计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目： 华为云 Seq2Seq 中英文翻译实验		学号：202020130190
日期：2022.12.15	班级： 人工智能	姓名：刘绪波
Email：2842353032@qq.com		

实验目的：

在本作业中，学习 Seq2Seq 中英文翻译实验的实验原理并进行编码测试；本实验将在华为云 ModelAtrs 平台，使用华为 MindSpore 深度学习框架，实现带有 Att ention 机制的 Seq2Seq（GRU）机器翻译模型。

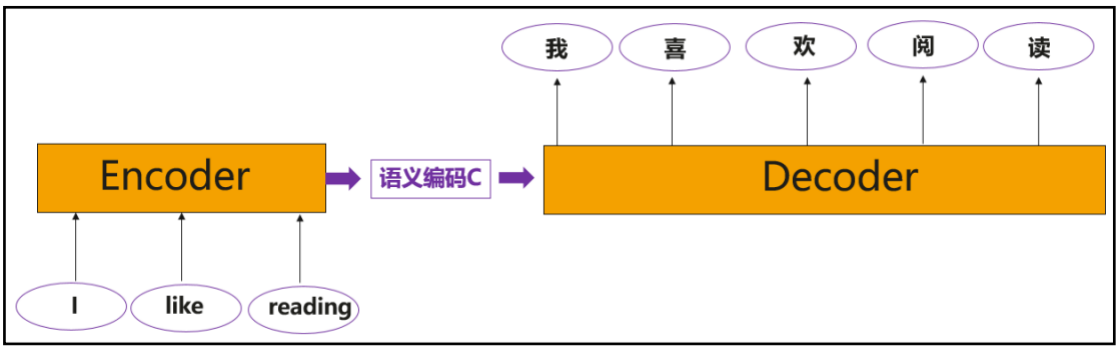
实验软件和硬件环境：

软件: 华为云 Model Arts

实验原理和方法：

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-D ecoder，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. 实验材料：

在提供的压缩包中，包含:

cmn_zhsim.txt ---------- 完整的双语数据集

cmn_zhsim_mini.txt ---------- 较小规模的双语数据集，用来快速训练

seq2seq-main.ipynb ---------- 你需要运行并完善的代码

华为云使用手册-2022.pdf ---------- ModelArts 及 OBS 使用手册

实验步骤：（不要求罗列完整源代码）

1. 环境准备：

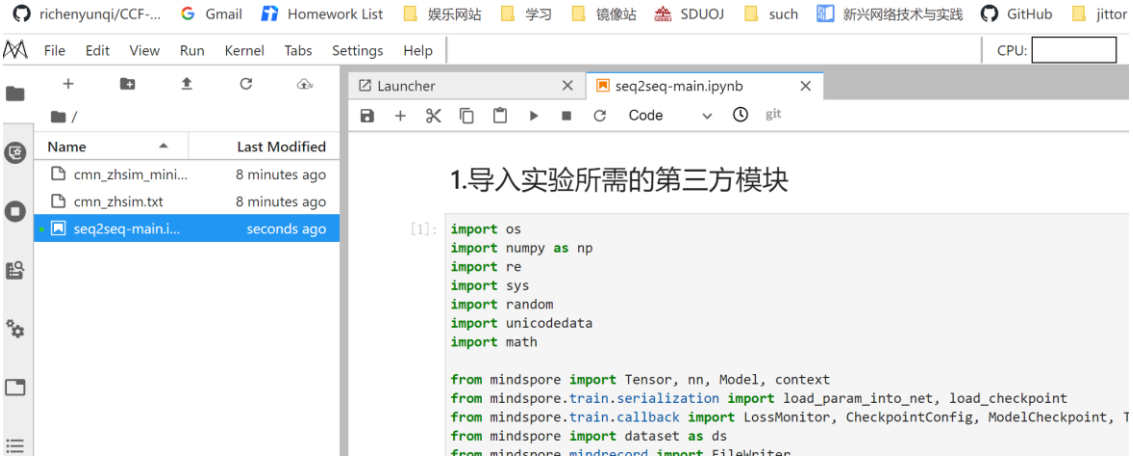
1. 登陆华为云，进入 ModelArts 开发环境。选择开发环境-Notebook，并进入旧版 Notebook 界面。

2. 点击创建，新建 Notebook。本次实验将使用 MindSpore 框架，硬件平台是华为 Ascend 910AI 处理器，如下图所示，需选择对应的工作环境。

3. 选择对象存储服务，并配置华为 OBS 桶。

4. 提交创建，创建成功后，Notebook 将处于“运行中”状态，点击打开 JupyterLab，进入 Notebook 界面。
5. 将本实验提供的数据集文件和代码文件上传到 OBS 桶中。

6. 文件上传成功后可在 Notebook 界面左侧的文件列表中显示。依次选中两个数据集文件， 并将其同步到工作环境（OBS 空间与工作环境并不互通，因此需要这一步将存放在 OBS 桶中的数据集上传到代码的运行环境中。）



2. 具体实验步骤以及结果展示：

① 打开 seq2seq-main.ipynb，阅读代码，根据提示补全代码，训练并测试模型。调整参数， 在小规模数据集 cmn_zhsim_mini.txt 测试成功后，继续调整参数，在 cmn_zhsim.txt 数 据集进行全量训练。

对于 cmn_zhsim.txt 数据集和 cmn_zhsim_mini.txt 数据集参数设置：

可以通过获取 mindrecord 文件步骤自行调整实现；

首先是 cmn_zhsim_mini.txt 文件，其参数设置：

然后根据此参数对下面的 cfg 超参数进行配置：

```
from easydict import EasyDict as edict
# hidden_size默认为1024, num_epochs默认为15
#注意！！！'en_vocab_size'与'ch_vocab_size'两项，使用cmn_zhsim.txt和cmn_zhsim_mini.txt会有不同的设置，具体值请查看步骤5

# 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,
```

代码修改：

```
def construct(self, decoder_input, hidden, encoder_output):
    embeddings = self.embedding(decoder_input)
    embeddings = self.dropout(embeddings)
    '''
    代码补充
    调用self.embedding，对输入decoder_input进行Embedding编码。
    并对其进行dropout操作

    将处理得到的输出命名为embeddings
    '''
    # calculate attn
    attn_weights = self.softmax(self.attn(embeddings)) # [1,1,10]

    '''
    代码补充
    调用self.attn，对embeddings计算注意力权重。
    并使用softmax处理注意力权重

    将处理得到的输出命名为attn_weights
    '''
    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)

    '''
    代码补充
    调用self.out，处理上步得到的output，将特征维度映射到词表大小。
    并使用self.logsoftmax处理输出

    将处理得到的输出继续命名为output
    '''
    output = self.out(output)
    output = self.logsoftmax(output)
    return output, hidden, attn_weights
```

模型训练：

14. 定义回调函数，构建模型，启动训练

```
loss_cb = LossMonitor()
config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps, keep_checkpoint_max=cfg.keep_checkp
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]
model.train(cfg.num_epochs, ds_train, callbacks=callbacks, dataset_sink_mode=True)
```

```
epoch: 1 step: 125, loss is 2.6842563
epoch time: 76045.874 ms, per step time: 608.367 ms
epoch: 2 step: 125, loss is 1.8440057
epoch time: 6564.371 ms, per step time: 52.515 ms
epoch: 3 step: 125, loss is 1.9872608
epoch time: 6547.056 ms, per step time: 52.376 ms
epoch: 4 step: 125, loss is 1.8343636
epoch time: 6548.226 ms, per step time: 52.386 ms
epoch: 5 step: 125, loss is 1.265866
epoch time: 6545.574 ms, per step time: 52.365 ms
epoch: 6 step: 125, loss is 0.91582394
epoch time: 6550.328 ms, per step time: 52.403 ms
epoch: 7 step: 125, loss is 0.8997347
```

查看 and 修改 chackpoint:

```
cd ckpt
```

```
/home/ma-user/work/ckpt
```

```
!ls
```

```
gru-10_125.ckpt  gru-13_125.ckpt  gru-6_125.ckpt  gru-9_125.ckpt
gru-11_125.ckpt  gru-14_125.ckpt  gru-7_125.ckpt  gru-graph.meta
gru-12_125.ckpt  gru-15_125.ckpt  gru-8_125.ckpt
```

```
network = Seq2Seq(cfg,is_train=False)
network = InferCell(network, cfg)
network.set_train(False)
# 注意，不同的checkpoint请自行设置
parameter_dict = load_checkpoint("./ckpt/gru-15_125.ckpt")
load_param_into_net(network, parameter_dict)
model = Model(network)
```

查看结果：使用 mini 训练效果不佳

18.在线推理测试

[24]:
```
translate('i love tom')
```

```
English ['i', 'love', 'tom']
中文 搬失同盒抵受
```

[25]:
```
translate('i hate tom')
```

```
English ['i', 'hate', 'tom']
中文  认跑受
```

[26]:
```
translate('Hi')
```

```
English ['hi']
中文 别报受
```

使用完整训练：具体修改过程不再赘述，直接看模型和结果：

[ ]:
```
!ls
```

```
gru-10_125.ckpt   gru_1-14_125.ckpt  gru_1-6_125.ckpt  gru-7_125.ckpt
gru_1-10_125.ckpt gru_1-15_125.ckpt  gru_1-7_125.ckpt  gru-8_125.ckpt
gru_1-11_125.ckpt gru-12_125.ckpt    gru_1-8_125.ckpt  gru-9_125.ckpt
gru-11_125.ckpt   gru-13_125.ckpt    gru_1-9_125.ckpt  gru-graph.meta
gru_1-12_125.ckpt gru-14_125.ckpt    gru_1-graph.meta
gru_1-13_125.ckpt gru-15_125.ckpt    gru-6_125.ckpt
```

18.在线推理测试

[30]:
```
translate('i love tom')
```

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

[31]:
```
translate('i hate tom')
```

```
English ['i', 'hate', 'tom']
中文 我恨汤姆。
```

[32]:
```
translate('Hi')
```

```
English ['hi']
中文 祝你好运。
```

可见使用完整的训练集训练效果很好！

② 阅读 MindSpore Api 文档，了解使用 P.DynamicGRUV2() api 定义的 GRU 模块输入输出 组成与计算原理，写入实验报告。

输入： x（Tensor）-当前单词。（num_step，batch_size，input_size）

weight_input（Tensor）-输入隐藏权重。数据类型必须为 float16。（input_size，3×hidden_size）

weight_hidden（Tensor）-隐藏权重。形状的张量。数据类型必须为 float16。（hidden_size，3×hidden_ssize）

init_h（Tensor）-初始时间的隐藏状态。形状的张量。数据类型必须为 float16 或 float32。（batch_size，hidden_size）

bias_input（Tensor）-输入隐藏偏置。形状张量或无。具有与输入 init_h 相同的数据类型。（3×hidden_size）

bias_hidden（Tensor）-隐藏偏置。形状张量或无。具有与输入 init_h 相同的数据类型。（3×hidden_size） seq_length（Tensor）-每个批次的长度。形状为的张量。当前仅支持 None。（batch_size）

输出： y（Tensor）-形状的张量： y_shape=：如果 num_proj>0，（num_step，batch_size，min（hidden_size，num_proj）） y_shape=：如果 num_proj=0。（num_step、batch_size、hidden_size）

output.h（Tensor）-（num_step，batch_size，hidden_size）

update（Tensor）-Tensor。（num_step，batch_size，hidden_size）

reset（Tensor）- Tensor。（num_step，batch_size，hidden_size）

new（Tensor）- Tensor。（num_step，batch_size，hidden_size）

hidden_new（Tensor）-（num_step，batch_size，hidden_size）

计算原理：

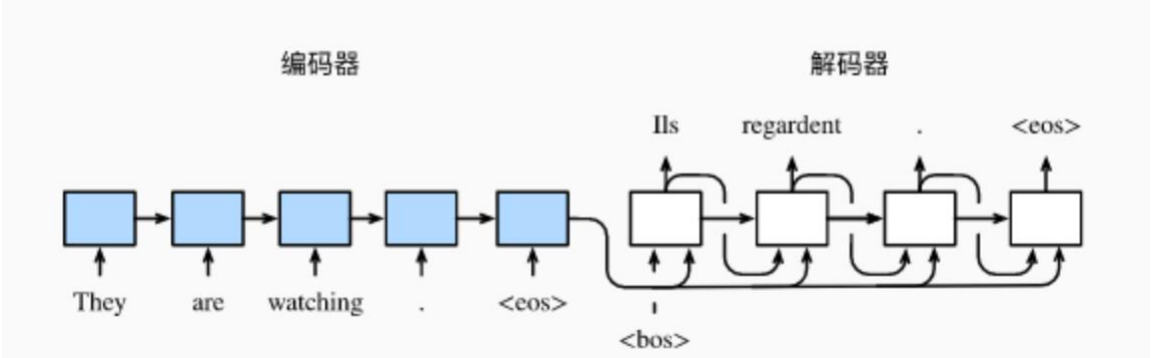
即用门控机制控制输入、记忆等信息而在当前时间步做出预测，表达式由以下给出：

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$
$$r = \sigma(x_t U^r + s_{t-1} W^r)$$
$$h = tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

GRU 有两个有两个门，即一个重置门（reset gate）和一个更新门（update gate）。从直观上来说，重置门决定了如何将新的输入信息与前面的记忆相结合，更新门定义了前面记忆保存到当前时间步的量。如果我们将重置门设置为 1，更新门设置为 0，那么我们将再次获得标准 RNN 模型。

③ 阅读代码，理解单独定义的 Encoder 和 Decoder 是怎么组成完整的 Seq2Seq 模型的，梳 理 Encoder 的哪些内容传递给了后面的 Decoder 结构，写入实验报告。

seq2seq 模型由编码器(Encoder)和解码器(Decoder)两部分组成，模型的输入和输出都是一个不定长文本序列。Encoder 将原始文本序列编码，在每个时间步中输出隐藏状态，在最终时间步隐藏状态可转换为固定长度的向量，这就是上下文向量。Decoder 会将向量再次转换为目标文本序列，从而实现文本翻译的目的。Encoder 和 Decoder 一般都为 RNN 网络。



④ 模型使用了 NLLLoss，梳理该损失函数的输入输出是什么，是什么样的数据类型与数据 维度，写入实验报告。

输入：

- **input**（张量） - 输入对数，带有形状。数据类型仅支持 float32 或 float16。(N,C)（北，丙）
- **target**（张量） - 具有形状的地面实况标签。数据类型仅支持 int32。(N)(N)
- **weight**（Tensor） - 重新缩放每个类的权重，仅具有形状和数据类型 支持浮点 32 或浮点 16'。(C)

输出：

由损失和 total_weight 组成的 2 个张量的元组。

- **loss**（Tensor） - 当归约为无且输入为 2D 张量时，损失形状为 （N,）。 否则，损失为标量。数据类型与输入的数据类型相同。
- **total_weight**（张量） -total_weight 是一个标量。数据类型与权重相同。

结论分析与体会：

1. MindSpore 与 pytorch 区别：

MindSpore 动态图模式支持数据并行，通过对数据按 batch 维度进行切分，将数据分配到各个计算单元中进行模型训练，从而**缩短训练时间**。

维度变化的区别：PyTorch 的 reshape，flatten，transpose，permute 可用 MindSpore 的 reshape，transpose 代替。MindSpore 的 transpose 要指定全部维度的顺序。

nn.conv2d：PyTorch：默认不对输入进行填充，bias 为 True。

MindSpore：默认对输入进行填充，使输出与输入维度一致，如果不需要 padding，可以将 pad_mode 参数设为“valid”；如果需要填充，将 pad_mode 参数设为 “pad”。默认 has_bias 为 False。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1－3 道问答题：

1. 调用 convert_to_mindrecord 方法报错：

由于工作空间中已经存在 preprocess 文件夹及 mindrecord 文件，第二次运行 convert_to_mindrecord 方法时会报错。

需要执行 linux 命令：!rm -rf preprocess/ 删除 preprocess 文件夹。

2. 实验中的步骤七八换版本问题：

更换了版本会报错，在没有换版本之前可以正常运行；

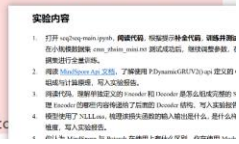
```
    10 from mindspore.train.serialization import load_param_into_net, load_checkpoint
    11 from mindspore.train.callback import LossMonitor, CheckpointConfig, ModelCheckpoint, TimeMonitor

~/miniconda3/envs/MindSpore-python3.7-aarch64/lib/python3.7/site-packages/mindspore/__init__.py in <module>
    16
    17 from .run_check import run_check
---> 18 from . import common, dataset, mindrecord, train, log
    19 from . import profiler, communication, numpy, parallel
    20 from .common import *

~/miniconda3/envs/MindSpore-python3.7-aarch64/lib/python3.7/site-packages/mindspore/common/__init__.py in <module>
    15 """Top-level reference to dtype of common module."""
    16 from __future__ import absolute_import
---> 17 from . import dtype
    18 from .api import ms_function, ms_memory_recycle, ms_class, _convert_python_data
    19 from .dtype import Type, int8, byte, int16, short, int32, intc, int64, intp, \

~/miniconda3/envs/MindSpore-python3.7-aarch64/lib/python3.7/site-packages/mindspore/common/dtype.py in <module>
    21 import numpy as np
    22 from mindspore import log as logger
---> 23 from .._c_expression import typing
    24 from .._c_expression.typing import Type
    25

ImportError: libacl_tdt_channel.so: cannot open shared object file: No such file or directc
```



3. 在查看 chackpoint 时运行 ls 命令，多次运行时，总会报错：

```
~/miniconda3/envs/MindSpore-python3.7-aarch64/lib/python3.7/site-packages/pexpect/pty_spawn.py in _wrap_ptyprocess_e
rr()
     23         yield
     24     except ptyprocess.PtyProcessError as e:
---> 25         raise ExceptionPexpect(*e.args)
     26
     27 PY3 = (sys.version_info[0] >= 3)

ExceptionPexpect: isalive() encountered condition where "terminated" is 0, but there was no child process. Did someo
ne else call waitpid() on our process?
```

尚未解决，不知道为什么!!!!!!!!!!