《自然语言处理》

-机器翻译实验

华为技术有限公司

目录

1 实验总览

1.1 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。

1.2 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。

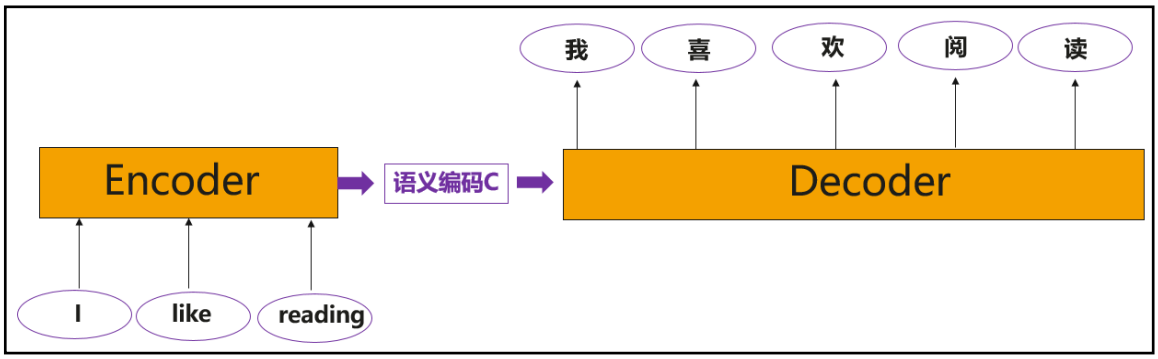
1.3 实验清单

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、MindSpore1.1	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、MindSpore1.1	ModelArts Ascend Notebook

2 Seq2Seq 中英文翻译实验

2.1 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"。

2.2 实验环境

ModelArts Ascend Notebook 环境，环境设置参考《华为云 ModelArts Ascend 环境配置》

2.3 实验步骤

2.3.1 实验准备

步骤 1　OBS 创建项目文件夹

使用 OBS Brower+登录 OBS

图2-1 OBS Brower+登录

进入之前创建的用于挂载 notebook 的 obs 目录，此处我们是建了一个“nlp”的目录用于挂载昇腾环境下的 notebook：

图2-2 进入课程主目录

创建“mt_seq2seq_mindspore”文件夹

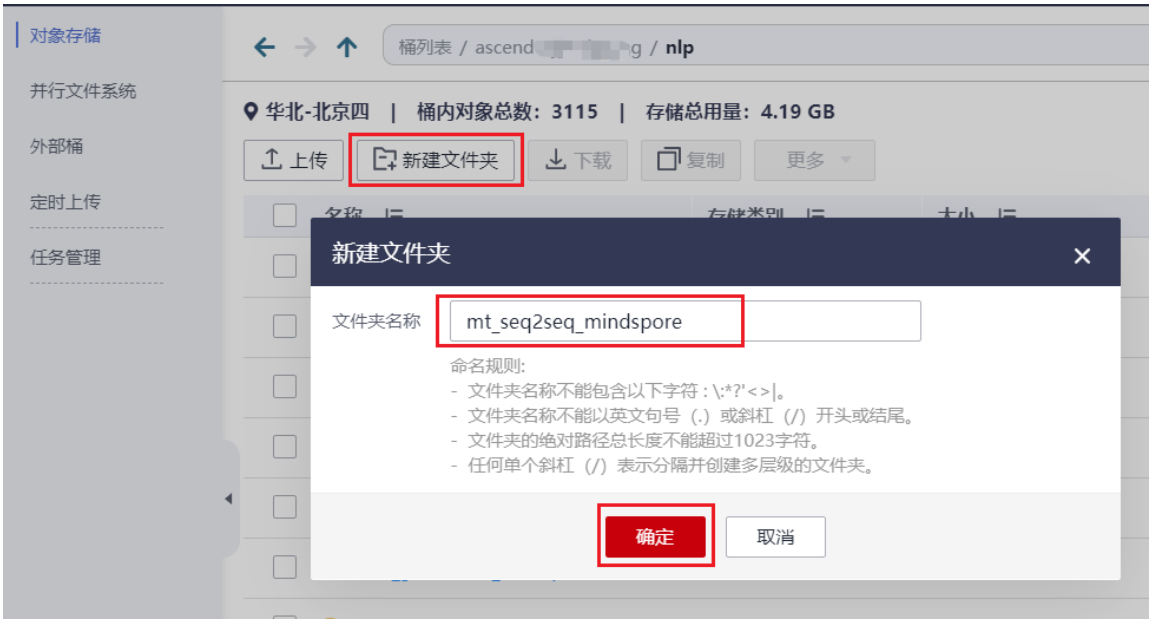


图2-3 创建项目文件夹

创建完如下所示：

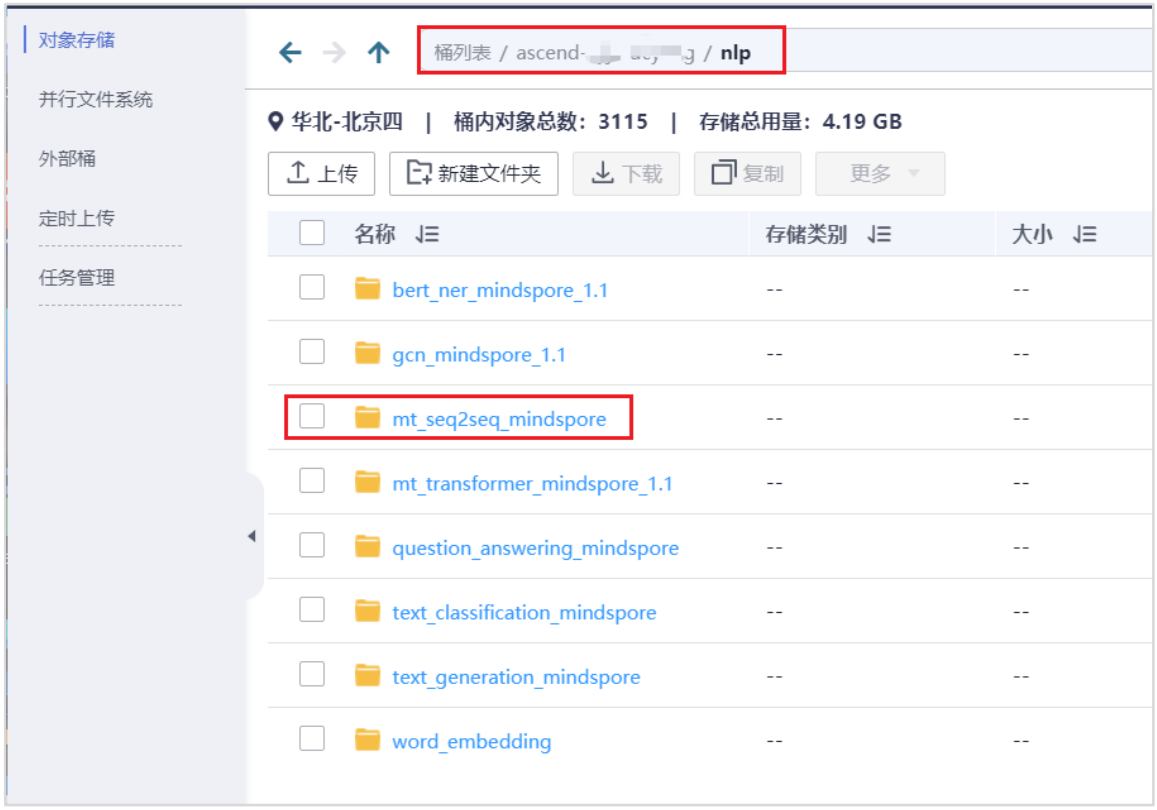


图2-4 项目文件夹创建成功

步骤 2　上传实验数据

进入刚创建的“mt_seq2seq_mindspore”文件夹，上传数据至该目录下。

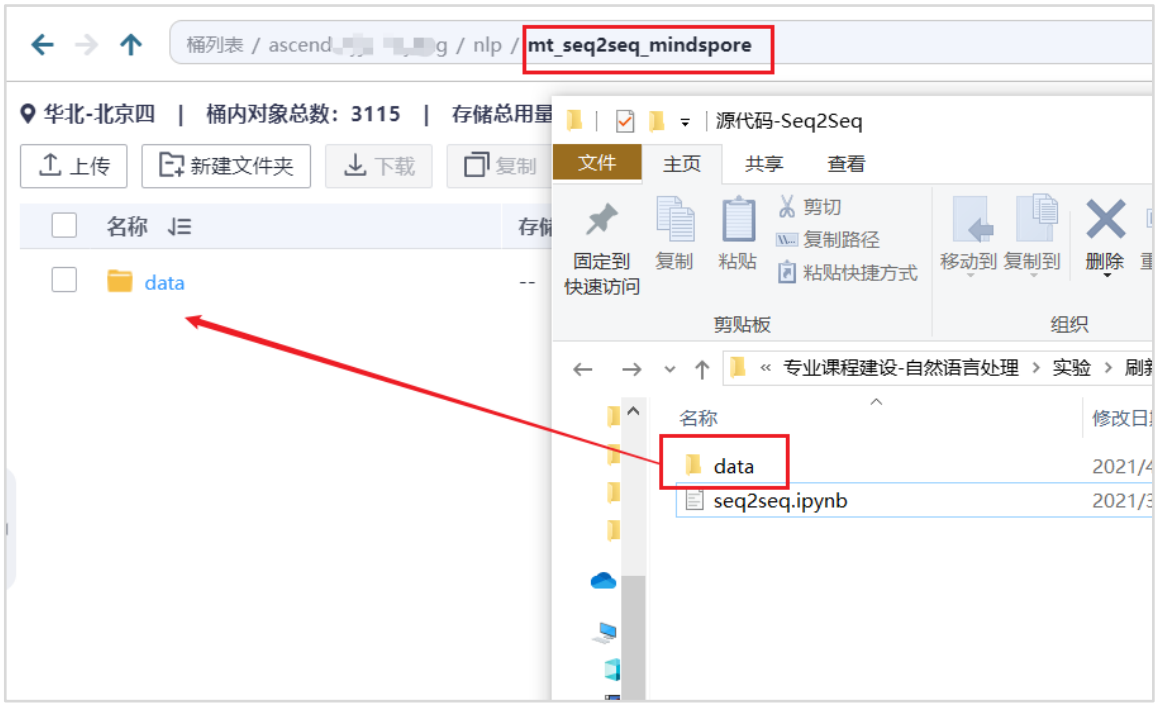


图2-5 上传实验数据

步骤 3　打开 JupyterLab

登录华为云，启动之前创建的 ModelArts Ascend notebook 环境

图2-6 启动 notebook

打开 JupyterLab，并进入之前创建的“mt_seq2seq_mindspore”文件夹

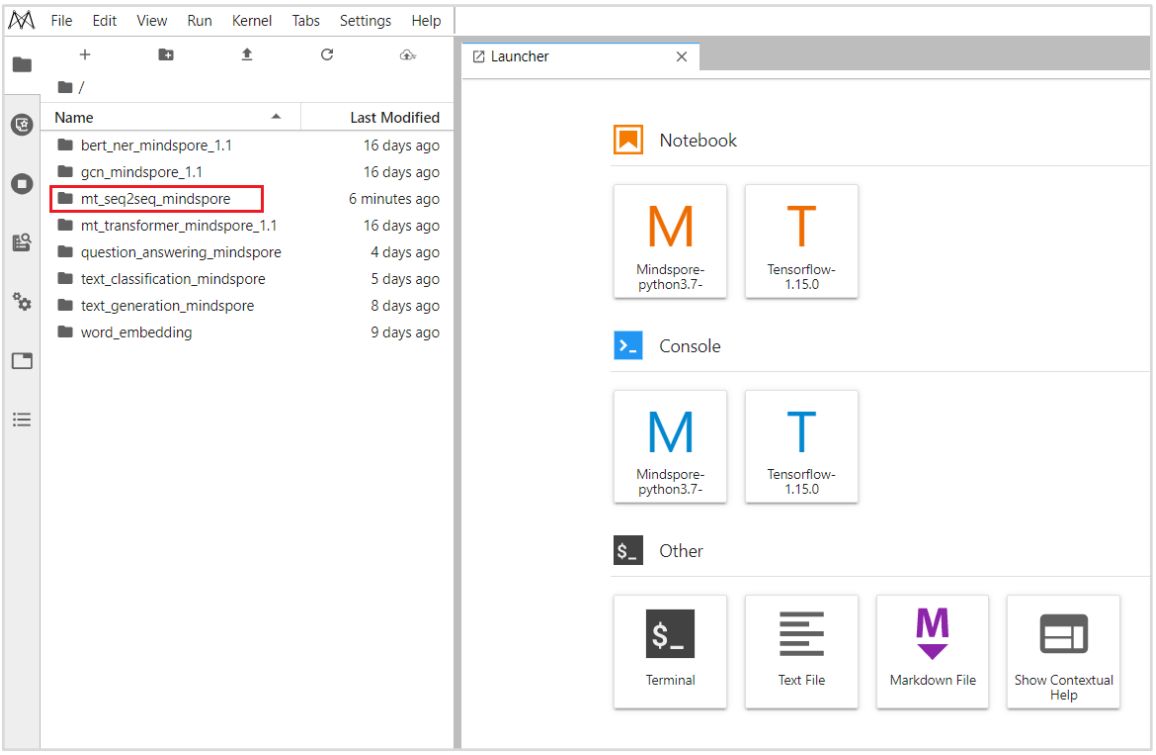


图2-7 进入项目文件夹

新建 notebook 文件：

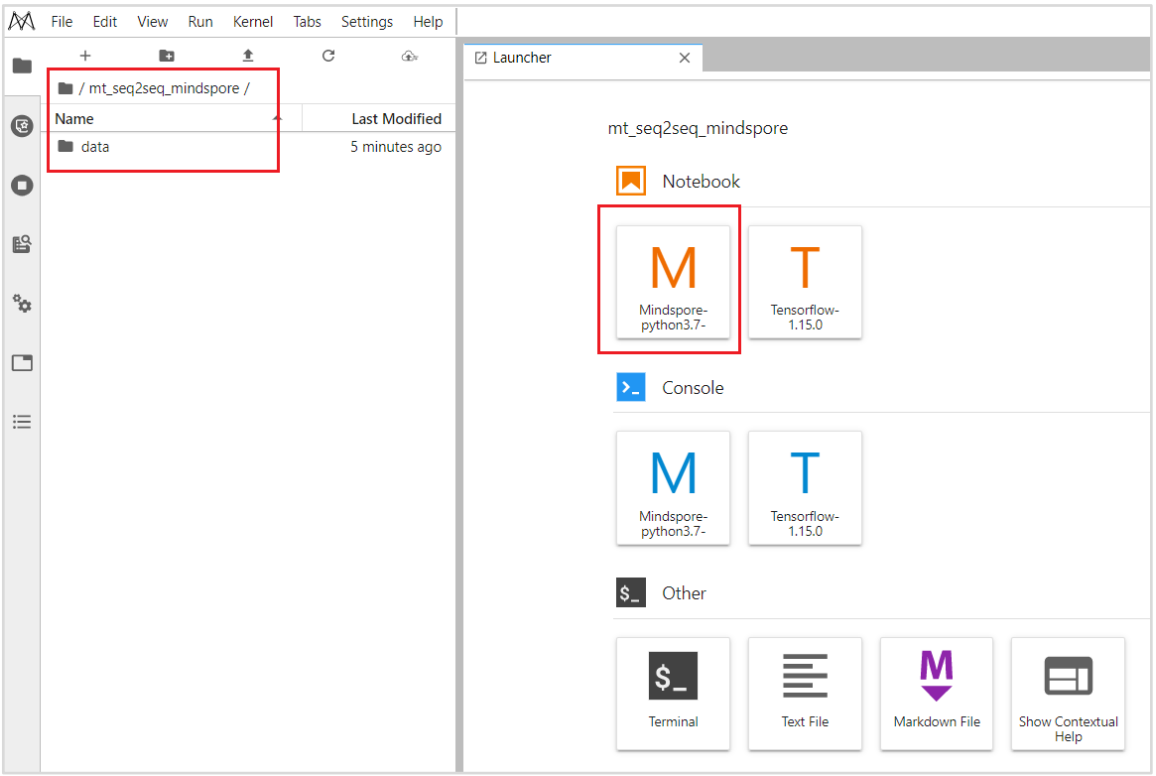


图2-8 新建 notebook 文件

重命名为“seq2seq.ipynb”并打开，右侧显示代码编辑区

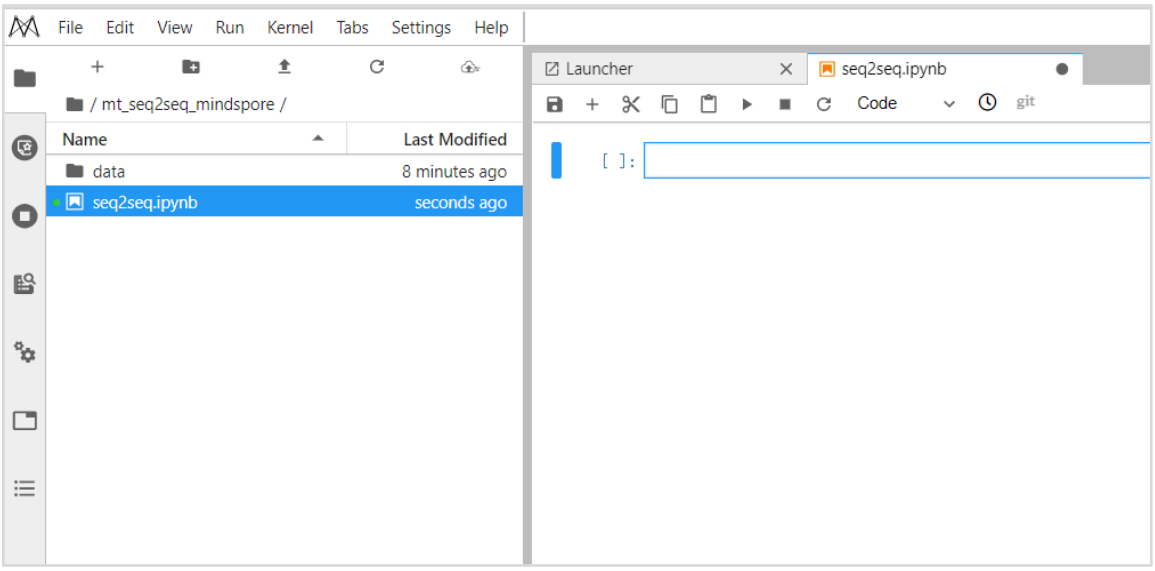


图2-9 重命名并打开 notebook 文件

2.3.2 实验过程

步骤 1　同步数据和源码至本地容器

因为 notebook 是挂载在 obs 上，运行的容器实例不能直接读取操作 obs 上的文件，需下载至容器本地环境中，请在 src_url 字段将 obs 桶名称替换成自己创建的 obs 桶名称。

输入：

```
import moxing as mox
mox.file.copy_parallel(src_url="s3://替换你自己的 obs 路径/nlp/mt_seq2seq_mindspore/data/", dst_url='./data/')
```

步骤 2　定义数据路径

输入：

```
path_to_file = "data/cmn.txt"          ##  数据集文件
```

步骤 3　设置运行环境

输入：

```
context.set_context(mode=context.GRAPH_MODE, save_graphs=False, device_target='Ascend')
```

步骤 4　查看数据内容

输入：

```
#查看训练数据内容前 10 行内容
with open("./data/cmn_zhsim.txt", 'r', encoding='utf-8') as f:
        for i in range(10):
            print(f.readline())
```

输出：

```
Hi.	嗨。
Hi.	你好。
Run.	你用跑的。
Wait!	等等！
Hello!	你好。
I try.	让我来。
I won!	我赢了。
Oh no!	不会吧。
Cheers!	干杯！
He ran.	他跑了。
```

步骤 5　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
```

步骤 6　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("./data/cmn_zhsim.txt", './preprocess', MAX_SEQ_LEN)
```

输出：

```
[1, 624, 1061, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 624, 1061, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 288, 1061, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 819, 893, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 969, 893, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 427, 119, 1061, 0, 0, 0, 0, 0, 0, 0]
[1, 427, 851, 893, 0, 0, 0, 0, 0, 0, 0]
[1, 169, 101, 893, 0, 0, 0, 0, 0, 0, 0]
[1, 378, 893, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 859, 446, 1061, 0, 0, 0, 0, 0, 0, 0]
en_vocab_size:  1154
ch_vocab_size:  1116
(1154, 1116)
```

步骤 7　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'
```

```
})
```

步骤 8　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)
```

步骤 9　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)
    # 思考题 1：这里 3*hidden_size 代表什么含义
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
```

步骤 10　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):
        ##########请将代码补充完整##############
        return output, hidden
```

步骤 11　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))
```

```
            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)
#思考题 2：请根据这里的代码，结合 Attention 机制的理论原理，绘制出模型的结构图
            return output, hidden, attn_weights
```

步骤 12　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
```

步骤 13　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):
        ##########请将代码补充完整##############
        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
```

步骤 14　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)
```

输入：

```
checkpoint_dir = './checkpoint'
checkpoint_prefix = checkpoint_dir +    '/seq2seq_mt_ckpt'
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
fine_tune = True
pretrain_ckpt = "pre_model/pretrain_ckpt"
if fine_tune:
    checkpoint.restore(pretrain_ckpt)
```

步骤 15　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]
```

步骤 16　启动训练

输入：

```
model.train(cfg.num_epochs, ds_train, callbacks=callbacks, dataset_sink_mode=True)
```

步骤 17　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)
```

步骤 18　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 = [[]]
    for i in str_vocab:
        str_id[0] += [en_vocab.index(i)]

    num = max_seq_len + 1 - len(str_id)
    if(num >= 0):
        str_id[0] += [0]*num
    else:
        str_id[0] = str_id[:max_seq_len] + [0]
    str_id = Tensor(np.array(str_id).astype(np.int32))

    out_id = Tensor(np.array([[1,0]]).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)
```

步骤 19 离线加载预测

在线推理测试：

输入：

```
translate('I love Tom ')
```

输出：

```
English ['i', 'love', 'tom']
中文 我爱汤姆。
```

2.4 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。

3 Transformer 中英文翻译实验

3.1 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。

3.2 实验环境

ModelArts Ascend Notebook 环境，MindSpore1.1.1

3.3 预备知识

Transformer 网络如下图所示，其中左边为编码网路，右边 为解码网络。

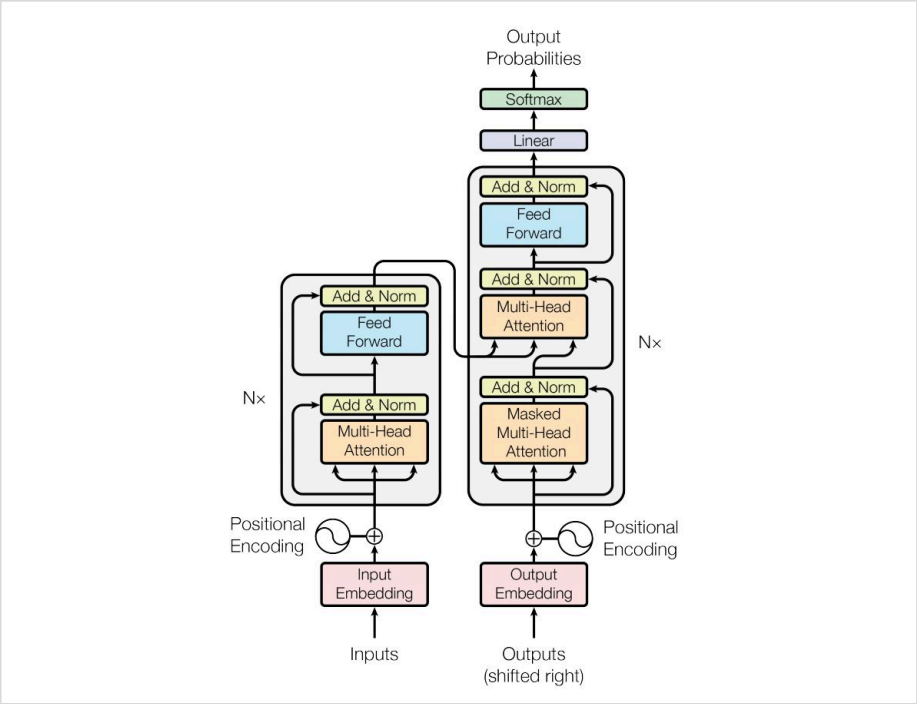


图3-1 Transformer 架构

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）。

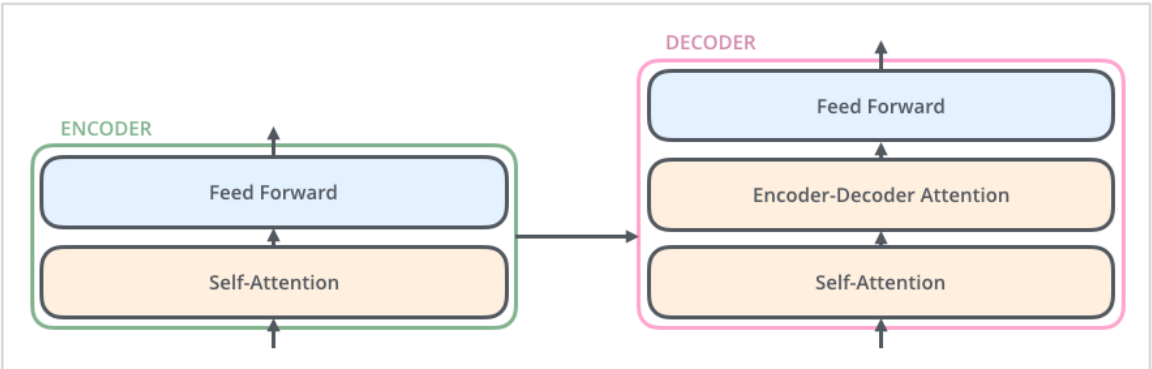


图3-2 Encoder-Decoder Attention

3.4 实验步骤

3.4.1 实验准备

步骤 1　上传实验数据及源码

参考上一节实验，OBS 中创建“nlp/mt_transformer_mindspore_1.1”文件夹，并上传实验数据和代码。

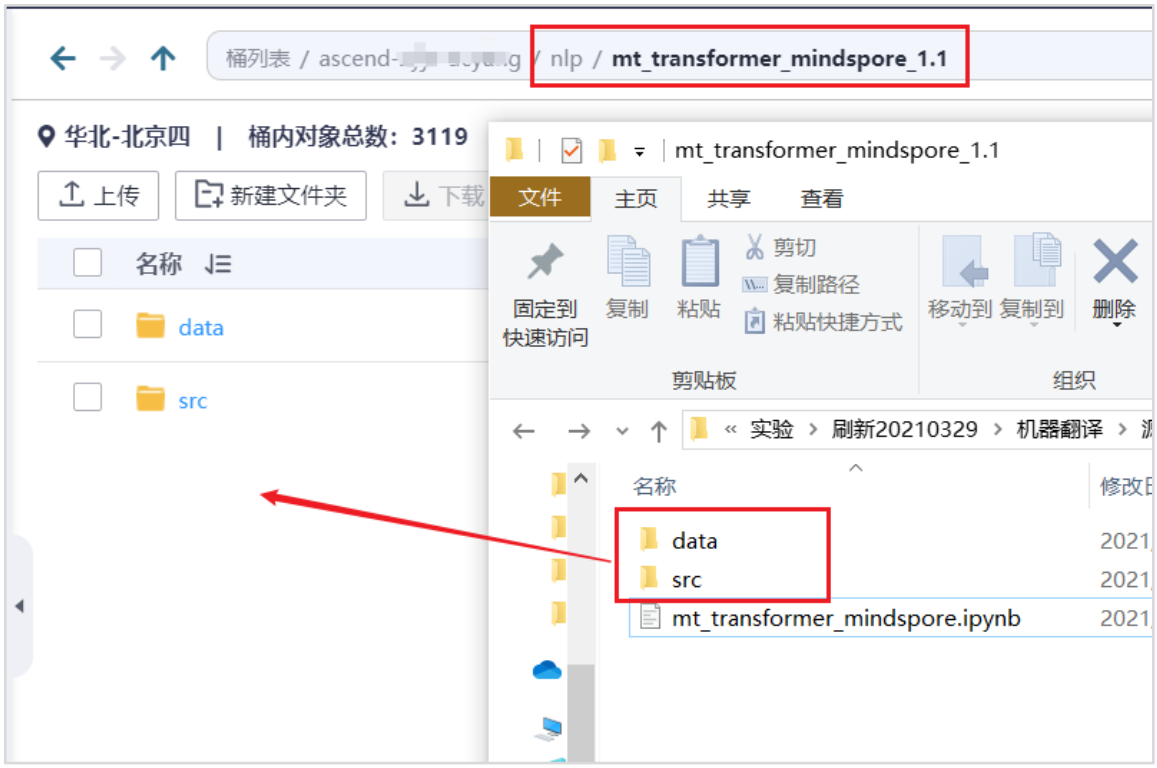


图3-3 上传源码及数据

步骤 2　启动 JupyterLab，进入对应项目文件夹，新建 notebook 文件

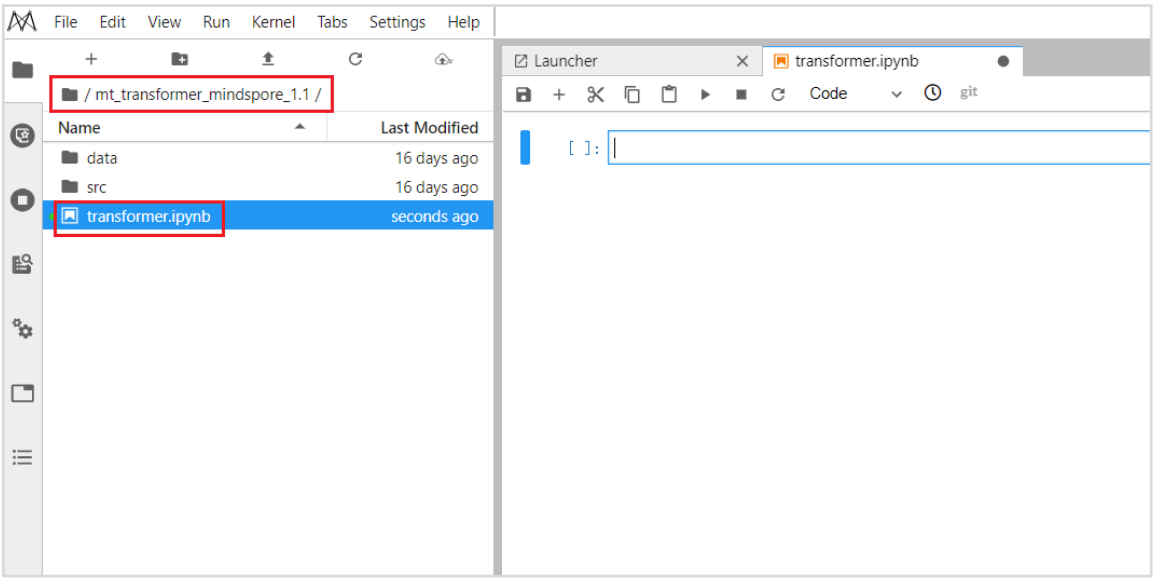


图3-4 新建 notebook 文件

3.4.2 实验过程

步骤 1　下载源码和数据至本地容器

因为 notebook 是挂载在 obs 上，运行的容器实例不能直接读取操作 obs 上的文件，需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="s3://你的 obs 桶/nlp/mt_transformer_mindspore_1.1/data/", dst_url='./data/')
mox.file.copy_parallel(src_url="s3://你的 obs 桶/nlp/mt_transformer_mindspore_1.1/src/", dst_url='./src/')
```

步骤 2　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
```

步骤 3　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")
```

步骤 4　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
})
```

步骤 5　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()
```

步骤 6　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)
```

步骤 7　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_id
s'][0]
```

输出：

```
Tensor(shape=[40], dtype=Int32, value= [3983,    3,    2,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0])
```

步骤 8　定义训练相关配置参数

输入：

```
train_cfg = edict({
    #------------------------------------nework config-----------------------------------
    '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 config-------------------------
    '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 config----------------------------
    '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!")
```

步骤 9　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)
```

步骤 10　启动训练

输入：

```
train(train_cfg)
```

输出：

```
time: 415181, epoch: 15, step: 8830, outputs are [2.8838322]
time: 415220, epoch: 15, step: 8831, outputs are [2.7487228]
time: 415259, epoch: 15, step: 8832, outputs are [2.846376]
time: 415297, epoch: 15, step: 8833, outputs are [2.7979784]
time: 415336, epoch: 15, step: 8834, outputs are [2.8634856]
time: 415375, epoch: 15, step: 8835, outputs are [2.9009814]
time: 415413, epoch: 15, step: 8836, outputs are [2.9312763]
time: 415452, epoch: 15, step: 8837, outputs are [2.9342737]
time: 415490, epoch: 15, step: 8838, outputs are [2.8199115]
time: 415529, epoch: 15, step: 8839, outputs are [2.8773308]
time: 415568, epoch: 15, step: 8840, outputs are [2.8958714]
time: 415606, epoch: 15, step: 8841, outputs are [3.0436244]
time: 415645, epoch: 15, step: 8842, outputs are [3.0285113]
time: 415684, epoch: 15, step: 8843, outputs are [3.0010152]
time: 415722, epoch: 15, step: 8844, outputs are [3.0235708]
time: 415761, epoch: 15, step: 8845, outputs are [3.221958]
time: 415799, epoch: 15, step: 8846, outputs are [3.041789]
time: 415838, epoch: 15, step: 8847, outputs are [3.218547]
time: 415877, epoch: 15, step: 8848, outputs are [3.3308377]
time: 415915, epoch: 15, step: 8849, outputs are [3.3308544]
time: 415954, epoch: 15, step: 8850, outputs are [3.797855]
Epoch time: 24102.532, per step time: 40.852
```

步骤 11　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!")
```

步骤 12　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()))
```

步骤 13　启动评估测试

输入：

```
evaluate(eval_cfg)
```

输出：

```
source: Wait !  等 等 !
result: 等 着 !
source: Wait !  等 一 下 !
result: 等 着 !
source: He ran .        他 跑 了 。
result: 他 跑 了 。
source: I ' m up .      我 已 经 起 来 了 。
result: 我 已 经 起 床 单 了 。
source: Listen .        听 着 。
result: 听 着 她 听 着 的 话 ， 听 着 听 着 听 着 。
source: No way !        没 门 !
result: 没 有 !
source: We try .        我 们 来 试 试 。
result: 我 们 试 着 试 试 试 试 著 试 试 。
source: Be fair .       公 平 点 。
result: 公 平 点 半 平 公 平 。
source: Be kind .       友 善 点 。
result: 友 善 点 。
source: Call me .       联 系 我 。
result: 打 电 话 给 我 打 电 话 打 电 话 打 电 话 。
source: Goodbye !       告 辞 !
result: 再 见 !
source: Hang on !       坚 持 。
result: 等 一 下 !
source: Hug Tom .       请 抱 紧 汤 姆 。
result: 抱 紧 汤 姆 ， 请 抱 紧 紧 汤 姆 。
source: I ' m wet .     我 湿 了 。
result: 我 湿 了 。
source: Keep it .       留 着 吧 。
result: 留 着 吧 。
source: Kiss me .       吻 我 。
result: 吻 我 。
source: Perfect !       完 美 !
```

3.5 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。