LSTM

长短期记忆网络的设计灵感来自于计算机的逻辑门。

长短期记忆网络引入了**记忆元（memory cell），又称为细胞单元（cell）**。

总共有三个门：忘记门、输入门、输出门。

一个时间步内LSTM发生了什么事？接下来是**我的理解**。

细胞单元乘坐3号线（日记本），前一时刻的隐藏状态与新输入乘坐1号线。

1号线双人组发生的事：

双人组首先到忘记门站，它们复制一份自己在忘记门换乘站经过Sigmoid函数处理成为了一个[0,1]的权重去3号线等待。

接下来，双人组在1号线到达输入门站。输入门站由Sigmoid函数和tanh函数组成。它们需要复制出两份自己去参加两个函数站。

Sigmoid函数的输出决定了哪些信息需要被更新或添加，而tanh函数则生成候选的新信息量。

在Sigmoid函数站中，Sigmoid函数将双人组结合起来，生成一个介于[0,1]之间的权重。这个权重用于控制需要更新的信息量。

与此同步，tanh函数会生成一组候选信息，这些信息代表着当前时刻的潜在更新内容。

之后，输入门的Sigmoid输出和tanh信息结合，生成新的细胞状态更新去3号线等待。

接下来，双人组的本体到达输出门进行Sigmoid处理生成一个权值，等待新的细胞单元进行tanh处理生成的权值进行相乘，成为新的隐藏状态。

注意是新的隐藏状态，不是新的双人组，等下个时间步的新输入来才能组成双人组。

3号线细胞单元发生的事：

旧的细胞单元首先碰上了忘记门站出来的权重，它们相乘进行了数据稀释成为半新细胞单元。

接着碰上了由输入门出来的新信息量。这时不再相乘而是相加成为全新细胞单元。

全新细胞单元到输出门复制一份自己进行tanh处理和新双人组经过Sigmoid处理。

当前隐藏状态输出不仅是LSTM的最终输出，也是下一时刻输入的隐藏状态，这样就实现了信息在时间上的传递和记忆更新。

这种设计使得LSTM能够在较长时间的序列中有效保留重要信息，同时忘记无关或过时的信息，避免了传统RNN中容易出现的梯度消失问题。

LSTM Code

```
import torch
from torch import nn
from d2l import torch as d2l

# Traditional LSTM
```

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

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))
    '''
    写段注释，方便看，我的大脑内存不是很大，笨就写在旁边方便自己看。
    W_xi：从输入到输入门的权重，相当于地铁类比中，前一时刻的隐藏状态和当前输入信息在1号线上的传递，决定输入信息进入输入门的程度。
    W_hi：从隐藏状态到输入门的权重，决定前一时刻的隐藏状态如何影响输入门，相当于前一站（隐藏状态）乘客换乘1号线到输入门的通道。
    b_i：输入门的偏置项，用于调整输入门的输出，相当于调整1号线的通行规则。
    W_xf：从输入到遗忘门的权重，相当于地铁类比中，前一时刻的隐藏状态和当前输入信息在1号线上的传递，决定信息如何进入遗忘门。
    W_hf：从隐藏状态到遗忘门的权重，决定前一时刻的隐藏状态如何影响遗忘门，相当于前一站（隐藏状态）乘客换乘1号线到遗忘门的通道。
    b_f：遗忘门的偏置项，用于调整遗忘门的输出，相当于调整1号线的通行规则。
    W_xo：从输入到输出门的权重，相当于地铁类比中，前一时刻的隐藏状态和当前输入信息在1号线上的传递，决定信息如何进入输出门。
    W_ho：从隐藏状态到输出门的权重，决定前一时刻的隐藏状态如何影响输出门，相当于前一站（隐藏状态）乘客换乘1号线到输出门的通道。
    b_o：输出门的偏置项，用于调整输出门的输出，相当于调整1号线的通行规则。
    W_xc：从输入到候选记忆细胞的权重，相当于地铁类比中，当前输入信息经过1号线处理后生成候选记忆细胞的通道。
    W_hc：从隐藏状态到候选记忆细胞的权重，影响候选记忆细胞的生成，相当于前一站（隐藏状态）乘客经过1号线生成候选记忆细胞的通道。
    b_c：候选记忆细胞的偏置项，用于调整候选记忆细胞的输出，相当于调整生成候选记忆细胞的通行规则。
    W_hq：从隐藏状态到输出的权重，相当于在3号线上的传递，将更新后的隐藏状态生成最终的输出结果。
    b_q：输出的偏置项，用于调整最终输出的结果，相当于调整3号线的通行规则。
    '''
    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):
    # 返回一个元组，包含两个张量：一个全零张量表示初始的隐藏状态，和一个全零张量表示初始的记忆细胞状态。
    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 - 输入门、F - 遗忘门、O - 输出门、C_tilda - 候选记忆细胞、C - 记忆细胞、H -
隐藏状态、Y - 输出
        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)

vocab_size, num_hiddens, device = len(vocab), 256, d2l.try_gpu()
num_epochs, lr = 500, 1
model = d2l.RNNModelScratch(len(vocab), num_hiddens, device, get_lstm_params,
init_lstm_state, lstm)
d2l.train_ch8(model, train_iter, vocab, lr, num_epochs, device)
d2l.plt.show()

# Pytorch LSTM
num_inputs = vocab_size
lstm_layer = nn.LSTM(num_inputs, num_hiddens)
model = d2l.RNNModel(lstm_layer, len(vocab))
mode = model.to(device)
d2l.train_ch8(model, train_iter, vocab, lr, num_epochs, device)
d2l.plt.show()
```