# 对NNET的深入理解（第3部分） - LSTM和GRU

### 回顾

在上一篇文章中，我们简要地讨论了RNNs，并讨论了状态性和递归性。我们还研究了消失/爆炸梯度问题，并了解了双向RNN是如何工作的。为了解决消失梯度问题，研究人员开发了一个已经存在的想法，并通过引入LSTM网络来捕获长期依赖性。在下一节中，我们将深入了解LSTM，并了解它是如何导致开发或GRUs或门控递归单元的。

### LSTM公司

用数学方法证明了香荚兰RNNS的消失敏感性，包括两个主要因素1。重量初始化2。反向传播

权重初始化不是避免渐变消失的直接解决方案，但它有助于避免任何直接问题。另一方面，反向传播是梯度消失的主要原因，当反向传播和同时向前传递来计算每个时间步相对于权重的误差梯度，读取实时递归学习（RTRL）以获得更多信息时，该问题变得更加严重。因此，截断反向传播似乎是一个好主意，但是知道何时截断反向传播是很重要的，因为我们需要相应地更新权重，从而允许模型进行。因此，消除梯度的方法分为两部分，知道截断反向传播的频率和更新模型的频率。

在解决了消失梯度后，研究人员还想解决香草RNNs所带来的信息形态学问题。简单地说，由于非线性，包含在先前状态中的信息被反复嵌入，并且不是当前状态下信息的最佳可用状态。实质上，变形后的信息会丢失原来可用的信息。

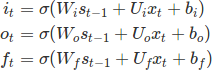
信息的原创性是可以保留的，这是由的地标性论文提出的。他们问：“我们怎样才能通过一个单独的单元，用一个单独的连接（即，一条单独的信息）来实现恒定的错误流？”？”

答案很简单，就是避免信息变形：通过显式的加减运算，对LSTM状态的改变被显式地写入，这样状态的每个元素都保持不变，而不受外界干扰：“单元的激活必须保持不变，这是通过使用标识函数来保证的。”。Hocreiter和Schmidhuber观察到，在每个状态下简单的信息加或减可能使状态保持孤立，但同时，加或减可能会抵消或更糟，它们可能使状态复杂化，只保留部分信息，难以恢复。

Hochreiter和Schmidhuber认识到了这个问题，把它分成几个子问题，他们称之为“输入权重冲突”、“输出权重冲突”、“滥用问题”和“内部状态漂移”。为了克服这些问题，从选择性的概念开始，LSTM体系结构被精心设计。

根据LSTM文献，LSTM应该有选择地决定三件事：“写什么，读什么，忘什么”。保持选择性的最基本和数学方法是门，我们称这些门为读、写和忘记门。我们在时间步骤t的三个门分别表示为i（t）、输入门（用于写入）、o（t）、输出门（用于读取）和f（t）、忘记门（用于记忆！）.

下面是闸门的数学定义（注意相似性）：



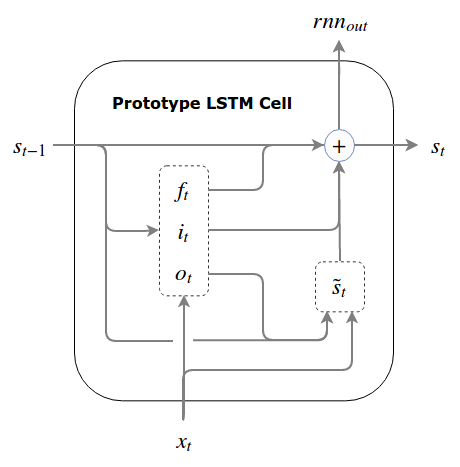
控制LSTM的方程。

定义了所有的门之后，我们现在通过定义所需的行为来开发LSTM原型。为了写出候选状态s（t），我们遵循一个简单的经验法则。

1. 使用写门获取输入
2. 使用读取门计算输出（输出是读取输入信息，以便您记住它使用读取门而不是输入）
3. 将输出与相关先验信息结合起来，为了保留相关信息，我们使用具有先验状态的遗忘门。

控制LSTM的方程。

下面是上述方程式的图示，箭头指向LSTM单元内的数据流。



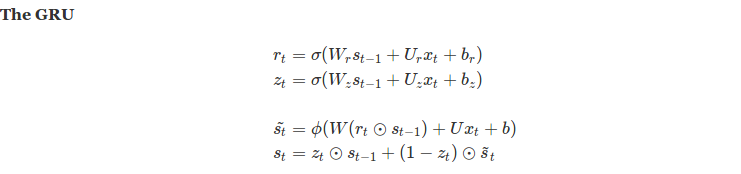
一个LSTM细胞。

从理论上讲，这个原型应该可以工作，但事实证明它没有。这是因为，即使在经过深思熟虑的初始化和写忘记门之后，这些门在训练的早期阶段之间的协调变得很棘手，并且在写的步骤中常常变得大而混乱。有关更多详细信息，请参阅“内部状态漂移”问题，此外，还可以在中找到对此的经验演示，其中包括LSTM的8个变体。

解决上述问题的方法是限制状态以防止其变得混乱或爆炸。LSTM有3种变体，使用此解决方案1。标准化LSTM、GRU和伪LSTM。我们将主要关注这篇文章中的GRU，但可以更深入地研究其他变体。

### 格鲁

我们通过显式地绑定write-and-forget门来对状态施加硬约束。换言之，我们将forget定义为1减去write gate，而不是进行选择性写入和选择性遗忘。所以没有写的东西都会被遗忘。在GRU术语中，遗忘门被重命名为update gate或z（t），其本质上意味着“不更新”。因此，对前一个状态进行元素级的更新将告诉不更新的内容，而1 - z（t）实际上会更新作为新写门的状态。



控制GRU的方程。

下面是上述方程式的图示，箭头指向GRU单元内的数据流。

一间地下室。

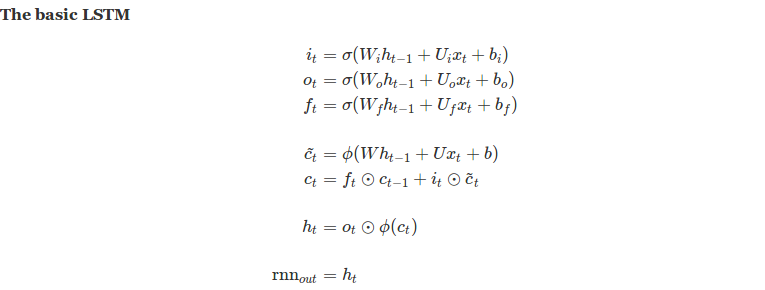
注意读和写之间的区别：如果我们选择不从一个单元中读，它不会影响我们状态的任何元素，而我们的读决定会影响整个状态。如果我们选择不给一个单位写信，那只会影响我们国家的一个元素。这并不意味着选择性读取的影响比选择性写入的影响更为显著：读取被非线性叠加和挤压，而写入是绝对的，因此读取决策的影响是广泛而浅的，写入决策的影响是窄而深的。

你可能还想知道，我们所说的LSTM细胞不安静，看起来像基本的LSTM细胞可在互联网上，你是对的。原因是我们没有在上面定义基本的LSTM单元，我们定义了一个原型单元，我们依次回答了vanilla RNNs面临的所有问题。现在我们将继续定义基本的LSTM单元。

### 基本LSTM单元

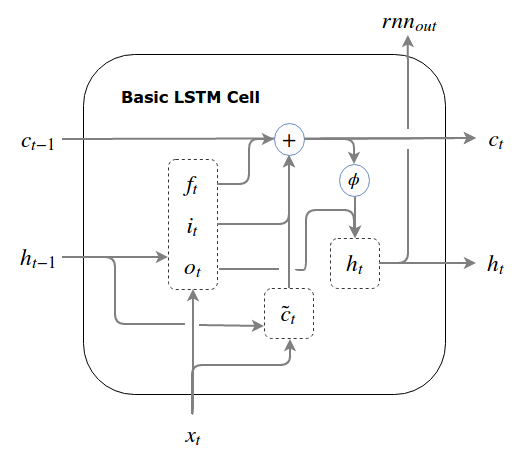
如上所述，读在写之后，因为单元将输入写入内存，然后在计算期间读取输出，最后应用忘记门并更新单元。在这里，我们松散地使用在基本LSTM单元的构造中起重要作用的术语存储器。基本的LSTM单元需要对原型稍作改动，我们现在将向一个单元输入2个prior，即先前的状态s（t）现在重命名为c（t）和阴影/隐藏状态h（t）。隐藏状态只不过是一个封闭的前一个状态，此外，前一个状态也会在单元格中流动。它的输出是一个更新的当前状态和一个隐藏的状态，这是一个门控的当前状态。

如果我们仔细考虑，基本的LSTM是以两种形式获取先前的状态，直接和选通（外部输入除外），并以两种形式生成当前状态，直接更新其选通版本。引入所有这些复杂性和隐藏状态的主要原因是“写-读-读”命令。我们需要读取前一个状态才能创建当前候选写入。但是，如果创建当前候选写操作的时间早于单元内的读取操作，则除非传递一个预先设置好的“前一状态”（previous state），使隐藏状态成为强制状态，否则无法执行该操作。因此，写入然后读取顺序迫使LSTM将隐藏状态从一个单元传递到另一个单元。



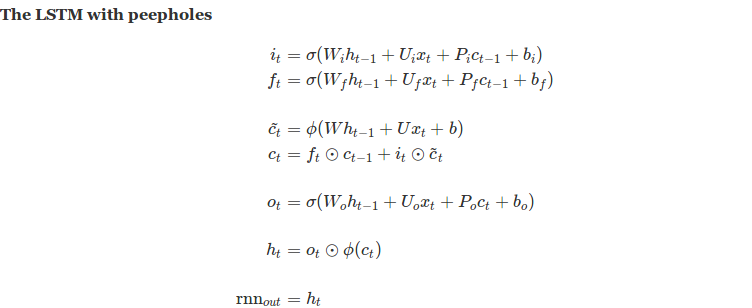
基本的LSTM方程根据内存单元视图更新。

下面是上述方程式的图示，箭头指向基本LSTM单元内的数据流。



一个基本的LSTM单元，可在深度学习库中使用。

虽然这种LSTM的实现是稳定的，并且具有很好的可伸缩性，但未经修改的前一状态输入有时被重新连接到门计算中，从而产生带有窥视孔的LSTM，这只是LSTM的另一种变体。与窥视孔LSTM的主要区别在于，更新的当前状态用于通过读取门输出，而不是由基本LSTM单元读取的先前状态。



带窥视孔的LSTM控制方程。

### 结论

LSTM及其变体解决了与RNNs相关的基本（信息变形）和技术（消失梯度）问题，因而获得了广泛的应用。与LSTM及其变体相关的意识形态也允许研究者在阅读和书写信息时实现类似的选择性思维过程。这种思想为残余网络或ResNet与非常深（高达100层）的架构相结合铺平了道路。该网络赢得了ImageNet 2015比赛。

这篇文章的内容可能会与视觉描述混淆，多亏了deepsystems.ai，您可以观看以获得更好的理解。他们的视频和引用的这篇文章的文字灵感来自R2RT的博客文章：。

In the next post, we will look at Auto Encoders in detail and also explore their utility in modern architectures.

**Thank you for reading, I hope it helped**