

法律声明

■ 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，北风网和讲师拥有完全知识产权；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或者机构不得盗版、复制、仿造其中的创意和内容，我们保留一切通过法律手段追究违反者的权利。

■ 课程详情请咨询

◆ 微信公众号：北风教育

◆ 官方网址：<http://www.ibeifeng.com/>



人工智能之深度学习

循环神经网络(RNN)

主讲人: Vincent Ying

上海育创网络科技有限公司



课程要求

■ 课上课下 “九字” 真言

- ◆ 认真听，善摘录，勤思考
- ◆ **多温故，乐实践**，再发散

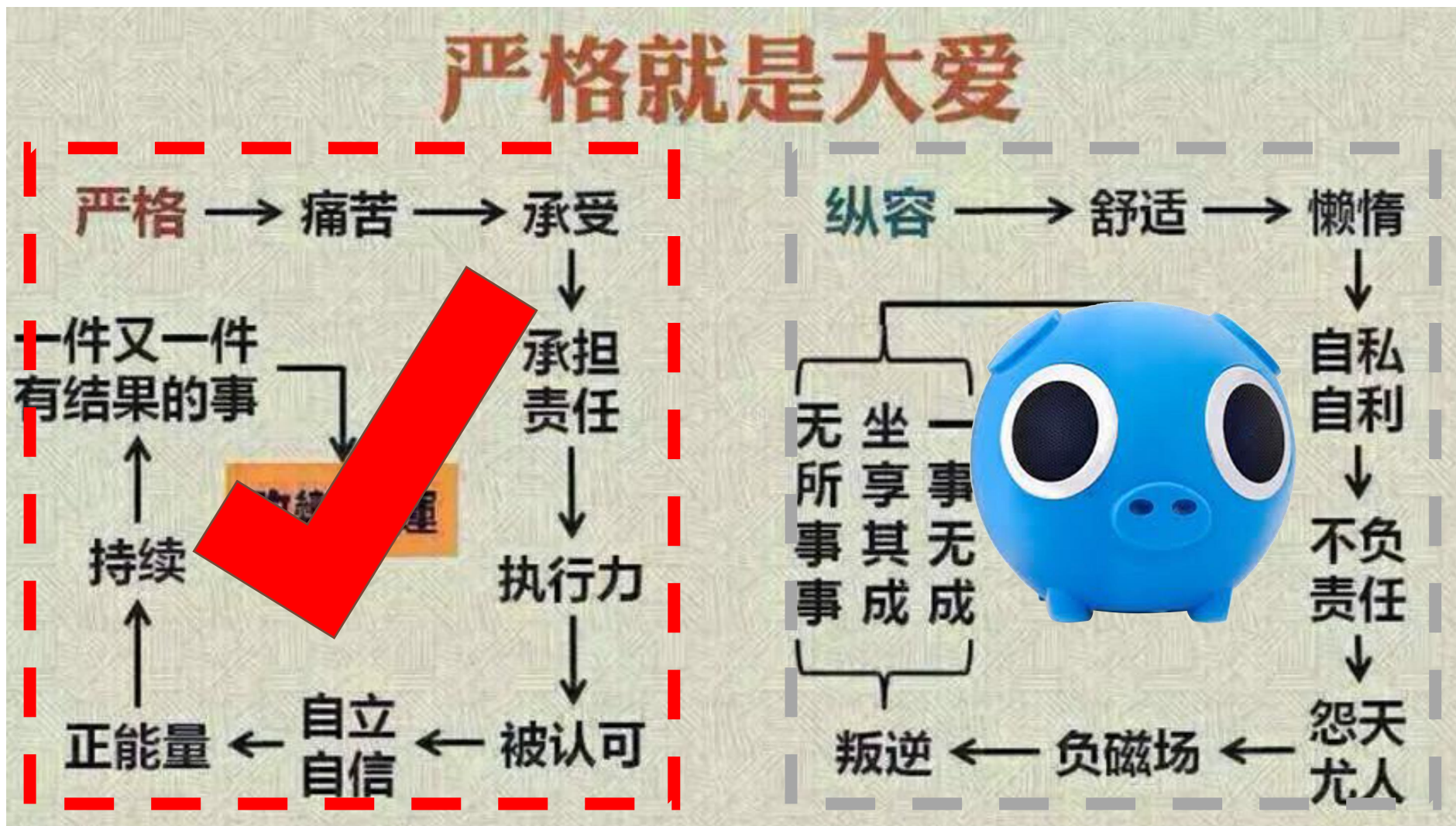
■ 四不原则

- ◆ **不懒散惰性，不迟到早退**
- ◆ **不请假旷课，不拖延作业**

■ 一点注意事项

- ◆ 违反 “四不原则” ， 不包就业和推荐就业

严格是大爱



寄语



做别人不愿做的事，
做别人不敢做的事，
做别人做不到的事。

课程内容

- 一、递归神经网络(RNN)
- 二、RNN反向传播
- 三、Word2vec
- 四、双向RNN
- 五、LSTM
- 六、RNN超参数

什么是递归神经网络

- 为什么有BP神经网络、CNN，还需要RNN？
 - ◆ BP神经网络和CNN的输入(问题)输出都是互相独立的；但是实际应用中有些场景输出内容和之前的内容是有关联的。
 - ◆ RNN引入“记忆”的概念；递归指其每一个元素都执行相同的任务，但是输出依赖于输入(问题)和“记忆”。

什么是递归神经网络

我们已经学习了前馈网络的两种结构——bp神经网络和卷积神经网络，这两种结构有一个特点，就是假设输入(问题)是一个独立的没有上下文联系的单位，比如输入(问题)是一张图片，网络识别是狗还是猫。但是对于一些有明显的上下文特征的序列化输入(问题)，比如预测视频中下一帧的播放内容，那么很明显这样的输出必须依赖以前的输入(问题)，也就是说网络必须拥有一定的“记忆能力”。为了赋予网络这样的记忆力，一种特殊结构的神经网络——递归神经网络(Recurrent Neural Network)便应运而生了。

递归神经网络RNN-应用场景

■ 自然语言处理(NLP)

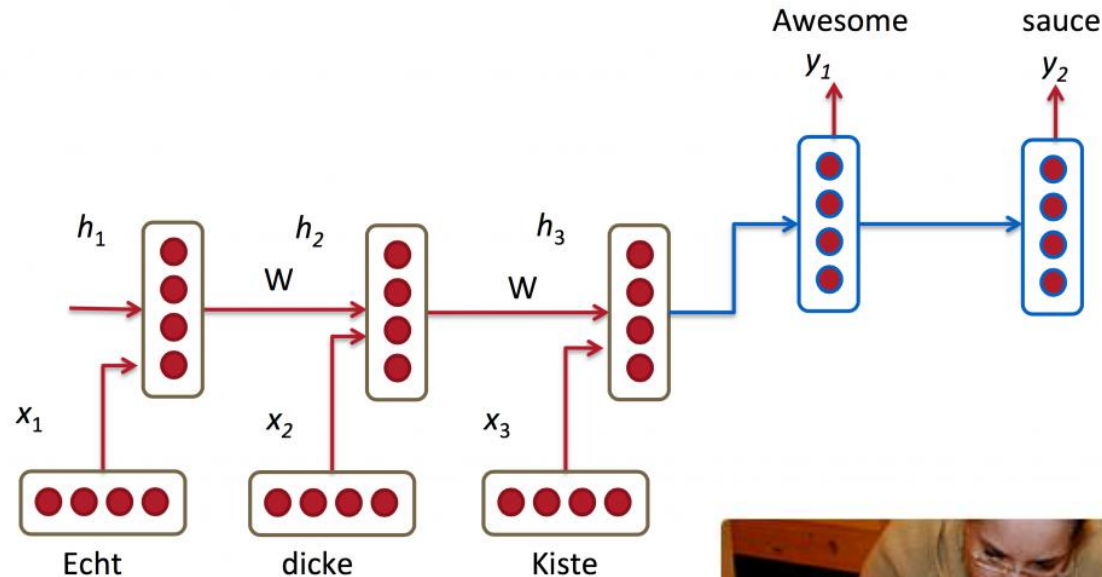
◆ 语言模型与文本生成

■ 机器翻译

■ 语音识别

■ 图像描述生成

■ 文本相似度计算等

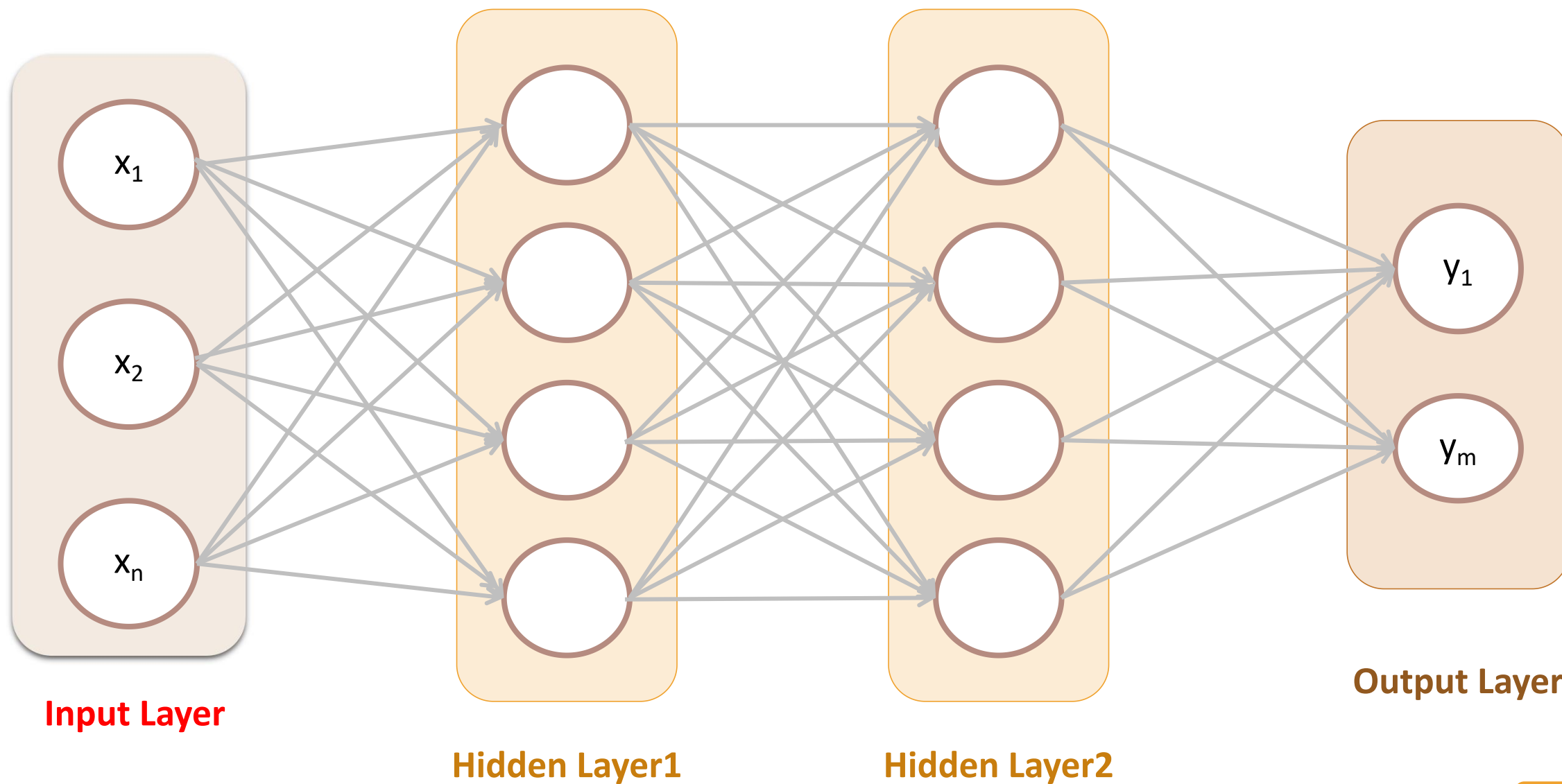


"man in black shirt is playing guitar."

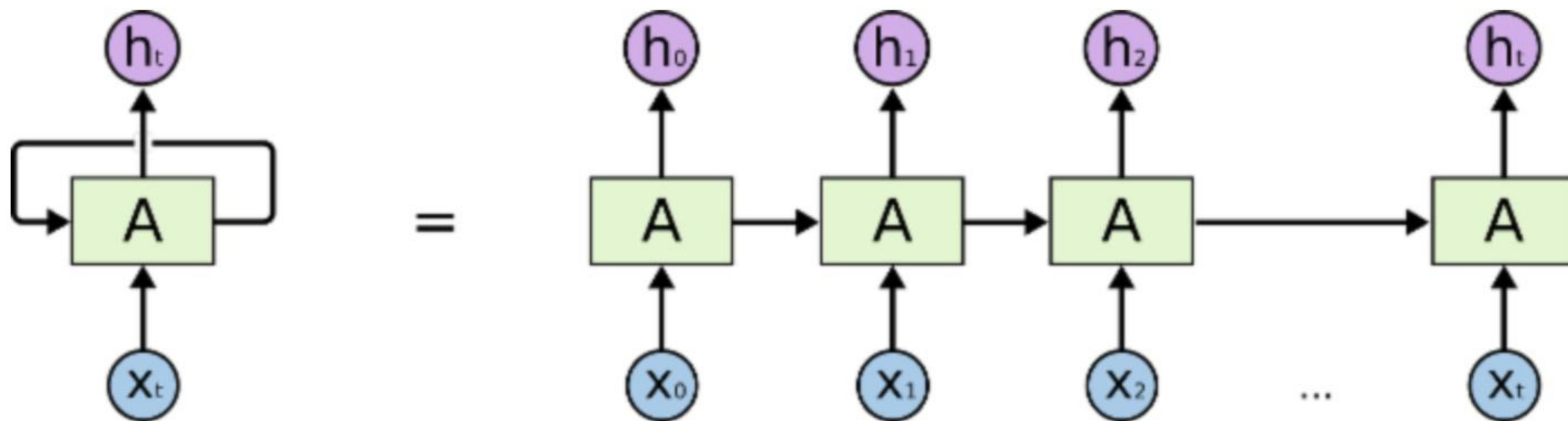


"two young girls are playing with lego toy."

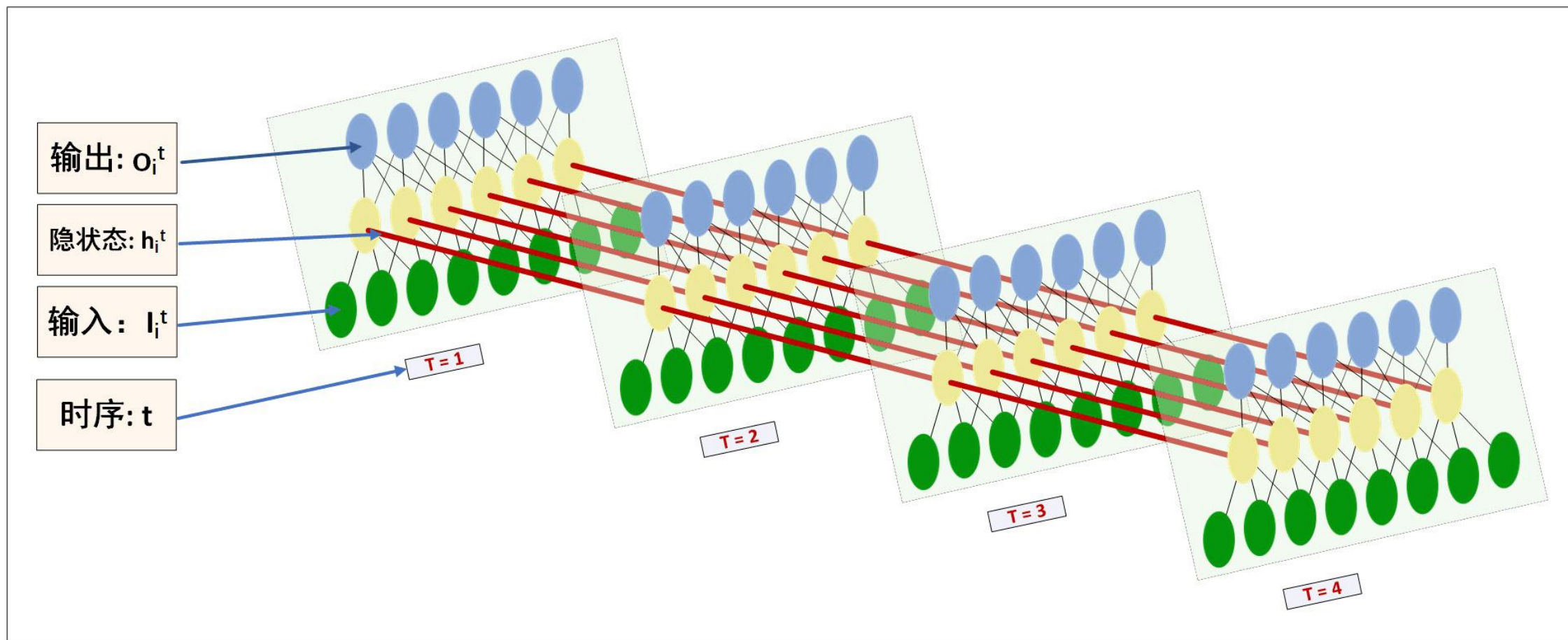
神经网络之结构



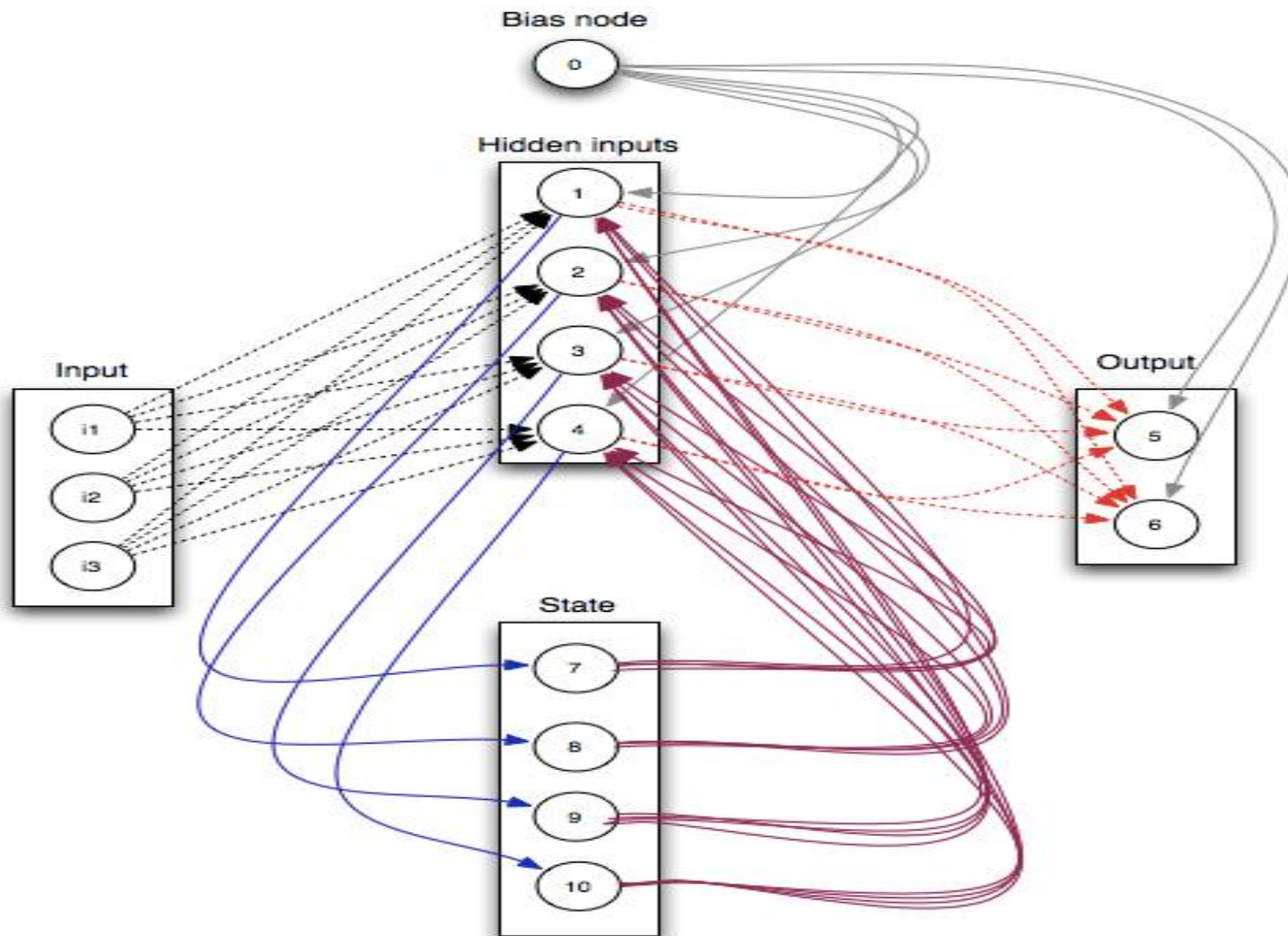
递归神经网络RNN-结构



递归神经网络RNN-结构



循环神经网络RNN-结构



循环神经网络RNN-结构

- 网络某一时刻的输入(问题) x_t ，和之前介绍的bp神经网络的输入(问题)一样， x_t 是一个n维向量，不同的是递归网络的输入(问题)将是一整个序列，也就是 $x=[x_1, \dots, x_{t-1}, x_t, x_{t+1}, \dots, x_T]$ ，对于语言模型，每一个 x_t 将代表一个词向量，一整个**序列**就代表一句话。
- h_t 代表时刻t隐神经元对于线性转换值， s_t 代表时刻t的隐藏状态
- o_t 代表时刻t的输出
- 输入(问题)层到隐藏层直接的权重由U表示
- 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

递归神经网络RNN正向传播阶段

- 在 $t=1$ 的时刻， U, V, W 都被随机初始化好， s_0 通常初始化为0，然后进行如下计算：

$$h_1 = Ux_1 + Ws_0$$

$$s_1 = f(h_1)$$

$$o_1 = g(Vs_1)$$

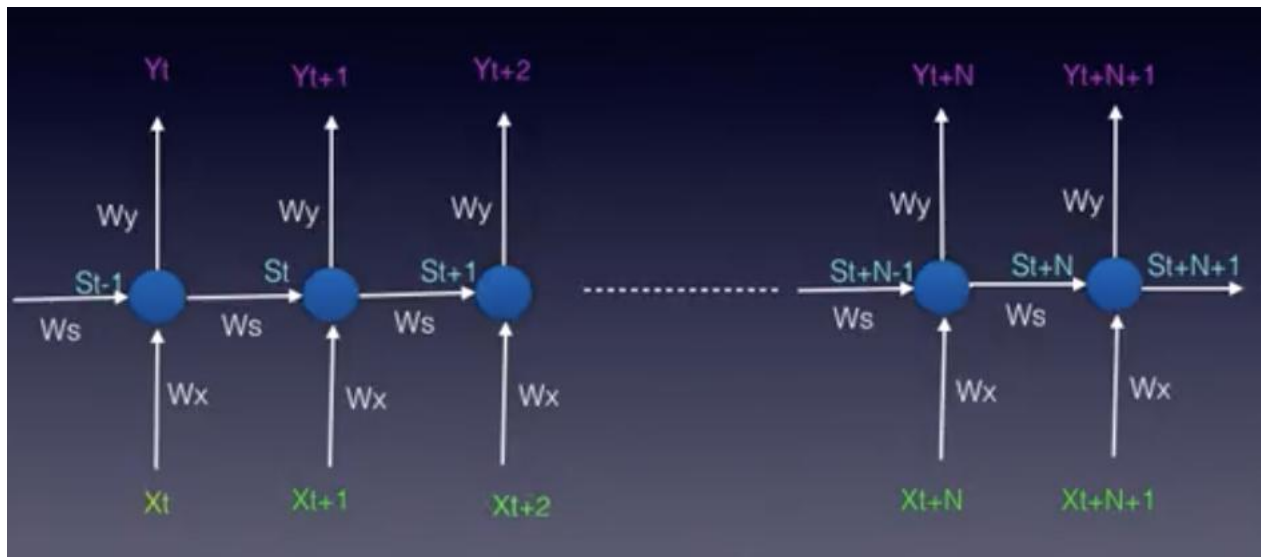
- 时间就向前推进，此时的状态 s_1 作为时刻1的记忆状态将参与下一个时刻的预测活动，也就是：

$$h_2 = Ux_2 + Ws_1$$

$$s_2 = f(h_2)$$

$$o_2 = g(Vs_2)$$

RNN展开模型



$$\bar{s}_t = \Phi(\bar{x}_t W_x + \bar{s}_{t-1} W_s)$$

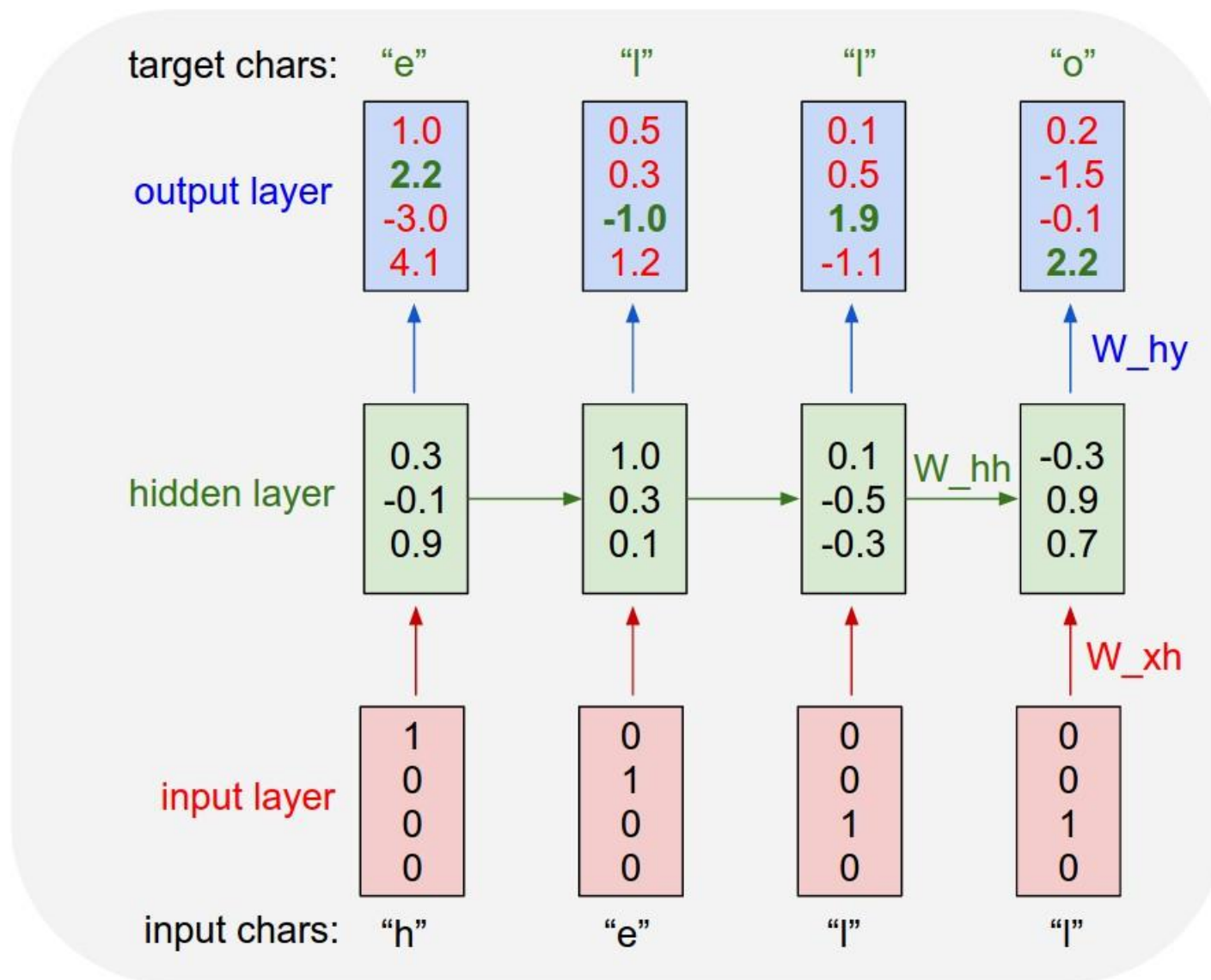
$$\bar{y}_t = \sigma(\bar{s}_t W_y)$$

举例-RNN是如何工作的?



- 1、对检测目标(回答)-我爱中国one-hot编码;
- 2、设置目标(回答), 一旦检测到, 标签为1; (人为设置的)
- 3、最终迫使模型学习, 如概率超过0.9认为检测出来了。

RNN



递归神经网络RNN正向传播阶段

- 以此类推，可得

$$h_t = Ux_t + Ws_{t-1}$$

$$s_t = f(h_t)$$

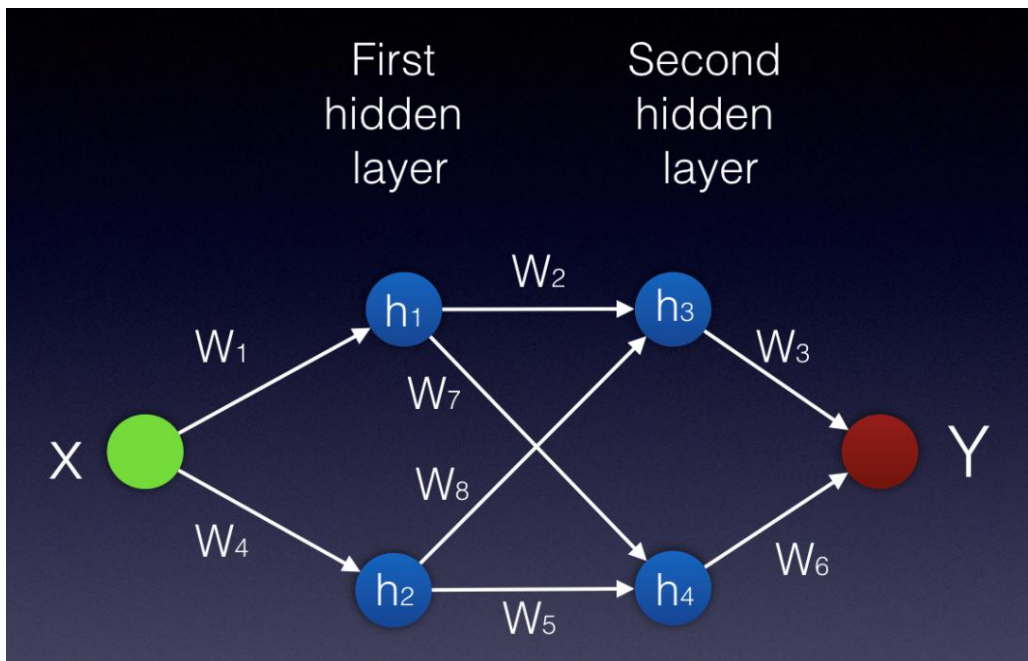
$$o_t = g(Vs_t)$$

- 其中f可以是tanh,relu,sigmoid等激活函数，g通常是softmax也可以是其他。

值得注意的是，我们说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(问题)状态进行总结，而作为下次输入(问题)的辅助。可以这样理解隐藏状态： $h=f(\text{现有的输入(问题)} + \text{过去记忆总结})$

RNN-反向传播

回顾普通反向传播-练习



Equation A
$$\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial h_1} \frac{\partial h_1}{\partial W_1} + \frac{\partial y}{\partial h_4} \frac{\partial h_4}{\partial h_1} \frac{\partial h_1}{\partial W_1}$$

Equation B
$$\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial h_1} \frac{\partial h_1}{\partial W_1} + \frac{\partial y}{\partial h_4} \frac{\partial h_4}{\partial h_2} \frac{\partial h_2}{\partial W_1}$$

Equation C
$$\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial h_3} \frac{\partial h_3}{\partial h_1} \frac{\partial h_1}{\partial W_1}$$

Equation D
$$\frac{\partial y}{\partial W_1} = \frac{\partial y}{\partial h_4} \frac{\partial h_4}{\partial h_1} \frac{\partial h_1}{\partial W_1}$$

请问：W1的梯度是？

基于时间的反向传播-明确我们公式定义

状态向量通过该方式计算: $\bar{s}_t = \Phi(\bar{x}_t W_x + \bar{s}_{t-1} W_s)$

$$\bar{y}_t = \bar{s}_t W_y$$

输出向量

Or

$$\bar{y}_t = \sigma(\bar{s}_t W_y)$$

误差函数 (为了简化
计算-没用交叉熵)

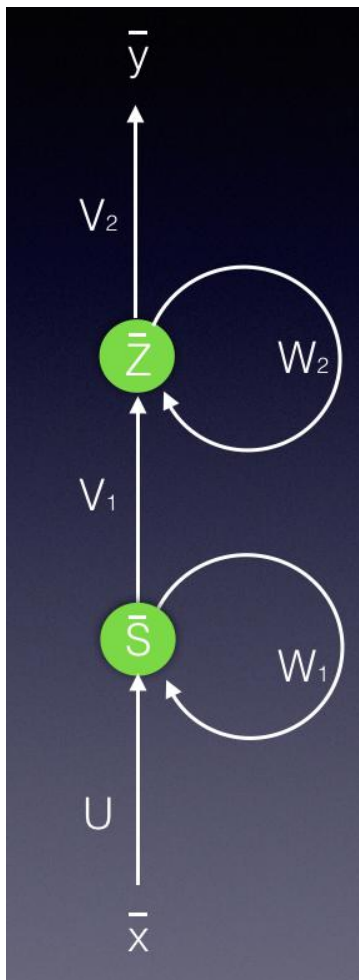
$$E_t = (\bar{d}_t - \bar{y}_t)^2$$

在 **基于时间的反向传播算法** 中, 我们在时间步长 t 训练网络, 也会考虑之前的所有步长。
例如:

我们将计算时间步长 $t=3$ 时**基于时间的反向传播算法**过程。你会发现为了调整所有三个权重矩阵 W_x, W_s, W_y , 我们需要考虑时间步长3、时间步长2和时间步长1。

求: W_s W_x 的反向传播 ★

练习1



思考在循环神经网络折叠模型中。状态 \mathbf{S} 和 \mathbf{Z} 在每层都有多个神经元。状态 \mathbf{Z} 在时间 t 的数学推导为：

Equation A $z_t = \phi(s_t v_1 + z_{t-1} w_2)$

Equation B $\bar{z}_t = \phi(\bar{s}_t v_1 + \bar{z}_t w_2)$

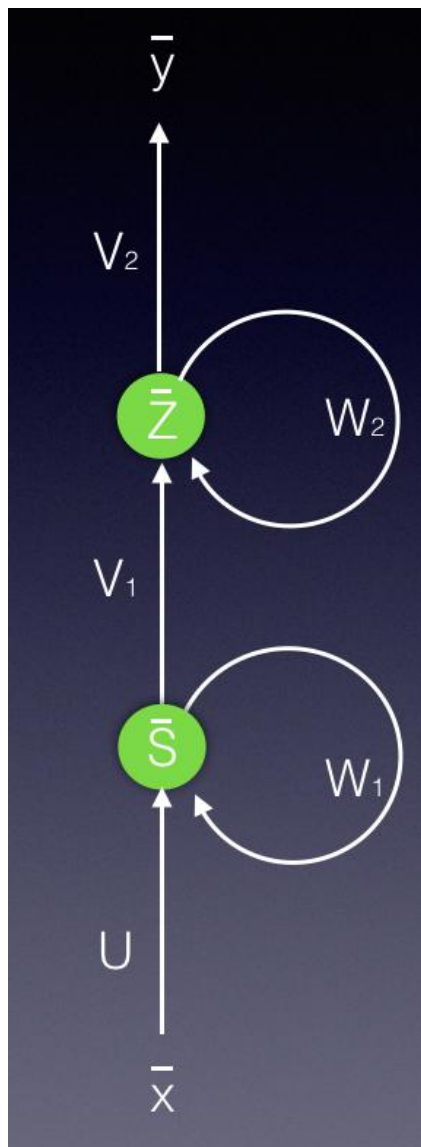
Equation C $z_t = \phi(s_t v_1 + z_t w_2)$

Equation D $\bar{z}_t = \phi(\bar{s}_t v_1 + \bar{z}_{t-1} w_2)$

答案： D

练习2

假设符号**E**表示误差。经过一个时间步长后，权重矩阵**V1**在时间**t**如何更新呢？



Equation A $\Delta v_1 = -\alpha \frac{\partial E_t}{\partial v_1} = -\alpha \frac{\partial E_t}{\partial \bar{y}_t} \frac{\partial \bar{Z}_t}{\partial \bar{v}_1}$

Equation B $\Delta v_1 = -\alpha \frac{\partial E_t}{\partial v_1} = -\alpha \frac{\partial E_t}{\partial \bar{y}_t} \frac{\partial \bar{y}_t}{\partial \bar{Z}_t} \frac{\partial \bar{Z}_t}{\partial \bar{v}_1}$

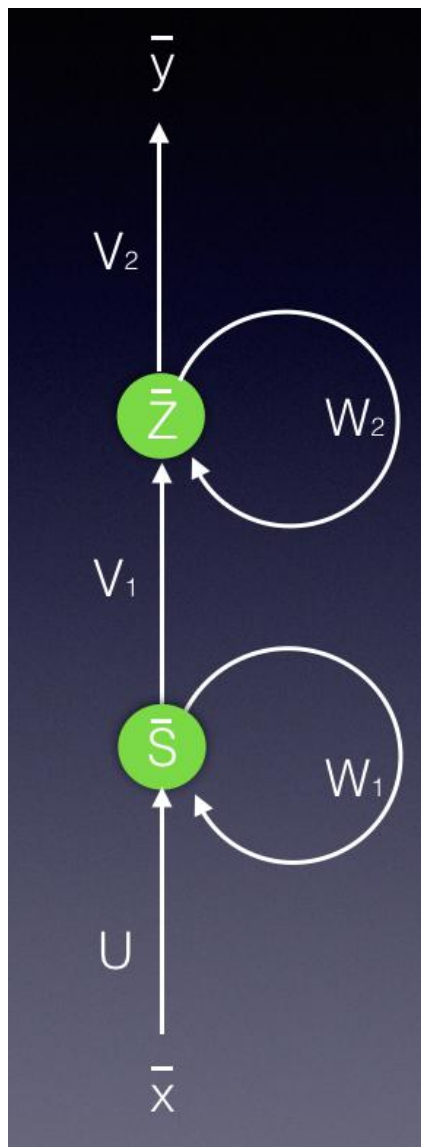
Equation C $\Delta v_1 = \frac{\partial E_t}{\partial v_1} = \frac{\partial E_t}{\partial \bar{y}_t} \frac{\partial \bar{y}_t}{\partial \bar{Z}_t} \frac{\partial \bar{Z}_t}{\partial \bar{v}_1}$

Equation D $\Delta v_1 = -\alpha \frac{\partial E_t}{\partial v_1} = -\alpha \frac{\partial E_t}{\partial \bar{y}_t} \frac{\partial \bar{y}_t}{\partial \bar{v}_1}$

答案： B

练习3

假设符号**E**表示误差。在时间**t+1**（经过两个时间步长），权重矩阵**U**如何更新？提示：使用展开模型，获得更好的可视化效果。



Equation A

$$\frac{\partial E_{t+1}}{\partial U} = \frac{\partial E_{t+1}}{\partial \bar{y}_{t+1}} \frac{\partial \bar{y}_{t+1}}{\partial \bar{z}_{t+1}} \frac{\partial \bar{z}_{t+1}}{\partial \bar{s}_{t+1}} \frac{\partial \bar{s}_{t+1}}{\partial U}$$

Equation B

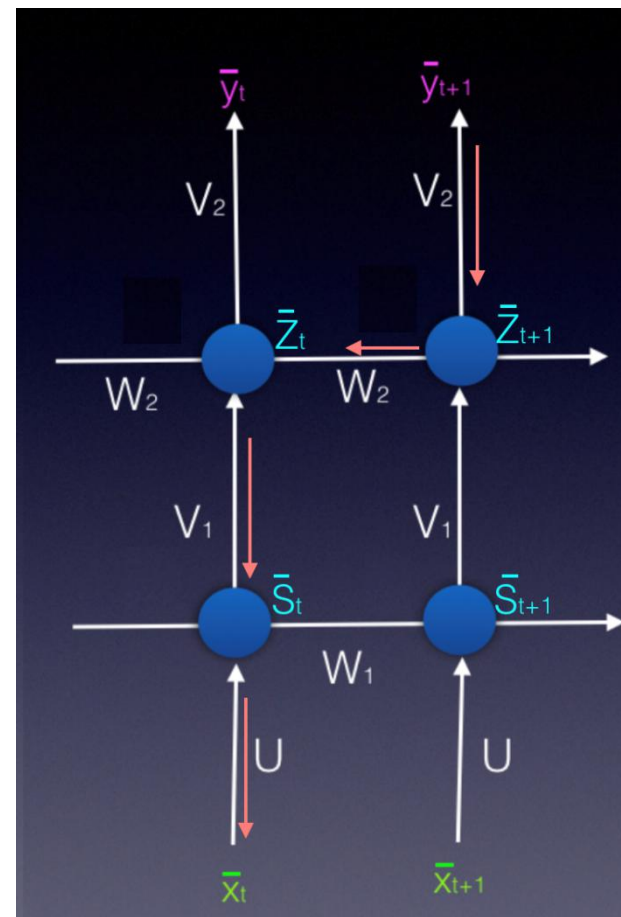
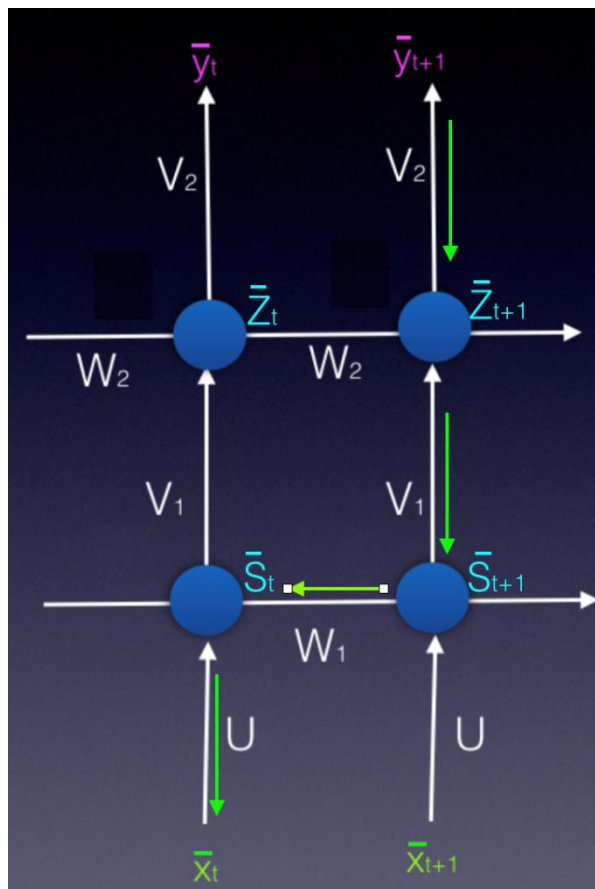
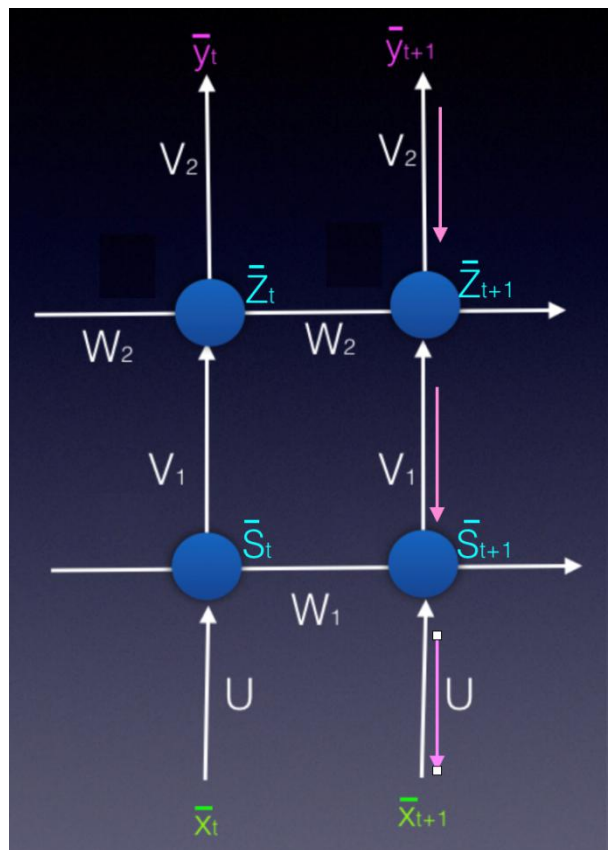
$$\frac{\partial E_{t+1}}{\partial U} = \frac{\partial E_{t+1}}{\partial \bar{y}_{t+1}} \frac{\partial \bar{y}_{t+1}}{\partial \bar{z}_{t+1}} \frac{\partial \bar{z}_{t+1}}{\partial \bar{s}_{t+1}} \frac{\partial \bar{s}_{t+1}}{\partial \bar{s}_t} \frac{\partial \bar{s}_t}{\partial U} + \frac{\partial E_{t+1}}{\partial \bar{y}_{t+1}} \frac{\partial \bar{y}_{t+1}}{\partial \bar{z}_{t+1}} \frac{\partial \bar{z}_{t+1}}{\partial \bar{z}_t} \frac{\partial \bar{z}_t}{\partial \bar{s}_t} \frac{\partial \bar{s}_t}{\partial U}$$

Equation C

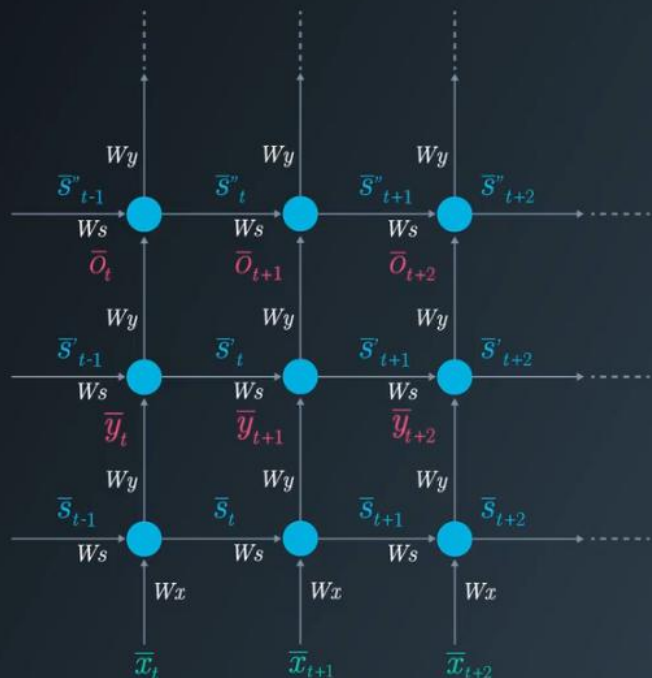
$$\begin{aligned} \frac{\partial E_{t+1}}{\partial U} = & \frac{\partial E_{t+1}}{\partial \bar{y}_{t+1}} \frac{\partial \bar{y}_{t+1}}{\partial \bar{z}_{t+1}} \frac{\partial \bar{z}_{t+1}}{\partial \bar{s}_{t+1}} \frac{\partial \bar{s}_{t+1}}{\partial U} + \frac{\partial E_{t+1}}{\partial \bar{y}_{t+1}} \frac{\partial \bar{y}_{t+1}}{\partial \bar{z}_{t+1}} \frac{\partial \bar{z}_{t+1}}{\partial \bar{z}_t} \frac{\partial \bar{z}_t}{\partial \bar{s}_t} \frac{\partial \bar{s}_t}{\partial U} \\ & + \frac{\partial E_{t+1}}{\partial \bar{y}_{t+1}} \frac{\partial \bar{y}_{t+1}}{\partial \bar{z}_{t+1}} \frac{\partial \bar{z}_{t+1}}{\partial \bar{s}_{t+1}} \frac{\partial \bar{s}_{t+1}}{\partial \bar{s}_t} \frac{\partial \bar{s}_t}{\partial U} \end{aligned}$$

练习3答案

共3条通路



What happens
when we have
many time steps?



使用基于时间的反向传播算法训练循环神经网络时，我们可以选择小批量进行训练，这里我们定期更新批次权重（而不是每次输入(问题)样本）。我们计算每一步的梯度，但不要立即更新权重。另外，我们对权重每次更新固定的步长数量。这样有利于降低训练过程的复杂性，并消除权重更新中的噪音

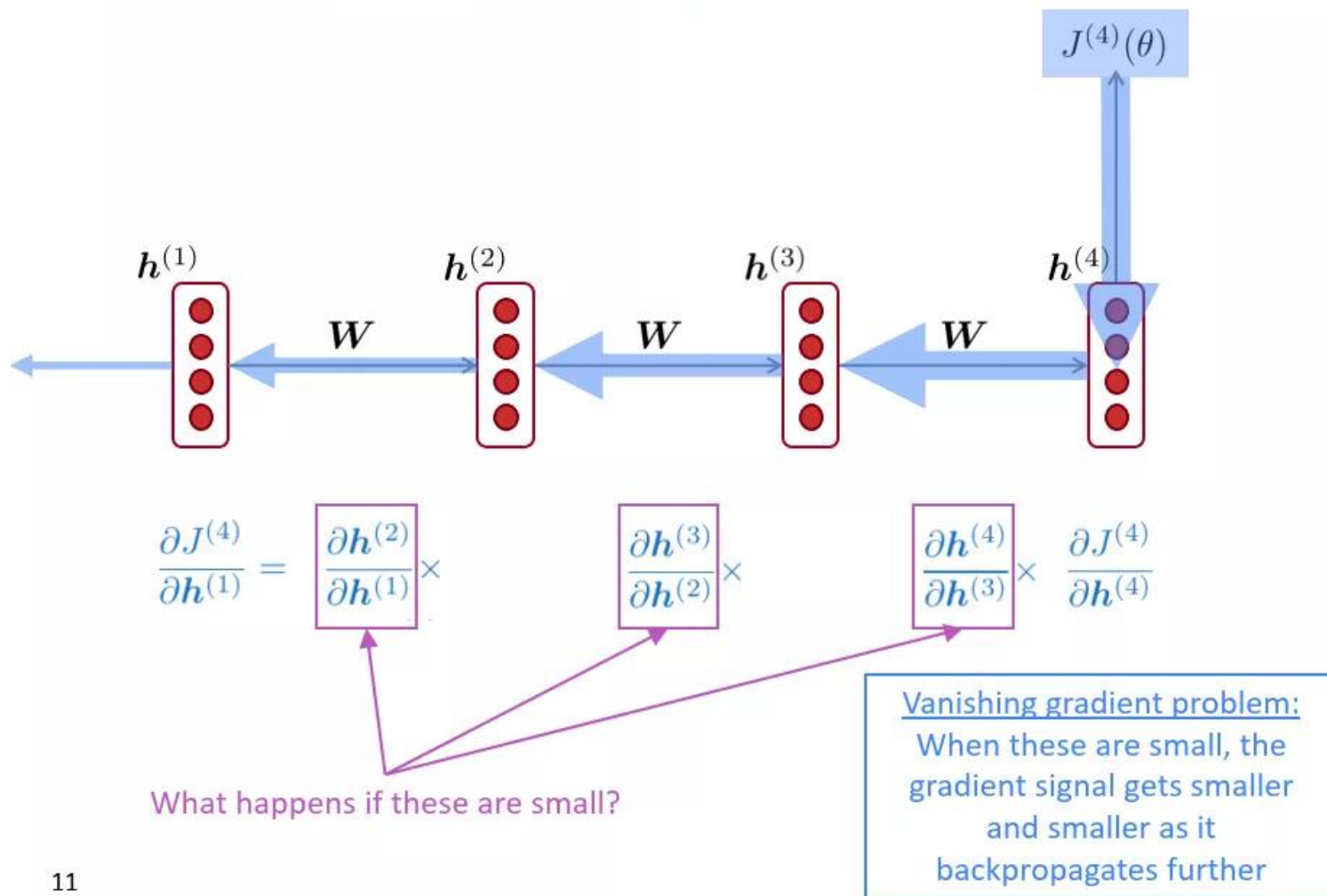
RNN原理就是:传递过程中不断 做乘法。连续乘一个小于1的数字，导致该数趋近于0，反之，趋近于无穷大。

如果反向传播超过10个时间步长，梯度会变得非常小。这种现象称为**梯度消失问题**

在循环神经网络中，我们可能也有相反的问题，称为**梯度膨胀问题**

code01 字符RNN

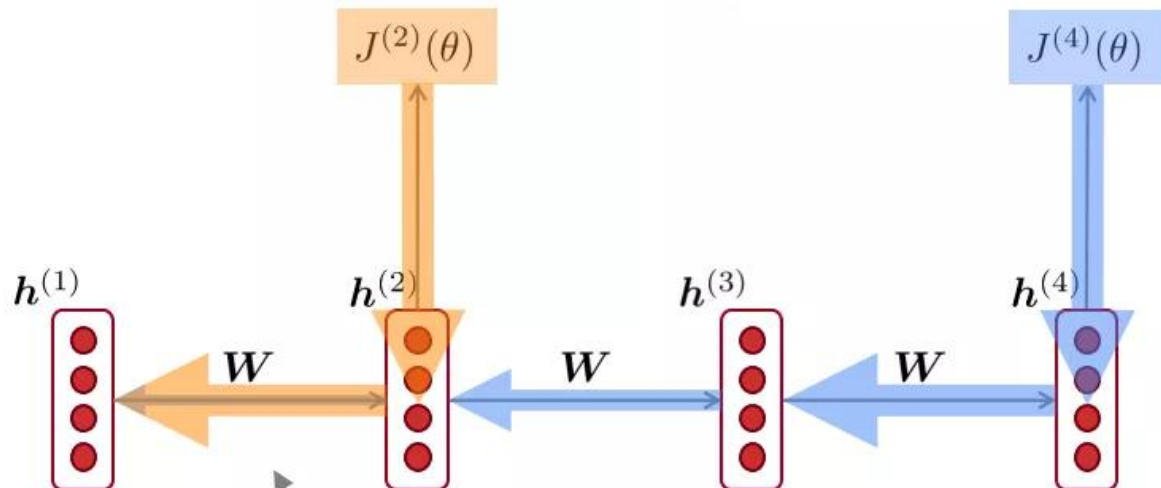
Vanishing gradient intuition



使用基于时间的反向传播算法训练循环神经网络时，核心问题是RNN在不同时间步使用共享参数 W ，导致 $t+n$ 时刻的损失对 t 时刻的参数的偏导数存在 W 的指数形式，一旦 W 很小或者很大就会导致梯度消失和梯度爆炸问题。

如果反向传播超过10个时间步长，梯度会变得非常小。这种现象称为**梯度消失问题**。在循环神经网络中，我们可能也有相反的问题，称为**梯度膨胀问题**。

Why is vanishing gradient a problem?



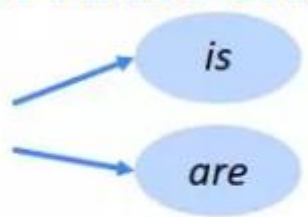


Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

问题：参数更新更多的受到临近词的影响，那些和当前时刻 t 较远的词对当前的参数更新影响很小。

如图所示， $h^{(1)}$ 对 $J^{(2)}(\theta)$ 的影响就比对 $J^{(4)}(\theta)$ 的影响大。因为梯度消失，我们就不知道 t 时刻是真的对 $t+n$ 时刻没影响，还是因为梯度消失导致没学习到这种影响。

Effect of vanishing gradient on RNN-LM

- **LM task:** *The writer of the books* ____ 
- **Correct answer:** *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (correct) 
- **Sequential recency:** *The writer of the books are* (incorrect) 
- Due to vanishing gradient, RNN-LMs are better at learning from **sequential recency** than **syntactic recency**, so they make this type of error more often than we'd like [Linzen et al 2016]

例子：
假设我们需要预测句子The writer of the books下一个单词，由于梯度消失，books对下一个词的影响比writer对下一个词的影响更大，导致模型错误的预测成了are。

Vanilla-RNN问题

Gradient clipping: solution for exploding gradient

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

```

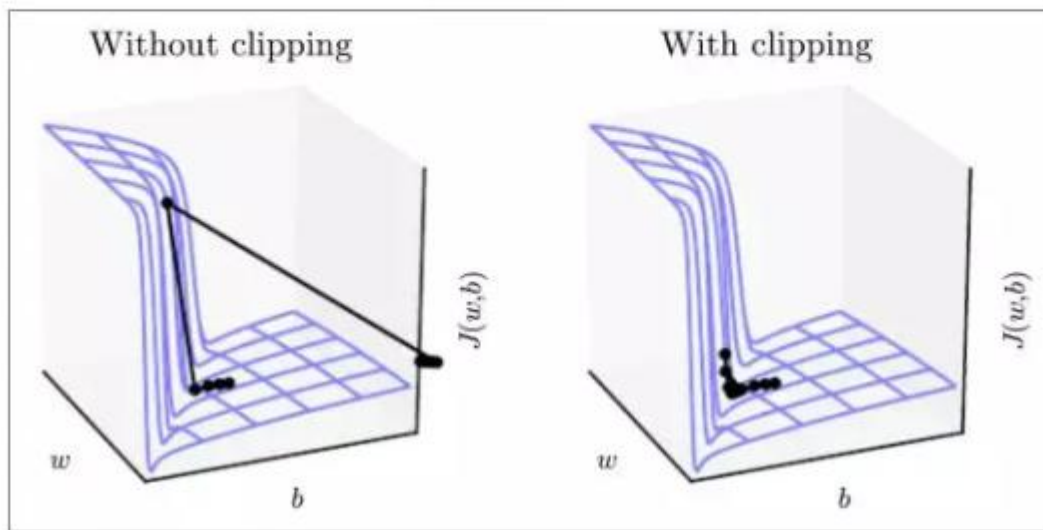
 $\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{\mathbf{g}}\| \geq threshold$  then
     $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$ 
end if

```

梯度爆炸一个比较好的解决方法是梯度裁剪，即如果发现梯度的范数大于某个阈值，则以一定的比例缩小梯度的范数，但不改变其方向。

- Intuition: take a step in the same direction, but a smaller step

Gradient clipping: solution for exploding gradient



- This shows the loss surface of a simple RNN (hidden state is a scalar not a vector)
- The “cliff” is dangerous because it has steep gradient
- On the left, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- On the right, gradient clipping reduces the size of those steps, so effect is less drastic

如图，左子图是没有梯度裁剪的，由于RNN的梯度爆炸问题，导致快接近局部极小值时，梯度很大，参数突然爬上悬崖，然后又飞到右边一个随机的区域，miss掉了中间的局部极小值。

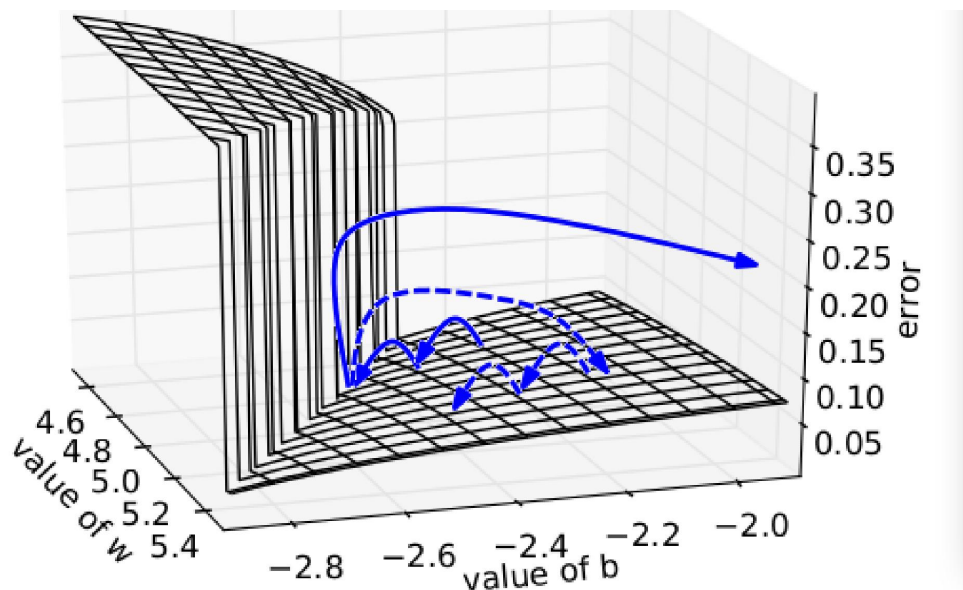
右子图是增加了梯度裁剪之后，更新步伐变小，参数稳定在局部极小值附近。

递归神经网络RNN反向传播阶段

- 为了克服梯度消失的问题，LSTM和GRU模型便后续被推出了，为什么LSTM和GRU可以克服梯度消失问题呢？由于它们都有特殊的方式存储“记忆”，那么以前梯度比较大的“记忆”不会像简单的RNN一样马上被抹除，因此可以一定程度上克服梯度消失问题。
- 另一个简单的技巧可以用来克服梯度爆炸的问题就是gradient clipping，也就是当你计算的梯度超过阈值 c 的或者小于阈值 $-c$ 时候，便把此时的梯度设置成 c 或 $-c$ 。

递归神经网络RNN反向传播阶段

- 下图所示是RNN的误差平面，可以看到RNN的误差平面要么非常陡峭，要么非常平坦，如果不采取任何措施，当你的参数在某一次更新之后，刚好碰到陡峭的地方，此时梯度变得非常大，那么你的参数更新也会非常大，很容易导致震荡问题。而如果你采取了gradient clipping这个技巧，那么即使你不幸碰到陡峭的地方，梯度也不会爆炸，因为梯度被限制在某个阈值 c 。

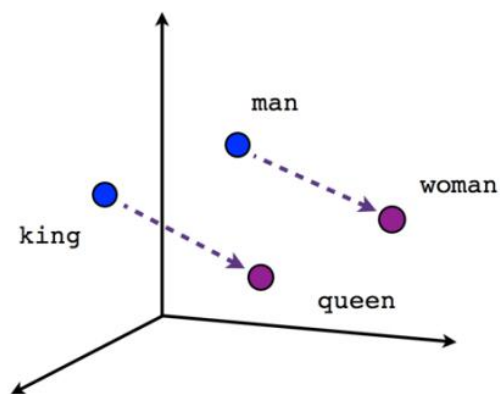


word2vec

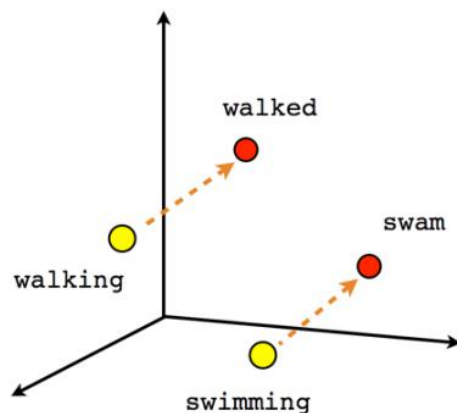
Word2Vec

嵌入：嵌入通过用更低维的向量表示数据，显著提高了网络从数据中学习规律的能力。

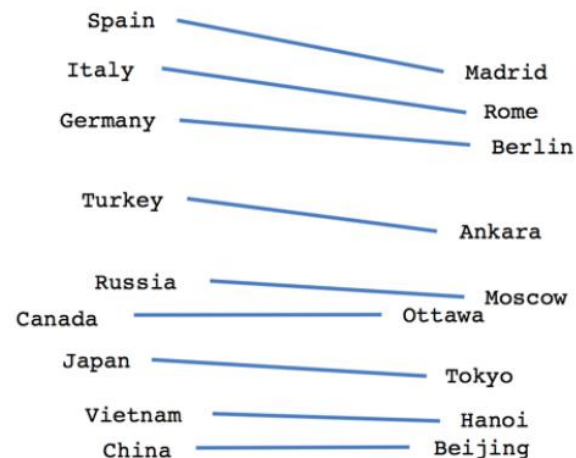
单词嵌入尤其有意思，因为网络能够学习单词之间的语义关系。例如，嵌入将知道男—女，女王—国王 等等。



Male-Female

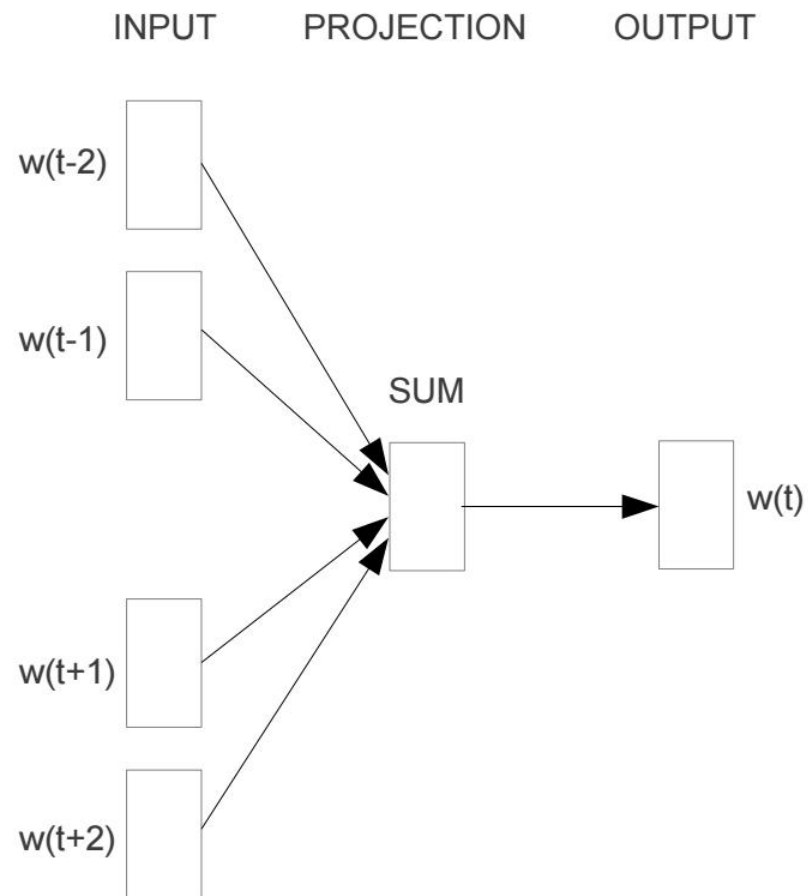


Verb tense

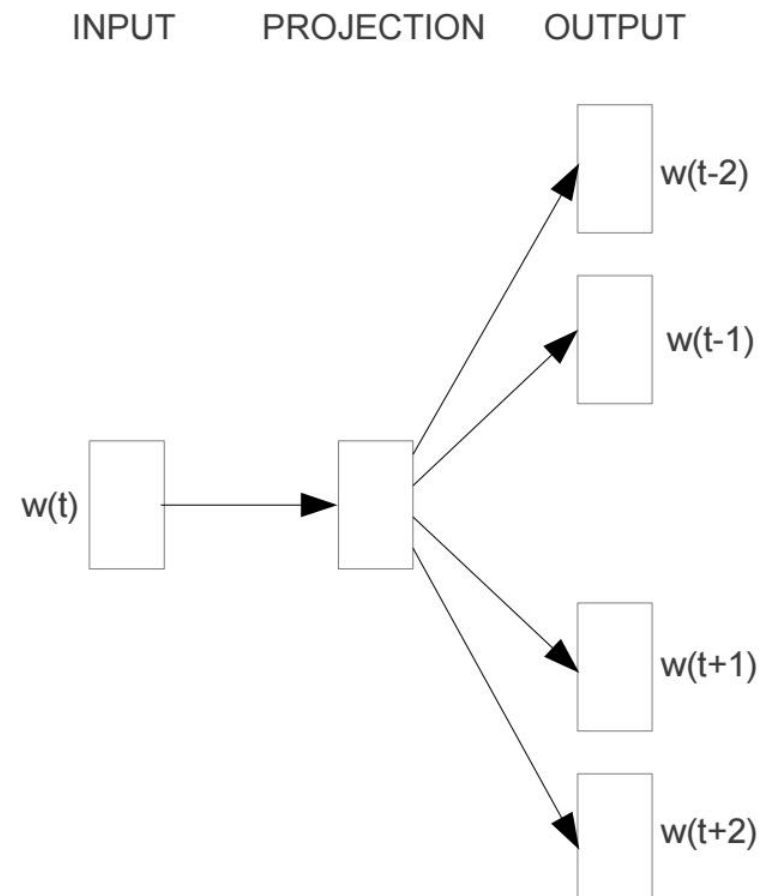


Country-Capital

Word2Vec

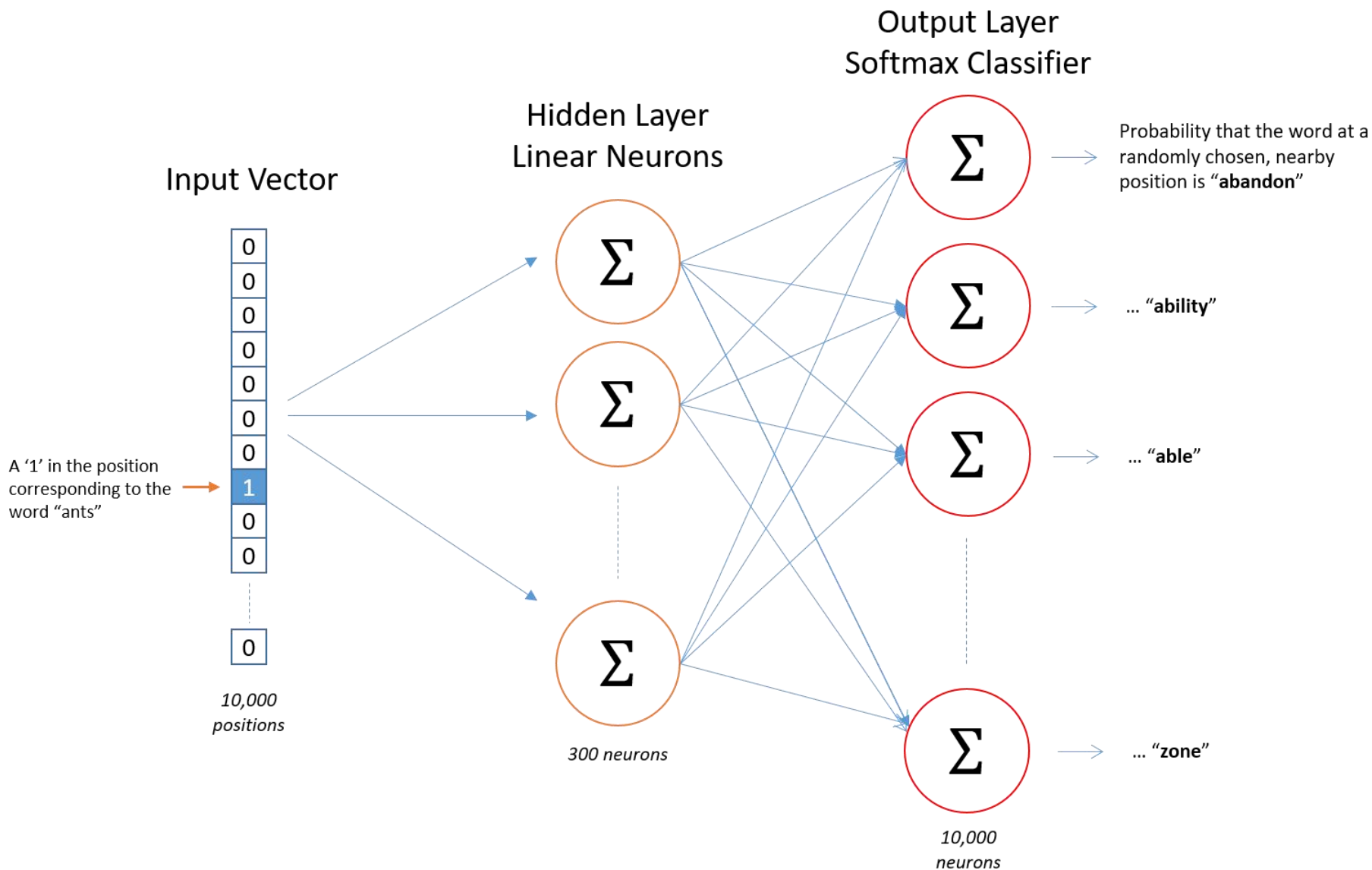


CBOW



Skip-gram

Word2Vec



Word2Vec

Source Text

Training Samples

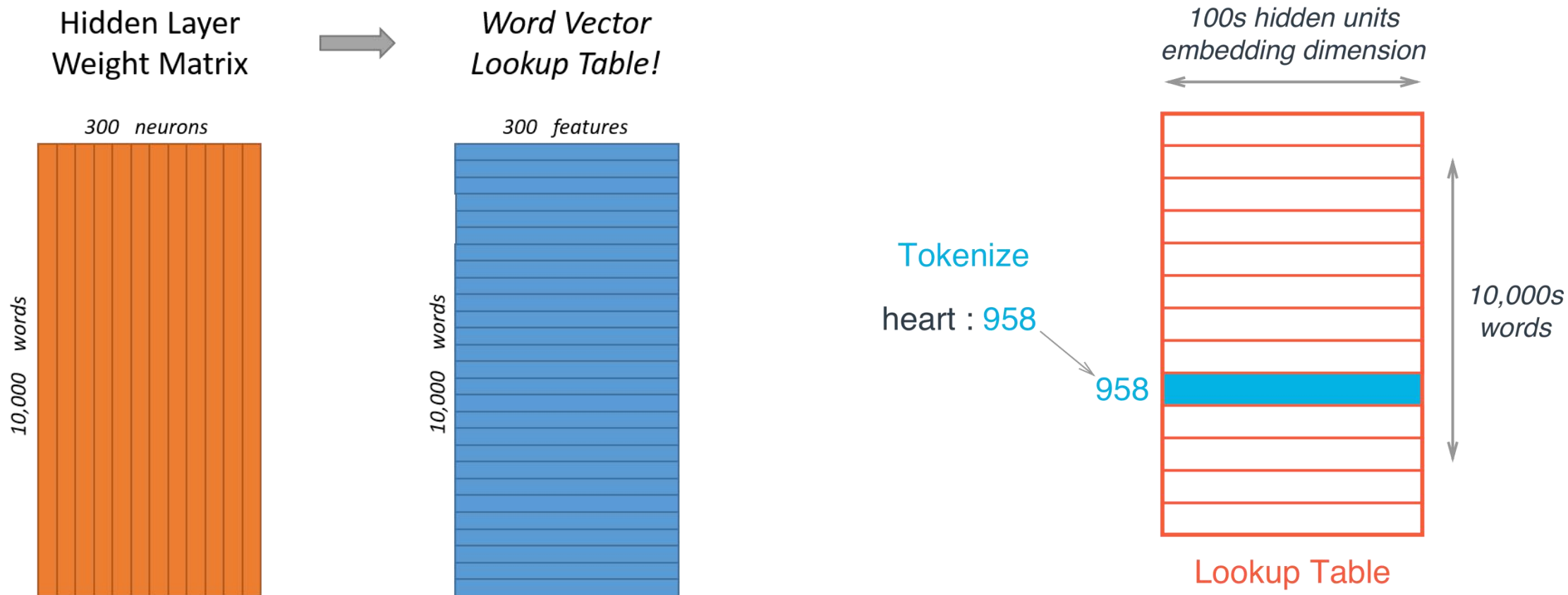
<div> <div>The</div> <div>quick</div> <div>brown</div> </div> fox jumps over the lazy dog.	→	(the, quick) (the, brown)
<div> <div>The</div> <div>quick</div> <div>brown</div> <div>fox</div> </div> jumps over the lazy dog.	→	(quick, the) (quick, brown) (quick, fox)
<div> <div>The</div> <div>quick</div> <div>brown</div> <div>fox</div> <div>jumps</div> </div> over the lazy dog.	→	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
<div> <div>quick</div> <div>brown</div> <div>fox</div> <div>jumps</div> <div>over</div> </div> the lazy dog.	→	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Word2Vec

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 8 & 2 & 1 & 9 \\ 6 & 5 & 4 & 0 \\ 7 & 1 & 6 & 2 \\ 1 & 3 & 5 & 8 \\ 0 & 4 & 9 & 1 \end{bmatrix} = [1 \ 3 \ 5 \ 8]$$

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

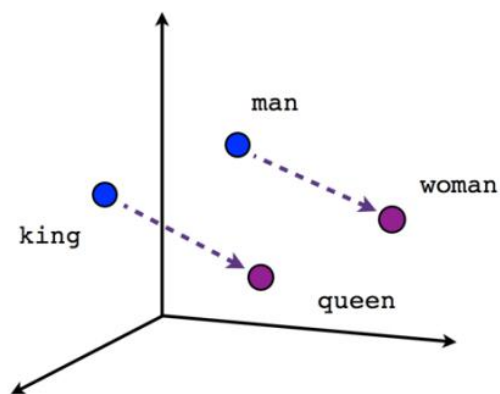
Word2Vec



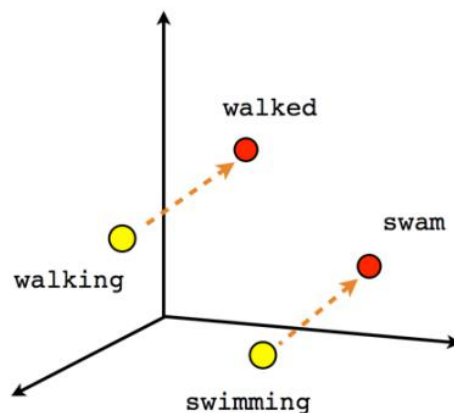
Word2Vec

嵌入：嵌入通过用更低维的向量表示数据，显著提高了网络从数据中学习规律的能力。

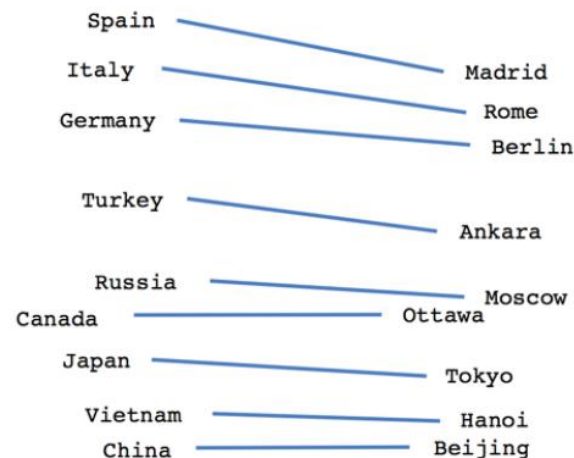
单词嵌入尤其有意思，因为网络能够学习单词之间的语义关系。例如，嵌入将知道男—女，女王—国王 等等。



Male-Female



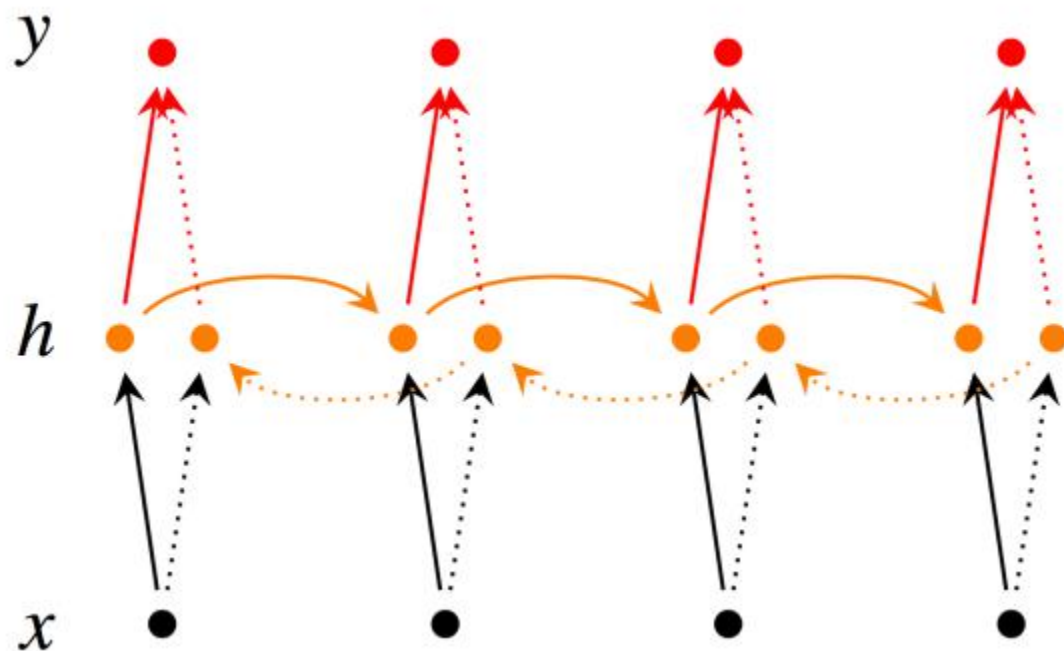
Verb tense



Country-Capital

循环神经网络RNN-Bidirectional RNN

- Bidirectional RNN(双向RNN)假设当前t的输出不仅仅和之前的序列有关，并且还与之后的序列有关，例如：预测一个语句中缺失的词语那么需要根据上下文进行预测；Bidirectional RNN是一个相对简单的RNNs，由两个RNNs上下叠加在一起组成。输出由这两个RNNs的隐藏层的状态决定。



$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

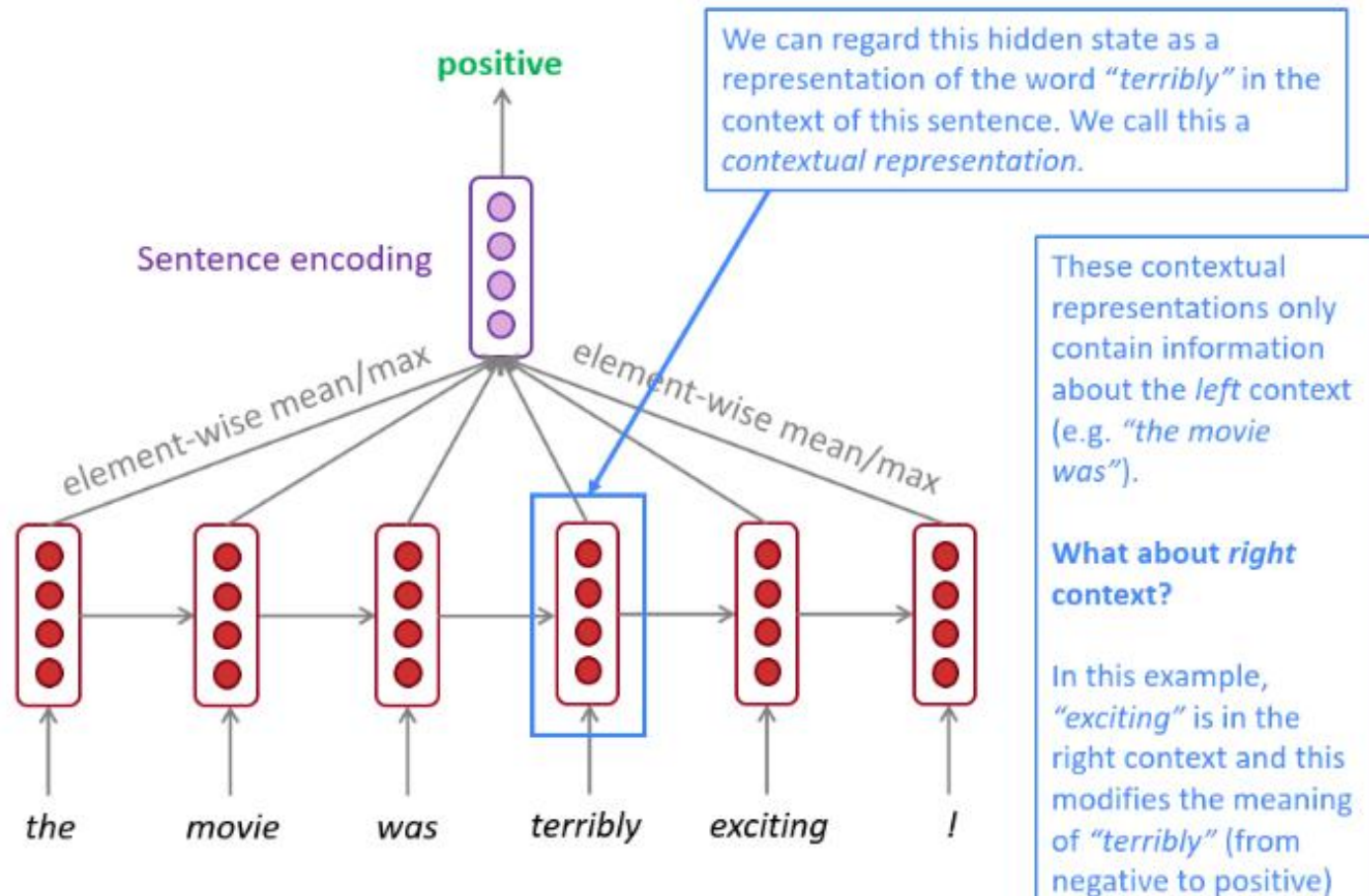
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

循环神经网络RNN-Bidirectional RNN

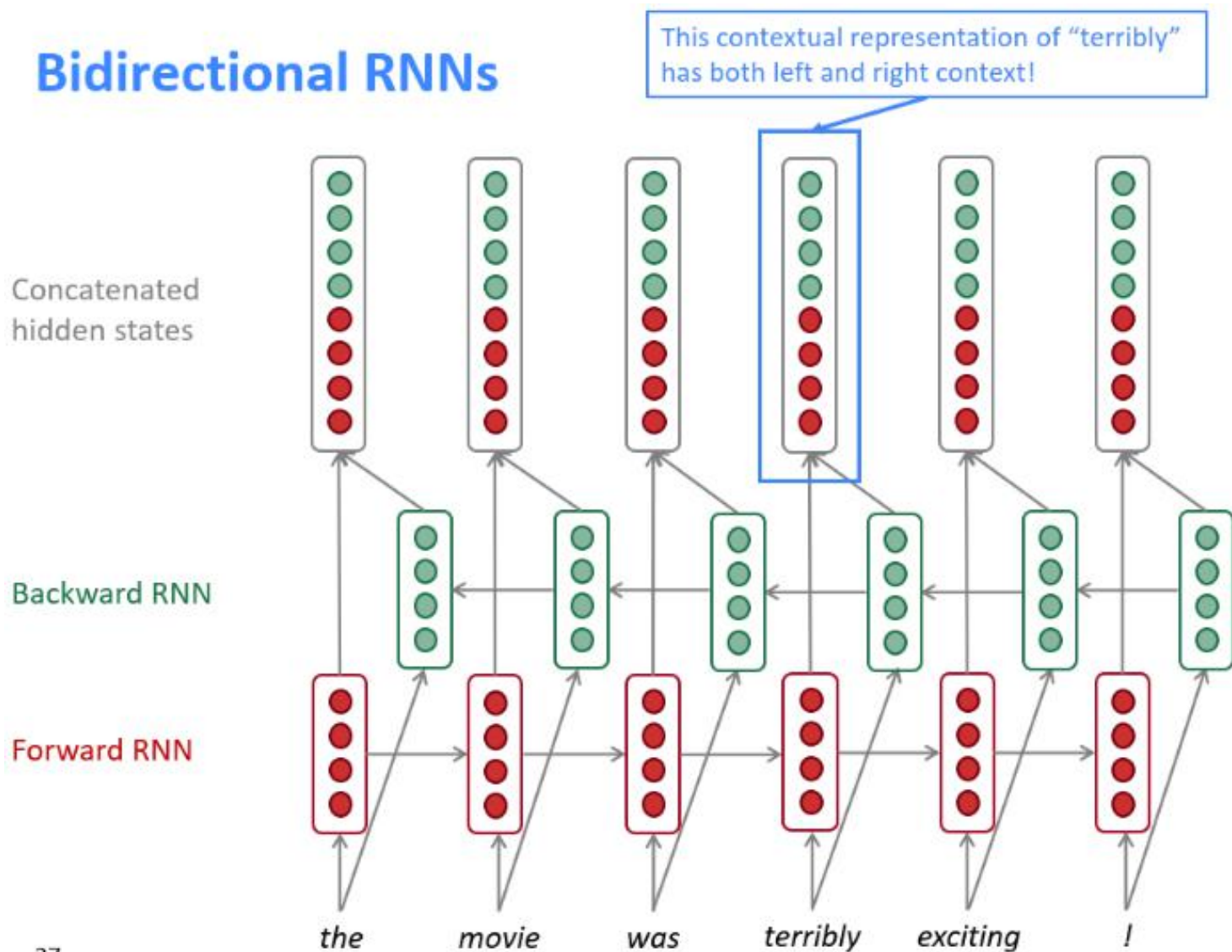
Bidirectional RNNs: motivation

Task: Sentiment Classification



循环神经网络-Bidirectional RNN

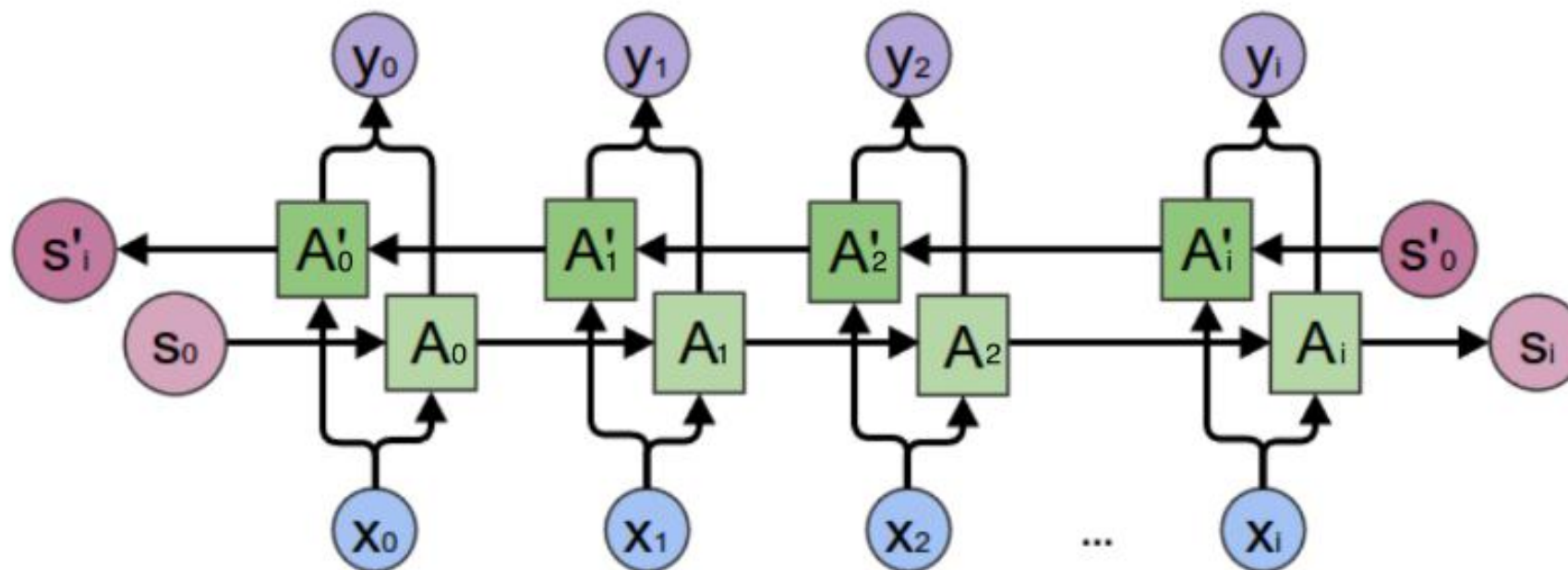
Bidirectional RNNs



双向RNN包含两个RNN，一个从左往右，一个从右往左，两个RNN的参数是独立的。最后把两个RNN的输出拼接起来作为整体输出。

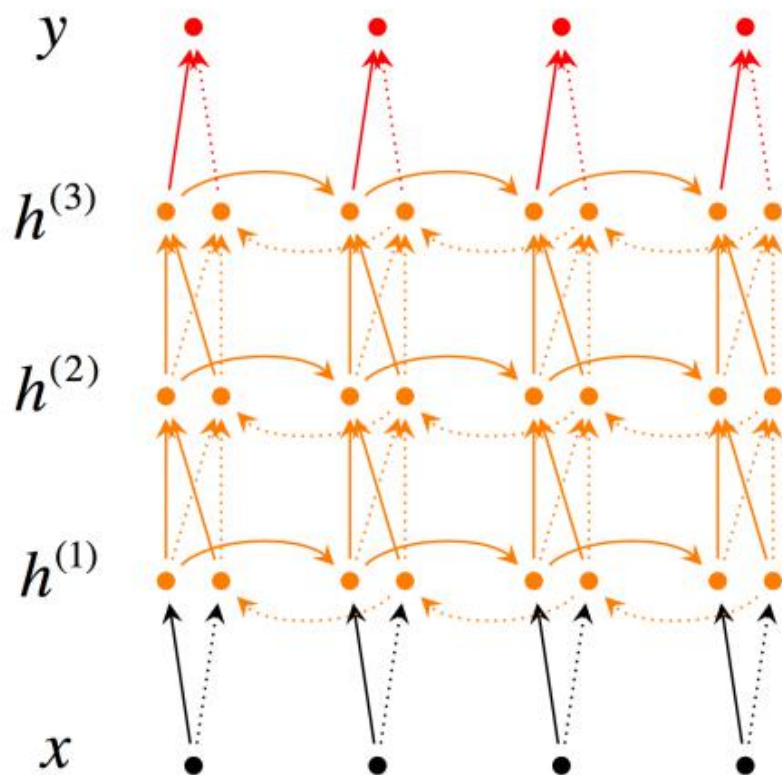
对于包含完整序列的NLP问题，双向RNN应该是默认选择，它通常比单向RNN效果更好。

循环神经网络RNN-Bidirectional RNN



循环神经网络RNN-Deep(Bidirectional) RNN

- Deep Bidirectional RNN(深度双向RNN)类似Bidirectional RNN，区别在于每个每一步的输入(问题)有多层网络，这样的话该网络便具有更加强大的表达能力和学习能力，但是复杂性也提高了，同时需要训练更多的数据。



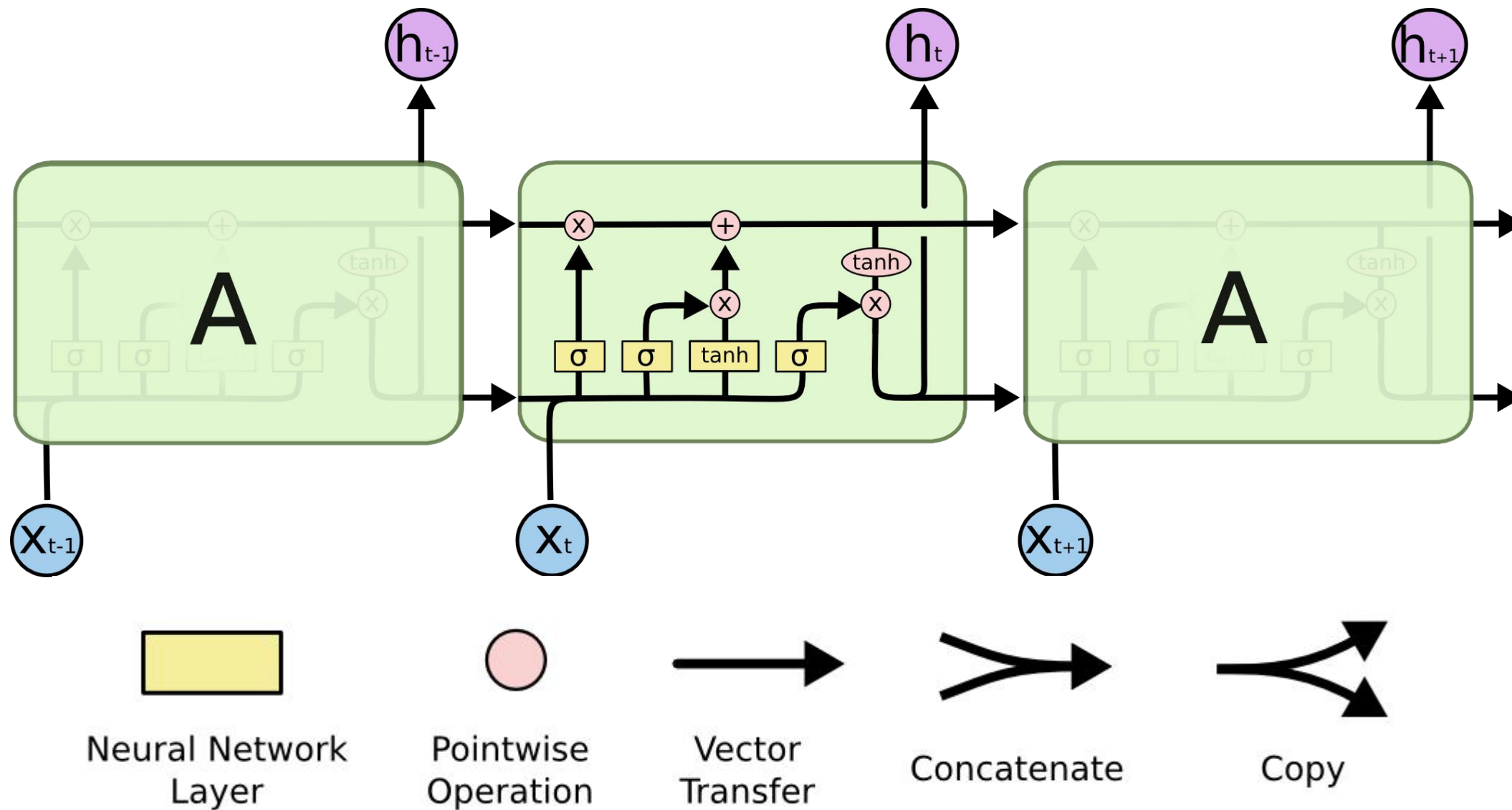
$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

LSTM-长短时记忆

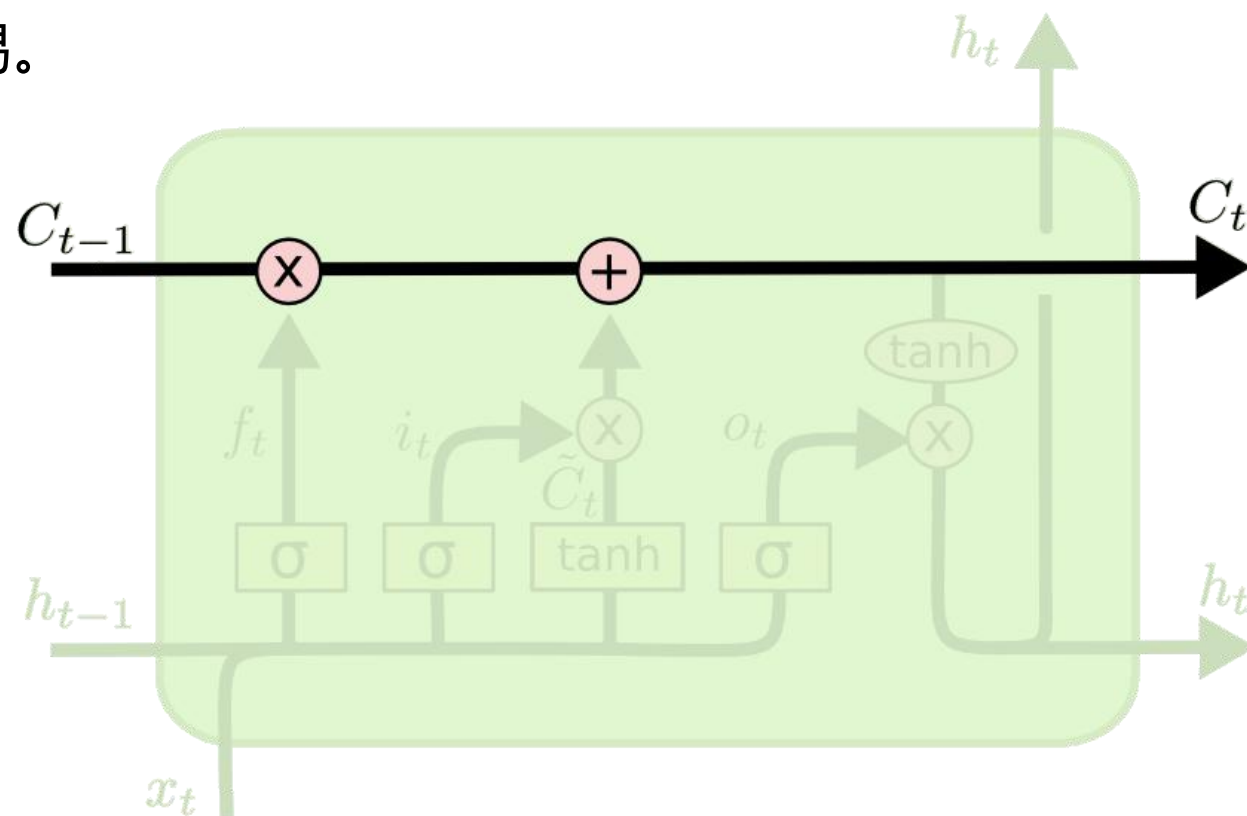
递归神经网络变形之LSTM



递归神经网络变形之LSTM

■ LSTM关键：“细胞状态”

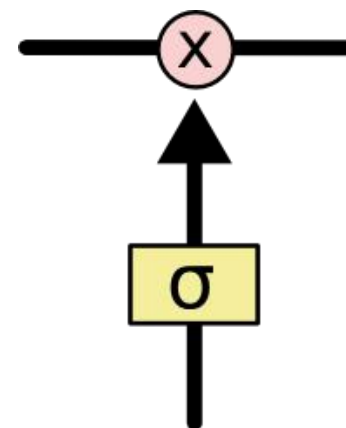
- ◆ 细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变很容易。



递归神经网络变形之LSTM

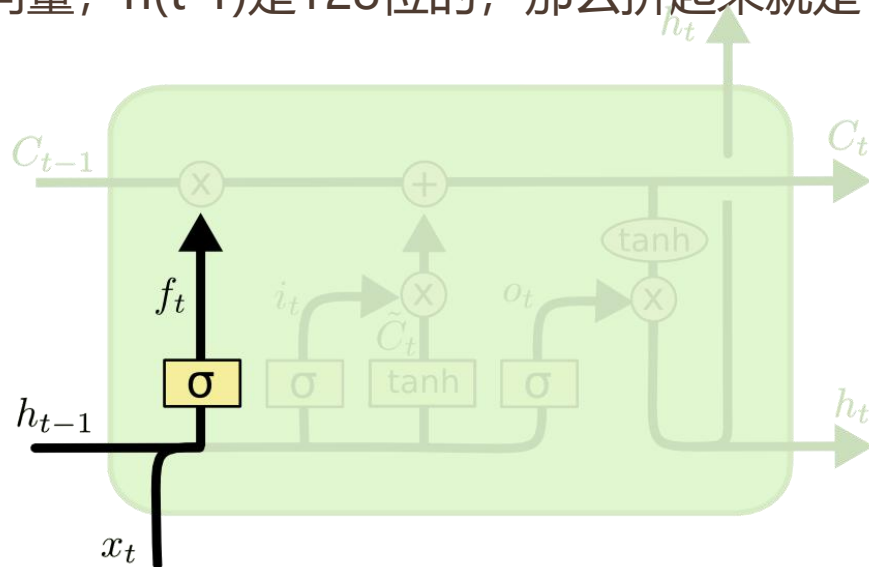
■ LSTM怎么控制 “细胞状态” ？

- ◆ LSTM可以通过gates(“门”)结构来去除或者增加 “细胞状态” 的信息
- ◆ 包含一个sigmoid神经网络层次和一个pointwist乘法操作
- ◆ Sigmoid层输出一个0到1之间的概率值，描述每个部分有多少量可以通过，0表示 “不允许任务变量通过” ， 1表示 “运行所有变量通过”
- ◆ LSTM中主要有四个 “门” 结构来控制 “细胞状态”



递归神经网络变形之LSTM

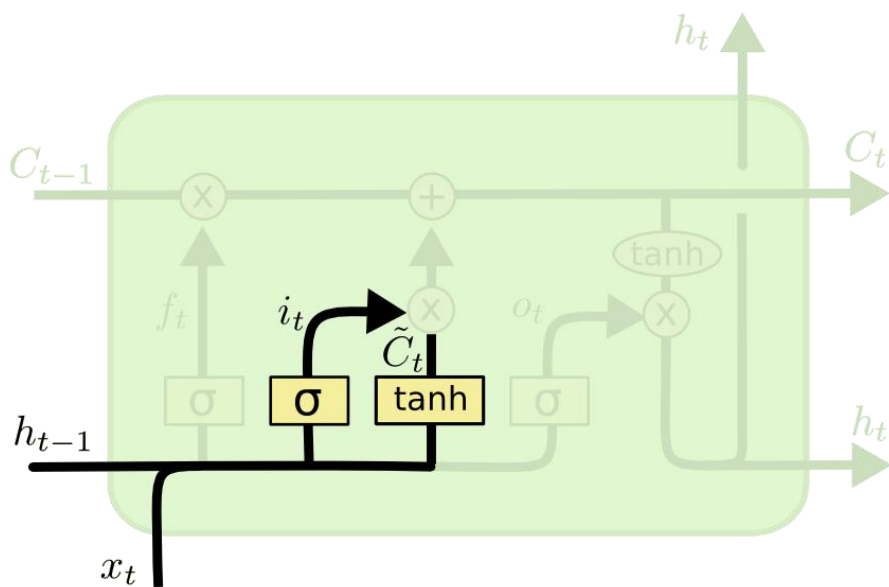
- 第一个“门” ==> “忘记门”：决定从“细胞状态”中丢弃什么信息；比如在语言模型中，细胞状态可能包含了性别信息(“他”或者“她”)，当我们看到新的代名词的时候，可以考虑忘记旧的数据
- 因为是通过sigmoid激活，接近0的值就遗忘了，接近1的值就记住并继续传递。
- 状态 $h(t-1)$ 和本次的输入(问题) $x(t)$ 结合 (concat) 起来的，concat，就是把二者直接拼起来，比如 x 是 28 位的向量， $h(t-1)$ 是 128 位的，那么拼起来就是 156 位的向量



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

递归神经网络变形之LSTM-学习门

- 第二个 “门” ==> “学习门”：决定放什么新信息到 “细胞状态” 中；
 - ◆ i_t (忽略因子, ignore factor) Sigmoid层决定什么值需要更新；
 - ◆ Tanh层创建一个新的信息向量 \tilde{C}_t ；
 - ◆ 最后： $i_t * \tilde{C}_t$ 构成学习门

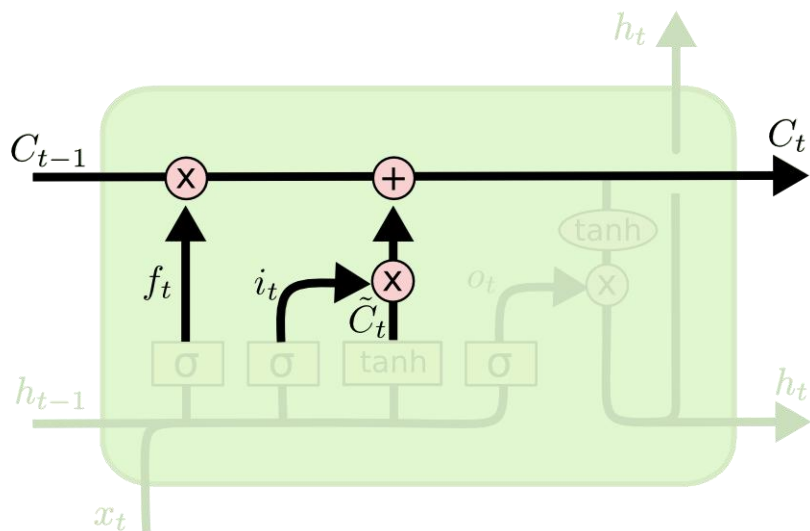


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

递归神经网络变形之LSTM-记忆门

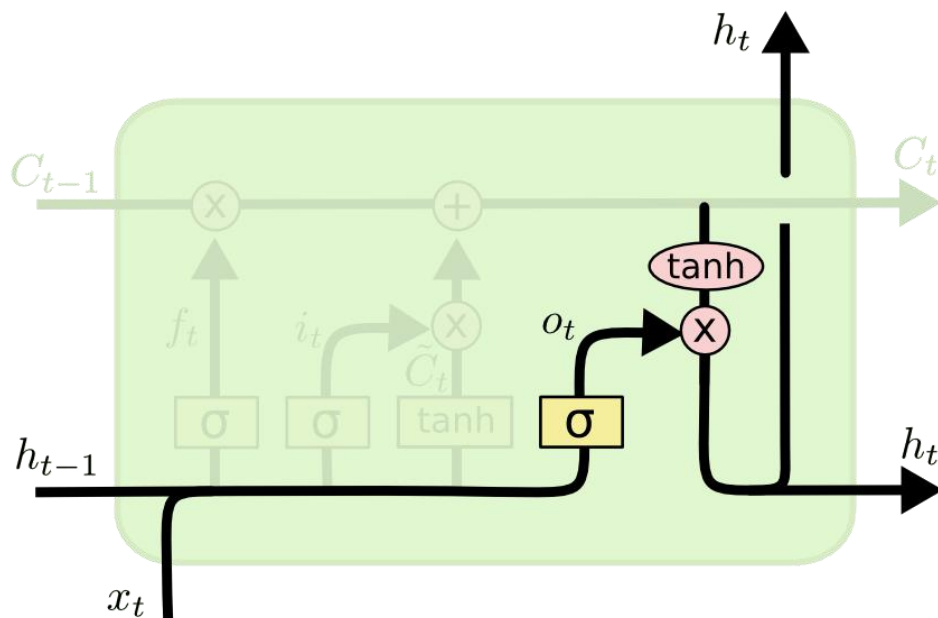
- 经过第一个和第二个“门”后，可以确定传递信息的删除和增加，即可以进行“细胞状态”的更新
 - ◆ 更新 C_{t-1} 为 C_t ;
 - ◆ 将旧状态与 f_t 相乘，丢失掉确定不要的信息;
 - ◆ 加上新的候选值 $i_t * C_t$ 得到最终更新后的“细胞状态”



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

递归神经网络变形之LSTM-使用门

- 第四个“门” ==> 基于“细胞状态”得到输出；
 - ◆ 首先运行一个sigmoid层来确定细胞状态的那个部分将输出
 - ◆ 使用tanh处理细胞状态得到一个-1到1之间的值，再将它和sigmoid门的输出相乘，输出程序确定输出的部分。



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

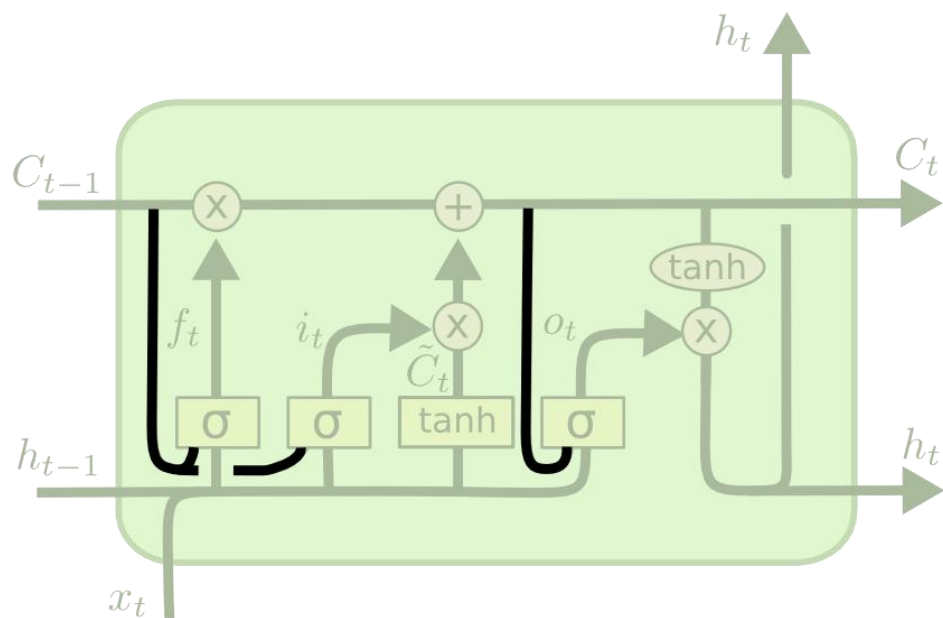
递归神经网络变形之LSTM-总结

- 另外一种理解方式，图中方框我们称为记忆单元，其中实线箭头代表当前时刻的信息传递，虚线箭头表示上一时刻的信息传递。从结构图中我们看出，LSTM模型共增加了四个门：**忘记门、学习门、记忆门、使用门**。
- **LSTM work原因**：LSTM的隐层神经元不仅包含隐状态 h_t ，还专门开辟了一个cell来保存过去的“记忆” c_t ，LSTM希望用 c_t 来传递很久以前的信息，以达到长距离依赖的目的。所以LSTM隐层神经元的输入是上一时刻的隐状态 h_{t-1} 和记忆 c_{t-1} ，输出是当前时刻的隐状态 h_t 和希望传递给下一个时刻的记忆 c_t 。

递归神经网络变形之变种

■ 变种1

- ◆ 增加 “peephole connections” 层（核心：为什么只有短时记忆可以决定门的控制，不合理，所以加入了长时记忆也参数门的开关控制！）
- ◆ 让门层也接受细胞状态的输入(问题)



$$f_t = \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

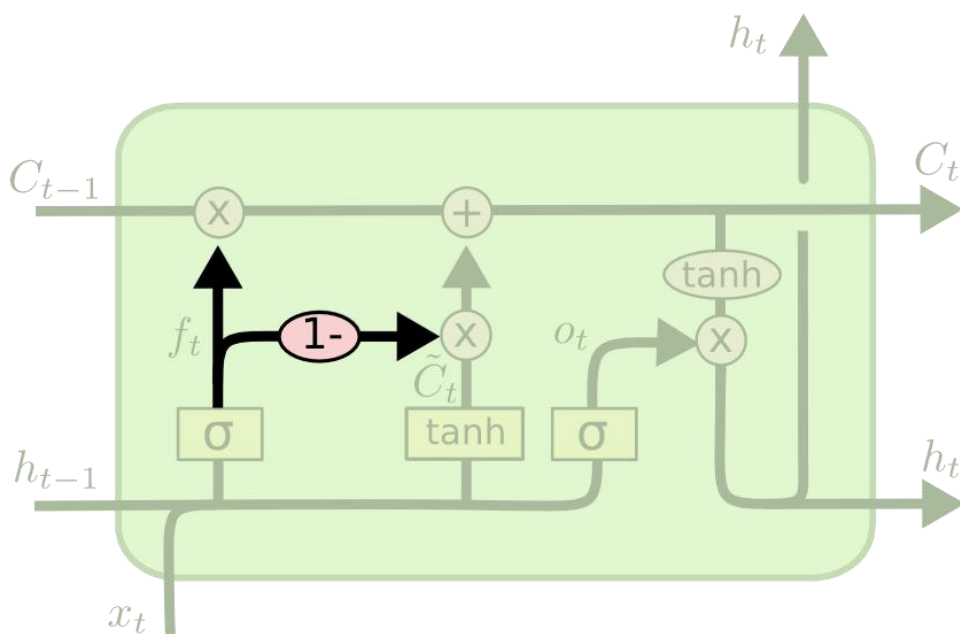
$$i_t = \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

递归神经网络变形之变种

■ 变种2

- ◆ 通过耦合忘记门和更新学习门(第一个和第二个门); 也就是不再单独的考虑忘记什么、增加什么信息, 而是一起进行考虑。

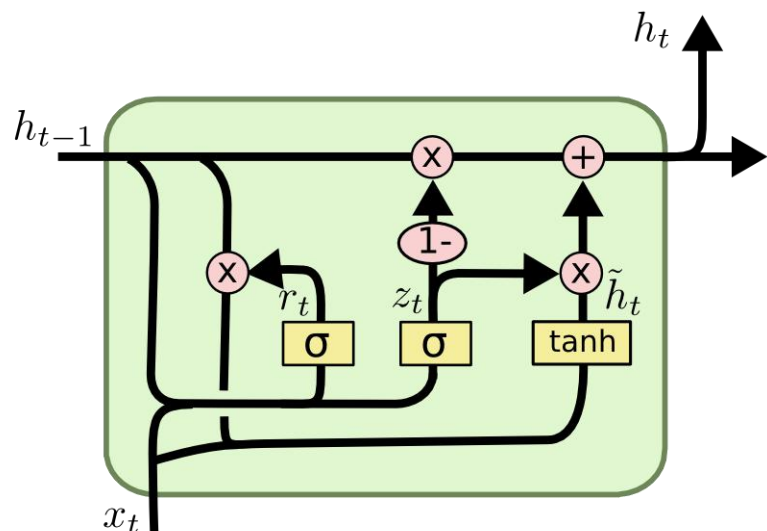


$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

递归神经网络变形之变种

■ Gated Recurrent Unit(GRU), 2014年提出

- ◆ 将忘记门和输入(问题)门合并成为一个单一的更新门
- ◆ 同时合并了数据单元状态和隐藏状态
- ◆ 结构比LSTM的结构更加简单



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gated Recurrent Units (GRU)

- Instead of $h_t = \tanh(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$ do
 - Update gate: $z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$
 - Reset gate: $r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$
 - New memory: $\tilde{h}_t = \tanh(W^{(hx)}x_t + r \circ W^{(hh)}h_{t-1})$
 - Final memory: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

RNN调参-学习率

问题1：假设你在训练一个模型。如果训练过程的输出如下所示，你将如何调整学习率来提高训练性能？

Epoch 1, Batch 1, Training Error: 8.4181

Epoch 1, Batch 2, Training Error: 8.4177

Epoch 1, Batch 3, Training Error: 8.4177

Epoch 1, Batch 4, Training Error: 8.4173

Epoch 1, Batch 5, Training Error: 8.4169

问题2：假设你在训练一个模型。如果训练过程的输出如下所示，你将如何调整学习率来提高训练性能？

Epoch 1, Batch 1, Training Error: 8.71

Epoch 1, Batch 2, Training Error: 3.25

Epoch 1, Batch 3, Training Error: 4.93

Epoch 1, Batch 4, Training Error: 3.30

Epoch 1, Batch 5, Training Error: 4.82

RNN调参- minibatch

Minibatch=2 4 8 16 32 64 128 256 512 1024

越大计算的越快，但也占用更多的计算资源，和内存。同时越大，会导致模型精度下降。

常见问题：OOM out of memory，内存溢出---降低batch size

所以：在时间允许情况下：32 64 128 256 都建议尝试下。

RNN调参- epochs

训练迭代次数是一个超参数，我们可以使用一种叫做“早期停止的技术自动优化

SessionRunHook

StopAtStepHook：用于在特定步数之后要求停止训练的 Monitor 函数

https://tensorflow.google.cn/versions/r1.4/api_docs/python/tf/train/StopAtStepHook

NanTensorHook：监控损失并在遇到 NaN 损失时停止训练的 Monitor 函数

https://tensorflow.google.cn/versions/r1.4/api_docs/python/tf/train/SessionRunHook

RNN超参数

参数	建议	备注
Epoch	Loss连续3-5个epoch没有下降，使用early stopping或者保存best model技术	
Batch size	你的内存限制，32起步，128 256 512都不错。但并非越大越好	
RNN隐藏层nodes	越大越好，256 可以	越大就越容易overfitting，花销也越大
RNN layers层数	2-3层	和CNN不同，CNN卷积层是越深越好
Sequence length	应该和生成的句子相匹配，例如整体句子平均长20，则 seq length=20	
Embedding dims	一般200—500之间	在词嵌入维度较小的时候，词汇容易被映射到相近的区域，互相之间缺乏有效区分，进而降低翻译质量



THANK YOU

上海育创网络科技有限公司