神经网络(1)

南开大学网络空间安全学院 计算机学院

引言

- 回归和分类模型
  - 使用固定基函数的线性组合
  - 具有有用的解析和计算性质，但实际应用受到维数灾难的限制
  - 应用大规模问题时有必要根据数据调整基函数

- 前馈神经网络
  - 预先确定基函数的数目
  - 在训练过程中，自适应地调整基函数的参数

前馈网络函数

- 回归和分类线性模型
  - 基于固定非线性基函数 $\phi_j(\mathbf{x})$ 的线性组合

$$y(\mathbf{x},\mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

其中：$f(\cdot)$ 是非线性激活函数（分类）或恒等函数（回归）

- 目标：扩展模型
  - 参数化基函数 $\phi_j(\mathbf{x})$
  - 训练过程中，调整参数和系数

基本神经网络模型

- 构造 M 个输入变量 $x_1,\ldots,x_D$ 的线性组合（称为净输入）

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

其中，$j=1,\ldots,M$，上标(1) 表示对应参数是网络第一层的。

- 激活函数 $h(\cdot)$
  - 可导且非线性
  - 通常选择 sigmoid 函数或 tanh 函数

$$z_j = h(a_j)$$

这些量对应着基函数的输出，称为隐含单元(hidden unit)。

基本神经网络模型

- 输出单元激活值

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

其中 $k = 1, \ldots, K$， K 是输出总数。

- 网络输出

$$y_k = \sigma(a_k)$$

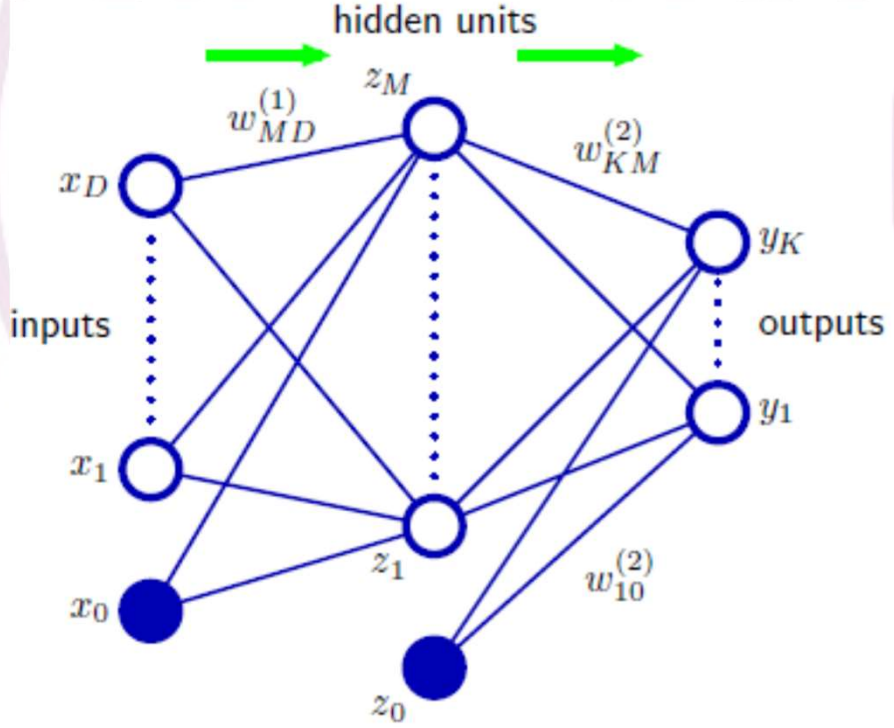
  - 标准回归问题，激活函数选择恒等函数
  - 多个二值分类问题，每个输出单元激活函数使用 logistic sigmoid函数
  - 多类问题，激活函数选择 softmax 函数

基本神经网络模型

- 整个网络函数

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

其中，所有权值和偏差参数表示为矢量 w。



基本神经网络模型

- 定义：附加输入变量 $x_0 = 1$，则有

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

和

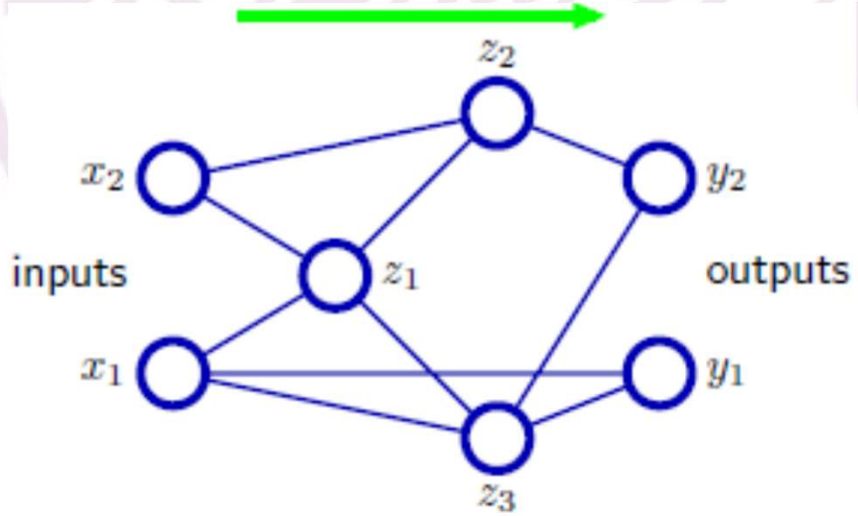
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right)$$

- 与感知器的区别
  - 前馈神经网络模型每层都类似感知器模型，故得名多层感知器 (multilayer perceptron, MLP)
  - 前馈神经网络在隐含层使用连续的 S 形非线性函数，而感知器使用阶梯非线性函数
  - S 形非线性函数是可微的，这个性质在网络训练中起着主要作用。

前馈神经网络模型

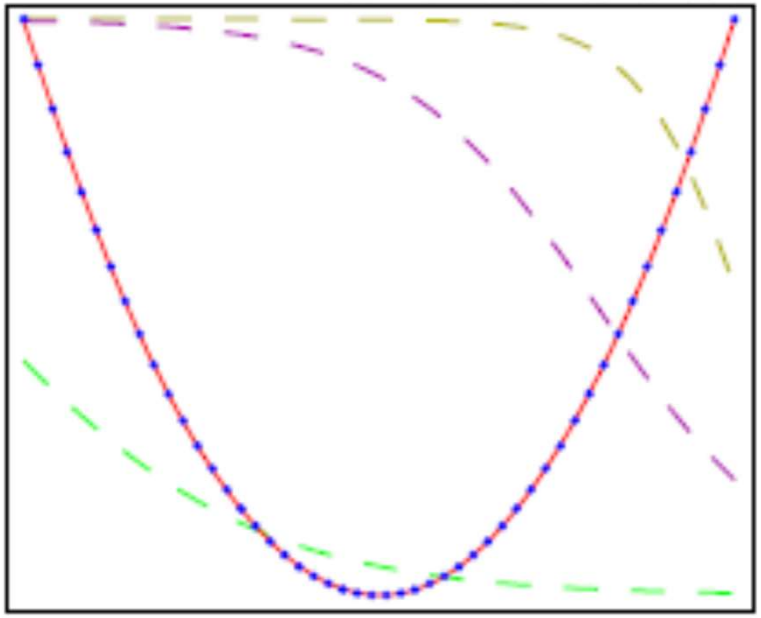
- 性质：如果网络中所有隐含单元的激活函数都是线性的，则总能找到一个没有隐含层的等价网络。

- 一般网络
  - 附加处理层：由加权线性组合形式组成，然后使用非线性激活函数进行元素间转换。
  - 跨层(skip-layer)连接：每个都有相应的自适应参数

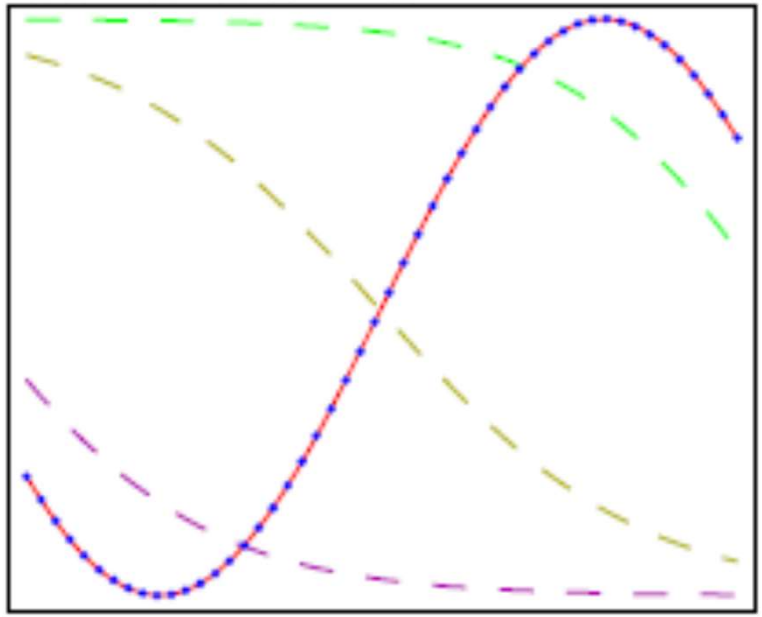


前馈神经网络模型

- 通用逼近器(universal approximators)：只要网络具有足够多的隐含单元，具有线性输出的两层网络可以在一个紧凑输入域上一致地逼近任意连续函数。
- 示意图
  - 红色曲线：网络逼近结果；虚线：隐含层输出

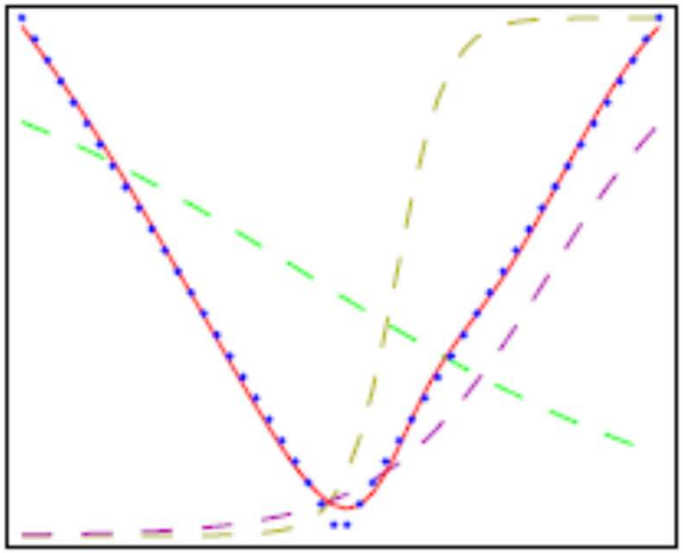


$f(x) = x^2$

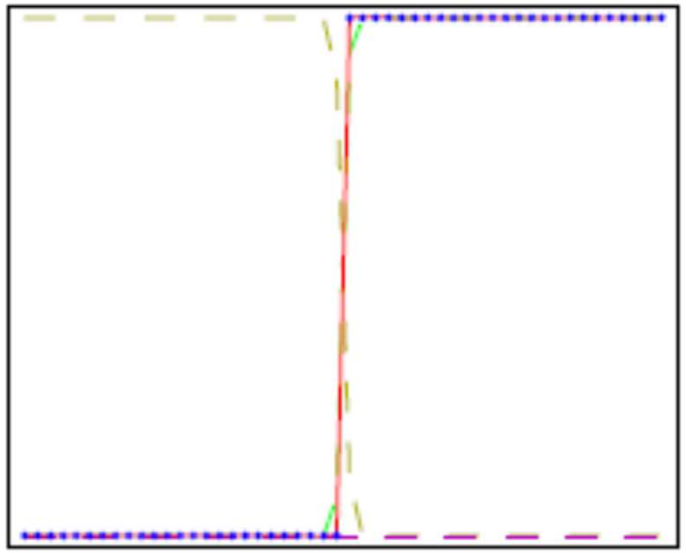


$f(x) = \sin(x)$

前馈神经网络模型



$f(x) = |x|$



$f(x) = \mathrm{H}(x)$ Heaviside step

权空间对称性

- 前馈网络的性质：权矢量 w 的多种不同选择都可以产生从输入到输出的相同映射函数。
  - 假设具有 M 个使用 tanh 激活函数的隐含单元和层间全连接两层网络

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} \tanh\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

  - 同时改变第 j 个隐含单元输入和输出权值，因为 $\tanh(-a) = -\tanh(a)$，则有

$$\left(-w_{kj}^{(2)}\right)\tanh\left(\sum_{i=0}^{D}\left(-w_{ji}^{(1)}\right)x_i\right) = -w_{kj}^{(2)}\left(-\tanh\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) = w_{kj}^{(2)} \tanh\left(\sum_{i=0}^{D} w_{ji}^{(2)} x_i\right)$$

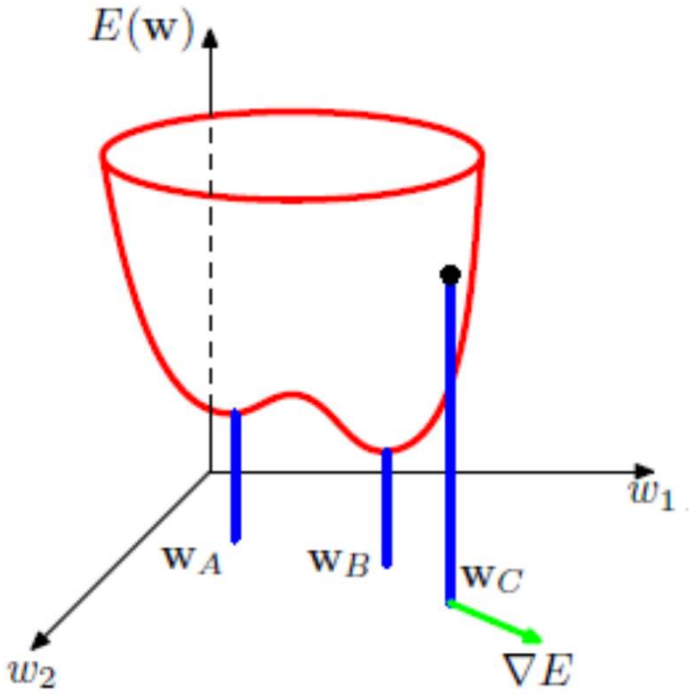
  - M 个隐含单元存在 M 个“符号翻转”对称性。
  - 任一权矢量将是 2^M 个等价权矢量之一
  - 交换任意两个隐含单元权值对应着 $M!$ 种不同的组合
  - 整个权空间的对称因子是 $M!2^M$

网络训练

- 已知：训练样本集合 $\{\mathbf{x}_n, \mathbf{t}_n\}, n = 1, \ldots, N$
- 最小化误差函数

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \left\| \mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n \right\|^2$$

- 误差函数的几何意义
  - 权空间：从 w 到 w+δw 移动
  - 误差函数：$\delta E \simeq \delta \mathbf{w}^{\mathrm{T}} \nabla E(\mathbf{w})$
  - 误差最小的条件：$\nabla E(\mathbf{w}) = 0$



- 权空间中满足 $\nabla E(\mathbf{w}) = 0$ 的点
  - 全局最小值：对任何权矢量，最小误差值的点
  - 局部最小值：其它对应较高误差值的权矢量
- 成功应用：不求全局最小值，但求“足够好”的局部最小值

局部二次近似

- 权空间点 $\hat{\mathbf{w}}$ 附近误差函数 $E(\mathbf{w})$ 的 Taylor 展开

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w}-\hat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w}-\hat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w}-\hat{\mathbf{w}})$$

其中，b 是在点 $\hat{\mathbf{w}}$ 处误差函数的梯度

$$\mathbf{b} \equiv \nabla E\big|_{\mathbf{w}=\hat{\mathbf{w}}}$$

Hessian矩阵 $\mathbf{H} = \nabla\nabla E$ 的元素为

$$(\mathbf{H})_{ij} \equiv \frac{\partial E}{\partial w_i \partial w_j}\bigg|_{\mathbf{w}=\hat{\mathbf{w}}}$$

- 从 $E(\mathbf{w})$ 的 Taylor 展开公式，可以得到梯度局部近似

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w}-\hat{\mathbf{w}})$$

局部二次近似

- 如果在点$\mathbf{w}^*$处$\nabla E = 0$，有

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^{\mathrm{T}}\mathbf{H}(\mathbf{w}-\mathbf{w}^*)$$

- Hessian 矩阵的特征值方程

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

其中，特征矢量$\mathbf{u}_i$形成完全正交规范集，使得

$$\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_j = \delta_{ij}$$

- 将$\mathbf{w}-\mathbf{w}^*$表示为特征矢量线性组合的形式

$$\mathbf{w}-\mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i$$

局部二次近似

- 误差函数

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

- 定义：矩阵 H 是正定的，当且仅当

$$\mathbf{v}^{\mathrm{T}}\mathbf{H}\mathbf{v} > 0 \quad \text{for all } \mathbf{v}$$

- 因为特征矢量 $\{\mathbf{u}_i\}$ 构成完备集，任意矢量 $\mathbf{v}$ 可表示为

$$\mathbf{v} = \sum_i c_i \mathbf{u}_i$$

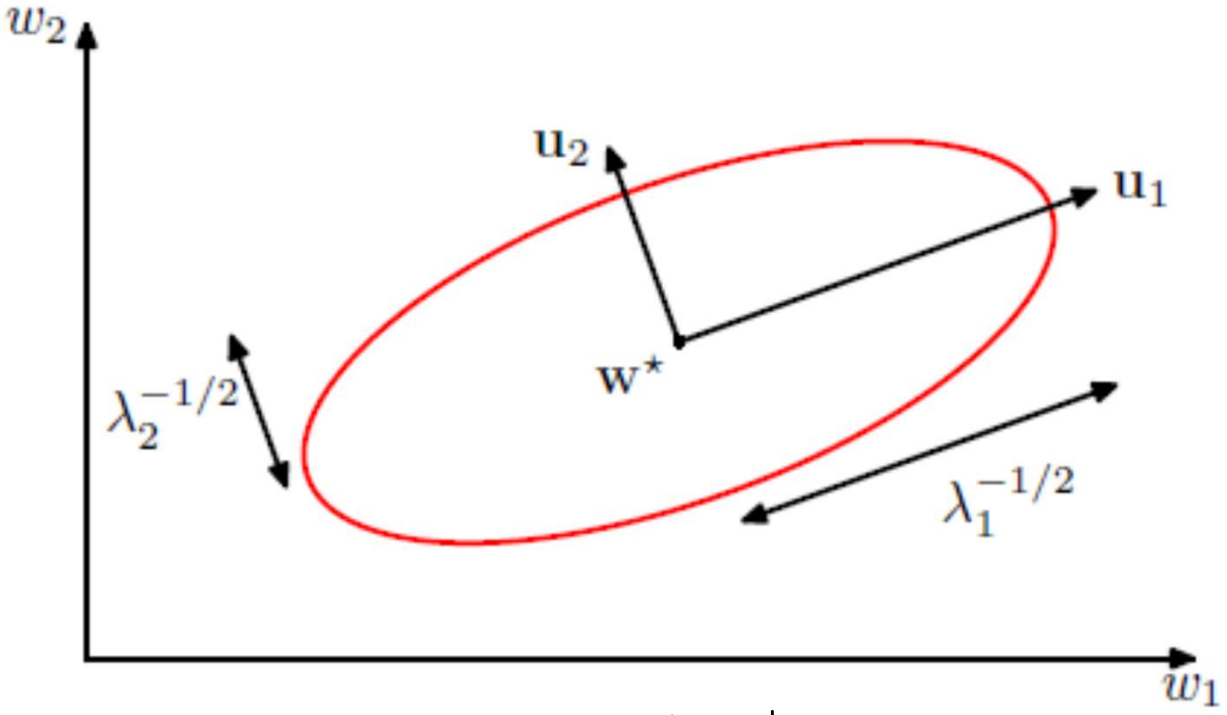
- 得到

$$\mathbf{v}^{\mathrm{T}}\mathbf{H}\mathbf{v} = \sum_i c_i^2 \lambda_i$$

当且仅当，所有特征值是正数，则 H 是正定的。

局部二次近似

- 在特征矢量 $\{\mathbf{u}_i\}$ 张开的坐标系下，常数 E 的轨迹是椭圆。



  - 稳定点$\mathbf{w}^*$是最小值的条件是 $\partial^2 E/\partial w^2\big|_{w^*} > 0$

- 在 D 维空间中误差函数 $E(\mathbf{w}^*)$ 为最小值的条件是：在 $\mathbf{w}^*$处 Hessian 矩阵是正定的。

梯度下降优化

- 使用梯度信息的权值更新公式

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right)$$

其中 $\eta > 0$ 是学习率。

- 批处理方法：使用整个训练集合计算误差函数梯度 ∇E
  - 算法直观合理，实际上糟糕。
- 随机梯度下降(stochastic gradient descent)
  - 也称为顺序梯度下降(sequential gradient descent)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n\left(\mathbf{w}^{(\tau)}\right)$$

基于对独立观测量集合最大似然的误差函数，即

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

梯度下降优化

- 随机梯度下降的优点
  - 比批处理方法计算误差函数效率更高
  - 逃离局部极小值的可能

误差反向传播

- 最小化误差函数的迭代过程
  - 计算误差函数对权矢量的导数
  - 使用导数计算权矢量的调整量

误差函数导数计算

- 误差函数通常表示为每个样本误差之和

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- 简单的线性模型

$$y_k = \sum_i w_{ki} x_i$$

- 某个输入 n 的误差函数

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

其中 $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$

误差函数导数计算

- 误差函数对权值 w_{ji} 的梯度

$$\frac{\partial E_n}{\partial w_{ji}} = \left(y_{nj} - t_{nj}\right) x_{ni}$$

  - 解释：计算连接 w_{ji} 的输出端误差信号 $y_{nj} - t_{nj}$ 和输入端变量 x_{ni} 的乘积

- 前馈网络的向前传播过程
  - 每个单元计算输入的加权和

$$a_j = \sum_i w_{ji} z_i$$

  - 通过非线性激活函数 $h(\cdot)$ 获得单元 j 的激活值 z_j

$$z_j = h\left(a_j\right)$$

  - 注意：最后一层单元的输出值表示为 y_{nj}

误差函数导数计算

- 误差 E_n 是通过单元 j 净输入 a_j 间接依赖于权值 w_{ji}，故

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

- 引入符号（通常称为误差）

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

- 因为

$$\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(\sum_i w_{ji} z_i\right) = z_i$$

有

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

误差函数导数计算

- 注意：导数

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

  - 单元权值输出端的δ值与单元权值输入端的z值的乘积

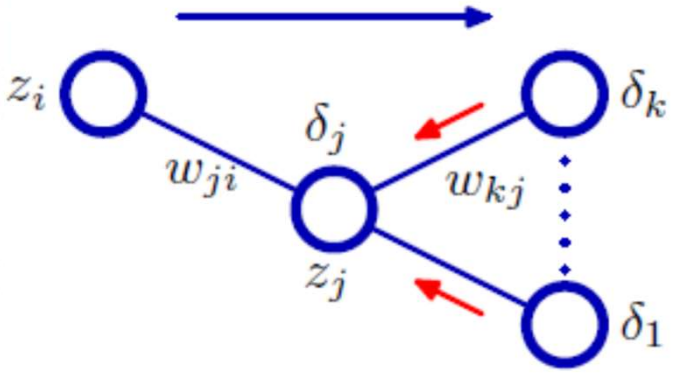
- 计算导数只需要计算网络中每个隐含和输出单元的δ_j
  - 输出单元

$$\delta_k = y_k - t_k$$

  - 隐含单元

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

对与单元 j 输出相连的所有单元 k 求和。



误差函数导数计算

- 反向传播公式

$$\delta_j = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \frac{\partial z_j}{\partial a_j}\frac{\partial a_k}{\partial z_j}\frac{\partial E_n}{\partial a_k} = h'\left(a_j\right)\sum_k w_{kj}\delta_k$$

  - 某个隐含单元的 δ 值可以通过网络中更高层单元的 δ 值反向传播得到

- 注意：
  - 对权值下标的第一个索引求和对应反向传播
  - 对权值下标的第二个索引求和对应向前传播。

反向传播算法

- 将矢量 $\mathbf{x}_n$ 输入到网络，通过公式

$$z_j = h\left(a_j\right) = h\left(\sum_i w_{ji} z_i\right)$$

在网络上向前传播，计算所有隐含和输出单元的激活值；

- 使用公式 $\delta_k = y_k - t_k$ 计算所有输出单元的 δ_k；
- 对网络中每个隐含单元，使用公式

$$\delta_j = h'\left(a_j\right)\sum_k w_{kj}\delta_k$$

反向传播 δ 值获得 δ_j；

- 使用公式

$$\partial E_n / \partial w_{ji} = \delta_j z_i$$

得到所求的导数。

误差函数导数计算

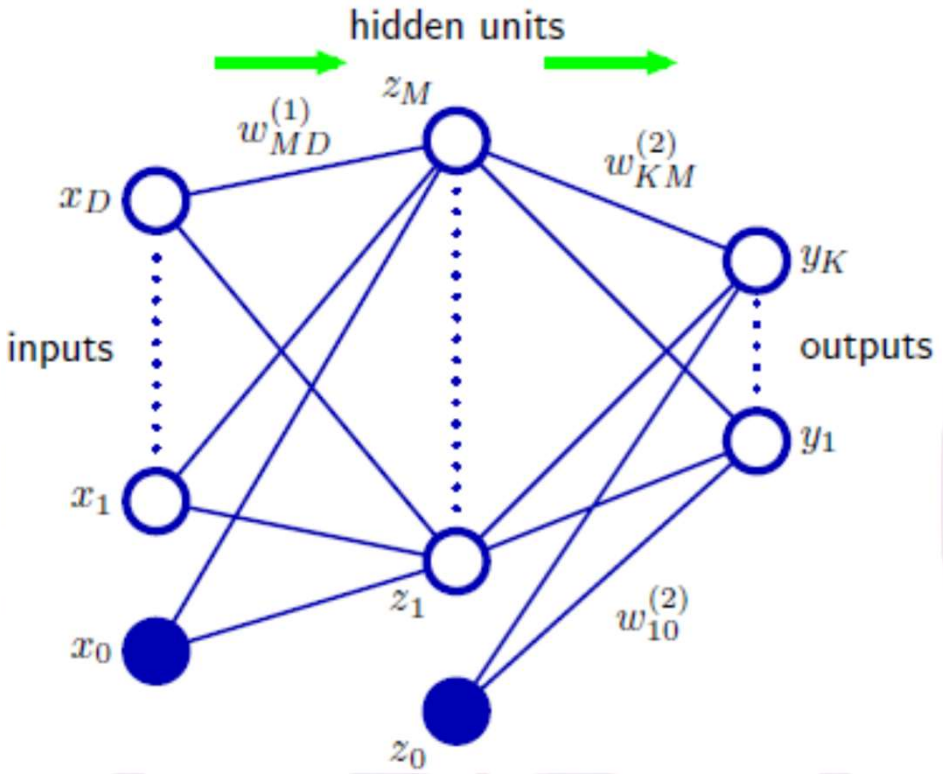
- 对于批处理方法，误差函数导数

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}$$

- 注意：对于使用不同激活函数的单元，只需关注激活函数的具体形式即可。

示例

- 两层网络拓扑结构



- 平方和误差函数
- 输出单元为线性激活函数 $y_k = a_k$

示例

- 隐含单元采用 logistic sigmoid 激活函数

$$h(a) \equiv \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

  - 这个函数的导数形式简单

$$h'(a) = 1 - h(a)^2$$

- 输入矢量 $\mathbf{x}_n$ 的标准平方和误差函数

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

y_k 是输出单元 k 的激活值，t_k 是对应的目标值。

示例

- 对训练样本 x_i，首先完成向前传播

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = \tanh\left(a_j\right)$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

- 对每个输出单元，计算误差

$$\delta_k = y_k - t_k$$

- 反向传播这些 δ_k，得到隐含单元的误差

$$\delta_j = \left(1 - z_j^2\right) \sum_{k=1}^{K} w_{kj} \delta_k$$

示例

- 最后，对第一层和第二层权值的导数为

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

反向传播的效率

- 假设网络中权值和偏差总数为 W，对给定输入的误差函数计算需要$O(W)$次操作。
  - 权值数目远远大于单元数目，激活函数的计算只占很小的开销
  - 求和计算需要的一次乘法和一次加法是总的计算代价 $O(W)$
- 另一种计算误差函数导数的反向传播方法是使用有限差分

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n\left(w_{ji}+\varepsilon\right)-E_n\left(w_{ji}\right)}{\varepsilon}+O(\varepsilon)$$

其中 $\varepsilon \ll 1$。

- 使用对称中心差分可以明显改进有限差分精度（两倍计算量）

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n\left(w_{ji}+\varepsilon\right)-E_n\left(w_{ji}-\varepsilon\right)}{2\varepsilon}+O\left(\varepsilon^2\right)$$

反向传播的效率

- 数值差分方法的主要问题是最理想的计算复杂度由 $O(W)$ 变成了$O(W^2)$ 。
  - 每次向前传播 $O(W)$ 步
  - 网络中的 W 个权值需要每个单独扰动

- 在实际应用中，由于反向传播算法具有最高的计算精度和数值效率，所以应该使用该算法。
- 对称中心差分方法一般做为检验反向传播算法是否正确实现的工具。

Jacobin矩阵

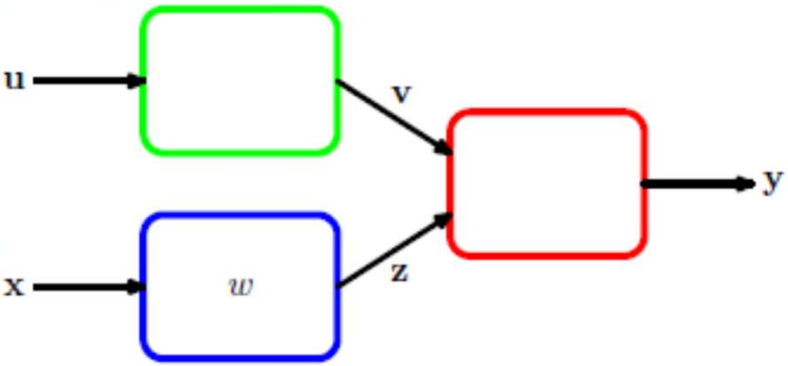
- 反向传播技术也可以用于其它导数计算
- Jacobian矩阵：矩阵元素为网络输出对输入的导数

$$J_{ki} \equiv \frac{\partial y_k}{\partial x_i}$$

计算每个导数时保持所有其它输入不变。

- 由多模块组成系统中，Jacobian矩阵具有重要作用。
  - 每个模块由固定或自适应的、线性或非线性的可微函数组成
  - 针对参数 w 最小化误差函数 E，误差函数的导数为

$$\frac{\partial E}{\partial w} = \sum_{k,j} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w}$$



Jacobian矩阵

- 输入误差 Δx_i 通过已训练网络对输出误差的贡献

$$\Delta y_k \simeq \sum_i \frac{\partial y_k}{\partial x_i} \Delta x_i$$

当 $|\Delta x_i|$ 很小时，上式成立。

- 反向传播计算

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_j w_{ji} \frac{\partial y_k}{\partial a_j}$$

求和项是与单元 i 输出相连的所有单元 j 。

Jacobian矩阵

- 反向传播的递归公式

$$\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l}\frac{\partial a_l}{\partial a_j}$$

$$= h'\left(a_j\right)\sum_l w_{lj}\frac{\partial y_k}{\partial a_l}$$

其中，求和项是与单元 j 输出相连的所有单元 l 。

- 从输出单元开始
  - 如果输出单元使用 sigmoid 激活函数，则

$$\partial y_k / \partial a_j = \delta_{kj}\sigma'\left(a_j\right)$$

  - 如果使用 softmax 激活函数，则

$$\partial y_k / \partial a_j = \delta_{kj} y_k - y_k y_j$$

Hessian矩阵

- 反向传播也可以计算误差的二阶导数

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}}$$

如果将所有权值和偏差参数 w_i 统一在单个矢量 w 中，上述二阶偏导数就是 Hessian 矩阵的元素 H_{ij}，$i, j \in \{1, \ldots, W\}$

- 在神经计算中的作用
  - 网络训练使用的非线性优化算法中误差曲面的二阶性质由其控制
  - 是根据训练数据变化快速调整预训练前馈网络的基础
  - 逆矩阵被用来检测网络中将被“剪枝”的最不重要权值
  - 在贝叶斯网络的 Laplace 近似中起着中心作用

对角化近似

- 引入：当需要使用 Hessian 矩阵的逆矩阵时，其对角化近似方便逆矩阵的计算。
- 对应输入变量 x_n 的 Hessian 矩阵对角线元素

$$\frac{\partial^2 E_n}{\partial w_{ji}^2} = \frac{\partial^2 E_n}{\partial a_j^2} z_i^2$$

考虑公式 $z_j = h(a_j),\ a_j = \Sigma_i w_{ji} z_i$

- 反向传播方程

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'\left(a_j\right)^2 \sum_k \sum_{k'} w_{kj} w_{k'j} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} + h''\left(a_j\right) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}$$

对角化近似

- 忽略二次偏导项中的非对角元素，有

$$\frac{\partial^2 E_n}{\partial a_j^2} = h'\left(a_j\right)^2 \sum_k w_{kj}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''\left(a_j\right) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}$$

- 注意：对角化近似的计算复杂度是 $O(W)$，而整个矩阵的计算复杂度是 $O(W^2)$。

- 由于在实际应用中 Hessian 矩阵是强非对角化的，故对角化近似主要是计算方便，使用时必须小心对待。

外积近似

- 当神经网络用于回归问题时，通常使用平方和误差函数

$$E=\frac{1}{2}\sum_{n=1}^{N}\left(y_n-t_n\right)^2$$

- Hessian矩阵

$$\mathbf{H}=\nabla\nabla E=\sum_{n=1}^{N}\nabla y_n\nabla y_n+\sum_{n=1}^{N}\left(y_n-t_n\right)\nabla\nabla y_n$$

- Levenberg-Marquardt近似或外积近似
  - 因为忽略第二项，只对经过适当训练的网络有效。

$$\mathbf{H}\simeq\sum_{n=1}^{N}\mathbf{b}_n\mathbf{b}_n^{\mathrm{T}}$$

其中 $\mathbf{b}_n=\nabla y_n=\nabla a_n$，只涉及误差一阶导数，时间复杂度$O(W)$

外积近似

- 输出单元 logistic sigmoid 激活函数，交叉熵误差函数的网络，对应的近似为

$$\mathbf{H} \simeq \sum_{n=1}^{N} y_n \left(1 - y_n\right) \mathbf{b}_n \mathbf{b}_n^{\mathrm{T}}$$

  - 输出单元 softmax 激活函数的多类网络，有类似结果。

逆 Hessian 矩阵

- Hessian 矩阵的外积近似

$$\mathbf{H}_N = \sum_{n=1}^{N} \mathbf{b}_n \mathbf{b}_n^{\mathrm{T}}$$

其中 $\mathbf{b}_n \equiv \nabla_{\mathbf{w}} a_n$ 是数据点 n 对输出单元激活值梯度的贡献。

- 假设，使用前 L 数据点获得了 Hessian 矩阵，将第 L+1 数据点的贡献分开表示为

$$\mathbf{H}_{L+1} = \mathbf{H}_L + \mathbf{b}_{L+1} \mathbf{b}_{L+1}^{\mathrm{T}}$$

- Woodbury 等式

$$\left(\mathbf{M} + \mathbf{v}\mathbf{v}^{\mathrm{T}}\right)^{-1} = \mathbf{M}^{-1} - \frac{\left(\mathbf{M}^{-1}\mathbf{v}\right)\left(\mathbf{v}^{\mathrm{T}}\mathbf{M}^{-1}\right)}{1 + \mathbf{v}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{v}}$$

逆 Hessian 矩阵

- 使用 $\mathbf{H}_L$ 替换 M，使用 $\mathbf{b}_{L+1}$ 替换 v，则有

$$\mathbf{H}_{L+1}^{-1} = \mathbf{H}_L^{-1} - \frac{\mathbf{H}_L^{-1}\mathbf{b}_{L+1}\mathbf{b}_{L+1}^T\mathbf{H}_L^{-1}}{1+\mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_L^{-1}\mathbf{b}_{L+1}}$$

  数据点依次被“吸收”，直到 $L+1=N$。

- 初始化矩阵 $\mathbf{H}_0 = \alpha\mathbf{I}$，其中 α 是一个较小的数，算法实际找到的是 $\mathbf{H}+\alpha\mathbf{I}$ 的逆矩阵。
  - 结果对 α 的取值不是特别敏感。

有限差分

- 可以利用有限差分求二阶导数，其精度受数值精度的约束。
- 扰动每一对可能的权值，有

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} = \frac{1}{4\varepsilon^2}\Big\{E\left(w_{ji}+\varepsilon, w_{lk}+\varepsilon\right) - E\left(w_{ji}+\varepsilon, w_{lk}-\varepsilon\right) - E\left(w_{ji}-\varepsilon, w_{lk}+\varepsilon\right) + E\left(w_{ji}-\varepsilon, w_{lk}-\varepsilon\right)\Big\} + O\left(\varepsilon^2\right)$$

  - 使用对称中心差分公式，残差 $O(\varepsilon^2)$ 而不是 $O(\varepsilon)$。
  - 算法复杂度为 $O(W^3)$
- 更有效的数值差分算法是对误差函数一阶导数做中心差分，而一阶导数计算使用反向传播：计算复杂度 $O(W^2)$

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} = \frac{1}{2\varepsilon}\left\{\frac{\partial E}{\partial w_{ji}}\left(w_{lk}+\varepsilon\right) - \frac{\partial E}{\partial w_{ji}}\left(w_{lk}-\varepsilon\right)\right\} + O\left(\varepsilon^2\right)$$

准确计算 Hessian 矩阵

- 假设两层前馈网络
  - 索引 i, i' 表示输入，索引 j, j' 表示隐含，索引 k, k' 表示输出
  - 定义

$$\delta_k = \frac{\partial E_n}{\partial a_k}, \quad M_{kk'} \equiv \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}}$$

    其中 E_n 表示数据点 n 对误差的贡献。
  - 第二层两个权值

$$\frac{\partial^2 E_n}{\partial w_{kj}^{(2)} \partial w_{k'j'}^{(2)}} = z_j z_{j'} M_{kk'}$$

  - 第一层两个权值

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{j'i'}^{(1)}} = x_i x_{i'} h''\left(a_{j'}\right) I_{jj'} \sum_k w_{kj'}^{(2)} \delta_k + x_i x_{i'} h'\left(a_{j'}\right) h'\left(a_j\right) \sum_k \sum_{k'} w_{k'j'}^{(2)} w_{kj}^{(2)} M_{kk'}$$

准确计算 Hessian 矩阵

- 在每层一个权值

$$\frac{\partial^2 E_n}{\partial w_{ji}^{(1)} \partial w_{kj'}^{(2)}} = x_i h'\left(a_{j'}\right)\left\{\delta_k I_{jj'} + z_j \sum_{k'} w_{k'j'}^{(2)} H_{kk'}\right\}$$

  其中 $I_{jj'}$ 是单位矩阵的 j, j' 元素。

- 注意：如果其中一个或者两个权值是偏差项，则通过令激活值为 1 来简单地获得对应的表达式。

快速 Hessian 矩阵乘法

- Hessian 矩阵 H 与矢量 v 的乘积 $\mathbf{v}^{\mathrm{T}}\mathbf{H}$
  - 注意到

$$\mathbf{v}^{\mathrm{T}}\mathbf{H} = \mathbf{v}^{\mathrm{T}}\nabla\left(\nabla E\right)$$

    其中∇ 表示权空间的梯度操作。
  - 相等于作用在正向传播和反向传播方程的导数操作 $\mathbf{v}^{\mathrm{T}}\nabla$，记作 $\mathcal{R}\{\cdot\}$ 。
  - 利用一般微积分规则，有

$$\mathcal{R}\{\mathbf{w}\} = \mathbf{v}$$

- 假设两层网络，线性输出函数与平方和误差函数
  - 向前传播方程

$$a_j = \sum_i w_{ji} x_i$$

$$z_j = h\left(a_j\right)$$

$$y_k = \sum_j w_{kj} x_j$$

快速 Hessian 矩阵乘法

- 将操作符$\mathcal{R}\{\cdot\}$ 作用在向前传播方程上，有

$$\mathcal{R}\{a_j\} = \sum_i v_{ji} x_i$$

$$\mathcal{R}\{z_j\} = h'(a_j)\mathcal{R}\{a_j\}$$

$$\mathcal{R}\{y_k\} = \sum_j w_{kj}\mathcal{R}\{z_j\} + \sum_j v_{kj} z_j$$

其中，v_{ji} 是矢量 v 对应权值 w_{ji} 的元素。

- 考虑平方和误差函数，标准反向传播表达式如下：

$$\delta_k = y_k - t_k$$

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

快速 Hessian 矩阵乘法

- 将操作符$\mathcal{R}\{\cdot\}$ 作用在反向传播方程上，有

$$\mathcal{R}\{\delta_k\} = \mathcal{R}\{y_k\}$$

$$\mathcal{R}\{\delta_j\} = h''(a_j)\mathcal{R}\{a_j\}\sum_k w_{kj}\delta_k + h'(a_j)\sum_k v_{kj}\delta_k + h'(a_j)\sum_k w_{kj}\mathcal{R}\{\delta_k\}$$

- 误差一阶导数

$$\frac{\partial E}{\partial w_{kj}} = \delta_k z_j$$

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i$$

快速 Hessian 矩阵乘法

- 将操作符 $\mathcal{R}\{\cdot\}$ 作用在误差一阶导数上，有

$$\mathcal{R}\left\{\frac{\partial E}{\partial w_{kj}}\right\}=\mathcal{R}\{\delta_k\}z_j+\delta_k\mathcal{R}\{z_j\} \tag{1}$$

$$\mathcal{R}\left\{\frac{\partial E}{\partial w_{ji}}\right\}=x_i\mathcal{R}\{\delta_j\} \tag{2}$$

- 算法实现需要隐含单元附加变量 $\mathcal{R}\{a_j\},\mathcal{R}\{z_j\},\mathcal{R}\{\delta_j\}$ ，输出单元附加变量 $\mathcal{R}\{\delta_k\},\mathcal{R}\{y_k\}$。
- 对每个输入模式，使用上述结果，$\mathbf{v}^{\mathrm{T}}\mathbf{H}$ 的元素由公式 (1) 和 (2) 给出。
- 计算 $\mathbf{v}^{\mathrm{T}}\mathbf{H}$ 的方程与标准向前和反向传播方程非常相似，可以使用已有算法稍加修改来实现计算。

❖ 计算整个 Hessian 矩阵：通过将矢量 v 选择为一系列单位矢量(0, 0,…,1,…,0)的形式，每个挑选 Hessian 矩阵的一列来计算。

诚信 创新 实践