\_vocab[x]

print("中文",out)

离线加载预测

在线推理测试：

输入：

translate('I love Tom ')

输出：



预测结果

## 实验小结

本节实验使用MindSpore实现了一个带有Attention注意力机制的Seq2Seq机器翻译模型，通过实验，使学员了解机器翻译的数据准备过程，以及如何构建编码器-解码器模型、注意力机制。

# Transformer中英文翻译实验

## 实验简介

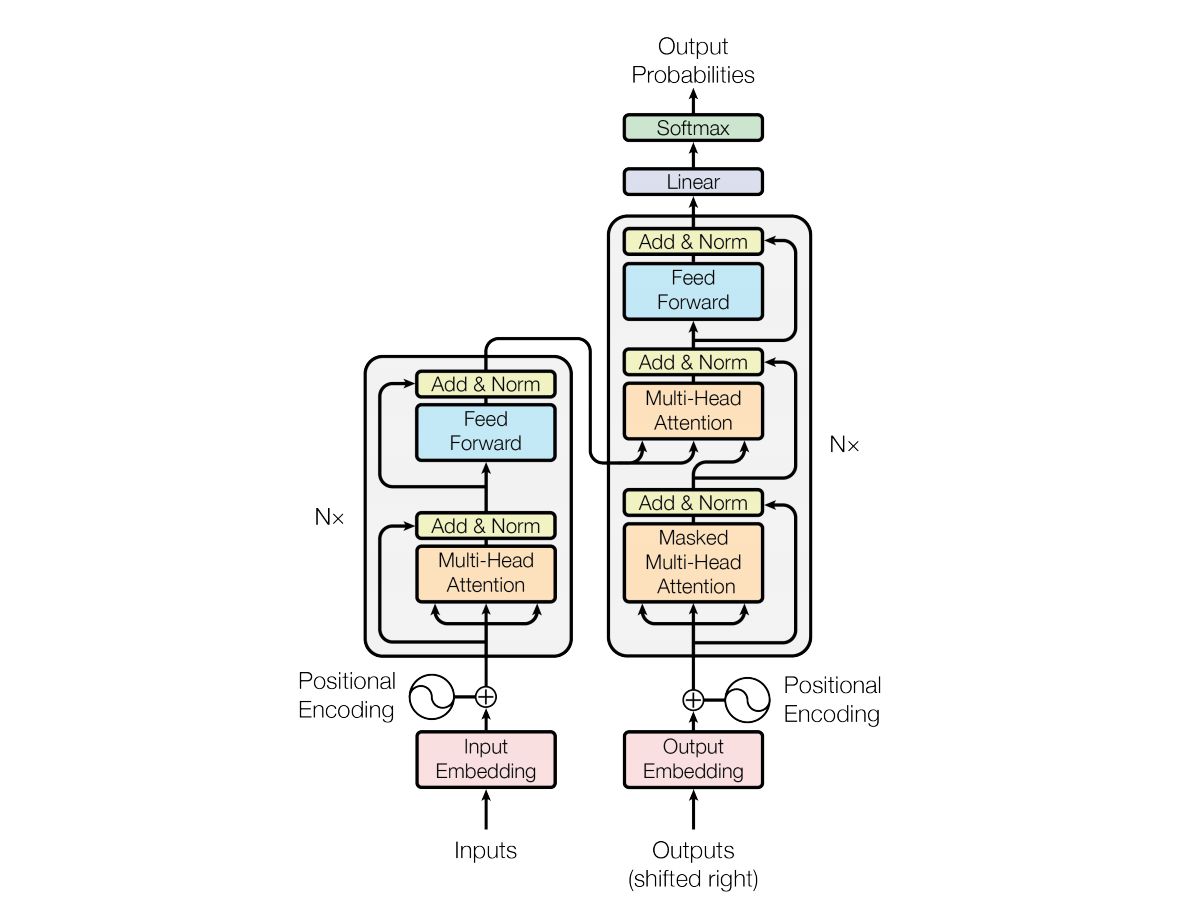
Transformer是谷歌的研究人员提出的一种全新的模型，Transformer在被提出之后，很快就席卷了整个自然语言处理领域。与循环神经网络等传统模型不同，Transformer模型仅仅使用一种被称作自注意力机制的方法和标准的前馈神经网络，完全不依赖任何循环单元或者卷积操作。自注意力机制的优点在于可以直接对序列中任意两个单元之间的关系进行建模，这使得长距离依赖等问题可以更好地被求解。本节实验将探索Transformer在机器翻译任务中的应用，实验将使用华为自研的MindSpore框架实现基于Transformer的中英文翻译。

## 实验环境

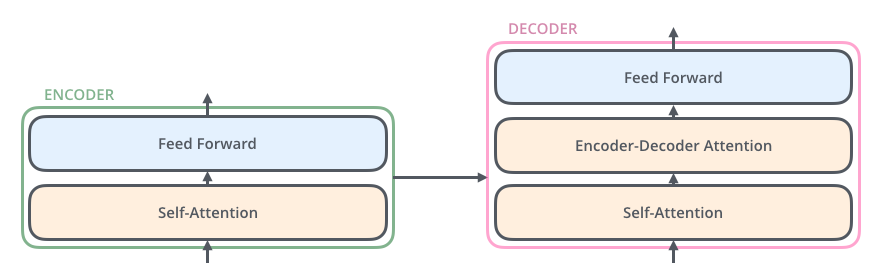
ModelArts Ascend Notebook环境，MindSpore1.5

## 预备知识

Transformer网络如下图所示，其中左边为编码网络，右边为解码网络。



Transformer架构

每一个编码器在结构上都是一样的，但它们的权重参数是不同的。每一个编码器里面，可以分为 2 层（Self-Attention 层、前馈神经网络）。输入编码器的文本数据，首先会经过一个 Self Attention 层，这个层处理一个词的时候，不仅会使用这个词本身的信息，也会使用句子中其他词的信息（可以类比为：当我们翻译一个词的时候，不仅会只关注当前的词，也会关注这个词的上下文的其他词的信息）。接下来，Self Attention 层的输出会经过前馈神经网络。同理，解码器也具有这两层，但是这两层中间还插入了一个 Encoder-Decoder Attention 层，这个层能帮助解码器聚焦于输入句子的相关部分（类似于 seq2seq 模型 中的 Attention）。

Encoder-Decoder Attention

## 实验步骤

### 实验准备

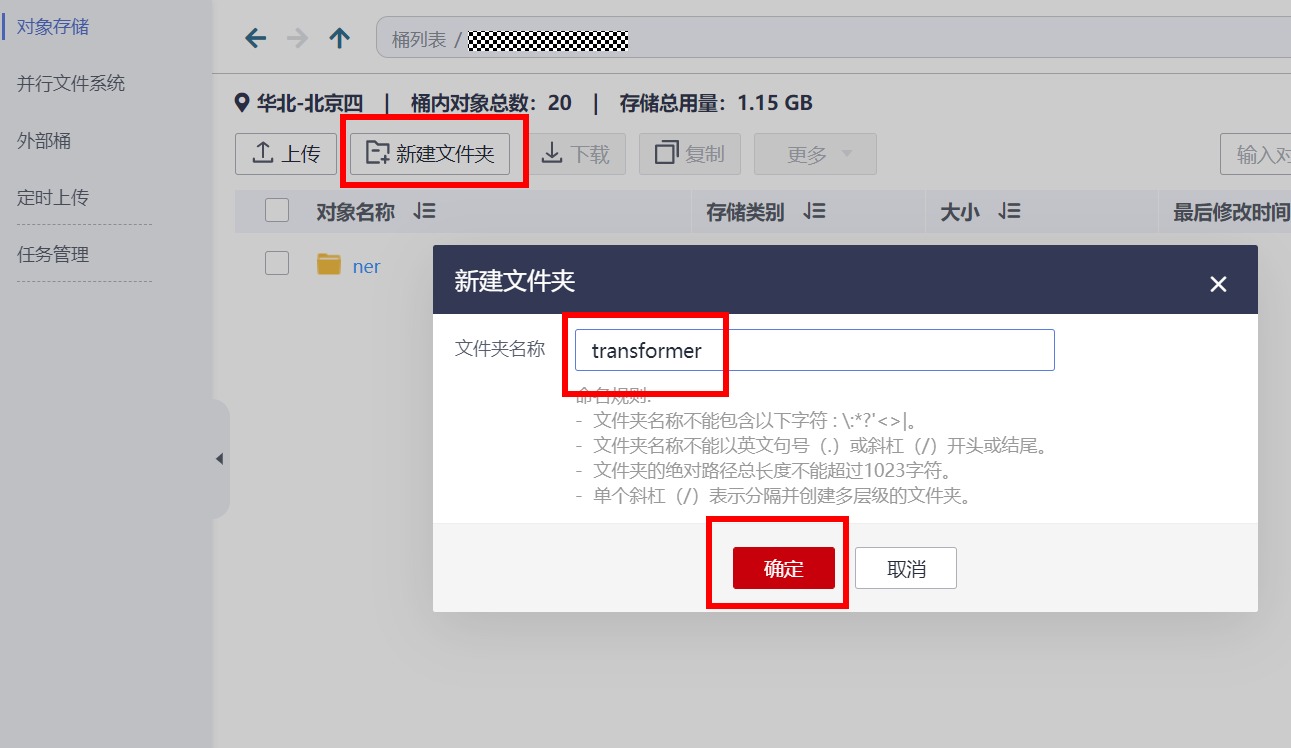
OBS创建项目文件夹

使用OBS Brower+登录OBS（注：OBS Browser+是一款用于访问和管理对象存储服务OBS的图形化工具，支持完善的桶管理和对象管理操作。OBS Browser+的图形化界面可以非常方便地让用户在本地对OBS进行管理。下载地址为：https://developer.huaweicloud.com/tools#section-1。可以根据自己的系统环境，下载对应版本的安装包，安装即可使用。）



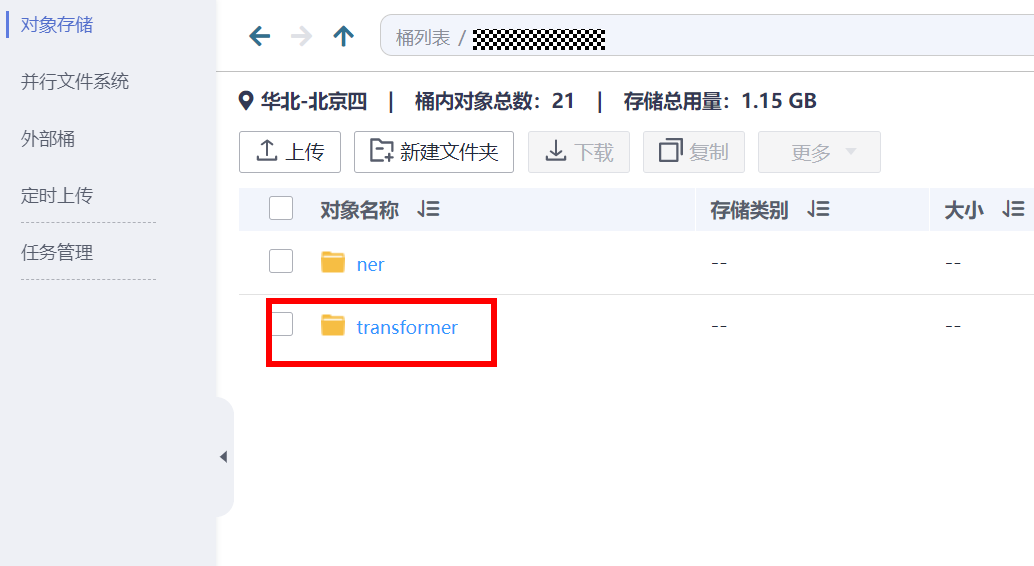
OBS Brower+登录

进入之后创建一个obs桶，用于存放实验所需的文件。在这个桶路径下，我们创建一个 “transformer”的目录用于本实验数据的存放：



创建项目文件夹

创建完如下所示：



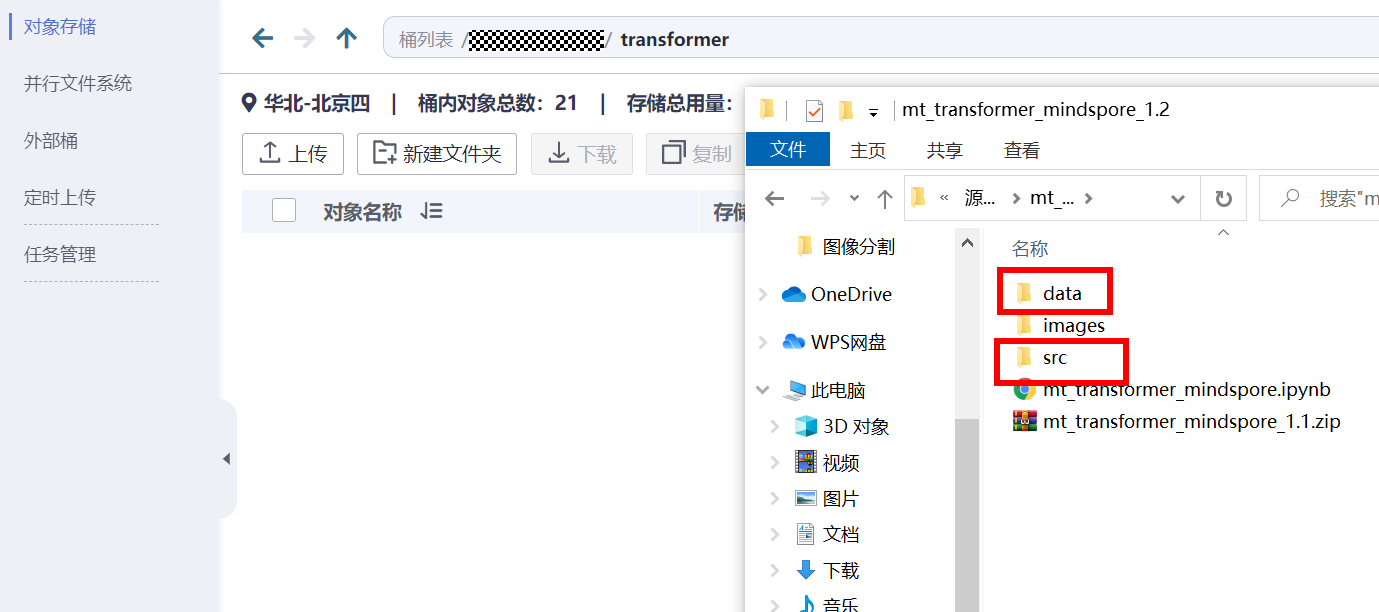
项目文件夹创建成功

下载自然语言处理包

下载链接:https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/NLP/NLP.zip，并在《自然语言处理》中的“实验指导书”-“机器翻译”模块下，获取Seq2Seq的cmn\_zhsim.txt文件。

上传实验源码及数据

进入刚创建的“transformer”文件夹，上传源码及数据至该目录下。直接将需要上传的文件和文件夹拖到该目录下即可。



上传实验数据

进入ModelArts开发环境

参考文末附录，创建ModelArts上的开发环境Notebook并进入。

### 实验过程

上传源码和数据至本地容器

进行实验前，我们需要将obs上的文件下载至容器本地环境中，此处需将obs桶名称换成自己创建的obs桶名称。

输入：

import moxing as mox

mox.file.copy\_parallel(src\_url="obs//你的obs桶/transformer/data/", dst\_url='./data/')

mox.file.copy\_parallel(src\_url="obs://你的obs桶/transformer/src/", dst\_url='./src/')

导入依赖库

输入：

import os

import numpy as np

from easydict import EasyDict as edict

import mindspore.nn as nn

from mindspore import context

import mindspore.dataset.engine as de

import mindspore.common.dtype as mstype

from mindspore.mindrecord import FileWriter

from mindspore.common.parameter import Parameter

import mindspore.dataset.transforms.c\_transforms as deC

from mindspore.common.tensor import Tensor

from mindspore.nn.optim import Adam

from mindspore.train.model import Model

from mindspore.train.loss\_scale\_manager import DynamicLossScaleManager

from mindspore.train.callback import CheckpointConfig, ModelCheckpoint

from mindspore.train.callback import Callback, TimeMonitor

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from src import tokenization

from src.train\_util import LossCallBack

from src.lr\_schedule import create\_dynamic\_lr

from src.transformer\_model import TransformerConfig, TransformerModel

from src.data\_utils import create\_training\_instance, write\_instance\_to\_file

from src.transformer\_for\_train import TransformerTrainOneStepCell, TransformerNetworkWithLoss, TransformerTrainOneStepWithLossScaleCell

import warnings

warnings.filterwarnings('ignore') #屏蔽WARNING信息

设置运行环境

将运行环境设置为昇腾处理器环境：

输入：

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")

定义数据处理相关参数

输入：

data\_cfg = edict({

'input\_file': './data/ch\_en\_all.txt',

'vocab\_file': './data/ch\_en\_vocab.txt',

'train\_file\_mindrecord': './data/train.mindrecord',

'eval\_file\_mindrecord': './data/test.mindrecord',

'train\_file\_source': './data/source\_train.txt',

'eval\_file\_source': './data/source\_test.txt',

'num\_splits':1,

'clip\_to\_max\_len': False,

'max\_seq\_length': 40

})

定义数据处理函数

加载原始数据，切分训练、测试数据，并预处理成模型输入所需的数据形式，并保存为mindrecord格式

输入：

def data\_prepare(cfg, eval\_idx):

tokenizer = tokenization.WhiteSpaceTokenizer(vocab\_file=cfg.vocab\_file)

writer\_train = FileWriter(cfg.train\_file\_mindrecord, cfg.num\_splits)

writer\_eval = FileWriter(cfg.eval\_file\_mindrecord, cfg.num\_splits)

data\_schema = {"source\_sos\_ids": {"type": "int32", "shape": [-1]},

"source\_sos\_mask": {"type": "int32", "shape": [-1]},

"source\_eos\_ids": {"type": "int32", "shape": [-1]},

"source\_eos\_mask": {"type": "int32", "shape": [-1]},

"target\_sos\_ids": {"type": "int32", "shape": [-1]},

"target\_sos\_mask": {"type": "int32", "shape": [-1]},

"target\_eos\_ids": {"type": "int32", "shape": [-1]},

"target\_eos\_mask": {"type": "int32", "shape": [-1]}

}

writer\_train.add\_schema(data\_schema, "tranformer train")

writer\_eval.add\_schema(data\_schema, "tranformer eval")

index = 0

f\_train = open(cfg.train\_file\_source, 'w', encoding='utf-8')

f\_test = open(cfg.eval\_file\_source,'w',encoding='utf-8')

f = open(cfg.input\_file, "r", encoding='utf-8')

for s\_line in f:

print("finish {}/{}".format(index, 23607), end='\r')

line = tokenization.convert\_to\_unicode(s\_line)

source\_line, target\_line = line.strip().split("\t")

source\_tokens = tokenizer.tokenize(source\_line)

target\_tokens = tokenizer.tokenize(target\_line)

if len(source\_tokens) >= (cfg.max\_seq\_length-1) or len(target\_tokens) >= (cfg.max\_seq\_length-1):

if cfg.clip\_to\_max\_len:

source\_tokens = source\_tokens[:cfg.max\_seq\_length-1]

target\_tokens = target\_tokens[:cfg.max\_seq\_length-1]

else:

continue

index = index + 1

# print(source\_tokens)

instance = create\_training\_instance(source\_tokens, target\_tokens, cfg.max\_seq\_length)

if index in eval\_idx:

f\_test.write(s\_line)

features = write\_instance\_to\_file(writer\_eval, instance, tokenizer, cfg.max\_seq\_length)

else:

f\_train.write(s\_line)

features = write\_instance\_to\_file(writer\_train, instance, tokenizer, cfg.max\_seq\_length)

f.close()

f\_test.close()

f\_train.close()

writer\_train.commit()

writer\_eval.commit()

数据处理，选择20%作为验证集

输入：

sample\_num = 23607

eval\_idx = np.random.choice(sample\_num, int(sample\_num\*0.2), replace=False)

data\_prepare(data\_cfg, eval\_idx)

定义数据加载函数

输入：

def load\_dataset(batch\_size=1, data\_file=None):

"""

Load mindrecord dataset

"""

ds = de.MindDataset(data\_file,

columns\_list=["source\_eos\_ids", "source\_eos\_mask",

"target\_sos\_ids", "target\_sos\_mask",

"target\_eos\_ids", "target\_eos\_mask"],

shuffle=False)

type\_cast\_op = deC.TypeCast(mstype.int32)

ds = ds.map(input\_columns="source\_eos\_ids", operations=type\_cast\_op)

ds = ds.map(input\_columns="source\_eos\_mask", operations=type\_cast\_op)

ds = ds.map(input\_columns="target\_sos\_ids", operations=type\_cast\_op)

ds = ds.map(input\_columns="target\_sos\_mask", operations=type\_cast\_op)

ds = ds.map(input\_columns="target\_eos\_ids", operations=type\_cast\_op)

ds = ds.map(input\_columns="target\_eos\_mask", operations=type\_cast\_op)

# apply batch operations

ds = ds.batch(batch\_size, drop\_remainder=True)

ds.channel\_name = 'transformer'

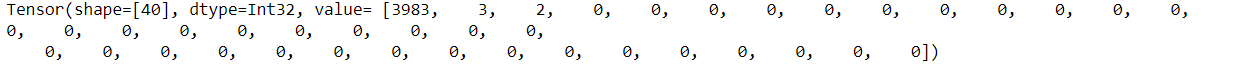
return ds

测试数据是否加载正常

输入：

next(load\_dataset(data\_file=data\_cfg.train\_file\_mindrecord).create\_dict\_iterator())['source\_eos\_ids'][0]

输出：



预处理后的训练数据

定义训练相关配置参数

输入：

train\_cfg = edict({

#--------------------------------------nework confige-------------------------------------

'transformer\_network': 'base',

'init\_loss\_scale\_value': 1024,

'scale\_factor': 2,

'scale\_window': 2000,

'lr\_schedule': edict({

'learning\_rate': 1.0,

'warmup\_steps': 8000,

'start\_decay\_step': 16000,

'min\_lr': 0.0,

}),

#-----------------------------------save model confige-------------------------

'enable\_save\_ckpt': True , #Enable save checkpointdefault is true.

'save\_checkpoint\_steps':590, #Save checkpoint steps, default is 590.

'save\_checkpoint\_num':2, #Save checkpoint numbers, default is 2.

'save\_checkpoint\_path': './checkpoint', #Save checkpoint file path,default is ./checkpoint/

'save\_checkpoint\_name':'transformer-32\_40',

'checkpoint\_path':'', #Checkpoint file path

#-------------------------------device confige-----------------------------

'enable\_data\_sink':False, #Enable data sink, default is False.

'device\_id':0,

'device\_num':1,

'distribute':False,

# -----------------mast same with the dataset-----------------------

'seq\_length':40,

'vocab\_size':10067,

#--------------------------------------------------------------------------

'data\_path':"./data/train.mindrecord", #Data path

'epoch\_size':15,

'batch\_size':32,

'max\_position\_embeddings':40,

'enable\_lossscale': False, #Use lossscale or not, default is False.

'do\_shuffle':True #Enable shuffle for dataset, default is True.

})

'''

two kinds of transformer model version

'''

if train\_cfg.transformer\_network == 'base':

transformer\_net\_cfg = TransformerConfig(

batch\_size=train\_cfg.batch\_size,

seq\_length=train\_cfg.seq\_length,

vocab\_size=train\_cfg.vocab\_size,

hidden\_size=512,

num\_hidden\_layers=6,

num\_attention\_heads=8,

intermediate\_size=2048,

hidden\_act="relu",

hidden\_dropout\_prob=0.2,

attention\_probs\_dropout\_prob=0.2,

max\_position\_embeddings=train\_cfg.max\_position\_embeddings,

initializer\_range=0.02,

label\_smoothing=0.1,

input\_mask\_from\_dataset=True,

dtype=mstype.float32,

compute\_type=mstype.float16)

elif train\_cfg.transformer\_network == 'large':

transformer\_net\_cfg = TransformerConfig(

batch\_size=train\_cfg.batch\_size,

seq\_length=train\_cfg.seq\_length,

vocab\_size=train\_cfg.vocab\_size,

hidden\_size=1024,

num\_hidden\_layers=6,

num\_attention\_heads=16,

intermediate\_size=4096,

hidden\_act="relu",

hidden\_dropout\_prob=0.2,

attention\_probs\_dropout\_prob=0.2,

max\_position\_embeddings=train\_cfg.max\_position\_embeddings,

initializer\_range=0.02,

label\_smoothing=0.1,

input\_mask\_from\_dataset=True,

dtype=mstype.float32,

compute\_type=mstype.float16)

else:

raise Exception("The src/train\_confige of transformer\_network must base or large. Change the str/train\_confige file and try again!")

定义训练过程函数

输入：

def train(cfg):

"""

Transformer training.

"""

train\_dataset = load\_dataset(cfg.batch\_size, data\_file=cfg.data\_path)

netwithloss = TransformerNetworkWithLoss(transformer\_net\_cfg, True)

if cfg.checkpoint\_path:

parameter\_dict = load\_checkpoint(cfg.checkpoint\_path)

load\_param\_into\_net(netwithloss, parameter\_dict)

lr = Tensor(create\_dynamic\_lr(schedule="constant\*rsqrt\_hidden\*linear\_warmup\*rsqrt\_decay",

training\_steps=train\_dataset.get\_dataset\_size()\*cfg.epoch\_size,

learning\_rate=cfg.lr\_schedule.learning\_rate,

warmup\_steps=cfg.lr\_schedule.warmup\_steps,

hidden\_size=transformer\_net\_cfg.hidden\_size,

start\_decay\_step=cfg.lr\_schedule.start\_decay\_step,

min\_lr=cfg.lr\_schedule.min\_lr), mstype.float32)

optimizer = Adam(netwithloss.trainable\_params(), lr)

callbacks = [TimeMonitor(train\_dataset.get\_dataset\_size()), LossCallBack()]

if cfg.enable\_save\_ckpt:

ckpt\_config = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps,

keep\_checkpoint\_max=cfg.save\_checkpoint\_num)

ckpoint\_cb = ModelCheckpoint(prefix=cfg.save\_checkpoint\_name, directory=cfg.save\_checkpoint\_path, config=ckpt\_config)

callbacks.append(ckpoint\_cb)

if cfg.enable\_lossscale:

scale\_manager = DynamicLossScaleManager(init\_loss\_scale=cfg.init\_loss\_scale\_value,

scale\_factor=cfg.scale\_factor,

scale\_window=cfg.scale\_window)

update\_cell = scale\_manager.get\_update\_cell()

netwithgrads = TransformerTrainOneStepWithLossScaleCell(netwithloss, optimizer=optimizer,scale\_update\_cell=update\_cell)

else:

netwithgrads = TransformerTrainOneStepCell(netwithloss, optimizer=optimizer)

netwithgrads.set\_train(True)

model = Model(netwithgrads)

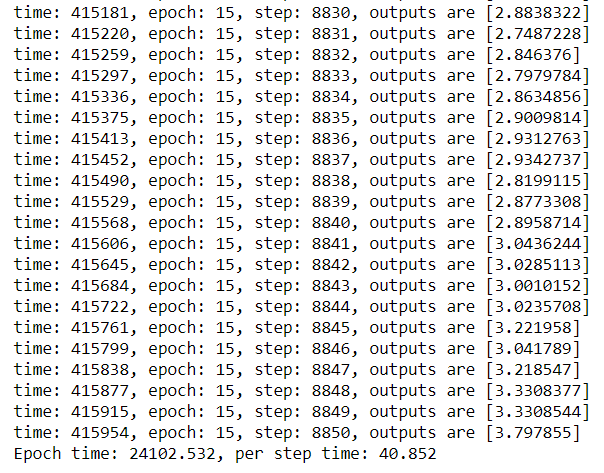
model.train(cfg.epoch\_size, train\_dataset, callbacks=callbacks, dataset\_sink\_mode=cfg.enable\_data\_sink)

启动训练

输入：

train(train\_cfg)

输出：



训练过程输出信息

定义推理相关配置参数

输入：

eval\_cfg = edict({

'transformer\_network': 'base',

'data\_file': './data/test.mindrecord',

'test\_source\_file':'./data/source\_test.txt',

'model\_file': './checkpoint/transformer-32\_40-15\_590.ckpt' ,

'vocab\_file':'./data/ch\_en\_vocab.txt',

'token\_file': './token-32-40.txt',

'pred\_file':'./pred-32-40.txt',

# -------------------mast same with the train config and the datsset------------------------

'seq\_length':40,

'vocab\_size':10067,

#-------------------------------------eval config-----------------------------

'batch\_size':32,

'max\_position\_embeddings':40 # mast same with the train config

})

'''

two kinds of transformer model version

'''

if eval\_cfg.transformer\_network == 'base':

transformer\_net\_cfg = TransformerConfig(

batch\_size=eval\_cfg.batch\_size,

seq\_length=eval\_cfg.seq\_length,

vocab\_size=eval\_cfg.vocab\_size,

hidden\_size=512,

num\_hidden\_layers=6,

num\_attention\_heads=8,

intermediate\_size=2048,

hidden\_act="relu",

hidden\_dropout\_prob=0.0,

attention\_probs\_dropout\_prob=0.0,

max\_position\_embeddings=eval\_cfg.max\_position\_embeddings,

label\_smoothing=0.1,

input\_mask\_from\_dataset=True,

beam\_width=4,

max\_decode\_length=eval\_cfg.seq\_length,

length\_penalty\_weight=1.0,

dtype=mstype.float32,

compute\_type=mstype.float16)

elif eval\_cfg.transformer\_network == 'large':

transformer\_net\_cfg = TransformerConfig(

batch\_size=eval\_cfg.batch\_size,

seq\_length=eval\_cfg.seq\_length,

vocab\_size=eval\_cfg.vocab\_size,

hidden\_size=1024,

num\_hidden\_layers=6,

num\_attention\_heads=16,

intermediate\_size=4096,

hidden\_act="relu",

hidden\_dropout\_prob=0.0,

attention\_probs\_dropout\_prob=0.0,

max\_position\_embeddings=eval\_cfg.max\_position\_embeddings,

label\_smoothing=0.1,

input\_mask\_from\_dataset=True,

beam\_width=4,

max\_decode\_length=80,

length\_penalty\_weight=1.0,

dtype=mstype.float32,

compute\_type=mstype.float16)

else:

raise Exception("The src/eval\_confige of transformer\_network must base or large and same with the train\_confige confige. Change the str/eval\_confige file and try again!")

定义评估测试函数

输入：

class TransformerInferCell(nn.Cell):

"""

Encapsulation class of transformer network infer.

"""

def \_\_init\_\_(self, network):

super(TransformerInferCell, self).\_\_init\_\_(auto\_prefix=False)

self.network = network

def construct(self,

source\_ids,

source\_mask):

predicted\_ids = self.network(source\_ids, source\_mask)

return predicted\_ids

def load\_weights(model\_path):

"""

Load checkpoint as parameter dict, support both npz file and mindspore checkpoint file.

"""

if model\_path.endswith(".npz"):

ms\_ckpt = np.load(model\_path)

is\_npz = True

else:

ms\_ckpt = load\_checkpoint(model\_path)

is\_npz = False

weights = {}

for msname in ms\_ckpt:

infer\_name = msname

if "tfm\_decoder" in msname:

infer\_name = "tfm\_decoder.decoder." + infer\_name

if is\_npz:

weights[infer\_name] = ms\_ckpt[msname]

else:

weights[infer\_name] = ms\_ckpt[msname].data.asnumpy()

weights["tfm\_decoder.decoder.tfm\_embedding\_lookup.embedding\_table"] = \

weights["tfm\_embedding\_lookup.embedding\_table"]

parameter\_dict = {}

for name in weights:

parameter\_dict[name] = Parameter(Tensor(weights[name]), name=name)

return parameter\_dict

def evaluate(cfg):

"""

Transformer evaluation.

"""

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend", reserve\_class\_name\_in\_scope=False)

tfm\_model = TransformerModel(config=transformer\_net\_cfg, is\_training=False, use\_one\_hot\_embeddings=False)

print(cfg.model\_file)

parameter\_dict = load\_weights(cfg.model\_file)

load\_param\_into\_net(tfm\_model, parameter\_dict)

tfm\_infer = TransformerInferCell(tfm\_model)

model = Model(tfm\_infer)

tokenizer = tokenization.WhiteSpaceTokenizer(vocab\_file=cfg.vocab\_file)

dataset = load\_dataset(batch\_size=cfg.batch\_size, data\_file=cfg.data\_file)

predictions = []

source\_sents = []

target\_sents = []

f2 = open(cfg.test\_source\_file, 'r', encoding='utf-8')

for batch in dataset.create\_dict\_iterator():

source\_sents.append(batch["source\_eos\_ids"])

target\_sents.append(batch["target\_eos\_ids"])

source\_ids = Tensor(batch["source\_eos\_ids"], mstype.int32)

source\_mask = Tensor(batch["source\_eos\_mask"], mstype.int32)

predicted\_ids = model.predict(source\_ids, source\_mask)

#predictions.append(predicted\_ids.asnumpy())

# ----------------------------------------decode and write to file(token file)---------------------

batch\_out = predicted\_ids.asnumpy()

for i in range(transformer\_net\_cfg.batch\_size):

if batch\_out.ndim == 3:

batch\_out = batch\_out[:, 0]

token\_ids = [str(x) for x in batch\_out[i].tolist()]

token=" ".join(token\_ids)

#-------------------------------token\_ids to real output file-------------------------------

token\_ids = [int(x) for x in token.strip().split()]

tokens = tokenizer.convert\_ids\_to\_tokens(token\_ids)

sent = " ".join(tokens)

sent = sent.split("<s>")[-1]

sent = sent.split("</s>")[0]

label\_sent = f2.readline().strip()+'\t'

print("source: {}".format(label\_sent))

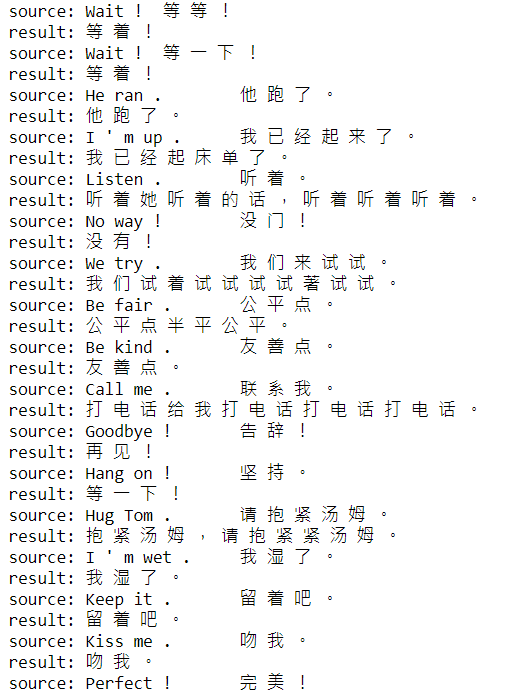
print("result: {}".format(sent.strip()))

启动评估测试

输入：

evaluate(eval\_cfg)

输出：



评估结果演示

## 实验小结

本节实验使用MindSpore实现了基于Transformer的中英文翻译模型，通过实验使学员了解MindSpore的基本用法，同时加深对Transformer的基本原理和网络结构的理解。

# 附录：ModelArts开发环境搭建

* ModelArts平台：Mindspore-1.5

进入ModelArts

在[华为云](https://www.huaweicloud.com/)主页搜索Modelarts，点击“AI开发平台ModelArts”中的“进入控制台”。

图形用户界面, 文本, 应用程序

描述已自动生成

进入ModelArts平台

选择训练作业

选择“北京四”地区，在左侧下拉框中点击“开发环境”中的“Notebook”：

电脑萤幕的截图

描述已自动生成

进入开发环境

创建Notebook

点击创建按钮来创建一个新的Notebook，选择如下配置：

名称：自定义。

**自动停止：1小时**

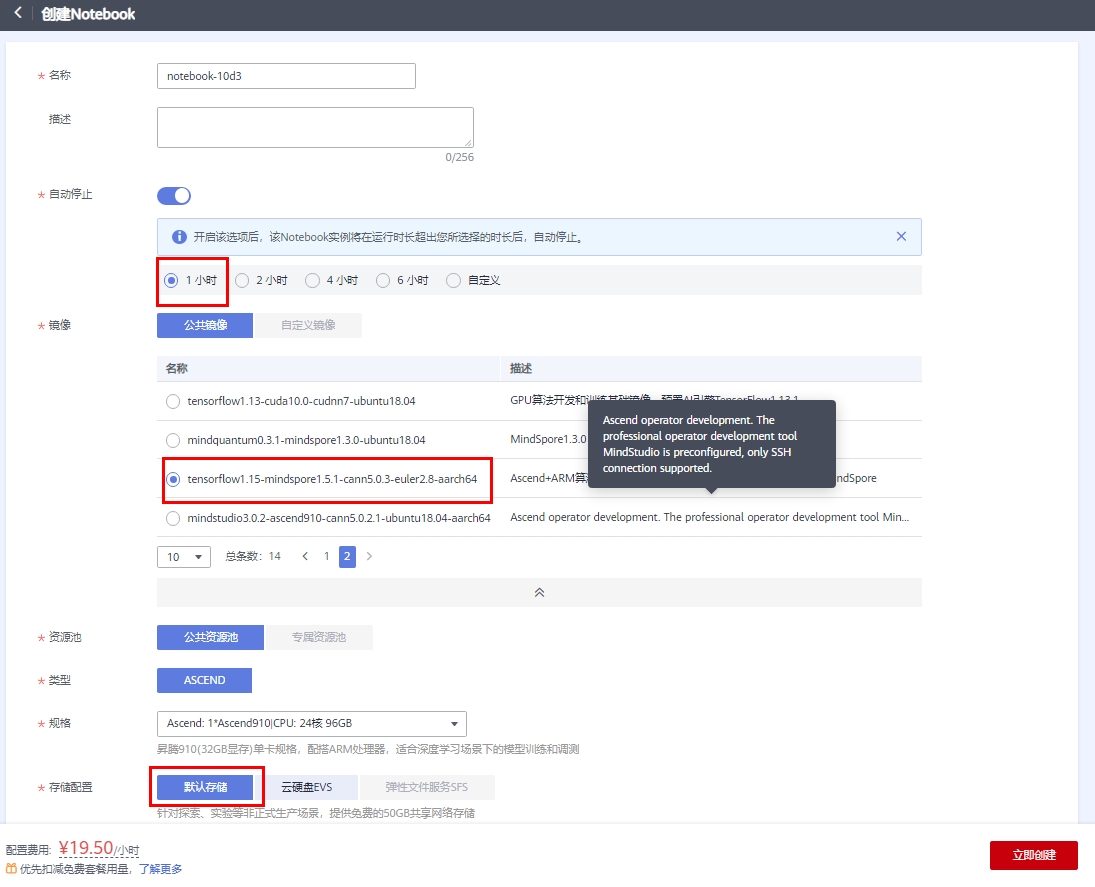
工作环境：选择tensorflow1.15-mindspore1.5.1-cann5.0.2-euler2.8-aarch64

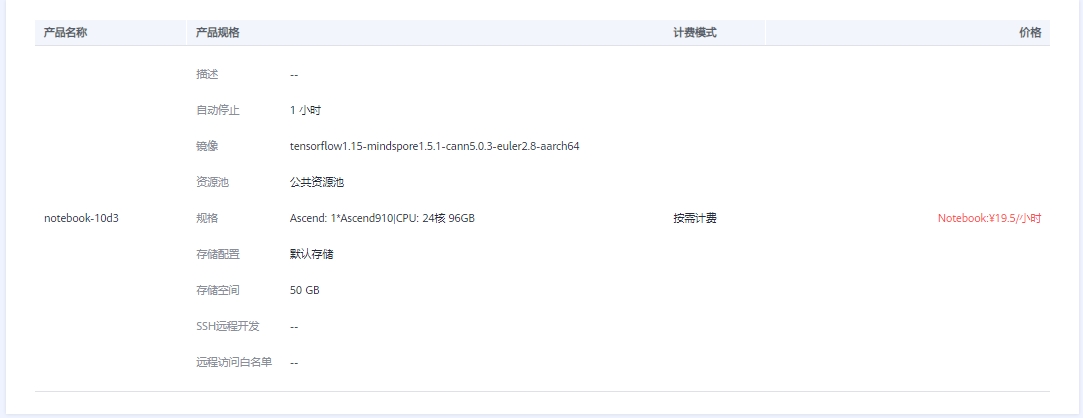
规格：Ascend: 1\*Ascend910|CPU:24核96GB

工作环境：Ascend+ARM算法开发和训练基础镜像。

存储配置：云硬盘EVS。

存储配置：默认存储。



点击“立即创建”，确认规格如下后选择提交：

创建Notebook

启动Notebook

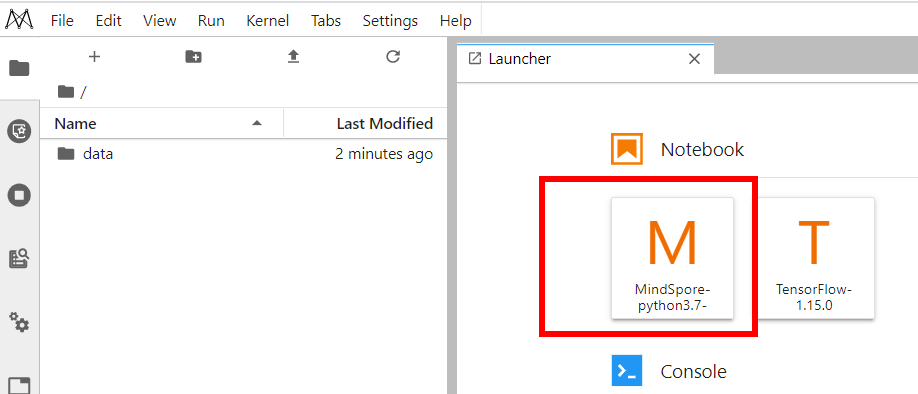
当Notebook状态变为“运行中”时，点击右侧“打开”按钮打开Notebook。



启动Notebook

进入Notebook实验环境

选择右侧“MindSpore-python3.7-aarch64”按钮，进入Notebook实验环境。



进入Notebook实验环境

停止Notebook训练作业

实验完成之后，请及时关闭Notebook训练作业，避免产生不必要的资源浪费。

登录[华为云ModelArts控制台](https://console.huaweicloud.com/modelarts/?region=cn-north-4#/dev-container)，在“操作”栏选择“停止”操作。

如下图所示：



及时停止Notebook

至此训练用的线上Notebook环境搭建完成。