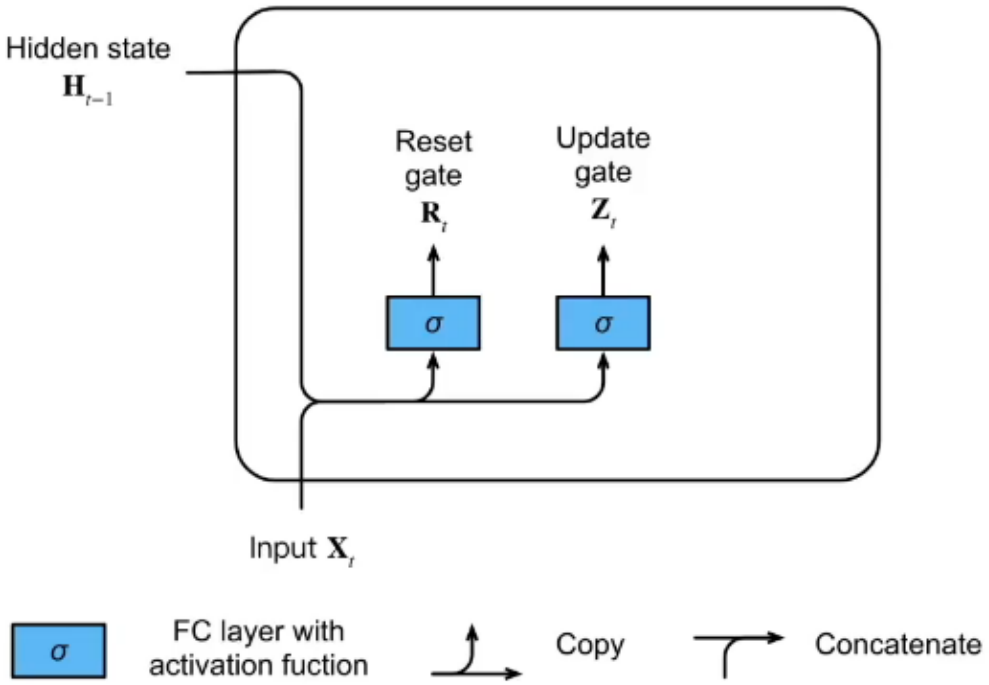
一、门

门

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$



二、隐藏状态如何计算

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$
$$\tilde{H}_t = \tanh(X_t W_{xh} + \left(R_t \odot H_{t-1}\right) W_{hh} + b_h)$$
$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

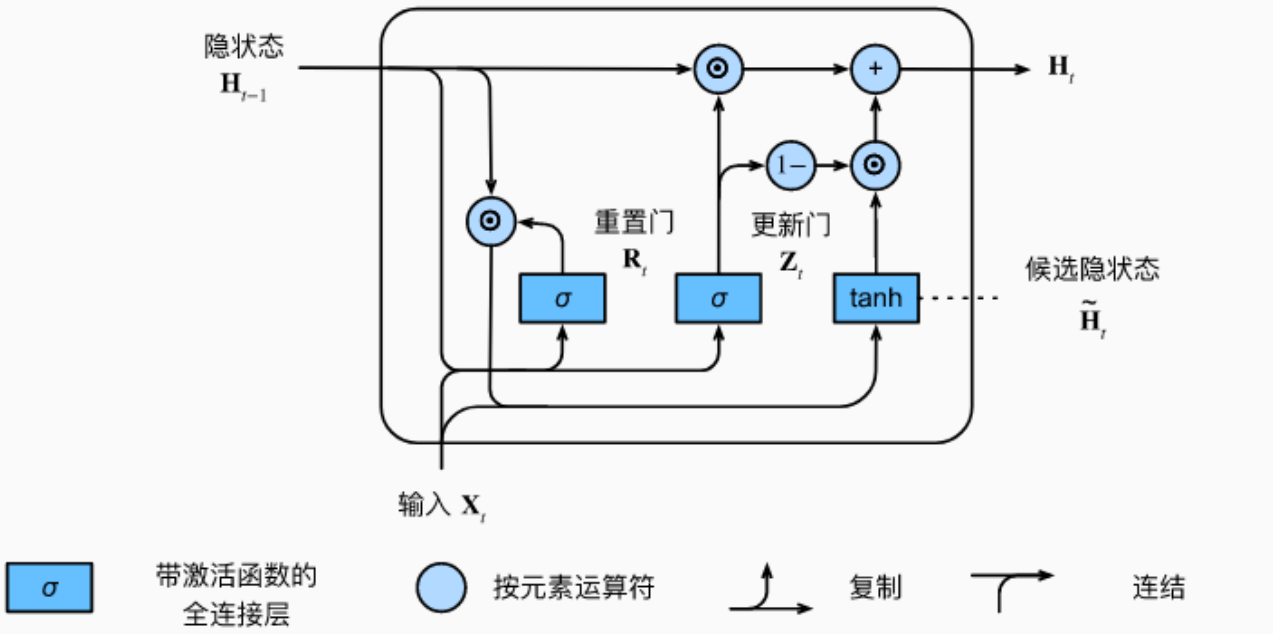


图9.1.3 计算门控循环单元模型中的隐状态

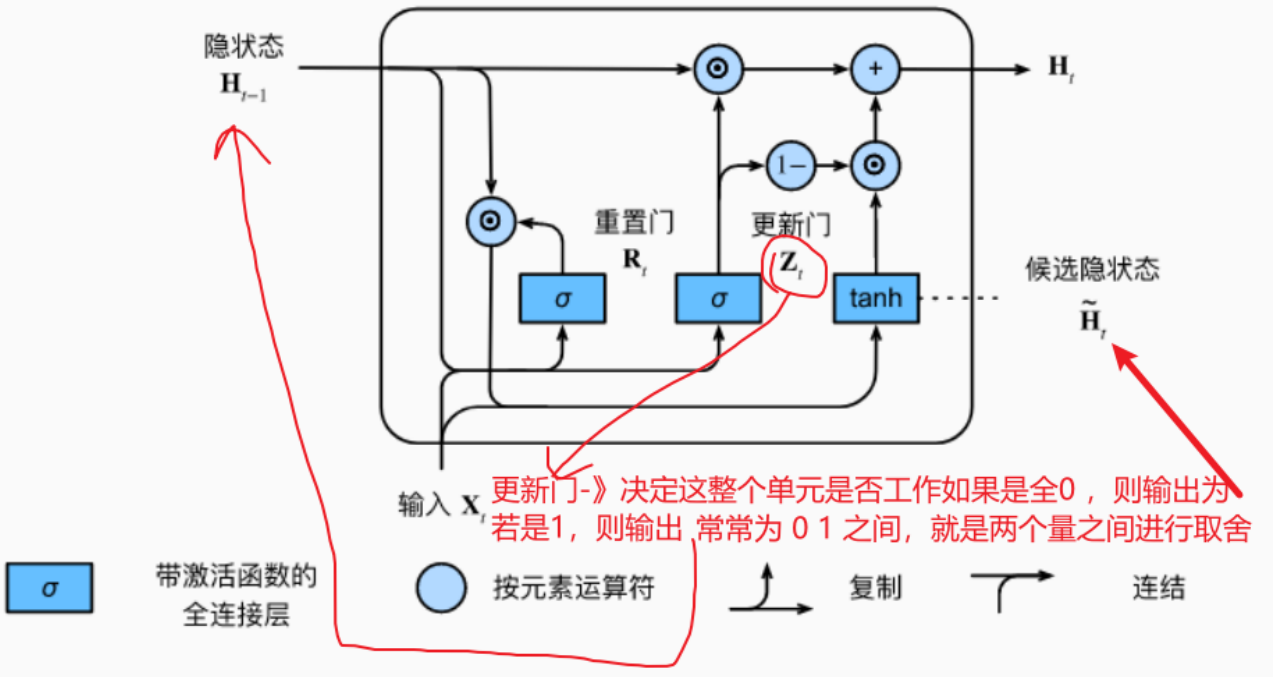


图9.1.3 计算门控循环单元模型中的隐状态

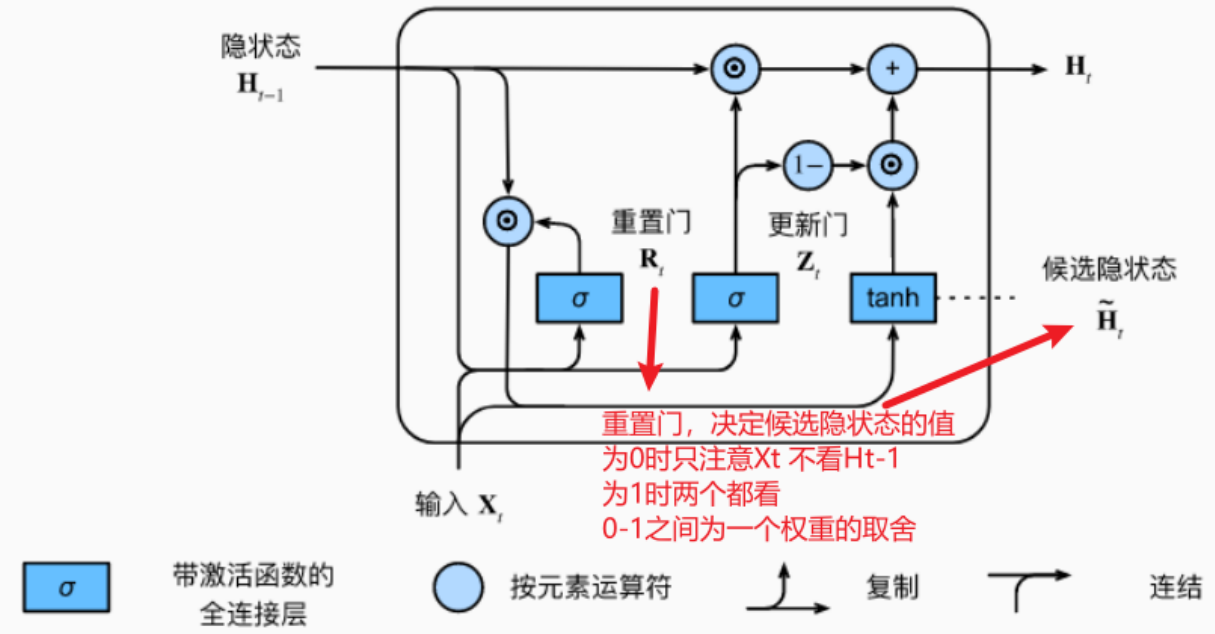


图9.1.3 计算门控循环单元模型中的隐状态

总结就是，这个GRU就是将xt 和 Ht-1 进行不同权重的取舍，极端情况就是可能只看xt 或只看Ht-1

三、代码

PYTHON

```
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)
# import torch
class MyGRU(torch.nn.Module):
    def __init__(self,input_size ,batch_size , num_layers,  hidden_siz
        super(MyGRU, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.batch_size = batch_size
        self.num_layers = num_layers
        self.embed = torch.nn.Embedding(self.input_size , self.hidden_
        self.GRU = torch.nn.GRU(input_size = self.hidden_size , hidden
```

```
        self.fc = torch.nn.Linear(hidden_size,input_size)

    def forward(self , x):
        # xs sel_len batch_size x_len
        print(x.shape)
        x = self.embed(x)
        print(x.shape)
        hidden = torch.zeros(self.num_layers , self.batch_size , self.
        x , hn = self.GRU(x,hidden)
        x = self.fc(x)
        return x


input_size = len(vocab)
batch_size = 32
hidden_size = 10
num_layers = 3

# input = torch.ones( batch_size , 5,dtype=torch.int)
net = MyGRU(input_size ,batch_size , num_layers, hidden_size)
# outs = net(input)
# print(out.shape)
lossfn = torch.nn.CrossEntropyLoss()
optim = torch.optim.Adam(net.parameters(),lr=0.05)
for x,y in train_iter:
    outs = net(x)
    loss = lossfn(outs.reshape(-1,input_size) , y.reshape(-1))
    optim.zero_grad()
    loss.backward()
    optim.step()
    print(f'loss:{loss}')
```