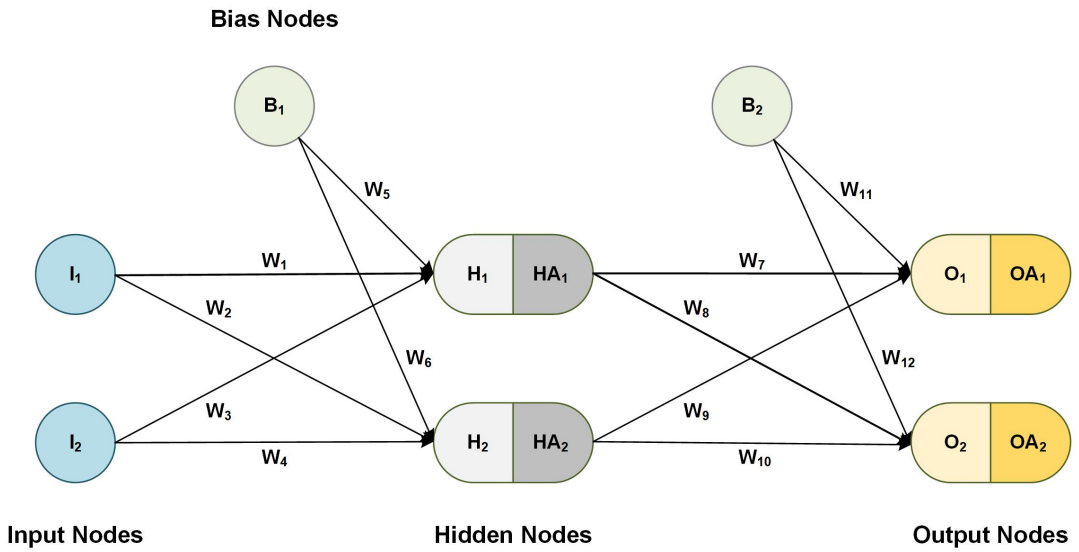
RNN及相关问题

- ✓ 前馈网络（Feedforward neural network, FNN）
- ✓ 循环神经网络（Recurrent neural network, RNN）
- ✓ 双向RNN（Bi-directional RNN）
- ✓ BP、BPTT、梯度消失（valishing）、梯度爆炸（explosion）
- ✓ 如何解决梯度爆炸 / 梯度消失问题？
  - ☐ 安全的激活函数
  - ☐ 梯度裁剪
  - ☐ LSTM

Grissom, 2019/07/11
sunbo2019.github.io

前馈网络（Feedforward neural network, FNN）



特点：

1）输入层-隐层、隐层-隐层、隐层-输出层，都是全连接。

2）每一层都可以有自己的偏置项（bias）

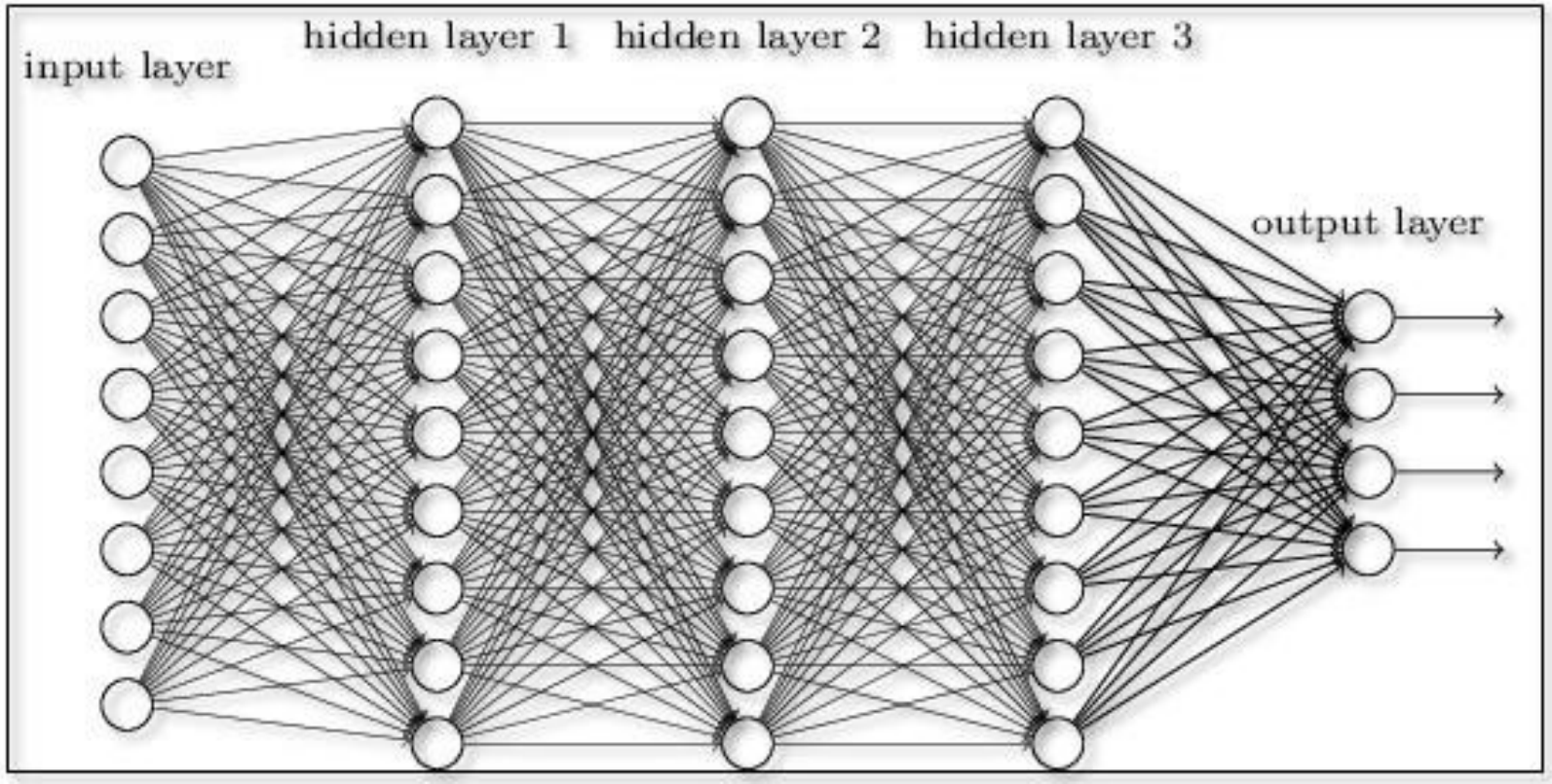
3）输入层、输出层只有一个，但隐层可以有多个。

4）隐层、输出层的节点（神经元）都有对应的激活函数，节点上的数据有两部分：激活前的数据（a）、激活后的数据（hidden status）。

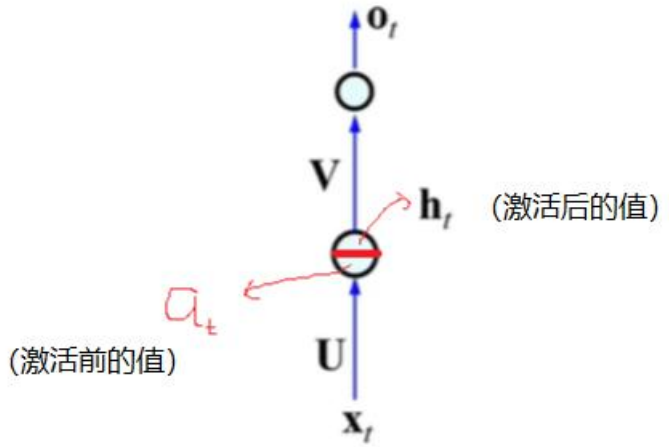
5）同一隐层的神经元之间，是无连接的。

6）适用于数据之间是独立的、无关联的情况。

有时为了简化，不会把偏置项单独写出来，而是在输入层增加一项x_0 = 1来使表达方式统一。



循环神经网络（Recurrent neural network, RNN）



如图所示，这是一个**前馈神经网络（FNN）**

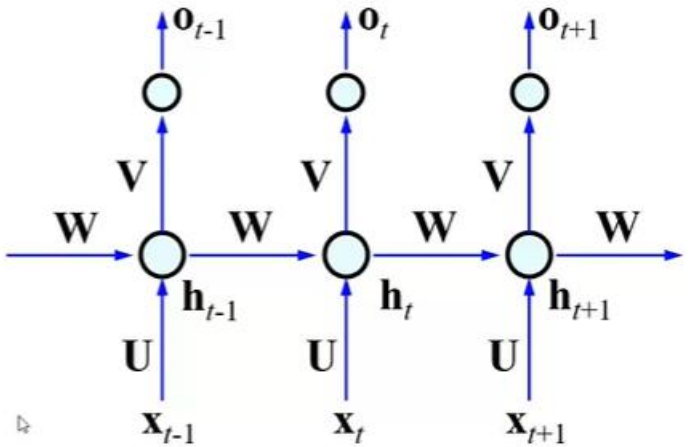
$$\mathbf{a}_t = \mathbf{b} + \mathbf{U}\mathbf{x}_t$$
$$\mathbf{h}_t = f(\mathbf{a}_t)$$
$$\mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{h}$$

b, c：偏置项（向量）
U, V, W：权重（矩阵）
$f(\,)$：激活函数

那么，我们现在将多个这样的FNN串起来：

增加了前一时刻的FNN的隐层信息

$$\mathbf{a}_t = \mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t$$
$$\mathbf{h}_t = f(\mathbf{a}_t)$$
$$\mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{h}$$



便构成了RNN

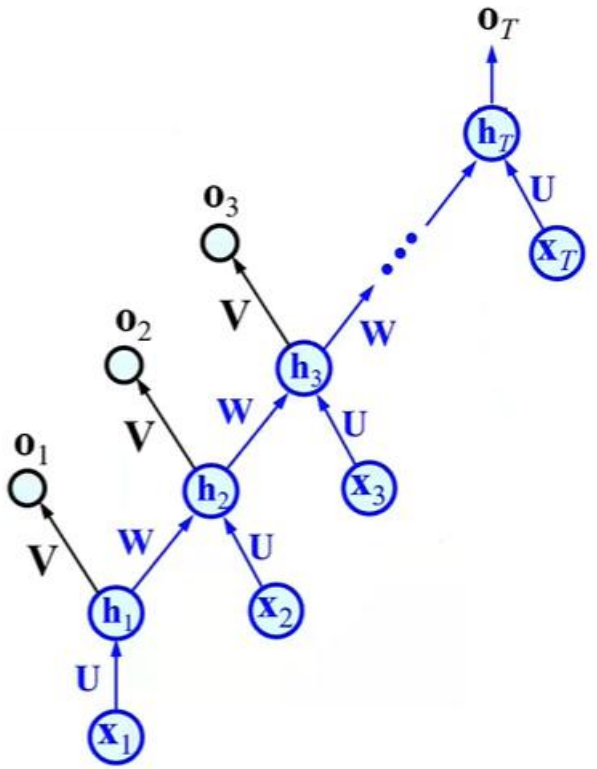
换个角度看（特点），

1) 一个RNN是由多个前馈神经网络（FNN）构成的。
[这些FNN，在tensorflow中被称为RNN Cell]

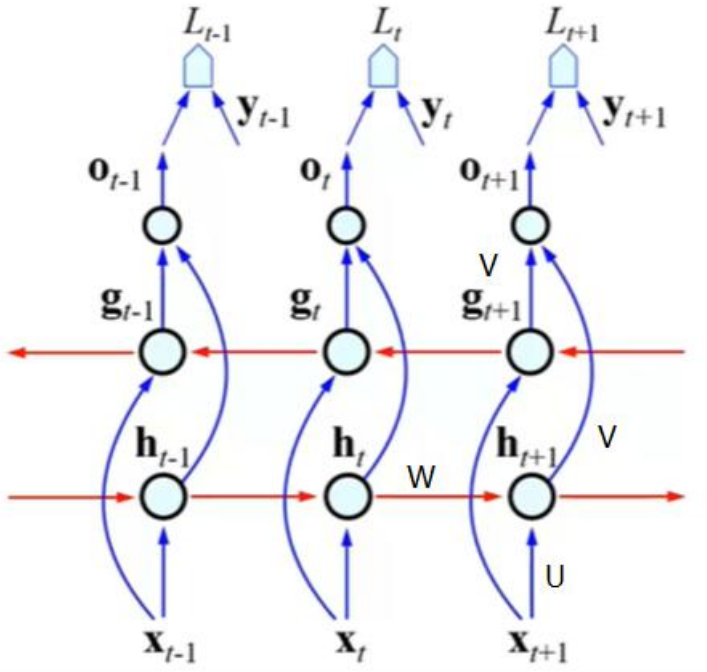
2) 每一个RNN Cell都有：
一个输入值（X_i）
两个输出值（o_i、h_i）

3) 所有的RNN Cell之间**共享权重U、V、W。**

4) RNN一般用于处理有“**时序依赖关系**”的数据！
（比如句子，因为句子中的词是有前后依赖关系的）



双向RNN（Bi-directional RNN）



这是最常用的一个RNN变种！

X：输入数据 （向量）

o：输出数据 （向量）

U： input layer -> hidden layer的权重 （矩阵）// 全连接，且所有RNN Cell共享

V： hidden layer -> output layer的权重 （矩阵）// 全连接，且所有RNN Cell共享

W： 一个RNN Cell的隐层 -> 另一个RNN Cell的**同级隐层** （方阵）// 全连接，且所有RNN Cell共享

另外需要注意的是，**Bi-RNN并不是深度网络！**

[在tensorflow中体现为depth=1，因为它的隐层是并行的（一个处理forward、一个处理backward），并不是串行的]

BP、BPTT、梯度消失（valishing） 、梯度爆炸（explosion）

1> BP - Back Propagation（反向传播）

(1) 正向传播、反向传播是一体的。[因为要先做正向传播，才能拿到反向传播时需要的数值]

(2) 反向传播只是“计算和存储梯度”的方法，而求解最佳参数的方法依然是“优化算法”（如：随机梯度下降）。

[只不过，这个优化算法依赖于反向传播来提供数据]

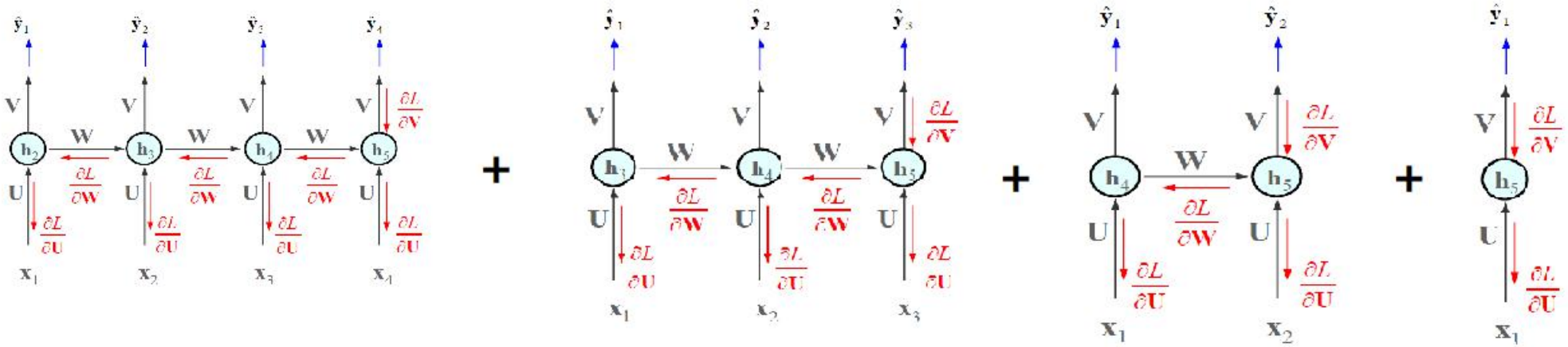
(3) 本质上，**正向传播**就是计算由激活函数一层层嵌套构成的复合函数的值；

反向传播就是在给定y的情况下，求解这个复合函数的参数u、w、v的梯度（链式法则）。

(4) 基于“计算图”可以清晰理解正向/反向传播的计算过程。

2> BPTT - Back Propagation Through Time (随时序进行的反向传播)

(1) 简言之，就是将某个参数在每一时刻的偏导数累加，作为这个参数在整个RNN上的求导结果。



以求解第k时刻隐层向量h的导数为例，根据RNN的特点 $\mathbf{h}_{k+1}= f(\mathbf{W}_{k+1}\mathbf{h}_k)$ 及链式求导法则：

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{h}_k} &= f'(\mathbf{h}_k)\mathbf{W}_{k+1}\frac{\partial L}{\partial \mathbf{h}_{k+1}} \\
&= f'(\mathbf{h}_k)f'(\mathbf{h}_{k+1})\mathbf{W}_{k+1}\mathbf{W}_{k+2}\frac{\partial L}{\partial \mathbf{h}_{k+3}} \\
&= \cdots \\
&\approx \left(f'(\mathbf{h})\mathbf{W}\right)^t \frac{\partial L}{\partial \mathbf{h}_{k+t}}
\end{aligned}
$$

因为：在RNN中，隐层权重矩阵W是各RNN Cell共享的，所可以可能存在以下问题
（1）当**W初始化值过大**时，W的t次方可能引起**梯度爆炸**
（2）当**激活函数f在h_i处的导数过小**时，t个-1~1的数连乘可能引起**梯度消失**

工程实践中，它们表现为：

梯度爆炸 -> 训练过程中，模型损失变成 NaN

梯度消失 -> 训练过程中，正确率很低且提高很慢（或停滞不前） // 也有可能是数据有问题

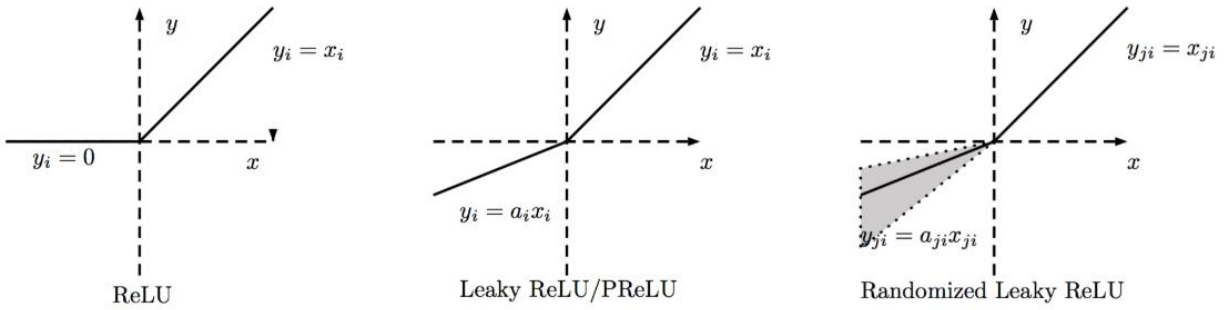
*** 无论是FNN还是RNN，只要是神经网络，都必然存着的梯度爆炸/梯度消失的问题。**

*** 梯度消失问题，更常见。**

如何解决梯度爆炸/消失问题？

常用的方法有三种（可组合使用）：

1）**选择安全的激活函数**（比如：ReLU及其变种。因为它的导数是1）



$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

2）梯度裁剪（Gradient Clipping）

简言之，就是在训练过程中，检查并将梯度限制在一定的范围内。具体是怎么做的呢？

$$g = \min\left(\frac{\theta}{\|g\|}, 1\right) g$$

解释：

若 $\|g\| \leq \theta$, 则 $\min\left(\frac{\theta}{\|g\|}, 1\right) = 1 \quad => g = g$

若 $\|g\| > \theta$, 则 $\min\left(\frac{\theta}{\|g\|}, 1\right) = \frac{\theta}{\|g\|} \quad => g = \frac{\theta}{\|g\|} g$

也就是将向量g的长度限制在 θ 倍长的范围内。
也可以认为是将向量g的各维度进行了等比例收缩。

3）LSTM（Long short-term memory，长短时记忆）

（请见“LSTM笔记专题”）

注意：LSTM也只是缓解了梯度消失、梯度爆炸，并不是完全解决了问题。因为倘若LSTM的层级（depth）较深或输入序列（sequence_length）很长，仍然有可能出现梯度爆炸，这时可以配合“梯度裁剪”使用。