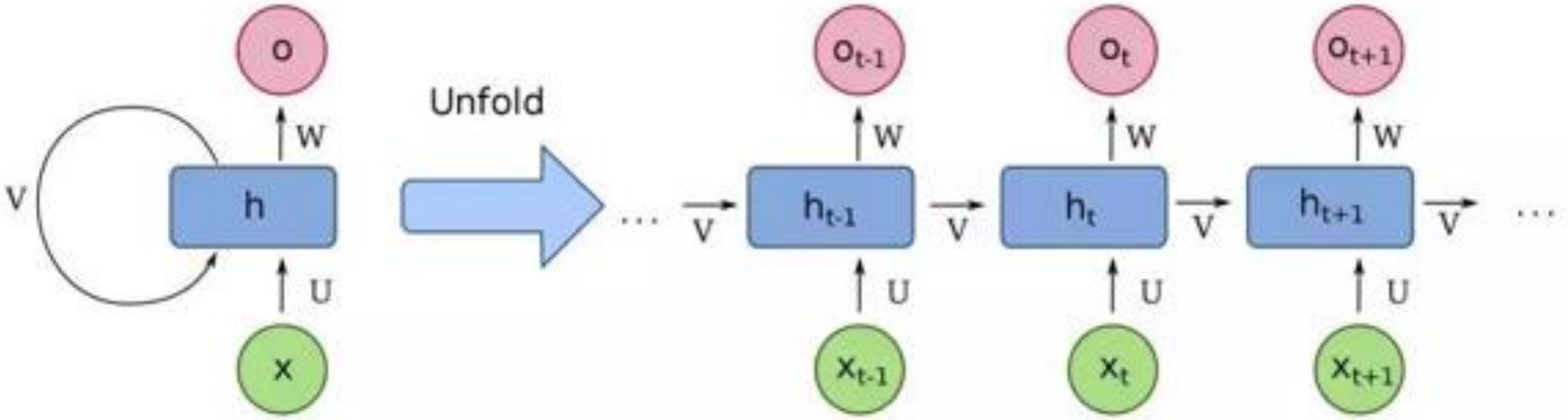
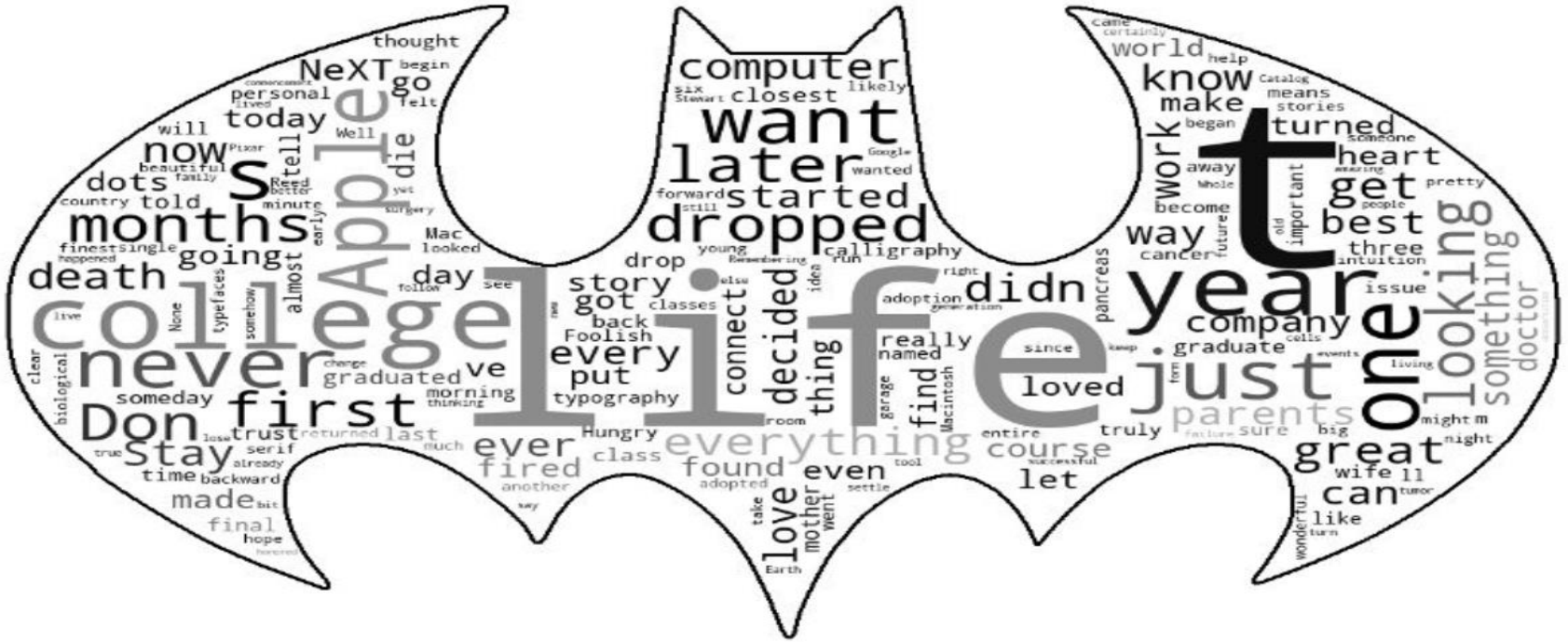
神经网络与序列建模

《人工神经网络》第 11 讲





主要内容

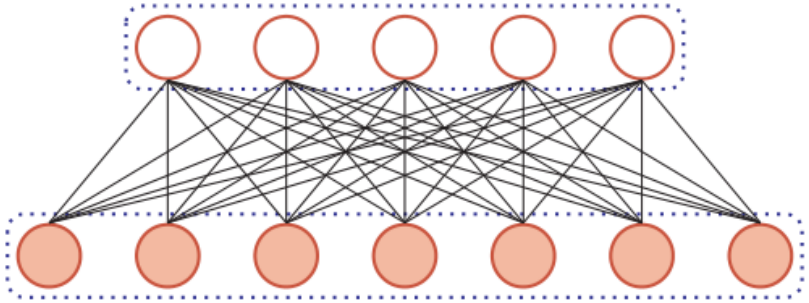
1. 循环神经网络
2. 文本特征提取
3. 几个应用展示

1 循环神经网络

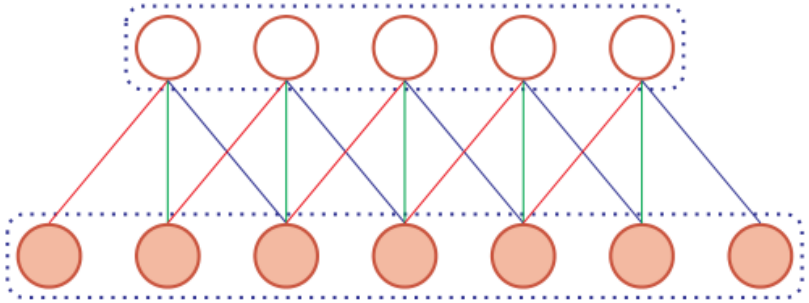
Recurrent Neural Network(RNN)

回顾：前馈神经网络

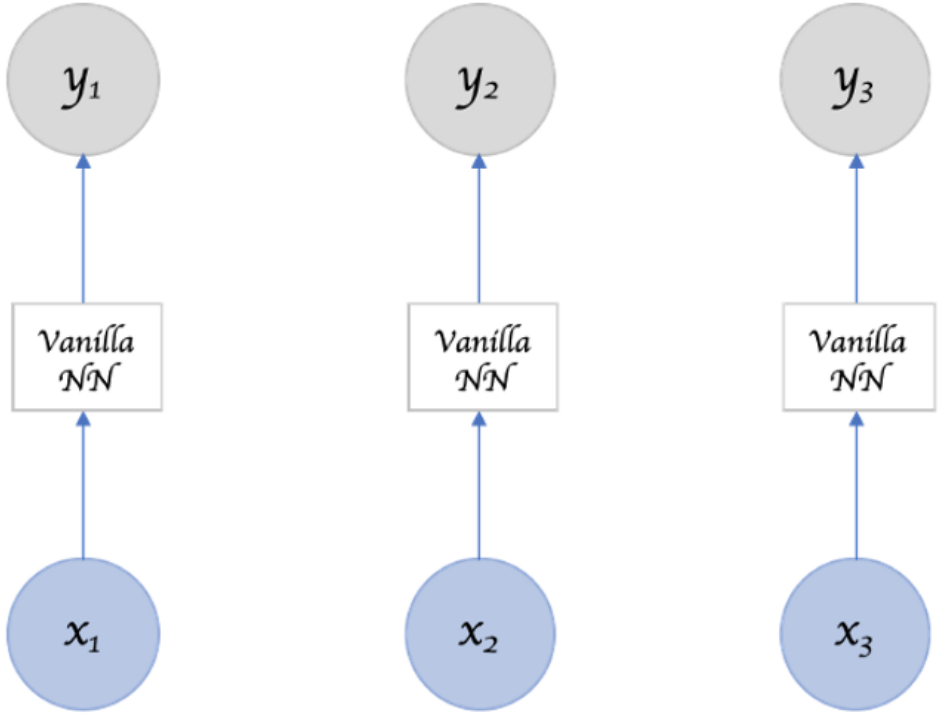
1. 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入。



(a) 全连接层



(b) 卷积层



2. 连接存在层与层之间，每层的节点之间是无连接的(有向无环图)

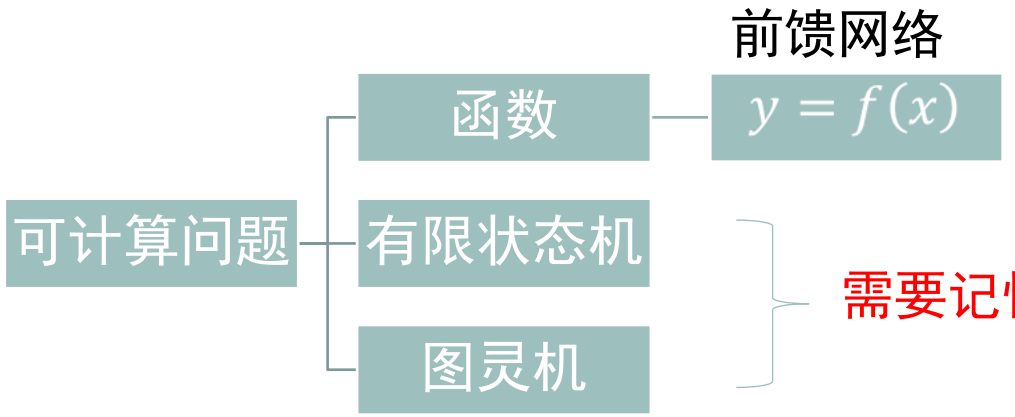
3. 输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据。

(时间)序列数据

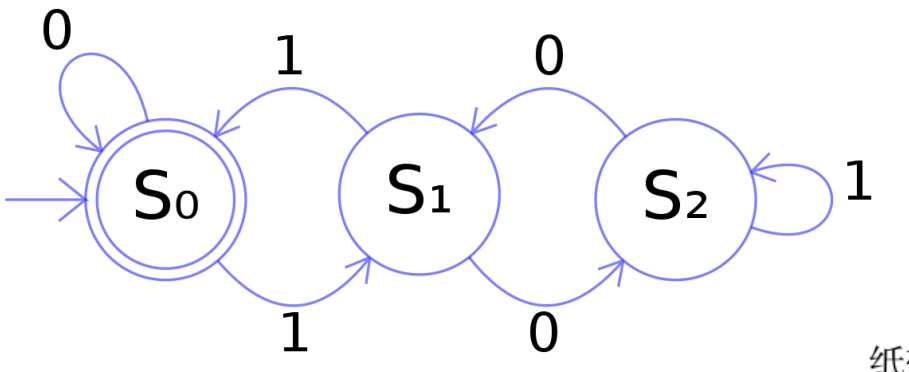
- 采集数据往往是对研究对象进行观测并记录，天然带有时间属性，几乎无处不在。比如：
  - 大气中每一时刻的温度、湿度、大气压强等；
  - 个人穿戴设备上采集的每分钟的心跳、体温、步行的步数等；
  - 股票、期货价格、大宗商品交易价格走势；
  - 医学检查心电图、血压监测等；
- 大数据时代各种类型的时序数据正迎来爆发式的增长　：
  - 社交平台图文数据
  - 电商平台交易数据
  - 各种监控视频

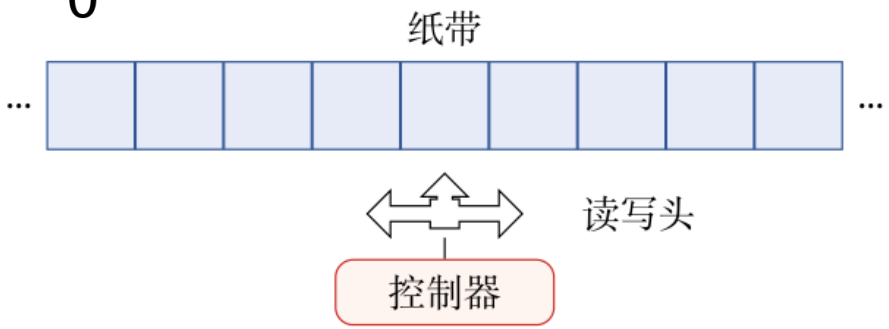
```
1   2016-01-01   -1.503070
2   2016-01-02    1.637771
3   2016-01-03   -1.527274
4   2016-01-04    1.202349
5   2016-01-05   -1.214471
6   2016-01-06    2.686539
7   2016-01-07   -0.665813
8   2016-01-08    1.210834
9   2016-01-09    0.973659
10  2016-01-10   -1.003532
11  2016-01-11   -0.138483
12  2016-01-12    0.718561
13  2016-01-13    1.380494
14  2016-01-14    0.368590
15  2016-01-15   -0.235975
16  2016-01-16   -0.847375
17  2016-01-17   -1.777034
18  2016-01-18    1.976097
19  2016-01-19   -0.631212
20  2016-01-20   -0.613633
21  Freq: D, dtype: float64
```

可计算问题



如何给网络增加记忆能力？

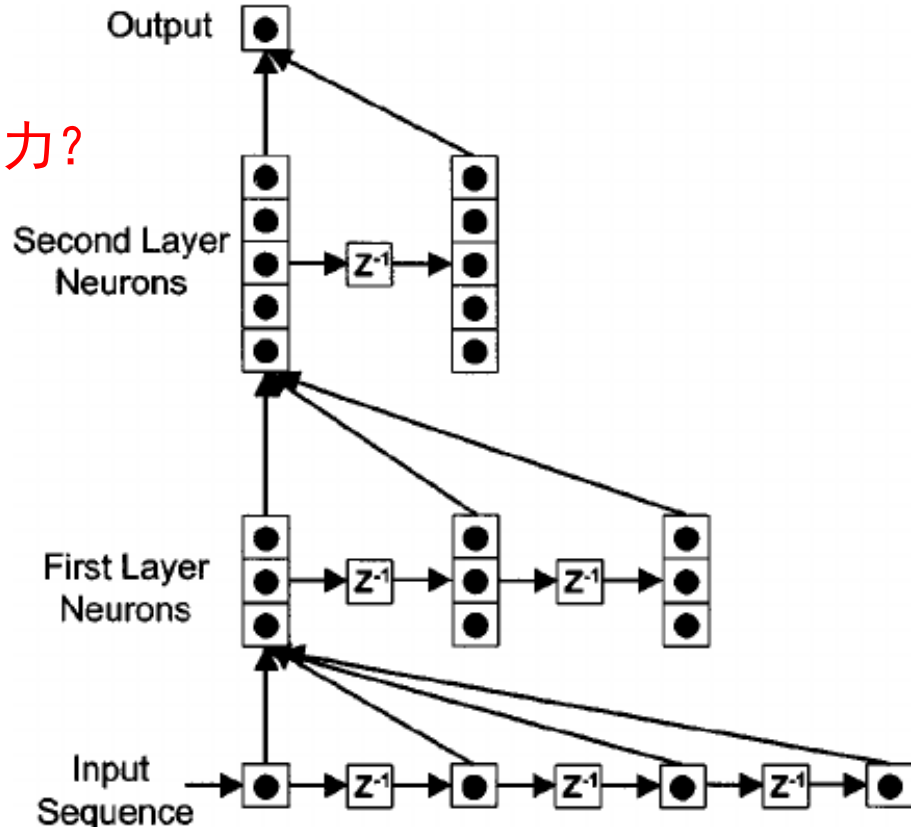




- 延时神经网络(Time Delay NN)
  - 建立一个额外的延时单元，用来存储网络的历史信息（可以包括输入、输出、隐状态等）

$$\boldsymbol{h}_t^{(l)} = f\left(\boldsymbol{h}_t^{(l-1)}, \boldsymbol{h}_{t-1}^{(l-1)}, \cdots, \boldsymbol{h}_{t-K}^{(l-1)}\right)$$

  - 这样，前馈网络就具有了短期记忆能力。



具有记忆能力的模型示例

- 自回归模型（Autoregressive Model，AR）
  - 一类时间序列模型，用变量y_t的历史信息来预测

$$\boldsymbol{y}_t = w_0 + \sum_{k=1}^{K} w_k \boldsymbol{y}_{t-k} + \epsilon_t$$

其中，$\epsilon_t \sim N(0, \sigma^2)$记为第t时刻的噪声.

- 有外部输入的非线性自回归模型，NARX

(Nonlinear Autoregressive with Exogenous Inputs Model)

$$\boldsymbol{y}_t = f(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \cdots, \boldsymbol{x}_{t-K_x}, \boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \cdots, \boldsymbol{y}_{t-K_y})$$

其中$f(\cdot)$ 表示非线性函数，可以是前馈网络， K_x 和 K_y 超参数.

Example 2 Use Eq. (5.31) with $m = 3$ to derive the three-step Adams-Bashforth technique.

Solution We have

$$\begin{aligned} y(t_{i+1}) &\approx y(t_i) + h\left[f(t_i, y(t_i)) + \frac{1}{2}\nabla f(t_i, y(t_i)) + \frac{5}{12}\nabla^2 f(t_i, y(t_i))\right] \\ &= y(t_i) + h\Big\{f(t_i, y(t_i)) + \frac{1}{2}[f(t_i, y(t_i)) - f(t_{i-1}, y(t_{i-1}))] \\ &\quad + \frac{5}{12}[f(t_i, y(t_i)) - 2f(t_{i-1}, y(t_{i-1})) + f(t_{i-2}, y(t_{i-2}))]\Big\} \\ &= y(t_i) + \frac{h}{12}[23f(t_i, y(t_i)) - 16f(t_{i-1}, y(t_{i-1})) + 5f(t_{i-2}, y(t_{i-2}))]. \end{aligned}$$

The three-step Adams-Bashforth method is, consequently,

$$w_0 = \alpha, \quad w_1 = \alpha_1, \quad w_2 = \alpha_2,$$

$$w_{i+1} = w_i + \frac{h}{12}[23f(t_i, w_i) - 16f(t_{i-1}, w_{i-1})] + 5f(t_{i-2}, w_{i-2})],$$

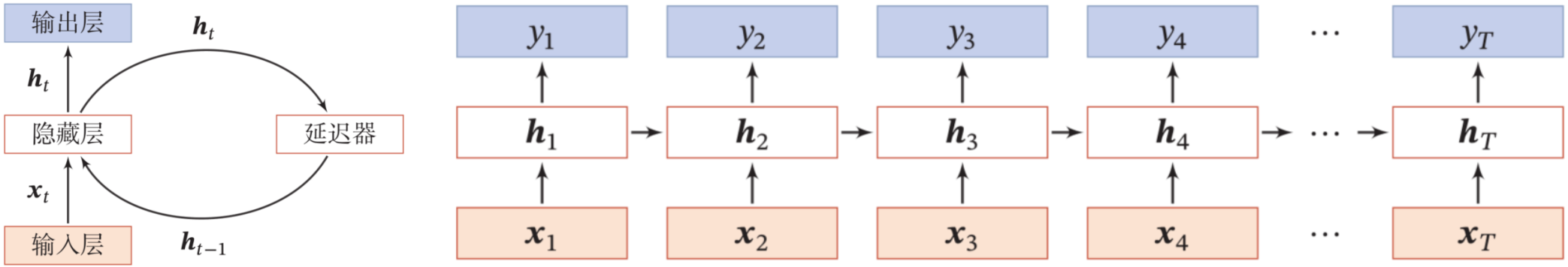
for $i = 2, 3, \ldots, N-1$. ■

循环(Recurrent)神经网络

- 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的时序数据

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

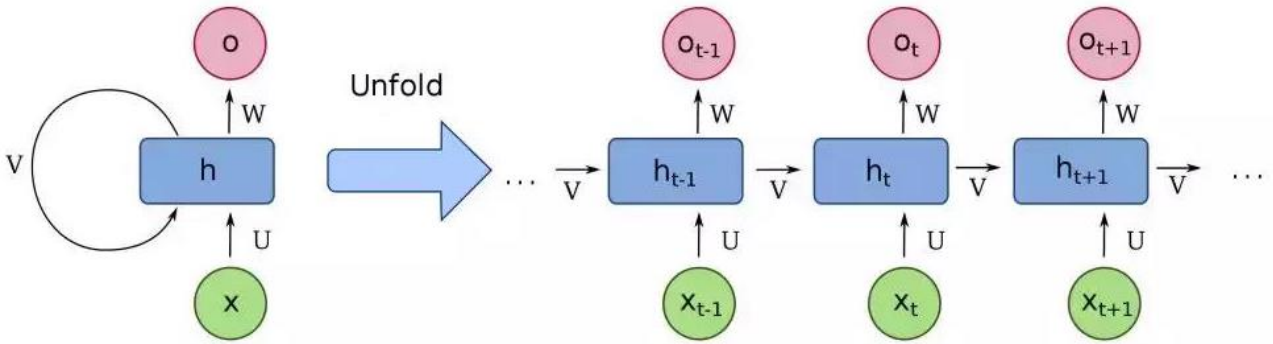
- 按时间展开：



- 循环神经网络比前馈神经网络更加符合生物神经网络的结构。
- 循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上

RNN 参数学习模型

demo_rnn.ipynb



$$\mathbf{h}_{t+1} = f(\mathbf{z}_{t+1}) = f(U\mathbf{h}_t + W\mathbf{x}_{t+1} + \mathbf{b})$$

- 机器学习
  - 给定一个训练样本(x,y)，其中
    - x = (x1 ,… ,xT)为长度是T 的输入序列，
    - y = (y1 ,… ,yT)是长度为T 的标签序列。

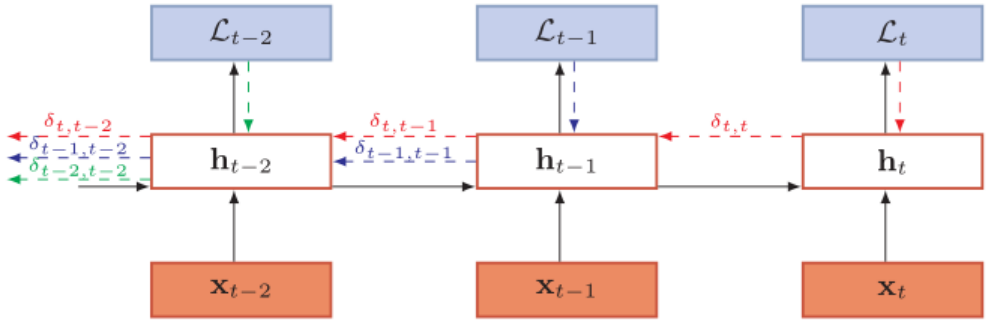
- 时刻t的瞬时损失函数为

$$\mathcal{L}_t = \mathcal{L}(\mathbf{y}_t, g(\mathbf{h}_t)),$$

- 总损失函数

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t.$$

- 随时间反向传播算法



$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t} \delta_{t,k}\mathbf{h}_{k-1}^{\mathrm{T}}$$

$$\delta_{t,k} = \prod_{\tau=k}^{t-1}\left(\mathrm{diag}(f'(\mathbf{z}_\tau))U^{\mathrm{T}}\right)\delta_{t,t}$$

$\delta_{t,k}$为第t时刻的损失对第k步隐藏神经元的净输入z_k的导数

RNN逼近性质

定理 6.1 – 循环神经网络的通用近似定理 [Haykin, 2009]： 如果一个完全连接的循环神经网络有足够数量的 sigmoid 型隐藏神经元，它可以以任意的准确率去近似任何一个非线性动力系统

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t), \tag{6.10}$$

$$\boldsymbol{y}_t = o(\boldsymbol{s}_t), \tag{6.11}$$

其中 $\boldsymbol{s}_t$ 为每个时刻的隐状态，$\boldsymbol{x}_t$ 是外部输入，$g(\cdot)$ 是可测的状态转换函数，$o(\cdot)$ 是连续输出函数，并且对状态空间的紧致性没有限制.

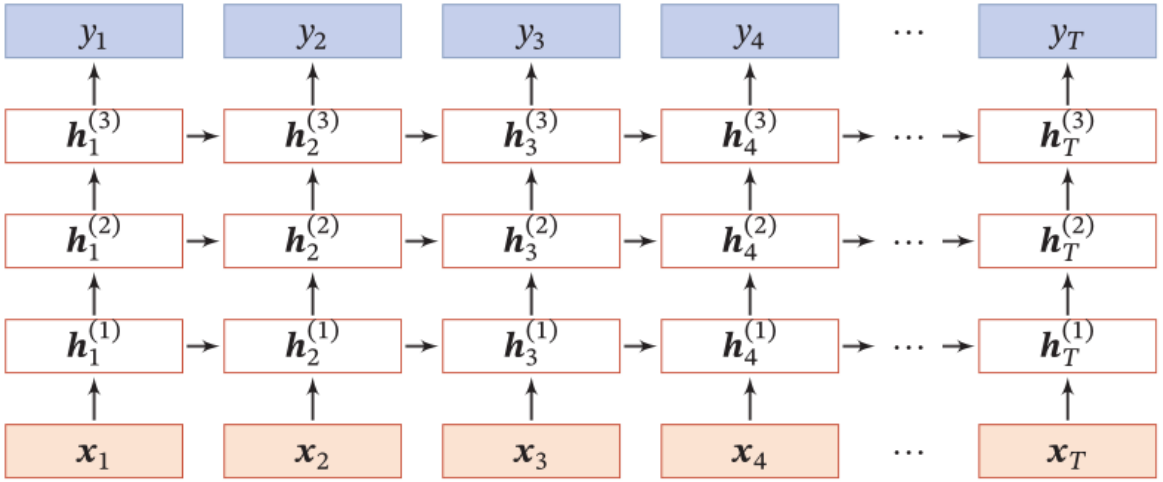
完全连接的循环网络是任何非线性动力系统的近似器

一个完全连接的循环神经网络可近似解决所有的可计算问题

- 图灵完备（Turing Completeness）是指一种数据操作规则，比如一种计算机编程语言，可以实现图灵机的所有功能，解决所有的可计算问题。

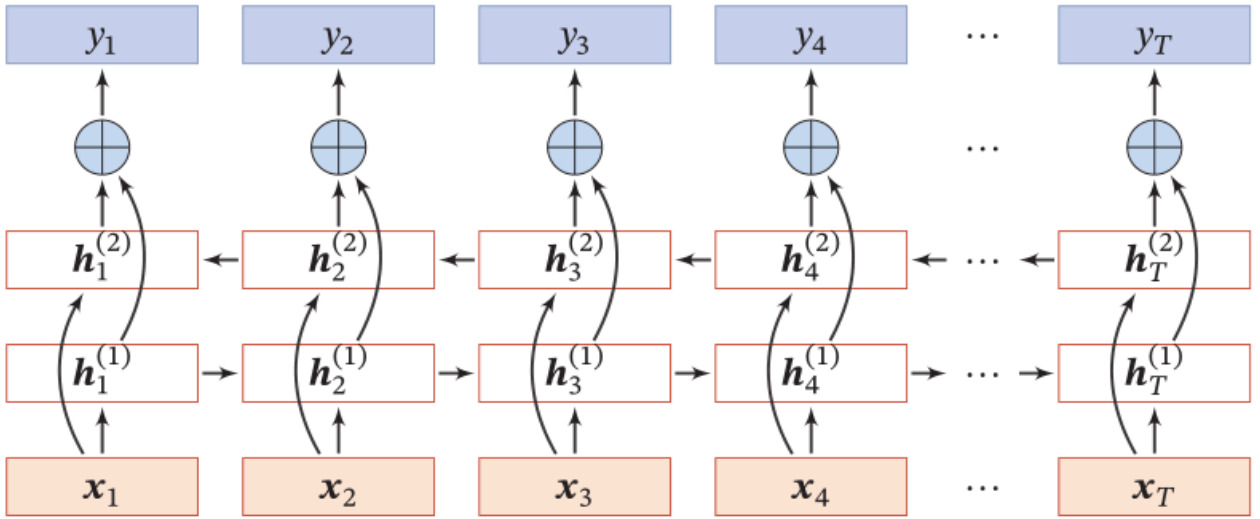
定理 6.2 – 图灵完备 [Siegelmann et al., 1991]： 所有的图灵机都可以被一个由使用 Sigmoid 型激活函数的神经元构成的全连接循环网络来进行模拟.

RNN的直接改进



堆叠循环神经网络

双向循环神经网络



长程依赖问题

- 改进方法
  - 循环边改为线性依赖关系

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t; \theta),$$

  - 增加非线性

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta),$$

残差网络？

梯度消失/爆炸现象

梯度

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T} \sum_{k=1}^{t} \delta_{t,k} \mathbf{h}_{k-1}^{\mathrm{T}}$$

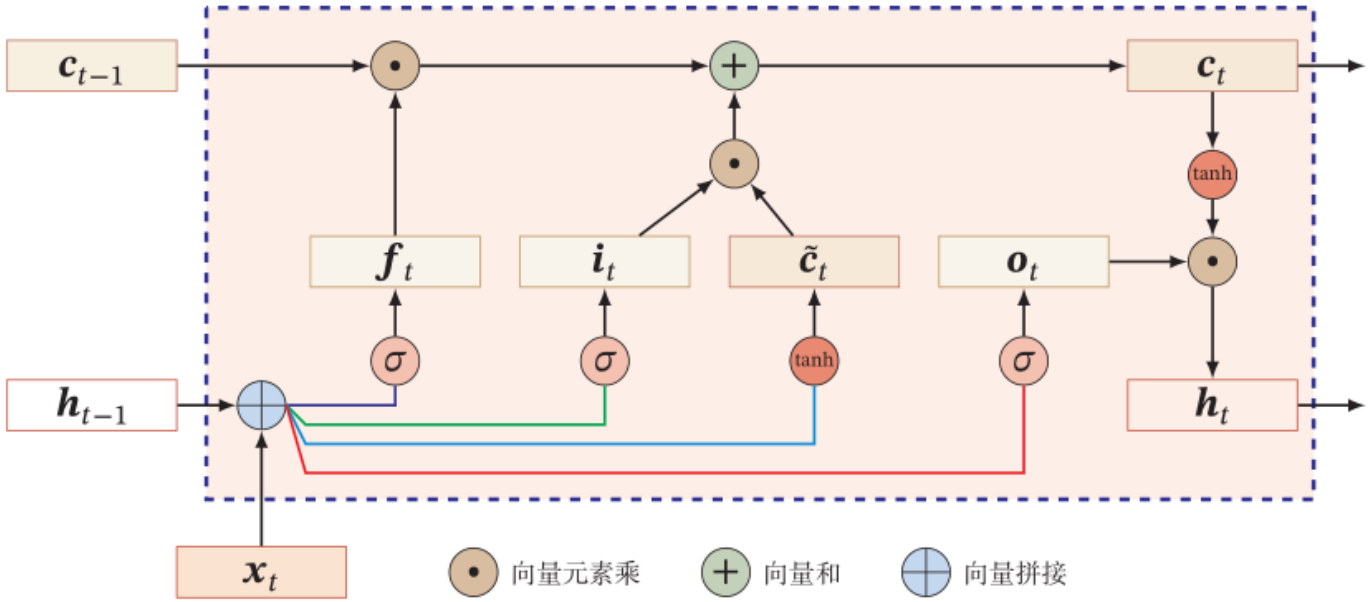
其中

$$\delta_{t,k} = \prod_{\tau=k}^{t-1} \left(\mathrm{diag}(f'(\mathbf{z}_\tau)) U^{\mathrm{T}} \right) \delta_{t,t}$$

- 循环神经网络在时间维度上非常深！
  - 梯度消失或梯度爆炸
  - 长程依赖问题:实际上只能学习到短周期的依赖关系
- 如何改进？
  - 梯度爆炸问题
    - 权重衰减
    - 梯度截断
  - 梯度消失问题
    - 改进模型

长短期记忆，LSTM

Long Short-Term Memory



$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i), \qquad \tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f), \qquad \mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,$$

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o), \qquad \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

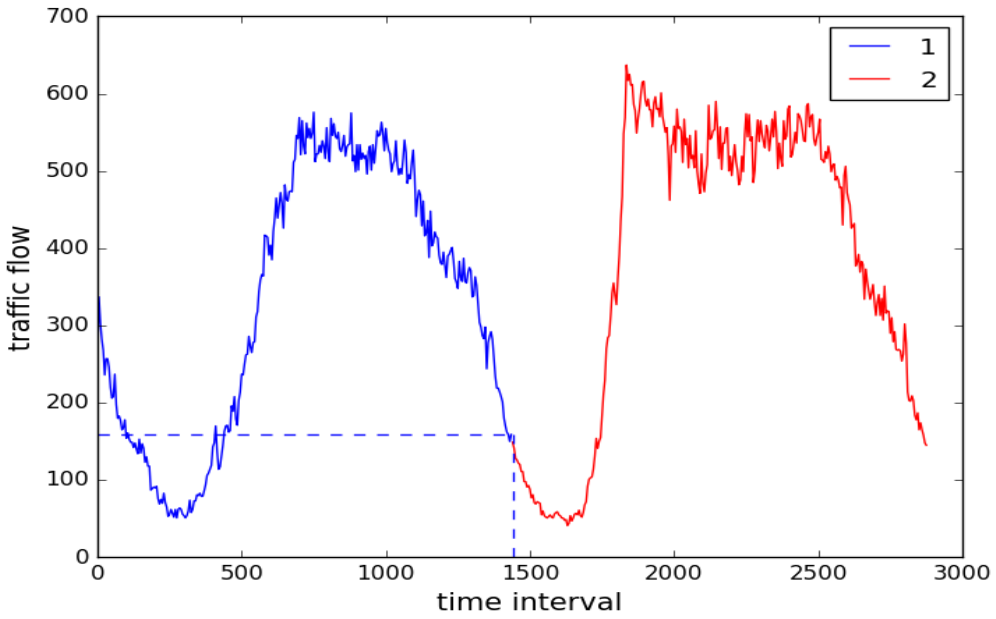
```python
def forward(self, x):
    self.times += 1
    # 遗忘门
    fg = self.calc_gate(x, self.Wfx, self.Wfh,
                        self.bf, self.gate_activator)
    self.f_list.append(fg)
    # 输入门
    ig = self.calc_gate(x, self.Wix, self.Wih,
                        self.bi, self.gate_activator)
    self.i_list.append(ig)
    # 输出门
    og = self.calc_gate(x, self.Wox, self.Woh,
                        self.bo, self.gate_activator)
    self.o_list.append(og)
    # 即时状态
    ct = self.calc_gate(x, self.Wcx, self.Wch,
                        self.bc, self.output_activator)
    self.ct_list.append(ct)
    # 单元状态
    c = fg * self.c_list[self.times - 1] + ig * ct
    self.c_list.append(c)
    # 输出
    h = og * self.output_activator.forward(c)
    self.h_list.append(h)
```

demo_lstm.ipynb

应用1、短时流量预测

traffic_predict_xxx.ipynb

根据历史交通数据，对未来一段 (5/15/30/45/60分钟) 时间的交通流数据做出预测。以某高速公路上某个检测点传感器每30s记录的通过车辆数为例，



- 预测任务描述：使用过去n个时间段的交通流量来预测未来h个时间段的交通流量

- 序列样本构建，以流量观测数据

  [261, 337, 303, 282, 268, 236, 257, 257]

为例，构造（输入，输出）样本对：

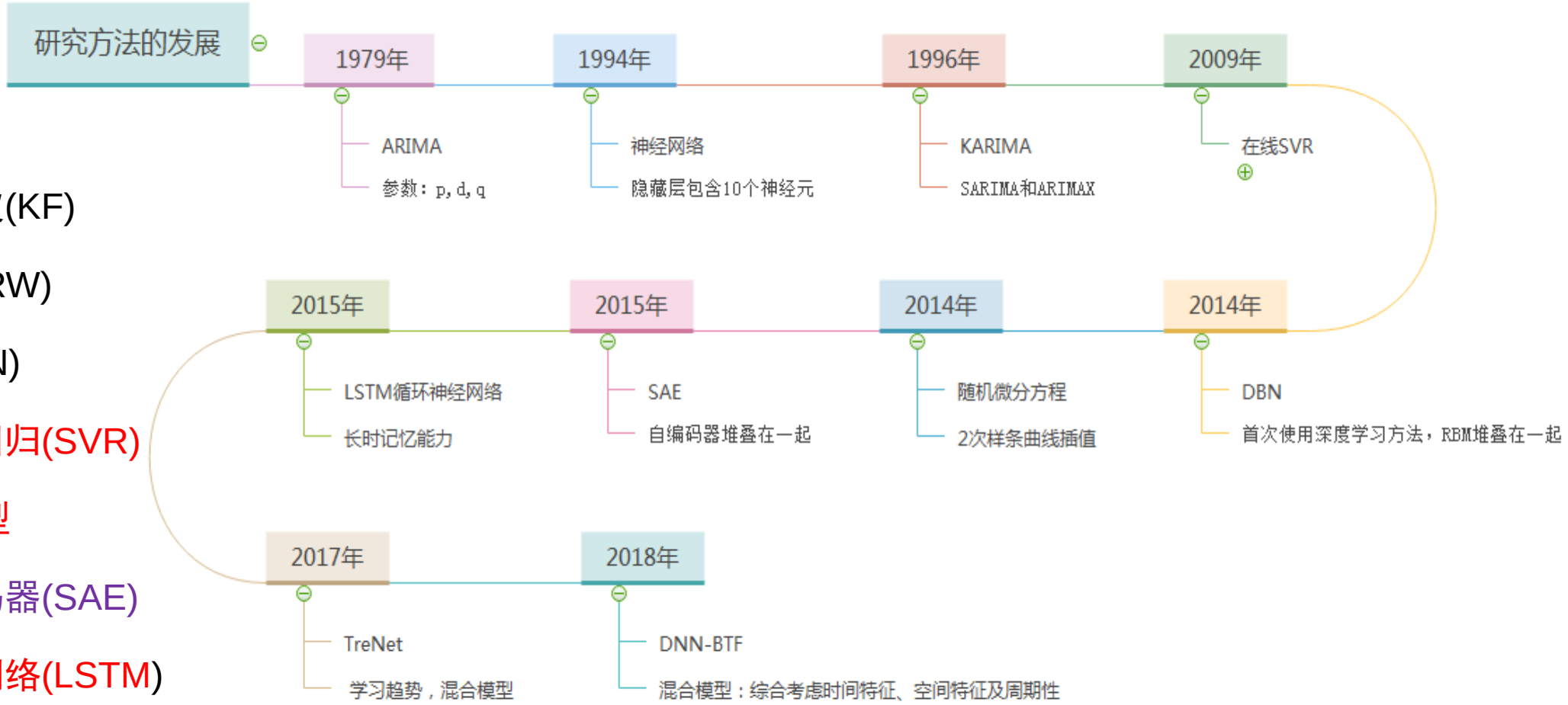
1. ([261, 337, 303, 282], 268)
2. ([337, 303, 282, 268], 236)
3. ([303, 282, 268, 236], 257)
4. ([282, 268, 236, 257], 257)

2002年，Abdulhai等人的研究表明：

如果要预测未来15分钟的交通流量，那么使用15分钟的历史数据进行预测是最准确的！

常用求解模型/方法

- 不同模型:
  - 卡尔曼滤波(KF)
  - 随机游走(RW)
  - K近邻(KNN)
  - 支持向量回归(SVR)
  - ARIMA模型
  - 堆栈自编码器(SAE)
  - 循环神经网络(LSTM)



- 基本流程包括数据预处理、建立模型、预测数据三部分

模型A:支持向量回归(SVR)

traffic_predict_svr.ipynb

- Support Vector Regression,是支持向量机模型运用于回归问题，运行时间短、准确率较高!
- SVR模型可描述为求解如下问题：给定数据点$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$，使用这些数据来拟合一个函数 $f(x) = \omega x + b$，使$f(x)$与y尽可能接近。在这里，使用一个ε-不敏感函数来计算两者之间的误差， ε为一个足够小的正数。

$$\min_{\omega,b} \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\max(0, |\omega^T x_i + b - y_i| - \varepsilon)$$

- 可以将其改为如下约束优化问题：

$$\min \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-)$$

$$s.t.\begin{cases} y_i - \omega^T x_i - b \le \varepsilon + \xi_i^+ \\ \omega^T x_i + b - y_i \le \varepsilon + \xi_i^- \\ \xi_i^+, \xi_i^- \ge 0 \end{cases}$$

模型B：自回归移动平均(ARIMA)

traffic_predict_svr.ipynb

- Auto-regressive Integrated Moving Average是自回归模型与移动平均模型的组合模型
- 时间序列若不平稳，则运用差分处理至将其转化为平稳的序列，得到参数d(差分阶数)
- 根据偏自相关系数PACF得出参数p(自回归模型的阶数)
- 根据自相关系数ACF得出参数q(移动平均模型的阶数)
- 建立模型ARIMA(p,d,q):

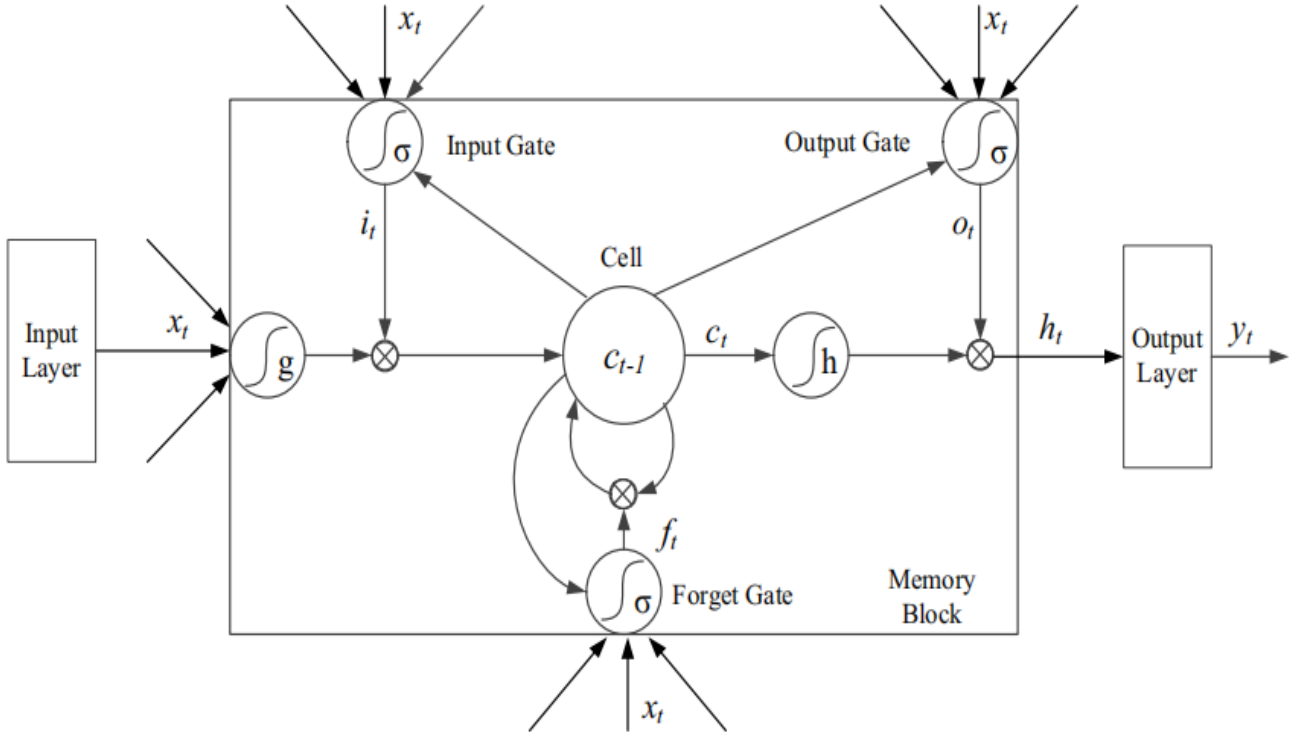
$$y_t = \mu + \sum_{i=1}^{p} a_i y_{t-i} + \sum_{i=1}^{q} b_i \varepsilon_{t-i} + \varepsilon_t$$

- 其中，μ是常数；a_i为自回归模型系数，b_i为移动平均模型系数，均为待求参数；p, q分别代表AR模型和MA模型的阶数；y_{t-i}为历史值，y_t为预测值；ε_t是随机白噪声，其均值为0，方差为常数，且对于任意$s \neq t$，ε_s与ε_t不相关，即$\mathrm{Cov}(\varepsilon_s, \varepsilon_t) = 0$。

模型C: LSTM

traffic_predict_lstm.ipynb

- 隐藏层使用1个重复的智能网络单元
- 四个门(前馈单元)的隐藏层神经元数量：
  - 30(5min)、120(15min)、128(30min)
- 在LSTM模块之后2个全连接层作回归:
  - 5min: [[30, 100], [100, 1]]
  - 15min: [[120, 150], [150, 1]]
  - 30min: [[128, 200], [200, 1]]
- 训练轮数：50000、60000、100000



实验结果

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\frac{|x_i - \hat{x}_i|}{x_i}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i|$$

$$RMSE = \left[\frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i|^2\right]^{\frac{1}{2}}$$

ARIMA	Criterion	5min	15min	30min
Station 1	MAE	25.663	74.012	175.213
	RMSE	35.620	104.320	251.059
Station 2	MAE	24.153	70.055	160.948
	RMSE	33.596	98.287	232.442

SVR	Criterion	5min	15min	30min
Station 1	MAE	24.747	64.744	127.787
	RMSE	34.221	90.942	176.503
Station 2	MAE	23.402	61.058	117.972
	RMSE	32.492	86.021	167.305

LSTM	Criterion	5min	15min	30min
Station 1	MAE	24.700	60.641	114.660
	RMSE	34.389	86.550	166.783
Station 2	MAE	23.153	57.971	105.403
	RMSE	32.317	82.578	156.047

更长时间间隔结果对比

Station 1	criterion	15min	30min
ARIMA	MAE	74.012	175.213
	RMSE	104.320	251.059
SVR	MAE	64.744	127.787
	RMSE	90.942	176.503
LSTM	MAE	60.641	114.660
	RMSE	86.550	166.783

LSTM的各种变体

- 没有遗忘门

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t.$$

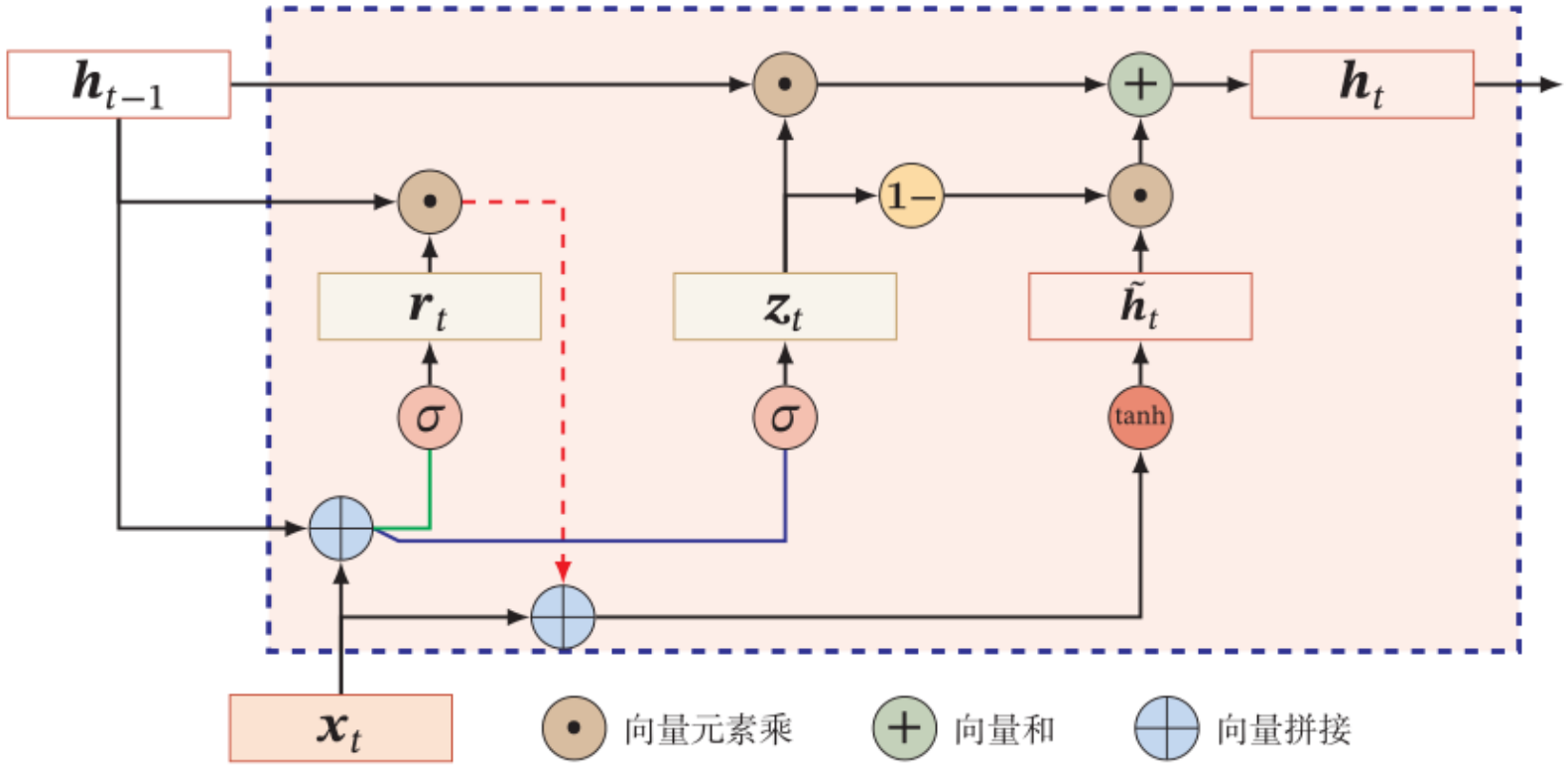
$$\mathbf{f}_t + \mathbf{i}_t = \mathbf{1}.$$

- 耦合输入门和遗忘门

$$\begin{aligned}
\mathbf{i}_t &= \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1} + \mathbf{b}_i), \\
\mathbf{f}_t &= \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1} + \mathbf{b}_f), \\
\mathbf{o}_t &= \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t + \mathbf{b}_o),
\end{aligned}$$

- peephole连接

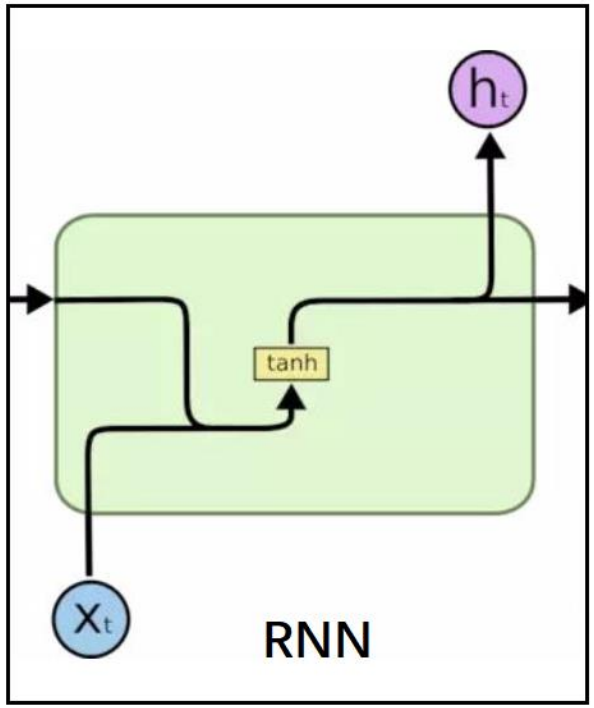
Gated Recurrent Unit, GRU

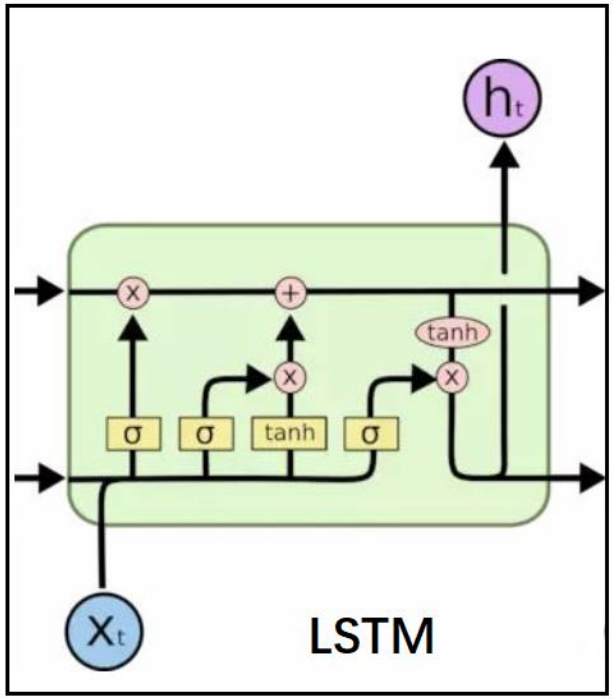


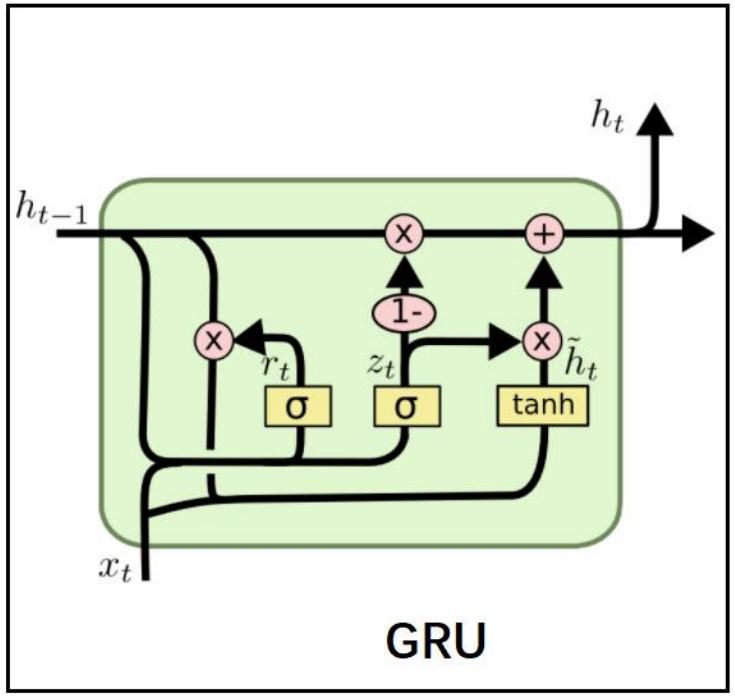
重置门 $\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r),$ $\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$

更新门 $\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z),$ $\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t,$

对比


h_t
tanh
X_t
RNN


h_t
tanh
σ
σ
tanh
σ
X_t
LSTM


h_t
h_{t-1}
1-
r_t
z_t
$\tilde{h}_t$
σ
σ
tanh
x_t
GRU

2 文本特征


特征表示
BoW 词袋模型
TF-IDF 词频-逆文档频率
N-gram
One-hot 独热码
Word2vec、Glove 词向量

向量化

Garbage In, Garbage Out

- ✓ 约90%的数据是非结构化的：图像、文本、音频、视频
- ✓ 文本数据以复杂形式出现：单词、列表、句子、段落
- ✓ 文本存储格式多源：HTML/WORD/TXT/PDF/EPUB

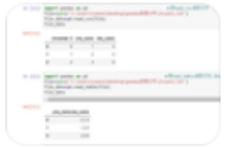
一、文本获取

csv 文本读写 pandas

网页 文库 资讯 贴吧 知道 图片

百度为您找到相关结果约44,300,000个

pandas读写文本(csv)文件_KJ.JK的博客-CSDN博客



2020年11月14日 接下来,通过一段代码来演示将
写入到CSV文件中: 2.通过read_csv()函数读取C
()函数的作用是将CSV文件的数据读取出来,并转

CSDN技术社区 百度快照

为您推荐: pandas中文本文件 pandas中属于文本文件的是 pa
csv读取为文本 pandas以文本格式读入 用来解析网

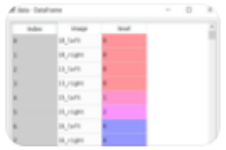
用pandas进行csv文件的读取和写入 - 百度文库

2页

最后可以封装一下pandas读写csv的方法,这样在工作中使用起来会比较方
(file_path): 2 # 读取csv数据,输入文件的路径,返回的是{[]}结构的数据 3 d

百度文库 保障 百度快照

pandas读写csv文件 全攻略_Huang_Fj的博客-CSDN



2021年10月21日 pandas读取csv文件 官网地址
g/pandas-docs/stable/reference/api/pandas.rea
的参数很多很多,但是日常用的就是 pandas.rea

CSDN技术社区 百度快照

Pandas操作CSV文件的读写实现方法_python_脚本之

2019年11月13日 这篇文章主要介绍了Pandas操作CSV文件的读写实现方
绍的非常详细,对大家的学习或者工作具有一定的参考学习价值,需要的朋

脚本之家 百度快照

json 文本读写 pandas

网页 文库 资讯 贴

百度为您找到相关结果约23,900,000个

pandas读写json文件_修勾勾

2021年10月22日 读取json文件 iris_data
data.head()) # 默认获取 前5条数据 取部

CSDN技术社区 百度快照

pandas读取json文件 Python_

2020年12月8日 使用pandas读取json文件
='utf-8')json_info=fr.read()df=pd.read_jso

CSDN技术社区 百度快照

pandas读取json文件 - CSDN

	Name	Gender	Nationality
0	John	Male	UK
1	Nick	Male	French
2	Ali	Female	USA
3	Joseph	Male	Brazil

2021年10月22
ndas库提供了可
的类和功能。

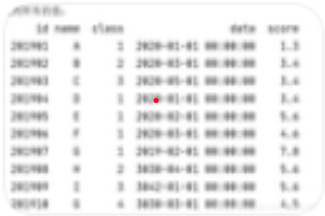
CSDN技术社

xlsx 文本读写 pandas

网页 贴吧 知道 文库 图片

百度为您找到相关结果约21,500,000个

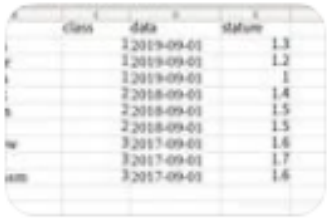
...读xlsx文件_RitaLoveCode的博客-CSDN



2020年9月2日 读取前n行数据 读
号和列标题 将数据转换为字典形
有数据 df1=pd.read_excel('d1.x

CSDN技术社区 百度快照

...writer读写xlsx文件_-牧野-的博客-CSDN



2019年3月8日 已有xlsx文件如下
mport pandas as pd # 1. 读取前
xlsx')#读取xlsx中第一个sheet d

CSDN技术社区 百度快照

python中pandas和xlsxwriter读写xlsx文件

2019年3月9日 本篇文章给大家带来的内容是关于python中
介绍(附代码),有一定的参考价值,有需要的朋友可以参考一

例1：json 格式

```
[
    {
    "id": "A001",
    "name": "搜狐",
    "url": "www.sohu.com",
    "likes": 1
    },
    {
    "id": "A002",
    "name": "Google",
    "url": "www.google.com",
    "likes": 2
    },
    {
    "id": "A003",
    "name": "淘宝",
    "url": "www.taobao.com",
    "likes": 3
    }
]
```

也可以从URL中读取 JSON 数据：

```
URL = 'https://www.test.com/test.json'
df = pd.read_json(URL)
```

将上面的数据以纯文本格式保存在test.json文件中

```
import pandas as pd
df = pd.read_json('test.json')
df
```

结果如下：

	id	name	url	likes
0	A001	搜狐	www.sohu.com	1
1	A002	Google	www.google.com	2
2	A003	淘宝	www.taobao.com	3

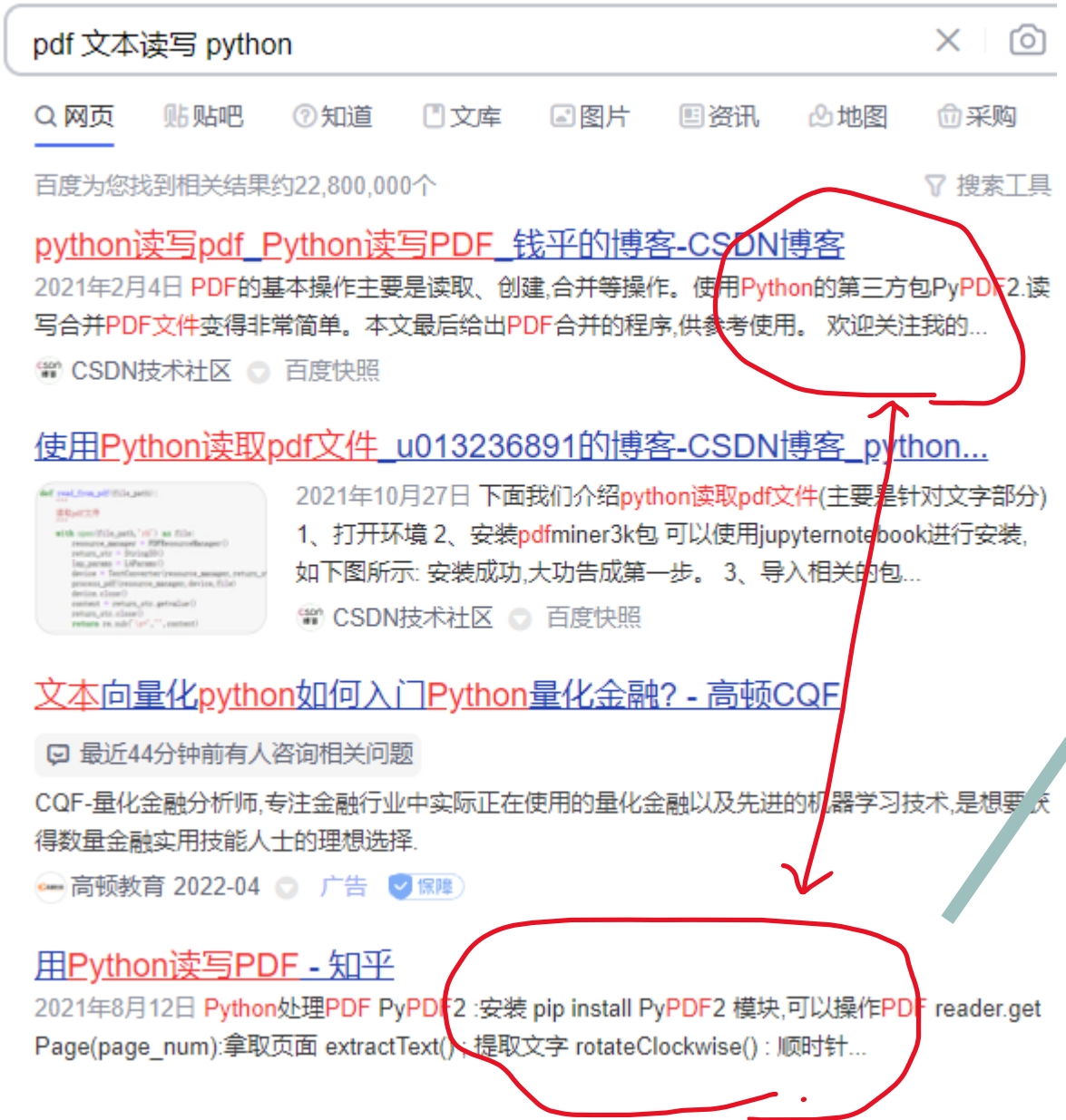
JSON 对象与 Python 字典具有相同的格式，可以直接

```
# 字典格式的 JSON
json_str = {
    "col1":{"row1":1,"row2":2,"row3":3},
    "col2":{"row1":"x","row2":"y","row3":"z"}
}
# 读取 JSON 转为 DataFrame
df = pd.DataFrame(json_str)
df
```

结果如下：

	col1	col2
row1	1	x
row2	2	y
row3	3	z

例2：PDF文件读写



Python处理PDF

PyPDF2 :安装 pip install PyPDF2 模块，

reader.getPage(page_num):拿取页面

extractText() ; 提取文字

rotateClockwise() : 顺时针旋转多少度

addBlanPage() : 添加空白页面

addPage() : 添加页面

os.path.abspath () 函数： os 模块下寻找

decrypt() : 解密函数

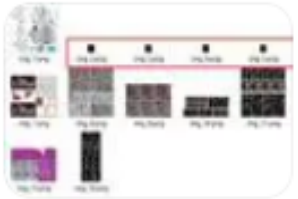
reportlab:创建PDF最强大的库

rotate() : 旋转画布角度

例2：PDF with PyPDF2

PyPDF2 文字提取

python提取pdf中的文字和图片_python 三种方法提取pdf..._...



2021年2月21日 今天就跟大家系统分享几种Python提取 PDF
法。其实没有非常完美的方法,每种方法提取效率都不是百分之
以考虑用多种方法进行互补,主要将涉及:基于 fitz 库和正则搜..

CSDN技术社区 百度快照

使用Python和PyPDF2从PDF文件中提取文本 - 我爱学习网

2021年3月25日 我想从给定的PDF中提取文本。使用的代码是:from PyPDF2 import Pd
r def extract_information(pdf_path): with open(pdf_path, 'rb') as f: pdf = Pd...

我爱学习网 百度快照

pypdf2提取文本 - CSDN

csdn已为您找到关于pypdf2提取文本相关内容,包含pypdf2提取

https://www.5axxw.com/questions/c

使用的代码是：

```
from PyPDF2 import PdfFileReader
def extract_information(pdf_path):
    with open(pdf_path, 'rb') as f:
        pdf = PdfFileReader(f)
        number_of_pages = pdf.getNumPages()
        for pages in range(number_of_pages):
            page=pdf.getPage(pages)
            page_content=page.extractText()
            print(page_content)


if __name__ == '__main__':
    path = 'test.pdf'
    extract_information(path)
```

但当我运行上述代码时，我得到了以下输出：

```
PS E:\Omkar\Coding\Python\pdfSearch> python .\sc
 !"#$%&!'()*+&,$ !")-!+)-. !"#$%$&'$%%()%*)(+(+$
;FM:0@HC$:FDDG$HU$%%/%?
V>%?W*%JPJ?++ A&3#=%(+,(>(?X:ED@@G$0FM:E9D%(+,(>
```

例：在线获取数据

事件文本分类与摘要生成的数据形式如下所示。

	dispute_brief_situation	dispute_eff
0	婚姻纠纷	简单纠纷
1	2019年5年1日，当事人赵光和蓝云鹏在碧湖蓝巨星门口引发争吵，互相扭打导致赵光受伤一事申请人民调解。	一般纠纷
2	被申请人拖欠申请人物业费	简单纠纷
3	双方当事人为村自来水不能使用发生争吵打架，导致夏勤良受伤遂产生纠纷。	一般纠纷
4	2019年7月至2020年1月叶奇木在物流城D区承包陶志刚的工地钢筋焊接，因双方在报酬上差价有争议，共计15232 元。	一般纠纷
5	2016年4月13日，被申请人鲍施有由中国建设银行股份有限公司云和支行确认信息真实性，向申请人申请提交《个人开户与银行签约服务申请书》，2017年5月21日、2018年1月6日、2018年2月8日通过互联网分别与申请人签订《中国建设银行借贷通服务协议》，被申请人分别向申请人借款人民币100000元、31100元、48100元并签订了电子合同《中国建设银行"快e贷"个人借款合	一般纠纷

```
# 导入模块
import pandas as pd
import requests
import json
# 第 1 页数据
url = "http://xxx.31.xxx.86:xxxxx/disputes/importData?page={}".format(1)
auth = {"Authorization":"Basic dGVzdDp0ZXN0drtughddU2"}
data_per = requests.get(url=url, headers=auth)
# json2dataframe
data_df = pd.DataFrame(json.loads(data_per.text)['results'])
```

二、文本预处理

1. 将文本转化为小写、拼写校正
2. 删除标点符号
3. 删除停用词
4. 文本分词(jieba)

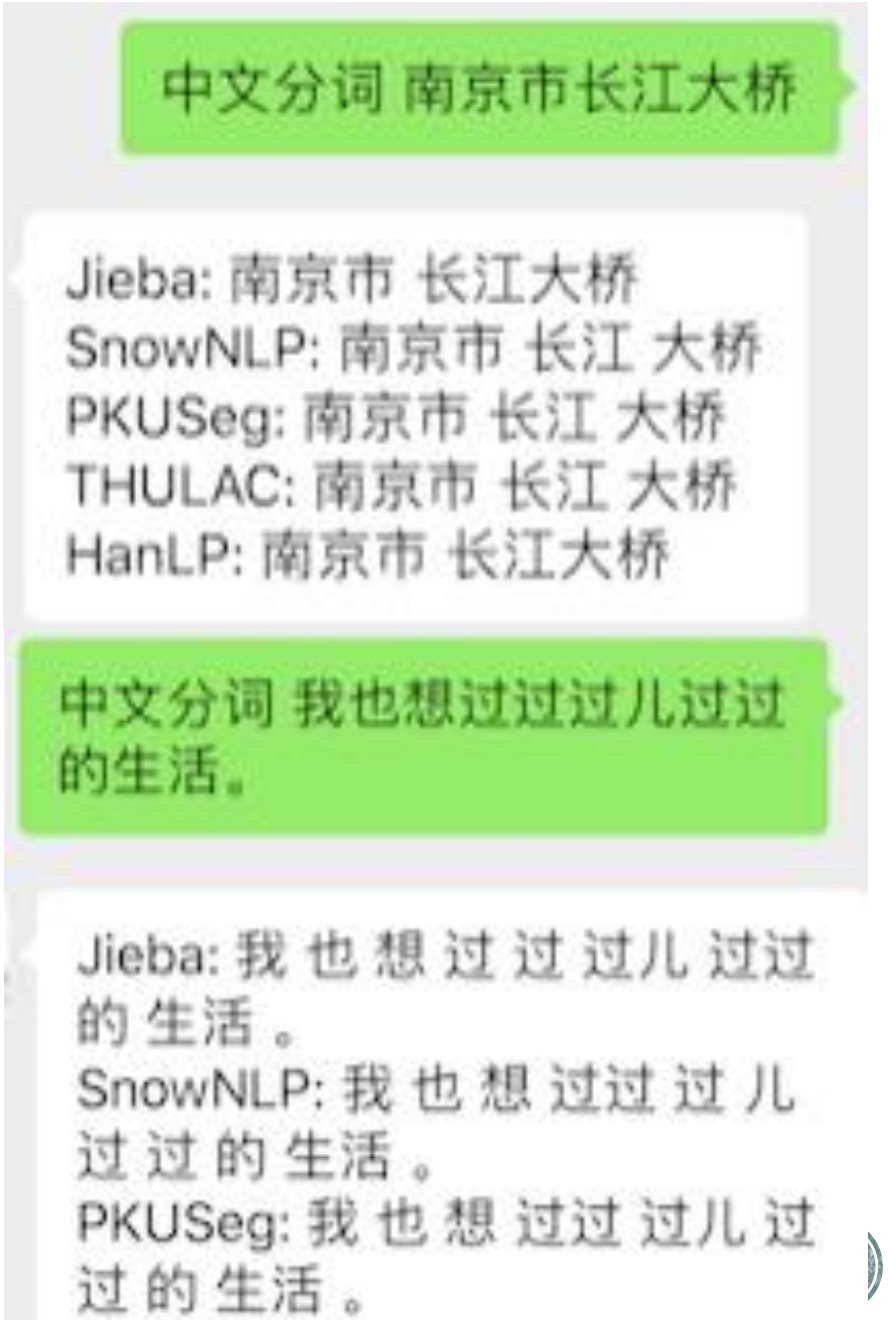
5. 词干提取(stemming): cats => cat, effective => effect
6. 词形还原(lemmatization): driving => drive, drove => drive

A)中文分词

1. **规则分词:** 通过人工设立词库词典，按照一定方式进行匹配切分
   - 将每个字符串与词表中的词**逐一匹配**，找到则分词，否则不予切分
   - 实现简单高效，但对新词很难进行处理
   - **正向最大匹配法、逆向最大匹配法、双向最大匹配**
2. **统计分词：**把每个词看成由各个字组成，如果**相连的字**在不同的文本中**出现次数越多**，就证明这相连的字很可能就是一个词。
   1. 建立统计语言模型
   2. 对句子分词的结果进行概率计算，获得概率最大的分词方式
3. **混合分词(规则 + 统计)**

- jieba 是基于Python的中文分词工具：

  https://github.com/fxsjy/jieba



B) 可视化：词云图

词云(Word Cloud)是文本数据的一种可视化表达。利用高频的关键词来传达出大量文本数据背后具有价值的信息，通过不同颜色和字体大小表达出不同程度的重要性，从而能够帮助我们快速的掌握文本所要表达的意思，更加具有视觉上的冲击力。

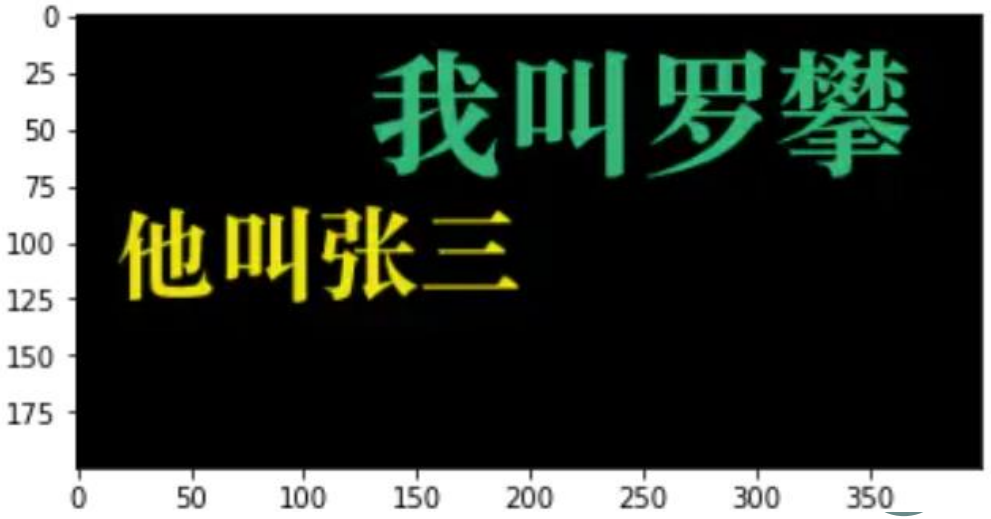
- 一个简单的(基于WordCloud)实现：

```
from matplotlib import pyplot as plt
from wordcloud import WordCloud

text = '我叫罗攀，他叫张三，我叫罗攀'

wc = WordCloud(font_path = r'/System/Library/Fonts/Supplemental/Songti.ttc')
wc.generate(text)

plt.imshow(wc)
```



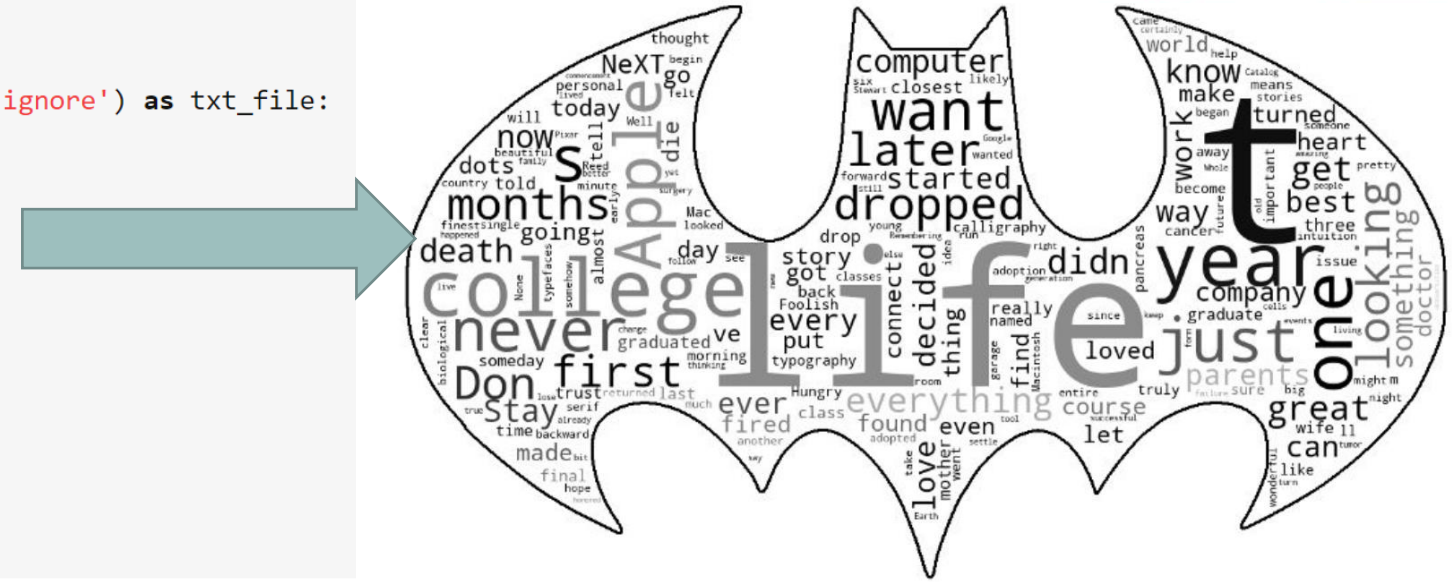
风格化

```
import stylecloud
stylecloud.gen_stylecloud(file_path='SJ-Speech.txt',icon_name="fas fa-apple-alt")
```

```
from wordcloud import WordCloud, ImageColorGenerator
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
# create a mask based on the image we wish to include
my_mask = np.array(Image.open('batman-logo.png'))
# create a wordcloud
wc = WordCloud(background_color='white',
                    mask=my_mask,
                    collocations=False,
                    width=600,
                    height=300,
                    contour_width=3,
                    contour_color='black',
                    stopwords=stop_words)

with open('SJ-Speech.txt',encoding='gb18030',errors='ignore') as txt_file:
  texto = txt_file.read()
wc.generate(texto)
image_colors = ImageColorGenerator(my_mask)
wc.recolor(color_func=image_colors)

plt.figure(figsize=(20, 10))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
wc.to_file('wordcloud2.png')
plt.show()
```



文本特征提取1： 词频统计

- **TF-IDF(Term Frequency–Inverse Document Frequency）**
  - 一种用于信息检索（information retrieval）与文本挖掘（text mining）的常用**加权技术**
  - 一种统计方法，评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度

$$TF-IDF(w) = TF_w \times IDF_w$$

其中，字词的重要性与出现的次数成正比，但与它在语料库中出现的频率成反比：

$$TF_w = \frac{\text{某类词条}w\text{出现次数}}{\text{该类中所有词条数}}, \quad IDF_w = log\frac{\text{语料库文档总数}}{\text{包含}w\text{文档数}+1}$$

  - 主要思想：如果某个单词在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

实例：利用gensim计算tfidf值

1

```
from gensim import corpora
# 给语料库中每个词一个从零开始的id
dictionary = corpora.Dictionary(word_list)
new_corpus = [dictionary.doc2bow(text) for text in word_list]
print(new_corpus)
##输出
[[(0, 1), (1, 1), (2, 1), (3, 1)], [(1, 1), (4, 1), (5, 1), (6, 1)], [(5, 1), (7, 1),

# 通过下面的方法可以看到语料库中每个词对应的id
print(dictionary.token2id)
{'我': 0, '电影': 1, '看': 2, '要': 3, '刘德华': 4, '播放': 5, '的': 6, '动画片': 7, '小
```

2

```
# 训练模型并保存
from gensim import models
tfidf = models.TfidfModel(new_corpus)
tfidf.save("my_model.tfidf")
```

3

```
# 使用这个训练好的模型得到单词的tfidf值
tfidf_vec = []
for i in range(len(corpus)):
    string = corpus[i]
    string_bow = dictionary.doc2bow(string.lower().split())
    string_tfidf = tfidf[string_bow]
    tfidf_vec.append(string_tfidf)
print(tfidf_vec)
```

其他python包的实现

Sklearn实现TF-IDF算法

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

x_train = ['TF-IDF 主要 思想 是','算法 一个 重要 特点 可以 脱离 语料库 背景',
           '如果 一个 网页 被 很多 其他 网页 链接 说明 网页 重要']
x_test=['原始 文本 进行 标记','主要 思想']

#该类会将文本中的词语转换为词频矩阵，矩阵元素a[i][j] 表示j词在i类文本下的词频
vectorizer = CountVectorizer(max_features=10)
#该类会统计每个词语的tf-idf权值
tf_idf_transformer = TfidfTransformer()
#将文本转为词频矩阵并计算tf-idf
tf_idf = tf_idf_transformer.fit_transform(vectorizer.fit_transform(x_train))
#将tf-idf矩阵抽取出来，元素a[i][j]表示j词在i类文本中的tf-idf权重
x_train_weight = tf_idf.toarray()

#对测试集进行tf-idf权重计算
tf_idf = tf_idf_transformer.transform(vectorizer.transform(x_test))
x_test_weight = tf_idf.toarray()  # 测试集TF-IDF权重矩阵
```

NLTK实现TF-IDF算法

```
from nltk.text import TextCollection
from nltk.tokenize import word_tokenize

#首先，构建语料库corpus
sents=['this is sentence one','this is sentence two','this is se
sents=[word_tokenize(sent) for sent in sents] #对每个句子进行分词
print(sents)  #输出分词后的结果
corpus=TextCollection(sents)  #构建语料库
print(corpus)  #输出语料库

#计算语料库中"one"的tf值
tf=corpus.tf('one',corpus)    # 1/12
print(tf)

#计算语料库中"one"的idf值
idf=corpus.idf('one')      #log(3/1)
print(idf)

#计算语料库中"one"的tf-idf值
tf_idf=corpus.tf_idf('one',corpus)
print(tf_idf)
```

TF-IDF不足之处

$$TF_w = \frac{\text{某类词条}w\text{出现次数}}{\text{该类中所有词条数}}, \quad IDF_w = log\frac{\text{语料库文档总数}}{\text{包含}w\text{文档数}+1}$$

1. 没有考虑特征词的位置因素对文本的区分度，词条出现在文档的不同位置时，对区分度的贡献大小是不一样的。
2. 按照传统TF-IDF，往往一些生僻词的IDF(反文档频率)会比较高、因此这些生僻词常会被误认为是文档关键词。
3. 传统TF-IDF中的IDF部分只考虑了特征词与它出现的文本数之间的关系，而忽略了特征项在一个类别中不同的类别间的分布情况。
4. 对于文档中出现次数较少的重要人名、地名信息提取效果不佳。

文本特征提取2： 独热(One-Hot)编码

又称为一位有效编码，主要是采用N位状态寄存器来对N个状态进行编码，每个状态都由独立的寄存器位，并且在任意时候只有一位有效。

足球 => 1000

篮球 => 0100

羽毛球 => 0010

乒乓球 => 0001

中国 => 100

美国 => 010

法国 => 001

男 => 10

女 => 01

样本为["男","中国","乒乓球"]的时候，

[1，0，1，0，0，0，0，0，1]

中国女足？

一个例子

来自https:// www.jianshu.com/p/3acd91b267cf

```
# 导入用于对象保存和加载的包
from sklearn.externals import joblib
# 导入keras中的词汇映射器Tokenizer
from keras.preprocessing.text import Tokenizer

# 初始化一个词汇表
vocab = {"周杰伦","陈奕迅","王力宏","李宗盛

# 实例化一个词汇映射器
t = Tokenizer(num_words=None, char_level=

# 在映射器上你和现有的词汇表
t.fit_on_texts(vocab)

# 循环遍历词汇表，将每一个单词映射为one-hot
for token in vocab:
    # 初始化一个全零向量
    zero_list = [0] * len(vocab)
    # 使用映射器转化文本数据，每个词汇对应从1开始
    token_index = t.texts_to_sequences([token])[0][0] -1
    # 将对应的位置复制为1
    zero_list[token_index] = 1
    print(token, "的one-hot编码为: ", zero_list)

# 将拟合好的词汇映射器保存起来
tokenizer_path = "./Tokenizer"
joblib.dump(t, tokenizer_path)
```

```
→ untitled2 python one-hot-test.py
周杰伦 的one-hot编码为:  [1, 0, 0, 0, 0, 0]
王力宏 的one-hot编码为:  [0, 1, 0, 0, 0, 0]
鹿晗 的one-hot编码为:  [0, 0, 1, 0, 0, 0]
     的one-hot编码为:  [0, 0, 0, 1, 0, 0]
陈奕迅 的one-hot编码为:  [0, 0, 0, 0, 1, 0]
李宗盛 的one-hot编码为:  [0, 0, 0, 0, 0, 1]
```

1、编写脚本：vim demo2.py

```
# 导入用于对象保存于加载的包
from sklearn.externals import joblib
# 将之前已经训练好的词汇映射器加载进来
t = joblib.load("./Tokenizer")

token = "李宗盛"
# 从词汇映射器中得到李宗盛的index
token_index = t.texts_to_sequences([token])[0][0] - 1
# 初始化一个全零的向量
zero_list = [0] * 6
zero_list[token_index] = 1
print(token, "的one-hot编码为: ", zero_list)
```

2、输出结果：

> 李宗盛 的one-hot编码为: [0, 0, 0, 0, 0, 1]

文本特征提取3：词袋(Bag of Word)模型

> John likes to watch movies. Mary likes too.
>
> John also likes to watch football games.

- Wikipedia给出了基于简单例子的解释:
- 根据示例中两句话里出现的单词, 先构建一个**字典** (dictionary):

{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10}

  - ***根据这个字典,*** 可以将上述两句话重新表达为下述两个向量:

$$[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]^T, \qquad [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]^T$$

其中,第i个元素表示字典中第i个单词在句子中出现的次数.

- 词汇鸿沟: ***没有表达单词在原来句子中出现的次序***
- 维度灾难：包含N个单词的词典，每个文档都可以被表示成为一个N维向量

语言模型（建模）

- 语言建模（Language modeling）：文本处理中，根据已经给出的词

  预测下一个词。

- 自然语言理解，即

  一个句子的可能性/合理性

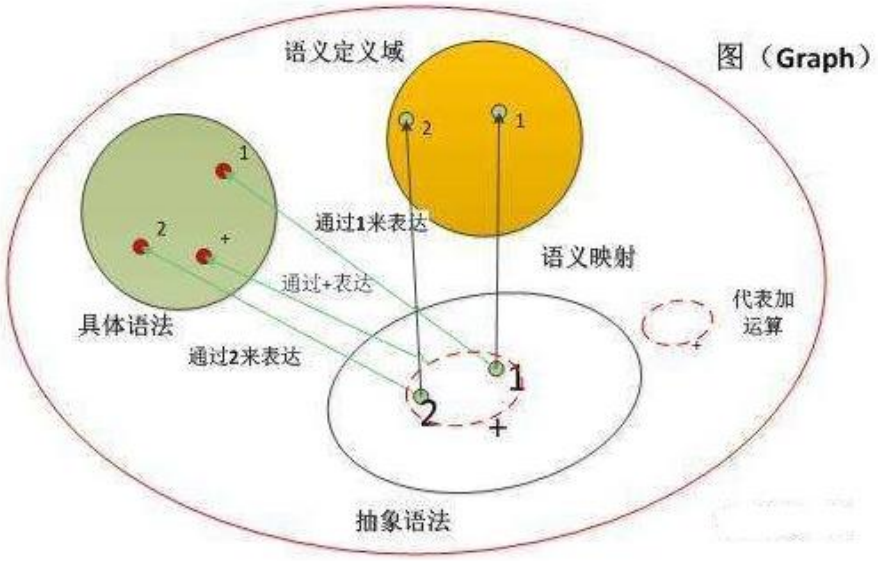
  - 在报那猫告做只!
  - 那只猫在作报告!
  - 那个人在作报告!

- 词袋(Bag of Word)模型（包括各种向量化方法）是 语言模型!

显式语言建模

- 借助Wikipedia等结构化知识库给出文本中每个词的概念(Conceptualization)分布
  - ① 根据给定词语从知识库预定义“概念”中找出相关的，构成概念集合
  - ② 采用诸如朴素贝叶斯方法，计算概念集合候选成员的得分
  - ③ 使用Laplace Smoothing等光滑技术过滤噪声，选取得分高的“概念”作为向量化结果

综上所述，即完成根据语境将文本映射为一个以概念为维度的向量。

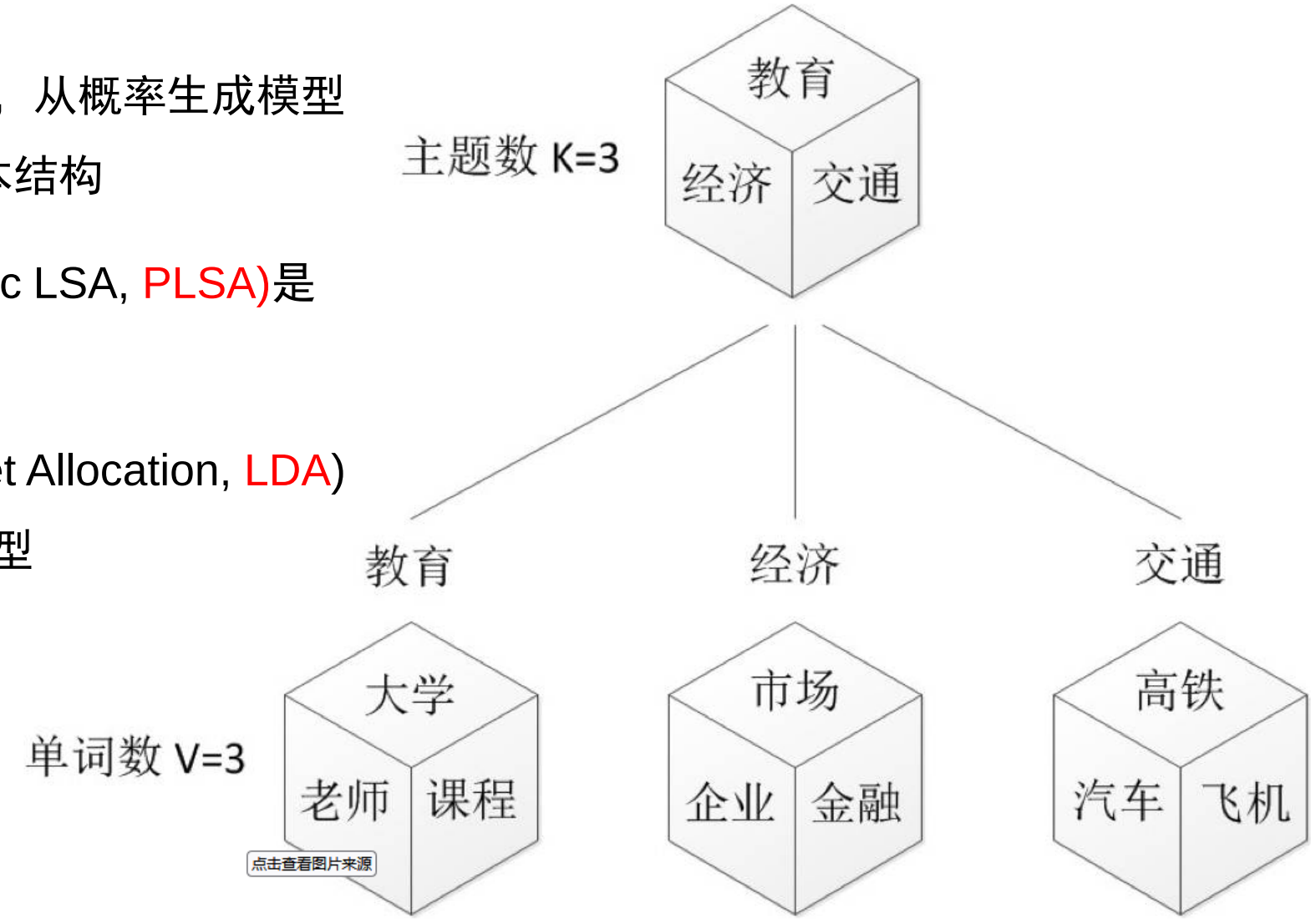


	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	点击查看图片来源	1

Figure 3.5 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

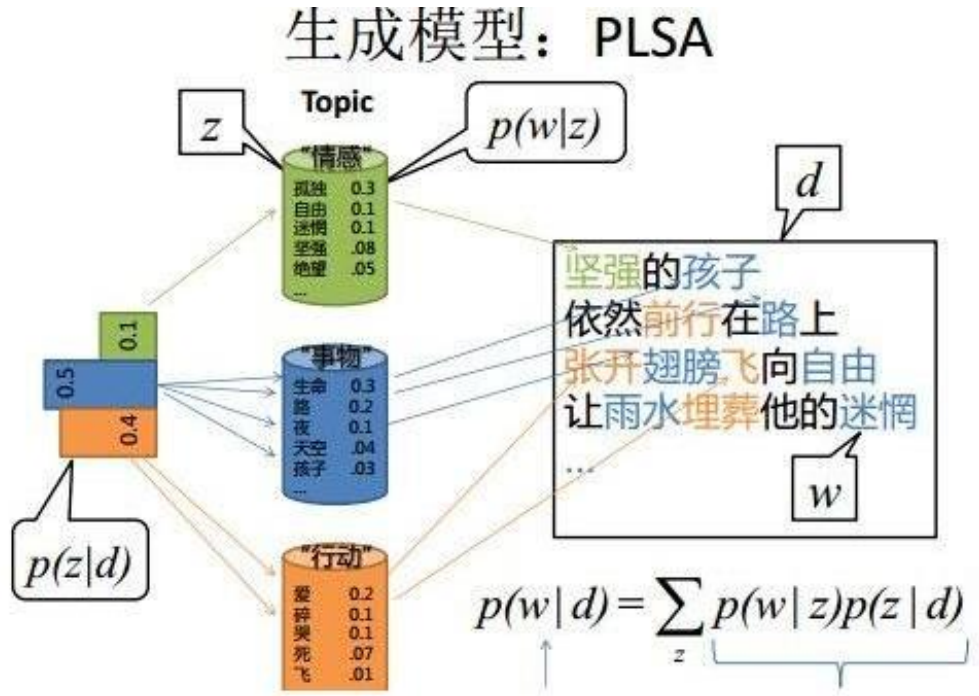
半显式语言建模

- 以“主题模型”类相关模型为主，从概率生成模型(Generative Model)角度分析文本结构
- 概率化潜在语义分析(Probabilistic LSA, PLSA)是最早的主题模型
- 潜在Dirichlet分布(Latent Dirichlet Allocation, LDA)是更为流行、更加完善的主题模型



概率化潜在语义分析(PLSA)

- 传统自然语言模型中，词语之间要求独立假设，但实际环境很难满足这个假设
- PLSA：用概率的观点，量化词语之间的关联性和语义模糊性
- 基本思想：
  - 认为词语再文档中的使用模式中存在隐含的潜在语义结构
  - 通过统计方法提取并量化潜在的语义结构，
  - 数据驱动生成词语-文本映射，如用SVD形成潜在语义索引空间



Index Words	Titles								
	T1	T2	T3	T4	T5	T6	T7	T8	T9
book			1	1					
dads						1			1
dummies		1						1	
estate							1		1
guide	1					1			
investing	1	1	1	1	1	1	1	1	1
market	1		1						
real							1		1
rich						2			1
stock	1		1					1	
value				1	1				

book	0.15	-0.27	0.04
dads	0.24	0.38	-0.09
dummies	0.13	-0.17	0.07
estate	0.18	0.19	0.45
guide	0.22	0.09	-0.46
investing	0.74	-0.21	0.21
market	0.18	-0.30	-0.28
real	0.18	0.19	0.45
rich	0.36	0.59	-0.34
stock	0.25	-0.42	-0.28
value	0.12	-0.14	0.23

*

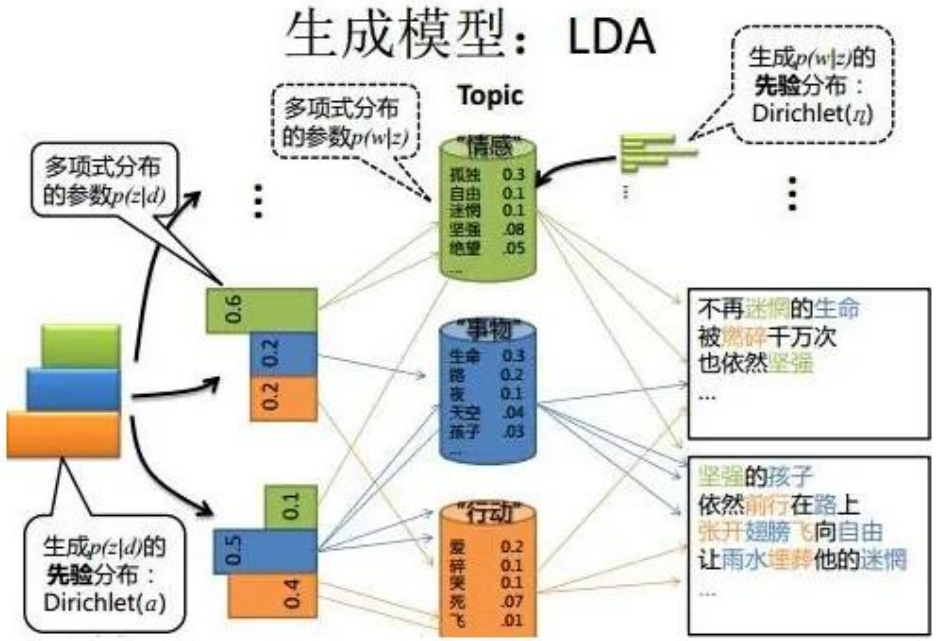
3.91	0	0
0	2.61	0
0	0	2.00

*

T1	T2	T3	T4	T5	T6	T7	T8	T9
0.35	0.22	0.34	0.26	0.22	0.49	0.28	0.29	0.44
-0.32	-0.15	-0.46	-0.24	-0.14	0.55	0.07	-0.31	0.44
-0.41	0.14	-0.16	0.25	0.22	-0.51	0.55	0.00	0.34

Latent Dirichlet Allocation,LDA

- **LDA模型是文本集合的生成概率模型**



  - 假设每个文本话题先验服从Dirichlet分布、生成话题单词的先验也服从Dirichlet分布
  - 先验分布的导入使LDA能够更好地应对话题模型学习中的过拟合现象

- **LDA的文本集合的生成过程**

1. 随机生成一个文本的话题分布
2. 在该文本的每个位置，依据该文本的话题分布随机生成一个话题
    1. 在该位置依据话题的单词分布随机生成一个单词
    2. 直至文本最后一个位置，生成完整文本
3. 重复以上过程生成所有文本

概率语言模型

一切都是概率!

- 设词串W $= w_1, w_2, \cdots, w_T$,，以$p(W)$表示该词串可能出现的概率，那么

$$p(W) = p(w_1)p(w_2|w_1)p(w_3|w_1w_2)\cdots p(w_T|w_1w_2\cdots w_{T-1})$$

- 更进一步：

$$p(w_1, w_2, \cdots, w_T) := \prod_i p(w_i|w_{i-1}, \cdots, w_1) \approx \prod_i p(w_i|w_{i-1}, \cdots, w_{i-n+1})$$

```
请输入两个字:中国
下个字/符号: 人 的可能性为: 0.0714
下个字/符号: 扶 的可能性为: 0.0556
下个字/符号: 的 的可能性为: 0.0397
下个字/符号: 社 的可能性为: 0.0397
下个字/符号: ， 的可能性为: 0.0397
```

https://github.com/surzia/go-n-gram

N-gram模型

- **1-gram模型（unigram）**

$$p(w_1, w_2, ..., w_n) \approx p(w_1)p(w_2)p(w_3)...p(w_n)$$

- **2-gram模型（bigram）**

$$p(w_1, w_2, ..., w_n) \approx p(w_1)p(w_2|w_1)p(w_3|w_2)...p(w_n|w_{n-1})$$

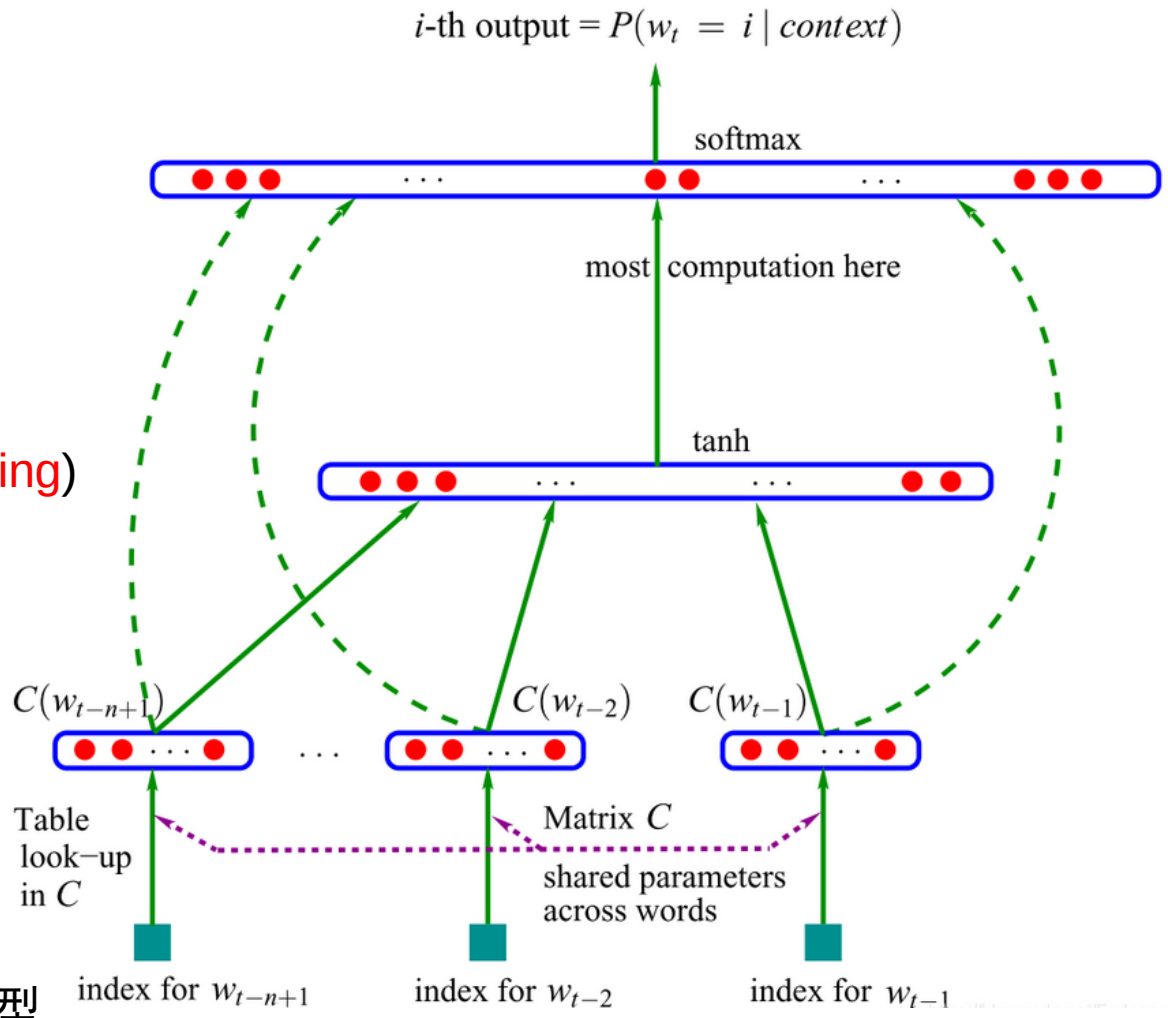
（此处w_{n-1}被称为历史词）

- **3-gram模型（trigram）**

$$p(w_1, w_2, ..., w_n) \approx p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)...p(w_n|w_{n-2}, w_{n-1})$$

神经语言模型

- 第一个神经语言模型： 前馈神经网络
  - Bengio, Y., Ducharme, R.， Vincent, P.,A Neural Probabilistic Language Model , NIPS, 2001.
  - 以n个先前的单词的表征向量(词向量，Word Embedding)做为输入,经过一个前馈神经网络输出预测结果
- 基于其他类型网络结构模型注记：
  - RNN和LSTM结构更适合于处理序列数据/文本
  - CNN在文本做了连续嵌入之后，也能用于构建文本模型



应用2：作诗词

- demo_poemtry.ipynb

白鹭窥鱼立， Egrets stood, peeping fishes. 青山照水开. Water was still, reflecting mountains. 夜来风不动， The wind went down by nightfall, 明月见楼台. as the moon came up by the tower.	满怀风月一枝春， Budding branches are full of romance. 未见梅花亦可人. Plum blossoms are invisible but adorable. 不为东风无此客， With the east wind comes Spring. 世间何处是前身. Where on earth do I come from?

本讲小结 - 11

1. 时间序列建模 – 流量预测
2. 文本特征提取 – AI写诗