清华大学电子工程系

媒体与认知 课堂 2

2021-2022 学年春季学期

作业 5

孙一 2022 年 5 月 9 日

理论部分

1 单选题（15 分）

1.1 B

1.2 D

1.3 B

1.4 D

1.5 C

2 计算题（15 分）

2.1 隐含马尔可夫模型的解码

某手机专卖店今年元旦新开业，每月上旬进货时，由专卖店经理决策，采用三种进货方案中的一种：高档手机 (H)，中档手机 (M)，低档手机 (L)。

当月市场行情假设分为畅销 (S_1) 和滞销 (S_2) 两种。畅销时，三种进货方案的概率分别为 0.4, 0.4, 0.2；滞销时，三种进货方案的概率分别为 0.2, 0.3, 0.5。

某月份市场行情为畅销，下一个月份为畅销和滞销的概率分别为 0.6 和 0.4；某月份市场行情为滞销，下一个月份为畅销和滞销的概率分别为 0.5 和 0.5。

开业第一个月市场行情为畅销和滞销的可能性均为 0.5。

(1) 如果我们采用隐含马尔可夫模型 (HMM) 对该专卖店进货环节建模，请写出 HMM 对应的参数 $\lambda = \{\pi, A, B\}$。

(2) 在第一季度中，采购业务员执行的进货方案为“高档手机，中档手机，低档手机”，即观测序列为 H, M, L。请利用 Viterbi 算法推测前三个月的市场行情。

解答过程见图 1

2.1

(1) $\pi = [0.5 \quad 0.5]$ S_1：畅销 S_2：滞销

$$A = \begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \qquad B = \begin{bmatrix} 0.4 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

(2) $O = [H \quad M \quad L]$

$\delta_1(1) = \pi_1 b_1(O_1) = \frac{1}{2} \times \frac{2}{5} = \frac{1}{5}$ $\quad \psi_1(1) = 0$

$\delta_1(2) = \pi_2 b_2(O_1) = \frac{1}{2} \times \frac{1}{5} = \frac{1}{10}$ $\quad \psi_1(2) = 0$

$$\delta_2(1) = \max_{1 \le i \le 2} [\delta_1(i) a_{i1}] b_1(O_2)$$

$$= \max \begin{cases} \frac{1}{5} \times \frac{3}{5} \checkmark \\ \frac{1}{10} \times \frac{1}{2} \end{cases} \times \frac{2}{5} = \frac{6}{125} \quad \psi_2(1) = 1$$

$$\delta_2(2) = \max_{1 \le i \le 2} [\delta_1(i) a_{i2}] b_2(O_2)$$

$$= \max \begin{cases} \frac{1}{5} \times \frac{2}{5} \checkmark \\ \frac{1}{10} \times \frac{1}{2} \end{cases} \times \frac{3}{10} = \frac{3}{125} \quad \psi_2(2) = 1$$

$$\delta_3(1) = \max_{1 \le i \le 2} [\delta_2(i) a_{i1}] b_1(O_3)$$

$$= \max \begin{cases} \frac{6}{125} \times \frac{3}{5} \checkmark \\ \frac{3}{125} \times \frac{1}{2} \end{cases} \times \frac{1}{5} = \frac{18}{3125} \quad \psi_3(1) = 1$$

$$\delta_3(2) = \max_{1 \le i \le 2} [\delta_2(i) a_{i2}] b_2(O_3)$$

$$= \max \begin{cases} \frac{6}{125} \times \frac{2}{5} \checkmark \\ \frac{3}{125} \times \frac{1}{2} \end{cases} \times \frac{1}{2} = \frac{6}{625} \quad \psi_3(2) = 1$$

$\frac{6}{625} > \frac{18}{3125} \longrightarrow q_3^* = 2 \longrightarrow q_2^* = \psi_3(2) = 1 \longrightarrow q_1^* = \psi_2(1) = 1$

∴ 推测前三个月的市场行情为 畅销、畅销、滞销

图 1: 2.1

2.2 循环神经网络的长时相关性建模能力

对序列中的长距离相关信息进行建模是涉及序列的任务中十分重要的一点，例如在阅读理解任务里，题目和正文中的关键词可能相距很远，这就需要模型具备足够好的长距离相关信息建模能力。传统 RNN 在训练时存在梯度消失问题，较远的误差无法得到有效传递，因此学习长距离相关信息时面临较大挑战，在本题中我们对传统 RNN 难以学习长距离相关信息的问题进行一个简单的讨论。

对 RNN 的计算过程进行简化，考虑一个暂不采用激活函数以及输入 x 的 RNN:

$$\boldsymbol{h}_t = U\boldsymbol{h}_{t-1} = U\left(U\boldsymbol{h}_{t-2}\right) = ... = U^t\boldsymbol{h}_0$$

其中 U^t 为 t 个 U 矩阵连乘。若矩阵 U 存在如下特征值分解：

$$U = Q\Lambda Q^\top$$

其中 Q 为单位正交矩阵（每一列为模长为 1 的特征向量），$Q^\top$ 为 Q 的转置，Λ 为特征值对角矩阵，则上述的 RNN 计算过程可表示为：

$$\boldsymbol{h}_t = Q\Lambda^t Q^\top \boldsymbol{h}_0$$

本题目包含以下三个问题：

(1) 假设某一特征值 $\lambda_i < 1$，当时刻 t 增大时，Λ^t 中第 i 行 i 列的值会怎样变化?

(2) 假设 $\boldsymbol{h}_0 = \boldsymbol{q}_i$，其中 $\boldsymbol{q}_i$ 为 U 矩阵的第 i 个特征向量（即 Q 的第 i 列），设 $\mathcal{L}$ 为目标函数计算出的 loss。试验证：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_0} = \lambda_i^t \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t}$$

(3) 对于更一般的 $\boldsymbol{h}_0$，由于 Q 中的特征向量构成一组完备正交基，可以将 $\boldsymbol{h}_0$ 分解为 Q 中不同特征向量的线性组合，即 $\boldsymbol{h}_0 = \sum_{i=1}^n k_i\boldsymbol{q}_i$。通过上述分析，请尝试解释传统 RNN 训练中的梯度消失现象，由此理解传统 RNN 对长距离相关信息建模的困难。

解答过程见图 2

2.2 $h_t = Q\Lambda^t Q^T h_0$

(1) Λ 是对角阵，故 Λ^t 中元素 $\Lambda_{ii} = (\lambda_i)^t$

若 $\lambda_i < 1$ 则 Λ_{ii} 会以指数形式趋于 0 ——→梯度逐渐消失

(2) $h_0 = g_i$

$$h_t = (g_1 \cdots g_n) \begin{bmatrix} \lambda_1^t & & & \\ & \lambda_2^t & & \\ & & \ddots & \\ & & & \lambda_n^t \end{bmatrix} \cdot \begin{bmatrix} g_1^T \\ \vdots \\ g_n^T \end{bmatrix} g_i$$

$$= \begin{bmatrix} \lambda_1^t g_1 & \lambda_2^t g_2 & \cdots & \lambda_n^t g_n \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \text{只有第 } i \text{ 行为 1}$$

$$= \lambda_i^t g_i$$

$$\therefore \frac{\partial L}{\partial h_0} = \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial h_0} = \lambda_i^t \frac{\partial L}{\partial h_t}$$

(3) $h_0 = \sum_{i=1}^n k_i g_i$ 由(2)中的分析可知

$$h_t = \sum_{i=1}^n k_i \lambda_i^t g_i$$

$$\frac{\partial L}{\partial g_j} = \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial g_j} = k_i \lambda_i^t \frac{\partial L}{\partial h_t}$$

若 λ_i 绝对值小于 1，则 $k_i \lambda_i^t$ 随 t 增大趋于 0

考虑到 $\frac{\partial L}{\partial h_t}$ 有界，$\frac{\partial L}{\partial g_j}$ 也随 t 增大趋于 0 } 指数级减小

∴ 当输入序列在时间上跨度较长时，新产生的 L 对 g_i 的梯度将趋于 0，导致无法进行长距离相关信息建模.

图 2: 2.2

3 编程作业报告

3.1 完成基于循环神经网络的场景文本识别程序代码

3.1.1 完成"CRNN" 模型的初始化

```
# step 1: define <self.cnn> for vision feature extraction
# -- hint: you may use the following classes:
#          nn.Conv2d(), nn.BatchNorm2d(), nn.ReLU(), nn.MaxPool2d(), and other layers
#          and you can use nn.Sequential() to stack all layers
# -- here we give an optional CNN configuration
# ------------------------------------------------------------------------------
# block |           convolution          | batch norm | activation |     max pool
#   0   | #k=16, ksize=3x3, s=2x2, p=1x1 |    yes     |    ReLU    | ksize=2x2, s=2x2
#   1   | #k=32, ksize=3x3, s=1x1, p=1x1 |    yes     |    ReLU    | ksize=2x1, s=2x1
#   2   | #k=48, ksize=3x3, s=1x1, p=1x1 |    yes     |    ReLU    | ksize=2x1, s=2x1
#   3   | #k=64, ksize=3x3, s=1x1, p=1x1 |    yes     |    ReLU    | ksize=2x1, s=2x1
#   4   | #k=64, ksize=1x1, s=1x1, p=0   |     no     |    None    |       None
# ------------------------------------------------------------------------------
self.cnn = nn.Sequential(
    nn.Conv2d(3, 16, 3, 2, 1),  # input channels are rgb
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(16,32,3,1,1),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d((2,1)),
    nn.Conv2d(32,48,3,1,1),
    nn.BatchNorm2d(48),
    nn.ReLU(),
    nn.MaxPool2d((2,1)),
    nn.Conv2d(48,64,3,1,1),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d((2,1)),
    nn.Conv2d(64,64,1)
)
```

```
# step 2: define self.rnn for sequence modeling
# -- hint 1: you may use nn.LSTM(), but it is also OK to use nn.RNN() or nn.GRU()
#            please refer to the following document to look up the usage of nn.LSTM()
#https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html?highlight=lstm#torch.nn
# -- hint 2: the input_size of RNN should be equal to the output channel numbers of C
#            which is 64 if you use the optional CNN
# -- here we give an optional RNN configuration
#    input_size = 64, hidden_size = 32, num_layers = 1, bidirectional = True
self.rnn = nn.LSTM(input_size = 64, hidden_size = 32, num_layers = 1,
                   bidirectional = True)
# step 3: define self.linear for classification
# -- hint 1: input dimension of self.linear should be output dimension of RNN,
#            if you set bidirectional=True in RNN, the output dimension
#            will be 2 times the hidden_size of RNN.
#            注意 LSTM 双向，输出的形状是 hidden_size * 2 = 64!!!
# -- hint 2: output dimension of self.linear should be number of classes,
#         in this task we have 38 classes (26 letters, 10 digits, 'blank' and '<unk>'
self.linear = nn.Linear(64,38)
# ========================================
# TODO 1: complete network initialization
# ========================================
```

这里需要注意的有两点：

1. MaxPool2d 和 Conv2d 函数的 kernel size, stride, padding, dilation 参数在设置时，如果只提供一个 int 数据，那么默认高度方向和宽度方向的值是相同的；如果提供了 tuple，那么高度和宽度分别取第一和第二个位置的值。可见 pytorch 官网对此的说明 3, 4

2. MaxPool2d 的 stride 默认和 kernel_size 的大小一致 5

3.1.2 完成模型的前向计算过程

一些变量的定义：

变量名	值
b(batch_size)	32
c(cnn_output_channels)	64
h(cnn_output_height)	32/32=1
w(cnn_output_width)	128/4=32

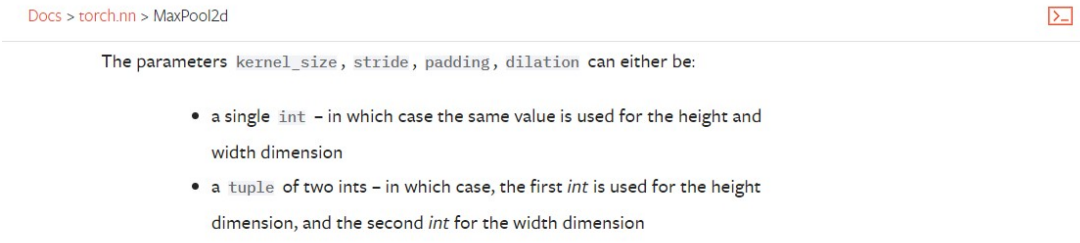


图 3: MaxPool2d 参数设置

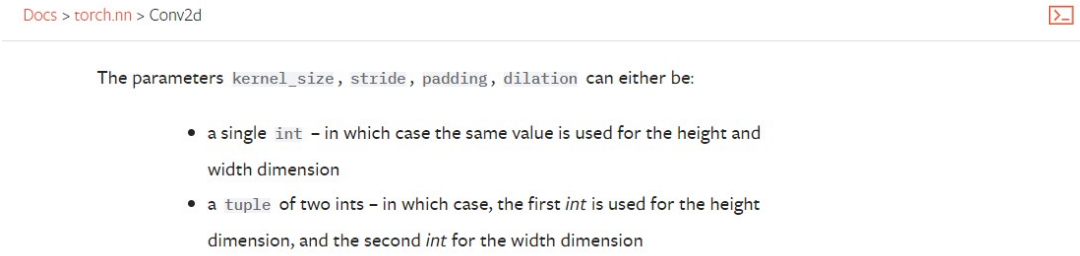


图 4: Conv2d 参数设置

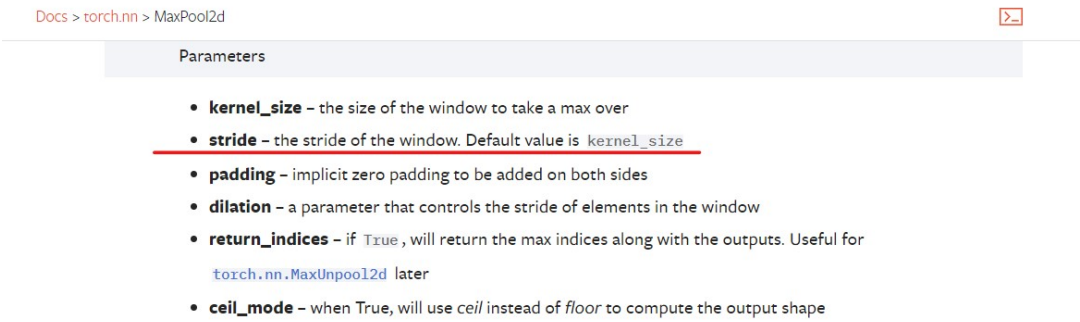


图 5: MaxPool2d stride 值

按照代码中所给的默认参数设计卷积网络，可使得最后一层输出数据的高度 (height) 恰好为 1（32/32=1），而宽度 (width) 变为了 (128/4=32)。同时，输入 LSTM 前需要把数据格式调整为 [w,b,c]，其中 w 相当于“时间总长度”或者 sequence length，c 相当于每一步输入数据的大小，b 还是 batch_size。由于 LSTM 设置成了双向，所以每一条输出数据的长度为 hidden_size*2 = 32*2=64。

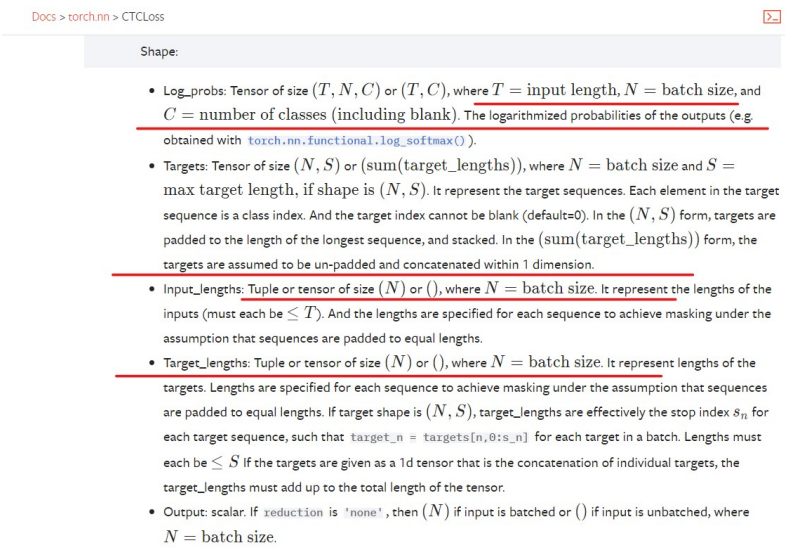
```
'''
:param x: input images with size [b, 3, H, W], b is batch size,
            3 refers to the dimension of RGB and H(32) and W(128)
            are height and width of the image
:return logits: logits with size [w, b, 38], w = 32 is the width
            of feature maps extracted by CNN
:return seq_lengths: torch.LongTensor with size [b], equals to [w, w, ..., w]
'''
# step 1: apply self.cnn on input "x" and get feature maps "feats"
feats = self.cnn(x) # [b, c, h, w]
# step 2: compute the length of feature sequence "seq_lengths" for CTC loss
# -- hint: seq_lengths is [w, w, ..., w] (b times)
#          you can use feats.size(0) and feats.size(3) to get "b" and "w"
seq_lengths = torch.LongTensor(feats.size(0) * [feats.size(3)])
# step 3: transform feature maps into RNN input
# -- hint: the size of feature maps is [batch_size, channels, h, w]
#          (h=1 if you use the optional CNN),
#          but the input size of RNN should be [w, b, c].
#          you may use functions such as tensor.squeeze(dim) and tensor.permute(*dims)
#          to change the shape of CNN output.
feats = feats.squeeze(2).permute(2, 0, 1)
# step 4: apply self.rnn on feature sequences
# -- hint: the outputs of RNN is "output, h_n" or "output, (h_n, c_n)"
#          we only need "output" for the subsequent recognition
output, _= self.rnn(feats)  # [w b c] --> [w b 32*2]
# step 5: apply self.linear on RNN output to obtain "logits"
logits = self.linear(output)    # [w b 32*2] --> [w, b, 38]
# ========================================
# TODO 2: complete network forward process
# ========================================
```

3.1.3 完成模型每轮训练过程代码

训练过程

```
# you may follow the below steps
# 1. set model into training mode
model.train()
# 2. initialize a "total_loss" variable to sum up losses from each training step
total_loss = 0.
# 3. start to loop, fetch images and labels of each step from "trainloader"
for ims, texts in trainloader:
# 4.   convert label texts into tensors by "label_converter"
    targets, target_lengths = label_converter.encode(texts)
    ims = ims.to(device)
    targets = targets.to(device)
    target_lengths = target_lengths.to(device)
# 5.   run the model forward process
    logits, seq_lengths = model(ims)
# 6.   compute loss by "criterion"
    loss = criterion(logits.log_softmax(2), targets, seq_lengths, target_lengths)
# 7.   run the backward process and update model parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
# 8.   update "total_loss"
    total_loss += loss.item()
# 9. return avg_loss, which is total_loss / len(trainloader)
avg_loss = total_loss / len(trainloader)
```

nn.CTCLOss 的输入参数定义见图 6。其中，第一个参数是修改后的 CRNN 模型输出值 [w,b,38]，需要把原来输出值的最后一维类别值改为类别概率（通过 softmax）；第二个参数是 label_converter 把样本标签转换得到的字母类别列表，总长度是 batch_size * target_lengths(也就是 batch_size)，对应图 6中的 (sum(target_lengths)) 模式；第三个参数是每个样本的输入时长（都是 w=32），共有 b 个；第四个参数是 target_lengths，自然是每个样本标签包含的字符数，总共有 b 个。


Docs > torch.nn > CTCLoss

Shape:

- Log_probs: Tensor of size (T, N, C) or (T, C), where T = input length, N = batch size, and C = number of classes (including blank). The logarithmized probabilities of the outputs (e.g. obtained with `torch.nn.functional.log_softmax()`).
- Targets: Tensor of size (N, S) or $(\text{sum}(\text{target\_lengths}))$, where N = batch size and S = max target length, if shape is (N, S). It represent the target sequences. Each element in the target sequence is a class index. And the target index cannot be blank (default=0). In the (N, S) form, targets are padded to the length of the longest sequence, and stacked. In the $(\text{sum}(\text{target\_lengths}))$ form, the targets are assumed to be un-padded and concatenated within 1 dimension.
- Input_lengths: Tuple or tensor of size (N) or $()$, where N = batch size. It represent the lengths of the inputs (must each be $\leq T$). And the lengths are specified for each sequence to achieve masking under the assumption that sequences are padded to equal lengths.
- Target_lengths: Tuple or tensor of size (N) or $()$, where N = batch size. It represent lengths of the targets. Lengths are specified for each sequence to achieve masking under the assumption that sequences are padded to equal lengths. If target shape is (N, S), target_lengths are effectively the stop index s_n for each target sequence, such that `target_n = targets[n,0:s_n]` for each target in a batch. Lengths must each be $\leq S$ If the targets are given as a 1d tensor that is the concatenation of individual targets, the target_lengths must add up to the total length of the tensor.
- Output: scalar. If `reduction` is `'none'`, then (N) if input is batched or $()$ if input is unbatched, where N = batch size.

图 6: CTCLoss 参数定义

验证过程

```
# hint: you may follow the below steps
# 1. set model into evaluation mode
model.eval()
# 2. initialize "n_correct" and "n_total" variables to save the
# numbers of correct and total images
n_correct = 0.
n_total = 0.
# 3. loop under the no-gradient environment, fetch images
# and labels of each step from "valloader"
with torch.no_grad():
    for ims, texts in valloader:
# 4.    run model forward process and compute "logits"
        logits, _ = model(ims)
# 5.    get raw predictions "raw_preds" by "logits.argmax(2)"
        raw_preds = logits.argmax(2)
# 6.    use "label_converter.decode(raw_preds)" to obtain decoded texts
        preds = label_converter.decode(raw_preds)
# 7.    update "n_total" and update "n_correct" by comparing decoded texts with labels
        n_total += len(texts)
        n_correct += sum([pred == text for pred, text in zip(preds, texts)])
```

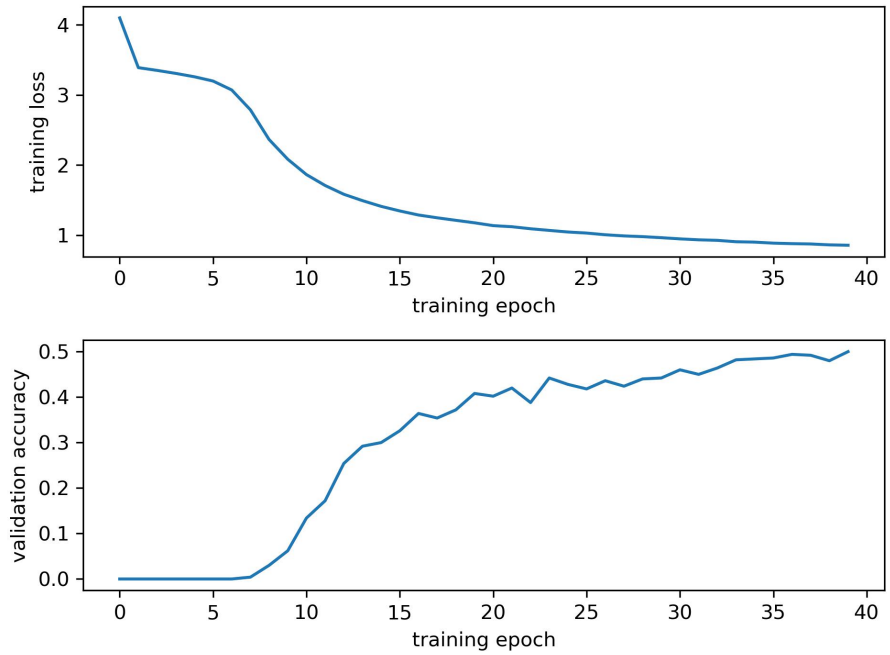


图 7: loss_and_accuracy

```
# 8. return accuracy, which is n_correct / n_total
accuracy = n_correct/n_total
# ================================
# TODO 4: complete val_one_epoch()
# ================================
```

3.2 训练/预测/可视化

3.2.1 模型的训练和验证

模型在训练集上的 loss 和验证集上的单词识别正确率（即完全识别正确的图像样本所占总样本的比例）变化情况如图 7。验证集上的最终正确率为 50%:

```
Epoch [40/40] start ...
train loss = 0.856, validation word accuracy = 50.0%
```

由于作业侧重于理解原理，数据量较小，在使用默认网络参数的条件下，训练 40 轮后在验证集上的单词识别正确率约为 50%，而且正确率在 30 轮以后正确率变化已经很小，说明此时模型在训练集上已经取得了比较稳定

的结果。可以通过增加训练样本或者适当提高模型的复杂度来获得更高的验证准确率。

3.2.2 使用训练好的模型预测新的文本图像

默认图片识别结果：

```
prediction: parking
CTC visualization has been saved as data/my_own/a_vis.jpg
```

可视化结果如图 8。从图中可以看出：

1. blank 符号占据识别结果的很大一部分。
2. 在字母开始发生“突变”的时刻会产生一段连续的识别结果，并且置信度较高。

自选图片 1 识别结果：

```
prediction: marvel
CTC visualization has been saved as data/my_own/marvel_vis.jpg
```

可视化结果如图 9。与默认图片情形相似。

自选图片 2 识别结果：

```
prediction: csmoeowe
CTC visualization has been saved as data/my_own/casino_royal_vis.jpg
```

可视化结果如图 10。此时由于字母间距较小，识别错误大大增加。

自选图片 3 识别结果：

```
prediction: mrastumis
CTC visualization has been saved as data/my_own/marvelstudios_vis.jpg
```

可视化结果如图 11。与图 9相比，此图多了一个单词”STUDIOS”，”MARVEL” 在图片中的占比明显下降，而且”STUDIOS” 上下还有两条横线作为干扰，因此识别准确率有所下降。

此外，还可以观察到一些有趣的现象：字母”e” 和”s” 一般不会识别错误；结构相对简单的字母，比如”o” 也不容易出错。前者可能是因为英文单词中字母”e” 和”s” 出现的概率非常高，等效于增大了”e” 和”s” 的训练样本数；考虑到本实验采用的训练集较小，模型也比较简单，结构简单字母的特征更容易被模型学习得出，因此识别率也相对更高。

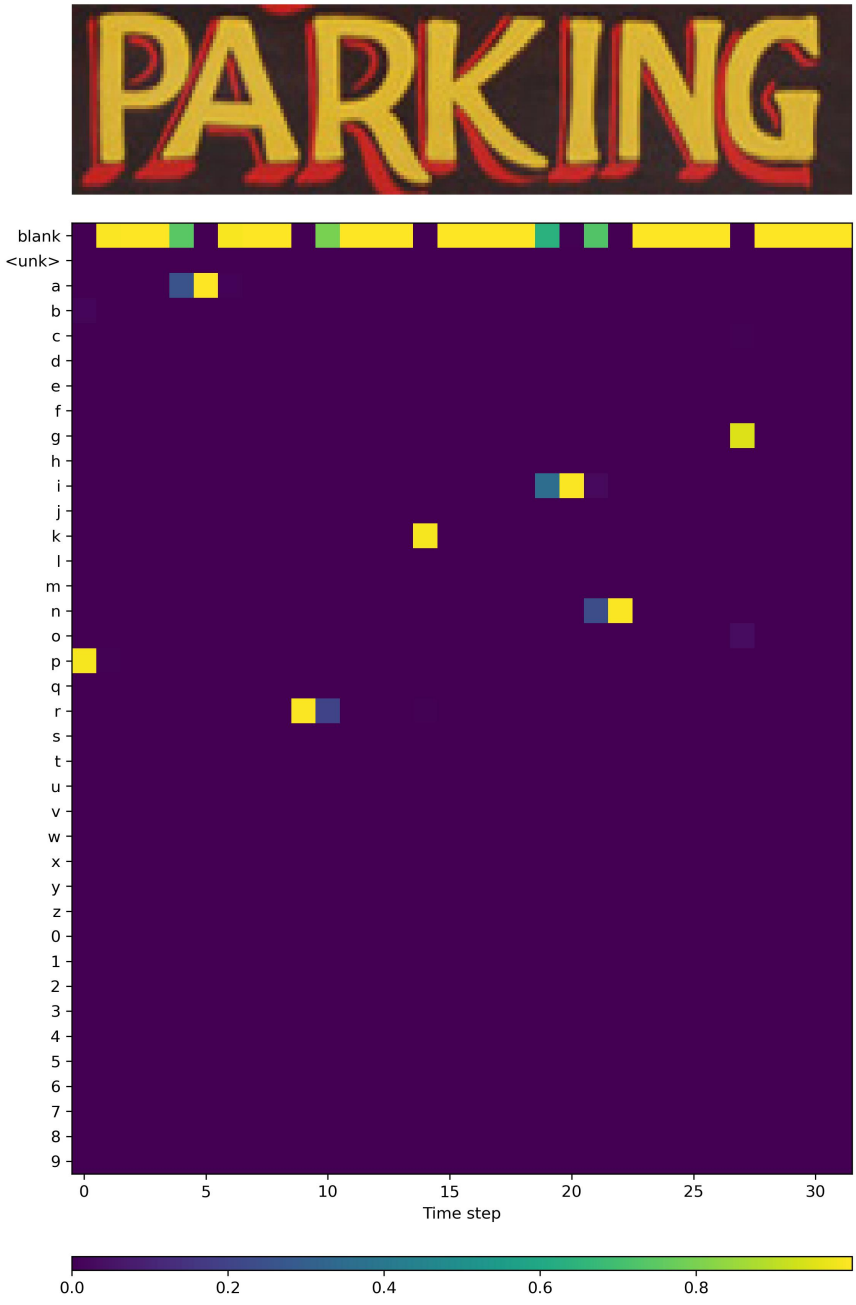

PARKING
blank
<unk>
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
0
1
2
3
4
5
6
7
8
9
0
5
10
15
20
25
30
Time step
0.0
0.2
0.4
0.6
0.8

图 8: a_vis

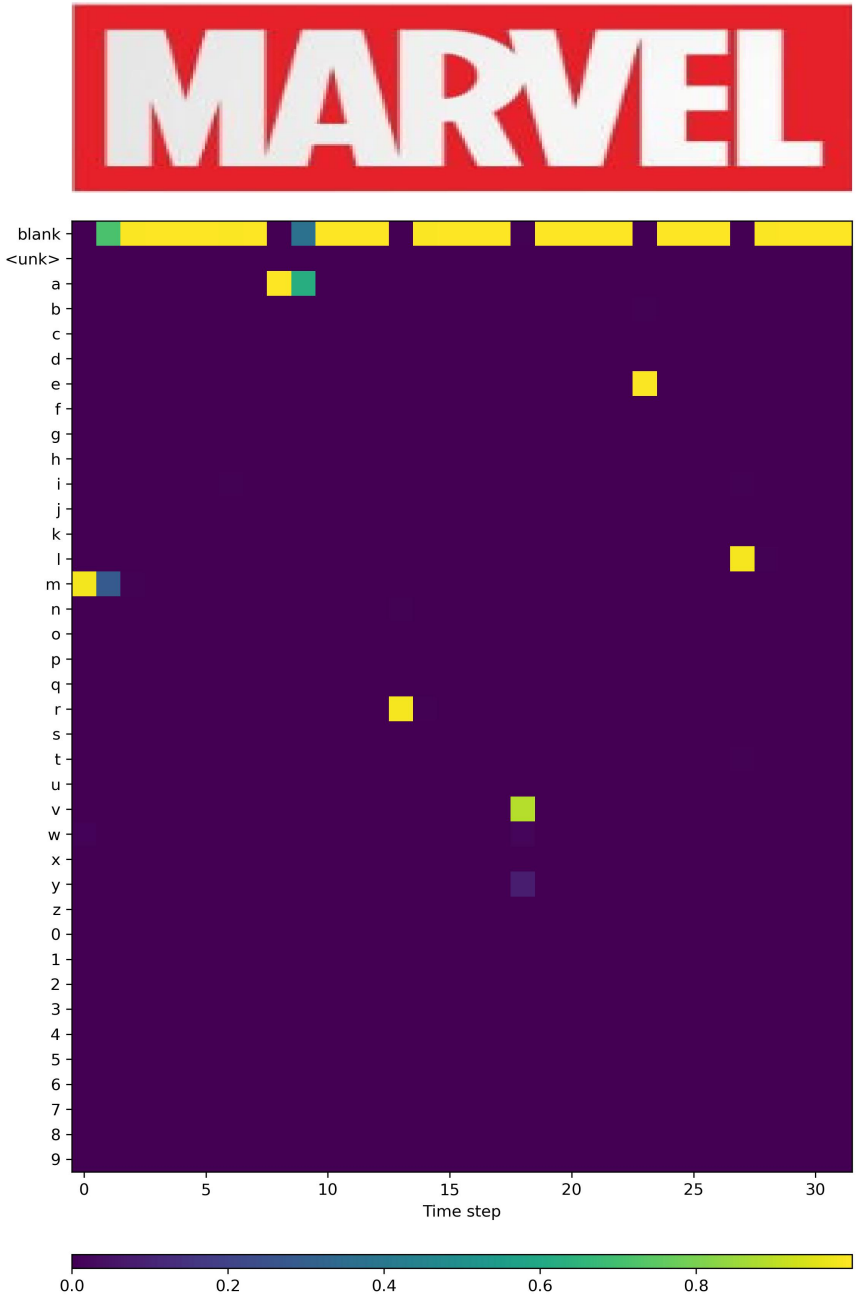


图 9: marvel_vis

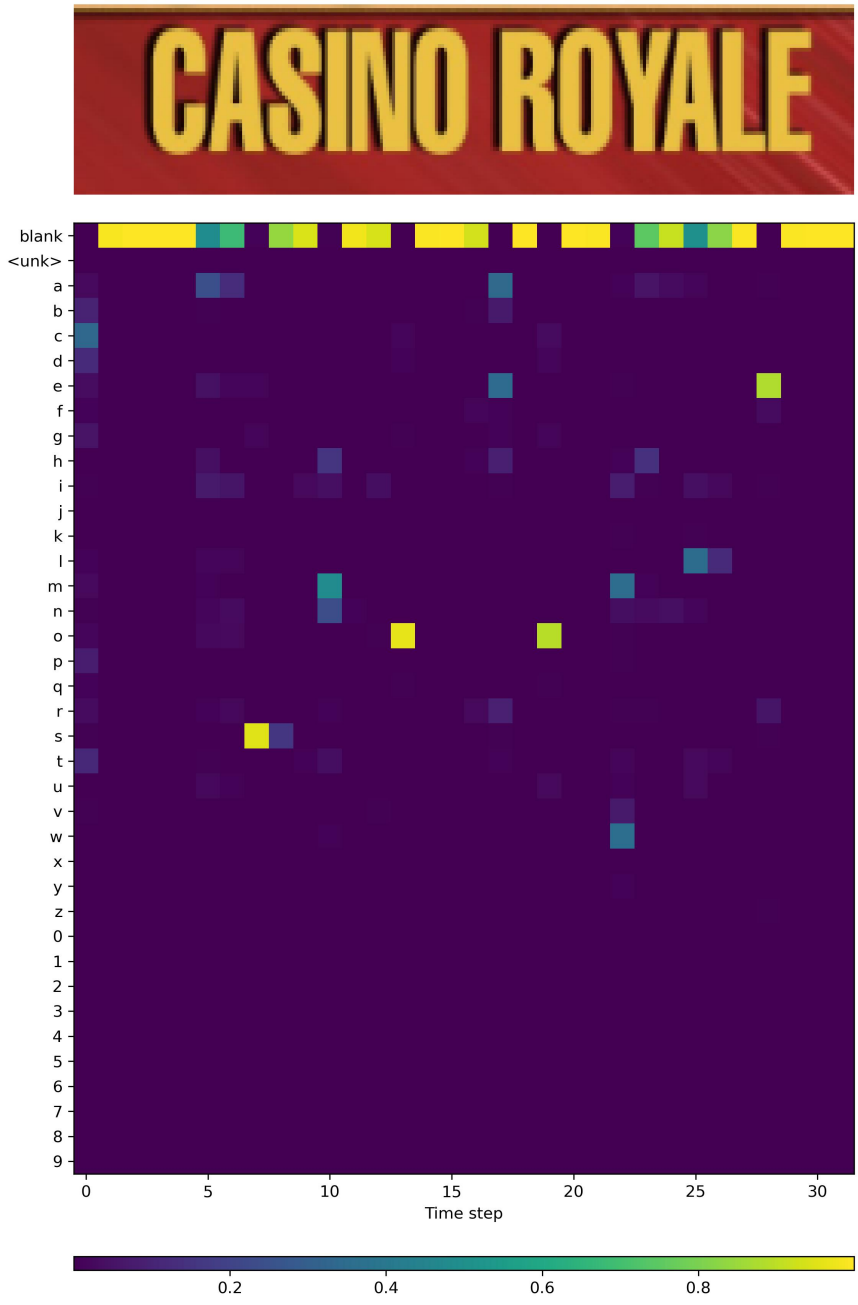


图 10: casino_royal_vis

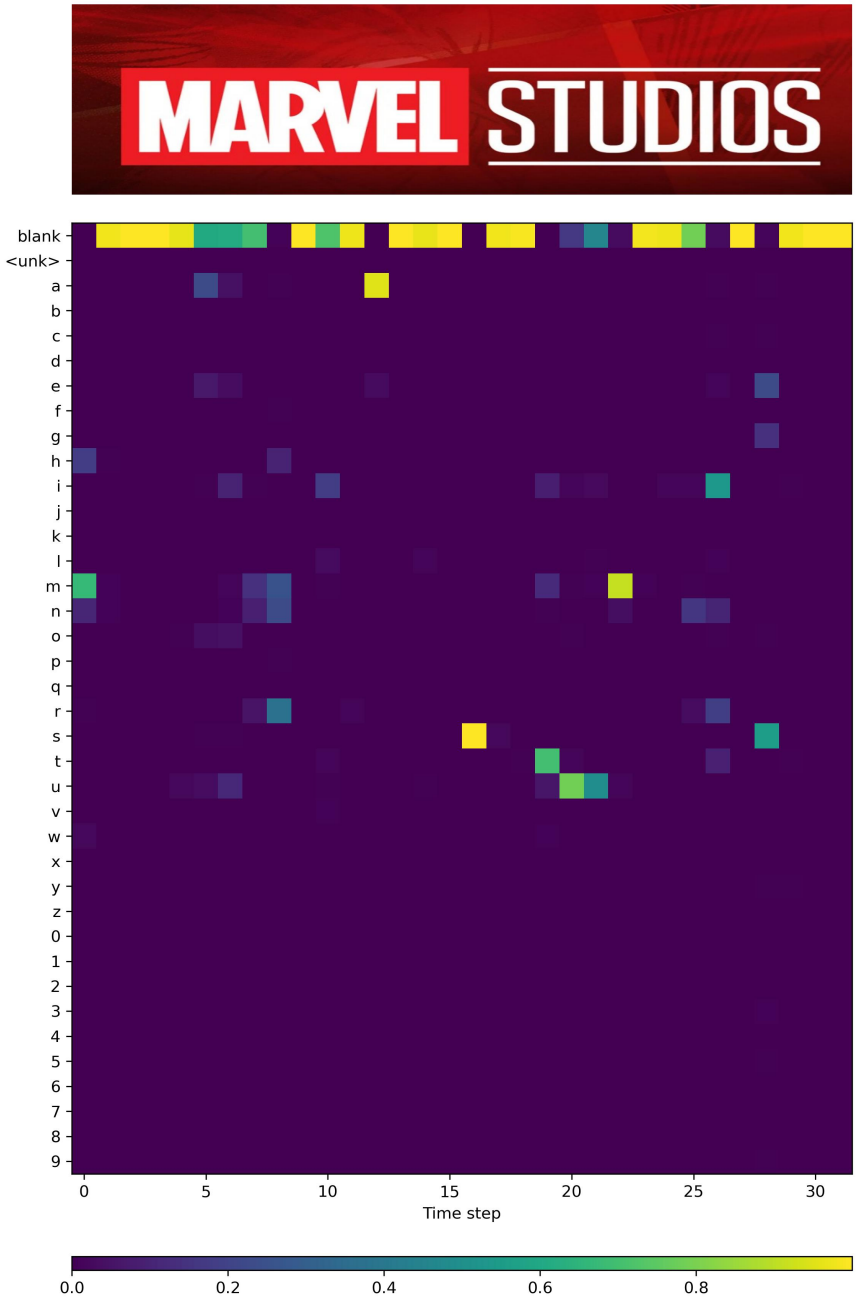


图 11: marvel_studios_vis