6.4 循环神经网络的从零开始实现

在本节中，我们将从零开始实现一个基于字符级循环神经网络的语言模型，并在周杰伦专辑歌词数据集上训练一个模型来进行歌词创作。首先，我们读取周杰伦专辑歌词数据集：

```
import time
import math
import numpy as np
import torch
from torch import nn, optim
import torch.nn.functional as F

import sys
sys.path.append("..")
import d2lzh_pytorch as d2l
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

(corpus_indices, char_to_idx, idx_to_char, vocab_size) = d2l.load_data_jay_lyrics()
```

6.4.1 one-hot向量

为了将词表示成向量输入到神经网络，一个简单的办法是使用one-hot向量。假设词典中不同字符的数量为N（即词典大小 `vocab_size` ），每个字符已经同一个从0到$N-1$的连续整数值索引一一对应。如果一个字符的索引是整数i, 那么我们创建一个全0的长为N的向量，并将其位置为i的元素设成1。该向量就是对原字符的one-hot向量。下面分别展示了索引为0和2的one-hot向量，向量长度等于词典大小。

> **pytorch没有自带one-hot函数(新版好像有了)，下面自己实现一个**

```
def one_hot(x, n_class, dtype=torch.float32):
    # X shape: (batch), output shape: (batch, n_class)
    x = x.long()
    res = torch.zeros(x.shape[0], n_class, dtype=dtype, device=x.device)
    res.scatter_(1, x.view(-1, 1), 1)
    return res

x = torch.tensor([0, 2])
one_hot(x, vocab_size)
```

我们每次采样的小批量的形状是(批量大小, 时间步数)。下面的函数将这样的小批量变换成数个可以输入进网络的形状为(批量大小, 词典大小)的矩阵，矩阵个数等于时间步数。也就是说，时间步t的输入为$\boldsymbol{X}_t \in \mathbb{R}^{n \times d}$，其中$n$为批量大小，$d$为输入个数，即one-hot向量长度（词典大小）。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def to_onehot(X, n_class):
    # X shape: (batch, seq_len), output: seq_len elements of (batch, n_class)
    return [one_hot(X[:, i], n_class) for i in range(X.shape[1])]

X = torch.arange(10).view(2, 5)
inputs = to_onehot(X, vocab_size)
print(len(inputs), inputs[0].shape)
```

输出：

```
5 torch.Size([2, 1027])
```

6.4.2 初始化模型参数

接下来，我们初始化模型参数。隐藏单元个数 `num_hiddens` 是一个超参数。

```
num_inputs, num_hiddens, num_outputs = vocab_size, 256, vocab_size
print('will use', device)

def get_params():
    def _one(shape):
        ts = torch.tensor(np.random.normal(0, 0.01, size=shape), device=device, dtype
        return torch.nn.Parameter(ts, requires_grad=True)

    # 隐藏层参数
    W_xh = _one((num_inputs, num_hiddens))
    W_hh = _one((num_hiddens, num_hiddens))
    b_h = torch.nn.Parameter(torch.zeros(num_hiddens, device=device, requires_grad=T
    # 输出层参数
    W_hq = _one((num_hiddens, num_outputs))
```

```
    b_q = torch.nn.Parameter(torch.zeros(num_outputs, device=device, requires_grad=T
    return nn.ParameterList([W_xh, W_hh, b_h, W_hq, b_q])
```

6.4.3 定义模型

我们根据循环神经网络的计算表达式实现该模型。首先定义 `init_rnn_state` 函数来返回初始化的隐藏状态。它返回由一个形状为(批量大小, 隐藏单元个数)的值为0的 `NDArray` 组成的元组。使用元组是为了更便于处理隐藏状态含有多个 `NDArray` 的情况。

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

下面的 `rnn` 函数定义了在一个时间步里如何计算隐藏状态和输出。这里的激活函数使用了tanh函数。3.8节（多层感知机）中介绍过，当元素在实数域上均匀分布时，tanh函数值的均值为0。

```
def rnn(inputs, state, params):
    # inputs和outputs皆为num_steps个形状为(batch_size, vocab_size)的矩阵
    W_xh, W_hh, b_h, W_hq, b_q = params
    H, = state
    outputs = []
    for X in inputs:
        H = torch.tanh(torch.matmul(X, W_xh) + torch.matmul(H, W_hh) + b_h)
        Y = torch.matmul(H, W_hq) + b_q
        outputs.append(Y)
    return outputs, (H,)
```

做个简单的测试来观察输出结果的个数（时间步数），以及第一个时间步的输出层输出的形状和隐藏状态的形状。

```
state = init_rnn_state(X.shape[0], num_hiddens, device)
inputs = to_onehot(X.to(device), vocab_size)
params = get_params()
outputs, state_new = rnn(inputs, state, params)
print(len(outputs), outputs[0].shape, state_new[0].shape)
```

输出：

```
5 torch.Size([2, 1027]) torch.Size([2, 256])
```

6.4.4 定义预测函数

以下函数基于前缀 `prefix` （含有数个字符的字符串）来预测接下来的 `num_chars` 个字符。这个函数稍显复杂，其中我们将循环神经单元 `rnn` 设置成了函数参数，这样在后面小节介绍其他循环神经网络时能重复使用这个函数。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def predict_rnn(prefix, num_chars, rnn, params, init_rnn_state,
                num_hiddens, vocab_size, device, idx_to_char, char_to_idx):
    state = init_rnn_state(1, num_hiddens, device)
    output = [char_to_idx[prefix[0]]]
    for t in range(num_chars + len(prefix) - 1):
        # 将上一时间步的输出作为当前时间步的输入
        X = to_onehot(torch.tensor([[output[-1]]], device=device), vocab_size)
        # 计算输出和更新隐藏状态
        (Y, state) = rnn(X, state, params)
        # 下一个时间步的输入是prefix里的字符或者当前的最佳预测字符
        if t < len(prefix) - 1:
            output.append(char_to_idx[prefix[t + 1]])
        else:
            output.append(int(Y[0].argmax(dim=1).item()))
    return ''.join([idx_to_char[i] for i in output])
```

我们先测试一下 `predict_rnn` 函数。我们将根据前缀“分开”创作长度为10个字符（不考虑前缀长度）的一段歌词。因为模型参数为随机值，所以预测结果也是随机的。

```
predict_rnn('分开', 10, rnn, params, init_rnn_state, num_hiddens, vocab_size,
            device, idx_to_char, char_to_idx)
```

输出：

```
'分开西圈绪升王凝瓜必客映'
```

6.4.5 裁剪梯度

循环神经网络中较容易出现梯度衰减或梯度爆炸。我们会在6.6节（通过时间反向传播）中解释原因。为了应对梯度爆炸，我们可以裁剪梯度（clip gradient）。假设我们把所有模型参数梯度的元素拼接成一个向量$\boldsymbol{g}$，并设裁剪的阈值是θ。裁剪后的梯度

$$\min\left(\frac{\theta}{\|\boldsymbol{g}\|}, 1\right)\boldsymbol{g}$$

的L_2范数不超过θ。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def grad_clipping(params, theta, device):
    norm = torch.tensor([0.0], device=device)
    for param in params:
        norm += (param.grad.data ** 2).sum()
    norm = norm.sqrt().item()
    if norm > theta:
        for param in params:
            param.grad.data *= (theta / norm)
```

6.4.6 困惑度

我们通常使用困惑度（perplexity）来评价语言模型的好坏。回忆一下3.4节（softmax回归）中交叉熵损失函数的定义。困惑度是对交叉熵损失函数做指数运算后得到的值。特别地，

- 最佳情况下，模型总是把标签类别的概率预测为1，此时困惑度为1；
- 最坏情况下，模型总是把标签类别的概率预测为0，此时困惑度为正无穷；
- 基线情况下，模型总是预测所有类别的概率都相同，此时困惑度为类别个数。

显然，任何一个有效模型的困惑度必须小于类别个数。在本例中，困惑度必须小于词典大小`vocab_size`。

6.4.7 定义模型训练函数

跟之前章节的模型训练函数相比，这里的模型训练函数有以下几点不同：

1. 使用困惑度评价模型。
2. 在迭代模型参数前裁剪梯度。
3. 对时序数据采用不同采样方法将导致隐藏状态初始化的不同。相关讨论可参考6.3节（语言模型数据集（周杰伦专辑歌词））。

另外，考虑到后面将介绍的其他循环神经网络，为了更通用，这里的函数实现更长一些。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def train_and_predict_rnn(rnn, get_params, init_rnn_state, num_hiddens,
                          vocab_size, device, corpus_indices, idx_to_char,
                          char_to_idx, is_random_iter, num_epochs, num_steps,
                          lr, clipping_theta, batch_size, pred_period,
                          pred_len, prefixes):
    if is_random_iter:
        data_iter_fn = d2l.data_iter_random
    else:
        data_iter_fn = d2l.data_iter_consecutive
    params = get_params()
    loss = nn.CrossEntropyLoss()

    for epoch in range(num_epochs):
        if not is_random_iter:  # 如使用相邻采样，在epoch开始时初始化隐藏状态
            state = init_rnn_state(batch_size, num_hiddens, device)
        l_sum, n, start = 0.0, 0, time.time()
        data_iter = data_iter_fn(corpus_indices, batch_size, num_steps, device)
        for X, Y in data_iter:
            if is_random_iter:  # 如使用随机采样，在每个小批量更新前初始化隐藏状态
                state = init_rnn_state(batch_size, num_hiddens, device)
            else:
            # 否则需要使用detach函数从计算图分离隐藏状态，这是为了
            # 使模型参数的梯度计算只依赖一次迭代读取的小批量序列(防止梯度计算开销太大)
                for s in state:
                    s.detach_()

            inputs = to_onehot(X, vocab_size)
            # outputs有num_steps个形状为(batch_size, vocab_size)的矩阵
            (outputs, state) = rnn(inputs, state, params)
            # 拼接之后形状为(num_steps * batch_size, vocab_size)
```

```
            outputs = torch.cat(outputs, dim=0)
            # Y的形状是(batch_size, num_steps)，转置后再变成长度为
            # batch * num_steps 的向量，这样跟输出的行一一对应
            y = torch.transpose(Y, 0, 1).contiguous().view(-1)
            # 使用交叉熵损失计算平均分类误差
            l = loss(outputs, y.long())

            # 梯度清0
            if params[0].grad is not None:
                for param in params:
                    param.grad.data.zero_()
            l.backward()
            grad_clipping(params, clipping_theta, device)  # 裁剪梯度
            d2l.sgd(params, lr, 1)  # 因为误差已经取过均值，梯度不用再做平均
            l_sum += l.item() * y.shape[0]
            n += y.shape[0]

        if (epoch + 1) % pred_period == 0:
            print('epoch %d, perplexity %f, time %.2f sec' % (
                epoch + 1, math.exp(l_sum / n), time.time() - start))
            for prefix in prefixes:
                print(' -', predict_rnn(prefix, pred_len, rnn, params, init_rnn_state
                    num_hiddens, vocab_size, device, idx_to_char, char_to_idx))
```

6.4.8 训练模型并创作歌词

现在我们可以训练模型了。首先，设置模型超参数。我们将根据前缀“分开”和“不分开”分别创作长度为50个字符（不考虑前缀长度）的一段歌词。我们每过50个迭代周期便根据当前训练的模型创作一段歌词。

```
num_epochs, num_steps, batch_size, lr, clipping_theta = 250, 35, 32, 1e2, 1e-2
pred_period, pred_len, prefixes = 50, 50, ['分开', '不分开']
```

下面采用随机采样训练模型并创作歌词。

```
train_and_predict_rnn(rnn, get_params, init_rnn_state, num_hiddens,
                      vocab_size, device, corpus_indices, idx_to_char,
                      char_to_idx, True, num_epochs, num_steps, lr,
```

```
                      clipping_theta, batch_size, pred_period, pred_len,
                      prefixes)
```

输出：

```
epoch 50, perplexity 70.039647, time 0.11 sec
 - 分开 我不要再想 我不能 想你的让我 我的可 你怎么 一颗四 一颗四 我不要 一颗两 一颗四 一颗四
 - 不分开 我不要再 你你的外 在人  别你的让我 狂的可 语人两 我不要 一颗两 一颗四 一颗四 我不
epoch 100, perplexity 9.726828, time 0.12 sec
 - 分开 一直的美栈人 一起看 我不要好生活 你知不觉 我已好好生活 我知道好生活 后知不觉 我跟了
 - 不分开堡 我不要再想 我不 我不 我不要再想你 不知不觉 你已经离开我 不知不觉 我跟了好生活 我
epoch 150, perplexity 2.864874, time 0.11 sec
 - 分开 一只会停留 有不它元羞 这蝎什么奇怪的事都有 包括像猫的狗 印地安老斑鸠 平常话不多 除非
 - 不分开扫 我不你再想 我不能再想 我不 我不 我不要再想你 不知不觉 你已经离开我 不知不觉 我跟
epoch 200, perplexity 1.597790, time 0.11 sec
 - 分开 有杰伦 干 载颗拳满的让空美空主 相爱还有个人 再狠狠忘记 你爱过我的证  有晶莹的手滴 让
 - 不分开扫 我叫你爸 你打我妈 这样对吗干嘛这样 何必让它牵鼻子走 瞎 说底牵打我妈要 难道球耳 
epoch 250, perplexity 1.303903, time 0.12 sec
 - 分开 有杰人开留 仙唱它怕羞 蜥蜴横著走 这里什么奇怪的事都有 包括像猫的狗 印地安老斑鸠 平常
 - 不分开简 我不能再想 我不 我不 我不能 爱情走的太快就像龙卷风 不能承受我已无处可躲 我不要再
```

接下来采用相邻采样训练模型并创作歌词。

```
train_and_predict_rnn(rnn, get_params, init_rnn_state, num_hiddens,
                      vocab_size, device, corpus_indices, idx_to_char,
                      char_to_idx, False, num_epochs, num_steps, lr,
                      clipping_theta, batch_size, pred_period, pred_len,
                      prefixes)
```

输出：

Copy to clipboard

```
epoch 50, perplexity 59.514416, time 0.11 sec
 - 分开 我想要这 我想了空 我想了空 我想了空 我想了空 我想了空 我想了空 我想了空 我想了空 我
 - 不分开 我不要这 全使了双 我想了这 我想了空 我想了空 我想了空 我想了空 我想了空 我想了空 
epoch 100, perplexity 6.801417, time 0.11 sec
 - 分开 我说的这样笑 想你都 不着我 我想就这样牵 你你的回不笑多难的  它在云实 有一条事 全你了
 - 不分开觉 你已经离开我 不知不觉 我跟好这节活 我该好好生活 不知不觉 你跟了离开我 不知不觉 我
```

```
epoch 150, perplexity 2.063730, time 0.16 sec
 - 分开 我有到这样牵着你的手不放开 爱可不可以简简单单没有伤  古有你烦 我有多烦恼向 你知带悄
 - 不分开觉 你已经很个我 不知不觉 我跟了这节奏 后知后觉 又过了一个秋 后哼哈兮 快使用双截棍 
epoch 200, perplexity 1.300031, time 0.11 sec
 - 分开 我想要这样牵着你的手不放开 爱能不能够永远单甜没有伤害 你 靠着我的肩膀 你 在我胸口睡
 - 不分开觉 你已经离开我 不知不觉 我跟了这节奏 后知后觉 又过了一个秋 后知后觉 我该好好生活 我
epoch 250, perplexity 1.164455, time 0.11 sec
 - 分开 我有一这样布 对你依依不舍 连隔壁邻居都猜到我现在的感受 河边的风 在吹着头发飘动 牵着你
 - 不分开觉 你已经离开我 不知不觉 我跟了这节奏 后知后觉 又过了一个秋 后知后觉 我该好好生活 我
```

小结

- 可以用基于字符级循环神经网络的语言模型来生成文本序列，例如创作歌词。
- 当训练循环神经网络时，为了应对梯度爆炸，可以裁剪梯度。
- 困惑度是对交叉熵损失函数做指数运算后得到的值。