6.5 循环神经网络的简洁实现

本节将使用PyTorch来更简洁地实现基于循环神经网络的语言模型。首先，我们读取周杰伦专辑歌词数据集。

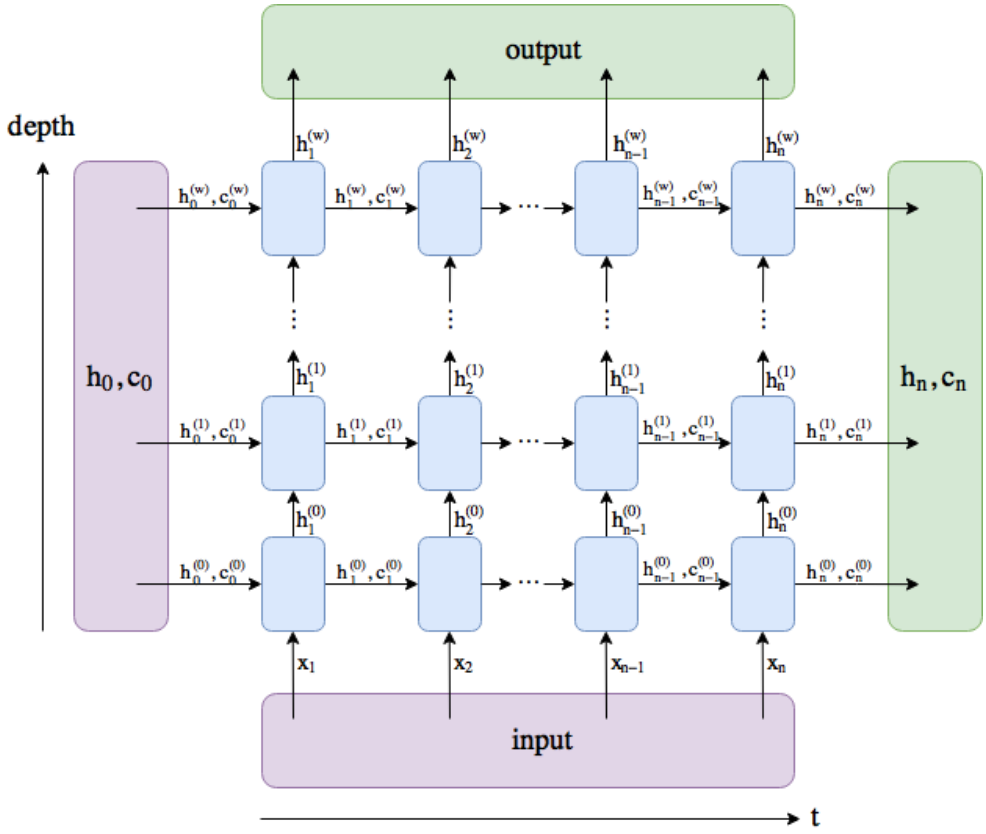
```
import time
import math
import numpy as np
import torch
from torch import nn, optim
import torch.nn.functional as F

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()
```

6.5.1 定义模型

PyTorch中的 `nn` 模块提供了循环神经网络的实现。下面构造一个含单隐藏层、隐藏单元个数为256的循环神经网络层 `rnn_layer` 。

```
num_hiddens = 256
# rnn_layer = nn.LSTM(input_size=vocab_size, hidden_size=num_hiddens) # 已测试
rnn_layer = nn.RNN(input_size=vocab_size, hidden_size=num_hiddens)
```

与上一节中实现的循环神经网络不同，这里 `rnn_layer` 的输入形状为(时间步数, 批量大小, 输入个数)。其中输入个数即one-hot向量长度（词典大小）。此外， `rnn_layer` 作为 `nn.RNN` 实例，在前向计算后会分别返回输出和隐藏状态h，其中输出指的是隐藏层在**各个时间步**上计算并输出的隐藏状态，它们通常作为后续输出层的输入。需要强调的是，该“输出”本身并不涉及输出层计算，形状为(时间步数, 批量大小, 隐藏单元个数)。而 `nn.RNN` 实例在前向计算返回的隐藏状态指的是隐藏层在**最后时间步**的隐藏状态：当隐藏层有多层时，每一层的隐藏状态都会记录在该变量中；对于像长短期记忆（LSTM），隐藏状态是一个元组(h, c)，即hidden state和cell state。我们会在本章的后面介绍长短期记忆和深度循环神经网络。关于循环神经网络（以LSTM为例）的输出，可以参考下图（**图片来源**）。



循环神经网络（以LSTM为例）的输出

来看看我们的例子，输出形状为(时间步数, 批量大小, 隐藏单元个数)，隐藏状态h的形状为(层数, 批量大小, 隐藏单元个数)。

```
num_steps = 35
batch_size = 2
state = None
X = torch.rand(num_steps, batch_size, vocab_size)
Y, state_new = rnn_layer(X, state)
print(Y.shape, len(state_new), state_new[0].shape)
```

输出：

```
torch.Size([35, 2, 256]) 1 torch.Size([2, 256])
```

> **如果 `rnn_layer` 是 `nn.LSTM` 实例，那么上面的输出是什么?**

接下来我们继承 `Module` 类来定义一个完整的循环神经网络。它首先将输入数据使用one-hot向量表示后输入到 `rnn_layer` 中，然后使用全连接输出层得到输出。输出个数等于词典大小 `vocab_size` 。

```
# 本类已保存在d2lzh_pytorch包中方便以后使用
class RNNModel(nn.Module):
    def __init__(self, rnn_layer, vocab_size):
        super(RNNModel, self).__init__()
        self.rnn = rnn_layer
        self.hidden_size = rnn_layer.hidden_size * (2 if rnn_layer.bidirectional else
        self.vocab_size = vocab_size
        self.dense = nn.Linear(self.hidden_size, vocab_size)
        self.state = None

    def forward(self, inputs, state): # inputs: (batch, seq_len)
        # 获取one-hot向量表示
        X = d2l.to_onehot(inputs, self.vocab_size) # X是个list
        Y, self.state = self.rnn(torch.stack(X), state)
        # 全连接层会首先将Y的形状变成(num_steps * batch_size, num_hiddens)，它的输出
        # 形状为(num_steps * batch_size, vocab_size)
        output = self.dense(Y.view(-1, Y.shape[-1]))
        return output, self.state
```

6.5.2 训练模型

同上一节一样，下面定义一个预测函数。这里的实现区别在于前向计算和初始化隐藏状态的函数接口。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def predict_rnn_pytorch(prefix, num_chars, model, vocab_size, device, idx_to_char,
                        char_to_idx):
    state = None
    output = [char_to_idx[prefix[0]]] # output会记录prefix加上输出
    for t in range(num_chars + len(prefix) - 1):
        X = torch.tensor([output[-1]], device=device).view(1, 1)
        if state is not None:
            if isinstance(state, tuple): # LSTM, state:(h, c)
                state = (state[0].to(device), state[1].to(device))
            else:
                state = state.to(device)

        (Y, state) = model(X, state)
        if t < len(prefix) - 1:
```

```
            output.append(char_to_idx[prefix[t + 1]])
        else:
            output.append(int(Y.argmax(dim=1).item()))
    return ''.join([idx_to_char[i] for i in output])
```

让我们使用权重为随机值的模型来预测一次。

```
model = RNNModel(rnn_layer, vocab_size).to(device)
predict_rnn_pytorch('分开', 10, model, vocab_size, device, idx_to_char, char_to_idx)
```

输出：

```
'分开戏想暖迎凉想征凉征征'
```

接下来实现训练函数。算法同上一节的一样，但这里只使用了相邻采样来读取数据。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def train_and_predict_rnn_pytorch(model, num_hiddens, vocab_size, device,
                                  corpus_indices, idx_to_char, char_to_idx,
                                  num_epochs, num_steps, lr, clipping_theta,
                                  batch_size, pred_period, pred_len, prefixes):
    loss = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    model.to(device)
    state = None
    for epoch in range(num_epochs):
        l_sum, n, start = 0.0, 0, time.time()
        data_iter = d2l.data_iter_consecutive(corpus_indices, batch_size, num_steps,
        for X, Y in data_iter:
            if state is not None:
                # 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)
                if isinstance (state, tuple): # LSTM, state:(h, c)
                    state = (state[0].detach(), state[1].detach())
                else:
                    state = state.detach()
```

```
            (output, state) = model(X, state) # output: 形状为(num_steps * batch_size

            # 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)
            l = loss(output, y.long())

            optimizer.zero_grad()
            l.backward()
            # 梯度裁剪
            d2l.grad_clipping(model.parameters(), clipping_theta, device)
            optimizer.step()
            l_sum += l.item() * y.shape[0]
            n += y.shape[0]

        try:
            perplexity = math.exp(l_sum / n)
        except OverflowError:
            perplexity = float('inf')
        if (epoch + 1) % pred_period == 0:
            print('epoch %d, perplexity %f, time %.2f sec' % (
                epoch + 1, perplexity, time.time() - start))
            for prefix in prefixes:
                print(' -', predict_rnn_pytorch(
                    prefix, pred_len, model, vocab_size, device, idx_to_char,
                    char_to_idx))
```

使用和上一节实验中一样的超参数（除了学习率）来训练模型。

```
num_epochs, batch_size, lr, clipping_theta = 250, 32, 1e-3, 1e-2 # 注意这里的学习率设置
pred_period, pred_len, prefixes = 50, 50, ['分开', '不分开']
train_and_predict_rnn_pytorch(model, num_hiddens, vocab_size, device,
                              corpus_indices, idx_to_char, char_to_idx,
                              num_epochs, num_steps, lr, clipping_theta,
                              batch_size, pred_period, pred_len, prefixes)
```

输出：

Copy to clipboard

```
epoch 50, perplexity 10.658418, time 0.05 sec
 - 分开始我妈  想要你 我不多 让我心到的 我妈妈 我不能再想 我不多再想 我不要再想 我不多再想 我
 - 不分开 我想要你不你 我 你不要 让我心到的 我妈人 可爱女人 坏坏的让我疯狂的可爱女人 坏坏的让
epoch 100, perplexity 1.308539, time 0.05 sec
 - 分开不会痛 不要 你在黑色幽默 开始了美丽全脸的梦滴 闪烁成回忆 伤人的美丽 你的完美主义 太彻
 - 不分开不是我不要再想你 我不能这样牵着你的手不放开 爱可不可以简简单单没有伤害 你 靠着我的肩
epoch 150, perplexity 1.070370, time 0.05 sec
 - 分开不能去河南嵩山 学少林跟武当 快使用双截棍 哼哼哈兮 快使用双截棍 哼哼哈兮 习武之人切记
 - 不分开 在我会想通 是谁开没有全有开始 他心今天 一切人看 我 一口令秋软语的姑娘缓缓走过外滩
epoch 200, perplexity 1.034663, time 0.05 sec
 - 分开不能去吗周杰伦 才离 没要你在一场悲剧 我的完美主义 太彻底 分手的话像语言暴力 我已无能为
 - 不分开 让我面到你 爱情来的太快就像龙卷风 离不开暴风圈来不及逃 我不能再想 我不能再想 我不
epoch 250, perplexity 1.021437, time 0.05 sec
 - 分开 我我外的家边 你知道这 我爱不看的太  我想一个又重来不以 迷已文一只剩下回忆 让我叫带你
 - 不分开 我我想想和 是你听没不  我不能不想  不知不觉 你已经离开我 不知不觉 我跟了这节奏 后
```

小结

- PyTorch的 `nn` 模块提供了循环神经网络层的实现。
- PyTorch的 `nn.RNN` 实例在前向计算后会分别返回输出和隐藏状态。该前向计算并不涉及输出层计算。