GRU

门控循环单元

马尔可夫模型和RNN中梯度裁剪有时会舍弃一些比较重要的早期信息，

所以需要一个新机制给这类信息一个较大的梯度保证它们的权重对后续有影响。

相反，有一些信息没啥用，需要一个机制跳过这些无用信息。

简单来说,RNN越往后的信息权重越大，越早的信息权重越小，这样是错误的。

谁规定的晚来的就是正确的了?

方法：**LSTM、GRU正是为了他们而生，GRU是LSTM的优秀变种（儿子）。**

以下理解是ChatGPT基于我对LSTM和地铁线进行类比生成的解释（感觉不如我写的生动）。

首先，前一时刻的隐藏状态和新的输入信息乘坐2号线，他们首先会到达**重置门**。在重置门处，他们通过一个Sigmoid函数的处理，得到一个在[0,1]之间的权重。

这就像乘客们决定要不要“重置”自己，选择忘记或保留一部分信息。如果Sigmoid函数输出0，那么他们会被“重置”成全新的信息；如果输出1，他们就继续保持原样。

接着，乘客们继续前行，这次2号线将他们带到下一站，这里他们遇到了**更新门**。更新门的作用类似于LSTM中的输入门和忘记门的结合。

在这里，前一时刻的隐藏状态和新输入的信息会再次通过Sigmoid函数的处理，得到一个新的权重。这个权重决定了当前的信息应该被保留多少，和应该更新多少。

此时，信息分流成了两条路线。一部分乘客继续乘坐2号线，前往下一站的隐藏状态；另一部分则换乘4号线，他们会和经过重置门处理的信息结合在一起产生新的候选隐藏状态。

最终，更新门的输出决定了4号线上的新信息应该有多少被保留，以及2号线上旧信息应该有多少被遗忘。

所有这些信息汇聚到了一起，形成了GRU的当前隐藏状态，并成为下一时刻的输入。这使得GRU能够有效地在时间序列中传递重要的信息，同时简化了计算流程。

通过这样的设计，GRU相比于LSTM减少了一些门和操作，虽然它的结构更为简洁，但依然保留了在处理长时间依赖问题上的有效性。

门控隐状态

支持隐状态的门控有专门的机制确定应该何时更新隐状态以及合适重置隐状态。

R_t和Z_t都是根据过去的状态H_{t-1}和当前输入X_t计算得到的【0,1】之间的量。

R_t首先与H_{t-1}进行元素积，由于R_t内部都是【0,1】的变量，

因此是对过去的状态H_{t-1}进行一次选择，R_t在某个位置的值越趋近于0，

则表示这个位置的过去信息越倾向于被丢弃，反之保留。

随后与X_t一起构成候选隐藏变量$\tilde{H}_t$。

同样由于R_t的值在【0,1】中，它只会削弱过去的状态，

而不会增强，因此被称为遗忘门（或重置门，重置过去的状态）。

Z_t被称为更新门，因为它控制了隐藏状态的更新。假如Z_t全为1，

则H_t将完全保留上一个时间的状态H_{t-1}；

反之，则全盘采用当前时刻的候选隐藏变量$\tilde{H}_t$。

或许各位会有疑问，感觉R_t已经对过去有所选择，

为何还要加上Z_t多此一举。

个人认为，Z_t实际上是对当前进行选择，根据老师的例子，

如果一个序列中已经有很多的\\猫//，那么再输入猫，实际上对于网络的正收益不大，

可以抛弃，而R_t只能选择过去，

不能抛弃当前，而Z_t可以。

总而言之，GRU通过两个门控网络，根据过去状态和当前输入，

一方面对过去状态进行选择，一方面对当前状态也进行选择。

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z),$$
$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h),$$
$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

GRU Code

```
import math
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)

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

    def normal(shape):
        return torch.randn(shape, device=device) * 0.01
    # sup function to create three sets of parameters
    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_xz：从输入到更新门的权重，相当于在地铁类比中，前一时刻的隐藏状态和当前输入信息在2号线上的传递。
```

W_hz：从隐藏状态到更新门的权重，决定了前一时刻的隐藏状态如何影响当前的更新。
b_z：更新门的偏置项，用于调整更新门的输出。
W_xr：从输入到重置门的权重，相当于在地铁类比中，决定前一时刻隐藏状态和当前输入信息在2号线上的传递情况。
W_hr：从隐藏状态到重置门的权重，决定前一时刻的隐藏状态如何被“重置”。
b_r：重置门的偏置项，用于调整重置门的输出。
W_xh：从输入到候选隐藏状态的权重，相当于重置门处理后的信息换乘4号线并生成候选信息。
W_hh：从隐藏状态到候选隐藏状态的权重，影响候选信息的生成。
b_h：生成候选隐藏状态的偏置项，用于调整候选信息的输出。
W_hq：从隐藏状态到输出的权重，相当于更新后的信息在4号线上的传递，最终生成当前的隐藏状态输出。
b_q：输出的偏置项，用于调整输出的最终结果。
'''

```
    W_xz, W_hz, b_z = three()
    W_xr, W_hr, b_r = three()
    W_xh, W_hh, b_h = three()
    W_hq = normal((num_hiddens, num_outputs))
    b_q = torch.zeros(num_outputs, device=device)
    params = [W_xz, W_hz, b_z, W_xr, W_hr, b_r, W_xh, W_hh, b_h, W_hq, b_q]
    for param in params:
        param.requires_grad_(True)
    return params

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

def gru(inputs, state, params):
    W_xz, W_hz, b_z, W_xr, W_hr, b_r, W_xh, W_hh, b_h, W_hq, b_q = params
    H, = state
    outputs = []
    for X in inputs:
        # @ 这个符号表示矩阵乘法，很多写法
        # Z-更新门 、 R-重置门 、 H_tilda-候选隐藏状态 、 H-隐藏状态 、 Y-输出
        Z = torch.sigmoid((X @ W_xz) + (H @ W_hz) + b_z)
        R = torch.sigmoid((X @ W_xr) + (H @ W_hr) + b_r)
        H_tilda = torch.tanh((X @ W_xh) + ((R * H) @ W_hh) + b_h)
        H = Z * H + (1 - Z) * H_tilda
        Y = H @ W_hq + b_q
        outputs.append(Y)

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

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_params,
                            init_gru_state, gru)
d2l.train_ch8(model, train_iter, vocab, lr, num_epochs, device)
d2l.plt.show()

# PyTorch 实现
num_inputs = vocab_size
gru_layer = nn.GRU(num_inputs, num_hiddens)
model = d2l.RNNModel(gru_layer, len(vocab))
model = model.to(device)
d2l.train_ch8(model, train_iter, vocab, lr, num_epochs, device)
d2l.plt.show()
```