机器学习2024
第4节

涉及知识点：
人工神经网络简介、人工神经网络发展史、普适逼近定理、反向传播算法介绍、反向传播算法示例、激活函数与损失函数、深度学习思想简介、梯度消失问题的解决方法、陷入局部最小的解决方法、深度学习中的正则化、卷积神经网络、循环神经网络、长短期记忆网络、长短期记忆网络使用案例

人工神经网络

张伟楠 – 上海交通大学

课程安排

参数化有监督学习

1. 机器学习概述
2. 线性模型
3. 双线性模型
4. 神经网络

非参数化有监督学习

5. 支持向量机
6. 决策树
7. 集成学习与森林模型

无监督学习部分

8. 概率图模型
9. 无监督学习

学习理论部分

10. 学习理论与模型选择

前沿话题部分

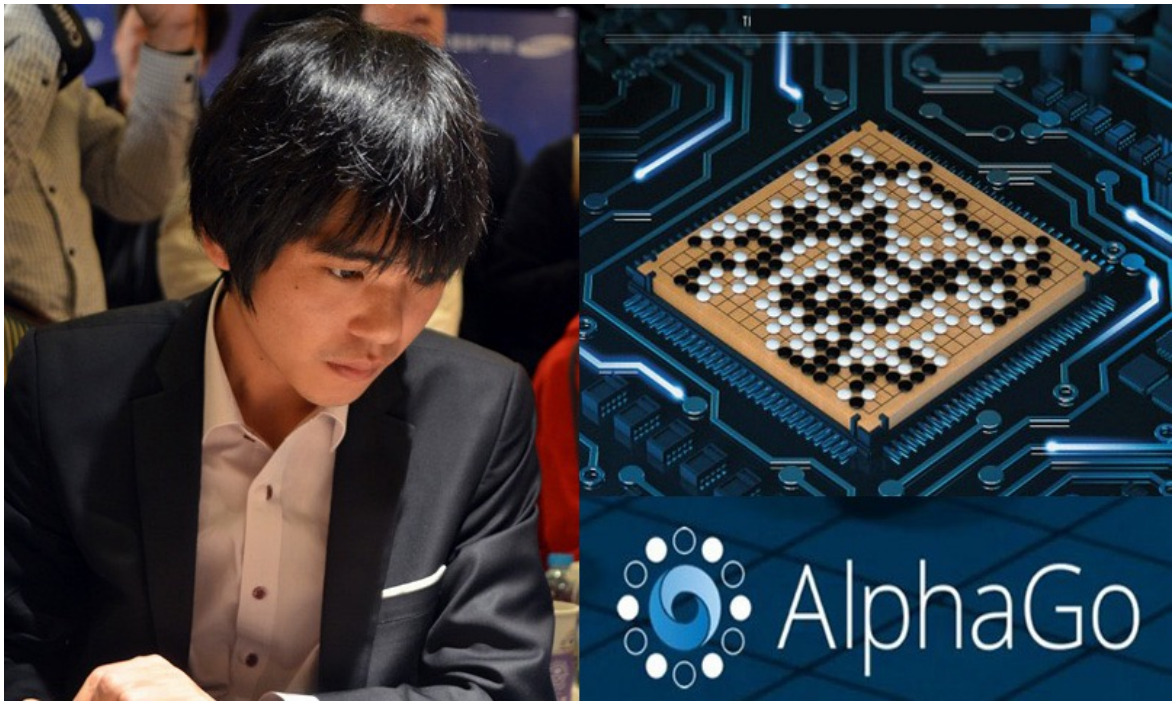
11. 迁移、多任务、元学习
12. System 1&2 机器意识

人工神经网络简介

张伟楠 - 上海交通大学

2016年AI界的爆炸新闻

- 2016年3月，AlphaGo战胜李世石（4-1）

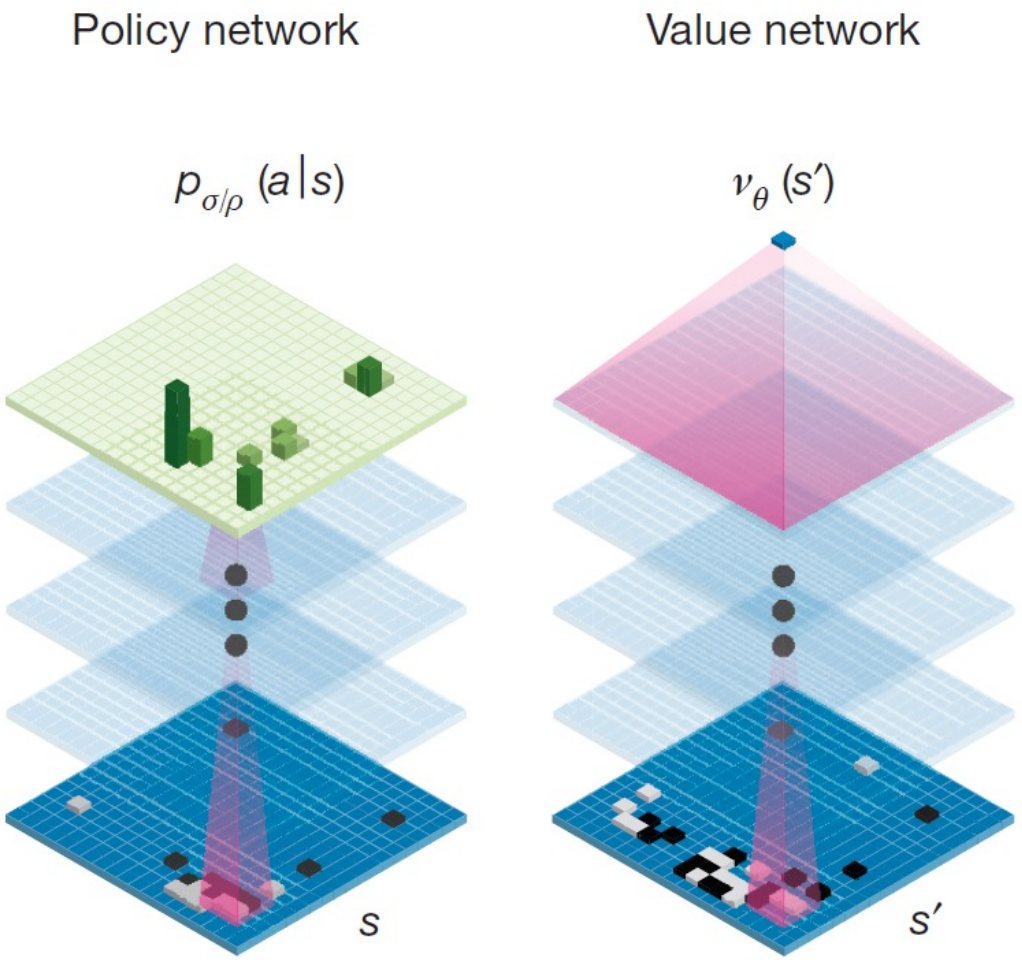


Rank	Name	Flag	Elo
1	Ke Jie		3628
2	AlphaGo		3598
3	Park Junghwan		3585
4	Tuo Jiaxi		3535
5	Mi Yuting		3534
6	Iyama Yuta		3525
7	Shi Yue		3522
8	Lee Sedol		3521
9	Zhou Ruiyang		3517
10	Shin Jinseo		3503
11	Chen Yaoye		3495
12	Lian Xiao		3493
13	Tan Xiao		3489
14	Kim Jiseok		3489
15	Choi Cheolhan		3482
16	Park Yeonghun		3482
17	Gu Zihao		3468
18	Fan Yunruo		3468
19	Huang Yunsong		3467
20	Li Qincheng		3465
21	Tang Weixing		3461
22	Lee Donghoon		3460
23	Lee Yeongkyu		3459
24	Fan Tingyu		3459
25	Tong Mengcheng		3447
26	Kang Dongyun		3442
27	Wang Xi		3439
28	Weon Seongjin		3439
29	Yang Dingxin		3439
30	Gu Li		3436

https://deepmind.com/research/alphago/
https://www.goratings.org/

AlphaGo中的机器学习

- 策略网络
  - 监督学习
    - 预测下一步人类移动的最佳结果
  - 强化学习
    - 学习去选择下一步移动去最大化获胜率
- 价值网络
  - 在给定当前状态的情况下获胜的期望
- 通过（深度）神经网络实现



2022年AI界的爆炸新闻

- 2022年11月， ChatGPT被推出，可以完成各类对话指令任务，推出2个月积累1亿月活用户

52 try to solve the programming min 2x^2 + 3x

The minimum value of the expression 2x^2 + 3x can be found by taking the derivative of the expression and setting it equal to zero. The critical points of the expression represent either the minimum or maximum values of the function.

The derivative of the expression 2x^2 + 3x is:

d/dx (2x^2 + 3x) = 4x + 3

Setting this equal to zero and solving for x, we get:

4x + 3 = 0
x = -3/4

We can then substitute this value back into the original expression to find the minimum value:

2x^2 + 3x = 2(-3/4)^2 + 3(-3/4) = -9/2

So, the minimum value of the expression 2x^2 + 3x is -9/2.

如何用golang写一段gzip压缩文件的代码

下面是一段用Go语言编写的代码，它可以用来压缩一个文件：

Copy code

```
package main

import (
    "compress/gzip"
    "os"
    "log"
)

func main() {
    // 打开要压缩的文件
    file, err := os.Open("test.txt")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    // 创建一个gzip压缩器
    gzipper, err := gzip.NewWriterLevel(file, gzip.BestCompression)
    if err != nil {
        log.Fatal(err)
    }
    defer gzipper.Close()

    // 将文件写入压缩器，完成压缩
    if _, err := gzipper.Write([]byte("Hello, world!")); err != nil {
        log.Fatal(err)
    }
}
```

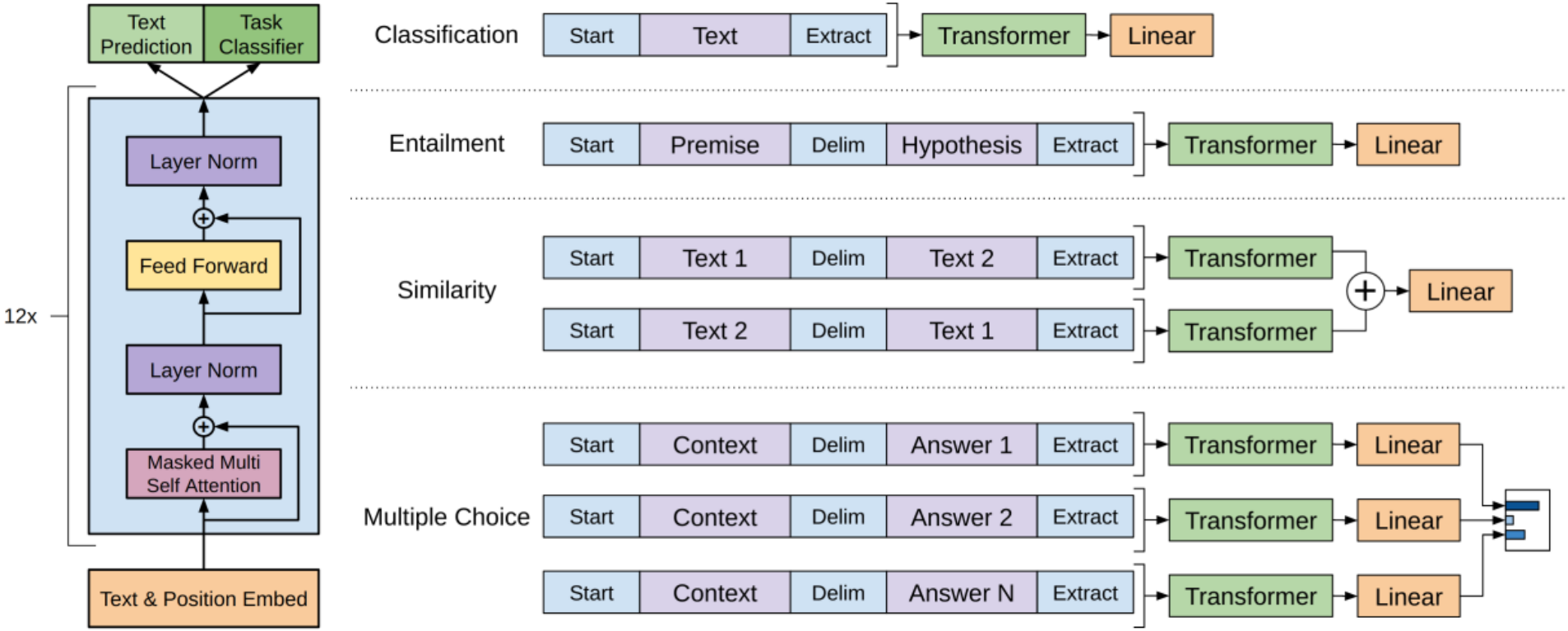
在上面的代码中，我们首先打开要压缩的文件，然后创建一个gzip压缩器。接着，我们使用`Write`方法将文件写入压缩器，完成压缩。最后，我们调用`Close`方法关闭压缩器，完成压缩操作。

注意：上面的代码仅供参考，并不能保证在所有情况下都能正常工作。如果你想要使用它，需要根据自己的需求进行相应的更改。

https://chat.openai.com/

2022年AI界的爆炸新闻

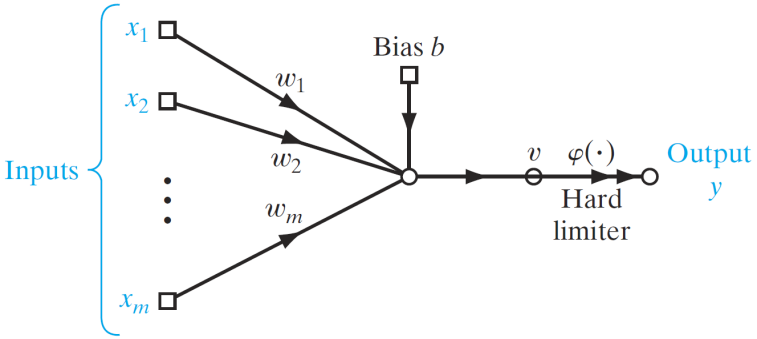
- ChatGPT背后的GPT深度神经网络架构和数据形式



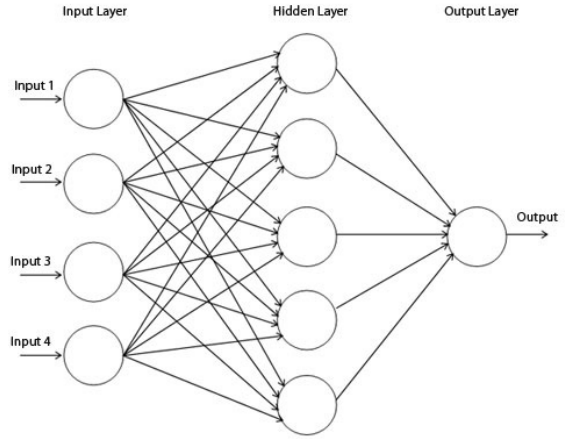
https://chat.openai.com/

人工神经网络

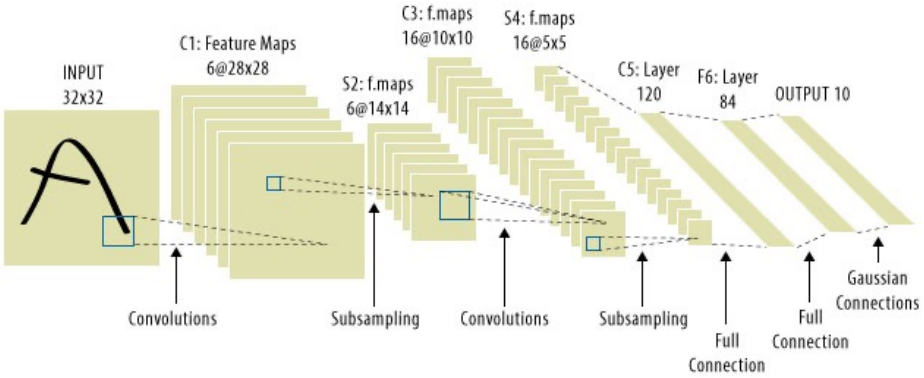
- 神经网络是深度学习的基础



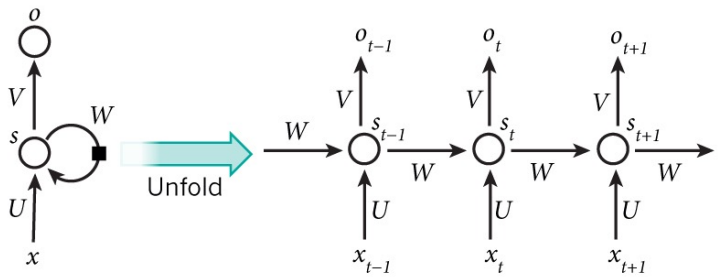
感知机



多层感知机



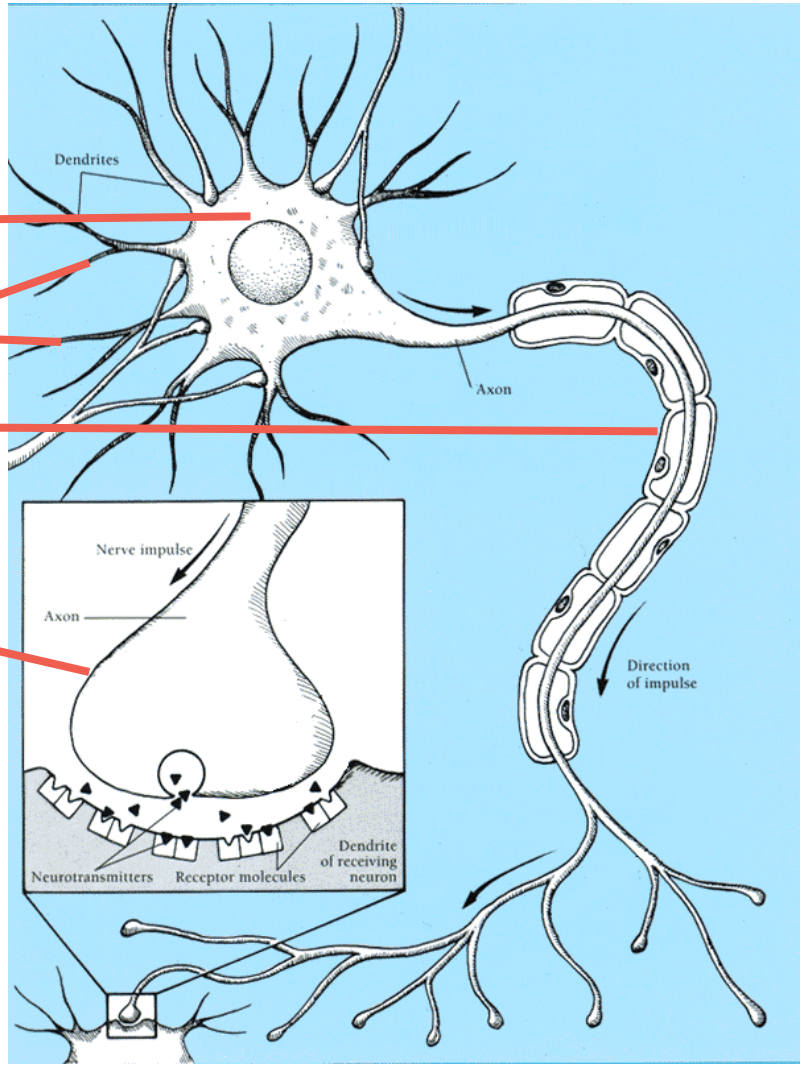
卷积神经网络



循环神经网络

真正的神经元

- 细胞结构
  - 细胞体
  - 树突
  - 轴突
  - 突触末梢

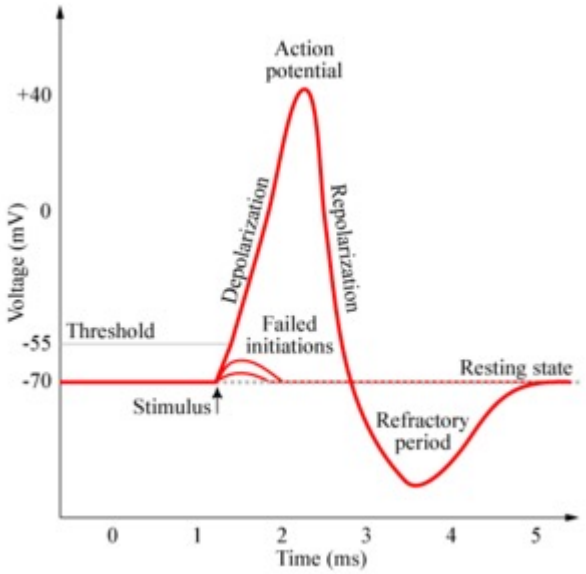


Slides credit: Ray Mooney

神经通信

- 细胞膜间的电位表现出的电信号称为动作电位
- 电信号从细胞体中产生，沿着轴突往下传，并且导致突触末梢释放神经递质介质
- 介质通过化学扩散从突触传递到其他神经元的树突
- 神经递质可以是兴奋的或者是抑制的
- 如果从其他神经元来的神经递质是兴奋的

  并且超过某个阈值，它就会触发一个运动电位



Slides credit: Ray Mooney

真正的神经元学习

- 突触通过经验改变尺寸和连接强度
- Hebbian学习：如果两个连接在一起的神经元同时被激发，那么他们之间的连接会变强
- “Neurons that fire together, wire together.”
- 这些都推动了人工神经网络的研究

Slides credit: Ray Mooney

人工神经网络发展史

张伟楠 - 上海交通大学

人工神经网络简史

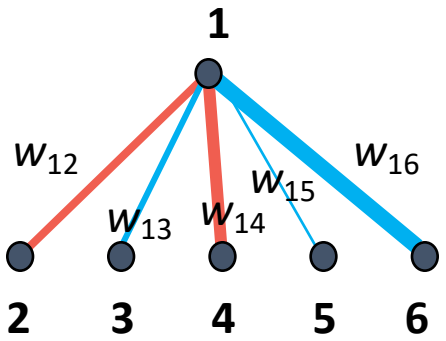
- 第一波浪潮
  - 1943年McCulloch和Pitts提出McCulloch-Pitts神经元模型
  - 1958年Rosenblatt提出了简单的单层神经网络，现在被称为感知机
  - 1969年Minsky和Papert所作的书Perceptrons论证了单层感知机的局限性，整个领域几乎进入冬眠期
- 第二波浪潮
  - 1986年对于多层感知机的反向传播学习算法被提出和宣扬，整个领域再次起飞
- 第三波浪潮
  - 2006年深度（神经网络）学习变得流行
  - 2012年在许多应用中都取得了重大突破
  - 2018年Hinton、LeCun和Bengio三人获得图灵奖

人工神经网络

McCulloch-Pitts神经元模型 [1943]

- 将网络建模成一个图，单元作为节点，突触连接作为从节点i到节点j的加权边，权重为w_{ji}
- 将单元的网络输入建模为

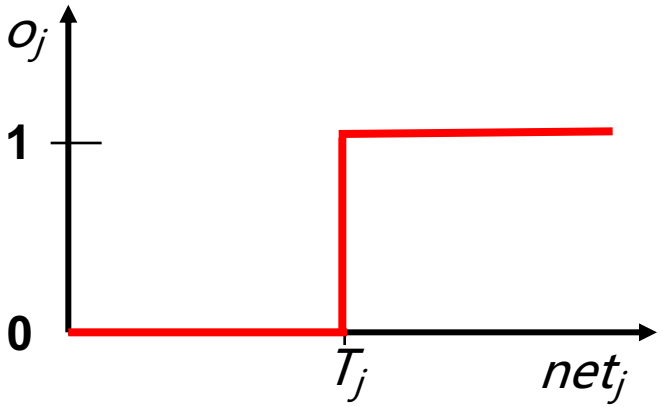
$$net_j = \sum_i w_{ji} o_i$$



- 单元输出为

$$o_j = \begin{cases} 0 & \text{if net}_j < T_j \\ 1 & \text{if net}_j \geq T_j \end{cases}$$

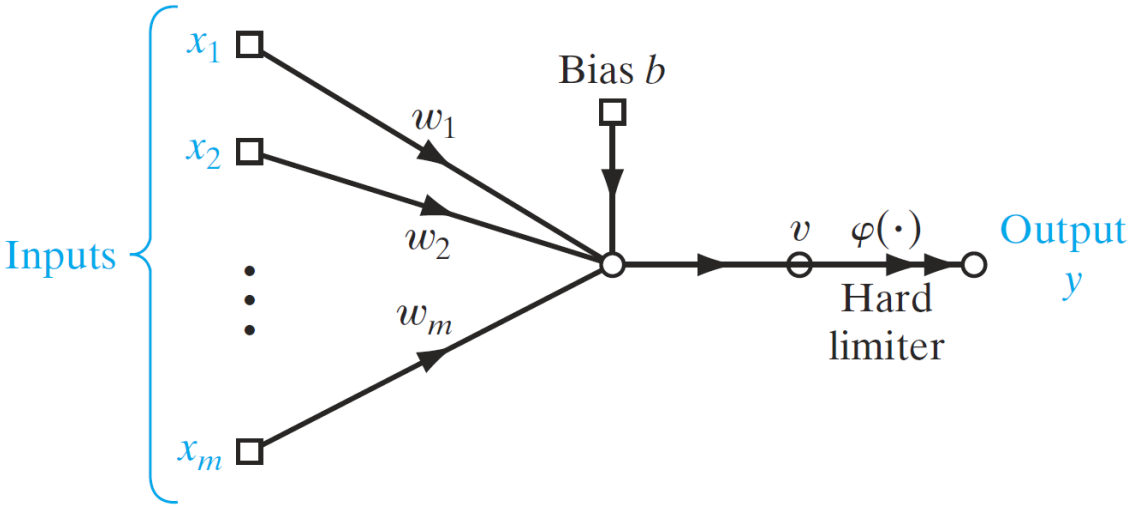
  - (T_j是单元j的阈值)



Slides credit: Ray Mooney

感知机模型

Rosenblatt的单层感知机 [1958]



- Rosenblatt [1958] 进一步提出感知机作为第一个在“老师”指导下进行学习的模型（即监督学习）
- 专注在如何找到合适的用于二分类任务的权重w_m
  - $y = 1$：类别1
  - $y = -1$：类别2

预测

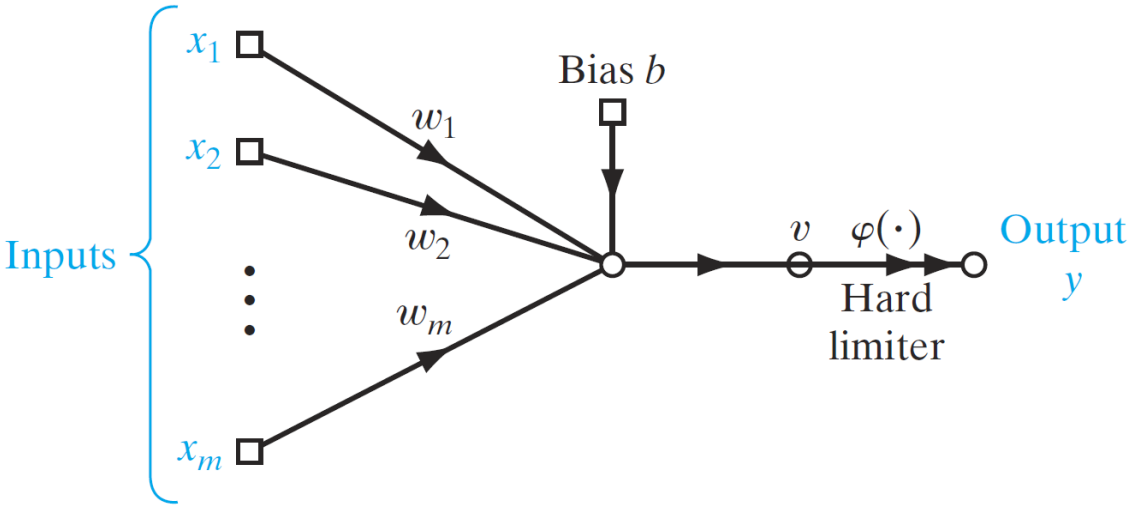
$$\hat{y} = \varphi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

激活函数

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

训练感知机

Rosenblatt的单层感知机 [1958]



预测

$$\hat{y} = \varphi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

激活函数

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

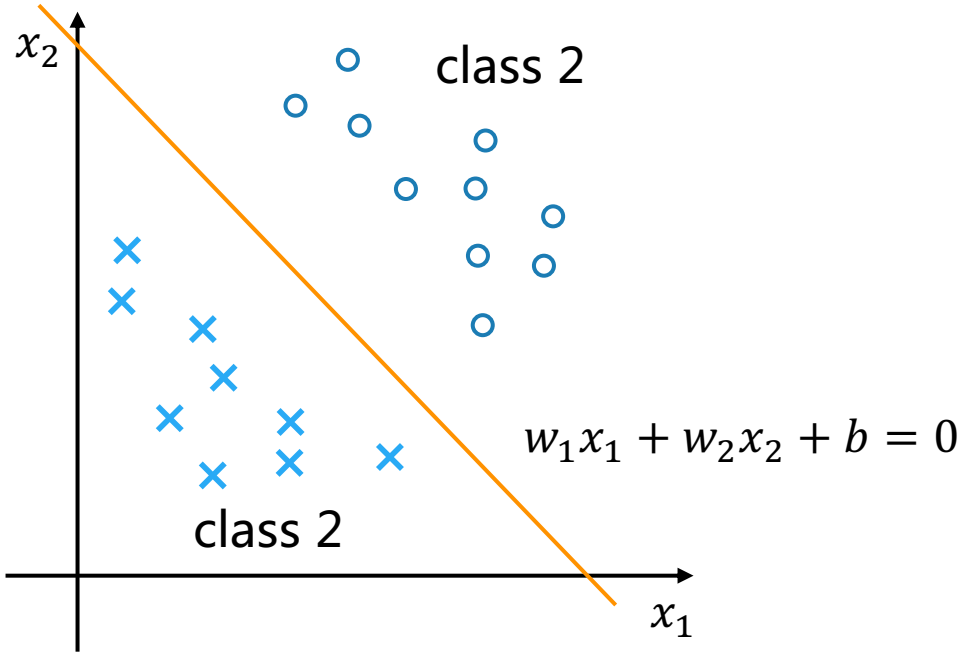
- 训练

$$w_i = w_i + \eta(y - \hat{y})x_i$$
$$b = b + \eta(y - \hat{y})$$

- 下列规则等价：
  - 如果输出正确，则不进行操作
  - 如果输出高了，降低正输入的权重
  - 如果输出低了，增加正输入的权重

感知机性质

Rosenblatt的单层感知机 [1958]

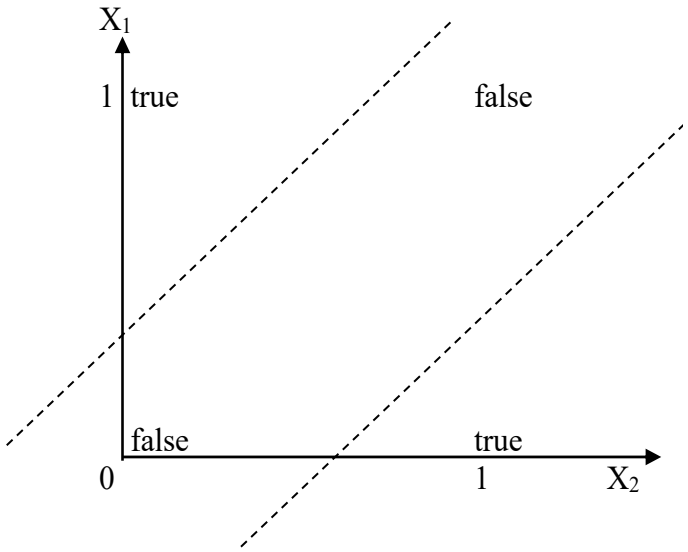


- Rosenblatt证明了学习算法的收敛性，如果两个类别是线性可分的（即不同的模式位于超平面的两侧）
- 许多人希望这种机器能成为人工智能的基础

感知机属性

异或问题

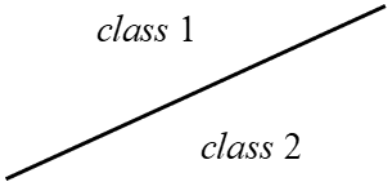
Input x		Output y
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

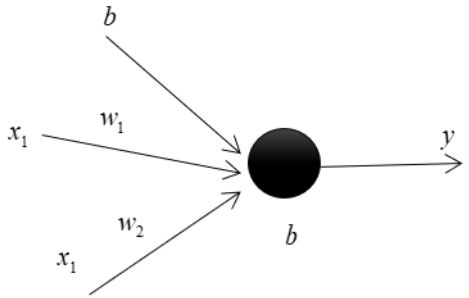


- Minsky和Papert [1969] 表明Rosenblatt的单层感知机甚至不能解决一些相当基本的计算（XOR问题）
- Rosenblatt认为如果添加更多的单元层能够克服这种限制，但是没有已知的算法来获得权重
- 由于缺乏学习算法，人们离开神经网络范式近20年

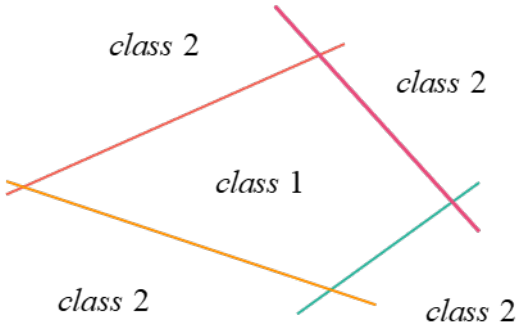
隐藏层和反向传播（1986~）

- 添加隐藏层（内部表示）允许去学习一个不局限于线性可分情况的映射

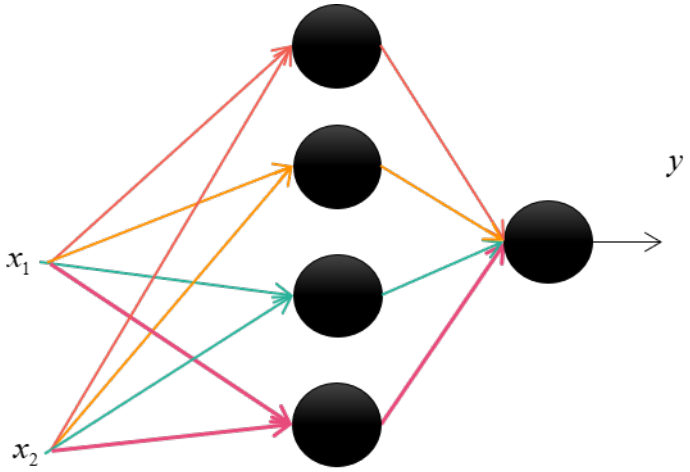




决策边界：$x_1 w_1 + x_2 w_2 + b = 0$



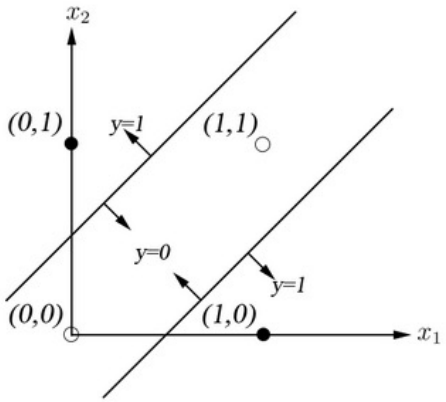
每个隐藏节点负责实现凸区域的一条边界线



隐藏层和反向传播（1986~）

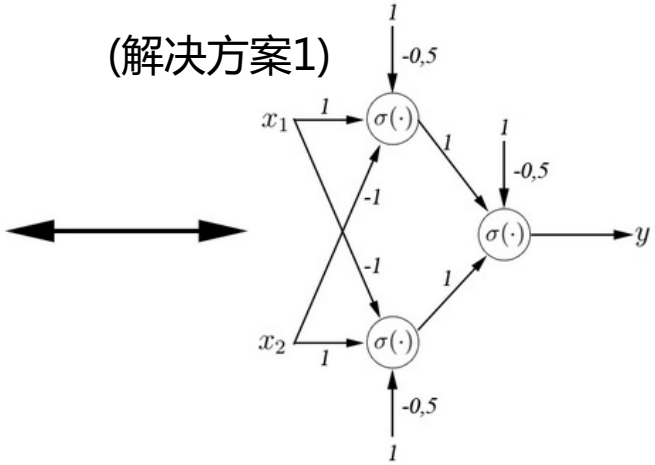
- 解决方案不唯一

Input x		Output y
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0



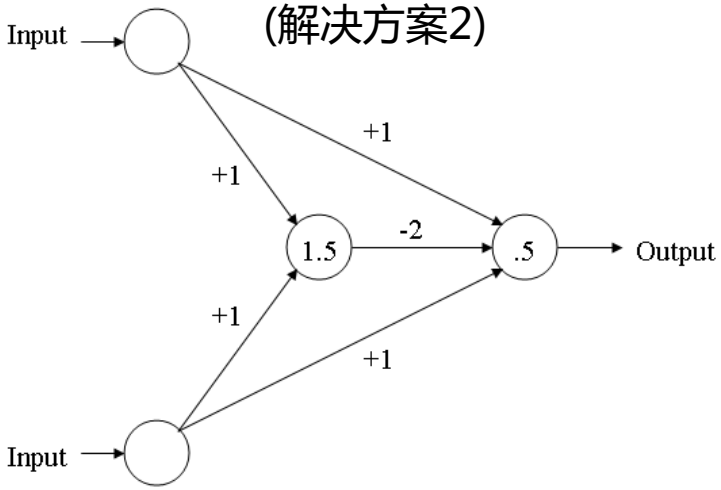
需要两条线来相应地划分样本空间

(解决方案1)



符号激活函数

(解决方案2)



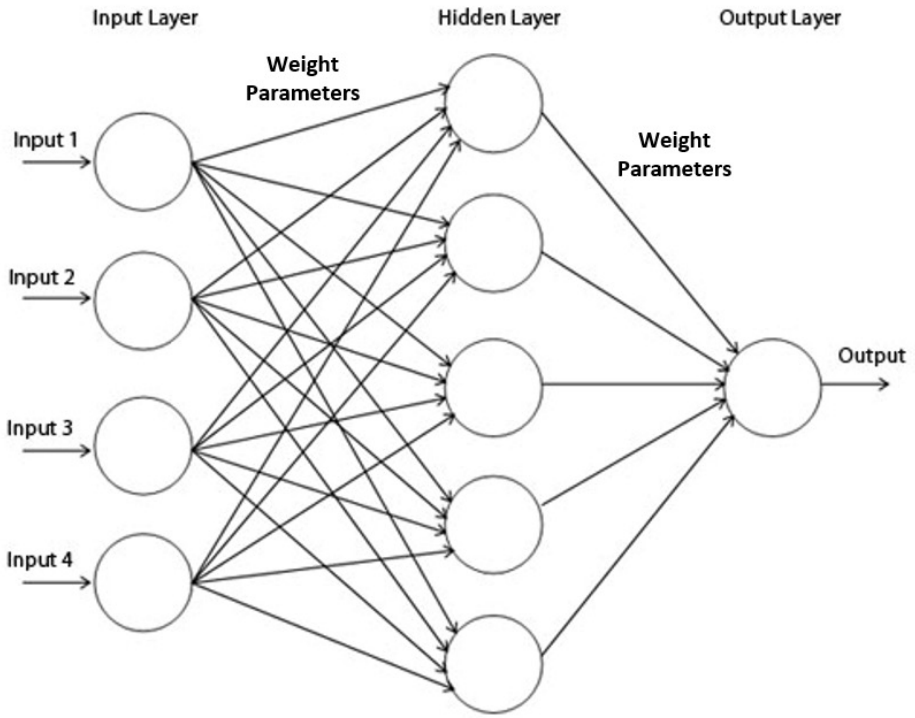
圆圈中的数字是一个阈值

http://www.cs.stir.ac.uk/research/publications/techreps/pdf/TR148.pdf
http://recognize-speech.com/basics/introduction-to-artificial-neural-networks

隐藏层和反向传播（1986~）

前馈

- 消息从输入节点出发，经过隐藏节点（如果有的话）到输出节点
- 网络中没有圈或者循环

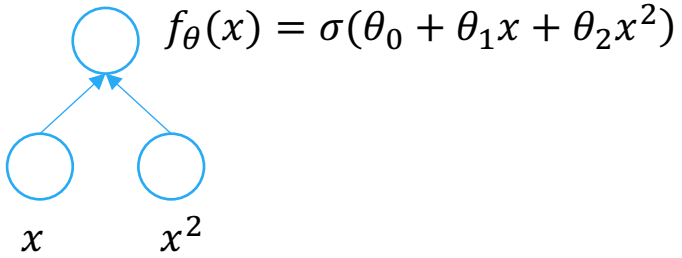


双层前馈神经网络

单层/多层计算

单层函数

$$f_\theta(x) = \sigma(\theta_0 + \theta_1 x + \theta_2 x^2)$$

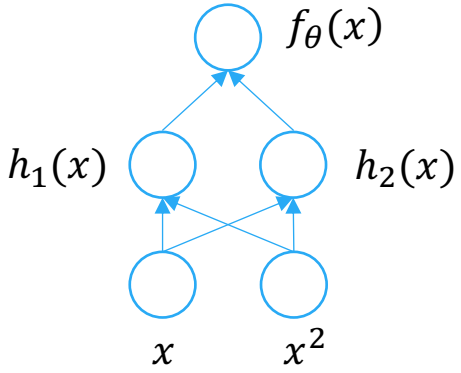


多层函数

$$h_1(x) = tanh(\theta_0 + \theta_1 x + \theta_2 x^2)$$

$$h_2(x) = tanh(\theta_3 + \theta_4 x + \theta_5 x^2)$$

$$f_\theta(x) = f_\theta(h_1(x), h_2(x)) = \sigma(\theta_6 + \theta_7 h_1 + \theta_8 h_2)$$



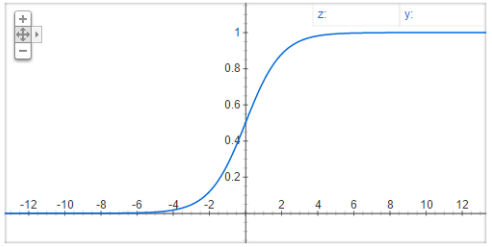
- 具有非线性激活函数

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$

非线性激活函数

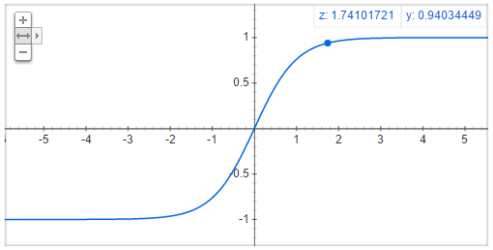
- Sigmoid激活函数

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



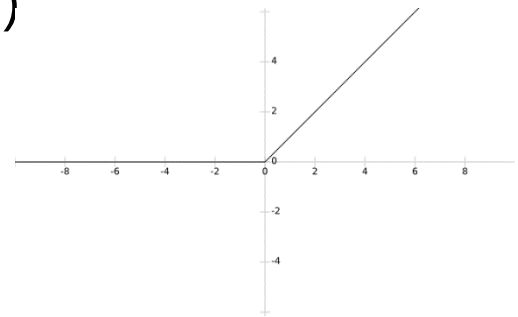
- Tanh激活函数

$$tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



- 线性整流函数（Rectified Linear Unit, ReLU)

$$ReLU(z) = max(0, z)$$

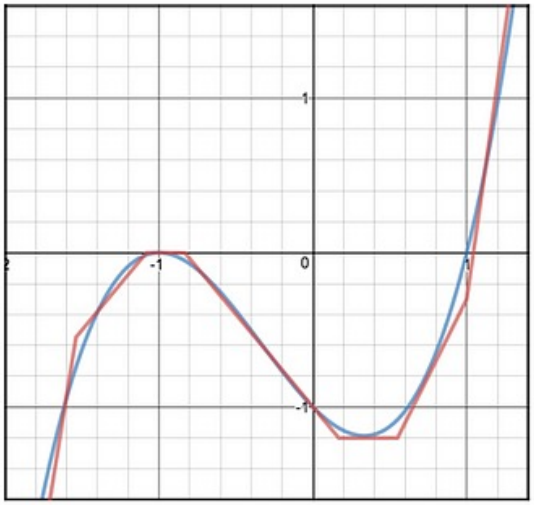


普适逼近定理

张伟楠- 上海交通大学

普适逼近定理

- 一个具有至少一层的隐藏层的前馈神经网络，并且隐藏层包含有限数量的神经元（即多层感知机），它可以以任意精度逼近任意一个定义在$\mathbb{R}^n$的闭集里的连续函数。
  - 前提是这个前馈神经网络的激活函数满足某些性质
    - 例如Sigmoid函数，Tanh函数，ReLU函数



$$
\begin{aligned}
n_1(x) &= Relu(-5x - 7.7) \\
n_2(x) &= Relu(-1.2x - 1.3) \\
n_3(x) &= Relu(1.2x + 1) \\
n_4(x) &= Relu(1.2x - .2) \\
n_5(x) &= Relu(2x - 1.1) \\
n_6(x) &= Relu(5x - 5)
\end{aligned}
$$

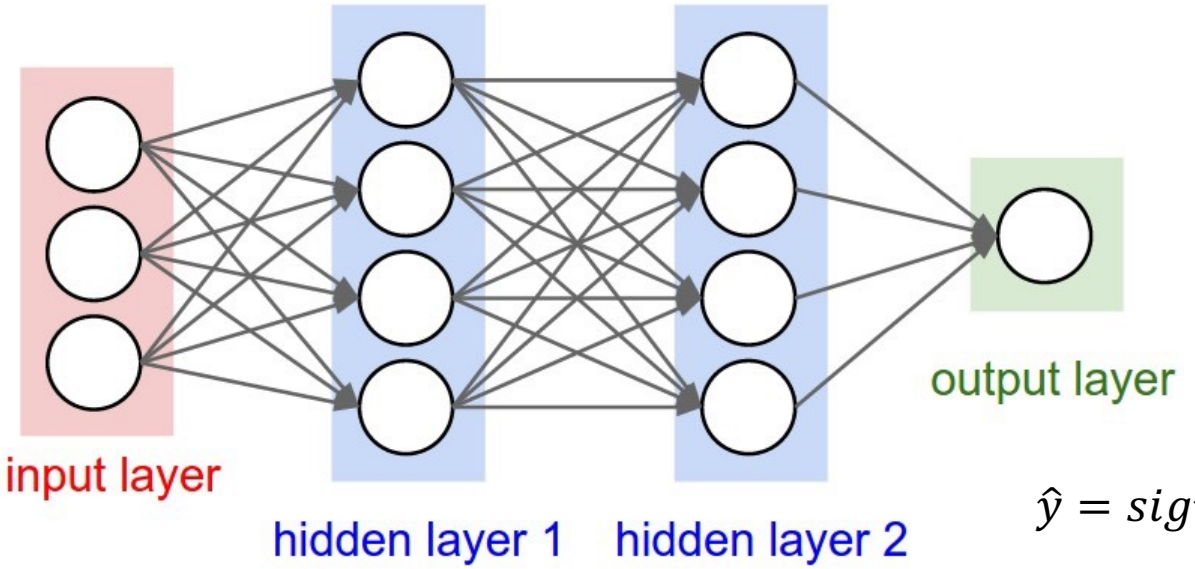
$$
\begin{aligned}
Z(x) = &-n_1(x) - n_2(x) - n_3(x) \\
&+ n_4(x) + n_5(x) + n_6(x)
\end{aligned}
$$

[Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.]

https://towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6

普适逼近定理

- 多层感知器近似于紧子集上的任何连续函数。



$$\hat{y} = sigmoid(\boldsymbol{W}_3 \boldsymbol{l}_2 + b_3)$$

$$l_1 = tanh(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) \qquad l_2 = tanh(\boldsymbol{W}_2 l_1 + \boldsymbol{b}_2)$$

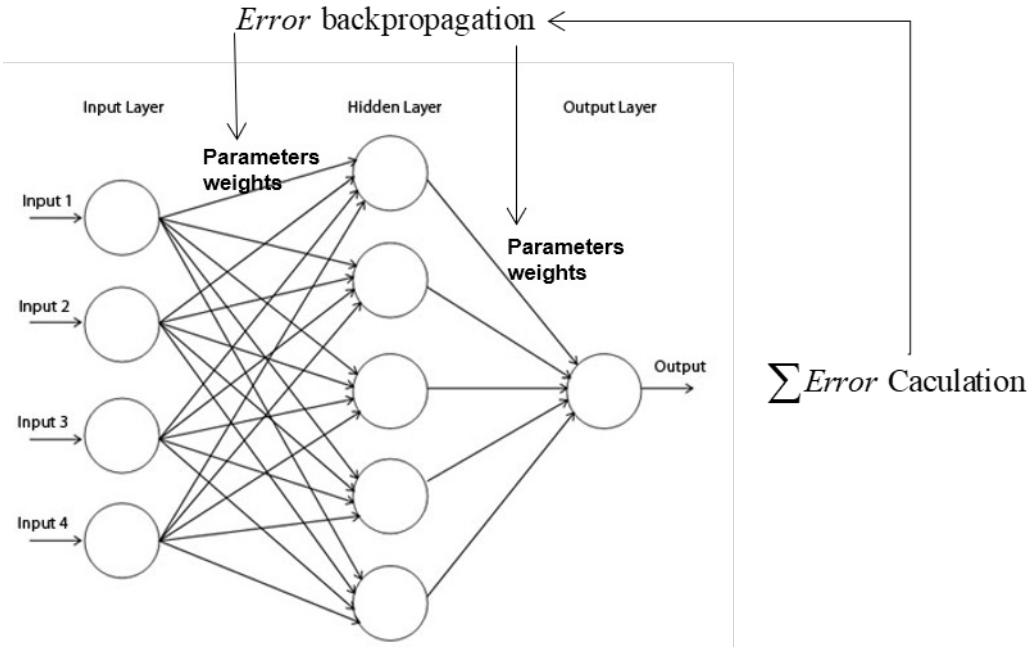
$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

反向传播算法介绍

张伟楠 - 上海交通大学

隐藏层和反向传播算法（1986~）

- 多层神经网络的有效算法之一是反向传播算法
- 它在1986年被重新提出，神经网络重新变得流行

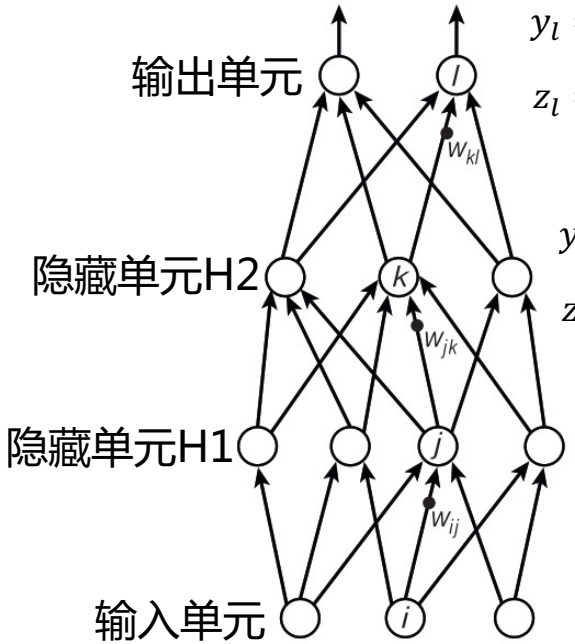


注意：反向传播算法最开始在1974年被Werbos[1]发现，之后在1985年左右被Rumelhart, Hinton, Williams[2] 以及Parker[3] 分别独立重新发现

[1] Werbos, Paul John (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University.
[2] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536.
[3] Parker, D. B. (1985). "Learning Logic," Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA.

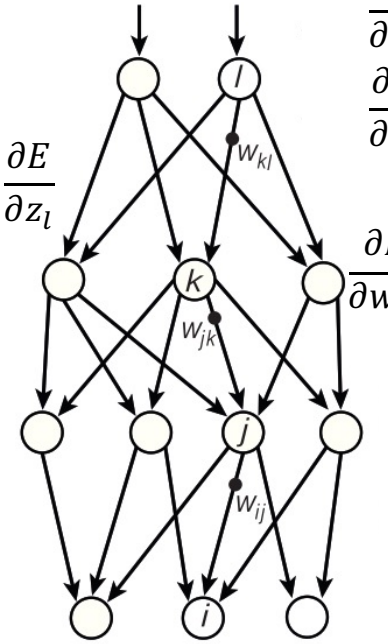
通过反向传播学习神经网络

输入数据逐层计
算得到输出



$$y_l = f(z_l)$$
$$z_l = \sum_{k \in H} w_{kl} y_k$$
$$y_k = f(z_k)$$
$$z_k = \sum_{j \in H1} w_{jk} y_j$$
$$y_j = f(z_j)$$
$$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$$

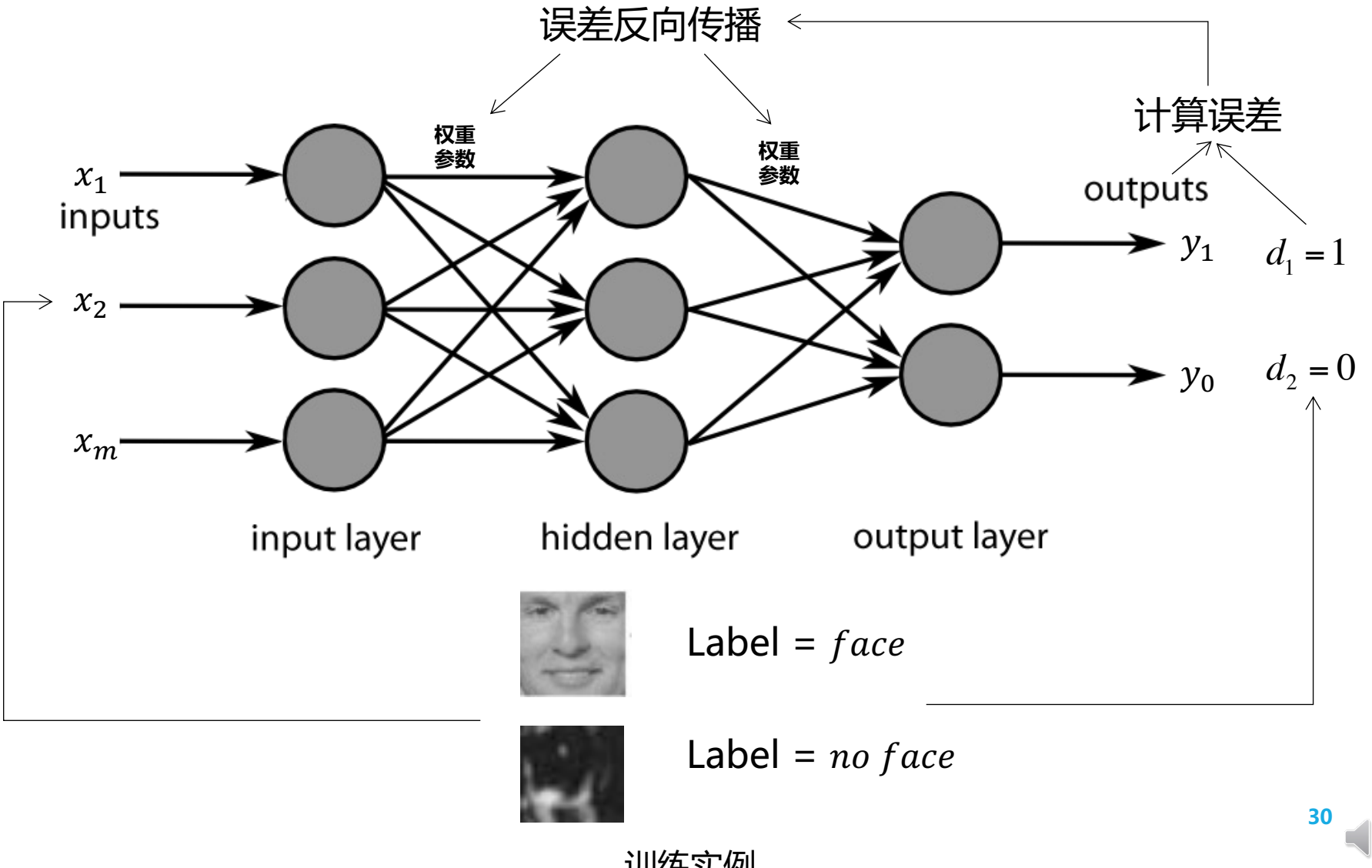
将输出与正确答案比较得到
误差，然后反向求导数



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$
$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$
$$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{Out}} w_{kl}\frac{\partial E}{\partial z_l}$$
$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} y_j$$
$$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk}\frac{\partial E}{\partial z_k}$$
$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

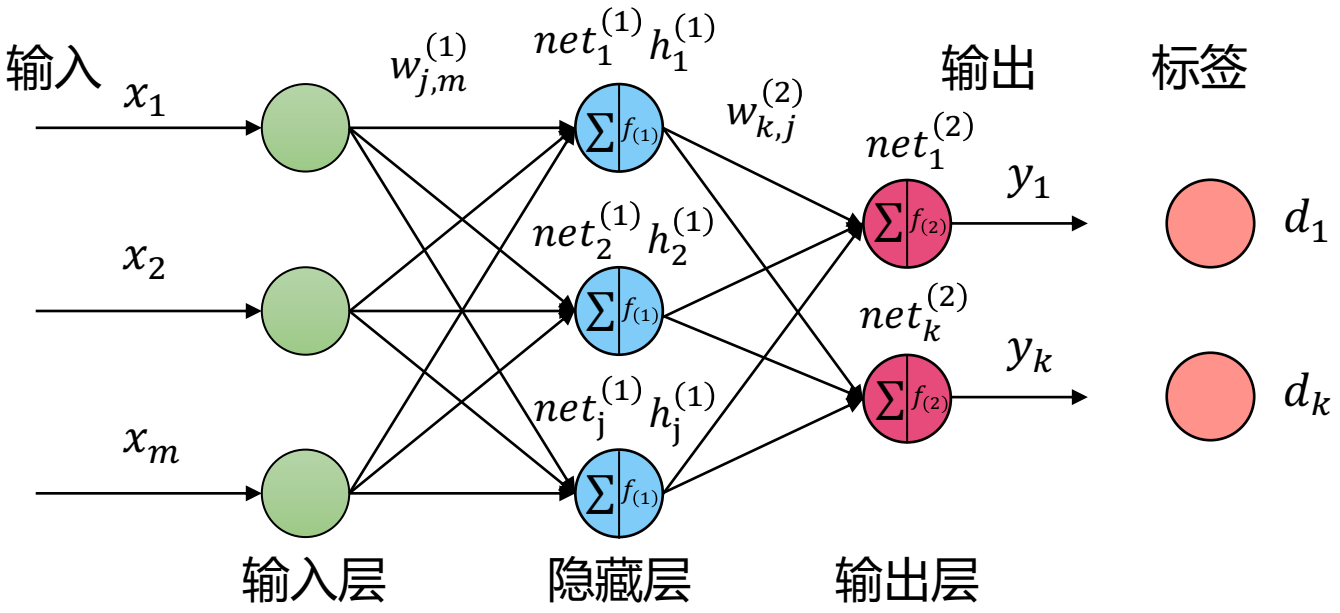
[LeCun, Bengio and Hinton. Deep Learning. Nature 2015.]

通过反向传播学习神经网络



训练实例

预测输出



双层前馈神经网络

前馈预测

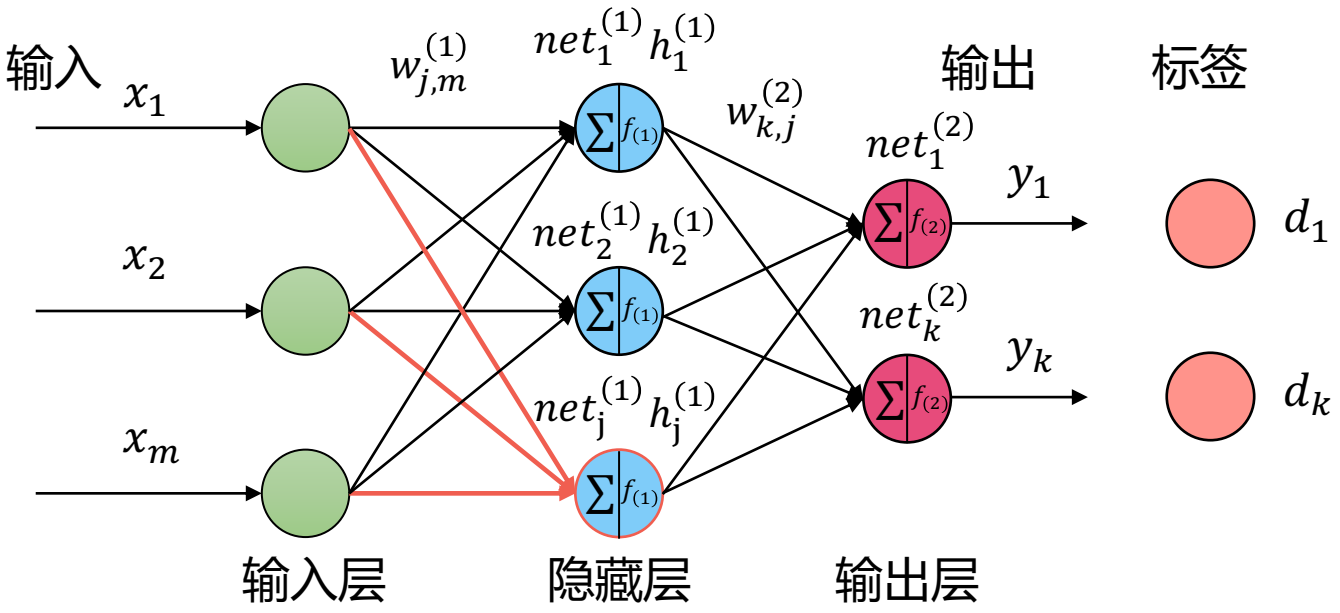
$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \longrightarrow y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right)$$

其中，

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

预测输出



双层前馈神经网络

前馈预测

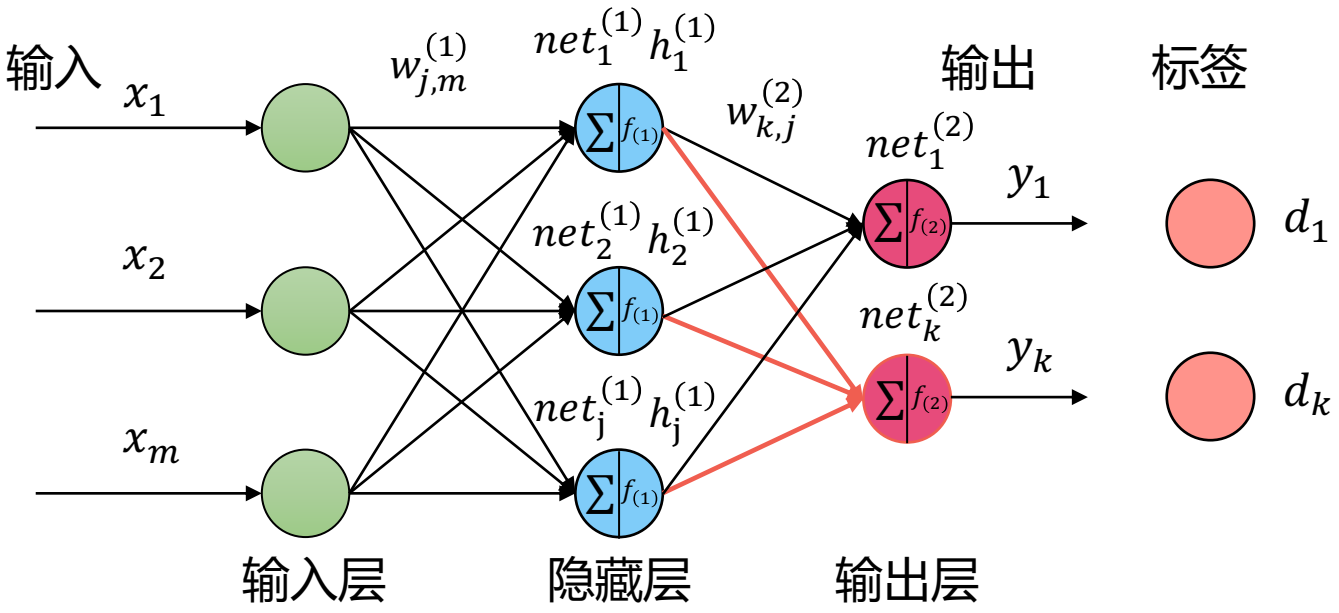
$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \qquad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(2)} h_j^{(1)})$$

$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

其中，

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

预测输出



双层前馈神经网络

前馈预测

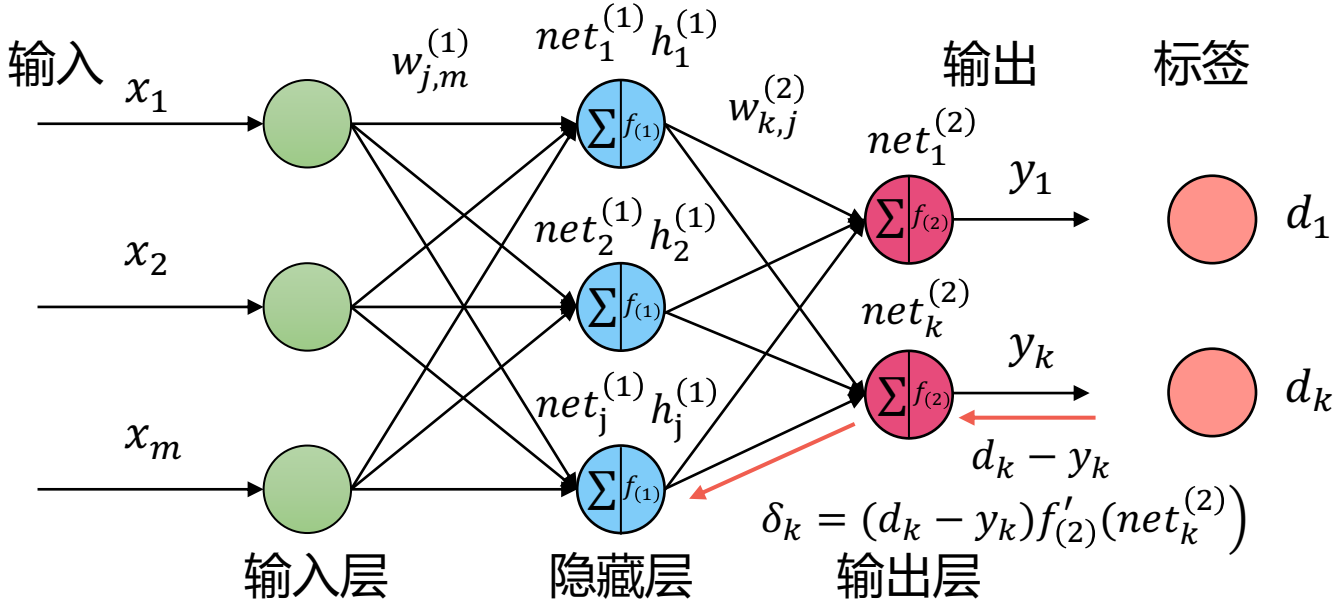
$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \longrightarrow y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(2)} h_j^{(1)}) \longrightarrow y_k$$

其中，

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

反向传播进行参数学习



双层前馈神经网络

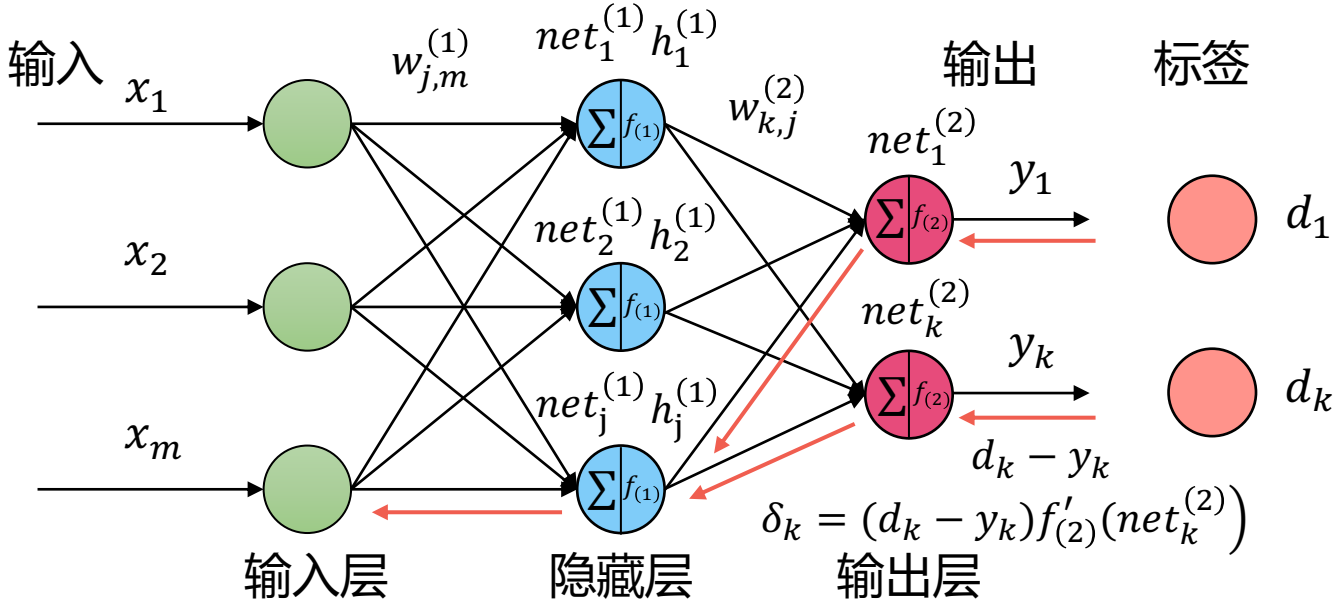
$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

$$w_{k,j}^{(2)} = w_{k,j}^{(2)} + \Delta w_{k,j}^{(2)}$$

$$\Delta w_{k,j}^{(2)} = \eta Error_k \, Output_j = \eta \delta_k h_j^{(1)} \longleftarrow E(W) = \frac{1}{2}\sum_k (y_k - d_k)^2$$

$$\Delta w_{k,j}^{(2)} = -\eta \frac{\partial E(W)}{\partial w_{k,j}^{(2)}} = -\eta (y_k - d_k) \frac{\partial y_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{k,j}^{(2)}} = \eta (d_k - y_k) f'_{(2)}(net_k^{(2)}) h_j^{(1)} = \eta \delta_k h_j^{(1)}$$

反向传播进行参数学习



双层前馈神经网络

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

$$E(W) = \frac{1}{2}\sum_k (y_k - d_k)^2$$

$$\Delta w_{j,m}^{(1)} = \eta Error_j Out_m = \eta \delta_j x_m$$

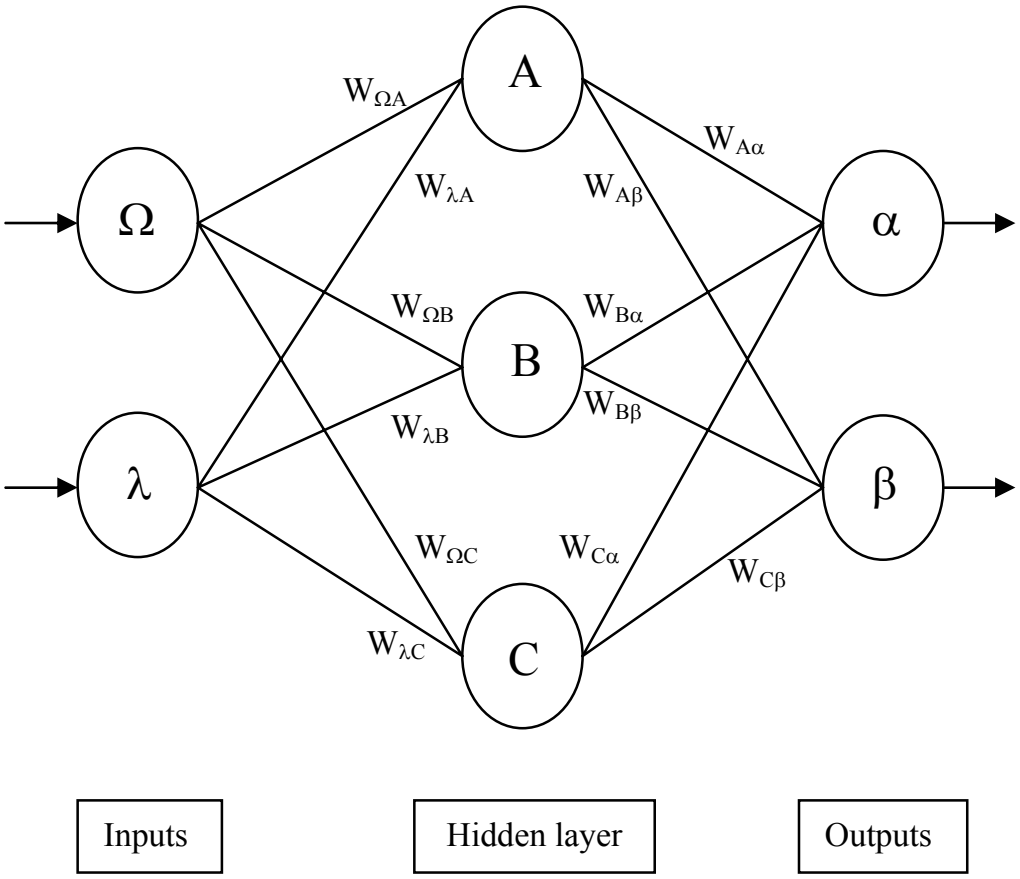
$$w_{j,m}^{(1)} = w_{j,m}^{(1)} + \Delta w_{j,m}^{(1)}$$

$$\Delta w_{j,m}^{(1)} = -\eta \frac{\partial E(W)}{\partial w_{j,m}^{(1)}} = -\eta \frac{\partial E(W)}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = \eta \sum_k (d_k - y_k) f'_{(2)}(net_k^{(2)}) w_{k,j}^{(2)} x_m f'_{(1)}(net_j^{(1)}) = \eta \delta_j x_m$$

反向传播算法示例

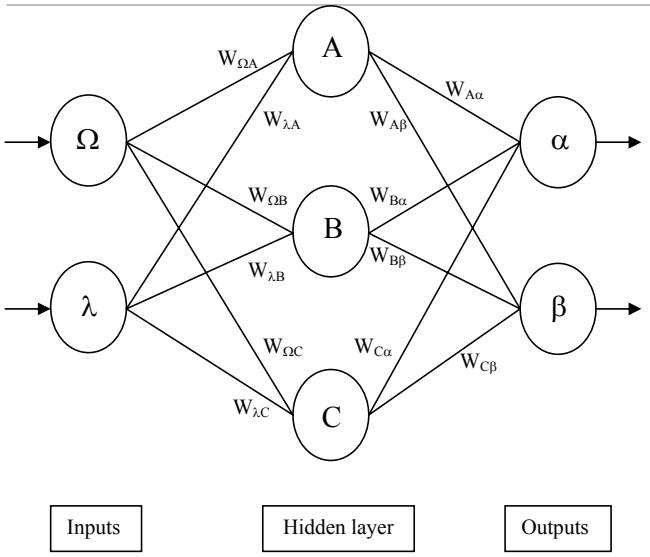
张伟楠- 上海交通大学

一个反向传播的例子



https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

一个反向传播的例子



- 计算输出节点的误差 $\delta_k = (d_k - y_k) f'_{(2)}(net_k^{(2)})$

$\delta_\alpha = out_\alpha\,(1 - out_\alpha)\,(Target_\alpha - out_\alpha)$
$\delta_\beta = out_\beta\,(1 - out_\beta)\,(Target_\beta - out_\beta)$

- 改变输出层的权重 $\Delta w_{k,j}^{(2)} = \eta Error_k\, Output_j = \eta \delta_k h_j^{(1)}$

$W^{+}_{A\alpha} = W_{A\alpha} + \eta\delta_\alpha\, out_A$　　$W^{+}_{A\beta} = W_{A\beta} + \eta\delta_\beta\, out_A$
$W^{+}_{B\alpha} = W_{B\alpha} + \eta\delta_\alpha\, out_B$　　$W^{+}_{B\beta} = W_{B\beta} + \eta\delta_\beta\, out_B$
$W^{+}_{C\alpha} = W_{C\alpha} + \eta\delta_\alpha\, out_C$　　$W^{+}_{C\beta} = W_{C\beta} + \eta\delta_\beta\, out_C$

- 计算（反向传播）隐藏层误差

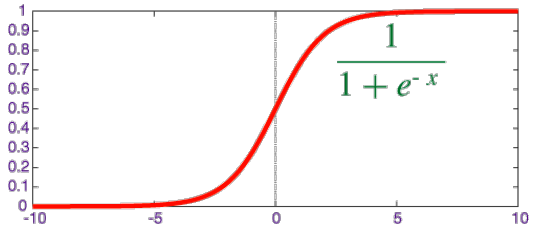
$\delta_A = out_A\,(1 - out_A)\,(\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$
$\delta_B = out_B\,(1 - out_B)\,(\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$
$\delta_C = out_C\,(1 - out_C)\,(\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$

$\delta_j = f'_{(1)}(net_j^{(1)}) \sum_k \delta_k w_{k,j}^{(2)}$

- 改变隐藏层权重 $\Delta w_{j,m}^{(1)} = \eta Error_j Out_m = \eta \delta_j x_m$

$W^{+}_{\lambda A} = W_{\lambda A} + \eta\delta_A\, in_\lambda$　　$W^{+}_{\Omega A} = W^{+}_{\Omega A} + \eta\delta_A\, in_\Omega$
$W^{+}_{\lambda B} = W_{\lambda B} + \eta\delta_B\, in_\lambda$　　$W^{+}_{\Omega B} = W^{+}_{\Omega B} + \eta\delta_B\, in_\Omega$
$W^{+}_{\lambda C} = W_{\lambda C} + \eta\delta_C\, in_\lambda$　　$W^{+}_{\Omega C} = W^{+}_{\Omega C} + \eta\delta_C\, in_\Omega$

假设激活函数都为sigmoid函数

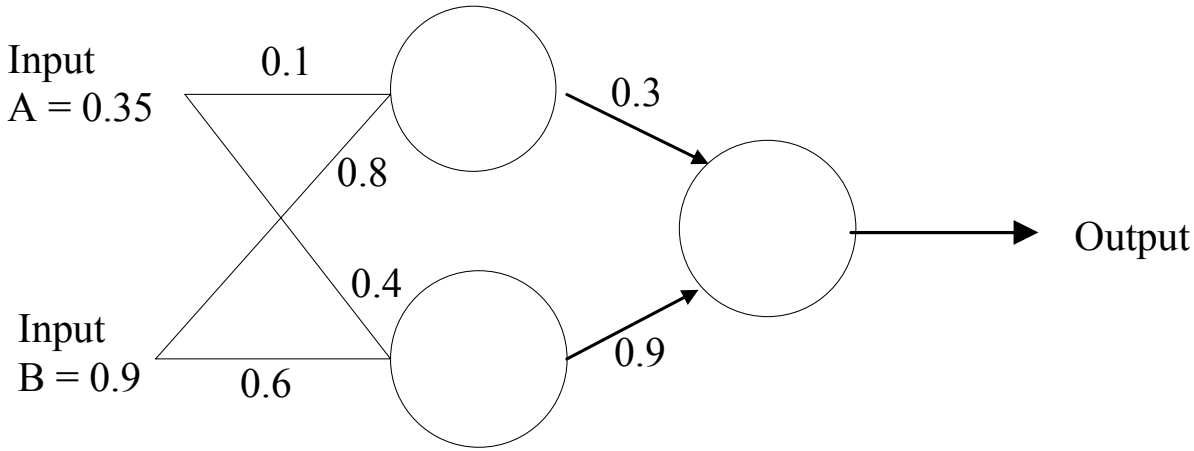


$f'_{Sigmoid}(x) = f_{Sigmoid}(x)(1 - f_{Sigmoid}(x))$

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

让我们做一些简单运算

- 考虑下面的简单网络：



- 假设神经元具有sigmoid激活功能：
  - 在网络上执行前向传播
  - 执行一次反向传播（目标=0.5）
  - 再次执行前向传播并更新预测结果

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

让我们做一些简单运算

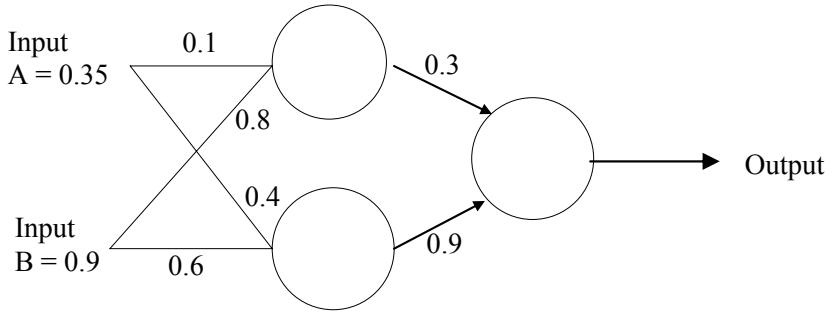
Answer:
(i)
Input to top neuron = (0.35x0.1)+(0.9x0.8)=0.755. Out = 0.68.
Input to bottom neuron = (0.9x0.6)+(0.35x0.4) = 0.68. Out = 0.6637.
Input to final neuron = (0.3x0.68)+(0.9x0.6637) = 0.80133. Out = 0.69.

(ii)
Output error $\delta=(t-o)(1-o)o = (0.5-0.69)(1-0.69)0.69 = -0.0406$.

New weights for output layer
$w1^+ = w1+(\delta \times input) = 0.3 + (-0.0406x0.68) = 0.272392$.
$w2^+ = w2+(\delta \times input) = 0.9 + (-0.0406x0.6637) = 0.87305$.

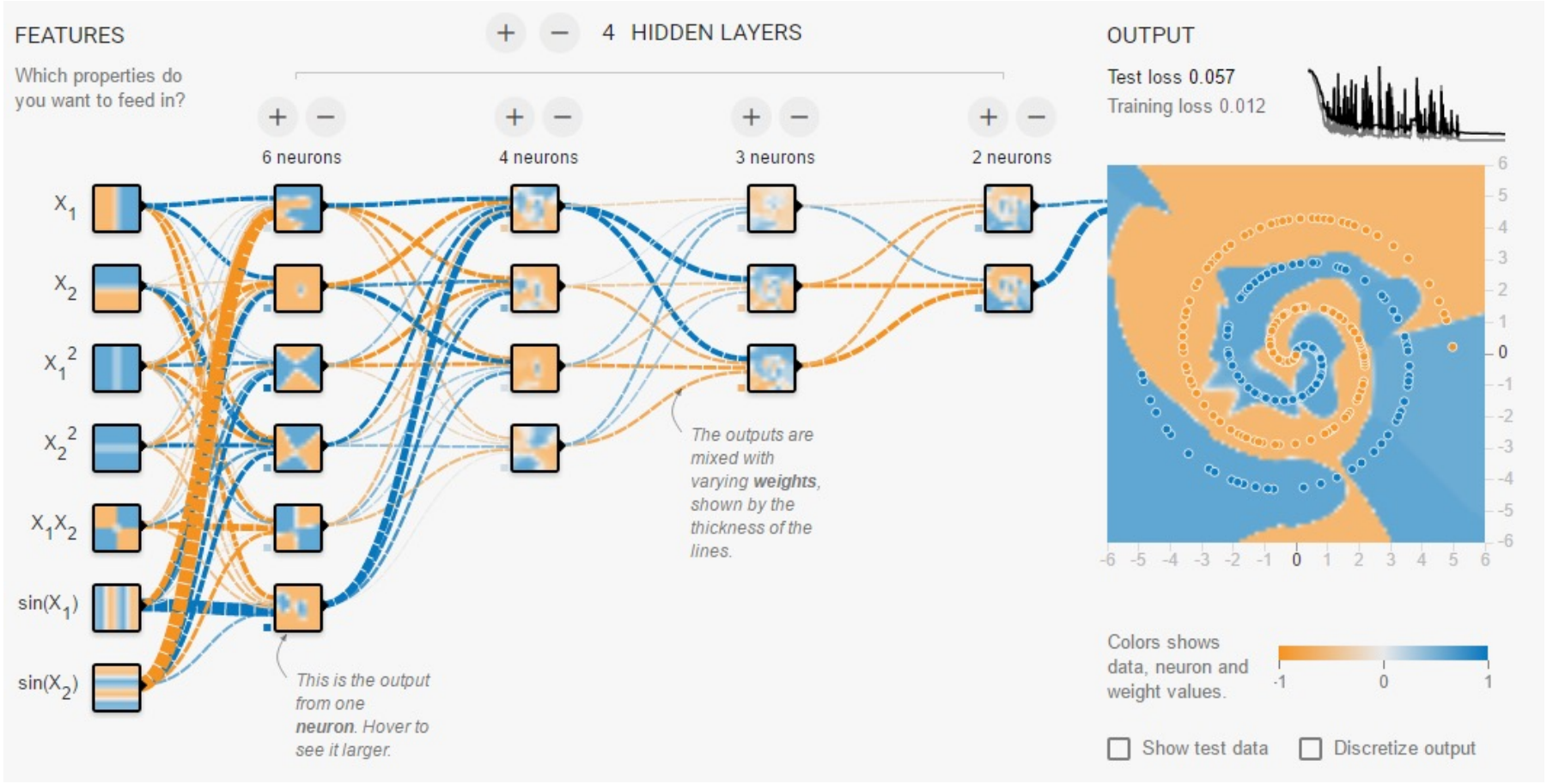
Errors for hidden layers:
$\delta1 = \delta \times w1 = -0.0406 \times 0.272392 \times (1-o)o = -2.406x10^{-3}$
$\delta2 = \delta \times w2 = -0.0406 \times 0.87305 \times (1-o)o = -7.916x10^{-3}$

New hidden layer weights:
$w3^+=0.1 + (-2.406 \times 10^{-3} \times 0.35) = 0.09916$.
$w4^+ = 0.8 + (-2.406 \times 10^{-3} \times 0.9) = 0.7978$.
$w5^+ = 0.4 + (-7.916 \times 10^{-3} \times 0.35) = 0.3972$.
$w6^+ = 0.6 + (-7.916 \times 10^{-3} \times 0.9) = 0.5928$.



(iii)
Old error was -0.19. New error is -0.18205. Therefore error has reduced.

一个Google的Demo



http://playground.tensorflow.org/

激活函数与损失函数

张伟楠 - 上海交通大学

目录

Contents

01 激活函数

02 损失函数

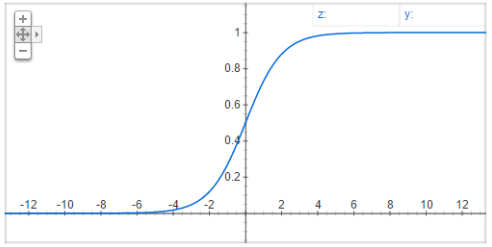
01

激活函数

非线性激活函数

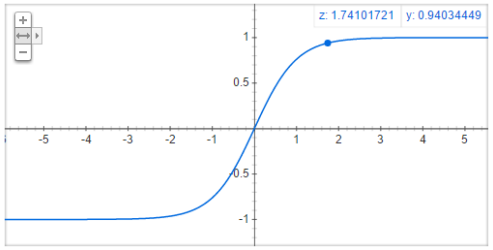
- Sigmoid激活函数

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



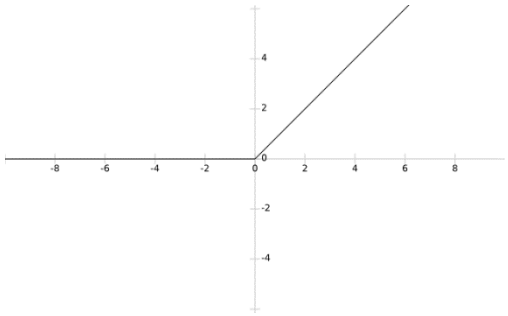
- Tanh激活函数

$$tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

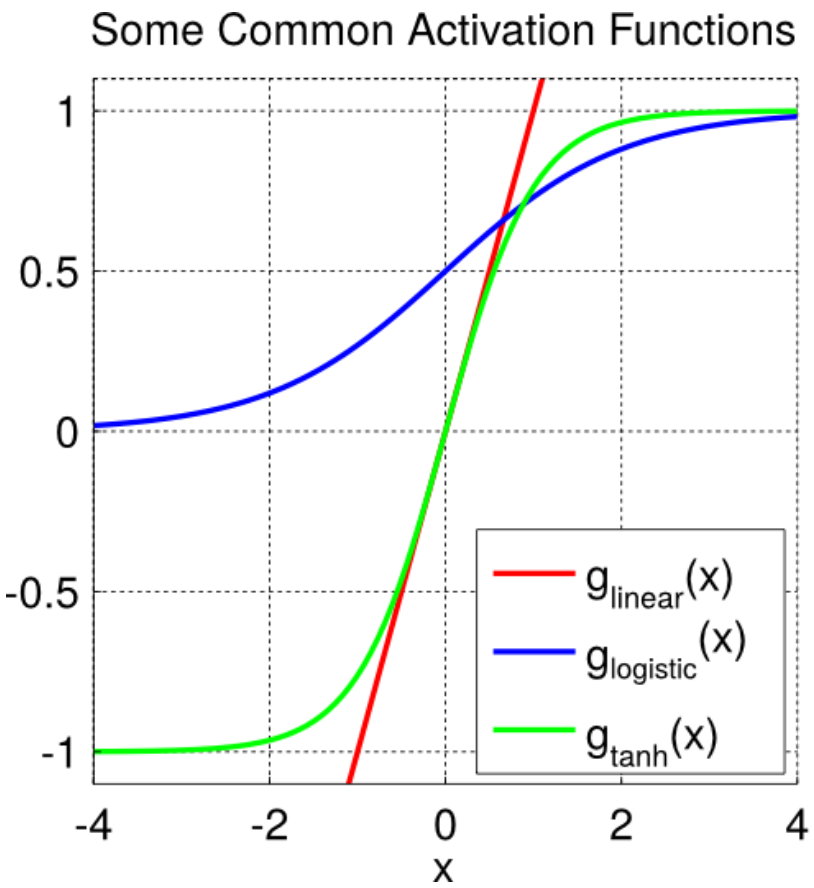


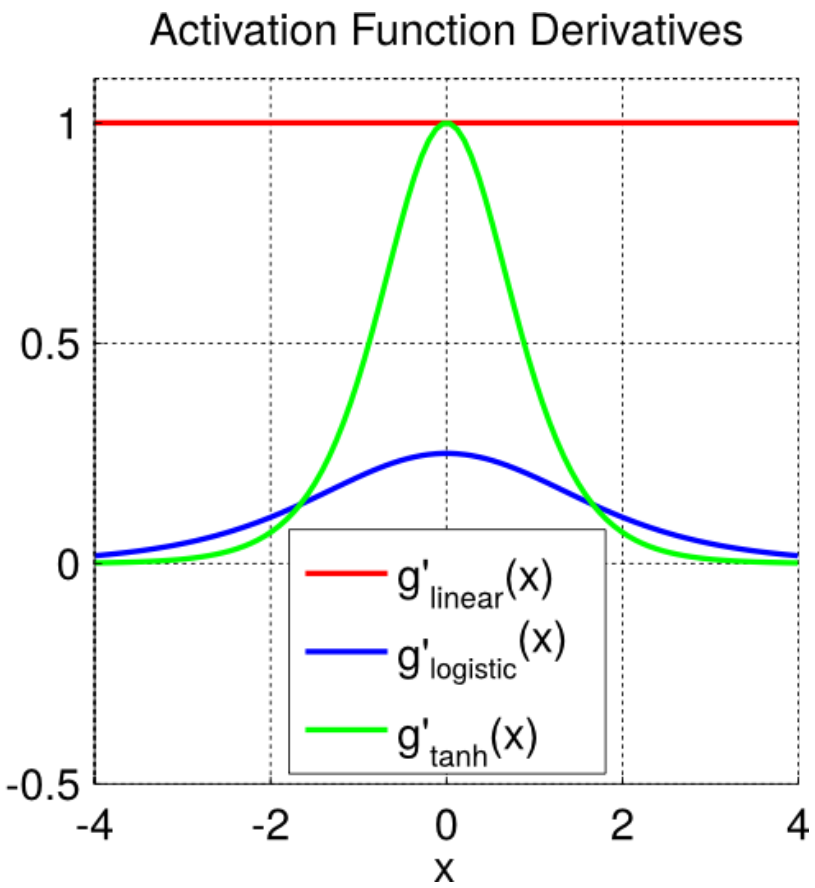
- 线性整流函数（Rectified Linear Unit,ReLU)

$$ReLU(z) = max(0, z)$$



激活函数



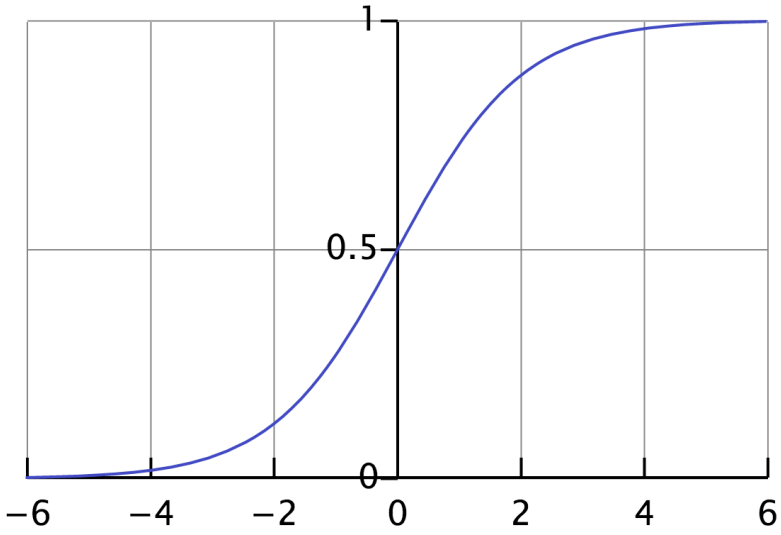


https://theclevermachine.wordpress.com/tag/tanh-function/

激活函数

逻辑Sigmoid函数

$$f_{Sigmoid}(x) = \frac{1}{1+e^{-x}}$$



- 导数：

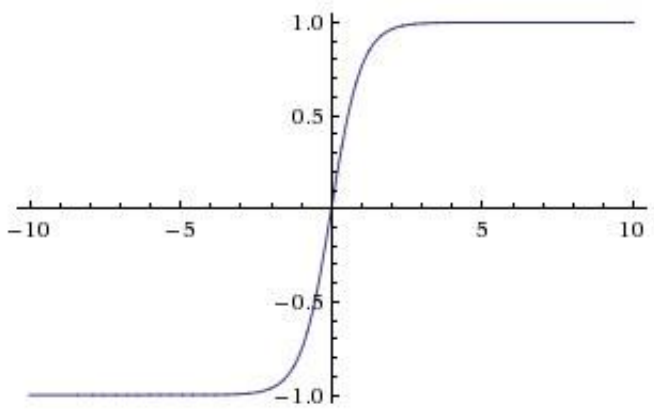
$$f'_{Sigmoid}(x) = f_{Sigmoid}(x)(1 - f_{Sigmoid}(x))$$

- 输出范围 $[0,1]$
- 受生物神经元的启发产生，可以被看做一个人工神经元在当前输入下被“激活”的概率
- 边界值会使梯度消失（为什么？）

激活函数

Tanh函数

$$f_{tanh}(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



- 导数：

$$f'_{tanh}(x) = 1 - f_{tanh}(x)^2$$

- 负值较大的输入经过tanh函数会映射为负输出
- 只有接近零的输入会被映射为接近零的输出
- 使网络训练时被“卡住”的可能性降低

https://theclevermachine.wordpress.com/tag/tanh-function/

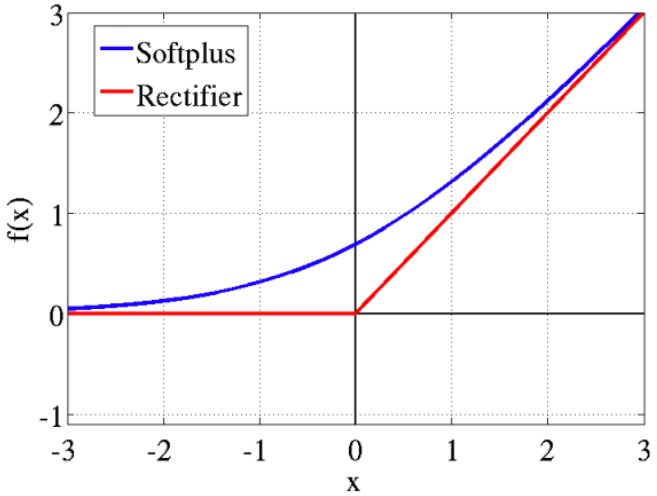
激活函数

ReLU函数(rectified linear unit)

$$f_{ReLU}(x) = max(0, x)$$

ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$



其他激活函数：

Leaky ReLU, Exponential LU, Maxout 等

- 导数：

$$f'_{ReLU}(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$$

- 另一个版本Noise ReLU：

$$f_{NoisyReLU}(x) = max(0, x + N(0, \delta(x)))$$

- ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$

- x增加时ReLU的梯度不会消失
- 可以用来对正值输入进行建模
- 由于无需计算指数函数，所以它的计算速度很快
- 使用它可以不再需要“预训练”过程

http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf

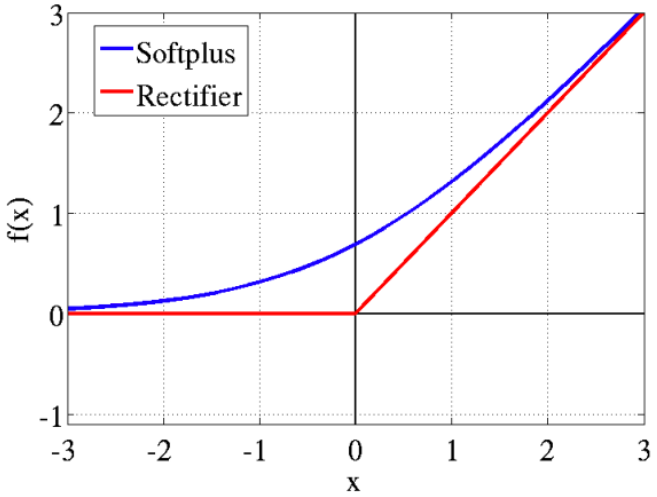
激活函数

ReLU函数(rectified linear unit)

$$f_{ReLU}(x) = max(0, x)$$

ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$

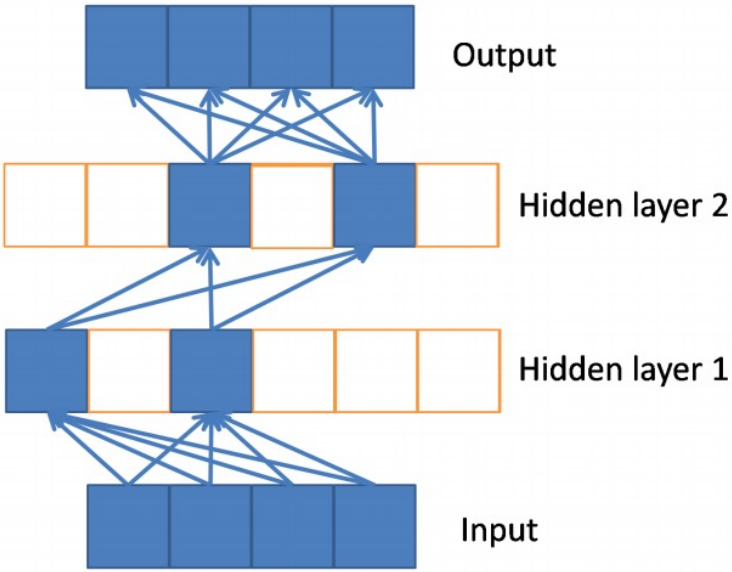


其他激活函数：

Leaky ReLU, Exponential LU, Maxout 等

http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

- 非线性取决于激活或者未激活的神经元构成的传播路径的选择
- 允许稀疏表示：
  - 对于特定的输入，只有一部分神经元是活跃的



激活信号和梯度的稀疏传播

02

损失函数

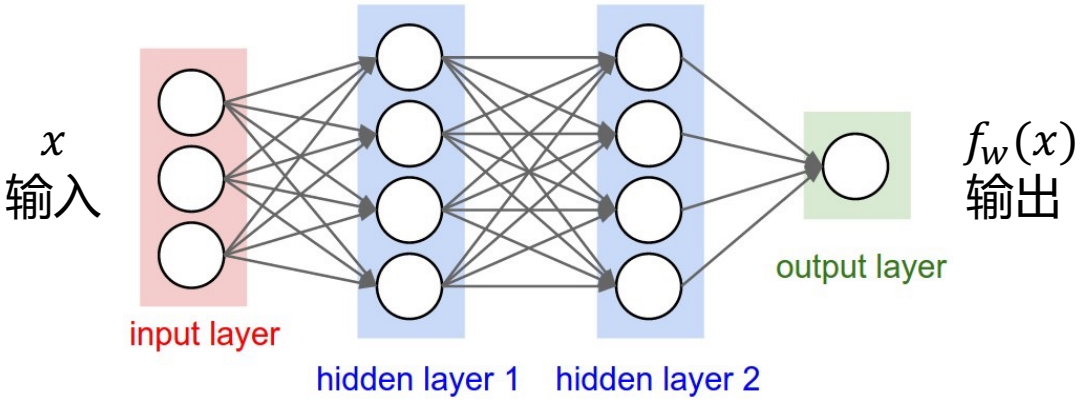
误差/损失函数

- 随机梯度下降
  - 从随机抽取的样本中进行更新（实际上执行批处理更新）：

$$w = w - \eta \frac{\partial \mathcal{L}(w)}{\partial w}$$

- 一个二值输入的平方误差损失：

$$\mathcal{L}(w) = \frac{1}{2}(y - f_w(x))^2$$



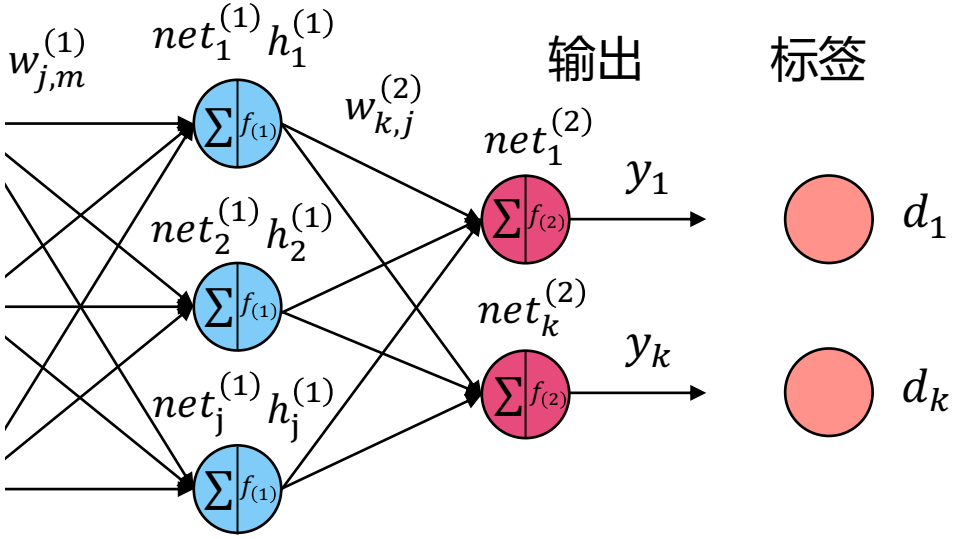
https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

误差/损失函数

多分类问题的Softmax损失（交叉熵损失）

（类标签服从多项分布）

$$\mathcal{L}(w) = -\sum_k d_k log\hat{y}_k \qquad 其中 \quad \hat{y}_k = \frac{exp(\sum_j w_{k,j}^{(2)} h_j^{(1)})}{\sum_{k'} exp\left(\sum_j w_{k',j}^{(2)} h_j^{(1)}\right)}$$



One hot编码的类标签

深度学习思想简介

张伟楠- 上海交通大学

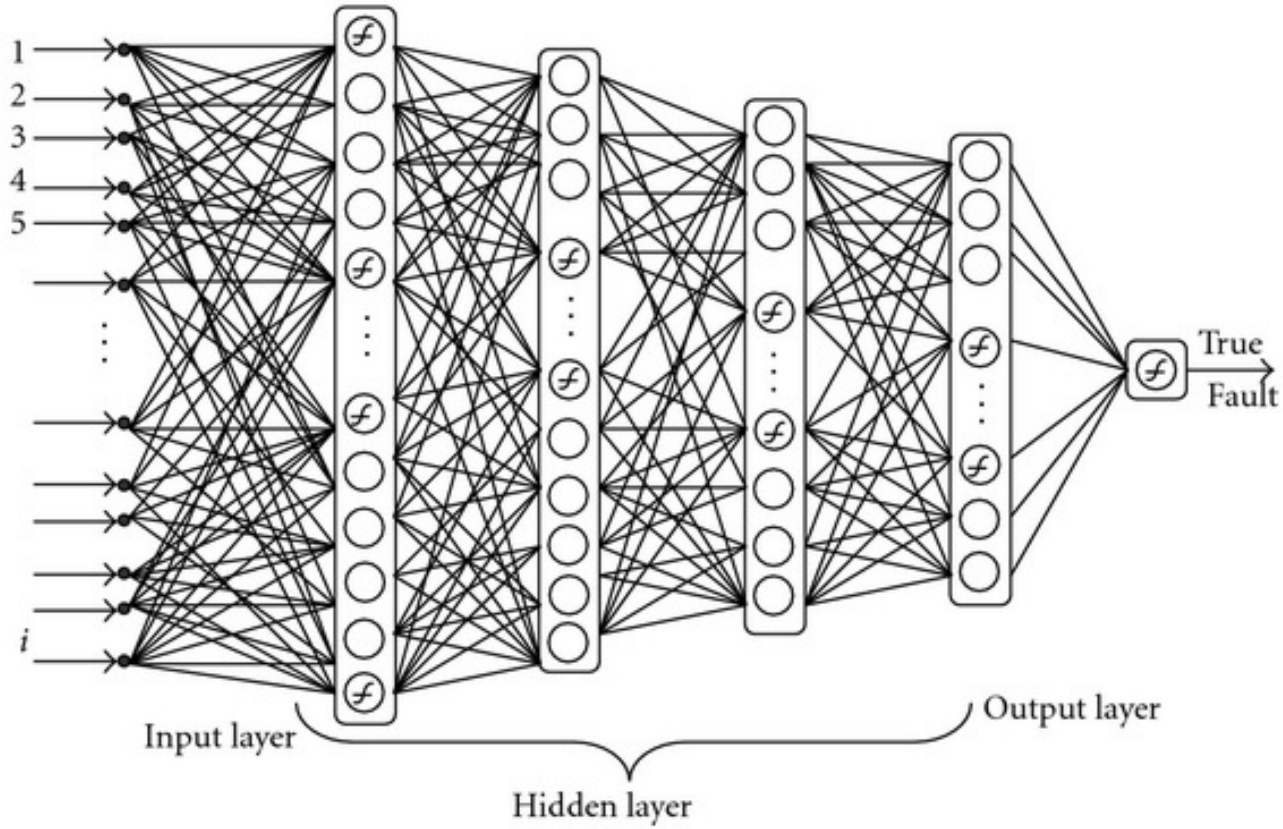
什么是深度学习？

定义

- 深度学习是一种有着多个层次的表征学习（representation-learning）方式
- 通过简单但非线性的模块自动地将每个层次的特征（从原始输入开始）转变为更高级抽象的表征形式
- 主要通过神经网络实现

LeCun, Bengio and Hinton. Deep Learning. Nature 2015.

深度神经网络 (DNN)



具有许多隐藏层的多层感知机

训练深度网络的难点

- 缺乏大数据
  - 现在我们有很多大数据
- 缺乏计算资源
  - 现在我们有 GPU 和 HPC
- 容易进入（坏的）局部最小值
  - 现在我们可以使用预训练技术和各种优化算法
- 梯度消失
  - 现在我们可以使用ReLU，批标准化，残差网络等方法
- 正则化
  - 现在我们可以使用dropout方法

大数据时代→新的AI时代

- 随着“大数据”概念的兴起，许多先进的大数据分布式数据处理系统被发明
  - MapReduce，BigTable，Hadoop，Spark，Ray等
- 公司有意识地收集、存储和规范化数据，并且进行数据挖掘
  - 谷歌，亚马逊，Facebook，百度，阿里巴巴，头条等
  - 高校、餐厅、交通局、医院等
- 有越来越多的大数据专家和技术人员
- 出现了收集和清理大数据的更好机制
  - 众包平台比如Amazon Mechanism Turk，CloudFlower等

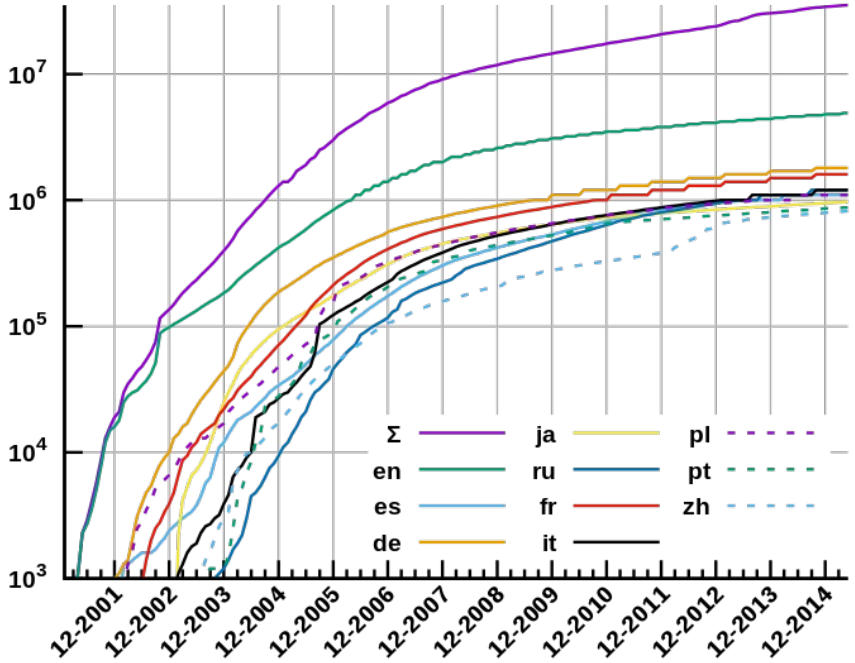
图像数据集：ImageNet

- 统计数据（2010年4月30日更新）
  - 非空概念总数: 21,841
  - 图像总数: 14,197,122
  - 带边界框注释的图像数量: 1,034,908
  - 有 SIFT 特征的概念数量: 1,000
  - 有 SIFT 特征的图像数量: 120 万张

http://image-net.org/about-stats

文本数据集：Wikipedia

- Wikipedia是一个基于网络、多语言、免费的百科全书，其内容是公开可编辑和可查看的



- 5,903,353 篇英文文章 (2019年8月更新)

http://wikipedialivecount.com/

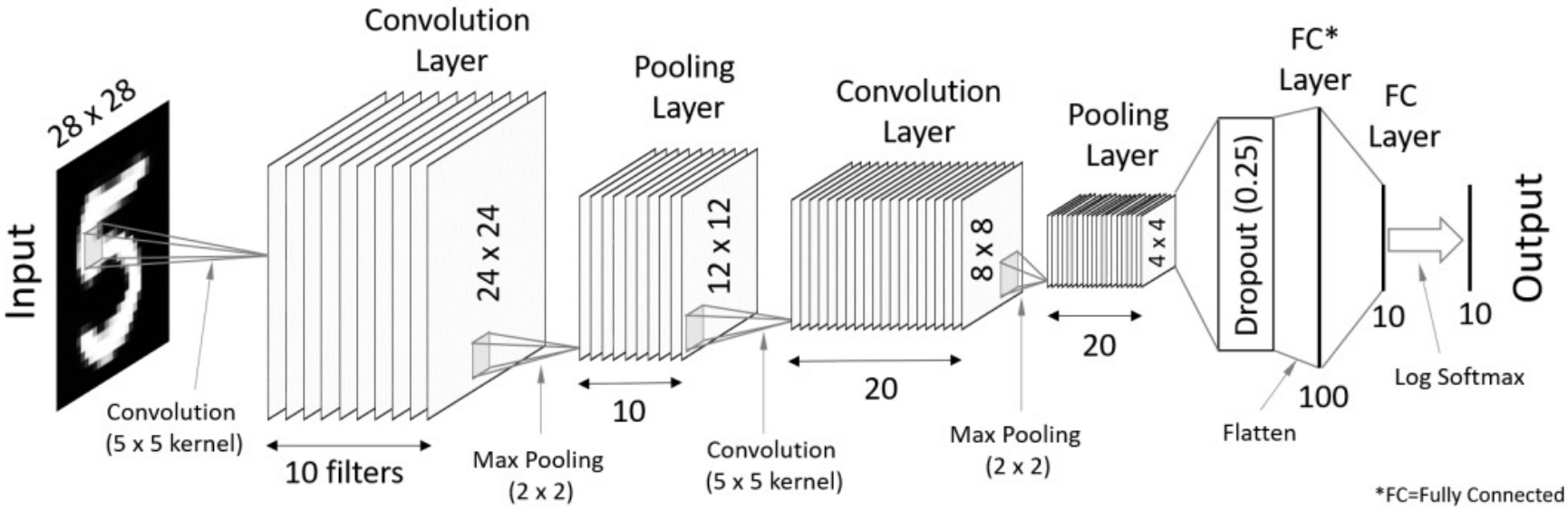
行为数据集:Criteo

- Criteo Terabyte 数据集包含一个月的点击日志，其中有数十亿个数据实例
- 统计
  - 实例: 10^8
  - 分类特征: 10^6
  - 域: 39

http://labs.criteo.com/2013/12/download-terabyte-click-logs/

通过 GPU 计算

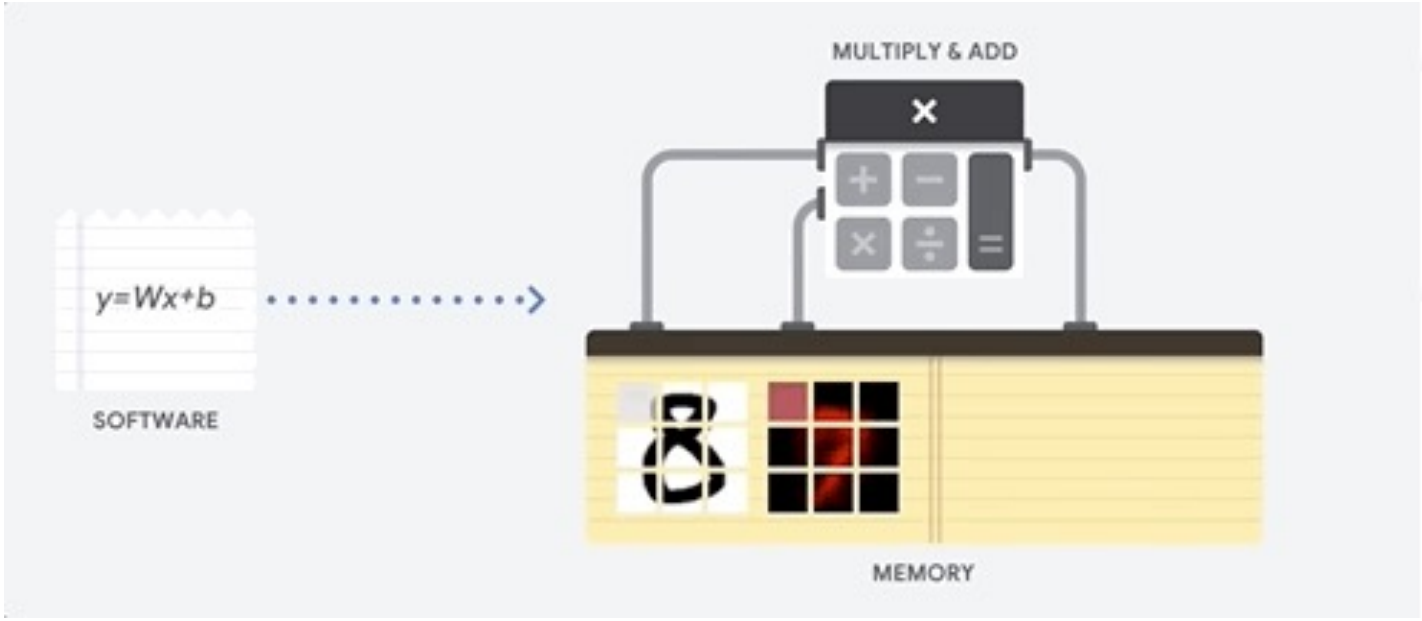
- 神经网络通过张量（tensor）计算



- GPU 之所以这么快，是因为它们对于矩阵乘法和卷积操作非常有效
- “CPU 优化延迟，而 GPU 优化带宽"

https://www.quora.com/Why-are-GPUs-well-suited-to-deep-learning

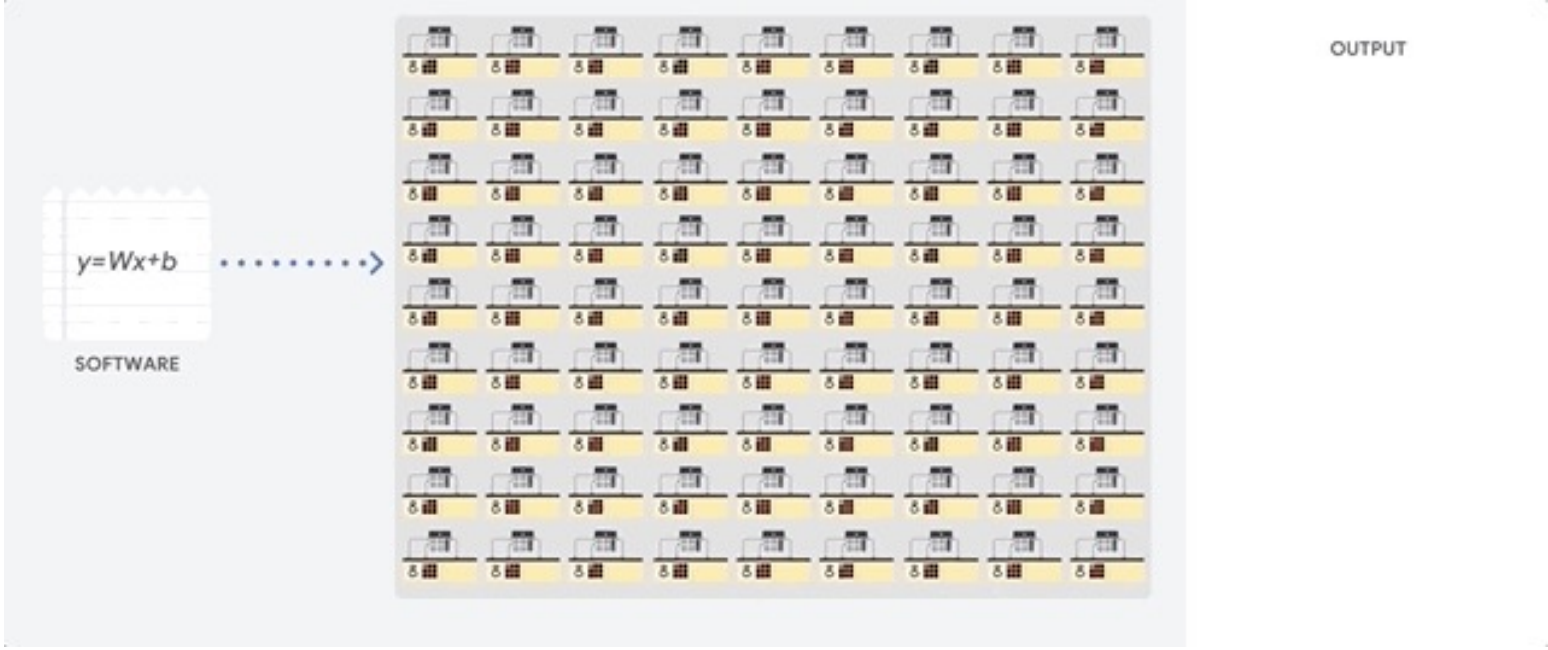
CPU 的工作原理



- CPU 的最大好处是它的灵活性
  - 凭借其 Von Neumann 架构可以为数百万种不同的应用程序加载任何类型的软件
- 硬件在从软件读取下一个指令之前并不确定其下一个计算是什么

https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning

GPU 的工作原理



- GPU 体系结构适用于具有大量并行性的应用程序
  - 神经网络中的矩阵乘法
- Nvidia 1080Ti GPU 有 3584 个内核，A100 GPU 有 6912 个内核
- GPU 仍然是一个通用处理器，必须支持数百万种不同的应用程序和软件
- TPU 是专门为深度学习中的张量计算设计的

https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning

梯度消失问题的解决方法

张伟楠 - 上海交通大学

目录

Contents

01 ReLU函数

02 深度残差网络

03 批标准化

01
ReLU
函数

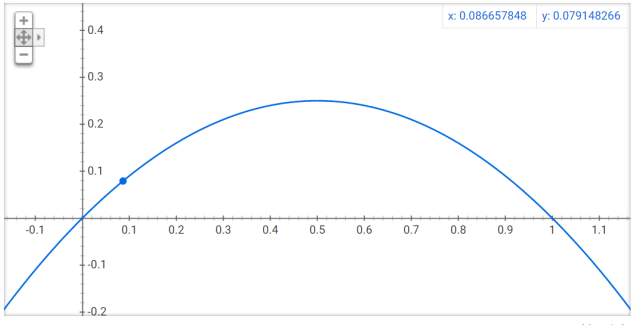
梯度消失问题

- Sigmoid激活函数

$$\sigma(z) = \frac{1}{1+e^{-z}} \in (0,1)$$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z)) \in (0,0.25)$$

Graph for x*(1-x)



- 梯度范围可能会变得越来越小

$$\Delta w_{j,m}^{(1)} = -\eta \frac{\partial E(W)}{\partial w_{j,m}^{(1)}} = -\eta \frac{\partial E(W)}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = \eta \sum_k (d_k - y_k) f'_{(2)}(net_k^{(2)}) w_{k,j}^{(2)} x_m f'_{(1)}(net_j^{(1)}) = \eta \delta_j x_m$$

(0,0.25) (0,0.25)

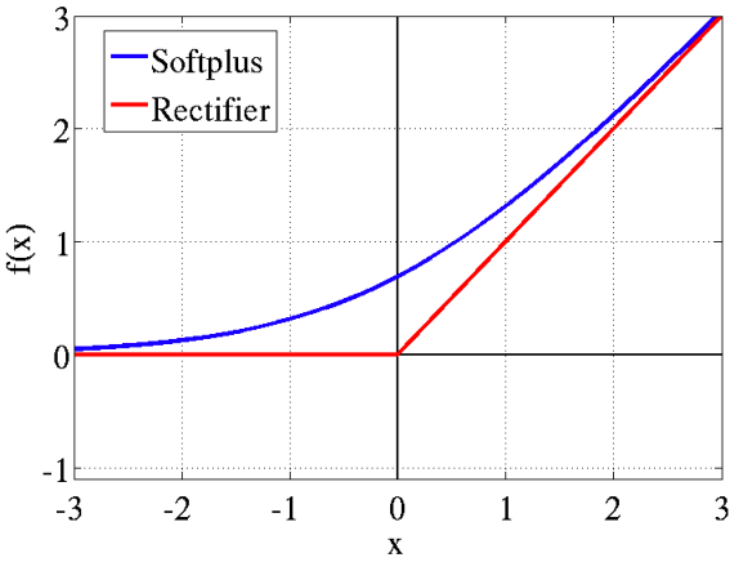
- 在反向传播超过 5 层后，梯度可能会消失

https://www.quora.com/Why-are-GPUs-well-suited-to-deep-learning

激活函数

ReLU函数(rectified linear unit)

$$f_{ReLU}(x) = max(0, x)$$



- 导数：

$$f'_{ReLU}(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$$

- 另一个版本Noise ReLU：

$$f_{NoisyReLU}(x) = max(0, x + N(0, \delta(x)))$$

- ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$

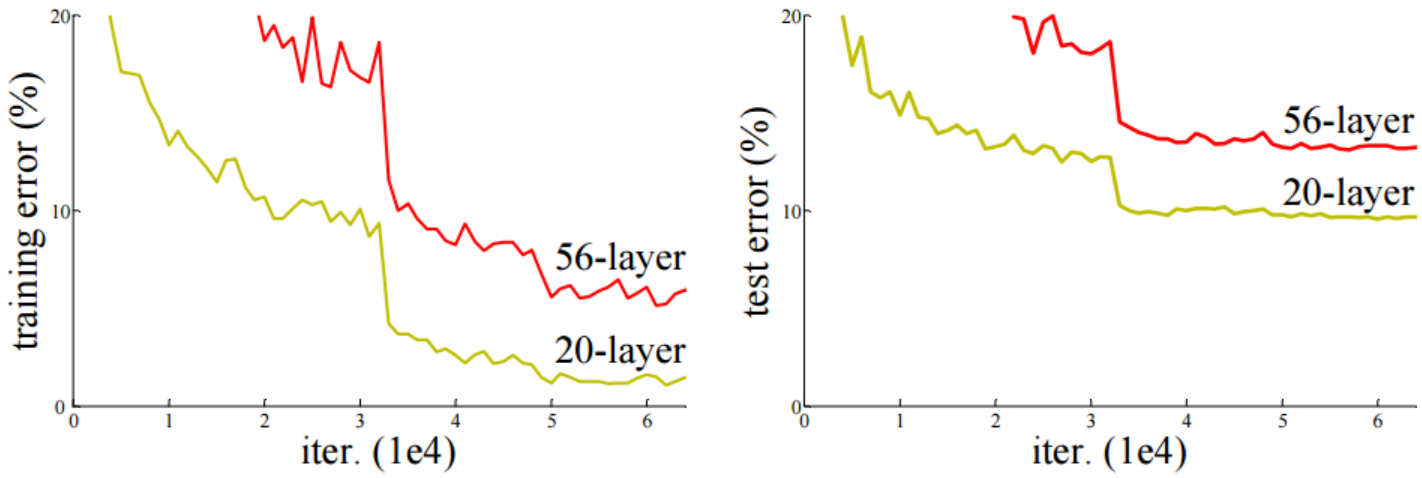
- x增加时ReLU的梯度不会消失
- 可以用来对正值输入进行建模
- 由于无需计算指数函数所以它的计算速度很快
- 使用它可以不再需要“预训练”过程

http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf

02
深度残差网络

ResNet：深度残差网络

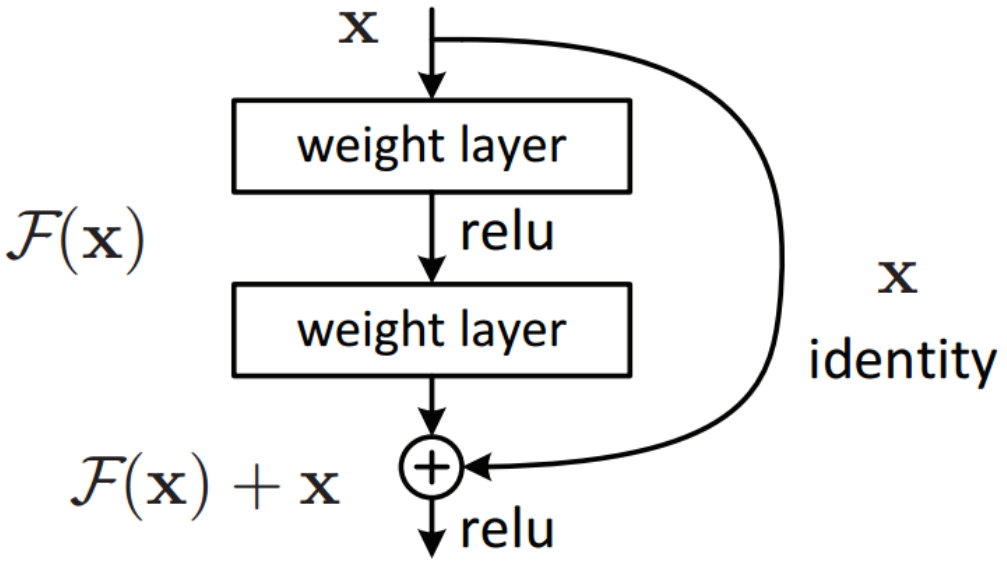
- 训练深度网络的困难性：

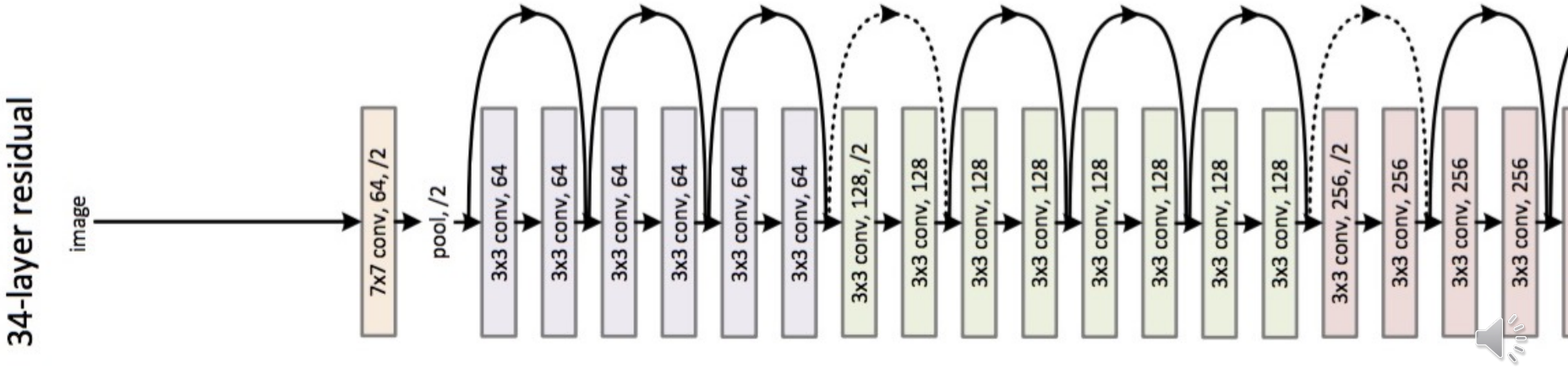


- 有时，即使是在训练数据上更深层的网络性能也可能比较浅层的网络差

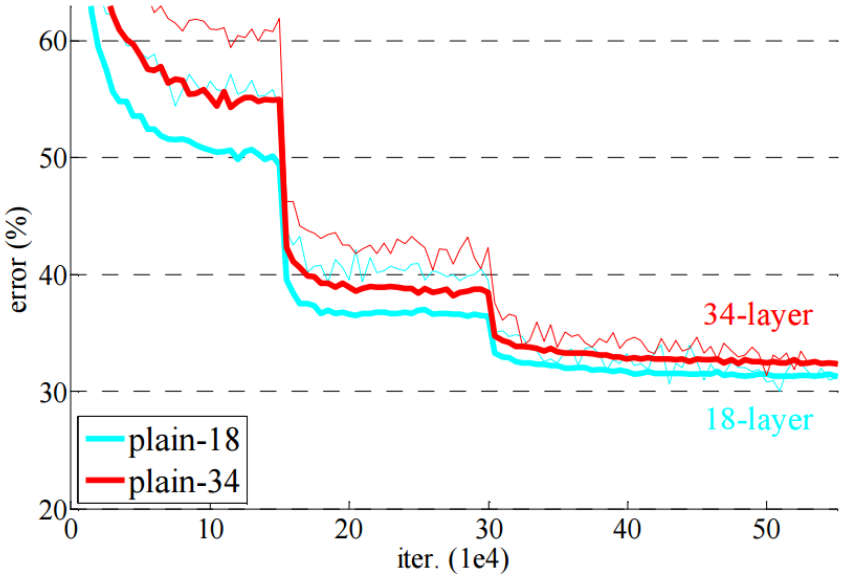
ResNet：深度残差网络

- 一个ResNet的构造块

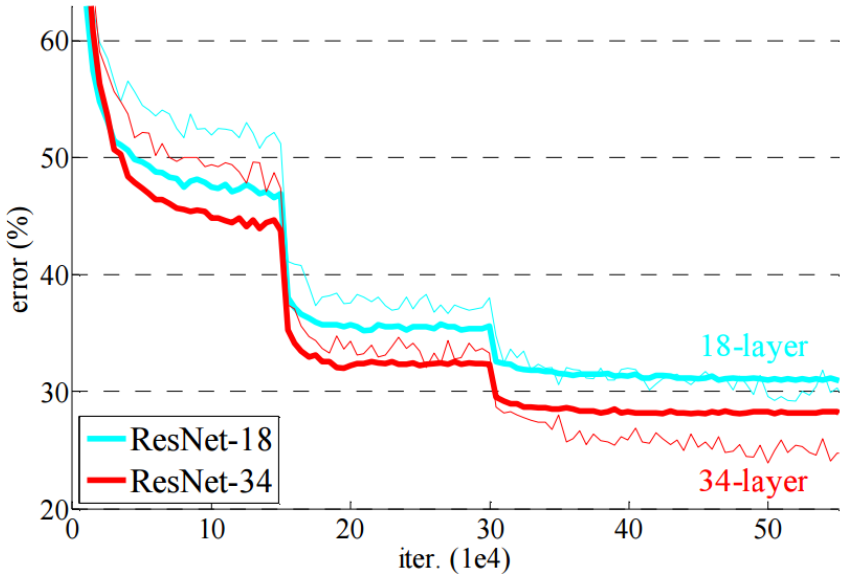




残差网络在ImageNet上的表现



18层和34层的普通网络



18层和34层的残差网络

- 细线表示训练误差，粗线表示验证误差
- 与普通网络相比，残差网络没有额外的参数

03
批标准化

批标准化（Batch Normalization）

- 内部协变量变化 (Covariate Shift)
  - 某一层神经网络的输入分布会发生变化

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

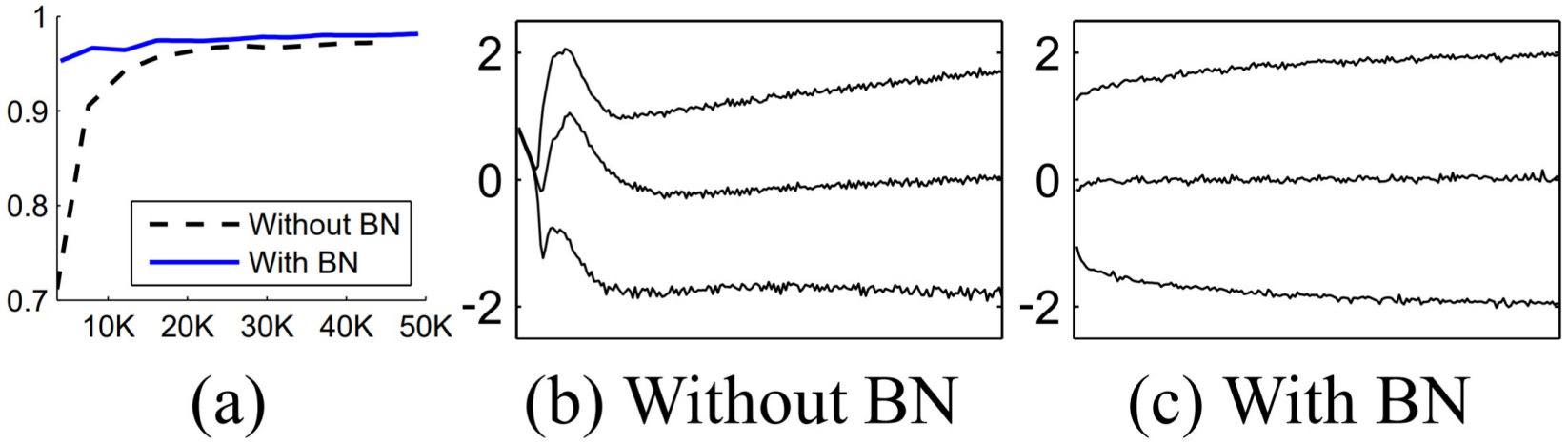
注意：
γ和β可以
被学出来

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Sergey Ioffe, Christian Szegedy. ICML2015
https://www.zhihu.com/question/283715823 https://zhuanlan.zhihu.com/p/33173246

批标准化实验



（a）在MNIST数据集上的测试准确率

（b,c）输入到sigmoid函数的数据在训练过程中的分布变化，图中分别代表15%，50%，85%的数据

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Sergey Ioffe, Christian Szegedy. ICML2015

陷入局部最小的解决方法

张伟楠 - 上海交通大学

目录

Contents

01 深度信念网络

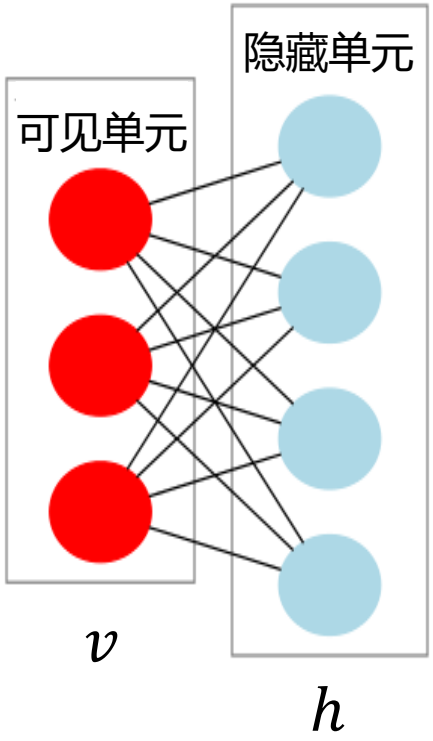
02 Adam算法

01
深度信念网络

差的局部最小值

预训练：寻找一个好的网络初始化

- 使用神经网络的无监督学习
- 基本思想：使用神经网络恢复数据
- 限制玻尔兹曼机



受限玻尔兹曼机（RBM）

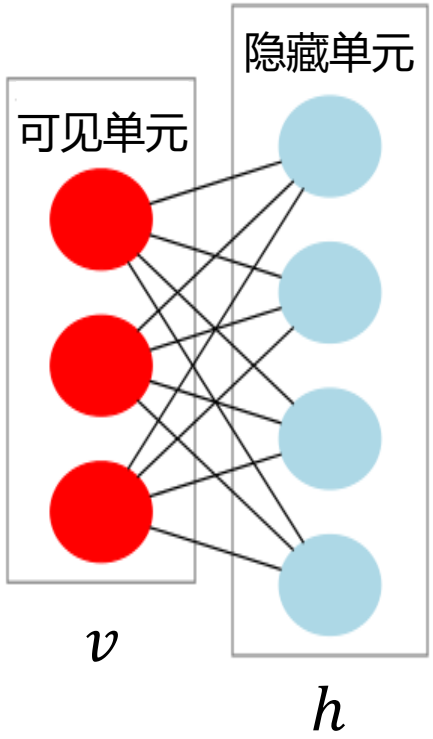
定义

- 限制玻尔兹曼机（restricted Boltzmann machine, RBM）是一种可以通过输入数据集学习概率分布的随机生成神经网络

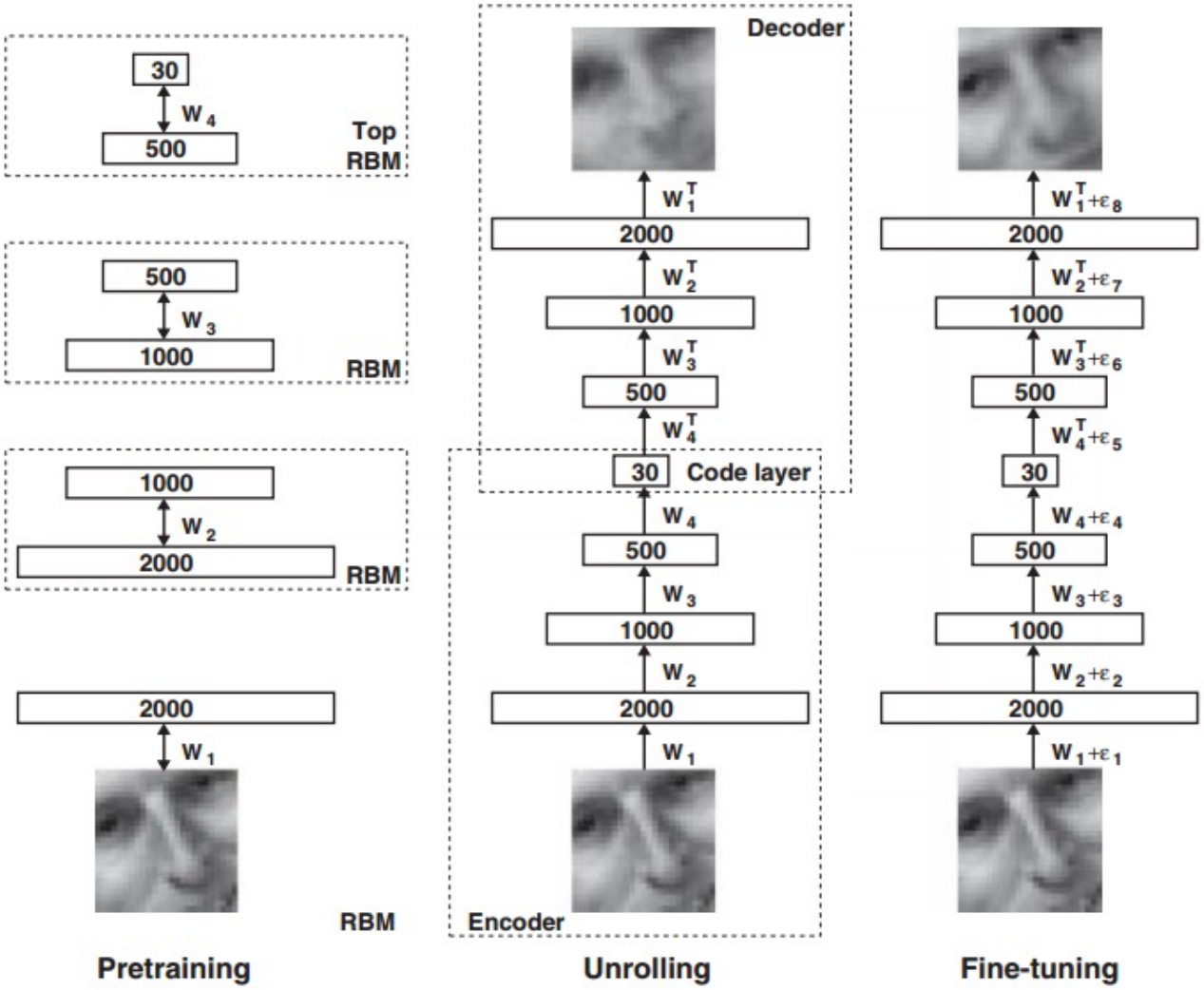
- 无向图模型
  - 受限：可见(隐藏)单元没有相互连接
  - 势能函数

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

$$p(v,h) = \frac{1}{Z} e^{-E(v,h)}$$



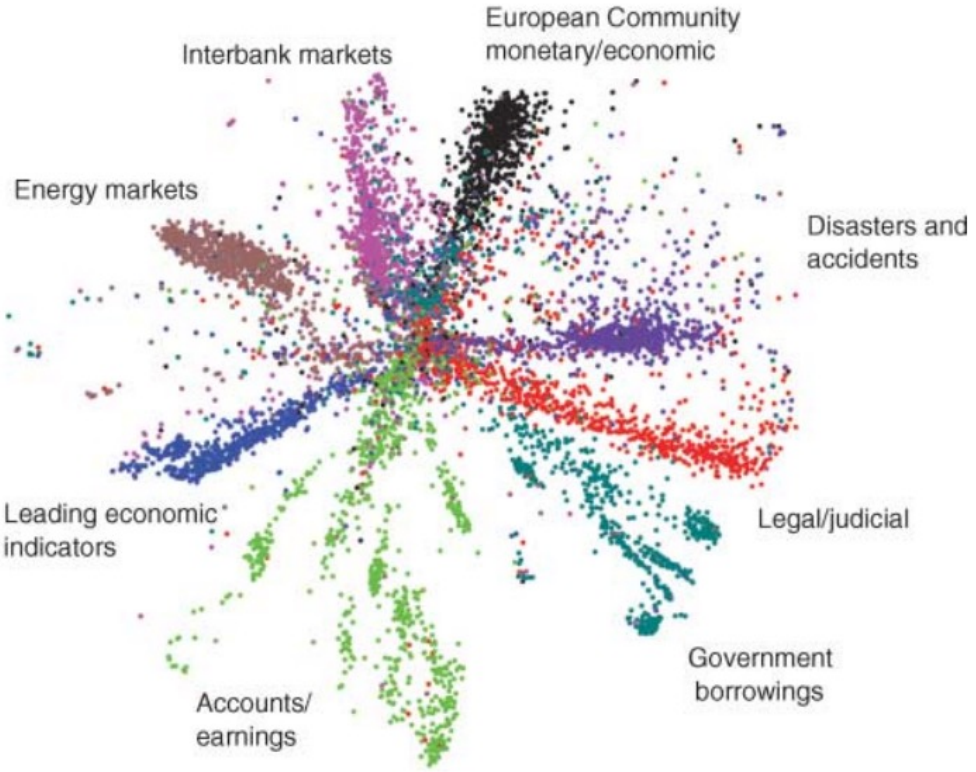
深度信念网络（DBN）



Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." science 313.5786 (2006): 504-507.

潜在因子分析

基于PCA的潜在因子分析



基于一个由DBN训练的2000-500-250-125-2的自编码器的潜在因子分析

Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." science 313.5786 (2006): 504-507.

02
Adam
算法

带动量的梯度算法

- 适应性矩估计(Adam: Adaptive Moment Estimation)
  - 存储过去时刻梯度的指数衰减平均值m_t，类似于动量，使用一阶指数平滑（对历史梯度进行正负抵消）
  - 存储过去时刻梯度平方的指数衰减平均值v_t，类似于动能，对应自适应学习率
    - 例如，某一维度在迭代过程中一直以很大的梯度进行更新，表明此方向梯度变换较为剧烈（不稳定），因此可减小学习率；反之，如果某一维度一直以很小的梯度进行更新，证明此方向梯度变换较为稳定，因此可以加大学习率。

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

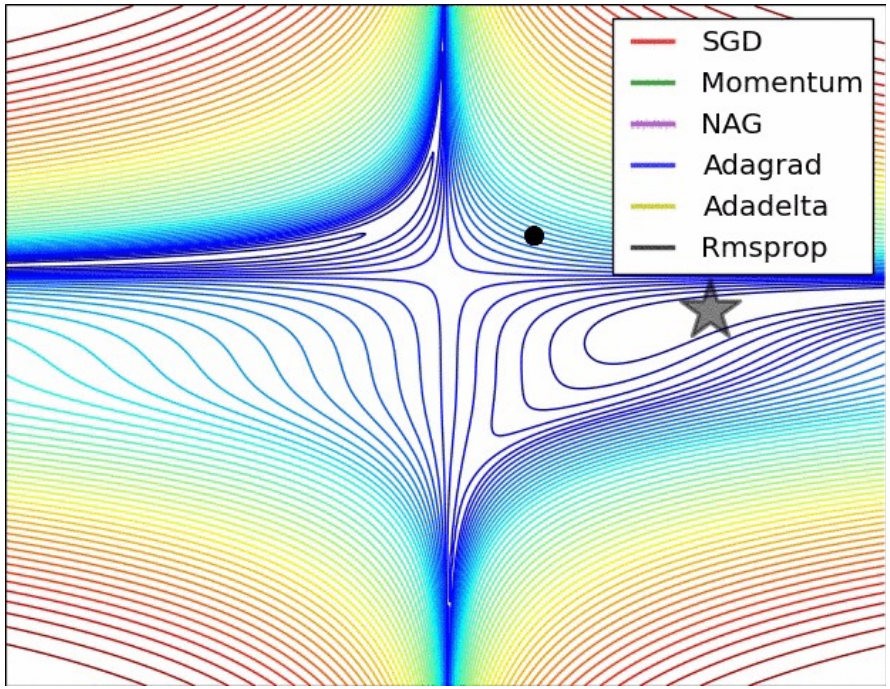
由于m_t和v_t初始化为0，需要偏置校正估计值

β_1的默认值为0.9，β_2的默认值为0.999，ϵ的默认值为10^{-8}

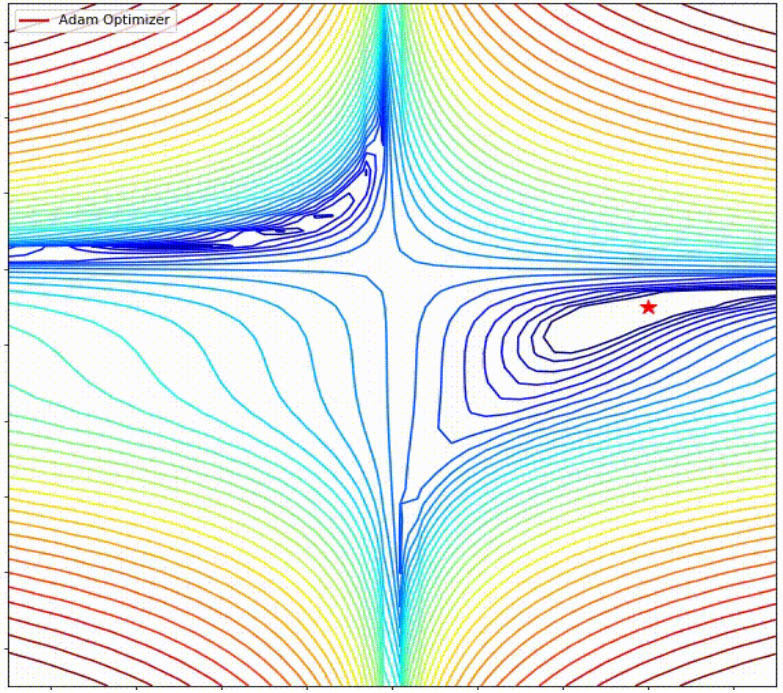
Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." ICLR 2015.

各类梯度方法的效果对比

各类梯度算法

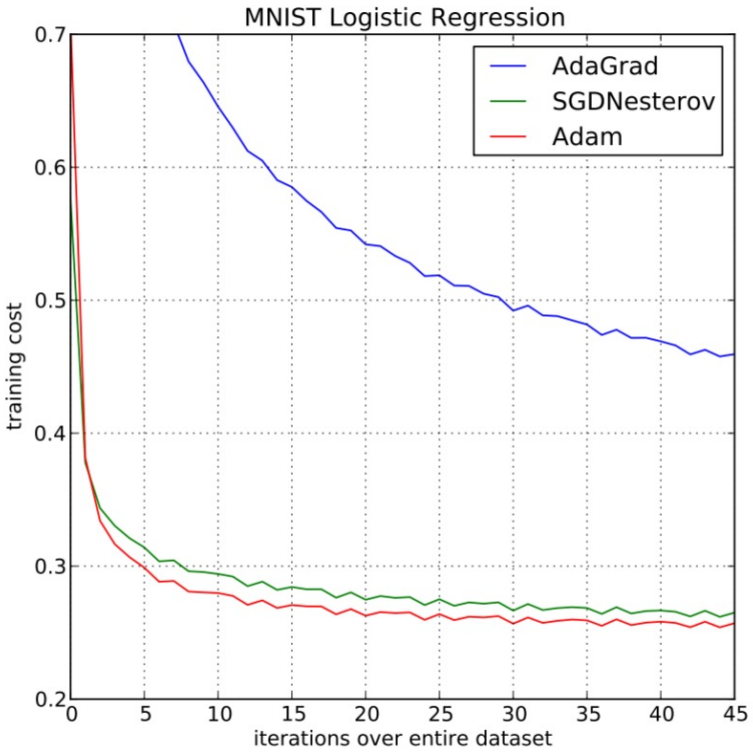


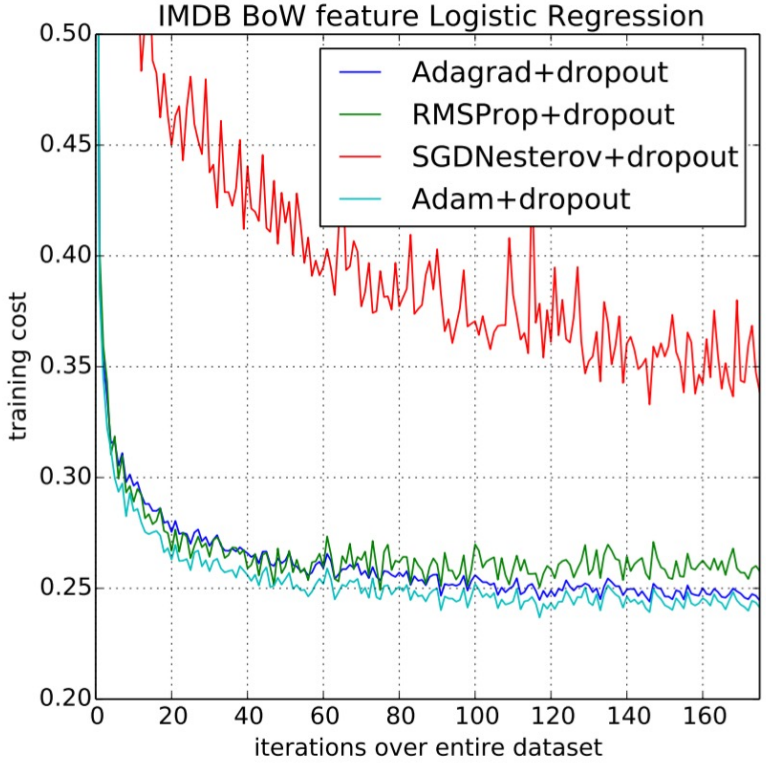
Adam梯度算法



Adam实验

- 在MNIST图像和IMDB电影评论上逻辑回归训练负对数似然
- IMDB电影评论具有10,000个词袋（BoW）特征向量





Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." ICLR 2015.

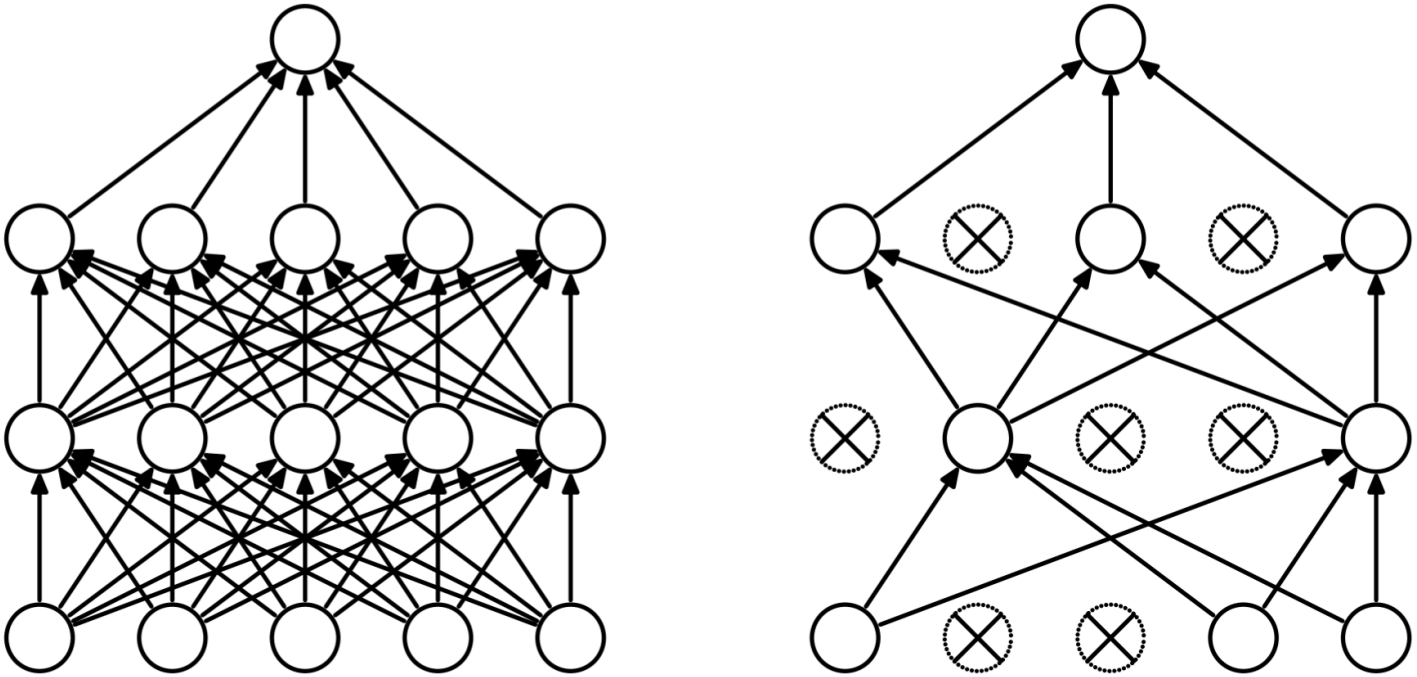
深度学习中的正则化

张伟楠- 上海交通大学

神经网络正则化

Dropout

- 在训练阶段，每个小批量训练时随机禁用网络中一部分节点及其连接

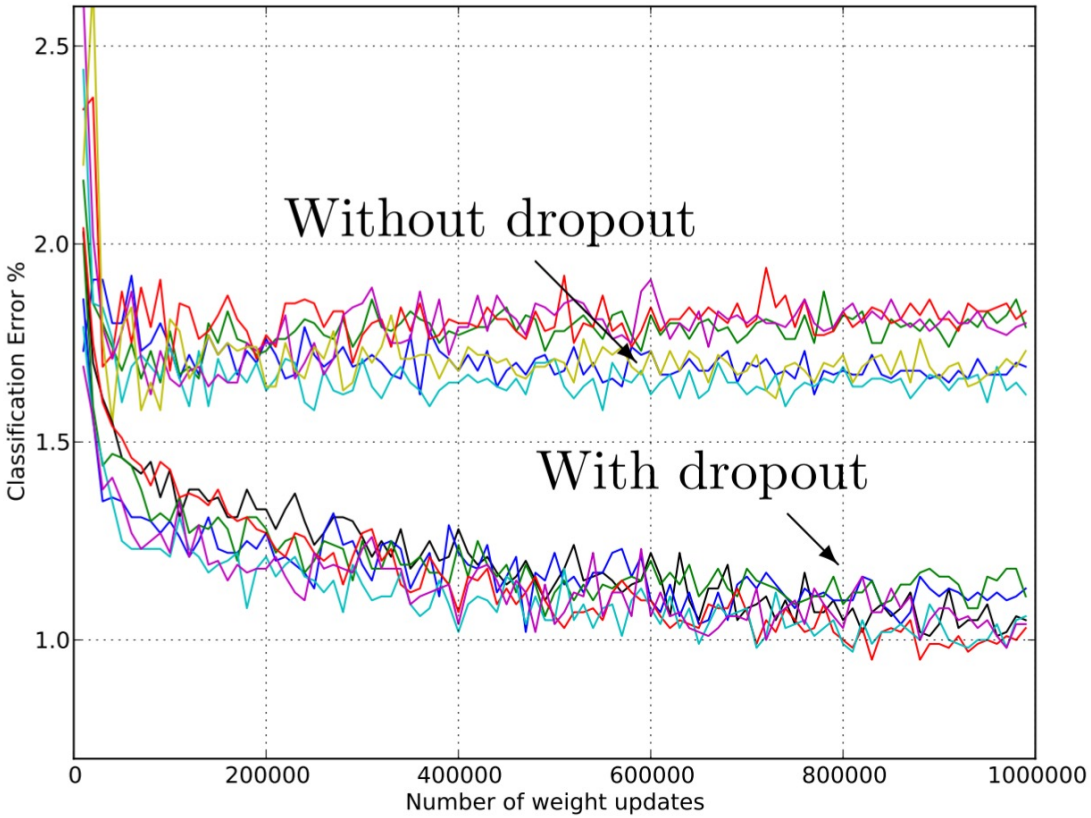


(a) Standard Neural Net　　(b) After applying dropout.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." JMLR 2014.

Dropout实验

- 使用dropout和不使用dropout的不同网络结构在MNIST上的测试误差
- 网络有2到4个隐藏层，每个层有1024到2048个单元



REVIEW：训练深度网络的难点及近年发展

- 缺乏大数据
  - 现在我们有很多大数据
- 缺乏计算资源
  - 现在我们有 GPU 和 HPC
- 容易进入（坏的）局部最小值
  - 现在我们可以使用预训练技术和各种优化算法
- 梯度消失
  - 现在我们可以使用ReLU，批标准化，残差网络等方法
- 正则化
  - 现在我们可以使用dropout方法

卷积神经网络

张伟楠 – 上海交通大学

目录

Contents

01 卷积神经网络

02 使用案例

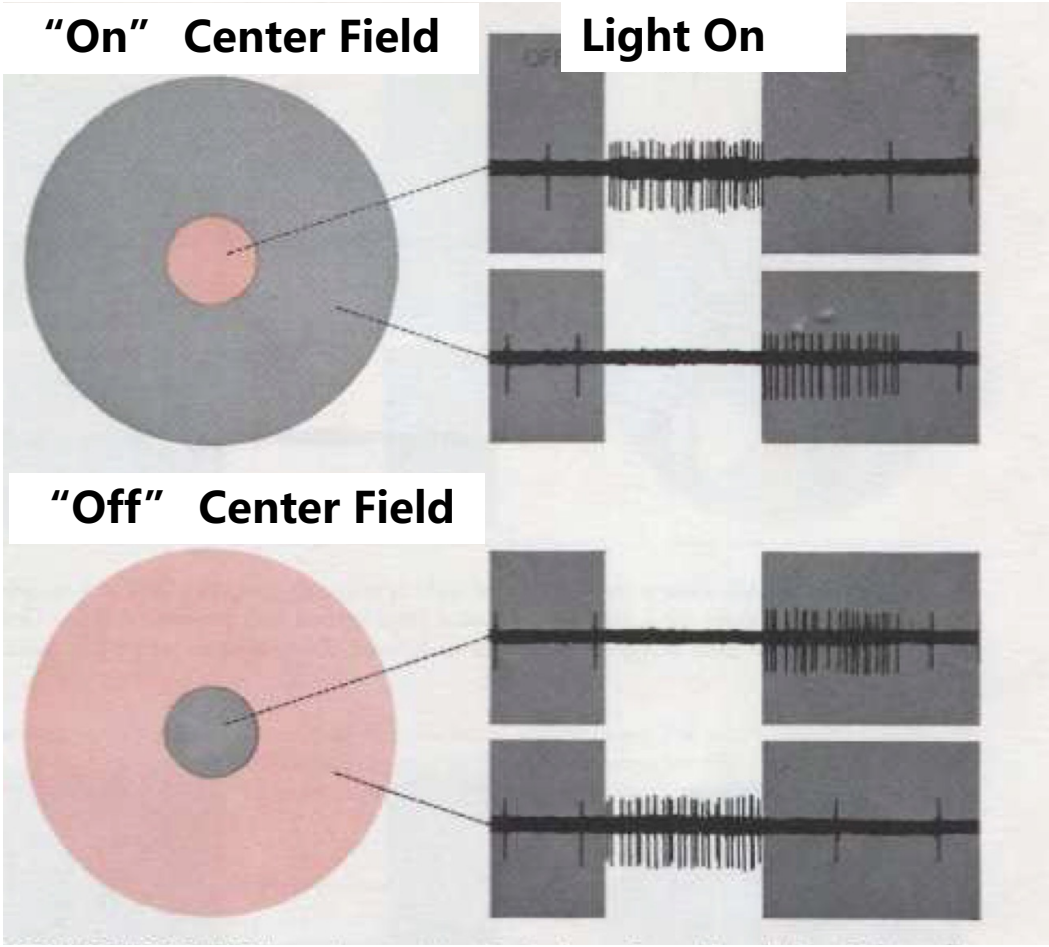
01

卷积神经网络

卷积神经网络：感受野

感受野

- 视网膜中的神经元只响应视野的受限区域中的光刺激
- 两个视网膜神经节细胞感受野的动物实验
  - 视野是视网膜的圆形区域
  - 当中心点亮周围变暗时，细胞（上部）响应
  - 当中心变暗并且周围照亮时，细胞（下部）响应
  - 当中心和周围都被照亮时，两个单元都给出响应和关闭响应，但都没有响应像只有中心或周围被照亮时那样强烈

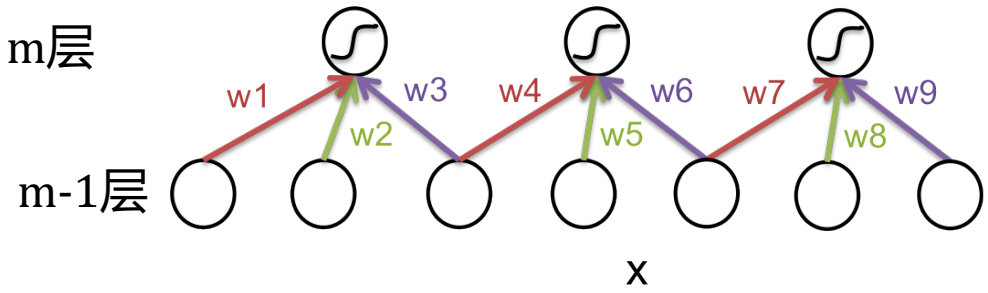


Hubel D.H. : The Visual Cortex of the Brain Sci Amer 209:54-62, 1963

卷积神经网络

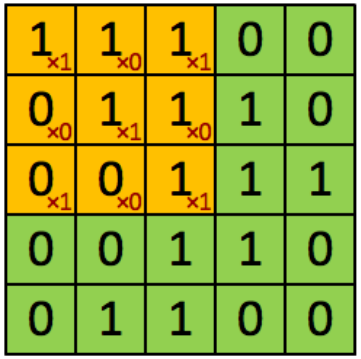
- 通过局部相关性稀疏连接
  - 过滤器：m层中隐藏单元的输入来自m-1层中具有空间连接的感受野的单元的子集
- 共享权重
  - 每个过滤器都在整个视野中复制。 这些复制的单元共享相同的权重并形成特征映射。

一维情况

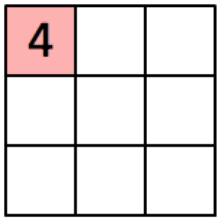


相同颜色的边权重相同

二维情况(下标是权重)

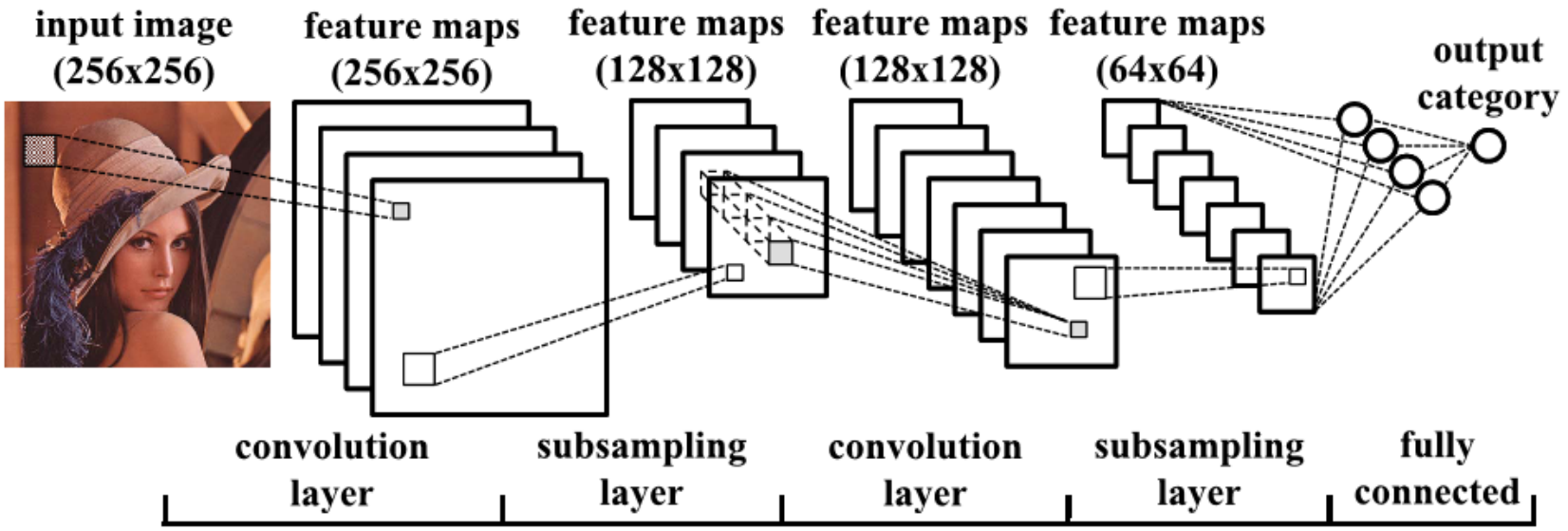


m-1层



m层的一个过滤器

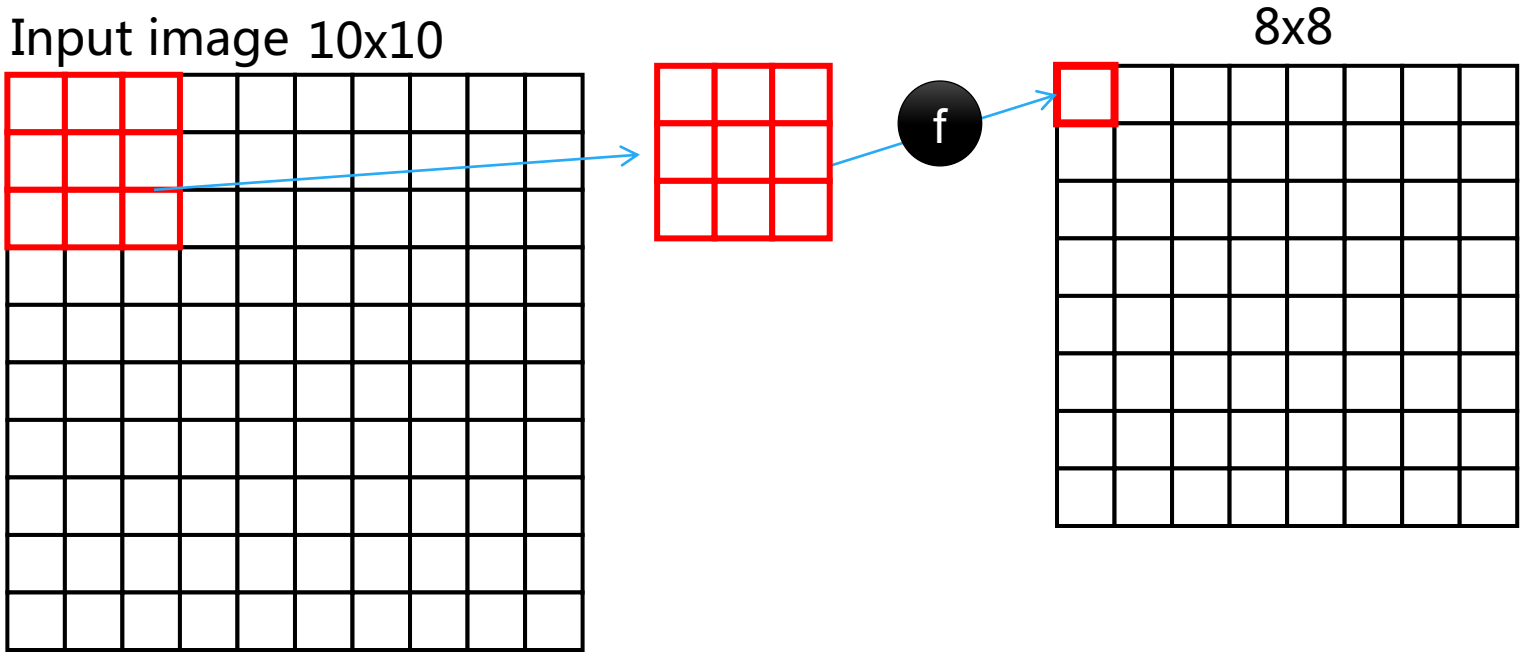
卷积神经网络(CNN)

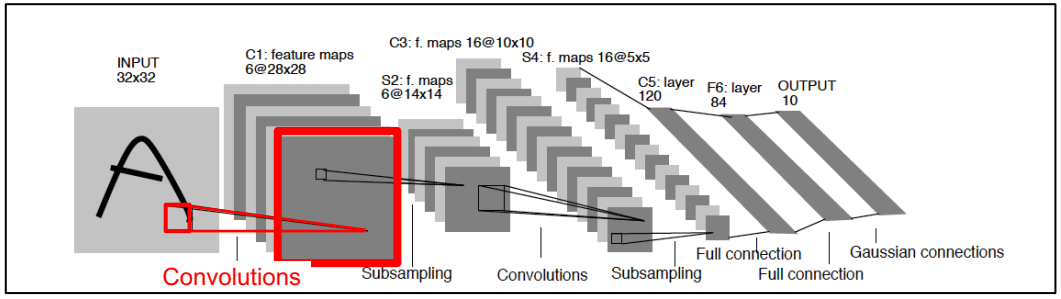


[Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11) 1998]

卷积层

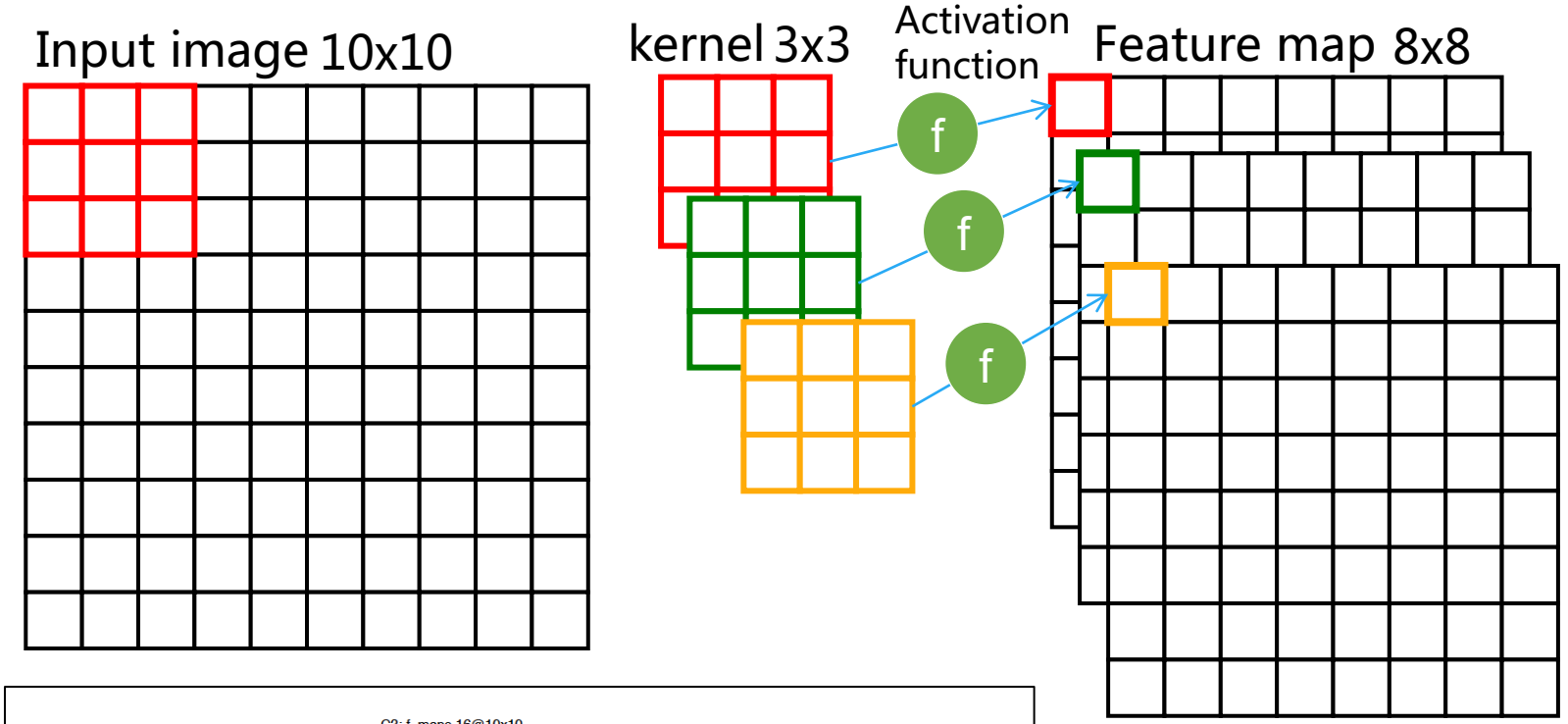
- 范例：具有3x3过滤器的10x10输入图像产生8x8输出图像

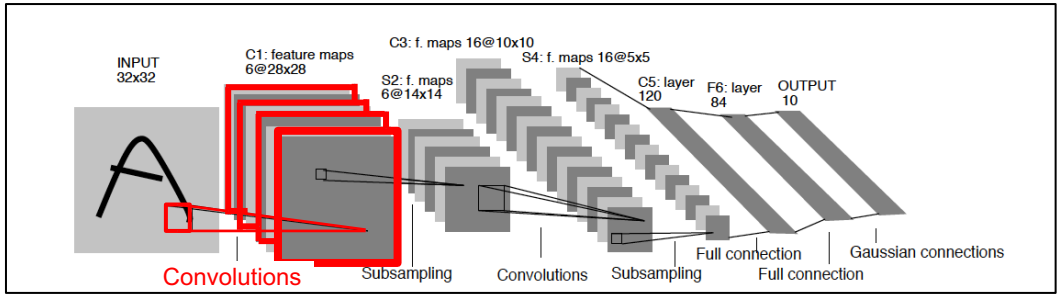




卷积层

- 范例：具有3x3过滤器的10x10输入图像产生8x8输出图像
- 三个不同的过滤器(权重不同)产生3张8×8的图像





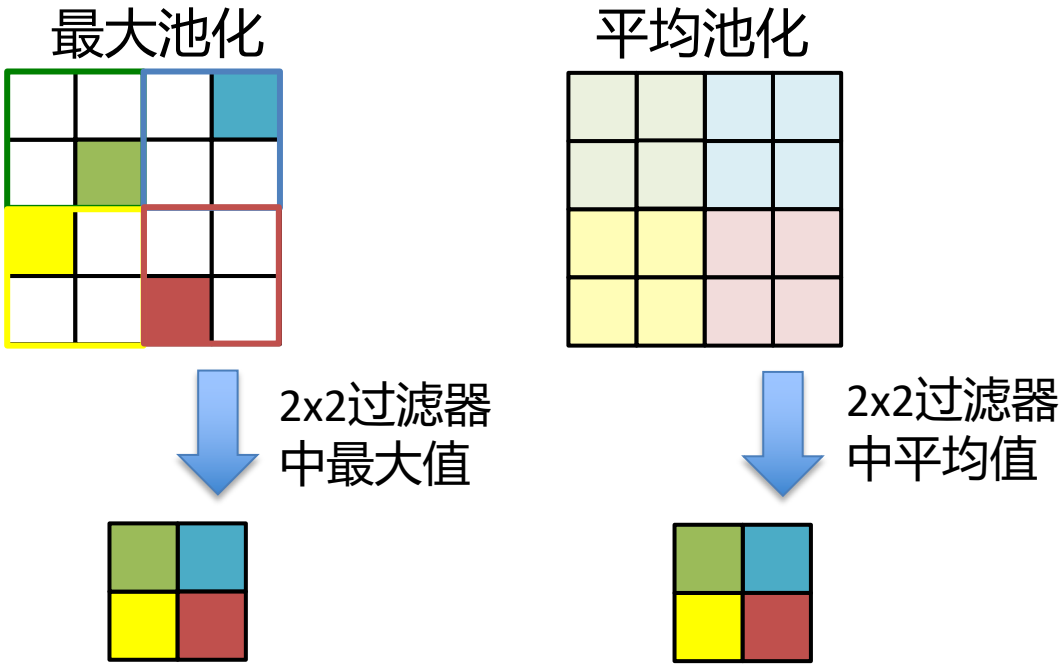
下采样池化层

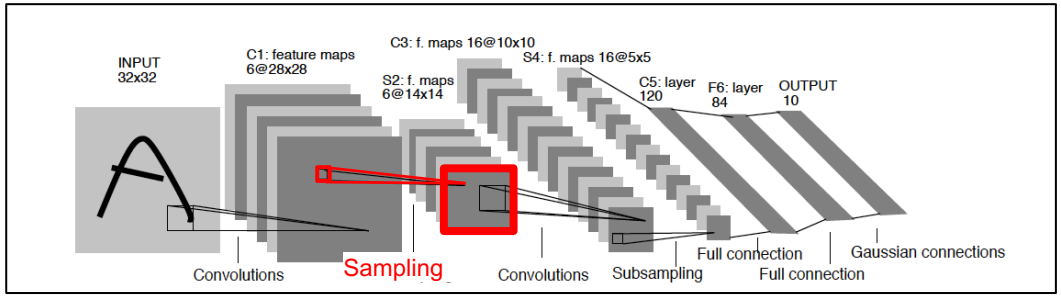
池化

- 将输入图像划分为一组非重叠矩形，并且对于每个这样的子区域，输出最大值或平均值

最大池化

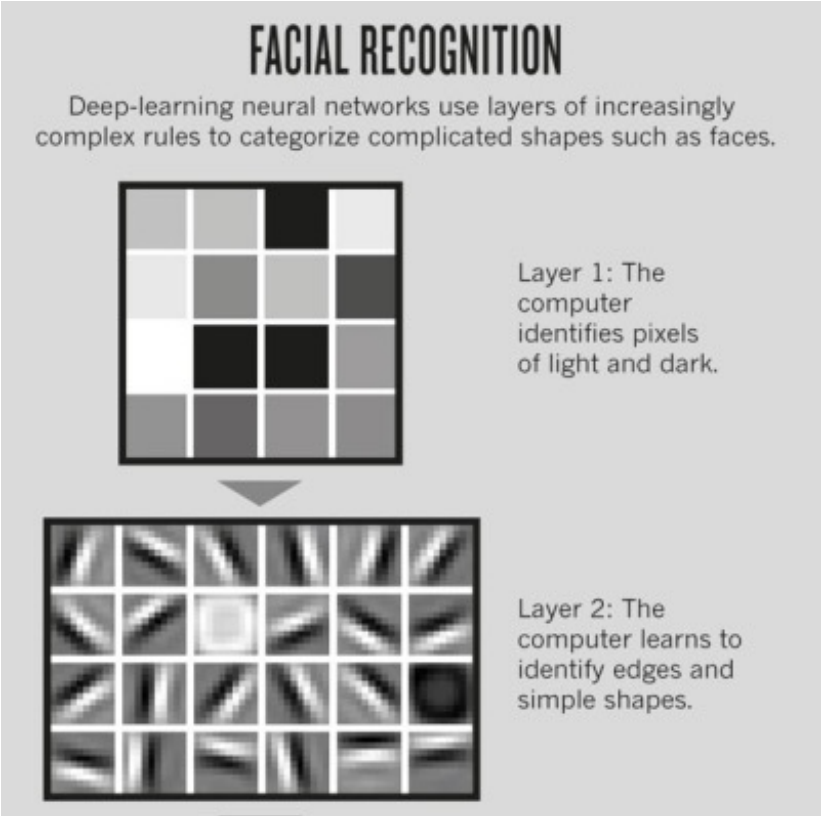
- 优点
  - 减少计算
  - 是一种获取给定兴趣区域响应最强烈的节点的方法
- 缺点
  - 可能会导致准确的空间信息丢失

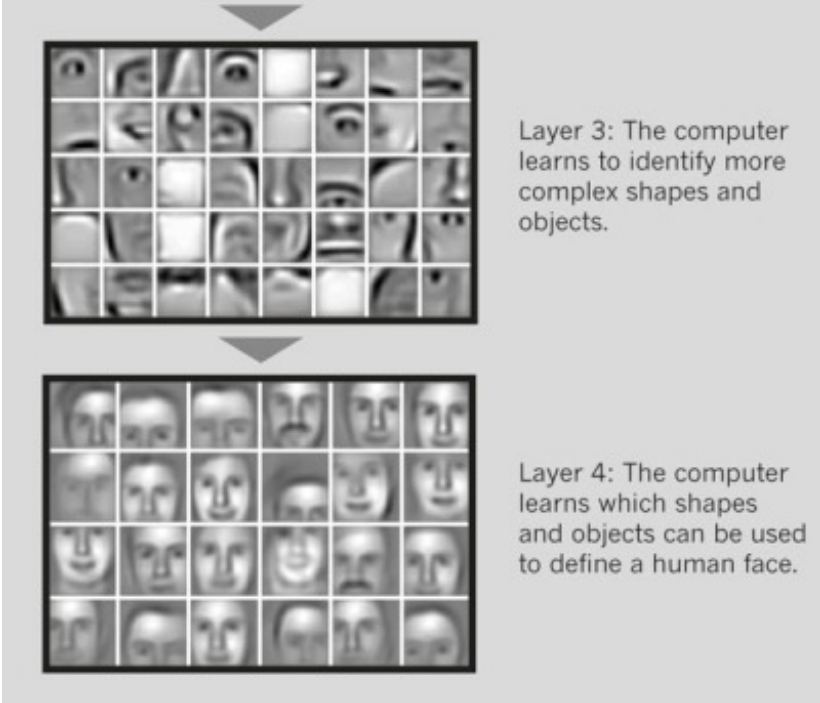




02
使用案例

使用案例：面部识别

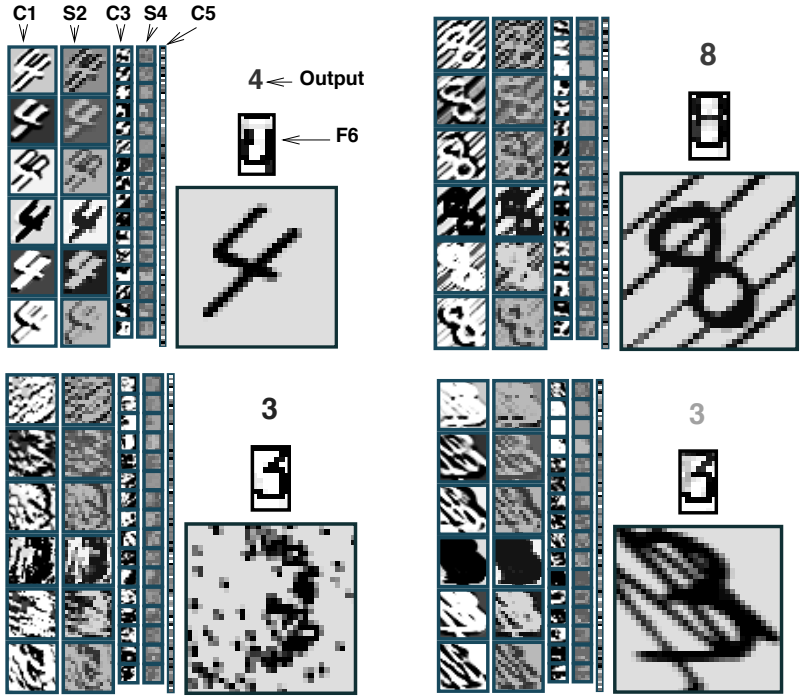

FACIAL RECOGNITION
Deep-learning neural networks use layers of increasingly complex rules to categorize complicated shapes such as faces.
Layer 1: The computer identifies pixels of light and dark.
Layer 2: The computer learns to identify edges and simple shapes.

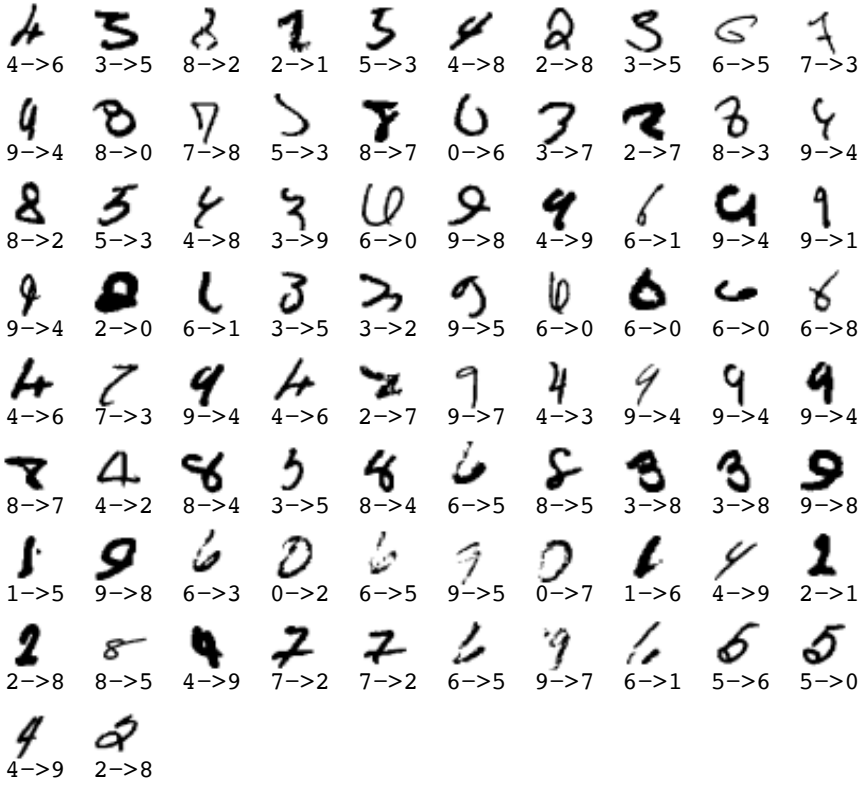

Layer 3: The computer learns to identify more complex shapes and objects.
Layer 4: The computer learns which shapes and objects can be used to define a human face.

使用案例：数字识别

- MNIST(手写数字)数据集
  - http://yann.lecun.com/exdb/mnist/
  - 60k训练和10k测试示例
- 测试错误率0.95%

用LeNet-5网络上总共只有82个错误
左边是正确的答案，右边是预测的答案

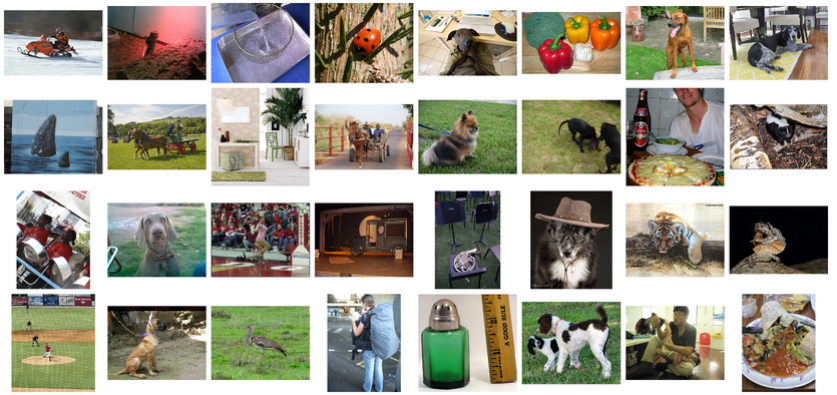


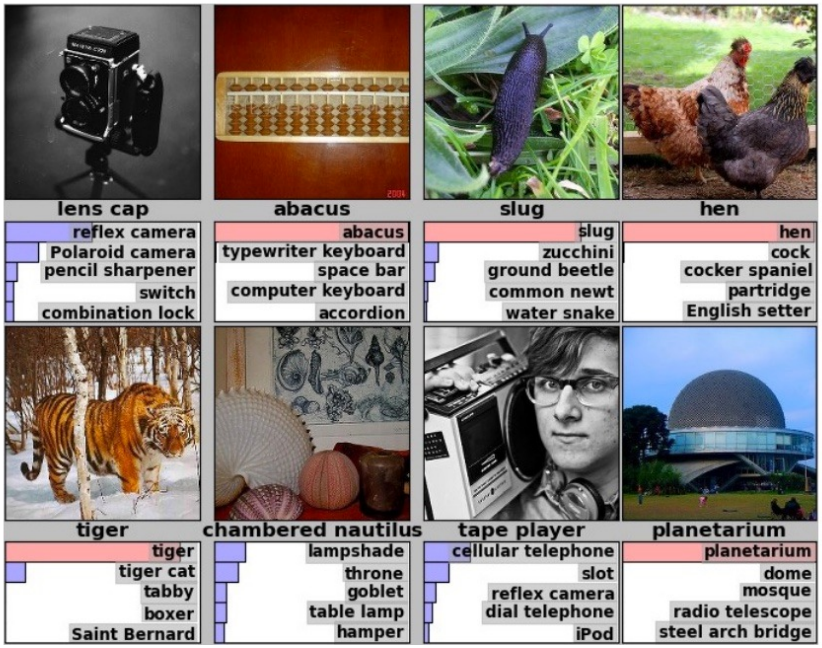


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998

更一般的图像识别

- ImageNet数据集
  - 超过15M标记的高分辨率图片
  - 大约22K类别
  - 从网站收集并由亚马逊Mechanical Turk标记
- 图像/场景分类挑战赛
  - 图像/场景分类
  - 指标： Hit@5错误率 - 对图像标签给出5个猜测





http://cognitiveseo.com/blog/6511/will-google-read-rank-images-near-future/

Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

ImageNet图像分类天梯图

2015 ResNet (ILSVRC'15) 3.57

微软残差网络，152层

Year	Codename	Error (percent)	99.9% Conf Int
2014	**GoogLeNet**	**6.66**	**6.40 - 6.92**
2014	VGG	7.32	7.05 - 7.60
2014	MSRA	8.06	7.78 - 8.34
2014	AHoward	8.11	7.83 - 8.39
2014	DeeperVision	9.51	9.21 - 9.82
2013	Clarifai†	11.20	10.87 - 11.53
2014	CASIAWS†	11.36	11.03 - 11.69
2014	Trimps†	11.46	11.13 - 11.80
2014	Adobe†	11.58	11.25 - 11.91
2013	**Clarifai**	**11.74**	**11.41 - 12.08**
2013	NUS	12.95	12.60 - 13.30
2013	ZF	13.51	13.14 - 13.87
2013	AHoward	13.55	13.20 - 13.91
2013	OverFeat	14.18	13.83 - 14.54
2014	Orange†	14.80	14.43 - 15.17
2012	SuperVision†	15.32	14.94 - 15.69
2012	**SuperVision**	**16.42**	**16.04 - 16.80**
2012	ISI	26.17	25.71 - 26.65
2012	VGG	26.98	26.53 - 27.43
2012	XRCE	27.06	26.60 - 27.52
2012	UvA	29.58	29.09 - 30.04

GoogLeNet, 22层

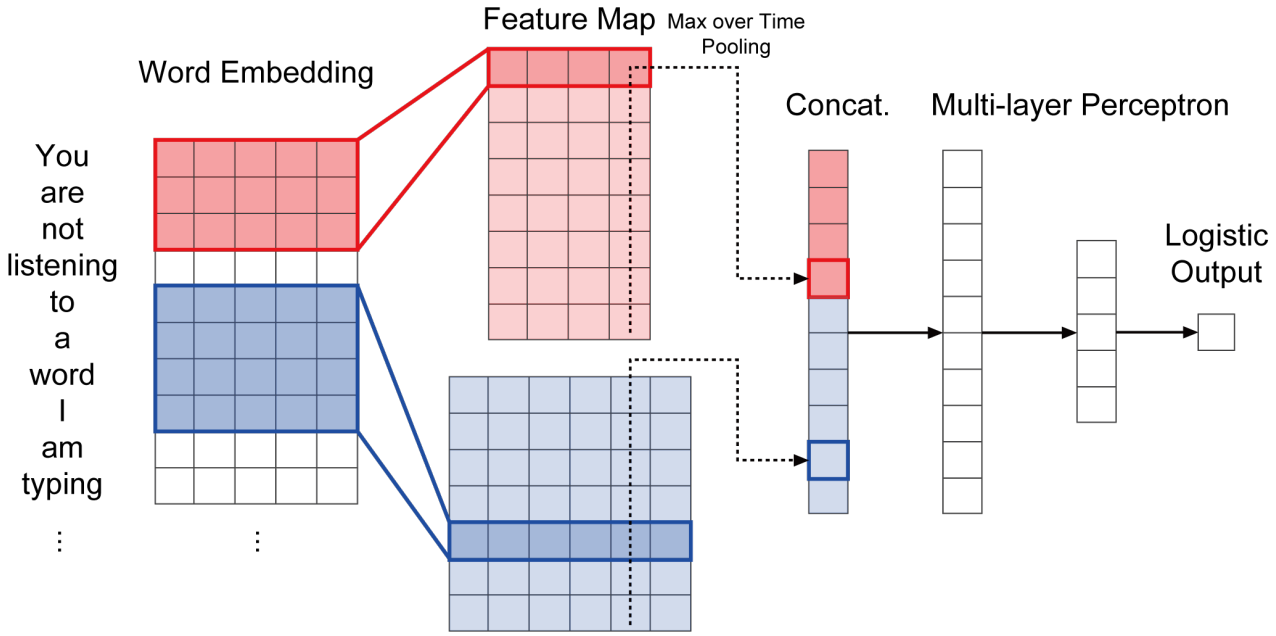
U. of Toronto, SuperVision, 7层

在一个子集非官方公布的人为错误约为5.1%

为什么还有人为错误？ 标记时，人类评价者判断它是否属于某一类（二分类）；挑战时是1000级分类问题。

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/)
Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

使用案例：文本分类



- 词嵌入：将每个单词映射到k维向量
- CNN核：nxk的矩阵，用于探索相邻的n个单词的模式(patterns)
- Max-per-time pooling：从每个核中找到最突出的文本模式
- MLP：进一步特征交互和提炼高级模式

Kim, Y. 2014. Convolutional neural networks for sentence classification. EMNLP 2014.

文本分类实验

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

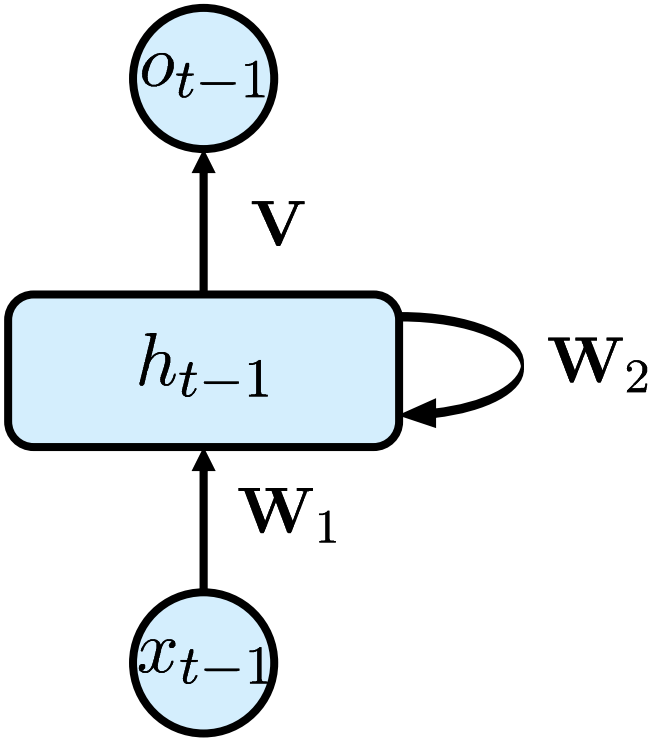
Kim, Y. 2014. Convolutional neural networks for sentence classification. EMNLP 2014.

循环神经网络

张伟楠 – 上海交通大学

循环神经网络(RNN)

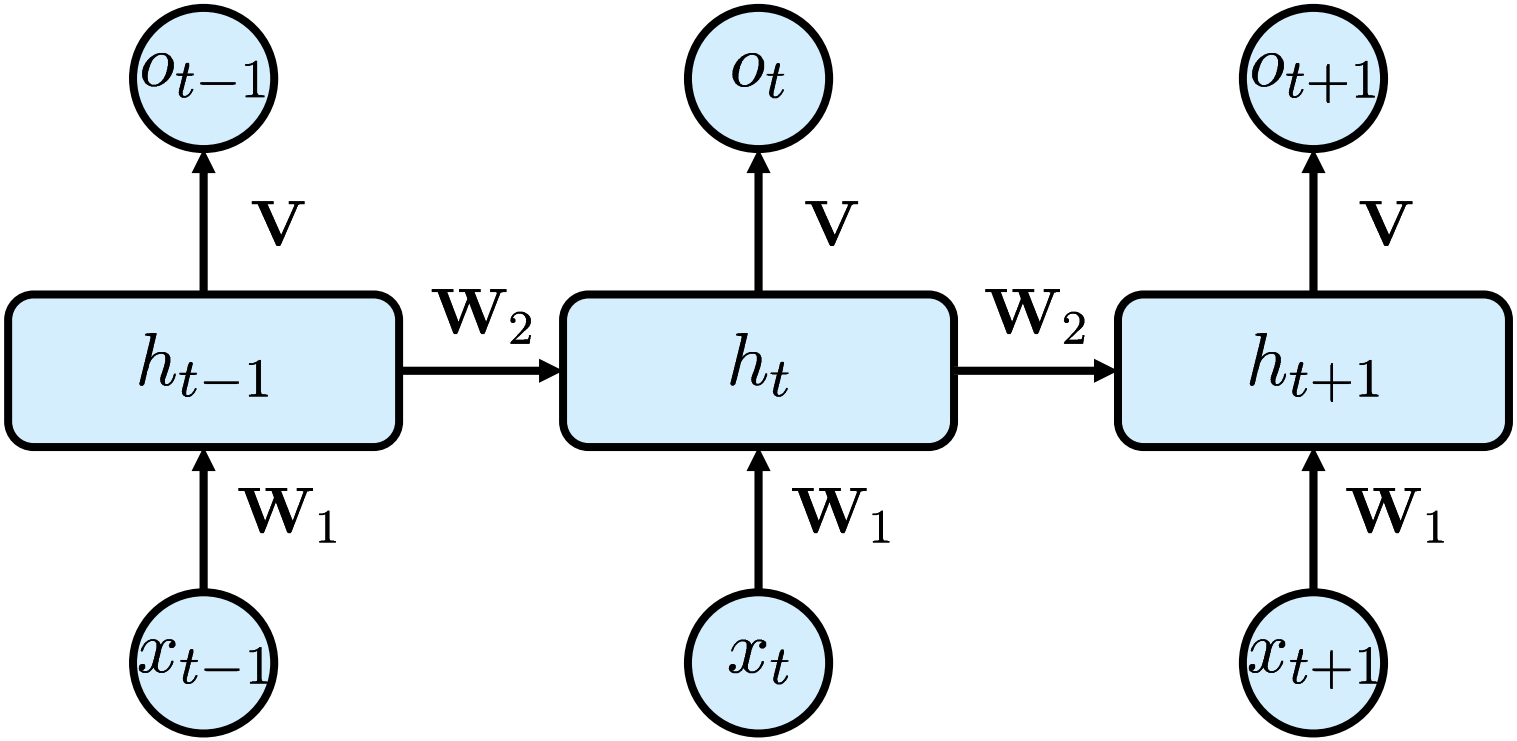
- 从前馈神经网络到循环神经网络
  - 前馈神经网络一个有向无环图，无自连边
  - 循环神经网络有一条自连的边，将隐层信息随着时间步往前传



http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns

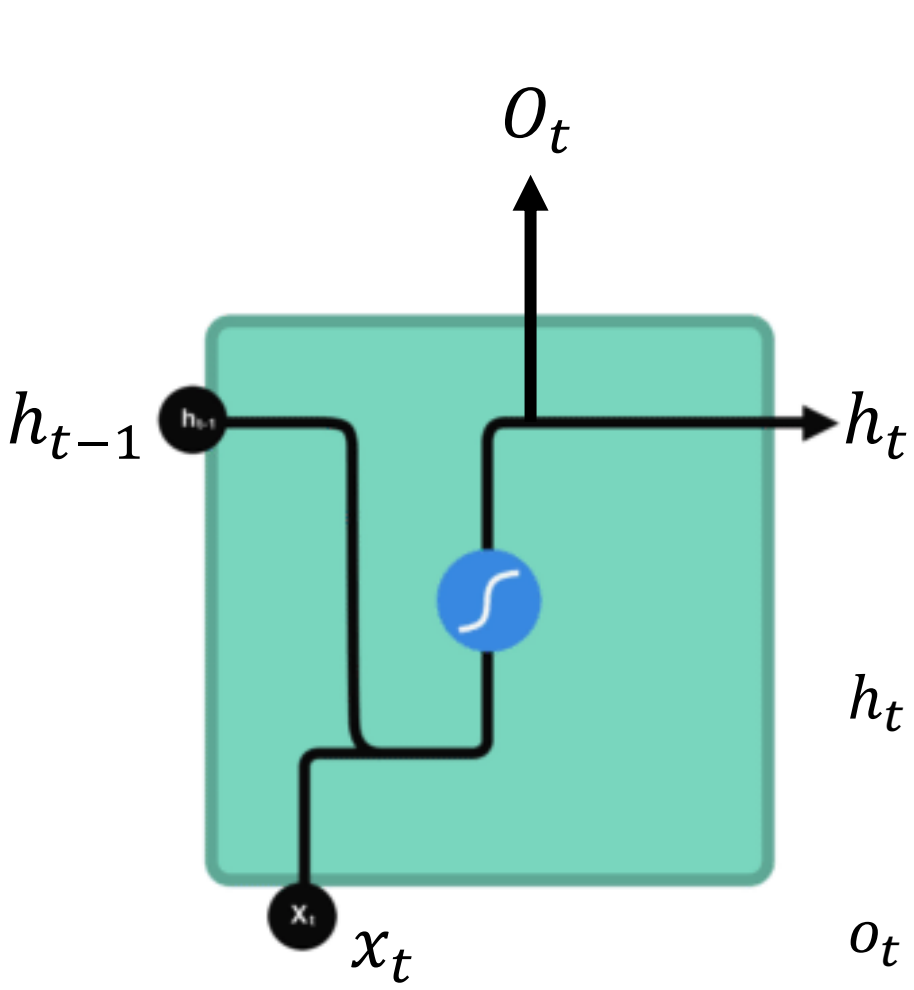
循环神经网络(RNN)

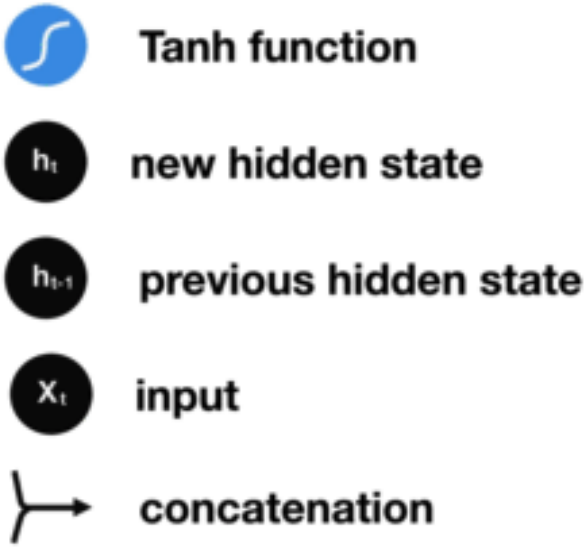
- 从前馈神经网络到循环神经网络
  - 前馈神经网络一个有向无环图，无自连边
  - 循环神经网络有一条自连的边，将隐层信息随着时间步往前传



http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns

RNN工作原理图示



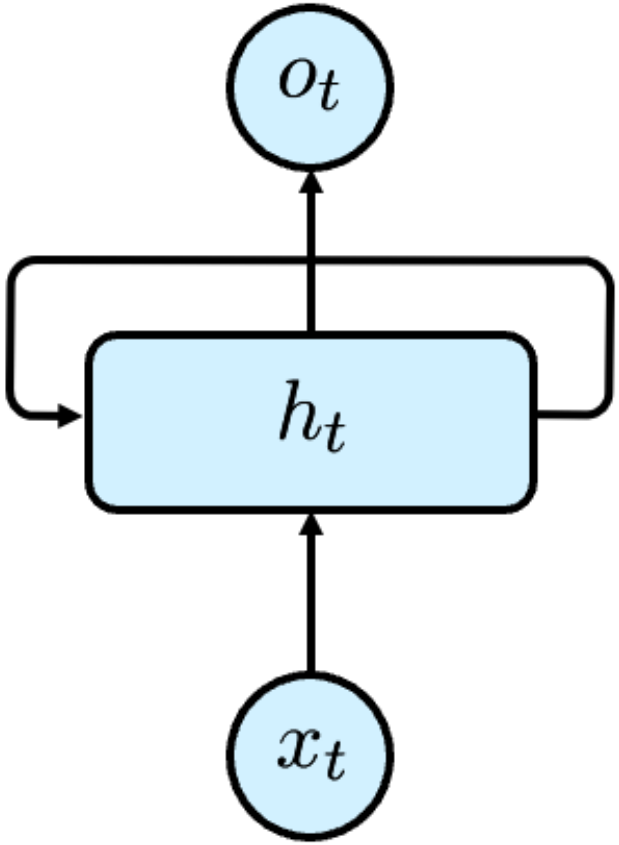


$$h_t = \tanh(\mathbf{W}_1 x_t + \mathbf{W}_2 h_{t-1})$$
$$= \tanh(\mathbf{W}[x_t, h_{t-1}])$$

$$o_t = \sigma(\mathbf{V} h_t)$$

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

RNN的工作方式

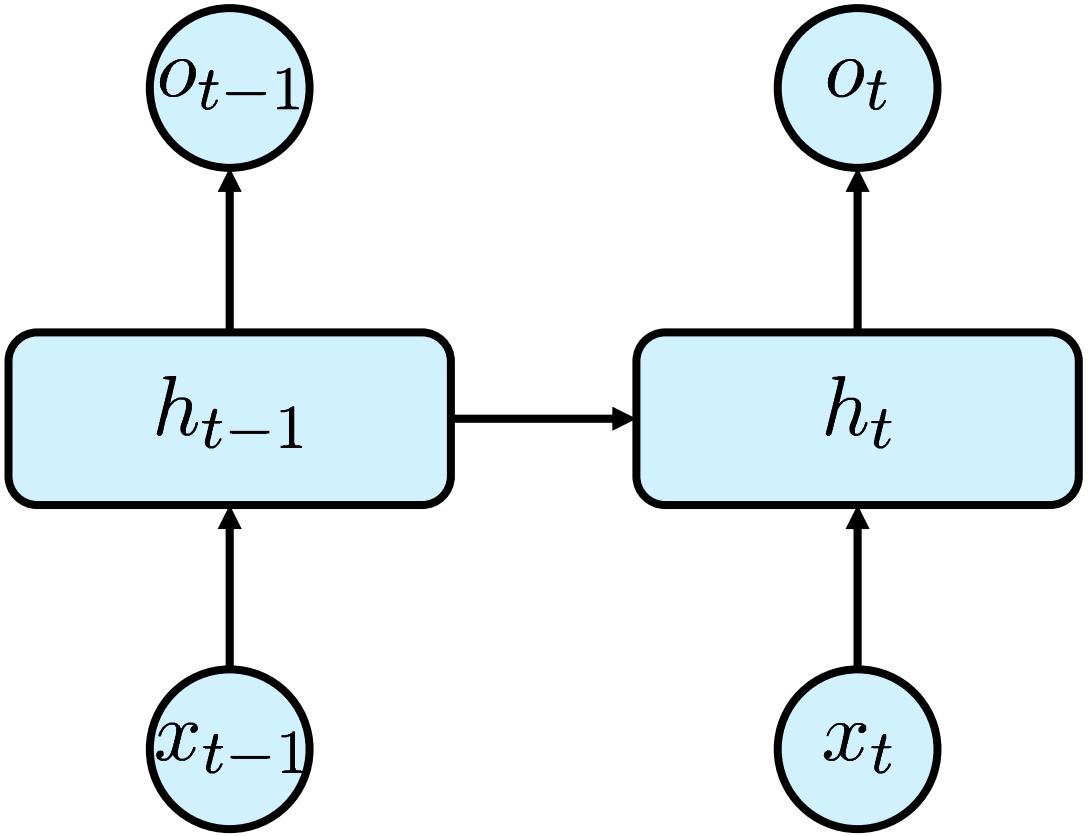


$$o_t = \sigma(\mathbf{V}h_t)$$

$$h_t = \tanh(\mathbf{W}_1 x_t + \mathbf{W}_2 h_{t-1})$$
$$= \tanh(\mathbf{W}[x_t, h_{t-1}])$$

把 x_t, h_{t-1} 接起来

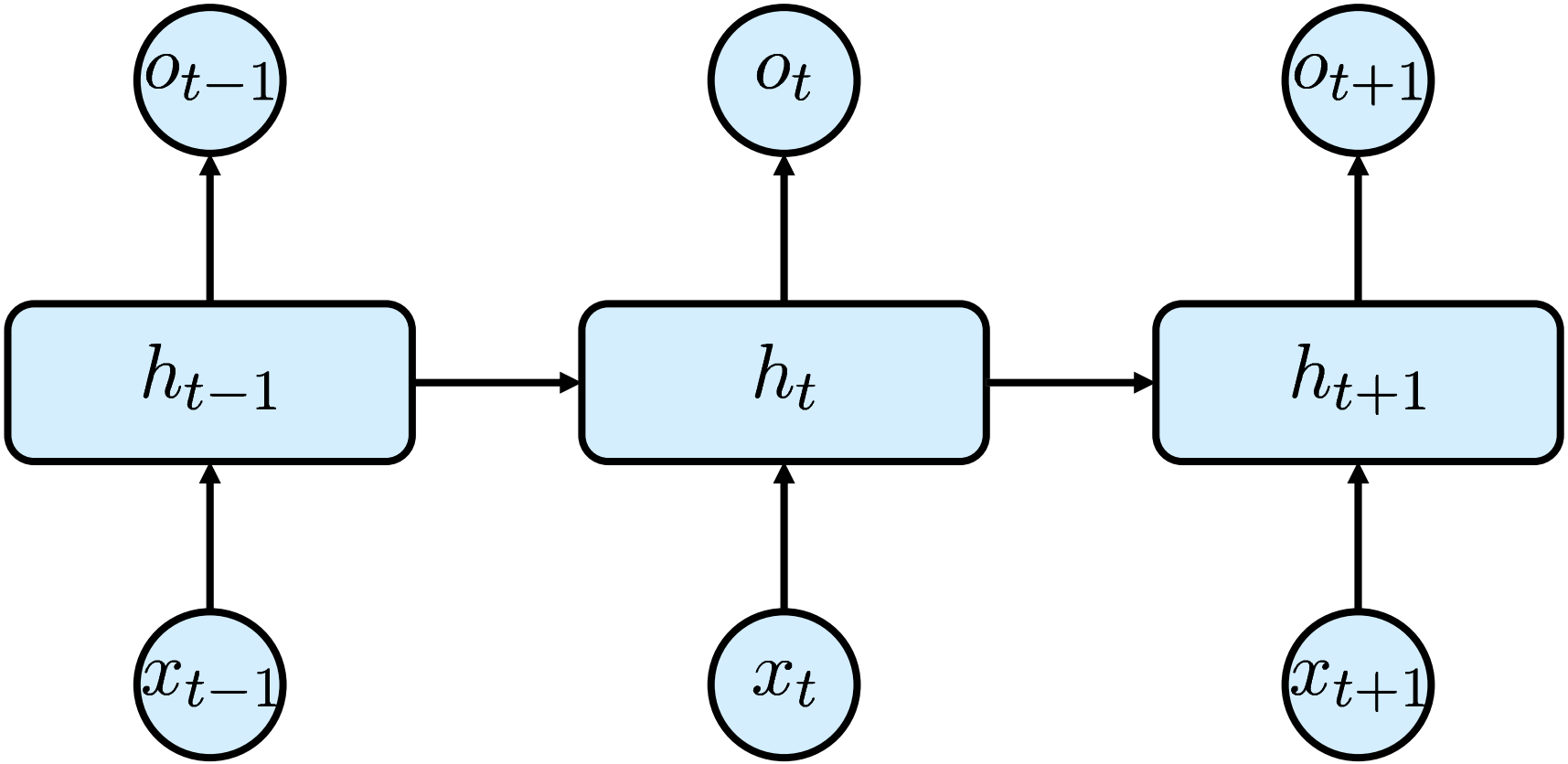
RNN的工作方式



$$o_t = \sigma(\mathbf{V}h_t)$$

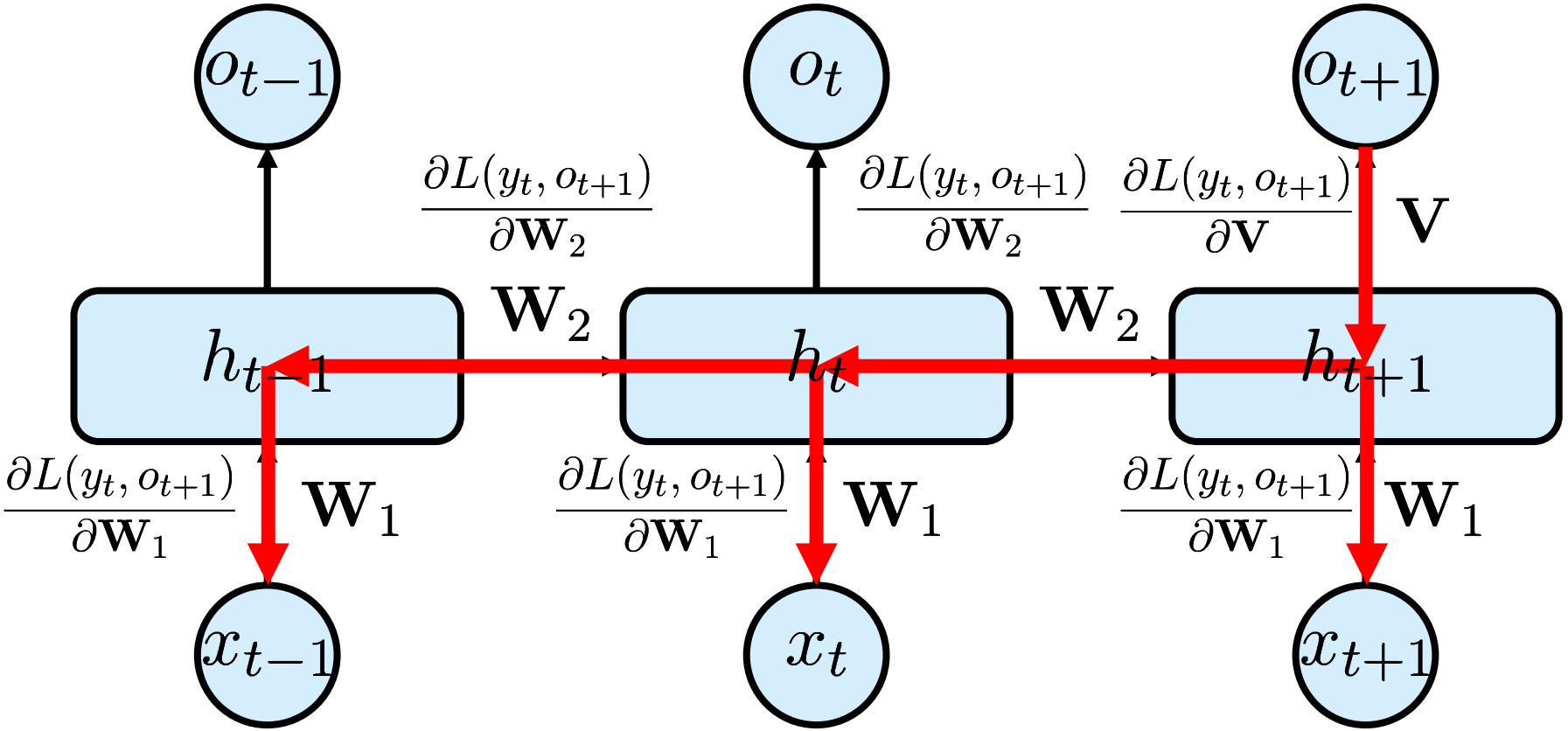
$$h_t = \tanh(\mathbf{W}[x_t, h_{t-1}])$$

RNN的工作方式



RNN的反向传播学习

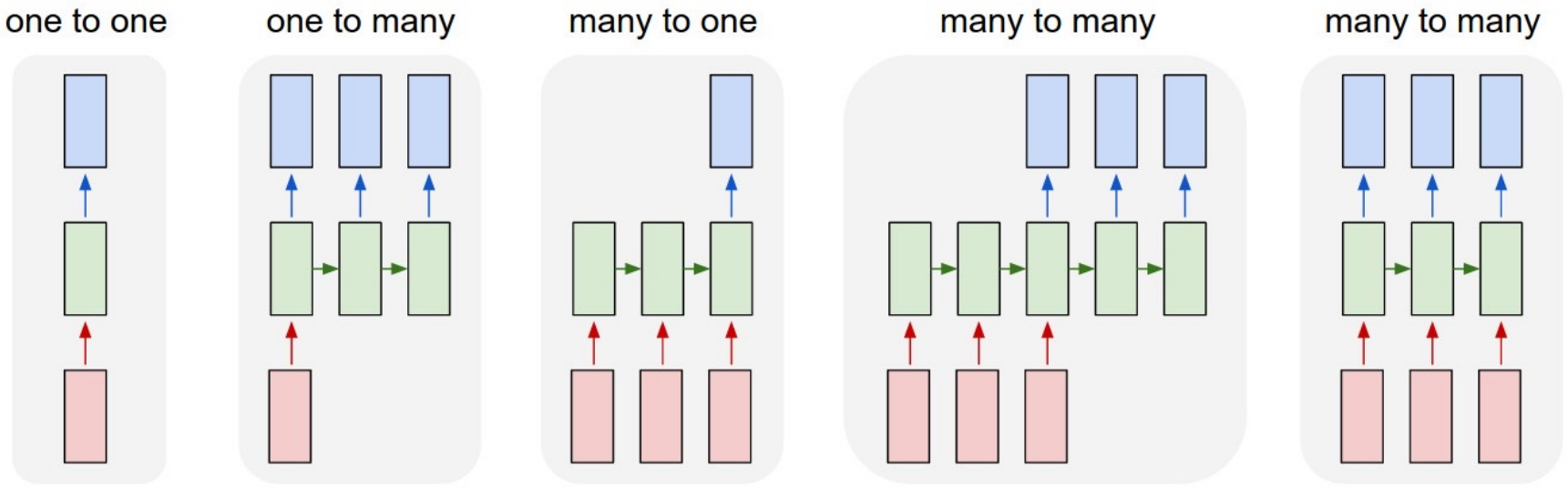
Backpropagation through time (BPTT) $L(y_{t+1}, o_{t+1})$



在上图例子中，BPTT让t+1时间步的损失梯度更新了：

- **V一次，W_2两次，W_1三次**

不同的RNNs



普通
神经网络

图片字幕
文本生成

文本分类
情感分析

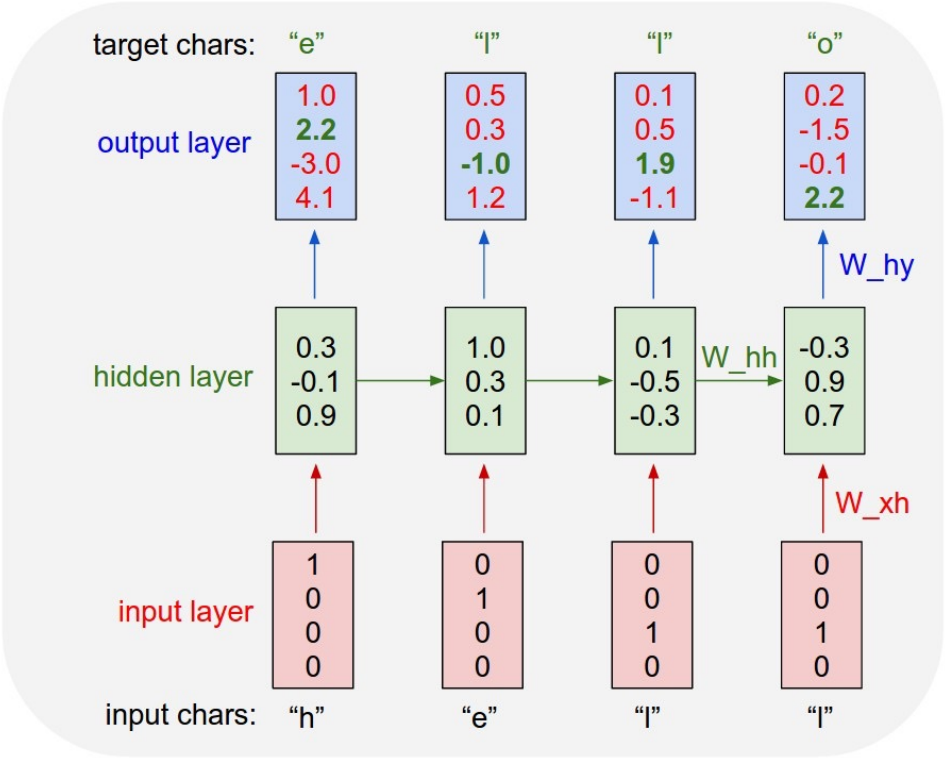
机器翻译
对话系统

股价估算
视频帧分类

- 不同任务不同结构
- 强烈推荐Andrej Karpathy的博客
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/

使用案例：语言模型

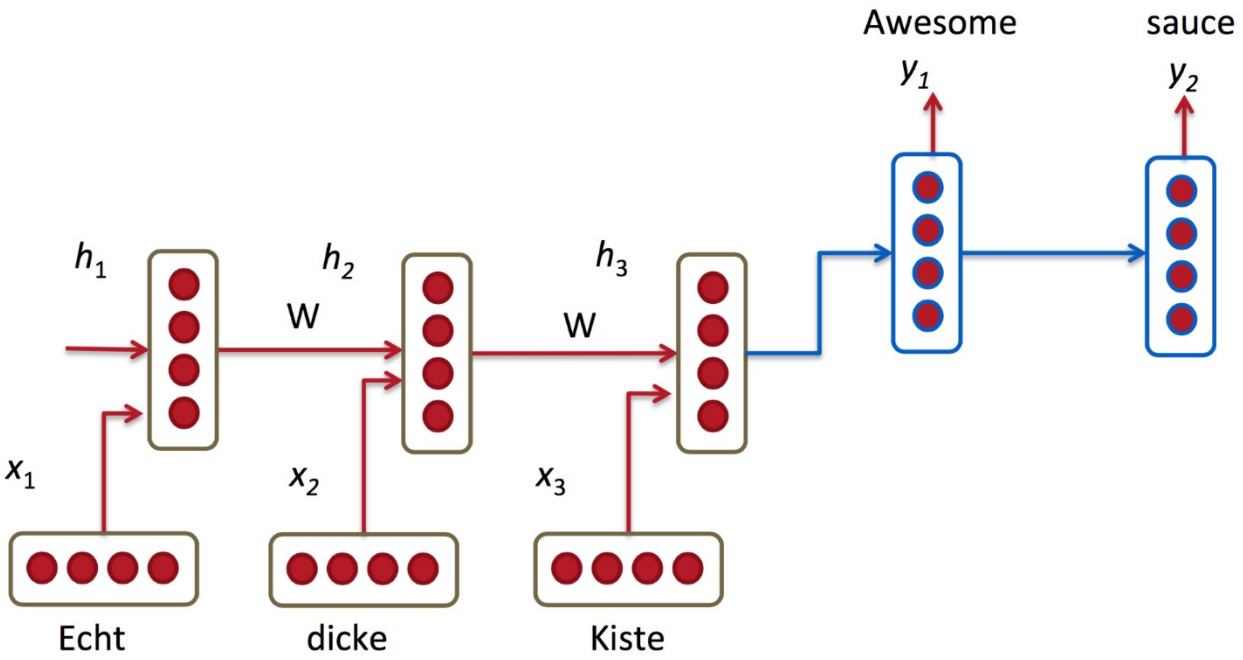
- 词级甚至字符级语言模型
  - 基于以前的单词/字符，预测下一个



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

使用案例：机器翻译

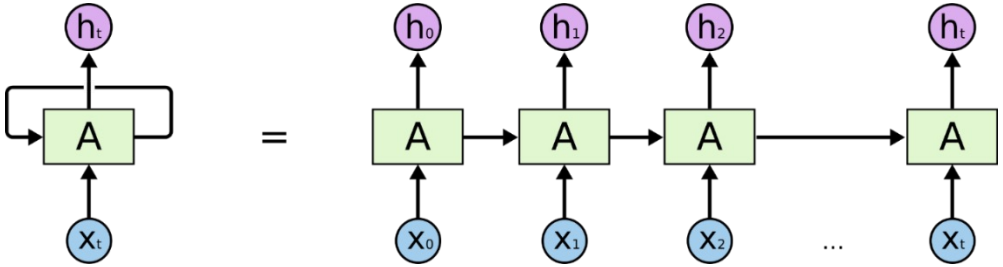
- 编码/解码RNN
  - 首先，对输入句子进行编码（转换为矢量，例如h_3）
  - 然后将矢量解码成另一种语言的句子



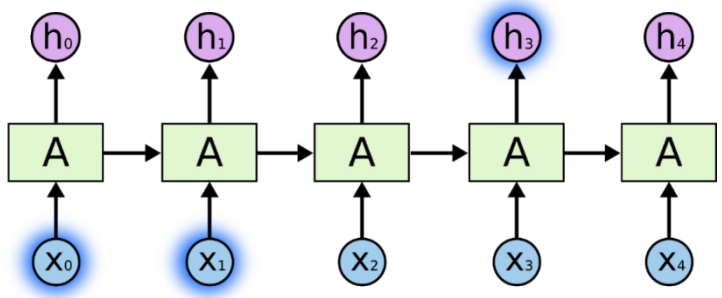
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN的问题

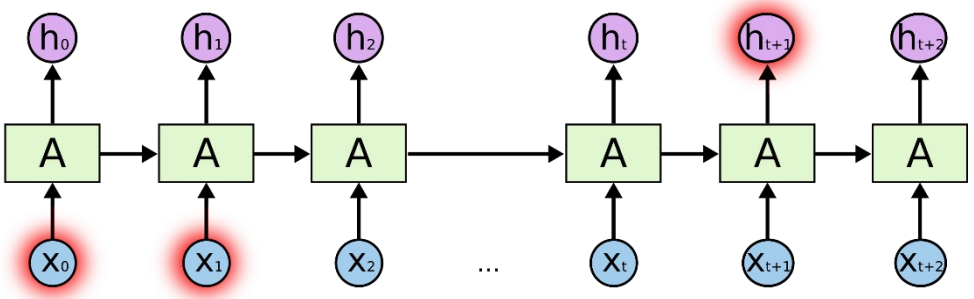
- 无法很好地利用早期信息



间隔依赖(Gap dependency)



长期依赖

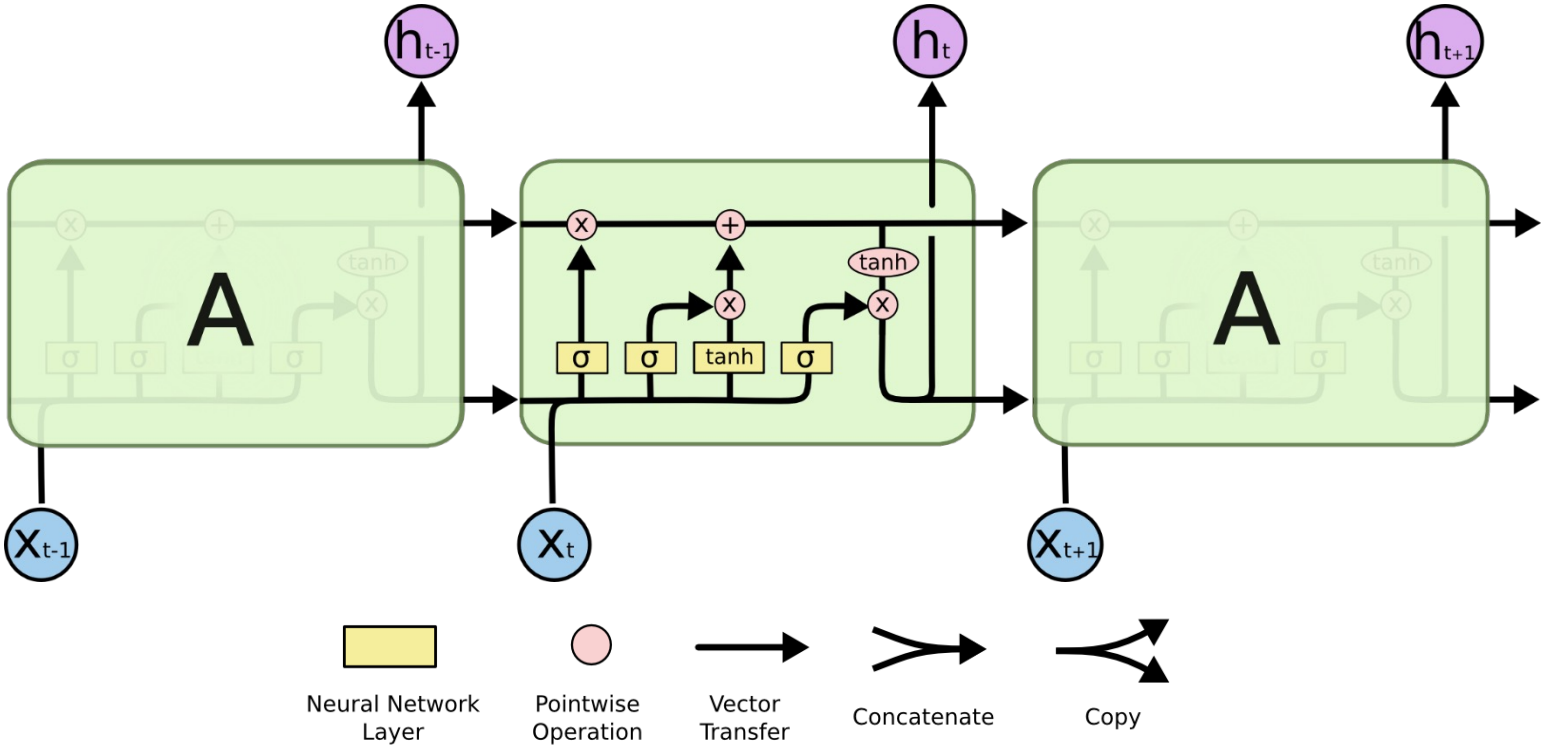


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

长短期记忆网络

张伟楠 – 上海交通大学

长短期记忆网络(LSTM)

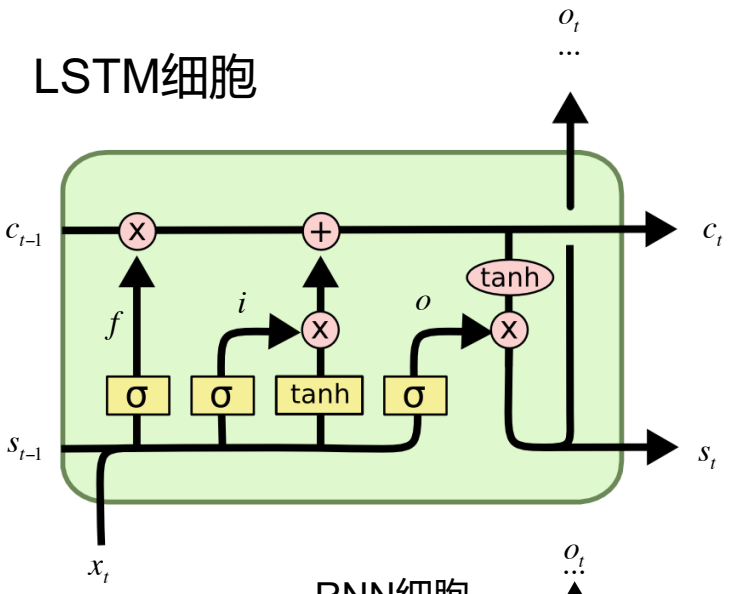


http://colah.github.io/posts/2015-08-Understanding-LSTMs
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780

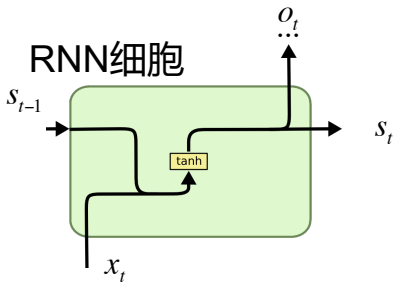
LSTM细胞

- LSTM细胞学会决定记住/忘记哪些内容

LSTM细胞



RNN细胞



σ：sigmoid(将信号控制在0和1之间；o:元素对位相乘)

$$i = \sigma(x_t U^i + s_{t-1} W^i)$$ 输入门

$$f = \sigma(x_t U^f + s_{t-1} W^f)$$ 遗忘门

$$o = \sigma(x_t U^o + s_{t-1} W^o)$$ 输出门

$$g = tanh(x_t U^g + s_{t-1} W^g)$$ “候选”隐藏状态

$$c_t = c_{t-1} \circ f + g \circ i$$ 细胞内存

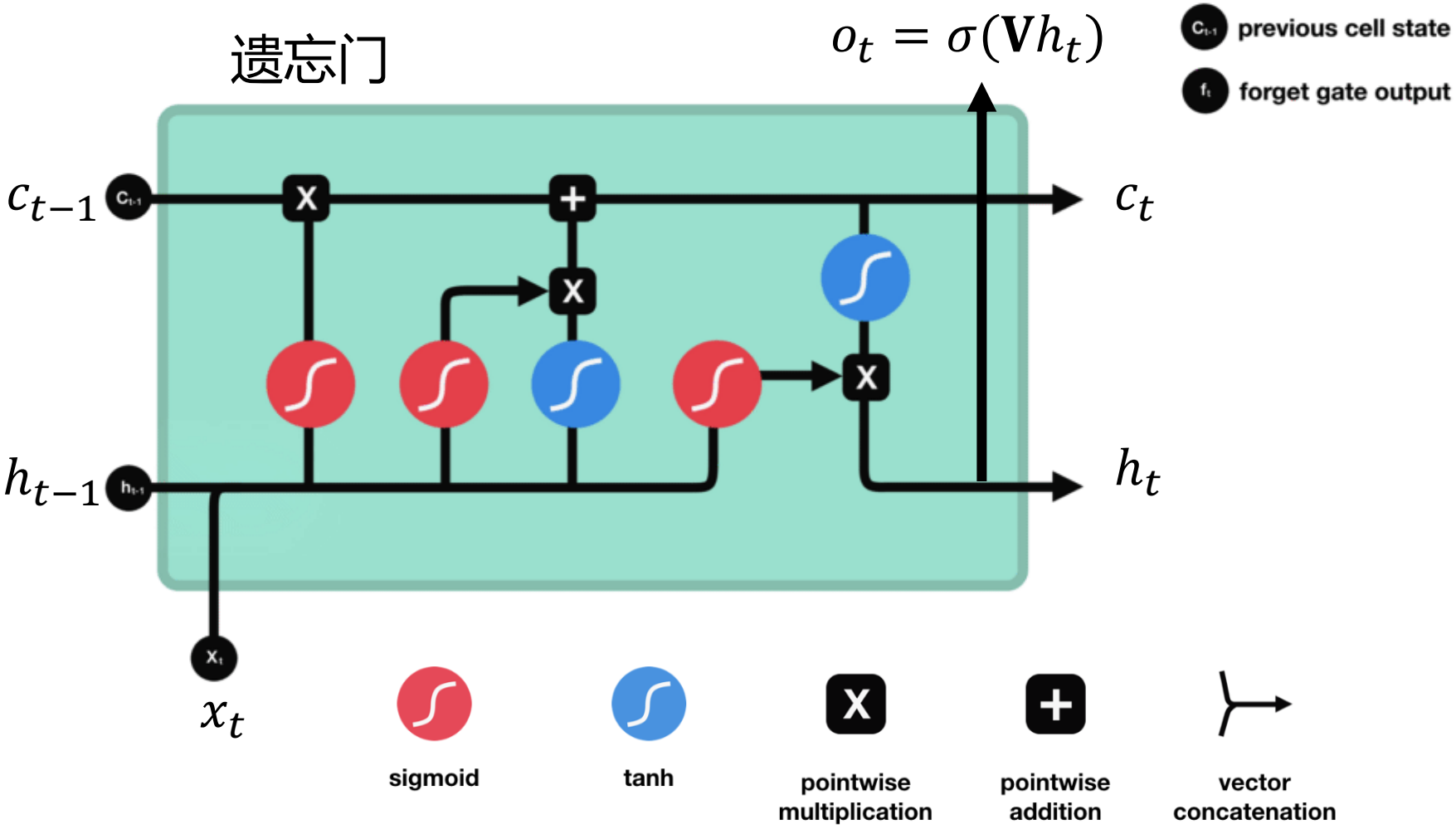
$$s_t = \tanh(c_t) \circ o$$ 隐藏状态

$$s_t = \tanh(x_t \mathbf{U} + s_{t-1} \mathbf{W})$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs

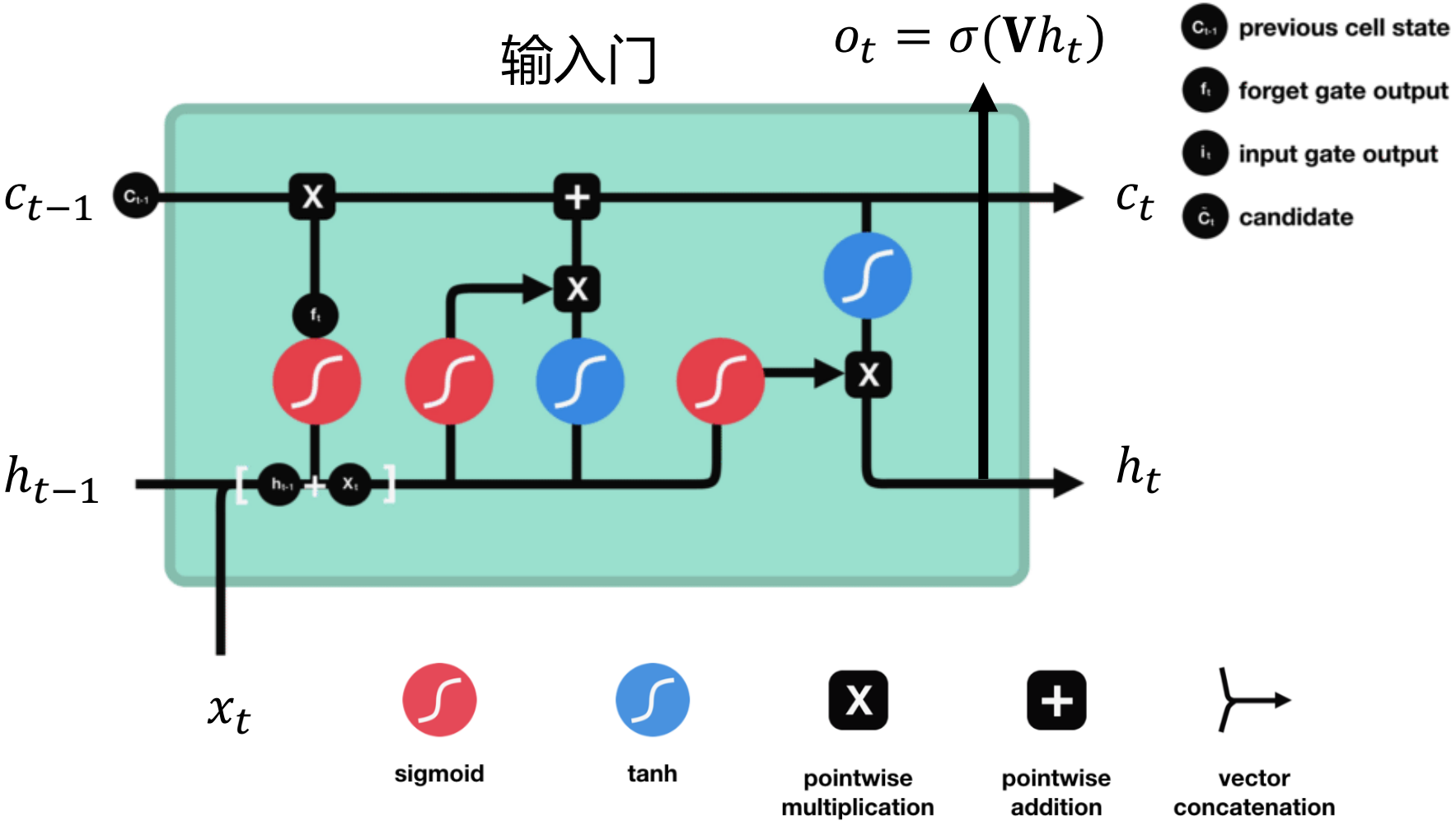
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780

LSTM工作原理图示



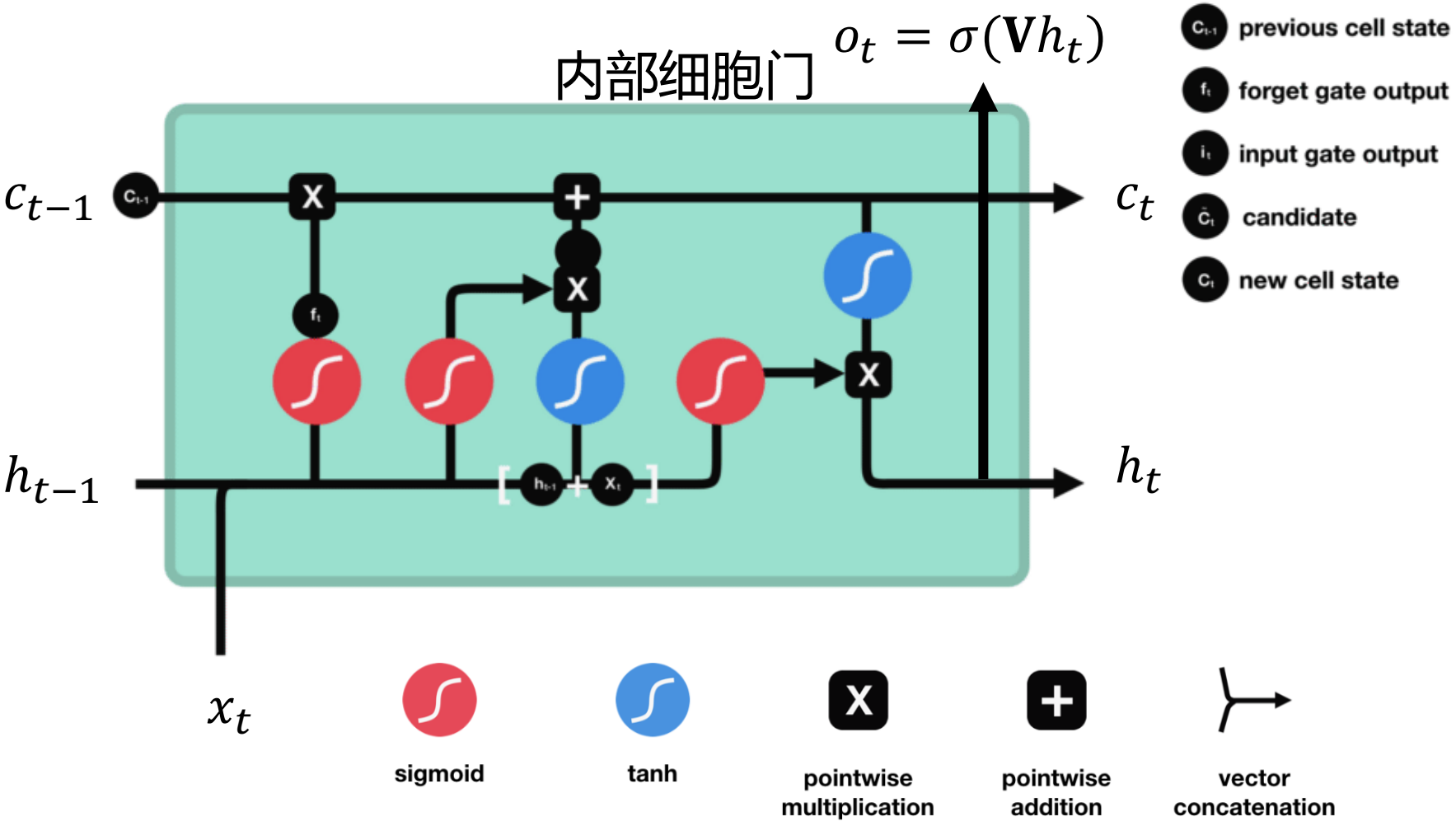
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM工作原理图示



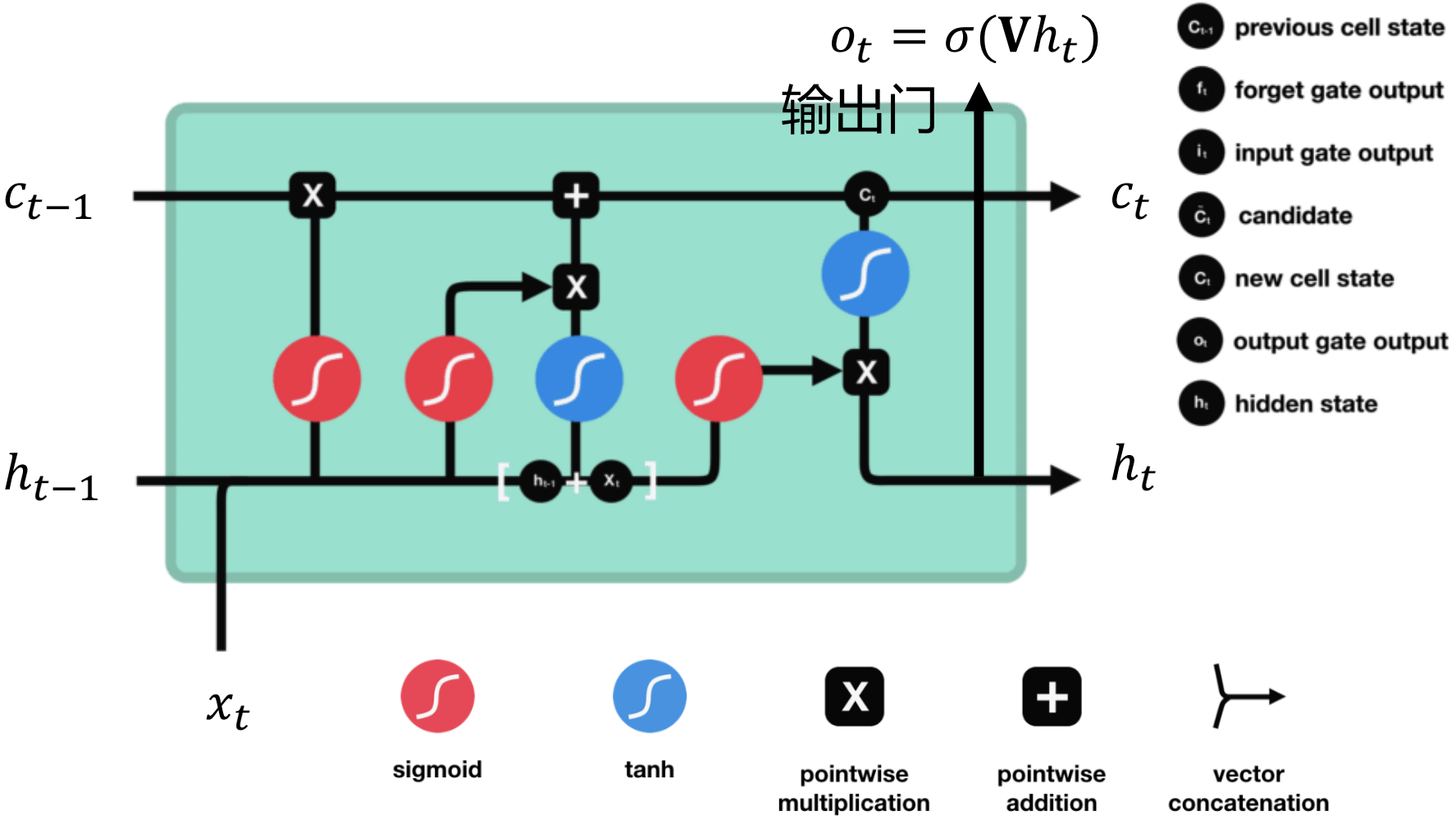
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM工作原理图示



https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM工作原理图示



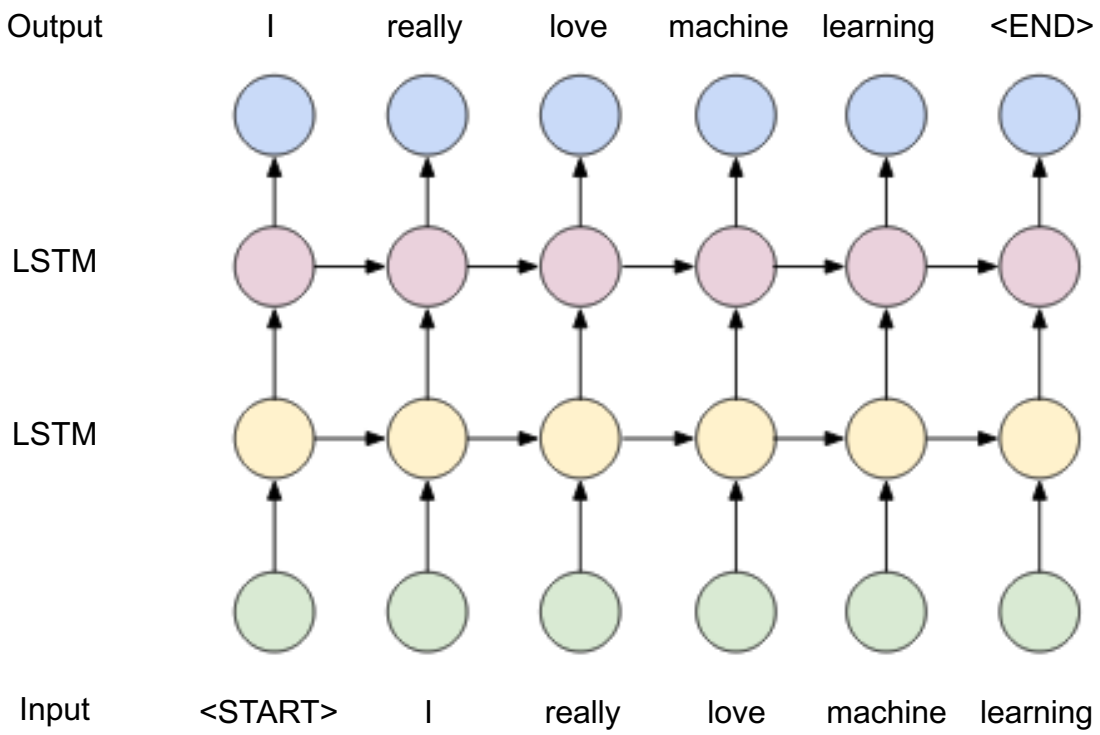
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

长短期记忆网络

使用案例

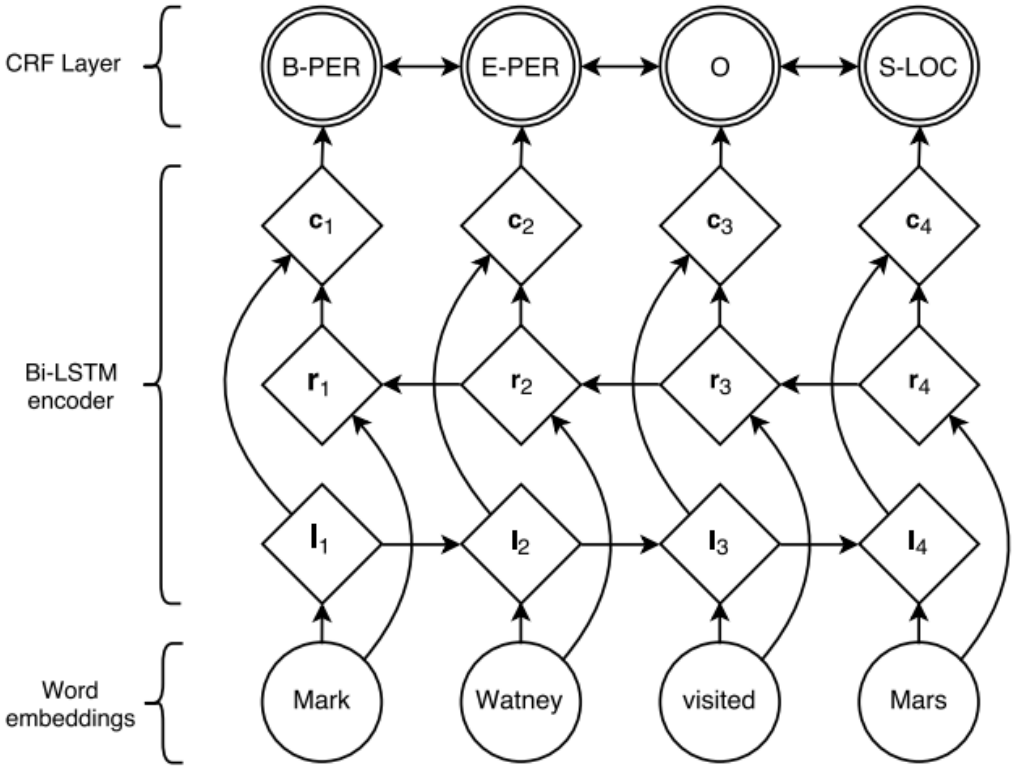
张伟楠 – 上海交通大学

使用案例：文本生成



- 关于字符级文本生成的演示
  - http://cs.stanford.edu/people/karpathy/recurrentjs/

使用案例：命名实体识别



Guillaume Lample et al. Neural Architectures for Named Entity Recognition. NAACL-HLT

词嵌入(Word Embedding)

- 从词袋(bag of words)到词嵌入
  - 用m维的实值向量($\mathbb{R}^m$)表示单词（概念）

$$v(\text{"cat"}) = (0.2, -0.4, 0.7, \cdots)$$

$$v(\text{"mat"}) = (0.0, 0.6, -0.1, \cdots)$$

- 连续词袋(CBOW)模型(word2vec)
  - 输入/输出 词 x/y 是独热编码(one-hot encoded)
  - 所有输入词共享隐藏层

词用N维向量表示

隐藏节点：

$$h = \frac{1}{C} W \cdot (x_1 + x_2 + \cdots + x_C)$$

$$= \frac{1}{C} \cdot (v_{w_1} + v_{w_2} + \cdots + v_{w_C})$$

交叉熵损失：

$$E = -\log p(\omega_O \mid \omega_{I,1}, \cdots, \omega_{I,C})$$

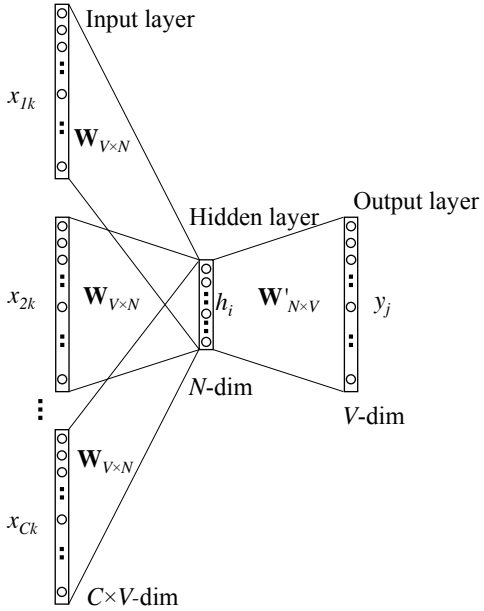
$$= -v'^{T}_{\omega_O} \cdot h + \log \sum_{j'=1}^{V} \exp(v'^{T}_{w_j} \cdot h)$$

梯度更新：

$$v'^{(new)}_{\omega_j} = v'^{(old)}_{\omega_j} - \eta \cdot e_j \cdot h \quad \text{for } j = 1,2,\cdots,V.$$

$$v^{(new)}_{\omega_{I,c}} = v^{(old)}_{\omega_{I,c}} - \frac{1}{C} \cdot \eta \cdot EH \quad \text{for } c = 1,2,\cdots,C.$$

V: 词汇量;
C: 输入词数量;
v: 输入矩阵W的行向量;
v' ：输出矩阵W' 的行向量



连续词袋(CBOW)模型

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot \omega'_{ij} := EH_i$$

Rong, Xin. "word2vec parameter learning explained." arXiv preprint arXiv:1411.2738 (2014)
Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

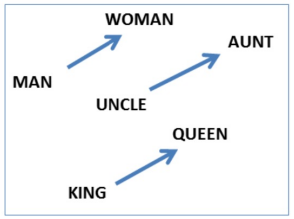
词嵌入的卓越属性

- 使用词向量进行简单的代数运算

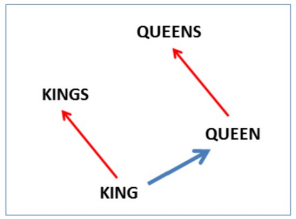
  使用X = v("biggest")- v("big")+ v("small") 作为查询，并基于余弦距离搜索向量空间中最近的点作为v(“smallest”)

  v("woman")−v("man") ≃ v("aunt")−v("uncle")

  v("woman")−v("man") ≃ v("queen")−v("king")



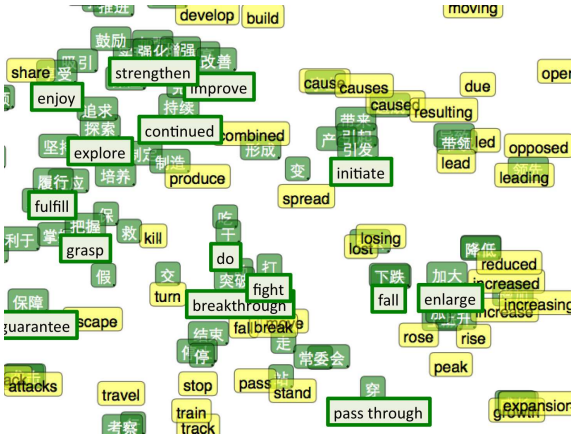
性别关系的矢量偏移



两个单词的单/复数关系

通过将两个单词向量相减来定义词的关系，并将结果加上另一个单词。例如，巴黎 (Paris)- 法国(France)+意大利(Italy)=罗马(Rome)。

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza



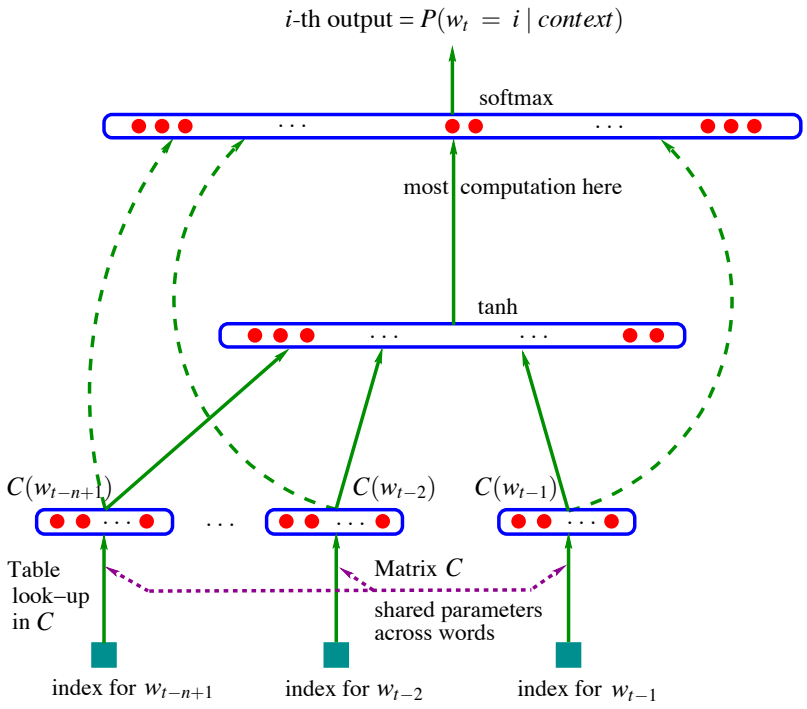
Zou, Will Y., et al. "Bilingual Word Embeddings for Phrase-Based Machine Translation." EMNLP. 2013.
Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic Regularities in Continuous Space Word Representations." HLT-NAACL. 2013.

神经语言模型

- N-gram模型
  - 给定最后n-1个单词的组合（上下文），构造下一个单词的条件概率：

$$\hat{P}(w_t|w_1^{t-1}) \approx \hat{P}(w_t|w_{t-n+1}^{t-1})$$

  - 其中 $w_i^j = (w_i, w_{i+1}, \cdots, w_{j-1}, w_j)$
- 神经语言模型
  - 将每个单词与用于词嵌入的分布式词向量相关联
  - 用词向量表达单词序列的联合概率函数
  - 同时学习词特征向量和概率函数的参数



Bengio, Yoshua, et al. "Neural probabilistic language models." Innovations in Machine Learning. Springer Berlin Heidelberg, 2006. 137-186.

基于RNN的语言模型

- 前馈神经网络的局限性：必须固定上下文长度
- RNN解决办法
  - 维护一个上下文表示并随着时间更新

x(t) 是输入向量: 通过拼接当前单词的词嵌入$w(t)$和在时间$t-1$时的隐藏状态$s(t-1)$而形成

$$x(t) = [w(t), s(t-1)]$$

s(t)表示网络状态(隐藏层):

$$S_j(t) = f\left(\sum_i x_i(t) u_{ji}\right)$$

输出被表示为y(t):

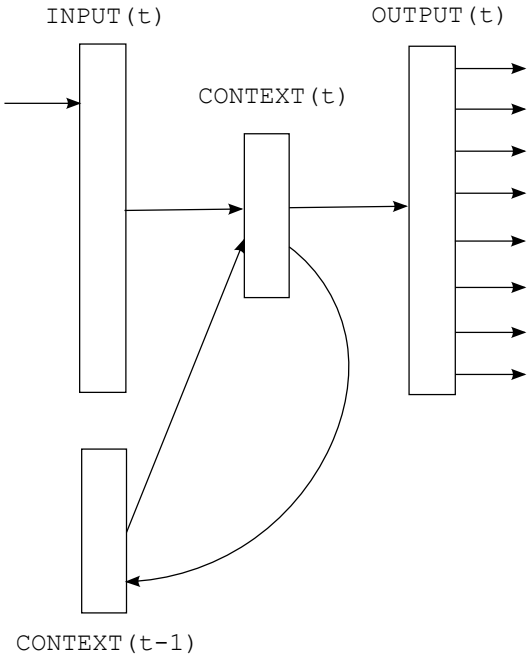
$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right)$$

对隐藏层使用sigmoid

$$f(z) = \frac{1}{1+e^{-z}}$$

输出层使用softmax

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

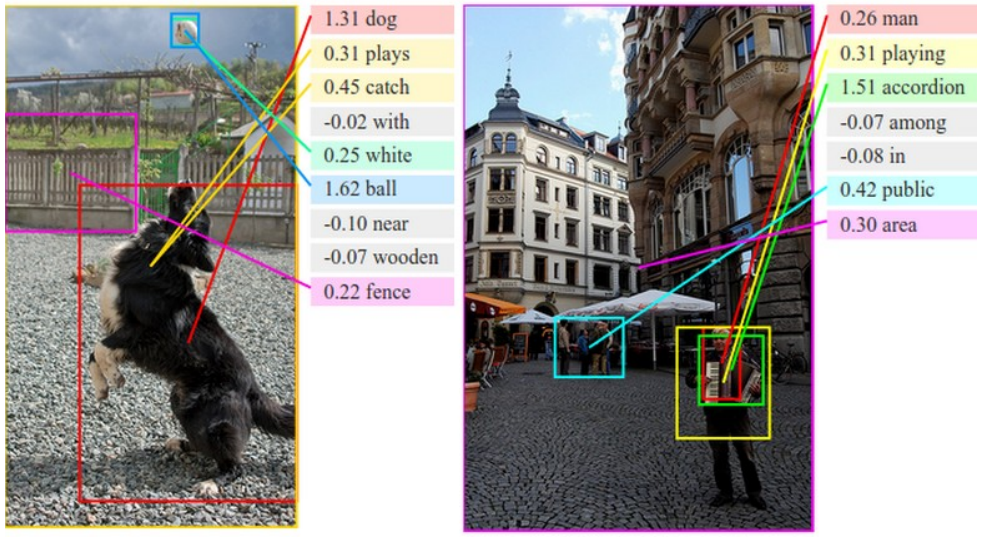


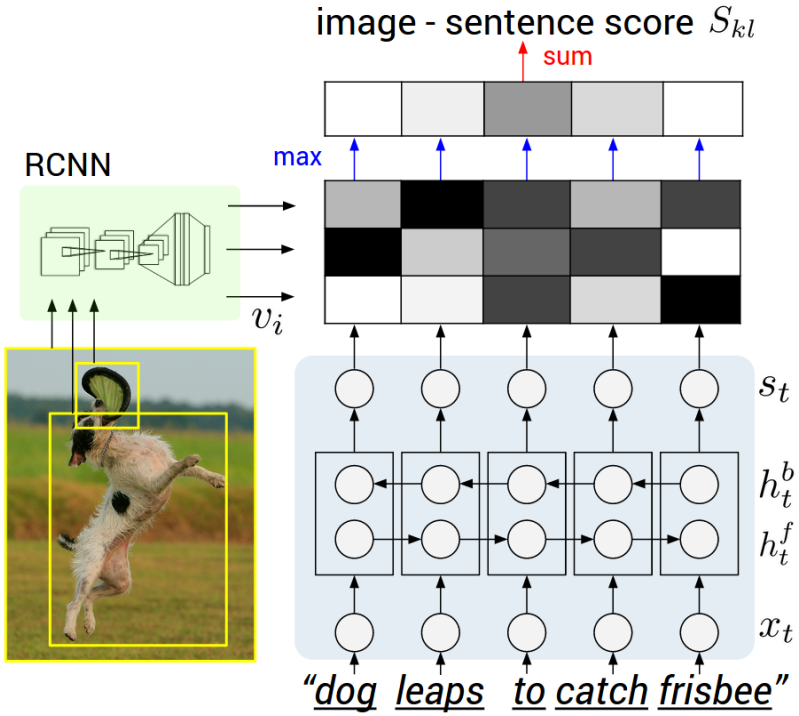
Elman的RNN语言模型

Mikolov, Tomas, et al. "Recurrent neural network based language model." INTERSPEECH. Vol. 2. 2010.
Elman J L. Finding structure in time[J]. Cognitive science, 1990, 14(2): 179-211.

学习视觉语言对齐

- 区域CNN +双向RNN
  - 通过一个共同的多模式嵌入空间将这两种方式联系起来



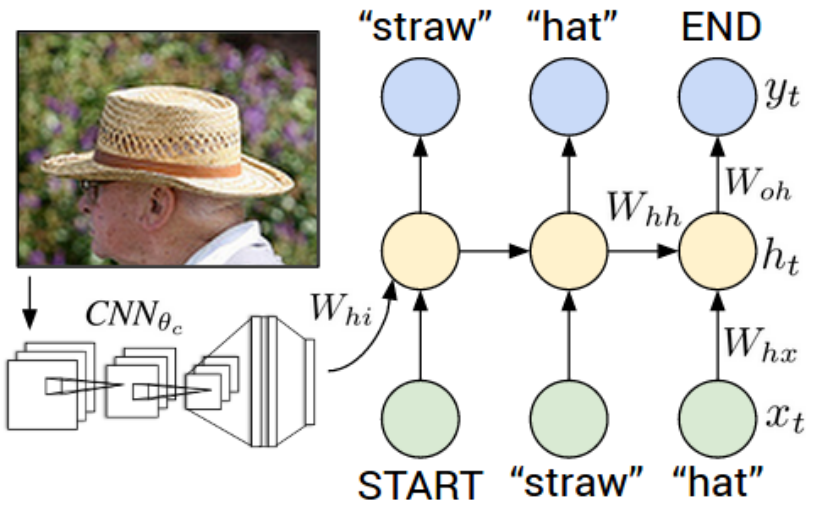


Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

学习生成图像描述

- 用图像训练CNN+用句子训练RNN
  - RNN接受一个单词和之前的上下文，输出下一个单词的分布
  - 图片信息一开始就输入到RNN中
  - START和END是特殊标志符

"two young girls are playing with lego toy."



Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

神经网络及深度学习总结

- 通用近似：双层神经网络可以逼近任何函数
- 到目前为止，反向传播是多层神经网络最重要的训练方案
- 深度学习，即用大数据训练的深度神经网络，效果非常好
  - MLP（DNN）、CNN、RNN、LSTM
  - 感兴趣的同学可以关注Attention和Transformer
- 结合其他机器学习模型的神经网络取得了进一步成功

参考资料

- Geoffery Hinton教授的Coursera课程
  - https://www.coursera.org/learn/neural-networks
- 汪军教授在UCL的深度学习教程
  - http://www.slideshare.net/JunWang5/deep-learning-61493694
- 李飞飞教授在斯坦福的CS231n课程
  - http://cs231n.stanford.edu/
- 俞凯教授在上海交大的课程
  - http://speechlab.sjtu.edu.cn/~kyu/node/10
- Michael Nielsen在线深度学习书籍
  - http://neuralnetworksanddeeplearning.com/
- 科研博客
  - 跟李沐学AI： https://space.bilibili.com/1567748478
  - Andrej Karpathy: http://karpathy.github.io/
  - Christopher Olah: http://colah.github.io/

THANK YOU