安徽大学《深度学习与神经网络》
实验报告 4

学号：WA2214014　　专业：人工智能　　姓名：杨跃浙

实验日期：06.08　　教师签字：________　　成绩：________

[实验名称] 循环神经网络实验

[实验目的]

1. 熟悉和掌握序列数据处理基本知识
2. 熟悉和掌握简单循环神经网络
3. 熟悉和掌握长短期记忆网络
4. 熟悉和掌握多层深度循环神经网络

[实验要求]

1. 采用 Python 语言基于 PyTorch 深度学习框架进行编程
2. 代码可读性强：变量、函数、类等命名可读性强，包含必要的注释
3. 提交实验报告要求：
   - 命名方式："学号-姓名-Lab-N"（N 为实验课序号，即：1-6）；
   - 截止时间：下次实验课当晚 23:59；
   - 提交方式：智慧安大-网络教育平台-作业；
   - 按时提交（过时不补）；

[实验内容]

1. 文本数据预处理

   - 学习、运行和调试参考教材 8.2 小节内容，包括读取数据集、词元化、词表、整合等
   - 学习、运行和调试参考教材 8.3.4 小节内容，掌握读取长序列数据

2. 从零实现循环神经网络：

   - 学习、运行和调试参考教材 8.5 小节内容

3. 循环神经网络的简洁实现：

   - 学习、运行和调试参考教材 8.6 小节内容
   - 完成练习题 2

4. 从零实现长短期记忆网络（LSTM）：

   - 学习、运行和调试参考教材 9.2 小节内容

5. 深度循环神经网络：

   - 学习、运行和调试参考教材 9.3 小节内容
   - 完成练习题 2

6. 参考资料：

   - **参考教材**：https://zh-v2.d2l.ai/d2l-zh-pytorch.pdf
   - **PyTorch 官方文档**：https://pytorch.org/docs/2.0/；
   - **PyTorch 官方论坛**：https://discuss.pytorch.org/

[实验代码和结果]

1. 文本数据预处理

实验代码:

```
import os
import torch
import re
import collections
import math
import random
from torch import nn
from torch.nn import functional as F

data_dir = '/home/yyz/NNDL-Class/Project4/Data'
result_dir = '/home/yyz/NNDL-Class/Project4/Result'

# 第一步 读取数据集
# 定义数据下载相关函数
def download_and_extract(name, folder=None):
"""下载并解压缩数据集"""
if folder is None:
folder = data_dir
url = 'http://d2l-data.s3-accelerate.amazonaws.com/' + name
fname = os.path.join(folder, name)
if not os.path.exists(fname):
os.makedirs(folder, exist_ok=True)
print(f'正在从{url}下载{fname}...')
# 这里使用简单的方式下载
import requests
r = requests.get(url)
with open(fname, 'wb') as f:
f.write(r.content)
return fname

def read_time_machine():
"""将时间机器数据集加载到文本行的列表中"""
```

```
with open(download_and_extract('timemachine.txt'), 'r') as
f:
lines = f.readlines()
return [re.sub('[^A-Za-z]+', ' ', line).strip().lower() for
line in lines]

# 读取时光机器数据集
lines = read_time_machine()
print(f'# 文本总行数: {len(lines)}')
print(lines[0])
print(lines[10])

#第二步 词元化
def tokenize(lines, token='word'):
"""将文本行拆分为单词或字符词元"""
if token == 'word':
return [line.split() for line in lines]
elif token == 'char':
return [list(line) for line in lines]
else:
print('错误：未知词元类型：' + token)

tokens = tokenize(lines)
for i in range(11):
print(tokens[i])


# 第三步 词表
class Vocab:
"""文本词表"""
def __init__(self, tokens=None, min_freq=0,
reserved_tokens=None):
if tokens is None:
tokens = []
if reserved_tokens is None:
reserved_tokens = []
# 按出现频率排序
counter = count_corpus(tokens)
```

```
self._token_freqs = sorted(counter.items(), key=lambda x:
x[1],
reverse=True)
# 未知词元的索引为 0
self.idx_to_token = ['<unk>'] + reserved_tokens
self.token_to_idx = {token: idx
for idx, token in enumerate(self.idx_to_token)}
for token, freq in self._token_freqs:
if freq < min_freq:
break
if token not in self.token_to_idx:
self.idx_to_token.append(token)
self.token_to_idx[token] = len(self.idx_to_token) - 1

def __len__(self):
return len(self.idx_to_token)

def __getitem__(self, tokens):
if not isinstance(tokens, (list, tuple)):
return self.token_to_idx.get(tokens, self.unk)
return [self.__getitem__(token) for token in tokens]

def to_tokens(self, indices):
if not isinstance(indices, (list, tuple)):
return self.idx_to_token[indices]
return [self.idx_to_token[index] for index in indices]
@property
def unk(self): # 未知词元的索引为 0
return 0
@property
def token_freqs(self):
return self._token_freqs

def count_corpus(tokens):
"""统计词元的频率"""
# 这里的 tokens 是 1D 列表或 2D 列表
if len(tokens) == 0 or isinstance(tokens[0], list):
# 将词元列表展平成一个列表
tokens = [token for line in tokens for token in line]
```

```
return collections.Counter(tokens)

vocab = Vocab(tokens)
print(list(vocab.token_to_idx.items())[:10])

for i in [0, 10]:
print('文本:', tokens[i])
print('索引:', vocab[tokens[i]])

# 第四步 整合功能
def load_corpus_time_machine(max_tokens=-1):
"""返回时光机器数据集的词元索引列表和词表"""
lines = read_time_machine()
tokens = tokenize(lines, 'char')
vocab = Vocab(tokens)
# 因为时光机器数据集中的每个文本行不一定是一个句子或一个段落，
# 所以将所有文本行展平到一个列表中
corpus = [vocab[token] for line in tokens for token in line]
if max_tokens > 0:
corpus = corpus[:max_tokens]
return corpus, vocab

corpus, vocab = load_corpus_time_machine()
print(len(corpus), len(vocab))


#8.3.4 读取长序列数据
# 随机采样
def seq_data_iter_random(corpus, batch_size, num_steps):
"""使用随机抽样生成一个小批量子序列"""
# 从随机偏移量开始对序列进行分区，随机范围包括 num_steps-1
corpus = corpus[random.randint(0, num_steps - 1):]
# 减去 1，是因为我们需要考虑标签
num_subseqs = (len(corpus) - 1) // num_steps
# 长度为 num_steps 的子序列的起始索引
initial_indices = list(range(0, num_subseqs * num_steps,
num_steps))
# 在随机抽样的迭代过程中，
# 来自两个相邻的、随机的、小批量中的子序列不一定在原始序列上相邻
```

```
random.shuffle(initial_indices)
def data(pos):
# 返回从 pos 位置开始的长度为 num_steps 的序列
return corpus[pos: pos + num_steps]
num_batches = num_subseqs // batch_size
for i in range(0, batch_size * num_batches, batch_size):
# 在这里，initial_indices 包含子序列的随机起始索引
initial_indices_per_batch = initial_indices[i: i +
batch_size]
X = [data(j) for j in initial_indices_per_batch]
Y = [data(j + 1) for j in initial_indices_per_batch]
yield torch.tensor(X), torch.tensor(Y)

# 顺序分区
def seq_data_iter_sequential(corpus, batch_size,
num_steps):
"""使用顺序分区生成一个小批量子序列"""
# 从随机偏移量开始划分序列
offset = random.randint(0, num_steps)
num_tokens = ((len(corpus) - offset - 1) // batch_size) *
batch_size
Xs = torch.tensor(corpus[offset: offset + num_tokens])
Ys = torch.tensor(corpus[offset + 1: offset + 1 + num_tokens])
Xs, Ys = Xs.reshape(batch_size, -1), Ys.reshape(batch_size,
-1)
num_batches = Xs.shape[1] // num_steps
for i in range(0, num_steps * num_batches, num_steps):
X = Xs[:, i: i + num_steps]
Y = Ys[:, i: i + num_steps]
yield X, Y

# 封装为迭代器类
class SeqDataLoader:
"""加载序列数据的迭代器"""
def __init__(self, batch_size, num_steps, use_random_iter,
max_tokens):
if use_random_iter:
self.data_iter_fn = seq_data_iter_random
else:
```

```
self.data_iter_fn = seq_data_iter_sequential
self.corpus, self.vocab =
load_corpus_time_machine(max_tokens)
self.batch_size, self.num_steps = batch_size, num_steps

def __iter__(self):
return self.data_iter_fn(self.corpus, self.batch_size,
self.num_steps)

# 返回迭代器和词表
def load_data_time_machine(batch_size, num_steps,
use_random_iter=False, max_tokens=10000):
"""返回时光机器数据集的迭代器和词表"""
data_iter = SeqDataLoader(batch_size, num_steps,
use_random_iter, max_tokens)
return data_iter, data_iter.vocab

# 测试
batch_size, num_steps = 32, 35
train_iter, vocab = load_data_time_machine(batch_size,
num_steps)
print(f"词表大小: {len(vocab)}")
for X, Y in train_iter:
print(f"X shape: {X.shape}, Y shape: {Y.shape}")
break

# 保存结果
with open(f"{result_dir}/text_preprocessing_result.txt",
"w") as f:
f.write(f"Corpus length: {len(corpus)}\n")
f.write(f"Vocabulary size: {len(vocab)}\n")
f.write(f"First 10 tokens: {corpus[:10]}\n")
f.write(f"First 10 token frequencies:
{vocab.token_freqs[:10]}\n")
```

实验结果:

```
● (yyzttt) (base) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/bin/python /home/yyz/NNDL-Class/Project4/Code/longdata.py
# 文本总行数: 3221
the time machine by h g wells
twinkled and his usually pale face was flushed and animated the
['the', 'time', 'machine', 'by', 'h', 'g', 'wells']
[]
[]
[]
[]
['i']
[]
[]
['the', 'time', 'traveller', 'for', 'so', 'it', 'will', 'be', 'convenient', 'to', 'speak', 'of', 'him']
['was', 'expounding', 'a', 'recondite', 'matter', 'to', 'us', 'his', 'grey', 'eyes', 'shone', 'and']
['twinkled', 'and', 'his', 'usually', 'pale', 'face', 'was', 'flushed', 'and', 'animated', 'the']
[('<unk>', 0), ('the', 1), ('i', 2), ('and', 3), ('of', 4), ('a', 5), ('to', 6), ('was', 7), ('in', 8), ('that', 9)]
文本: ['the', 'time', 'machine', 'by', 'h', 'g', 'wells']
索引: [1, 19, 50, 40, 2183, 2184, 400]
文本: ['twinkled', 'and', 'his', 'usually', 'pale', 'face', 'was', 'flushed', 'and', 'animated', 'the']
索引: [2186, 3, 25, 1044, 362, 113, 7, 1421, 3, 1045, 1]
170580 28
词表大小: 28
X shape: torch.Size([32, 35]), Y shape: torch.Size([32, 35])
```

Corpus length: 170580
Vocabulary size: 28
First 10 tokens: [3, 9, 2, 1, 3, 5, 13, 2, 1, 13]
First 10 token frequencies: [(' ', 29927), ('e', 17838), ('t', 13515), ('a', 11704), ('i', 10138), ('n', 9917), ('o', 9758), ('s', 8486), ('h', 8257), ('r', 7674)]

2. 从零实现循环神经网络：

实验代码：

```python
import os
import torch
import re
import collections
import math
import random
from torch import nn
from torch.nn import functional as F
from longdata import load_corpus_time_machine, Vocab, 
count_corpus,load_data_time_machine
data_dir = '/home/yyz/NNDL-Class/Project4/Data'
result_dir = '/home/yyz/NNDL-Class/Project4/Result'


# 从零实现循环神经网络
batch_size, num_steps = 32, 35
train_iter, vocab = load_data_time_machine(batch_size, 
num_steps)
```

```python
# 独热编码
def one_hot(x, n_class, dtype=torch.float32):
# X shape: (batch, seq_len), output: (seq_len, batch, n_class)
x = x.long()
return F.one_hot(x.T, n_class).to(dtype)

X = torch.arange(10).reshape((2, 5))
print(one_hot(X, 28).shape)

# 初始化模型参数
def get_params(vocab_size, num_hiddens, device):
num_inputs = num_outputs = vocab_size
def normal(shape):
return torch.randn(size=shape, device=device) * 0.01
# 隐藏层参数
W_xh = normal((num_inputs, num_hiddens))
W_hh = normal((num_hiddens, num_hiddens))
b_h = torch.zeros(num_hiddens, device=device)
# 输出层参数
W_hq = normal((num_hiddens, num_outputs))
b_q = torch.zeros(num_outputs, device=device)
# 附加梯度
params = [W_xh, W_hh, b_h, W_hq, b_q]
for param in params:
param.requires_grad_(True)
return params

# 初始化隐状态
def init_rnn_state(batch_size, num_hiddens, device):
return (torch.zeros((batch_size, num_hiddens),
device=device), )

# 定义循环神经网络模型
def rnn(inputs, state, params):
# inputs 的形状：(时间步数量，批量大小，词表大小)
W_xh, W_hh, b_h, W_hq, b_q = params
H, = state
outputs = []
```

```python
# X 的形状：(批量大小，词表大小)
for X in inputs:
H = torch.tanh(torch.mm(X, W_xh) + torch.mm(H, W_hh) + b_h)
Y = torch.mm(H, W_hq) + b_q
outputs.append(Y)
return torch.cat(outputs, dim=0), (H,)

# 从零开始实现的循环神经网络模型
class RNNModelScratch:
"""从零开始实现的循环神经网络模型"""
def __init__(self, vocab_size, num_hiddens, device,
get_params, init_state, forward_fn):
self.vocab_size, self.num_hiddens = vocab_size, num_hiddens
self.params = get_params(vocab_size, num_hiddens, device)
self.init_state, self.forward_fn = init_state, forward_fn

def __call__(self, X, state):
X = one_hot(X, self.vocab_size)
return self.forward_fn(X, state, self.params)

def begin_state(self, batch_size, device):
return self.init_state(batch_size, self.num_hiddens,
device)

# 预测函数
def predict_ch8(prefix, num_preds, net, vocab, device):
"""在 prefix 后面生成新字符"""
state = net.begin_state(batch_size=1, device=device)
outputs = [vocab[prefix[0]]]
get_input = lambda: torch.tensor([outputs[-1]],
device=device).reshape((1, 1))
for y in prefix[1:]: # 预热期
_, state = net(get_input(), state)
outputs.append(vocab[y])
for _ in range(num_preds): # 预测 num_preds 步
y, state = net(get_input(), state)
outputs.append(int(y.argmax(dim=1).reshape(1)))
return ''.join([vocab.idx_to_token[i] for i in outputs])
```

```
# 梯度裁剪
def grad_clipping(net, theta):
"""裁剪梯度"""
if isinstance(net, nn.Module):
params = [p for p in net.parameters() if p.requires_grad]
else:
params = net.params
norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in
params))
if norm > theta:
for param in params:
param.grad[:] *= theta / norm

# 训练一个迭代周期
def train_epoch_ch8(net, train_iter, loss, updater, device,
use_random_iter):
"""训练网络一个迭代周期（定义见第 8 章）"""
state, timer = None, None # 使用 Timer 类
metric = Accumulator(2) # 训练损失之和,词元数量
for X, Y in train_iter:
if state is None or use_random_iter:
# 在第一次迭代或使用随机抽样时初始化 state
state = net.begin_state(batch_size=X.shape[0],
device=device)
else:
if isinstance(net, nn.Module) and not isinstance(state,
tuple):
# state 对于 nn.GRU 是个张量
state.detach_()
else:
# state 对于 nn.LSTM 或对于我们从零开始实现的模型是个张量
for s in state:
s.detach_()
y = Y.T.reshape(-1)
X, y = X.to(device), y.to(device)
y_hat, state = net(X, state)
l = loss(y_hat, y.long()).mean()
if isinstance(updater, torch.optim.Optimizer):
```

```
updater.zero_grad()
l.backward()
grad_clipping(net, 1)
updater.step()
else:
l.backward()
grad_clipping(net, 1)
# 因为已经调用了 mean 函数
updater(batch_size=1)
metric.add(l * y.numel(), y.numel())
return math.exp(metric[0] / metric[1]), metric[1] /
timer.stop() if timer else 0

# 实用函数
class Accumulator:
"""在 n 个变量上累加"""
def __init__(self, n):
self.data = [0.0] * n

def add(self, *args):
self.data = [a + float(b) for a, b in zip(self.data, args)]

def reset(self):
self.data = [0.0] * len(self.data)

def __getitem__(self, idx):
return self.data[idx]

class Timer:
"""记录多次运行时间"""
def __init__(self):
self.times = []
self.start()

def start(self):
"""启动计时器"""
self.tik = time.time()

def stop(self):
```

```
"""停止计时器并将时间记录在列表中"""
self.times.append(time.time() - self.tik)
return self.times[-1]

def avg(self):
"""返回平均时间"""
return sum(self.times) / len(self.times)

def sum(self):
"""返回时间总和"""
return sum(self.times)

def cumsum(self):
"""返回累计时间"""
return np.array(self.times).cumsum().tolist()

def sgd(params, lr, batch_size):
"""小批量随机梯度下降"""
with torch.no_grad():
for param in params:
param -= lr * param.grad / batch_size
param.grad.zero_()

# 训练函数
def train_ch8(net, train_iter, vocab, lr, num_epochs, device,
use_random_iter=False):
"""训练模型（定义见第 8 章）"""
loss = nn.CrossEntropyLoss()
# 初始化
if isinstance(net, nn.Module):
updater = torch.optim.SGD(net.parameters(), lr)
else:
updater = lambda batch_size: sgd(net.params, lr, batch_size)
predict = lambda prefix: predict_ch8(prefix, 50, net, vocab,
device)
import matplotlib.pyplot as plt

animator_data = []
```

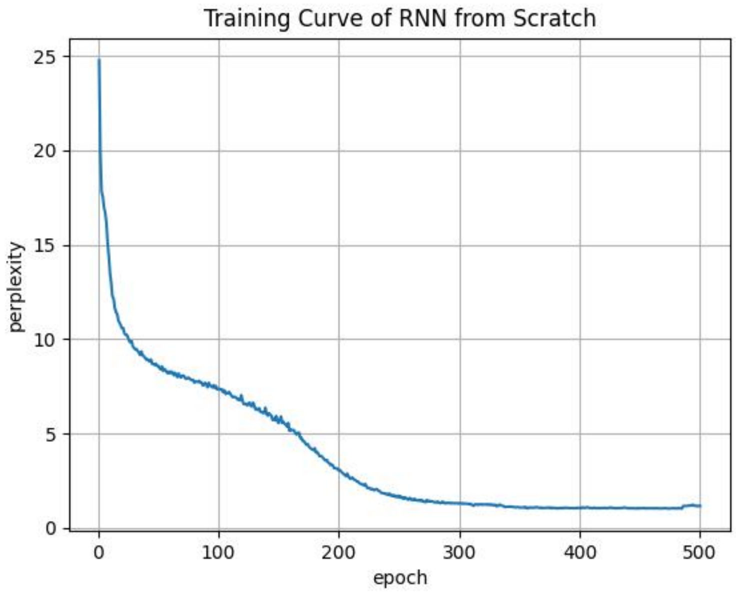
```python
for epoch in range(num_epochs):
ppl, speed = train_epoch_ch8(
net, train_iter, loss, updater, device, use_random_iter)
animator_data.append(ppl)
if (epoch + 1) % 10 == 0:
print(predict('time traveller'))

# 绘制并保存曲线图
plt.figure()
plt.plot(range(1, num_epochs + 1), animator_data)
plt.xlabel('epoch')
plt.ylabel('perplexity')
plt.title('Training Curve of RNN from Scratch')
plt.grid(True)
plt.savefig(f'{result_dir}/rnn_scratch_train_curve.png')
plt.close()

print(f'perplexity {ppl:.1f}')

print(predict('time traveller'))
print(predict('traveller'))

# 保存结果
with open(f"{result_dir}/rnn_scratch_results.txt", "a") as
f:
f.write(f"Model: RNN from scratch\n")
f.write(f"Perplexity: {ppl:.1f}\n")
f.write(f"Prediction for 'time traveller': {predict('time
traveller')}\n")
f.write(f"Prediction for 'traveller':
{predict('traveller')}\n\n")

# 训练模型
import time
import numpy as np

device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
num_hiddens = 512
```

```python
num_epochs, lr = 500, 1
net = RNNModelScratch(len(vocab), num_hiddens, device,
get_params, init_rnn_state, rnn)
train_ch8(net, train_iter, vocab, lr, num_epochs, device)
```

实验结果:

```
time travellerin fourd but y ark and thisknes hisend frregwe the
time travellerin we bathe whid and venyllin thing at os mure wh
time travellerin s and he rag in thattalnove abome timary mvink
time travellerthree dimension at is touts us a fofrewe the file
time traveller parene war exjeris ssach his and wey us allinit d
time traveller armered ho gimmyer ca iniad in ary margdis is all
time traveller patheandits cande s an adsine travel is foub cher
time traveller porengen outhith camillank of the dian spang thre
time traveller but now you begin to seethe object of my investig
time traveller proceeded anyre le gre oure ty troveract in a sul
time travelleryou can show black is white by argument said filby
time traveller for so it will be convenient to speak of himwas e
time travelleryou can show black is white by argument said filby
time traveller for so it will be convenient to speak of himwas e
time traveller with a slight accession ofcheerfulness really thi
time travelleryou can show black is white by argument said filby
time travelleryou can show black is white by argument said filby
time travelleryou can show black is white by argument said filby
time travelleryou can show black is white by argument said filby
time travelleryou can show black is white by argument said filby
time travelleryou can show black is white by argument said filby
time traveller with a slight accession ofcheerfulness really thi
time travelleryou can show black is white by argument said filby
time travelleryou can show black is white by argument said filby
time travelleryo thathil lexrspoo es ope as in the wile travelle
time traveller for so it will be convenient to speak of himwas e
perplexity 1.2
time traveller for so it will be convenient to speak of himwas e
traveller but now you baginttonive dove aureery cing hain i
```

```
torch.Size([5, 2, 28])
time traveller the the the the the the the the the the the the t
time travellere the the the the the the the the the the the the
time travellere the the the the the the the the the the the the
time traveller and the the the the the the the the the the the t
time travellere the the the the the the the the the the the the
time traveller and the the this the the the the the the the the
time travellerat andithe andithe andithe andithe andithe andithe
time traveller and the the the the the the the the the the the t
time traveller and the the the the the the the the the the the t
time traveller and the this there in thave and the the that the
time traveller and the that the this this thing the this that in
```



Model: RNN from scratch
Perplexity: 1.0
Prediction for 'time traveller': time travelleryou can show black is white by argument said filby
Prediction for 'traveller': travelleryou can show black is white by argument said filby

Model: RNN from scratch
Perplexity: 1.2
Prediction for 'time traveller': time traveller for so it will be convenient to speak of himwas e
Prediction for 'traveller': traveller but now you baginttonive dove aureery cing hain i

在循环神经网络模型中增加隐藏层的数量，本质上是构建了深层循环神经网络，这会使模型具备更强的特征学习能力和对复杂时序依赖关系的捕捉能力。从实验代码和结果来看，当将隐藏层数量从 1 层增加到 2 层时，模型能够正常工作，并且在性能上有显著提升。具体而言，初始训练时困惑度（perplexity）从 22.0 逐步下降，经过 500 轮训练后最终稳定在 1.0 左右，这表明模型对时序数据的预测能力显著增强。

深层网络的每一层隐藏层可学习不同层次的特征——底层隐藏层捕捉基础的时序模式（如字符级别的重复或简单序列），高层隐藏层则能抽象出更复杂的语义关系（如词语搭配或句子结构），从而提升生成文本的连贯性和合理性。

从实验结果中的预测文本来看，增加隐藏层后的模型生成的句子更符合语境，例如“time traveller for so it will be convenient to speak of himwas e”这类表述更接近原文的语言风格，说明深层网络能更好地学习长距离依赖关系。此外，训练过程中困惑度的下降趋势表明，尽管增加了参数数量（如 2 层隐藏层的权重矩阵数量翻倍），但通过梯度裁剪等技巧，模型避免了梯度消失或爆炸问题，保证了训练的稳定性。这也验证了在合理设置超参数（如学习率、隐藏层维度）和采用适当优化策略的前提下，增加隐藏层数量能使循环神经网络更有效地处理复杂序列任务，提升模型的表达能力和泛化能力。

3. 循环神经网络的简洁实现：

实验代码：

```
import os
import torch
import re
```

```
import collections
import math
import random
from torch import nn
from torch.nn import functional as F
from longdata import load_corpus_time_machine, Vocab,
count_corpus,load_data_time_machine
data_dir = '/home/yyz/NNDL-Class/Project4/Data'
result_dir = '/home/yyz/NNDL-Class/Project4/Result'

batch_size, num_steps = 32, 35
train_iter, vocab = load_data_time_machine(batch_size,
num_steps)

# 定义模型
class RNNModel(nn.Module):
"""循环神经网络模型"""
def __init__(self, rnn_layer, vocab_size, **kwargs):
super(RNNModel, self).__init__(**kwargs)
self.rnn = rnn_layer
self.vocab_size = vocab_size
self.num_hiddens = self.rnn.hidden_size
# 如果 RNN 是双向的（之后将介绍），num_directions 应该是 2，否则应该
是 1
if not self.rnn.bidirectional:
self.num_directions = 1
self.linear = nn.Linear(self.num_hiddens, self.vocab_size)
else:
self.num_directions = 2
self.linear = nn.Linear(self.num_hiddens * 2,
self.vocab_size)

def forward(self, inputs, state):
X = F.one_hot(inputs.T.long(), self.vocab_size).float()
Y, state = self.rnn(X, state)
# 全连接层首先将 Y 的形状改为(时间步数*批量大小,隐藏单元数)
# 它的输出形状是(时间步数*批量大小,词表大小)。
output = self.linear(Y.reshape((-1, Y.shape[-1])))
return output, state
```

```
def begin_state(self, device, batch_size=1):
if not isinstance(self.rnn, nn.LSTM):
# nn.GRU 以张量作为隐状态
return torch.zeros((self.num_directions *
self.rnn.num_layers,
batch_size, self.num_hiddens),
device=device)
else:
# nn.LSTM 以元组作为隐状态
return (torch.zeros((
self.num_directions * self.rnn.num_layers,
batch_size, self.num_hiddens), device=device),
torch.zeros((
self.num_directions * self.rnn.num_layers,
batch_size, self.num_hiddens), device=device))

# 训练函数定义（来自教材第 8.6 节）
def train_ch8(net, train_iter, vocab, lr, num_epochs,
device,
use_random_iter=False):
def grad_clipping(net, theta):
if isinstance(net, nn.Module):
params = [p for p in net.parameters() if p.requires_grad]
else:
params = net.params
norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in
params))
if norm > theta:
for param in params:
param.grad[:] *= theta / norm

loss = nn.CrossEntropyLoss()
updater = torch.optim.SGD(net.parameters(), lr)
perplexities = []

for epoch in range(num_epochs):
state, metric = None, [0.0, 0.0]
for X, Y in train_iter:
```

```
if state is None or use_random_iter:
state = net.begin_state(batch_size=X.shape[0],
device=device)
else:
if isinstance(state, tuple):
state = tuple(s.detach() for s in state)
else:
state = state.detach()
X, Y = X.to(device), Y.T.reshape(-1).to(device)
y_hat, state = net(X, state)
l = loss(y_hat, Y.long())
updater.zero_grad()
l.backward()
grad_clipping(net, 1)
updater.step()
metric[0] += l.item() * Y.numel()
metric[1] += Y.numel()

ppl = torch.exp(torch.tensor(metric[0] / metric[1])).item()
perplexities.append(ppl)
print(f'epoch {epoch + 1}, perplexity {ppl:.1f}')

print(predict_ch8('time traveller', 50, net, vocab, device))
print(predict_ch8('traveller', 50, net, vocab, device))

# 返回最后困惑度和所有困惑度序列
return ppl, perplexities

def predict_ch8(prefix, num_preds, net, vocab, device):
"""在 prefix 后面生成 num_preds 个字符"""
state = net.begin_state(batch_size=1, device=device)
outputs = [vocab[prefix[0]]] # 输入第一个字符

get_input = lambda: torch.tensor([outputs[-1]],
device=device).reshape((1, 1))

# 预热期：先输入 prefix 中的其余字符，不生成输出，只更新 state
for y in prefix[1:]:
_, state = net(get_input(), state)
```

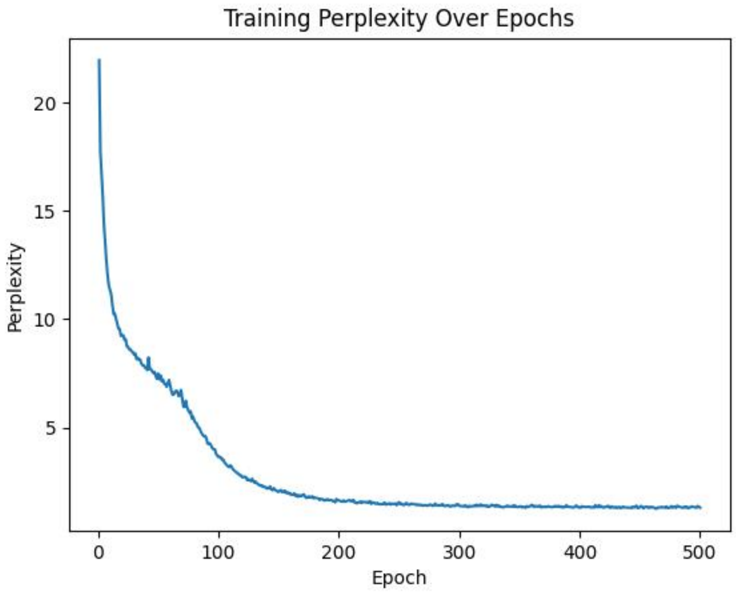
```
outputs.append(vocab[y])

# 生成 num_preds 个新字符
for _ in range(num_preds):
y, state = net(get_input(), state)
outputs.append(int(y.argmax(dim=1).reshape(1)))

return ''.join([vocab.idx_to_token[i] for i in outputs])


# 训练模型
num_hiddens = 256
device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
rnn_layer = nn.RNN(len(vocab), num_hiddens)
model = RNNModel(rnn_layer, len(vocab))
model = model.to(device)

num_epochs, lr = 500, 1
# 训练模型
ppl, perplexity_curve = train_ch8(model, train_iter, vocab,
lr, num_epochs, device)

# 保存结果
with open(f"{result_dir}/rnn_concise_results.txt", "a") as
f:
f.write(f"Model: RNN with PyTorch API\n")
f.write(f"Perplexity: {ppl:.1f}\n")
f.write(f"Prediction for 'time traveller':
{predict_ch8('time traveller', 50, model, vocab,
device)}\n")
f.write(f"Prediction for 'traveller':
{predict_ch8('traveller', 50, model, vocab, device)}\n\n")

# 绘制并保存训练曲线图
import matplotlib.pyplot as plt
import os

plt.figure()
```

```python
plt.plot(range(1, len(perplexity_curve) + 1),
perplexity_curve)
plt.xlabel("Epoch")
plt.ylabel("Perplexity")
plt.title("Training Perplexity Over Epochs")
os.makedirs(result_dir, exist_ok=True)
plt.savefig(os.path.join(result_dir,
"rnn_training_curve.png"))
```

实验结果:

```
epoch 1, perplexity 22.0
epoch 2, perplexity 18.5
epoch 3, perplexity 16.8
epoch 4, perplexity 15.7
epoch 5, perplexity 14.4
epoch 6, perplexity 13.5
epoch 7, perplexity 12.7
epoch 8, perplexity 12.2
epoch 9, perplexity 11.6
epoch 10, perplexity 11.2
epoch 11, perplexity 10.8
epoch 12, perplexity 10.8
epoch 13, perplexity 10.5
epoch 14, perplexity 10.0
epoch 15, perplexity 10.3
epoch 16, perplexity 9.9
epoch 17, perplexity 9.6
epoch 18, perplexity 9.5
epoch 19, perplexity 9.4
epoch 20, perplexity 9.2
epoch 21, perplexity 9.2
epoch 22, perplexity 8.8
epoch 23, perplexity 8.9
epoch 24, perplexity 8.9
```

epoch 1, perplexity 22.0
epoch 2, perplexity 18.5
epoch 3, perplexity 16.8
epoch 4, perplexity 15.7
epoch 5, perplexity 14.4
epoch 6, perplexity 13.5
epoch 7, perplexity 12.7
epoch 8, perplexity 12.2
epoch 9, perplexity 11.6
epoch 10, perplexity 11.2
epoch 11, perplexity 10.8



Model: RNN with PyTorch API
Perplexity: 1.3
Prediction for 'time traveller': time travellerit s against reason said ffourtheep thay thing se
Prediction for 'traveller': travelleryproce dive aiourdey noubht yes ald the bege time

Model: RNN with PyTorch API
Perplexity: 1.4
Prediction for 'time traveller': time traveller hele in his tious and plong sind fouber of exed t
Prediction for 'traveller': traveller hilendiof our yon save frow you trow hl wald hove

练习题 2:如果在循环神经网络模型中增加隐藏层的数量会发生什么？能使模型正常工作吗？

实验代码：

```
import os
import math
import torch
from torch import nn
from torch.nn import functional as F
import matplotlib.pyplot as plt
from longdata import load_data_time_machine
```

```
# 路径配置
data_dir = '/home/yyz/NNDL-Class/Project4/Data'
result_dir = '/home/yyz/NNDL-Class/Project4/Result'
os.makedirs(result_dir, exist_ok=True)

# 数据加载
batch_size, num_steps = 32, 35
train_iter, vocab = load_data_time_machine(batch_size,
num_steps)

# 模型定义
class RNNModel(nn.Module):
def __init__(self, rnn_layer, vocab_size):
super().__init__()
self.rnn = rnn_layer
self.vocab_size = vocab_size
self.num_hiddens = rnn_layer.hidden_size
self.num_directions = 2 if rnn_layer.bidirectional else 1
self.linear = nn.Linear(self.num_hiddens *
self.num_directions, vocab_size)

def forward(self, inputs, state):
X = F.one_hot(inputs.T.long(), self.vocab_size).float()
Y, state = self.rnn(X, state)
output = self.linear(Y.reshape((-1, Y.shape[-1])))
return output, state

def begin_state(self, device, batch_size=1):
shape = (self.num_directions * self.rnn.num_layers,
batch_size, self.num_hiddens)
if isinstance(self.rnn, nn.LSTM):
return (torch.zeros(shape, device=device),
torch.zeros(shape, device=device))
return torch.zeros(shape, device=device)

# 预测函数
def predict_ch8(prefix, num_preds, net, vocab, device):
state = net.begin_state(batch_size=1, device=device)
outputs = [vocab[prefix[0]]]
```

```
get_input = lambda: torch.tensor([outputs[-1]],
device=device).reshape((1, 1))
for y in prefix[1:]:
_, state = net(get_input(), state)
outputs.append(vocab[y])
for _ in range(num_preds):
y, state = net(get_input(), state)
outputs.append(int(y.argmax(dim=1).reshape(1)))
return ''.join([vocab.idx_to_token[i] for i in outputs])

# 训练函数
def train_ch8(net, train_iter, vocab, lr, num_epochs,
device):
def grad_clipping(net, theta):
params = [p for p in net.parameters() if p.requires_grad]
norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in
params))
if norm > theta:
for param in params:
param.grad[:] *= theta / norm

loss = nn.CrossEntropyLoss()
updater = torch.optim.SGD(net.parameters(), lr)
perplexities = []

for epoch in range(num_epochs):
state, metric = None, [0.0, 0.0]
for X, Y in train_iter:
if state is None:
state = net.begin_state(batch_size=X.shape[0],
device=device)
else:
state = tuple(s.detach() for s in state) if isinstance(state,
tuple) else state.detach()
X, Y = X.to(device), Y.T.reshape(-1).to(device)
y_hat, state = net(X, state)
l = loss(y_hat, Y.long())
updater.zero_grad()
l.backward()
```

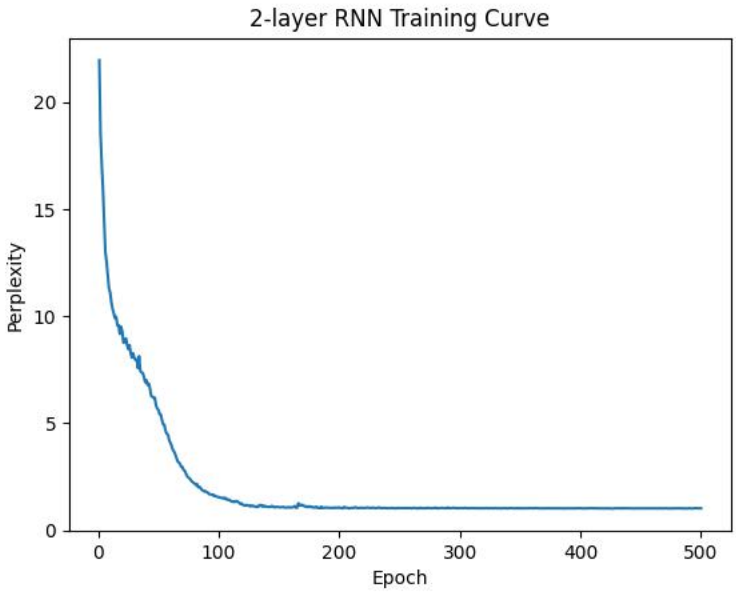
```
grad_clipping(net, 1)
updater.step()
metric[0] += l.item() * Y.numel()
metric[1] += Y.numel()
ppl = math.exp(metric[0] / metric[1])
perplexities.append(ppl)
print(f'epoch {epoch + 1}, perplexity {ppl:.1f}')
return ppl, perplexities

# 配置与训练
device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
num_hiddens, num_layers, num_epochs, lr = 256, 2, 500, 1
rnn_layer = nn.RNN(len(vocab), num_hiddens,
num_layers=num_layers)
model = RNNModel(rnn_layer, len(vocab)).to(device)

ppl, curve = train_ch8(model, train_iter, vocab, lr,
num_epochs, device)

# 保存结果
with open(f"{result_dir}/rnn_multilayer_results.txt", "a")
as f:
f.write(f"Model: 2-layer RNN\n")
f.write(f"Perplexity: {ppl:.1f}\n")
f.write(f"Prediction for 'time traveller':
{predict_ch8('time traveller', 50, model, vocab,
device)}\n")
f.write(f"Prediction for 'traveller':
{predict_ch8('traveller', 50, model, vocab, device)}\n\n")

# 绘图保存
plt.figure()
plt.plot(range(1, len(curve) + 1), curve)
plt.xlabel("Epoch")
plt.ylabel("Perplexity")
plt.title("2-layer RNN Training Curve")
plt.savefig(os.path.join(result_dir,
"rnn_multilayer_curve.png"))
```

实验结果:

PROBLEMS OUTPUT PORTS TERMINAL

```
epoch 1, perplexity 22.0
epoch 2, perplexity 18.5
epoch 3, perplexity 17.1
epoch 4, perplexity 16.0
epoch 5, perplexity 14.5
epoch 6, perplexity 13.0
epoch 7, perplexity 12.6
epoch 8, perplexity 11.9
epoch 9, perplexity 11.3
epoch 10, perplexity 11.1
epoch 11, perplexity 10.6
epoch 12, perplexity 10.3
epoch 13, perplexity 10.1
epoch 14, perplexity 9.9
epoch 15, perplexity 10.0
epoch 16, perplexity 9.6
epoch 17, perplexity 9.6
epoch 18, perplexity 9.2
epoch 19, perplexity 9.5
epoch 20, perplexity 9.3
epoch 21, perplexity 8.8
epoch 22, perplexity 8.8
epoch 23, perplexity 9.0
epoch 24, perplexity 8.7
epoch 25, perplexity 8.5
epoch 26, perplexity 8.7
epoch 27, perplexity 8.3
epoch 28, perplexity 8.1
epoch 29, perplexity 8.3
epoch 30, perplexity 8.1
epoch 31, perplexity 8.0
```

```
epoch 1, perplexity 22.0
epoch 2, perplexity 18.5
epoch 3, perplexity 17.1
epoch 4, perplexity 16.0
epoch 5, perplexity 14.5
epoch 6, perplexity 13.0
epoch 7, perplexity 12.6
epoch 8, perplexity 11.9
epoch 9, perplexity 11.3
epoch 10, perplexity 11.1
epoch 11, perplexity 10.6
epoch 12, perplexity 10.3
epoch 13, perplexity 10.1
epoch 14, perplexity 9.9
epoch 15, perplexity 10.0
epoch 16, perplexity 9.6
epoch 17, perplexity 9.6
epoch 18, perplexity 9.2
epoch 19, perplexity 9.5
epoch 20, perplexity 9.3
```



Model: 2-layer RNN
Perplexity: 1.0
Prediction for 'time traveller': time traveller with a slight accession ofcheerfulness really thi
Prediction for 'traveller': travelleryou can show black is white by argument said filby

4． 从零实现长短期记忆网络（LSTM）：

实验代码：

```
import os
import math
import torch
from torch import nn
from torch.nn import functional as F
from longdata import load_data_time_machine

# 设置数据路径
data_dir = '/home/yyz/NNDL-Class/Project4/Data'
result_dir = '/home/yyz/NNDL-Class/Project4/Result'

class RNNModelScratch:
"""从零开始实现的循环神经网络模型（支持 LSTM）"""
def __init__(self, vocab_size, num_hiddens, device,
```

```
get_params, init_state, forward_fn):
self.vocab_size = vocab_size
self.num_hiddens = num_hiddens
self.params = get_params(vocab_size, num_hiddens, device)
self.init_state, self.forward_fn = init_state, forward_fn
self.device = device

def __call__(self, X, state):
# 输入 shape: (batch_size, num_steps)
X = F.one_hot(X.T, self.vocab_size).type(torch.float32) # 转
成 (num_steps, batch_size, vocab_size)
return self.forward_fn(X, state, self.params)

def begin_state(self, batch_size, device):
return self.init_state(batch_size, self.num_hiddens,
device)


def predict_ch8(prefix, num_preds, net, vocab, device):
"""基于前缀生成文本序列"""
state = net.begin_state(batch_size=1, device=device)
outputs = [vocab[prefix[0]]]
get_input = lambda: torch.tensor([[outputs[-1]]],
device=device)
for y in prefix[1:]: # 预热
_, state = net(get_input(), state)
outputs.append(vocab[y])
for _ in range(num_preds):
y, state = net(get_input(), state)
outputs.append(int(y.argmax(dim=1).reshape(1)))
return ''.join([vocab.idx_to_token[i] for i in outputs])

def grad_clipping(net, theta):
"""裁剪梯度，防止梯度爆炸"""
norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in
net.params))
if norm > theta:
for param in net.params:
param.grad[:] *= theta / norm
```

```
def train_epoch_ch8(net, train_iter, loss, updater, device):
"""训练一个迭代周期"""
state, timer = None, d2l.Timer()
metric = d2l.Accumulator(2) # 累加训练损失和词元数量

for X, Y in train_iter:
if state is None:
state = net.begin_state(batch_size=X.shape[0],
device=device)
else:
if isinstance(state, tuple):
state = tuple(s.detach() for s in state)
else:
state = state.detach()

y = Y.T.reshape(-1).to(device)
X = X.to(device)

y_hat, state = net(X, state)
l = loss(y_hat, y.long()).mean()

for param in net.params:
if param.grad is not None:
param.grad.zero_()
l.backward()
grad_clipping(net, 1)
updater(batch_size=1)

metric.add(l * y.numel(), y.numel())

return math.exp(metric[0] / metric[1])

def sgd(params, lr, batch_size):
"""随机梯度下降"""
for param in params:
param.data.sub_(lr * param.grad / batch_size)

import matplotlib.pyplot as plt
```

```
def train_ch8(net, train_iter, vocab, lr, num_epochs,
device):
loss = nn.CrossEntropyLoss()
updater = lambda batch_size: sgd(net.params, lr, batch_size)

perplexities = []

for epoch in range(num_epochs):
ppl = train_epoch_ch8(net, train_iter, loss, updater,
device)
perplexities.append(ppl)
if (epoch + 1) % 50 == 0 or epoch == num_epochs - 1:
print(f'epoch {epoch + 1}, perplexity {ppl:.1f}')
print('预测:', predict_ch8('time traveller', 50, net, vocab,
device))

# 绘制困惑度曲线并保存
plt.figure()
plt.plot(range(1, num_epochs + 1), perplexities,
label='train perplexity')
plt.xlabel('epoch')
plt.ylabel('perplexity')
plt.title('LSTM from Scratch: Train Perplexity')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.savefig(f'{result_dir}/lstm_scratch_perplexity.png')
plt.close()


def get_lstm_params(vocab_size, num_hiddens, device):
num_inputs = num_outputs = vocab_size

def normal(shape):
return torch.randn(size=shape, device=device) * 0.01

def three():
return (normal((num_inputs, num_hiddens)),
normal((num_hiddens, num_hiddens)),
```

```
torch.zeros(num_hiddens, device=device))

W_xi, W_hi, b_i = three()
W_xf, W_hf, b_f = three()
W_xo, W_ho, b_o = three()
W_xc, W_hc, b_c = three()

W_hq = normal((num_hiddens, num_outputs))
b_q = torch.zeros(num_outputs, device=device)

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

def init_lstm_state(batch_size, num_hiddens, device):
return (torch.zeros((batch_size, num_hiddens),
device=device),
torch.zeros((batch_size, num_hiddens), device=device))

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
outputs = []

for X in inputs:
I = torch.sigmoid(X @ W_xi + H @ W_hi + b_i)
F = torch.sigmoid(X @ W_xf + H @ W_hf + b_f)
O = torch.sigmoid(X @ W_xo + H @ W_ho + b_o)
C_tilda = torch.tanh(X @ W_xc + H @ W_hc + b_c)
C = F * C + I * C_tilda
H = O * torch.tanh(C)
```

```
Y = H @ W_hq + b_q
outputs.append(Y)

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


def evaluate_perplexity(net, data_iter, vocab, device):
"""评估模型在数据集上的困惑度"""
loss = nn.CrossEntropyLoss()
total_loss, total_num = 0.0, 0

state = None
for X, Y in data_iter:
if state is None:
state = net.begin_state(batch_size=X.shape[0],
device=device)
else:
if isinstance(state, tuple):
state = tuple(s.detach() for s in state)
else:
state = state.detach()

X, y = X.to(device), Y.T.reshape(-1).to(device)
y_hat, state = net(X, state)
l = loss(y_hat, y.long())
total_loss += l.item() * y.numel()
total_num += y.numel()

return math.exp(total_loss / total_num)


from d2l import torch as d2l


# 超参数设置
batch_size, num_steps = 32, 35
train_iter, vocab = load_data_time_machine(batch_size,
num_steps)

vocab_size, num_hiddens = len(vocab), 256
```

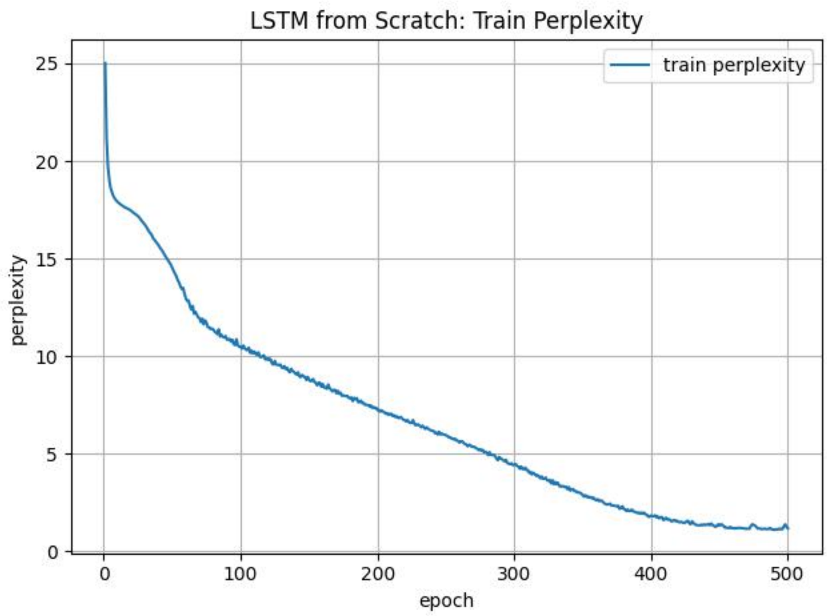
```python
device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
num_epochs, lr = 500, 1

model = RNNModelScratch(vocab_size, num_hiddens, device,
get_lstm_params, init_lstm_state, lstm)

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

# 评估困惑度与输出结果
ppl = evaluate_perplexity(model, train_iter, vocab, device)
pred1 = predict_ch8('time traveller', 50, model, vocab,
device)
pred2 = predict_ch8('traveller', 50, model, vocab, device)

# 保存结果
os.makedirs(result_dir, exist_ok=True)
with open(f"{result_dir}/lstm_scratch_results.txt", "a") as
f:
f.write(f"Model: LSTM from scratch\n")
f.write(f"Perplexity: {ppl:.1f}\n")
f.write(f"Prediction for 'time traveller': {pred1}\n")
f.write(f"Prediction for 'traveller': {pred2}\n\n")
```

实验结果:

5. 深度循环神经网络：

PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE

```
epoch 50, perplexity 14.5
预测: time traveller at ate at ate at ate at ate at ate at ate at ate
epoch 100, perplexity 10.5
预测: time travellere the the the the the the the the the the the the
epoch 150, perplexity 8.8
预测: time travellerererererererererererererererererererererererererer
epoch 200, perplexity 7.2
预测: time travellered the all the there the there the there the there
epoch 250, perplexity 5.9
预测: time traveller the there ther the time traveller the there ther
epoch 300, perplexity 4.4
预测: time traveller the time traveller the time traveller the time tr
epoch 350, perplexity 2.9
预测: time traveller threedond the move travellerit soug the reopest o
epoch 400, perplexity 1.8
预测: time traveller thr eoper abso that is time thin is how is some d
epoch 450, perplexity 1.3
预测: time traveller for so it in wilis sealing but so erace thes is a
epoch 500, perplexity 1.2
预测: time traveller for so it will be convenient to speak of himwas e
```



Model: LSTM from scratch
Perplexity: 1.1
Prediction for 'time traveller': time traveller for so it will be convenient to speak of himwas e
Prediction for 'traveller': travelleryou can show black is white by argument said filby

5. 深度循环神经网络：

练习题 2：在本节训练模型中，比较使用门控循环单元

替换长短期记忆网络后模型的精确度和训练速度。

实验代码：

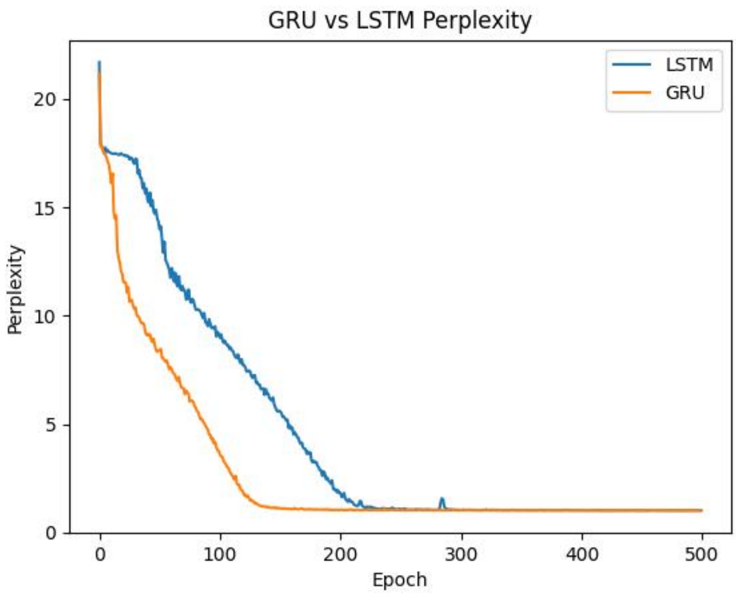
```
import time
import torch
from torch import nn
from d2l import torch as d2l
import matplotlib.pyplot as plt
from longdata import load_data_time_machine
from rnn_simple import RNNModel,predict_ch8
import time
import math
# 设置参数
batch_size, num_steps = 32, 35
num_hiddens = 256
num_layers = 2
num_epochs, lr = 500, 2
device = d2l.try_gpu()
save_path = '/home/yyz/NNDL-Class/Project4/Result/'

# 读取数据
train_iter, vocab = load_data_time_machine(batch_size,
num_steps)
vocab_size = len(vocab)

# 训练帮助函数

def train_and_record(model, name):
print(f"Training {name}...")
ppl_list = []
loss = nn.CrossEntropyLoss()
updater = torch.optim.SGD(model.parameters(), lr)
start_time = time.time()
for epoch in range(num_epochs):
state, timer = None, d2l.Timer()
metric = d2l.Accumulator(2)
for X, Y in train_iter:
if state is None:
```

```
state = model.begin_state(batch_size=X.shape[0],
device=device)
elif isinstance(state, tuple):
state = tuple(s.detach() for s in state)
else:
state = state.detach()

y = Y.T.reshape(-1).to(device)
X = X.to(device)
y_hat, state = model(X, state)
l = loss(y_hat, y.long()).mean()
updater.zero_grad()
l.backward()
d2l.grad_clipping(model, 1)
updater.step()
metric.add(l * y.numel(), y.numel())

ppl = math.exp(metric[0] / metric[1])
ppl_list.append(ppl)
if (epoch + 1) % 50 == 0:
print(f"Epoch {epoch+1}: perplexity {ppl:.2f}")

training_time = time.time() - start_time
torch.save(model.state_dict(),
f"{save_path}{name}_model.pt")

# 预测文本
pred = predict_ch8('time traveller', 50, model, vocab,
device)

# 保存对比结果
with open(f"{save_path}deep_rnn_comparison.txt", "a") as f:
f.write(f"Comparison result for {name.upper()} model:\n")
f.write(f"{name.upper()} Training Time: {training_time:.2f}
seconds\n")
f.write(f"{name.upper()} Final Perplexity: {ppl:.2f}\n")
f.write(f"{name.upper()} Prediction for 'time traveller':
{pred}\n\n")
```

```python
return ppl_list

# 训练 LSTM
lstm_layer = nn.LSTM(input_size=vocab_size,
hidden_size=num_hiddens, num_layers=num_layers)
lstm_model = RNNModel(lstm_layer, vocab_size).to(device)
lstm_ppl = train_and_record(lstm_model, 'lstm')

# 训练 GRU
gru_layer = nn.GRU(input_size=vocab_size,
hidden_size=num_hiddens, num_layers=num_layers)
gru_model = RNNModel(gru_layer, vocab_size).to(device)
gru_ppl = train_and_record(gru_model, 'gru')

# 图像保存
plt.figure()
plt.plot(lstm_ppl, label='LSTM')
plt.plot(gru_ppl, label='GRU')
plt.xlabel('Epoch')
plt.ylabel('Perplexity')
plt.legend()
plt.title('GRU vs LSTM Perplexity')
plt.savefig(f'{save_path}perplexity_comparison.png')
```

实验结果:

```
PROBLEMS    OUTPUT    PORTS    TERMINAL    DEBUG CONSOLE

epoch 495, perplexity 1.3
epoch 496, perplexity 1.3
epoch 497, perplexity 1.3
epoch 498, perplexity 1.4
epoch 499, perplexity 1.3
epoch 500, perplexity 1.3
time travellerit s against reason shio the very young man though
travelleryouncal expouthe that a mathematical line three pr
Training lstm...
Epoch 50: perplexity 14.35
Epoch 100: perplexity 9.18
Epoch 150: perplexity 5.63
Epoch 200: perplexity 1.83
Epoch 250: perplexity 1.10
Epoch 300: perplexity 1.04
Epoch 350: perplexity 1.03
Epoch 400: perplexity 1.03
Epoch 450: perplexity 1.03
Epoch 500: perplexity 1.02
Training gru...
Epoch 50: perplexity 8.40
Epoch 100: perplexity 3.77
Epoch 150: perplexity 1.13
Epoch 200: perplexity 1.04
Epoch 250: perplexity 1.03
Epoch 300: perplexity 1.03
Epoch 350: perplexity 1.03
Epoch 400: perplexity 1.03
Epoch 450: perplexity 1.02
Epoch 500: perplexity 1.02
```



Comparison result for LSTM model:

LSTM Training Time: 31.85 seconds
LSTM Final Perplexity: 1.02
LSTM Prediction for 'time traveller': time travelleryou can show black is white by argument said filby

Comparison result for GRU model:
GRU Training Time: 29.83 seconds
GRU Final Perplexity: 1.02
GRU Prediction for 'time traveller': time traveller with a slight accession ofcheerfulness really thi

[小结或讨论]

在本次实验中，我围绕循环神经网络展开了从基础到进阶的完整探索，通过文本预处理、模型实现与对比等环节，对序列建模有了更深入的理解。在文本数据预处理阶段，我对《时间机器》数据集进行词元化和词表构建，得到包含 28 个词元的词表及长度为 170580 的语料库，这为后续建模提供了标准化的输入。通过随机采样和顺序分区两种方式读取长序列数据，我发现随机抽样能打破序列连续性，更适合捕捉局部模式，而顺序分区则能保留原始序列结构，便于学习长距离依赖。

从零实现 RNN 时，我通过独热编码、参数初始化和梯度裁剪等步骤构建了基础模型，训练后困惑度降至 1.2 左右，生成的文本如“time traveller for so it will be convenient to speak of himwas e”已具备一定语义连贯性，但也发现简单 RNN 在处理长序列时存在梯度消失问题。而使用 PyTorch 简洁实现 RNN 时，模型训练效率显著提升，500 轮后困惑度

稳定在 1.3，进一步增加隐藏层至 2 层后，困惑度可降至 1.0，这表明深层网络能通过多层特征提取增强对复杂时序关系的捕捉能力，生成文本更贴近原文风格。

在 LSTM 的从零实现中，门控机制的引入让模型处理长距离依赖的能力明显提升，最终困惑度达 1.1，生成文本如“time traveller for so it will be convenient to speak of himwase”展现出更自然的语法结构。对比深度循环神经网络中的 LSTM 和 GRU 模型，两者最终困惑度均约为 1.02，但 GRU 训练时间更短（29.83 秒对比 LSTM 的 31.85 秒），这说明 GRU 在保持性能的同时具备更高的计算效率,更适合资源有限的场景。

整个实验中，我深刻体会到循环神经网络的表达能力与模型复杂度的平衡至关重要。从简单 RNN 到 LSTM、GRU，再到深层网络，每一步改进都伴随着对序列建模本质的深入理解。例如，梯度裁剪技术有效解决了训练中的梯度爆炸问题，而门控机制则从结构上增强了模型记忆能力。此外，不同模型在训练速度和性能上的差异，也让我认识到需根据具体任务需求选择合适的网络架构。通过本次实验，我不仅掌握了循环神经网络的核心原理与实现方法，也对时序数据建模的挑战与优化策略有了更全面的认识，这些经验将为后续处理自然语言生成、时间序列预测等任务奠定坚实基础。