**RNN的基本结构**  


* 网络某一时刻的输入*xt*，*xt*是一个*n*维向量，不同的是递归网络的输入将是一整个序列，也就是*x*=[*x*1,...,*xt*−1,*xt*,*xt*+1,...*xT*]，对于语言模型，每一个*xt*将代表一个词向量，一整个序列就代表一句话。
* *ht*代表时刻*t*的隐藏状态
* *ot*代表时刻*t*的输出
* 输入层到隐藏层直接的权重由*U*表示，它将我们的原始输入进行抽象作为隐藏层的输入
* 隐藏层到隐藏层的权重*W*，它是网络的记忆控制者，负责调度记忆。
* 隐藏层到输出层的权重*V*，从隐藏层学习到的表示将通过它再一次抽象，并作为最终输出。

**RNN的Forward阶段**

在*t*=0的时刻，*U*,*V*,*W*随机初始化，*h*0通常初始化为0，然后进行如下计算：

*s*1=*Ux*1+*Wh*0

*h*1=*f*(*s*1)

*o*1=*g*(*Vh*1)

这样时间就向前推进，此时的状态*h*1作为时刻0的记忆状态将参与下一次的预测活动，也就是

*s*2=*Ux*2+*Wh*1

*h*2=*f*(*s*2)

*o*2=*g*(*Vh*2)

，以此类推

*st*=*Uxt*+*Wht*−1

*ht*=*f*(*Uxt*+*Wht*−1)

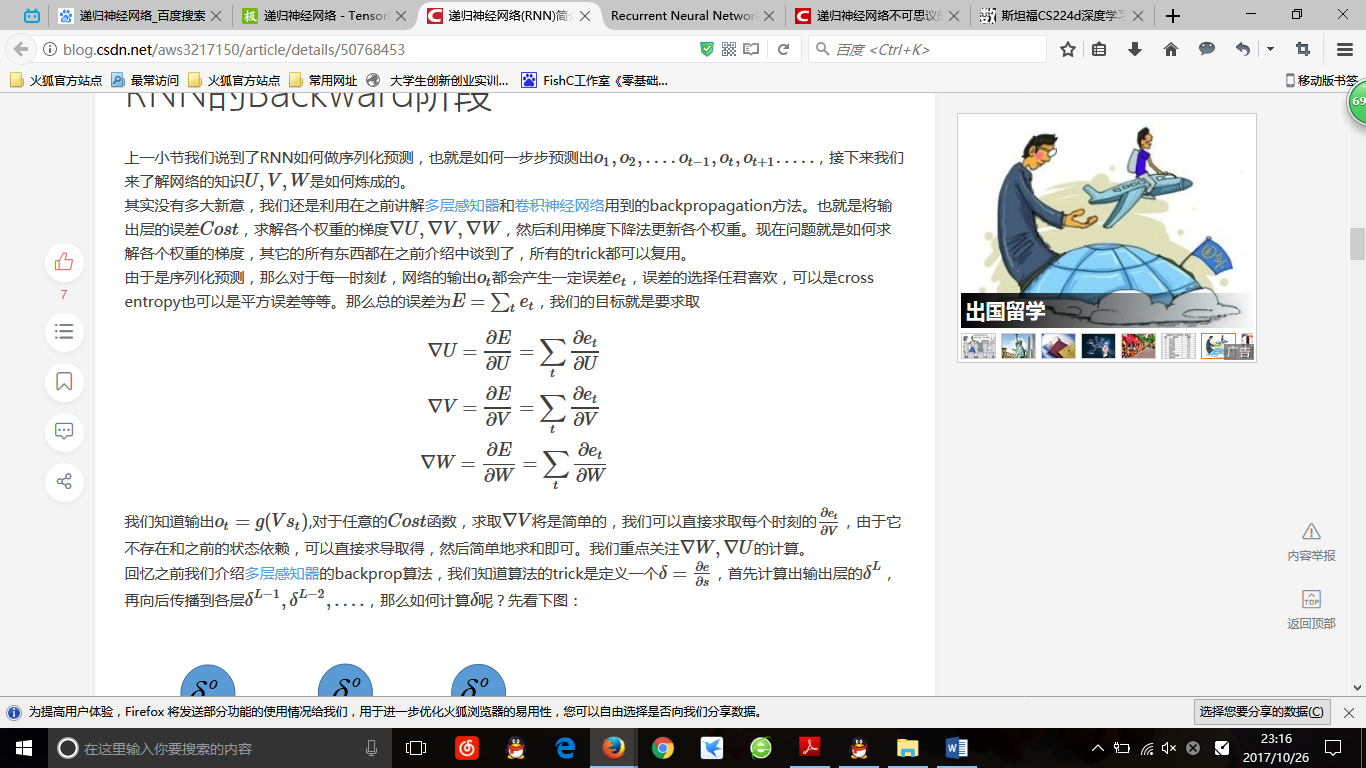
*ot*=*g*(*Vht*)

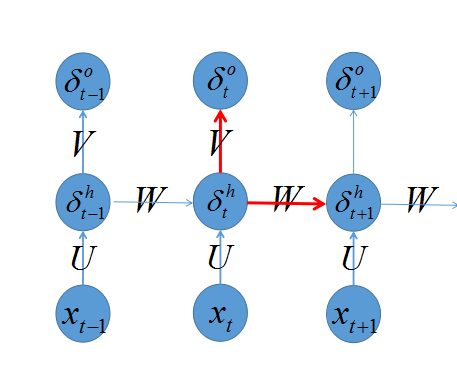
其中*f*可以是*tanh*,*relu*,*logistic*等，*g*通常是*softmax*。   
 递归神经网络拥有记忆能力，而这种能力就是通过*W*将以往的输入状态进行总结，而作为下次输入的辅助。可以这样理解隐藏状态：

*h*=*f*(现有的输入+过去记忆总结)

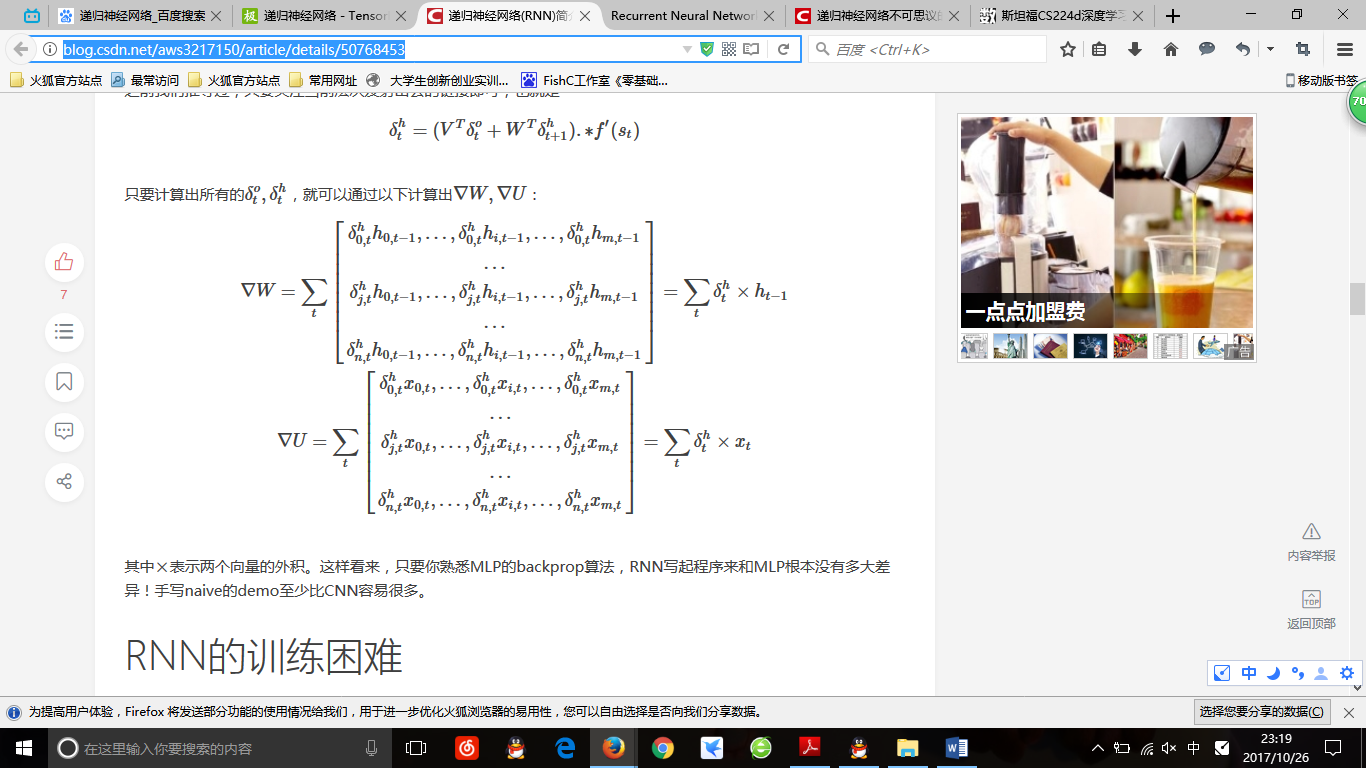
**RNN的Backward阶段**

RNN利用反向传播来更新参数，也就是对输出层的误差*Cost*，反向求解各个权重的梯度∇*U*,∇*V*,∇*W*，然后利用梯度下降法更新各个权重。由于是序列化预测，那么对于每一时刻*t*，网络的输出*ot*都会产生一定误差*et*，误差可以是cross entropy也可以是平方误差等。那么总的误差为*E*=∑*tet*，我们的目标就是要求取



我们知道输出*ot*=*g*(*Vst*),对于任意的*Cost*函数，求取∇*V*将是简单的，可以直接求取每个时刻的∂*et/*∂*V*，由于它不存在和之前的状态依赖，可以直接求导取得，然后简单地求和即可。我们重点关注∇*W*,∇*U*的计算。   
我们知道算法的trick是定义一个*δ*=∂*e/*∂*s*，首先计算出输出层的*δL*，再向后传播到各层*δL*−1,*δL*−2,....，那么如何计算*δ*呢？先看下图：   
  
之前我们推导过，只要关注当前层次发射出去的链接即可，也就是

*δht*=(*VTδot*+*WTδht*+1).∗*f*′(*st*)  
只要计算出所有的*δot*,*δht*，就可以通过以下计算出∇*W*,∇*U*：



参考: http://blog.csdn.net/aws3217150/article/details/50768453