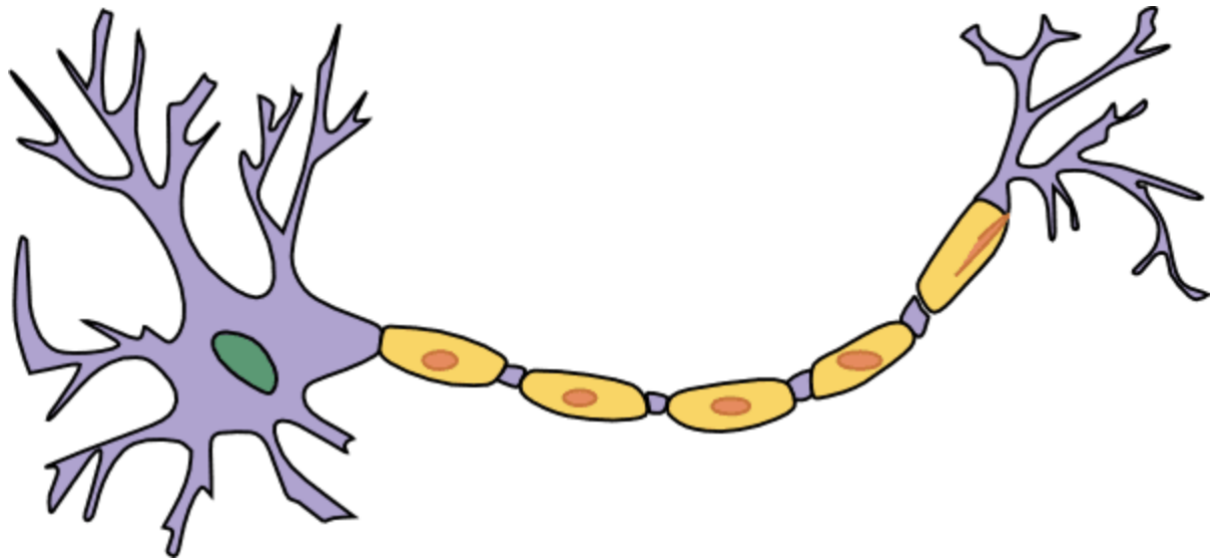
感知机、logistic回归和多层神经网络

江西财经大学 | 易亚伟 | 2202291160@stu.jxufe.edu.cn

感知机

感知机的引入

机器学习中如果我们的模型预测的结果是离散值，则此类学习任务称为‘分类’。如果预测值是连续值，则此类任务称为‘回归’。



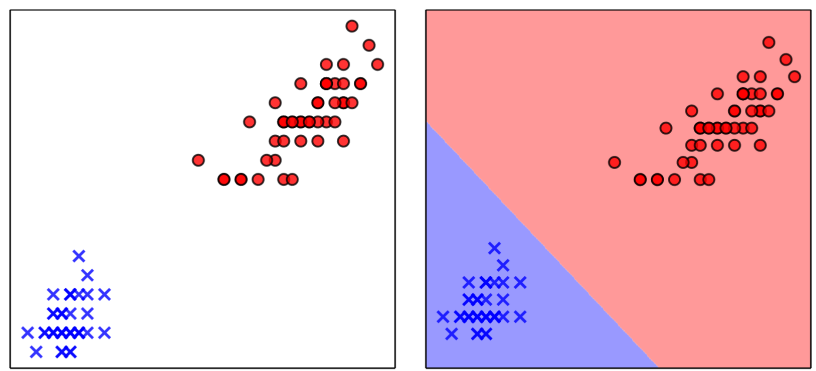
感知机：在自然界中，一个神经元会感知周围的环境，然后对外发出两种类别的信号，我们可以对神经元来抽象解决分类问题，建立的模型称为感知机。

根据我们之前的线性回归模型。可以根据数据得到以下模型

$$f(x) = \omega^T x + b$$

如果要解决一个二分类问题，很自然地，我们可以按照$f(x) >= 0$和$f(x) < 0$进行分类 即：

$$a\text{类：}\ f(x) >= 0$$
$$\text{非}a\text{类：}\ f(x) < 0$$

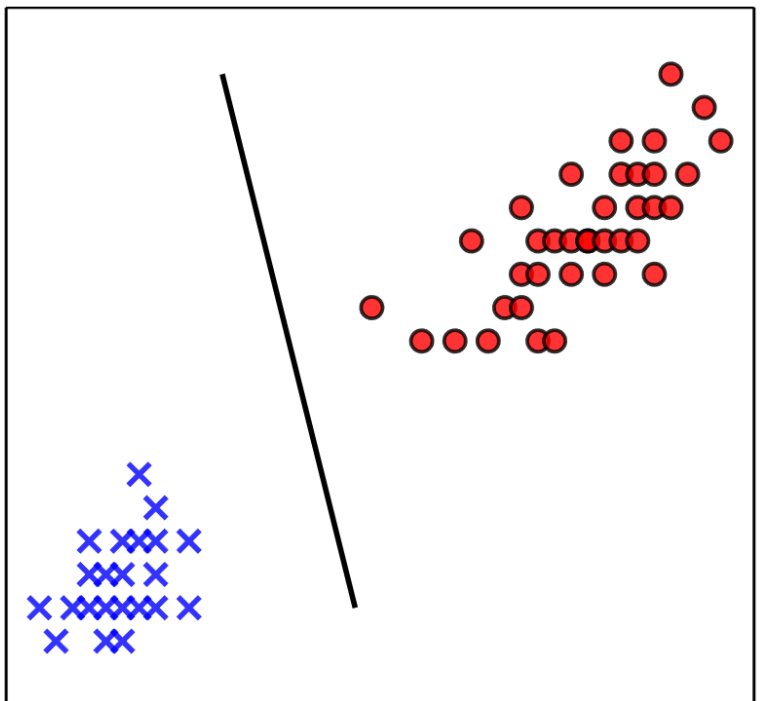


两种类别的数据　　感知机找到的分界线

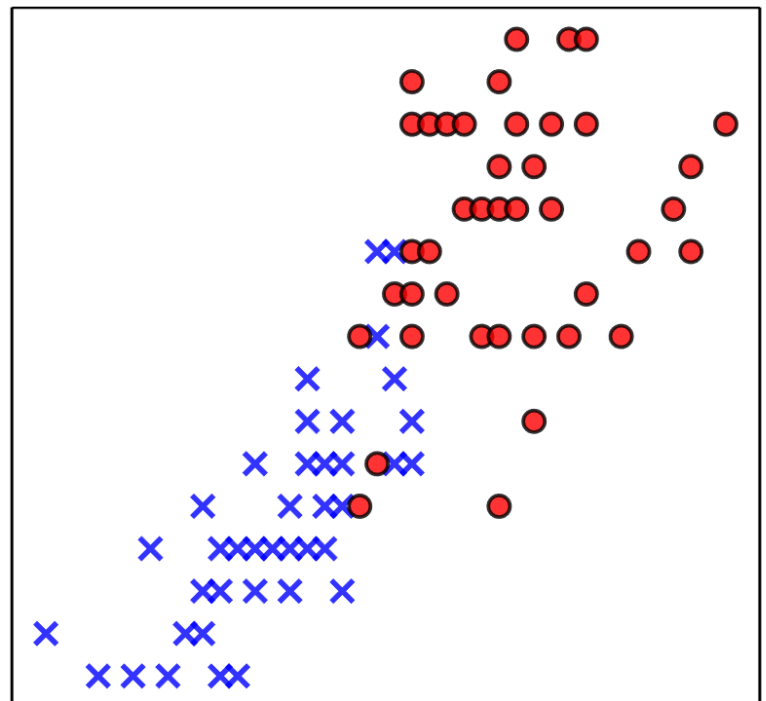
感知机本质上就是找一条分界线，把数据分开，感知机只能将数据分为两类，并且分界线是直线，所以又称它为**二分类线性模型**

感知机的缺陷

感知机只能应用在**线性可分**的数据集中，如果数据集不是线性可分的，感知机无法找到分界线



线性可分的数据集



非线性可分的数据集

logistic回归模型

核心公式

logistic核心函数

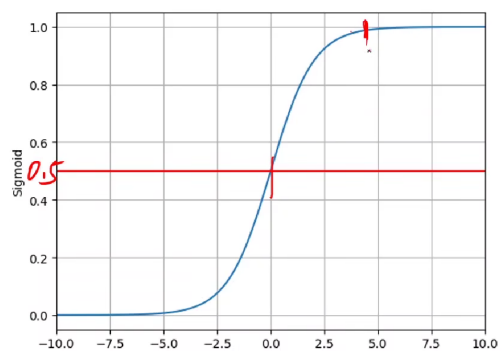
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

把y映射到[0,1]区间

$$x \to \infty 时 y \to 1$$
$$x = 0 时 y = 0.5$$
$$x \to -\infty 时 y \to 0$$

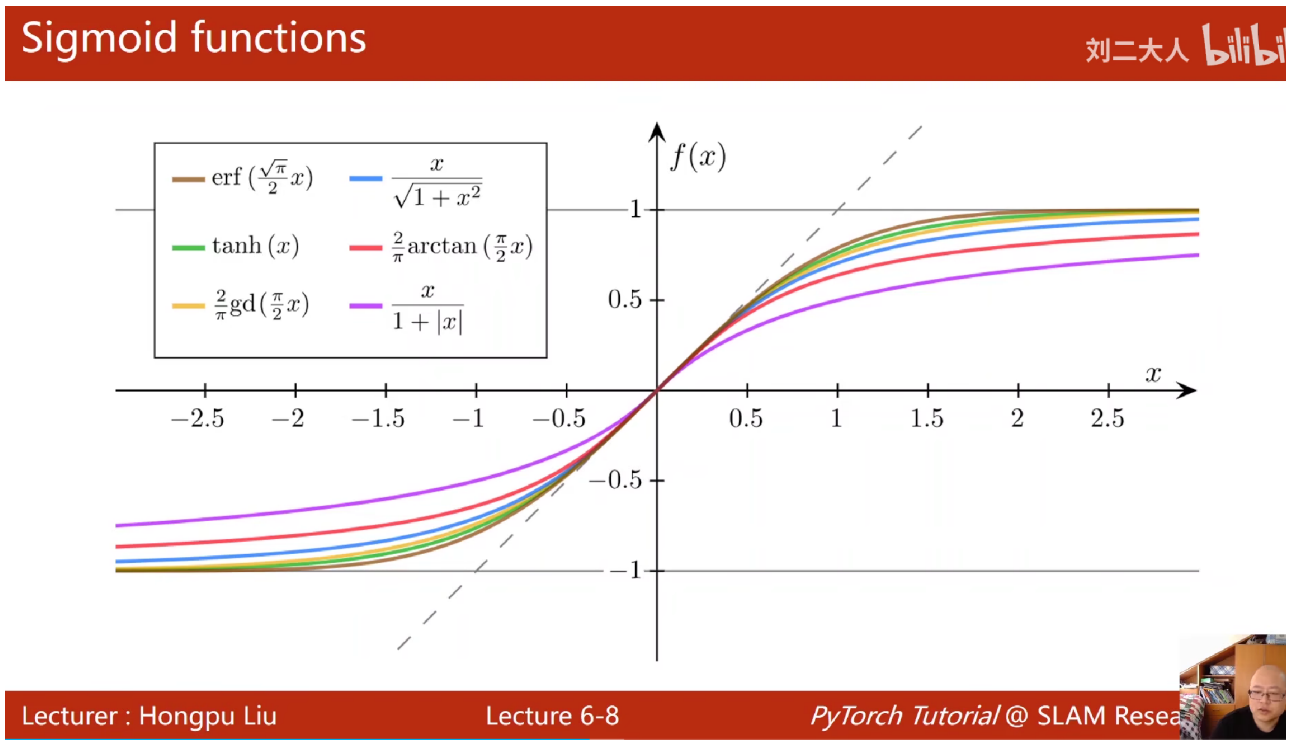
饱和函数：x大于0后导数逐渐下降,趋近于0;x小于0x越小导数逐渐下降，趋近于0

通过引入logistic函数，可以将分类问题转化为回归问题，将线性模型的结果带入logistic函数，输出结果表示概率。



激活函数

在神经网络中,输入经过权值加权计算并求和之后,需要经过一个函数的作用,这个函数就是激活函数(Activation Function)。



如果不经过激活函数，神经网络中的每一层的输出都是上一层输入的线性函数，这时神经网络就相当于感知机。

激活函数的特点

所有sigmoid函数，都满足以下条件：

- 有极限
- 单调递增
- 饱和函数

模型

之前的模型：

$$\hat{y} = x * \omega + b$$

现在的模型

$$\hat{y} = \sigma(x * \omega + b)$$

保证输出值在0-1之间

模型改进

二分类损失函数（BCE）

$$
\begin{gathered}
loss = -(ylog\hat{y} + (1-y)log(1-\hat{y})) \\
y = 1 \rightarrow loss = -ylog\hat{y}, \hat{y} \rightarrow 1 \\
y = 0 \rightarrow loss = -log(1-\hat{y}), \hat{y} \rightarrow 0
\end{gathered}
$$

可以让$\hat{y}$趋近于真实值

小批量二分类损失函数

$$
loss = -\frac{1}{N}\sum_{n-1}^{N} y_n \log \hat{y}_n \log(1-\hat{y})
$$

双列布局

主要代码

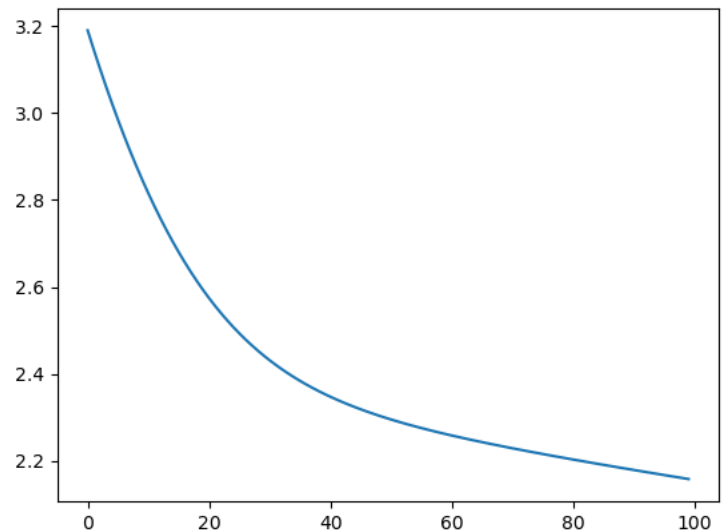
```
# 构建模型
class LogisticModel(torch.nn.Module):  # 都要继承只Module
    def __init__(self):
        # super(LinearModel, self).__init__()  # 调用父类构
        super(LogisticModel,self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # 构造对象包含

    def forward(self, x):
        # y_pred = self.linear(x)  # 在liner会调用forward
        y_pred = F.sigmoid(self.linear(x)) #把结果应用sigm
        return y_pred

epoch_list = []
loss_list = []

model = LogisticModel()
# criterion = torch.nn.MSELoss(size_average=False)#是否求均
criterion = torch.nn.BCELoss(size_average=False)#求损失值,
optimizer = torch.optim.SGD(model.parameters(),lr=0.01)#优
```

运行效果

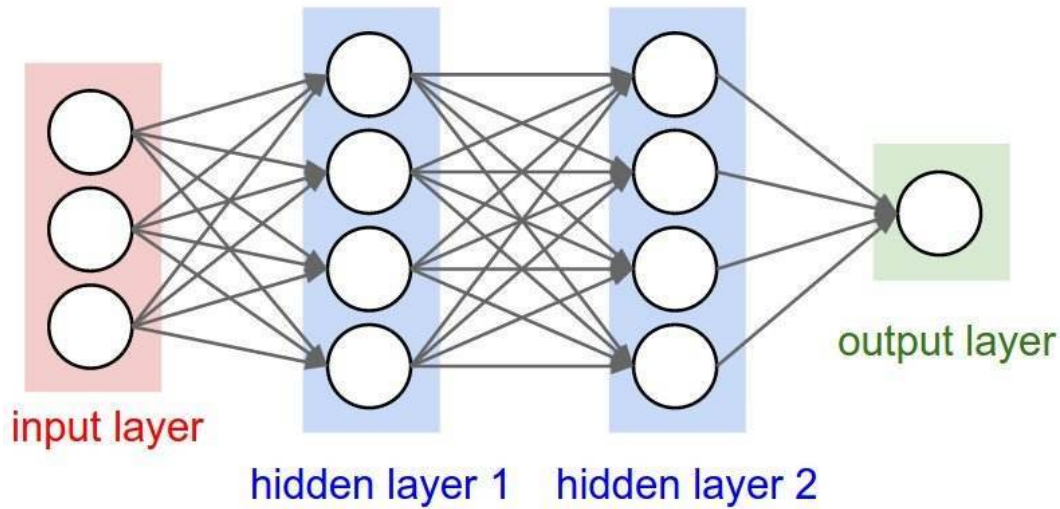


多层神经网络

多层神经网络的实现

将多个神经元按一定的层次结构连接起来，就得到了**神经网络**。

隐层：输入层与输出层之间的一层神经元，对输入层的信号进行加工，最终结果由输出层神经元输出。



神经网络之间的隐层越多，学习能力越强。但有隐层过多可能造成学到数据中的噪声，造成过拟合

糖尿病数据分类实践

X1	X2	X3	X4	X5	X6	X7	X8	Y
-0.29	0.49	0.18	-0.29	0.00	0.00	-0.53	-0.03	0
-0.88	-0.15	0.08	-0.41	0.00	-0.21	-0.77	-0.67	1
-0.06	0.84	0.05	0.00	0.00	-0.31	-0.49	-0.63	0
-0.88	-0.11	0.08	-0.54	-0.78	-0.16	-0.92	0.00	1
0.00	0.38	-0.34	-0.29	-0.60	0.28	0.89	-0.60	0
-0.41	0.17	0.21	0.00	0.00	-0.24	-0.89	-0.70	1
-0.65	-0.22	-0.18	-0.35	-0.79	-0.08	-0.85	-0.83	0
0.18	0.16	0.00	0.00	0.00	0.05	-0.95	-0.73	1
-0.76	0.98	0.15	-0.09	0.28	-0.09	-0.93	0.07	0
-0.06	0.26	0.57	0.00	0.00	0.00	-0.87	0.10	0

数据读取

```python
import numpy as np
import torch
import matplotlib.pyplot as plt
# x = np.loadtxt(r'./diabetes_data.csv/X.csv',delimiter=' ',dtype=np.float32)
# y = np.loadtxt(r'./diabetes_target.csv/y.csv',delimiter=' ',dtype=np.float32)
#
# x = torch.from_numpy(x)
# y = torch.from_numpy(y)
# 数据是这个老师自己写的
xy = np.loadtxt('diabetes.csv.gz',delimiter=',',dtype = np.float32)
#除了最后一行数据均放入x_data矩阵中
x_data = torch.from_numpy(xy[:,:-1])
#只把最后一行放入y_data向量
y_data = torch.from_numpy(xy[:,[-1]])
# 创建了两个tensor
```

模型构建

```
# 构建模型
class Model(torch.nn.Module):  # 都要继承只Module
    def __init__(self):
        # super(LinearModel, self).__init__()  # 调用父类构造函数
        super(Model,self).__init__()
        self.linear1 = torch.nn.Linear(8, 6)  # 第一层 8维到6维
        self.linear2 = torch.nn.Linear(6, 4)  # 第二层 6维到4维
        self.linear3 = torch.nn.Linear(4, 1)  # 第三层 4维到1维
        self.sigmoid = torch.nn.Sigmoid()  # 使用的激活函数，直接使用sigmoid

    def forward(self, x):
        # y_pred = self.linear(x)  # 在liner会调用forward
        x = self.sigmoid(self.linear1(x))
        x = self.sigmoid(self.linear2(x))
        x = self.sigmoid(self.linear3(x))
        return x
model = Model()
```

优化器构建

```
criterion = torch.nn.BCELoss(size_average=True)#二分类交叉熵
optimizer = torch.optim.SGD(model.parameters(),lr=0.01)#优化器 梯度下降算法
```

训练

```
epoch_list = []
loss_list = []
# 训练
for epoch in range(100000):
    y_pred = model(x_data)
    loss = criterion(y_pred,y_data)
    print(epoch,loss.item())
    epoch_list.append(epoch)
    loss_list.append(loss.item())

    # 反馈
    optimizer.zero_grad()
    loss.backward()
    # 更新
    optimizer.step()
plt.plot(epoch_list,loss_list)
plt.show()
```