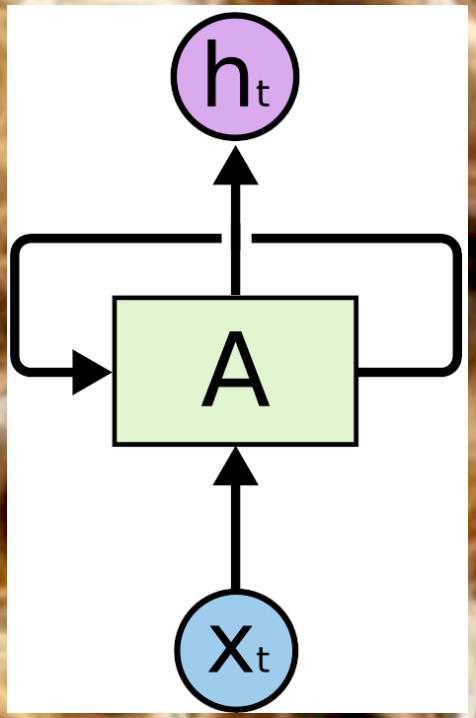
数据驱动的人工智能（5）循环神经网络

Data Driven Artificial Intelligence

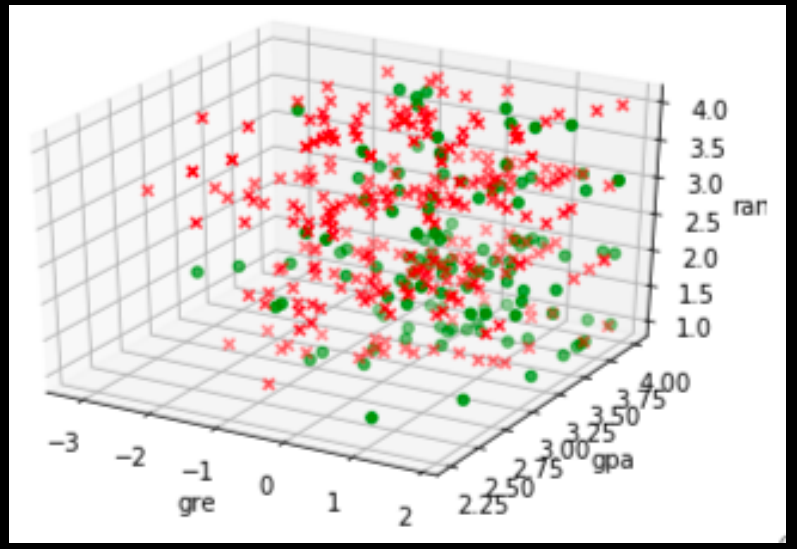
邬学宁 SAP硅谷创新中心

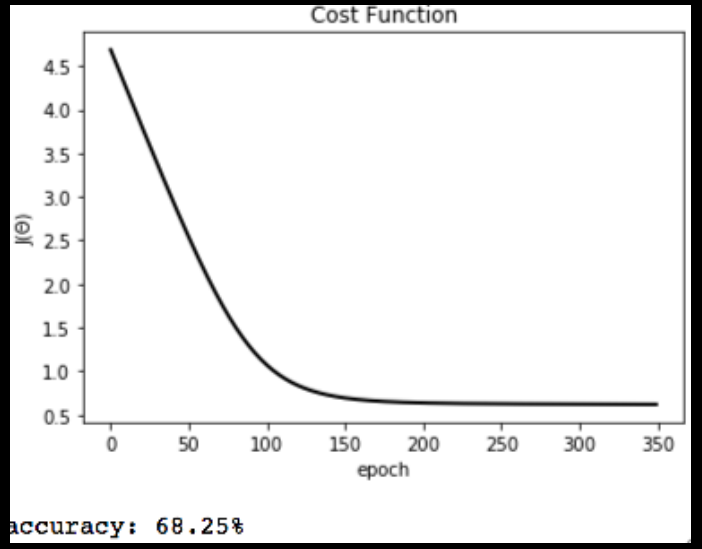
2017 / 04

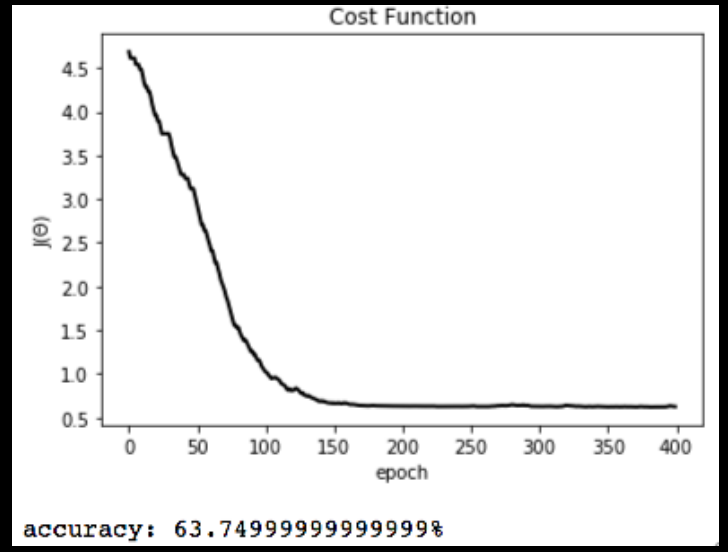


Logistic Regression Assignment

Actual

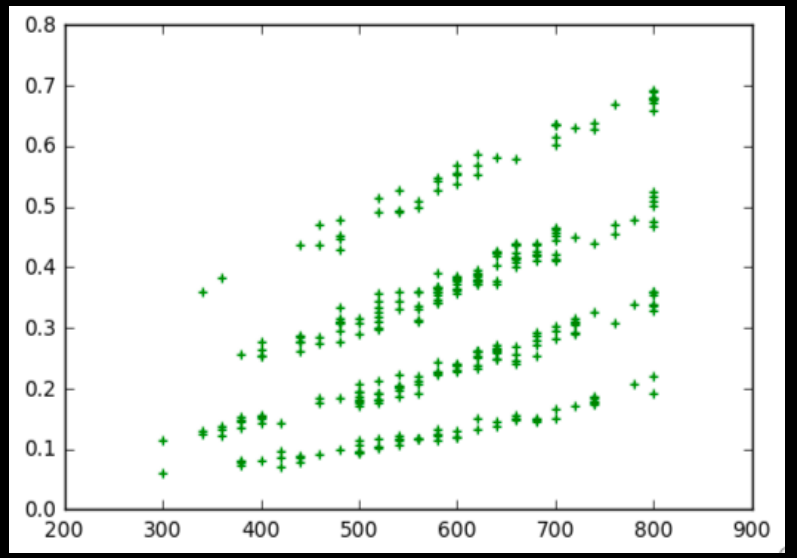


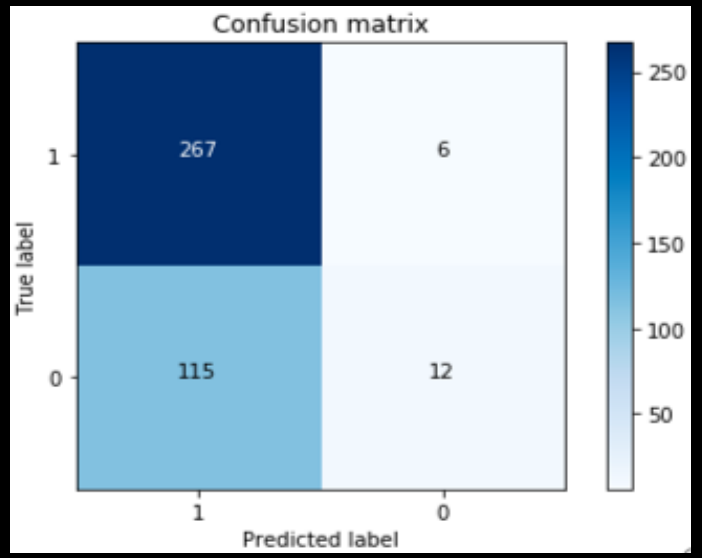




陈继麟

Predict





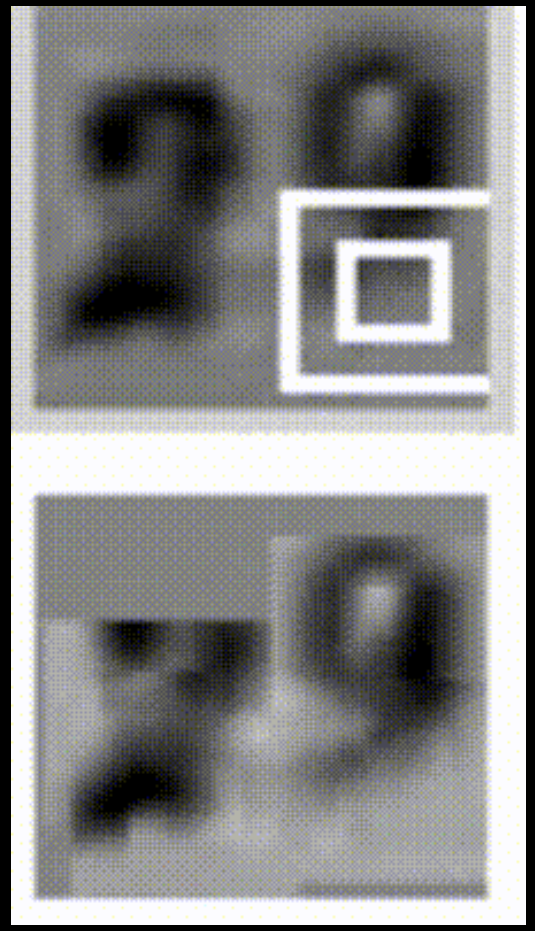
日程

RNN应用概述
时间序列 / LSTM／GRU
自然语言处理基础
Word Embedding
RNN实现案例
Bayes Learning / Generative Learning
Bayes Network

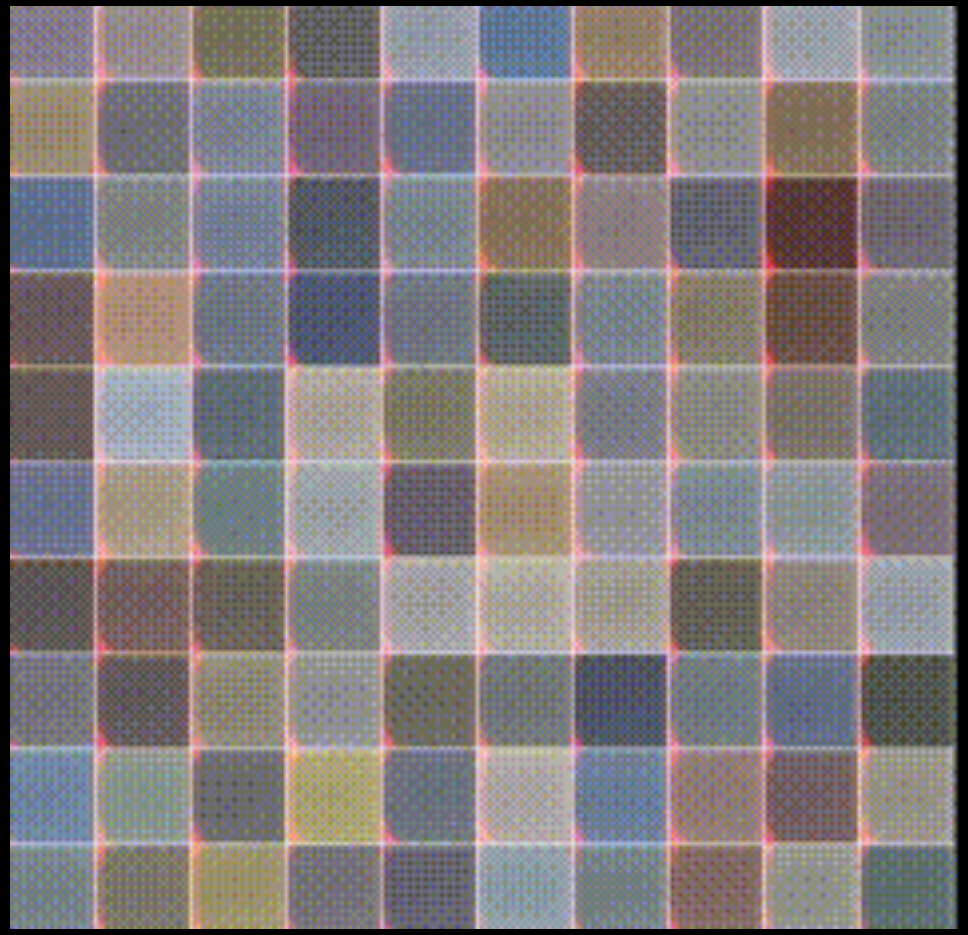
SAP

FUDAN UNIVERSITY 1905 復旦大學

RNN with attention / RNN generate



Ba et al.



Gregor et al.

AI play Atari = CNN + 强化学习

Training after **10min**

Training after **120min**

Training after **240min**

自然语言处理NLP

- NLP是CS，AI和语言学的交叉领域
- 目标：计算机理解和处理自然语言
- 完全的理解和表达语言的含义非常困难
- 完美的语言理解已属于“AI-Complete”

NLP应用

- 拼写检查，联想输入法，关键字搜索
- 网站信息提取（产品价格，日期，公司等信息）
- 情绪分析，分档分类
- 机器翻译，语音识别
- 对话系统，机器人客服
- 复杂问题回答

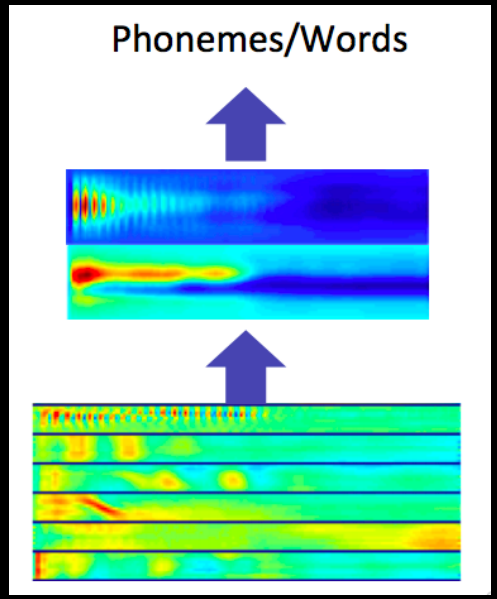
难度增大

深度学习的突破始于语音识别

- 基于上下文预训练的深度神经网络方法应用于大词汇量语音识别

Dahl et al. (2010)

Acoustic model	Recog \ WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass –adapt	**27.4**	**23.6**
Deep Learning	1-pass –adapt	**18.5** (–33%)	**16.1** (–32%)



音位表达

CONSONANTS (PULMONIC) © 2005 IPA

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

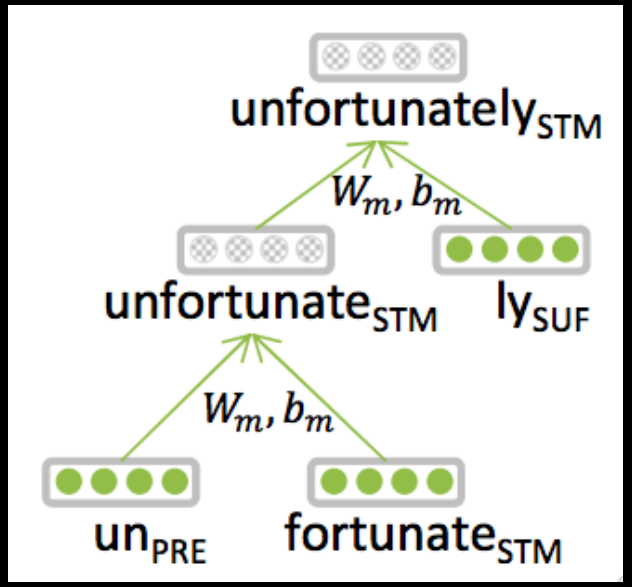
- 基于声音特征，深度学习预测音位或直接预测Word，并以向量来表达

词法表达

传统表达：词素　　前缀　　词干　　后缀

un　　fortunate　　ly

- 深度学习：
  - 每个词素对应一个向量
  - 算法对词素向量进行组合
  - Thang et al. 2013

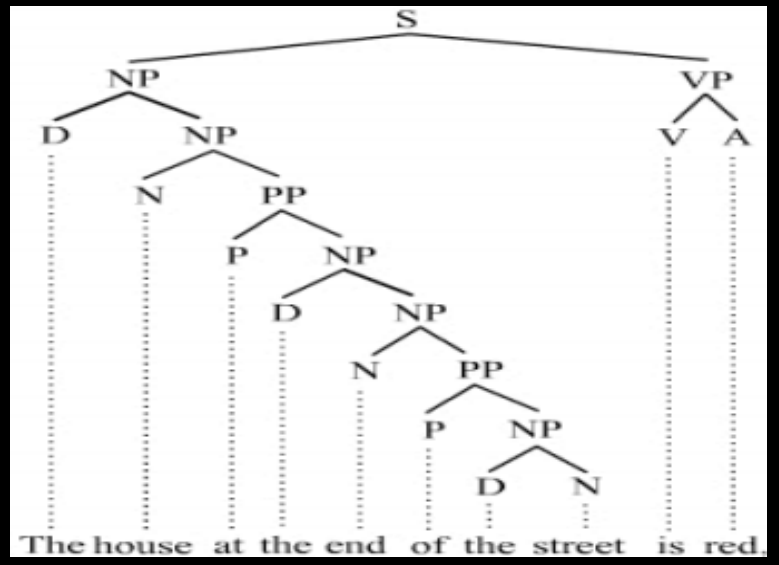


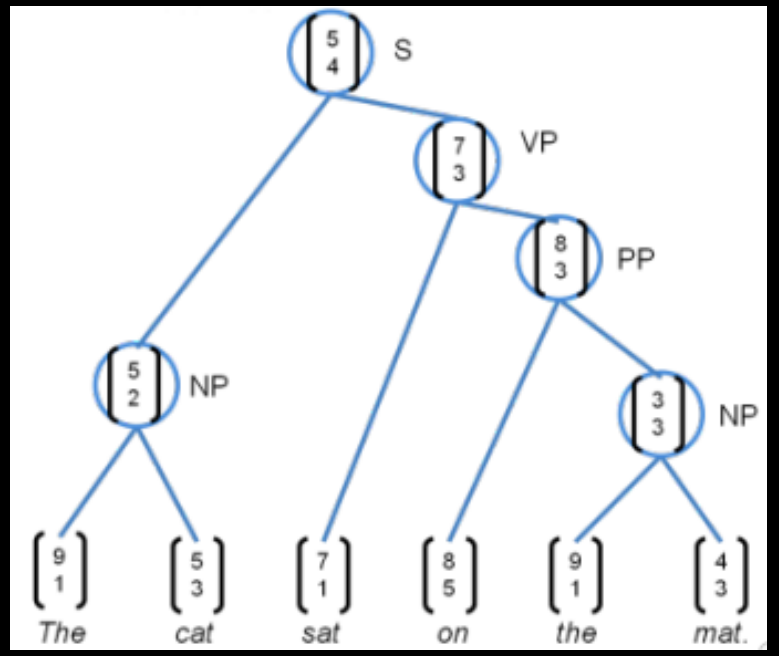
cs224d

语法表达

- 传统表达：短语
- 离散分类如：
  - 名词短语（NP）
  - 动词短语（VP）
  - 介词短语（PP）

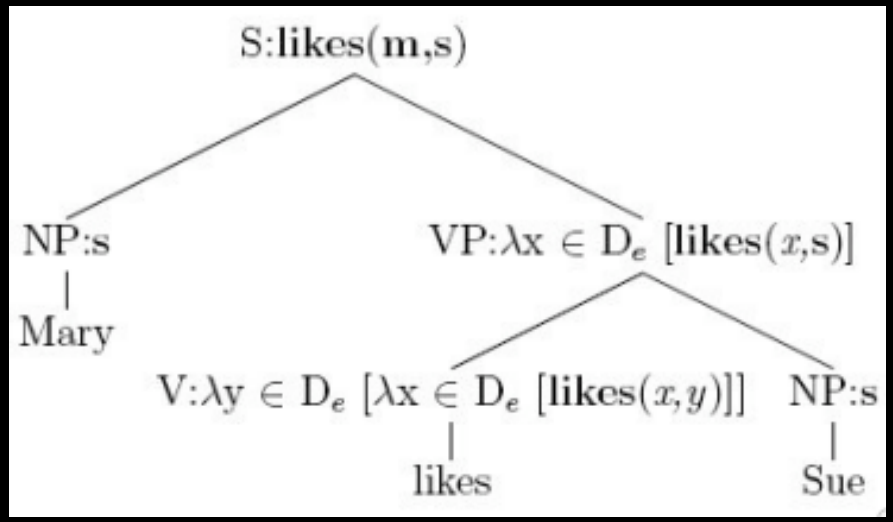
- 深度学习：
  - 每个词或短语对应一个向量
  - 算法对向量进行组合
  - Socher et al. 2011



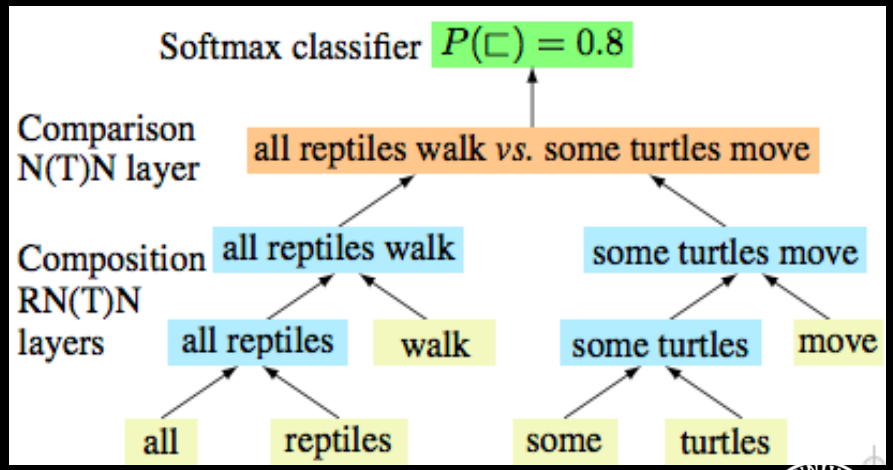


语义表达

- 传统表达：Lambda Calculus
  - 精心设计的函数(无函数名，1个输入变量)
  - 作为其他函数的输入
  - 缺乏语义的相似度标记



- 深度学习：
  - 每个词或短语对应一个向量
  - 算法对向量进行组合
  - Socher et al. 2011

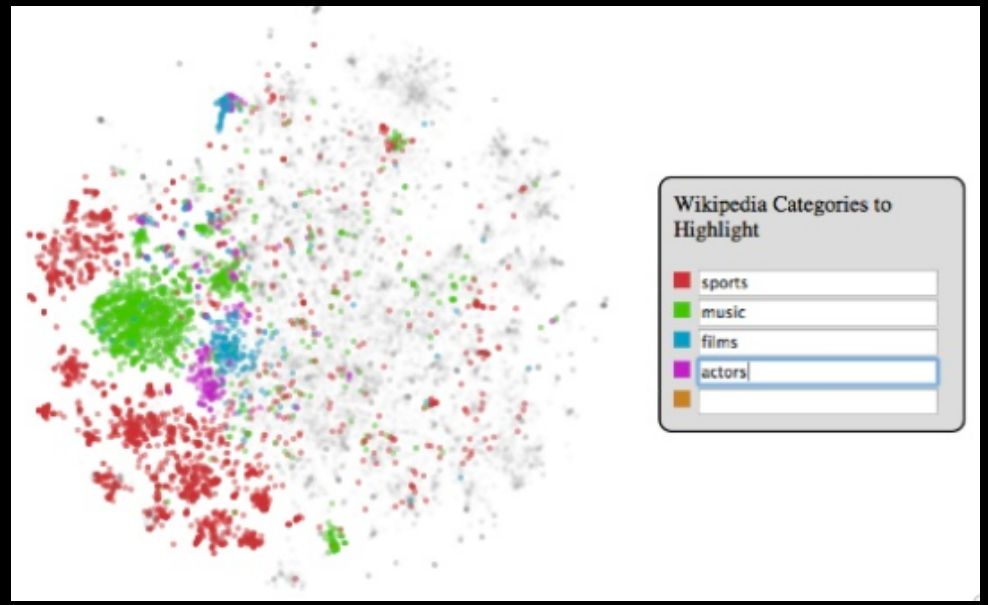


cs224d

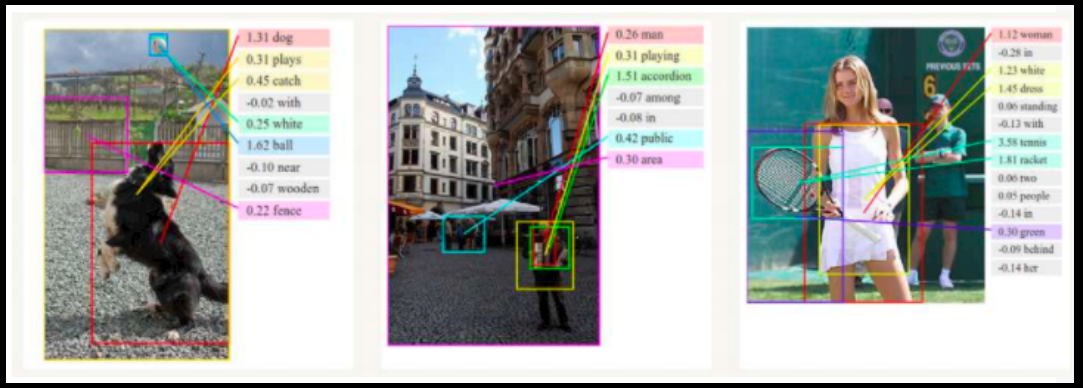
问题回答

- 传统方法：利用正则表达式等方法进行复杂的特征工程，以捕捉知识

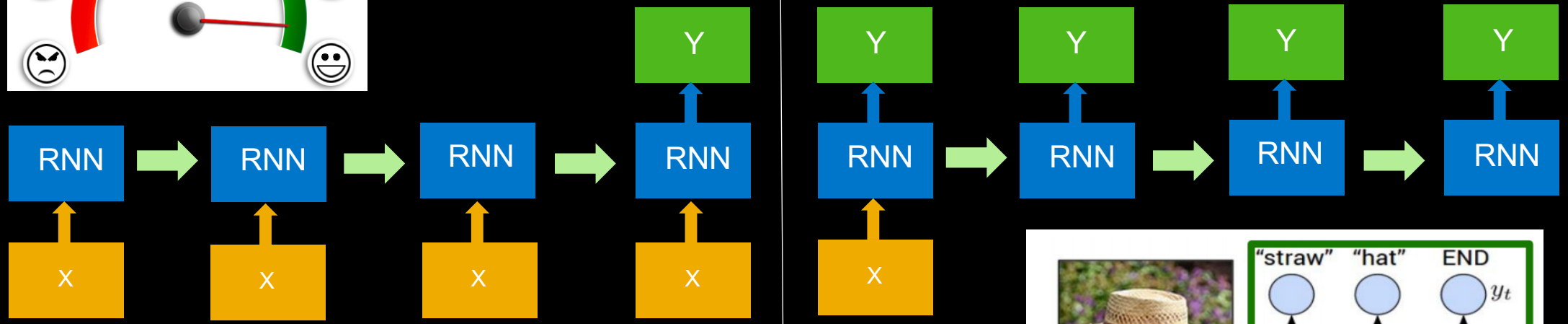
- 深度学习：
  - 可以使用词法、语法、语义和情绪使用的深度学习模型
  - Facts被作为矢量存储

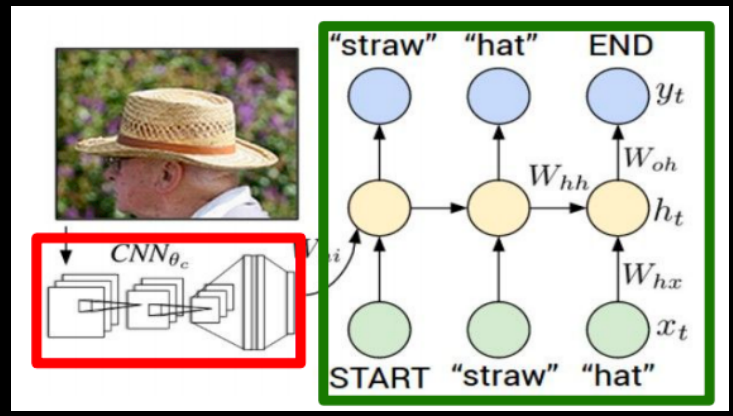


情绪分析 / 图片描述

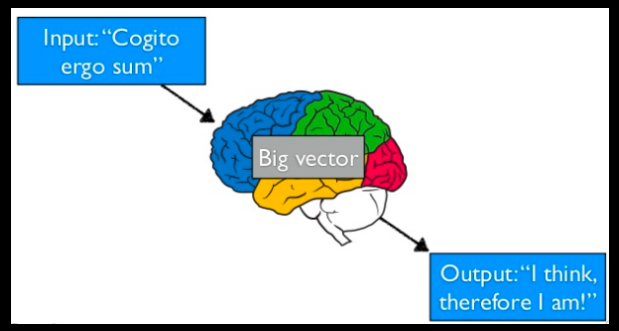


http://cs.stanford.edu/people/karpathy/deepimagesent/



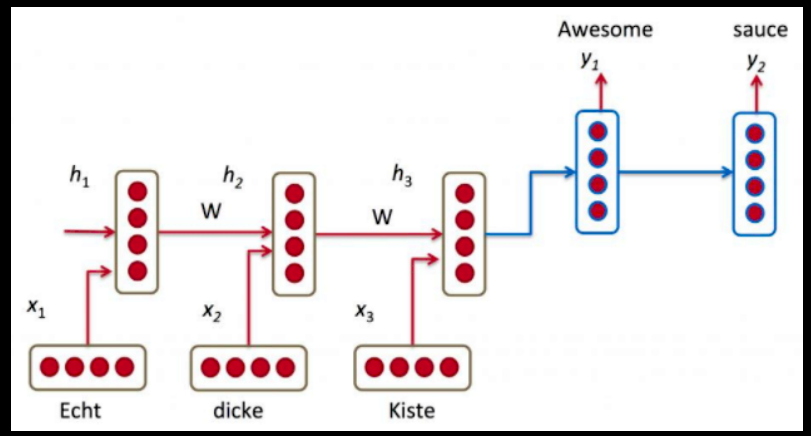


机器翻译 / 写论文 / 写代码 / 作曲



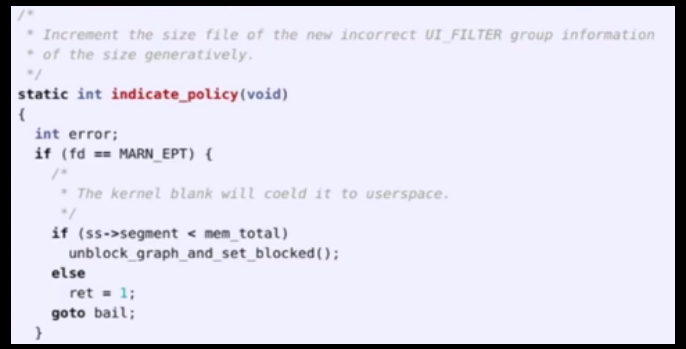
Jeff Dean

- 深度学习正在替换手工的异常复杂的架构
- Google Neural Machine Translation (2016 Nov)
- sentence *embedding*
- Skype translator (API 2016)
- 作业：https://www.tensorflow.org/versions/master/tutorials/seq2seq

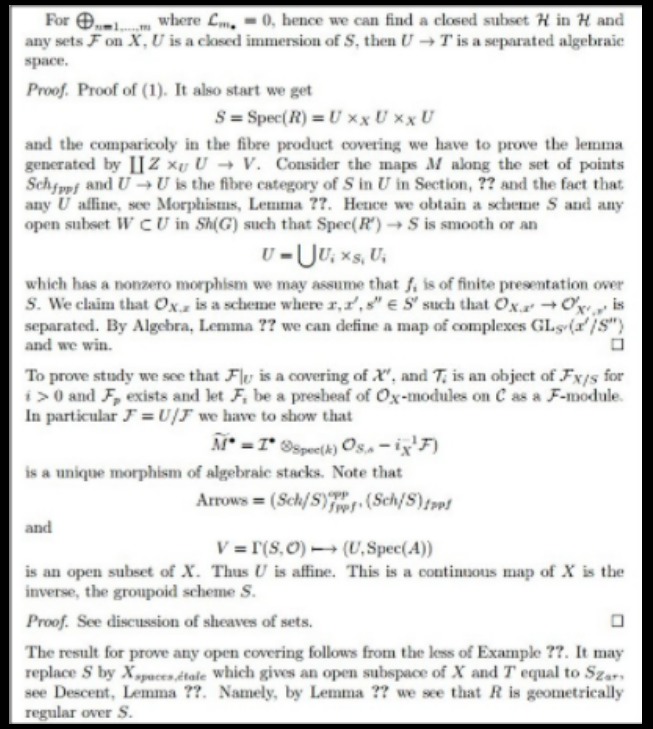


NMT

Seq2Seq



```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
```



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_*} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariccly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',s''}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longrightarrow (U, \mathrm{Spec}(A))$$

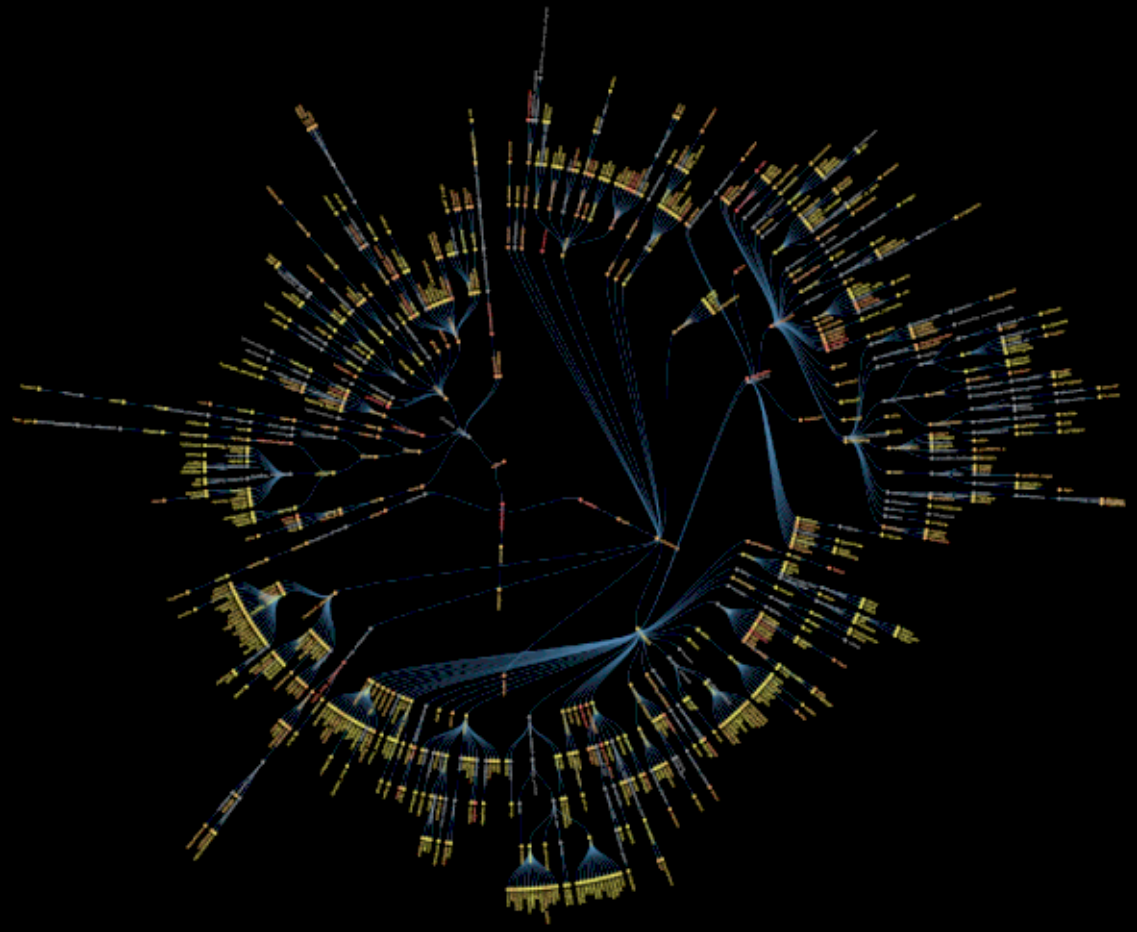
is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Justin Johnson

语言学家如何表达Word的含义



Emiel van Miltenburg

Wordnet Graph 'entity'

通过同义词和近义词来表示

但存在明显的不足：

- 近义词之间的细微差别缺失
- 新词缺失
- 主观
- 人力成本高
- 词的相似度难以计算

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
```

统计学家：Language Model

- 语言模型：估计**word sequence**的概率
  - Word Sequence: $w_1, w_2, w_3, \ldots, w_n$
  - P$(w_1, w_2, w_3, \ldots, w_n)$
- 应用：语音识别，语句生成
  - 发音类似语句的语音识别，选择概率大的Word Sequence

recognize speech

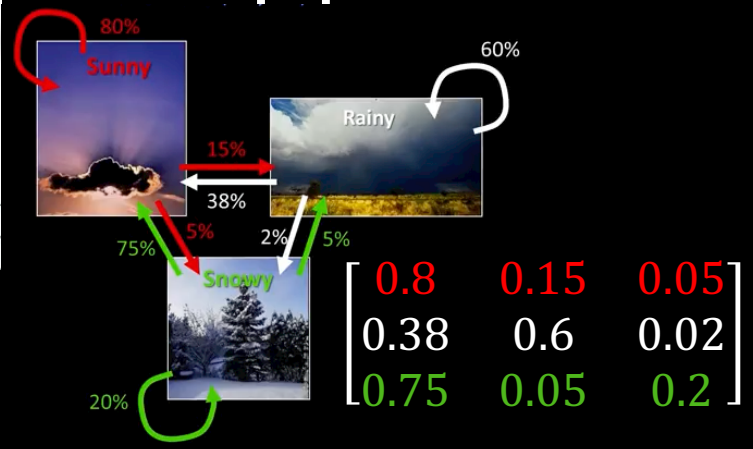
or

wreck a nice beach

语言模型 Language Model

- 语言模型计算词序列的概率 $P(w_1, w_2, w_3, \ldots, w_n)$
- 应用场景例如：机器翻译时(FR->EN)：
  - 词序 p(the cat is small) > p(small the is cat)
  - 词选择 p(walking home after school) > p(walking house after school)
- $P(w_1, w_2, w_3, \ldots, w_m) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_m|w_1, \cdots, w_{m-1})$

$$= \prod_{i=1}^{m} P(w_i|w_1 \cdots w_{i-1}) \approx \prod_{i=1}^{n} P\left(w_i|w_{i-(n-1)} \cdots w_{i-1}\right)$$



$$\begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.38 & 0.6 & 0.02 \\ 0.75 & 0.05 & 0.2 \end{bmatrix}$$

Markov Assumption

- 1st order Markovian: Probability of moving to a giving state depends only on current state
- 2nd order Markovian: Probability of moving to a giving state depends on current state & previous state

N-gram

- 目的：估算 P($w_1, w_2, w_3, \ldots, w_n$)
- 直觉：对包含大量文本的语料库(Corpus)进行统计
- 问题：在整个语料库中可能也找不到Word Sequence: $w_1, w_2, w_3, \ldots, w_n$
- 方法：Bi-gram 语言模型：

$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1|START)P(w_2|w_1) \cdots P(w_n|w_{n-1})$$

$$P(w_i|w_{i-1}) = \frac{Count(w_{i-1} \& w_i)}{Count(w_{i-1})}$$

- 以此类推： Bi-gram只考虑前一个字，考虑前2个词就是Tri-gram, 前3个词就是4-gram … …

语言模型的评价标准: Perplexity

- 以在整个数据集（词的数量）T的上的Log概率的均值作为评估函数：

$$\mathcal{J} = -\frac{1}{T}\sum_{T=1}^{T}\sum_{j=1}^{|V|} y_{t,j} log\hat{y}_{t,j}$$

$\hat{y} \in \mathbb{R}^{|V|}$, 是在整个词汇表上的概率分布

- 常用Perplexity（ 2^J ）来评价语言模型（整个语料库的概率分布）(越小越好)

Perplexity(复杂度 / 混乱度)

Goodman的“A bit of progress in language modeling”中的分析结果($|v| = 50{,}000$)：

- Trigram Model： $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i|w_{i-1}, w_{i-2})$

  Perplexity = 74

  as if the model has to choose uniformly or independently among 74 possibility each word

- Bigram Model： $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i|w_{i-1})$

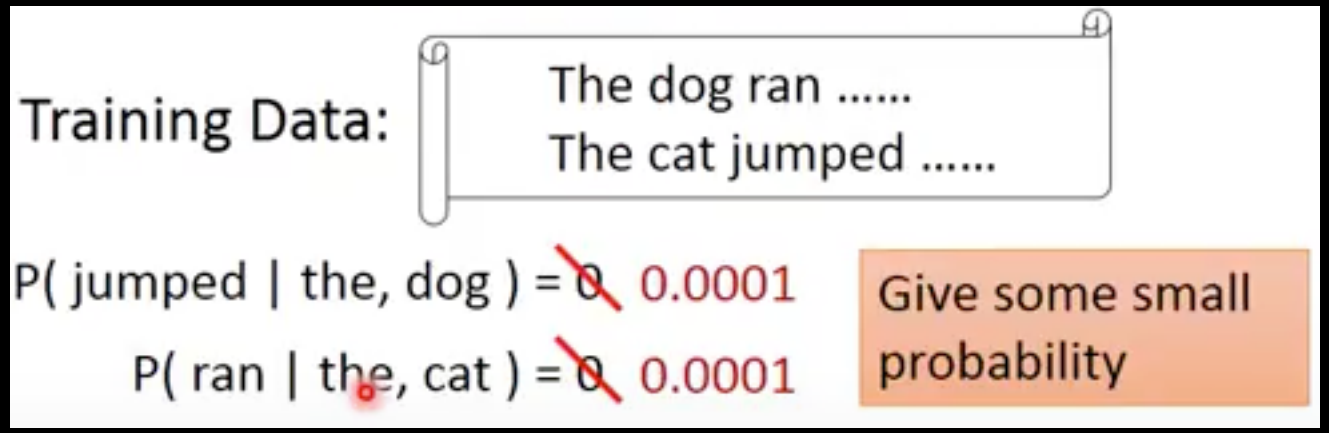
  Perplexity = 137

- Unigram Model： $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} p(w_i)$

  Perplexity = 955

传统语言模型的不足

- 更高的n-grams模型表现更好
- N很大时，导致巨量的n-grams，和巨量的内存消耗
- 也会导致data sparsity (大模型，数据量不够)
- 目前最高纪录，使用一台140GB内存的机器运行2.8天，在1260亿个Token上建立了一个未被修剪的模型



语言模型平滑

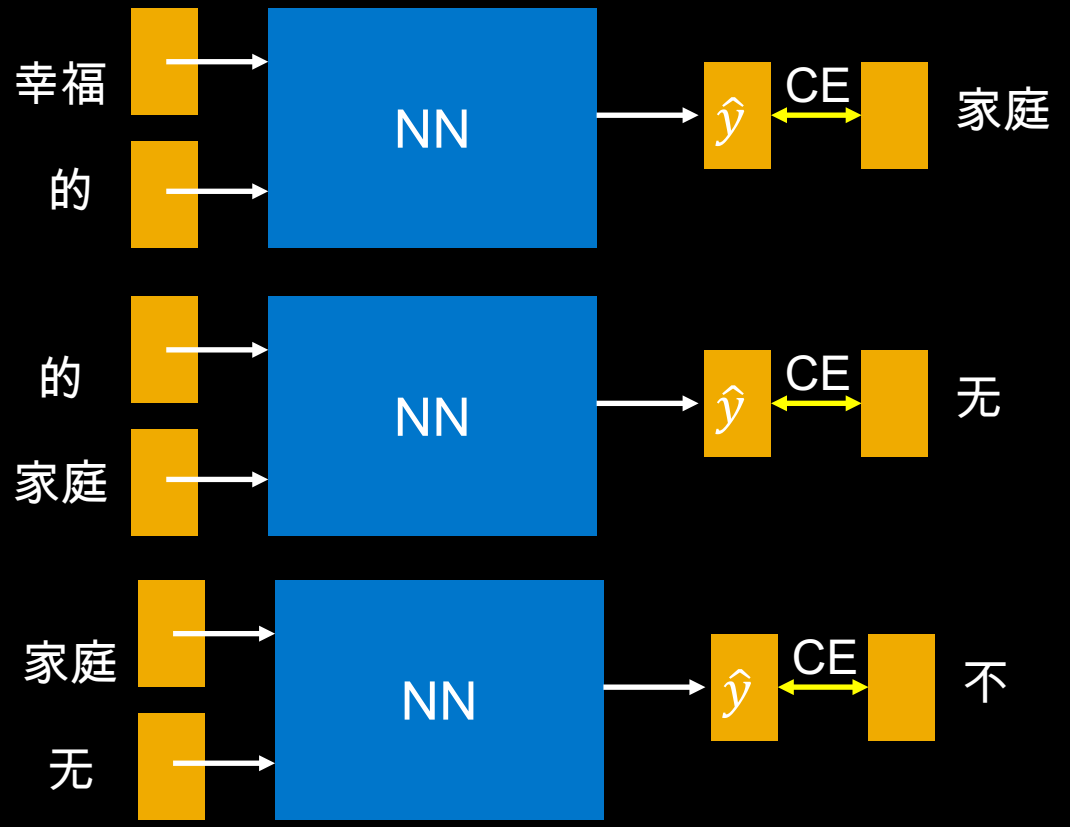
用神经网络来实现语言模型

Training：

语料库：

【幸福的家庭無不相似，不幸的家庭各有不幸。】

最小化交叉熵CE



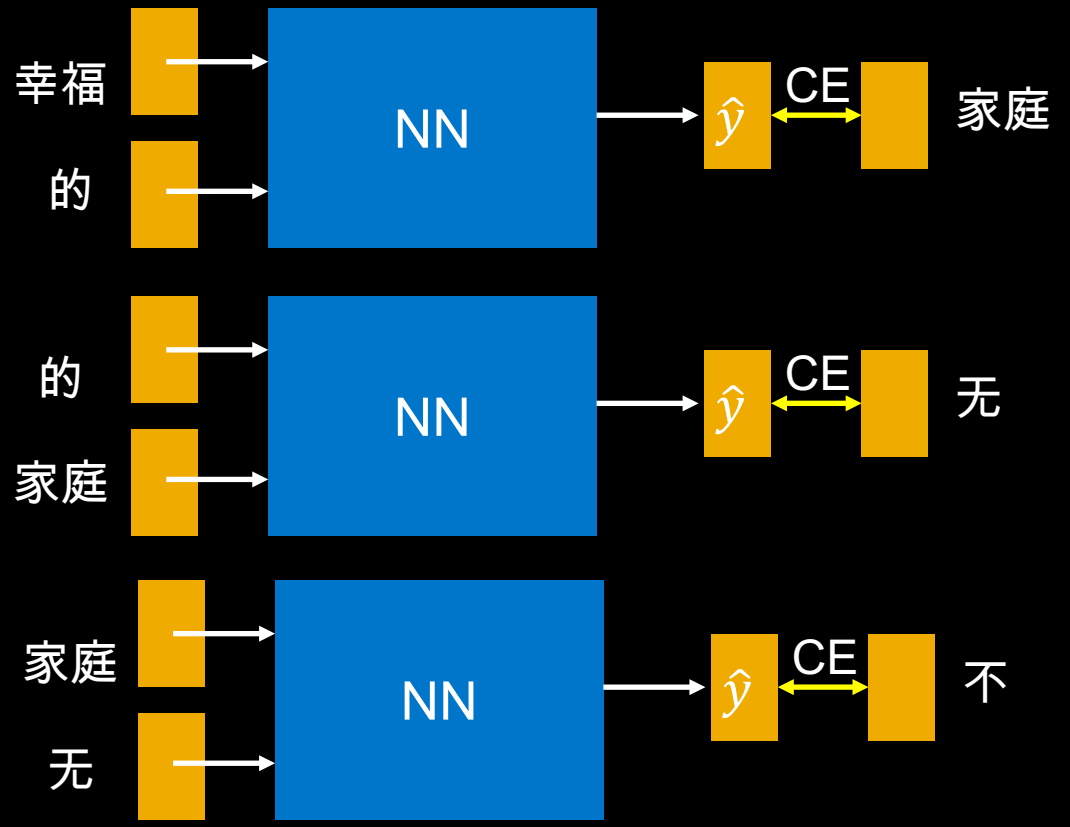
用神经网络来实现语言模型

$P(w_1, w_2, w_3, \ldots, w_n)$
$= P(w_1|START)P(w_2|w_1) \cdots P(w_n|w_{n-1})$

$P(w_i|w_{i-1})$：NN预测的下一个词的概率
（不再是基于统计的结果）

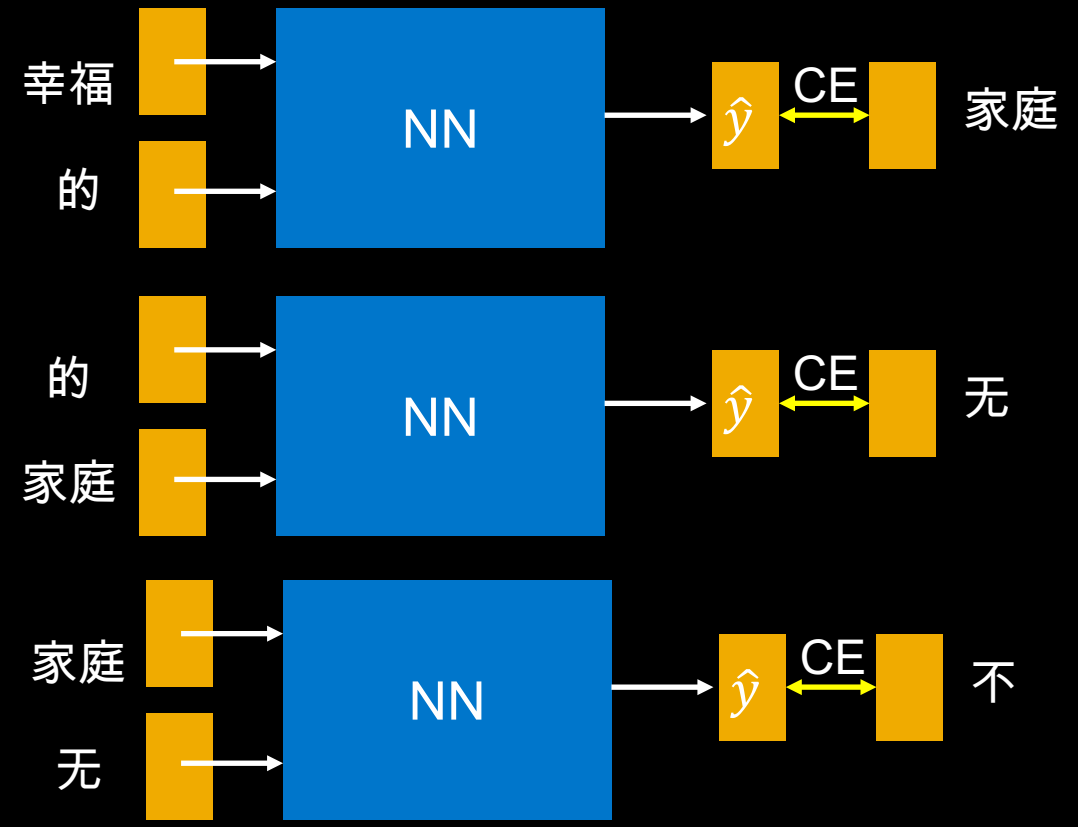
NN的输入： one-hot vector

NN的输出： Softmax多分类概率P

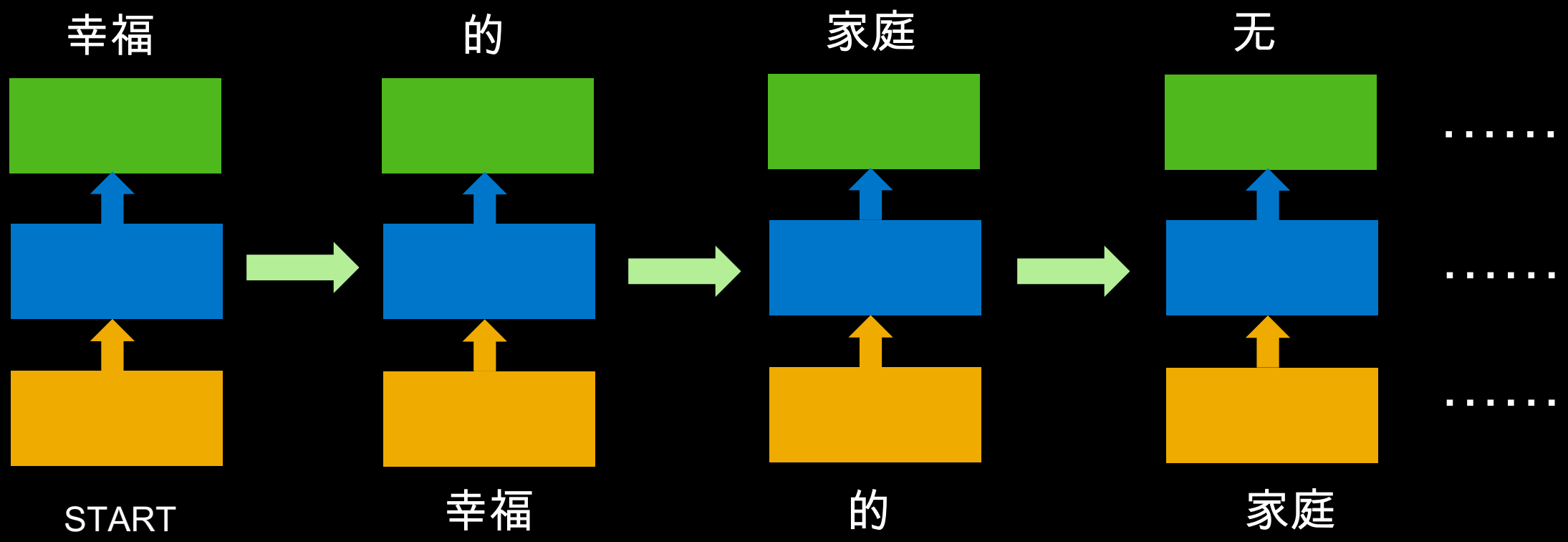


前馈式神经网络的不足

虽然MLP取得了极大的成功，但是因为MLP假设所有数据点的输入矢量都是固定长度的，这限制了对时间关系的建模。

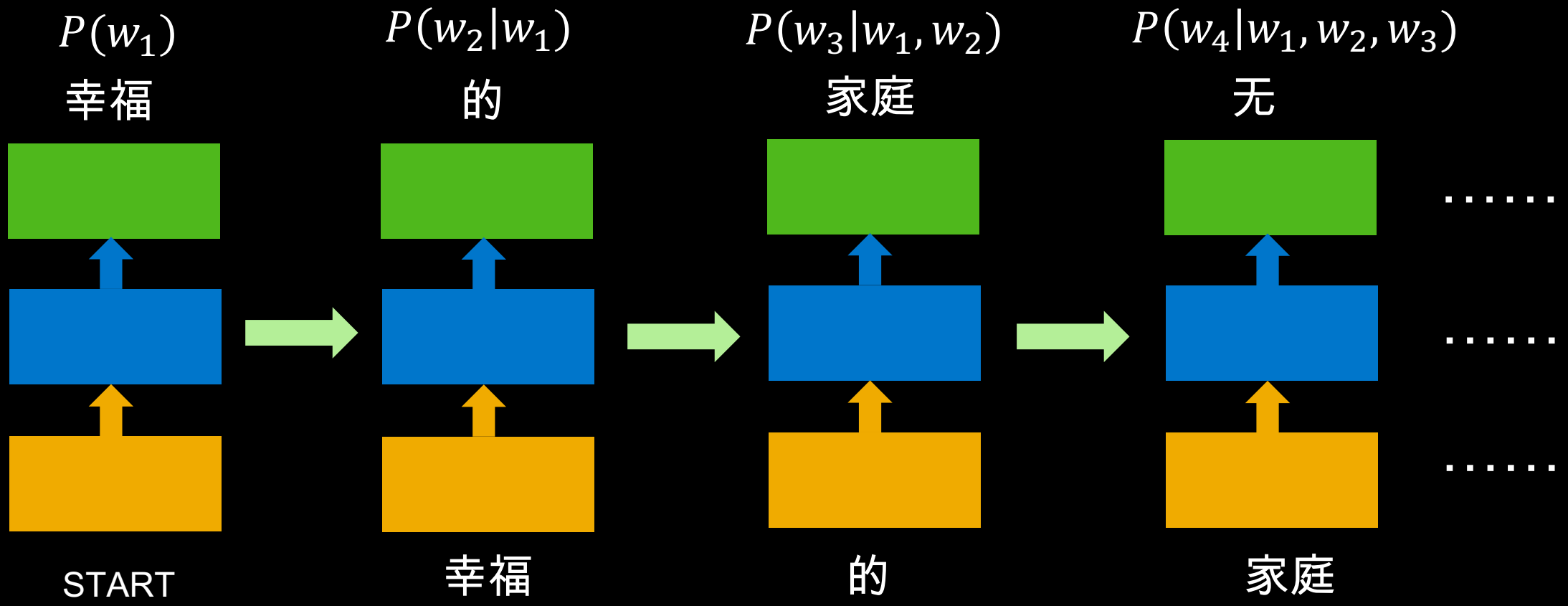


基于循环神经网络RNN的语言模型



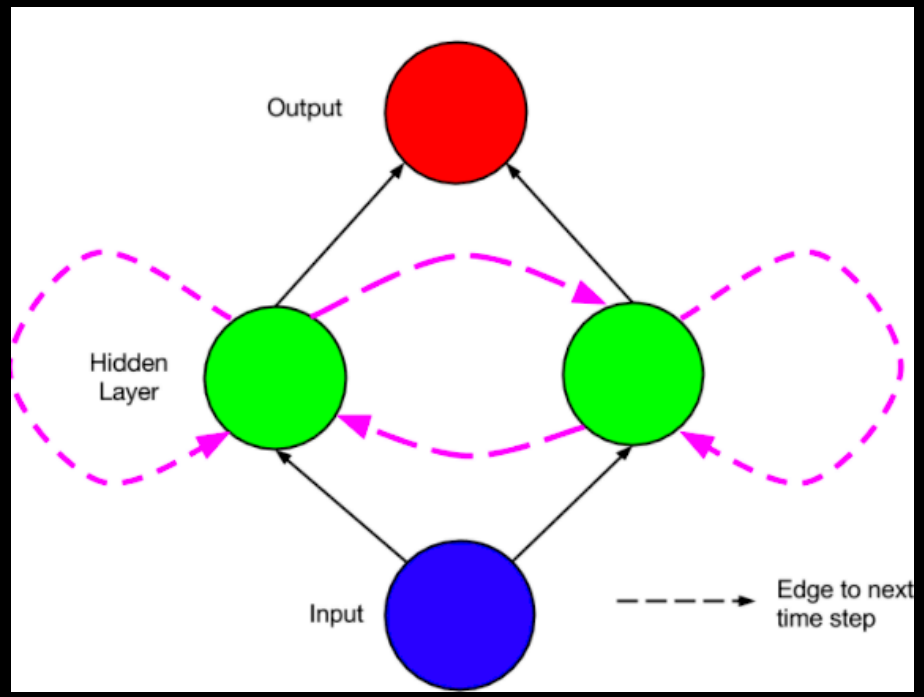
基于循环神经网络RNN的语言模型实现

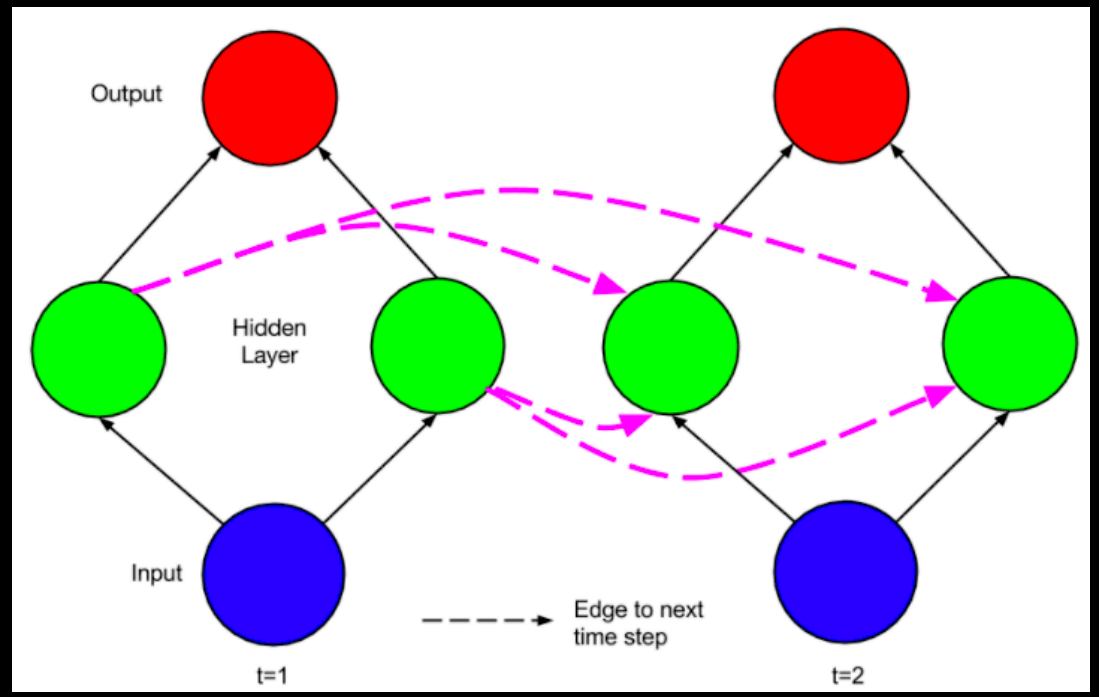
$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \cdots, w_{n-1})$$



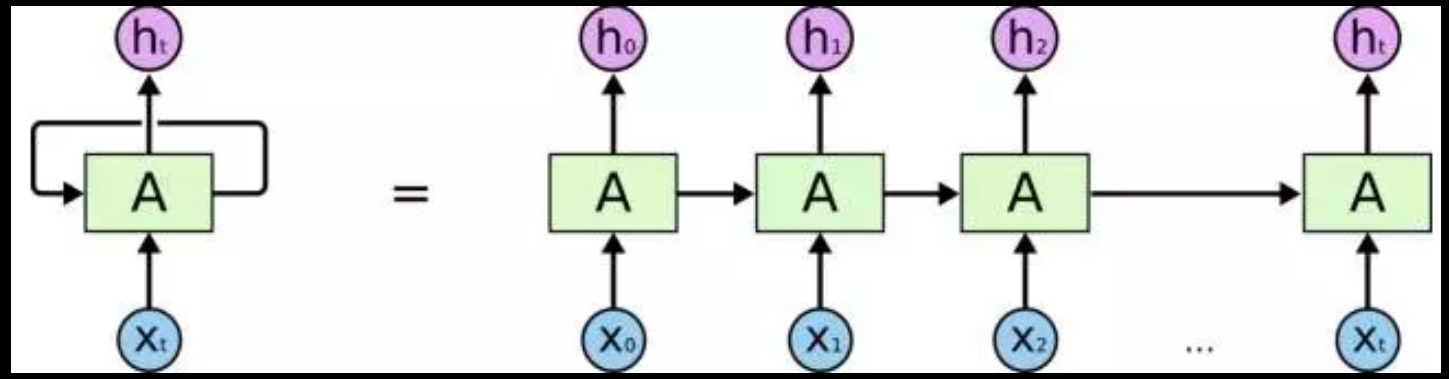

```
lstm = rnn_cell.BasicLSTMCell(lstm_size)
# 初始化 LSTM 存储状态.
state = tf.zeros([batch_size, lstm.state_size])
```

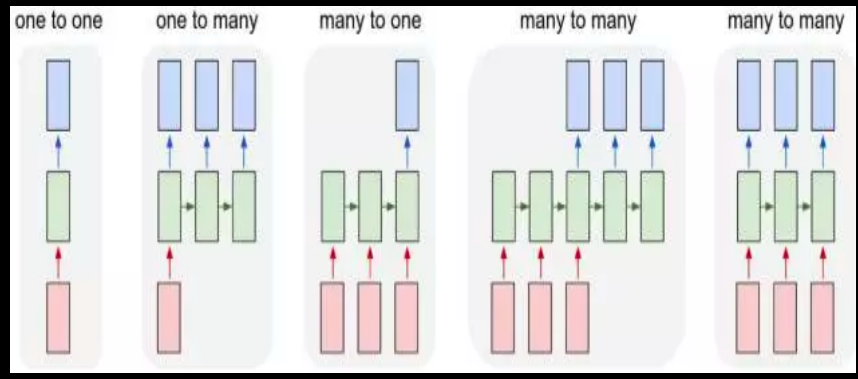
RNN工作原理

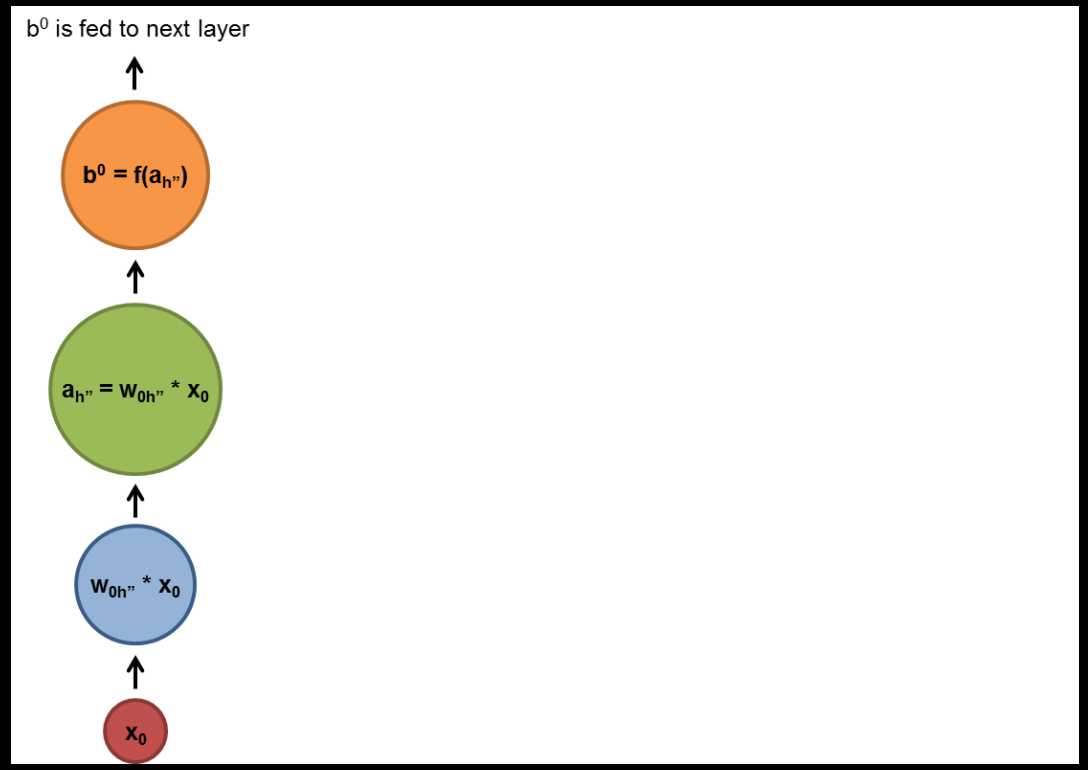

Output
Hidden
Layer
Input
Edge to next
time step

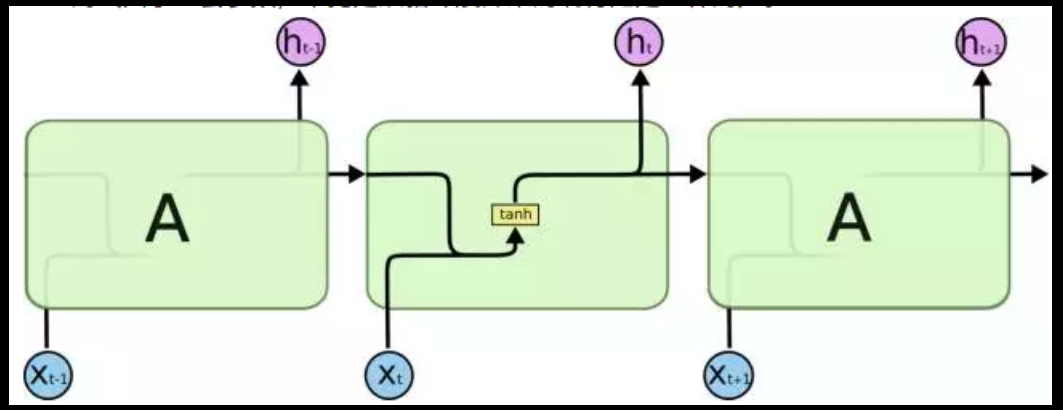

Output
Hidden
Layer
Input
Edge to next
time step
t=1
t=2

RNN工作原理





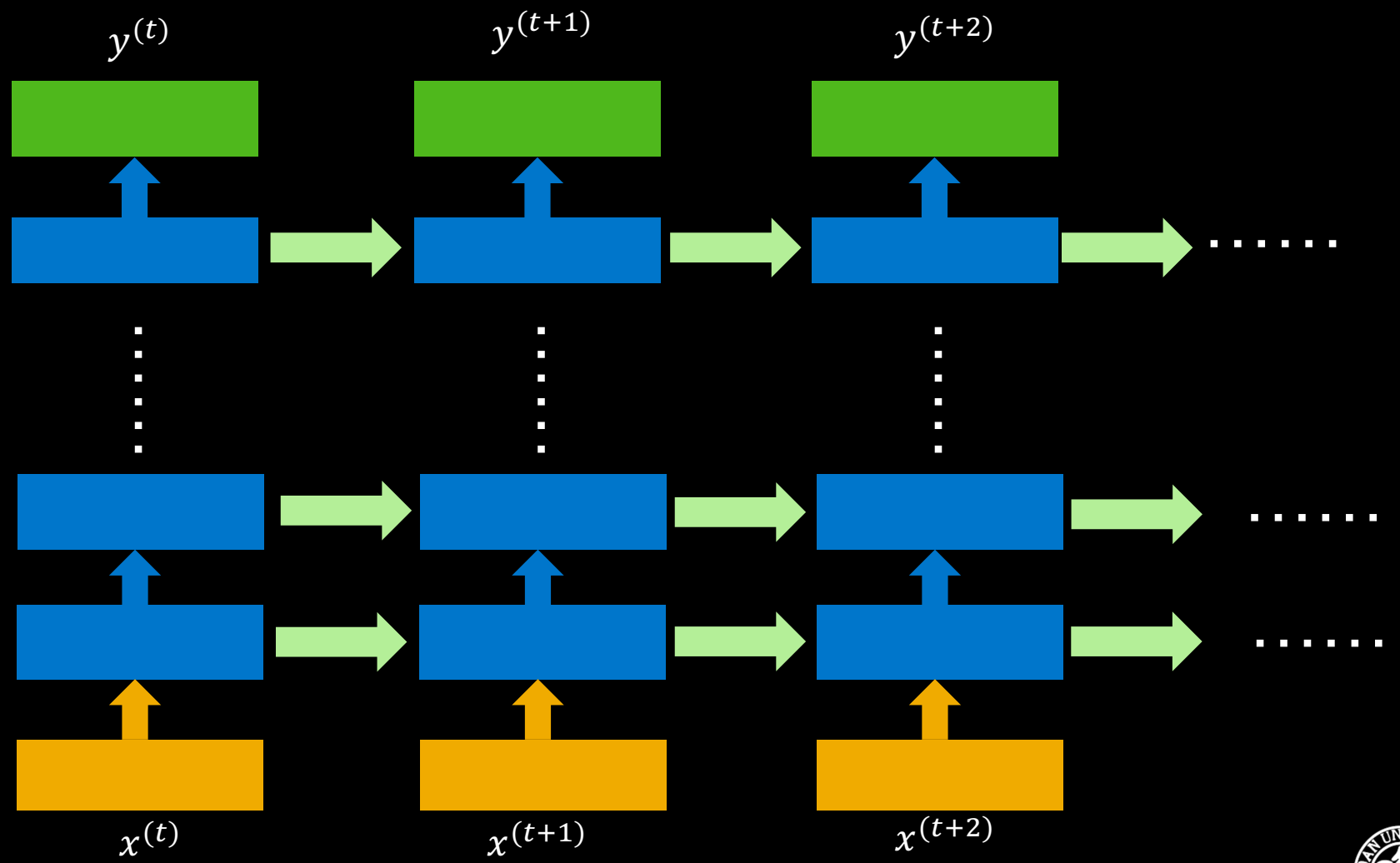




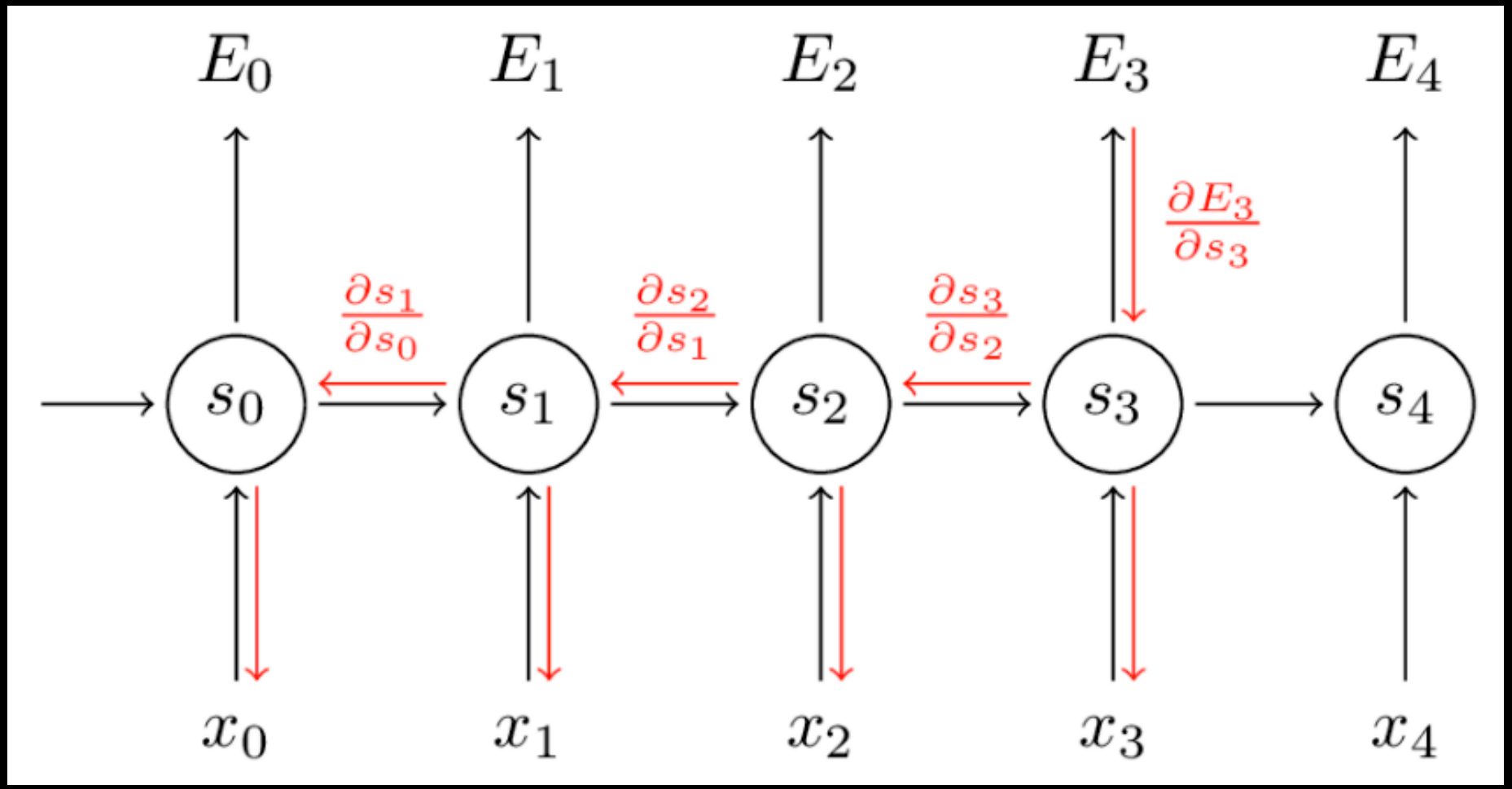

```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

深度RNN

```
lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
cell = tf.contrib.rnn.MultiRNNCell([lstm] * num_layers)
```



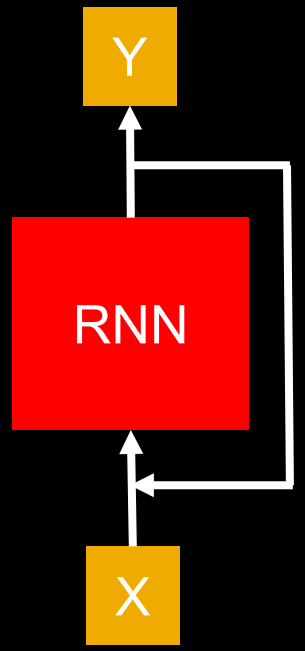
BPTT（截断）


E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

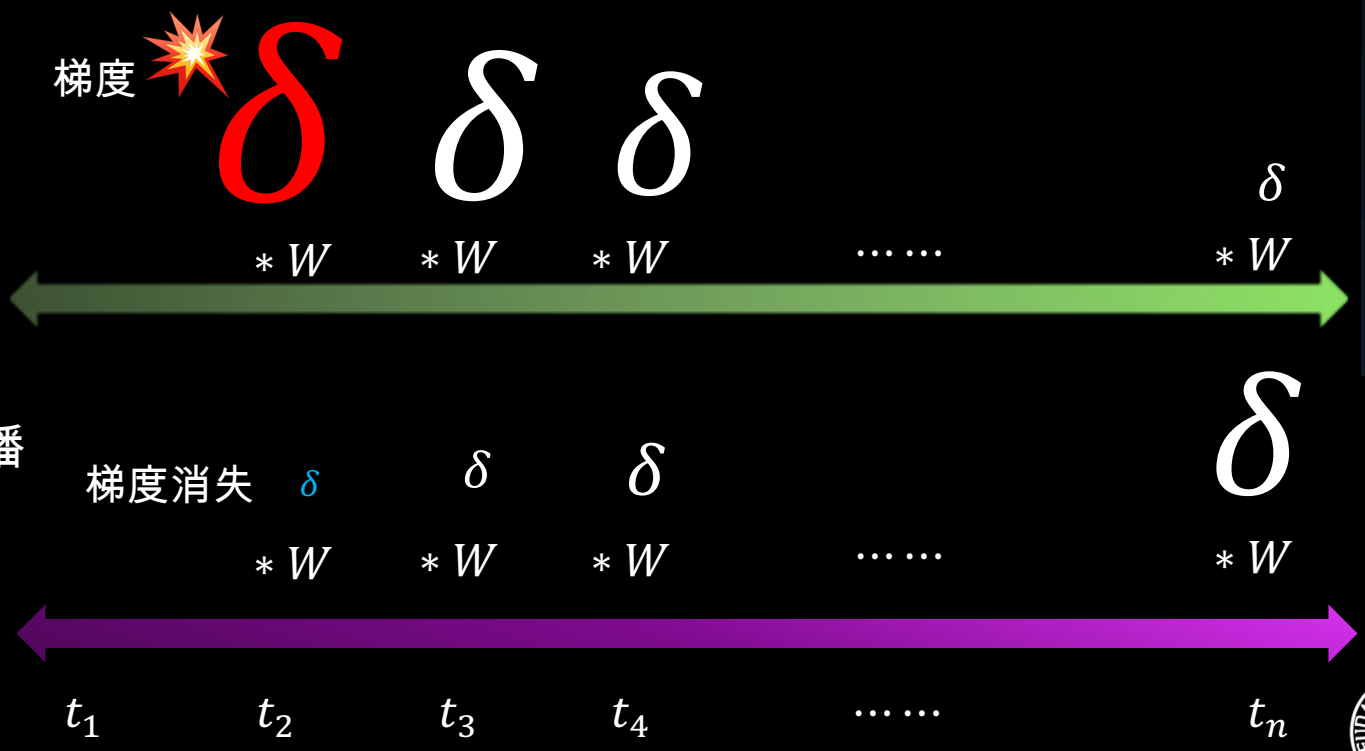
RNN的困难：梯度消失与梯度爆炸

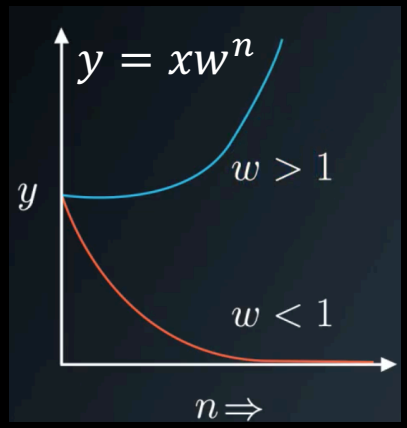
Exploding & Vanishing Gradients

- Sentence 1：“Jane walked into the room. John walked in too. Jane said hi to ___“
- Sentence 2："Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to ___"



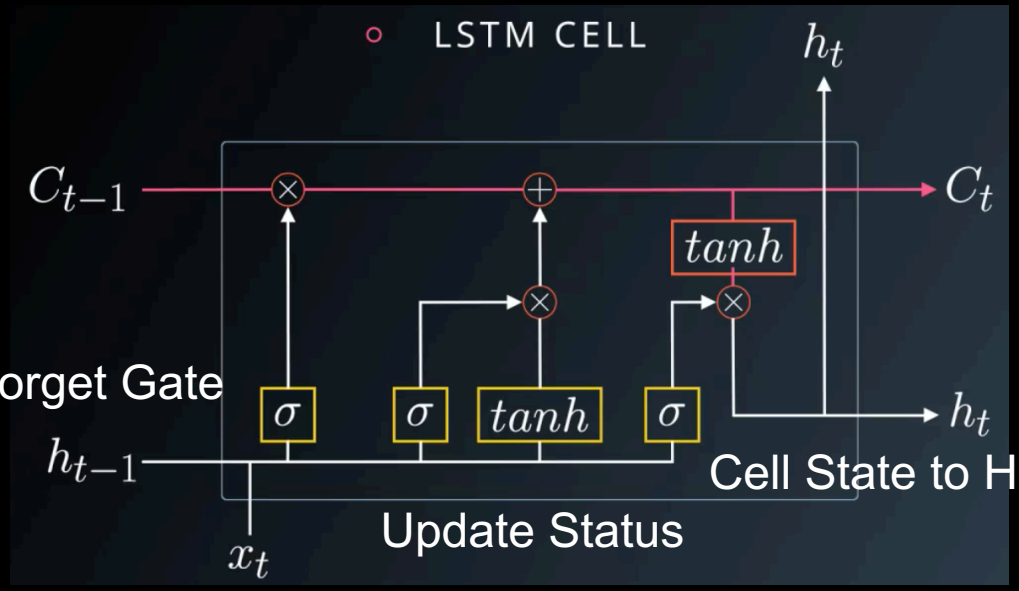
误差传播





解决方法：Long Short Term Memory Units (LSTMs)

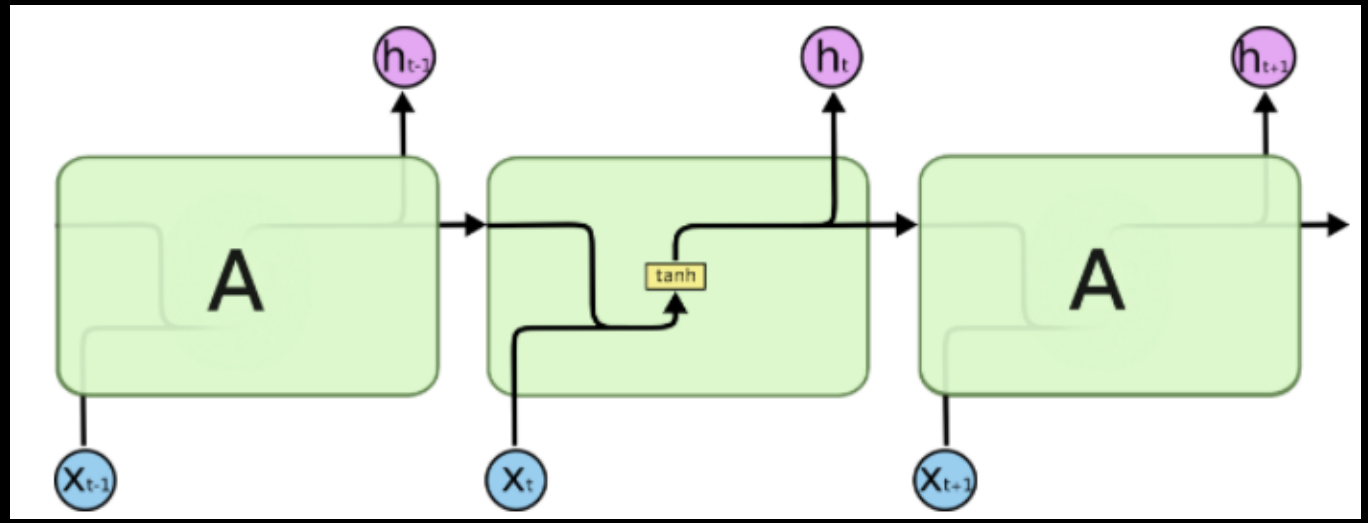
When: 1997
Who: Sepp Hochreiter & Juergen Schmidhuber
How Many Steps: 1000， 打开了远程连接Cause/Effect的可能
How: 将信息存于门单元（Gated Cell）之中，Cell通过门的开关决定存储的内容，允许读写删的时间，不同与计算机中常用的数字门，LSTM使用的是基于Sigmoid的模拟门，范围在0-1之间，可微分，适合BP

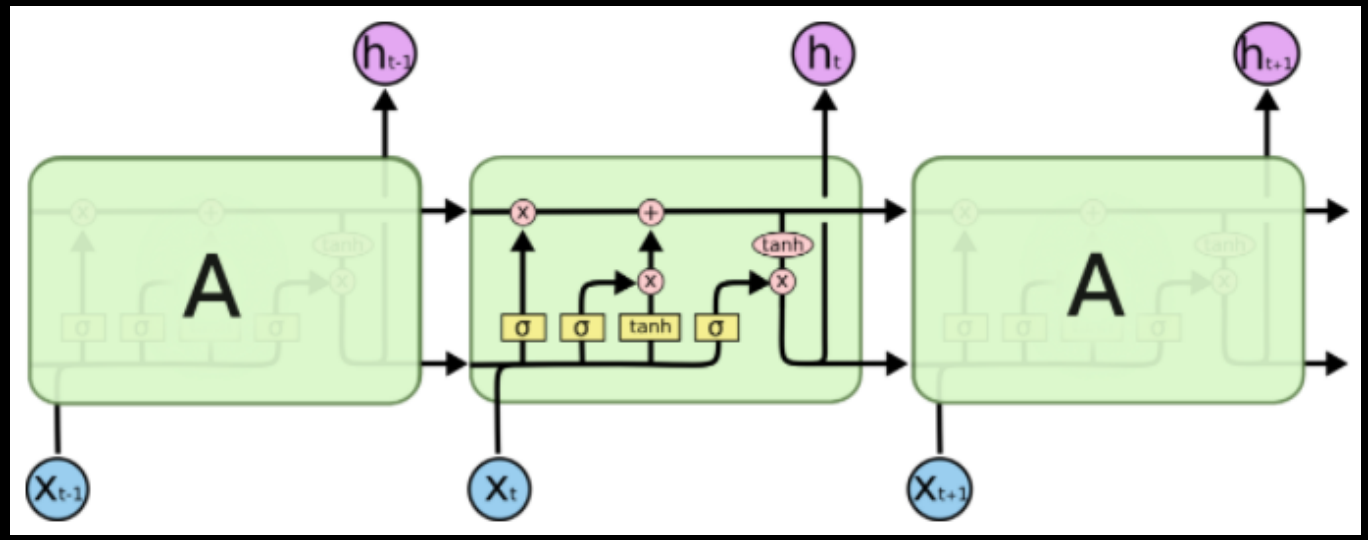


Cell State, 传送带

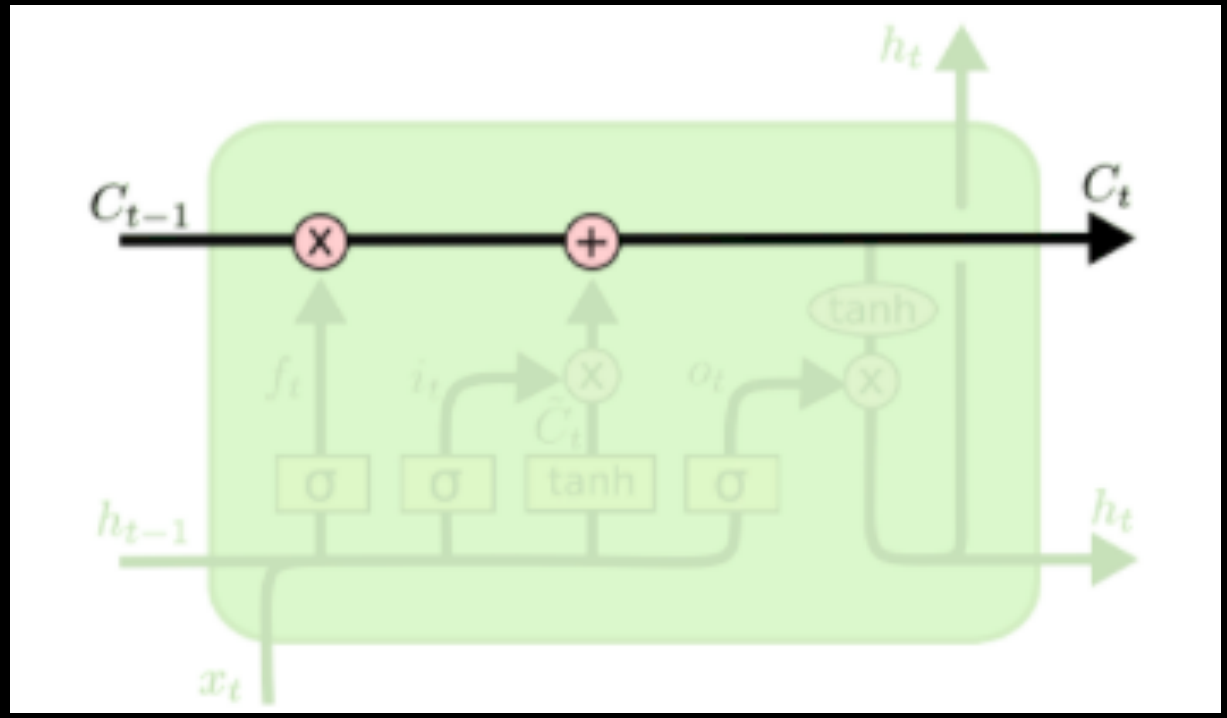
Udacity

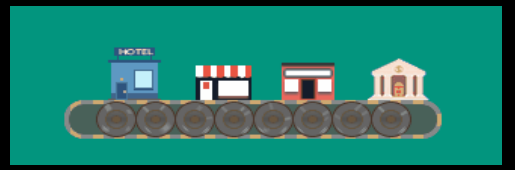
标准RNN v.s. LSTM





LSTM 拆解（1）：Cell Status：传送带



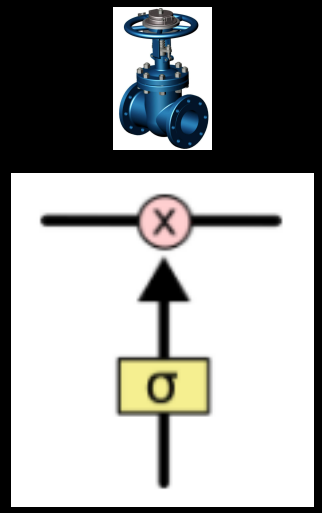


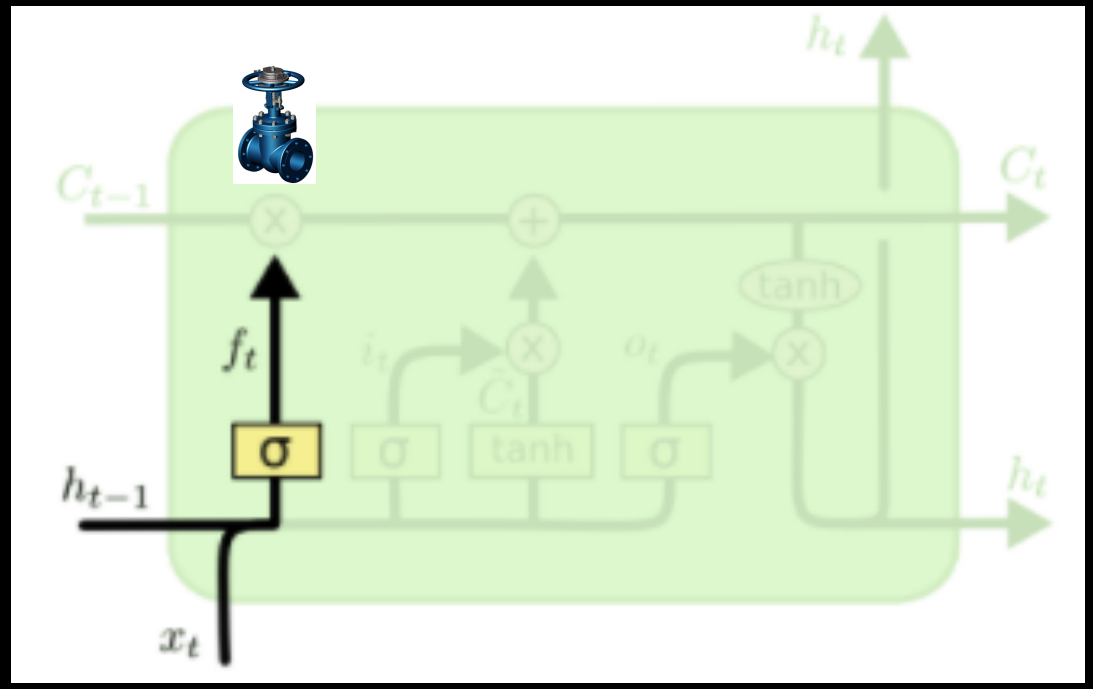
传送带沿途仅少量线性操作，信息可以很容易地不被改变的流过

LSTM 拆解(2)：Forget Gate：遗忘门

pointwise multiplication operation

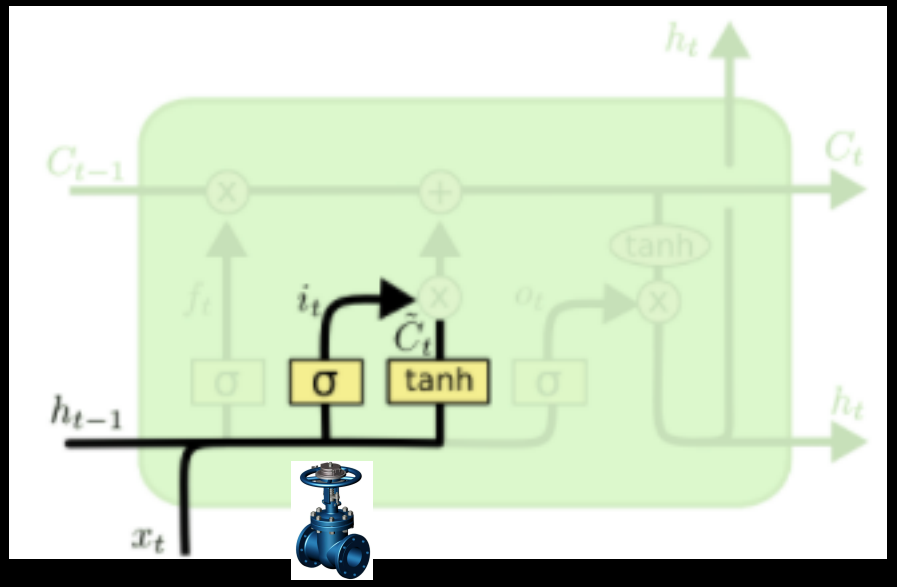
Sigmoid层输出0到1之间的数

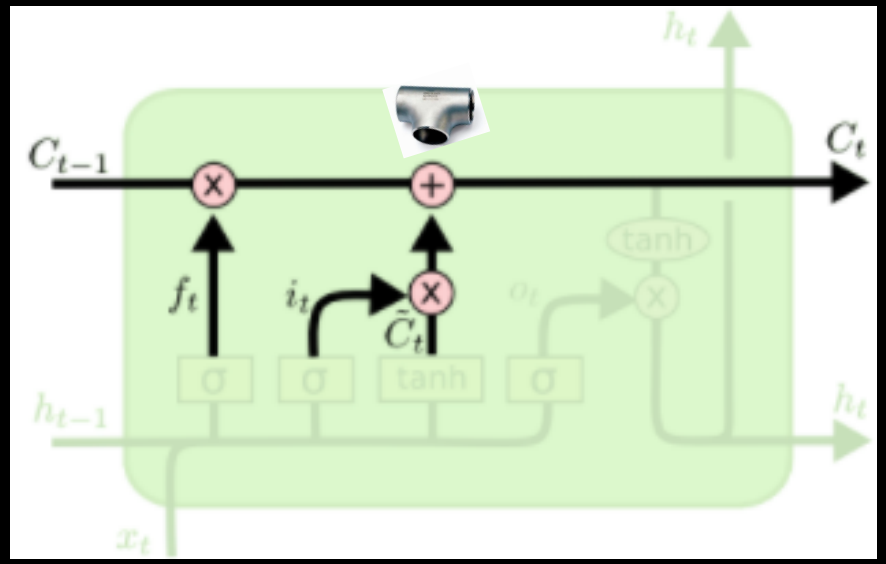




遗忘门　　$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f)$

LSTM 拆解(3)：阀门2决定如何更新Cell的状态



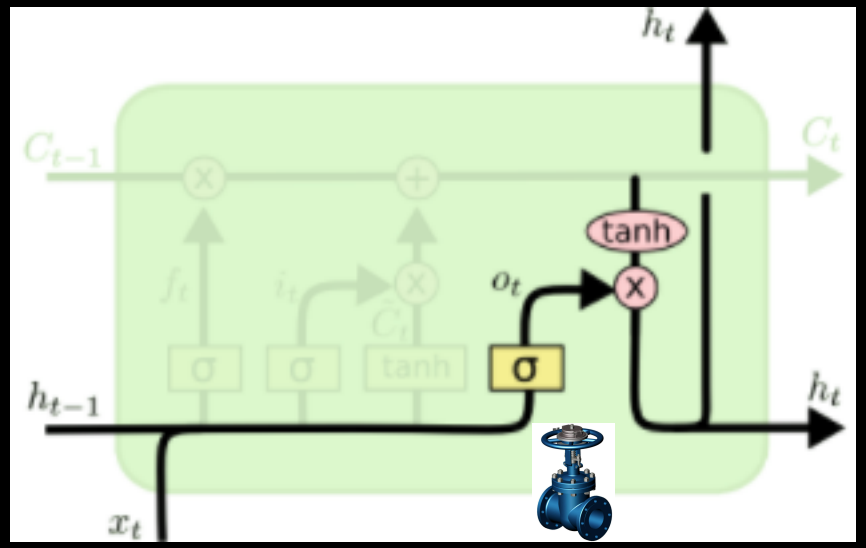


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

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

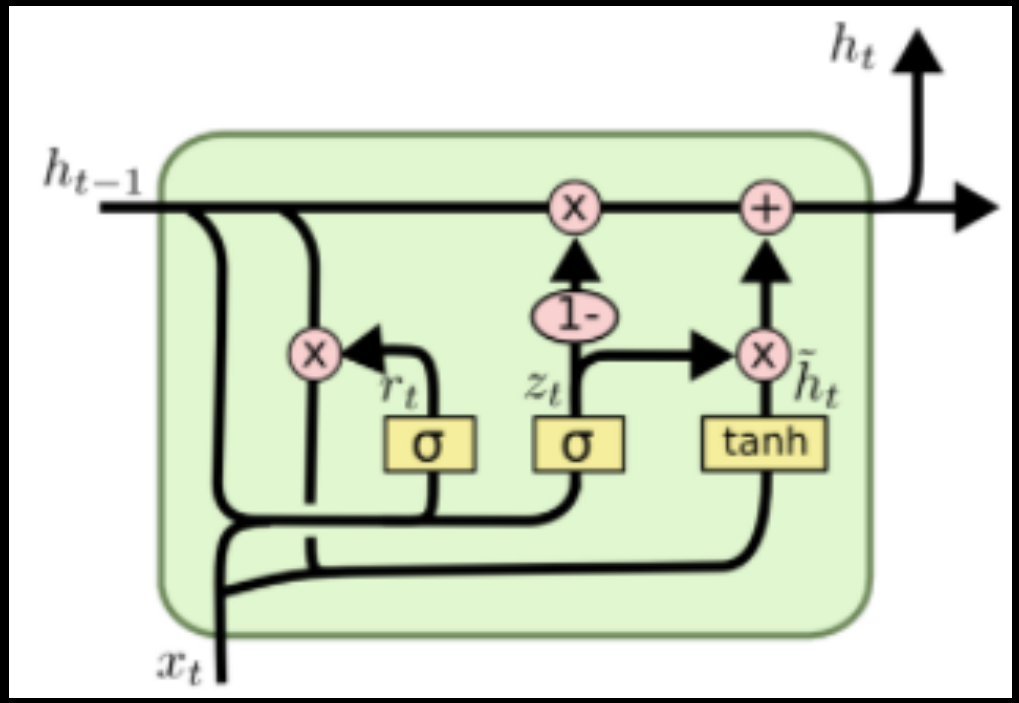
LSTM 拆解（4）：阀门3决定如何输出



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

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

Gated Recurrent Unit (GRU，2014)



- 将Forget Gate和Input Gate合并为Update Gate
- 同样还混合了Cell Status和Hidden Status
- 因为比LSTM简单，近年来比较流行

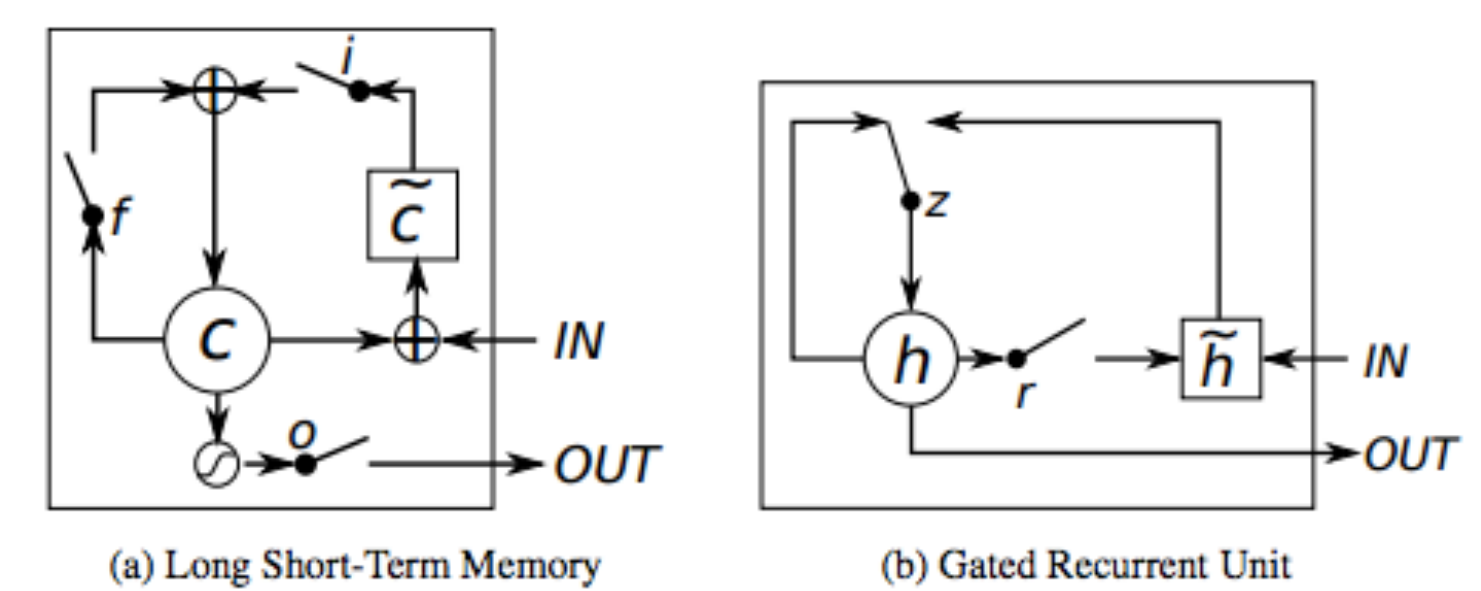
$$z_t = \sigma(w_z \cdot [h_{t-1}, x_t])$$

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

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

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

LSTM v.s. GRU



(a) Long Short-Term Memory

(b) Gated Recurrent Unit

词的离散表达（one-hot）的不足

基于规则和统计的NLP将词看成是独立的符号，例如：

- 旅馆
- 会议
- 汽车
- ……

词由one-hot vector 来表达，例如[0，0，0，0，1，…，0]，向量的维度是有词汇量决定，常用的口语词汇大约2万，比较大的词汇表达到50万

Hotel [0，0，0，0，1，…，0]

Motel [0，0，1，0，0，…，0]

Hotel · Motel = 0

现代统计NLP中最成功的想法

“You shall know a word by the company it keeps”

J. R. Firth 1957

通过利用一个词周围的词来获得表达这个词能力。

This cat hunts mice

kitty

如何使用周围的词来表达词？

Co-occurrence 矩阵！

示例语料库：

I Like deep learning.

I like NLP.

I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Co-occurrence 矩阵的缺陷与对策

缺陷

- 随着词汇量的增长而增长
- 超高维度需要占据大量的内存
- 分类模型存在稀疏性的问题

对策

- 将最重要的信息存储于固定长度的较低维度的稠密向量
- 通常25-1000维

复习：特征向量与矩阵的对角化

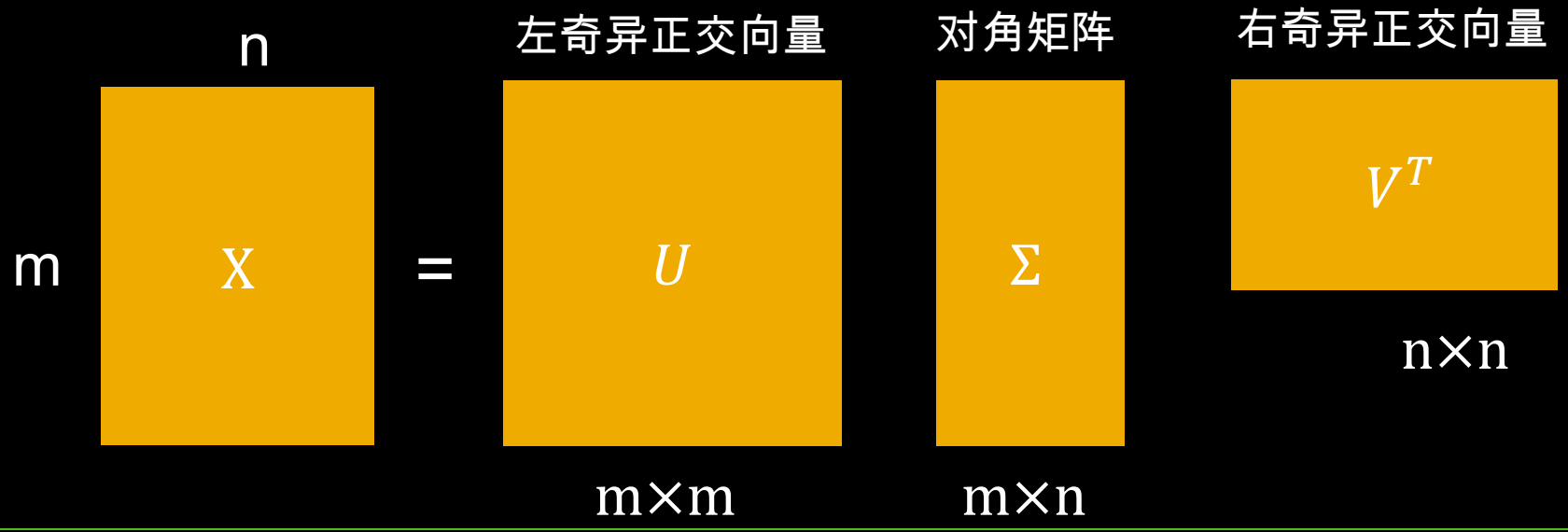
特征向量与特征值：$Ax = \lambda x$

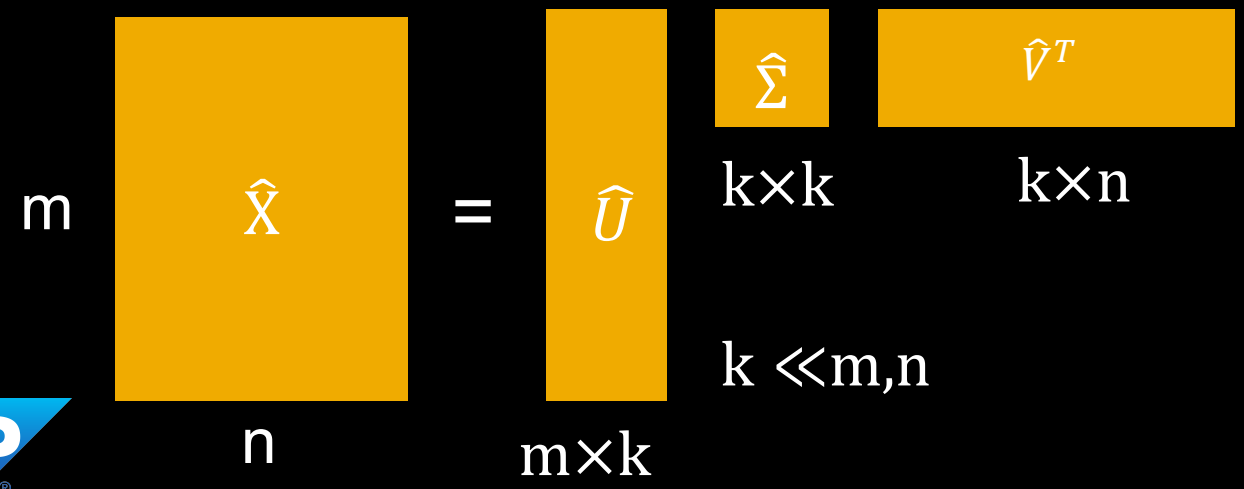
假设矩阵A存在n个独立的特征向量$x_1 x_2 \cdots x_n$ ，并将这些特征向量按列放到矩阵S中：

$$AS=A[x_1 x_2 \cdots x_n]= A[x_1 x_2 \cdots x_n]=[\lambda_1 x_1 \cdots \lambda_n x_n]= [x_1 \cdots x_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} = S\Lambda$$

$$AS = S\Lambda \qquad A = S\Lambda S^{-1}$$

方法1:SVD矩阵降维





$\hat{X}$是X最佳的Rank K逼近

SVD矩阵降维

语料库：

I Like deep learning.

I like NLP.

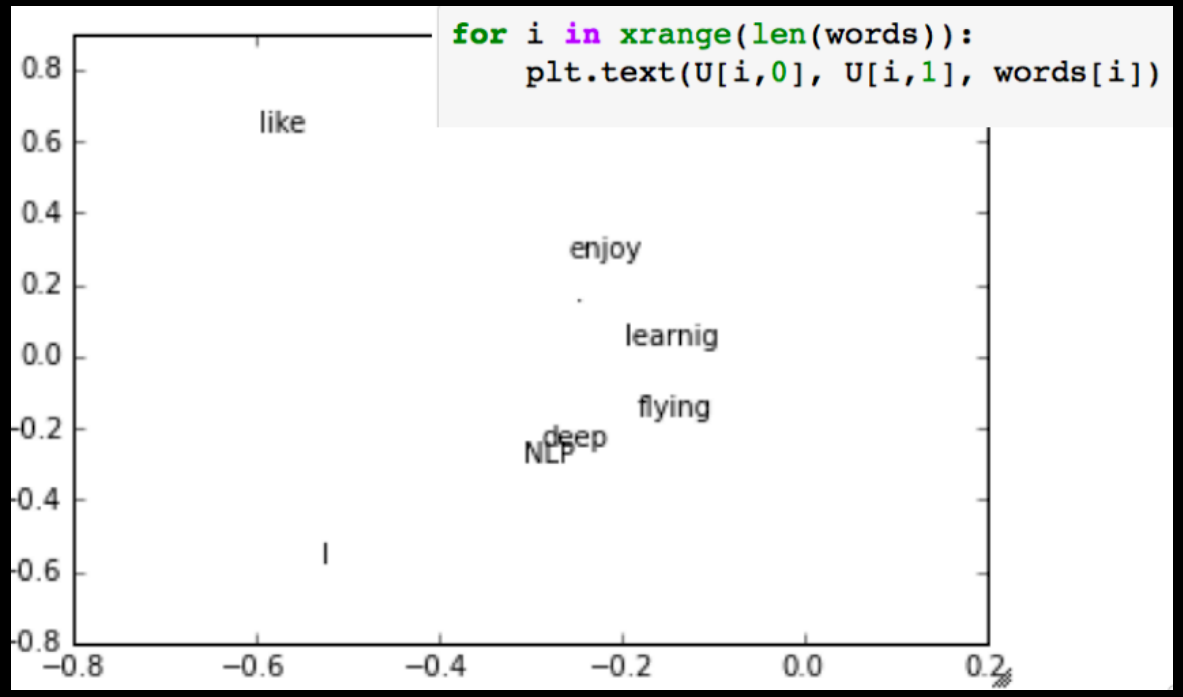
I enjoy flying.

```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

CS224d

SVD矩阵降维

取U的前2列做图（对应最大的2个Singular值）

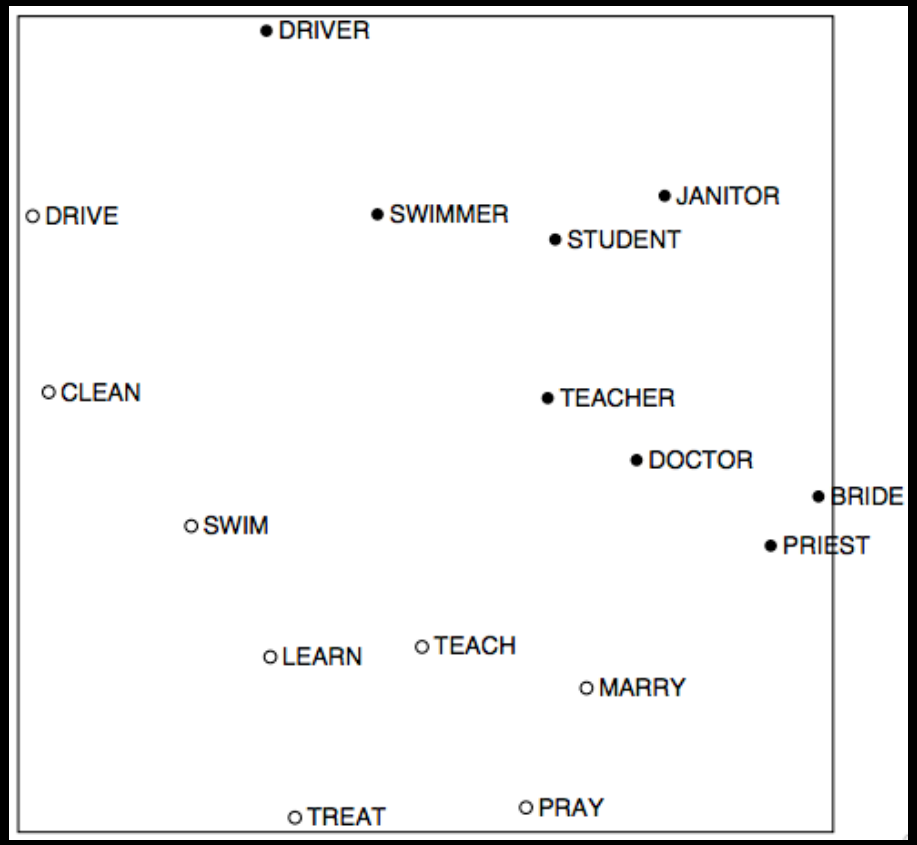


CS224d

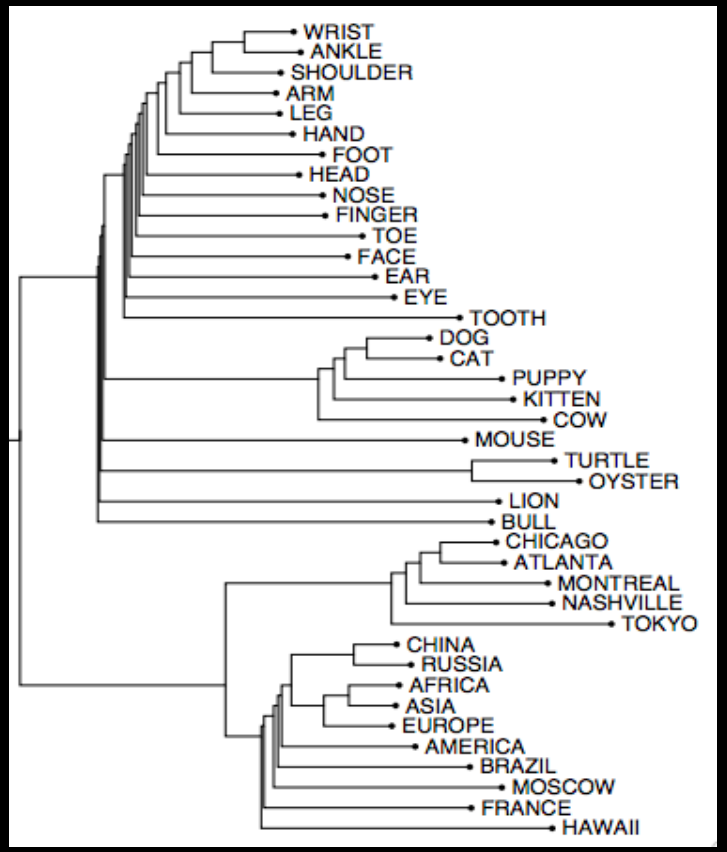
由此可见，Word可以以Dense Vector来表达：

$$\text{Enjoy} = \begin{bmatrix} 0.235 \\ -0.177 \\ -0.342 \\ 0.341 \\ 0.543 \\ -0.271 \end{bmatrix}$$

SVD矩阵降维



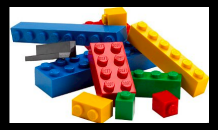
Rohde et at. 2005

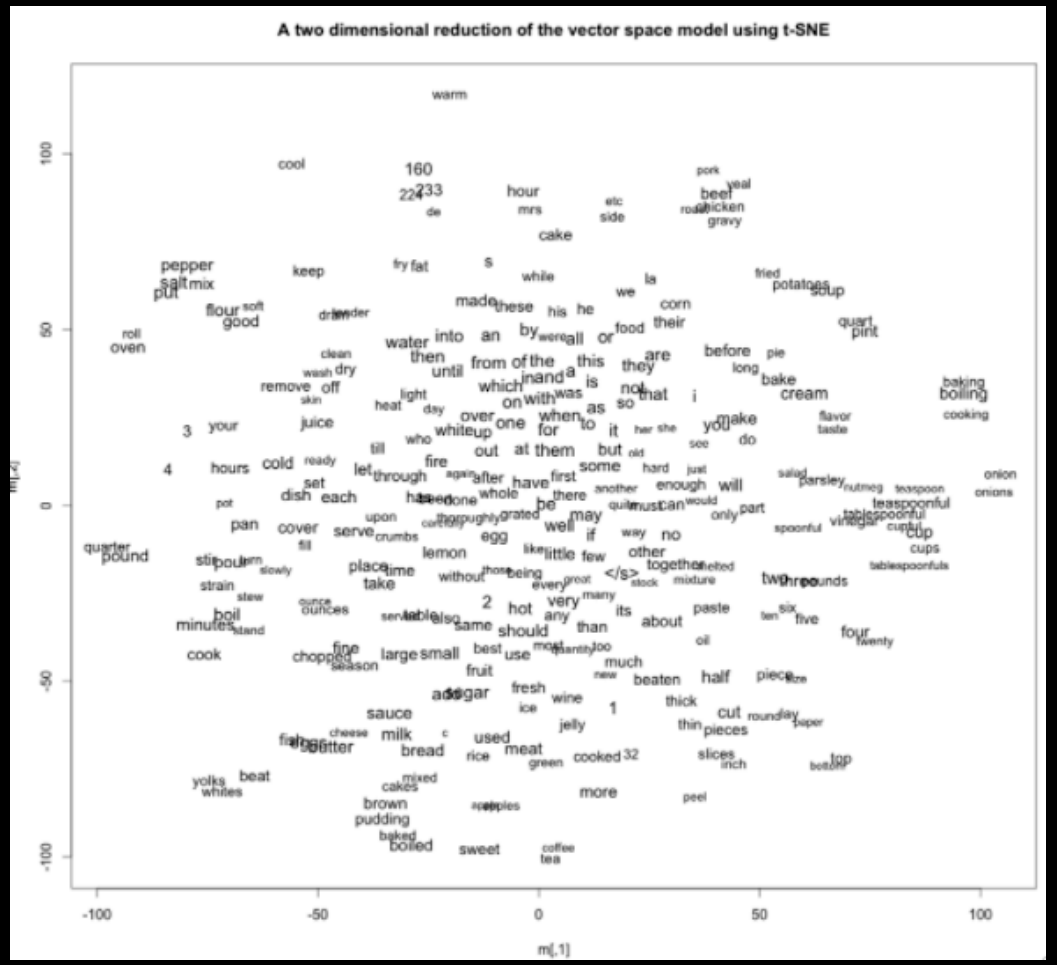


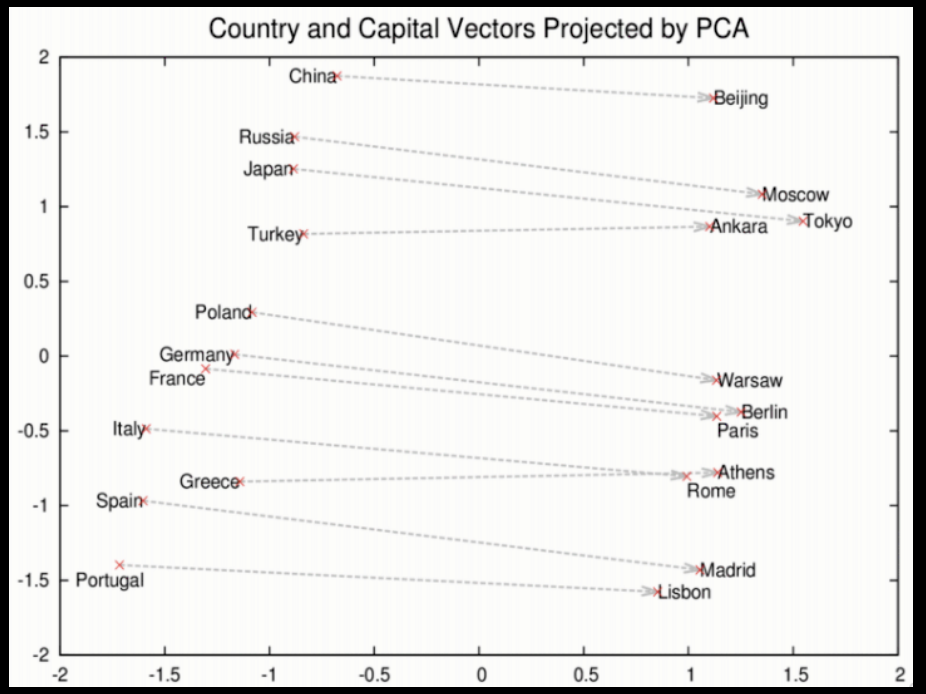
SVD方法的缺点：

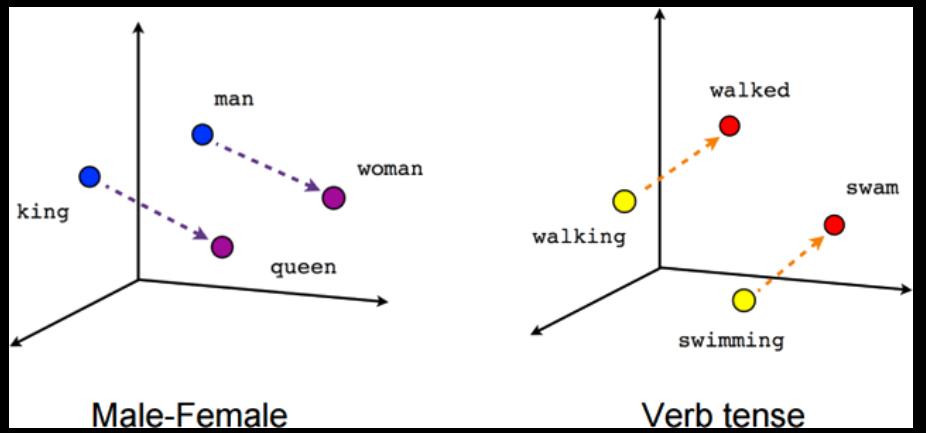
- 对于上百万字的语料库而言，计算成本高
- 难以加入新的Word
- 与机器学习的其他算法的Learning Regime不同

Word2Vec



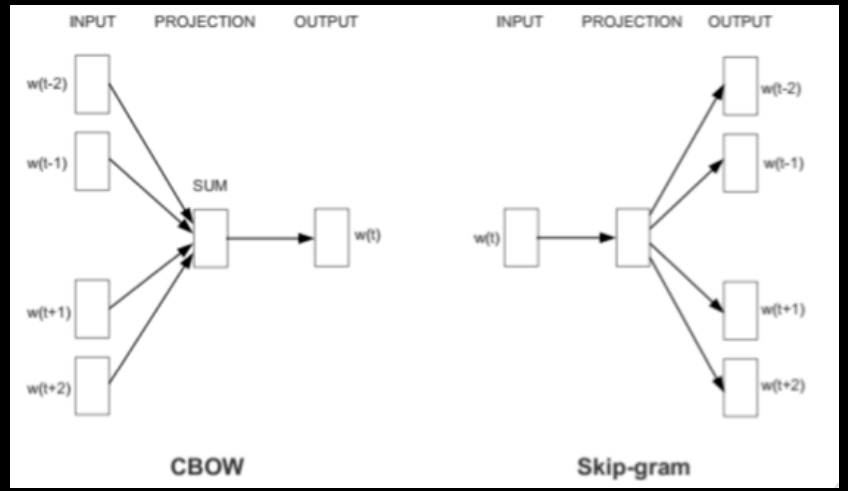

A two dimensional reduction of the vector space model using t-SNE
m[,1]
m[,2]


Country and Capital Vectors Projected by PCA
China
Beijing
Russia
Japan
Moscow
Ankara
Tokyo
Turkey
Poland
Germany
France
Warsaw
Berlin
Paris
Italy
Athens
Rome
Greece
Spain
Madrid
Portugal
Lisbon


man
woman
king
queen
Male-Female
walked
swam
walking
swimming
Verb tense

Word2Vec的基本思想

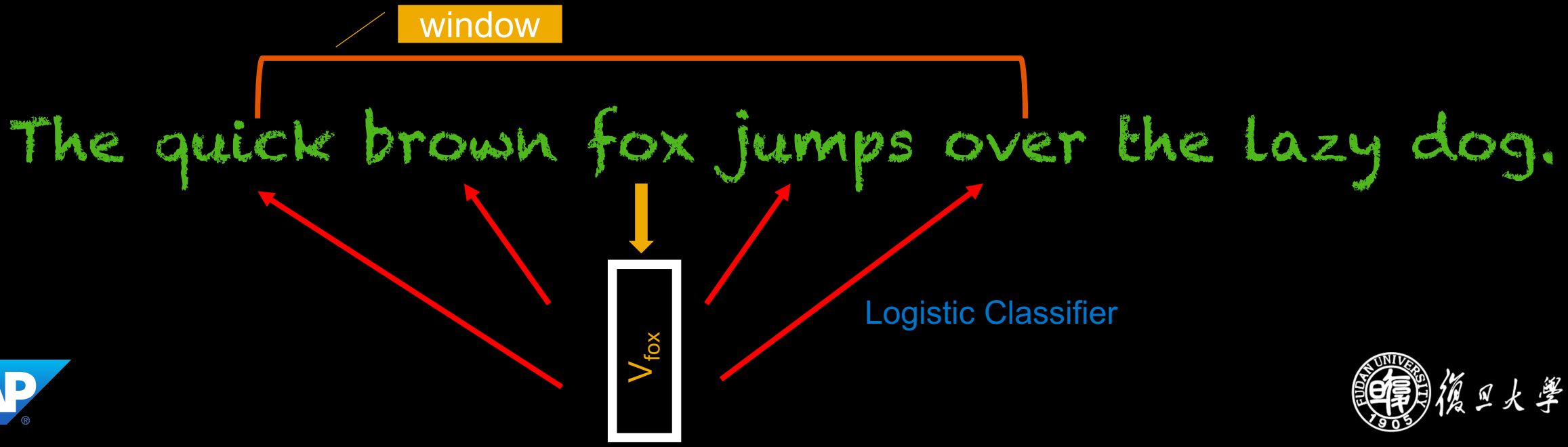
- 不直接捕捉co-occurrence的次数，而是预测每个词周围的词
- 预测每个词周围窗口大小为m周遭的词
- Skip-gram & CBOW
- 训练一个单隐层的神经网络，但却不用它预测，目的仅仅是学习隐层的权重（词向量）
- 类似的Trick：Auto Encoder
- 容易将新词加入词汇表中



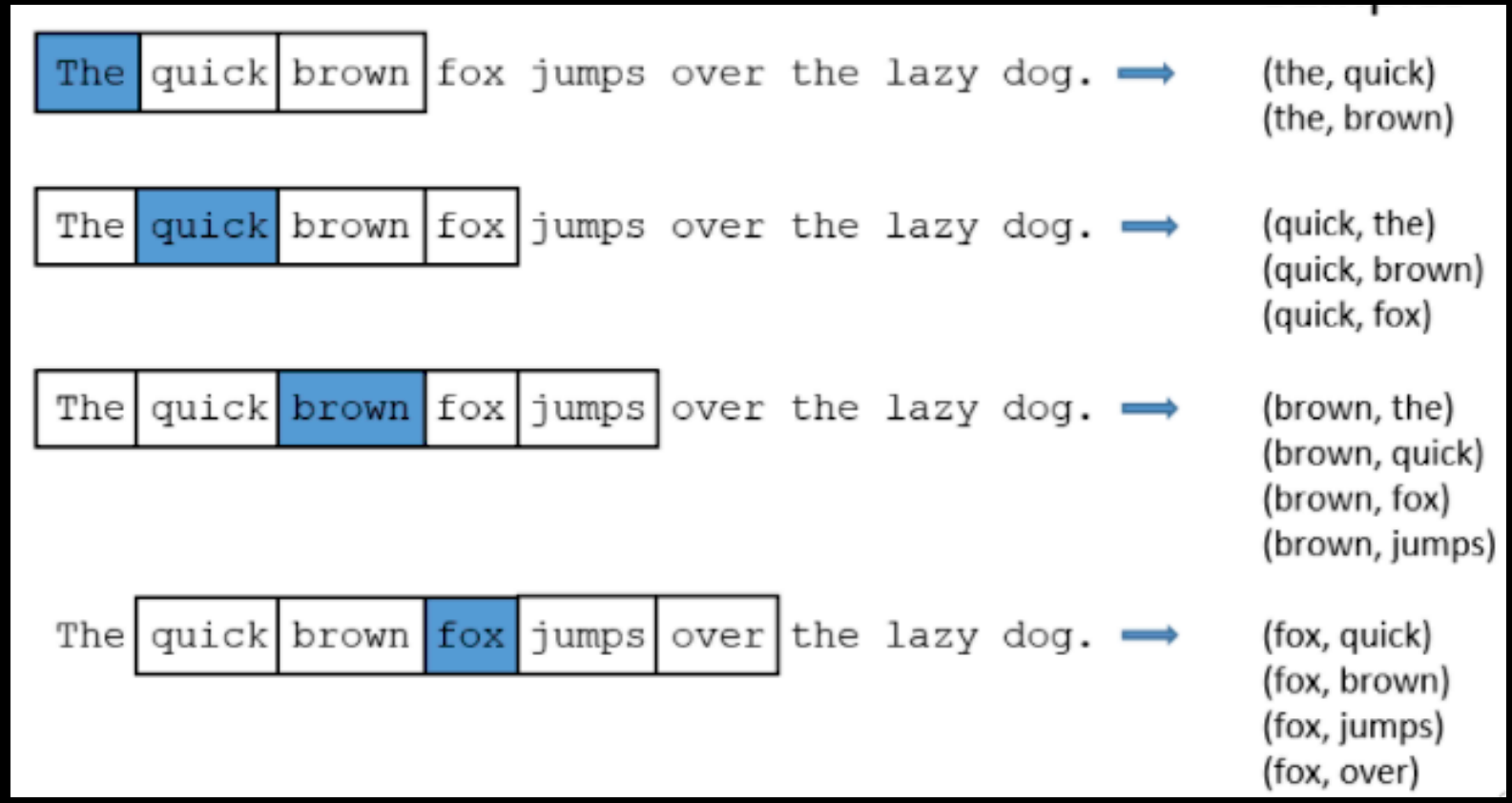
Word2Vec Objective Function

- 目标函数：给定当前词的前提下，最大化上下文词的log Probability:

$$\mathcal{J}(\theta) = \frac{1}{T}\sum_{i=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} logP(w_{t+j}|w_t)$$

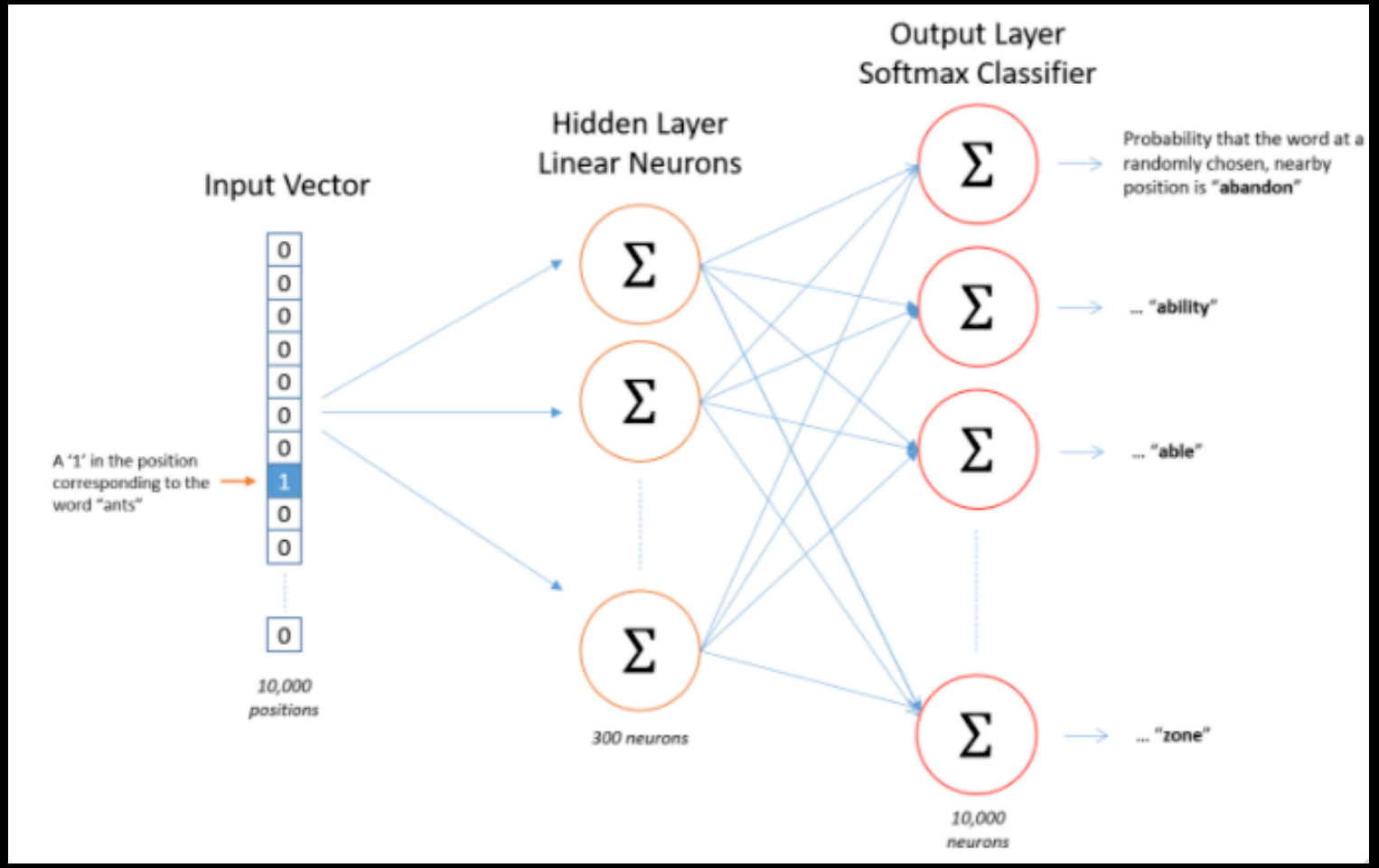


Training Example



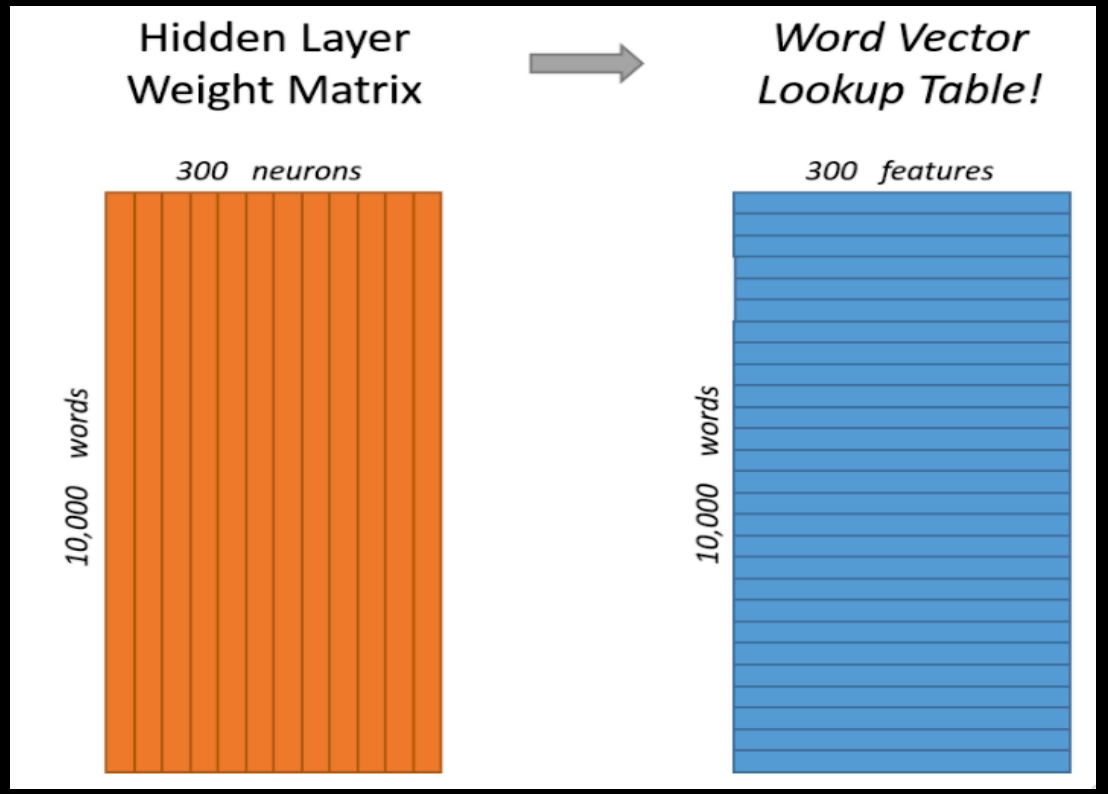
http://mccormickml.com/

Word2Vec Network



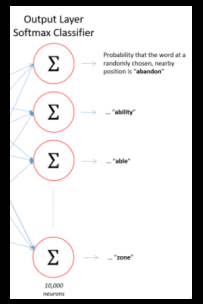
隐层的权重矩阵就是Word Vector

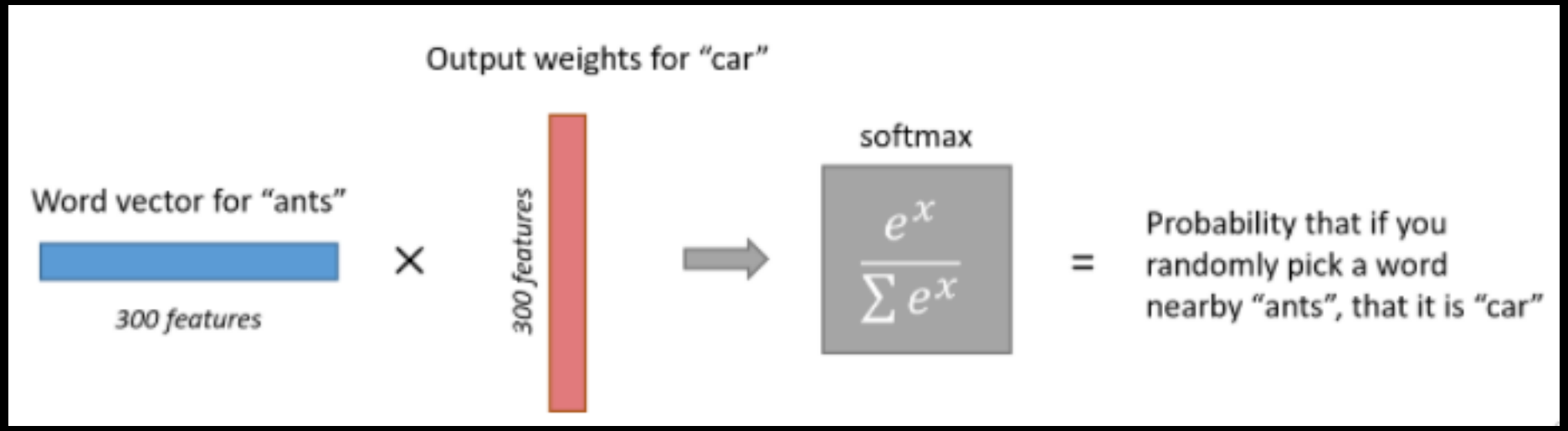
$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$



http://mccormickml.com/

输出层





http://mccormickml.com/

Insights

如果两个词具有非常类似的上下文（在该词周围经常出现的词），模型可能会输出很相似的结果;

例1: 含义接近的词：“Intelligent”和“Smart”

例2: 含义相关的词：“Engine ”和“transmission”

例3: 含义相反的词：“好”和“坏”

例4: 词根: “car”和“cars”

Word2Vec 庞大的权重矩阵训练技巧

在上面的例子中，存在2个庞大的权重参数矩阵（10K*300=3M），在如此之大的矩阵上进行梯度下降的速度很慢，还需要庞大的数据集，避免过拟合。

Word2Vec的作者们提出了解决方法：

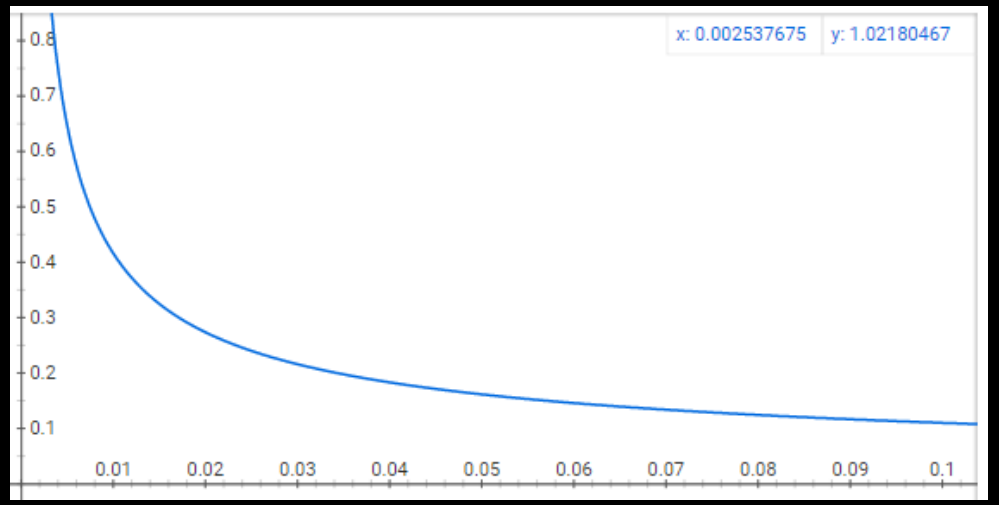
- 将常见的词或词组视为单个词
- 对常见的词进行下采样(Subsampling)，以减少训练样本的数量
- 采用“负采样”（Negative Sampling）的技术，调整优化目标，使得每个训练样本仅更新一小部分权重

Word2Vec 训练技巧

- 将常见的词或词组视为单个词（一份报纸： Boston Globe ）
- 对常见的词进行下采样(Subsampling)：
  - 例如： The quick brown fox jumps over the lazy dog.
    - “The”并没有告诉我们太多关于“Fox”的含义，因为出现在太多词的上下文中了
    - 我们有太多“The”的样本，超过了我们需要用来学习“The”的向量
  - 解决方法：假设窗口=10，删除某个“The”，“The”将不会出现的任何其他词的上下文中，同时，当“the”作为输入词时，也少了10个训练样本。

Word2Vec 训练技巧:Subsampling

- Subsampling Rate
  - $z(w_i)$是词w_i在整个语料库中的占比，例如peanut在10亿的词的语料库中出现了1000词，$z(\text{'peanuts'}) = 10^{-6}$
  - 保持的概率：$p(w_i) = (\sqrt{\frac{z(w_i)}{0.001}}) \cdot \frac{0.001}{z(w_i)}$



- $p(w_i)$ =1.0 when $z(w_i)$<=0.0026
- $p(w_i)$ =0.5 when $z(w_i)$=0.0075

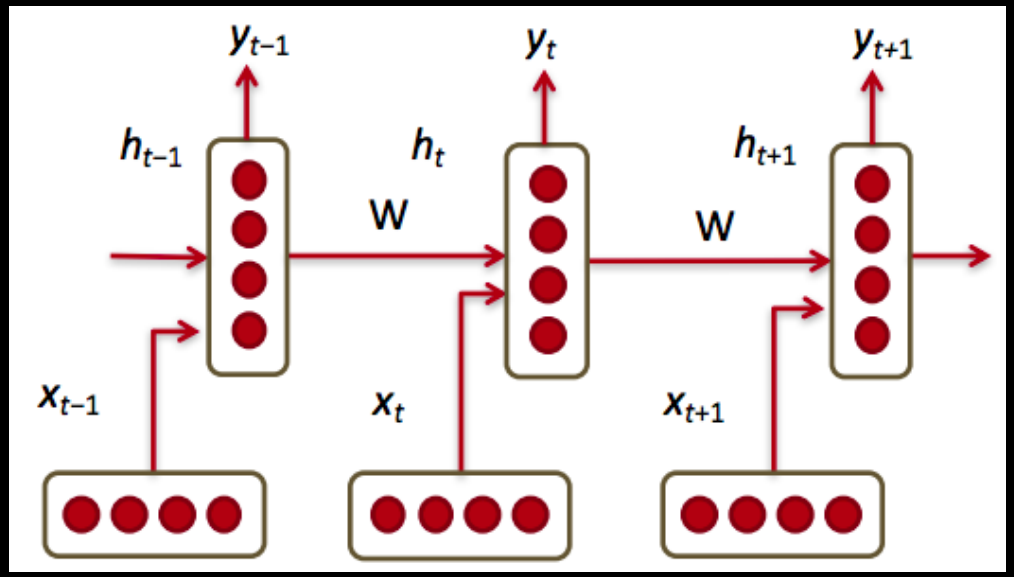
Word2Vec 训练技巧:Negative Sampling

- 训练神经网络的过程就是用一个样本去微调所有权重的过程。
- 由于词汇表很大，导致权重非常多，而每个权重都要被数十亿个训练样本微调！
- Negative Sampling通过每个训练样本仅调整一小部分权重来解决这个问题
- 例如：训练word pair(“fox", "quick”)时，输出的标签是one-hot vector，只有“quick”是1，其他几千个词都是0
- Negative Sampling从这几千个（Negative=0）词中只选择一小部分词（例如5个），进行权重更新
- 原先输出层10000个权重变成之需要更新6个（5个Negative+1个Positive），工作量仅0.06%
- 选择Negative Samples时，采用Unigram Distribution：$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}}$

  Unigram $Model$: $P(w_1, w_2, w_3, \ldots, w_n) \approx \prod_i P(w_i)$

RNN

- 核心思想：RNN的所有的(时间)步都使用相同的权重，这样可以将该模型延伸到任意时间步骤
- 后面的输出依赖之前所有的词
- 内存需求与词的数量呈线性关系



Richard Socher

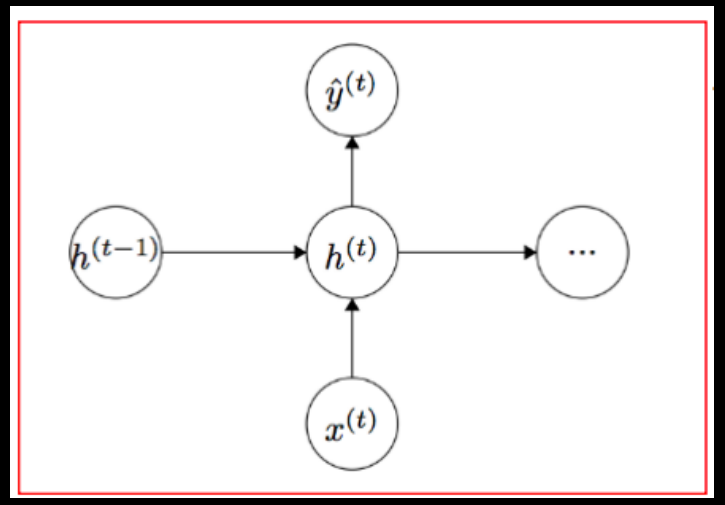
基于RNN的语言模型

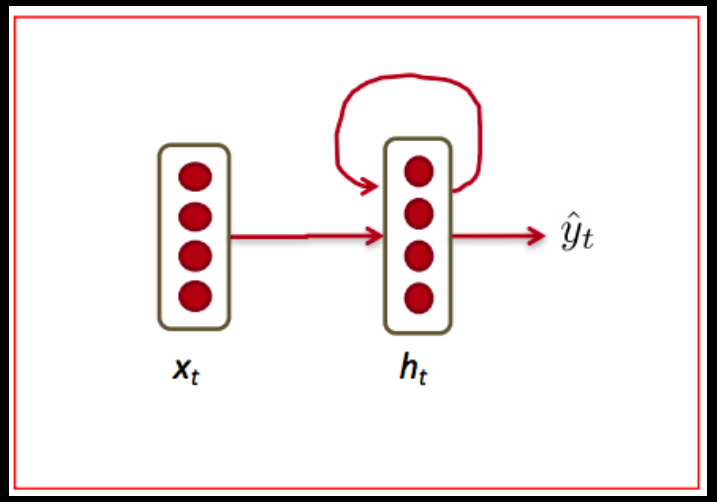
- 给定词向量List：$x_1, x_2, \ldots, x_t, \ldots, x_n$
- 每一步：$h_t = \sigma\left(W^{(hh)}h_{t-1}\right) + W^{(hx)}x_{[t]}$

$$\hat{y}_t = softmax\left(W^{(S)}h_t\right)$$

$$\hat{P}(x_{t+1} = v_j | x_1, x_2, \ldots, x_t) = \hat{y}_{t,j}$$

虽然在语言模型中，RNN被用来预测下一个词，但也可以被用于情绪分析，或词的其他标签的分类





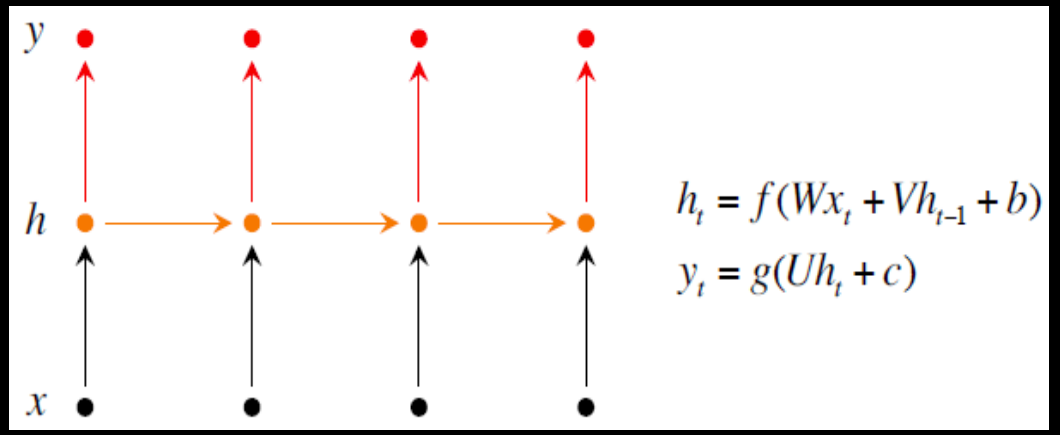
Cost Function:

$$J^{(t)} = -\sum_{j=1}^{|V|} y_{t,j} log\hat{y}_{t,j}$$

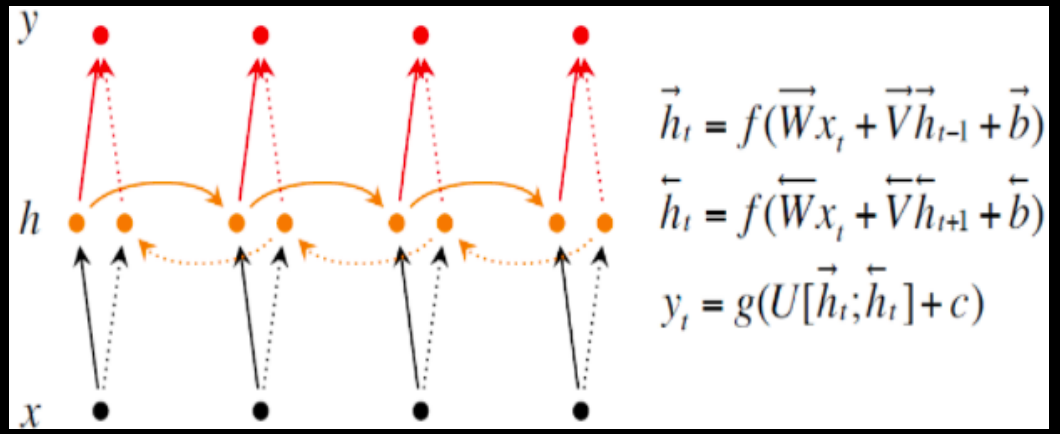
绝大部分是0

Richard Socher

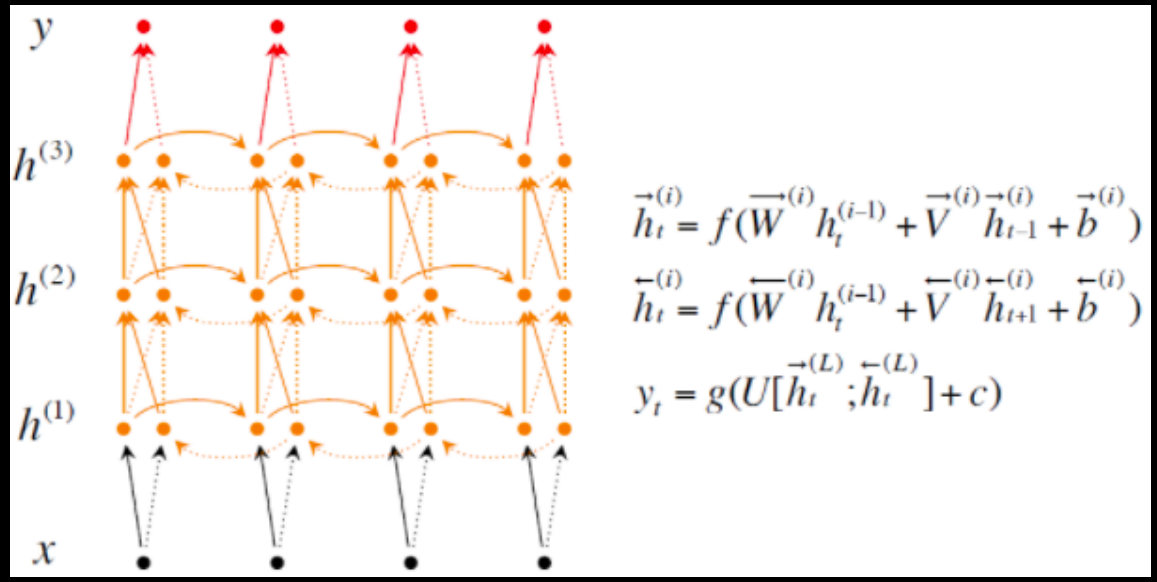
RNN,双向RNN,深度双向RNN



$$h_t = f(Wx_t + Vh_{t-1} + b)$$
$$y_t = g(Uh_t + c)$$



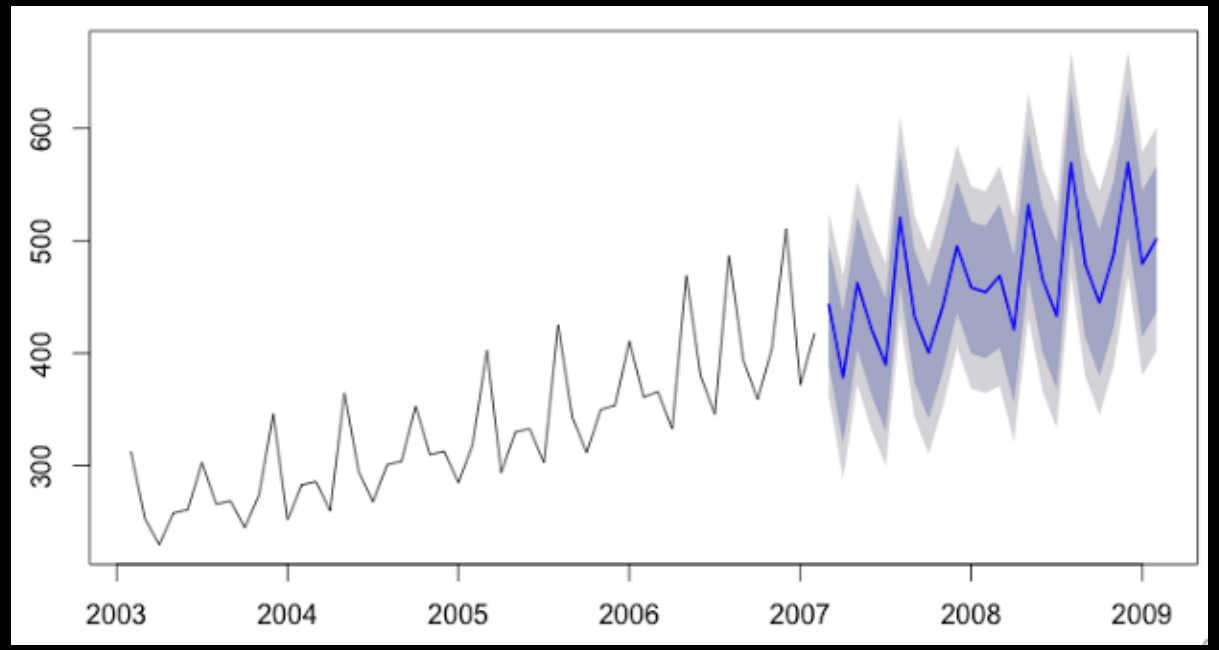
$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$
$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$
$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$
$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

时间序列：Box-Jenkins Forecasting (ARIMA)

1970年代，2位英国统计学家George Box , Gwilym Jenkins提出自回归积分移动平均模型（Autoregressive Integrated Moving Average Model， ARIMA）。



自回归模型（AR）：

当期值等于一个或数个前期值的线性组合，+常数项+白噪声

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

模型参数

移动平均模型（MA）：

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

均值　　参数　　白噪声误差项

Keras 简介

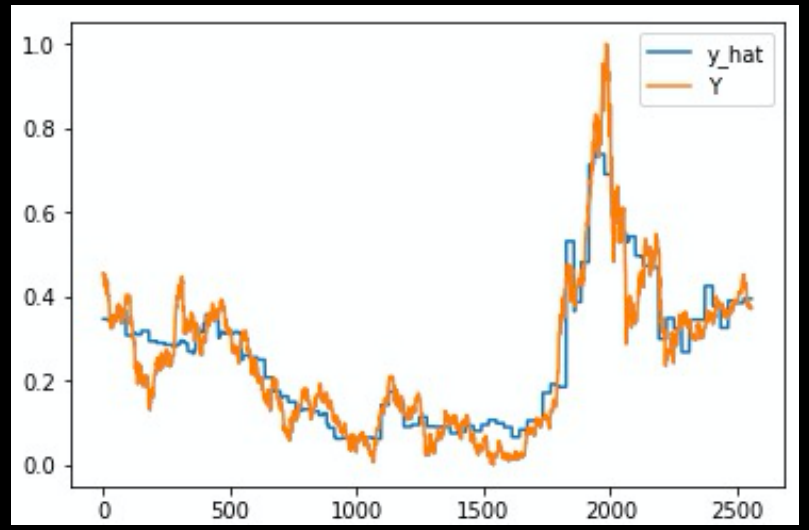
LSTM应用（1）股价预测（此处应有风险提示！☺）

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm, time #helper libraries
```

```
X_train, y_train, X_test, y_test = lstm.load_data('stock.csv', 50, True)

model = Sequential()
model.add(LSTM(
    input_dim=1,
    output_dim=50,
    return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(
    100,
    return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(
    output_dim=1))
model.add(Activation('linear'))
```

LSTM应用（1）股价预测

```
model.compile(loss='mse', optimizer='rmsprop')

model.fit(
    X_train,
    y_train,
    batch_size=512,
    nb_epoch=1,
    validation_split=0.05)

predictions = lstm.predict_sequences_multiple(model, X_test, 50, 50)
lstm.plot_results_multiple(predictions, y_test, 50)
```



600XXX预测与实际走势对比（更复杂的RNN）

LSTM应用（2）自动产生电视脚本

```
def create_lookup_tables(text):

    from collections import Counter
    counts = Counter(text)
    vocab = set(sorted(counts, key=counts.get, reverse=True))
    vocab_to_int = {word:i for i, word in enumerate(vocab)}
    int_to_vocab = {i:word for i, word in enumerate(vocab)}

    return vocab_to_int, int_to_vocab
```

```
def token_lookup():
    keys = ['.',',','"',';','!','?','(',')','--', "\n"]
    values = ['||Period||','||Comma||','||Quoatation_Mark||','||Semicolon||','||Exclamation_mark||','\
    ||Question_mark||','||Left_Parentheses||','||Right_Parentheses||','||Dash||','||Return||']
    return dict(zip(keys, values))
```

```
def get_inputs():
    input = tf.placeholder(tf.int32, [None, None], name = 'input')
    targets = tf.placeholder(tf.int32, [None, None], name = 'targets')
    learning_rate = tf.placeholder(tf.float32, name = 'learning_rate')
    return (input, targets, learning_rate)
```

SAP

復旦大學

LSTM应用（2）自动产生电视脚本

```
def get_init_cell(batch_size, rnn_size):

    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    cell = tf.contrib.rnn.MultiRNNCell([lstm] * 2)
    X = cell.zero_state(batch_size,tf.float32)
    InitialState = tf.identity(X,'initial_state')

    return cell, InitialState
```

```
def get_embed(input_data, vocab_size, embed_dim):

    embedding  = tf.Variable(tf.random_uniform([vocab_size,embed_dim],-1,1))
    embedded = tf.nn.embedding_lookup(embedding, input_data)

    return embedded
```

```
def build_rnn(cell, inputs):

    outputs, state = tf.nn.dynamic_rnn(cell,inputs,dtype=tf.float32)
    final_state = tf.identity(state, name="final_state")
    return outputs, final_state
```

LSTM应用（2）自动产生电视脚本

```python
def build_nn(cell, rnn_size, input_data, vocab_size):

    embedded = get_embed(input_data, vocab_size, rnn_size)
    output, final_state = build_rnn(cell, embedded)
    logits = tf.contrib.layers.fully_connected(output, vocab_size, \
        weights_initializer = tf.truncated_normal_initializer(stddev=0.1),biases_initializer = tf.zeros_initializer(), activation_fn=None)

    return logits, final_state
```

```python
def get_batches(int_text, batch_size, seq_length):
    n_batches = (len(int_text)-1)//(batch_size*seq_length)
    int_text = int_text[:n_batches*batch_size*seq_length+1]
    batch = np.ndarray(shape=(n_batches,2,batch_size,seq_length), dtype=int)

    for b in range(n_batches):
        batch_idx = b*seq_length
        x , y= [], []
        for bb in range(batch_size):
            idx = batch_idx + (bb*n_batches*seq_length)
            batch[b][0][bb] = int_text[idx:idx+seq_length]
            batch[b][1][bb] = int_text[idx+1:idx+seq_length+1]
    return batch
```

```python
# Number of Epochs
num_epochs = 150
# Batch Size
batch_size = 128
# RNN Size
rnn_size = 100
# Sequence Length
seq_length = 10
# Learning Rate
learning_rate = 0.001
# Show stats for every n number of batches
show_every_n_batches = 26
```

SAP

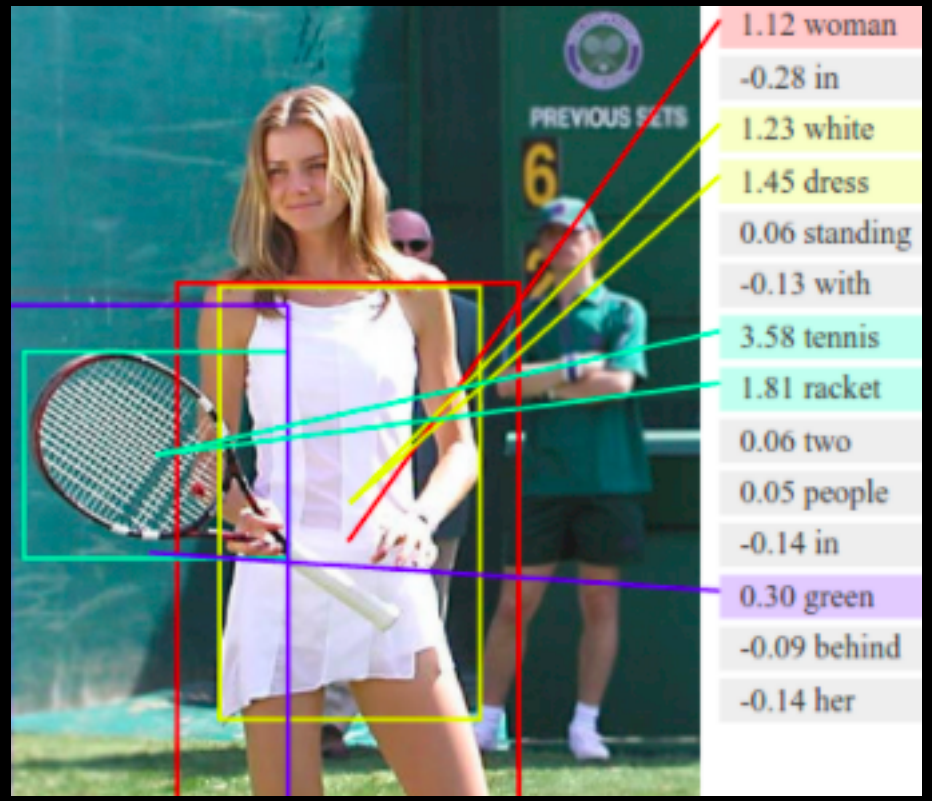
復旦大學

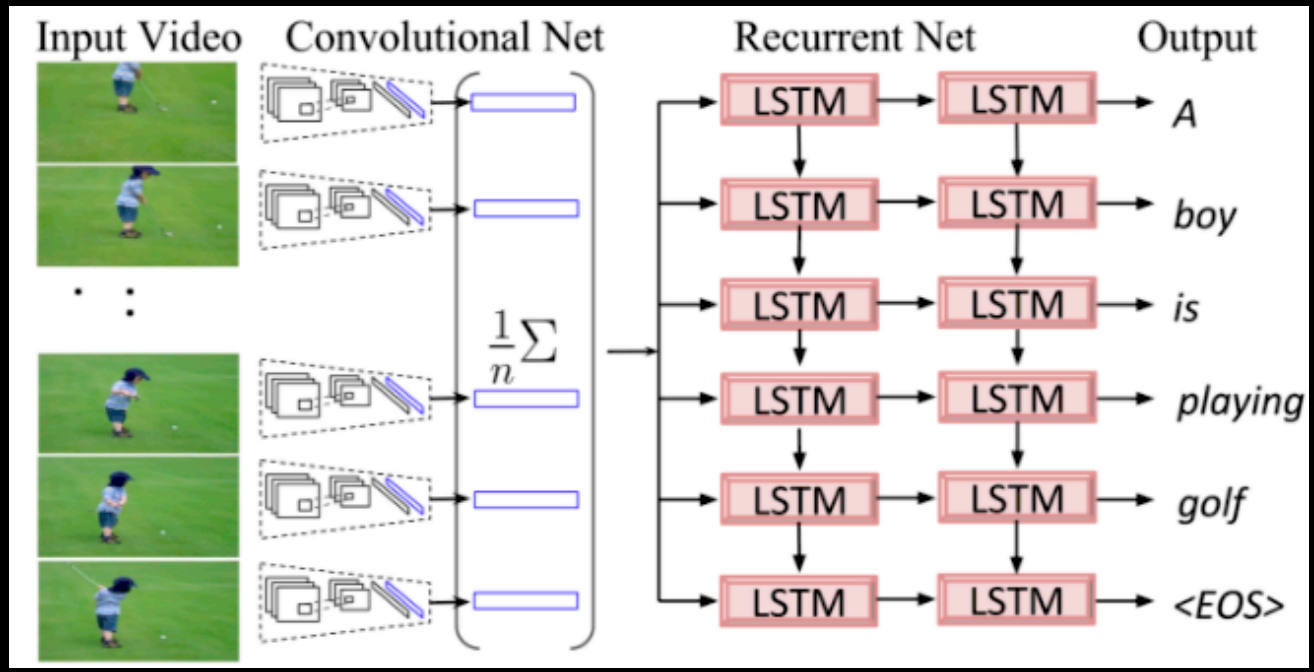
LSTM应用（2）自动产生电视脚本

```python
def pick_word(probabilities, int_to_vocab):
    """
    Pick the next word in the generated text
    :param probabilities: Probabilites of the next word
    :param int_to_vocab: Dictionary of word ids as the keys and words as the values
    :return: String of the predicted word
    """
    return np.random.choice(list(int_to_vocab.values()), 1, p=probabilities)[0]
```

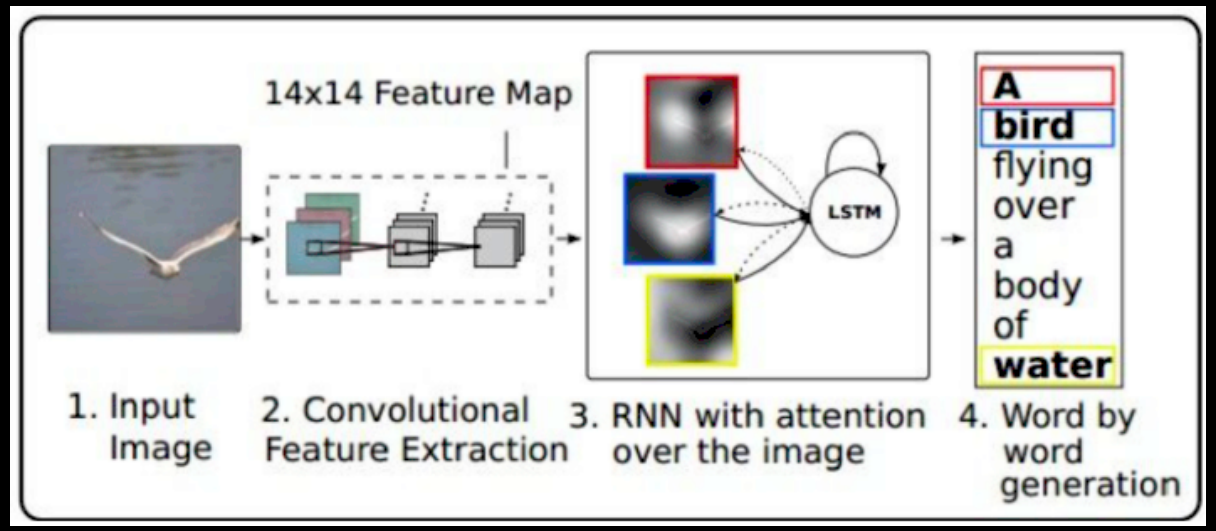
```
moe_szyslak:(under who's) uh.. _montgomery_burns:(sings)
kent_brockman:(sucked barney ladies full)) yes...
moe_szyslak: yeah, i'm not lookin'...(handed strips) eggshell?
moe_szyslak:(" shoo die?
marge_simpson: so that's right. i demand not drop, in the bee stirrers?
moe_szyslak: meyerhof cloudy.
moe_szyslak:(thoughtful) hi, him lisa. what grammar.
homer_simpson: ow!
moe_szyslak: oh, take it as you felt?
moe_szyslak: oh, really?
moe_szyslak: she's be food! mmmmm, yeah. what? i answer i did you got your face, love.
kent_brockman: lenny, you're be into the one at raking i can do.
moe_szyslak:(glowers) she's a matter, limber into a head.


lenny_leonard: this tramp's have would afraid that got as my bright na(dashes in tv)
duffman: that's feeling i'm going...(towed) the prayer i popped down out here.
moe_szyslak:(disapproving) well,
```

LSTM应用（2）图像/视频生成文字

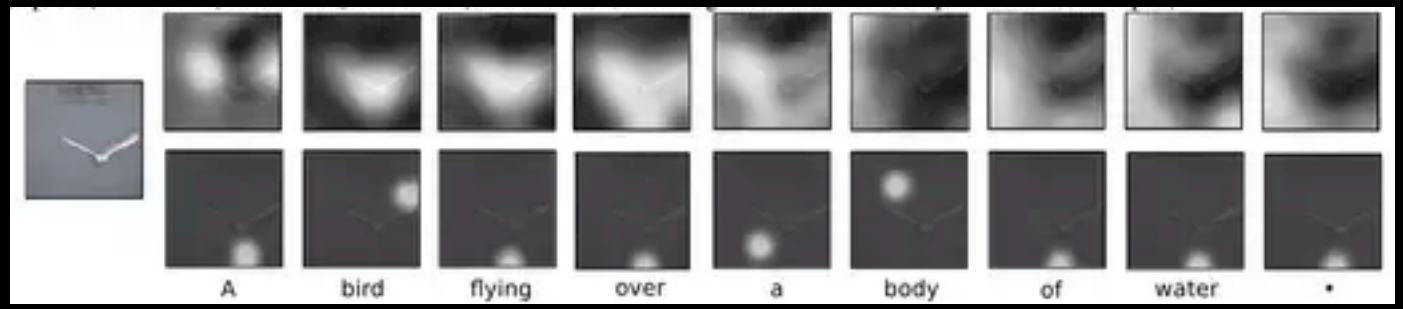




Attention



- 灵感获得于人类视觉的注意力模型
- 存在多种模式，共性是聚焦于图像的某一特定区域
- 对于该区域启用“高分辨率”模式，而对周围区域保持“低分辨率”
- 随时间变化，调整聚焦点



- 当模型生成语句的每个词时，注意力发生变化，并对应到图像相应位置，“软注意”v.s.”硬注意“

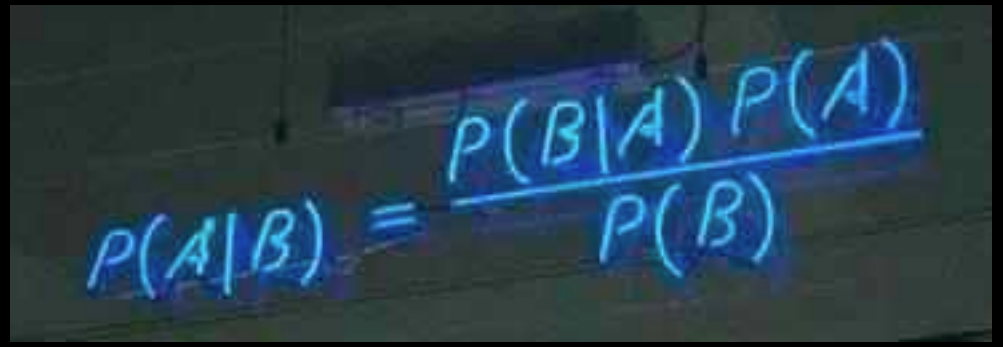
逆概问题

“正概问题”：如“假设袋子里面有N个白球，M个黑球，你伸手进去摸一把，摸出黑球的概率是多大”。

但是现实中要解决的问题往往是相反的：”我们事先并不知道袋子里面黑白球的比例，而是闭着眼睛摸出几个球，通过对这些取出来的球的颜色进行观察，而对袋子里面的黑白球的比例作出的推测”。

Bayesian 定律的通俗解释

- 在运用概率对某一事件进行推断之前，我们往往已经事先掌握了关于这一事件的概率，这个概率可能是主观概率或者相对概率，这种初始的概率可以称为**先验概率**（Prior）。
- 如果在后续的研究中，通过抽样调查样本等消息源又获得了有关该事件的信息，我们就可以根据这些新信息对先验概率进行修正，使先验概率变为**后验概率**（Posterior）。
- 这个修正概率的定理就称为**贝叶斯定理**。



Bayes 思想

将$P(Hyposis\,|Data)$记为$P(H|D)$，在给定数据D的情况下，最大化H的概率。

由模型H自动生成（Generate）数据的概率分布

后验概率　　likelihood　　先验概率

$$P(H|D) = \frac{P(D|H) * P(H)}{P(D)}$$

正则项

Bayes 公式应用的一个例子

某种罕见疾病，发病率为0.1%，某测试结果的误诊率为95%（病患100%被检出，5%的正常人显示假阳性），现某人的测试结果为阳性，其确实患该病的几率为多少？

已知：　先验概率：　　条件概率：

P(Disease) = 0.001　　P(+|Disease) = 1.00

P(Normal) = 0.999　　P(+|Normal) = 0.05

P(Disease|+) = P(+|Disease) * P(Disease) / P(+)

= 1.00 *0.001 / (P(+|Disease) * P(Disease) + P(+|Normal) * P(Normal))

= 0.001 / (1.00* 0.001+ 0.05* 0.999) = 0.0198 (< 2%)

Bayesian Learning

$$p(\mathrm{H}|\mathrm{D}) = \frac{p(D|\mathrm{H}) * p(\mathrm{H})}{p(D)}$$

Learn the BEST Hypothesis Given Data & Domain Knowledge.

Learn the most Probable $h \in \mathcal{H}$ Given Data & Domain Knowledge.

Generative Learning algorithm

机器学习算法的2个分类：

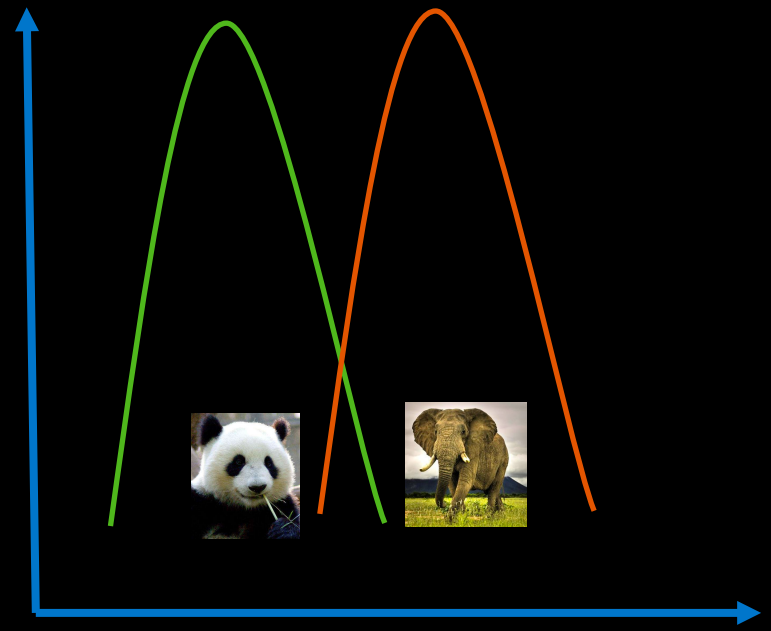
Discriminative: $p(y|x;\theta)$, 总是希望在不同分类间，直接用 Decision Boundary 区分开…

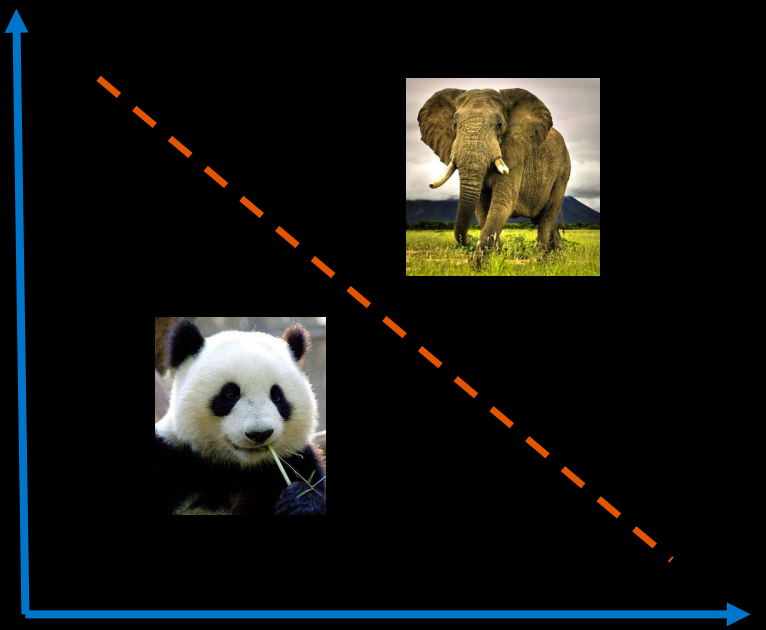
Generative:

$p(x|y)$

$p(y)$: class priors

$$p(y|x) = \frac{p(x|y) * p(y)}{p(x|y)}$$





生成型学习算法

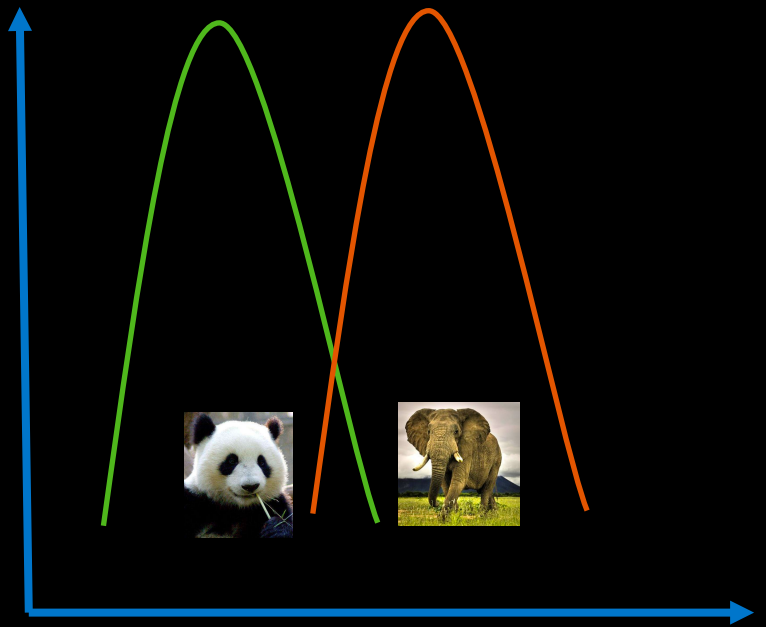
Generative algorithm: For each $y \epsilon Y$

计算：$p(y|x) = \dfrac{p(x|y) * p(y)}{p(x)}$

其中：$p(x) = p(x|y=1) * p(y=1) + p(x|y=0) * p(y=0)$

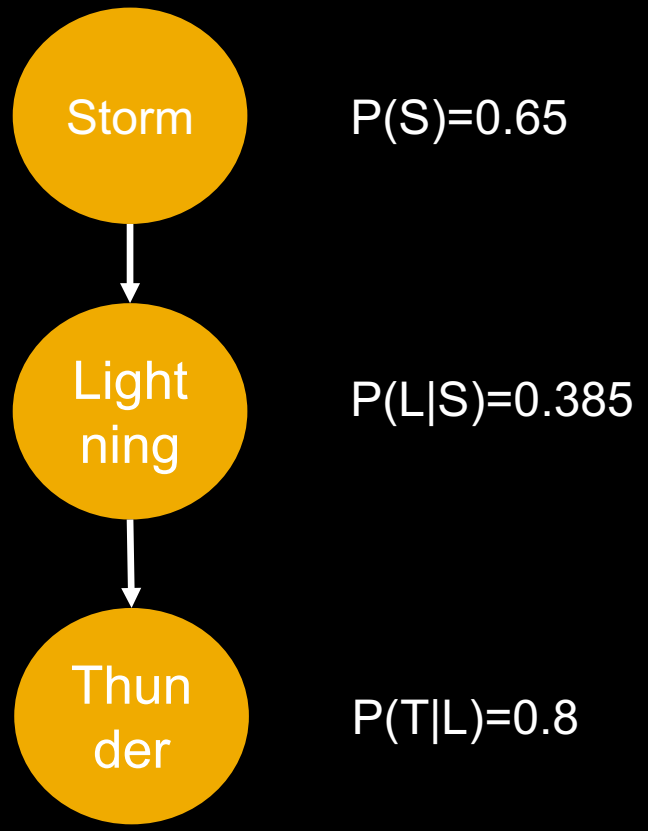
在计算 $p(y|x)$ 进行预测的时候，可不必计算分母$p(x)$，因为，我们关心的是分类的概率，而不是数据的概率，数据的概率也不会影响分类的概率。

输出：$y = \underset{y \epsilon \mathbb{Y}}{augmax}\ p(y|x)$



Bayesian Networks （Belief Net）

Storm	Lightning	%	Thunder	
T	T	0.25	T	0.20
			F	0.05
T	F	0.40	T	0.04
			F	0.36
F	T	0.05	T	0.04
			F	0.01
F	F	0.30	T	0.03
			F	0.27



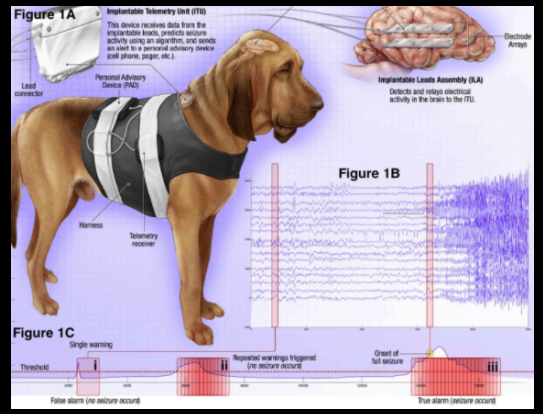
Confusion Matrix

		Condition (as determined by "Gold standard")		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	**False Positive** (Type I error)	Positive predictive value = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Test Outcome Positive}}$
	Test Outcome Negative	**False Negative** (Type II error)	**True Negative**	Negative predictive value = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Test Outcome Negative}}$
		Sensitivity = $\frac{\Sigma \text{ True Positive}}{\Sigma \text{ Condition Positive}}$	**Specificity** = $\frac{\Sigma \text{ True Negative}}{\Sigma \text{ Condition Negative}}$	

Assignment

TensorFlow RNN / LSTM Tutorial

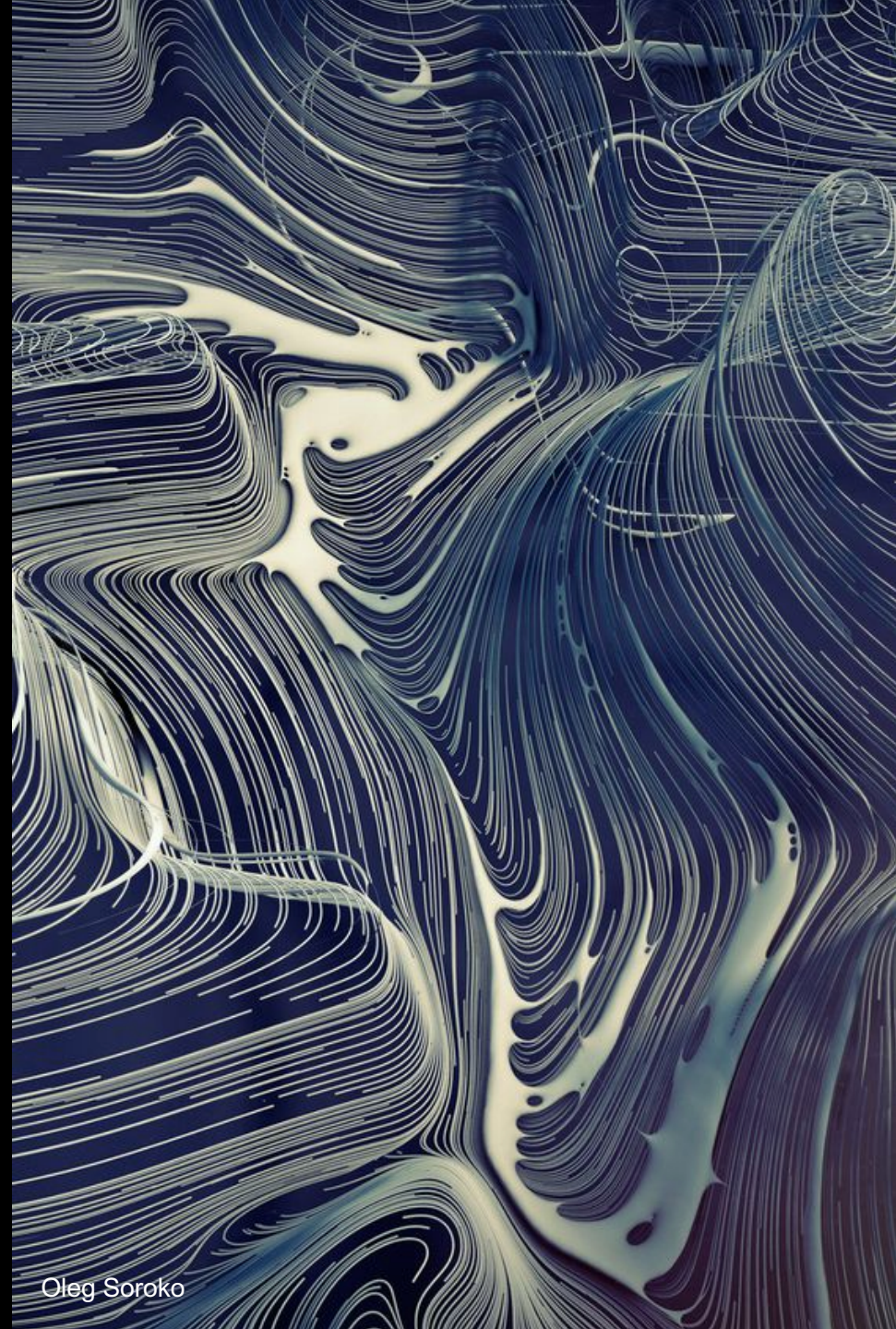
补充作业：
DeepBach: https://arxiv.org/pdf/1612.01010.pdf
癫痫发作预测：https://www.kaggle.com/c/seizure-prediction



Thank you!

Contact information:

鄔学宁（i025497)
Chief Data Scientist, SAP Silicon Valley Innovation Center
Address: No. 1001, Chenghui Road, Shanghai, 201023
Phone number: +8621-6108 5287
Email: x.wu@sap.com


Oleg Soroko