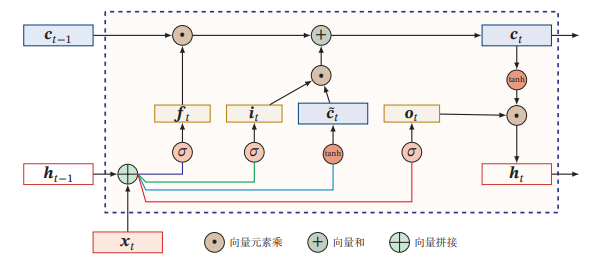
**1 LSTM和GRU的数学原理**

**1.1 LSTM(Long Short-Term Memory Network)**



**数学模型**

遗忘门

输入门

输出门

候选状态

内部状态 表示逐元素相乘(哈达玛积)

外部状态(隐状态)

实际输出通过变换得到，如

其中，

，其中

**核心思想**

外部状态每次更新幅度大，可以看作是一种短期记忆。神经网络的参数可看作长期记忆，其更新周期远远慢于短期记忆。内部状态可以在某一时刻捕捉到某个关键信息，并将其保存一定的时间间隔，其中保存的信息的生命周期长于短期记忆，但又远远短于长期记忆，因此称为长短期记忆(长的短期记忆)。内部状态是LSTM的关键

有变换的是短期，含自身上一时刻值做哈达玛积的是长短期

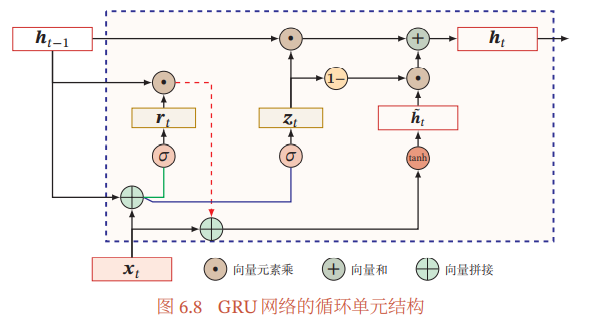
此外，LSTM通过有精心设计的“门”结构，在数字电路中门是一个二值变量，0表示关闭，1表示打开，LSTM中用sigmoid函数模拟此效果，其中：

(1)遗忘门控制将从上一时刻的内部状态中忘掉哪些信息，其每一维度上的值都在0到1之间，0表示完全遗忘，1表示完全保留，表示当前时刻内部状态是由上一时刻内部状态和当前时刻候选状态变换得来

(2)输入门控制当前时刻的候选状态中哪些被存入内部状态

(3)输出门控制当前时刻的内部状态(用tanh处理后)中哪部分传递需要传递给外部状态

**1.2 GRU(Gated Recurrent Unit)**



**数学模型**

更新门

重置门

候选状态

隐状态

实际输出通过变换得到，如

其中，

，其中

**核心思想**

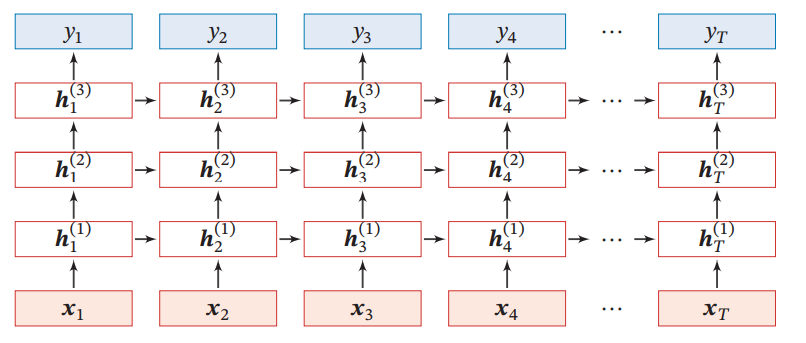
GRU舍弃了内部状态**，**通过捕捉短期依赖关系，再通过捕捉长期和短期依赖关系，相比LSTM的显著优势是计算量小，容易训练

LSTM中的遗忘门和输入门具有一定的互补冗余性，可将这两个门合并成一个门来控制遗忘和输入之间的平衡，即令。在GRU中，更新门相当于遗忘门和输入门的综合，可控制当前隐状态需要从上一时刻的隐状态中保留多少信息，以及需要从候选状态中接受多少信息

重置门控制候选状态是否依赖上一时刻的隐状态

**2多层循环神经网络和双向循环神经网络的数学原理**

**2.1多层循环神经网络**



**数学模型**

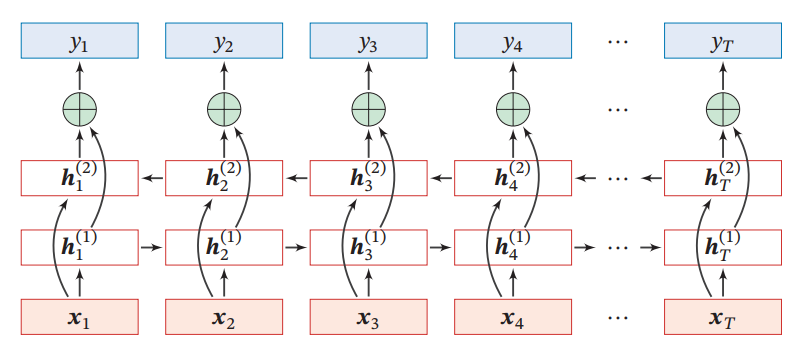
第

其中，视具体网络单元结构而定

**核心思想**

通过增加同一时刻网络输入到输出之间的路径来增加循环神经网络的深度

**2.2 双向循环神经网络**



**数学模型**

表示向量拼接操作

**核心思想**

某个时刻的输出不仅和过去时刻的信息有关，也和后续时刻的信息有关。在这些任务中，可增加一个按照时间的逆序来传递信息的网络层

**3 nn.LSTM&nn.GRU**

**nn.LSTM(input\_size,hidden\_size, num\_layers=1，bias=True,batch\_first=False,droupt=0, bidirectional=False)**

参数意义和nn.GRU相同

**输入输出**

设lstm是nn.LSTM的实例，则lstm(input,hidden)自动进行前向计算

当batch\_first=True时，

输入：input,(hidden0,cell0)

input是batch\_size×length\_seq×input\_size的输入矩阵

(hidden,cell)是num\_layers\*directions×batch\_size×hidden\_size的初始隐状态矩阵hidden0和同样大小的初始内部状态矩阵cell0组成的元组

输出：output, (hidden,cell)

output是batch\_size×length\_seq×hidden\_size\*directions的输出矩阵

(hidden,cell)是num\_layers\*directions×batch\_size×hidden\_size的最终隐状态矩阵hidden和同样大小的最终内部状态矩阵cell组成的元组

embed = torch.randn(2,2,3)

*#embed\_size:(batch\_size,length\_seq,input\_size)=(2,2,3)*

hidden0 = torch.randn(4,2,5)

*#hidden\_size:(num\_layers\*directions,batch\_size,hidden\_size)=(4,2,5)*

cell0 = torch.randn(4,2,5)

*#cell\_size:(num\_layers\*directions,batch\_size,hidden\_size)=(4,2,5)*

lstm = nn.LSTM(3,5,batch\_first=True,num\_layers=2,bidirectional=True)

*#input\_size=3,output\_size=5*

output,(hidden,cell) = lstm(embed,(hidden0,cell0))

*#output\_size:(batch\_size,length\_seq,directions\*hidden\_size)=(2,2,10)*

*#hidden\_size:(num\_layers\*directions,batch\_size,hidden\_size)=(4,2,5)*

print(output.size(),hidden.size(),cell.size())

>>>

torch.Size([2, 2, 10]) torch.Size([4, 2, 5]) torch.Size([4, 2, 5])

**nn.GRU(input\_size,hidden\_size, num\_layers=1，bias=True,batch\_first=False,droupt=0, bidirectional=False)**

**参数解释**

input\_size：表示输入矩阵单个样本的特征数#单词的嵌入维度

hidden\_size：隐藏层节点数#输出向量维度与隐层节点数相同

num\_layers：隐藏层层数，默认为1

bias：隐层是否带偏置，默认带偏置

batch\_first：默认为False，若为True，则input应为batch\_size×length\_seq×input\_size；若为False，则input应为length\_seq×batch\_size×input\_size。

dropout：取值在0-1之间，除最后一层，每一层的输出都按概率进行dropout，默认值为0，即不丢弃

bidirectional：是否用双向LSTM

**输入输出**

设gru是nn.GRU的实例，则gru(input,hidden)自动进行前向计算

当batch\_first=True时，

输入：input,hidden0

input是batch\_size×length\_seq×input\_size的输入矩阵

hidden0是num\_layers\*directions×batch\_size×hidden\_size的初始隐状态矩阵

输出：output,hidden

output是batch\_size×length\_seq×hidden\_size\*directions的输出矩阵

hidden是num\_layers\*directions×batch\_size×hidden\_size的最终隐状态矩阵

embed = torch.randn(2,2,3)

*#embed\_size:(batch\_size,length\_seq,input\_size)=(2,2,3)*

hidden0 = torch.randn(4,2,5)

*#hidden\_size:(num\_layers\*directions,batch\_size,hidden\_size)=(4,2,5)*

gru = nn.GRU(3,5,batch\_first=True,num\_layers=2,bidirectional=True)

*#input\_size=3,output\_size=5*

output,hidden = gru(embed,hidden0)

*#output\_size:(batch\_size,length\_seq,directions\*hidden\_size)=(2,2,10)*

*#hidden\_size:(num\_layers\*directions,batch\_size,hidden\_size)=(4,2,5)*

print(output.size(),hidden.size())

>>>

torch.Size([2, 2, 10]) torch.Size([4, 2, 5])