深度学习
Lab8-attention mechanism

兰韵诗

本次Lab有作业，请在4月21日结束之前提交！

Lab6参考答案

```python
class Sequence_Modeling(nn.Module):
    def __init__(self, vocab_size, embedding_size, num_outputs, hidden_size):
        super(Sequence_Modeling, self).__init__()
        self.emb_layer = nn.Embedding(vocab_size, embedding_size)
        self.gru_layer = GRU(embedding_size, hidden_size)
        self.linear = nn.Linear(hidden_size, num_outputs)

    def forward(self, sent, state):
        '''
        sent --> (B, S) where B = batch size, S = sequence length
        sent_emb --> (B, S, I) where B = batch size, S = sequence length, I = num_inputs
        state --> (B, 1, H), where B = batch_size, num_hiddens
        你需要利用定义好的emb_layer, gru_layer和linear,
        补全代码实现歌词预测功能,
        sent_outputs的大小应为(B, S, O) where O = num_outputs, state的大小应为(B, 1, H)
        '''
        sent_emb = self.emb_layer(sent)
        sent_hidden, state = self.gru_layer(sent_emb, state)
        sent_states = self.linear(sent_hidden)

        return sent_states, state
```

```python
def forward(self, inputs, state):
    '''
    补全GRU的前向传播,
    不能调用pytorch中内置的GRU函数及操作
    '''

    H = state
    outputs = []
    for i in range(inputs.size(1)):
        Z = torch.sigmoid(torch.matmul(inputs[:, i, :], self.W_xz) + torch.matmul(H, self.W_hz) + self.b_z)
        R = torch.sigmoid(torch.matmul(inputs[:, i, :], self.W_xr) + torch.matmul(H, self.W_hr) + self.b_r)
        H_tilda = torch.tanh(torch.matmul(inputs[:, i, :], self.W_xh) + torch.matmul(R * H, self.W_hh) + self.b_h)
        H = Z * H + (1 - Z) * H_tilda
        outputs.append(H)
    outputs = torch.cat(outputs, 0)
    return outputs.transpose(1, 0), H
```

Lab6常见问题

状态更新出错

示例二：没有用H_new更新状态H，导致输出也错误

代码块

```
    def forward(self, inputs, H):
        '''
        利用定义好的参数补全GRU的前向传播，
        不能调用pytorch中内置的GRU函数及操作
        '''
        # ==========
        # todo '''请补全GRU网络前向传播'''
        outputs=[]
        for X in inputs:
            z_t = torch.sigmoid(X @ self.Wz + H @ self.Uz + self.bz)
            r_t = torch.sigmoid(X @ self.Wr + H @ self.Ur + self.br)
            h_tilde_t = torch.tanh(X @ self.Wh + (r_t * H) @ self.Uh + self.bh)
            H_new = z_t * H + (1 - z_t) * h_tilde_t
            outputs.append(H.unsqueeze(0))
        outputs = torch.cat(outputs,dim=0)

        # ==========
        return outputs,H_new
```

Lab6常见问题

循环封装方式效率低

```
def forward(self, inputs, H):
        '''
        利用定义好的参数补全GRU的前向传播，
        不能调用pytorch中内置的GRU函数及操作
        '''
        # ==========
        # todo '''请补全GRU网络前向传播'''
        Z = torch.sigmoid(torch.matmul(inputs, self.W_xz) + torch.matmul(H, self.W_hz) +
        R = torch.sigmoid(torch.matmul(inputs, self.W_xr) + torch.matmul(H, self.W_hr) +
        H_tilde = torch.tanh(torch.matmul(inputs, self.W_xh) + torch.matmul(R * H, self.W
        H_new = Z * H + (1 - Z) * H_tilde

        return H_new


def forward(self, sent, state):
        '''
        sent --> (B, S) where B = batch size, S = sequence length
        sent_emb --> (B, S, I) where B = batch size, S = sequence length, I = num_inputs
        state --> (B, 1, H), where B = batch_size, num_hiddens
        你需要利用定义好的emb_layer, gru_layer和linear,
        补全代码实现歌词预测功能，
        sent_outputs的大小应为(B, S, O) where O = num_outputs, state的大小应为(B, 1, H)
        '''

        sent_emb = self.emb_layer(sent)

        # ==========
        # todo '''请补全代码'''
        sent_outputs = []

        for i in range(sent_emb.size(1)):
            input_emb = sent_emb[:, i, :]  # (B, I)
            state = self.gru_layer(input_emb, state.squeeze(0))
            output = self.linear(state)
            sent_outputs.append(output.unsqueeze(1))
```

Lab8

1.熟悉用编码器-解码器的框架解决序列逆置任务的流程
2.补全rnn_with_atten.py文件中的融合注意力机制的RNN模型

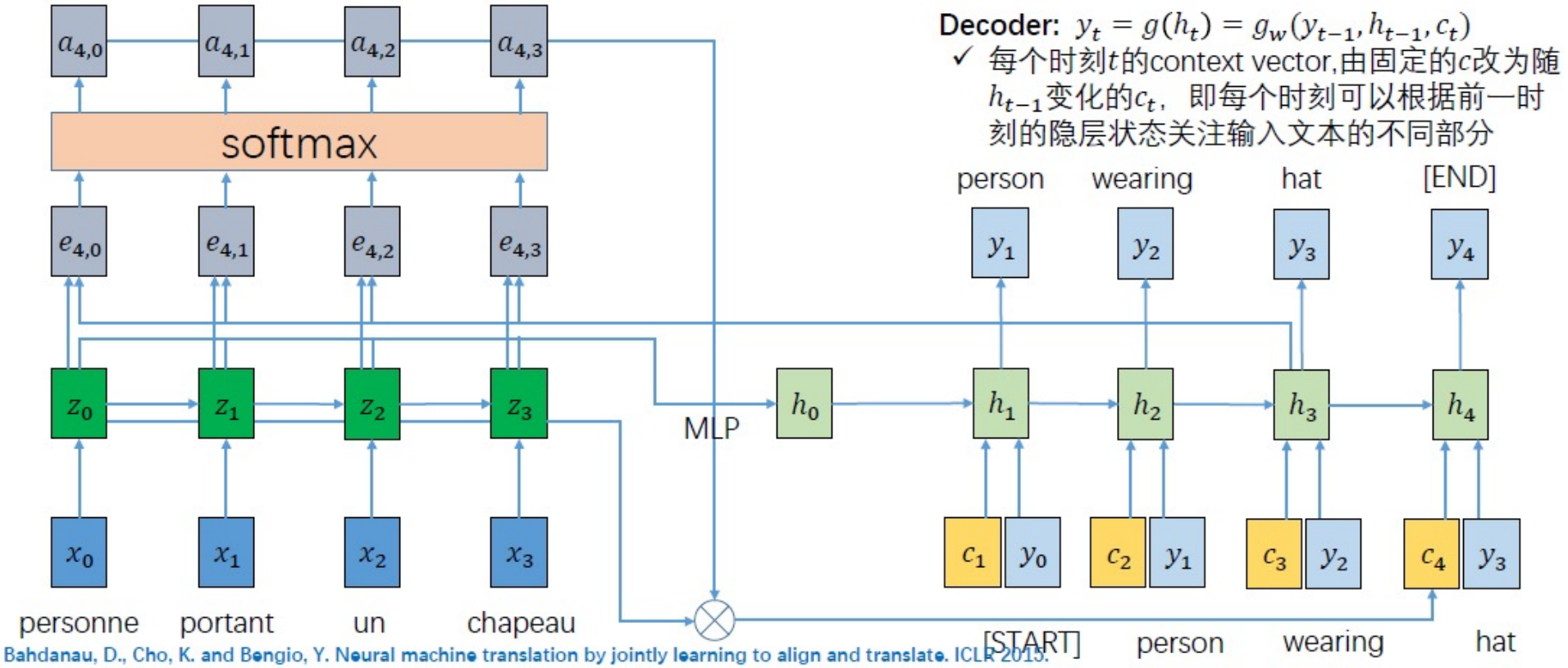
Attention Mechanism

- 根据提示，补全**融合注意力机制的解码器**代码，实现序列逆置任务
  - 利用设定好的输入完成**要求的RNN with attention模型**
  - 所有预设的网络层都应当用到
  - 不能修改给定的对象属性，不能调用其他工具包，只能在“to do”下面书写代码，**可以调试超参数**
  - 提交之后，测试集上的准确率应该提升到一个正确的范围内
  - 可多次提交。即使对自己的代码没有自信也一定要提交，我们会酌情给过程分
  - 本次Lab在截止日期之前将不公开榜单
- **TO DO**：完成《Attention Mechanism》项目。补全rnn_with_atten.py文件使exercise_reverse_sequence.py文件中的train_with_RNN()可以顺利执行。

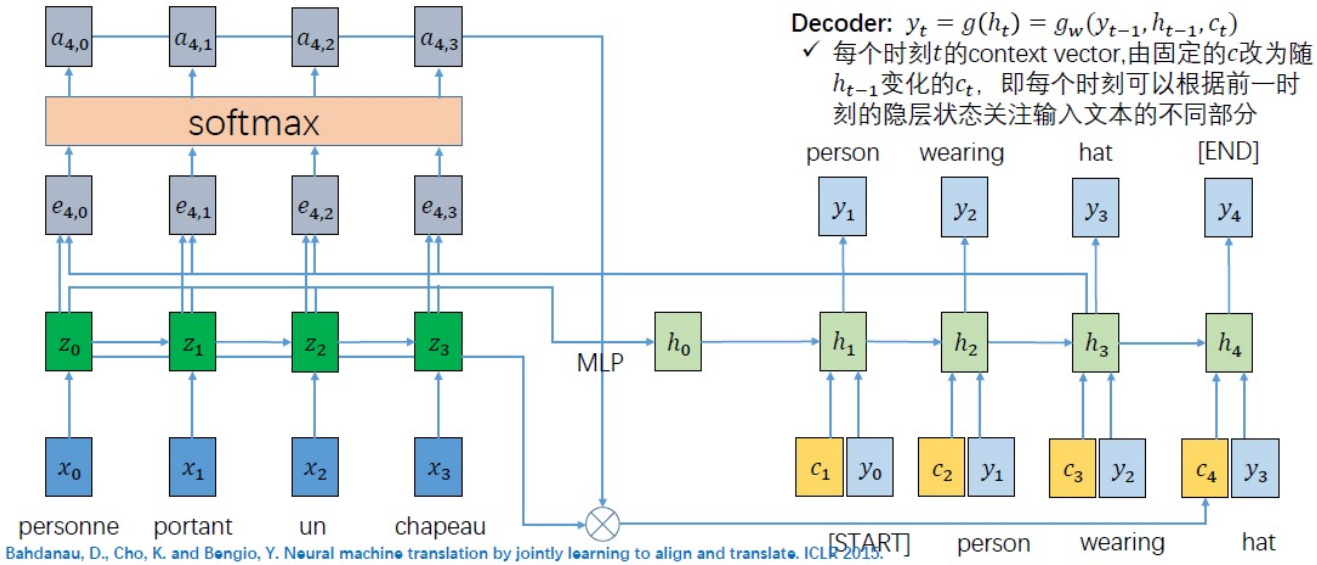
序列逆置任务

- 输入一个序列要求模型输出序列的逆置
  - 如：输入 “ABCDEFG”， 输出 “GFEDCBA”

模型结构


Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$
✓ 每个时刻t的context vector,由固定的c改为随h_{t-1}变化的c_t，即每个时刻可以根据前一时刻的隐层状态关注输入文本的不同部分
softmax
MLP
personne
portant
un
chapeau
[START]
person
wearing
hat
[END]
Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

模型结构



Input: $x = x_1, x_2, .., x_T$
Output: $y = y_1, y_2, .., y_T$

Encoder: $z_t = RNN(x_t, z_{t-1})$
z_t为编码器的隐藏状态
注意，解码器初始的隐藏状态设置为：

$$h_0 = \frac{1}{T}\sum_{i=1}^{T}\{MLP_1([z_1, z_1, \dots, z_t])\}$$

Decoder:
$h_t = RNN(h_{t-1}, [c_t \oplus y_{t-1}])$
h_t为解码器的隐藏状态

$c_t = attn(Z, h_{t-1})$
其中$Z = [\mathrm{z}_1, \mathrm{z}_2, \dots, \mathrm{z}_\mathrm{T}]$

$$\alpha_i = \frac{exp^{z_i^T h_{t-1}}}{\sum_{j=1}^{T} exp^{z_j^T h_{t-1}}}$$

$$c_t = \sum_{i=1}^{T} \alpha_i \mathrm{z_i}$$

Output： $\hat{y_t} = MLP_2(h_t)$
$\hat{y_t}$为预测的字符

Evaluation脚本

```
def compute_acc(pred_file):
    with open('data/test_X.txt') as f:
        gold = f.readlines()
    gold = [sent.strip() for sent in gold]
    gold = [''.join([o for o in reversed(e_idx)]) for e_idx in gold]

    with open(pred_file) as f:
        pred = f.readlines()
    pred = [sent.strip() for sent in pred]
    correct_case = [i for i, _ in enumerate(gold) if gold[i] == pred[i]]

    acc = len(correct_case)*1./len(gold)
    print('The predicted accuracy is %s' %acc)
```