RNN

循环神经网络

太难了，感觉要听三遍，今天先把代码Debug捋顺了一点点。

潜变量自回归模型

以下公式块中的函数映射省略了参数和噪声。

$$
\begin{gathered}
\text{潜变量}H_T\text{与}H_{T-1}\text{和}X_{T-1}\text{相关} \\
\text{潜变量}X_T\text{与}X_{T-1}\text{和}H_T\text{相关} \\
H_T = f(H_{T-1}, X_{T-1}) \\
X_T = g(X_{T-1}, H_T)
\end{gathered}
$$

说人话：潜变量相关与前一个潜变量和前一个输入变量，输入变量相关与前一个输入变量和前一个输入变量推出的潜变量相关。

循环神经网络

$$
\begin{gathered}
X_T\text{是观察数据、}H_T\text{是潜变量、}O_T\text{是输出} \\
\text{潜变量}H_T\text{与}X_{T-1}\text{相关} \\
\text{输出}O_T\text{与}H_T\text{相关} \\
H_T = f(X_{T-1}) \\
O_T = g(H_T)
\end{gathered}
$$

困惑度

$$
\exp\left(-\frac{1}{n}\sum_{t=1}^{n}\log P(x_t \mid x_{t-1}, \ldots, x_1)\right).
$$

所谓的困惑度就是交叉熵加了个指数运算，AI的很多名词其实就是把以前因为算力不足而没实现的技术换了个高大上的名字。

MLP - DL、Weighted Average - Self-Attention、Seq2Seq Models - Transformer、Adversarial Learning - GAN

困惑度1代表完美，无穷大为最糟糕。

梯度裁剪

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

防止梯度爆炸。

独热编码函数one_hot

字面意思，独热：**唯一热，只有一个位置是1。**

将分类变量转换为数值向量的方法。

每个向量中只有一个元素为1，其余为0。

```
F.one_hot(torch.tensor([0,2]),len(vocab))
```

输出为tensor([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0]])

为什么要有隐状态

不深究，**别管**。它很好，我还没到研究理论的时候。

Traditional RNN Code

```
import math
import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l

# Tradional RNN
batch_size, num_steps = 32, 35
# 训练迭代器和词表
train_iter, vocab = d2l.load_data_time_machine(batch_size, num_steps)

print('vocab:',len(vocab))
print('hot_vector:',F.one_hot(torch.tensor([0,2]),len(vocab)))

'''
Demo of one_hot function
[0,1,2,3,4,5,6,7,8,9] -> [[0,1,2,3,4],[5,6,7,8,9]] ->
([[0, 5],[1, 6],[2, 7],[3, 8],[4, 9]]) ->
([[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]],)
......
]]])(太长了，仅演示了[0, 5]的one_hot)
'''
X = torch.arange(10).reshape((2, 5))
print(X)
print('X_one_hot',F.one_hot(X.T, 28))

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

    W_xh = normal((num_inputs, num_hiddens)) # input to hidden weights
    W_hh = normal((num_hiddens, num_hiddens)) # hidden to hidden weights
```

```
    b_h = torch.zeros(num_hiddens, device=device) # hidden bias
    W_hq = normal((num_hiddens, num_outputs)) # hidden to output weights
    b_q = torch.zeros(num_outputs, device=device) # output bias
    params = [W_xh, W_hh, b_h, W_hq, b_q]
    for param in params:
        param.requires_grad_(True)
    return params

# hidden state initialization
def init_rnn_state(batch_size, num_hiddens, device):
    return (torch.zeros((batch_size, num_hiddens), device=device), )

# hidden and output in a time step
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 # params已经可以进行梯度计算了
    # 通过写H, = state 会使 H 被解包为一个张量，而不是一个元组。
    H, = state
    outputs = []
    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,)

class RNNModelScratch:
    # 这里的参数和函数前面都定义了自己对应
    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 = F.one_hot(X.T, self.vocab_size).type(torch.float32)
        # 此时的X的形状是(时间步数，批量大小，词表大小)
        return self.forward_fn(X, state, self.params)

    # 初始化隐藏状态
    def begin_state(self, batch_size, device):
        return self.init_state(batch_size, self.num_hiddens, device)

# Testing the model
num_hiddens = 512
# 词汇表大小、隐藏单元数、设备、参数列表、初始化隐藏状态函数和前向传播函数
net = RNNModelScratch(len(vocab), num_hiddens, d2l.try_gpu(), get_params,
                      init_rnn_state, rnn)
state = net.begin_state(X.shape[0], d2l.try_gpu())
Y, new_state = net(X.to(d2l.try_gpu()), state)
print(Y.shape, len(new_state), new_state[0].shape)

# prefix为前缀字符串，num_preds为后续预测字符数
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])
print(predict_ch8('i like studying', 10, net, vocab, d2l.try_gpu()))

# 梯度裁剪，防止超过阈值导致梯度爆炸
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
    # 沐神的源代码返回的是int而不是tensor，也可能是我弄巧成拙，原码在下行
    # norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in params))
    grads = [p.grad for p in params if p.grad is not None]
    if not grads:
        return
    norm = torch.sqrt(torch.stack([torch.sum(grad ** 2) for grad in grads]).sum())
    if norm > theta:
        for param in params:
            param.grad[:] *= theta / norm

def train_epoch_ch8(net, train_iter, loss, updater, device, use_random_iter):
    state, timer = None, d2l.Timer()
    metric = d2l.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:
            state = net.begin_state(batch_size=X.shape[0], device=device)
        else: # 分离张量，使用历史状态，防止梯度传递
            if isinstance(net, nn.Module) and not isinstance(state, tuple):
                state.detach_()
            else:
                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()
        # 如果不使用PyTorch的优化器，则需要使用上文的梯度裁剪
        if isinstance(updater, torch.optim.Optimizer):
            updater.zero_grad()
            l.backward()
            grad_clipping(net, 1)
            updater.step()
        else:
            l.backward()
            grad_clipping(net, 1)
            updater(batch_size=1)
        metric.add(l * y.numel(), y.numel())
    return math.exp(metric[0] / metric[1]), metric[1] / timer.stop()

def train_ch8(net: object, train_iter: object, vocab: object, lr: object, num_epochs:
object, device: object,
              use_random_iter: object = False) -> object:
    loss = nn.CrossEntropyLoss()
    animator = d2l.Animator(xlabel='epoch', ylabel='perplexity',
                            legend=['train'], xlim=[10, num_epochs])
```

```
    if isinstance(net, nn.Module):
        updater = torch.optim.SGD(net.parameters(), lr)
    else:
        updater = lambda batch_size: d2l.sgd(net.params, lr, batch_size)
        predict = lambda prefix: predict_ch8(prefix, 50, net, vocab, device)
    for epoch in range(num_epochs):
        ppl, speed = train_epoch_ch8(
            net, train_iter, loss, updater, device, use_random_iter)
        if (epoch + 1) % 10 == 0:
            print(predict('time traveller'))
            animator.add(epoch + 1, [ppl])
    print(f'困惑度 {ppl:.1f}, {speed:.1f} 词元/秒 {str(device)}')
    print(predict('time traveller'))
    print(predict('traveller'))

num_epochs, lr = 500, 1
train_ch8(net, train_iter, vocab, lr, num_epochs, d2l.try_gpu())
d2l.plt.show()

# 随机抽样训练
train_ch8(net, train_iter, vocab, lr, num_epochs, d2l.try_gpu(),
use_random_iter=True)
d2l.plt.show()

# Pytorch RNN
import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l

batch_size, num_steps = 32, 35
train_iter, vocab = d2l.load_data_time_machine(batch_size, num_steps)
num_hiddens = 256

rnn_layer = nn.RNN(len(vocab), num_hiddens)
state = torch.zeros((1, batch_size, num_hiddens))
print(state.shape)

X = torch.rand(size=(num_steps, batch_size, len(vocab)))
Y, state_new = rnn_layer(X, state)
print(Y.shape, state_new.shape)
```

Pytorch RNN Code

```
import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l

batch_size, num_steps = 32, 35
train_iter, vocab = d2l.load_data_time_machine(batch_size, num_steps)
num_hiddens = 256

rnn_layer = nn.RNN(len(vocab), num_hiddens)
state = torch.zeros((1, batch_size, num_hiddens))
print(state.shape)
```

```
X = torch.rand(size=(num_steps, batch_size, len(vocab)))
Y, state_new = rnn_layer(X, state)
print(Y.shape, state_new.shape)

class RNNModel(nn.Module):
    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
        # 判断网络是否双向
        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)
        X = X.to(torch.float32)
        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):
        if not isinstance(self.rnn, nn.LSTM):# 还没学LSTM，先扔这，过几天学完再回头看
            return torch.zeros((self.num_directions * self.rnn.num_layers,
                                batch_size, self.num_hiddens),
                               device=device)
        else:
            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))

device = d2l.try_gpu()
net = RNNModel(rnn_layer, vocab_size=len(vocab))
net = net.to(device)
d2l.predict_ch8('time traveller', 10, net, vocab, device)
num_epochs, lr = 500, 1
d2l.train_ch8(net, train_iter, vocab, lr, num_epochs, device)
d2l.plt.show()
```