2021年10月29日总结

问题解决：

1. 关于Shell variables：
   - 不能有不必要的空格，比如：use SET alpha=beta, not SET alpha = beta
   - Windows下：设置shell variable需要用到set语句，Linux下不需要，都可以通过$来引用值。
   - 下图应该是对于linux（或者git bash）下，运行的示例：

```
train_file=$data_dir/$lang/train.jsonl
dev_file=$data_dir/$lang/valid.jsonl
eval_steps=1000 #400 for ruby, 600 for javascript, 1000 for others
train_steps=50000 #20000 for ruby, 30000 for javascript, 50000 for others
pretrained_model=microsoft/codebert-base #Roberta: roberta-base

python run.py --do_train --do_eval --model_type roberta --model_name_or_path $pretrained_model --train_filename $tr
```

bert-for-tf2

1. 是为tensorflow-v2写的一个库，可以直接构建一个在tf-keras中构建一个bert层。

python相关

1. @staticmethod的作用：

   staticmethod用于修饰类中的方法,使其可以在不创建类实例的情况下调用方法，这样做的好处是执行效率比较高。静态方法就是类对外部函数的封装，有助于优化代码结构和提高程序的可读性。

2. eval()函数：返回括号中运算的结果，比如eval(f.read())，返回读取的f的结果。

3. zip()函数，将对象中对应的元素打包成一个元组。

```
>>>a = [1,2,3]
>>> b = [4,5,6]
>>> c = [4,5,6,7,8]
>>> zipped = zip(a,b)      # 打包为元组的列表
[(1, 4), (2, 5), (3, 6)]
>>> zip(a,c)              # 元素个数与最短的列表一致
[(1, 4), (2, 5), (3, 6)]
>>> zip(*zipped)          # 与 zip 相反，*zipped 可理解为解压，返回二维矩阵式
[(1, 2, 3), (4, 5, 6)]
```

4. 字典的建立可以直接通过{}来实现，array的建立可以直接通过[]来实现。

5. 字符串相关：
   - startwith(string)，判断是否以指定字符串开头，是返回true，不是返回false。

6. 实例化一个类的时候，__init__函数会被执行，可以在__init__中调用这个类的的其它函数，就完成了调用。python中以双下划线__修饰的函数称为魔法函数，都会在类实例化时被调用。

7. tensorflow相关：
   - self.input_spec = keras.layers.InputSpec(shape=input_shape)

     指定图层的每个输入的ndim,dtype和形状，上述代码是指定输入维度为input_shape。
   - keras中的initializer：规定了具体层的权重随机初始化的方法。
   - tf.keras.layers.Embedding()，实现从文本语句到向量的转换，只能作为模型的第一层。

```
 #vocab_size:字典大小
#embedding_dim:本层的输出大小，也就是生成的embedding的维数
#input_length:输入数据的维数，因为输入数据会做padding处理，所以一般是定义的
max_length
keras.layers.Embedding(vocab_size, embedding_dim, input_length =
max_length)
```

- tf.matmul(a, b) 矩阵乘法a 乘b，tf.multiply是元素相乘。
- tf.cast(x,dtype,name)，将x转化为dtype类型。
- tf.transpose(x, perm=[0, 2, 1, 3])，实现转置，多维时按照perm参数安排的顺序进行转置。
- tf.reduce_sum(x, n)，对于tensor x，在地n个维度上进行相加。
- tf.train.Checkpoint：变量的保存与恢复，Tensorflow的Checkpoint机制将可追踪变量以二进制的方式储存成一个.ckpt文件，储存了变量的名称及对应张量的值， 只保存模型的参数，不保存模型的计算过程 。
- 高维tensor相乘：https://www.jianshu.com/p/e649b7f04ee1，往往是从后面能够相乘的二维开始（比如： M1的shape为 (c,n,s), M2的shape为（c,s,m）时，M3的shape为（c,n,m)），然后如果前面维度不相同，能够使用广播的就先使用广播（比如： M1的shape为(n,c,h,s), M2的shape为（1,1,s,w）时，M3的shape为（n,c,h,w)。

问题

1. 关于conda进行版本控制的问题，使用命令conda install tensorflow==2.0.0是可以正常安装的，但是使用conda install tensorflow>=2.0.0时就会卡死没有反应（使用pip效果相同）。
2. 在bert-for-tf2中，单步调试会卡死。
3. 关于GPU，电脑是配的轻薄本，配置的GPU不在CUDA支持的列表里。
4. 函数定义与引用参数不一致。

```
class DecoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(DecoderLayer, self).__init__()

    self.mha1 = MultiHeadAttention(d_model, num_heads)
    self.mha2 = MultiHeadAttention(d_model, num_heads)

    self.ffn = point_wise_feed_forward_network(d_model, dff)

    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm3 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)
    self.dropout3 = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,
           look_ahead_mask, padding_mask):
    # enc_output.shape == (batch_size, input_seq_len, d_model)

    attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)
    attn1 = self.dropout1(attn1, training=training)
    out1 = self.layernorm1(attn1 + x)
```

总结

1. 在真正写代码的时候，因为操作的都是类似于矩阵的tensor，所以要很清楚地明白，对应的每一个维度的含义，还有就是经过了不同的层后矩阵的维度。
2. 克服对英语网页的排斥，从英文网页中的检索很重要。
3. conda、pip和pycharm在联网时都使用了自己的代理，所以要将自己电脑的代理关掉，不然会报错。