南開大學
Nankai University

深 度 学 习 实 验 报 告

实验名称：循环网络

姓　　名：张恒硕

学　　号：2212266

专　　业：智能科学与技术

目录

一、实验目的

实现一个 LSTM（Long Short-Term Memory，长短期记忆）RNN（Recurrent Neural Network，循环神经网络），并用 The Time Machine 数据集进行测试。另尝试改进模型或调整参数来改善效果。

二、实验原理

1. RNN

作为一类用于处理序列数据的神经网络，RNN 改变了 FCNN 等模型对于同一单词不同实例的理解语境问题，通过上下文信息，获取了区分一词多义的能力。同时，这种前后文处理能力赋予了模型真正“理解”语句的能力。

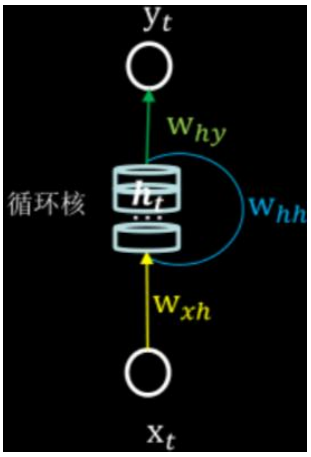

图 2-1 RNN 基本单元

上图图 2-1 给出了 RNN 的基本功能单元——循环核。其计算公式如下：

$$h_t = f_w(h_{t-1}, x_t) = tanh(w_{xh}x_t + w_{hh}h_{t-1} + b_h)$$

$$y_t = f_{why}(h_t) = softmax(w_{hy}h_t + b_y)$$

其中h_t是状态信息，x_t是输入特征，y_t是输出特征。在前向传播时，更新h_t，不改变参数矩阵w_{xh}、w_{hy}、w_{hh}；在反向传播时，更新参数矩阵。

多个循环核连接在一起，便构成了 RNN。不同的输入输出模式对应不同功能的 RNN。siso 是 Vanilla 神经网络，simo 是图像描述的词语序列模型，miso 是动作预测的视频帧序列模型，mimo 则是视频描述或分类的视频帧序列模型。

传统 RNN 在处理长时间序列数据时会遇到梯度消失或梯度爆炸问题。改进得到下面的 LSTM 模型。

2. LSTM

LSTM 引入了细胞状态结构和三个门控机制来控制信息的流动，这使得其能更有效地学习长期依赖关系，并在处理和预测时间序列数据方面表现出色。以下图 2-2 是细胞状态结构。

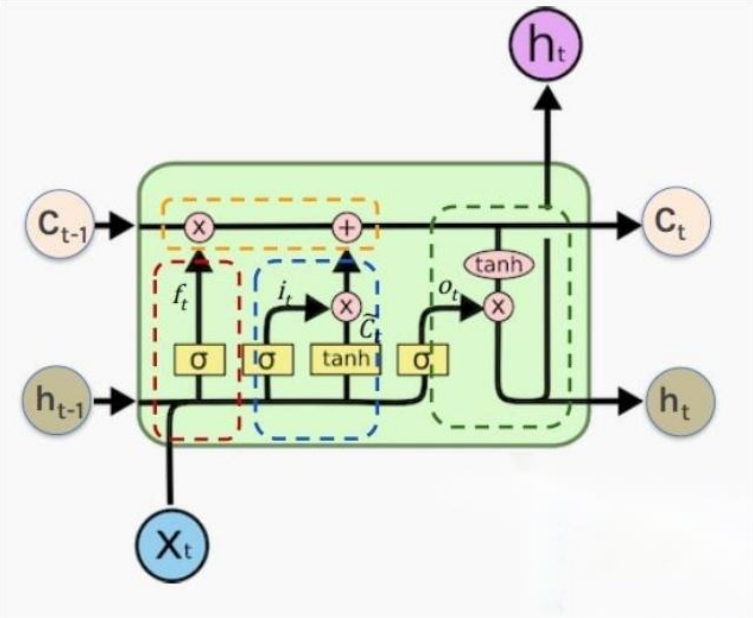

图 2-2 细胞状态结构

1) 细胞状态（Cell State）

贯穿模型，在整个序列上进行少量的线性交互，使信息流动而不发生太多改变。其中黄色框部分对应细胞状态的更新，其根据遗忘门和输入门的结果进行。

2) 遗忘门（Forget Gate）

对应上图红色框部分，决定从细胞状态中丢弃哪些信息。其查看前一时刻的隐藏状态h_{t-1}和当前输入x_t，输出介于 0 和 1 之间的值给细胞状态c_{t-1}的每个元素。0 意味着完全丢弃，1 意味着完全保留。

3) 输入门（Input Gate）

对应上图蓝色框部分，包含一个用于确定要更新哪些部分的 sigmoid 层，和一个创建新候选值向量的 tanh 层。

4) 输出门（Output Gate）

对应上图绿色框部分，决定细胞状态的输出。sigmoid 层决定细胞状态的哪些部分将输出，tanh 层缩放细胞状态，与 sigmoid 门的输出相乘，得到本时刻输出。

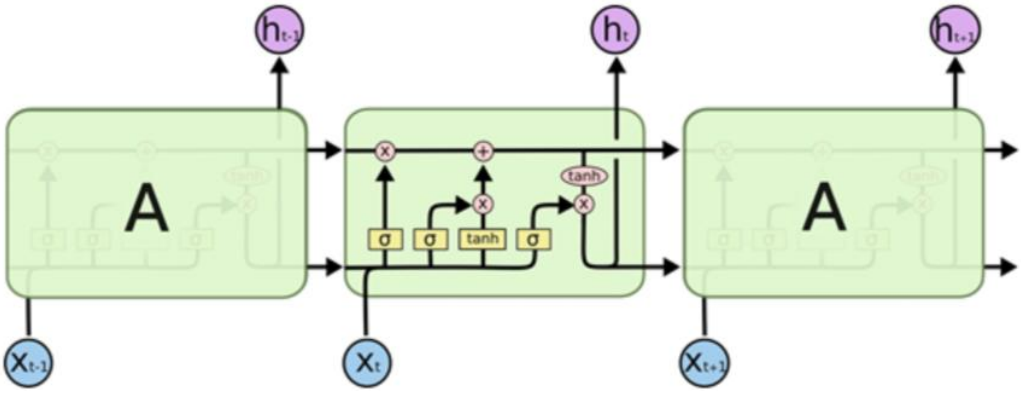

图 2-3 LSTM 模型结构

将细胞状态排布起来，便得到了 LSTM 模型，如上图图 2-3。相比于 RNN，其大幅改善了梯度消失和梯度爆炸问题。

- 梯度消失：细胞状态和门控机制对信息流的精细控制，使 LSTM 能够有效减

少梯度在反向传播过程中的衰减。

- 梯度爆炸：门控机制由 sigmoid 函数产生的数值控制，将权重更新限制在合理范围内，减轻了梯度爆炸风险。

3. 困惑度（Perplexity）

作为衡量语言模型性能的常用指标，反映了模型预测文本序列的不确定性或复杂性。其计算公式如下：

$$\text{Perplexity} = P(w_1, w_2, \ldots\ldots, w_3)^{-\frac{1}{n}} = \left(\prod_{i=1}^{n} \frac{1}{P(w_i|w_{<i})}\right)^{-\frac{1}{n}}$$

对数形式则为如下：

$$\text{Perplexity} = \exp\left(-\frac{1}{n}\sum_{i=1}^{n} \log P(w_i|w_{<i})\right)$$

其中，n 是序列中词的数量，$P(w_i|w_{<i})$表示在给定前面所有词的情况下，模型为第 i 个词w_i分配的概率。

更低的困惑度表明模型更准确地预测了文本序列中的词语，即模型具有更好的泛化能力。

三、实验步骤

1. 数据集导入与划分

从 The Time Machine 数据集导入序列长度的词汇表，并按批量大小进行划分。

2. 长短期记忆循环神经网络搭建

设定 LSTM 模型，初始化参数和状态，生成 RNN 实例。

3. 模型训练

在训练中，使用交叉熵损失函数和 SGD 优化器，进行前向传播和后向传播。在传播过程中，要确保状态间没有依赖关系。在后向传播时要进行梯度裁剪，以防止爆炸。

4. 模型预测

预测时要获取输入的张量表示，在更新状态中记录字符索引，并生成指定长度的新字符。

5. 模型调试

调整批量大小、序列长度、学习率、隐藏层大小等参数并变更优化器和权重初始化方法等进行调试。

四、教材代码（LSTM_easy.py）

由于自己实现的完整代码运行较慢，这里补充教材上的代码，后续的多数调试过程通过更改该代码实现。

```
import torch

from torch import nn

from d2l import torch as d2l

import time
```

```
# 数据与参数

batch_size = 32          # 批量大小

num_steps = 35           # 序列长度

train_iter, vocab = d2l.load_data_time_machine(batch_size, num_steps) # 数据集

num_epochs = 500         # 训练轮数

lr = 1                   # 学习率

vocab_size = len(vocab) # 词汇表大小

hiddens = 256            # 隐藏层大小

device = d2l.try_gpu()  # 训练设备

num_inputs = vocab_size # 输入大小

# 模型

lstm = nn.LSTM(num_inputs, hiddens)          # lstm 层

model = d2l.RNNModel(lstm, len(vocab)).to(device) # 模型实例

# 训练

start_time = time.time()

d2l.train_ch8(model, train_iter, vocab, lr, num_epochs, device)

end_time = time.time()

print("运行时间：%.5f秒" % (end_time - start_time))

# 预测

model.eval()

num_predicts = 100       # 预测字符数
```

```
start_time = time.time()

phrases = [

    "time traveller",

    "traveller",

    "the time traveller says that",

    "when the time traveller returns to the garden",

    "the time traveller begins learning the language",

    "the time traveller determines that",

    "the time traveller knows he will have to stop",

    "when he wakes up",

    "the time traveller finds himself",

    "the time traveller tells the narrator to wait for him"

]

for phrase in phrases:

    print(d2l.predict_ch8(phrase, num_predicts, model, vocab, device))

end_time = time.time()

print("运行时间：%.5f秒" % (end_time - start_time))
```

五、完整代码（LSTM.py）

本代码完整地给出了模型搭建、训练、预测三个部分的主体代码，但在一些细节部分仍选择调用 d2l 库中的函数来简便实现。代码中给出了详细注释，这里不作额外阐述。另外，后续调试中，针对模型部分的变更在本代码基础上实现。

```
import math

import torch

from torch import nn
```

```
from torch.nn import functional as F

from d2l import torch as d2l

import time



# 获取模型参数

def get_params(vocab_size, hiddens, device):

    num_inputs = num_outputs = vocab_size # 输入和输出与词汇表同维

    def three():

        return (torch.randn(size=(num_inputs, hiddens), device=device) * 0.01,

                torch.randn(size=(hiddens, hiddens), device=device) * 0.01,

                torch.zeros(hiddens, device=device))

    W_xi, W_hi, b_i = three() # 输入门

    W_xf, W_hf, b_f = three() # 遗忘门

    W_xo, W_ho, b_o = three() # 输出门

    W_xc, W_hc, b_c = three() # 候选记忆元

    # 输出层

    W_hq = torch.randn(size=(hiddens, num_outputs), device=device) * 0.01

    b_q = torch.zeros(num_outputs, device=device)

    # 添加梯度

    params = [W_xi, W_hi, b_i, W_xf, W_hf, b_f, W_xo, W_ho, b_o, W_xc, W_hc, b_c, W_hq,

b_q]

    for param in params:
```

```
        param.requires_grad_(True)
    return params


# 初始化状态：为一个 batch 的数据初始化隐藏状态和细胞状态
def init_state(batch_size, hiddens, device):
    return torch.zeros((batch_size, hiddens), device=device), torch.zeros((batch_size,
hiddens), device=device)


# LSTM 前向传播
def lstm(inputs, state, params):
    [W_xi, W_hi, b_i, W_xf, W_hf, b_f, W_xo, W_ho, b_o, W_xc, W_hc, b_c, W_hq, b_q] =
params
    (H, C) = state # 隐藏状态和细胞状态
    outputs = []
    for X in inputs:
        I = torch.sigmoid((X @ W_xi) + (H @ W_hi) + b_i)     # 输入门
        F = torch.sigmoid((X @ W_xf) + (H @ W_hf) + b_f)     # 遗忘门
        O = torch.sigmoid((X @ W_xo) + (H @ W_ho) + b_o)     # 输出门
        C_tilda = torch.tanh((X @ W_xc) + (H @ W_hc) + b_c) # 候选记忆元
        C = F * C + I * C_tilda # 更新细胞状态
        H = O * torch.tanh(C)    # 更新隐藏状态
        Y = (H @ W_hq) + b_q     # 输出
```

```
        outputs.append(Y)
    return torch.cat(outputs, dim=0), (H, C)


# RNN 类
class RNN:
    def __init__(self, vocab_size, hiddens, device, params, init_state, forward_fn):
        self.vocab_size = vocab_size
        self.hiddens = hiddens
        self.params = params(vocab_size, hiddens, device)
        self.init_state = init_state # 初始化状态函数
        self.forward_fn = forward_fn # 前向传播函数
    def __call__(self, X, state):
        X = F.one_hot(X.T, self.vocab_size).type(torch.float32) # 输入独热编码
        return self.forward_fn(X, state, self.params) # 前向传播
    def begin_state(self, batch_size, device):
        return self.init_state(batch_size, self.hiddens, device) # 初始化状态


# 梯度裁剪，防止梯度爆炸
def grad_clipping(net, theta):
    params = net.params
    norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in params)) # L2 范数
    if norm > theta:
        for param in params:
```

```
            param.grad[:] *= theta / norm # 缩放梯度


# 训练
def train(net, train_iter, vocab, lr, epochs, device):
    loss = nn.CrossEntropyLoss()                # 交叉熵损失函数
    updater = torch.optim.SGD(net.params, lr) # SGD 优化器
    animator = d2l.Animator(xlabel='epoch', ylabel='perplexity', legend=['train'],
xlim=[10, epochs])
    for epoch in range(epochs):
        state = None
        timer = d2l.Timer()
        metric = d2l.Accumulator(2) # 累计损失和样本数量
        for X, Y in train_iter:
            if state is None: # 初始化状态
                state = net.begin_state(batch_size=X.shape[0], device=device)
            else: # 断开历史依赖
                if isinstance(state, tuple): # 如果 state 是元组（LSTM 的状态）
                    for s in state:
                        s.detach_() # 创建新张量，切断与旧梯度的联系
                else:
                    state.detach_()
            y = Y.T.reshape(-1) # 展平
            X, y = X.to(device), y.to(device)
```

```
            y_hat, state = net(X, state) # 前向传播
            l = loss(y_hat, y.long()).mean() # 计算平均损失
            updater.zero_grad() # 清空梯度
            l.backward() # 反向传播
            grad_clipping(net, 1) # 梯度裁剪
            updater.step() # 参数更新
            metric.add(l * d2l.size(y), d2l.size(y)) # 累计损失和样本数量
        ppl = math.exp(metric[0] / metric[1]) # 计算困惑度
        speed = metric[1] / timer.stop() # 计算处理速度
        if (epoch + 1) % 10 == 0:
            print(predict('time traveller', 50, net, vocab, device))
            animator.add(epoch + 1, [ppl])
    print(f'perplexity {ppl:.1f}, {speed:.1f} tokens/sec on {str(device)}')


# 预测
def predict(prefix, len_preds, net, vocab, device):
    state = net.begin_state(batch_size=1, device=device) # 初始化
    outputs = [vocab[prefix[0]]] # 初始化输出列表
    # 获取输入张量表示，调整形状以适应模型输入要求
    get_input = lambda: d2l.reshape(d2l.tensor([outputs[-1]], device=device), (1, 1))
    # 更新状态并记录字符索引
    for y in prefix[1:]:
```

```
        _, state = net(get_input(), state)
        outputs.append(vocab[y])
    # 生成指定长度的新字符
    for _ in range(len_preds):
        y, state = net(get_input(), state)
        outputs.append(int(y.argmax(dim=1).reshape(1)))
    # 将索引转换回字符，连接成完整字符串
    return ''.join([vocab.idx_to_token[i] for i in outputs])


# 设置数据、参数与模型
batch_size = 32         # 批量大小
num_steps = 35          # 序列长度
train_iter, vocab = d2l.load_data_time_machine(batch_size, num_steps) # 数据集
num_epochs = 500        # 训练轮数
lr = 1                  # 学习率
hiddens = 256           # 隐藏层大小
device = d2l.try_gpu() # 训练设备
model = RNN(len(vocab), hiddens, device, get_params, init_state, lstm) # 模型实例
# 训练
start_time = time.time()
train(model, train_iter, vocab, lr, num_epochs, device)
end_time = time.time()
```

```
print("运行时间：%.5f秒" % (end_time - start_time))

# 预测

num_predicts = 100 # 预测字符数

start_time = time.time()

phrases = [

    "time traveller",

    "traveller",

    "the time traveller says that",

    "when the time traveller returns to the garden",

    "the time traveller begins learning the language",

    "the time traveller determines that",

    "the time traveller knows he will have to stop",

    "when he wakes up",

    "the time traveller finds himself",

    "the time traveller tells the narrator to wait for him"

]

for phrase in phrases:

    print(predict(phrase, num_predicts, model, vocab, device))

end_time = time.time()

print("运行时间：%.5f秒" % (end_time - start_time))
```

六、调试训练与分析

（注：括号中对应相应的代码）

1. 教材代码结果展示（LSTM_easy.py）

困惑度	1.1	速度（tokens/s）	298764.3
训练时间（s）	18.24286	预测时间（s）	0.36789
time traveller for so it will be convenient to speak of himwas expounding a recondite matter to us his grey eyes s travelleryou can show black is white by argument said filby but you willnever convince mepossibly not said th the time traveller says that neis expothat me had in lepped is newtime frawe he eab the prou dime to eed thi k wull who and gert when the time traveller returns to the gardent as teref al unstane of spained brop the verllans he ganint spast and alm hown dime yisparowe andwh the time traveller begins learning the language as our think of his four dimensionedbeing which is a fixed and unalterable thingscientific people p the time traveller determines that said a veryayllascemt ans ther seen at instancat of has in exratilays of hat no weind a sighad i je the time traveller knows he will have to stop ofacters of the chand of momererthtand have elen time pracolle batt agell that al shat and they th when he wakes upreat fraided anditss ne lagne tran e yin twyycaicesuitt thereght you shablune excound wains of space the time traveller finds himselfaro ie simwhere time trevell chowg said the time traveller but now you begin to seethe object of my the time traveller tells the narrator to wait for himene why han gnie to sing i sale all rad deat all gan goube abowe erome and larys on a gutten of wo c			
分析：生成的结果中，单词的拼写正确率较高，虽然能够理解语句中包含的事物和大概的意思，但是空格占位和语法的使用较差，语句还是很让人困惑。			

2. 完整代码结果展示（LSTM.py）

困惑度	1.1	速度（tokens/s）	28158.1
训练时间（s）	165.26667	预测时间（s）	0.59110
time traveller care we can go backward and forward freely enoughand men always have done so i admit we move freely travelleryou can show black is white by argument said filby but you willnever convince mepossibly not said th the time traveller says that us a a filed of can ancurferens his said whe time travellerit would be remarkably convenient for th when the time traveller returns to the gardent of move and the fime travellerit s against reason said filbywhy canthry seal dimensioneswire sot h the time traveller begins learning the language ar he has al now said the moticayelo an wore hen a said a onee arowli go om atainowo come a soree i the time traveller determines that us a soatr and that is that whethe			

sothadd orever and so it you wayd the w oth weometht ficetsey hi the time traveller knows he will have to stop ouci thre time ssmeer three thing trover thisg it atweroon upace a soved blas no that i ale sming t when he wakes upouthine move at is intont goven thefoutly nowt faillbabe thisg ancuruly of the fromely move a sot i the time traveller finds himself than at aly the wrodely moven a triet ony that haid the pery young man thoughtin which case they wo the time traveller tells the narrator to wait for himistance that ustals utat uly in a balloont ffor the caushe t are asm the periment a dover and shine
分析：与教材代码相比，自行搭建的代码的效果与其相当，但是运行效率慢很多倍。

3. 模型调试

1） 批量大小（LSTM_easy.py）

批量大小增加（64）			
困惑度	2.3	速度（tokens/s）	560241.0
训练时间（s）	11.34689	预测时间（s）	0.34057
批量大小减少（16）			
困惑度	1.0	速度（tokens/s）	194261.0
训练时间（s）	28.09903	预测时间（s）	0.32900
分析：适宜的批量大小能达到训练效果和运行时间的平衡。增加批量大小降低了模型的泛化能力，增加了困惑度；减小批量大小则延长了训练时间。			

2） 序列长度（LSTM_easy.py）

序列长度增加（60）			
困惑度	1.1	速度（tokens/s）	383954.6
训练时间（s）	16.03161	预测时间（s）	0.35301
分析：更长的序列长度可以让模型学习到更长时间依赖的关系，但也增加了计算复杂度。这里由于给出的文本较少，增加依赖关系的长度并无作用。			

3） 学习率（LSTM_easy.py）

学习率增加（5）			
困惑度	1.0	速度（tokens/s）	298662.2
训练时间（s）	18.47233	预测时间（s）	0.34000
学习率减少（0.1）			
困惑度	9.9	速度（tokens/s）	389662.1
训练时间（s）	15.42866	预测时间（s）	0.33200
分析：合适的学习率对收敛至关重要。太高可能导致无法收敛，太低则可能需要更多轮次才能收敛。			

4） 隐藏层大小（LSTM_easy.py）

隐藏层大小增加			
困惑度	1.0	速度（tokens/s）	296892.6
训练时间（s）	19.18445	预测时间（s）	0.32898

分析：增加隐藏层单元数可能会提升模型的表达能力，但也容易导致过拟合。

5） 优化器选择（LSTM.py）

Adam（lr=0.001）			
困惑度	1.3	速度（tokens/s）	24480.2
训练时间（s）	176.22707	预测时间（s）	0.61801
分析：不同的优化器可能需要不一样的参数设置，这里尝试的 Adam 并没有取得更好的成果。			

6） 权重初始化方法（LSTM.py）

Xavier 初始化			
困惑度	1.3	速度（tokens/s）	32820.3
训练时间（s）	165.21400	预测时间（s）	0.58500
Kaiming 初始化			
困惑度	1.0	速度（tokens/s）	33964.1
训练时间（s）	170.30910	预测时间（s）	0.58886
分析：本实验中，相比于随机初始化，合理的初始化方法最明显的作用是减少运行时间。			

从上述实验来看，调节参数和改进模型都能在困惑度或运行时间方面改善模型的效果。

七、附加题

1. 模型调试

见六、3.。

除了本次实验调试的内容，也可以尝试以下调试手段：

- 修改梯度裁剪中的 theta 值。
- 引入正则化技术。
- 使用其他优化器和权重初始化方法。