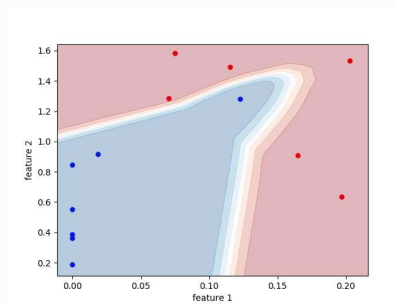


# 媒体与认知

## Media and Cognition



V 1.0 2022.3.10

清华大学电子工程系

王生进 彭良瑞 李亚利

Email: {wgsgj, penglr, liyali13}@tsinghua.edu.cn

# 第3讲 神经网络

- 一、神经网络基本原理
- 二、神经网络的训练方法
- 三、编程示例

# 问题

- 前馈神经网络为何具有较强的非线性表示能力？
- 多层神经网络训练中的误差反向传播过程是怎样的？

# 一、神经网络基本原理

- 人工神经网络(Artificial Neural Network, ANN)是一种机器学习方法
  - ◆ 深层神经网络 (Deep Neural Network, DNN) 或称 深度学习 (Deep Learning) 的技术兴起得益于：
    - 大数据
    - 并行计算能力

# 神经网络与深度学习发展历史

## ➤ 神经网络数学模型的开创性工作

- ◆ 1943年 McCulloch 和 Pitts 提出的神经元模型

## ➤ 感知机

- ◆ 1958年由Rosenblatt提出

## ➤ 自1960年逐步发展的误差反向传播算法

- ◆ 来自控制等领域的启发

## ➤ 经典的神经网络误差反向传播算法

- ◆ 1986年由Rumelhart, Hinton 和 Williams提出

## ➤ 具有一个隐含层的神经网络的非线性函数拟合能力

- ◆ 1989年由Hecht-Nielsen提出

# 发展历史

- 卷积神经网络 (Convolutional Neural Network, CNN)
  - ◆ 1980年由 Fukushima提出
  - ◆ 1998年由 LeCun等改进，局部感受野、权值共享
- 循环神经网络 (Recurrent Neural Network, RNN)
  - ◆ Hopfield神经网络：1983年 Hopfield 在网络中引入反馈
  - ◆ 长短时记忆(LSTM)网络：1997年 Hochreiter等引入门控机制
- 深度置信网络 (Deep Belief Net, DBN)
  - ◆ 2006年由Hinton等提出，并引入神经网络预训练方法
- 自动编码器 (Auto Encoder)
  - ◆ 2006年由Hinton等提出，基于神经网络的编码及解码重构
- 生成对抗网络 (Generative Adversarial Network, GAN)
  - ◆ 2014年由Goodfellow提出，两个神经网络(生成器和鉴别器)相互博弈
- 基于自注意力机制的Transformer
  - ◆ 2017年由Vaswani等提出，利用数据相关性作为自注意力系数进行加权

# 人工神经元 (Neuron)

► 模拟神经元的计算单元——人工神经元包括两个计算步骤：

◆ 线性加权求和，相当于两个向量 $\mathbf{x}$ ,  $\mathbf{w}$ 求内积

$$z = \sum_{i=1}^d w_i x_i + b$$
$$= \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x} = \mathbf{x}^T \mathbf{w}$$

$\mathbf{x} = (x_0, x_1, \dots, x_d)^T$  为输入数据 $(x_1, \dots, x_d)^T$ 的增广向量,  
 $x_0=1$

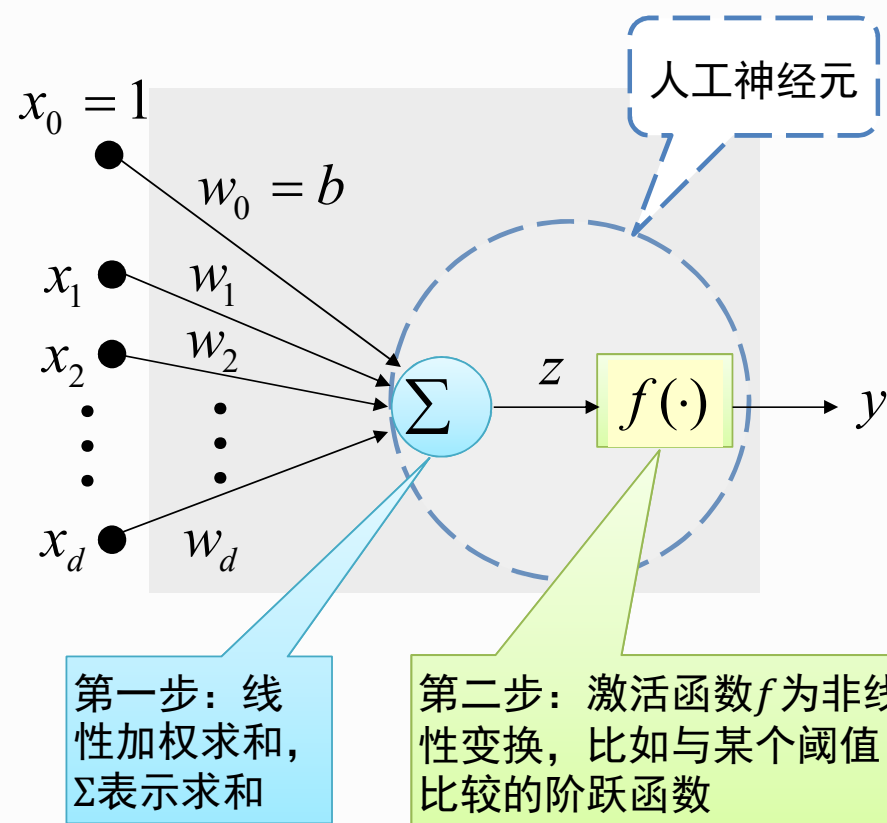
$\mathbf{w} = (w_0, w_1, \dots, w_d)^T$  为权值向量,  
 $w_0 = b$ ,  $b$ 为偏置量

◆ 激活函数

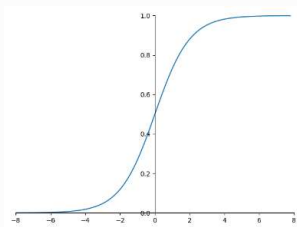
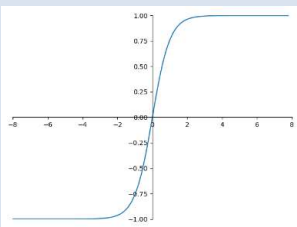
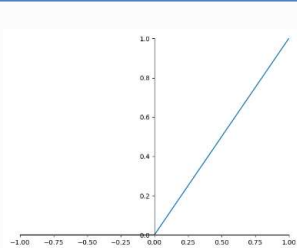
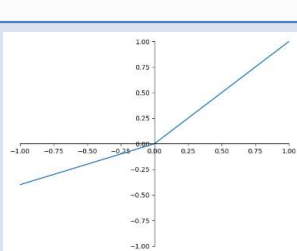
删减信息

• 对数据进行非线性变换

$$y = f(z)$$



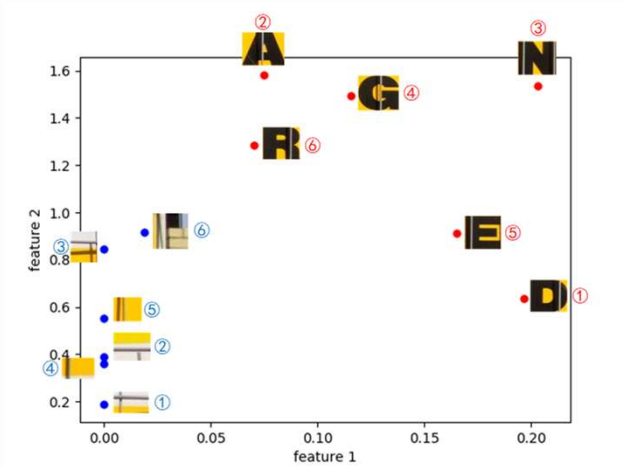
# 激活函数(Activation Function)

Sigmoid	$\sigma=f(z)=\frac{1}{1+e^{-z}}$	$f'(z)=\frac{e^{-z}}{(1+e^{-z})^2}$ $=\sigma(1-\sigma)$		<ol style="list-style-type: none"> <li>1. 非零中心;</li> <li>2. 饱和区域<b>梯度消失</b>;</li> </ol>
Tanh双曲正切	$\delta=f(z)=2\sigma(2z)-1$ $=\frac{e^z - e^{-z}}{e^z + e^{-z}}$	$f'(z)=1-\left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)^2$ $=1-\delta^2$		<ol style="list-style-type: none"> <li>1. 零中心;</li> <li>2. 饱和区域梯度消失;</li> <li>3. 应用于循环神经网络(RNN)</li> </ol>
ReLU (Rectified Linear Unit)	$f(z)=\max(0,z)$	$f'(z)=\begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$		<ol style="list-style-type: none"> <li>1. <b>正值部分梯度有效传递，解决梯度消失问题</b>;</li> <li>2. 非零中心;</li> <li>3. 负值信息截断产生 dead neurons</li> </ol>
Leaky ReLU /PReLU(Parametric ReLU)	$f(z)=\begin{cases} z & z > 0 \\ \alpha z & z \leq 0 \end{cases}$	$f'(z)=\begin{cases} 1 & z > 0 \\ \alpha & z < 0 \end{cases}$		部分解决负值信息截断问题



# 基于逻辑回归的线性分类

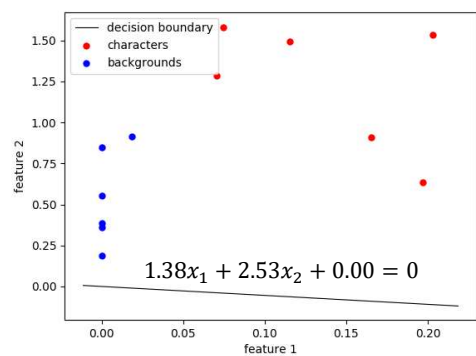
## 逻辑回归模型训练过程中决策面变化情况



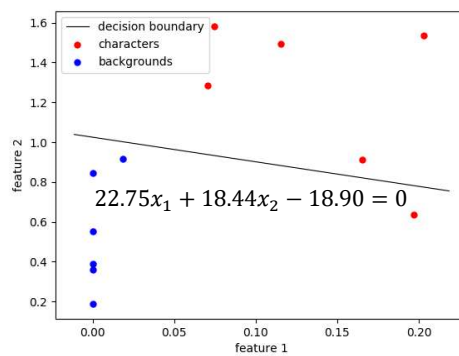
(a) 特性向量数据分布

各样本的特征向量取值

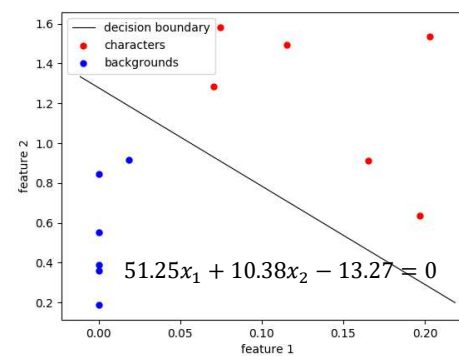
	样本1	样本2	样本3	样本4	样本5	样本6
字符	$\begin{bmatrix} 0.197 \\ 0.635 \end{bmatrix}$	$\begin{bmatrix} 0.075 \\ 1.581 \end{bmatrix}$	$\begin{bmatrix} 0.203 \\ 1.534 \end{bmatrix}$	$\begin{bmatrix} 0.116 \\ 1.492 \end{bmatrix}$	$\begin{bmatrix} 0.165 \\ 0.910 \end{bmatrix}$	$\begin{bmatrix} 0.070 \\ 1.286 \end{bmatrix}$
背景	$\begin{bmatrix} 0.000 \\ 0.190 \end{bmatrix}$	$\begin{bmatrix} 0.000 \\ 0.388 \end{bmatrix}$	$\begin{bmatrix} 0.000 \\ 0.847 \end{bmatrix}$	$\begin{bmatrix} 0.000 \\ 0.360 \end{bmatrix}$	$\begin{bmatrix} 0.000 \\ 0.553 \end{bmatrix}$	$\begin{bmatrix} 0.019 \\ 0.915 \end{bmatrix}$



(b) 随机初始化模型



(c) 训练轮数 = 10



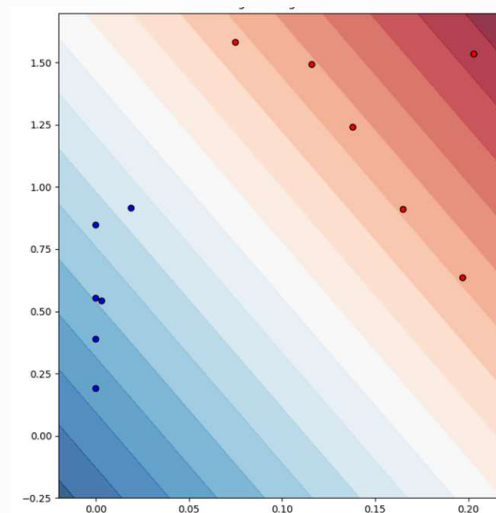
(d) 训练轮数 = 100

# 训练过程中各样本属于各类的概率

不同训练阶段，各个样本属于字符或背景的概率，被错分的样本用红色标出

样本ID	随机初始化		训练轮数 = 10		训练轮数 = 100	
	分类为字符的概率	分类为背景的概率	分类为字符的概率	分类为背景的概率	分类为字符的概率	分类为背景的概率
字符①	0.87	0.13	0.06	0.94	0.97	0.03
字符②	0.98	0.02	1.00	0.00	1.00	0.00
字符③	0.98	0.02	1.00	0.00	1.00	0.00
字符④	0.98	0.02	1.00	0.00	1.00	0.00
字符⑤	0.93	0.07	0.84	0.16	0.99	0.01
字符⑥	0.97	0.03	1.00	0.00	0.98	0.02
背景①	0.62	0.38	0.00	1.00	0.00	1.00
背景②	0.73	0.27	0.00	1.00	0.00	1.00
背景③	0.89	0.11	0.04	0.96	0.01	0.99
背景④	0.71	0.29	0.00	1.00	0.00	1.00
背景⑤	0.80	0.20	0.00	1.00	0.00	1.00
背景⑥	0.91	0.09	0.17	0.83	0.06	0.94

# 基于单个神经元的线性分类的本质



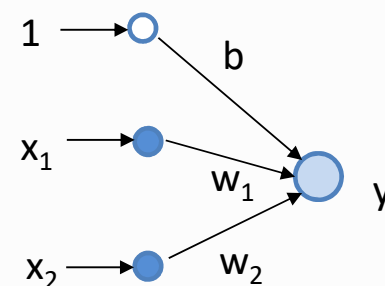
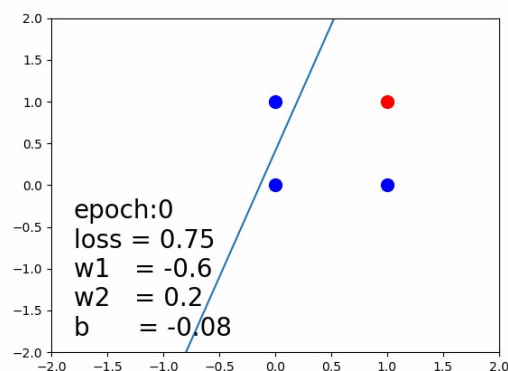
逻辑回归

- 逻辑回归的分类预测概率与样本点离决策平面的距离相关
- 逻辑回归的决策等高线均为直线

# 基于感知机的线性模型

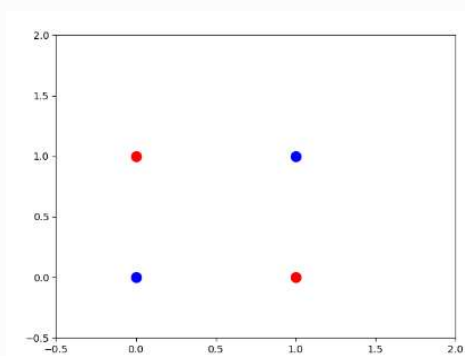
## ➤ 感知机求解 逻辑与(AND)问题

$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1



## ➤ 感知机是线性分类模型，不能求解线性不可分问题，如 逻辑异或(XOR)问题

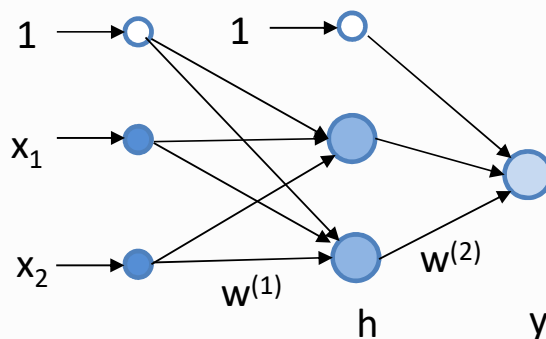
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0



# 从线性分类到非线性分类

## ► 求解异或问题的思路

- ◆ 采用具有一个隐含层(hidden layer)的多层感知机



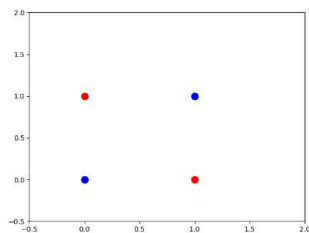
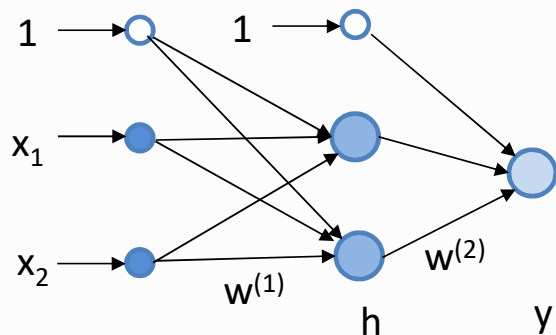
# 求解异或问题

## ➤ 网络设计过程:

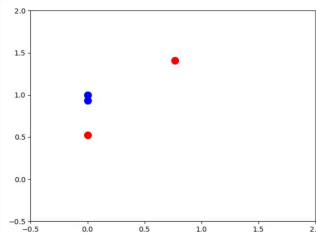
输入增广数据:  $X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

$x_0$   
 $x_1$   
 $x_2$

4个数据点



(a) 原始数据: 线性不可分



(b) 经过隐含层之后:  
线性可分

隐含层系数:  $w^{(1)} = \begin{bmatrix} -0.78 & 0.78 & 0.78 \\ 0.52 & 0.42 & 0.48 \end{bmatrix}^T$

隐含层输出:  $\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \text{ReLU}((w^{(1)})^T x)$

$$= \begin{bmatrix} 0 & 0 & 0 & 0.78 \\ 0.52 & 1.0 & 0.94 & 1.42 \end{bmatrix}$$

输出层增广输入:  $h = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0.78 \\ 0.52 & 1.0 & 0.94 & 1.42 \end{bmatrix}$

输出层系数:  $w^{(2)} = [0.23 \quad -1.03 \quad 0.51]^T$

计算线性层输出:  $z = (w^{(2)})^T h$

$$= [0.4952 \quad 0.74 \quad 0.7094 \quad 0.1508]$$

取阈值  $th$  为 0.5, 输出为:

$y = th(z) = [0 \quad 1 \quad 1 \quad 0]$

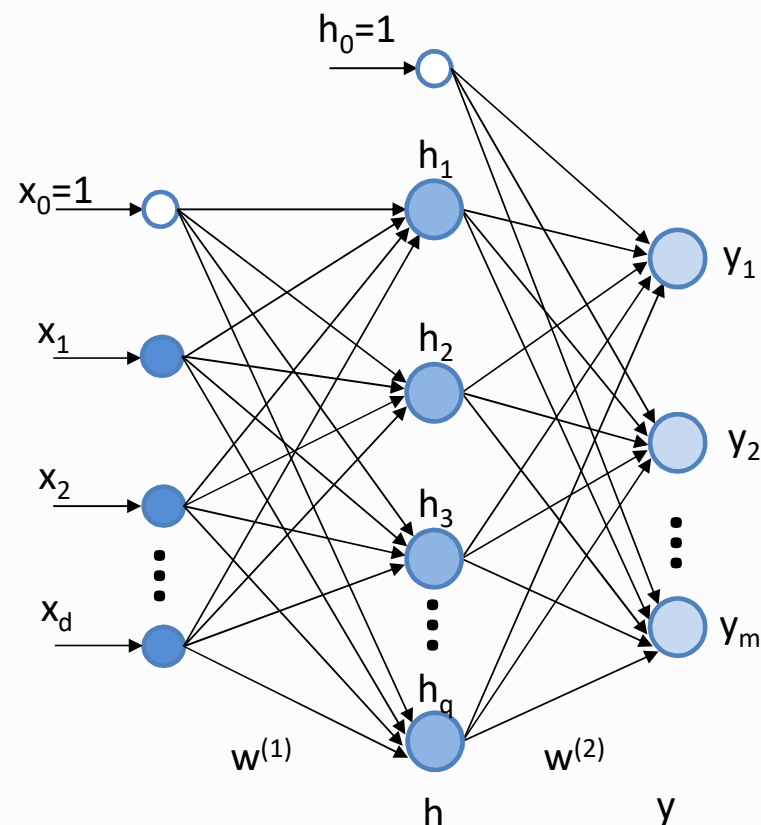
4个数据点判决结果

# 多层感知机

➤ 在输入层和输出层之间增加一个或多个隐含层，每层有多个神经元节点，构成**多层感知机** (Multi-Layer Perceptron, MLP)，即**前馈神经网络**

◆ 每一层的神经元节点与前一层各节点均有线性加权连接，称为：

- **全连接层** (fully connected layer, fc)
- 或 **线性层** (linear layer)
  - 此时非线性激活函数作为后续处理步骤
- 或 **密集层** (dense layer)



# 前馈神经网络

➤ 前馈神经网络计算过程是前向的：

◆ 数据 $x$ 通过用函数 $f$ 定义的中间计算过程，输出 $y = f(x)$

➤ 前馈神经网络可用多个函数复合表示，相当于有向无环图结构

例如由三个函数 $f^{(1)}, f^{(2)}, f^{(3)}$ 复合形成：

$$f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right)$$

◆ 训练过程中，使用输入样本 $x$ 的网络输出预测值 $y = f(x)$ 去估计真值 $y^* = f^*(x)$



# 架构设计

➤ 架构（architecture）是指网络的整体结构

- ◆ 前馈神经网络包括若干层，每一层有多个计算节点，每一层的输出作为下一层的输入，例如：

$$\text{第一层: } \mathbf{h}^{(1)} = f^{(1)}((\mathbf{W}^{(1)})^T \mathbf{x} + b^{(1)})$$

$$\text{第二层: } \mathbf{h}^{(2)} = f^{(2)}((\mathbf{W}^{(2)})^T \mathbf{h}^{(1)} + b^{(2)})$$

- ◆ 模型配置参数包括网络层数和每一层节点数
- ◆ 还需要指定每一层所用的激活函数，以及是否使用偏置量

# 前馈神经网络的拟合能力

## ➤ 拟合逻辑运算

- ◆ 具有一个隐含层、隐含层节点（单元）数目可任意设置、激活函数采用阈值函数的网络，可以实现任意的二值逻辑运算函数

## ➤ 非线性回归任务中拟合非线性函数

- ◆ 具有一个隐含层、隐含层节点（单元）数目可任意设置、激活函数采用S型非线性函数的网络，可以一致逼近紧集上的连续函数或按范数逼近紧集上的平方可积函数

## ➤ 非线性分类任务中拟合决策面

- ◆ 二分类任务中，假设二维特征空间中正类样本位于一个凸多边形内部，具有一个隐含层的网络（输入层、隐含层、输出层）可模拟任一凸多边形或无界的凸区域，即隐含层每个神经元拟合凸多边形的一条边
- ◆ 多层网络可模拟更为复杂的图形

此题未设置答案，请点击右侧设置按钮

神经网络中的非线性表示能力由哪些因素决定？

- ☐ A 非线性激活函数
- ☐ B 隐含层中的节点数
- ☐ C 网络层数

提交

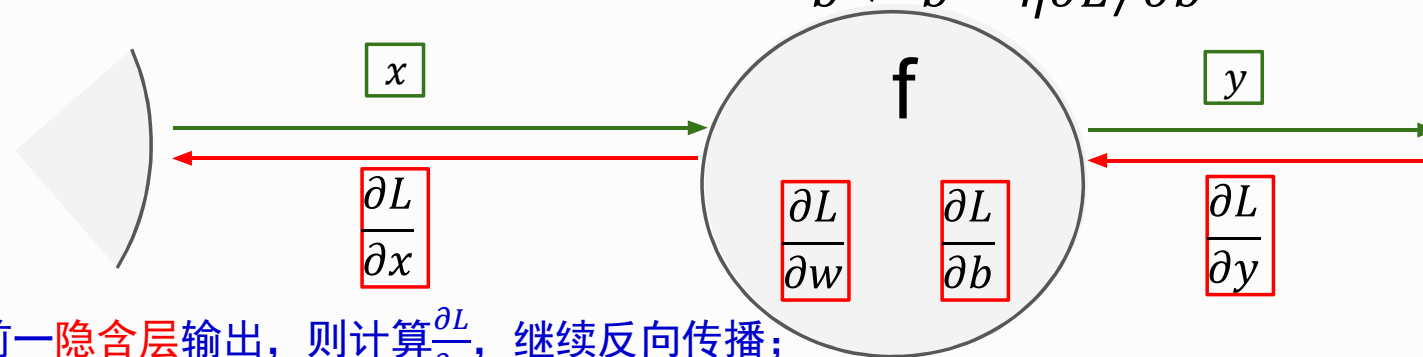
## 二、神经网络的训练方法

- 神经网络相当于一个复杂的复合函数
- 训练时利用误差反向传播算法，采用梯度下降法调整模型参数
  - ◆ 复合函数梯度求导采用链式法则

# 误差反向传播算法(Backward Propagation, BP)

► 训练过程中的误差反向传播，以网络中的最后一层为例：

- ◆ 前向**计算预测值**， $y = f(w^T x + b)$ ，此处 $x$ 为前一层的输出
- ◆ 利用预测值 $y$ 和真值 $y^*$ ，根据目标函数**计算误差**  $L$
- ◆ 误差反向传播，利用**复合函数求导链式法则**（如 $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w}$ ）**计算梯度**  $\frac{\partial L}{\partial y}$ ， $\frac{\partial L}{\partial w}$ ， $\frac{\partial L}{\partial b}$ ，以及 $\frac{\partial L}{\partial x}$
- ◆ **设定学习率 $\eta$** ，利用梯度下降法**更新模型参数**  
 $w \leftarrow w - \eta \frac{\partial L}{\partial w}$   
 $b \leftarrow b - \eta \frac{\partial L}{\partial b}$

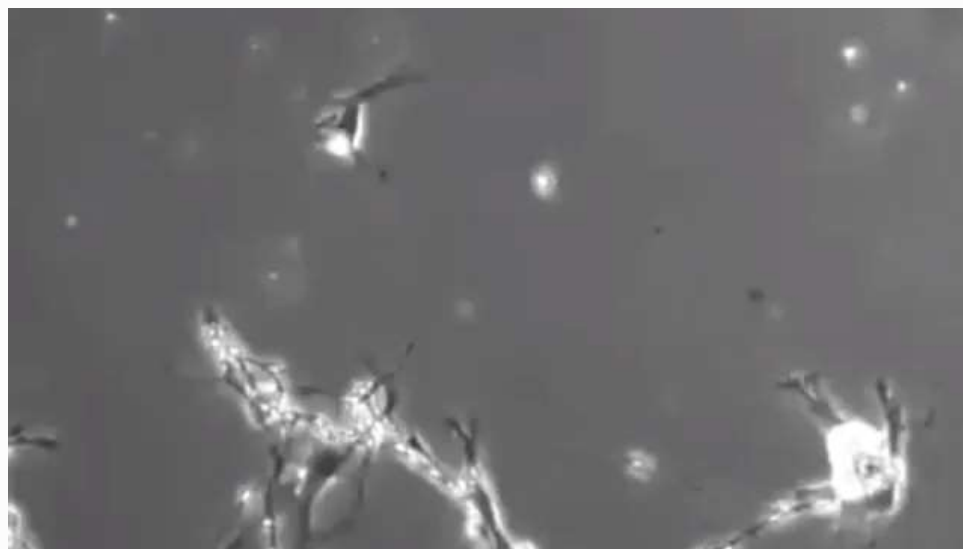


若此处 $x$ 为前一**隐含层**输出，则计算 $\frac{\partial L}{\partial x}$ ，继续反向传播；

若此处 $x$ 为网络的输入层，则不需要进一步计算 $\frac{\partial L}{\partial x}$

# 思考

► 误差反向传播方法是否模拟了人脑中的神经网络学习机制？



神经元连接的形成过程

## 目标函数(一): 用于回归任务的均方误差 MSE

- 考察一个神经网络, 输入 $N$ 个数据 $x^{(i)}, i = 1, \dots, N$  得到网络预测值 $y_i$ 和真值 $y_i^*$ 的误差

$$e_i = y_i - y_i^*$$

- 均方误差(Mean Square Error, MSE)目标函数:

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^N e_i^2$$

- 求解最优化问题:

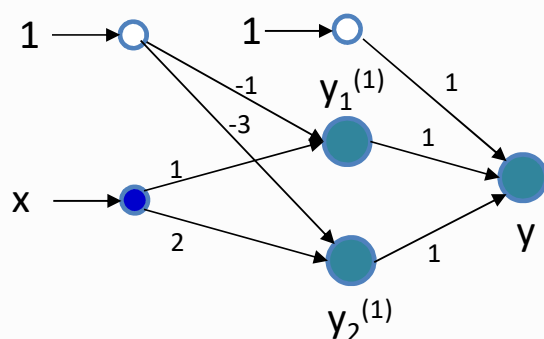
$$\arg \min_{\theta} L(\theta)$$

得到模型参数 $\theta$ , 即网络中各神经元的权值向量 $\mathbf{w}$ 和偏置量 $b$

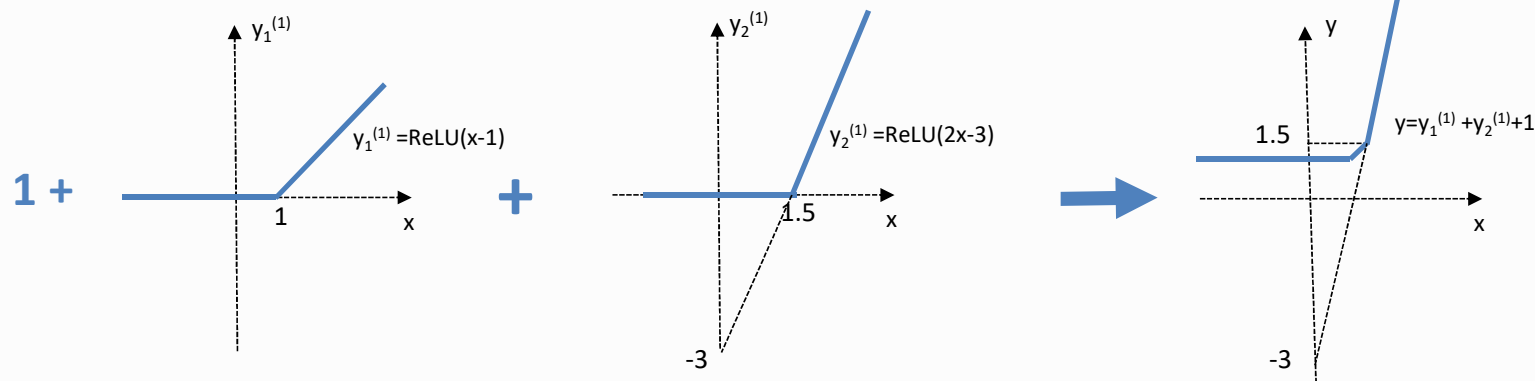
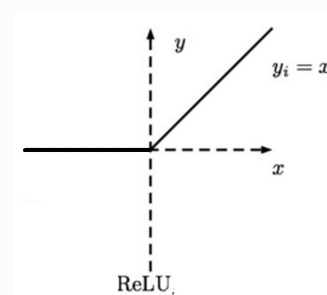
# 基于神经网络的非线性回归

## ► 增加节点及层数、使用非线性激活函数（如ReLU）

具有一个隐含层的神经网络：



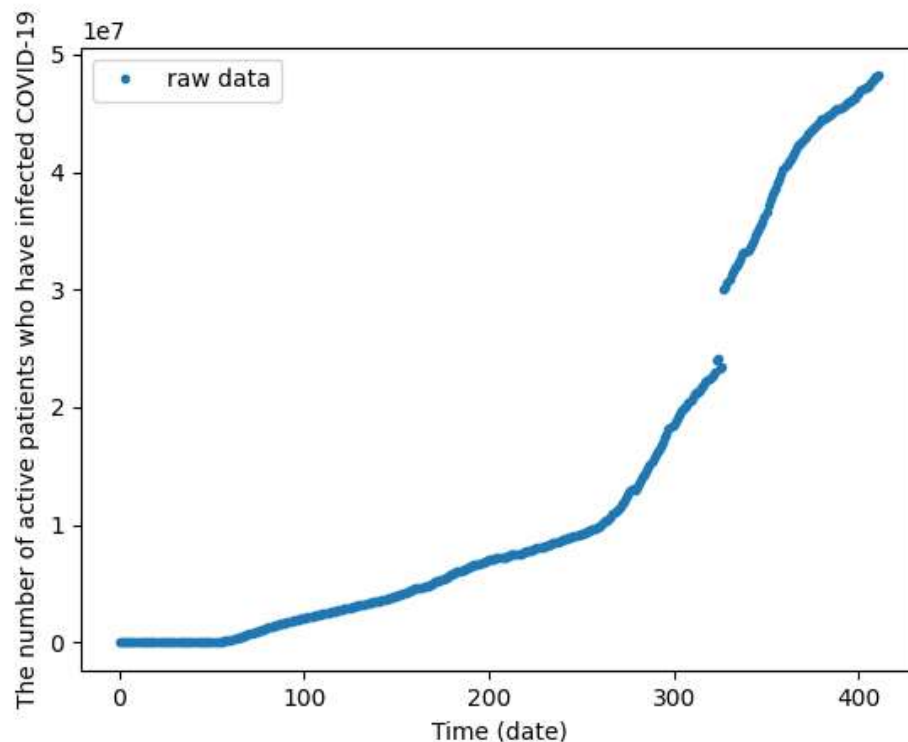
ReLU:  $y = \max(0, x)$ ,





# 求解实际非线性回归问题

➤ 全球新冠肺炎累计感染人数：

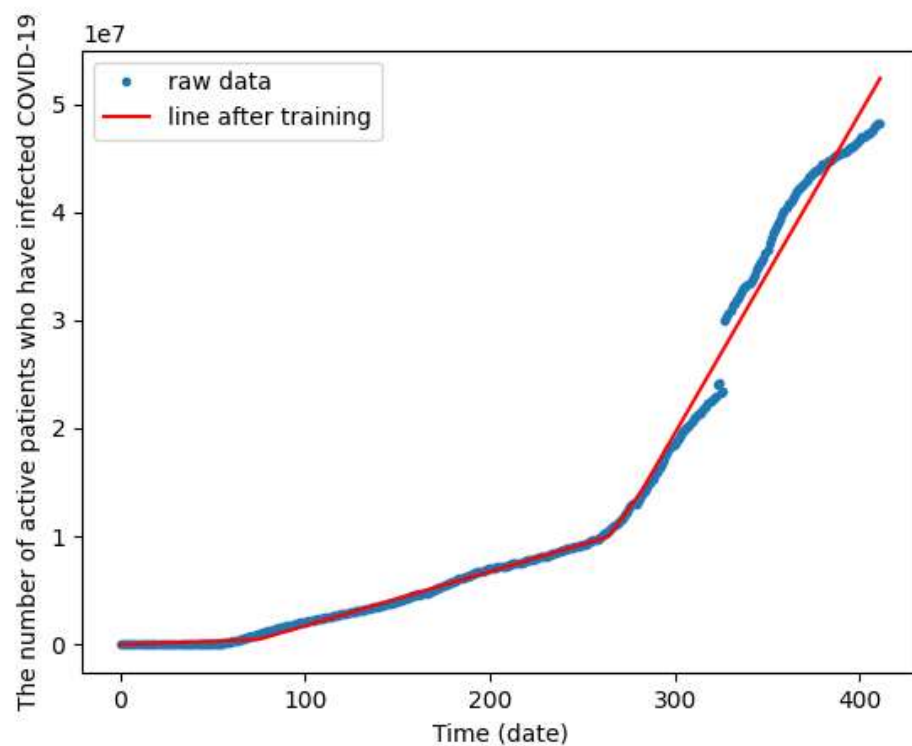


横轴：距2020年1月22日的天数  
纵轴：全球累计感染人数

# 求解实际非线性回归问题

## ► 多层神经网络回归结果：

神经网络各层节点数分别为8、4、4、1



# 用于多类分类任务的Softmax回归

- 用于多类分类任务时，输出层节点数设为模式分类问题的类别总数C相同，即输出层每一节点预测属于对应类别的概率，取值为0与1之间的浮点数
  - ◆ 网络输出层C个节点的输出为一个C维向量
  - ◆ 类别真值(最理想的网络输出)也用C维向量表示，仅有一个元素为1，其余元素为0，称为one-hot编码
    - 例：手写数字识别类别序号取值为0~9，某数字图片样本类别真值为9，采用one-hot编码为 $y^* = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1)^T$
- 输出层节点激活函数采用Softmax函数，适用于多类分类任务
  - ◆ 用指数函数exp处理线性层输出结果，得到大于0的值
  - ◆ 归一化处理得到分类预测的概率分布

$$q_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)}$$

## 目标函数(二)：用于分类任务的交叉熵

- 交叉熵(Cross Entropy Loss)用于度量两个概率分布P和Q之间的差异

$$H(P, Q) = - \sum_{x_i \in X} p(x_i) \log q(x_i)$$

- 类别真值看作一种特殊的概率分布

$$y^* = (y_1, y_2, \dots, y_C)^T, y_i = \begin{cases} 1, & \text{样本类别真值为第} i \text{类} \\ 0, & \text{上述条件不成立} \end{cases}$$

- Softmax输出为属于各类别的概率估计

$$q_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)}$$

- Softmax回归的交叉熵目标函数

若输入样本的类别真值为第*i*类，则 $y_i = 1$ ;  $y_j = 0, j = 1, \dots, C, j \neq i$

$$\begin{aligned} \text{cross entropy loss} &= - \sum_{j=1}^C y_j \log q_j = -y_1 \log q_1 - y_2 \log q_2 \cdots - y_i \log q_i \cdots - y_C \log q_C \\ &= -0 \cdot \log q_1 - 0 \cdot \log q_2 \cdots - 1 \cdot \log q_i \cdots - 0 \cdot \log q_C \\ &= -\log q_i \end{aligned}$$

# 自定义类别标签转换的编程示例

## ➤ 用lambda自定义匿名函数

- ◆ 定义一个具有10个元素的数组，初始化数值为0
- ◆ 然后根据类别y的取值，在对应数组下标处赋值为1

```
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor, Lambda

ds = datasets.FashionMNIST(
    root="./Datasets/FashionMNIST",
    train=True,
    download=True,
    transform=ToTensor(),
    target_transform=Lambda(lambda y: torch.zeros(10, dtype=torch.float).scatter_(0, torch.tensor(y), value=1))
)
```

# 交叉熵

【例】：手写数字识别任务中类别数 $C=10$ 。训练过程中，输入一个数字9的图片样本，样本真值为： $y^* = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1)^T$   
初始模型softmax输出为均匀分布：

$$y = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)^T$$

交叉熵  $loss = -\log(0.1) = -\log\left(\frac{1}{C}\right) = \log(C) = \log(10) = 2.3026$

在一次训练结束后，对于一个数字9的图片样本，模型softmax输出：

$$y = (0.02, 0, 0, 0, 0, 0, 0.03, 0, 0, 0.95)^T$$

交叉熵  $loss = -\log(0.95) = 0.0513$

注：在机器学习常用软件工具包中，交叉熵中的对数计算  $\log$  默认采用自然常数  $e=2.71828$  作为底数。另有  $\log_2(\cdot)$  表示底数为2的对数。

利用对数计算换底公式，则有： $\log(x) = \frac{\log_2(x)}{\log_2(e)}$

```
>>> import torch
>>> -torch.log(torch.tensor(0.1))
tensor(2.3026)
>>> -torch.log(torch.tensor(0.95))
tensor(0.0513)
```

# PyTorch中的交叉熵

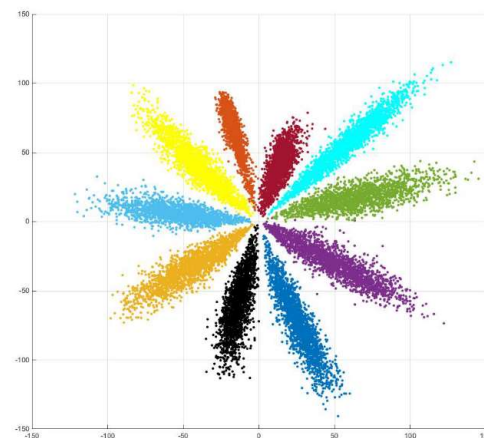
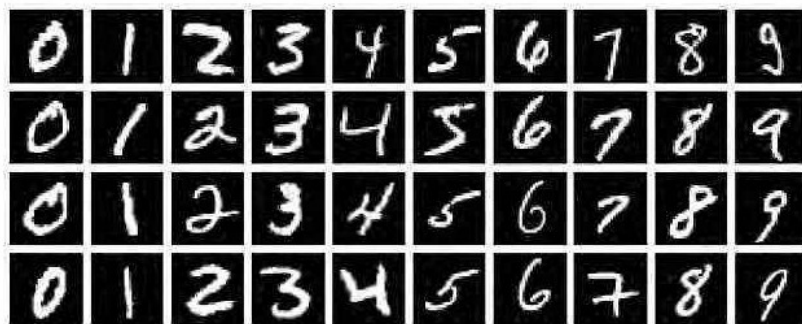
- CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)`
  - ◆ <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>
  - ◆ 包括Softmax归一化处理和交叉熵计算
  - ◆ 类别真值(代码示例中的target变量) 可以用类别序号或one-hot编码形式表示

```
import torch
from torch import nn

# Example of target with class indices
loss = nn.CrossEntropyLoss()
input = torch.randn(3, 5, requires_grad=True)
target = torch.empty(3, dtype=torch.long).random_(5)
output = loss(input, target)
output.backward()
```

# Softmax 及 交叉熵的应用

## ➤ Softmax 及 交叉熵用于多类分类任务



采用 LeNets++ (包含Softmax with Cross Entropy Loss)的MNIST手写数字样本特征分布

Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In Proceedings of European Conference on Computer Vision, 2016, 499–515.



# Softmax函数实现

$$\text{softmax}(z)_i = q_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)}$$

```
import numpy as np
def softmax(x):
    """Compute the softmax of vector x. """
    exps = np.exp(x)
    return exps / np.sum(exps)
```

```
softmax([1,2,3])
```

输出: array([0.09003057, 0.24472847, 0.66524096])

# Softmax函数计算中的问题

```
softmax([1000, 2000, 3000])
```

```
输出: array([ nan, nan, nan])
```

- Numpy中的浮点数的数值范围是有限的，对于float64，最大的数为 $10^{308}$ ，softmax中的指数计算很容易超出这个范围。

# Softmax计算改进方法

$$\begin{aligned}\text{softmax}(z)_i &= q_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)} = \frac{A \exp(z_i)}{\sum_{j=1}^C A \exp(z_j)} \\&= \frac{\exp(\log A) \exp(z_i)}{\sum_{j=1}^C \exp(\log A) \exp(z_j)} = \frac{\exp(z_i + \log A)}{\sum_{j=1}^C \exp(z_j + \log A)} \\&= \frac{\exp(z_i + B)}{\sum_{j=1}^C \exp(z_j + B)}\end{aligned}$$

取  $B = \log A = -\max_{j=1,\dots,C} \{z_j\}$

```
def stablesoftmax(x):  
    """Compute the softmax of vector x in a numerically stable way."""  
    shiftx = x - np.max(x)  
    exps = np.exp(shiftx)  
    return exps / np.sum(exps)  
stablesoftmax([1000, 2000, 3000])  
输出: array([ 0.,  0.,  1.]
```

# Softmax函数的导数

- 对输出层中类别真值对应的节点的 $z_i$ 求导:

$$\frac{\partial q_i}{\partial z_i} = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)} = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)} - \frac{\exp(z_i) \cdot \exp(z_i)}{(\sum_{j=1}^n \exp(z_j))^2} = q_i(1 - q_i)$$

- 对输出层中其他节点的 $z_j$ 求导:

$$\frac{\partial q_i}{\partial z_j} = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)} = -\frac{\exp(z_i) \cdot \exp(z_j)}{(\sum_{j=1}^n \exp(z_j))^2} = -q_i q_j$$

- 上述两种情形的统一形式:

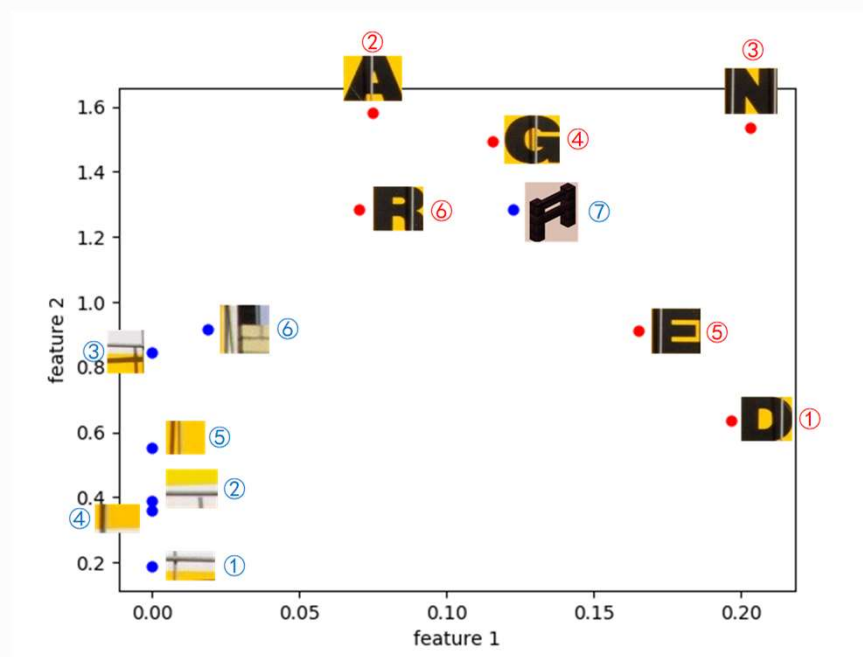
$$\frac{\partial q_i}{\partial z_j} = q_i(\delta_{ij} - q_j) \quad \delta_{ij} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}$$

# 基于神经网络的非线性分类

► 在字符/背景分类案例中，增加一张“背景”样本：



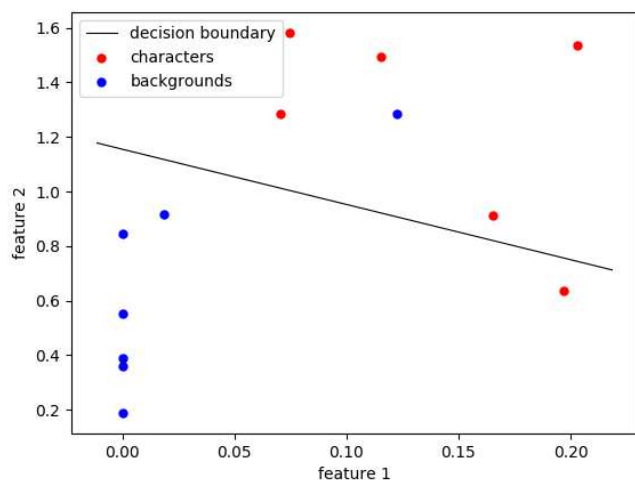
◆ 样本特征空间分布为：



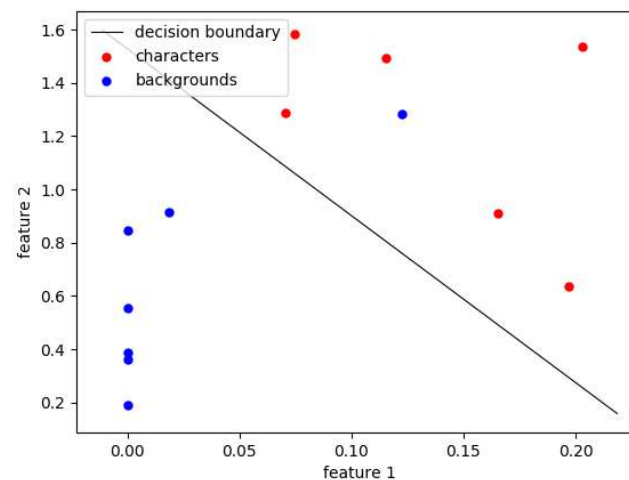
◆ 是否可以用一条直线将字符和背景分开？

# 非线性分类问题

## ► 采用逻辑回归模型作为分类器



(a) 训练轮数 = 100

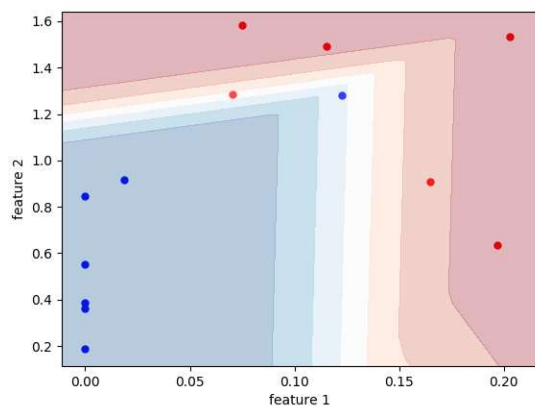


(b) 训练轮数 = 1000

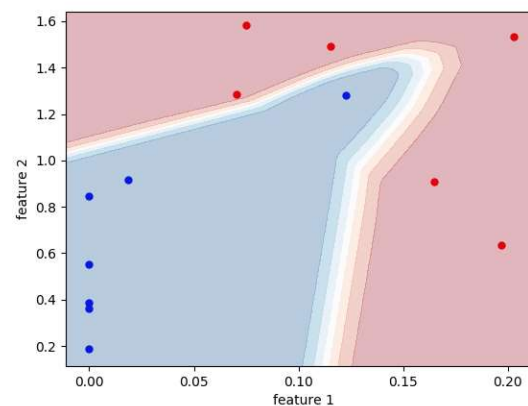
- ◆ 逻辑回归无法将两类线性不可分的样本完全分开
- ◆ 解决方法：采用多层感知机作为分类器

# 非线性分类问题

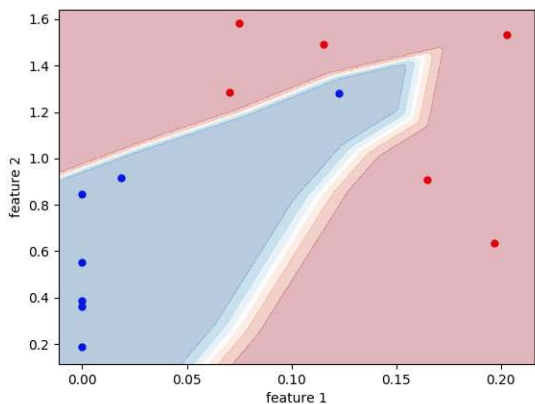
## ► 采用多层感知机作为分类器



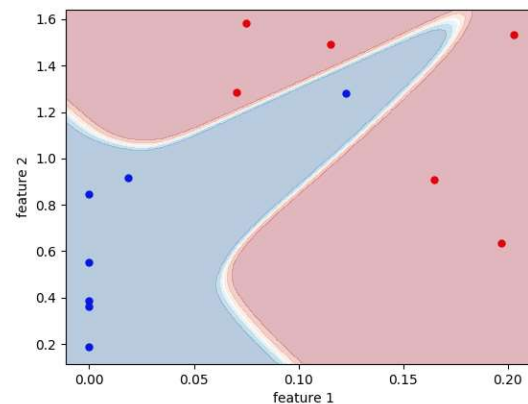
(a) 1层隐含层，节点数8，激活函数ReLU



(b) 1层隐含层，节点数32，激活函数ReLU



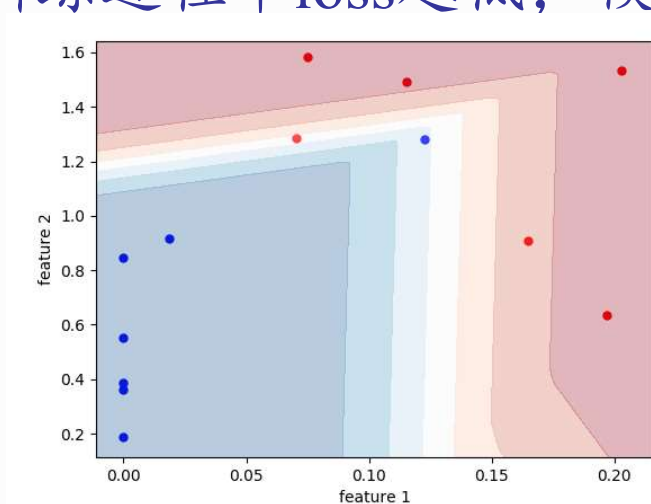
(c) 2层隐含层，节点数均为8，激活函数ReLU



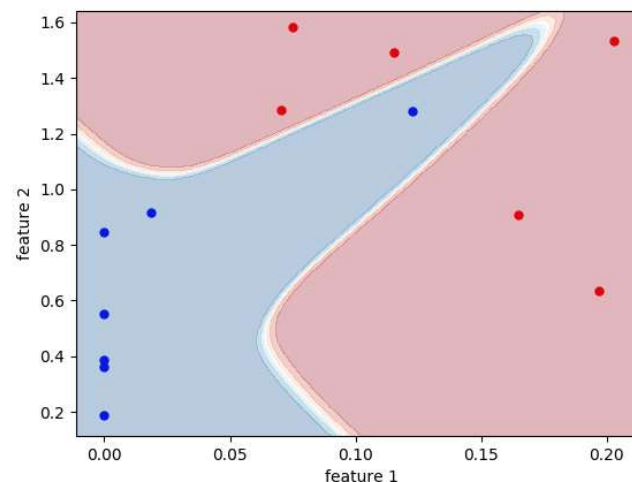
(d) 2层隐含层，节点数均为8，激活函数Tanh

# 非线性分类问题

► 思考：训练过程中loss越低，模型是否一定越好？



loss = 0.125



loss = 0.000

◆ 哪个模型具有较好的泛化能力？

- 可以采用正则化方法防止模型在训练集上过拟合，例如：
  - 在目标函数中引入对模型复杂度的惩罚项
  - 当验证集上误差不再降低就提前终止训练



## 三、编程示例

### ➤ Fashion-MNIST分类任务

#### ◆ 10个类别

- t-shirt (T恤)
- trouser (裤子)
- pullover (套衫)
- dress (连衣裙)
- coat (外套)
- sandal (凉鞋)
- shirt (衬衫)
- sneaker (运动鞋)
- bag (包)
- ankle boot (短靴)

#### ◆ 图像数据

- 数据尺寸是 (C x H x W)
  - 第一维是通道数，灰度图像通道数为1
  - 后面两维分别是图像的高和宽，高和宽均为28像素
  - 一张图像像素数为  $28 \times 28 = 784$



# PyTorch网站上的入门基础教程

## 0. [Quickstart](#)

[https://pytorch.org/tutorials/beginner/basics/quickstart\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html)

## 1. [Tensors](#)

[https://pytorch.org/tutorials/beginner/basics/tensorqs\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html)

## 2. [Datasets and DataLoaders](#)

[https://pytorch.org/tutorials/beginner/basics/data\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/data_tutorial.html)

## 3. [Transforms](#)

[https://pytorch.org/tutorials/beginner/basics/transforms\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/transforms_tutorial.html)

## 4. [Build Model](#)

[https://pytorch.org/tutorials/beginner/basics/buildmodel\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html)

## 5. [Automatic Differentiation](#)

[https://pytorch.org/tutorials/beginner/basics/autogradqs\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html)

## 6. [Optimization Loop](#)

[https://pytorch.org/tutorials/beginner/basics/optimization\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html)

## 7. [Save, Load and Use Model](#)

[https://pytorch.org/tutorials/beginner/basics/saveloadrun\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html)

# 导入函数库、装载数据

## ➤ 导入函数库

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor, Lambda
```

## ➤ 装载数据

```
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)

train_dataloader = DataLoader(training_data, batch_size=64)
test_dataloader = DataLoader(test_data, batch_size=64)
```

# 设计模型

## ➤ PyTorch编程实现MLP的三种方法

- ◆ 利用顺序容器`torch.nn.Sequential`
- ◆ 利用自定义的`torch.nn.Module`子类
- ◆ 利用张量计算的底层方法 `torch.tensor`

# 设计模型：方法一

## ➤ 利用顺序容器torch.nn.Sequential

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork()
```

# 设计模型：方法二

## ➤ 利用自定义torch.nn.Module子类

### ◆ 在初始化函数中用nn.Parameter 定义模型参数

```
def relu(X):  
    return torch.max(input=X, other=torch.tensor(0.0))  
  
class NeuralNetwork(nn.Module):  
    def __init__(self):  
        super(NeuralNetwork, self).__init__()  
        self.flatten = nn.Flatten()  
        num_inputs, num_outputs, num_hiddens = 784, 10, 256  
  
        self.W1 = torch.nn.Parameter(torch.randn((num_inputs, num_hiddens), dtype=torch.float))  
        self.b1 = torch.nn.Parameter(torch.zeros(num_hiddens, dtype=torch.float))  
        self.W2 = torch.nn.Parameter(torch.randn((num_hiddens, num_outputs), dtype=torch.float))  
        self.b2 = torch.nn.Parameter(torch.zeros(num_outputs, dtype=torch.float))  
  
    def forward(self, X):  
        X = self.flatten(X)  
        H = relu(torch.matmul(X, self.W1) + self.b1)  
        logits = torch.matmul(H, self.W2) + self.b2  
        return logits
```

# 设计模型：方法二的训练过程

```
learning_rate = 0.1
batch_size = 64
epochs = 5
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f} [{current:>5d}/{size:>5d}]")

    for t in range(epochs):
        print(f"Epoch {t+1}\n-----")
        train_loop(train_dataloader, model, loss_fn, optimizer)
```

# 设计模型：方法三

## ➤ 利用张量计算的底层方法 torch.tensor

```
num_inputs, num_outputs, num_hiddens = 784, 10, 256
```

```
W1 = torch.tensor(np.random.normal(0, 0.01, (num_inputs, num_hiddens)), dtype=torch.float)
b1 = torch.zeros(num_hiddens, dtype=torch.float)
W2 = torch.tensor(np.random.normal(0, 0.01, (num_hiddens, num_outputs)), dtype=torch.float)
b2 = torch.zeros(num_outputs, dtype=torch.float)
```

```
params = [W1, b1, W2, b2]
for param in params:
    param.requires_grad_(requires_grad=True)
```

```
def relu(X):
    return torch.max(input=X, other=torch.tensor(0.0))
```

```
def net(X):
    X = X.view((-1, num_inputs))
    H = relu(torch.matmul(X, W1) + b1)
    return torch.matmul(H, W2) + b2
```



# 设计模型：方法三的训练过程

```
def train(net, train_iter, test_iter, loss, num_epochs, params, lr):
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, test_acc_sum, n_train, n_test = 0.0, 0.0, 0.0, 0, 0
        for X, y in train_iter:
            #X是当前批次的训练数据, y是对应的标签
            y_hat = net(X)
            l = loss(y_hat, y).sum()

            # 梯度清零
            if params[0].grad is not None:
                for param in params:
                    param.grad.data.zero_()

            #误差反向传播
            l.backward()

            #小批量梯度下降算法
            if params[0].grad is not None:
                for param in params:
                    param.data -= lr * param.grad # 注意这里更改param时用的param.data

        train_l_sum += l.item()
        train_acc_sum += (y_hat.argmax(dim=1) == y).sum().item() #统计训练集识别正确的样本数
        n_train += y.shape[0] #统计训练集样本总数
```

# 小结

## ➤ 神经网络是一种非线性建模方法

- ◆ 网络模型
- ◆ 目标函数
- ◆ 训练过程中的误差反向传播方法

# 本周需要掌握的内容

## ➤ 原理

- ◆ 前馈神经网络（多层感知机）
- ◆ 目标函数
  - 回归：MSE
  - 多类分类：网络输出层采用softmax激活函数 + 交叉熵
- ◆ 误差反向传播、梯度下降法

## ➤ 数学基础

- ◆ 复合函数求导：链式法则

◆ Softmax

◆ 交叉熵

## ➤ 编程实践

- ◆ 网络学堂“课程文件”→“编程实践”栏目中的“第三周演示程序与教程.zip”
  - week3-demo.py
  - tutorial-3-MLP.ipynb

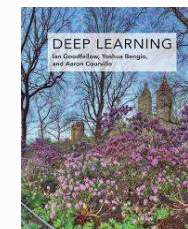
# 参考书

## ➤ 参考书

- ◆ Ian Goodfellow and Yoshua Bengio et al., Deep Learning, MIT Press, 2016.

<http://www.deeplearningbook.org>

- Chapter 6



- ◆ 周志华,机器学习, 清华大学出版社, 2016年

- 第5章 神经网络

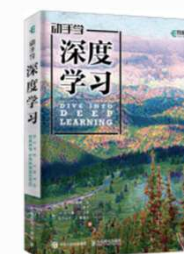



- ◆ 《动手学深度学习》(PyTorch版)

- 第三章第3.4 -3.10节

• <https://tangshusen.me/Dive-into-DL-PyTorch/#/>

• <https://github.com/ShusenTang/Dive-into-DL-PyTorch>





谢谢大家！