# 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:Regularization and Batch Normalization | | 学号：202000130047 |
|---|---|---|
| 日期：2022-12-4 | 班级： 智能 20 | 姓名： 夏再禹 |
| Email：842649082@qq.com | | |

实验目的：

在华为云 ModelAtrs 平台，使用华为 MindSpore 深度学习框架，实现带有 Attention 机制的 Seq2Seq（GRU）机器翻译模型

实验软件和硬件环境：

软件：jupyter notebook

硬件：cpu: Intel i5

实验原理和方法：

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”。

实验步骤：（不要求罗列完整源代码）

## 一、实验环境准备

按照实验指导书，建立 ModelArts 开发环境，其配置为 Ascend910AI。


然后将本实验提供的数据集文件和代码文件上传到 OBS 桶中，并将其同步到工作环境。再通过 pip 安装命令安装 1.8.1 版本的 MindSpore。

## 二、补全实验代码

(1) Encoder:

```
'''
代码补充
调用self.embedding，对输入encoder_input进行Embedding编码。
并根据self.perm定义的维度，通过self.trans进行维度变换。

将处理得到的输出命名为embeddings
'''

output1 = self.embedding(encoder_input)
embeddings = self.trans(output1,self.perm)
```

(2) Decoder

```
'''
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
'''

out = self.embedding(decoder_input)
embeddings = self.dropout(out)

        '''
        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
        '''

        out = self.attn(embeddings)
        attn_weights = self.softmax(out)


    '''
    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)
```

## 三、训练、调参、测试模型

在 cmn_zhsim_mini.txt 上训练成功后，改为在 cmn_zhsim.txt 数据集进行全量训练与调参。经过多次训练的观察，训练 50 轮后 loss 基本都收敛在 0.02 左右，故不再调节 learning_rate 与 momentum，而是主要调节 hidden_size

（1）训练模型 50 轮，'hidden_size': 1024
最后 5 轮的记录为：

```
epoch: 46 step: 125, loss is 0.03349245712161064
epoch time: 5458.425 ms, per step time: 43.667 ms
epoch: 47 step: 125, loss is 0.018626783043146133
epoch time: 5428.638 ms, per step time: 43.429 ms
epoch: 48 step: 125, loss is 0.043408509343862534
epoch time: 5526.392 ms, per step time: 44.211 ms
epoch: 49 step: 125, loss is 0.049716848880052567
epoch time: 5627.708 ms, per step time: 45.022 ms
epoch: 50 step: 125, loss is 0.025355283170938492
epoch time: 5593.450 ms, per step time: 44.748 ms
```

翻译结果：

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

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

English ['i', 'will', 'do', 'my', 'best']
中文 我会尽力而为。

English ['i', 'want', 'a', 'dog']
中文 我想要一只狗。

English ['i', 'want', 'a', 'phone']
中文 我想要一只狗。

English ['he', 'is', 'a', 'good', 'boy']
中文 他是个好人。

English ['he', 'is', 'a', 'bad', 'boy']
中文 他是一个好人。

English ['i', 'would', 'like', 'to', 'eat']
中文 我要你去。
```

可见有一些语句是能准确翻出的。

对于错误的例子，值得注意的是，它可能是受数据集里相似数据的影响，如“I want a phone”，被翻译成了“我想要一只狗”，这与“I want a dog”的翻译结果是一样的。这可能是因为数据集里有 I want a dog（或类似的），然后 I want a phone 与 I want a dog 因前三个单词相似，也被翻译成了我想要一只狗。“he is a bad boy”与”he is a good boy”也是类似的，这说明我们训出的模型高度依赖于数据集，最好需要数据集有各种各样分布广泛的数据。

（2）训练模型 50 轮，'hidden_size': 512

翻译结果：

```
English ['he', 'is', 'a', 'good', 'boy']
中文 他是个大男孩。

English ['i', 'want', 'a', 'phone']
中文 我要多一条狗。

English ['i', 'hate', 'tom']
中文 我尊敬汤姆。
```

这里我们将 hidden_size 调成一半，发现果然结果变差了一些，如 I hate tom 在 hidden_size=1024 时能翻译对，而现在就不行，因为 hidden_size 大时模型容量大，能力还是更强一些。

（3）训练模型 50 轮，'hidden_size': 1536

```
epoch: 46 step: 125, loss is 0.010656364262104034
epoch time: 8865.932 ms, per step time: 70.927 ms
epoch: 47 step: 125, loss is 0.00204438716173172
epoch time: 8835.911 ms, per step time: 70.687 ms
epoch: 48 step: 125, loss is 0.0587829487347603
epoch time: 8959.496 ms, per step time: 71.676 ms
epoch: 49 step: 125, loss is 0.023042259737849236
epoch time: 8938.631 ms, per step time: 71.509 ms
epoch: 50 step: 125, loss is 0.0275175292044878
epoch time: 9283.266 ms, per step time: 74.266 ms
```

翻译结果：

```
English ['i', 'hate', 'tom']
中文 我恨蚊子。

English ['he', 'is', 'a', 'good', 'boy']
中文 他是一个好男孩。

English ['he', 'is', 'a', 'bad', 'boy']
中文 他是个胖子

English ['i', 'would', 'like', 'to', 'eat']
中文 我会在这里吃饭。
```

把 hidden_size 再提高到 1536，发现在我的测试样例中，有两个测试结果变好了，“I would like to eat”翻译出来了“我会在这里吃饭”，而之前是“我要你去”，“he is a good boy”翻译出来了“好男孩”，而之前翻译的是“好人”；还有一个变差了，“I hate tom”翻译错了。还有一些翻译错误的结果比 hidden_size 为 1024 时有所变化，但还是错的。

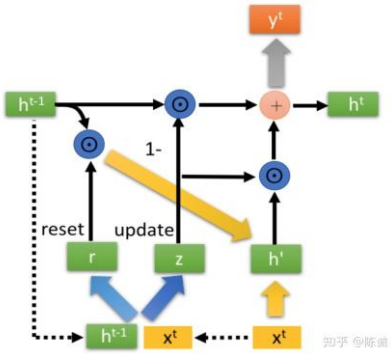
对比 hidden_size 为 1024 的结果，可以说因为模型容量的上升，模型的拟合能力会更强，但也更容易过拟合数据集，因此需要更大更全面的数据集。还有，其 hidden_size 增大 1/3 后，训练时间变长了大概 60%。

## 四、MindSpore 中 GRU 模块输入输出组成与计算原理

（1）阅读 MindSpore Api 文档，可知 GRU 模块的计算原理为：

$$
\begin{aligned}
r_{t+1} &= \sigma(W_{ir}x_{t+1} + b_{ir} + W_{hr}h_{(t)} + b_{hr}) \\
z_{t+1} &= \sigma(W_{iz}x_{t+1} + b_{iz} + W_{hz}h_{(t)} + b_{hz}) \\
n_{t+1} &= \tanh(W_{in}x_{t+1} + b_{in} + r_{t+1} * (W_{hn}h_{(t)} + b_{hn})) \\
h_{t+1} &= (1 - z_{t+1}) * n_{t+1} + z_{t+1} * h_{(t)}
\end{aligned}
$$

其中 $h_{t+1}$ 是在时刻 t+1 的隐藏状态， $x_{t+1}$ 是时刻 t+1 的输入， $h_t$ 为时刻 t 的隐藏状态或时刻 0 的初始隐藏状态。$r_{t+1}$ 、$z_{t+1}$ 、$n_{t+1}$ 分别为重置门、更新门和当前候选集。W，b 为可学习权重和偏置。σ 是 sigmoid 激活函数，* 为 Hadamard 乘积。




根据上图更便于理解，其计算流程就是先利用 $x_{t+1}$ 和上一轮的 $h_t$ 以及权重和偏置算出经重置门和更新门后的值 $r_{t+1}$ 、$z_{t+1}$，这里通过 sigmoid 函数可以将数据变换为 0-1 范围内的数值，从而来充当门控信号（也就是用于控制信号的强弱，值越接近 0，就把对应的值降得越弱）。

接下来用 $h_t$ 经过一个线性变换后的值乘上重置门 $r_{t+1}$，再加上 $x_{t+1}$ 经过一个线性变换后的值，再经过一个 tanh 激活函数，即可算出 $n_{t+1}$（即图中的 $y^t$）。

然后再由 $h_t$ 与更新门 $z_{t+1}$ 的积，加上$(1- z_{t+1})* n_{t+1}$ ,即可得到 $h_{t+1}$ ,作为输入下一层的向量。

（2）MindSpore 中 GRU 模块的 API: mindspore.ops.DynamicGRUV2

定义其类别可指定的参数包括：

direction (str) - 指定 GRU 方向，str 类型。默认值：”UNIDIRECTIONAL”。目前仅支持”UNIDIRECTIONAL”。

cell_depth (int) - GRU 单元深度。默认值：1。

keep_prob (float) - Dropout 保留概率。默认值：1.0。

cell_clip (float) - 输出裁剪率。默认值：-1.0。

num_proj (int) - 投影维度。默认值：0。

time_major (bool) - 如为 True，则指定输入的第一维度为序列长度 num_step ，如为 False 则第一维度为 batch_size 。默认值：True。

activation (str) - 字符串，指定 activation 类型。默认值：”tanh”。目前仅支持取值”tanh”。

gate_order (str) - 字符串，指定 weight 和 bias 中门的排列顺序，可选值为”rzh”或”zrh”。默认值：”rzh”。”rzh”代表顺序为：重置门、更新门、隐藏门。”zrh”代表顺序为：更新门，重置门，隐藏门。

reset_after (bool) - 是否在矩阵乘法后使用重置门。默认值：True。

is_training (bool) - 是否为训练模式。默认值：True。

其输入为：

x (Tensor) - 输入词序列。shape: (num_step,batch_size,input_size) 。数据类型支持 float16。

weight_input (Tensor) - 权重 W{ir,iz,in} 。 shape： (input_size,3×hidden_size) 。 数据类型支持 float16。

weight_hidden (Tensor) - 权重 W{hr,hz,hn} 。 shape： (hidden_size,3×hidden_size) 。 数据类型支持 float16。

bias_input (Tensor) - 偏差 b{ir,iz,in} 。shape： (3×hidden_size) ，或 None 。与输入 init_h 的数据类型相同。

bias_hidden (Tensor) - 偏差 b{hr,hz,hn} 。shape： (3×hidden_size) ，或 None 。与输入 init_h 的数据类型相同。

seq_length (Tensor) - 每个 batch 中序列的长度。shape： (batch_size) 。 目前仅支持 None 。

init_h (Tensor) - 初始隐藏状态。shape： (batch_size,hidden_size) 。 数据类型支持 float16 和 float32。

输出为：
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）
具有与输入 bias_type 相同的数据类型。
output.h（Tensor）-形状的 Tensor。具有与输入 bias_type 相同的数据类型。（num_step，batch_size，hidden_size）
update（Tensor）-形状的 Tensor。具有与输入 bias_type 相同的数据类型。（num_step，batch_size，hidden_size）
reset（Tensor）-形状的 Tensor。具有与输入 bias_type 相同的数据类型。（num_step，batch_size，hidden_size）
new（Tensor）-形状的 Tensor。具有与输入 bias_type 相同的数据类型。（num_step，batch_size，hidden_size）
hidden_new（Tensor）-形状的 Tensor。具有与输入 bias_type 相同的数据类型。（num_step，batch_size，hidden_size）
如果 bias_input 和 bias_hidden 都为 None，则 bias_type 为 init_h 的数据类型。
如果 bias_input 不是 None，则 bias_type 是 bias_input 的数据类型。
如果 bias_input 为 None 且 bias_hidden 为 None，则 bias_type 为 bias_hided 的数据类型。

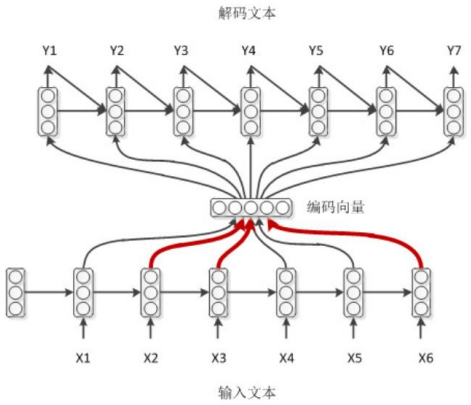
## 五、理解单独定义的 Encoder 和 Decoder 是怎么组成完整的 Seq2Seq 模型的

从代码上看，encoder 就是把输入的语句先 embedding 编码，然后经过一个 GRU，得到编码向量与隐藏层状态。Decoder 就是把输入的 decoder_input 先 embedding 编码，并经过 dropout 层，然后在送入 GRU, 其中输入隐藏层 hidden, 再经过一个线性映射与 logsoftmax 得到最终的 output，同时根据 encoder_output 计算出 attn_applied。

看 Seq2Seq 整体结构的代码即可知道怎么用 encoder 和 decoder 组成完整的 seq2seq 模型。对于输入的源语句 src，先把 src 交给 encoder 编码出 encoder_output，变形得到 decoder_hidden，就是所谓的编码向量。然后往 decoder 里面输入 dst，也就是目标语句，在训练时就是 src 翻译的结果，以及刚刚得到的 decoder_hidden、encoder_output，得到 decoder 的输出，推理时取概率最大的结果作为输出。总结来说，encoder 把输入语句编码为编码向量，作为 decoder 的隐藏层状态，最终可通过 GRU，得到目标输出。

Encoder-Decoder 是一个模型构架，其中编码（encode）由一个编码器将输入序列转化成一个固定维度的稠密向量，解码（decode）阶段将这个稠密向量生成目标译文。

在这里的机器翻译中，encode 就像把语言 A 的意思通过理解转码成了代表其意思的自己的编码，而 decode 就像是利用这个编码把意思用语言 B 表达出来，从而就实现了从语言 A 到语言 B 的翻译。




更具体地说，我们可以用上图来理解，输入语句 X 通过 GRU，编码成了一个编码向量，然后再通过 GRU 解码出语句 Y，以实现从语言 A 到语言 B 的翻译

## 六、梳理 NLLLoss 损失函数

NLLLoss 损失函数用于多分类任务，输入预测值 logits，真实类别 label，实质就是取真实类别的预测值之和，乘以-1，就得到了其损失。

一般我们会用 one-hot 编码实现，对真实类别 label one-hot 编码（真实类别取 1，其它取 0），然后乘以预测值 logits（哈达玛积），然后乘以-1 即可。

```
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))
    self.print('NLLLoss label_one_hot:',label_one_hot, label_one_hot.shape)
    self.print('NLLLoss logits:',logits, logits.shape)
    self.print('xxx:', logits * label_one_hot)
    loss = self.reduce_sum(-1.0 * logits * label_one_hot, (1,))
    self.print('NLLLoss loss:',loss)
    return self.get_loss(loss)
```

由于这里打印不出来这些中间量，我们只能自己计算一下。

在实验中，由于预测的是中文词，个数为 ch_vocab_size 个，一句话填充后固定为 max_seq_length 个，因此输入的 Logits 维度即为 max_seq_length*ch_vocab_size，而 label 的维度显然为 max_seq_length，最后它们经过哈达玛积，再求和得到 loss 的维度为 max_seq_length，最后经过平均变成了一个标量。

实验中 max_seq_length 为 10，ch_vocab_size 为 1116，则：

logits：维度为 10*1116，数据类型为 float.

label: 维度为 10，数据类型为 int

返回值 loss：最后返回的是求平均值过后的 loss，是标量，float。

## 七、MindSpore

(1)pytorch 与 mindspore 的区别

Pytorch 是动态图计算，自由度非常高，至少在训练过程中，想加入一些特殊的输出，或想调试一下等等，比 mindspore 方便太多。Mindspore 是想要走出一条独立于静态图+Eager 和动态图+Trace 的路线，想要做到动态图和静态图的大一统，也就是其能自动调优，动态图+静态图同一套代码。

Mindspore 一优势就在于其速度更快。

(2) ModelArts 与 MindSpore 的问题

ModelArts 旧版 Notebook 的问题很多，一大点就是不好调试。

Mindspore 的问题就是比较难用，为性能而失去了灵活性，感觉自由度比较低，调试也比较麻烦。

结论分析与体会：

本次实验对 seq2seq 进行机器翻译进行了实验，第一次采用 MindSpore 进行实验，第一次对 encoder-decoder 框架进行实验。

首先在配环境上就踩了很大的坑，MindSpore 包升级容易，但包相关的依赖等，“Ascend AI software package”的升级就可不轻松了。最终采用 1.3 的 MindSpore 或者是用新版 notebook 直接选择 1.7 的 MindSpore 环境。

其次对于 encoder-decoder 框架，与本次实验的 seq2seq 模型，也花了很多时间去理解，Encoder-Decoder 是一个模型构架，其中编码（encode）由一个编码器将输入序列转化成一个固定维度的稠密向量，解码（decode）阶段将这个稠密向量生成目标译文。

最后对于 NLLLoss 损失函数，其实这个损失函数非常简单，在代码里用的就是取真实类别的预测值之和，乘以-1，就得到了其损失，但是 MindSpore 在训练过程中，想要自己在函数里面加一点 print，则根本不会输出，就算尝试了它的 ops.Print 算子，发现还是不会输出，而用 callback 的话，又不知道怎么把 NLLLoss 里面的变量保存下来并让 callback 的接口能够访问到，总之非常难搞。最后只能自己计算相关变量的维度。

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. 按实验指导书上方法安装了 MindSpore1.8.1 版本，发现只是给终端中的 python 环境安装了，在运行 ipynb 的 kernel 中并未安装。

解决：需在终端配置好 base 环境后，让 base 环境成为 jupyter 的 kernel。或直接给原有的 kernel 安装 MindSpore1.8.1 版本。

2. 安装 MindSpore1.8.1 版本后，运行时报异常：

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

解决：

仔细翻看输出，发现 MindSpore version 1.8.1 需要的 Ascend AI software package (Ascend Data Center Solution)版本是 1.82，而此环境安装的版本 1.78，不匹配。然后尝试去安装 1.82 版本的 Ascend AI software package，发现并不好搜，而且各种版本号也与 1.82 相差甚远，也就是说很难找到所谓的 1.82 版本的 Ascend AI software package。再然后发现，1.3 的版本其实也能运行代码，只不过有一点警告。

如果按实验指导书所说，要使用>=1.5 的版本，可以用华为云的新版 Notebook，那里有相对应的环境。