HW4-批改标准

本次作业基础分15分，扣分制。有加分项，加分最多2分。

代码

- `main.py` 能正常运行，否则扣10分。
- 相关代码填补正确。参考代码如下：

`rnn_cell.py`

1. GRUCell的call函数实现正确

```
with vs.variable_scope("candidate"):
    c = self._activation(_linear([inputs, r * state], self._num_units, True,
                  self._bias_initializer, self._kernel_initializer))
new_h = u * state + (1 - u) * c
return new_h, new_h
```

只要new_h计算正确即可，如果new_h计算错误，扣1分。c作为中间变量可以灵活处理。new_h用其他API（如tf.layers.dense等）计算亦可，self._activation可以直接用tanh函数替换，u和(1-u)的位置可互换。易错点如下：

- 未使用激活函数

2. BasicLSTMCell的call函数实现正确

```
def call(self, inputs, state):
    """Long short-term memory cell (LSTM)."""
    sigmoid = math_ops.sigmoid
    # Parameters of gates are concatenated into one multiply for efficiency.
    if self._state_is_tuple:
      c, h = state
    else:
      c, h = array_ops.split(value=state, num_or_size_splits=2, axis=1)

    concat = _linear([inputs, h], 4 * self._num_units, True)

    # i = input_gate, j = new_input, f = forget_gate, o = output_gate
    i, j, f, o = array_ops.split(value=concat, num_or_size_splits=4, axis=1)

    new_c = (
        c * sigmoid(f + self._forget_bias) + sigmoid(i) * self._activation(j))
    new_h = self._activation(new_c) * sigmoid(o)

    if self._state_is_tuple:
      new_state = LSTMStateTuple(new_c, new_h)
    else:
      new_state = array_ops.concat([new_c, new_h], 1)
    return new_h, new_state
```

该部分需要正确地计算new_c和new_h，中间步骤可灵活处理。易错点如下：

- 计算公式错误。如果i, j, f, o任意一个计算错误，扣1分；如果new_c计算错误，扣1分；如果new_h计算错误，扣1分。
- 计算concat时（即计算i, j, f, o时）没有加偏置，即没有将_linear的第3个参数设为true，判为i, j, f, o计算错误，扣1分。
- forget gate的偏置不作要求，即不使用self._forget_bias也不扣分。

`model.py`

1. Placeholders部分实现正确

```
self.texts_length = tf.placeholder(tf.int32, (None) , 'texts_length')  # shape: [batch]
self.labels = tf.placeholder(tf.int64, (None) , 'labels')  # shape: [batch]
```

2. embedding inputs部分实现正确

```
self.embed_input = tf.nn.embedding_lookup(self.embed, self.index_input) #shape: [batch, length,
num_embed_units]
```

3. RNNCell部分实现正确

```
cell_fw = MultiRNNCell([BasicLSTMCell(num_units) for _ in range(num_layers)])
cell_bw = MultiRNNCell([BasicLSTMCell(num_units) for _ in range(num_layers)])
```

该部分要求实现num_layers层含有num_units个神经元的RNNCell，实现方法不唯一。如果没有使用num_units或num_layers参数，即认为该部分实现错误，扣1分。

4. 双向RNN部分实现正确

```
outputs, states = tf.nn.bidirectional_dynamic_rnn(cell_fw, cell_bw, self.embed_input,
self.texts_length, dtype=tf.float32, scope="rnn")
```

5. self-attention部分实现正确

```
with tf.variable_scope('logits'):
#todo: implement self-attention mechanism, feel free to add codes to calculate temporary results
    Ws1 = tf.get_variable("Ws1", [2*num_units, param_da])
    Ws2 = tf.get_variable("Ws2", [param_da, param_r])

    fc_in = tf.nn.tanh(tf.einsum('aij,jk->aik', H, Ws1)) # (batch, length, param_da)
    A_T = tf.nn.softmax(tf.einsum('aij,jk->aik', fc_in, Ws2), dim=1) # (batch, length, param_r)
    H_T = tf.transpose(H, perm=[0, 2, 1]) # shape: (batch, 2*num_units, length)
    M_T = tf.matmul(H_T, A_T) # shape: (batch, 2*num_units, param_r)

    M = tf.transpose(M_T, perm=[0, 2, 1]) # shape: [batch, param_r, 2*num_units]
    flatten_M = tf.reshape(M, shape=[batch_size, param_r*2*num_units]) # shape: [batch,
param_r*2*num_units]
    logits = tf.layers.dense(flatten_M, num_labels, activation=None, name='projection') # shape:
[batch, num_labels]
```

该部分的实现方式很多，根据公式是否正确实现来给分：

$$A = softmax(W_{s2}tanh(W_{s1}H^T))$$
$$M = AH$$

易错点如下：

- A/A^T 计算错误，扣1分。易错点在于$softmax$的dim参数设置，如果该处计算采用H连续右乘W_{s1}和W_{s2}，则计算结果应为A^T，需要对length所在的维度（即上述标准代码中的dim=1）进行softmax归一化；如果计算结果为A，则同样应对length所在维度（即dim=2）进行softmax归一化。
- M 计算错误，扣1分。

6. 惩罚项部分实现正确

```
identity = tf.reshape(tf.tile(tf.diag(tf.ones([param_r])), [batch_size, 1]), [batch_size,
param_r, param_r])
A = tf.transpose(A_T, perm=[0, 2, 1]) # (batch, param_r, length)
AA_T = tf.matmul(A, A_T)
self.penalized_term = tf.reduce_mean(tf.square(tf.norm(AA_T - identity, ord='euclidean', axis=
[1, 2])), name='penalize')
```

该部分仍根据公式正确与否来给分，如果惩罚项计算错误，扣1分：

$$P = ||(AA^T - I)||_F^2$$

惩罚项部分需要用到A，如果self-attention部分中A的计算错误，但惩罚项的计算过程正确，则本部分不再重复扣分。易错点如下：

- Frobenius范数计算错误。矩阵的Frobenius范数计算公式如下：

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

使用tf.norm计算时应注意将ord设置为euclidean，同时将axis设置为除了batch_size的另外两维（即1和2），以保证结果正确。如果直接当作向量来求2范数，也算正确，但无论哪种方式均需要平方。

- 如果将tf.reduce_mean写为tf.reduce_sum，不扣分。

报告

- 一共包含三个主要实验
    - 完整地分析了BasicRNNCell、GRUCell和BasicLSTMCell的性能。该实验应包含3张loss-epoch的图像和3张accuracy-epoch的图像（可画在同一张图里），以及对3种cell效果的文字分析。缺一张图扣1分，缺少文字分析扣1分。
    - 汇报了最终网络的结构与性能。该实验应包含最终网络的loss-epoch图像和accuracy-epoch图像，网络结构和超参数的详细描述，以及分类准确率的数值结果。缺一张图扣1分，缺少网络结构和超参描述扣1分，缺少最终网络分类准确率的数值结果扣1分。
    - 最终网络的测试。该实验由助教根据学生提交的result.txt和SST的测试集结果来计算，以得到最终网络在测试集上的分类准确率。该部分分数分为3档：准确率<42%，扣1分；准确率处于42%~46%之间，该项不加分也不扣分；准确率>46%，加1分。如果未提交result.txt，扣2分。
- 加分项（不超过2分）
    - 最终网络的测试集准确率高于46%，加1分
    - 汇报最终网络性能时总结了超参的探索过程，详细分析了不同超参对分类结果的影响，加1分。
    - 使用Tensorboard绘制实验图像。该功能需要对`main.py`进行改写，正确改写代码并在报告中使用Tensorboard进行画图则加1分。
- 抄袭：
    - 本次实验的大部分代码逻辑较为简单，只需关注self-attention和惩罚项计算。