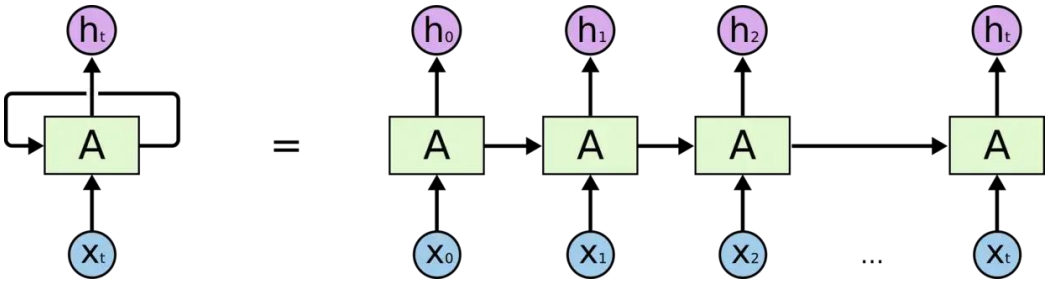
1 RNN 的缺陷——长期依赖的问题（The Problem of Long-Term Dependencies）

前面一节我们学习了 RNN 神经网络，它可以用来处理序列型的数据，比如一段文字，视频等等。RNN 网络的基本单元如下图所示，可以将前面的状态作为当前状态的输入。



但也有一些情况，我们需要更“长期”的上下文信息。比如预测最后一个单词“我在中国长大……我说一口流利的**。”“短期”的信息显示，下一个单词很可能是一种语言的名字，但如果我们想缩小范围，我们需要更长期语境——“我在中国长大”，但这个相关信息与需要它的点之间的距离完全有可能变得非常大。

不幸的是，随着这种距离的扩大，RNN 无法学会连接这些信息。

从理论上讲，RNN 绝对有能力处理这种“长期依赖性”。人们可以为他们精心选择参数，以解决这种形式的问题。遗憾的是，在实践中，RNN 似乎无法学习它们。

幸运的是，GRU 也没有这个问题！

2、GRU

什么是 GRU

GRU（Gate Recurrent Unit）是循环神经网络（Recurrent Neural Network, RNN）的一种。和 LSTM（Long-Short Term Memory）一样，也是为了解决长期记忆和反向传播中的梯度等问题而提出来的。

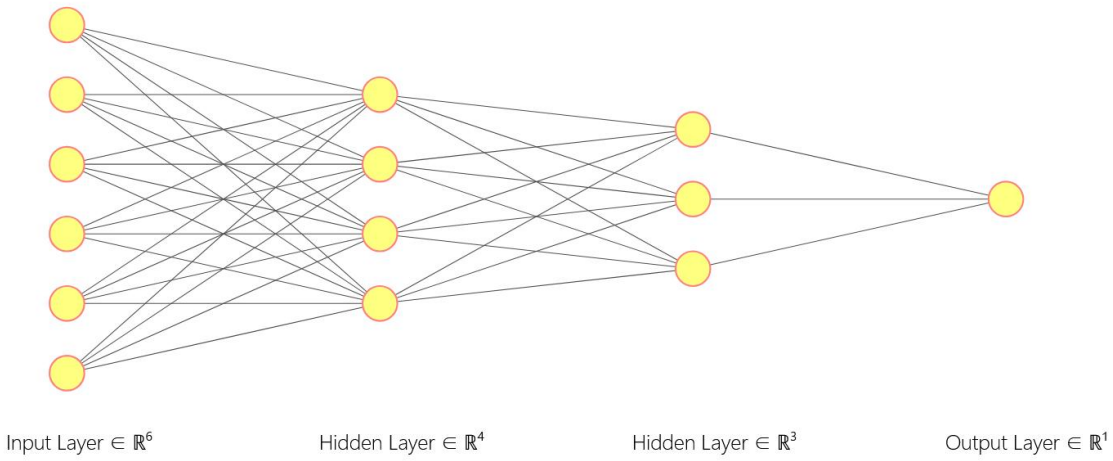
GRU 和 LSTM 在很多情况下实际表现上相差无几，那么为什么我们要使用新人 GRU（2014 年提出）而不是相对经受了更多考验的 LSTM（1997 提出）呢。

用论文中的话说，相比 LSTM，使用 GRU 能够达到相当的效果，并且相比之下更容易进行训练，能够很大程度上提高训练效率，因此很多时候会更倾向于使用 GRU。

2.1 总体结构框架

前面我们讲到，神经网络的各种结构都是为了挖掘变换数据特征的，所以下面我们也将结合数据特征的维度来对比介绍一下 RNN&&LSTM 的网络结构。

多层感知机（线性连接层）



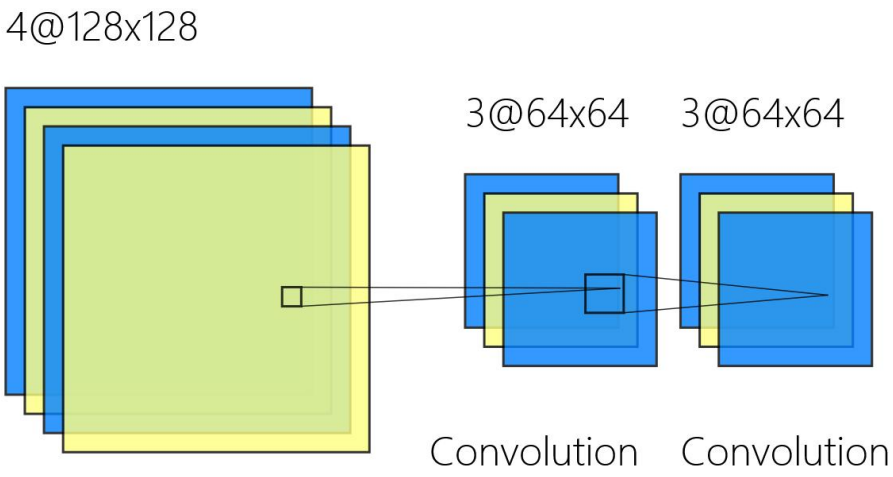
从特征角度考虑：

输入特征：是 n*1 的单维向量（这也是为什么卷积神经网络在 linear 层前要把所有特征层展平），

隐藏层：然后根据隐藏层神经元的数量 m 将前层输入的特征用 m*1 的单维向量进行表示（对特征进行了提取变换，隐藏层的数据特征），单个隐藏层的神经元数量就代表网络参数，可以设置多个隐藏层；

输出特征：最终根据输出层的神经元数量 y 输出 y*1 的单维向量。

卷积神经网络结构



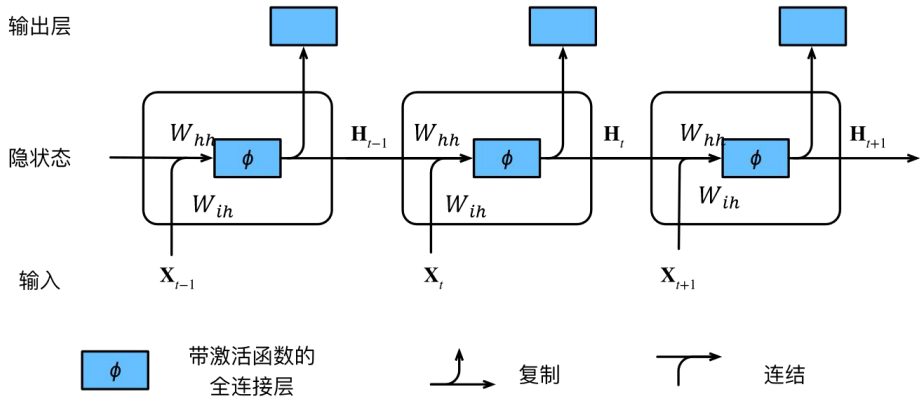
从特征角度考虑：

输入特征：是(batch)*channel*width*height 的张量，

卷积层（等）：然后根据输入通道 channel 的数量 c_in 和输出通道 channel 的数量 c_out 会有 c_out*c_in*k*k 个卷积核将前层输入的特征进行卷积（对特征进行了提取变换，k 为卷积核尺寸），卷积核的大小和数量 c_out*c_in*k*k 就代表网络参数，可以设置多个卷积层；每一个 channel 都代表提取某方面的一种特征，该特征用 width*height 的二维张量表示，不同特征层之间是相互独立的（可以进行融合）。

输出特征：根据场景的需要设置后面的输出，可以是多分类的单维向量等等。

循环神经网络 RNN 系列结构



从特征角度考虑：

输入特征：是(batch)*T_seq*feature_size 的张量（T_seq 代表序列长度，注意不是 batch_size）.

我们来详细对比一下卷积神经网络的输入特征，

(batch)*T_seq*feature_size

(batch)*channel*width*height，

逐个进行分析，RNN 系列的基础输入特征表示是 feature_size*1 的单维向量，

比如一个单词的词向量，比如一个股票价格的影响因素向量，而 CNN 系列的基础输入特征是 width*height 的二维张量；

再来看一下序列 T_seq 和通道 channel，RNN 系列的序列 T_seq 是指一个连续的输入，比如一句话，一周的股票信息，而且这个序列是有时间先后顺序且互相关联的，而 CNN 系列的通道 channel 则是指不同角度的特征，比如彩色图像的 RGB 三色通道，过程中每个通道代表提取了每个方面的特征，不同通道之间是没有强相关性的，不过也可以进行融合。

最后就是 batch，两者都有，在 RNN 系列，batch 就是有多个句子，在 CNN 系列，就是有多张图片（每个图片可以有多个通道）

隐藏层：明确了输入特征之后，我们再来看看隐藏层代表着什么。隐藏层有 T_seq 个隐状态 H_t(和输入序列长度相同)，每个隐状态 H_t 类似于一个 channel，对应着 T_seq 中的 t 时刻的输入特征；而每个隐状态 H_t 是用 hidden_size*1 的单维向量表示的，所以一个隐含层是 T_seq*hidden_size 的张量；对应时刻 t 的输入特征由 feature_size*1 变为 hidden_size*1 的向量。如图中所示，同一个隐含层不同时刻的参数 W_ih 和 W_hh 是共享的；隐藏层可以有 num_layers 个（图中只有 1 个）

以 t 时刻具体阐述一下：

X_t 是 t 时刻的输入，是一个 feature_size*1 的向量

W_ih 是输入层到隐藏层的权重矩阵

H_t 是 t 时刻的隐藏层的值,是一个 hidden_size*1 的向量

W_hh 是上一时刻的隐藏层的值传入到下一时刻的隐藏层时的权重矩阵

Ot 是 t 时刻 RNN 网络的输出

从上右图中可以看出这个 RNN 网络在 t 时刻接受了输入 Xt 之后，隐藏层的值是 St，输出的值是 Ot。但是从结构图中我们可以发现 St 并不单单只是由 Xt 决定，还与 t-1 时刻的隐藏层的值 St-1 有关。

2.2 GRU 的输入输出结构

GRU 的输入输出结构与普通的 RNN 是一样的。有一个当前的输入 xt，和上一个节点传递下来的隐状态（hidden state）ht-1 ，这个隐状态包含了之前节点的相关信息。结合 xt 和 ht-1，GRU 会得到当前隐藏节点的输出 yt 和传递给下一个节点的隐状态 ht。

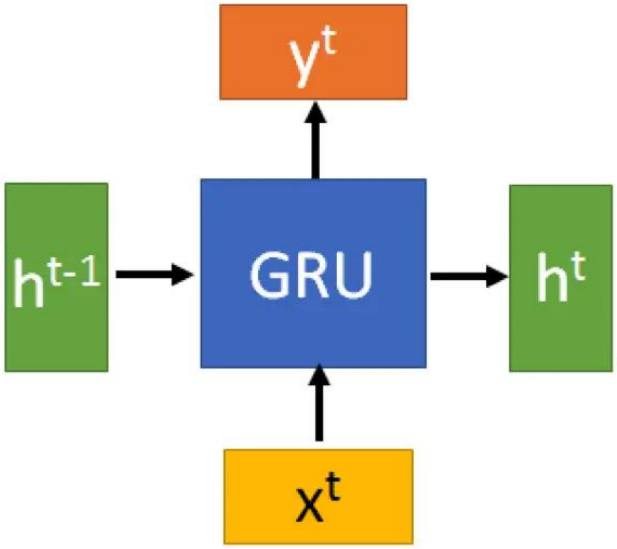


图 GRU 的输入输出结构

那么，GRU 到底有什么特别之处呢？下面来对它的内部结构进行分析！

2.3 GRU 的内部结构

不同于 LSTM 有 3 个门控，GRU 仅有 2 个门控，

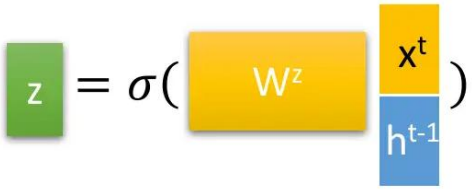
第一个是“重置门”（reset gate），其根据当前时刻的输入 xt 和上一时刻的隐状态 ht-1 变换后经 sigmoid 函数输出介于 0 和 1 之间的数字，用于将上一时刻隐状态 ht-1 重置为 ht-1’，即 ht-1’=ht-1*r。

$$r = \sigma(W^r \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$

再将 ht-1’与输入 xt 进行拼接，再通过一个 tanh 激活函数来将数据放缩到**-1~1** 的范围内。即得到如下图 2-3 所示的 h’。

$$h' = tanh(W \begin{bmatrix} x^t \\ h^{t-1'} \end{bmatrix})$$

第二个是“更新门”（update gate），其根据当前时刻的输入 xt 和上一时刻的隐状态 ht-1 变换后经 sigmoid 函数输出介于 0 和 1 之间的数字，



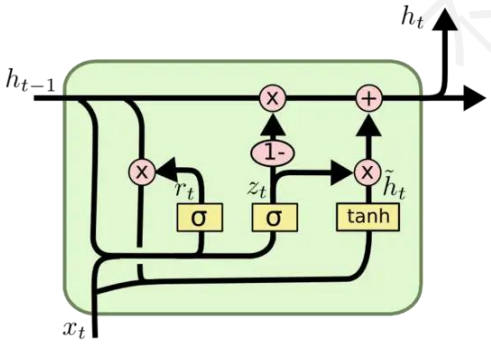
最终的隐状态 ht 的更新表达式即为：

$$h^t = z \odot h^{t-1} + (1-z) \odot h'$$

再次强调一下，门控信号（这里的 z）的范围为 0~1。门控信号越接近 1，代表”记忆“下来的数据越多；而越接近 0 则代表”遗忘“的越多。

2.4 小结

GRU 很聪明的一点就在于，**使用了同一个门控 z 就同时可以进行遗忘和选择记忆（LSTM 则要使用多个门控）**。与 LSTM 相比，GRU 内部少了一个”门控“，参数比 LSTM 少，但是却也能够达到与 LSTM 相当的功能。考虑到硬件的**计算能力**和**时间成本**，因而很多时候我们也就会选择更加”实用“的 GRU。



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

3 代码

```
import torch
import torch.nn as nn


def my_gru(input,initial_states,w_ih,w_hh,b_ih,b_hh):
    h_prev=initial_states
```

```
    batch_size,T_seq,feature_size=input.shape
    hidden_size=w_ih.shape[0]//3

    batch_w_ih=w_ih.unsqueeze(0).tile(batch_size,1,1)
    batch_w_hh=w_hh.unsqueeze(0).tile(batch_size,1,1)

    output=torch.zeros(batch_size,T_seq,hidden_size)

    for t in range(T_seq):
        x=input[:,t,:]
        w_times_x=torch.bmm(batch_w_ih,x.unsqueeze(-1))
        w_times_x=w_times_x.squeeze(-1)

       # print(batch_w_hh.shape,h_prev.shape)
        # 计算两个 tensor 的矩阵乘法，torch.bmm(a,b),tensor a 的 size 为(b,h,w),tensor b 的 size 为(b,w,m)
        # 也就是说两个 tensor 的第一维是相等的，然后第一个数组的第三维和第二个数组的第二维度要求一样，
        # 对于剩下的则不做要求，输出维度 （b,h,m）
        # batch_w_hh=batch_size*(3*hidden_size)*hidden_size
        # h_prev=batch_size*hidden_size*1
        # w_times_x=batch_size*hidden_size*1
        ##squeeze，在给定维度（维度值必须为 1）上压缩维度，负数代表从后开始数
        w_times_h_prev=torch.bmm(batch_w_hh,h_prev.unsqueeze(-1))
        w_times_h_prev=w_times_h_prev.squeeze(-1)


r_t=torch.sigmoid(w_times_x[:,:hidden_size]+w_times_h_prev[:,:hidden_size]+b_ih[:hidden_size]
                                  +b_hh[:hidden_size])

z_t=torch.sigmoid(w_times_x[:,hidden_size:2*hidden_size]+w_times_h_prev[:,hidden_size:2*hidden_size]

+b_ih[hidden_size:2*hidden_size]+b_hh[hidden_size:2*hidden_size])

n_t=torch.tanh(w_times_x[:,2*hidden_size:3*hidden_size]+w_times_h_prev[:,2*hidden_size:3*hidden_size]

+b_ih[2*hidden_size:3*hidden_size]+b_hh[2*hidden_size:3*hidden_size])

        h_prev=(1-z_t)*n_t+z_t*h_prev
        output[:,t,:]=h_prev
```

```
    return output,h_prev


if __name__=="__main__":

    fc=nn.Linear(12,6)


    batch_size=2
    T_seq=5
    feature_size=4

    hidden_size=3
   # output_feature_size=3

    input=torch.randn(batch_size,T_seq,feature_size)
    h_prev=torch.randn(batch_size,hidden_size)

    gru_layer=nn.GRU(feature_size,hidden_size,batch_first=True)
    output,h_final=gru_layer(input,h_prev.unsqueeze(0))
    # for k,v in gru_layer.named_parameters():
    #         print(k,v.shape)
    # print(output,h_final)

    my_output,
my_h_final=my_gru(input,h_prev,gru_layer.weight_ih_l0,gru_layer.weight_hh_l0,gr
u_layer.bias_ih_l0,gru_layer.bias_hh_l0)

    # print(my_output, my_h_final)
    # print(torch.allclose(output,my_output))
```

参考资料

https://zhuanlan.zhihu.com/p/32481747
https://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2018/Lecture/Seq%20(v2).pdf
https://www.bilibili.com/video/BV1jm4y1Q7uh/?spm_id_from=333.788&vd_source=cf7630d31a6ad93edecfb6c5d361c659