智能系统原理与开发
第11章 前馈神经网络

李斌

复旦大学 计算机科学技术学院

总览

1 神经元与多层感知器

2 反向传播算法

3 深度前馈神经网络

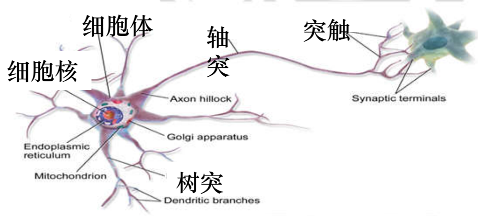
神经元与多层感知器

- 神经元模型
- 多层感知器
- 多层感知机参数学习

生物神经元

- 神经科学表明
  - 一个神经元可以接受从多个神经元传来的脉冲输入
  - 当输入累计到一定程度，这个神经元将发射一个新的脉冲
  - 传入之后的神经元，同时自身电位急剧衰减

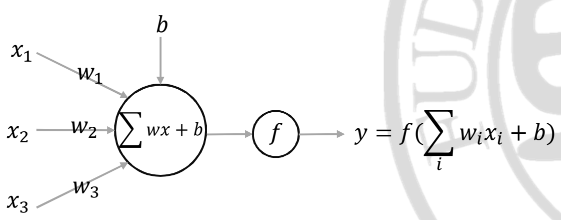
→



维基百科

人工神经元

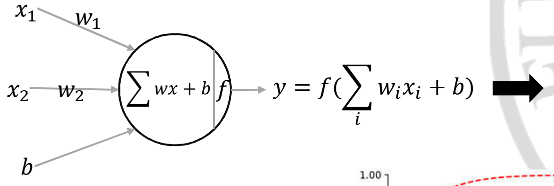
- 神经元模型是人工神经网络的基本成分，它从神经科学中获得灵感，但工作方式与大脑神经元并不相同



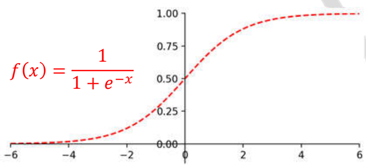
神经元对输入进行简单的加权求和，再通过非线性激活函数f得到神经元的输出

神经元模型

- 用感知器可以实现一些简单的运算
  - 令f为$Sigmoid$函数，即$f(x)=\frac{1}{1+e^{-x}}$，$w_1=20$，$w_2=20$，$b=-10$
  - 此时感知器实现了或运算

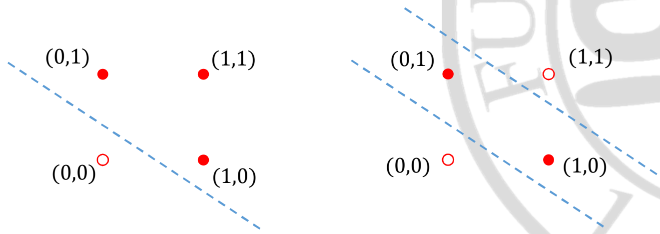


x_1	x_2	f
0	0	≈ 0
0	1	1
1	0	1
1	1	1



神经元模型

- 感知器的局限性
  - 感知器只能处理线性可分的问题，如上述的或运算，但是异或问题是线性不可分的



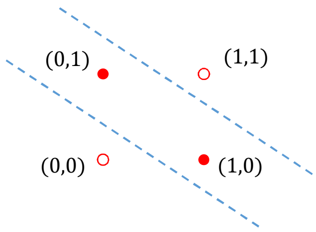
或　　异或

神经元与多层感知器

- 神经元模型
- **多层感知器**
- 多层感知机参数学习

多层感知器

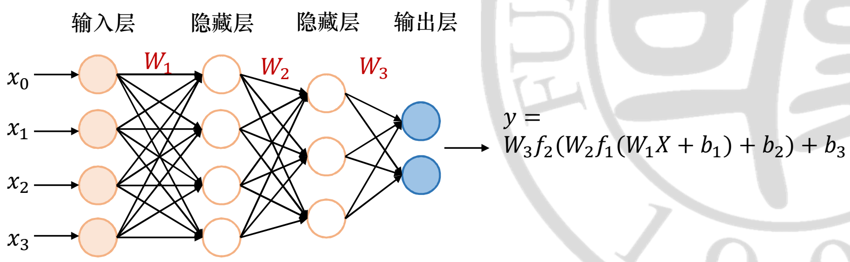
- 1969年，Marvin Minsky 和 Seymour Papert 发表《Perceptrons: An introduction to computational geometry》一书
- 该书从数学的角度证明了单层神经网络无法处理简单的“异或”问题，要处理异或问题，需要接下来介绍的多层感知器



异或

多层感知器

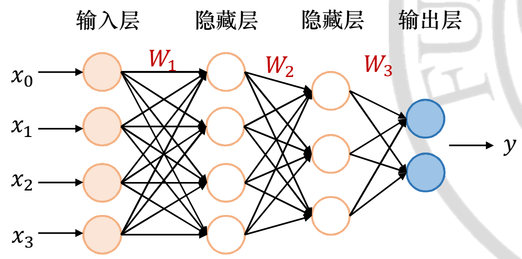
- 多层感知器（前馈神经网络）
  - 多层感知器由输入层、隐藏层、输出层构成
  - 相邻两层神经元完全两两相连
  - 数据从输入层到输出层单向传播



含有两层隐含层的多层感知机

多层感知器

- 上述多层感知器的前馈过程：
  - 第一层隐藏层拿到输入进行第一次变换
  - 第二层隐藏层拿第一层隐藏层的输出进行第二次更抽象的变换
  - 最后一层隐藏层使用高层抽象信息进行最终的决策



含有两层隐含层的多层感知机

隐藏单元

- 隐藏层中非线性激活函数对网络的表现起着关键作用，它在神经网络中引入了非线性变换，若没有这样的非线性变换，网络的输出始终是输入的线性组合。

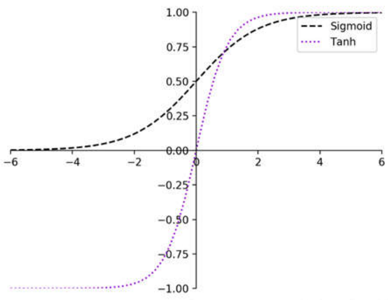
- 一个良好的激活函数需要满足的性质：
  - 连续并可导的非线性函数（可部分点上不可导）
  - 要尽可能简单，方便计算
  - 导数的值域要在一个合适的区间内，否则会影响训练的效率和稳定性

激活函数

■ sigmoid与双曲正切函数

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

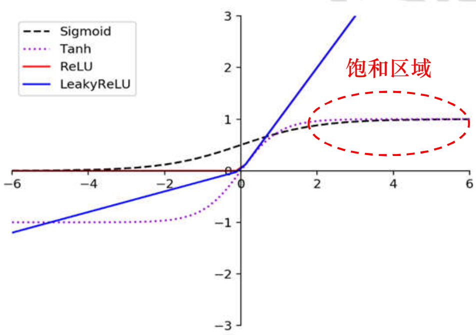
$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$= \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



激活函数

■ sigmoid与双曲正切函数

- □ sigmoid和tanh的优势是全程可导
- □ sigmoid和tanh的梯度在饱和区域接近于0，很容易造成梯度消失的问题，减缓收敛速度

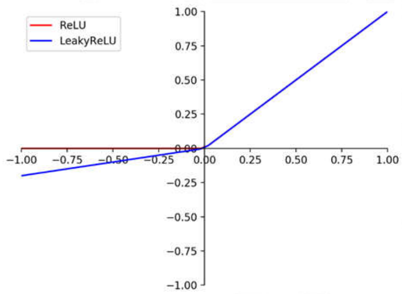


激活函数

- 整流线性单元（rectified linear unit）
  - 整流线性单元$ReLU$的特点在于它在一半的定义域上输出为零
  - 这使得神经元的梯度是分段线性的，易于优化，并且$x > 0$时不存在饱和区域；为了防止$x < 0$时的硬饱和问题，后又衍生出$LeakyReLU$。

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

$$LeakyReLU(x) = \begin{cases} x & if\ x > 0 \\ \lambda x & if\ x \le 0 \end{cases}$$



输出层单元

- 对于不同的任务需求，通常有几种常见的输出层选择：
  - 线性单元——用于回归问题

$$y = Wh + b$$

  - sigmoid 单元——用于二分类问题

$$y = \sigma(Wh + b)$$

  - softmax 单元——用于多分类问题

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}, \qquad Z = Wh + b$$

神经元与多层感知器

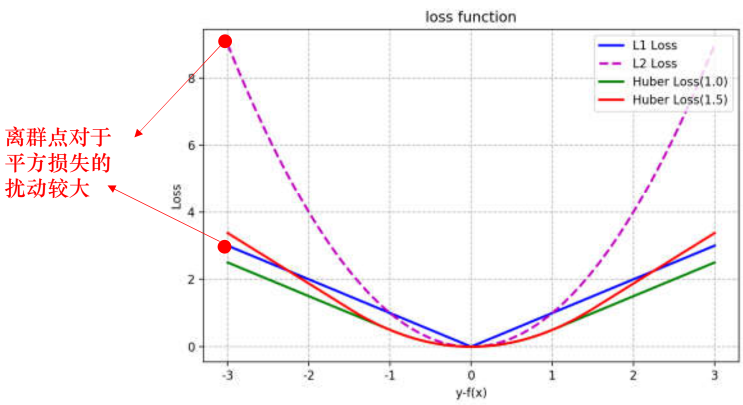
- 神经元模型
- 多层感知器
- **多层感知器参数学习**

损失函数

- 损失函数是基于梯度的参数学习的目标函数，损失函数越小，说明网络对训练数据拟合的越好。
- 回归问题常用损失函数：
  - 绝对值损失函数（MAE）： $L(y,f(x)) = |y-f(x)|$ ，收敛速度慢，但是不容易受离群点影响
  - 平方损失函数（MSE）：$L(y,f(x)) = \left(y-f(x)\right)^2$，收敛速度快，但是容易受离群点影响
  - Huber损失函数：
    $$L(y,f(x)) = \begin{cases} 0.5(y-f(x))^2, & |y-f(x)| \le \delta \\ \delta|y-f(x)| - 0.5\delta^2, & |y-f(x)| > \delta \end{cases},$$
    综合了MSE和MAE的优点。

损失函数

■ 三种回归损失函数



损失函数

■ 分类问题常用损失函数：

- □ Hinge损失函数： Hinge Loss的目的就是让正确类的评分越大越好，错误类的评分越小越好

  二分类：$Loss = max(0,1 - y \cdot f(x))$

  多分类：$\text{Loss} = \sum_{i \neq t} max(0, y_i - y_t + 1)$，$y_t$ 表示正确类对应的评分

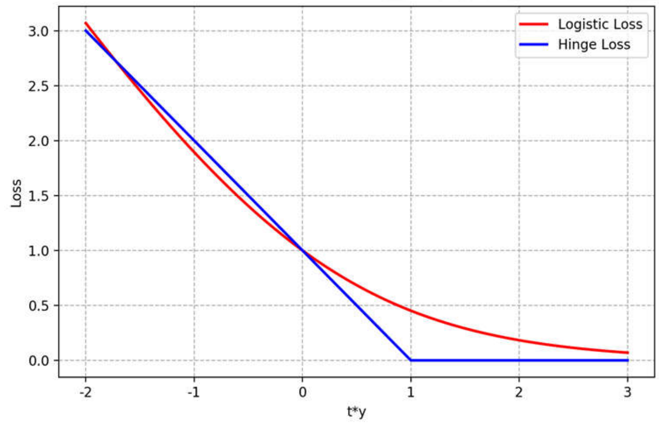
- □ 交叉熵损失函数：交叉熵损失求导简单（仅与正确类别的概率有关），并且梯度不受激活函数影响，收敛较快

  二分类：$\text{Loss} = \sum_i -t_i log(y_i) = -t_1 log(y_1) - (1 - t_1) log(1 - y_1)$,
  等价于二分类的Logistic损失

  多分类：$\text{Loss} = \sum_i -t_i log(y_i)$，$y_i$是经过softmax之后第$i$类的评分；

损失函数

■ 二分类损失函数：



梯度下降算法

■ 有了损失函数，我们需要通过梯度下降算法来更新模型参数。

□ 简化版的梯度下降算法流程

参数：学习率ϵ、参数θ

While 未收敛 *do*：

  计算损失函数对参数的梯度g

  计算更新：$\Delta\theta = -\epsilon \odot g$

  更新参数：$\theta = \theta + \Delta\theta$

End

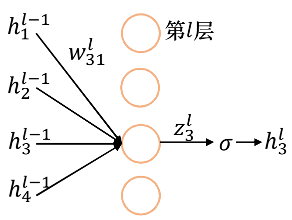
■ 此时关键问题是计算损失函数对于模型参数的梯度。需要用到反向传播算法。

反向传播算法

- 梯度推导
- 自动微分

梯度推导

- 1986年，Hinton在Nature上提出反向传播算法
- 设w_{jk}^l表示第$l-1$层第k个神经元与第l层第j个神经元连接的权重；b_j^l表示第l层第j个神经元的偏置； $\sigma(\cdot)$表示激活函数。于是对于单个样本有以下三式：



$$Loss：C = \frac{1}{2}\left\|y - h^L\right\|^2$$

第l层第j个神经元的输入：

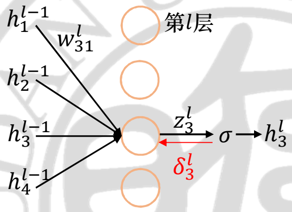
$$z_j^l = \sum_k w_{jk}^l h_k^{l-1} + b_j^l$$

第l层第j个神经元的输出：

$$h_j^l = \sigma(z_j^l) = \sigma(\sum_k w_{jk}^l h_k^{l-1} + b_j^l)$$

——Learning internal representations by back-propagating errors. Nature, 323(99): 533-536, 1986.

梯度推导



- 反向传播的目标是计算损失函数对于参数的偏导$\frac{\partial C}{\partial w_{jk}^l}$与$\frac{\partial C}{\partial b_j^l}$，为此我们先定义第$l$层第$j$个神经元的误差为：

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

- 由链式法则最后一层的误差为：

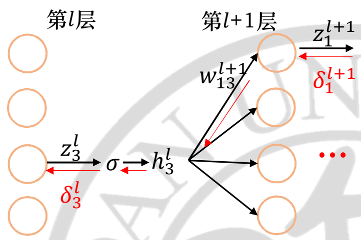
$$\boldsymbol{\delta}^L = \frac{\partial C}{\partial \boldsymbol{z}^L} = \frac{\partial C}{\partial [z_1^L, z_2^L \cdots]^T} = \left[\frac{\partial C}{\partial z_1^L}, \frac{\partial C}{\partial z_2^L}\cdots\right]^T$$

$$= \left[\frac{\partial C}{\partial h_1^L}\cdot\frac{\partial h_1^L}{\partial z_1^L}, \frac{\partial C}{\partial h_2^L}\cdot\frac{\partial h_2^L}{\partial z_2^L}\cdots\right]^T = \frac{\partial C}{\partial \boldsymbol{h}^L}\odot\frac{\partial \boldsymbol{h}^L}{\partial \boldsymbol{z}^L}$$

$$= \nabla_{\boldsymbol{h}} C \odot \sigma'(\boldsymbol{z}^L)$$

$\odot$为向量点对点乘法

梯度推导



- 下面计算中间层产生的误差：

每个神经元的误差为：$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_m \frac{\partial C}{\partial z_m^{l+1}}\cdot\frac{\partial z_m^{l+1}}{\partial h_j^l}\cdot\frac{\partial h_j^l}{\partial z_j^l}$

$$= \sum_m \delta_m^{l+1}\cdot\frac{\partial(w_{mj}^{l+1}h_j^l + b_m^{l+1})}{\partial h_j^l}\cdot\sigma'(z_j^l)$$

$$= \sum_m \delta_m^{l+1}\cdot w_{mj}^{l+1}\cdot\sigma'(z_j^l)$$

于是l层整体为：$\boldsymbol{\delta}^l = \left[\left((\boldsymbol{w}_1^{l+1})^T\boldsymbol{\delta}^{l+1}\right)\cdot\sigma'(z_1^l),\cdots\right]^T$

$$= \left((\boldsymbol{w}^{l+1})^T\boldsymbol{\delta}^{l+1}\right)\odot\ \sigma'(\boldsymbol{z}^l)$$

梯度推导

■ 下面计算对权重w_{jk}^l 的偏导：

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l}$$
$$= \delta_j^l \cdot \frac{\partial (w_{jk}^l h_k^{l-1} + b_j^l)}{\partial w_{jk}^l}$$
$$= h_k^{l-1} \delta_j^l$$

■ 同样可以推到出对偏置b_j^l的偏导：

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \cdot \frac{\partial \left(w_{jk}^l h_k^{l-1} + b_j^l\right)}{\partial b_j^l} = \delta_j^l$$

梯度推导

■ 计算出梯度后，使用前述的梯度下降算法更新参数：

以一个样本为例：

$$\boldsymbol{w}^l = \boldsymbol{w}^l - \epsilon \boldsymbol{\delta}^l \cdot \left(\boldsymbol{h}^{l-1}\right)^T$$
$$\boldsymbol{b}^l = \boldsymbol{b}^l - \epsilon \boldsymbol{\delta}^l$$

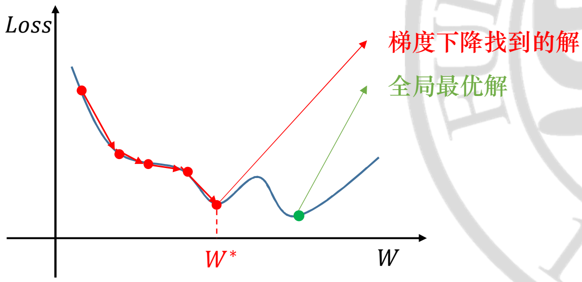
当按批训练时，假设批大小为N：

$$\boldsymbol{w}^l = \boldsymbol{w}^l - \epsilon \frac{1}{N} \sum_x \boldsymbol{\delta}^{x,l} \cdot \left(\boldsymbol{h}^{x,l-1}\right)^T$$

$$\boldsymbol{b}^l = \boldsymbol{b}^l - \epsilon \frac{1}{N} \sum_x \boldsymbol{\delta}^{x,l}$$

局部最优解

- 神经网络多层的代价函数是非凸的，所以前面提到的梯度下降很难找到全局最优解
- 但是找到的局部最优解的效果已经可以接受



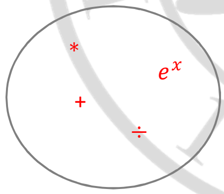
反向传播算法

- 梯度推导
- 自动微分

自动微分

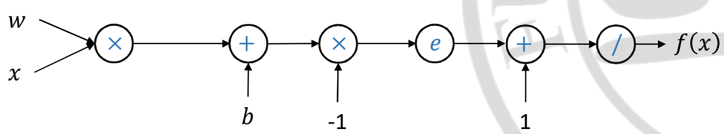
- 前面讲了用反向传播推导梯度的方法，在具体计算梯度时，手动求导很容易出错，即便使用计算机也只能是编写具体的求导函数，函数变化，就得重新编程；
- 因此，实际梯度计算的实现中，使用自动微分的思想，它是一种利用链式法则来自动计算一个复合函数梯度的方法。
- 自动微分基于这样一个观察：所有数值计算归根结底是一系列有限的可微算子的组合，比如下面的$f(x)$可拆分成右侧四种算符。

$$f(x) = \frac{1}{1 + e^{-(wx+b)}}$$



自动微分

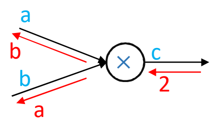
- 自动微分涉及一个核心概念：计算图。流行的深度学习框架TensorFlow、Pytorch都是使用计算图机制来计算梯度。
  - 计算图中每个节点为一个运算操作
  - 例：$f(x) = \frac{1}{1 + e^{-(wx+b)}}$的计算图为：



梯度计算

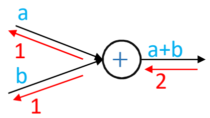
- 计算图中梯度的传播只和当前节点有关

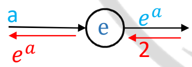
例：乘法节点



因为当前节点相当于$a*b=c$，所以对a方向的导数为b，对b方向的导数为a，与后面传过来的梯度2无关。

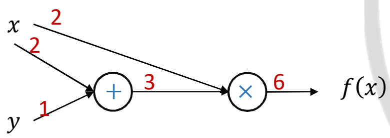
同理：加法节点、指数节点如下





自动微分

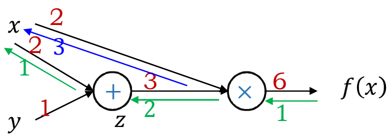
- 利用计算图求反向传播的梯度——简单的例子
  - 第一步进行前向运算
  - $f(x)=x*(x+y)$ ，令$x=2, y=1$



自动微分

■ 利用计算图求反向传播的梯度——简单的例子

- 第二步进行反向求导
- 令$f = x * (x + y) = x * z$



求导路线一:

$\frac{\partial f}{\partial f} = 1$

$\frac{\partial f}{\partial z} = x = 2$

$\frac{\partial z}{\partial x} = 1$

求导路线二:

$\frac{\partial f}{\partial f} = 1$

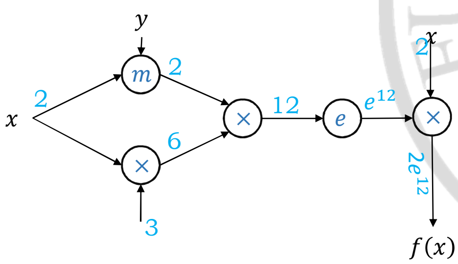
$\frac{\partial f}{\partial x} = z = 3$

$$\Rightarrow \frac{\partial f}{\partial x} = 1 * 2 * 1 + 1 * 3 = 5$$

梯度计算

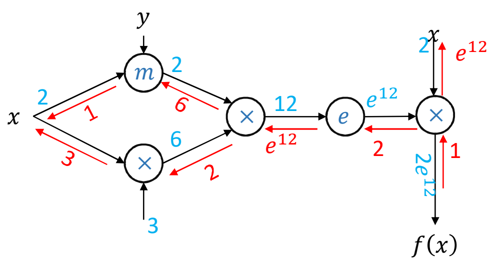
■ 利用计算图求反向传播的梯度——再一个例子

- 第一步进行前向计算　　设输入$x = 2, y = 1$
- 令$f = x * e^{(x*3)*max(x,y)}$



梯度计算

- 利用计算图求反向传播的梯度——再一个例子
  - 第二步进行反向求导　　设输入$x = 2, y = 1$
  - 令$f = x * e^{(x*3)*max(x,y)}$



$$\frac{\partial f}{\partial x} = 1 * 6 * e^{12} * 2 * 1 + 3 * 2 * e^{12} * 2 * 1 + e^{12} * 1$$

$$= 25e^{12}$$

深度前馈神经网络

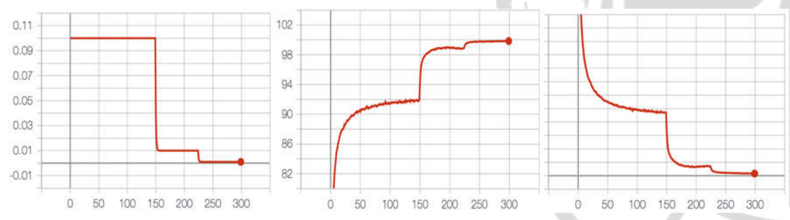
- 训练细节与技巧
- 前馈神经网络的问题
- 前馈神经网络发展回顾

权重初始化

- 梯度下降对参数的初始值较为敏感 ，好的初始值可以带来更快更好的收敛效果
- 在训练前馈神经网络时需要将所有的权重值初始化为小随机数。当初始化权重太大时，计算的梯度会累积多个大的权重，导致“梯度爆炸”
- 通常使用从高斯或均匀分布中的随机抽取的值来初始化模型的权重

学习率的设置策略

- 模型训练初期需要较大的学习率来让它快速找到*Loss*较小的区域，之后则需要恰当时机减小学习率，来让*Loss*继续下降



学习率　　训练集准确率　　训练*Loss*

深层网络的训练

- 为了提高网络的表达能力，可以使用多隐层的神经网络
- 但是多隐层网络带来新的问题，就是网络难以收敛，这可以从多种角度进行解决：
  - 使用批归一化（Batch Normalization）
  - 使用比随机梯度下降法更强的优化算法（比如Adam）
- 下面介绍这两种方法。

深层网络的训练

- 批归一化算法：在激活函数前对中间层特征进行归一化。设网络中间某一层在一个Batch中得到m个特征向量$x_{1\ldots m}$，批归一化首先对特征数据进行归一化：

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu)^2$$

$$\Longrightarrow \quad \widehat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

归一化后，为了弥补丢失的表达能力，需再进行一次仿射变换：

$$y_i = \gamma\widehat{x_i} + \beta$$　　γ、β为可学习参数

深层网络的训练

- Adam算法：

参数：学习率ϵ、衰减率ρ_1、 ρ_2、参数θ、常数$\delta = 10^{-6}$

初始化一阶和二阶矩动量项 $s = 0, r = 0$

初始化时间步 $t = 0$

$While$ 未收敛 do：

计算梯度g，令$t = t + 1$

更新有偏一阶矩估计：$s = \rho_1 s + (1 - \rho_1) g$

更新有偏二阶矩估计：$r = \rho_2 r + (1 - \rho_2) g \odot g$

修正偏差$\hat{s} = \frac{s}{1-\rho_1^t}$、$\hat{r} = \frac{r}{1-\rho_2^t}$

计算更新：$\Delta\theta = -\frac{\epsilon \hat{s}}{\delta + \sqrt{\hat{r}}}$

更新参数：$\theta = \theta + \Delta\theta$

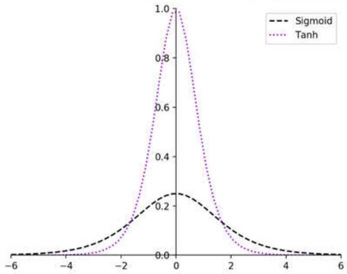
End

深度前馈神经网络

- 训练细节与技巧
- 前馈神经网络的问题
- 前馈神经网络发展回顾

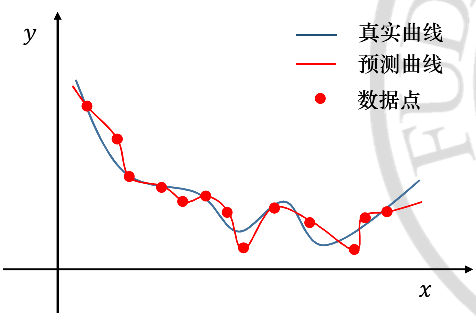
前馈神经网络的问题

- 梯度消失
  - sigmoid和tanh激活函数的导数取值如图，反向传播时多个导数相乘，导致梯度接近于0
- 解决方法
  - 使用ReLU系列激活函数



前馈神经网络的问题

- 过拟合问题
  - 预测出的曲线拟合了训练集所有的点，但是偏离了真实的数据分布



前馈神经网络的问题

- 缓解过拟合问题
  - 早停止策略
    - 将数据划分为训练集和验证集，验证集用来估计误差
    - 若训练集误差降低，验证集误差升高，则停止训练
  - 正则化策略:
    - 基本思想是在损失函数中增加描述模型复杂度的部分
    - 例如权重的平方和，以此来限制模型的学习能力，缓解过拟合
  - 增大加训练数据:
    - 增加训练数据可以有效防止过拟合，因为大数据量可以抵消模型拟合能力过强的问题

深度前馈神经网络

- 训练细节与技巧
- 前馈神经网络的问题
- 前馈神经网络发展回顾

前馈神经网络发展回顾

- 几个关键节点

1958年，就职于Cornell航空实验室的Frank Rosenblatt发明了的一种称为感知器（Perceptron）的人工神经网络。

1974 年，哈佛大学Paul Werbos在其博士论文中发明了影响深远的著名BP神经网络学习算法。但没有引起重视。

1986年，David E. Rumelhart, Geoffrey E. Hinton 和 Ronald J. Williams重新报道这一方法，BP神经网络学习算法才受到重视。

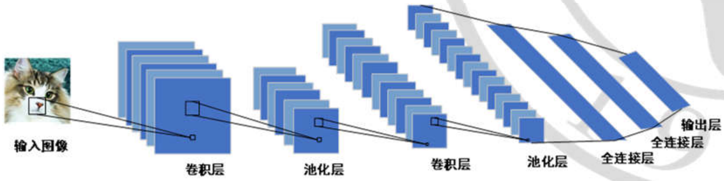
前馈神经网络发展回顾

- 20 世纪 80 年代以来前馈神经网络的核心思想没有发生重大变化，但其性能有着显著地提高
  - 足够大的数据集提高了网络的泛化能力
  - 计算机计算性能的提高
  - 交叉熵和整流线性单元的使用也大大提高了神经网络的能力

其它类型神经网络

■ 卷积神经网络

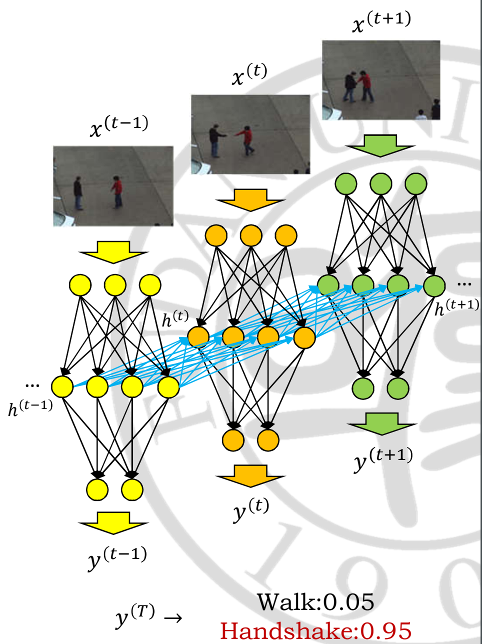
- ☐ 卷积网络是专门用于处理网格化数据（图片等）的网络，在图像分类中首次取得重大成功。
- ☐ 不同于全连接神经网络，卷积网络的卷积核每次只计算局部的特征，因此卷积网络能更多地关注图片的局部特性。
- ☐ 卷积网络大大减少了全连接神经网络的参数量。



其它类型神经网络

■ 循环神经网络

- ☐ 是一类用于处理序列数据的神经网络，允许网络出现环形结构，t步的输出不仅与t步的输入有关还与t-1步的网络状态有关
- ☐ 如右图是处理视频数据的网络，最终的输出结合了每个时间步的信息。



Thanks

Email: libin@fudan.edu.cn