

机器学习与数据科学

Machine Learning and Data Science

主讲: 李春光

www.pris.net.cn/teacher/lichunguang

模式识别与智能系统实验室

信息与通信工程学院 网络搜索教研中心

北京邮电大学



专题 三：线性模型的扩展

- 内容提要

- 引言

- 广义线性模型

- 核方法

- 多层感知器(MLP)

- 误差反向传播算法

函数的表达形式

- 线性函数

- 线性泛函

- Rietz表现定理: $\forall \mathbf{x} \in H, \exists \mathbf{x}_0 \in H, f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{x}_0 \rangle = \mathbf{x}_0^T \mathbf{x}$

- 非线性函数

- 如何表示？如何参数化？

- 借助再生核希尔伯特空间构造

- 借助RKHS表现定理，即 核方法

- 直接显式地给出参数化定义

- 直接定义非线性变换 Φ

- 通过交替使用简单的线性函数和非线性函数的复合

- 在多层感知器中，通过输入线性组合的非线性函数来构造非线性模型

- 在卷积神经网络中，卷积 \ 采样 \ ReLU

专题 三：线性模型的扩展

- 内容提要

- 引言
- 广义线性模型
 - 核方法
- 多层感知器(MLP)
 - 误差反向传播算法

回顾: 感知器模型

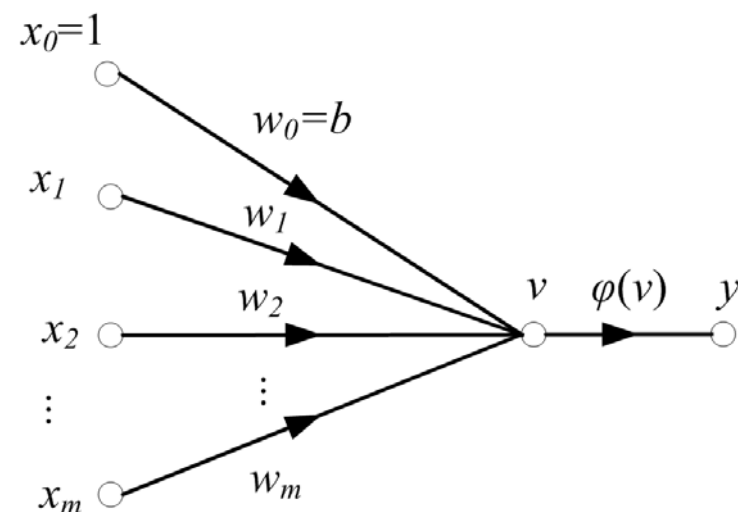
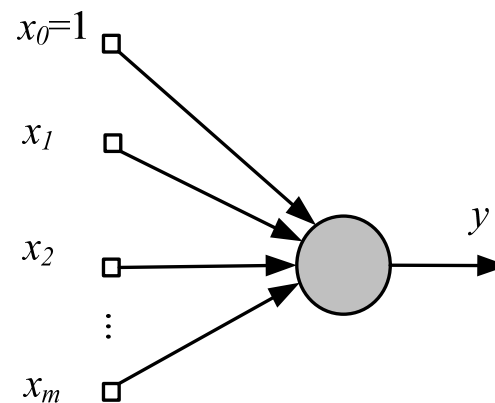
- 1个非线性神经元

- 不可微

- 数学表达

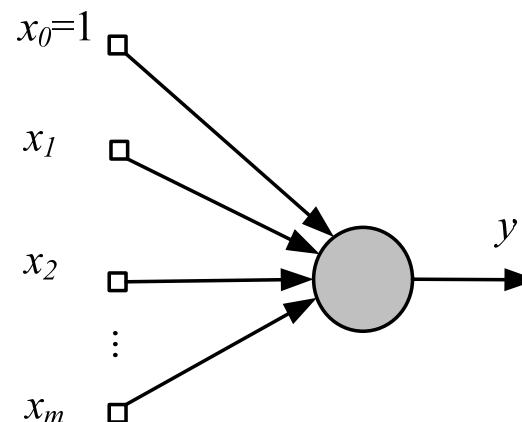
$$y = \varphi(\mathbf{w}^T \mathbf{x}) = \varphi\left(\sum_{i=0}^m w_i x_i\right)$$

- 其中 $\varphi(v) = \begin{cases} 1 & v > 0 \\ -1 & v \leq 0 \end{cases}$



回顾: 线性神经元模型

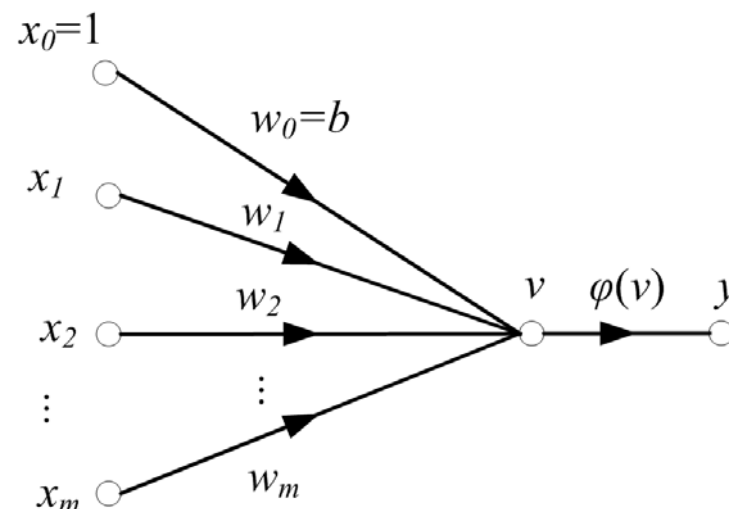
- 1个线性神经元
 - Adaline



– 数学表达

$$y = \varphi\left(\sum_{i=0}^m w_i x_i\right) = \varphi(\mathbf{w}^T \mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- 其中 $\varphi(v) = v$



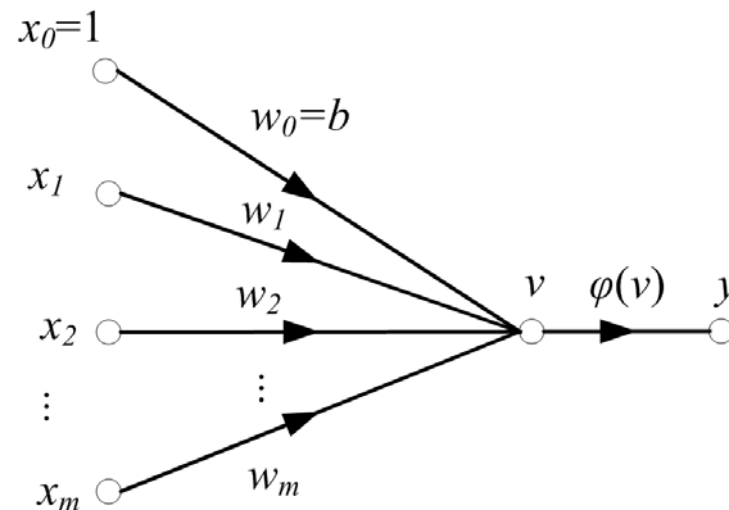
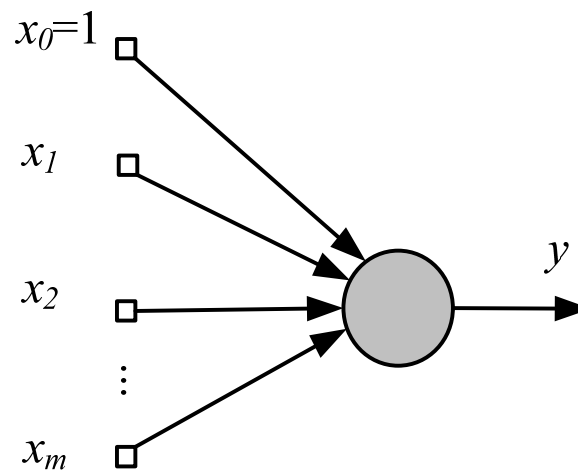
非线性神经元模型

- 1个非线性神经元

— 数学表达

$$y = \varphi\left(\sum_{i=0}^m w_i x_i\right) = \varphi(\mathbf{w}^T \mathbf{x})$$

- 其中 $\varphi(v) = \frac{1}{1 + e^{-v}}$



最小均方(LMS)算法

- 最小均方(LMS : Least Mean Square)

- 定义代价函数为均方误差 $\varepsilon(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^N e^2(n)$

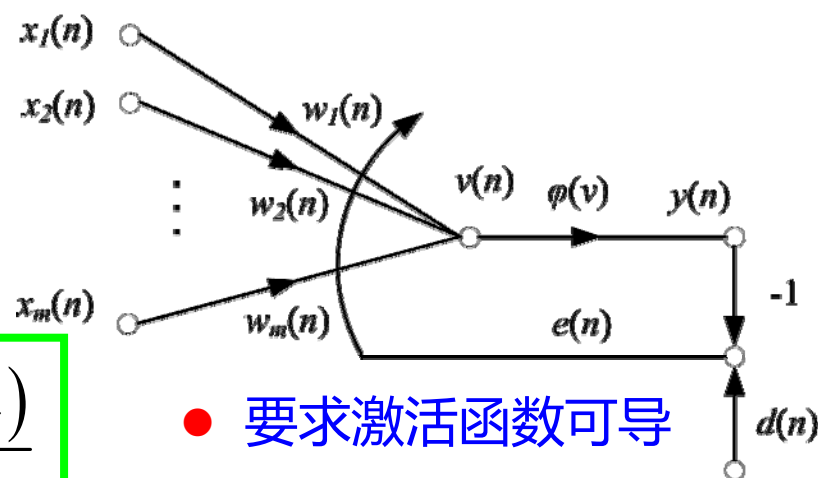
其中 $e(n)$ 为时刻 n 测得的误差

- 定义权值更新规则:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta \cdot \Delta \mathbf{w}$$

其中, 基于最速下降法, 则

$$\Delta \mathbf{w} = -\frac{\partial \varepsilon(\mathbf{w})}{\partial \mathbf{w}} = -\frac{1}{N} \sum_{n=1}^N e(n) \frac{\partial e(n)}{\partial \mathbf{w}}$$



- 权值更新特点:

- 训练集内的全部 N 个样本都呈现之后, 才进行一次权值更新
 - 称为**集中方式**、**批处理方式**
 - 训练集内的全部 N 个样本都呈现一次, 称为一个回合(epoch);

最小均方(LMS)算法

- 最小均方(LMS : Least Mean Square)

- 定义代价函数为瞬时误差的平方

$$\varepsilon(\mathbf{w}) = \frac{1}{2} e^2(n)$$

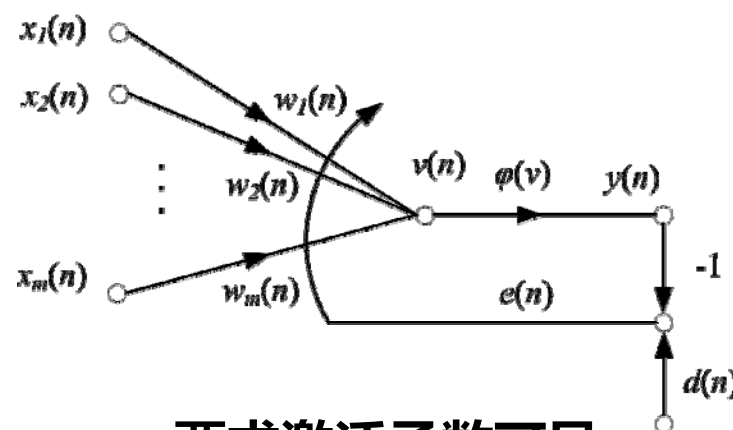
其中 $e(n)$ 为时刻 n 测得的误差

- 权值更新规则：

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta \cdot \Delta \mathbf{w}$$

其中，基于最速下降法，则

$$\Delta \mathbf{w} = -\frac{\partial \varepsilon(\mathbf{w})}{\partial \mathbf{w}} = -e(n) \frac{\partial e(n)}{\partial \mathbf{w}}$$



● 要求激活函数可导

- 权值更新特点：

- 每个训练样本呈现之后，即进行权值更新

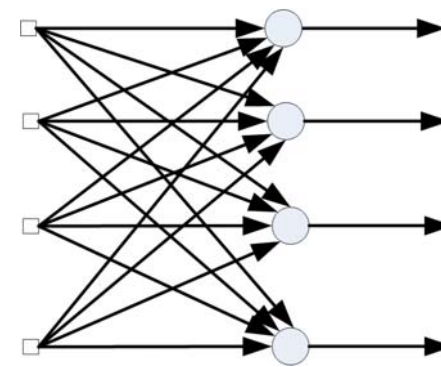
- 称为**串行方式、在线方式、随机方式**

- 训练集内的全部 N 个样本都呈现一次，称为一个回合(epoch)；



单层感知器网络

- 网络结构:
 - 基于**sigmoid**激活函数的非线性神经元构成单层感知器网络
 - 多个非线性神经元
- 训练算法
 - 最小均方算法
 - 基于LMS算法分别给出各个神经元的权值更新规则



$$w_{kj}(n+1) \leftarrow w_{kj}(n) + \Delta w_{kj}(n)$$

单层感知器网络

- 定义代价函数

- 串行方式: $\varepsilon(\mathbf{w}) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$

- **C** : 输出层的所有神经元

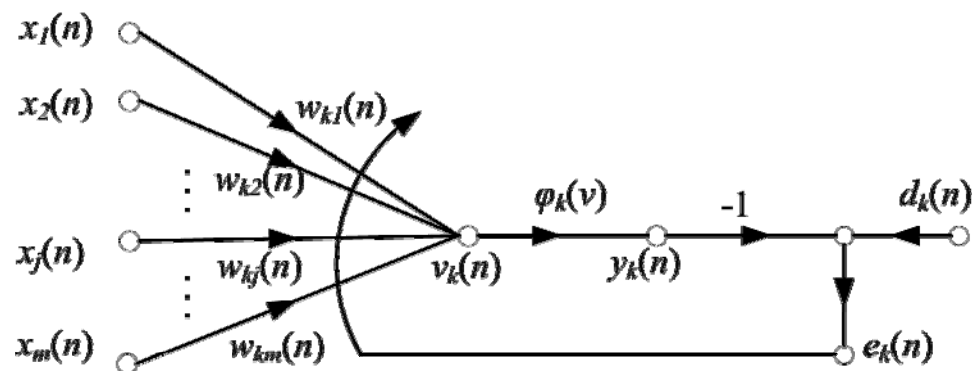
$$\Delta w_{kj}(n) = -\frac{\partial \varepsilon(\mathbf{w})}{\partial w_{kj}(n)} = -\frac{\partial \varepsilon(\mathbf{w})}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial w_{kj}(n)} = -e_k(n) \frac{\partial e_k(n)}{\partial w_{kj}(n)}$$

- 集中方式:

$$\varepsilon_{av}(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^N \sum_{k \in C} e_k^2(n)$$

- MiniBatch 方式:

$$\varepsilon_{\text{mini}}(\mathbf{w}) = \frac{1}{2L} \sum_{n=1}^L \sum_{k \in C} e_k^2(n)$$



专题三：线性模型的扩展

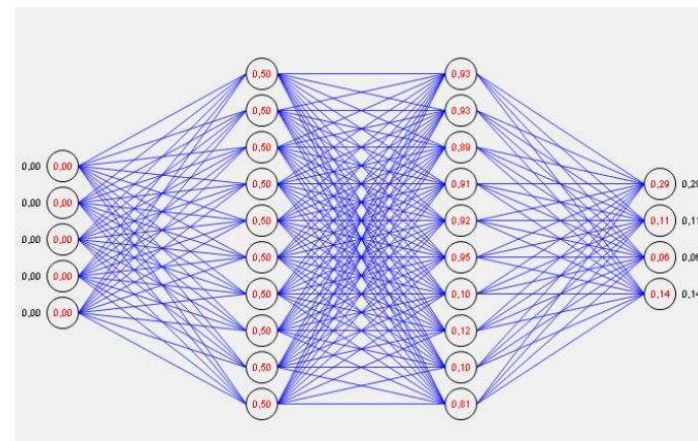
- 内容提要

- 引言
- 广义线性模型
 - 核方法
- 多层感知器(MLP)
 - 误差反向传播算法

从单层网络到多层网络

- **多层感知器**

- 如何给出权值更新规则？

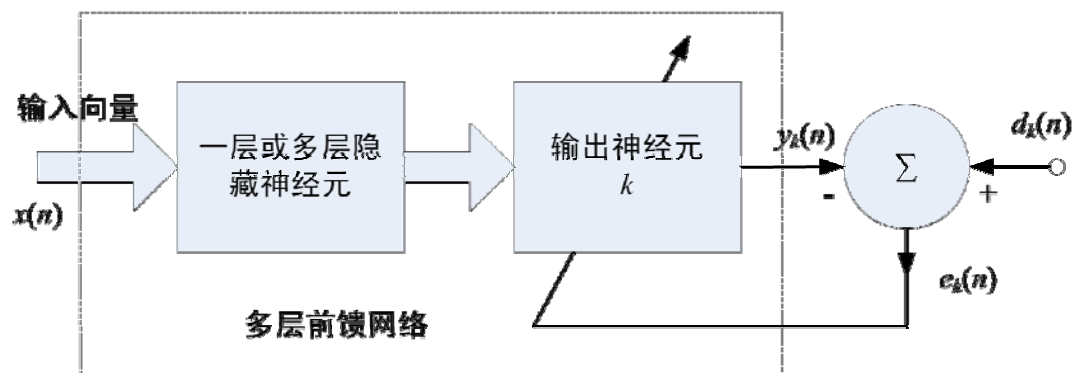


- **误差-修正学习的前提**

- 误差信号是直接可以测量的

- 在单层感知器中误差信号是直接可以测量的
 - 在多层感知器中，隐层神经元是不可见的，其误差信号无法直接测量

误差反向传播算法



反向传播算法: 基本概念

- 多层感知器(MLP: Multi-Layer Perceptron)3个特点:

- 非线性激活函数

- 非线性且处处可微

- Logistic函数

- 双曲正切

- 包括隐藏层神经元

- 高度连通性

- 网络中传递的两种信号:

- 函数信号

被当成输入及权值的函数

- 来自于输入端, 到达输出端

- 误差信号

计算过程涉及误差

- 产生于输出端, 反向传播

- 每个神经元完成两种计算:

- 计算输出函数信号

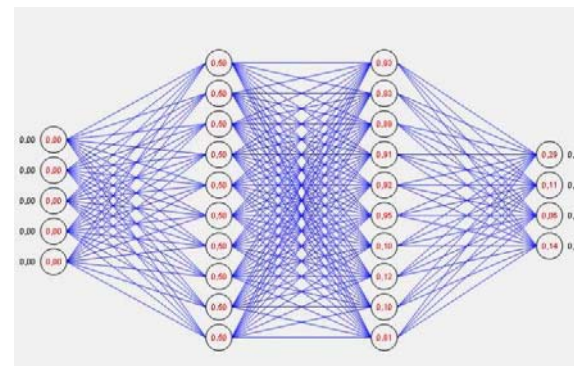
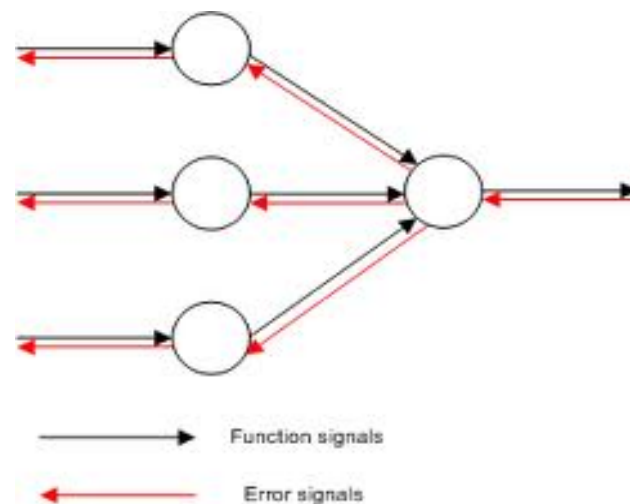
- 输入信号和权值的连续非线性函数

- 计算梯度向量

- 误差曲面对于权值的梯度的估计

$$\varphi(t) = \frac{1}{1 + \exp(-a \cdot t)}$$

$$\varphi(t) = \tanh(a \cdot t) = \frac{e^{a \cdot t} - e^{-a \cdot t}}{e^{a \cdot t} + e^{-a \cdot t}}$$

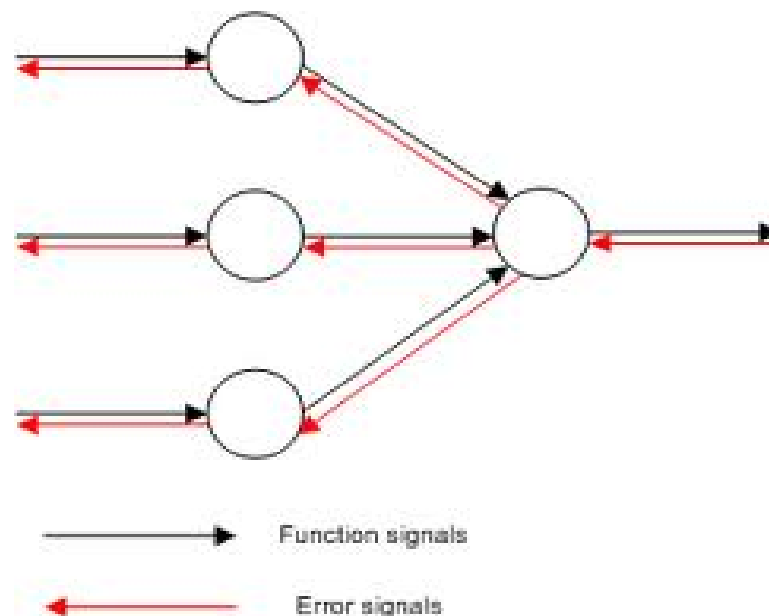


反向传播算法: 基本思想

- **反向传播(BP : back-propagation)**
 - 反向传播的是 “误差信号”

Rumelhart, Hinton & Williams, “Learning representations by back-propagating errors”, Nature, 1986

- **两个计算过程：**
 - 信号的前向通过
 - 产生网络的实际响应
 - 权值固定
 - 误差的反向通过
 - 提供修正权值的 “误差” 信号
 - 权值被调整



反向传播算法: 两种工作方式

- 串行方式

– 定义代价函数: $\varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$

– 提交一个样本, 进行一次前向运算, 再进行一次反向运算

- 每提交1个样本, 更新一次权值

- 集中方式

– 定义代价函数: $\varepsilon_{\text{ini}} = \frac{1}{2L} \sum_{n=1}^L \varepsilon(n) = \frac{1}{2L} \sum_{n=1}^L \sum_{j \in C} e_j^2(n)$

– 提交一个样本, 进行一次前向运算, 等待全部N个样本都提交完毕再进行反向运算

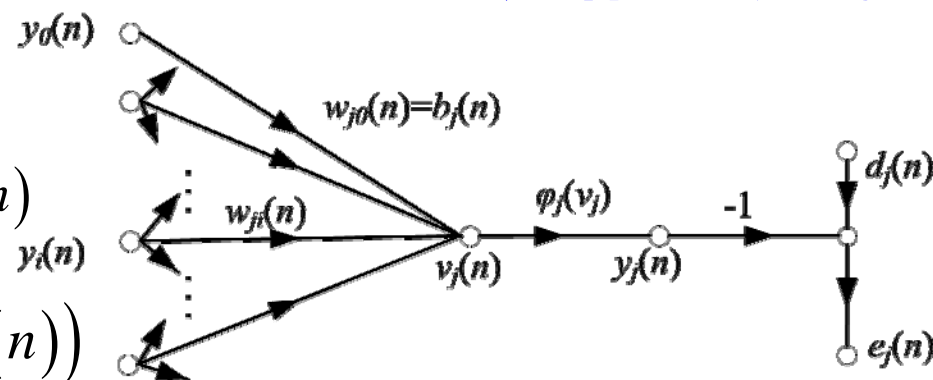
$$\Delta w_{ji}(n) = -\eta \cdot \frac{\partial \varepsilon_{\text{ini}}}{\partial w_{ji}} = -\frac{\eta}{N} \sum_{n=1}^N e_j(n) \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$$

反向传播算法：各处信号的定义

- 误差信号定义： $e_j(n) = d_j(n) - y_j(n)$
 - 如果神经元j是输出节点，则误差可直接测量
- “可见”误差的瞬时能量： $\varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$ $\varepsilon_j(n) = \frac{1}{2} e_j^2(n)$
 - C: 输出层的所有神经元

这里介绍串行方式

- 均方误差能量： $\varepsilon_{av} = \frac{1}{N} \sum_{n=1}^N \varepsilon(n)$
 - 对所有n，对瞬时能量求和
- 诱导局部域： $v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$
 - 神经元j的激活函数输入处
- 输出的函数信号： $y_j(n) = \varphi_j(v_j(n))$
 - 神经元j输出处的函数信号
- 突触权值的修正：
 - 权值修正量正比于偏导 $w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n)$



$$\Delta w_{ji}(n) \propto \partial \varepsilon(n) / \partial w_{ji}(n)$$

偏导怎样计算？

串行方式工作的反向传播算法

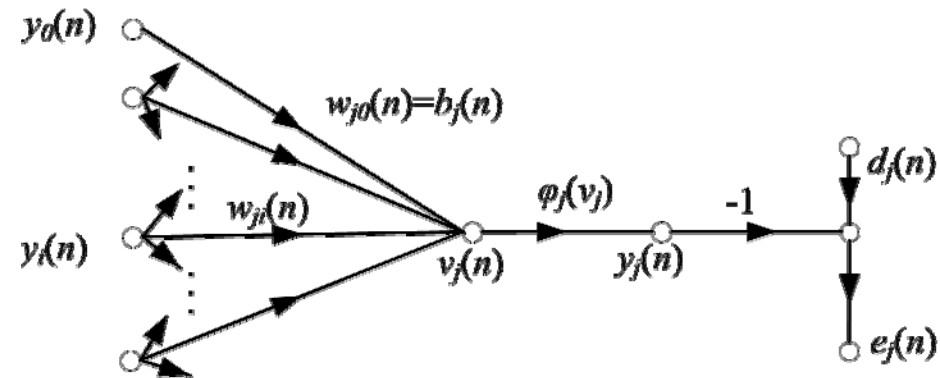
- 误差信号定义： $e_j(n) = d_j(n) - y_j(n)$ $\varepsilon_j(n) = \frac{1}{2} e_j^2(n)$
 - 神经元j是输出节点
- “可见”误差的瞬时能量： $\varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$
 - **C**: 输出层的所有神经元

- 诱导局部域：
$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$$

偏导的计算：

$$\begin{aligned} \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} &= \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \\ &= -e_j(n) \varphi'(v_j(n)) y_i(n) \end{aligned}$$

- 输出的函数信号：
$$y_j(n) = \varphi_j(v_j(n))$$
- 突触权值的修正：
 - 权值修正量正比于偏导
$$\Delta w_{ji}(n) \propto \partial \varepsilon(n) / \partial w_{ji}(n)$$



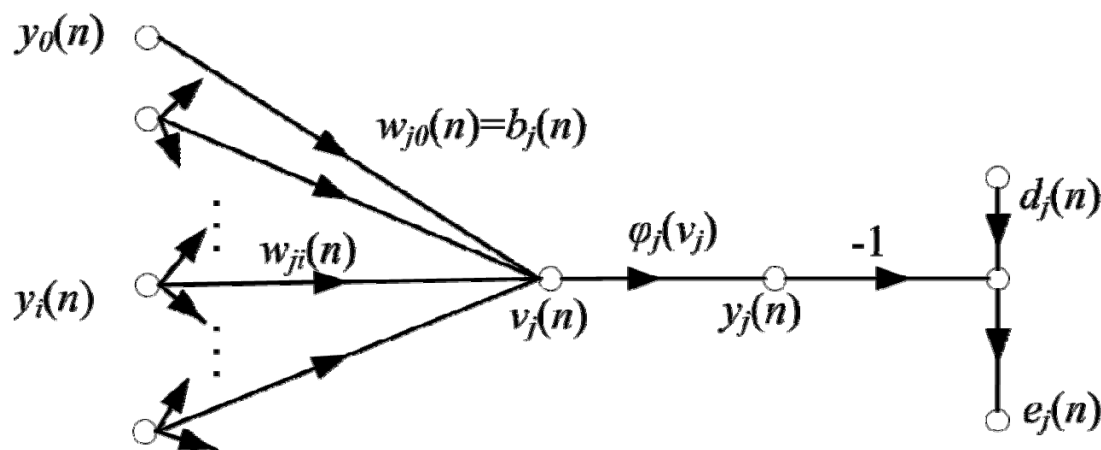
输出层神经元的权值修正法则

- 权值修正的delta法则 $w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n)$

– 其中 $\Delta w_{ji}(n) = -\eta \cdot \partial \varepsilon(n) / \partial w_{ji}(n)$

– 即 $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} = -e_j(n) \varphi'(v_j(n)) y_i(n)$

$$w_{ji}(n+1) = w_{ji}(n) + \eta e_j(n) \varphi'_j(v_j(n)) y_i(n)$$



反向传播算法？哪里
有反向传播的影子。。。



隐藏层的权值如何调整？
其误差信号如何计算？

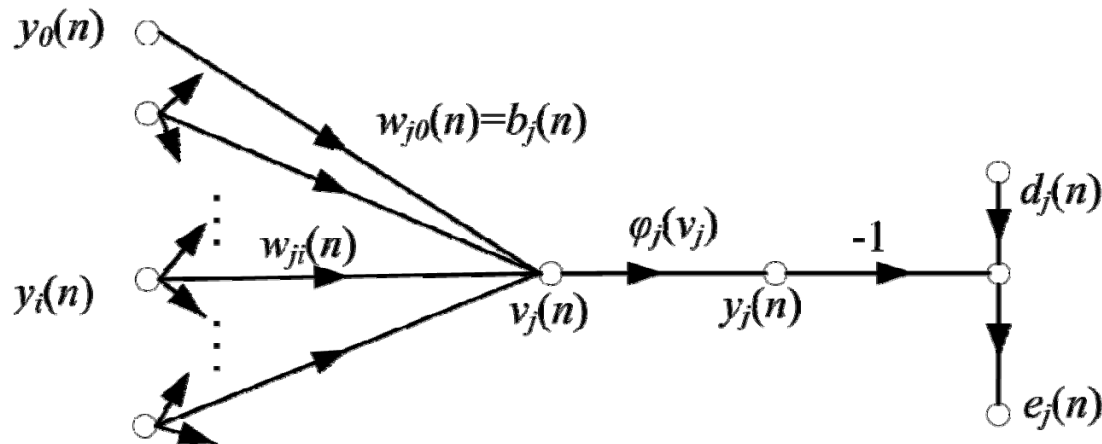
定义神经元的局部梯度

- 神经元局部梯度的定义： $\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)}$

– 对于输出层神经元j

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi'_j(v_j(n))$$

➡ $\Delta w_{ji}(n) = \eta e_j(n) \varphi'_j(v_j(n)) y_i(n) = \eta \delta_j(n) y_i(n)$



定义神经元的局部梯度

- 神经元局部梯度的定义： $\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)}$

– 对于输出层神经元 j

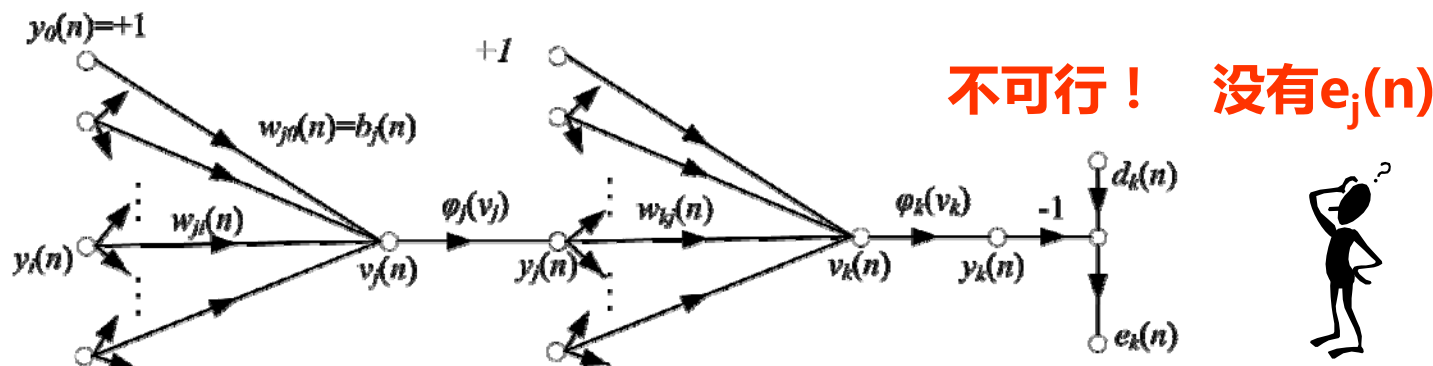
$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi'_j(v_j(n))$$

– 对于隐藏层神经元 j

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

X

如此链导可行么？



隐藏层神经元的局部梯度计算

- 神经元局部梯度的定义

- 对于隐藏层神经元j $\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n))$

- 其中 $\varepsilon(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$, $e_k(n) = d_k(n) - y_k(n)$, $y_k(n) = \varphi_k(v_k(n))$,

- 于是 $v_k(n) = \sum_{j=0}^m w_{kj}(n) y_j(n)$

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k \in C} e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_{k \in C} e_k(n) \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

$$= \sum_{k \in C} e_k(n) (-\varphi'_k(v_k(n))) w_{kj}(n) = -\sum_{k \in C} \delta_k(n) w_{kj}(n) \quad \text{其中:}$$

$$= -\tilde{e}_j(n) \quad \text{神经元 j 应该承担的“误差信号”} \quad \delta_k(n) = e_k(n) \varphi'_k(v_k(n))$$

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n)) = \tilde{e}_j(n) \varphi'_j(v_j(n))$$

误差信号反向传播的过程

- 输出层

$$\delta_k(n) = e_k(n) \left(\varphi'_k(v_k(n)) \right)$$

隐藏层神经元j被“折算出来”的误差信号

- 隐藏层

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n)) = \varphi'_j(v_j(n)) \tilde{e}_j(n)$$

其中：

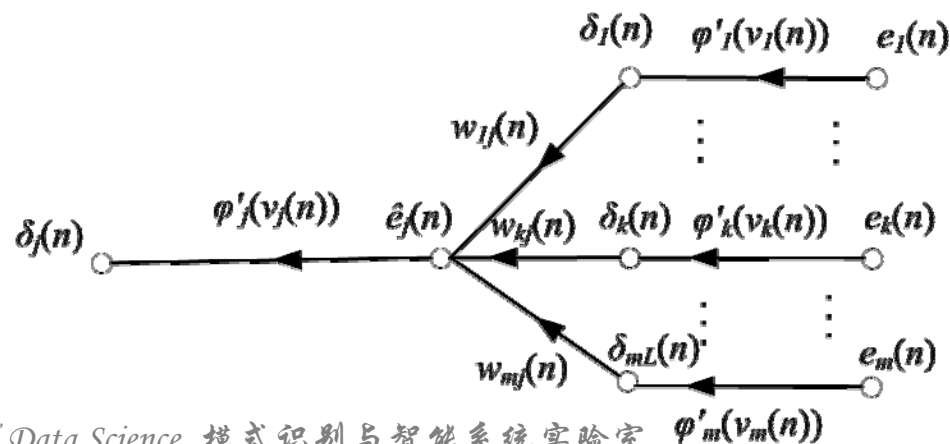
$$\tilde{e}_j(n) = \sum_{k \in C} \delta_k(n) w_{kj}(n)$$

无法直接测量隐藏层的误差，但可以“折算”隐藏层的误差

- 隐藏层误差信号涉及：

– 对所有与隐藏层直接相连的神经元k的局部梯度加权求和

$$\tilde{e}_j(n) = \sum_{k \in C} \delta_k(n) w_{kj}(n)$$



误差信号分析与权值更新

- 误差信号分析与权值更新法则：

- 输出节点 j : $e_j(n) \rightarrow \delta_j(n) = e_j(n) \varphi'_j(v_j(n))$

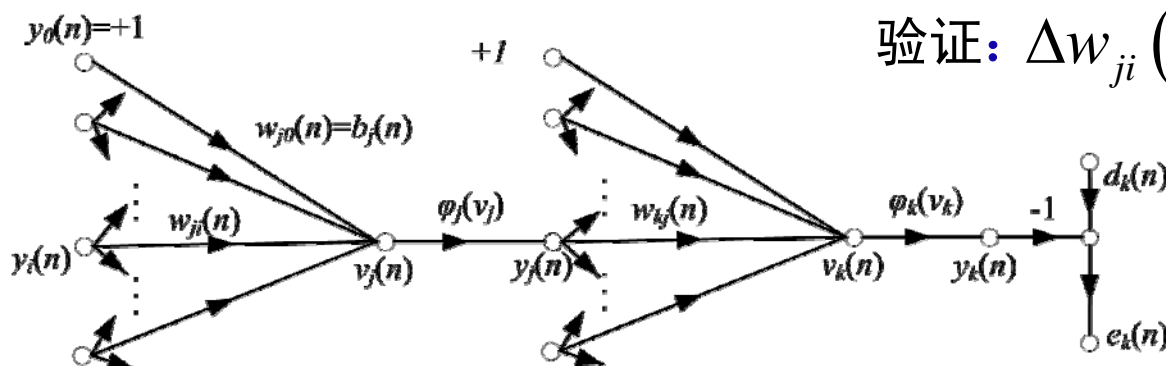
- $\rightarrow \Delta w_{ji}(n) = \eta e_j(n) \varphi'_j(v_j(n)) y_i(n) = \eta \delta_j(n) y_i(n)$

- 隐藏层节点 j : $\tilde{e}_j(n) = -\sum_{k \in C} \delta_k(n) w_{kj}(n)$

隐藏层的误差信号需通过
所有与隐藏层直接相连
的神经元的误差递归计算

- $\rightarrow \delta_j(n) = \tilde{e}_j(n) \varphi'_j(v_j(n))$

- $\rightarrow \Delta w_{ji}(n) = \eta \tilde{e}_j(n) \varphi'_j(v_j(n)) y_i(n) = \eta \delta_j(n) y_i(n)$



验证: $\Delta w_{ji}(n) = -\eta \cdot \partial \varepsilon(n) / \partial w_{ji}(n)$

$$= -\eta \frac{\partial \varepsilon(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

权值修正的delta规则

- 由神经元 i 连接到神经元 j 的权值的校正值由delta规则定义如下：

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$

学习速率参数

权值校正

局部梯度

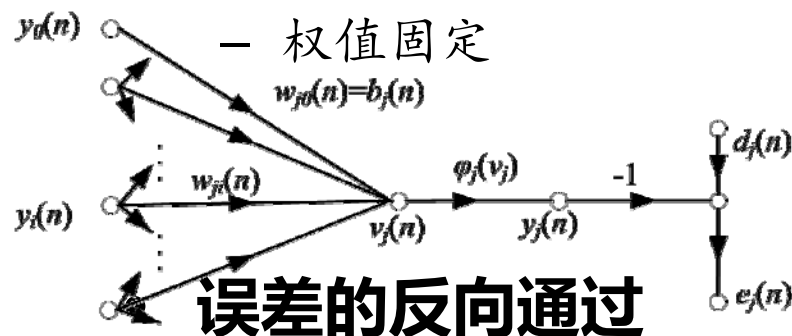
神经元j的输入信号

- 局部梯度取决于**神经元j**是一个输出层节点还是隐藏层节点
 - 神经元 j 是输出层节点： $\delta_j(n) = \varphi'_j(v_j(n))e_j(n)$
 - 神经元 j 是隐藏层节点： $\delta_j(n) = \varphi'_j(v_j(n)) \sum_{k \in C} \delta_k(n) w_{kj}(n)$
 $= \varphi'_j(v_j(n)) \tilde{e}_j(n)$

小结：两个计算过程

• 信号的前向通过

- 产生网络的实际响应 $y_j(n) = \varphi_j(v_j(n)) \quad v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$
- 权值固定



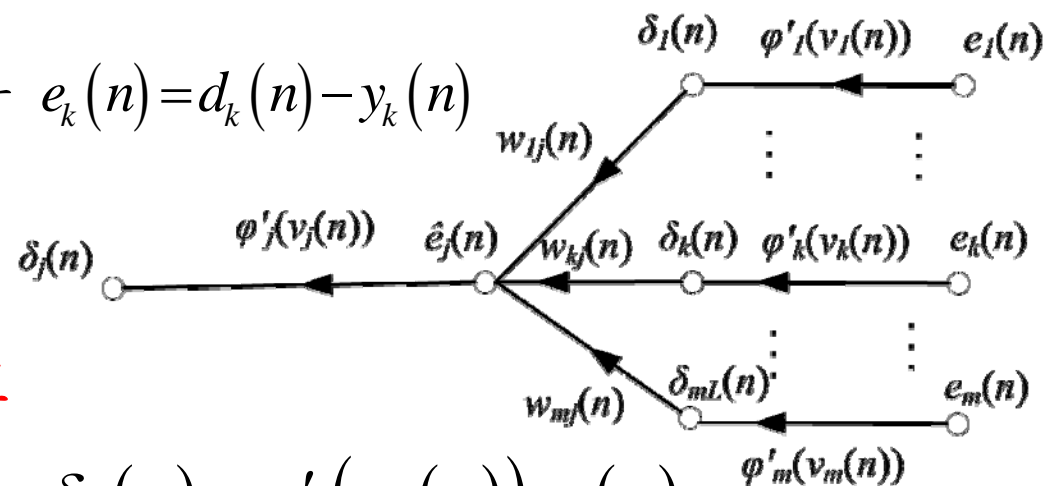
若i在输入层: $y_i(n) = x_i(n)$

若j在输出层: $y_j(n) = o_j(n)$

误差的反向通过

- 提供修正权值的“误差”信号 $e_k(n) = d_k(n) - y_k(n)$
- 权值调整

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$



其中, 局部梯度递推计算

1. 神经元j在输出层: $\delta_j(n) = \varphi'_j(v_j(n)) e_j(n)$

2. 神经元j在隐藏层: $\delta_j(n) = \varphi'_j(v_j(n)) \tilde{e}_j(n)$

激活函数

- Logistic函数

$$\varphi(t) = \frac{1}{1 + \exp(-a \cdot t)} \quad \varphi'(t) = a \cdot \varphi(t)(1 - \varphi(t))$$

$$\varphi'_j(v_j(n)) = a \cdot \varphi(v_j(n))(1 - \varphi(v_j(n))) = a \cdot y_j(n)(1 - y_j(n))$$

若j在输出层: $y_j(n) = o_j(n)$

$$\delta_j(n) = \varphi'_j(v_j(n))e_j(n) = a \cdot (d_j(n) - o_j(n)) \cdot o_j(n) \cdot (1 - o_j(n))$$

若j在隐藏层:

$$\begin{aligned} \delta_j(n) &= \varphi'_j(v_j(n)) \sum_{k \in C} \delta_k(n) w_{kj}(n) \\ &= a \cdot y_j(n)(1 - y_j(n)) \sum_{k \in C} \delta_k(n) w_{kj}(n) \end{aligned}$$

激活函数

- 双曲正切 $\varphi(t) = \tanh(a \cdot t) = \frac{e^{a \cdot t} - e^{-a \cdot t}}{e^{a \cdot t} + e^{-a \cdot t}}$
 $\varphi'(t) = a \cdot (1 - \varphi(t))(1 + \varphi(t))$

若j在输出端:

$$\delta_j(n) = \varphi'_j(v_j(n)) e_j(n)$$

若j在隐藏层:

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_{k \in C} \delta_k(n) w_{kj}(n)$$

串行方式BP算法小结

- **给定训练数据集** $D = \{(\mathbf{x}_i, \mathbf{d}_i)\}_{i=1}^N$

1. 初始化：随机选择权值和阈值(均匀分布)

2. 训练样本呈现：每次随机地提交1个训练样本给网络

3. 前向计算：

- 在第 l 层的神经元 j 的诱导局部域为：
$$v_j^{(l)}(n) = \sum_{i=0}^m w_{ji}^{(l)}(n) y_i^{(l-1)}(n)$$
- 神经元 j 的输出信号为：

$$y_j^{(l)}(n) = \varphi_j(v_j^{(l)}(n)), \quad y_j^{(0)}(n) = x_j(n), \quad y_j^{(L)}(n) = o_j(n)$$

- 计算误差信号：

$$e_j(n) = d_j(n) - o_j(n)$$

4. 反向计算：

- 计算局部梯度：
$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(l)}(n) \varphi_j'(v_j^{(L)}(n)) & j \in \text{output } L \\ \varphi_j'(v_j^{(l)}(n)) \sum_{k \in C} \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) & j \in \text{hidden } l \end{cases}$$

- 调整权值：
$$w_{kj}^{(l)}(n+1) = w_{kj}^{(l)}(n) + \eta \cdot \delta_j^{(l)}(n) y_i^{(l-1)}(n)$$

5. 若不满足停止条件，呈现训练样本继续前向和反向计算

停止准则

- **收敛准则——1**

- 梯度向量的欧氏范数足够小
 - **驻点：即误差曲面对权值向量的梯度向量为零向量**

- **收敛准则——2**

- 当每个回合的均方误差的变化的绝对速率足够小
 - **一个回合(epoch): 训练集中N个样本每个都依次呈现一次**

- **收敛准则——3**

- 在每次学习迭代后，检查网络的泛化性能
 - **反向传播的学习过程：把输入/输出映射编码为多层感知器的突触权值, 训练过程也可以看作对候选模型**

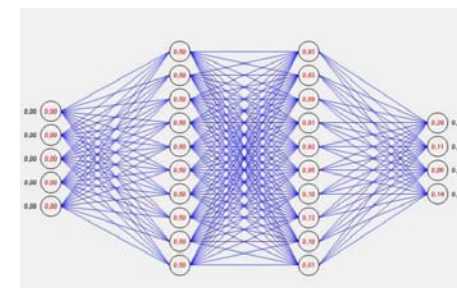
集中方式的反传算法

- 误差信号定义： $e_j(n) = d_j(n) - y_j(n)$ $\varepsilon_j(n) = \frac{1}{2} e_j^2(n)$
 - 神经元j是输出节点
- “可见”误差的瞬时能量： $\varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$
 - **C**: 输出层的所有神经元
- 均方误差能量： $\varepsilon_{av} = \frac{1}{N} \sum_{n=1}^N \varepsilon(n)$
 - 对所有n，对瞬时能量求和
- 诱导局部域： $v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$
 - 神经元j的激活函数输入处
- 输出的函数信号： $y_j(n) = \varphi_j(v_j(n))$
 - 神经元j输出处的函数信号
- 突触权值的修正： $w_{ji}(n) \leftarrow \Delta w_{ji}(n)$
 - 权值修正量正比于偏导

$$\Delta w_{ji}(n) \propto \partial \varepsilon_{av}(n) / \partial w_{ji}(n)$$

改善BP算法性能的方法

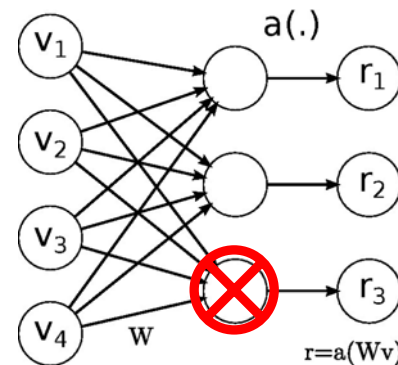
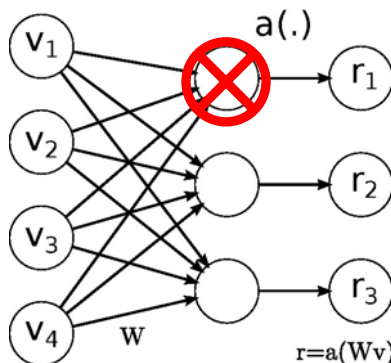
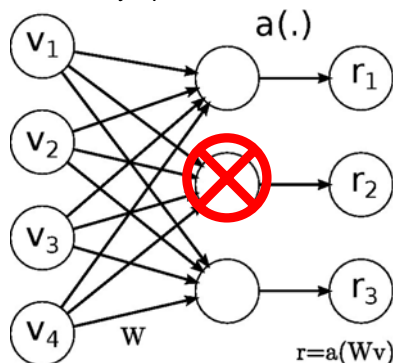
- 提高网络性能可通过：
 - 采用串行更新，而不是集中更新
 - 最大可能的信息内容
 - 使用训练误差最大的样本、把样本顺序随机化
 - 激活函数
 - 采用反对称函数，如双曲正切，ReLU
 - 目标值
 - 期望响应值要偏离sigmoid函数的极限值(采用 ϵ 扰动法)
 - 输入规整化
 - 消除均值、去相关性、协方差均衡(白化过程)
 - 初始化
 - 避免过大或过小的初始权值
 - 学习速率
 - 最后一层的学习速率应设得比别的层小
 - 扰动技术
 - Dropout (G. E. Hinton, JMLR 2014)
 - DropConnect (L. Wan et al. ICML 2013)



深度学习

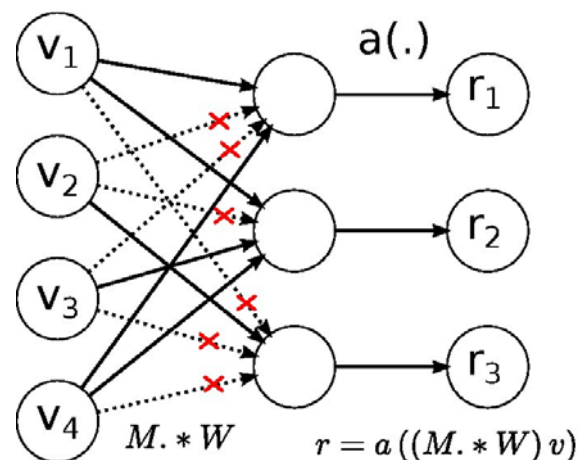
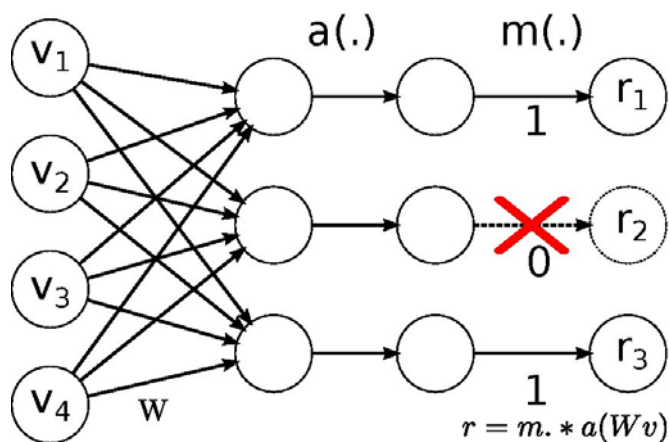
提高网络泛化能力的训练策略-1

- Dropout (G. E. Hinton, 2014)
 - 考虑使用**On-line**方式训练具有**1**个隐藏层的**2**层网络
 - 训练阶段: 每次呈现一个样本, 以**概率0.5**随机地忽略各个隐藏层**神经元**
 - 等效于我们随机地从 2^H 个不同体系结构中对结构采样
 - 正则化效果: 使权值趋向其它模型想要的值
 - 测试阶段: 从不同结构中采样, 取不同输出分布的几何均值

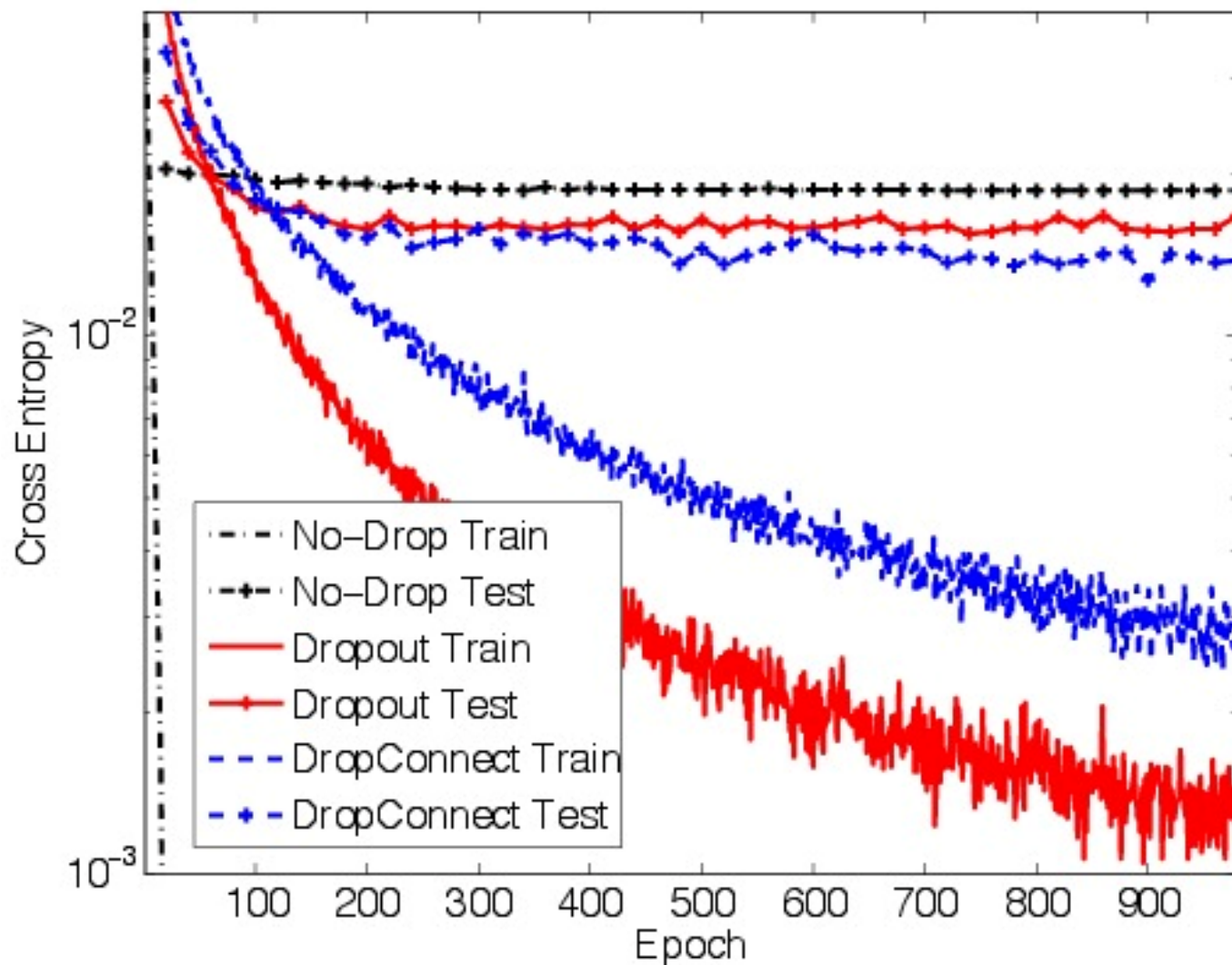


提高网络泛化能力的训练策略-2

- DropConnect (L. Wan et al. ICML2013)
 - 仅限于在全连接层(full-connect layer)使用
 - 在训练时，以概率 p ($p=0.5$)随机地忽略各个网络中的连接权值($w_{ij}=0$)
 - 在测试时，从网络结构中采样，取均值



实验性能比较



MNIST: 手写数字分类数据集

- 使用 784-800 x 800 x 10的3层网络
 - 输入维数 **784**，输出层 **10**个神经元
 - 具有**2**个隐藏层，各**800**个神经元

neuron	model	error(%) 5 network	voting error(%)
<i>relu</i>	No-Drop	1.62 ± 0.037	1.40
	Dropout	1.28 ± 0.040	1.20
	DropConnect	1.20 ± 0.034	1.12
<i>sigmoid</i>	No-Drop	1.78 ± 0.037	1.74
	Dropout	1.38 ± 0.039	1.36
	DropConnect	1.55 ± 0.046	1.48
<i>tanh</i>	No-Drop	1.65 ± 0.026	1.49
	Dropout	1.58 ± 0.053	1.55
	DropConnect	1.36 ± 0.054	1.35

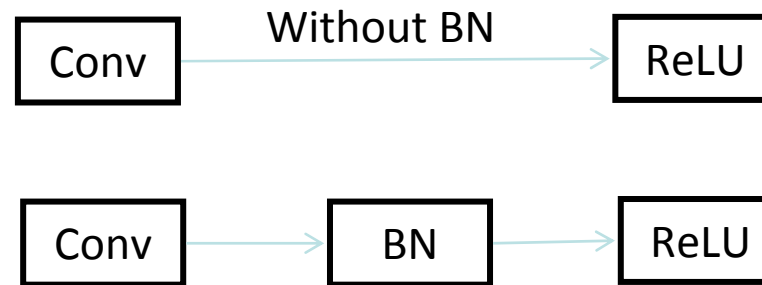
1.6% BP MLP
(Random init.)

1.4% SVM

1.2% RBM (784-500-
500-2000-10)

Batch Normalization

- 在训练过程的mini-batch中，对卷积的输出规范化



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

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

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

[1] Sergey Ioffe, Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift". ICML, 2015.

其它的有用技术

- 初始化
 - **Xavier weight initialization**
- 激活函数
 - **ReLU**
- 随机梯度下降
 - **Stochastic gradient decent (SGD)**
- 动量随机梯度下降
 - **Momentum SGD**
- Nesterov加速梯度下降
 - **Nesterov accelerated gradient (NAG)**

[1] Xavier Glorot & Yoshua Bengio, "Understanding the difficulty of training deep feed-forward neural networks", AI&STATS 2010.

动量项与学习速率

- 为了加快学习速度，同时保持稳定，通常增加一个动量项

$$\Delta w_{ji}(n) = \alpha \cdot \Delta w_{ji}(n-1) + \eta \cdot \delta_j(n) \cdot y_i(n)$$


- $1 > \alpha > 0$: 动量常数

广义delta规则

– 动量项对于避免学习过程停止在误差曲面的一个浅层局部最小是有益的

- 若偏导数在连续迭代中具有相同的符号，则动量项的存在会导致权值修正量增加
 - 若偏导数在连续的迭代中具有相反的符号，则动量项的存在会导致权值修正量减少
- 学习速率应该是依赖于连接的
 - η 应是 η_{ij} ，即网络的不同权值应使用不同的学习速率

输出的表示与决策规则

- 对于M(M>2)类的分类问题，用于网络输出的最优决策规则是什么？
最优权值下，网络实际输出值是期望响应向量的条件期望的LMS估计，
对于1对M分类问题，期望响应向量的条件期望等于后验类概率 $P(C_k|x)$
 - y_{kj} 表示网络响应于 \mathbf{x}_j 的第k个神经元的输出： $y_{kj} = F_k(\mathbf{x}_j)$
 - 定义向量值函数 $\mathbf{F}(\mathbf{x}_j)$
$$\mathbf{y}_j = (y_{1,j}, y_{2,j}, \dots, y_{M,j})^T = (F_1(\mathbf{x}_j), F_2(\mathbf{x}_j), \dots, F_M(\mathbf{x}_j))^T = \mathbf{F}(\mathbf{x}_j)$$
 - 连续函数、并使经验风险泛函最小： $R = \frac{1}{2N} \sum_{j=1}^N \|\mathbf{d}_j - \mathbf{F}(\mathbf{x}_j)\|^2$
第k个元素 
 - 类别 C_k 的期望目标输出： $\mathbf{d}_j = (0, 0, \dots, 1, \dots, 0)^T$
 - 适当的决策规则是由后验概率估计产生的近似Bayes规则
 - 如果 $F_k(\mathbf{x}) > F_j(\mathbf{x}) \quad \forall j \neq k$ 则把输入向量 \mathbf{x} 分类为 C_k ，其中
 $F_k(\mathbf{x})$ 和 $F_j(\mathbf{x})$ 是下列向量值函数的分量
$$\mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), F_2(\mathbf{x}), \dots, F_M(\mathbf{x}))^T$$

[1] Richard & Lippmann: Neural Network classifiers estimate Bayesian a posteriori probabilities, Neural Computation. 1991.

隐藏层神经元的角色

- 特征检测器：
 - 通过非线性变换把输入数据变换到隐藏空间(或特征空间)中，从而发现训练数据的潜在特征
 - 多层分类器网络的最优内在表达完成非线性判别分析

[1] Webb A. R. and Lowe D.: The optimal internal representation of multilayer classifier networks performs nonlinear discriminant analysis, Neural Networks, vol.5, pp.480-488, 1990.

- 线性判别分析：假设有2个类别 C_1 和 C_2
 - » 在分类意义上，从多维到1维的最优线性变换

专题 三：线性模型的扩展

- 内容提要
 - 引言
 - 广义线性模型
 - 核方法
 - 多层感知器(MLP)
 - 反向传播算法

Q / A

- Any Question? ...

分层(Hierarchical)学习机器举例

- Perceptron → MLP
- Clustering
 - **Hierarchical Clustering**
- Gaussian 模型
 - → **Gaussian Model of Mixture**
 - → **Hierarchical Gaussian Mixture Model**
- Mixture of Experts (混合专家) → Hierarchical ME
- Topic model中, Latent Dirichlet Allocation
 - → **Hierarchical Dirichlet Process**
- 在稀疏编码中,
 - **Sparse coding**
 - → **Hierarchical Sparse coding**
- 在Kernel machine中,
 - **Kernel Machine** → **Hierarchical Kernel Machine**
- **Deep Learning** → ...

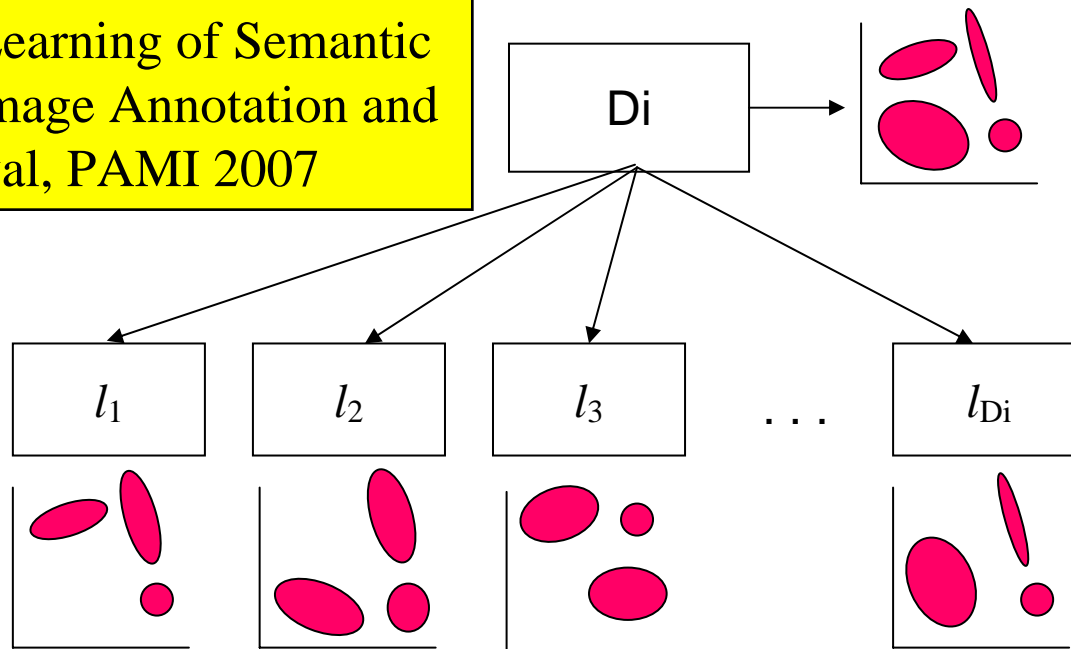
分层的GMM

- 举例：

Learning Mixture Hierarchies, NIPS1998

- 每个类别包含多张图片
- 每张图片训练一个**GMM**
- 由多个**GMM**获得该类的**GMM**

Supervised Learning of Semantic
Classes for Image Annotation and
Retrieval, PAMI 2007



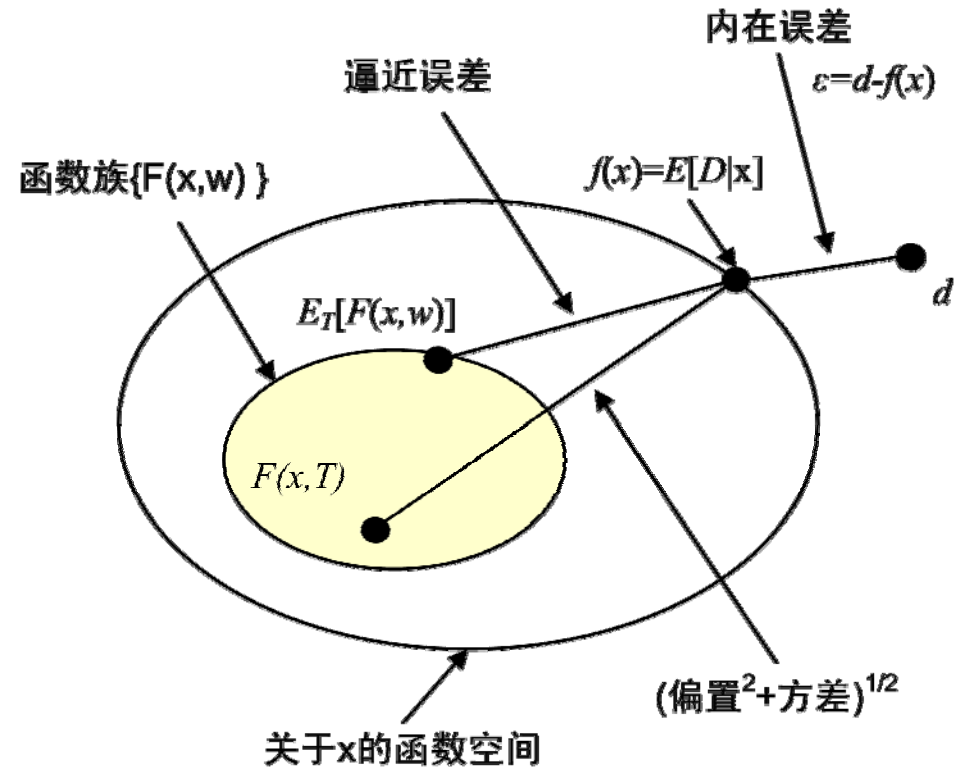
Q / A

- Any Question? ...

多层感知器网络

- 权值更新算法

- 牛顿法
- 拟牛顿法
- 共轭梯度法



- 可以作为通用函数逼近器

- 理论上，一个隐藏层就足够作为通用函数逼近器
 - 学习时间？实现难易程度？泛化能力？未作回答。

统计学习理论界对神经网络的评价

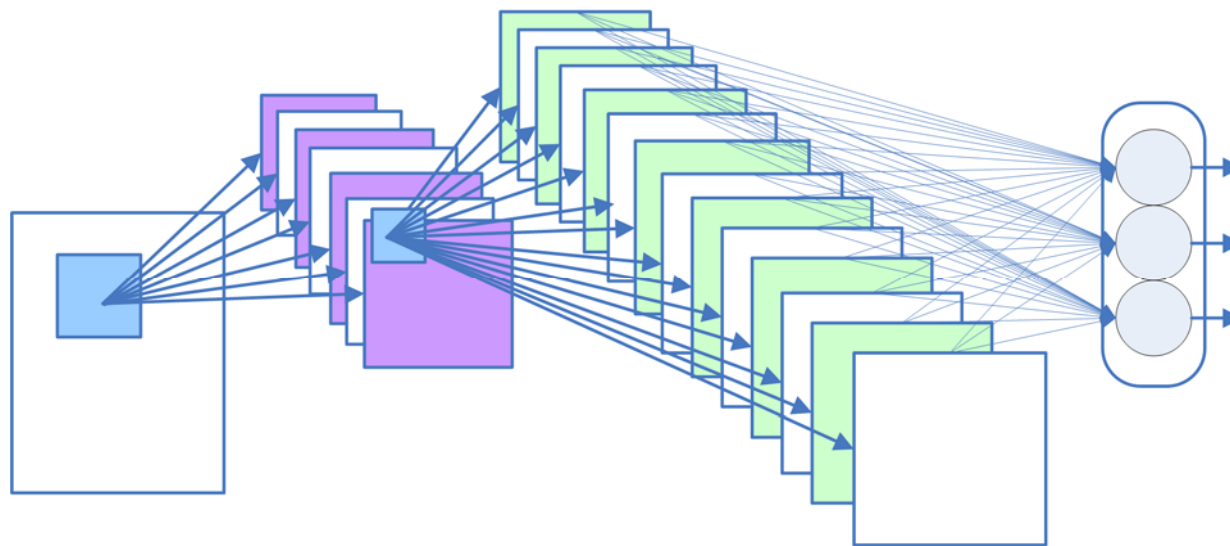
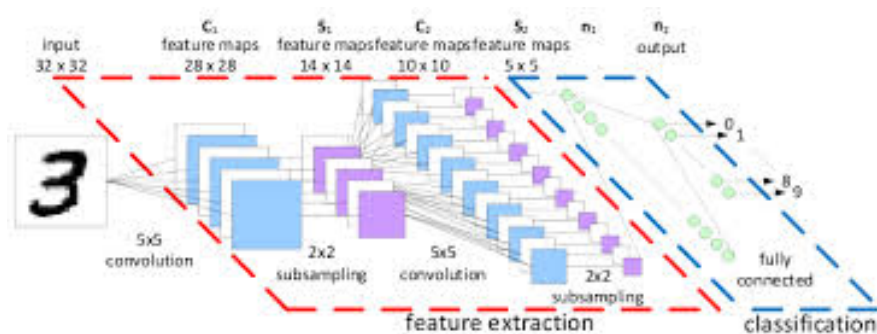
- “神经网络的存在可以看作是对理论学家的挑战。”



- “从严格的角度看, 我们无法保证神经网络能够较好地推广, 因为根据理论, 我们需要控制两个因素: 经验风险值和置信范围值. 然而, 神经网络不能控制两者中的任何一个.
- “然而, 神经网络的设计者们用高超的工程技巧弥补了数学上的缺陷.....从而使得神经网络显示了出人意料的好结果.”

卷积神经网络 (CNN)

- 基本结构:



卷积神经网络 (CNN)

- CNN的每层由下述4个基本单元构成:
 - 卷积(**Convolution**):
 - 使用一组学习到的filters检测局部特征
 - ReLU校正(**Rectification**):
 - $\text{ReLU}(x) = \max(x, 0)$
 - 在邻域内汇集(**Pooling**):
 - 把一定区域内的特征融合起来
 - 局部对比规范化(**Local Contrast Normalization**)

[1] Krizhevsky, A., Sutskever, I., and Hinton, G.E. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

1个8层CNN的体系结构

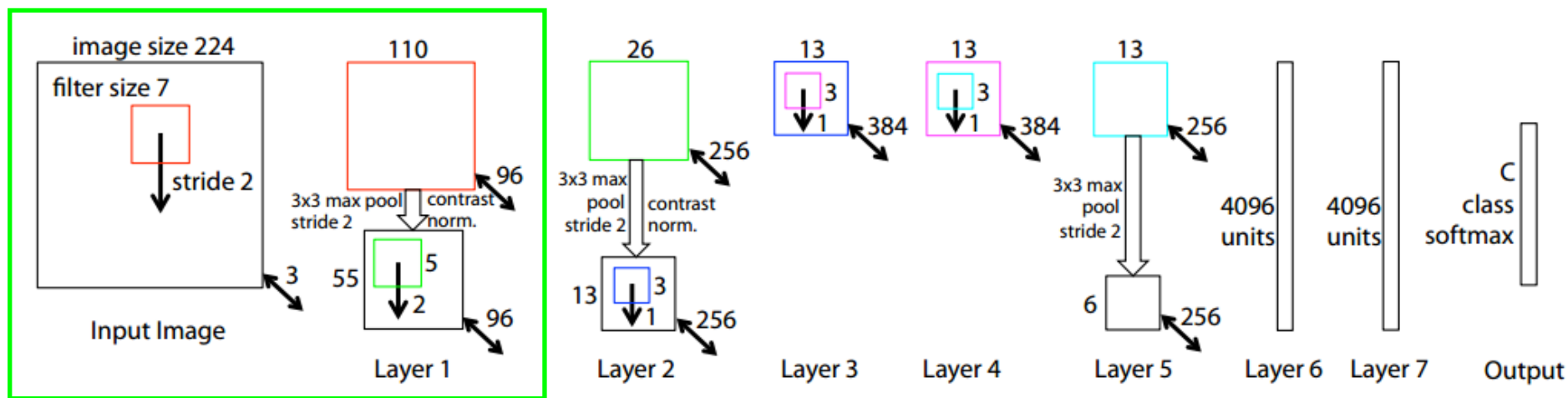


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks",
<http://arxiv.org/pdf/1311.2901.pdf>.

机器学习与数据科学 - Machine Learning & Data Science 模式识别与智能系统实验室

1个8层CNN的体系结构

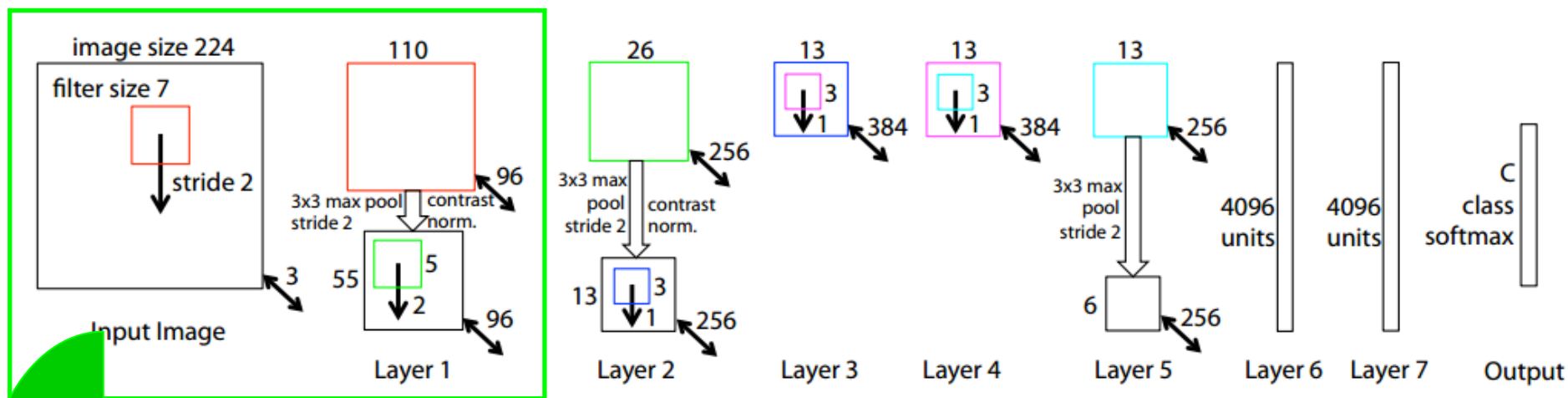


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

- Stride 2: 相邻窗口每次滑动2个像素(或单位), i.e. 5个像素重叠(overlapping)
- Filter size 7: 使用7x7的kernel作用在每个窗口内, 输出一个响应值
- 3x3 max pooling: 下采样(Downsampling)步骤, 在一个3x3邻域内的9个响应值里选取最大值作为输出
 - 3x3 max pooling stride 2: 相邻的3x3窗口每次滑动2个像素, 即重叠1个像素

1个8层CNN的体系结构

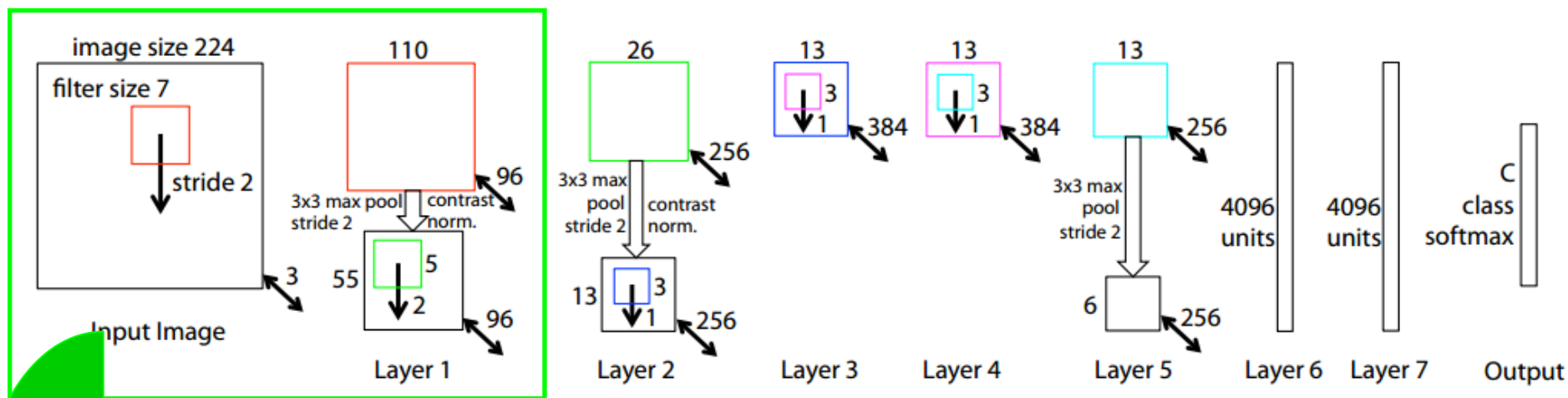


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

- 输入为彩色图像，包含3个通道
- $110 \times 110 \rightarrow 55 \times 55$: 使用3x3 max pooling with stride 2之后尺寸减半
- 3通道 \rightarrow 96 通道: 选用96个7x7x3的filters对224x224x3的图像卷积，每个filter给出一个特征图(feature map)

1个8层CNN的体系结构

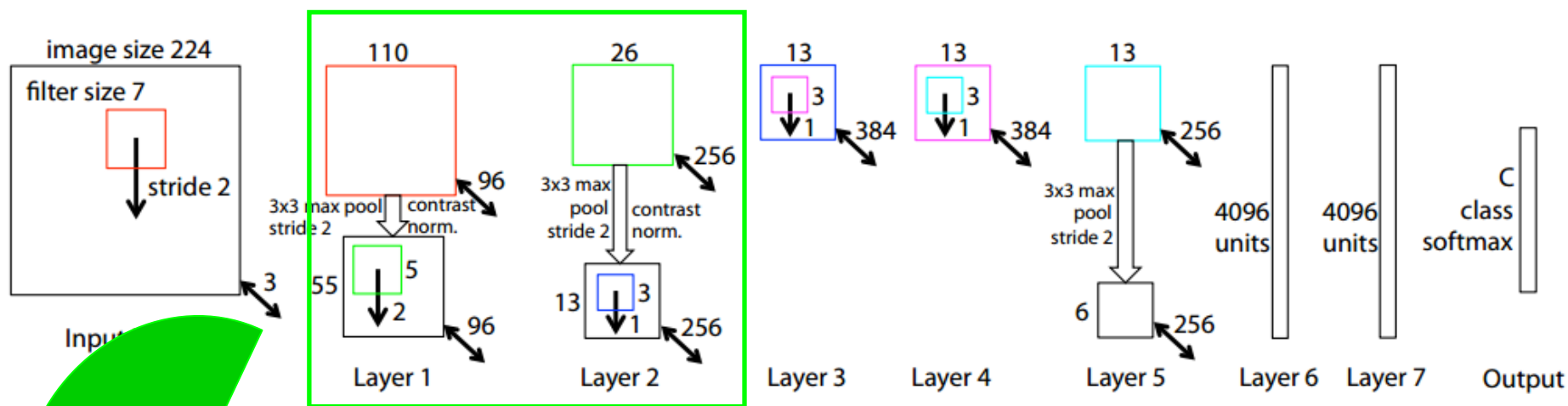


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax layer, C being the number of classes. All filters and feature maps are square in shape.

第1层输出为96个55x55的feature map，可以看做96通道的“特征图”

- 第2层选用256个filters，其中每个filter是5x5x96，作用在55x55的特征图上，得到256个26x26的feature maps，然后MaxPooling之后变成13x13x256
- 第6层和第7层为全连接层(Full Connected Layers)

CNN的不同实现

- LeNet

- [1] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." *Neural computation*, 1989.
- [2] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* (1998): 2278-2324.

- AlexNet

- [3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *NIPS*. 2012.

- VGGNet

- GoogleNet

- ResNet

- He, K., Ren, S., Sun, J., & Zhang, X.: *Deep Residual Learning for Image Recognition*, CVPR 2016.

Residual network (ResNet)

- ResNet的基本模块

$$y = \mathcal{F}(x) + x.$$

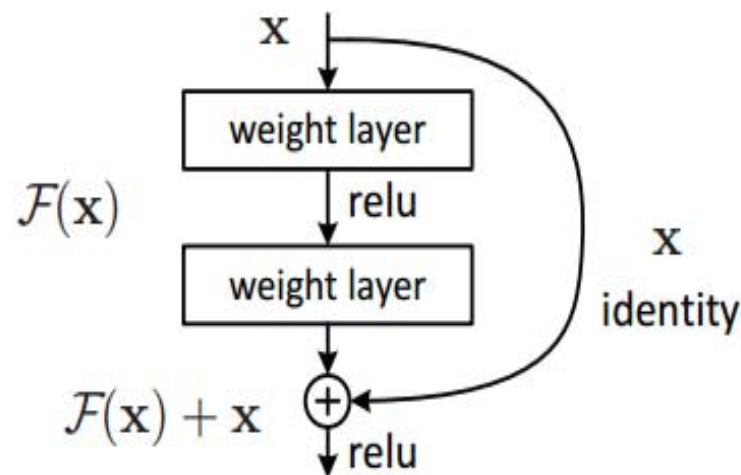
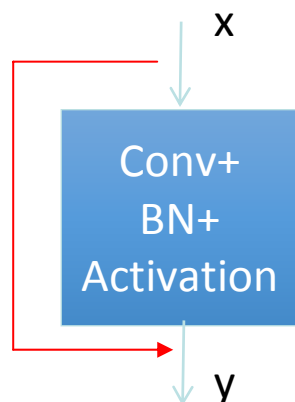


Figure 2. Residual learning: a building block.

He, K., Ren, S., Sun, J., & Zhang, X.: Deep Residual Learning for Image Recognition, CVPR 2016.

相关参考文献

- [1] Hinton, G.E., Krizhevsky, A., Srivastava, N., Sutskever, I., & Salakhutdinov, R. : **Dropout**: a simple way to prevent neural networks from overfitting, Journal of Machine Learning Research, 15, 1929-1958, 2014.
- [2] He, K., Ren, S., Sun, J., & Zhang, X.: Deep Residual Learning for Image Recognition, arXiv, 2016.
- [3] Sergey Ioffe, Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift". ICML, 2015.
- [4] Fei-Fei, L., Karpathy, A., Leung, T., Shetty, S., Sukthankar, R., & Toderici, G. (2014): Large-Scale Video Classification with Convolutional Neural Networks. IEEE CVPR.
- [5] Bengio, Y., Courville, A.C., Goodfellow, I.J., Mirza, M., Ozair, S., Pouget-Abadie, J., Warde-Farley, D., & Xu, B. (2014): Generative Adversarial Nets, NIPS.
- [6] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks", AI&STATS 2010.