

# 机器学习

xbZhong

2024-02-18

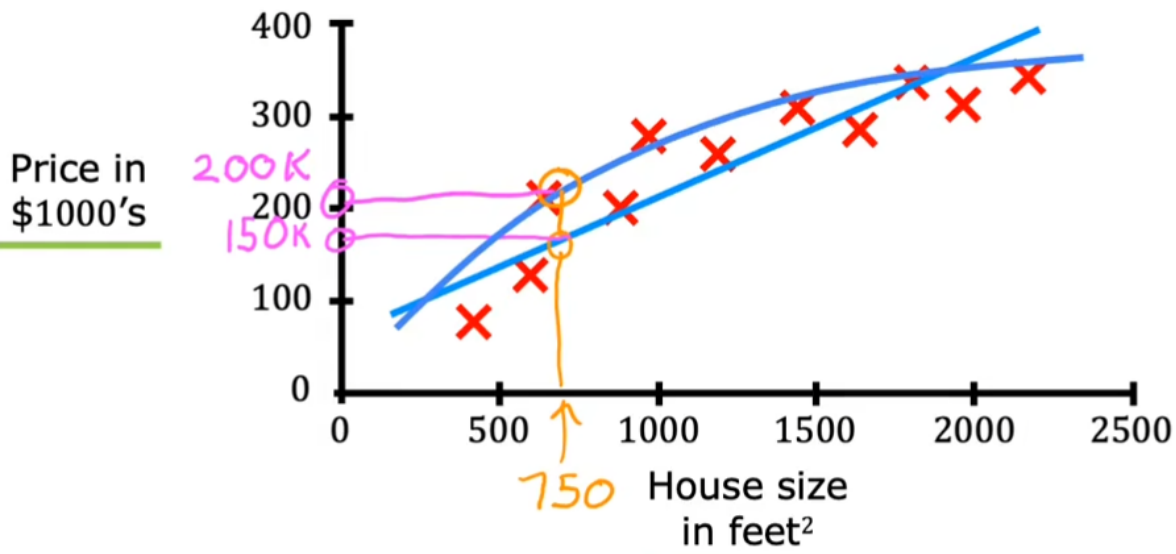
[本页PDF](#)

## 机器学习

### 监督学习 (Supervised learning)

指的是学习x到y或输入到输出映射的算法 关键特征：提供学习算法示例以供学习 ##### 回归

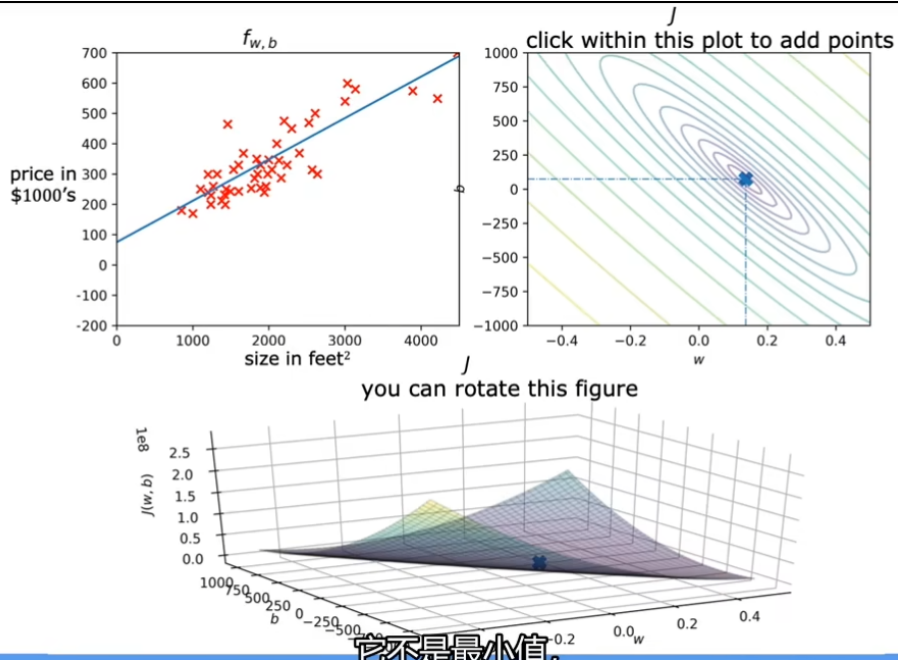
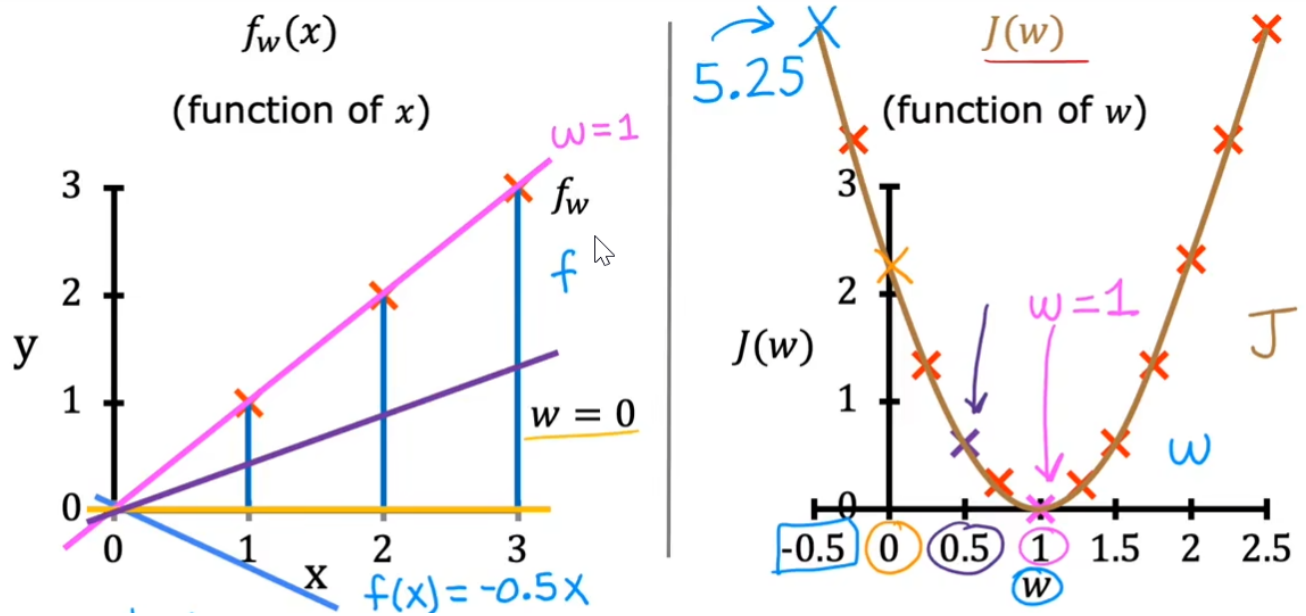
### Regression: Housing price prediction



\* 拟合曲线或更复杂的函数 \* 指的是通过建立数学模型(算法)来对因变量进行分析，算法预测的结果是无数可能值中的一个 \* 其学习从无限多的可能数字中预测数字

#### 线性回归模型

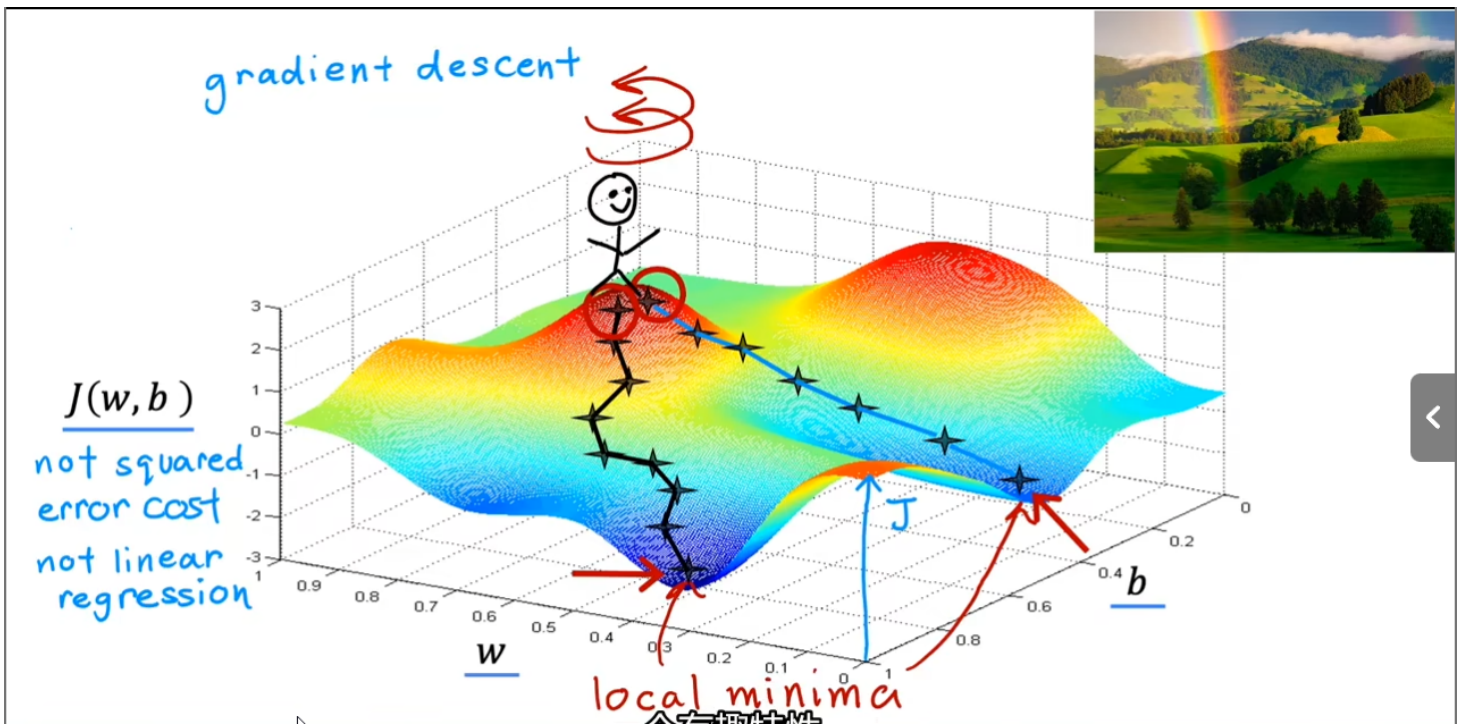
- 根据数据进行模型训练，然后根据数据拟合成一条直线
- 字母仅仅是 $y$ 时，表明是训练集中的真实值；是 $\hat{y}$ 则是模型的预测值
- $f(x) = wx + b$  ##### 成本函数
- 成本函数（平方误差成本函数）： $J = \frac{\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}{2m}$
- 成本函数是 $w$ 和 $b$ 的函数,取斜率为0的切线的那个切点即为极小值，使得成本函数最小



alt text

### 梯度下降

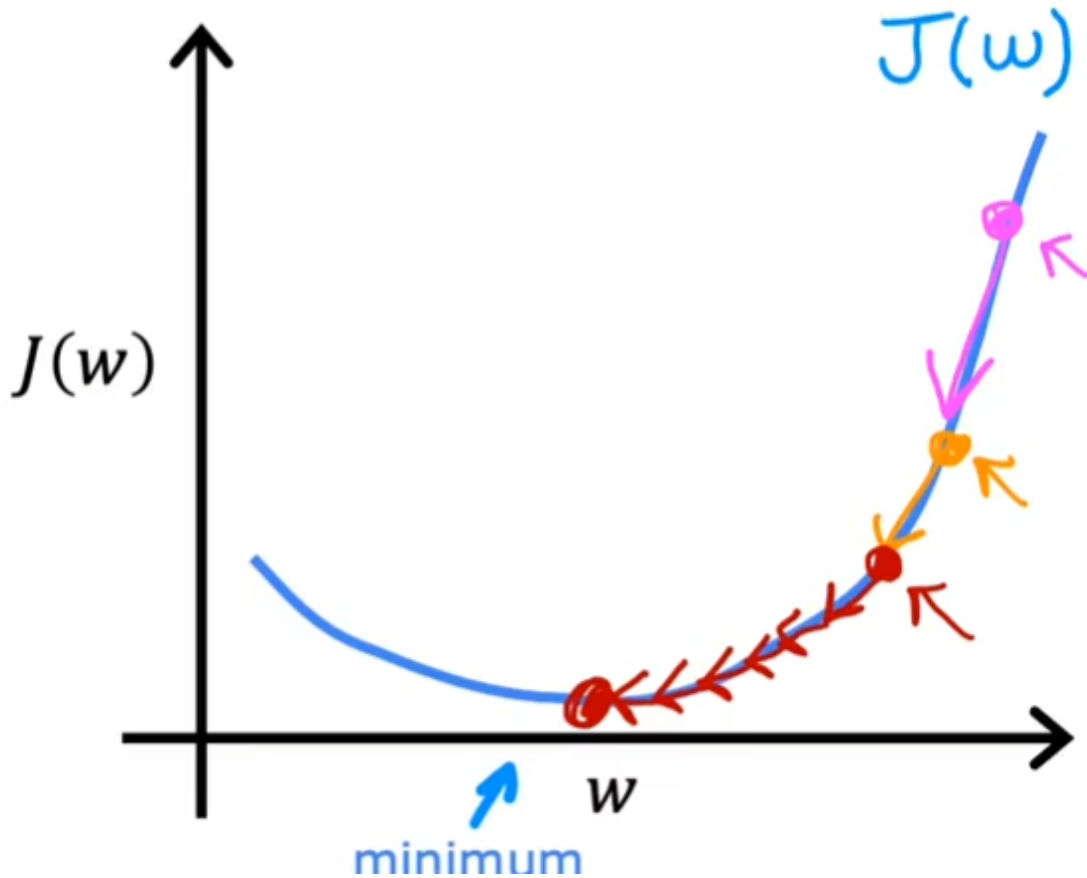
- 其是一种可用于尝试最小化任何函数的算法，适用于具有两个以上参数的模型



\* 梯度下降会用最快的方式把自己带到局部最小值，但局部最小值不一定相同，因为参数初始化的不同会让自己站在不同的位置，从而使得选择路线发生变化 ##### 梯度下降算法  $w = w - \alpha \frac{\partial}{\partial w} J(w, b)$   $b = b - \alpha \frac{\partial}{\partial b} J(w, b)$  \* 当 $\alpha$ (学习率)非常大时，那么其对应一个非常激进的梯度下降过程，其就是在尝试采取最大的步骤下坡 \* 学习率太小，梯度下降会起作用，但需要花费十分多时间 \* 学习率太大，会导致发散，找不到局部最优解 \* 随着导数的不断变化自动调整步长，不断逼近最小值 \*

# fixed learning rate

$\alpha$



左边为正确的代码写法：要实现并行计算

Correct: Simultaneous update	Incorrect
$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w, b)$	$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w, b)$
$tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w, b)$	$w = tmp\_w$
$w = tmp\_w$	$tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w, b)$
$b = tmp\_b$	$b = tmp\_b$

用于线性回归的梯度下降

通过求偏导可得： $w = w - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$   $b = b - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$

多元线性回归

多维特征

有多个因素可以影响 $y$ ，因此自变量可以看成是一个行向量 模型可以写成： $f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$   
 即： $\vec{w} = [w_1, w_2, w_3, w_4]$   $\vec{x} = [x_1, x_2, x_3, x_4]$  模型可以写成： $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x}$

向量化

使用numpy进行运算，使运算速率加快

## Vectorization

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

$$\mathbf{f} = \text{np.dot}(\mathbf{w}, \mathbf{x}) + \mathbf{b}$$



带有向量化多元线性回归的梯度下降

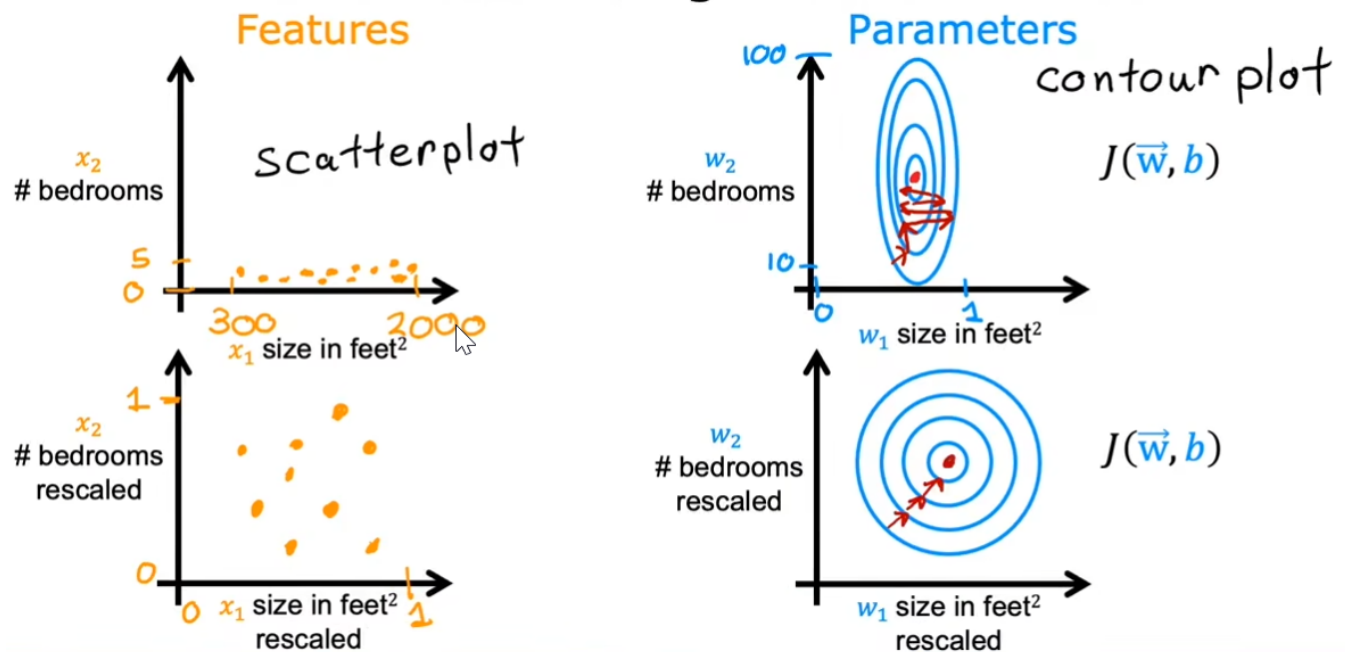
成本函数： $J(\vec{w}, b)$  梯度下降：

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w_1, \dots, w_n, b) = w_j - \alpha \frac{\partial}{\partial w_j} J(w_1, \dots, w_n, b) = w_j - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

特征缩放

通过缩放特征使得梯度下降更为方便，更易找到局部最小值

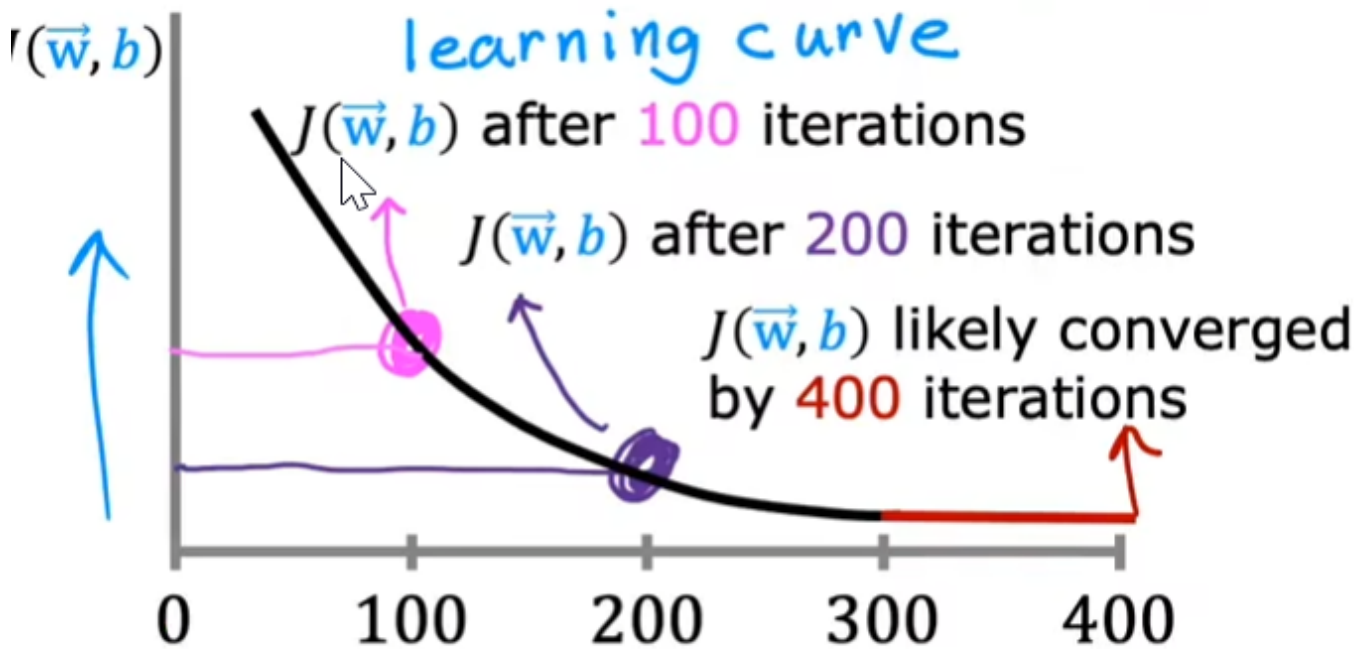
## Feature size and gradient descent



方法：1. 使用数据除以原始数据的范围的最大值，得出来的数据范围在x轴正半轴 2. 归一化： $x_1 = \frac{x_1 - \bar{x}}{x_{max} - x_{min}}$

学习曲线：利于帮助我们看到梯度下降迭代次数和成本函数之间的关系

objective:  $\min_{\vec{w}, b} J(\vec{w}, b)$   $J(\vec{w}, b)$  should decrease after every iteration



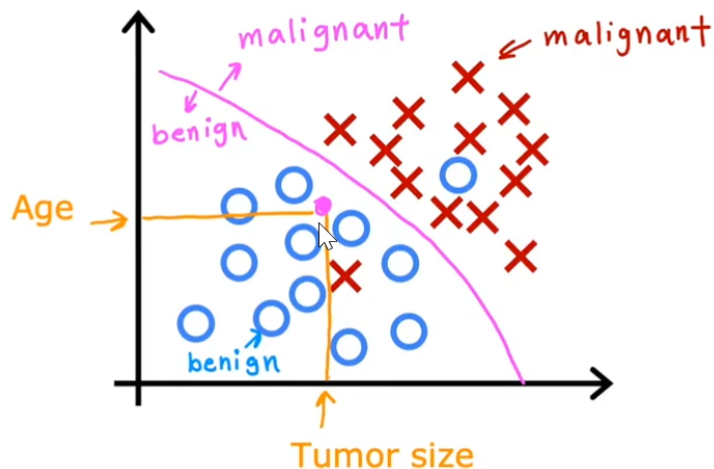
选择好的学习率

- 从小的学习率开始试，尝试几个不同的值后便可选取较为合适的学习率

## 分类

- 预测的是可能输出类别的有限集合，但不是介于两者之间的所有可能数字

## Two or more inputs



- alt text

- 拟合边界线

### 逻辑回归(单一分类算法)

- sigmoid函数:  $g(z) = \frac{1}{1+e^{-z}}$   $0 < g(z) < 1$
- 使用sigmoid函数将实数域映射到(0,1),并将其作为概率值

### 逻辑回归函数

- $f_{w,b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\vec{x}+b)}}$
- 模型可以有多个特征,即多元;也可以有高次幂
- 决策边界:不同类别的分界线

### 逻辑回归中的成本函数

损失函数是对单个样本的预测误差,成本函数通常是损失函数求和的平均 线性回归中的成本函数在逻辑回归中并不适用,因为其有许多极小值,不利于找到最低成本

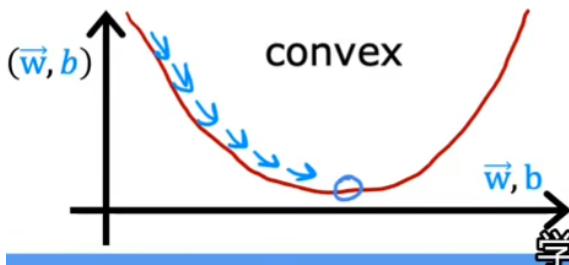
## Squared error cost

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$

loss  $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$

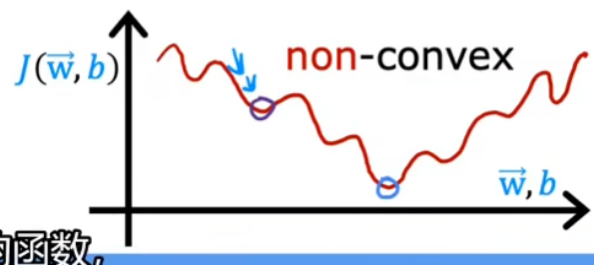
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$



学习算法预测的函数

所以要构建一个新的损失函数,即对数似然损失

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

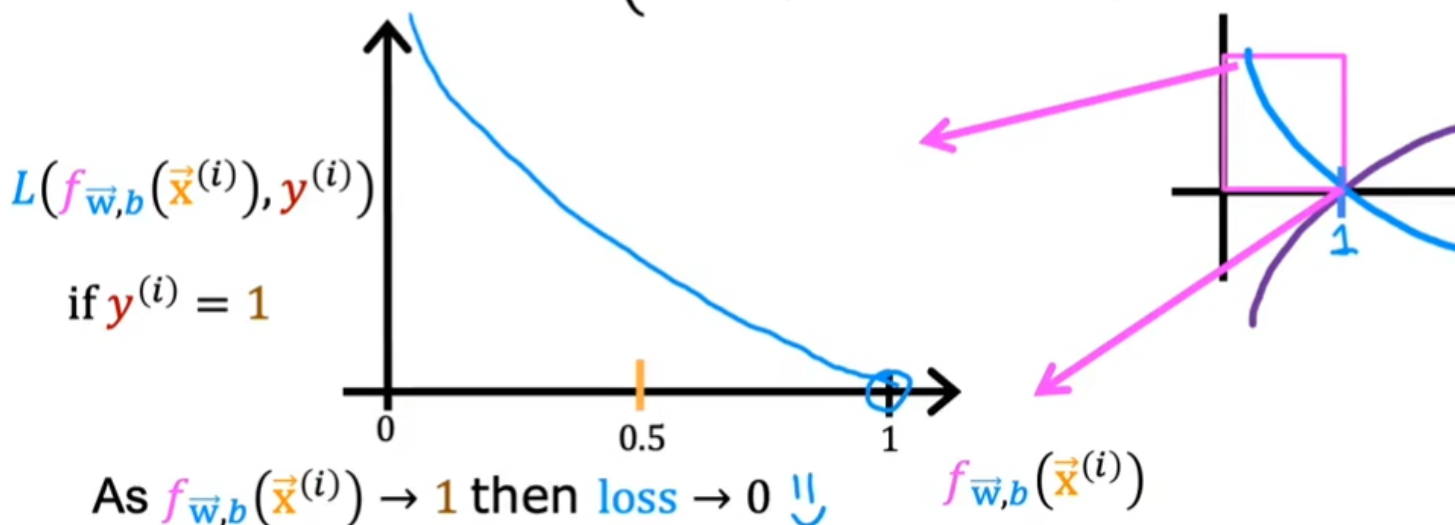
$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$$

if  $y^{(i)} = 1$ :

下图为真实值为1的损失函数图像,当 $f > 1$ ,即预估值趋近1时, $y$ 的值很小且斜率趋于0,表明此时损失很小

# Logistic loss function

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$



目标就是找到最适合的参数使得损失函数(成本函数)的值最小 损失函数:

$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$  成本函数:

$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$

使用梯度下降来拟合模型

其梯度下降算法与线性回归的基本相同(要进行数学推导), 但f的定义不一样, 线性回归的f是f=wx+b, 而逻辑回归的是sigmoid函数 梯度下降算法:

$$w_j = w_j - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad b = b - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

过拟合与欠拟合

- 欠拟合: 模型与数据不匹配, 具有高偏差(特征较少)
- 过拟合: 模型太过完美, 能够通过所有训练集, 但具有高方差(特征较多, 高阶多项)
- 防止过度拟合:
  1. 使用更多的训练数据
  2. 使用更少的特征
  3. 正则化

正则化(线性回归)

通过缩小系数使得特征减少, 从而使拟合的曲线更加平滑, 不容易产生过拟合 \* 正则化参数:  $\lambda$  \* 正则化项:  $\frac{\lambda}{2m} \sum_{j=1}^n (w_j)^2$  \* 成本函数:  $J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n (w_j)^2$  正则化参数的大小决定了正则化项所占的权重, 我们最后的目的是取什么参数时成本函数能取到最小值。如果正则化参数很大, 说明w要很小, 这时候回到模型去看会导致欠拟合;



如果正则化参数为0, 说明w随意取, 这时候会导致过拟合 梯度下降算法: 正则化的本质就是让w在每次更新中减小一点, 从而让w更小防止过拟合

$$w_j = w_j - \alpha \left( \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j \right) b = b - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

正则化(逻辑回归)

成本函数:  $J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$

$$w_j = w_j - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j b = b - \frac{1}{m} \alpha \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

正则化(神经网络)

# Neural network regularization

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m} \sum_{i=1}^m L(f(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m} \sum_{\text{all weights } \mathbf{W}} (w^2)$$

b

regularized MNIST model

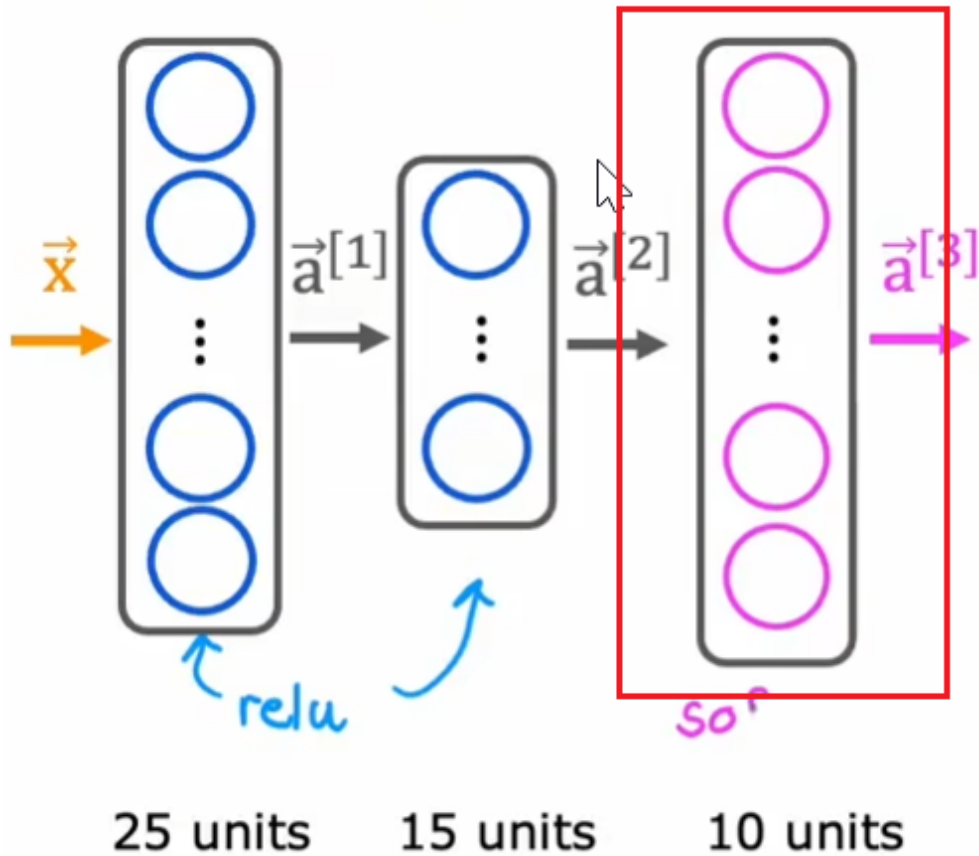
alt text

多类分类

每个样本只能属于一个类别

逻辑回归里面的分类不局限于二元, 可以是多元 在输出的时候要换成多个输出单元

# Neural Network wi



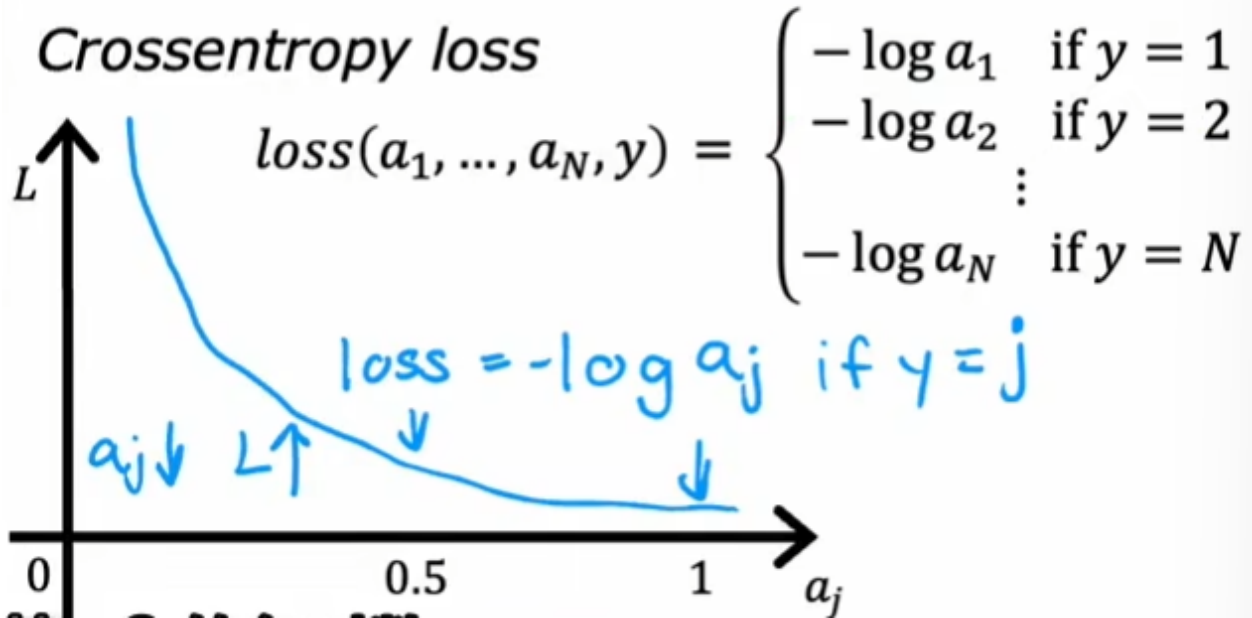
softmax回归

$$z_1 = w_1 x + b_1 z_2 = w_2 x + b_2 z_3 = w_3 x + b_3 a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 1 | \vec{x}) a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

softmax回归成本函数

# Softmax regression

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = 1 | \vec{x})$$
$$\vdots$$
$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = N | \vec{x})$$



alt text

**softmax**激活函数是多元函数 误差问题：在编写代码的时候，最好不要设置太多的中间变量，不然计算结果会有较大的误差

## 多标签分类

每个样本可以属于多个类别 可以构建多层神经层，一个神经层来判别一个类别

## 无监督学习 (Unsupervised Learning)

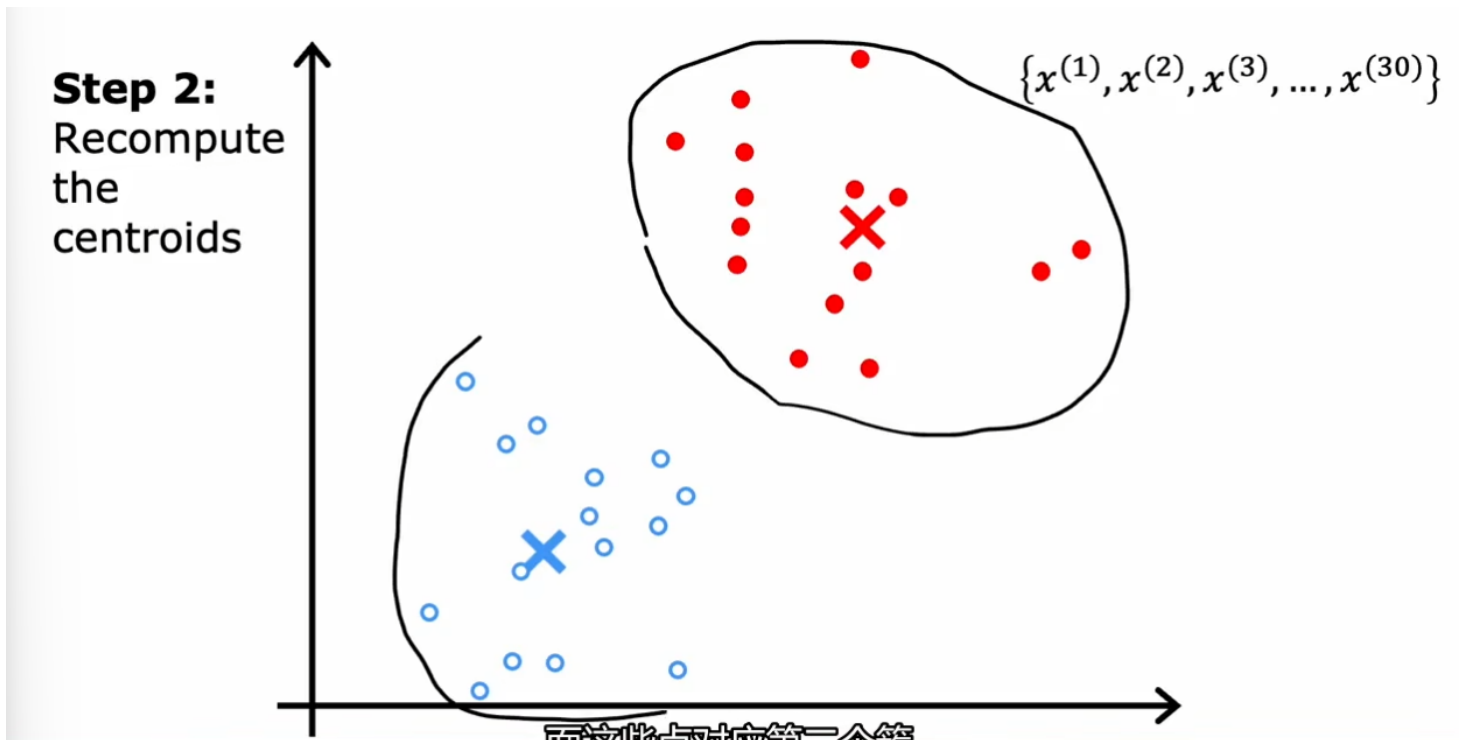
数据仅带有输入 $x$ 而无输出标签 $y$  #### 聚类算法 \* 数据集无标签，会把数据分成不同的集群

# Clustering: Grouping customers



## K-means 聚类算法

会重复做的两件事：将点分配给集群质心与移动集群质心 \* 随机猜测你可能要求它找到的两个聚类的中心位置 \* 遍历每一个点，查看其是否接近红十字还是蓝十字，将点染成对应的颜色 \* 查看所有红点的平均位置，并将红十字移到这个平均位置，蓝十字同理



## 成本函数

- $c^{(i)}$  可以看作集群中心的索引
- $\mu_k$  指的是集群中心k的位置

- $\mu_{c^{(i)}}$ 指的是这个数字所属的集群中心的位置

**成本函数(失真函数)**  $J = (c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$  \* m是样本的总体数量,  $x^{(i)}$ 是训练样例 \*  
降低成本函数: \* 将点分配给最近的集群中心, 对 $c_i$ 操作 \* 移动集群中心, 移到点的中间位置, 对 $u_i$ 操作

## 随机猜测

随机抽取训练样例, 让他们成为集群中心, 但很有可能会使成本函数达到局部最小值, 从而分类不太完美 可以运行多次, 找到多次里面成本函数的最小值

## 选择集群数量

- 绘制集群数量和成本函数的关系, 选择最小的成本函数

## 异常检测与降维

- 异常检测: 金融系统中的欺诈检测
- 降维: 将一个大数据集神奇地压缩成一个小的多的数据集, 同时丢失尽可能少的信息

## 密度估计

根据密度来进行区域划分, 然后根据概率来判断是否是异常 根据高斯分布来进行异常检测, 概率的判断

## 参数调整

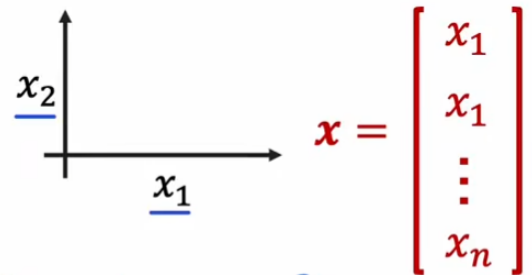
- 根据训练集, 交叉验证集, 测试集
- 可以混入一些异常样例, 在模型对交叉验证集进行异常检测的时候调整参数

## 多种特征

- 用一个向量来表示多个特征
- 出现的特征是相互独立的, 直接相乘

## Density estimation

Training set:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$   
Each example  $x_i$  has  $n$  features



$$p(x) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) *$$

$$= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

$$p(x_n; \mu_n, \sigma_n^2)$$

$$\begin{aligned} p(x_1 = \text{high temp}) &= 1/10 \\ p(x_2 = \text{high vibra}) &= 1/20 \\ p(x_1, x_1) &= p(x_1) * p(x_2) \\ &= \frac{1}{10} * \frac{1}{20} = \frac{1}{200} \end{aligned}$$

## 监督学习和异常检测的选择

- 例子少可以用异常检测, 且异常检测检测的是从来没有在训练集中出现的样本

- 使用监督学习会需要使用更多的样本来判断正例是什么样的，从而做出正确的判断，只会判断未来的样本和训练集的是否相似

## 选择合适特征(后续补充)

- 数据选择高斯分布的分布特征

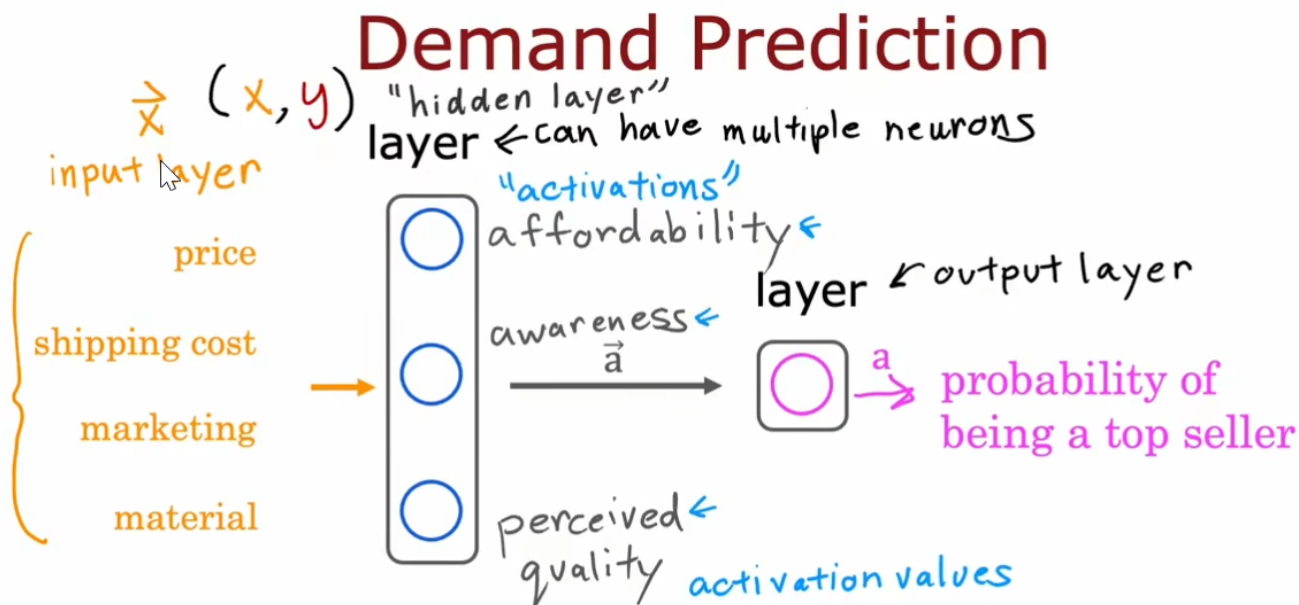
## 神经网络

神经网络中， $w$ 称为权重(一定有)， $b$ 称为偏置(可有可无)，神经元会根据权重对信号进行缩放，权重是神经元之间的连接强度

与传统的机器学习算法如逻辑回归，线性回归相比，神经网络能带来显著的性能的提高可以自己学习不同隐藏层的特征检测器

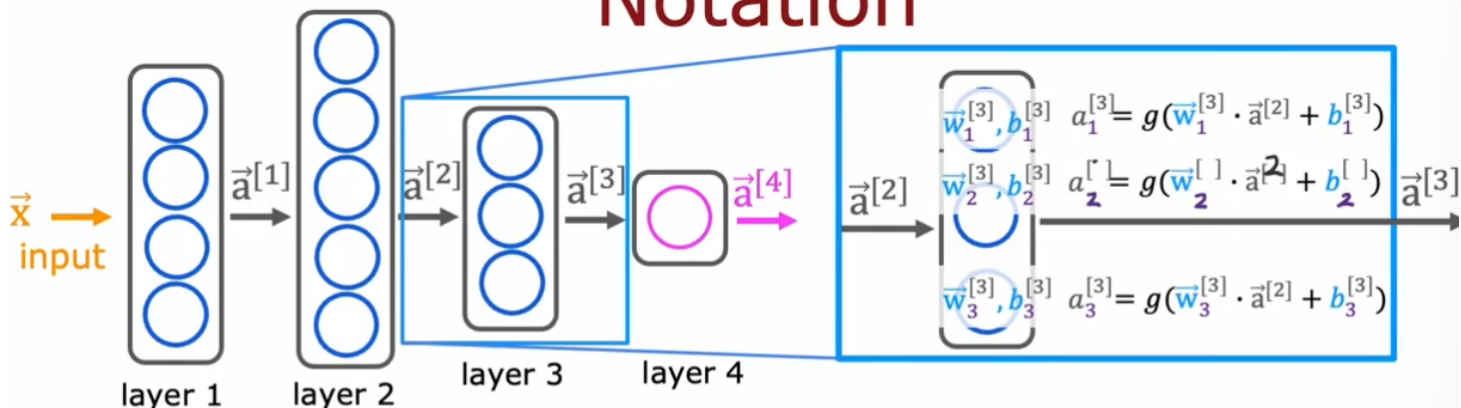
工作原理：\* 神经元是最小工作单元，多个神经元在一起组成层 \* 一个神经元有一个输出值，前层的输出(激活)是后层的输入 \* 一个神经网络由多个神经层组成，可以接受数字向量的输入和输出 \* 有输入层，输出层和多个隐藏层

- 本质上是用多个神经元分别处理特征，然后将他们的输出值作为一个神经元的输入值，再由其输出概率
- 如下图所示，将处理特征的多个神经元称为层，最左边为输入层，中间为隐藏层，右边为输出层



每个输出和参数右上角的数字表示他隶属于第几层的神经层

## Notation



#### 激活 指的是一个神经元向他下游的其他神经元发送多少高输出，换句话说就是神经元输出的计算过程 激活函数：处理输入，得到输出的那部分

## 泛化

指的是模型不仅在训练集上表现良好，还能再从未见过的数据上保持良好表现

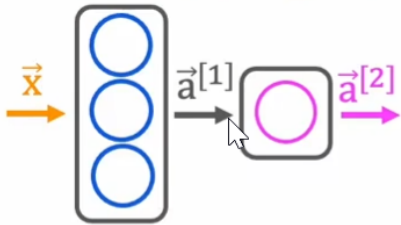
## 前向传播

在传播神经元的激活

## tensorflow

使用sequential方法将神经层串联起来

# Building a neural network architecture



```
→ layer_1 = Dense(units=3, activation="sigmoid")
→ layer_2 = Dense(units=1, activation="sigmoid")
→ model = Sequential([layer_1, layer_2])
```

```
x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

4 x 2

targets  $y = np.array([1, 0, 0, 1])$

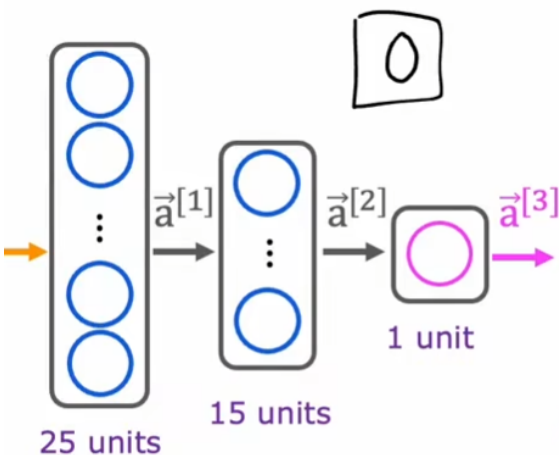
```
model.compile(...) ← more about this next week!
model.fit(x,y)
```

```
→ model.predict(x_new)
```

		y
200	17	1
120	5	0
425	20	0
212	18	1

使用tensorflow来编写神经网络代码 1. 指定模型 2. 使用特定损失函数编译模型 3. 训练模型

# Train a Neural Network in TensorFlow



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

```
model = Sequential([
    Dense(units=25, activation='sigmoid')
    Dense(units=15, activation='sigmoid')
    Dense(units=1, activation='sigmoid')
])
```

```
from tensorflow.keras.losses import BinaryCrossentropy
```

```
model.compile(loss=BinaryCrossentropy())
```

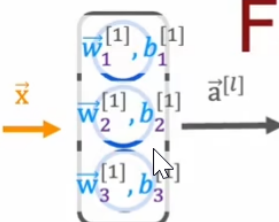
```
model.fit(X, Y, epochs=100)
```

given set of  $(x,y)$  examples

密集函数及其聚合

看懂代码实现，要有一定numpy基础，要注意是向量还是矩阵

# Forward prop in NumPy



$\vec{x}$  →  $\vec{a}^{[l]}$

$\vec{w}_1^{[1]} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$   $\vec{w}_2^{[1]} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$   $\vec{w}_3^{[1]} = \begin{bmatrix} 5 \\ -6 \end{bmatrix}$

$W = \text{np.array}(\begin{bmatrix} 1 & -3 & 5 \\ 2 & 4 & -6 \end{bmatrix})$  2 by 3

$b_1^{[l]} = -1$   $b_2^{[l]} = 1$   $b_3^{[l]} = 2$

$b = \text{np.array}([-1, 1, 2])$

$\vec{a}^{[0]} = \vec{x}$

$a\_in = \text{np.array}([-2, 4])$

```
def dense(a_in, W, b, g):  
    3 units = W.shape[1] [0,0,0]  
    a_out = np.zeros(units)  
    for j in range(units): 0,1,2  
        w = W[:,j]  
        z = np.dot(w, a_in) + b[j]  
        a_out[j] = g(z)  
    return a_out
```

```
def sequential(x):  
    a1 = dense(x, W1, b1)  
    a2 = dense(a1, W2, b2)  
    a3 = dense(a2, W3, b3)  
    a4 = dense(a3, W4, b4)  
    f_x = a4  
    return f_x
```

capital W refers to a matrix

## 激活函数

- 线性激活函数:  $g(z) = z$
- sigmoid函数, 值域范围是(0,1), 适合处理二元分类问题, 因为其有两个平坦面, 其梯度下降速度会更慢
- ReLU函数:  $g(z) = \max(0, z)$

## 优化算法

### Adam算法(自动调整学习率)

当梯度下降的方向确实朝着目标方向移动, 增加学习率; 当参数来回震荡, 降低学习率 1. 导入必要的库(torch.optim) 2. 将模型参数和初始学习率导入优化器 3. 记得要把梯度清零

## 网络层类型

- 密集层: 该层中的每个神经元都从前一层获得所有激活的输入
- 卷积层: 通过卷积核在图片上进行滑动提取特征, 输出通常是三维的

## 模型性能评估

将数据集分成测试集和训练集(计算训练集和测试集成本时不用加正则化项), 通过对测试集和训练集进行成本函数计算, 可以知道模型性能



# Train/test procedure for linear regression (with squared error cost)

Fit parameters by minimizing cost function  $J(\vec{w}, b)$

$$J(\vec{w}, b) = \min_{\vec{w}, b} \left[ \frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^n w_j^2 \right]$$

Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[ \sum_{i=1}^{m_{test}} (f_{\vec{w}, b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)})^2 \right]$$

Compute training error:

$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[ \sum_{i=1}^{m_{train}} (f_{\vec{w}, b}(\vec{x}_{train}^{(i)}) - y_{train}^{(i)})^2 \right]$$

可将数据分为测试集，训练集，交叉验证集，并且基于这三个数据子集的成本函数选择模型 通过交叉验证集去验证模型泛化性能，选择模型，找到最合适的模型

## Training/cross validation/test set

Training error:  $J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[ \sum_{i=1}^{m_{train}} (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 \right]$

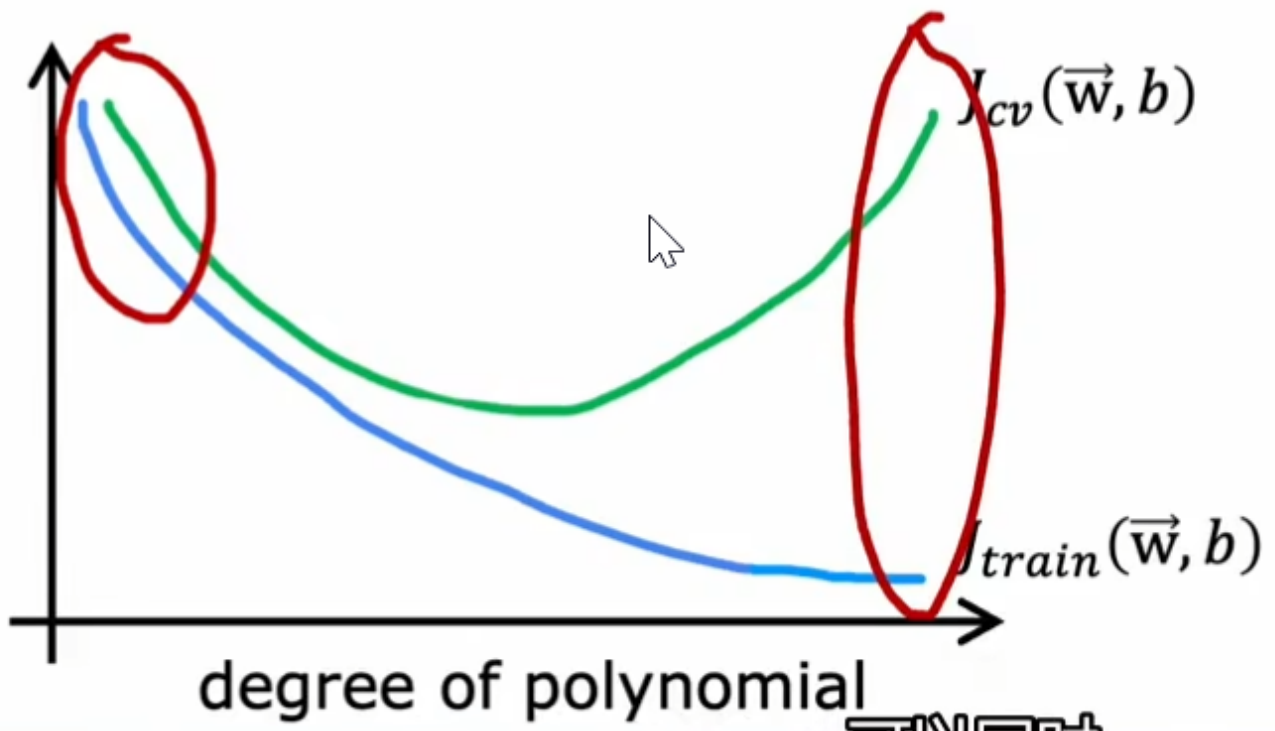
Cross validation error:  $J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}} \left[ \sum_{i=1}^{m_{cv}} (f_{\vec{w}, b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)})^2 \right]$  (validation dev errc)

Test error:  $J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[ \sum_{i=1}^{m_{test}} (f_{\vec{w}, b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)})^2 \right]$

通过偏差和方差评估模型性能

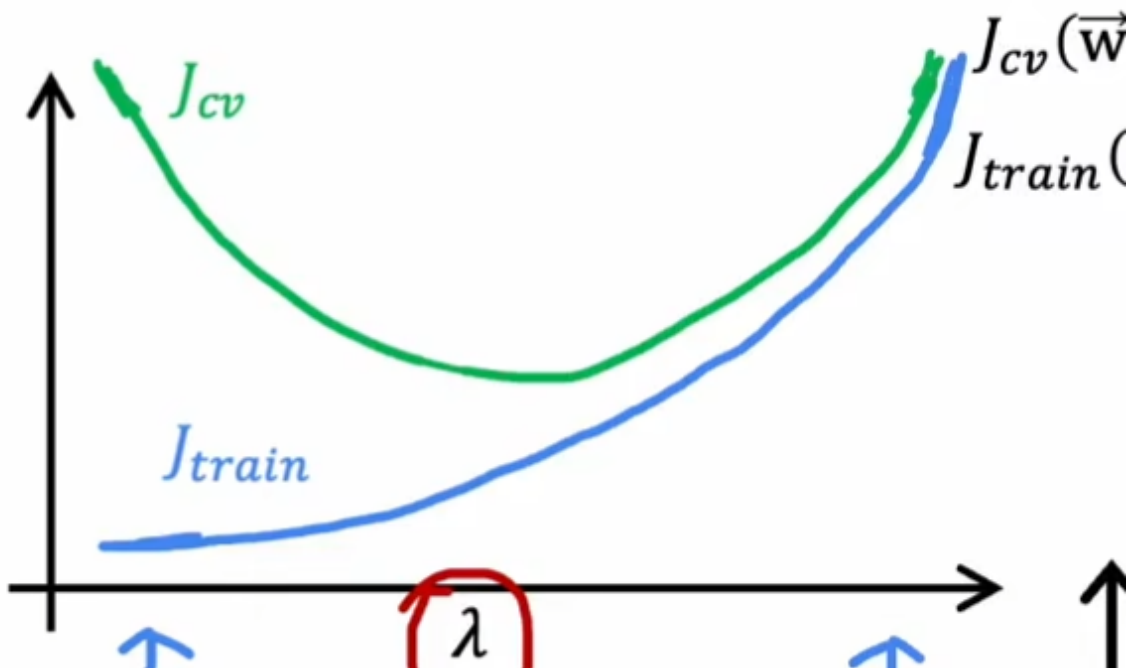
- 高偏差(欠拟合): 训练集的成本函数高
- 高方差(过拟合): 训练集的成本函数低, 但交叉验证集的成本函数高

多项式次数和成本函数之间关系,  $J_{cv}$ 为交叉验证集的成本函数



交叉验证集帮助确定正则化参数

使用不同的正则化参数，得出不同的成本函数，找到成本函数的最小值 正则化参数和成本函数关系图像



学习曲线

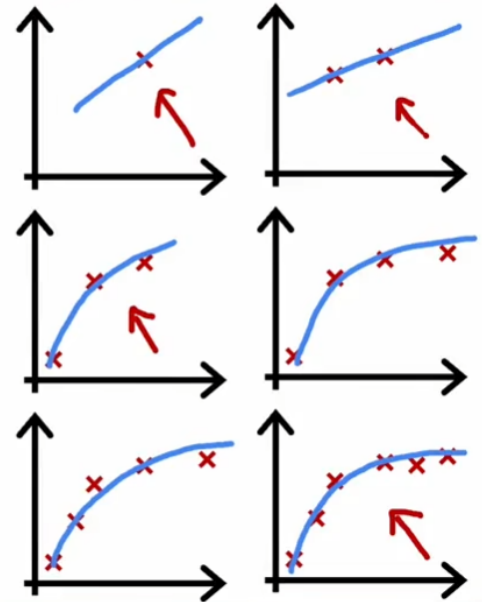
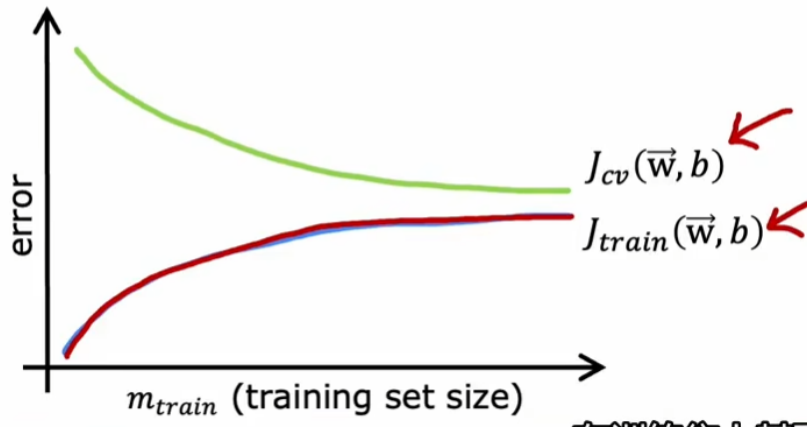
训练集大小和成本函数的关系 增加数据集不会改变成本函数的大小

# Learning curves

$J_{train}$  = training error

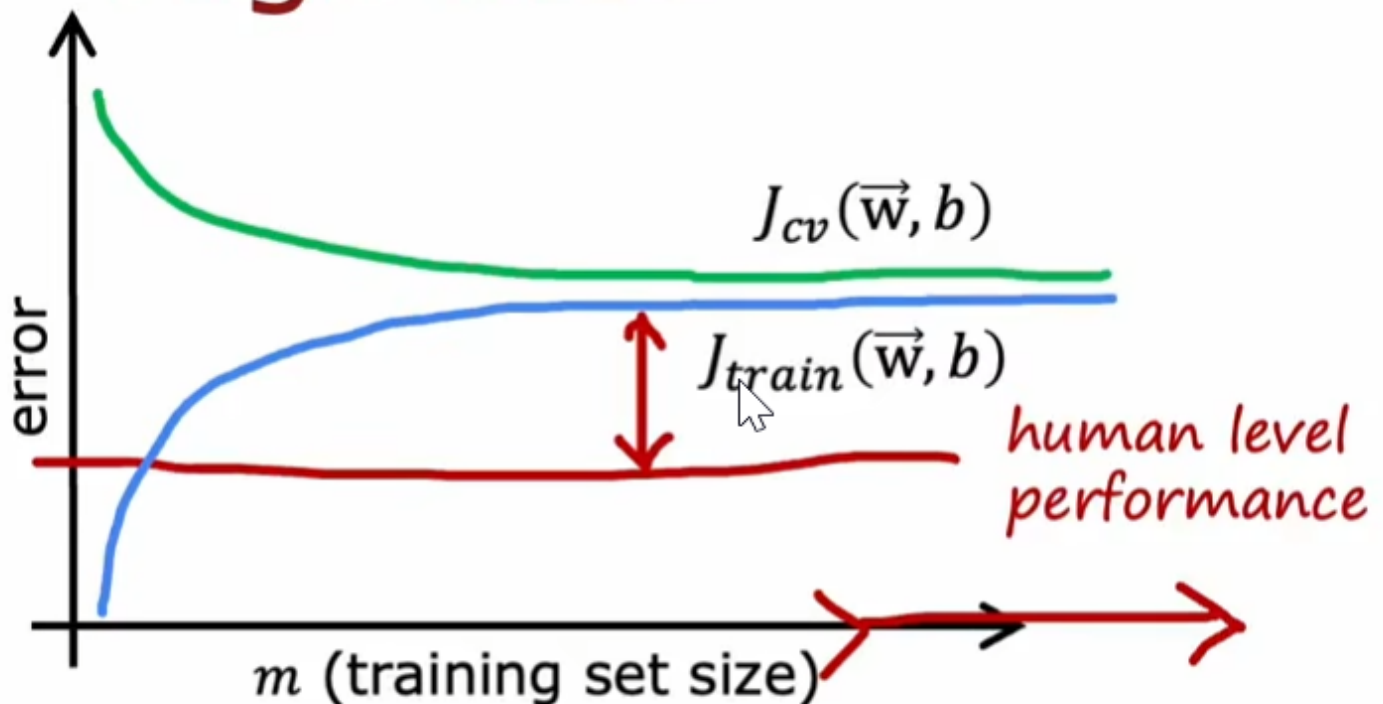
$J_{cv}$  = cross validation error

$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$$



高偏差图像 因为其模型拟合有问题，所以增加再多的训练集也不会提升模型性能

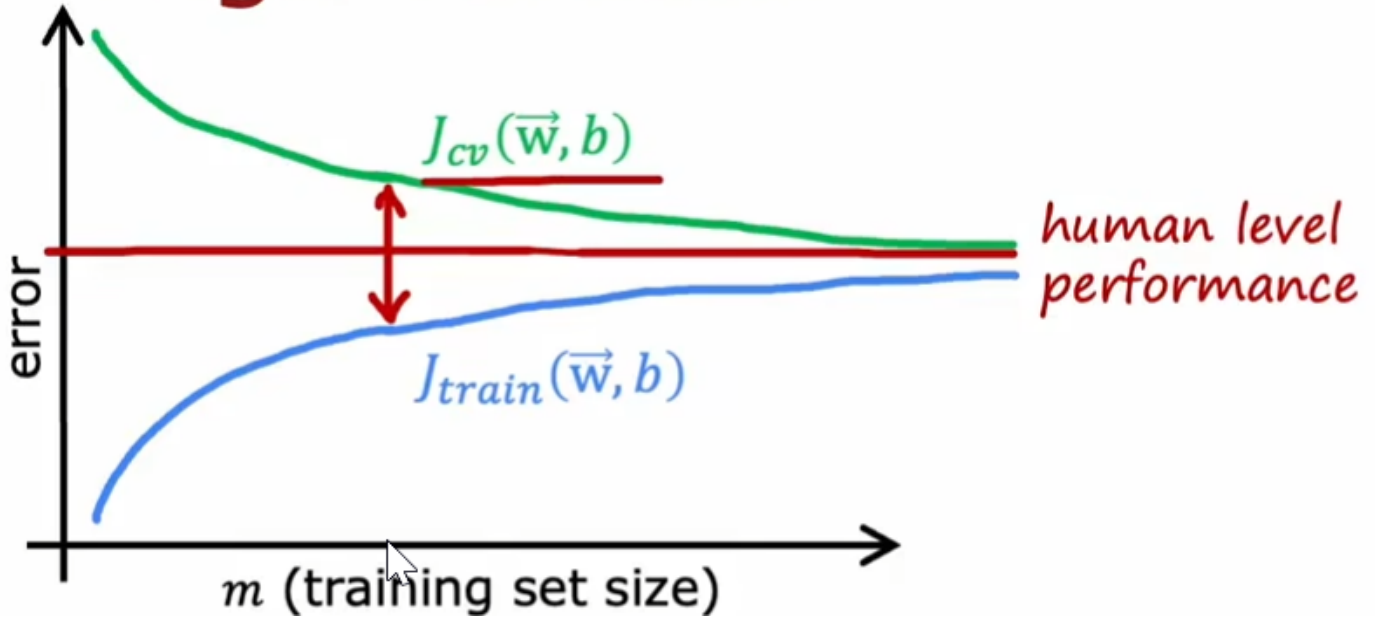
# High bias



高方差图像 增加足够多的训练集可以帮助提升模型性能

# High variance

f



解决高偏差高方差的方法

- ▶ Get more training examples
- ▶ Try smaller sets of features  $x, x^2, \cancel{x^3}, \cancel{x^4}, \cancel{x^5} \dots$
- ▶ Try getting additional features ←
- ▶ Try adding polynomial features  $(x_1^2, x_2^2, x_1 x_2, \text{etc})$
- ▶ Try decreasing  $\lambda$  ←
- ▶ Try increasing  $\lambda$  ←

- fixes high variance
- fixes high variance
- fixes high bias
- fixes high bias
- fixes high bias
- fixes high variance

大型神经网络可以降低高偏差

法则：\* 分析偏差方差帮助我们找到解决问题方法，分析数据帮助我们往哪个方向解决问题 \* 尝试添加更多有帮助的数据，可以使用数据增强方式来修改原有数据以获取更多数据 \* 也可以通过数据合成获取全新数据样例，需要花费较长时间 \* 迁移学习：可以复制除输出层之外的参数，然后使用优化算法或者梯度下降更新输出层参数；也可以重新训练所有参数。（先在大型数据集上进行训练，再在较小的数据集上进一步调整参数）使用别人预训练好的神经网络再对输出层进行微调也是很常见的。

## 误差指标

横轴为真实类别 纵轴为预测类别

		Actual Class	
		1	0
Predict-ed Class	1	True positive 15	False positive 5
	0	False negative 10	True negative 70
		↓ 25	↓ 75

\* 左上角为真阳性，右上角为假阳性 \* 左下角为假阴性，右下角为真阴性 \* 精度： $\frac{\text{真阳性}}{\text{真阳性} + \text{假阳性}}$  \* 召回率： $\frac{\text{真阳性}}{\text{真阳性} + \text{假阴性}}$  高精度意味着如果患者被诊断患有这种罕见疾病，那么他确实可能患有这种疾病，高召回率意味着如果患者确实有这种疾病，那么他有很大的概率会被诊断出患病

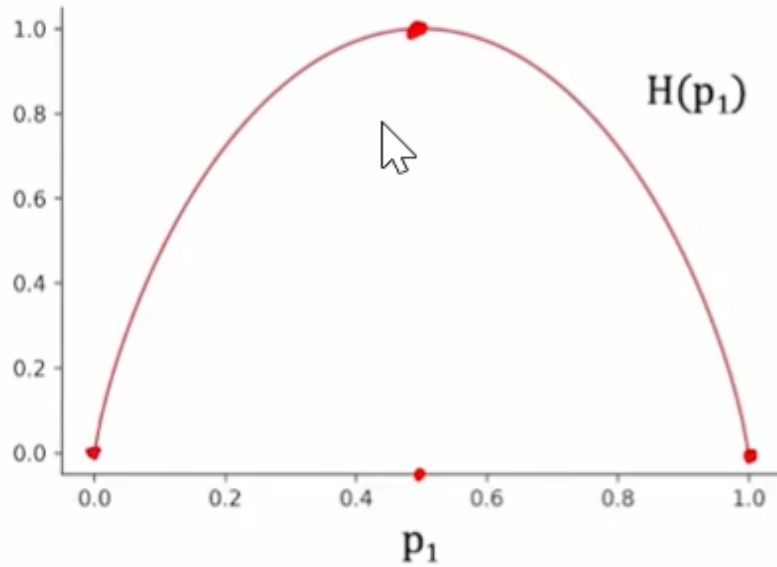
- 二元分类中提高阈值会降低召回率，提高精度
- 降低阈值会降低精度，提高召回率
- 要权衡精确率与召回率，使用F1分数方法(调和平均数)，越高越好  $F1_{SCORE} = \frac{1}{\frac{1}{P} + \frac{1}{R}} = 2 \frac{PR}{P+R}$

## 决策树

- 决定在根节点用什么特征
- 选择能获得最大纯度(区分后都是一个类型)的特征

熵：衡量不纯程度的指标 熵和样品比例的关系

$p_1$  = fraction of examples that are cats



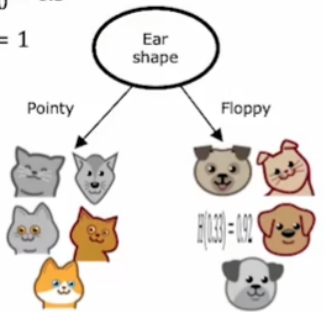
两个类别的特殊情况，多类别参考周志华的西瓜书

$$p_0 = 1 - p_1 \quad H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0) = -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$

熵的减少称为信息增益 使用根节点的熵减去左右分支熵的加权平均(加权的是样本的数量)，可以把左右分支看作是不不断减熵优化的过程。最后要选择结果最大的那个

$$p_1 = 5/10 = 0.5$$

$$H(0.5) = 1$$



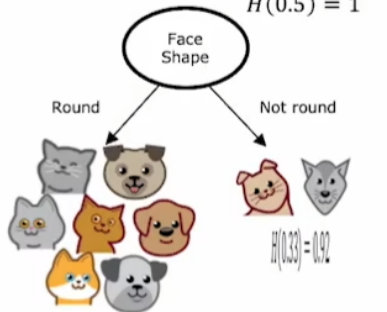
$$p_1 = 4/5 = 0.8 \quad p_1 = 1/5 = 0.2$$

$$H(0.8) = 0.72 \quad H(0.2) = 0.72$$

$$H(0.5) - \left( \frac{5}{10} H(0.8) + \frac{5}{10} H(0.2) \right)$$

$$= 0.28$$

$$H(0.5) = 1$$



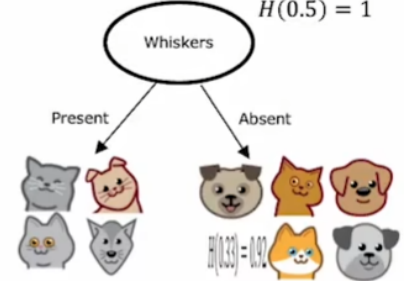
$$p_1 = 4/7 = 0.57 \quad p_1 = 1/3 = 0.33$$

$$H(0.57) = 0.99 \quad H(0.33) = 0.92$$

$$H(0.5) - \left( \frac{7}{10} H(0.57) + \frac{3}{10} H(0.33) \right)$$

$$= 0.03$$

$$H(0.5) = 1$$



$$p_1 = 3/4 = 0.75 \quad p_1 = 2/6 = 0.33$$

$$H(0.75) = 0.81 \quad H(0.33) = 0.92$$

$$H(0.5) - \left( \frac{4}{10} H(0.75) + \frac{6}{10} H(0.33) \right)$$

$$= 0.12$$

$p$ 表示具有正标签的分数， $w$ 表示样本数量占总样本的分数 信息增益计算公式：

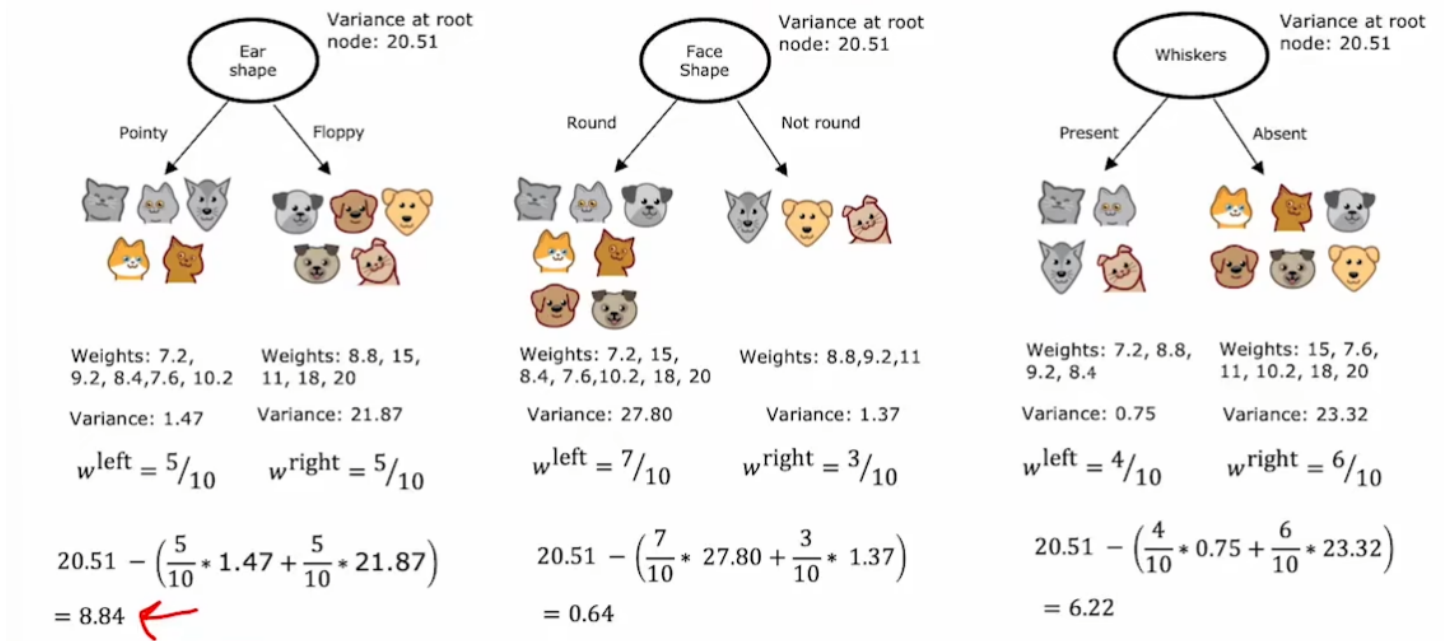
$$H(p_1^{root}) - (w^{left} H(p_1^{left}) + w^{right} H(p_1^{right}))$$

单热编码 如果一个分类特征可以取k个可能的值，那么我们将通过创建k个只能取值0或1的二进制特征来替换他 对于其它特征也可以采用0，1来替换 \* 如果遇到特征是用多个数字表示，那么我们要选择能获得最大信息增益的阈值来进行划分

## 回归树

我们要尝试减少每个数据子集的值Y的权重方差 使用根节点的方差减去左右分支方差的加权平均(加权的是样本的数量), 要选择的是方差减小最多的那个。

## Choosing a split



## 使用多个决策树(树集成)

使用多个树对同一特征进行递归判断, 最后让他们投票, 判断最后结果 ##### 替换采样(有放回采样) \* 在已有数据里面进行随机抽样, 抽取后记录并放回 \* 得到一个新的数据集, 和原有数据集有些不同但大致相同 \* 根据得到的数据集创建新的决策树

## 随机森林算法

- 训练多个决策树, 并将他们合并为一个集合(一般不超过100棵)
- 随机抽取k个特征, 在这k个特征里面找到最大的信息增益并选取他作为划分依据

## XGBoost

- 对分类错误的样本进行刻意的训练, 使得他们能做的更好
- 内置正则化, 防止过拟合
- 一个很方便的库, 可用于分类与回归

## 决策树和神经网络优缺点

- 决策树适用于结构性数据, 即表格化数据, 训练时间短
- 神经网络适用于非结构性数据和结构性数据, 如图片, 音频, 文字等, 训练时间长

## 推荐系统

- 有一定数量的用户和一定数量的项目
- 其目标是向特定用户进行个性化推荐

## 开发算法

- 拥有电影特征和数据集(用户对电影的评分)
- 会为不同的用户设置不同的权重和偏置

参数 \*  $r(i, j)$  为1表示用户参与评分, 反之则没有 \*  $y^{(i,j)}$  表示用户对电影的评分 \*  $w^{(j)}, b^{(j)}$  表示为每个用户设定的参数 \*  $X^{(i)}$  表示电影i的特征向量 \*  $m^{(j)}$  表示用户评价的电影数量

预测:  $w^{(j)} \cdot X^{(i)} + b^{(j)}$

成本函数(对于所有用户): 包括了正则化系数

To learn parameters  $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$  for all users :

$$J\left(\begin{matrix} w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \end{matrix}\right) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \underbrace{(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2}_{f(x)} + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

### 协同过滤算法

- 根据用户的评分来推荐
- 在开发中我们往往不知道电影特征的参数值, 但我们知道用户的评分
- 根据w值进行特征值的选取, 即使用  $w \cdot x + b$  去拟合用户评分, 选取恰当的x特征值

电影特征的成本函数

→ To learn  $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ :

$$J(x^{(1)}, x^{(2)}, \dots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

将他们放在一起, 成本函数是w, b, x的函数

Put them together:

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

总和和这里的总和是

梯度下降



$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, x)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w, b, x)$$

## 泛化到二进制标签

- 在用户在网站进行操作的时候分配二进制标签
- 使用逻辑回归中的sigmoid函数
- 损失函数使用二元交叉熵成本函数

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1 - y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

## 均值归一化

有利于对未评级的用户做出良好的预测

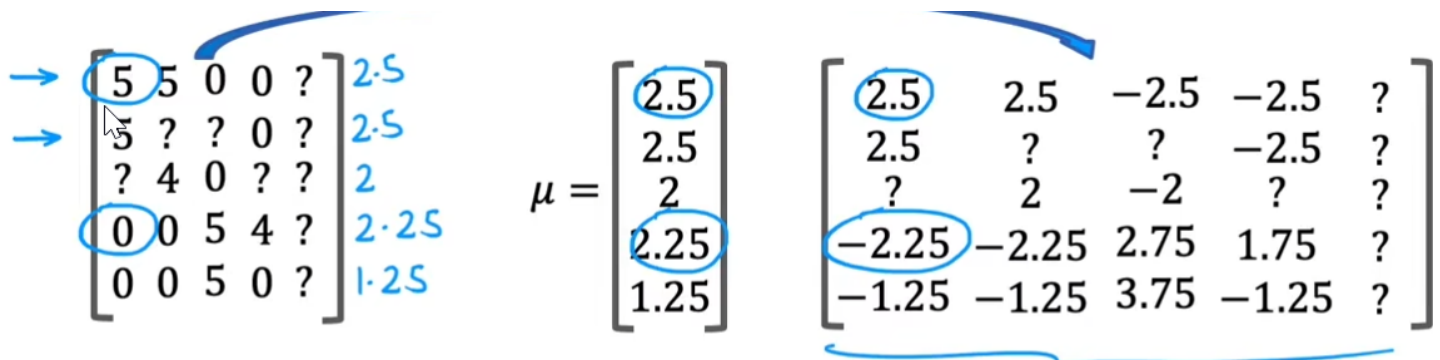
Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

Eve 做出良好预测的前提

\* 对于未评级的用户，其前面的损失函数为0，因此w和x会为0，这会导致预测值wx+b为0

解决方法：



For user  $j$ , on movie  $i$  predict:

$$w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$$

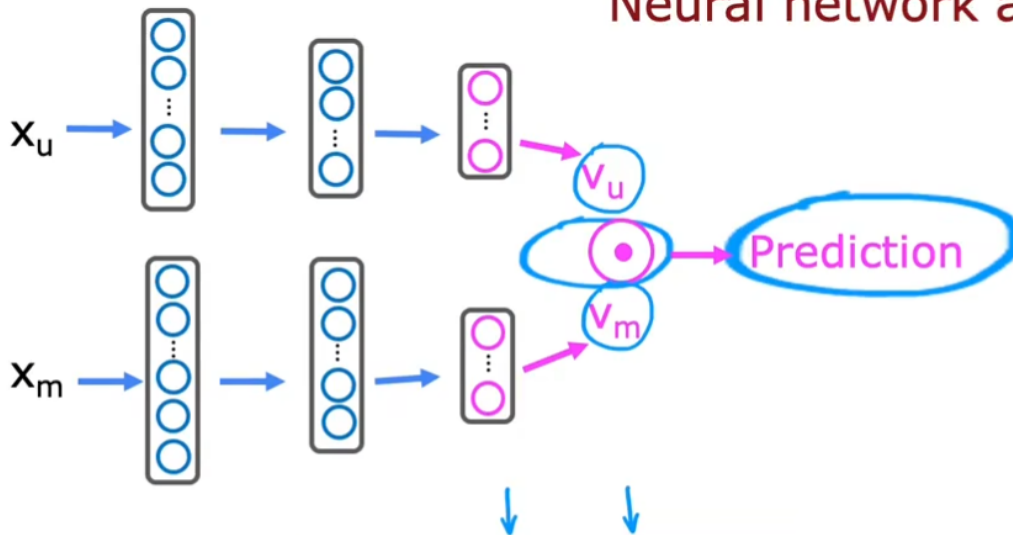
$$y^{(i,j)}$$

\* 将电影的评分放在一个矩阵中，计算每一部电影的评分均值得到一个列向量 \* 让评分分别减去对应电影的均值得到新的矩阵 \* 此时预测函数为  $w x + b + \mu$ ,  $\mu$  为电影均值，这样未评级的用户就会被预测较为正常的值

### 基于内容的过滤

根据用户的特征和物品特征向你推荐物品，需要特征来决定哪些项目可能和用户相匹配 \* 根据用户的特征来进行推荐，但用户特征和物品特征大小很可能不一样 \* 拟合的模型： $v_u^{(j)} \cdot v_m^{(j)}$ ，通过这两个特征来预测用户的评分 \* 使用神经网络进行拟合

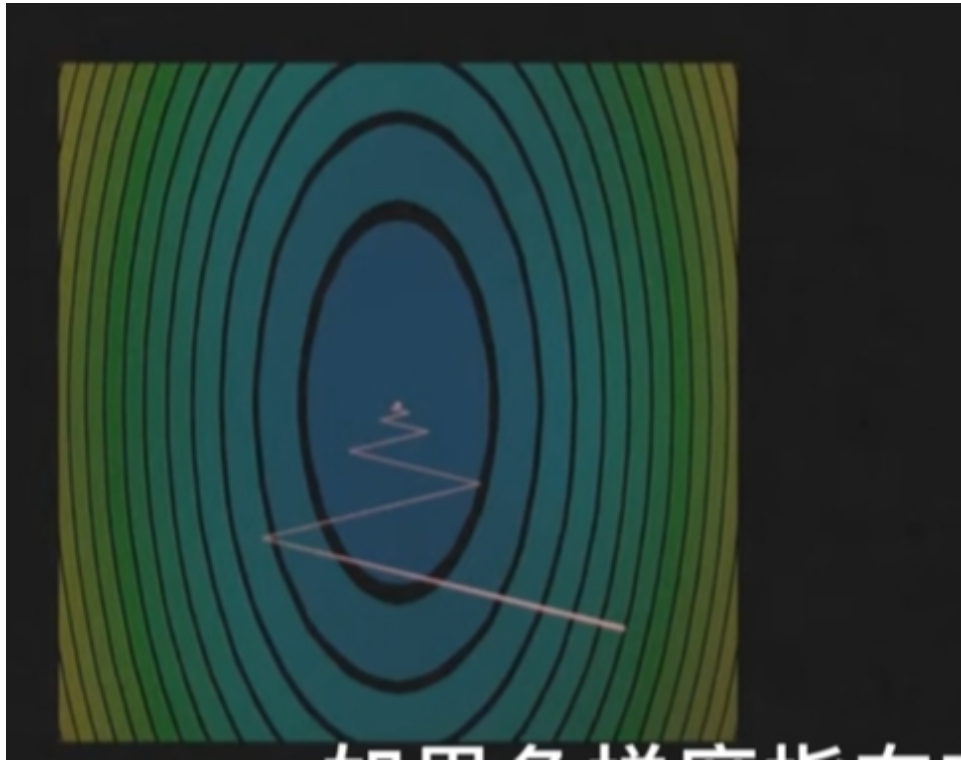
### Neural network architecture



Cost function  $J = \sum_{(i,j):r(i,j)=1} (v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2 + \text{NN regularization term}$

### Momentum 梯度下降优化算法

- 在常规的SGD中，梯度的下降效率并不是那么高，其会在某一方向上不断震荡，导致下降的速度慢，并且可能会陷入鞍点无法逃出



alt text

- 我们在常规的梯度下降中加入一个超参数 $\beta$ ，这个值一般取0.9
- 我们在这里使用指数加权平均，这个方法可以让之前的梯度占取一部分的权重，并且离该点越近权重越大，使得当前的下降方向不完全由当前梯度决定，使得震荡不那么剧烈。并且在陷入鞍点的时候，之前的梯度的累加可以帮助其不陷入鞍点

- $$v_t = \beta v_{t-1} + (1 - \beta) \frac{\partial J}{\partial w} w = w - \alpha v_t (\alpha \text{为学习率})$$

