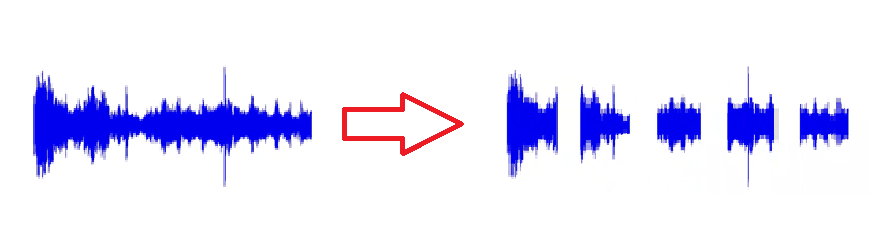
# 循环神经网络

## 循环神经网络

### 序列数据

对于图像数据或者表格数据，我们可以设计专门的卷积神经网络架构来为这类特殊的数据结构建模。但是这样的数据都来自于某种分布，并且所有样本都是独立同分布的。而在我们的身边还有许多数据并非如此，比如序列数据，一段音频就是一段序列数据，它可以分解成一系列的波形，如图所示：



一段文本可以分解为字符序列或者一串串的单词：



除了文本和音频，还存在着许多的序列数据，如心电图、股票、DNA序列等等。

### 什么是循环神经网络？

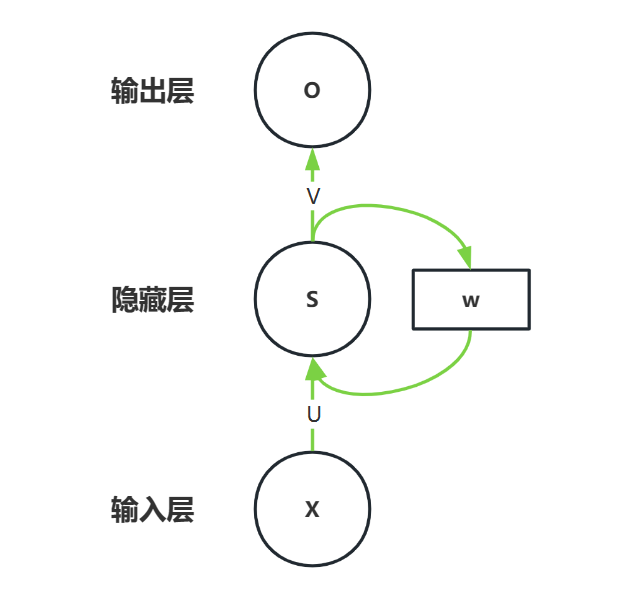
循环神经网络（Rerrent Neural Network, RNN）是一种神经网络结构，其主要特点是能够处理序列数据，即具有时间维度的数据。与其他类型的神经网络不同，RNN可以捕捉先前时间步骤的信息并将其传递到后续时间步骤中，从而在整个序列上产生输出。

RNN通过在网络中引入循环连接来实现这种能力，使得信息可以在时间步骤之间传递。具体来说，每个时间步骤上的输入被送入网络中，网络根据当前输入和先前时间步骤的状态计算输出和当前时间步骤的状态，然后将当前时间步骤的状态传递到下一个时间步骤中，以处理下一个输入。这样，RNN可以对输入序列的每个时间步骤进行建模，并生成与序列相关的输出。

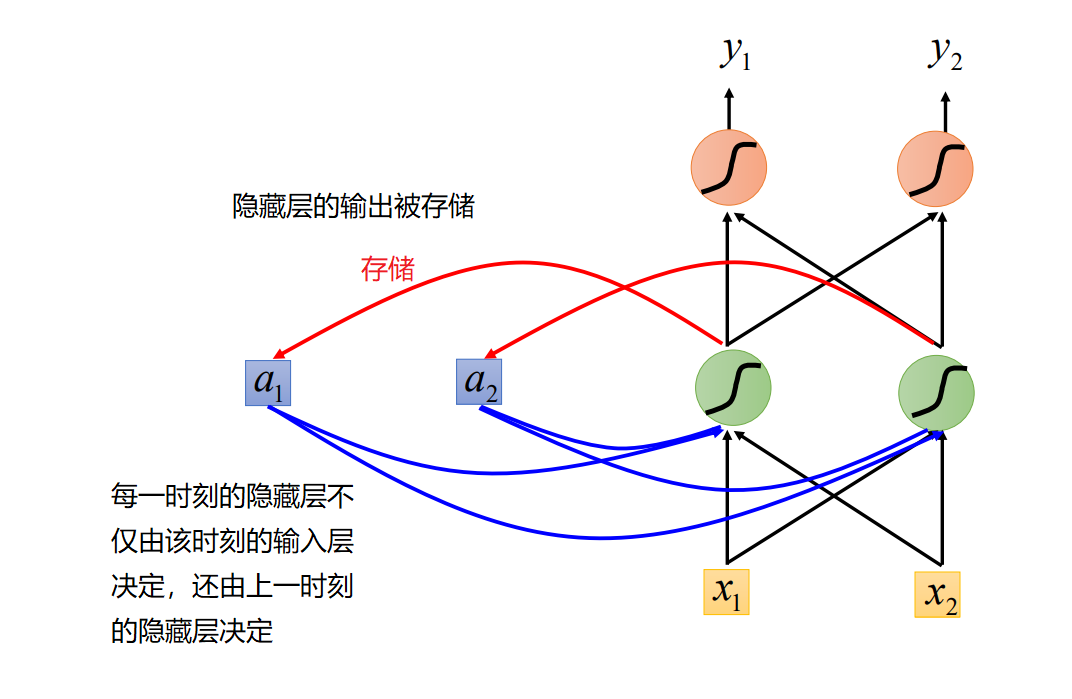
循环神经网络具有广泛的应用，包括语音识别、自然语言处理、视频分析和时间序列预测等领域。

### 循环神经网络组成部分和基本原理

循环神经网络（RNN）主要由三个基本组成部分组成：输入层、隐藏层和输出层。其中，输入层接受序列数据作为输入，隐藏层用于捕捉序列中的时间依赖关系并生成隐藏状态，输出层产生与序列相关的输出。



在每个时间步骤上，输入序列被馈送到输入层，然后将输入传递到隐藏层中。隐藏层使用激活函数来计算当前时间步骤的隐藏状态，并将该状态传递到下一个时间步骤中，以便能够捕捉时间上的依赖关系。最后，在每个时间步骤结束时，输出层使用隐藏状态计算输出，并生成与输入序列相关的输出结果。



除此之外，循环神经网络中还有一个重要的组成部分：循环连接。循环连接是由隐藏层的节点自己与前一个时间步骤的隐藏状态连接而成，用于实现信息在时间步骤之间的传递和保持记忆。这种循环连接使得循环神经网络具有对序列数据进行建模的能力，从而可以用于处理自然语言、语音、时间序列等具有时间性质的数据。

### RNN的简单实现

下面是使用Numpy来简单实现一个循环神经网络：

与传统神经网络不同，RNN拥有3个权重参数，即输入权重、内部状态权重（用于存储记忆）和输出权重。我们首先用随机值初始化这些参数。我们将单词嵌入维度和输出维度分别初始化为100和80。

1. hidden\_dim = 100
2. output\_dim = 80
3. input\_weights = np.random.uniform(0, 1, (hidden\_dim, hidden\_dim))
4. internal\_state\_weights = np.random.uniform(0,1, (hidden\_dim, hidden\_dim))
5. output\_weights = np.random.uniform(0,1, (output\_dim,hidden\_dim))

假如有一个句子 "我喜欢打篮球。"。.在词汇表中，我们假设“我”被映射到索引2，“喜欢”被映射到索引45，“打”被映射到索引10，“篮球”被映射到索引64，标点符号“。”被映射到索引1。为了得到一个从输入到输出的真实场景，让我们随机地初始化每个词的单词嵌入。

1. input\_string = [2,45,10,65]
2. embeddings = []
3. for i in range(0,T):
4. x = np.random.randn(hidden\_dim,1)
5. embeddings.append(x)

现在我们已经完成了输入，我们需要考虑每个单词输入的输出。RNN单元应该输出当前输入的下一个最有可能的词。为了训练RNN，我们提供第t+1个词作为第t个输入值的输出，例如：对于给定的输入词“我”，RNN单元应该输出“喜欢”这个词。

现在，输入是嵌入向量的形式，计算损失所需的输出格式应该是独热编码向量。这是对输入字符串中除第一个字以外的每个字进行的，因为我们只考虑一个例句供神经网络学习，而初始输入是该句子的第一个字。

1. output\_mapper = {}
2. for index\_value in output\_string :
3. output\_mapper[index\_value] = identity\_matrix[index\_value,:]

在实施过程中，我们需要注意输出映射器（output\_mapper）的关键值。我们需要将关键值重置为其时间戳值，以便算法知道在特定时间戳时需要使用哪个真实值单词来计算损失。

1. output\_t = {}
2. i=0
3. for key,value in output\_mapper.items():
4. output\_t[i] = value
5. i+=1

定义激活函数Tanh和softmax函数

1. def tanh\_activation(Z):
2. return (np.exp(Z)-np.exp(-Z))/(np.exp(Z)-np.exp(-Z))
3. def softmax\_activation(Z):
4. e\_x = np.exp(Z - np.max(Z))
5. return e\_x / e\_x.sum(axis=0)

前向传播的计算公式如下：

这里，代表输入权重（input\_weights），代表内部状态权重（internal\_state\_weights），代表输出权重（output\_weights）。输入权重与输入（）相乘，内部状态权重与前一次激活值相乘，而在我们的表示法中，前一次激活值被称为prev\_memory。在这里我们忽略Bias，前向传播的代码如下：

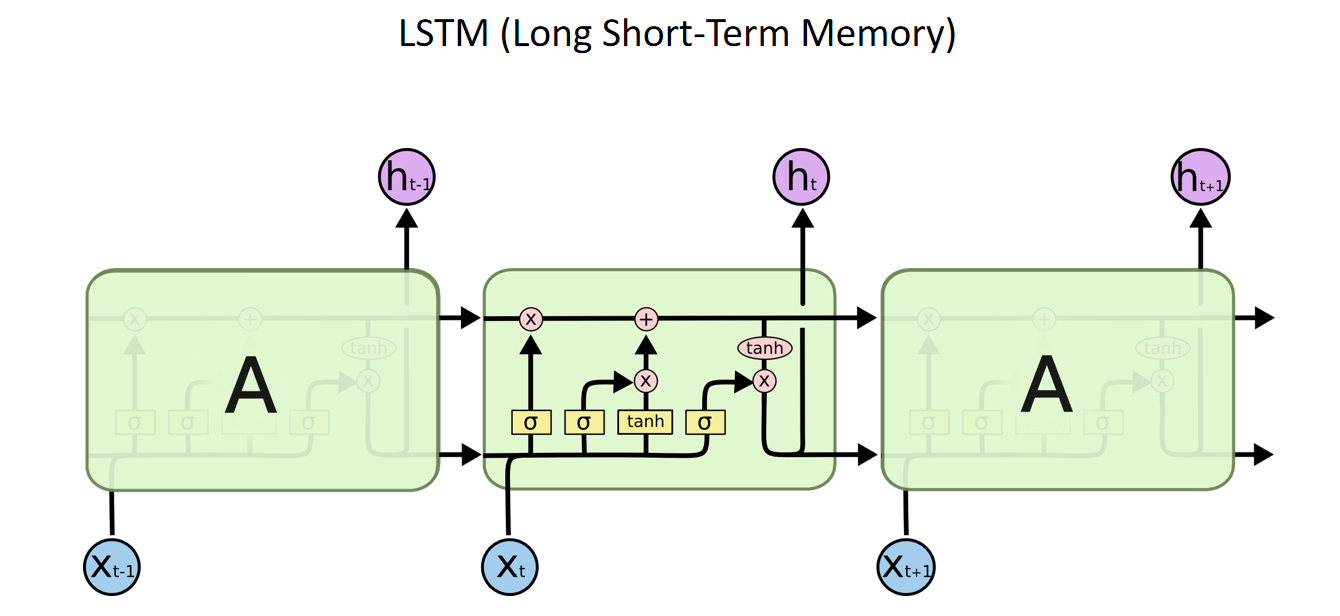
1. def Rnn\_forward(input\_embedding, input\_weights, internal\_state\_weights, prev\_memory,output\_weights):
2. forward\_params = []
3. W\_frd = np.dot(internal\_state\_weights,prev\_memory)
4. U\_frd = np.dot(input\_weights,input\_embedding)
5. sum\_s = W\_frd + U\_frd
6. ht\_activated = tanh\_activation(sum\_s)
7. yt\_unactivated = np.asarray(np.dot(output\_weights, tanh\_activation(sum\_s)))
8. yt\_activated = softmax\_activation(yt\_unactivated)
9. forward\_params.append([W\_frd,U\_frd,sum\_s,yt\_unactivated])
10. return ht\_activated,yt\_activated,forward\_params

## 长短时记忆（LSTM）

### 什么是LSTM

LSTM，全称长短期记忆网络（Long Short-Term Memory，LSTM），是一种循环神经网络（Recurrent Neural Network，RNN），它可以处理序列数据，如文本、语音、视频等。LSTM的特点是能够有效地解决长期依赖问题，即在处理长序列时，能够保留之前的重要信息，并且避免梯度消失或爆炸的问题。

LSTM的结构如图



假设我们有一个当前发生的事件，我们需要预测游戏中的下一个事件。我们也有在游戏中收集的长期和短期信息。长期记忆是很久以前收集的记忆，短期记忆是在几个时间戳之前收集的信息。为了预测下一个事件，我们显然需要当前的事件。但只有一些长期和短期记忆才是有用的。我们如何获得这些？

在LSTM中，我们将使用这个长期记忆和短期记忆，以及当前的事件，来生成一个新的修改过的长期记忆。在这样做的时候，我们将只记住那些有用的东西，我们将抛弃所有不相关的信息。同样地，我们将通过使用一些信息和抛弃其他信息来更新短期记忆。简而言之，在每个时间步骤，我们将过滤需要传递到下一个时间步骤的记忆。这些经过修改的信息被用来预测下一个事件。

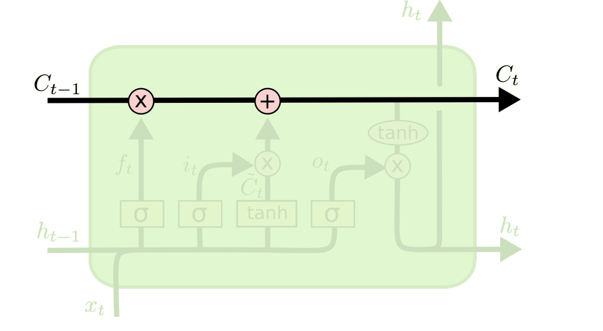
LSTM在自然语言处理、语音识别、机器翻译、图像描述等领域都有广泛的应用，并且有许多变体和改进方法。下一节将具体介绍LSTM的结构。

### 详解LSTM结构

LSTM和RNN在结构上的区别。相对于RNN只有一个传递状态，LSTM有两个传输状态 （cell state）和（hidden state）。

**Cell**

LSTM中的核心概念：cell



相对于RNN，LSTM引入了一个新的隐状态单元状态cell state,如上图中的水平线，他类似于一个传送带，贯穿整个LSTM的计算过程。

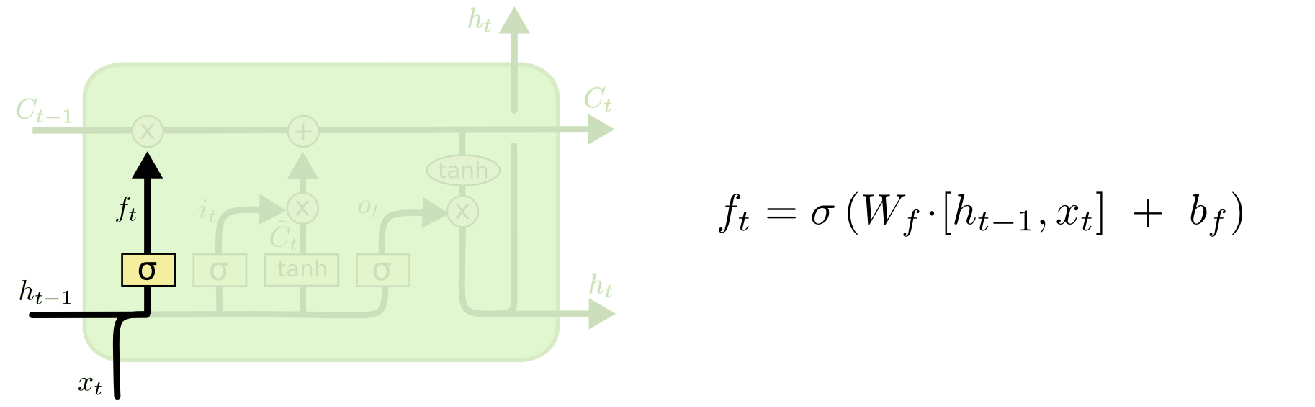
cell state保存的是当前时间隐状态信息，隐状态信息包含来自之前时间步的隐状态和当前时间步的临时隐状态。

接下来介绍LSTM中的3个门单元。

**遗忘门：**决定在新的储存单元中cell需要丢掉/遗忘哪些信息（这里的“信息”指的是cell state中存储的信息）

正如我们前面所讨论的，我们可以用门来添加和删除单元状态的信息。下图只不过是遗忘门，它只过滤需要的信息，并删除其余的信息。这是如何实现的呢？

在这个图中，将上一个时间步骤的单元状态或长期记忆与函数ft相乘，得到新的经过过滤的记忆，其中ft是遗忘因子。遗忘因子使用下面所示的公式计算。

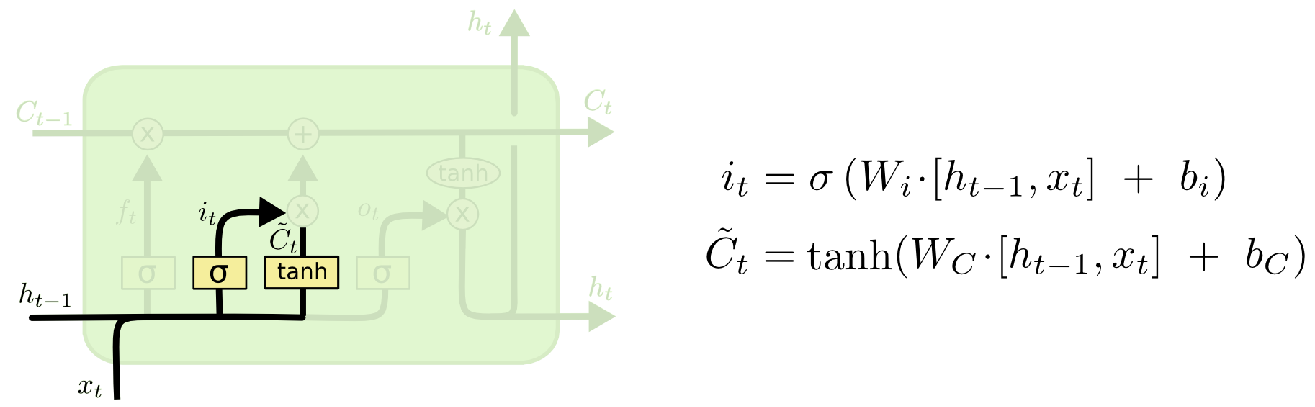


使用上一个时间戳的短期记忆和当前事件来计算遗忘因子。将短期记忆和当前事件连接起来，然后在该向量上应用一个Sigmoid层。Sigmoid函数将产生一个范围从0到1的输出，然后将其与上一个细胞状态中的每个值相乘。0值表示信息将被完全丢弃，而值为1表示信息保持不变。

**输入门：**决定在cell里存储哪些新的信息

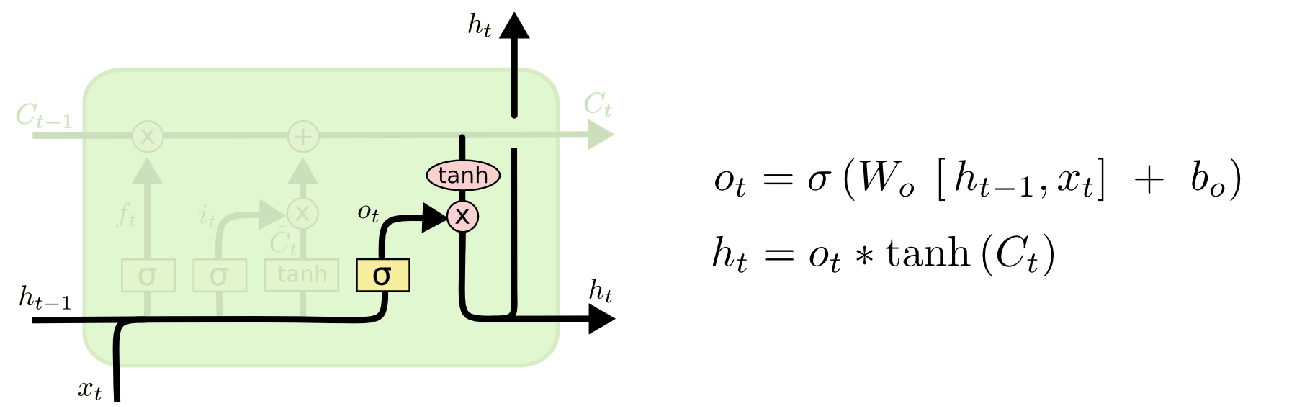
将先前的短期记忆和当前事件连接起来，然后通过一个tanh层。这将生成新的值，即新的信息。然而，我们并不需要整个新的信息。我们如何忽略其中的一部分呢？

每个新候选值的更新程度是通过另一个遗忘门来确定的。遗忘门的输出将与我们的新信息相乘，并生成最终的输出。

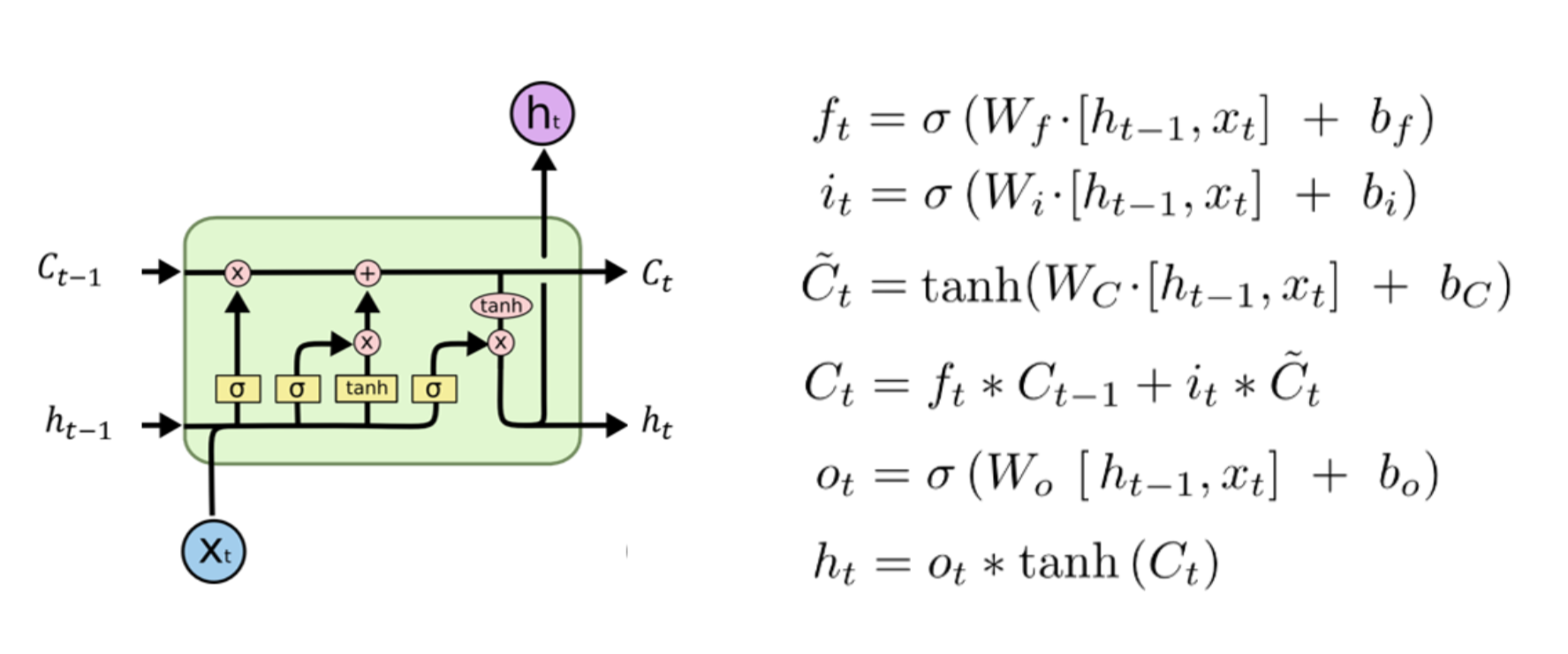


为什么在某些地方使用tanh激活函数？tanh激活函数会输出在-1到1范围内的向量，其值以0为中心。这将很好地分配梯度，并允许细胞状态运行更长时间。最终，这将解决梯度消失或梯度爆炸的问题。

**输出门**：从当前时刻的隐状态中，输出哪些信息到中



总结如下：



### LSTM的简单实现

遗忘门的代码实现：新事件和上一个周期的隐藏状态被逐元素相加，然后通过一个Sigmoid函数进行转换。因此，该输出是一个介于0和1之间的向量。当上一个长期记忆状态与该向量逐元素相乘时，效果是上一个长期记忆状态中每个值的一部分（介于0和1之间）通过门被保留，而其余部分被遗忘

1. import numpy as np
2. from scipy.special import expit as sigmoid
4. def forget\_gate(x, h, Weights\_hf, Bias\_hf, Weights\_xf, Bias\_xf, prev\_cell\_state):
5. forget\_hidden = np.dot(Weights\_hf, h) + Bias\_hf
6. forget\_eventx = np.dot(Weights\_xf, x) + Bias\_xf
7. return np.multiply( sigmoid(forget\_hidden + forget\_eventx), prev\_cell\_state )

输入门包括两个组成部分，一种是“忽略”新信息的方式，另一种是“学习”新信息的方式。在每种情况下，新事件和上一个周期的隐藏状态被相加并进行转换。忽略组件的转换逻辑与遗忘门类似：通过Sigmoid函数创建一个比例向量（取值介于0和1之间）。学习组件使用双曲正切函数进行转换，返回一个取值介于-1和1之间的向量；这有助于模型学习数据中的正向和负向关系。当学习组件逐元素与忽略组件相乘时，效果是学习组件中每个值的一部分（比例介于0和1之间）通过门被保留，而其余部分被忽略。代码实现如下：

1. def input\_gate(x, h, Weights\_hi, Bias\_hi, Weights\_xi, Bias\_xi, Weights\_hl, Bias\_hl, Weights\_xl, Bias\_xl):
2. ignore\_hidden = np.dot(Weights\_hi, h) + Bias\_hi
3. ignore\_eventx = np.dot(Weights\_xi, x) + Bias\_xi
4. learn\_hidden = np.dot(Weights\_hl, h) + Bias\_hl
5. learn\_eventx = np.dot(Weights\_xl, x) + Bias\_xl
6. return np.multiply( sigmoid(ignore\_eventx + ignore\_hidden), np.tanh(learn\_eventx + learn\_hidden) )

在每个时间步骤中，细胞状态（Cell State）的计算是通过将两个向量相加得到的：来自遗忘门（Forget Gate）的向量和来自输入门（Input Gate）的向量。细胞状态在输出门（Output Gate）中被用于确定模型的当前输出；它也会被保留下来，以便在下一个事件的前向传递中使用。

1. def cell\_state(forget\_gate\_output, input\_gate\_output):
2. return forget\_gate\_output + input\_gate\_output

输出门（Output Gate）返回一个向量，既是该事件的模型输出，也是新的隐藏状态h（STM）。这个隐藏状态会在下一个事件的前向传递中被保留下来。细胞状态、前一个隐藏状态和新事件都对该向量作出贡献：新事件和前一个隐藏状态被合并，并通过转换后的细胞状态进行逐元素相乘。

1. def output\_gate(x, h, Weights\_ho, Bias\_ho, Weights\_xo, Bias\_xo, cell\_state):
2. out\_hidden = np.dot(Weights\_ho, h) + Bias\_ho
3. out\_eventx = np.dot(Weights\_xo, x) + Bias\_xo
4. return np.multiply( sigmoid(out\_eventx + out\_hidden), np.tanh(cell\_state) )

通常，LSTM会将输出传送到最终的全连接线性层。

1. input\_size = 2
2. hidden\_dim = 3
3. output\_size = 1
5. def model\_output(lstm\_output, fc\_Weight, fc\_Bias):
6. return np.dot(fc\_Weight, lstm\_output) + fc\_Bias

## 门控循环单元（GRU）

### GRU的基本结构和基本原理

门控循环单元的结构包括两个门：更新门和重置门，以及一个候选隐藏状态和一个隐藏状态.。更新门用于控制隐藏状态的更新程度，它可以学习在每个时间步保留多少过去的信息和接受多少当前的信息。重置门用于控制隐藏状态的重置程度，它可以学习在每个时间步忽略多少过去的信息和复位多少当前的信息。候选隐藏状态是根据当前的输入和重置后的上一时刻的隐藏状态计算得到的，它可以看作是当前时刻的新信息。隐藏状态是根据更新门的输出和候选隐藏状态的输出组合得到的，它可以看作是当前时刻的记忆。

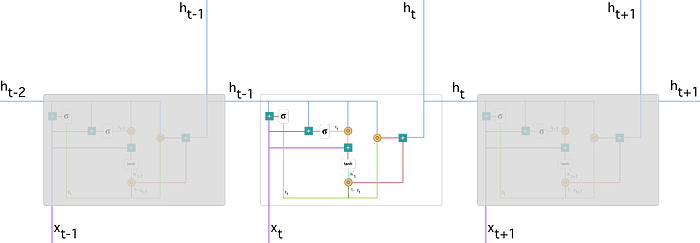
**GRU如何解决梯度消失的问题？**

为了解决基本的递归神经网络运行过程中经常遇到的梯度消失（爆炸）问题，人们开发了许多变体。最著名的变体之一是LSTM。其中一个不太知名但同样有效的变体是GRU。

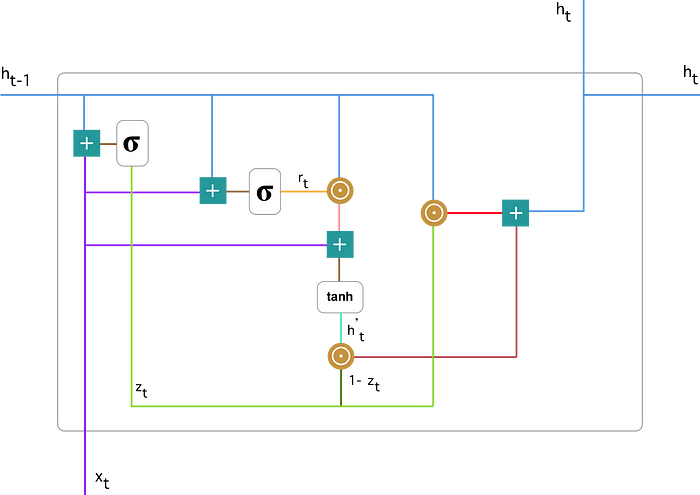
门控循环单元的原理是通过更新门和重置门来控制信息的流动，从而解决梯度消失问题。

更新门可以帮助捕捉长期依赖关系，因为它可以让隐藏状态在一段时间内保持不变，从而保存历史信息。重置门可以帮助捕捉短期依赖关系，因为它可以让隐藏状态在某些时间步忽略不相关的历史信息，从而重点关注当前信息。

### 详解GRU结构



下图是此GRU单元的细节：

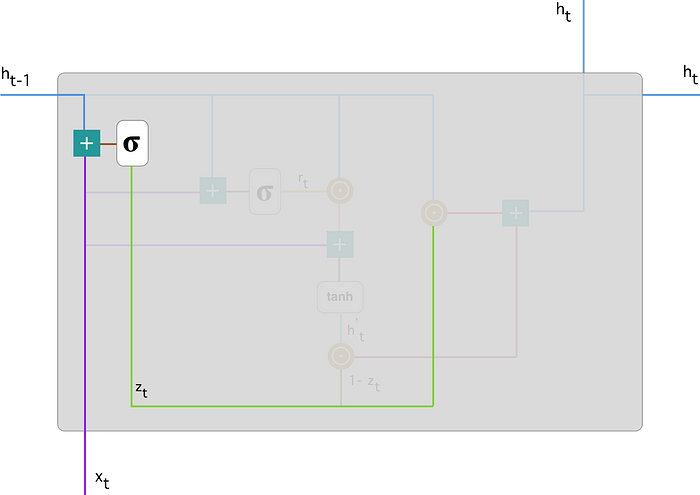


其中σ指激活函数sigmoid，接下来介绍GRU中的更新门和重置门：

**更新门**

我们首先用公式计算时间步骤t的更新门：

当被插入网络单元时，它被乘以它自己的权重。也是如此，它保存了前t-1个单元的信息，并被乘以自己的权重。这两个结果加在一起，然后应用一个sigmoid激活函数，将结果限制在在0和1之间：

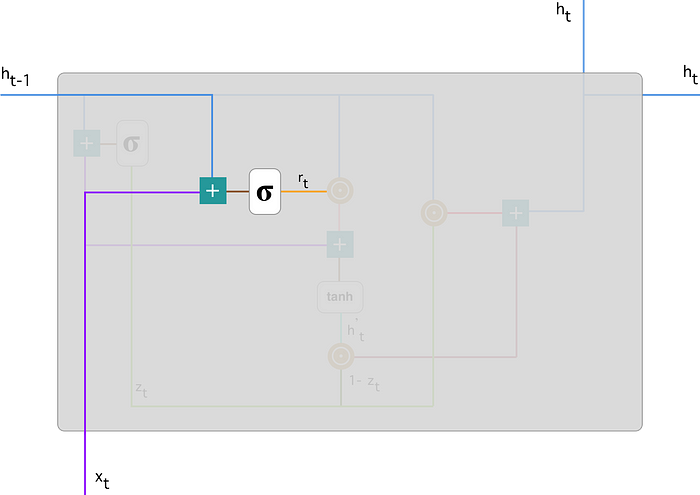


更新门帮助模型确定有多少过去的信息（来自以前的时间步骤）需要传递到未来。因为模型可以决定从过去复制所有的信息，消除梯度消失问题的风险。我们将在后面看到更新门的用法。现在请记住的公式。

**重置门**

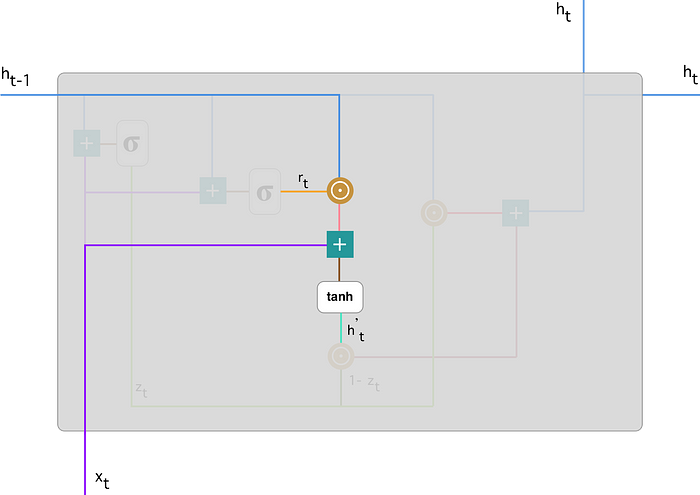
从本质上讲，这个门是由模型来决定遗忘多少过去的信息的。为了计算它，我们使用：

这个公式和更新门的公式是一样的。区别在于权重和门的使用，这一点我们稍后会看到。下面的图示显示了重置门的位置：



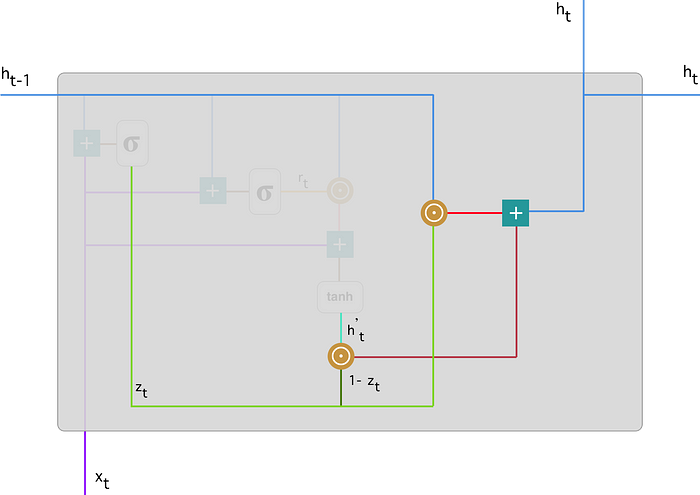
让我们看看这些门到底会如何影响最终的输出。首先，我们从重置门的使用开始。我们引入一个新的内存内容，它将使用重置门来存储过去的相关信息。它的计算方式如下：

1. 输入的乘以一个权重，乘以一个权重。
2. 计算重置门和之间的Hadamard积。这将决定从以前的时间步骤中删除什么。假设我们有一个情感分析问题，从一个人写的评论中确定他对一本书的看法。文本以 "这是一本幻想书，说明了...... "开始，在几段之后以 "我不太喜欢这本书，因为我认为它抓住了太多的细节 "结束。为了确定该书的总体满意程度，我们只需要评论的最后一部分。在这种情况下，当神经网络接近文本的结尾时，它将学会把向量分配到接近0，洗掉过去的内容，只关注最后的句子。
3. 将步骤1和2的结果加起来。
4. 应用非线性激活函数。



作为最后一步，网络需要计算--持有当前单元信息的向量，并将其传递给网络。为了做到这一点，就需要更新门。它决定从当前的内存内容--和从以前的步骤--收集什么。具体操作如下：

1. 对更新门和应用元素相乘法。
2. 对和应用元素相乘法。
3. 将步骤1和2的结果加起来。



### 与LSTM的区别

GRU和LSTM都是循环神经网络的变体，它们都可以处理序列数据，并且通过门机制来缓解梯度消失和长期依赖的问题。但是它们存在着一些区别：首先，LSTM有三个门（输入门、遗忘门和输出门），而GRU只有两个门（重置门和更新门）。其次LSTM有两个状态（cell state和隐藏状态hidden state），而GRU只有一个状态（隐藏状态hidden state）。LSTM的cell state是用来保存长期记忆的信息载体，而GRU的隐藏状态hidden state既包含了长期记忆又包含了短期记忆。LSTM在每个时间步都会输出隐藏状态，而GRU会根据更新门的值来决定是否输出隐藏状态。

一般来说，LSTM比GRU更复杂，参数更多，训练更困难，但也可能更精确。GRU比LSTM更简单，参数更少，训练更快，但也可能更粗糙。它们之间的性能优劣并没有定论，需要根据具体的任务和数据集来选择合适的模型。

## 参考文献

1. 李沐，动手学深度学习
2. 李宏毅机器学习课程
3. MIT 6.S191