

计算机应用编程

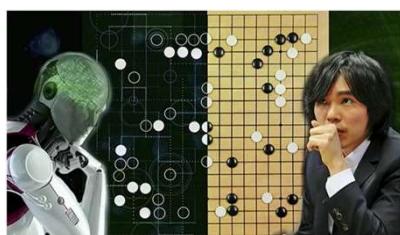
数字识别神经网络

周一 13:30-17:20@3-235

2018.7.9

实验意义

- 各种实际应用
 - 各种火的一塌糊涂的应用
 - 图像识别
 - 自动驾驶
 - 语音识别
 - 自然语言处理
- 机器学习人才的市场需求旺盛





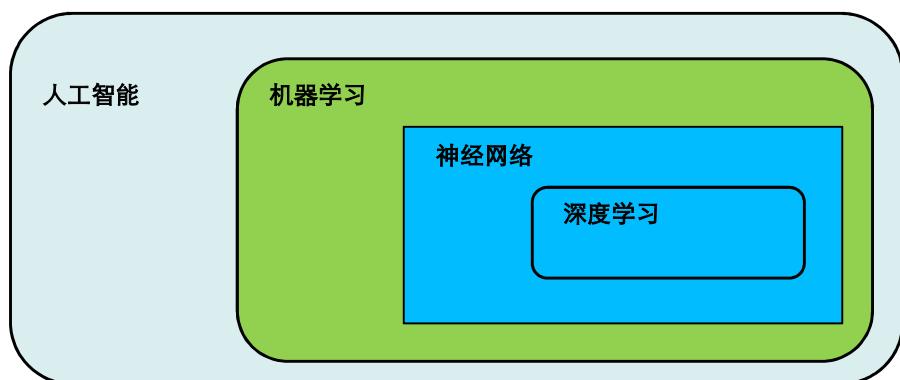
实验任务

- 目标
 - 实现一个多层神经网络分类器
 - 根据给定的手写数字数据集进行训练并优化
- 编程技能
 - python语言编程
 - 神经网络原理
 - Cos激活函数和交叉熵代价函数
 - 微积分应用
 - 机器学习核心思想



神经网络是个什么东西？

- 神经网络是机器学习的一个分支，全名应该叫人工神经网络，与之相对应的是生物神经网络（**Biological Neural Networks, BNN**）



What's involved in Intelligence?

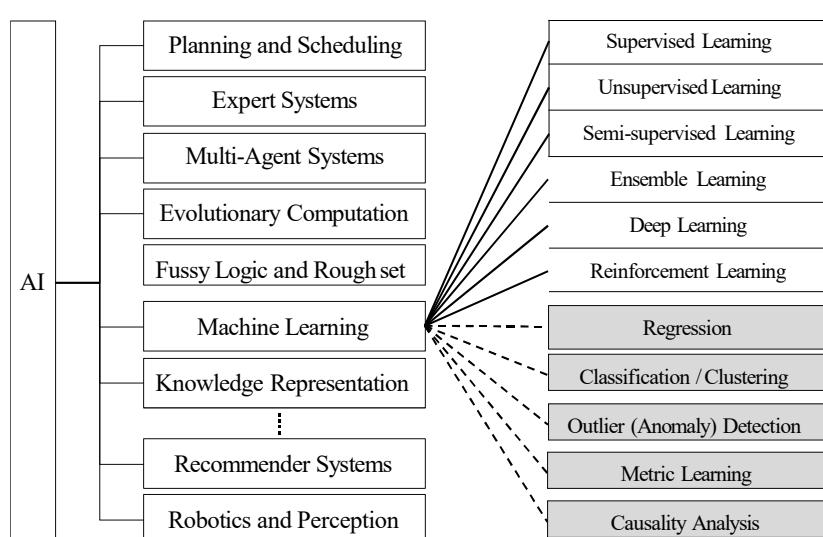
- Ability to interact with the real world
 - to perceive, understand, and act
 - e.g., speech recognition and understanding and synthesis
 - e.g., image understanding
 - e.g., ability to take actions, have an effect
- Reasoning and Planning
 - modeling the external world, given input
 - solving new problems, planning, and making decisions
 - ability to deal with unexpected problems, uncertainties
- Learning and Adaptation
 - we are continuously learning and adapting
 - our internal models are always being “updated”
 - e.g., a baby learning to categorize and recognize animals

弱人工智能

强人工智能

计算机应用编程实验2018

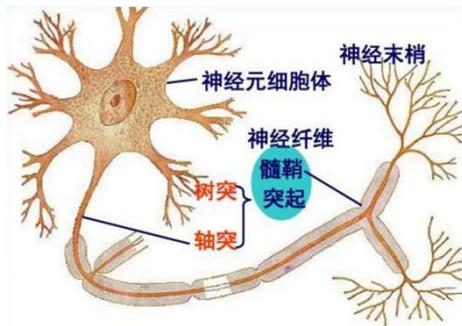
人工智能的子领域



计算机应用编程实验2018

神经网络的仿生学基础

- 神经元细胞体：通过化学反应，为神经活动提供能量，并大量制造用于传递信息的化学物质。
- 树突：呈树枝状，接受其他神经元的信息并传至细胞体
- 轴突：把冲动由细胞体传至远处，传给另一个神经元的树突或肌肉与腺体



计算机应用编程实验2018

背景知识

计算机应用编程实验2018

链式法则

- 情况1:

- 复合函数 $y = f(u)$, $u = \varphi(x)$, 求导法则:

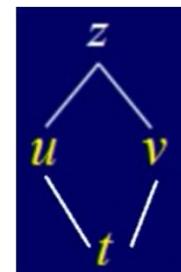
$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

- 情况2:

- $u = \varphi(t)$, $v = \psi(t)$ $z = f(u, v)$ 在点 (u, v) 处可导,

处可以表示为: $z = f(\varphi(t), \psi(t))$

$$\frac{dz}{dt} = \frac{\partial z}{\partial u} \cdot \frac{du}{dt} + \frac{\partial z}{\partial v} \cdot \frac{dv}{dt}$$



计算机应用编程实验2018

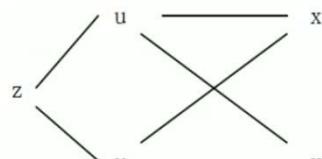
链式法则

- 情况3:

- 如果函数 $u = u(x, y)$, $v = v(x, y)$, 在点 (x, y) 处都具有对 x 及对 y 的偏导数, 函数 $z = f(x, y)$ 在对应点 (u, v) 处具有连续偏导数, 则复合函数 $z = f[u(x, y), v(x, y)]$ 在点 (x, y) 处存在两个偏导数, 且具有下列公式:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u} \frac{\partial u}{\partial x} + \frac{\partial z}{\partial v} \frac{\partial v}{\partial x}$$

$$\frac{\partial z}{\partial y} = \frac{\partial z}{\partial u} \frac{\partial u}{\partial y} + \frac{\partial z}{\partial v} \frac{\partial v}{\partial y}$$

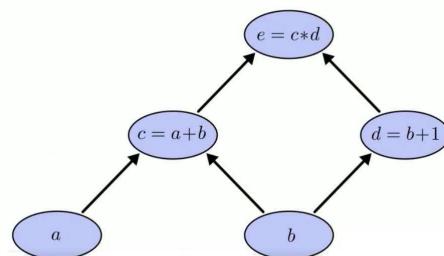


- 一般地, 无论符合函数的复合关系如何, 因变量有几条路径, 就有几项相加。一条路径中有几个环节, 这一项就有几个偏导数相乘。

计算机应用编程实验2018

计算图 (Computational Graph)

- 计算图是一种可以描述函数功能的语言，是一种对神经网络等机器学习模型进行形式化表示的强大工具。
- 计算图是一种有向图，它描述了由一系列的计算步骤构成的某些模型，例如，一个FNN（前馈神经网络）。
- 计算图中的节点（vertex）代表一个计算，图中的有向边（edge）指定每个节点所依赖的输入（可以是常数、向量、张量等）。



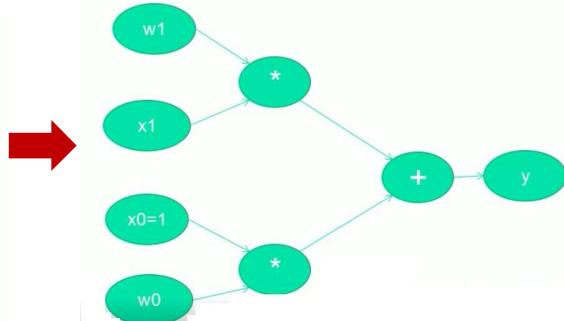
计算机应用编程实验2018

计算图 ≠ 神经网络

输入



输出



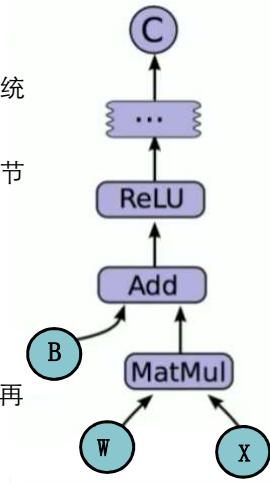
神经网络

计算图

计算机应用编程实验2018

Tensorflow中的计算图

- 计算图是Tensorflow中最核心的概念
 - Tensorflow是一个通过计算图的形式来表述计算的编程系统，计算图也叫数据流图。
 - Tensorflow中的每一个计算都是计算图上的一个节点，而节点之间的边描述了计算之间的依赖关系
- 图的构成要素：节点，边
- 图中的边
 - 正常边：tensor可以流动数据的边
 - 特殊边：节点之间的依赖关系，如可以让起始节点执行完再执行目标节点，比如起到“限制内存峰值”的作用



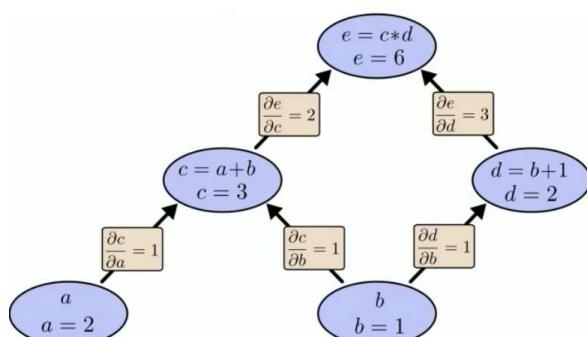
计算机应用编程实验2018

链式法则应用于计算图

- 计算图中如何求取导数？

$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$

$$\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$



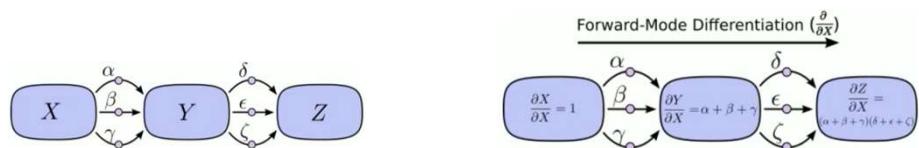
- 不相邻的两个节点之间应该如何求其偏导数呢？

计算机应用编程实验2018

前向微分 VS 反向微分

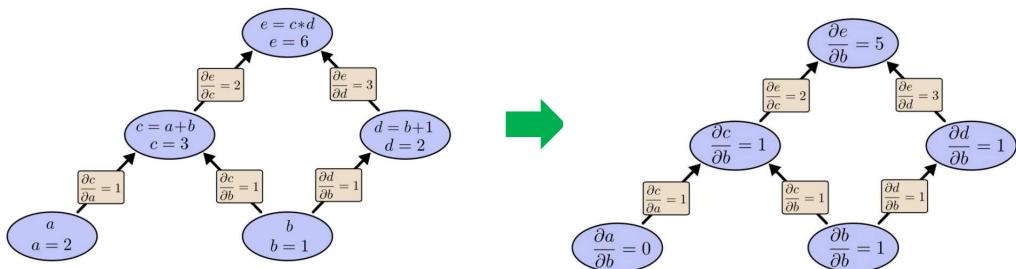
• 前向微分

- 前向微分从图的输入开始，一步一步到达终点。在每个节点处，对输入的路径进行求和。每个这样的路径都表示输入影响该节点的一个部分。通过将这些影响加起来，我们就得到了输入影响该节点的全部，也就是关于输入的导数。



计算机应用编程实验2018

前向微分举例

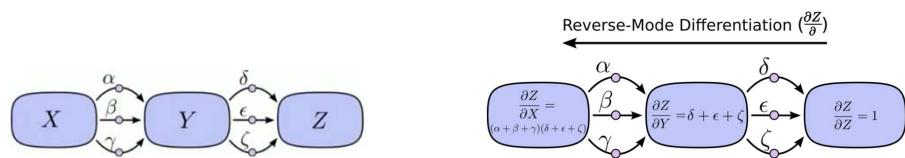


计算机应用编程实验2018

前向微分 VS 反向微分

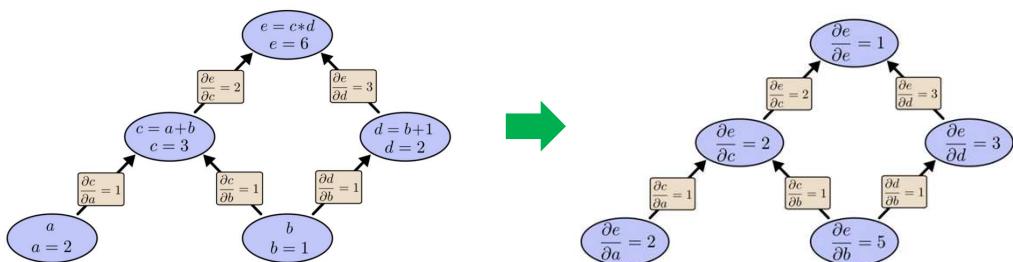
• 反向微分

- 反向微分是从图的输出开始，反向一步一步抵达最开始输入处。在每个节点处，会合了所有源于该节点的路径。
- 前向微分 跟踪了输入如何改变每个节点的情况。反向微分则跟踪了每个节点如何影响输出的情况。也就是说，前向微分应用操作 d/dX 到每个节点，而反向微分应用操作 dZ/d 到每个节点。



计算机应用编程实验2018

反向微分举例



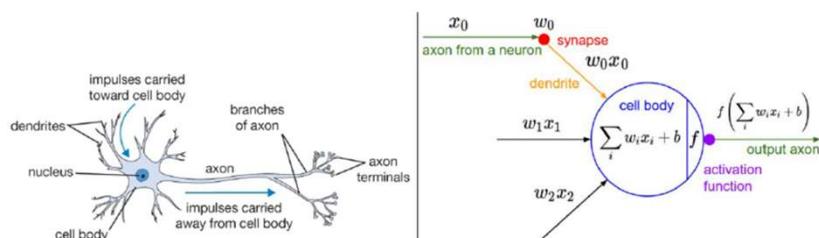
计算机应用编程实验2018

神经网络基础概念

计算机应用编程实验2018

神经元

- 神经网络(neural networks)是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应。



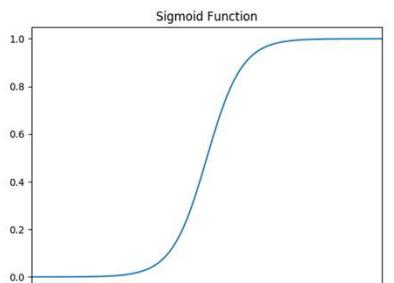
- 一个以 x_1, x_2, x_3 , 以及bias 1为输入值的神经元, 其对应的输出是:

$$h_{W,b}(x) = f(W^T x) = f\left(\sum_{i=1}^3 W_i x_i + b\right)$$

- 激活函数: f
 - 模拟神经元对电信号进行反应的函数

计算机应用编程实验2018

Sigmoid激活函数



Sigmoid函数

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

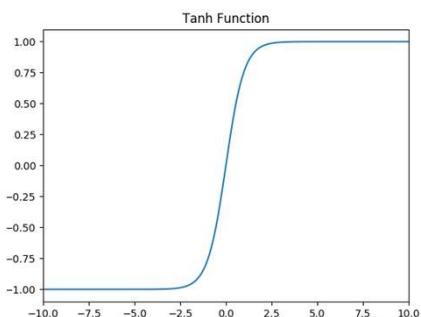
$$f'(z) = f(z)(1 - f(z))$$

- 缺点

- 激活函数计算量大，反向传播求误差梯度时，求导涉及除法反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练
- 反向传播算法中，要对激活函数求导，sigmoid 的导数表达式为：

计算机应用编程实验2018

Tanh激活函数（双曲正切）



Tanh函数

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

$$f'(z) = 1 - (f(z))^2$$

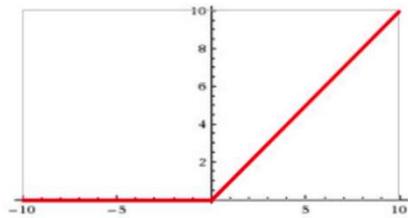
- 与Sigmoid函数比较类似
- 与 sigmoid 的区别是，tanh 是 0 均值的，因此实际应用中 tanh 会比 sigmoid 更好

计算机应用编程实验2018

ReLU激活函数（修正线性函数）

- 函数定义

$$\phi(x) = \max(0, x)$$



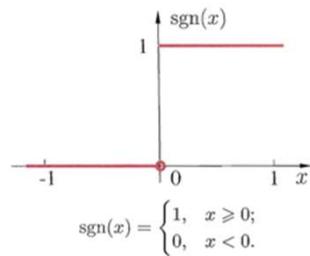
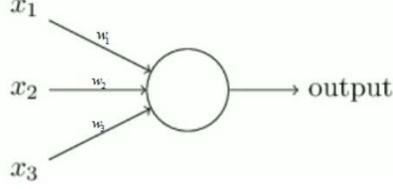
- 特性

- 优点: Krizhevsky et al. 发现使用 ReLU 得到的 SGD 的收敛速度会比 sigmoid/tanh 快很多
- 缺点: 训练的时候很”脆弱”，很容易就”die”
 - 例如，一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了，那么这个神经元的梯度就永远都会是 0。

计算机应用编程实验2018

感知器

- 在上个世纪50-70年代很流行，也成功解决了很多问题
- **输入权值**: 一个感知器可以接收多个输入，每个输入上有一个权值，此外还有一个偏置项
- **激活函数**: 采用**阶跃函数**来作为激活函数



阶跃函数

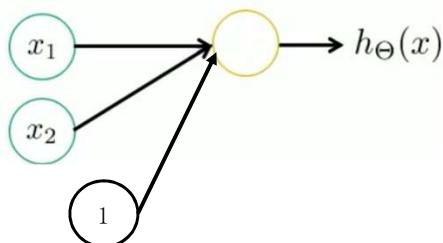
计算机应用编程实验2018

感知器的简单实例：AND运算

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ AND } x_2$$

X1	X2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$\begin{cases} b < 0 \\ w_2 + b < 0 \\ w_1 + b < 0 \\ w_1 + w_2 + b \geq 0 \end{cases}$$

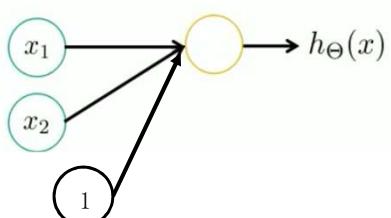
计算机应用编程实验2018

感知器无法解决异或运算

$$x_1, x_2 \in \{0, 1\}$$

$$Y=X1 \text{ NOR } X2$$

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	0

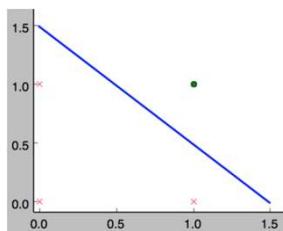


$$\begin{cases} b < 0 \\ w_2 + b \geq 0 \\ w_1 + b \geq 0 \\ w_1 + w_2 + b < 0 \end{cases} \Rightarrow \begin{cases} w_1 + w_2 + 2b \geq 0 \\ w_1 + w_2 + 2b < 0 \end{cases} \text{矛盾}$$

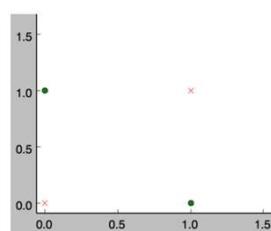
计算机应用编程实验2018

为什么感知器无法解决异或问题？

- 线性可分
 - AND操作



- 线性不可分
 - NOR操作



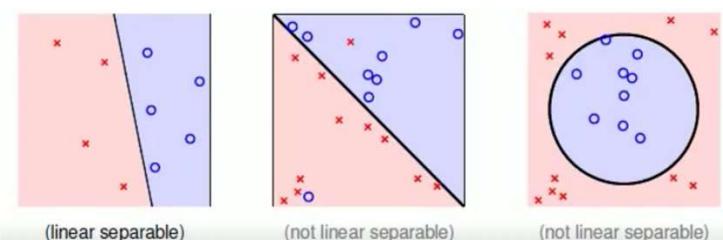
- 既然单层的感知器无法解决异或问题，那么多层次感知器能否解决呢？

$$XOR(g1, g2) = OR(AND(NOT(g1), g2), AND(g1, NOT(g2)))$$

计算机应用编程实验2018

如何训练单层感知器？

- 基本前提：训练数据必须线性可分



- 感知器的符号表示：

$$f(x; w, b) = \text{sgn}(wx + b)$$

- 两种手段：

- 梯度下降：定义目标函数 loss function
- 参数迭代更新

计算机应用编程实验2018

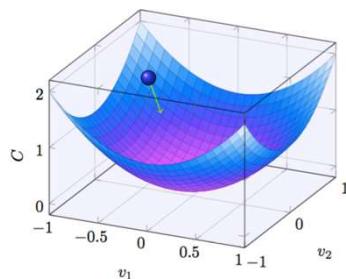
梯度下降求解

- 思想

- 每次选择 $\Delta v = -\eta \nabla C$
- 以保证 $\Delta C \leq 0$

- 迭代求解

- 迭代更新 $v \rightarrow v' = v - \eta \nabla C$
- 不停的进行迭代更新，那么C最终会降低到我们想要的全局最小值。



计算机应用编程实验2018

单层感知器参数更新

- 假定挑选样本 (x_1, x_2, \dots, x_m) 作为输入对神经元进行刺激，令

$$\begin{aligned} w &= (w_1, w_2, \dots, w_m, b) \\ x &= (x_1, x_2, \dots, x_m, 1) \end{aligned}$$

$$\text{则 } y = \text{sgn}(v) = \begin{cases} +1 & v \geq 0 \\ 0 & v < 0 \end{cases}, v = w \cdot x$$

- 输出值y与真实值(标注值)y*之间可能存在的情况：

- $y=y^*$: 感知器权重不需要进行调整
- $y=1, y^*=0$: 说明 $w \cdot x$ 偏大，应该通过减小w的值，使得其小于0;
- $y=0, y^*=1$: 说明 $w \cdot x$ 偏小，应该通过增大w的值，使得其大于0;

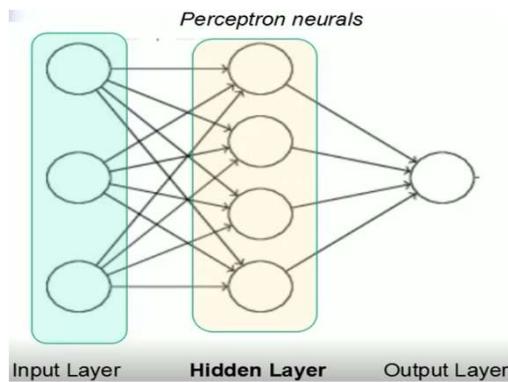
- 怎么调整w的值呢？

$$w := w + \text{learning_rate} \times (y^* - y) \times x$$

计算机应用编程实验2018

前馈神经网络

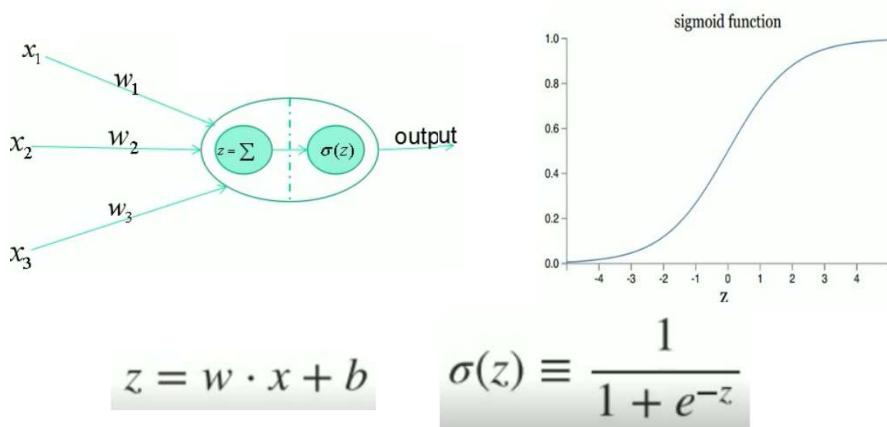
- 前馈神经网络 (feedforward neural network)，简称前馈网络，是人工神经网络的一种。在此种神经网络中，各神经元从输入层开始，接收前一级输入，并输出到下一级，直至输出层。整个网络中无反馈，可用一个有向无环图表示。
- 前馈神经网络采用一种单向多层结构。其中每一层包含若干个神经元，同一层的神经元之间没有互相连接，层间信息的传送只沿一个方向进行。其中第二层称为输入层。最后一层为输出层。中间为隐含层，简称隐层。隐层可以是一层。也可以是多层。



计算机应用编程实验2018

对感知器的改造 (一)

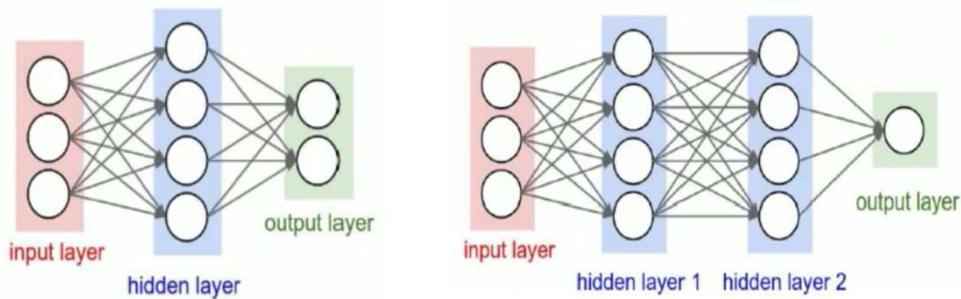
- 激活函数：
 - 阶跃函数被改造为其他类型的激活函数，例如：sigmoid函数、Relu函数、Tanh函数等等....



计算机应用编程实验2018

对感知器的改造 (二)

- 增加层数：
 - 在输入层及输出层之间可以添加1层或多次隐含层
 - 隐含层大于等于5层的就是深度神经网络



计算机应用编程实验2018

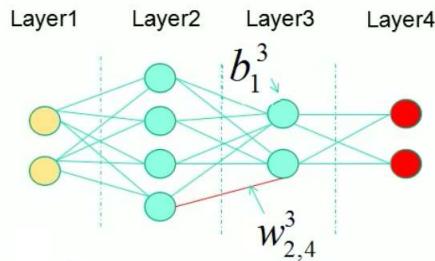
激活函数的性质

- 激活函数通常拥有如下性质：
 - 非线性：**当激活函数是线性的时候，一个两层的神经网络就可以逼近基本上所有的函数了。但是，如果激活函数是恒等激活函数的时候，就不满足这个性质了，而且如果MLP使用的是恒等激活函数，那么其实整个网络跟单层神经网络是等价的。
 - 可微性：**当优化方法是基于梯度的时候，这个性质是必须的。
 - 单调性：**当激活函数是单调的时候，单层网络能够保证是凸函数。
 - 输出值的范围：**当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况下，一般需要更小的learning rate。

计算机应用编程实验2018

前馈神经网络的符号定义

1、层 (layer)



2、权重 weight: w_{jk}^l

- 从第 $l-1$ 层的第 k 个神经元链接向第 l 层的第 j 个神经元的边的权重值

3、偏置: b_j^l

- 第 l 层的第 j 个神经元的偏置

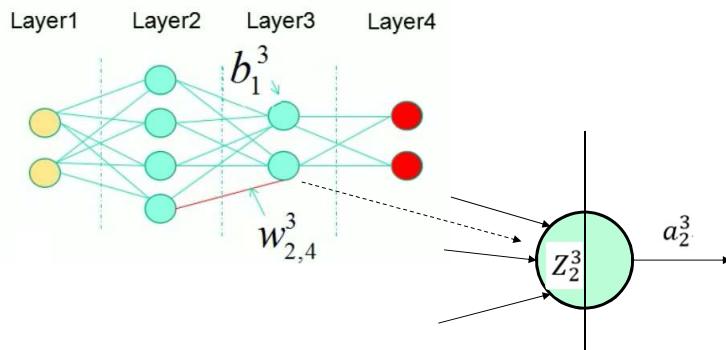
计算机应用编程实验2018

前馈神经网络的符号定义

4、神经元输入 z_j^l

- 第 l 层第 j 个神经元的输入值

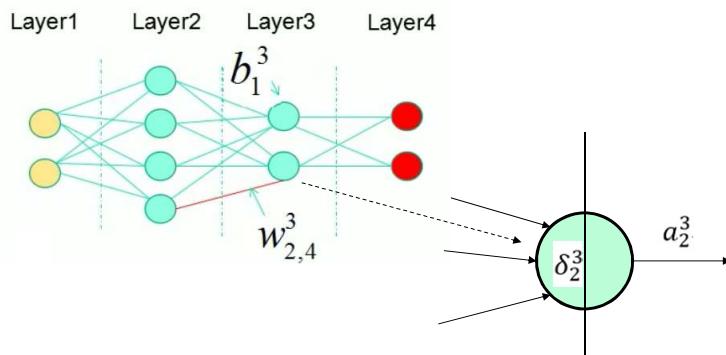
5、神经元输出值 (激活值) : $a_j^l = \sigma(z_j^l)$



计算机应用编程实验2018

前馈神经网络的符号定义

6、第 l 层第 j 个神经元的误差： $\delta_j^l = \frac{\partial C}{\partial z_j^l}$



计算机应用编程实验2018

前馈神经网络梯度下降训练法

- 损失函数与梯度下降
 - 单样本损失函数： $\frac{\partial C}{\partial y_i} = y_i - \hat{y}_i$
 - 全样本损失函数 累积BP算法

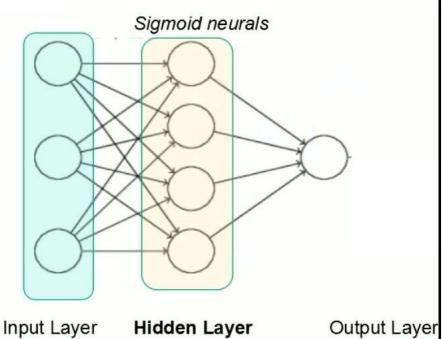
$$C_{sum} = \frac{1}{N} \sum_{n=1}^N C^{(n)} \quad \rightarrow \quad \frac{\delta C_{sum}}{\delta w} = \frac{1}{N} \sum_1^N \frac{\delta C^{(n)}}{\delta w}$$

- 基于损失函数，使用梯度下降方法：

$$w_k' = w_k + \Delta w_k = w_k - \eta \frac{\partial C}{\partial w_k} \quad b_k' = b_k - \eta \frac{\partial C}{\partial b_k}$$

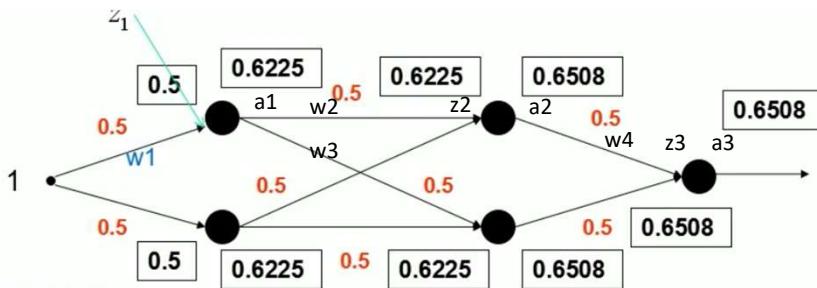
- 问题：如何计算

$$\frac{\delta c}{\delta w_k} \quad \frac{\delta c}{\delta b_k}$$



计算机应用编程实验2018

一个简单的BP网络实例



- 说明

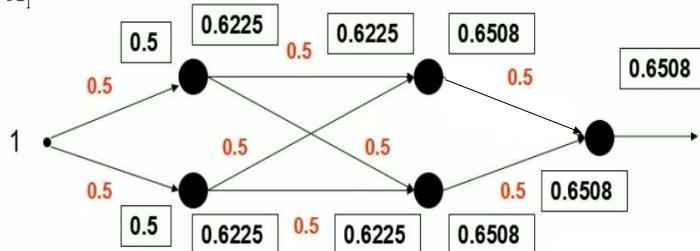
- 训练目标：通过不断训练，使得最终满足输入为1时，输出也为1
- 为了简化训练过程，做了如下简化：
 - 上下两条通路对称设置
 - 中间层的激活函数采用sigmoid
 - 输出层只有一个节点，且不设置激活函数
 - 去掉偏置
 - 损失函数：MSE(Mean-square Error 均方误差)

计算机应用编程实验2018

第一轮求梯度

$$\frac{\partial C}{\partial z_2} = \frac{\partial C}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} = (-0.3492) \times 0.5 \times 0.6508 \times (1 - 0.6508) = -0.0397$$

$$\frac{\partial C}{\partial z_1} = 2 \times \frac{\partial C}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} = 2 \times (-0.0397) \times 0.5 \times 0.6225 \times (1 - 0.6225) = 2 \times (-0.0093)$$



$$\frac{\delta c}{\delta y} = 0.6508 - 1 = -0.3492$$

$$\frac{\delta c}{\delta z_3} = \frac{\delta c}{\delta y} \frac{\delta y}{\delta z_3} = -0.3492$$

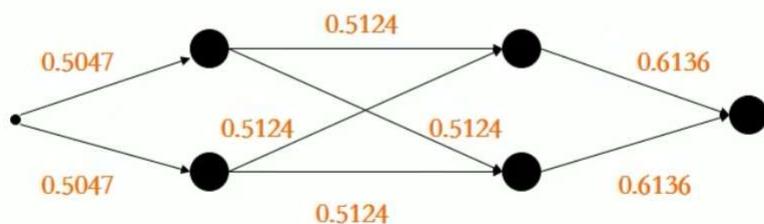
计算机应用编程实验2018

第一轮更新权重

$$0.5 - (0.5)(-0.0397)(0.6225)$$

$$0.5 - (0.5) * 2 * (-0.0093)(1)$$

$$0.5 - (0.5)(-0.3492)(0.6508)$$



$$w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

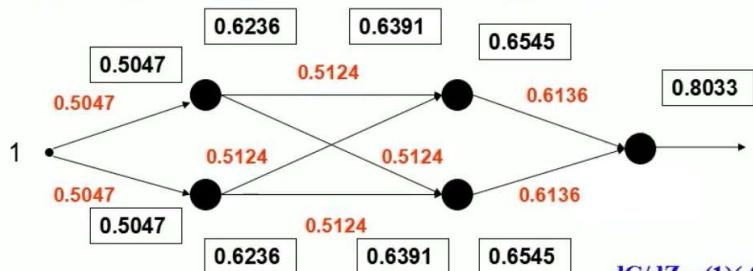
$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_k}$$

计算机应用编程实验2018

第二轮求梯度

$$\frac{dC}{dZ_1} = (0.6236)(1 - 0.6236)(0.5124)(-0.0273)(2) = -0.0066$$

$$\frac{dC}{dZ_2} = (0.6545)(1 - 0.6545)(-0.1967)(0.6136) = -0.0273$$

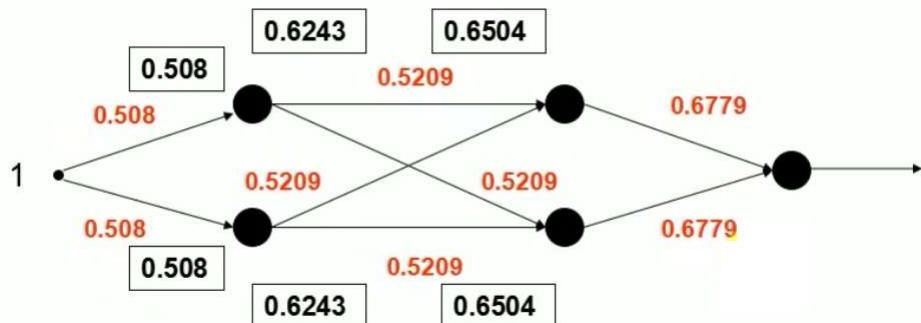


$$\frac{dC}{dZ_3} = (1)(-0.1967) = -0.1967$$

$$\frac{dC}{dy} = 0.8033 - 1 = -0.1967$$

计算机应用编程实验2018

第二轮更新后的输出



计算机应用编程实验2018

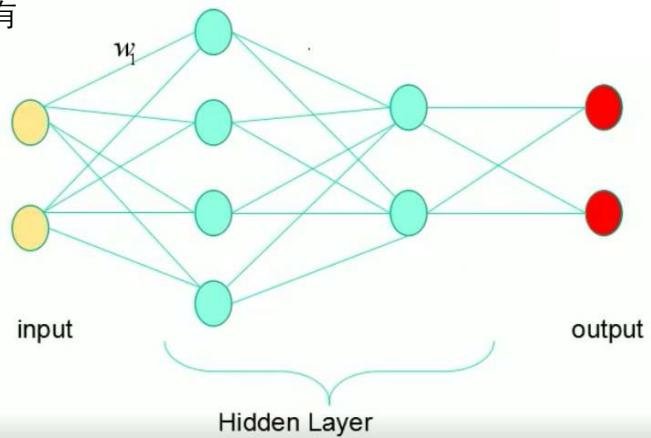
迭代过程输出

	Weights			Output	Expected	Error
	w1	w2	w3			
Initial conditions	0.5	0.5	0.5	0.6508	1	0.3492
Pass 1 Update	0.5047	0.5124	0.6136	0.8033	1	0.1967
Pass 2 Update	0.508	0.5209	0.6779	0.8909	1	0.1091

计算机应用编程实验2018

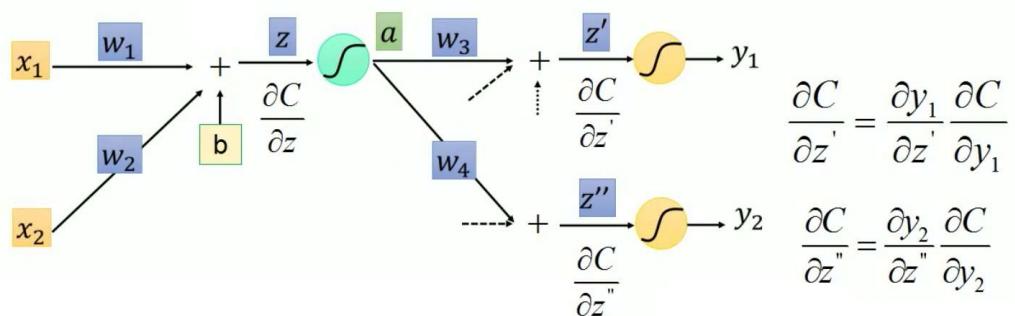
BP (Back Propagation) 算法原理

- 两个过程
 - 信号正向传播
 - 计算所有每个神经元的输入、输出，最终输出及损失函数
 - 误差反向传播
 - 计算损失函数对所有 z 及权重的偏导数



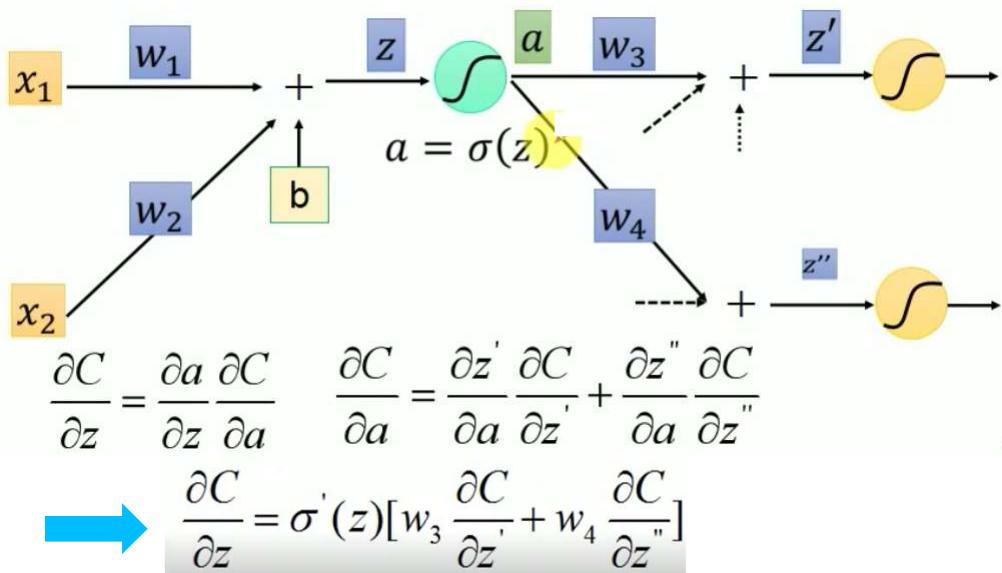
计算机应用编程实验2018

输出层的反向传播



计算机应用编程实验2018

隐含层的反向传播



计算机应用编程实验2018

BP算法的公式化表示

$$\text{BP1} \quad \delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \longrightarrow \delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\text{BP2} \quad \delta_j^l = \sum_k w_{kj}^{l+1} \delta_j^{l+1} \sigma'(z_j^l) \longrightarrow \delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\text{BP3} \quad \frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\text{BP4} \quad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

计算机应用编程实验2018

证明

- BP2的证明：

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}\end{aligned}$$

- 对右式求导并带回：

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \quad \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

- 得： $\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$ 证明了其中一项

- 而BP1, BP3, BP4都是应用链式法则求导便可证明出。

计算机应用编程实验2018

Hadamard乘积

- 假设s和t是两个同样维度的向量，那么我们使用 $s \odot t$ 来表示按元素的乘积。所以， $s \odot t$ 的元素就是

$$(s \odot t)_j = s_j t_j$$

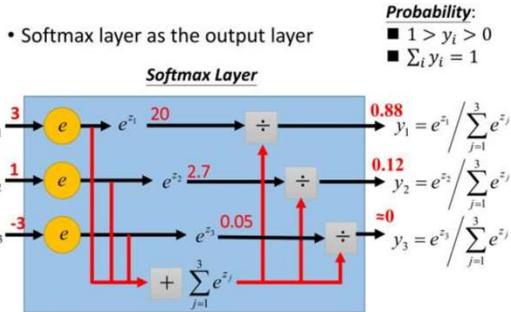
$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

计算机应用编程实验2018

softmax函数（输出层）

- 定义

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$



- 作用

- 就是如果某一个 z_j 大过其他 z , 那这个映射的分量就逼近于 1, 其他就逼近于 0
- 主要应用就是多分类, 等价于2分类的sigmoid
- 取指数: 模拟 max 的行为, 大的更大, 可导的函数。

计算机应用编程实验2018

交叉熵代价函数

- Kullback - Leibler divergence (K-L 散度)

- 设 $p(x)$ 和 $q(x)$ 是 X 取值的两个概率分布, 则 p 对 q 的相对熵为:

$$\begin{aligned} D_{KL}(p||q) &= \sum_{i=1}^n p(x_i) \log \frac{p(x_i)}{q(x_i)} \\ &= \sum_{i=1}^n p(x_i) \log p(x_i) - \sum_{i=1}^n p(x_i) \log q(x_i) \end{aligned}$$

- 基于KL散度的代价函数定义

$$H(\rho||\hat{\rho}) = - \sum_{j=1}^m [\rho_j \log(\hat{\rho}_j) + (1 - \rho_j) \log(1 - \hat{\rho}_j)]$$

- 神经网络交叉熵代价函数

$$C = - \frac{1}{n} \sum_x y \ln a + (1 - y) \ln(1 - a)$$

计算机应用编程实验2018

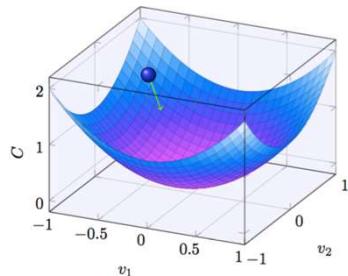
梯度下降求解

- 思想

- 每次选择 $\Delta v = -\eta \nabla C$
- 以保证 $\Delta C \leq 0$

- 迭代求解

- 迭代更新 $v \rightarrow v' = v - \eta \nabla C$
- 不停的进行迭代更新，那么C最终会降低到我们想要的全局最小值。



计算机应用编程实验2018

回到神经网络

- 求解过程

- 在神经网络中我们通过
- $$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$
- $$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

- 来对参数进行更新，以便达到想要的最小的代价函数C，C是每个样本代价 C_x 的平均值。

$$C = \frac{1}{n} \sum_x C_x \quad C_x \equiv \frac{\|y(x)-a\|^2}{2}$$

- 计算梯度 $\nabla C = \frac{1}{n} \sum_x \nabla C_x$
- 需要计算每个样本x的梯度值导致计算代价很高，使得学习过程相当的缓慢

计算机应用编程实验2018

随机梯度下降SGD

- 思想

- 全民普选→民意调查
- 不是每次计算所有样本，而是随机选取少量的m个训练数据，并对数据进行标号， $X_1, X_2, \dots, X_{m\text{-batch}}$ ，使得：

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^m \nabla C_{X_j}$$

- 可以通过计算随机选取的mini-batch的梯度来估计整体的梯度。
 - 应用到神经网络中，以epoch为单位迭代训练
- $$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$
- $$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$
- 每个epoch迭代随机选取1个mini-batch训练，直到用完了所有的训练数据

计算机应用编程实验2018

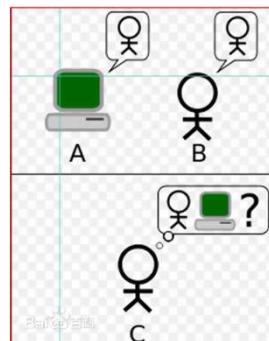


Turing Test (图灵测试)

The **Turing Test** was devised by Alan Turing as a method of determining if a machine exhibits human intelligence.



图灵测试用来判断机器是否具备了人的智能。

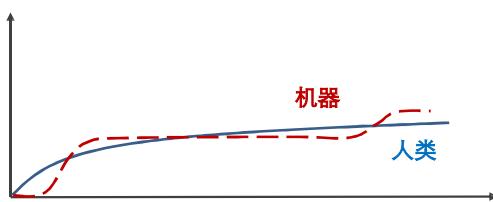


计算机应用编程实验2018

Singularity (奇点) ?

The **Singularity**, or more specifically, the technological Singularity, is the point in time at which computers exceed humans in intelligence, launching a new era of innovation.

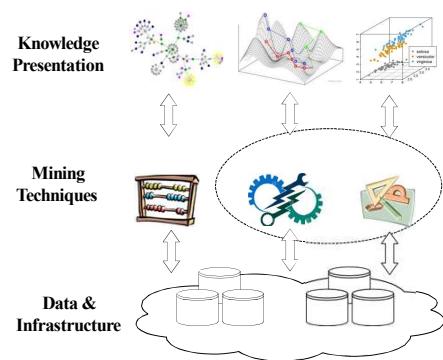
奇点：机器智能超越人类智能的那个时间点



计算机应用编程实验2018

数据挖掘 - 从数据中挖掘出知识和智能

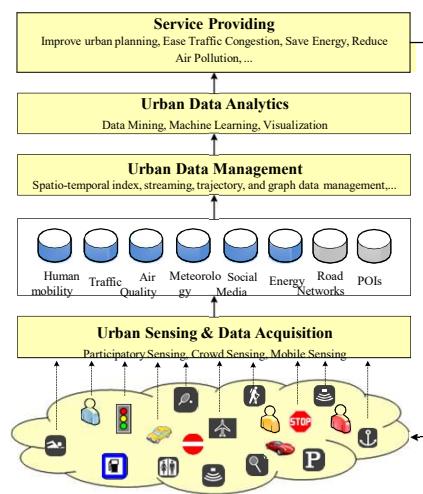
- Machine Learning Approaches
 - More accurate: minimize errors or maximum probabilities
 - Linear regression
 - Probabilistic methods: Graphical models
- Database Approaches
 - More efficient
 - Frequent Pattern Mining, clustering....
- Machine learning + Databases (data management) + Visualization



计算机应用编程实验2018

大数据- Big Data

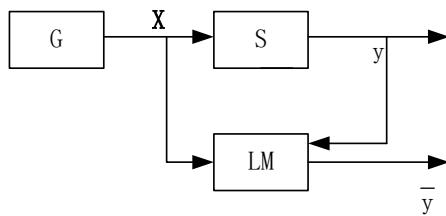
- 以数据为中心的思维方式
- 以数据解决行业问题的途径 (Data Science + Domain Knowledge)
- 大数据是一种从数据采集、管理、分析、可视化并提供服务的端到端的过程和能力
- Volume, Velocity, Variety, Value



计算机应用编程实验2018

机器学习的基本问题

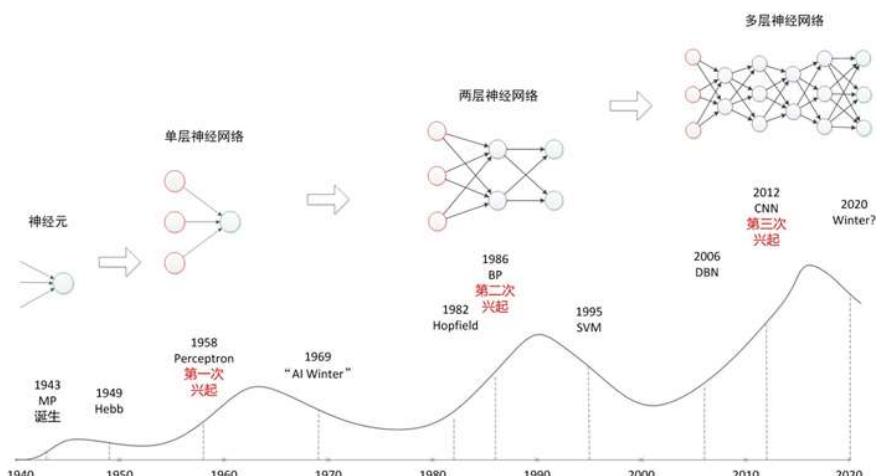
• 机器学习问题的表示



- 产生器(G)，产生随机向量 x 属于 R^n ，它们是从固定但未知的概率分布函数 $F(x)$ 中独立抽取的。
- 训练器(S)，对每个输入向量 x 返回一个输出值 y ，产生输出的根据是同样固定但未知的条件分布函数 $F(y|x)$ 。
- 学习机器(LM)，它能够实现一定的函数集 $f(x, a)$ ， a 属于 A ，其中 A 是参数集合

计算机应用编程实验2018

从层次增加看深度学习发展



计算机应用编程实验2018

浅层机器学习模型（第二次浪潮）

- 1-2层隐层的BP网络（20世纪80年代末期）
- SVM、Boosting等，模型的结构可视为带一层隐层节点或没有隐层节点（20世纪90年代）

Neural Network问题

- 比较容易过拟合，参数比较难tune，而且需要不少技巧
- 训练速度比较慢，
- 有限样本和计算单元情况下对复杂函数的表示能力有限
- 梯度越来越稀疏：从输出层越往输入层，误差校正信号越来越小
- 收敛到局部极小值

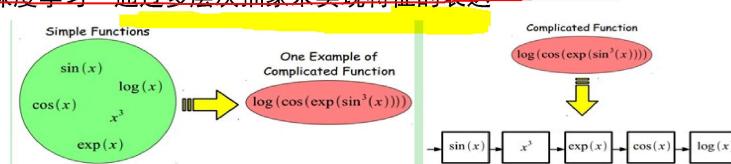
近20多年，主要SVM和boosting算法

计算机应用编程实验2018

深度机器学习模型（第三次浪潮）

Geoffrey Hinton（加拿大多伦多大学教授、机器学习领域的泰斗）及其学生2006年在科学杂志发表“Deep Learning”文章，开启了深度学习在学术界和工业界的浪潮。

- 多隐层的人工神经网络具有优异的特征学习能力
- 通过无监督学习的“逐层初始化”（layer-wise pre-training）来有效克服深度神经网络在训练上的难度
- 深度学习可通过学习一种深层非线性网络结构，实现复杂函数逼近，表征输入数据分布式表示，并展现了强大的从少数样本集中学习数据集本质特征的能力。
- 强调了模型结构的深度，通常有5层、6层，甚至10多层的隐层节点
- 深度学习就是特征学习，通过逐层特征变换，将样本在原空间的特征表示变换到一个新特征空间，从而使分类或预测更加容易
- “深度学习”通过多层次抽象来实现特征的表达

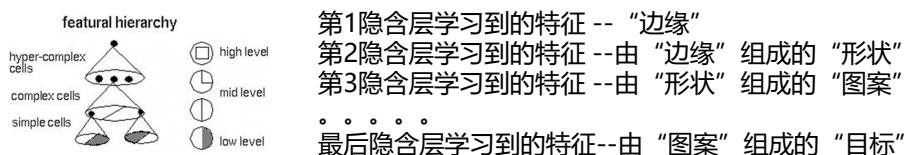


计算机应用编程实验2018

深度学习基本思想

- 神经网络本质

- 模拟特征与目标之间函数映射关系。
- 层数多，参数多，模拟映射函数更复杂、更多容量
- 层数保持不变，神经网络的参数数量增加，从而带来了更好的表示（representation）能力
- 增加更多层次，更深入的特征表示，以及更强的函数模拟能力
- 随着网络的层数增加，每一层对于前一层次的抽象表示更深入。即每一层神经元学习到的是前一层神经元值的更抽象的表示



计算机应用编程实验2018

深度网络训练方法

- 加拿大多伦多大学的Geoffery Hinton教授2006年首次提出了“深度信念网络”的概念。
- 与传统的训练方式不同，为大幅度减少了训练多层神经网络的时间，采用两个技术
 - “预训练” (pre-training)：无监督学习→参数初始值
 - 逐层贪婪训练，就是先训练网络第1个隐含层，再训练第2个…，最后将训练好的网络参数作为整个网络参数的初值（预训练，找到神经网络中一个接近最优解的权值）
 - “微调” (fine-tuning)
 - 监督学习进一步优化训练整个网络，对神经网络参数（权值）改变很小

计算机应用编程实验2018

深度学习与神经网络

- 深度学习

- ——是模拟人脑进行分析学习，称Unsupervised Feature Learning
- —源于人工神经网络，含多隐层的多层感知器就是一种深度学习结构
- 用深度网络结构来抽象和迭代组合低层特征形成更加抽象的高层表示属性类别或特征，以发现数据的分布式特征表示。
- 深度网络训练的主要思想是用非标签数据进行逐层贪婪训练和用有标记数据来进行整个网络的微调

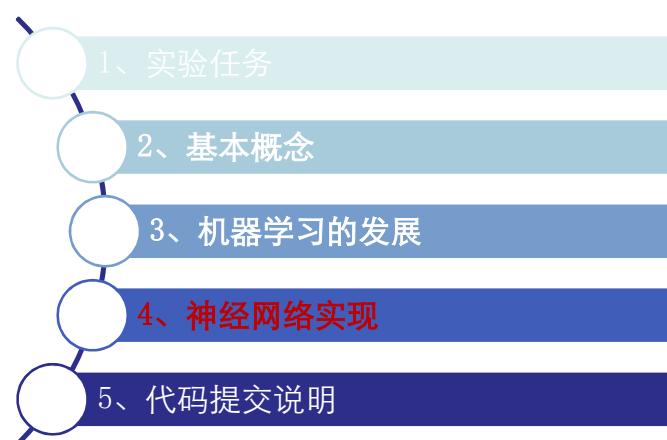
- 相同之处

- —采用分层结构：输入层、隐层（多层）、输出层
- —连接方式：相邻层节点之间有连接，同一层以及跨层节点之间相互无连接

- 不同之处

- —训练机制不同：ANN采用后向传播机制，DL layer-wise的训练机制
- —层数多，BP残差传播到最前面的层已经变得太小（梯度扩散）

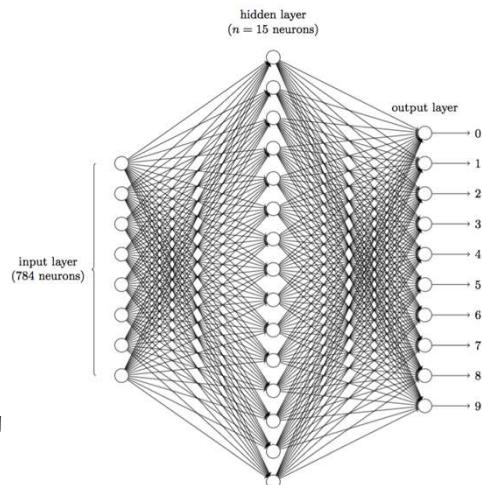
计算机应用编程实验2018



利用神经网络识别数字

• 识别模型

- 输入层是对输入像素编码的神经元。
 - 训练数据是 28×28 手写数字的位图
 - 输入层包含了 $784 = 28 \times 28$ 个神经元
 - 0代表白色，1代表黑色，中间值表示不同程度的灰度值。
- 网络的第二层是隐层
 - 本例子仅包含30个神经元。
- 网络的输出层包含10个神经元。
 - 如果第一个神经元被激活，输出约等于1，可以推断出这个数字是0。
 - 如果6号神经元有最高值，那么我们的神经网络预测输入数字是6。



问题建模

• 定义

- 用 x 去定义训练输入，每一个训练输入 x 视为一个 $28 \times 28 = 784$ 维的向量，向量中的一个元素代表图像的一个像素点的灰度值。
- 定义输出为 $y=y(x)$ ，每一个 y 是一个10维的向量。
 - 举例来说，对于一个数字的训练图像，输出是 $y(x)=(0,0,0,0,0,0,1,0,0,0)$ 。

• 训练模型

- 训练算法以找到合适的权重和偏置，从而对所有的训练输入为 x 的输出都近似于 $y(x)$ 。
- 为了衡量当前取得结果距离目标结果的好坏程度，定义一个代价函数：

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

- w 表示所有的权重的集合， b 表示所有的偏置， n 是训练数据的个数， a 代表 x 作为输入时输出层的向量，求和针对所有的训练输入 x 。
- 取值非负，越小越好

SGD训练

- 流程

- SGD方法

```
def SGD(self, training_data, epochs, mini_batch_size, eta,
       test_data=None):
    if test_data is None:
        test_data = list(training_data)
    n = len(training_data)

    for j in range(epochs):
        random.shuffle(training_data)
        mini_batches = [
            training_data[k:k+mini_batch_size]
            for k in range(0, n, mini_batch_size)]
        for mini_batch in mini_batches:
            self.update_mini_batch(mini_batch, eta)
        if test_data:
            print("Epoch {} : {} / {}".format(j, self.evaluate(test_data), n))
        else:
            print("Epoch {} complete".format(j))

def update_mini_batch(self, mini_batch, eta):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw
                   for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                  for b, nb in zip(self.biases, nabla_b)]
```

反向传播代码实现

```
def backprop(self, x, y):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # feedforward
    activation = x
    activations = [x] # list to store all the activations, layer by layer
    zs = [] # list to store all the z vectors, layer by layer
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)
    # backward pass
    delta = self.cost_derivative(activations[-1], y) *
        sigmoid_prime(zs[-1])
    nabla_b[-1] = delta
    nabla_w[-1] = np.dot(delta, activations[-2].transpose())
    for l in range(2, self.num_layers):
        z = zs[-l]
        sp = sigmoid_prime(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)
```

由四个基本等式我们可以从输出层一直反向的对每一层的权重进行更新



MNIST数据集

- 手写数字识别数据集

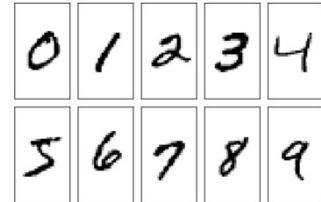
- 28 X 28 像素点灰度图像
- 数据构成
 - Training set images: train-images-idx3-ubyte.gz (47 MB, 含 60,000 个样本)
 - Training set labels: train-labels-idx1-ubyte.gz (29 KB, 含 60,000 个标签)
 - Test set images: t10k-images-idx3-ubyte.gz (7.8 MB, 含 10,000 个样本)
 - Test set labels: t10k-labels-idx1-ubyte.gz (10 KB, 含 10,000 个标签)

```

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset] [type]          [value]         [description]
0000   32 bit integer  0x00000801(2049)  magic number (MSB first)
0004   32 bit integer  60000            number of items
0008   unsigned byte   ??              label
0009   unsigned byte   ??              label
...
xxxx   unsigned byte   ??              label
The labels values are 0 to 9.

```



网络训练代码

- 流程

- 加载数据

- mnist_loader.py文件中的函数来对mnist数据集进行训练集和测试集的划分

- 网络初始化

- 调用了network类进行初始化 `net = network.Network([784, 30, 10])`

- 根据输入的参数对网络进行构建和并对 ω 和 b 进行初始化。输入[784,30,10]代表了一共三层的系统及每层的神经元个数

```
class Network(object):
    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                       for x, y in zip(sizes[:-1], sizes[1:])]
```

- 网络训练

```
net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

```
Epoch 20 : 9512 / 10000
Epoch 21 : 9518 / 10000
Epoch 22 : 9517 / 10000
Epoch 23 : 9513 / 10000
Epoch 24 : 9516 / 10000
Epoch 25 : 9498 / 10000
Epoch 26 : 9489 / 10000
Epoch 27 : 9516 / 10000
Epoch 28 : 9589 / 10000
Epoch 29 : 9492 / 10000
```

- 保存模型

```
np.save("w.npy", self.weights)
np.save("b.npy", self.biases)
```

计算机应用编程实验2018

识别手写数字

- 识别流程

- 正向传播识别

```
im=Image.open("image/0.png")
imarray = np.array(im)
w=np.load("w.npy")
b=np.load("b.npy")
imarray=imarray.reshape([784,1])
for n in range(w.size):
    imarray = sigmoid(np.dot(w[n],imarray)+b[n])
print(imarray)
```



可以看到输出结果中，下标为7的值最接近于1，由此可以分类的结果图片是7。

计算机应用编程实验2018

实验要求

- 编程语言
 - python
- 网络模型
 - 可以利用已有代码
 - 实现3层网络（增加1层隐藏层）
 - 实现cos函数作为激活函数
 - 使用交叉熵代价函数
- 实验报告
 - 包括实验目标、系统设计实现、实验结果、结论等几个部分
 - 主要逻辑
 - 算法流程图

计算机应用编程实验2018

进度要求

- 实验进度
 - W1: 理解神经网络原理，并能从数学公式中推导关键步骤
 - W2: 在python中运行给定的代码，并理解程序逻辑，增加1层隐藏层，对比识别准确率
 - W3: 修改激活函数为cos函数，推导反向传播过程，并实现程序逻辑，对比识别准确率
 - W4: 修改代价函数为交叉熵代价函数，并实现最终逻辑，对比识别准确率和训练速度

计算机应用编程实验2018

