COMP9444 Final B

Yihan Xiao 2018

Section4. Recurrent Networks

- Introduction

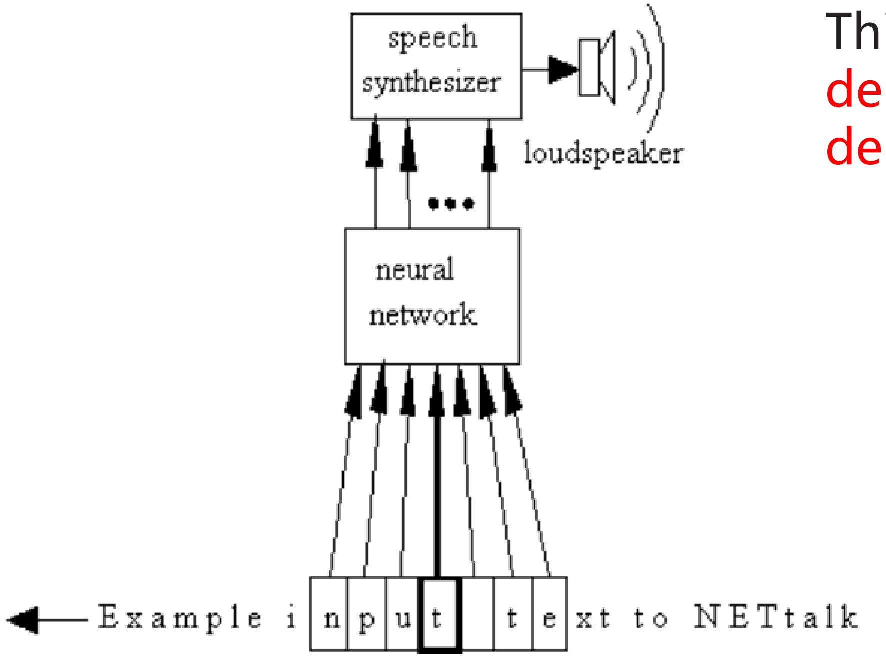
There are many tasks which require a sequence of inputs to be processed rather than a single input

- speech recognition
- time series prediction
- machine translation
- handwriting recognition

Section4. Recurrent Networks

- 早期历史 - Sliding Window

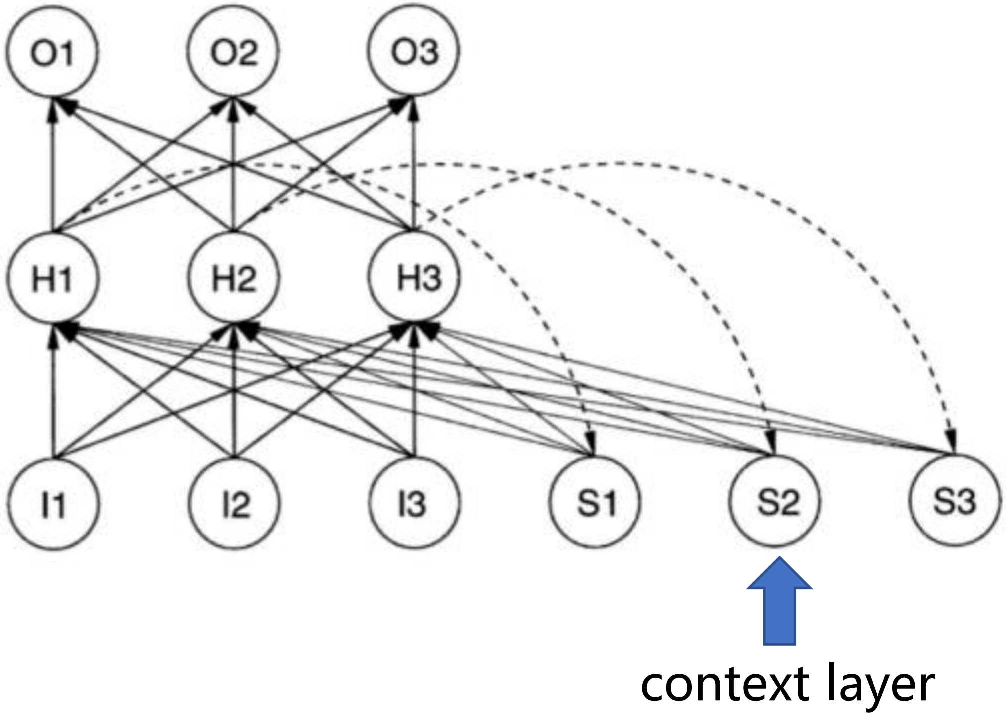
NetTalk system



This kind of approach can only learn short term dependencies, not the medium or long term dependencies that are required for some tasks.

Section4. Recurrent Networks

- 早期历史 - Simple Recurrent Network (Elman)

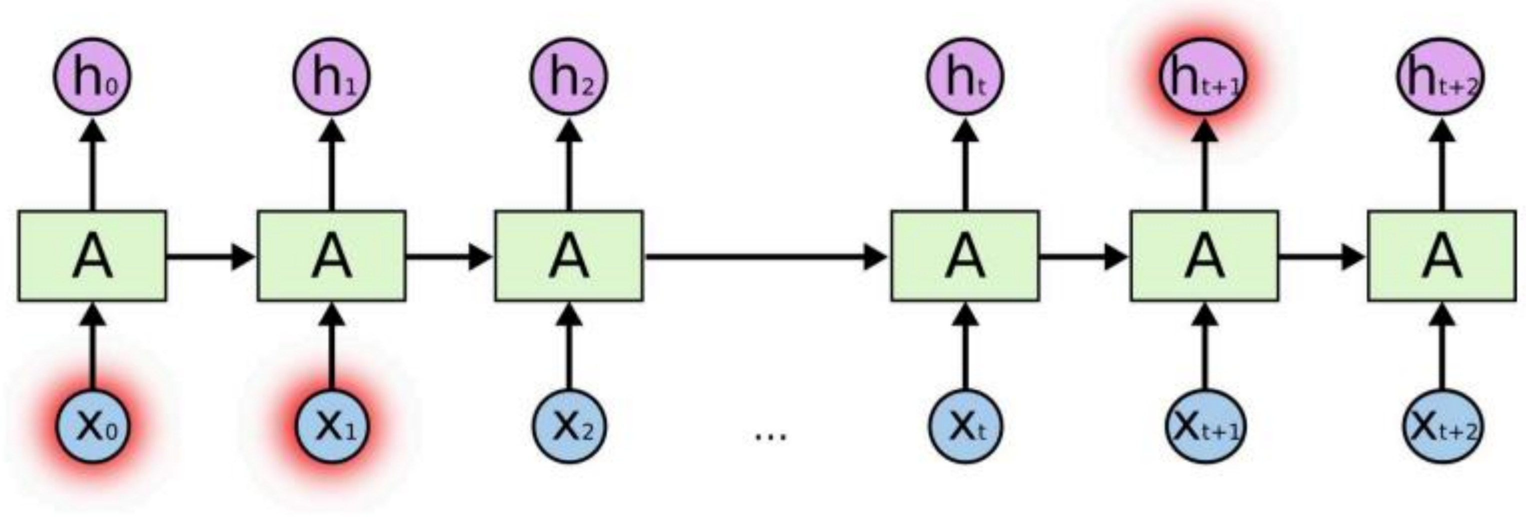


hidden layer的信息被暂时储存在context layer中，下次计算时context layer和input layer同时向hidden layer传递信息

context layer发挥了记忆顺序的作用

Section4. Recurrent Networks

- Long Range Dependencies

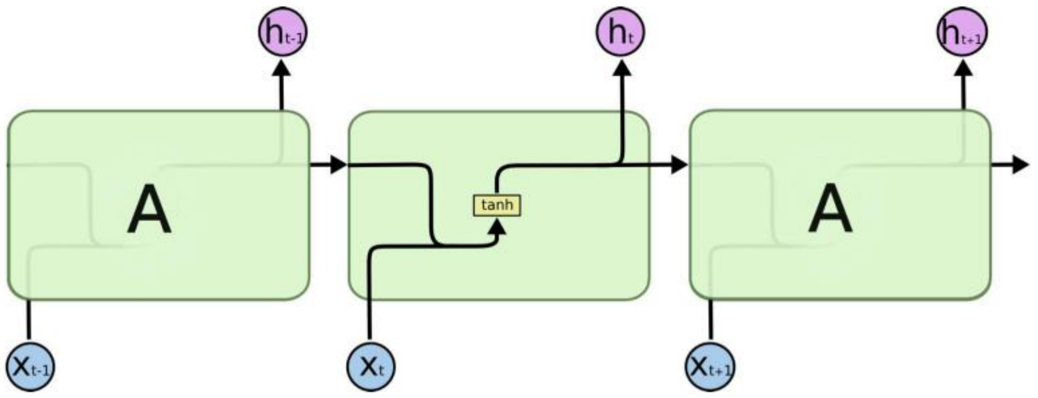


Simple Recurrent Networks (SRNs) can learn medium-range dependencies but have difficulty learning long range dependencies

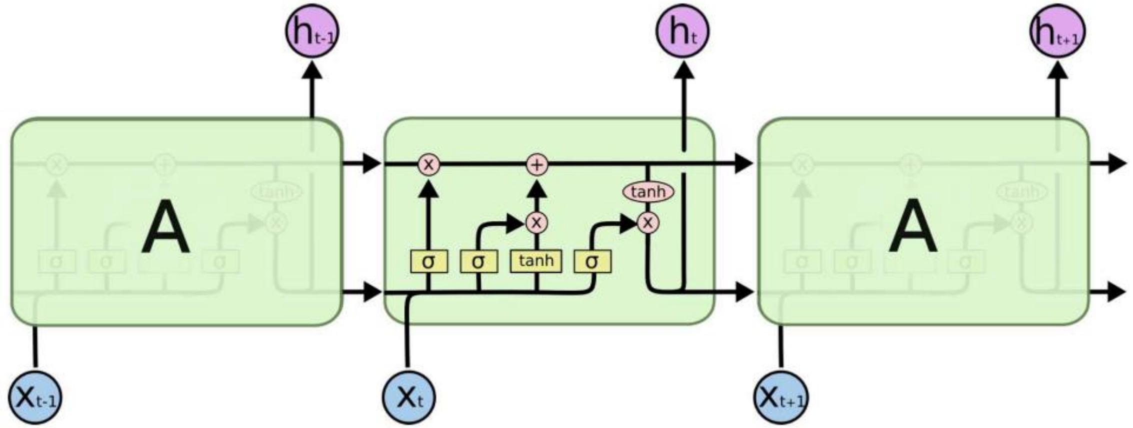
SRN无法学习到长期依赖关系

Section4. Recurrent Networks

- Long Short Term Memory



Simple Recurrent Network



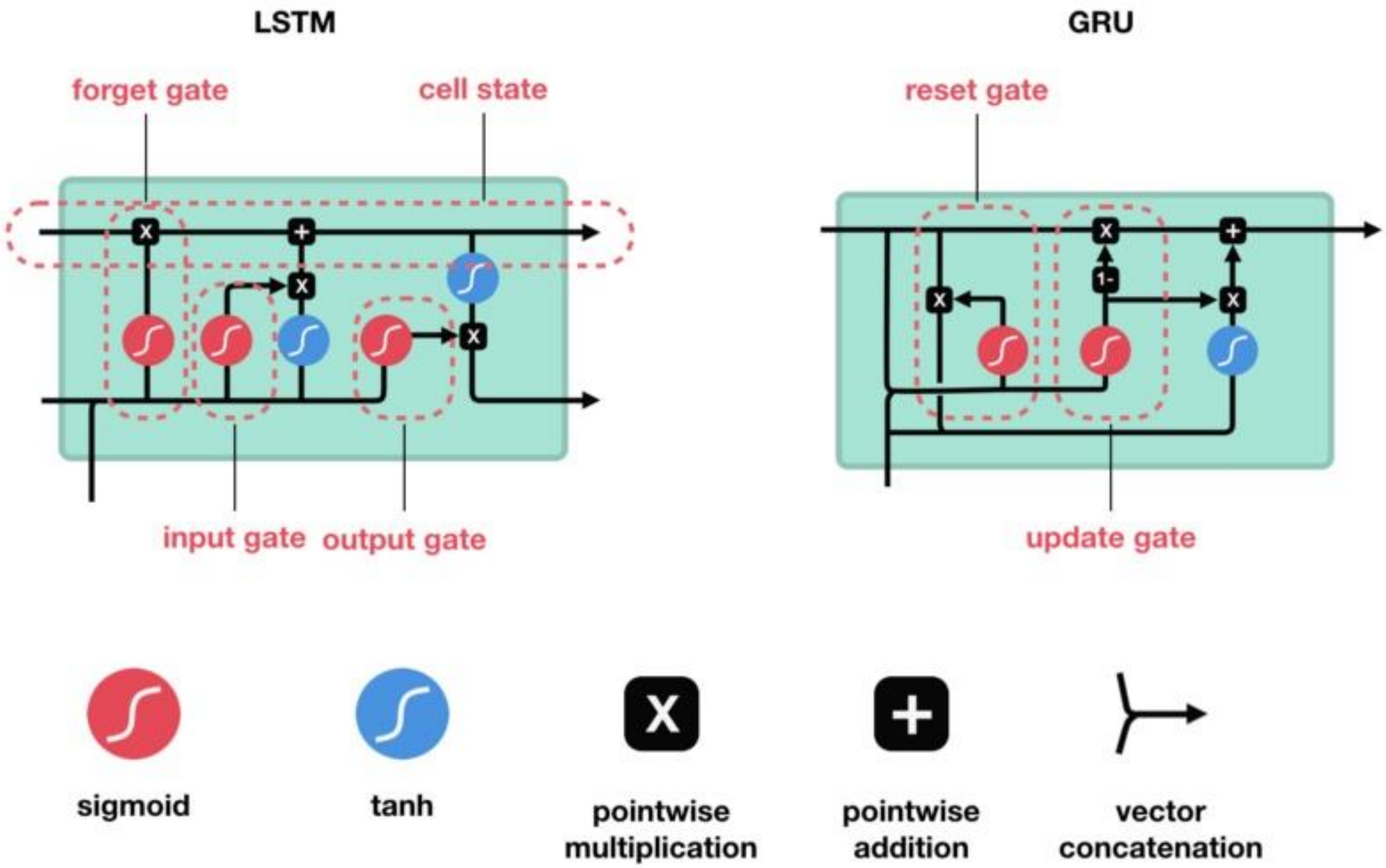
Long Short Term Memory

Section4. Recurrent Networks

- Long Short Term Memory

一个cell的信息输出有两种：
cell state和hidden state(output)

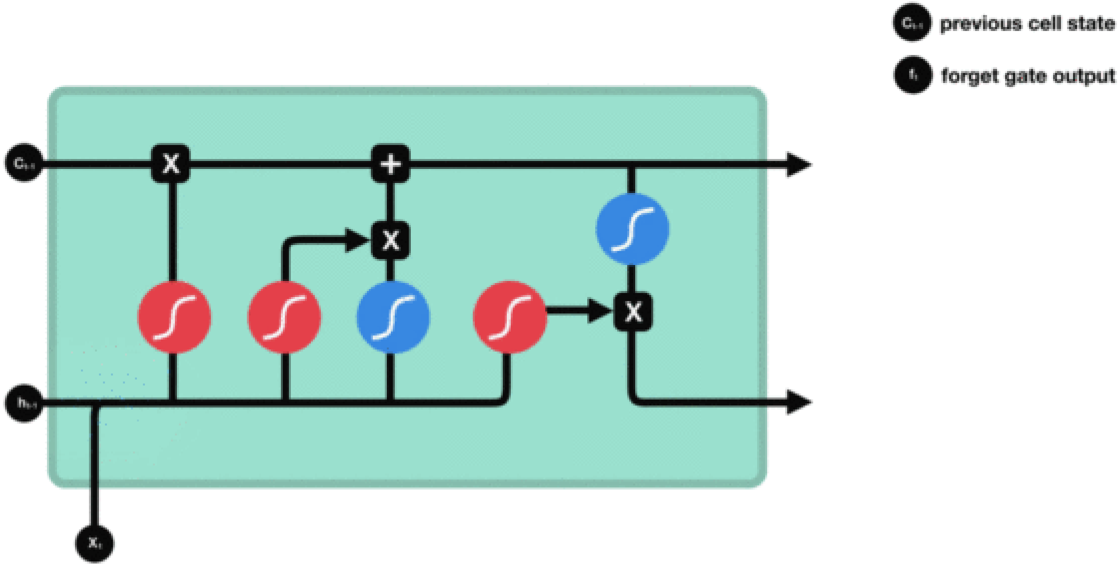
输入也有两种：
直接的数据输入和上一次的state



- Long Short Term Memory

遗忘门
这个门决定应该丢弃哪些信息。当来自先前先前隐藏状态的信息和来自当前输入的信息进入cell时，它们经sigmoid函数激活，向量的各个值介于0-1之间。越接近0意味着越容易被忘记，越接近1则越容易被保留。

$$f_t = \sigma(W_f\, x_t + U_f\, h_{t-1} + b_f) \quad \text{[forget gate]}$$

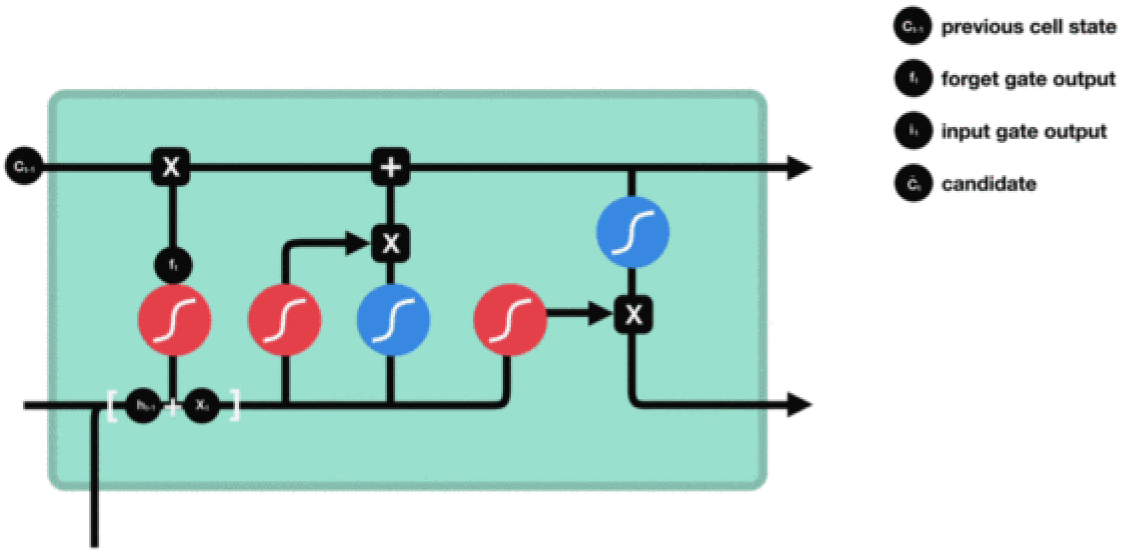


Section4. Recurrent Networks

- Long Short Term Memory

输入门
更新cell状态的重要步骤。首先，我们把先前隐藏状态和当前输入传递给sigmoid函数，由它计算出哪些值更重要（接近1），哪些值不重要（接近0）。其次，同一时间，我们也把原隐藏状态和当前输入传递给tanh函数，由它把向量的值推到-1和1之间，防止神经网络数值过大。最后，我们再把tanh的输出与sigmoid的输出相乘，由后者决定对于保持tanh的输出，原隐藏状态和当前输入中的哪些信息是重要的，哪些是不重要的。

$$i_t = \sigma(W_i\, x_t + U_i\, h_{t-1} + b_i) \quad \text{[input gate]}$$
$$g_t = \tanh(W_g\, x_t + U_g\, h_{t-1} + b_g)$$



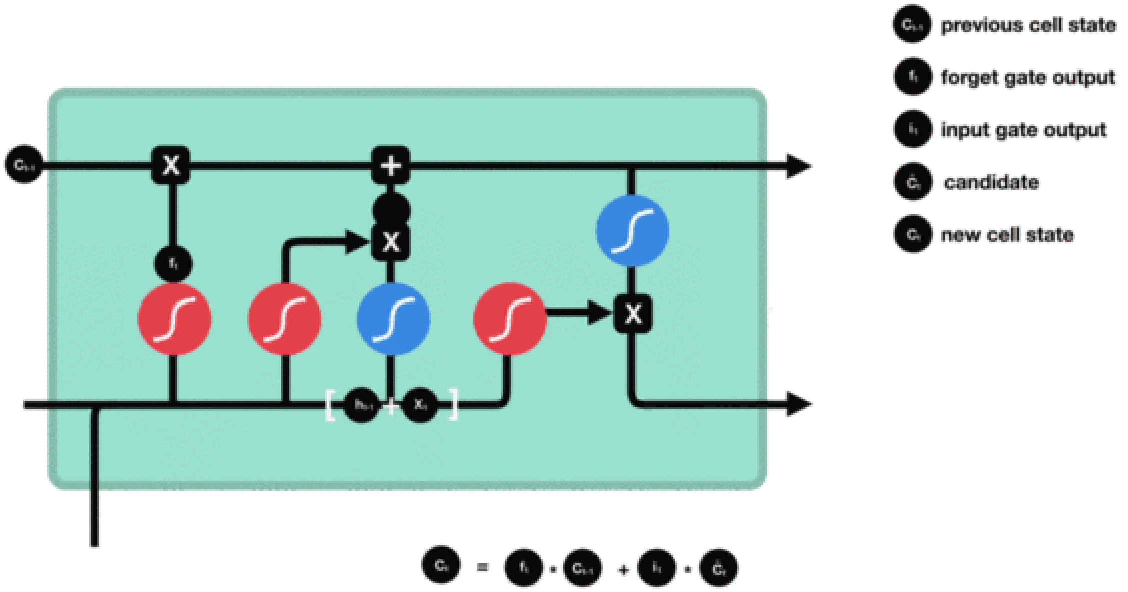
- Long Short Term Memory

cell状态

到现在为止，我们就可以更新cell状态了。首先，将先前隐藏状态和遗忘门输出的向量进行点乘，这时因为越不重要的值越接近0，原隐藏状态中越不重要的信息也会接近0，更容易被丢弃。之后，利用这个新的输出，我们再把它和输入门的输出点乘，把当前输入中的新信息放进cell状态中，最后的输出就是更新后的cell状态。

State:

$$c_t = c_{t-1} \otimes f_t + i_t \otimes g_t$$



Section4. Recurrent Networks

- Long Short Term Memory

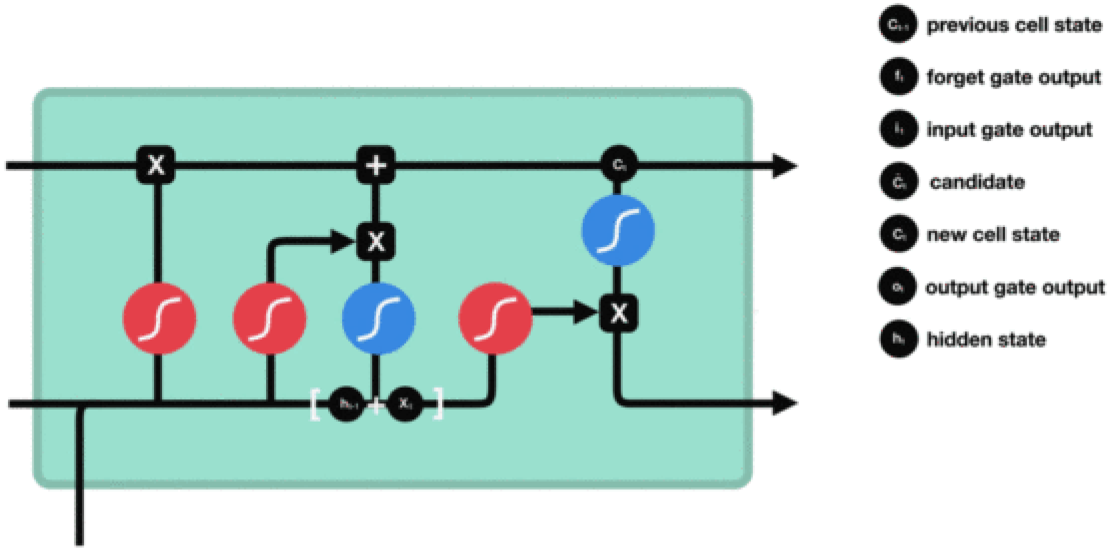
输出门

最后是输出门，它决定了下一个隐藏状态应该是什么。隐藏状态和cell状态不同，它包含有关先前输入的信息，神经网络的预测结果也正是基于它。首先，我们将先前隐藏状态和当前输入传递给sigmoid函数，其次，我们再更新后的cell状态传递给tanh函数。最后，将这两个激活函数的输出相乘，得到可以转移到下一时间步的新隐藏状态。

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

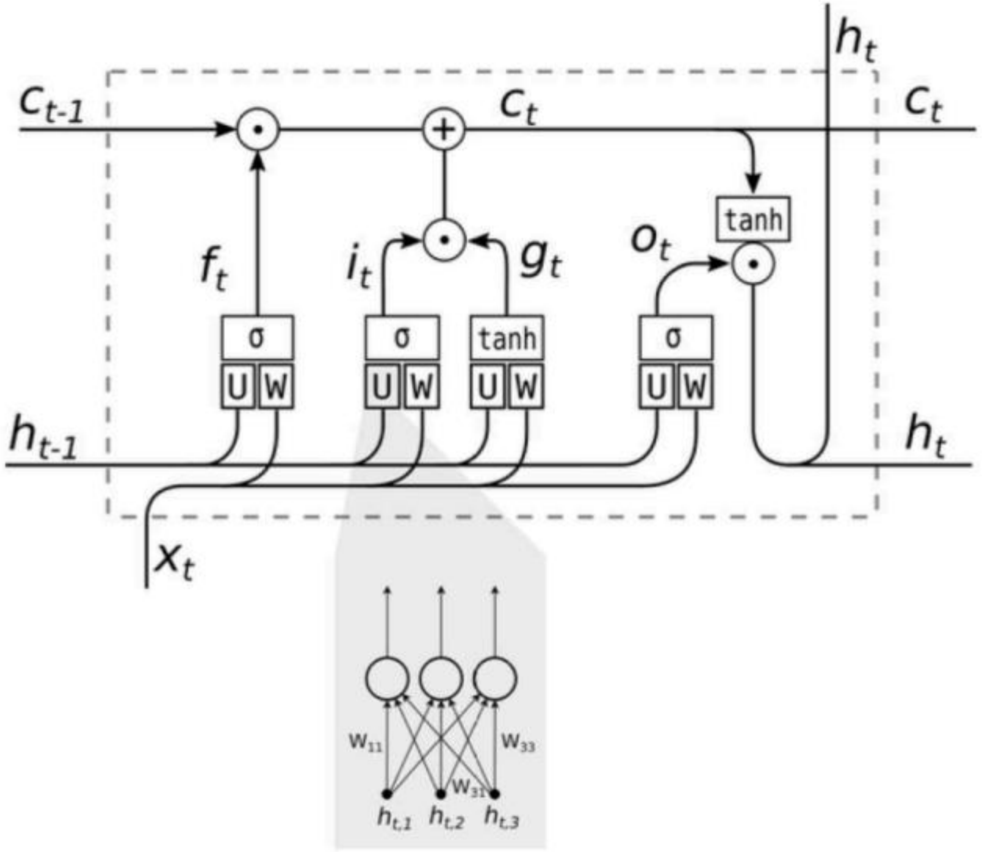
Output:

$$h_t = \tanh(c_t) \otimes o_t$$



Section4. Recurrent Networks

- Long Short Term Memory 所有公式都要熟记



Gates:

$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$ [forget gate]

$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$ [input gate]

$g_t = \tanh(W_g x_t + U_g h_{t-1} + b_g)$

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

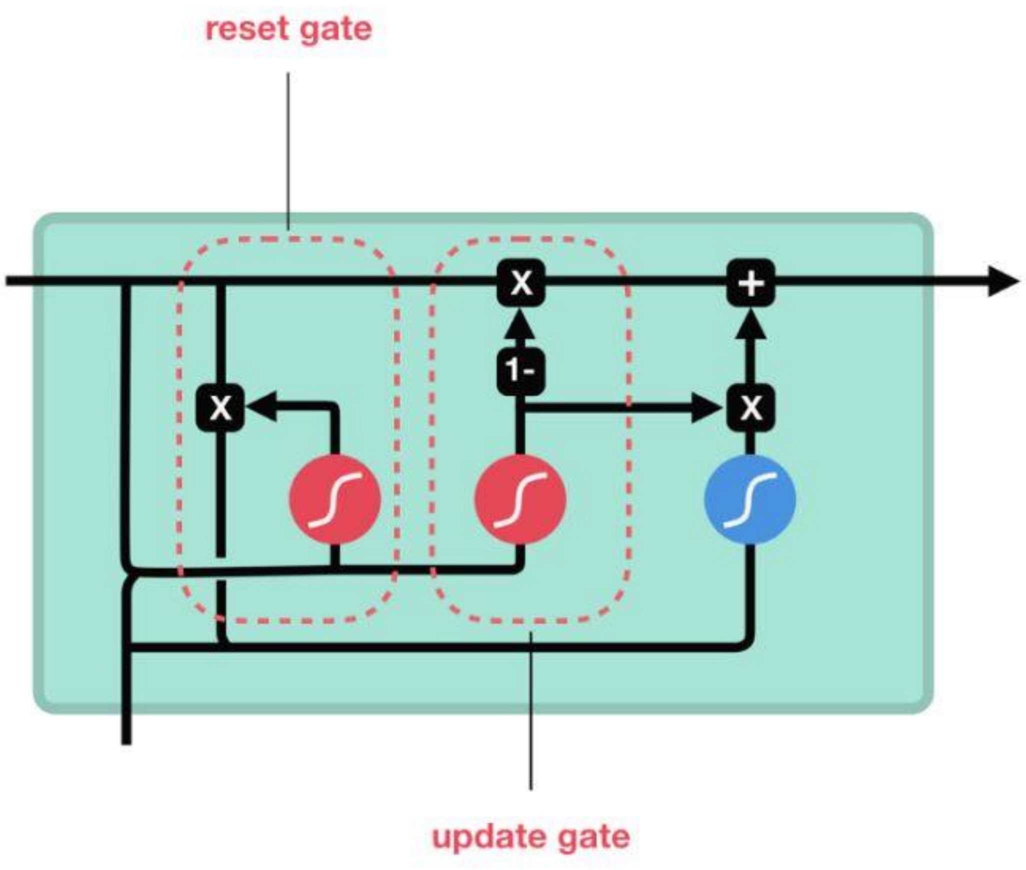
State:

$c_t = c_{t-1} \otimes f_t + i_t \otimes g_t$

Output:

$h_t = \tanh(c_t) \otimes o_t$

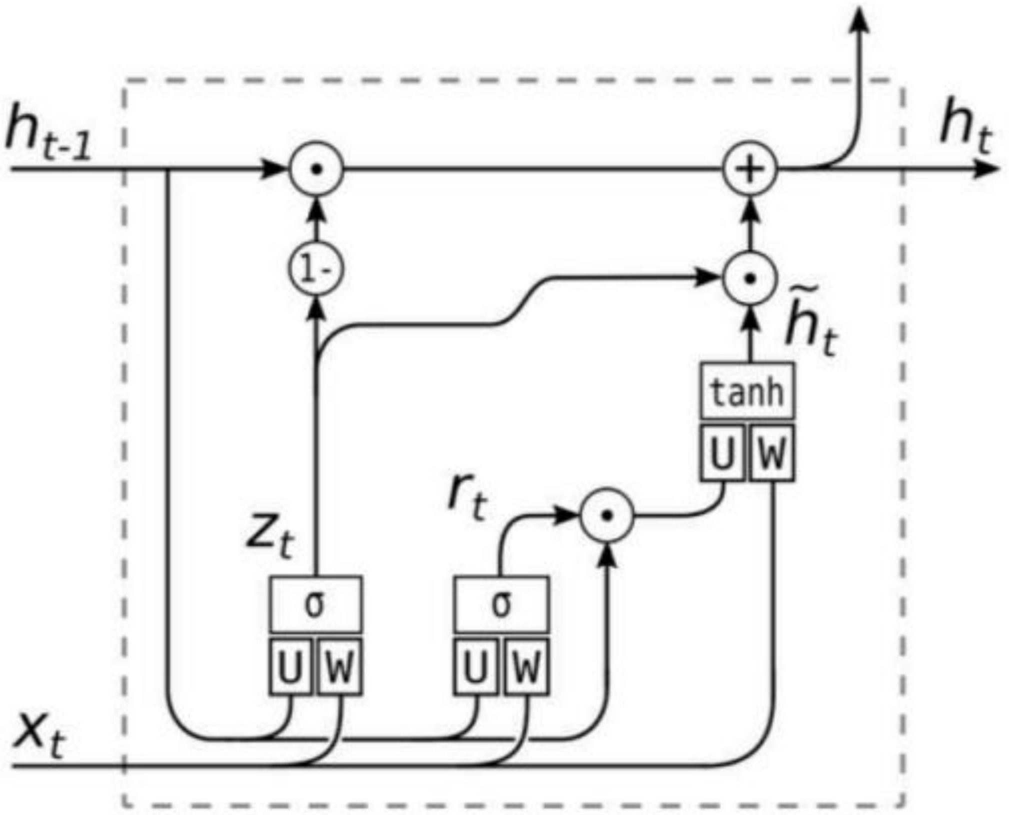
- Long Short Term Memory - GRU



更新门
更新门的作用类似LSTM的遗忘门和输入门，它决定要丢弃的信息和要新添加的信息。

重置门
重置门的作用是决定要丢弃多少先前信息。
相比LSTM，GRU的张量操作更少，所以速度也更快。
但它们之间并没有明确的孰优孰劣，只有适不适合。

- Long Short Term Memory - GRU



GRU is similar to LSTM but has only two gates instead of three

Gates:

$$z_t = \sigma(W_z\, x_t + U_z\, h_{t-1} + b_z)$$
$$r_t = \sigma(W_r\, x_t + U_r\, h_{t-1} + b_r)$$

Candidate Activation:

$$\hat{h}_t = \tanh(W\, x_t + U(r_t \otimes h_{t-1}) + b_h)$$

Output:

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \hat{h}$$

Section5. Language Processing

- Introduction

Synonyms Antonyms Taxonomy

同义词，反义词，单词分类学 –
需要大量的人工，单词关系离散，难以发现潜在的联系

Continuous representation

单词向量化/连续表达 –
可以发现单词之间的层次关系，更好地发掘潜在联系，不需要人工介入

- N/1-Gram Word Model

统计一个单词后面可能跟着哪些单词

There was a crooked man,
who walked a crooked mile
And found a crooked sixpence
upon a crooked stile.
He bought a crooked cat,
who caught a crooked mouse
And they all lived together
in a little crooked house.

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a							6					1													
all													1												
and								1												1					
bought	1																								
cat																									1
caught	1																								
crooked					1					1				1	1	1	1	1							
found	1																								
he				1																					
house																									
in	1																								
little							1																		
lived																					1				
man																									1
mile			1																						
mouse			1																						
sixpence																						1			
stile									1																
there																								1	
they		1																							
together											1														
upon	1																								
walked	1																								
was	1																								
who						1																	1		

- N/1-Gram Word Model

归一化后即可得到简单的概率预测

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a							$\frac{6}{7}$					$\frac{1}{7}$													
all													1												
and								$\frac{1}{2}$												$\frac{1}{2}$					
bought	1																								
cat																									1
caught	1																								
crooked					$\frac{1}{7}$					$\frac{1}{7}$				$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$							
found	1																								
he				1																					
house																									
in	1																								
little							1																		
lived																					1				
man																									1
mile			1																						
mouse			1																						
sixpence																						1			
stile									1																
there																								1	
they		1																							
together											1														
upon	1																								
walked	1																								
was	1																								
who						$\frac{1}{2}$																	$\frac{1}{2}$		

N/1-Gram model的假设前提为：
一个单词之和它之前的单词有关系
如果只考察前面一个单词就是1-Gram，
N个就是N-Gram

Section5. Language Processing

- Co-occurrence Matrix

2-word window

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a				1		1	6	1			1	1										1	1	1	
all													1							1					
and								1							1	1				1					
bought	1								1																
cat							1																		1
caught	1																								1
crooked	6				1					1		1		1	1	1	1	1							
found	1		1																						
he				1														1							
house							1																		
in	1																				1				
little	1						1																		
lived		1																			1				
man							1																		1
mile			1				1																		
mouse			1				1																		
sixpence							1															1			
stile							1		1																
there																								1	
they		1	1																						
together											1		1												
upon	1																1								
walked	1																								1
was	1																		1						
who					1		1							1									1		

更多的时候我们其实不想知道下个词是什么，而是想知道一个单词和它附近的单词是什么关系

使用窗口window考察单词的前后N个单词

- by aggregating over many documents, pairs (or groups) of words emerge which tend to occur near each other (but not necessarily consecutively)
  - “cat”, “caught”, “mouse”
  - “walked”, “mile”
  - “little”, “house”

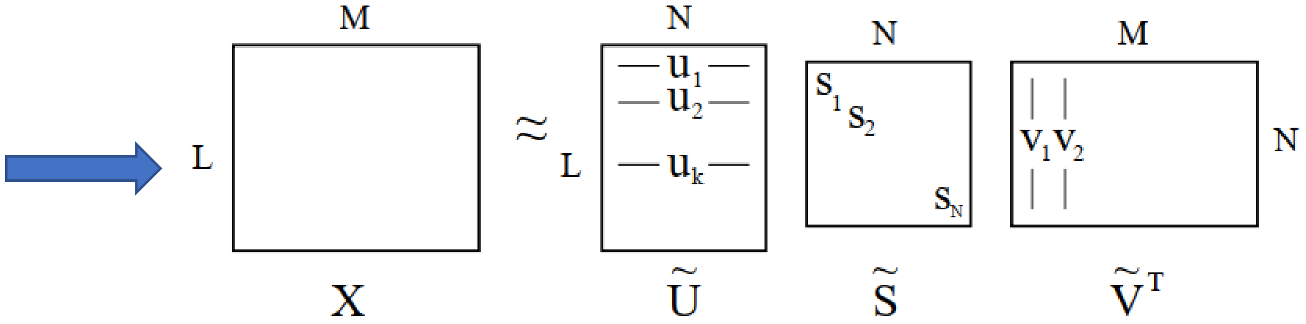
Section5. Language Processing

- Word Embedding

find the vector representation of word
close representations should have similar meanings

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a				1		1	6	1			1	1										1	1	1	
all													1							1					
and								1							1	1				1					
bought	1								1																
cat							1																		1
caught	1																								1
crooked	6				1					1		1		1	1	1	1	1							
found	1		1																						
he				1														1							
house							1																		
in	1																				1				
little	1						1																		
lived		1																			1				
man							1																		1
mile			1				1																		
mouse			1				1																		
sixpence							1															1			
stile							1		1																
there																								1	
they		1	1																						
together											1		1												
upon	1																1								
walked	1																								1
was	1																		1						
who					1	1								1									1		

Singular Value Decomposition

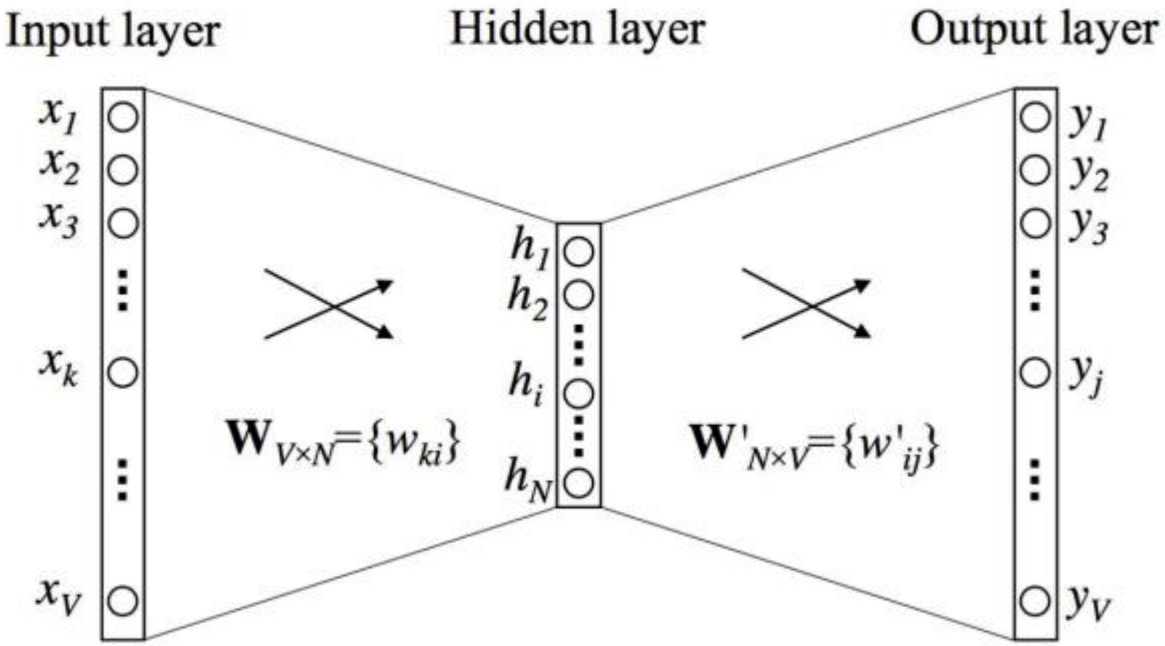


U矩阵的每一行就是一个单词的vector
SVD计算复杂，$O(LM^2)$复杂度$(L \geq M)$

Section5. Language Processing

- Word Embedding

word2vec



One-hot encoding
若总共N个单词，则第i个单词用
$[0,0,..1_{ith}..,0,0]$来表示

$$\mathrm{prob}(j|k) = \frac{\exp(u_j)}{\sum_{j'=1}^{V}\exp(u_{j'})} = \frac{\exp(\mathbf{v}_j'^{\mathrm{T}}\mathbf{v}_k)}{\sum_{j'=1}^{V}\exp(\mathbf{v}_{j'}'^{\mathrm{T}}\mathbf{v}_k)}$$

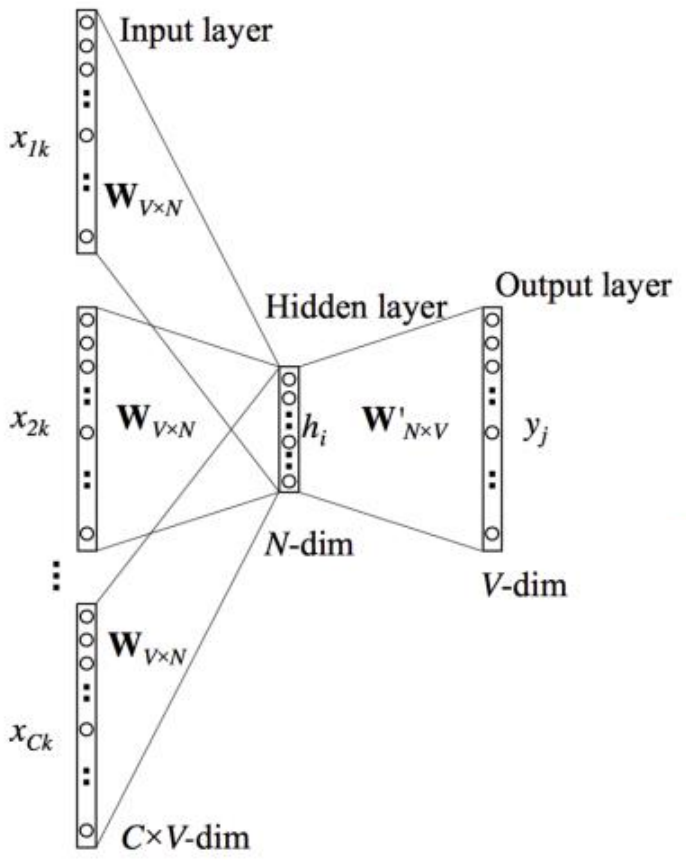
Maximize:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le r\le c, r\ne 0}\log\,\mathrm{prob}(w_{t+r}|w_t)$$

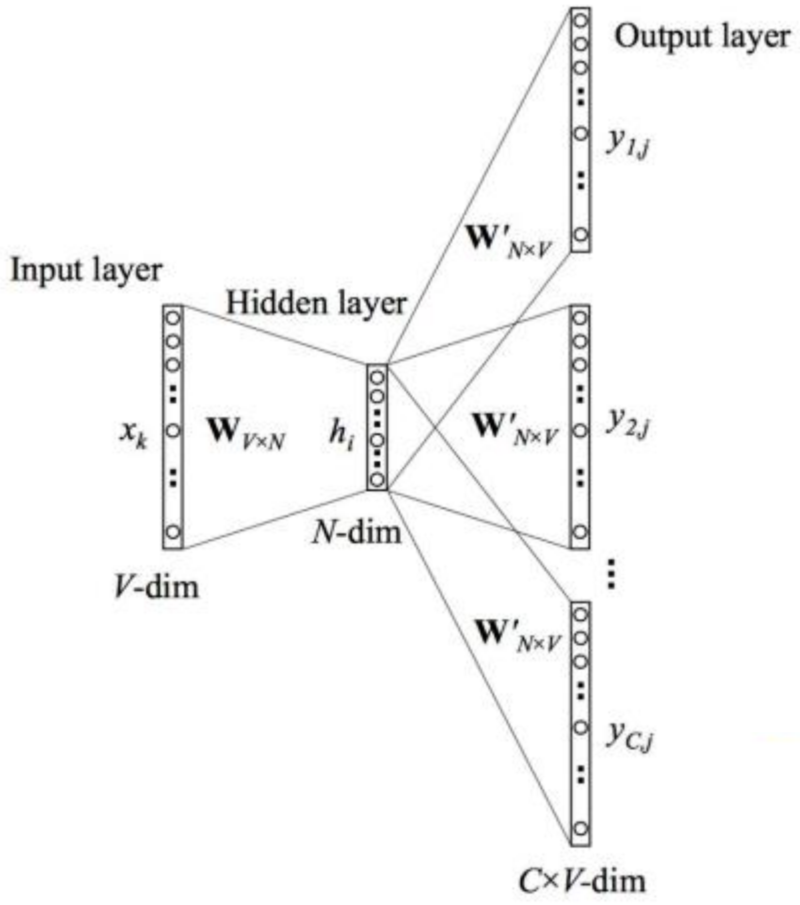
The k^{th} row $\mathbf{v}_k$ of $\mathbf{W}$ is a representation of word k.
The j^{th} column $\mathbf{v}_j'$ of $\mathbf{W}'$ is an (alternative) representation of word j.
If the (1-hot) input is k, the linear sum at each output will be $u_j = \mathbf{v}_j'^{\mathrm{T}}\mathbf{v}_k$

- Word Embedding

Continuous Bag Of Words



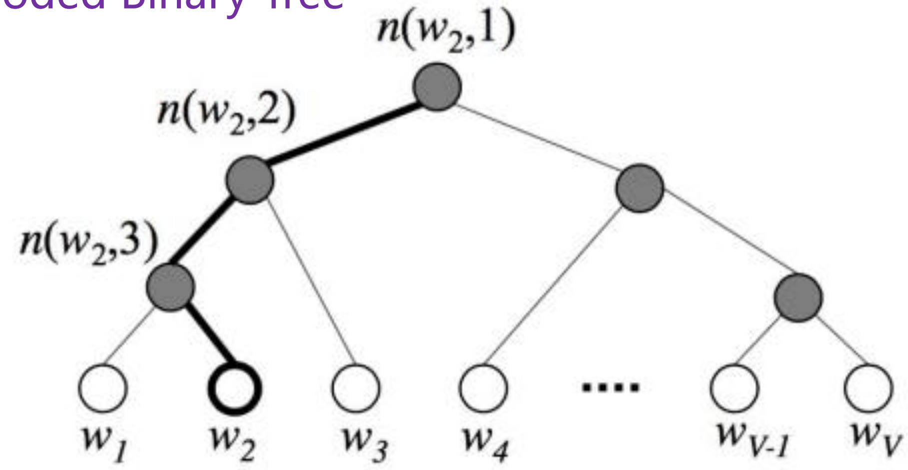
Skip-Gram



Section5. Language Processing

- 加快计算的方法 1.Hierarchical Softmax

Huffman-coded Binary Tree



$$[n' = \text{child}(n)] = \begin{cases} +1, & \text{if } n' \text{ is left child of node } n, \\ -1, & \text{otherwise.} \end{cases}$$

$$\sigma(u) = 1/(1 - \exp(-u))$$

$$\text{prob}(w = w_t) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j+1) = \text{child}(n(w,j))] \mathbf{v}'_{n(w,j)}{}^{\mathrm{T}} \mathbf{h})$$

为何要用HS?

传统的Softmax需要对所有输出节点都进行计算，如果单词数量巨大（词典型向量长度可以轻易上万），则计算过于耗时

- 加快计算的方法 2. Negative Sampling

$$E = -\log\sigma(\mathbf{v}'_{j^*}{}^{\mathrm{T}}\mathbf{h}) - \sum_{j\in\mathcal{W}_{\text{neg}}} \log\sigma(-\mathbf{v}'_j{}^{\mathrm{T}}\mathbf{h})$$

负采样的思想是每次训练只随机取一小部分的负例使他们的概率最小，以及对应的正例概率最大。

随机采样需要假定一个概率分布，word2vec中直接使用词频作为词的分布，不同的是频数上乘上0.75，相比于直接使用频次作为权重，取0.75幂的好处可以减弱不同频次差异过大带来的影响，使得小频次的单词被采样的概率变大。

$$P(w) = U(w)^{3/4}/Z$$

另一种减弱高频词的方法是以一定的概率discard单词，词频越高概率越大

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Section6. Reinforcement Learning

- Framework

  - An agent interacts with its environment.
  - There is a set $\mathcal{S}$ of *states* and a set $\mathcal{A}$ of *actions*.
  - At each time step t, the agent is in some state s_t.
    It must choose an action a_t, whereupon it goes into state
    $s_{t+1} = \delta(s_t, a_t)$ and receives reward $r_t = \mathcal{R}(s_t, a_t)$
  - Agent has a *policy* $\pi : \mathcal{S} \to \mathcal{A}$. We aim to find an optimal policy π^* which maximizes the cumulative reward.
  - In general, δ, $\mathcal{R}$ and π can be multi-valued, with a random element, in which case we write them as probability distributions

$$\delta(s_{t+1} = s \mid s_t, a_t) \quad \mathcal{R}(r_t = r \mid s_t, a_t) \quad \pi(a_t = a \mid s_t)$$

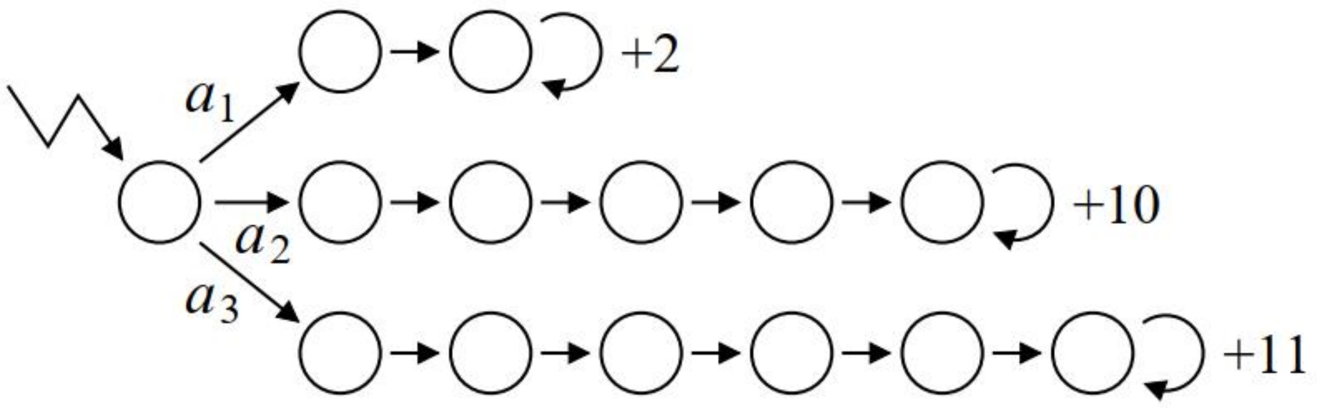
Section6. Reinforcement Learning

- Framework - reward

Finite horizon reward $\sum_{i=0}^{h-1} r_{t+i}$

Infinite discounted reward $\sum_{i=0}^{\infty} \gamma^i r_{t+i}, \quad 0 \leq \gamma < 1$

Average reward $\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i}$

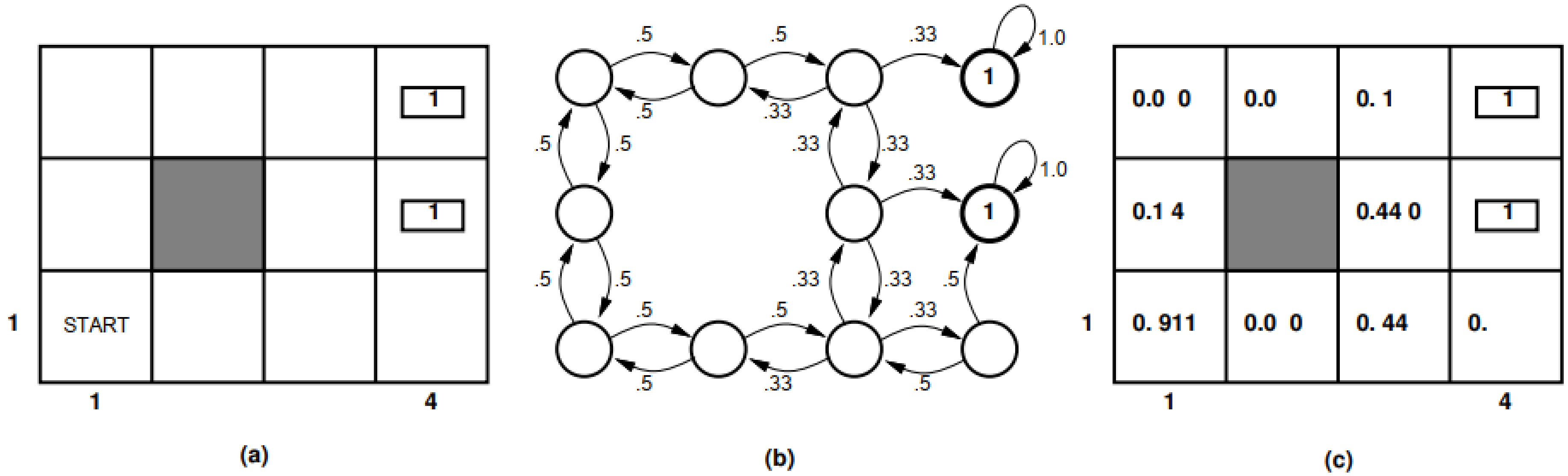


- Finite horizon, $k = 4$ $\rightarrow$ a_1 is preferred
- Infinite horizon, $\gamma = 0.9$ $\rightarrow$ a_2 is preferred
- Average reward $\rightarrow$ a_3 is preferred

Section6. Reinforcement Learning

- Value Function

Every policy π determines a Value Function $V^{\pi} : \mathcal{S} \to \mathrm{R}$ where $V^{\pi}(s)$ is the average discounted reward received by an agent who begins in state s and chooses its actions according to policy π.



Section6. Reinforcement Learning

- Temporal Difference Learning

$$V^*(s) = \mathcal{R}(s,a) + \gamma V^*(\delta(s,a))$$

$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$

Most of the time we should choose what we think is the best action.

However, in order to ensure convergence to the optimal strategy, we must occasionally choose something different from our preferred action, e.g.

我们需要强制agent以一定的概率探索非当前最佳的选择

If $\mathcal{R}$ and δ are stochastic (multi-valued), it is not safe to simply replace $V(s)$ with the expression on the right hand side. Instead, we move its value fractionally in this direction, proportional to a learning rate η

$$V(s_t) \leftarrow V(s_t) + \eta\,[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

- **Q-Learning**

For a deterministic environment, π^*, Q^* and V^* are related by

$$\pi^*(s) = \text{argmax}_a\, Q^*(s,a)$$
$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma V^*(\delta(s,a))$$
$$V^*(s) = \max_b Q^*(s,b)$$

So

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \max_b Q^*(\delta(s,a),b)$$

This allows us to iteratively approximate Q by

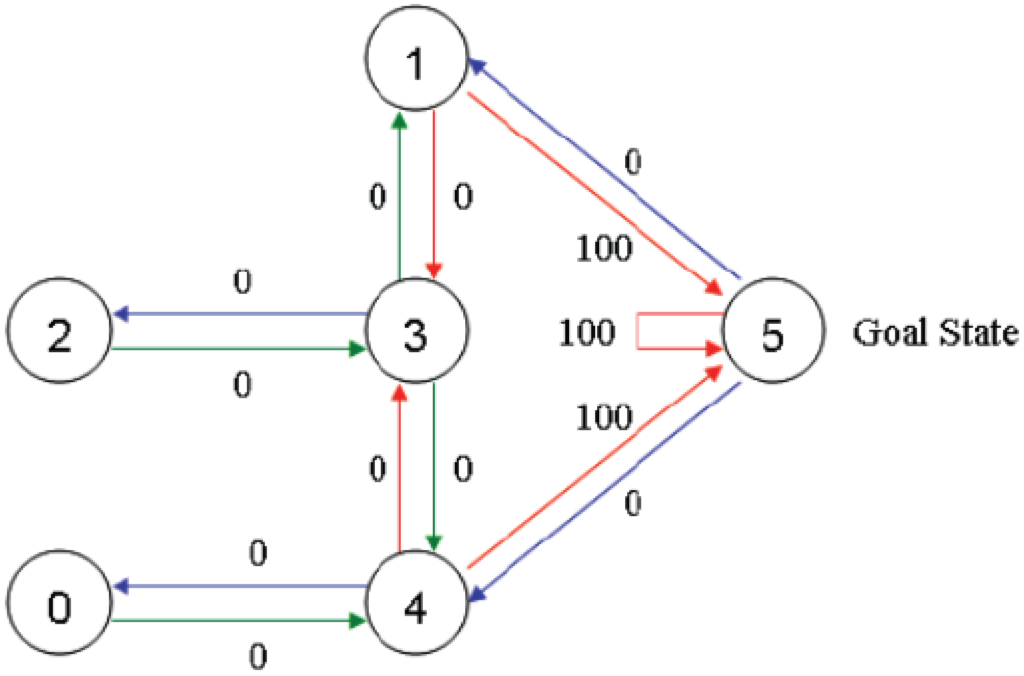
$$Q(s_t,a_t) \leftarrow r_t + \gamma \max_b Q(s_{t+1},b)$$

If the environment is stochastic, we instead write

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta\,[r_t + \gamma \max_b Q(s_{t+1},b) - Q(s_t,a_t)]$$

- **Q-Learning** Example

$\gamma = 0.8$



$$R = \begin{array}{c c} & \text{Action} \\ \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

- **Q-Learning** Example

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \max_b Q^*(\delta(s,a),b)$$

首先初始化Q矩阵

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

随机选择一个状态开始，根据R表格计算下一步

Action

$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

假设我们选择state 1开始

- **Q-Learning** Example

$$
\begin{aligned}
Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\
&= 100 + 0.8 * \max\{0,0,0\} \\
&= 100.
\end{aligned}
$$

5是最终目的地，一次迭代结束

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

$$\Downarrow$$

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Section6. Reinforcement Learning

- **Q-Learning** Example

再重新随机选择一个状态开始，
假设选state 3

$$
\begin{array}{cc}
 & \text{Action} \\
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}
\end{array}
$$

$$
R = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
$$

$$
Q = \begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 100 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

$$
\begin{aligned}
Q(3,1) &= R(3,1) + 0.8 * \max\{Q(1,3), Q(1,5)\} \\
&= 0 + 0.8 * \max\{0, 100\} \\
&= 80.
\end{aligned}
$$

$$
Q = \begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 100 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 80 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

- **Q-Learning** Example

state 1 不是目的地，继续计算

假设我们再次选择了到state 5

$$
R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}
$$

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
$$

$$
\begin{aligned}
Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\
&= 100 + 0.8 * \max\{0, 0, 0\} \\
&= 100.
\end{aligned}
$$

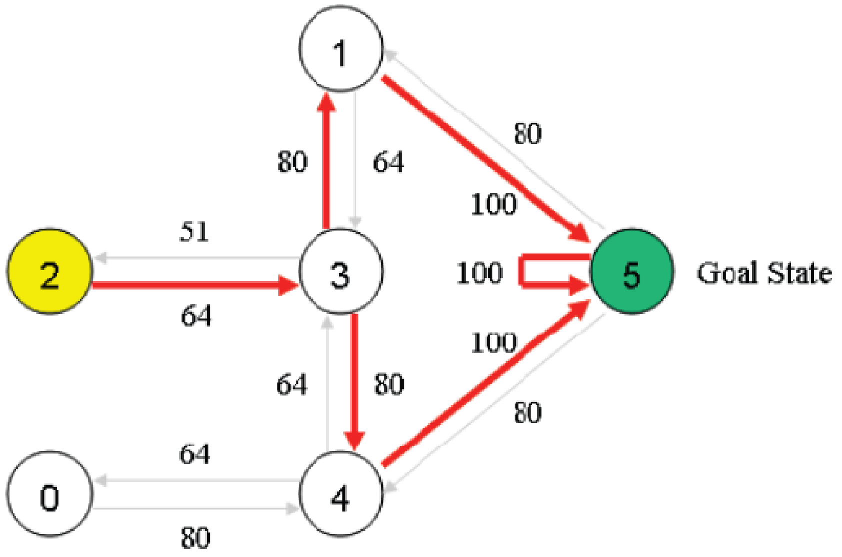
$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
$$

Q(1,5)的值没有发生变化，但因为5已经是目的地，这次迭代结束

- **Q-Learning** Example

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

经过多次迭代后，
Q会逐渐收敛



$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

进行标准化，每个元素都除
以矩阵中最大的元素

Section6. Reinforcement Learning

- Hill Climbing (Evolution Strategy)

  - Initialize "champ" policy $\theta_{\text{champ}} = 0$
  - for each trial, generate "mutant" policy

$$\theta_{\text{mutant}} = \theta_{\text{champ}} + \text{Gaussian noise (fixed } \sigma)$$

  - champ and mutant are evaluated on the same task(s)
  - if mutant does "better" than champ,

$$\theta_{\text{champ}} \leftarrow (1-\alpha)\theta_{\text{champ}} + \alpha\,\theta_{\text{mutant}}$$

Section6. Reinforcement Learning

- REINFORCE Algorithm **Policy Gradients**

Let's first consider episodic games. The agent takes a sequence of actions

$$a_1\ a_2\ \ldots\ a_t\ \ldots\ a_m$$

At the end it receives a reward r_{total}. We don't know which actions contributed the most, so we just reward all of them equally. If r_{total} is high (low), we change the parameters to make the agent more (less) likely to take the same actions in the same situations. In other words, we want to increase (decrease)

$$\log \prod_{t=1}^{m} \pi_\theta(a_t|s_t) = \sum_{t=1}^{m} \log \pi_\theta(a_t|s_t)$$

$$\nabla_\theta\, r_{\text{total}} \sum_{t=1}^{m} \log \pi_\theta(a_t|s_t) = r_{\text{total}} \sum_{t=1}^{m} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

求导可以使用Softmax

We then get the following REINFORCE algorithm:

for each trial
    run trial and collect states s_t, actions a_t, and reward r_{total}
    for t = 1 to length(trial)
        $\theta \leftarrow \theta + \eta(r_{\text{total}} - b)\nabla_\theta \log \pi_\theta(a_t|s_t)$
    end
end

This algorithm has successfully been applied, for example, to learn to play the game of Pong from raw image pixels.

Section6. Deep Reinforcement Learning

- Deep Q-Learning

之前的Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t) \right]$$

现在的Loss Function

sample asynchronously from database and apply update, to minimize

$$\left[r_t + \gamma \max_b Q_w(s_{t+1}, b) - Q_w(s_t, a_t) \right]^2$$

求导项

- Prioritised Replay
  - weight experience according to surprise
- Double Q-Learning
  - current Q-network w is used to select actions
  - older Q-network $\overline{w}$ is used to evaluate actions

Section6. Deep Reinforcement Learning

- Prioritised Replay

将过去的state存入队列中，按照Error从大到小的顺序排列，越大的则越容易被选到

$$|r_t + \gamma \max_b Q_w(s_{t+1}, b) - Q_w(s_t, a_t)|$$

this ensures the system will concentrate more effort on situations where the Q value was “surprising” (in the sense of being far away from what was predicted)

Section6. Deep Reinforcement Learning

- Double Q-Learning

in the context of Deep Q-Learning, a simpler approach is to use the current "online" version of w for selection, and an older "target" version $\overline{w}$ for evaluation; we therefore minimize

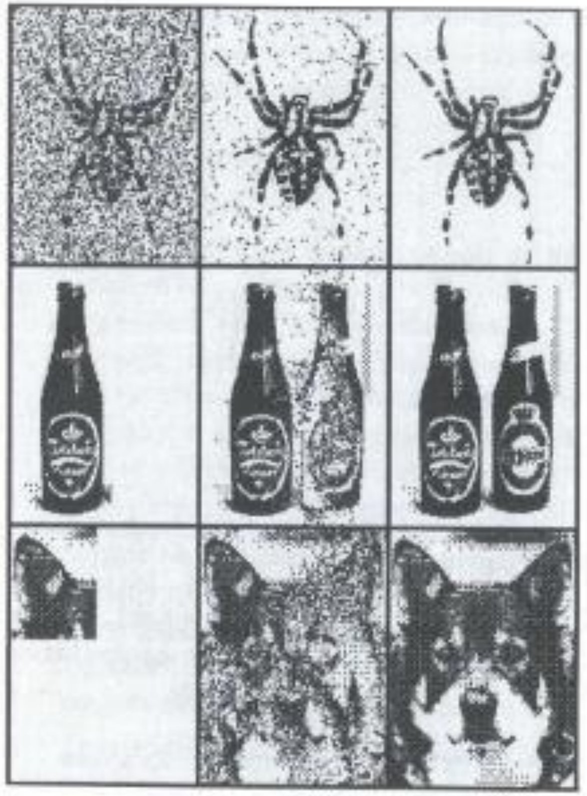
$$[r_t + \gamma Q_{\overline{w}}(s_{t+1}, \operatorname{argmax}_b Q_w(s_{t+1}, b)) - Q_w(s_t, a_t)]^2$$

旧的$\overline{w}$用于evaluate

新的w用于select action

Section7. Boltzmann Machines

- Auto-Associative Memory



我们希望网络可以记忆图片，复原受干扰或者缺损的图片

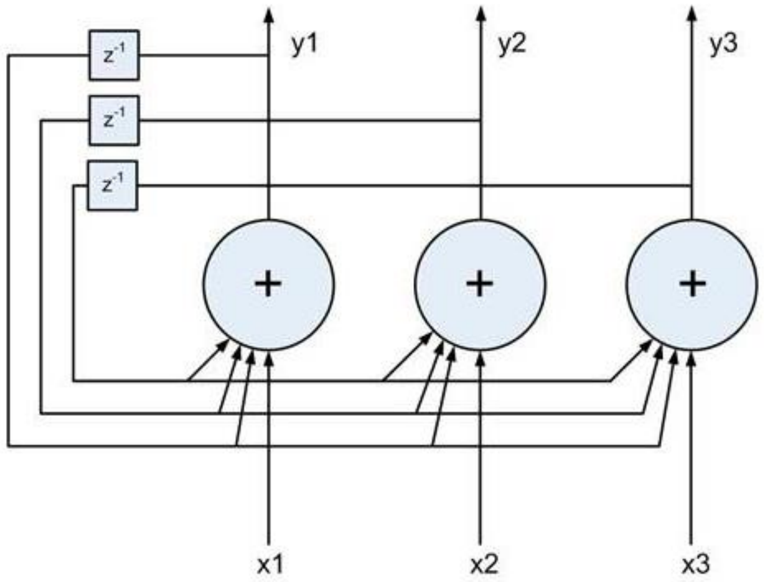
Section7. Boltzmann Machines

- Hopfield Network

$$E(x) = -(\frac{1}{2}\sum_{i,j} x_i w_{ij} x_j + \sum_i b_i x_i)$$

$$w_{ij} = \frac{1}{d}\sum_{k=1}^{p} x_i^{(k)} x_j^{(k)}$$

x的取值只能是1或-1



迭代过程

$$\mathbf{W}_{n\times n} = \frac{1}{n}\sum_{k=1}^{K}\mathbf{y}_k\mathbf{x}_k^T = \frac{1}{n}\sum_{k=1}^{K}\mathbf{y}_k\mathbf{y}_k^T = \frac{1}{n}\sum_{k=1}^{K}\begin{bmatrix} y_1^{(k)} \\ \vdots \\ y_n^{(k)} \end{bmatrix}[y_1^{(k)}, \cdots, y_n^{(k)}]$$

$$x_i \leftarrow \begin{cases} +1, & \text{if } \sum_j w_{ij}x_j + b_i > 0, \\ x_i, & \text{if } \sum_j w_{ij}x_j + b_i = 0, \\ -1, & \text{if } \sum_j w_{ij}x_j + b_i < 0. \end{cases}$$

两种更新方式：同步synchronous / 异步asynchronous

Section7. Boltzmann Machines

- Hopfield Network - Exercise

1. Can the vector [1, 0, –1, 0, 1] be stored in a 5-neuron discrete Hopfield network? If so, what would be the weight matrix for a Hopfield network with just that vector stored in it? If not, why not?

Section7. Boltzmann Machines

- Hopfield Network - Exercise

2.

a. Compute the weight matrix for a Hopfield network with the two memory vectors [1, –1, 1, –1, 1, 1] and [1, 1, 1, –1, –1, –1] stored in it.

$$\mathbf{W}_{n\times n} = \frac{1}{n}\sum_{k=1}^{K}\mathbf{y}_k\mathbf{x}_k^T = \frac{1}{n}\sum_{k=1}^{K}\mathbf{y}_k\mathbf{y}_k^T = \frac{1}{n}\sum_{k=1}^{K}\begin{bmatrix} y_1^{(k)} \\ \vdots \\ y_n^{(k)} \end{bmatrix}[y_1^{(k)},\cdots,y_n^{(k)}]$$

$$x_i \leftarrow \begin{cases} +1, & \text{if } \sum_j w_{ij}x_j + b_i > 0, \\ x_i, & \text{if } \sum_j w_{ij}x_j + b_i = 0, \\ -1, & \text{if } \sum_j w_{ij}x_j + b_i < 0. \end{cases}$$

b. Confirm that both these vectors are stable states of this network.

Section7. Boltzmann Machines

- Hopfield Network

- The number of patterns p that can be reliably stored in a Hopfield network is proportional to the number of neurons d in the network.
- A careful mathematical analysis shows that if $p/d < 0.138$, we can expect the patterns to be stored and retrieved successfully.
- If we try to store more patterns than these, additional, "spurious" stable states may emerge.

$\frac{p}{d} < 0.138$ p: 待存储的模式数量，d:网络中神经元的数量

Section7. Boltzmann Machines

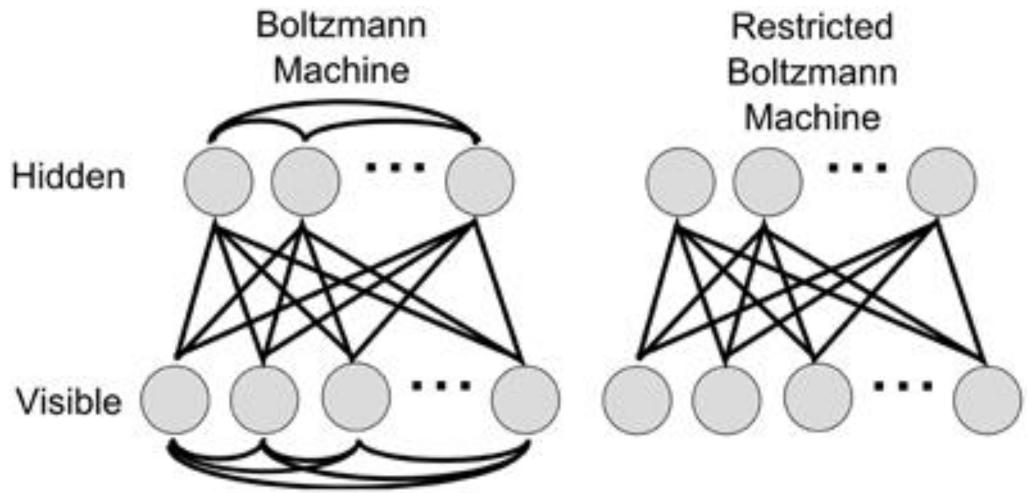
- Boltzmann Machine

Normal

$$E(x) = -(\sum_{i<j} x_i w_{ij} x_j + \sum_i b_i x_i)$$

Restricted

$$E(v,h) = -(\sum_i b_i v_i + \sum_j c_j h_j + \sum_{i,j} v_i w_{ij} h_j)$$



$$p = \frac{1}{1 + e^{-\Delta E/T}}$$

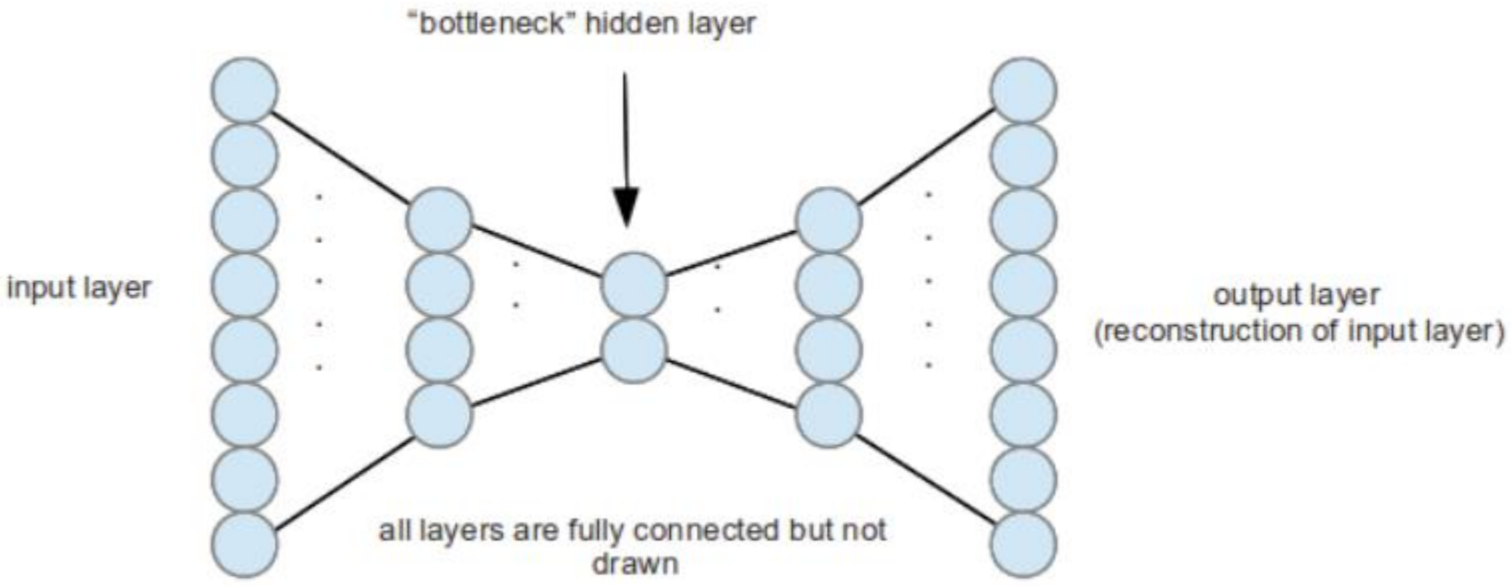
更新概率（模拟退火）

The Boltzmann Machine is very similar to the Hopfield Network, except that

- components (neurons) x_i take on the values $0, 1$ instead of $-1, +1$
- used to generate new states rather than retrieving stored states
- update is not deterministic but stochastic, using the sigmoid

Section8. Auto-Encoder

- Autoencoder Networks



一般用于Pretraining

$$E = L(x, g(f(x)))$$

先编码后再解码，看和原来的输入相差多大

- after an autoencoder is trained, the decoder part can be removed and replaced with, for example, a classification layer
- this new network can then be trained by backpropagaiton
- the features learned by the autoencoder then serve as initial weights for the supervised learning task

Section8. Auto-Encoder

- Regularization **Avoiding Trivial Identity**

如果网络的hidden layer中node太多超过输入层，则有可能学到太多不重要的细节

Sparse Autoencoder

$$E = L(x, g(f(x)) + \lambda \sum_i |h_i|$$

把activation的总量作为penalty –
降低激活节点数

Contractive Autoencoder

$$E = L(x, g(f(x)) + \lambda \sum_i ||\nabla_x h_i||^2$$

把导数值总和作为penalty –
强迫网络学习不太变化的主要特征，放
弃变化剧烈的细节

Section8. Auto-Encoder

- Generative Models

For autoencoders, the decoder can be seen as defining a conditional probability distribution $p_\theta(x|z)$ of output x for a certain value z of the hidden or "latent" variables.

我们希望网络能通过输入样本学习到能生成类似数据的参数分布

In some cases, the encoder can also be seen as defining a conditional probability distribution $q_\phi(z|x)$ of latent variables z based on an input x.

Variational Autoencoder

In other words, we want to be able to choose latent variables z from a standard Normal distribution $p(z)$, feed these values of z to the decoder, and have it produce a new item x which is somehow similar to the training items.

maximize $$\mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)}|z)\right] - D_{\mathrm{KL}}\left(q_\phi(z|x^{(i)})\|p(z)\right)$$

在参数z时能够生成样本x的条件概率（越大越好）

参数z的实际概率分布和条件概率分布尽可能一致（越小越好）

从样本学到的参数分布（符合正态分布）

Section9. Adversarial Training and GANs

- Generative Adversarial Networks

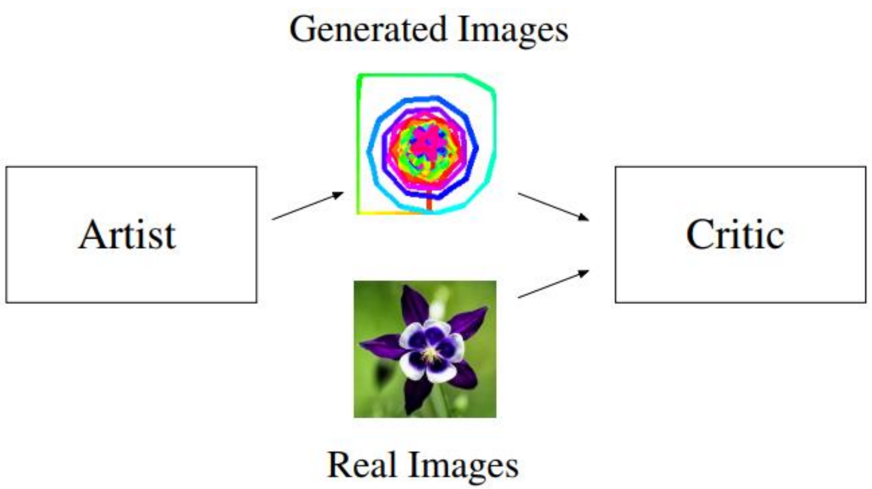
Gradient ascent on Discriminator:

$$\max_{\psi}\left(\mathbf{E}_{x\sim p_{\text{data}}}\left[\log D_{\psi}(x)\right]+\mathbf{E}_{z\sim p_{\text{model}}}\left[\log\left(1-D_{\psi}(G_{\theta}(z))\right)\right]\right)$$

Gradient descent on Generator, using:

$$\min_{\theta}\ \mathbf{E}_{z\sim p_{\text{model}}}\left[\log\left(1-D_{\psi}(G_{\theta}(z))\right)\right]$$

D_{ψ} 图像被判定为真实的概率



Section9. Adversarial Training and GANs

- Generative Adversarial Networks

  - Like any coevolution, GANs can sometimes oscillate or get stuck in a mediocre stable state.
  - **oscillation**: GAN trains for a long time, generating a variety of images, but quality fails to improve (compare IPD)
  - **mode collapse**: Generator produces only a small subset of the desired range of images, or converges to a single image (with minor variations)

Methods for avoiding mode collapse:

- Conditioning Augmentation
- Minibatch Features (Fitness Sharing)
- Unrolled GANs

Nash equilibrium is not promised to achieve. Your opponent may always find a countermeasure.

Questions