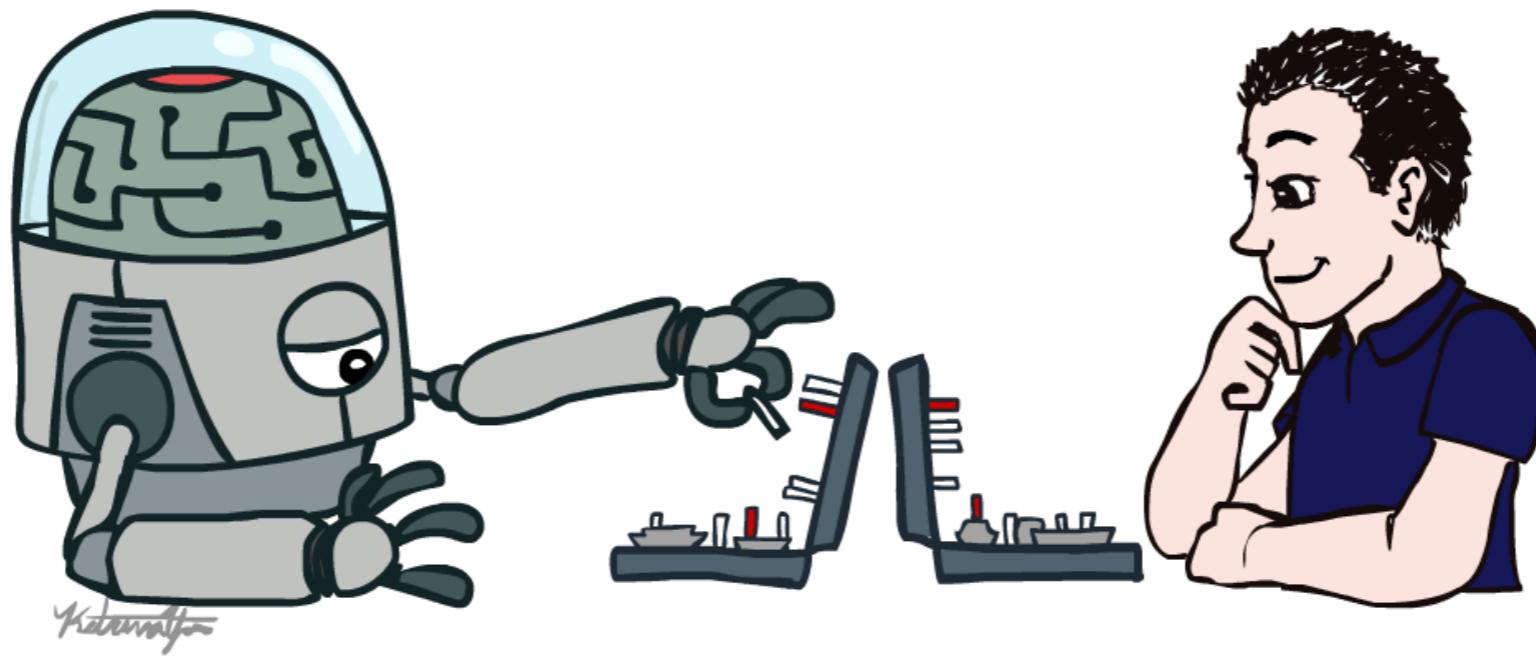


# 人工智能

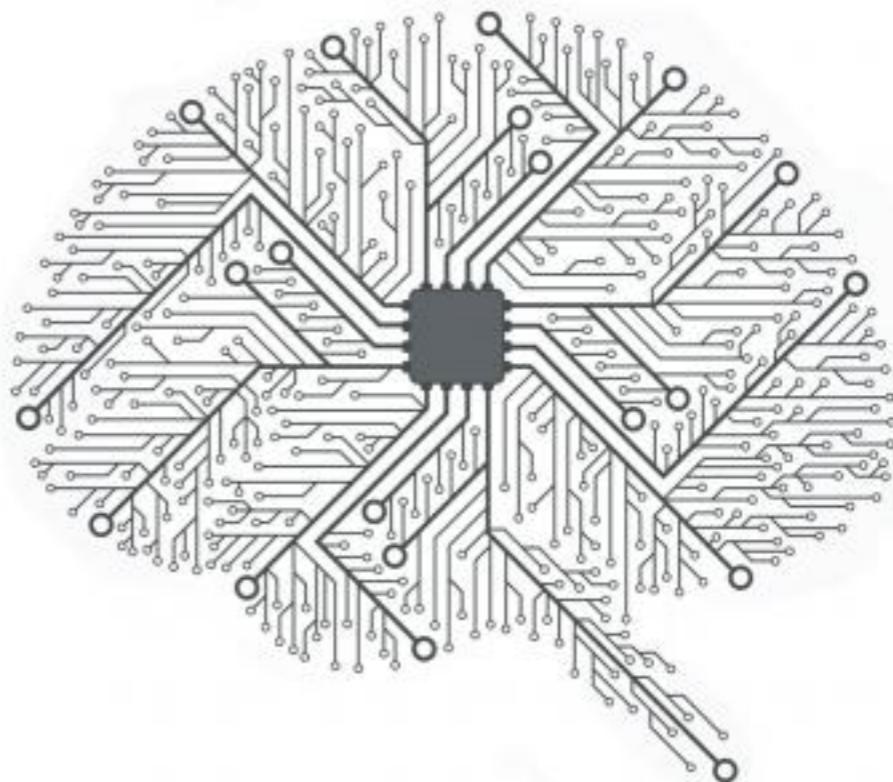
---



# 第九章·感知机与神经网络

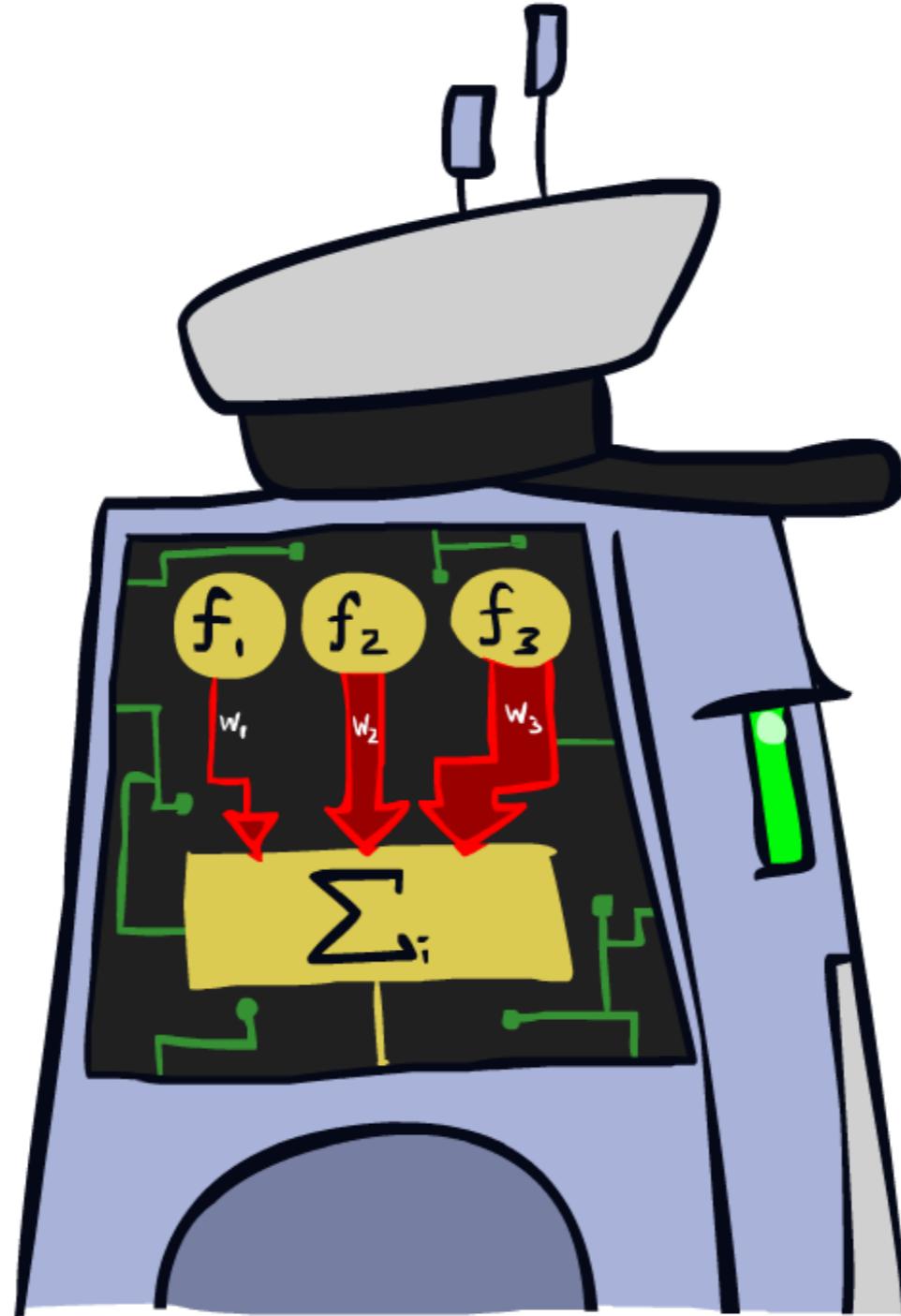
---

- 线性分类器
- 感知机
- 确定性决策与概率决策
- 逻辑回归



# 线性分类器

---



# 特征向量

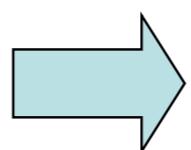
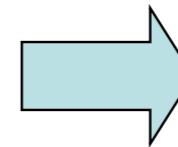
---

$x$

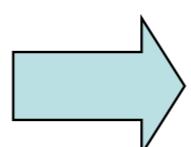
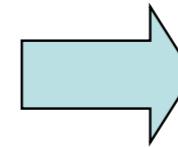
$f(x)$

$y$

Hello,  
Do you want free print  
cartridges? Why pay more  
when you can get them  
ABSOLUTELY FREE! Just


$$\begin{cases} \# \text{ free} & : 2 \\ \text{YOUR\_NAME} & : 0 \\ \text{MISSPELLED} & : 2 \\ \text{FROM\_FRIEND} & : 0 \\ \dots \end{cases}$$


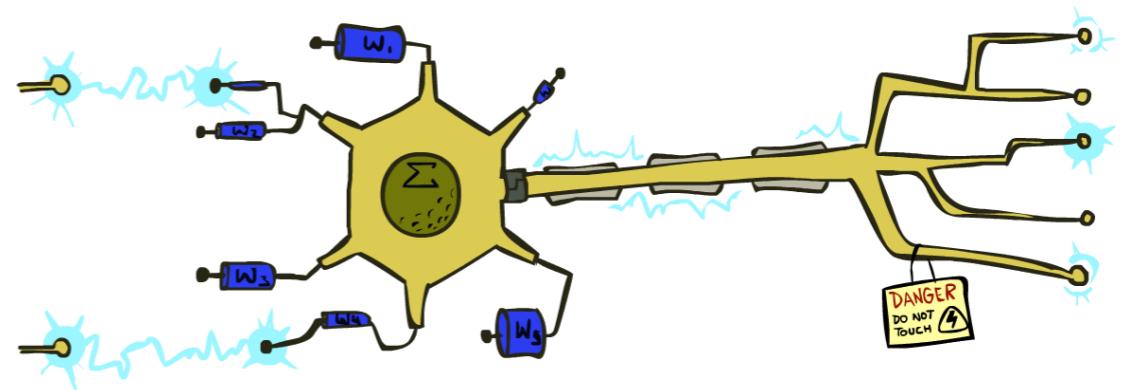
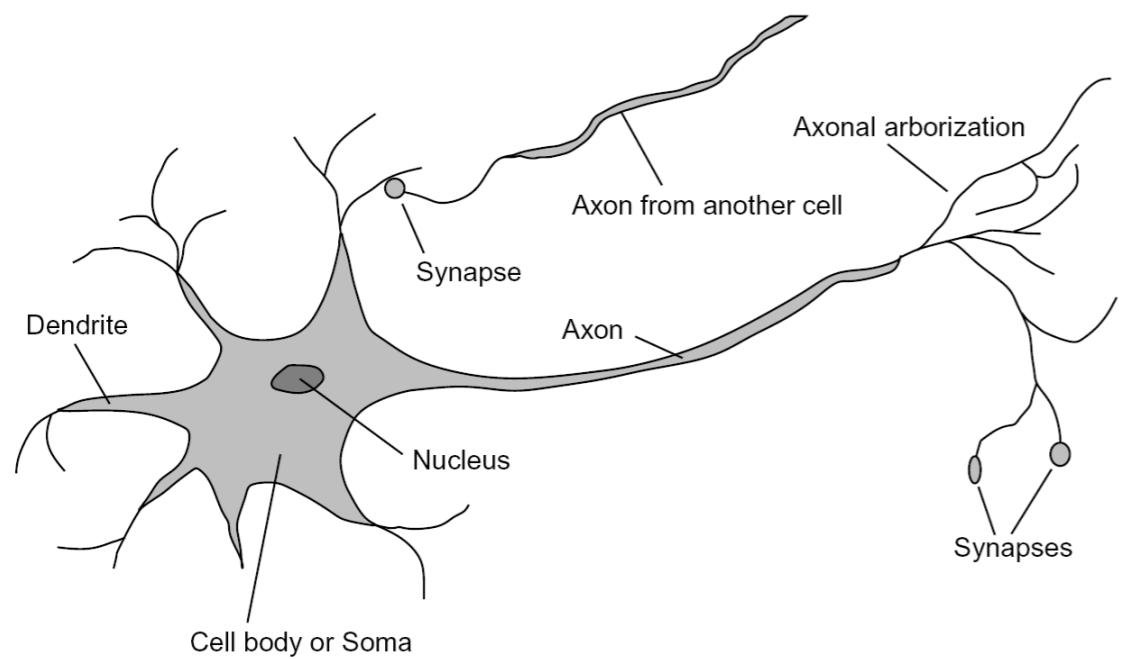
**SPAM**  
or  
**HAM**


$$\begin{cases} \text{PIXEL-7,12} & : 1 \\ \text{PIXEL-7,13} & : 0 \\ \dots \\ \text{NUM\_LOOPS} & : 1 \\ \dots \end{cases}$$


“2”

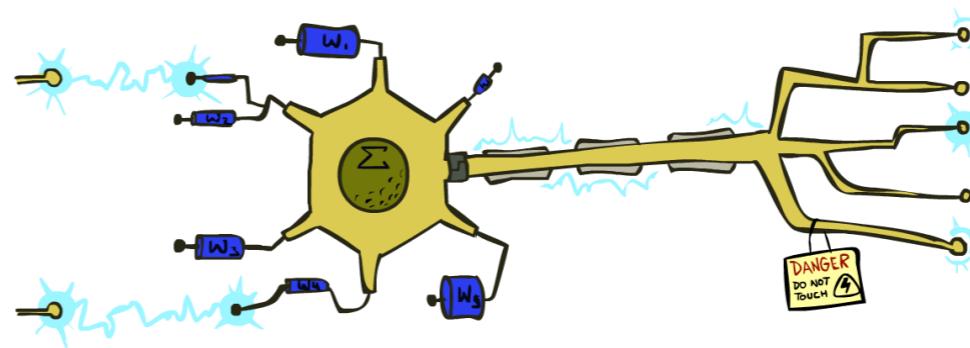
# 人类神经元

---



# 线性分类器

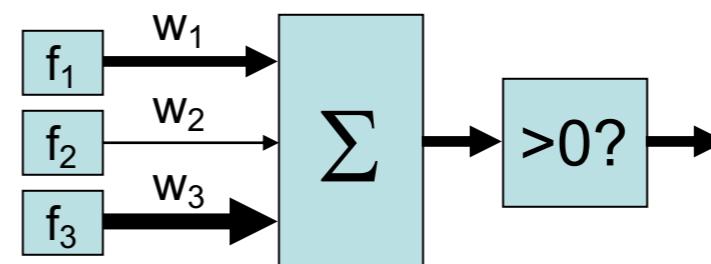
- 输入是特征值
- 每个特征都有权重
- 激活函数是求和



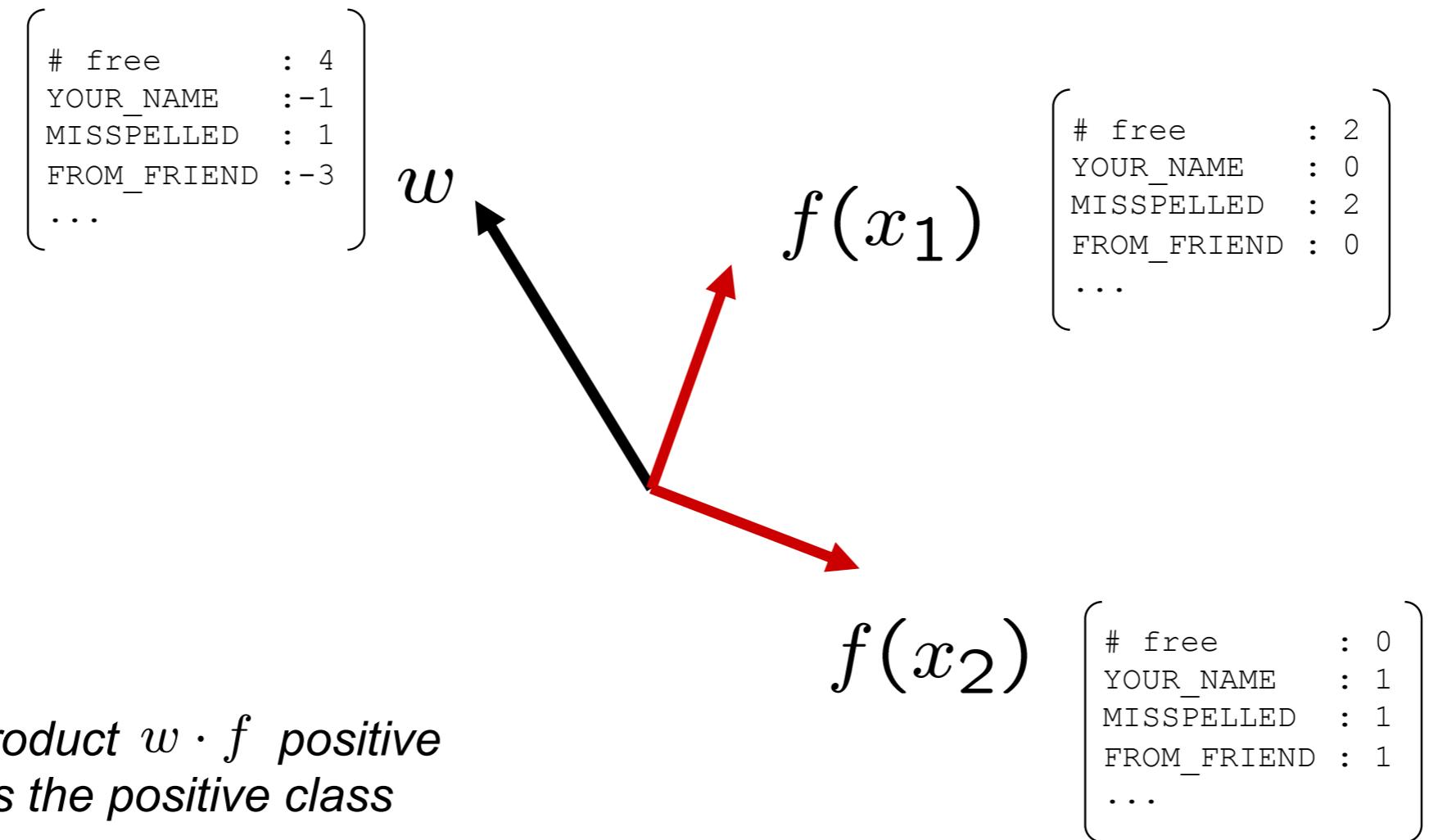
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- 如果激活函数的值是:

- 正数, 那么输出 +1
- 负数, 那么输出 -1



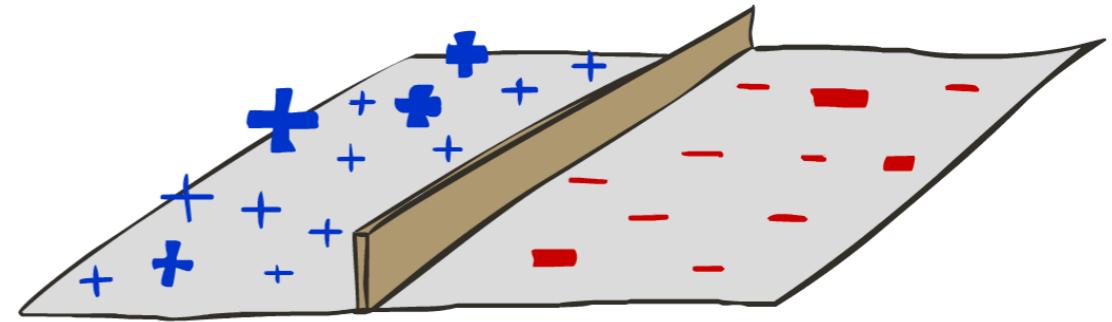
# 权重



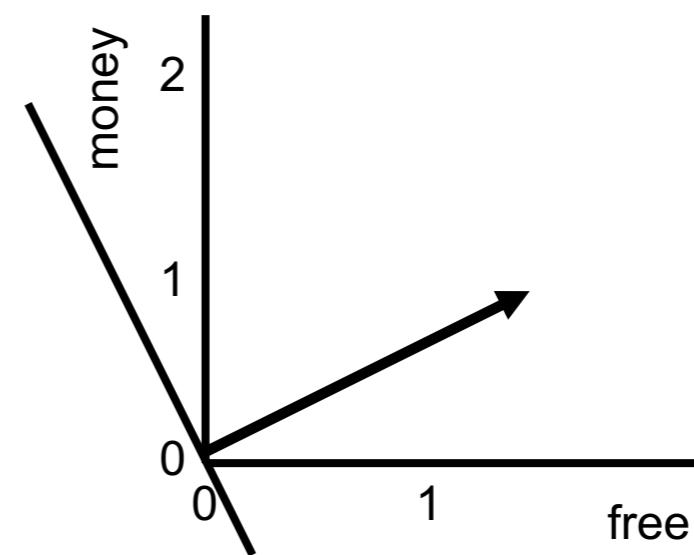
- 学习：从训练样本中得到合适的权重向量

# 二元决策

- 在特征向量空间中
  - 学习样本是一些点
  - 权重向量确定了一个超平面 $\omega^T \cdot f(x) = 0$
  - 这个超平面的一个侧面对应 $Y=+1$
  - 另一个对应于 $Y=-1$

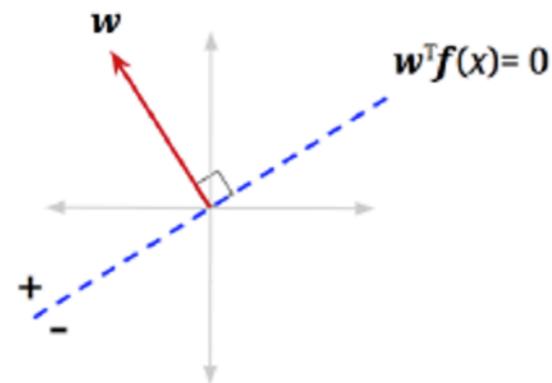


$$\begin{matrix} w \\ \hline \text{BIAS} & : & -3 \\ \text{free} & : & 4 \\ \text{money} & : & 2 \\ \dots & & \end{matrix}$$

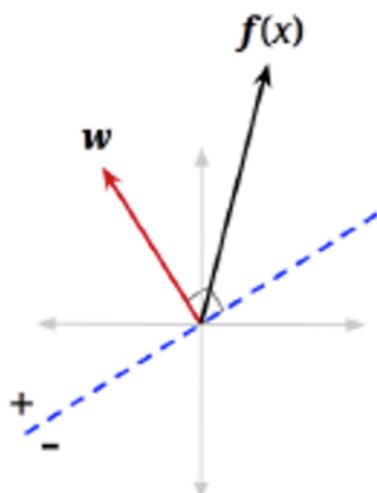


# 二元决策

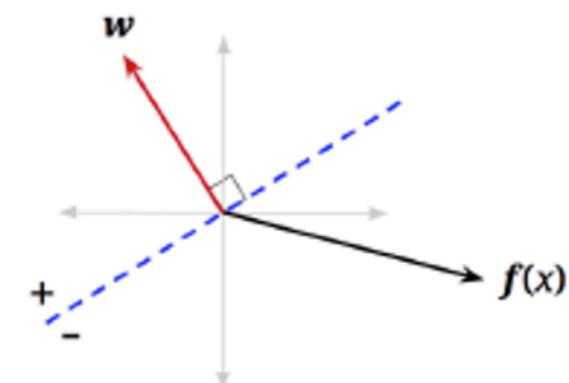
---



Decision Boundary



$x$  classified into positive class



$x$  classified into negative class

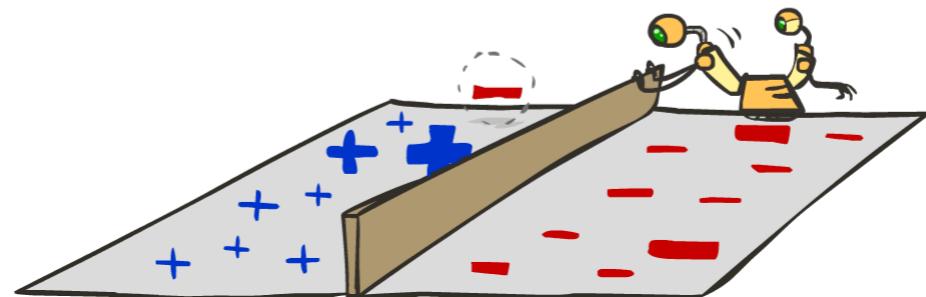
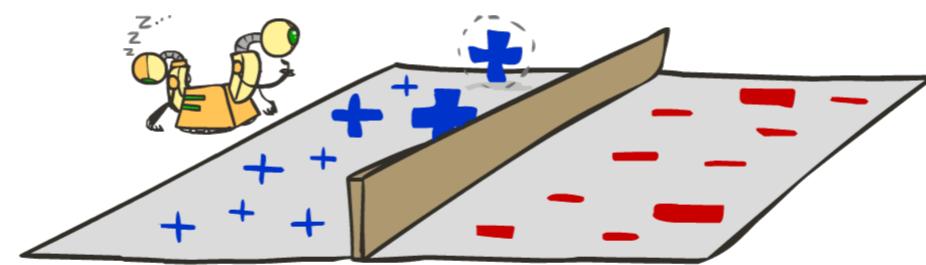
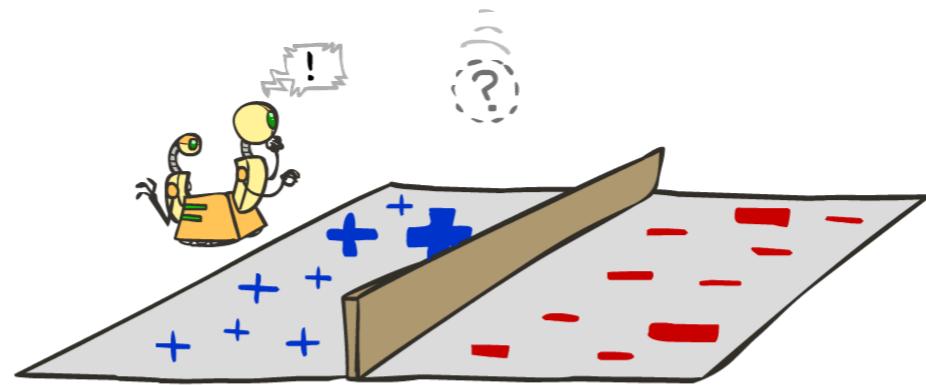
# 权重学习

---



# 学习：二进制感知机

- 开始时权重向量weights = 0
- 对每个训练样本：
  - 按当前权重分类
  - 如果正确 ( $y=y^*$ ), 不作处理!
  - 如果错误: 修正权重向量



# 学习：二进制感知机

---

- 开始时 权重向量weights = 0

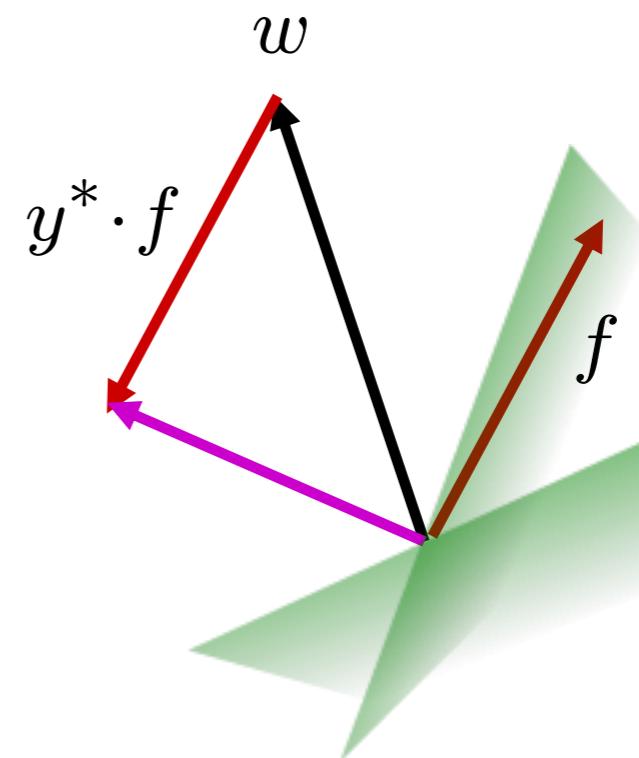
- 对每个训练样本：

- 按当前权重分类

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

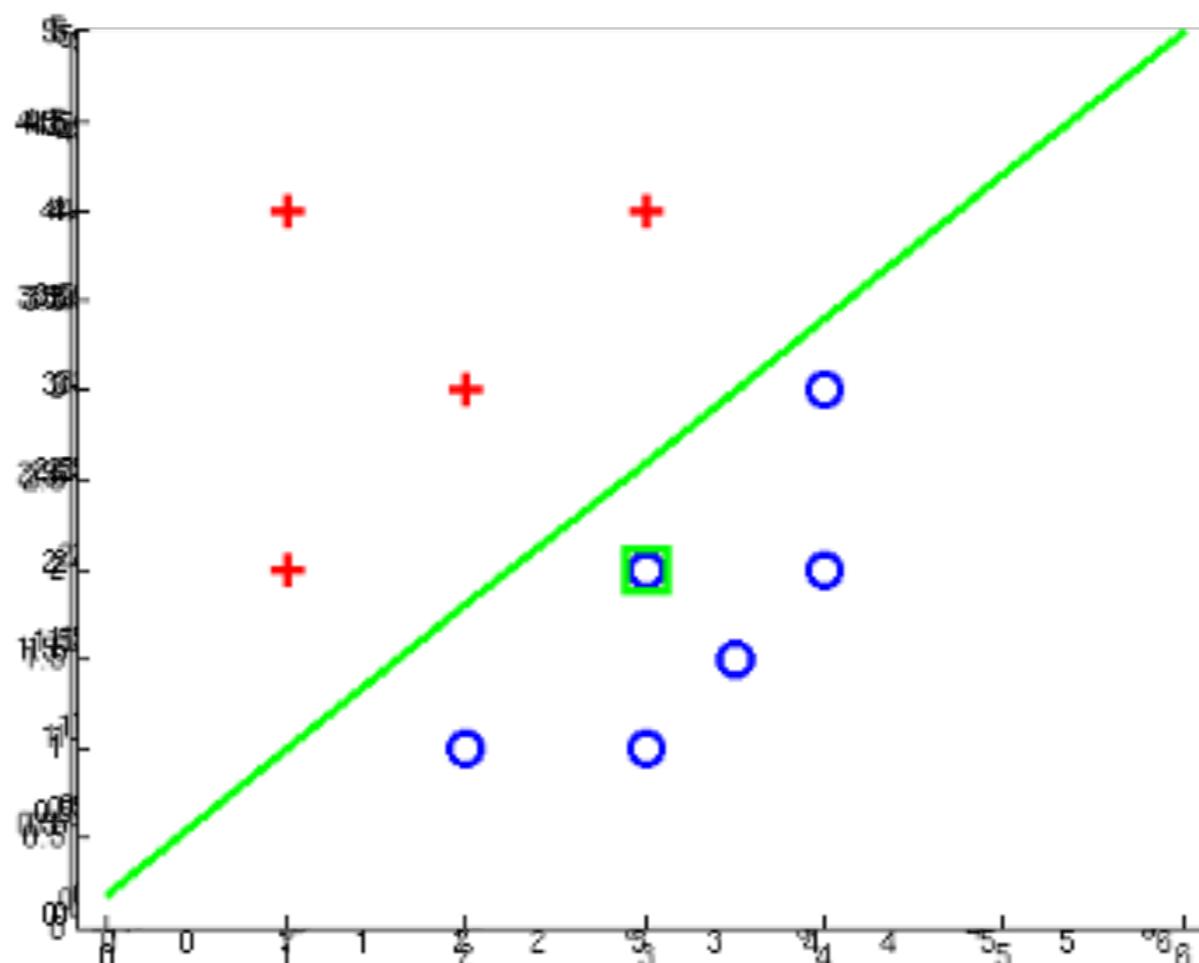
- 如果正确 ( $y=y^*$ ), 不作处理!
    - 如果错误: 权重向量加上或减去特征向量

$$w = w + y^* \cdot f$$



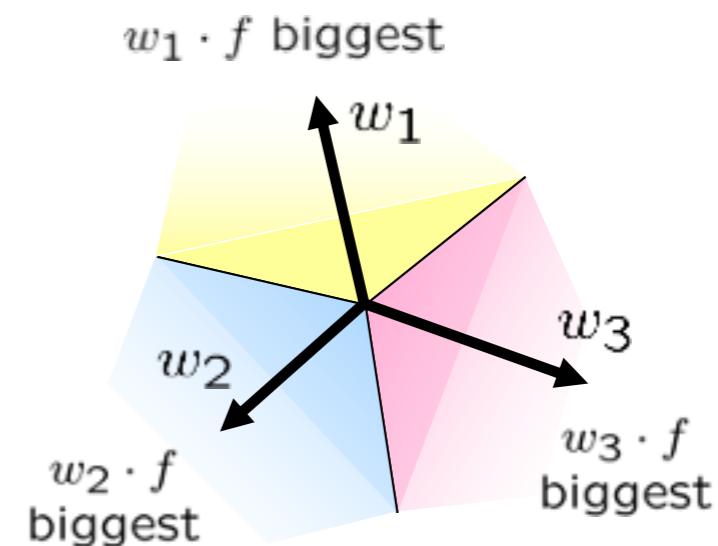
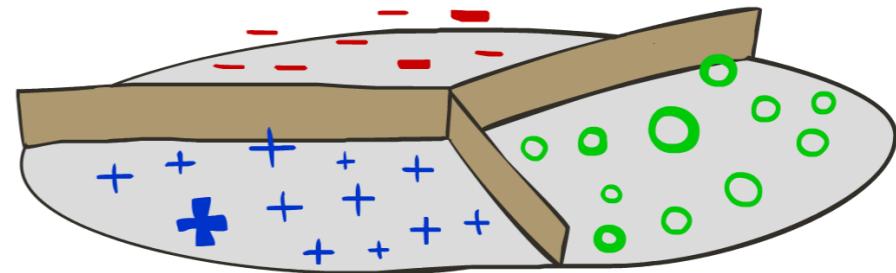
## 示例：感知机

---



# 多类别决策

- 如果有多个类别
    - 每个类的权重向量： $w_y$
    - 计算样本在每个y类的得分： $w_y \cdot f(x)$
    - 将样本判断为最高得分的类别
- $$y = \arg \max_y w_y \cdot f(x)$$



*Binary = multiclass where the negative class has weight zero*

# 学习：多类别感知机

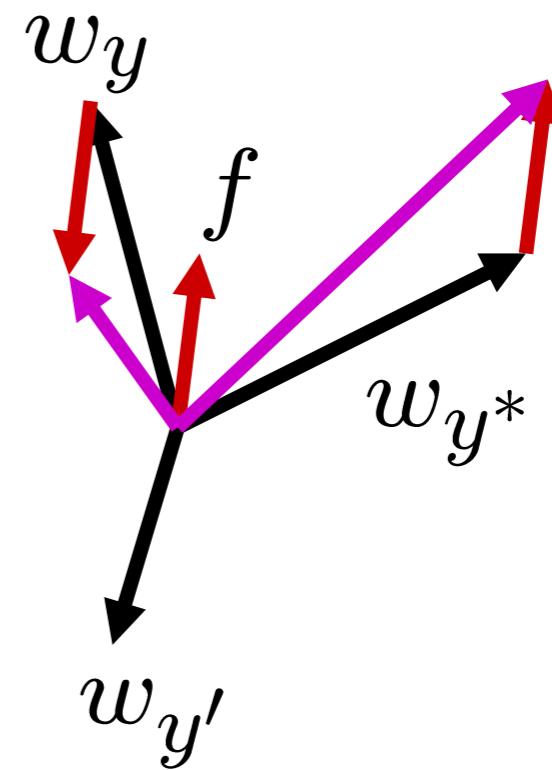
- 开始时 所有的权重向量weights = 0
- 对每个训练样本：
  - 用当前权重进行预测

$$y = \arg \max_y w_y \cdot f(x)$$

- 如果正确，不作处理！
- 如果错误：降低错误类别的分数，增加正确类别的分数

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



# 示例：多类别感知机

---

“win the vote”

“win the election”

“win the game”

$w_{SPORTS}$

```
win    : 0
game   : 0
vote   : 0
election : 0
the    : 0
...
```

$w_{POLITICS}$

```
win    : 0
game   : 0
vote   : 0
election : 0
the    : 0
...
```

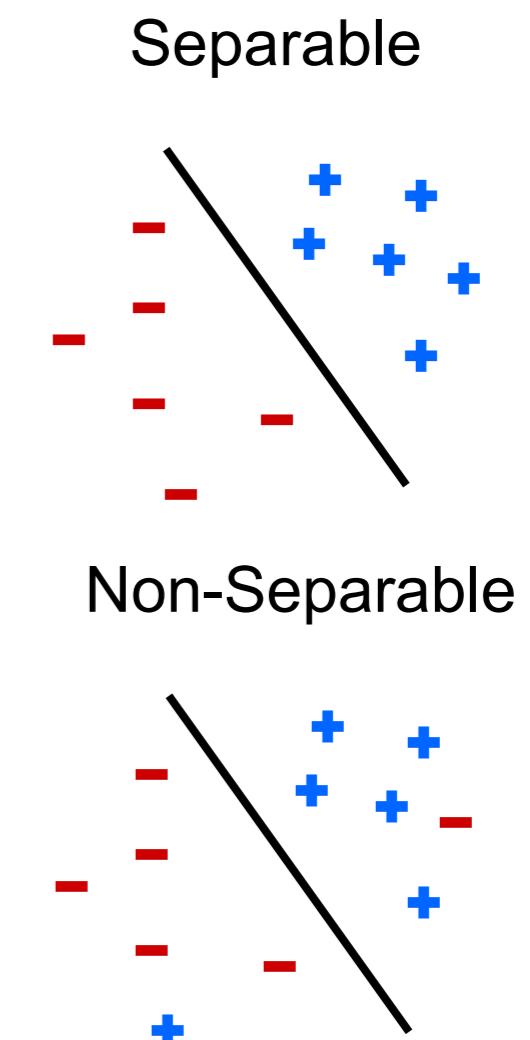
$w_{TECH}$

```
win    : 0
game   : 0
vote   : 0
election : 0
the    : 0
...
```

# 感知机的性质

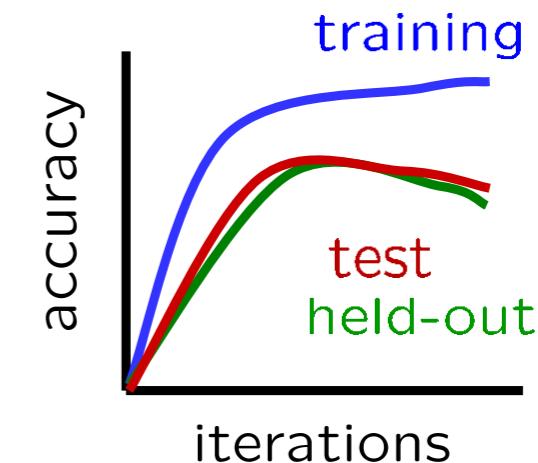
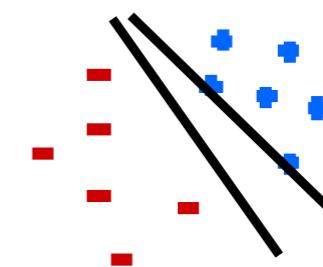
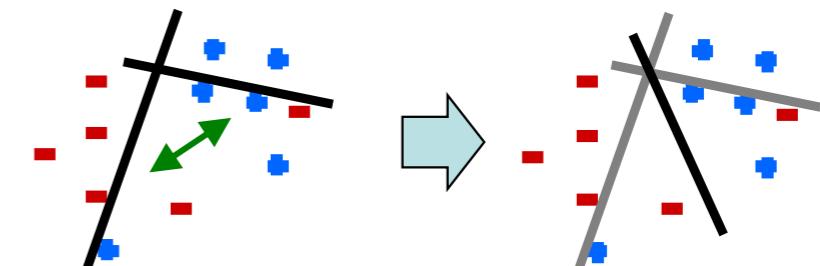
- 可分性：是否存在某些参数使训练集完全正确
- 收敛性：如果训练样本是可分的，则感知机最终会收敛（二分类情况下）
- 错误界限：在训练集上最多会犯多少次错误

$$\text{mistakes} < \frac{k}{\delta^2}$$



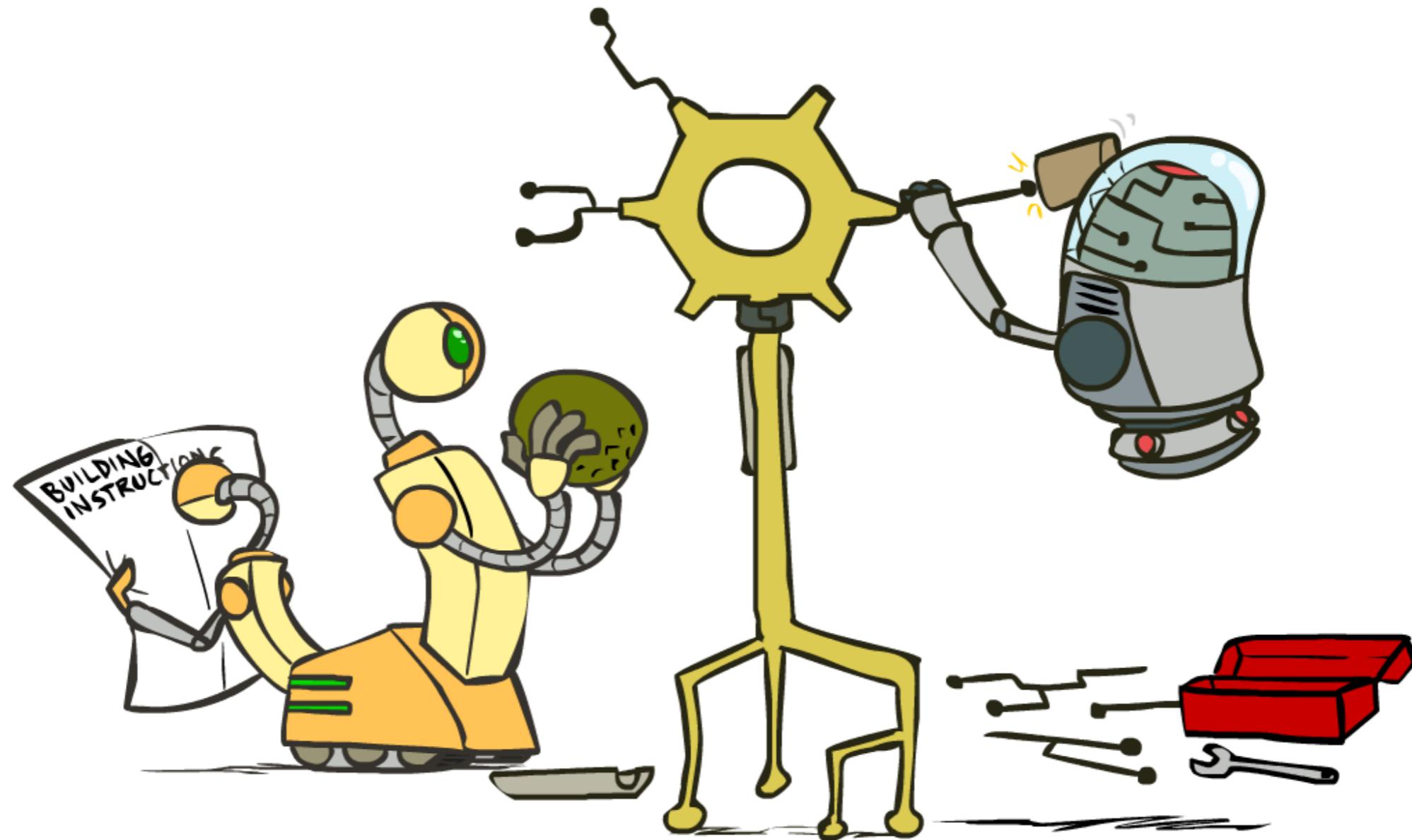
# 感知机的问题

- 噪声：如果数据不可分离，权重可能会剧烈振荡：
  - 时间差分学习会降低权重的振荡
- 泛化能力差：只能找到一个把正负样本“勉强”分开的解决方案
- 过度训练：测试/保持的准确性通常先上升后下降
  - 过度训练是一种过拟合

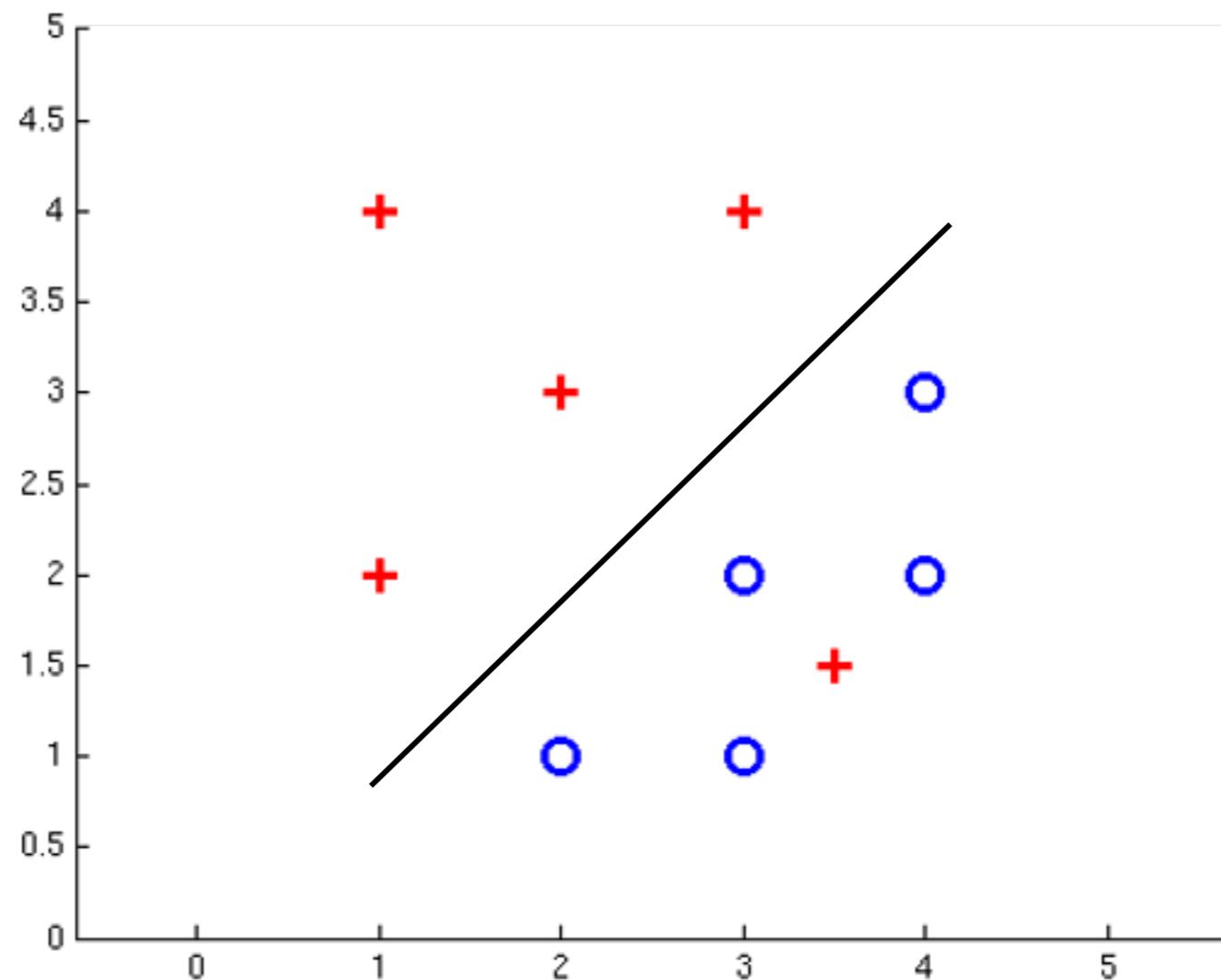


# 改进感知机

---



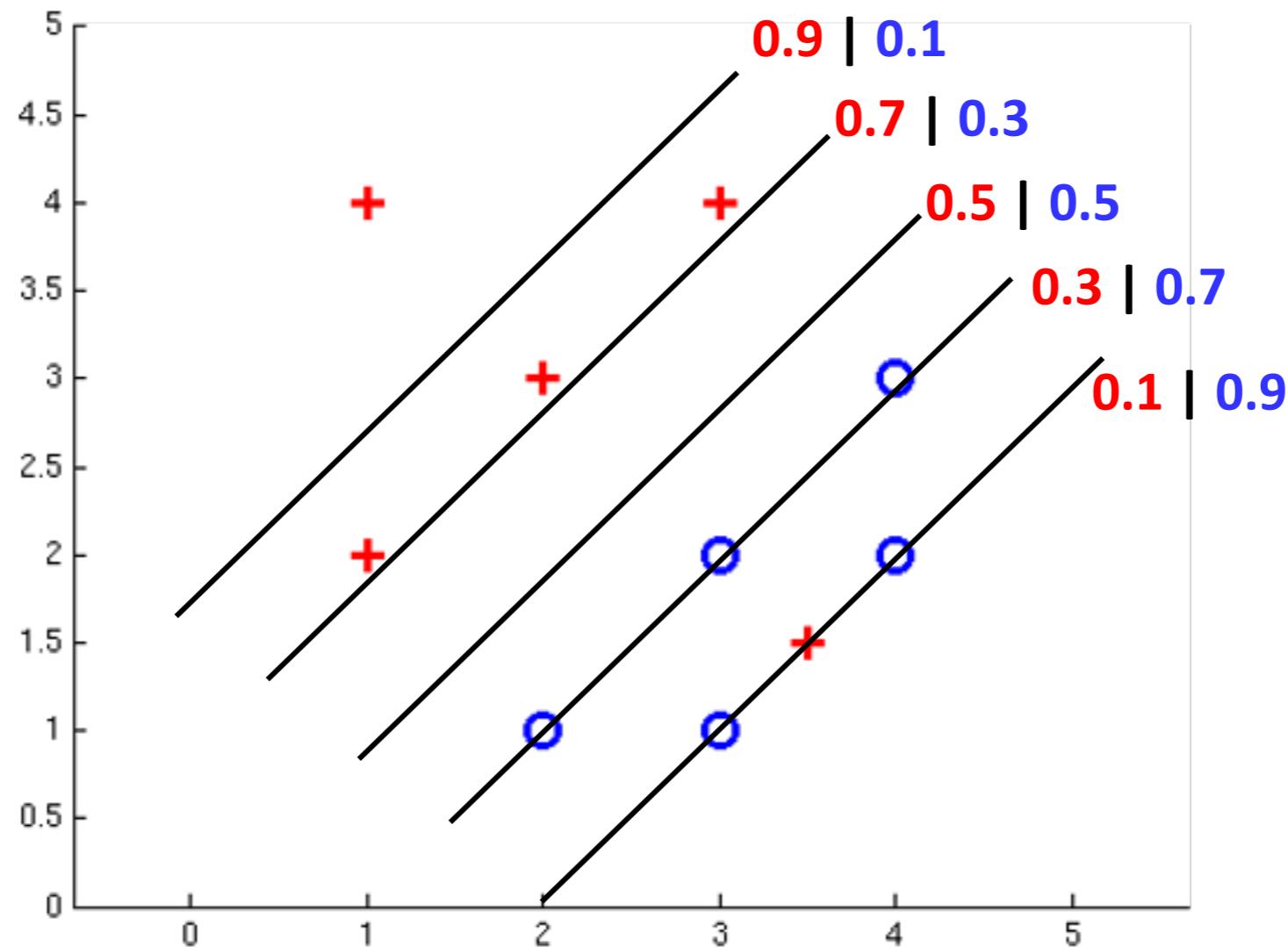
# 不可分情形：确定性决策



即使是最  
好的线性  
边界也至  
少会犯一  
个错误

# 不可分情形：概率决策

---

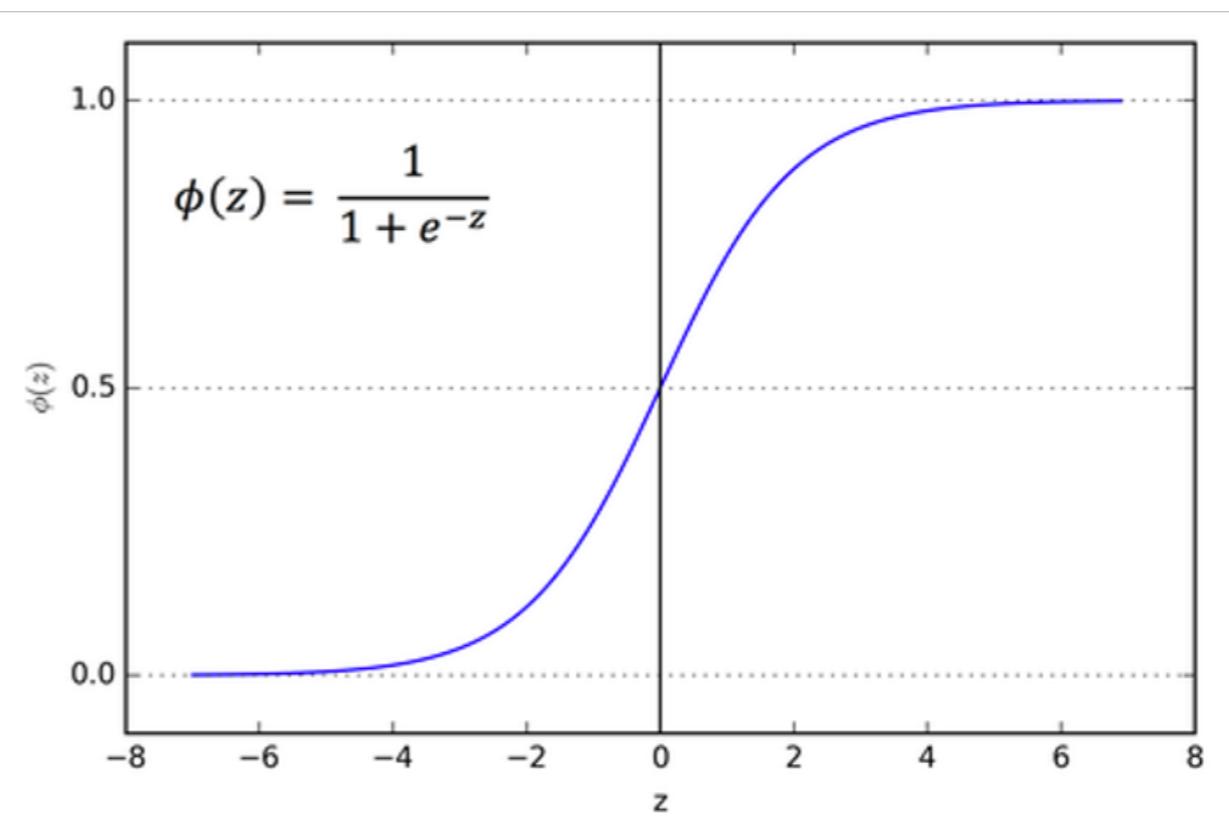


# 如何获得概率决策?

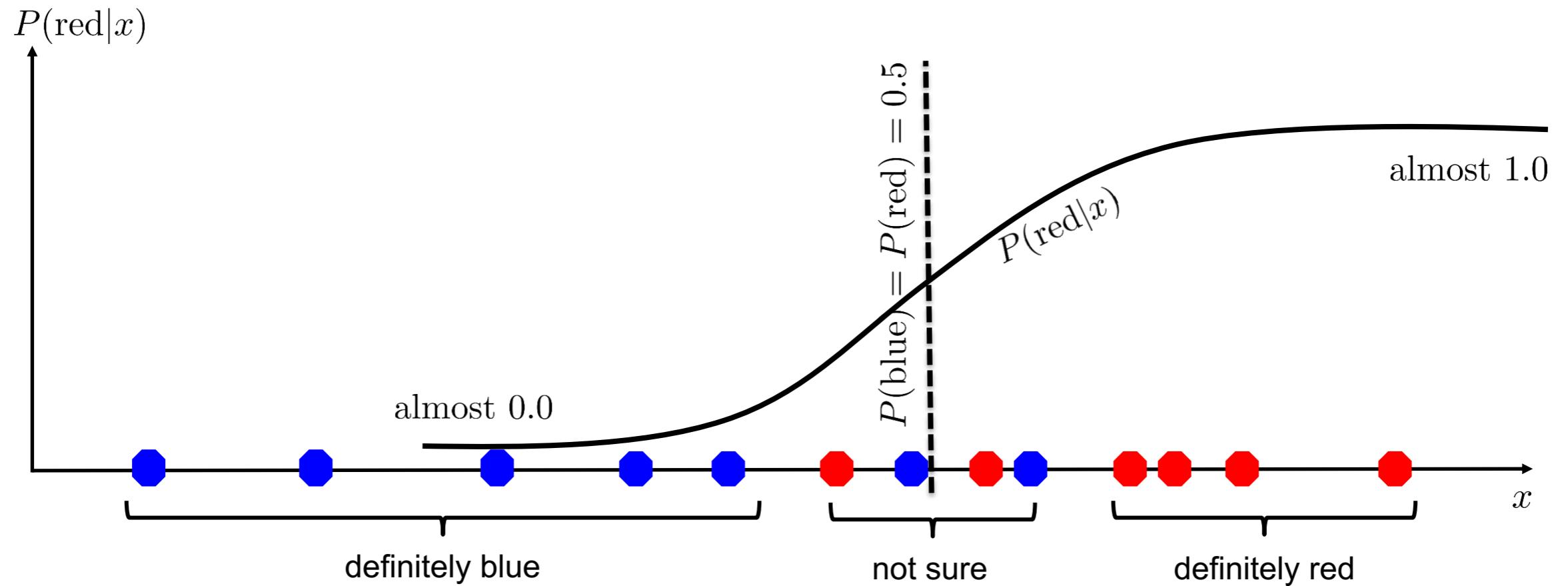
---

- 感知机评分:  $z = w \cdot f(x)$ 
  - 如果  $z = w \cdot f(x)$  是很大的正数, 那么希望它的值趋近于1
  - 如果  $z = w \cdot f(x)$  是很大的正数, 那么希望它的值趋近于0
- Sigmoid 函数

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



# Sigmoid 二分类的例子



$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

概率呈指数增长  
归一化

# 求解最佳权重

---

- 最大似然估计：

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

- 其中

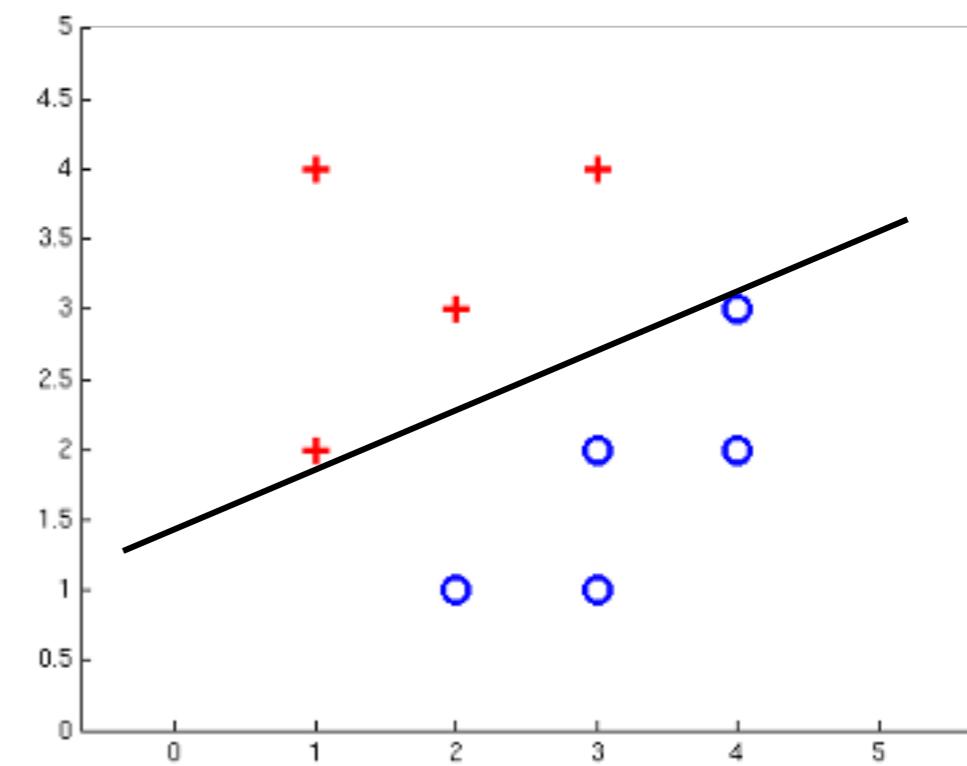
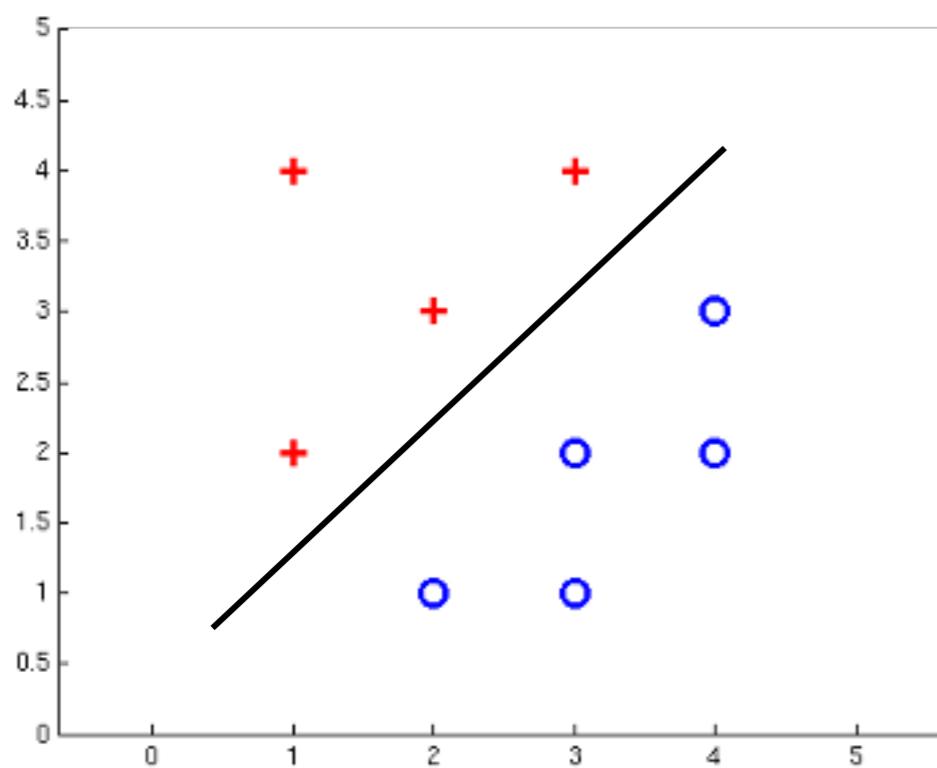
$$P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

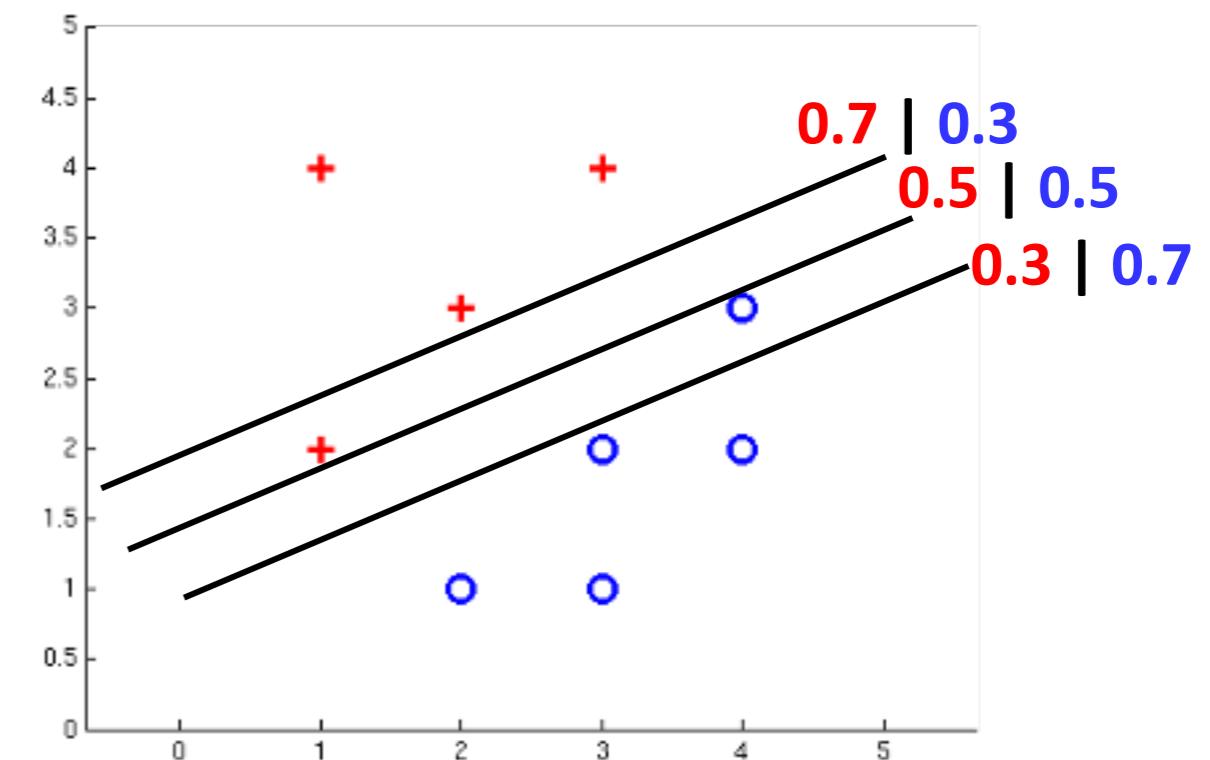
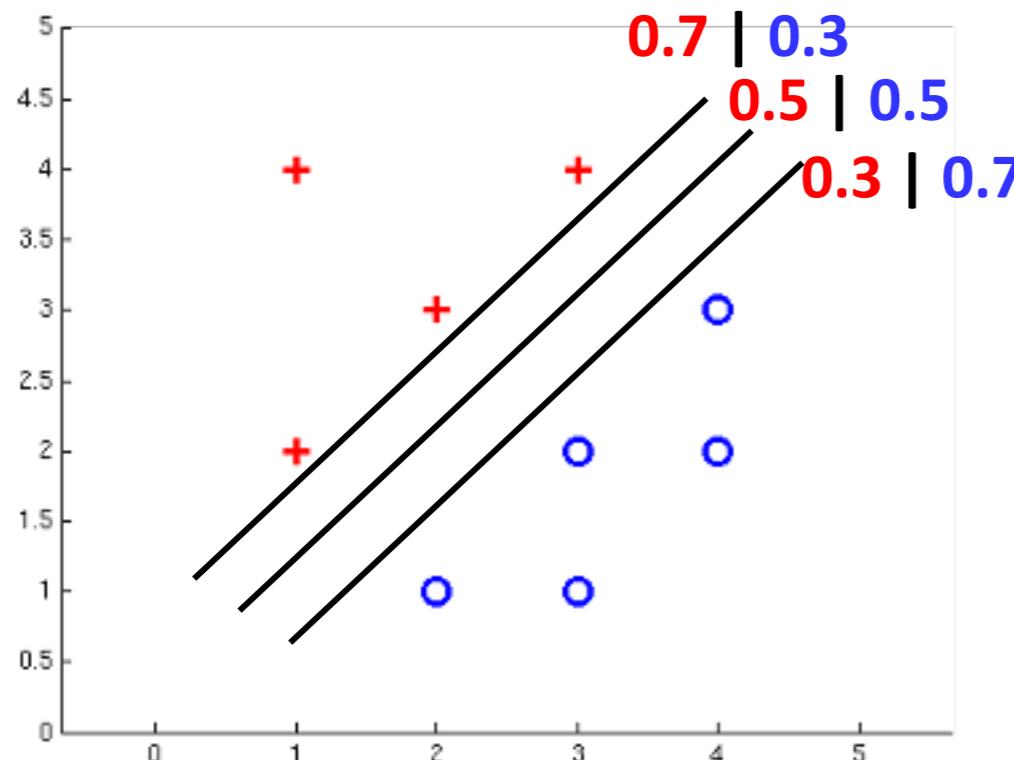
- 这就是逻辑回归 Logistic Regression

# 可分离案例：确定性决策-多种可能的选择

---



# 可分离案例：概率决策-明确的偏好



# 多类别的逻辑回归

---

- 对于感知机：

- 每个类的权重向量：

$$w_y$$

- 计算样本在每个y类的得分：

$$w_y \cdot f(x)$$

- 将样本判断为最高得分的类别

$$y = \arg \max_y w_y \cdot f(x)$$

- 如何将分数转化为概率？

$$z_1, z_2, z_3 \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

original activations

# 求解最佳权重

---

- 最大似然估计：

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

- 其中

$$P(y^{(i)} | x^{(i)}; w) = \frac{e^{w_y \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

- 这就是多分类逻辑回归 Logistic Regression

# 优化问题

---

- 如何求解

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

# 爬山算法

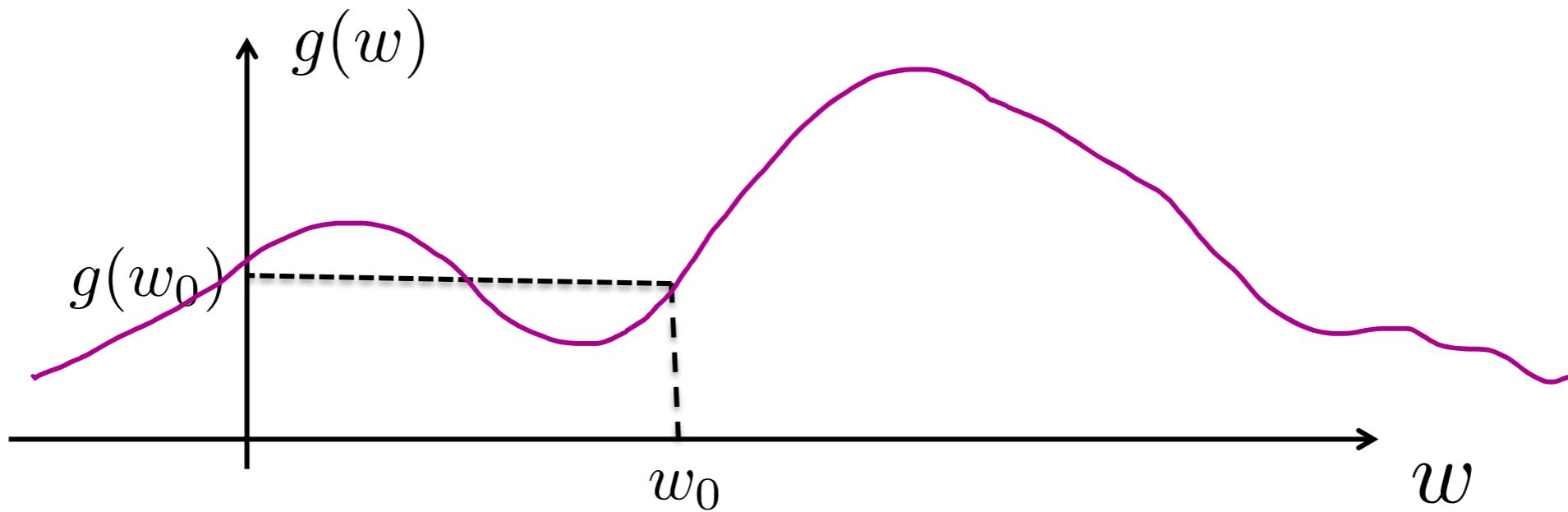
---

- CSPs中的爬山算法
  - 随机选取开始状态
  - 重复：移动到最佳邻近状态
  - 如果没有比当前更好的邻居，算法结束
- 如何应用到逻辑回归上?
  - 在连续空间上进行优化
  - 无限多个邻居



# 一维优化

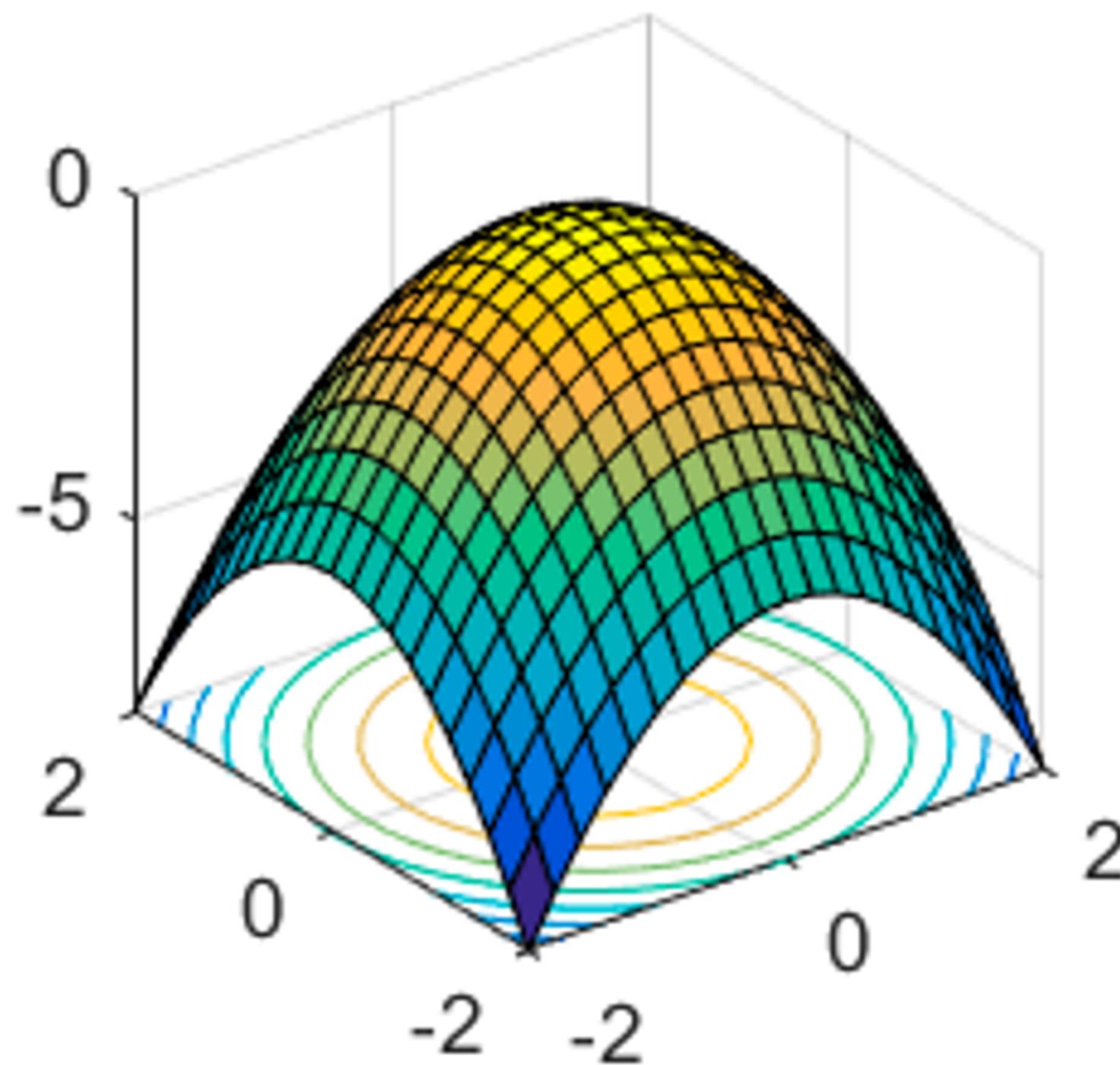
---



- 可以计算  $g(w_0 + h)$  和  $g(w_0 - h)$ 
  - 然后朝最佳方向前进
- 或者, 求值导数:  $\frac{\partial g(w_0)}{\partial w} = \lim_{h \rightarrow 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$ 
  - 告诉我们要进入哪个方向

## 二维优化

---



# 梯度上升

---

- 在上山方向为每个坐标执行更新
- 坡度越陡 (即导数越高) , 该坐标的步长越大
- 例如, 考虑:  $g(w_1, w_2)$ 
  - 更新:
  - 矢量表示法的更新:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

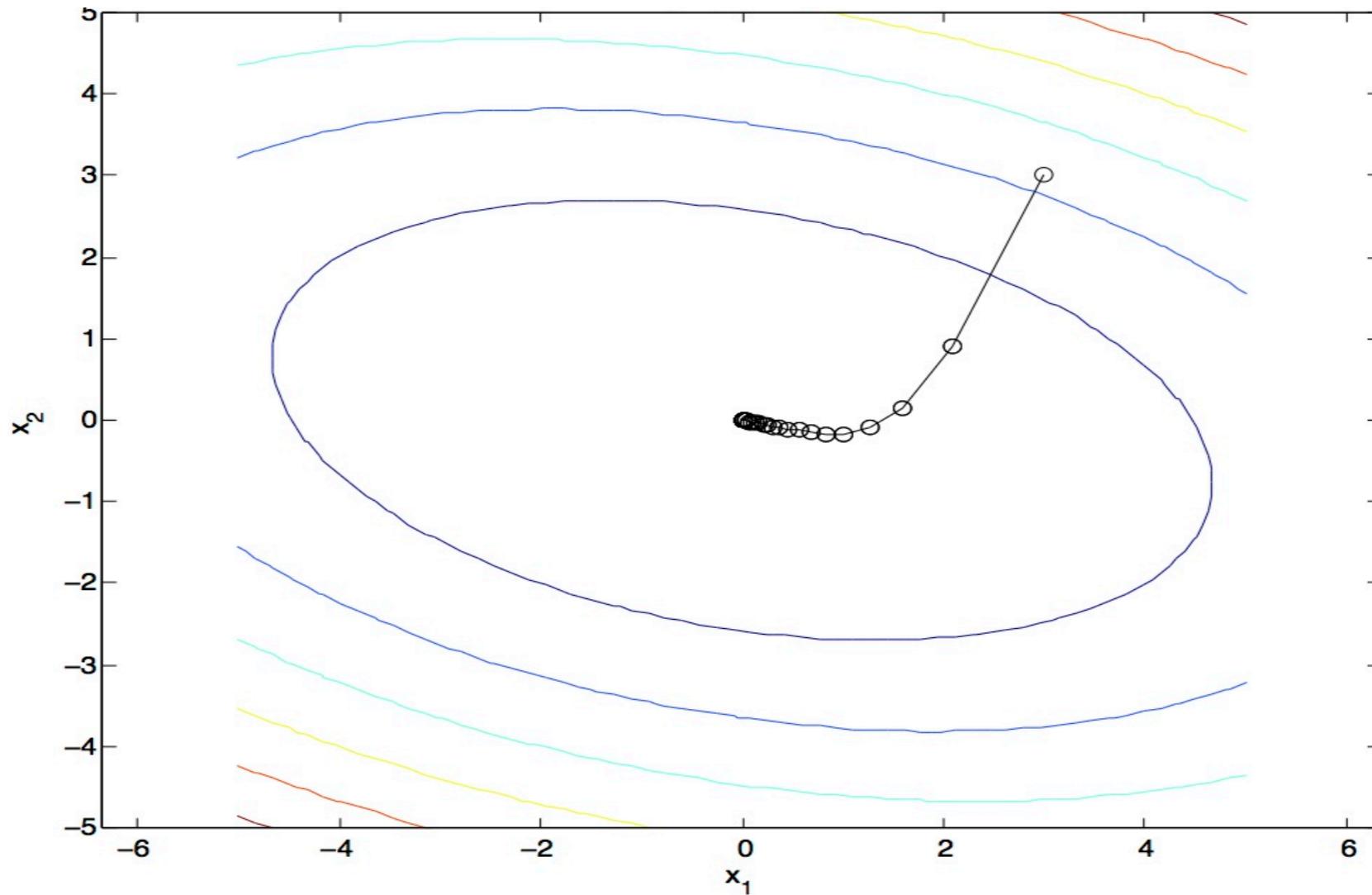
$$w \leftarrow w + \alpha * \nabla_w g(w)$$

- 其中梯度为  $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$

# 梯度上升

---

- 随机选取开始状态
- 重复：向着梯度的方向前进



# 最陡的方向是什么?

---

$$\max_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \varepsilon} g(w + \Delta)$$

- 一阶泰勒展开:  $g(w + \Delta) \approx g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$
- 最陡上升方向:  $\max_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \varepsilon} g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$
- 因为:  $\max_{\Delta: \|\Delta\| \leq \varepsilon} \Delta^\top a \quad -> \quad \Delta = \varepsilon \frac{a}{\|a\|}$
- 因此:  $\Delta = \varepsilon \frac{\nabla g}{\|\nabla g\|} \quad \nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$

# 优化程序：梯度上升

---

- `init w`
- `for iter = 1, 2, ...`

$$w \leftarrow w + \alpha * \nabla g(w)$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

- $\alpha$ : 学习率—梯度上升的调整参数，需要仔细选择
- 如何选择学习率？多多尝试
  - 粗略的经验法则：每次更新使得权重的变化大约0.1–1%

# 基于对数似然目标的批梯度上升

---

$$\max_w \text{ll}(w) = \max_w \underbrace{\sum_i \log P(y^{(i)}|x^{(i)}; w)}_{g(w)}$$

- `init  $w$`
- `for iter = 1, 2, ...`

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)$$

# 对数似然目标下的随机梯度上升

---

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- 一旦计算了一个训练样本的梯度，可以在计算下一个之前进行更新

```
■ init w  
■ for iter = 1, 2, ...  
    ■ pick random j
```

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

# 对数似然目标下的小批量梯度上升

---

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

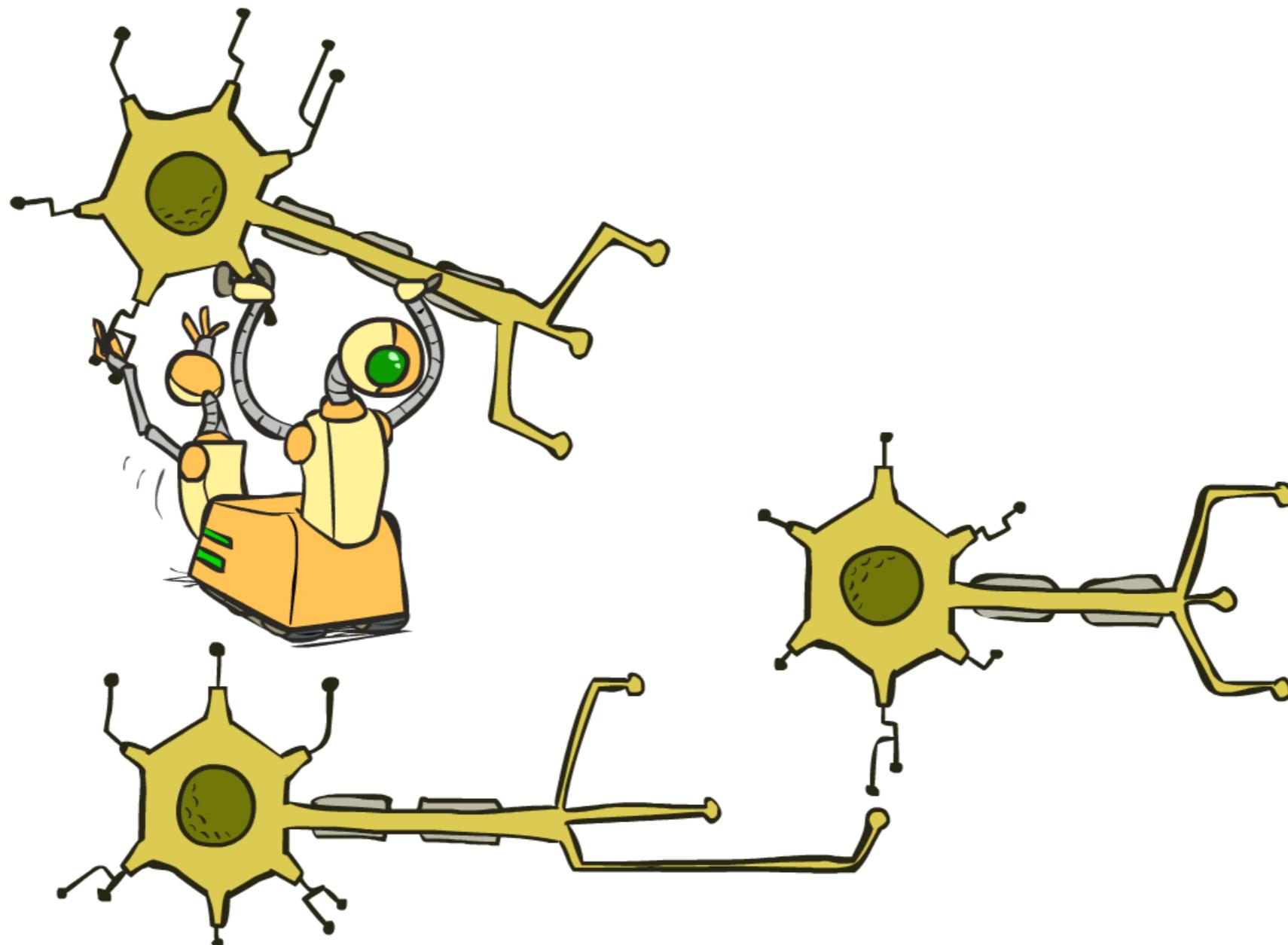
- 可以用一个小的数据集(mini-batch)上的梯度来代替单一样本上的梯度

```
■ init  $w$ 
■ for iter = 1, 2, ...
    ■ pick random subset of training examples  $J$ 
```

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)} | x^{(j)}; w)$$

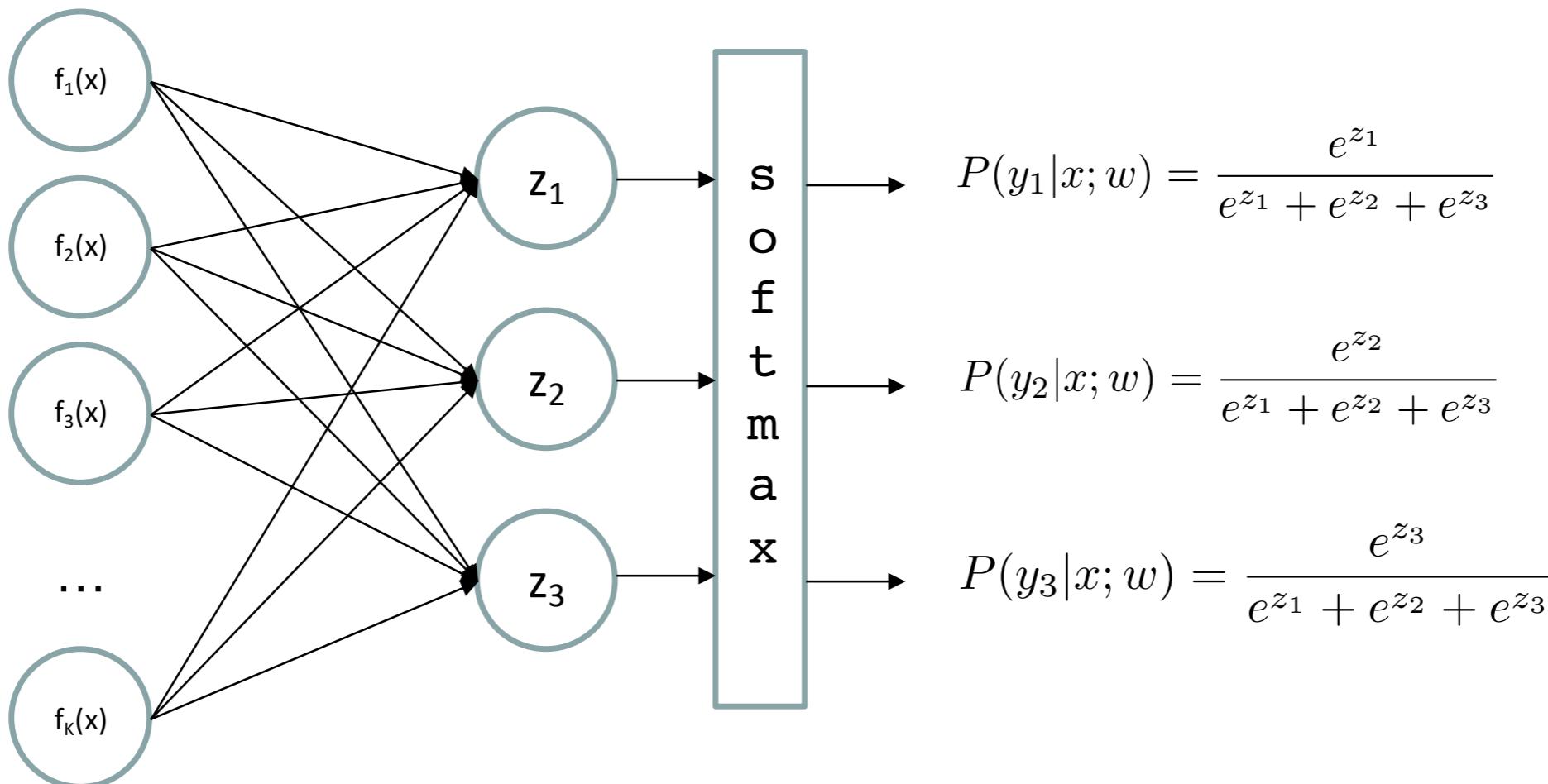
# 神经网络

---



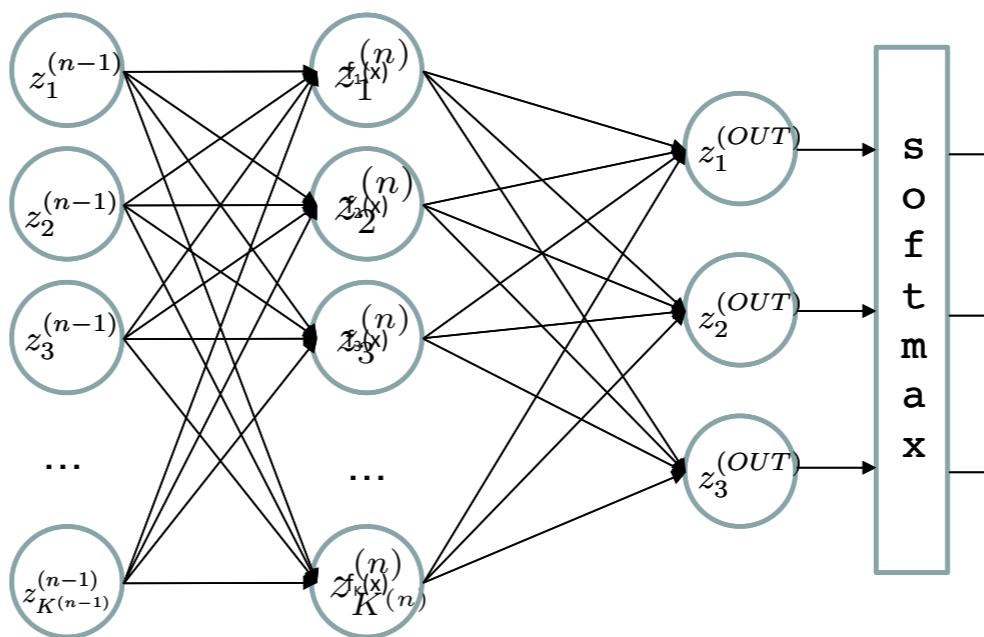
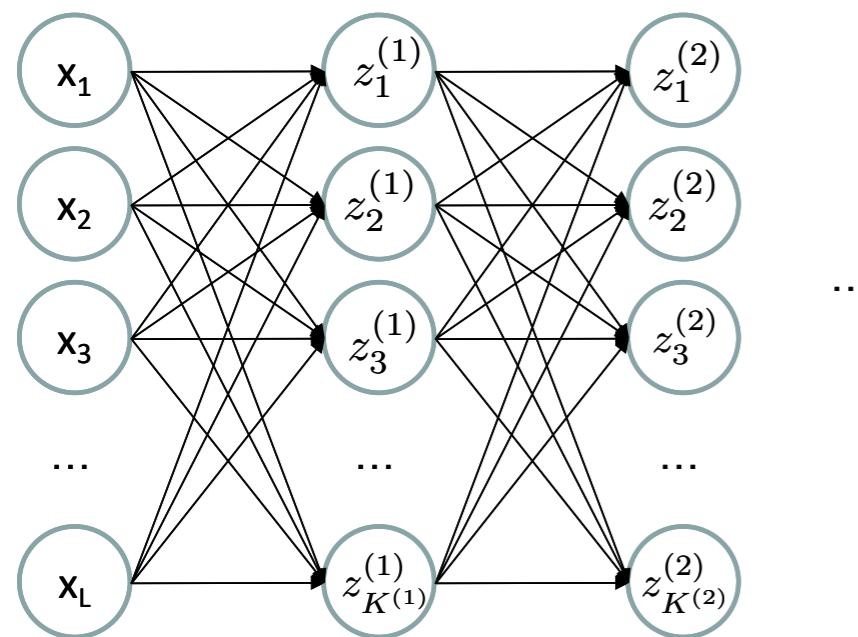
# 多分类逻辑回归

- 其实是神经网络的一种特殊形式



•

# 深层神经网络也学习特征



$$P(y_1|x; w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x; w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

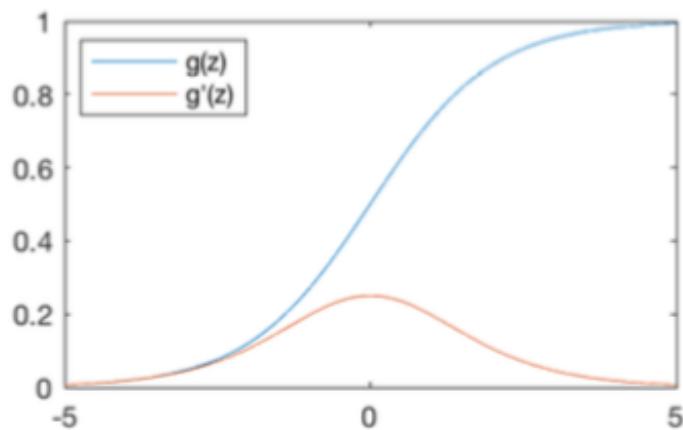
$$P(y_3|x; w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

**g = nonlinear activation function**

# 常用激活函数

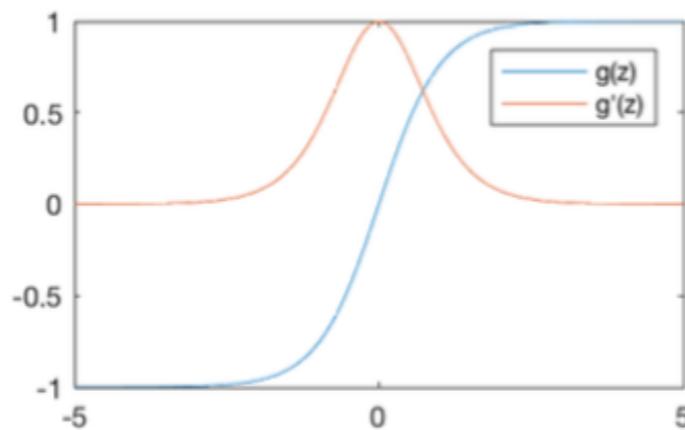
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

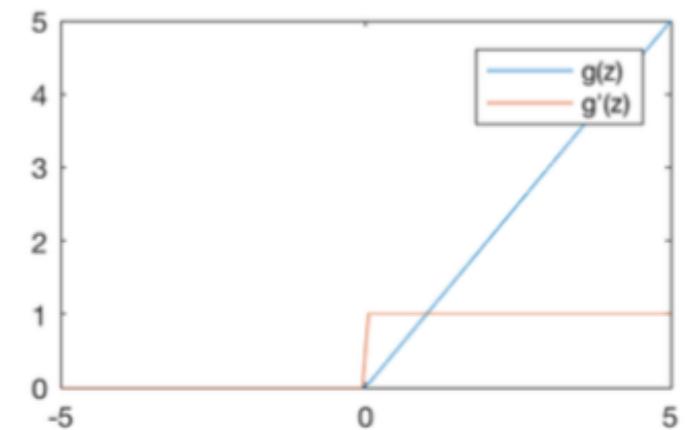
Hyperbolic Tangent



$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

# 深层神经网络也学习特征

---

- 训练深层神经网络和逻辑回归类似：
  - 运行梯度上升
  - 当在留出集上对数似然开始下降时，停止迭代过程

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

# 神经网络特性

---

- 定理（通用函数逼近器）。一个具有足够数量神经元的双层神经网络可以将任意连续函数近似到任何期望的精度。
- 实际考虑
  - 可以看作是特征的学习
  - 大量神经元会导致较高的过拟合风险

# 如何计算所有的导数?

---

- 链式法则

If  $f(x) = g(h(x))$

Then  $f'(x) = g'(h(x))h'(x)$

# 自动微分

---

- 自动微分软件
  - Theano, TensorFlow, PyTorch, Chainer
  - 只需通过编程定义函数 $g(x, y, w)$
  - 可以自动计算所有导数w.r.t.所有条目
  - 这通常是通过在f的正向计算过程中缓存信息，然后进行反向传递=“反向传播”来完成的
  - 自动差分/反向传播通常可以在计算成本与前向传递相当的情况下完成

# 作业

---

- 假设给定以下的 5 个编号的数据向量点，每个数据向量由 A 和 B 两个维度构成，标签列给出的是对应数据向量的分类，假设是二分类的情况。

| 数据索引 | A | B | 分类标签 |
|------|---|---|------|
| 1    | 1 | 1 | -    |
| 2    | 3 | 2 | +    |
| 3    | 2 | 4 | +    |
| 4    | 3 | 4 | +    |
| 5    | 2 | 3 | -    |

- 请回答：
  - 检查上面这些不同类别的数据点是否可以被线性分割辨别出来？
  - 训练一个感知机分类器。假设初始权值向量是 [-1,0,0]。依次使用上面给出的数据训练一轮该感知机模型，得到的权值向量是什么？假设我们对所有的输入数据都增加一维的常量值，即  $f_0 = 1$ ,  $f_1 = A$  的值  $f_2 = B$  的值。
  - 在 2 中得到的感知机是否可以正确分类以上数据？