2151131-朱沙桐-课后作业4

2151131 朱沙桐

1. 代码补全

1.1. PyTorch

```
class RNN_model(nn.Module):
    def __init__(self, batch_sz, vocab_len, word_embedding, embedding_dim, lstm_hidden_dim):
        super(RNN_model, self).__init__()

        self.word_embedding_lookup = word_embedding
        self.batch_size = batch_sz
        self.vocab_length = vocab_len
        self.word_embedding_dim = embedding_dim
        self.lstm_dim = lstm_hidden_dim

        ##############################################
        # here you need to define the "self.rnn_lstm"  the input size is "embedding_dim" and the output size is "lstm_hidden_dim"
        # the lstm should have two layers, and the  input and output tensors are provided as (batch, seq, feature)
        # ???
        '''
        在RNN_model类的初始化方法__init__中定义LSTM网络
        它的输入大小是embedding_dim，输出大小（隐藏状态的大小）是lstm_hidden_dim
        网络有两层
        且输入和输出张量的格式是(batch, seq, feature)，这是通过batch_first=True参数指定的
        '''
        self.rnn_lstm = nn.LSTM(
            input_size=embedding_dim, hidden_size=lstm_hidden_dim, num_layers=2, batch_first=True)
        ##############################################

        self.fc = nn.Linear(lstm_hidden_dim, vocab_len)
        self.apply(weights_init)  # call the weights initial function.

        self.softmax = nn.LogSoftmax()  # the activation function.
        # self.tanh = nn.Tanh()

    def forward(self, sentence, is_test=False):
        batch_input = self.word_embedding_lookup(
            sentence).view(1, -1, self.word_embedding_dim)

        ################################################
        # here you need to put the "batch_input"  input the self.lstm which is defined before.
        # the hidden output should be named as output, the initial hidden state and cell state set to zero.
        # ???
```

```
        '''
        在RNN_model类的forward方法中，将batch_input传递给LSTM网络并处理输出
        (hn, cn)代表LSTM网络的最终隐藏状态和细胞状态，但由于初始化状态没有被明确设置，我们传入None作为网络的初始隐藏状态和细胞状态，让网络自动处理它们
        '''
        output, (hn, cn) = self.rnn_lstm(batch_input, None)
        ################################################

        out = output.contiguous().view(-1, self.lstm_dim)

        out = F.relu(self.fc(out))

        out = self.softmax(out)

        if is_test:
            prediction = out[-1, :].view(1, -1)
            output = prediction
        else:
            output = out
        # print(out)
        return output
```

在RNN_model类的初始化方法**init**中定义LSTM网络，它的输入大小是embedding_dim，输出大小（隐藏状态的大小）是lstm_hidden_dim。网络有两层，且输入和输出张量的格式是(batch, seq, feature)，这是通过batch_first=True参数指定的。

`self.rnn_lstm = nn.LSTM(input_size=embedding_dim, hidden_size=lstm_hidden_dim, num_layers=2, batch_first=True)`

在RNN_model类的forward方法中，将batch_input传递给LSTM网络并处理输出。(hn, cn)代表LSTM网络的最终隐藏状态和细胞状态，但由于初始化状态没有被明确设置，我们传入None作为网络的初始隐藏状态和细胞状态，让网络自动处理它们：

`self.rnn_lstm = nn.LSTM(input_size=embedding_dim, hidden_size=lstm_hidden_dim, num_layers=2, batch_first=True)`

1.2. TensorFlow

Learn2Carry-exercise.ipynb

```
@tf.function
def call(self, num1, num2):
    '''
    此处完成上述图中模型
    '''
    num1_emb = self.embed_layer(num1)
    num2_emb = self.embed_layer(num2)
    input_emb = tf.concat([num1_emb, num2_emb], axis=-1)
    rnn_out = self.rnn_layer(input_emb)
    logits = self.dense(rnn_out)

    return logits
```

- `num1_emb = self.embed_layer(num1)`：使用 `embed_layer`（一个预先定义的嵌入层）将输入 `num1` 转换成嵌入表示 `num1_emb`。

- `num2_emb = self.embed_layer(num2)`：将输入 `num2` 转换成嵌入表示 `num2_emb`。
- `input_emb = tf.concat([num1_emb, num2_emb], axis=-1)`：将 `num1_emb` 和 `num2_emb` 沿着最后一个维度（`axis=-1`）拼接起来，生成一个新的嵌入输入 `input_emb`。
- `rnn_out = self.rnn_layer(input_emb)`：通过一个预定义的循环神经网络层（`rnn_layer`）处理拼接后的嵌入输入 `input_emb`，得到循环网络的输出 `rnn_out`。
- `logits = self.dense(rnn_out)`：通过一个全连接层（`dense`）处理循环网络的输出 `rnn_out`，生成最终的输出 `logits`。

poem_generation_with_RNN-exercise.ipynb

```
@tf.function
def call(self, inp_ids):
    '''
    此处完成建模过程，可以参考Learn2Carry
    '''
    input_emb = self.embed_layer(inp_ids)
    rnn_out = self.rnn_layer(input_emb)
    logits = self.dense(rnn_out)
    return logits
```

- `input_emb = self.embed_layer(inp_ids)`：调用一个嵌入层（`embed_layer`），将输入的 `inp_ids` 转换为嵌入向量 `input_emb`。嵌入层处理具有大量类别的离散数据，如单词或标识符，将它们映射到一个连续的、更小维度的向量空间中。
- `rnn_out = self.rnn_layer(input_emb)`：使用一个循环神经网络层（`rnn_layer`）处理嵌入向量 `input_emb`。
- `logits = self.dense(rnn_out)`：通过一个全连接层（`dense`）处理RNN的输出 `rnn_out`，得到最终的输出 `logits`。将网络的学习表示转换为最终的输出格式，如分类问题中的类别得分。

```
@tf.function
def train_one_step(model, optimizer, x, y, seqlen):
    '''
    完成一步优化过程，可以参考之前做过的模型
    '''
    with tf.GradientTape() as tape:
        logits = model(x)
        loss = compute_loss(logits, y, seqlen)
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
    return loss
```

`tf.GradientTape()`：上下文管理器，用于自动跟踪在其作用域内执行的操作，以便后续计算梯度。这对于自动微分非常有用。

`logits = model(x)`：通过模型传递输入数据 `x`，获取模型输出 `logits`。

`loss = compute_loss(logits, y, seqlen)`：计算损失函数，根据模型的输出 `logits`、真实标签 `y` 和序列长度 `seqlen` 来计算损失值。是模型性能的一个量度，优化的目标是最小化这个值。

`grads = tape.gradient(loss, model.trainable_variables)`：计算损失函数关于模型可训练参数的梯度。训练过程中的关键，梯度指示了损失函数如何随模型参数变化而变化。

`optimizer.apply_gradients(zip(grads, model.trainable_variables))`：应用计算出的梯度来更新模型的参数。进行模型优化，根据梯度和指定的优化策略（如 SGD、Adam 等）调整参数值以减少损失。

```python
def gen_tang_poem(begin_word, max_length=100, max_sentences=10):
    # 假设模型和必要的映射已经被加载和初始化
    state = [tf.random.normal(shape=(1, 128), stddev=0.5),
             tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id[begin_word]], dtype=tf.int32)  # 使用开始字初始化
    poem = [begin_word]
    end_token_id = word2id['eos']  # 假设'eos'为结束符的ID

    sentences_generated = 0  # 已生成的句子数量
    for _ in range(max_length):  # 限制诗的最大长度
        cur_token, state = model.get_next_token(cur_token, state)
        token_id = cur_token.numpy()[0]

        if token_id == end_token_id:
            sentences_generated += 1
            if sentences_generated >= max_sentences:
                break  # 如果生成了足够数量的句子，则结束循环
            else:
                poem.append('\n')  # 用换行符代替'eos'，分隔句子
        else:
            poem.append(id2word.get(token_id, '<unk>'))

    return ''.join(poem)


# 生成不同开头词汇的唐诗
words = ["春", "日", "红", "山", "夜", "湖", "海", "月"]
for word in words:
    print(f"《{word}》")
    print(gen_tang_poem(word))
    print()
```

生成诗歌。

2. 模型解释

2.1. RNN

循环神经网络（RNN）是一种用于处理序列数据的神经网络。与传统的神经网络不同，RNN具有内部状态（记忆）来处理输入序列中的元素。这使得RNN特别适用于处理时间序列数据、自然语言文本、语音等序列化信息。

RNN的核心是一个循环单元，该单元在处理序列的每个时间步时都会被激活。在每个时间步，循环单元接收两个输入：当前时间步的输入数据和上一个时间步的隐藏状态。基于这两个输入，循环单元会更新自己的隐藏状态，并可能生成一个输出。这个隐藏状态可以被视为网络的“记忆”，它捕获了到目前为止处理的序列的信息。

ENN的主要优点是：

- **处理序列数据的能力：** 由于RNN设计之初就考虑到了时间序列的特点，因此它非常适合处理任何形式的序列数据。
- **灵活的输入/输出长度：** RNN能够处理不同长度的输入和输出序列，这使得它在多种应用场景中非常有用，比如在自然语言处理任务中。

2.2. LSTM

长短期记忆网络（LSTM）是一种特殊类型的循环神经网络（RNN），专为解决标准RNN在处理长序列数据时面临的梯度消失和梯度爆炸问题而设计。由Hochreiter和Schmidhuber在1997年首次提出，LSTM通过引入几个门控机制来调节信息的流动，使得网络能够在长时间序列中保持信息并捕获长期依赖关系。

LSTM的核心在于其内部结构的设计，它包含以下几个关键部分：

- **遗忘门（Forget Gate）**：决定从单元状态中丢弃什么信息。它通过观察当前输入和上一个时间步的隐藏状态，生成一个在0到1之间的数值，用以表示保留多少旧信息。
- **输入门（Input Gate）**：决定哪些新信息将被添加到单元状态中。它由两部分组成：一个“输入门层”决定哪些值我们将更新，和一个“候选值层”创建一个候选值向量，该向量将被加到状态中。
- **单元状态（Cell State）**：网络的“记忆”部分，贯穿整个链，只有微小的线性交互，信息流动几乎不受阻碍。单元状态有能力在整个处理过程中携带相关信息，遗忘门和输入门共同作用来更新它。
- **输出门（Output Gate）**：根据单元状态和当前的输入，决定最终的输出。输出是单元状态的一个过滤版本，只输出我们想要的部分。

LSTM的主要优点是：

- **解决梯度消失问题**：通过精心设计的门控机制，LSTM能够在长序列中保持梯度的稳定，使得训练过程更加高效。
- **灵活性**：LSTM能够学习和记忆长期和短期的依赖关系，这使得它在各种序列任务中都非常有效，特别是那些需要理解长距离上下文的任务。

2.3. GRU

门控循环单元（GRU）是一种循环神经网络（RNN）的变体，旨在解决标准RNN在处理长序列数据时遇到的梯度消失问题。GRU由Cho等人于2014年提出，其设计思想与长短期记忆网络（LSTM）类似，但更为简化，这使得GRU在某些任务上比LSTM更高效，同时仍然保持了处理长期依赖关系的能力。

GRU通过引入两个主要的门控机制来调节信息的流动，简化了LSTM的结构，这两个门分别是：

- **更新门（Update Gate）**：决定多少之前的记忆会被保留下来。更新门帮助模型决定在当前状态的记忆中保留多少之前的状态。这类似于LSTM中的遗忘门和输入门的组合。
- **重置门（Reset Gate）**：决定多少过去的信息会被忽略。它允许模型丢弃与当前任务不相关的状态信息，这对于模型捕获时间序列中的短期依赖关系非常有用。

GRU相比于LSTM的优点：

- **模型简化**：GRU有更少的参数，因为它合并了遗忘门和输入门到一个单一的更新门，并且没有单元状态，只有隐藏状态，这使得GRU比LSTM更简单，训练速度通常更快。
- **参数效率**：由于结构简化，GRU在某些情况下能够使用更少的参数达到与LSTM相似或甚至更好的性能，特别是在参数数量和计算资源有限的情况下。
- **灵活性**：虽然GRU简化了门控机制，但它仍然非常灵活，能够捕捉序列中的长距离依赖关系，对于各种序列建模任务都非常有效。

3. 诗歌生成过程

`process_poems1` 和 `process_poems2` 函数被用于处理诗歌数据集。读取文本文件中的诗歌，清洁和格式化文本数据，然后将每个字转换为对应的整数索引。在这个过程中，诗歌被过滤和排序，以确保数据质量和一致性。最终，这些函数返回诗歌的向量表示和字到整数索引的映射。

`generate_batch` 函数用于创建训练批次，这是模型训练过程中的一个关键步骤。它将诗歌向量分割成多个批次，并为每个批次准备好输入数据和目标数据。

`run_training` 函数是模型训练的核心。它首先加载和准备数据，然后初始化RNN模型和优化器。接下来，它进入训练循环，不断地进行前向传播、计算损失、进行反向传播和参数更新。

`to_word` 函数将模型预测的整数索引转换回对应的字，而 `pretty_print_poem` 函数则负责格式化和打印生成的诗歌，以便人类阅读。

`gen_poem` 函数使用训练好的模型来生成诗歌。它接受一个开始字作为输入，然后模型基于这个开始字连续生成下一个字，直到遇到结束标记或达到一定长度限制。

4. 运行截图

PyTorch版本训练截图

```
epoch   8 batch number 187 loss is:  5.719720840454102
prediction [10, 63, 102, 25, 53, 0, 6, 62, 10, 8, 21, 1, 6, 30, 4, 45, 161, 0, 4, 51, 48, 65, 326, 1, 16, 108, 25, 32, 45, 0, 6, 21, 77, 63, 7, 1, 15, 7, 10, 16, 42, 0, 10, 177, 10, 23, 21, 1, 3, 3]
b_y       [703, 522, 259, 282, 177, 0, 5, 62, 191, 160, 21, 1, 109, 562, 708, 274, 1663, 0, 1115, 601, 1382, 102, 84, 1, 2568, 575, 27, 38, 186, 0, 134, 1435, 322, 218, 7, 1, 25, 115, 785, 119, 189, 0, 390, 1203, 191, 118, 133, 1, 3, 3]
********************************
epoch   8 batch number 188 loss is:  6.06257438659668
prediction [10, 44, 9, 132, 63, 0, 66, 7, 8, 117, 102, 1, 396, 79, 6, 104, 218, 0, 17, 57, 126, 208, 79, 1, 53, 72, 9, 466, 0, 0, 5, 14, 4, 19, 7, 1, 15, 23, 4, 15, 16, 0, 10, 95, 49, 172, 205, 1, 3, 3]
b_y       [97, 44, 7, 14, 114, 0, 46, 33, 182, 117, 127, 1, 11, 301, 576, 618, 63, 0, 17, 311, 249, 71, 25, 1, 1520, 587, 136, 826, 929, 0, 24, 899, 422, 1139, 777, 1, 65, 670, 92, 75, 16, 0, 481, 95, 314, 1373, 205, 1, 3, 3]
********************************
epoch   8 batch number 189 loss is:  5.932765007019043
prediction [10, 7, 9, 62, 20, 0, 6, 30, 39, 19, 131, 1, 4, 141, 6, 18, 63, 0, 5, 18, 4, 9, 132, 1, 17, 290, 7, 69, 63, 0, 7, 381, 10, 17, 141, 1, 4, 30, 9, 19, 43, 0, 15, 182, 92, 9, 205, 1, 3, 3]
b_y       [13, 73, 258, 168, 135, 0, 86, 30, 376, 36, 131, 1, 114, 1948, 330, 19, 2181, 0, 22, 442, 72, 9, 25, 1, 16, 592, 80, 800, 1255, 0, 402, 141, 203, 539, 105, 1, 37, 505, 27, 1206, 141, 0, 66, 197, 637, 1373, 205, 1, 3, 3]
********************************
epoch   8 batch number 190 loss is:  5.8139147758483889
prediction [10, 14, 102, 26, 23, 0, 102, 246, 39, 31, 5, 1, 6, 23, 9, 45, 45, 0, 6, 64, 4, 9, 326, 1, 72, 44, 9, 25, 29, 0, 5, 32, 41, 10, 21, 1, 15, 373, 9, 48, 63, 0, 4, 96, 49, 47, 23, 1, 3, 3]
b_y       [667, 660, 876, 234, 25, 0, 32, 1230, 991, 18, 127, 1, 23, 119, 113, 139, 2264, 0, 5, 64, 391, 181, 88, 1, 697, 1684, 708, 129, 212, 0, 185, 183, 1233, 160, 135, 1, 365, 861, 6, 567, 96, 0, 656, 73, 251, 773, 570, 1, 3, 3]
********************************
epoch   8 batch number 191 loss is:  5.781478404998779
prediction [10, 236, 6, 6, 63, 0, 21, 66, 16, 10, 5, 1, 6, 7, 9, 19, 160, 0, 4, 160, 4, 20, 5, 1, 72, 104, 9, 6, 42, 0, 6, 279, 4, 96, 20, 1, 15, 30, 4, 31, 188, 0, 4, 51, 39, 64, 151, 1, 3, 3]
b_y       [236, 41, 66, 14, 348, 0, 33, 131, 1007, 212, 30, 1, 196, 223, 20, 19, 100, 0, 667, 1667, 42, 9, 11, 1, 315, 3191, 40, 630, 436, 0, 1267, 279, 195, 177, 1247, 1, 27, 151, 5, 422, 188, 0, 103, 37, 2999, 578, 228, 1, 3, 3]
********************************
epoch   8 batch number 192 loss is:  5.74879932403564453
prediction [6, 197, 66, 110, 39, 0, 25, 74, 49, 15, 15, 1, 39, 18, 6, 21, 159, 0, 6, 93, 17, 18, 7, 1, 7, 6, 25, 31, 8, 0, 10, 104, 45, 32, 79, 1, 4, 161, 9, 26, 41, 0, 15, 132, 4, 142, 6, 1, 3, 3]
b_y       [158, 5, 68, 43, 1798, 0, 25, 74, 192, 47, 15, 1, 539, 169, 11, 343, 1145, 0, 594, 93, 883, 1840, 55, 1, 2037, 512, 279, 143, 701, 0, 22, 3372, 45, 103, 139, 1, 233, 161, 452, 26, 244, 0, 7, 35, 342, 249, 224, 1, 3, 3]
********************************
epoch   8 batch number 193 loss is:  5.819192409515381
prediction [10, 8, 9, 9, 399, 0, 26, 279, 4, 39, 58, 1, 6, 119, 154, 16, 145, 0, 7, 17, 41, 6, 14, 1, 72, 74, 9, 407, 38, 0, 7, 5, 79, 4, 144, 1, 15, 5, 4, 10, 170, 0, 15, 179, 4, 37, 305, 1, 3, 3]
b_y       [73, 682, 149, 881, 525, 0, 1267, 440, 91, 696, 58, 1, 3449, 319, 181, 53, 193, 0, 33, 1678, 529, 6, 658, 1, 1668, 17, 3733, 407, 63, 0, 1474, 20, 3456, 5649, 506, 1, 532, 111, 1556, 99, 73, 0, 9, 1320, 149, 1025, 305, 1, 3, 3]
********************************
epoch   8 batch number 194 loss is:  5.9436397552490234
prediction [10, 18, 25, 58, 162, 0, 7, 119, 77, 108, 72, 1, 7, 7, 25, 42, 16, 0, 4, 104, 4, 6, 7, 1, 72, 77, 25, 174, 72, 0, 7, 119, 17, 194, 93, 1, 15, 23, 19, 20, 17, 0, 10, 39, 77, 30, 20, 1, 3, 3]
b_y       [140, 17, 91, 666, 380, 0, 332, 119, 1237, 52, 127, 1, 1810, 156, 9, 2057, 96, 0, 1280, 1548, 1010, 46, 25, 1, 193, 434, 299, 174, 289, 0, 733, 52, 298, 194, 241, 1, 453, 1
```

PyTorch版本生成结果

```
(pwsh) exercise master ↑1 ?1 ~2 -1 27ms                                                    base 3.11.4  100%  27, 20:55
>> & D:/Python/Python311/python.exe e:/#03-Homework/6大三下-深度学习/exercise/chap6_RNN/tangshi_for_pytorch/main.py
error
inital  linear weight
D:\Python\Python311\Lib\site-packages\torch\nn\modules\module.py:1518: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  return self._call_impl(*args, **kwargs)
error
inital  linear weight
自有人心事，何如此路长。
浮生如可坐，一片一片石。
error
inital  linear weight
山翠长秋月，风光不可闻。
风尘不可见，风雨不成尘。
error
inital  linear weight
风前一片石，风雨一相携。
一去无人处，看君不可留。
error
inital  linear weight
湖风送客心，风尘不可闻，一生何处闻时看。
风前一片前落，花发千。
error
inital  linear weight
海上秋风发，一片落花心。
error
inital  linear weight
坐见一时节，一望一相见。
error
inital  linear weight
君子不能里，一望一相携。
```

TensorFlow版本生成结果

```
《春》
春雨落，月色寒花，一枝。
生不可知，不见君人事。
有不可知，不知何处处。
来不可知，不见君人事。
有不可知，不知何处处。
来不可知，不见君人事。
有不可知，不知何处处。
来不可知，不见君人事。
有不可知

《日》
日来无事不知君。
有无人事，何人不可知。
生无事事，不得不知君。
马无人事，何人不可知。
生无事事，不得不知君。
马无人事，何人不可知。
生无事事，不得不知君。
马无人事，何人不可知。
生无事事，不得不

《红》
红裳上柳花花落，红叶红花一片花。
上不知人不得，不知何处是君人。
...
生不可知，不知何处处。
来不可知，不见君人事。
有不可知，

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```