

第三讲：Sigmoid 神经元和多层 Sigmoid 神经元网络

张盛平

s.zhang@hit.edu.cn

计算学部
哈尔滨工业大学

2021 年秋季学期

致谢

- 讲稿中很多资料或素材来源于网络，包括国外一些大学的相关课程、一些博客、维基百科等。后面不一一列举来源，在此一并表示感谢
- 引用网络资源时，由于本人的理解能力，可能存在一些偏差
- 本讲稿会经常更新



Sigmoid 神经元



背景与动机...

- 多层感知机可以实现任意的布尔函数

背景与动机...

- 多层感知机可以实现任意的布尔函数
- 如何实现任意的函数 $y = f(x)$, 其中 $x \in \mathbb{R}^n$ (而不是 $\{0, 1\}^n$), $y \in \mathbb{R}$ (而不是 $\{0, 1\}$) ?

背景与动机...

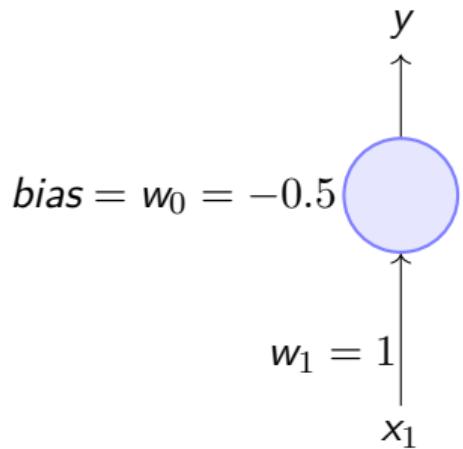
- 多层感知机可以实现任意的布尔函数
- 如何实现任意的函数 $y = f(x)$, 其中 $x \in \mathbb{R}^n$ (而不是 $\{0, 1\}^n$), $y \in \mathbb{R}$ (而不是 $\{0, 1\}$) ?
- 能找到一个网络来 (近似) 表示这样的函数吗 ?

背景与动机...

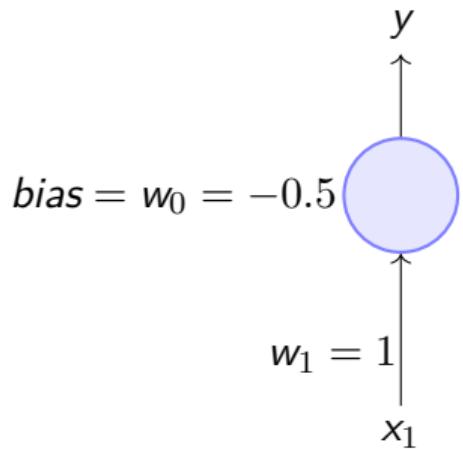
- 多层感知机可以实现任意的布尔函数
- 如何实现任意的函数 $y = f(x)$, 其中 $x \in \mathbb{R}^n$ (而不是 $\{0, 1\}^n$), $y \in \mathbb{R}$ (而不是 $\{0, 1\}$) ?
- 能找到一个网络来 (近似) 表示这样的函数吗 ?
- 回答这个问题之前, 看看如何从 感知机进化到 *sigmoidal* 神经元...

回顾

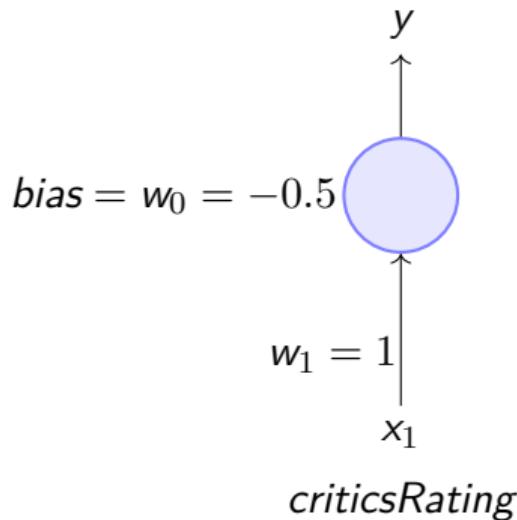
- 如果感知机输入的加权求和大于阈值 ($-w_0$)，这个感知机将『开火』



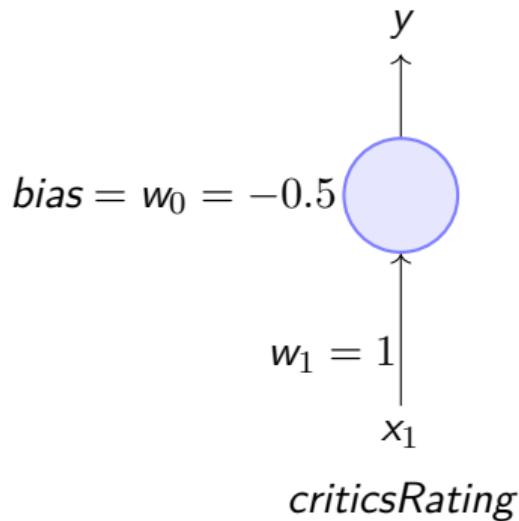
- 感知机所使用的阈值操作过于『硬』



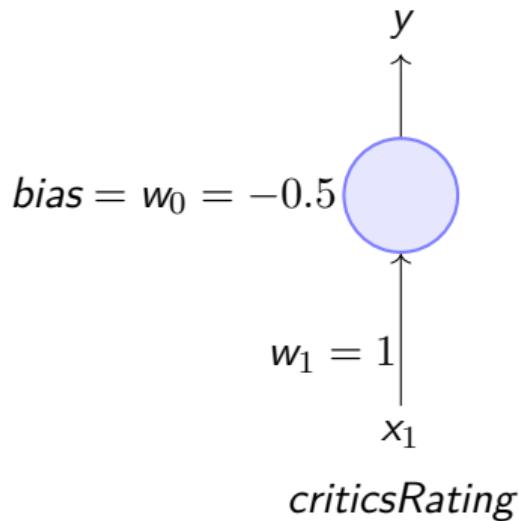
- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例



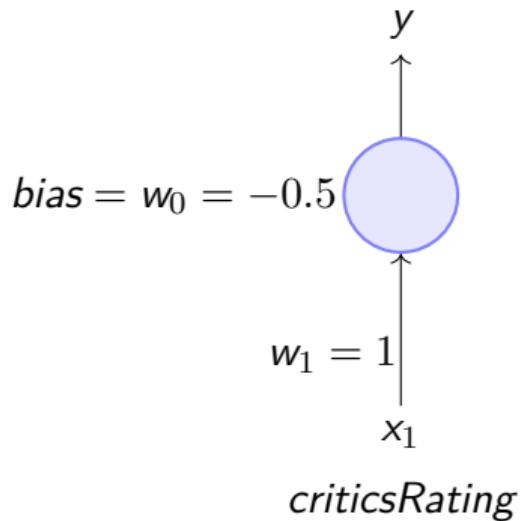
- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例
- 假设预测仅仅基于一个输入 ($x_1 = \text{criticsRating} \in [0, 1]$)



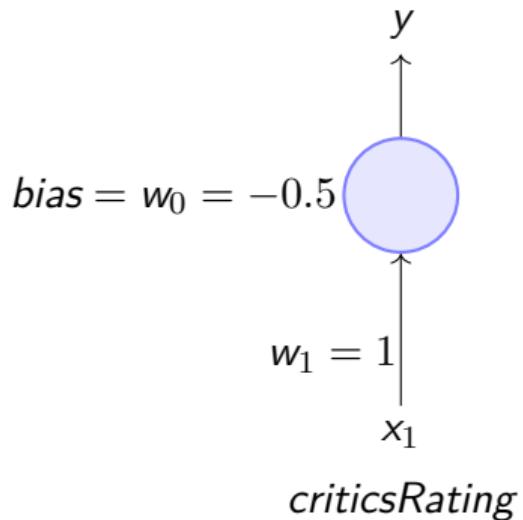
- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例
- 假设预测仅仅基于一个输入 ($x_1 = criticsRating \in [0, 1]$)
- 如果阈值是 0.5 ($w_0 = -0.5$), $w_1 = 1$, 如何预测一个 $criticsRating = 0.51$ 的电影?



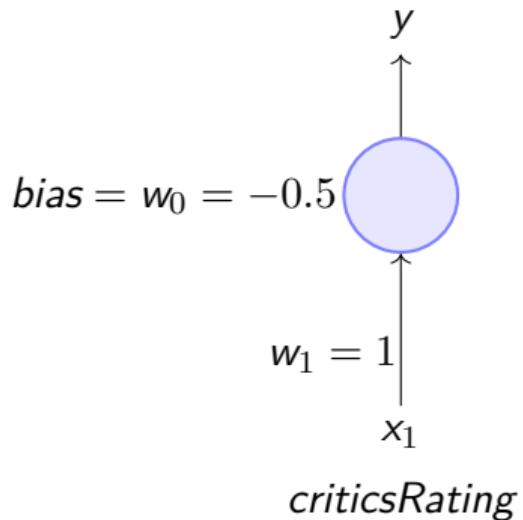
- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例
- 假设预测仅仅基于一个输入 ($x_1 = \text{criticsRating} \in [0, 1]$)
- 如果阈值是 0.5 ($w_0 = -0.5$), $w_1 = 1$, 如何预测一个 $\text{criticsRating} = 0.51$ 的电影? (喜欢)



- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例
- 假设预测仅仅基于一个输入 ($x_1 = criticsRating \in [0, 1]$)
- 如果阈值是 0.5 ($w_0 = -0.5$), $w_1 = 1$, 如何预测一个 $criticsRating = 0.51$ 的电影? (喜欢)
- 如何预测一个 $criticsRating = 0.49$ 的电影?

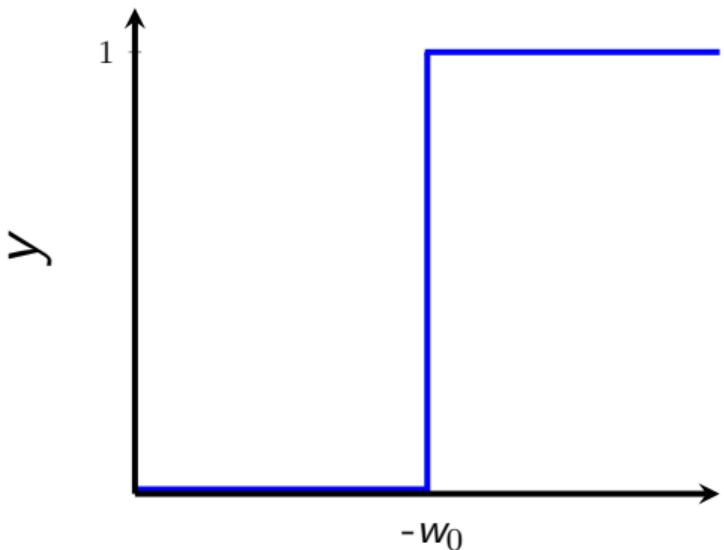


- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例
- 假设预测仅仅基于一个输入 ($x_1 = criticsRating \in [0, 1]$)
- 如果阈值是 0.5 ($w_0 = -0.5$), $w_1 = 1$, 如何预测一个 $criticsRating = 0.51$ 的电影? (喜欢)
- 如何预测一个 $criticsRating = 0.49$ 的电影? (不喜欢)



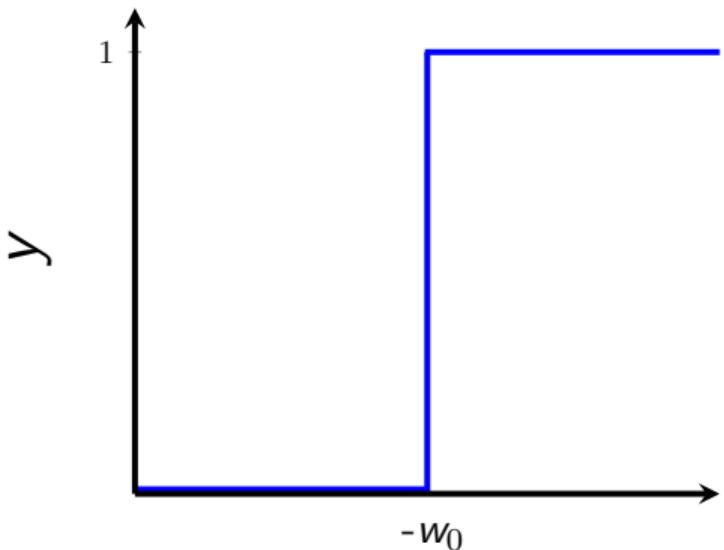
- 感知机所使用的阈值操作过于『硬』
- 以预测是否喜欢看一部电影为例
- 假设预测仅仅基于一个输入 ($x_1 = criticsRating \in [0, 1]$)
- 如果阈值是 0.5 ($w_0 = -0.5$), $w_1 = 1$, 如何预测一个 $criticsRating = 0.51$ 的电影? (喜欢)
- 如何预测一个 $criticsRating = 0.49$ 的电影? (不喜欢)
- 预测我们喜欢一个打分为 0.51 的电影而不喜欢一个打分为 0.49 的电影, 打分相差 0.02, 我们的预测却完全相反

- 这种输入轻微变化带来相反的预测不是因为问题本身，也不是因为我们选择的权重和阈值



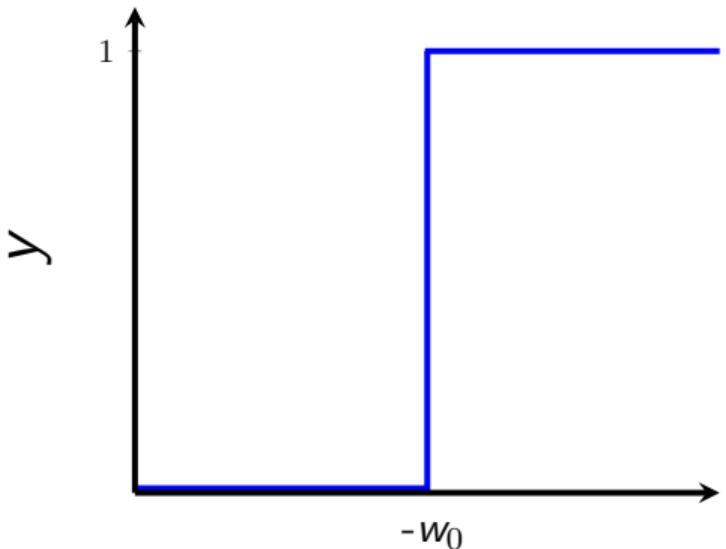
$$z = \sum_{i=1}^n w_i x_i$$

- 这种输入轻微变化带来相反的预测不是因为问题本身，也不是因为我们选择的权重和阈值
- 而是因为感知机的输出函数是一个阶跃函数



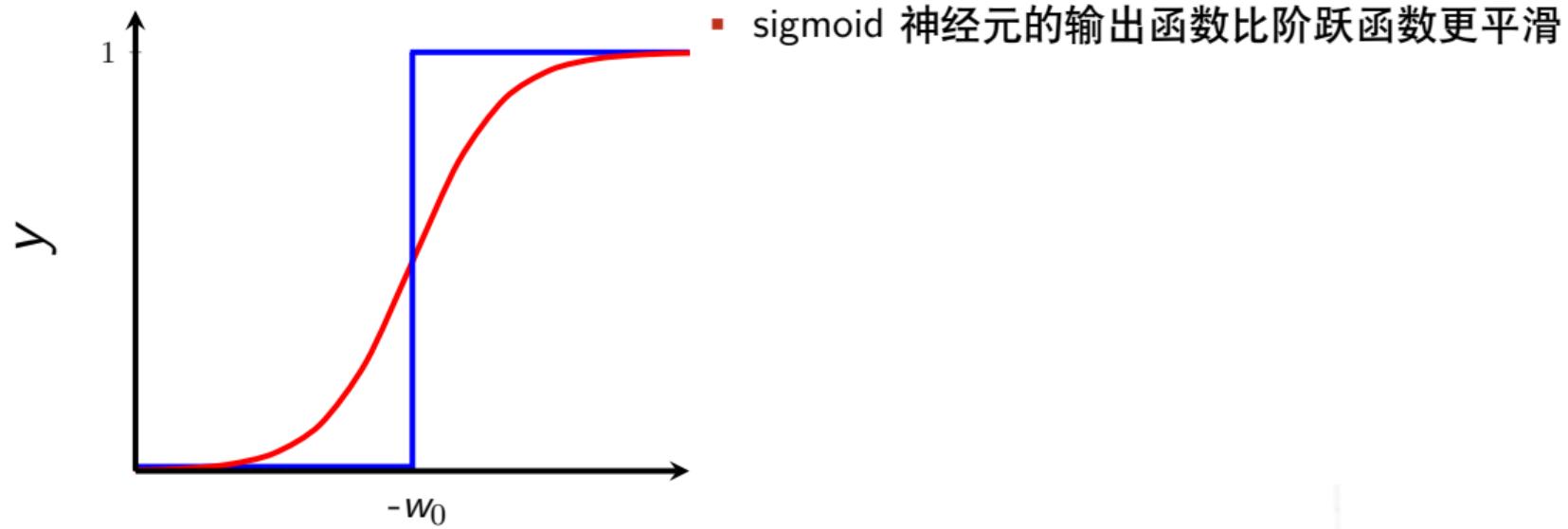
- 这种输入轻微变化带来相反的预测不是因为问题本身，也不是因为我们选择的权重和阈值
- 而是因为感知机的输出函数是一个阶跃函数
- 当输入的加权求和 $\sum_{i=1}^n w_i x_i$ 在阈值 (-w₀) 的两侧时，感知机的输出将会有突然的跳跃（从 0 到 1）

$$z = \sum_{i=1}^n w_i x_i$$

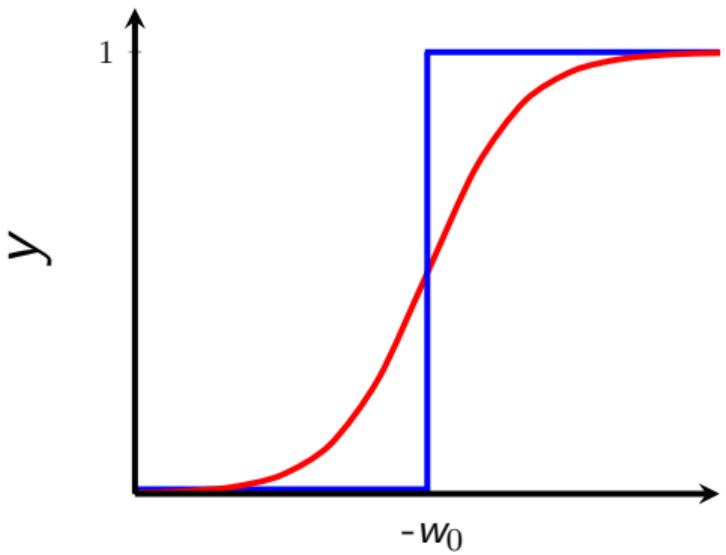


$$z = \sum_{i=1}^n w_i x_i$$

- 这种输入轻微变化带来相反的预测不是因为问题本身，也不是因为我们选择的权重和阈值
- 而是因为感知机的输出函数是一个阶跃函数
- 当输入的加权求和 $\sum_{i=1}^n w_i x_i$ 在阈值 $(-w_0)$ 的两侧时，感知机的输出将会有突然的跳跃（从 0 到 1）
- 对大多数实际的应用，我们期望一个更平滑的输出函数，能够慢慢从 0 变到 1



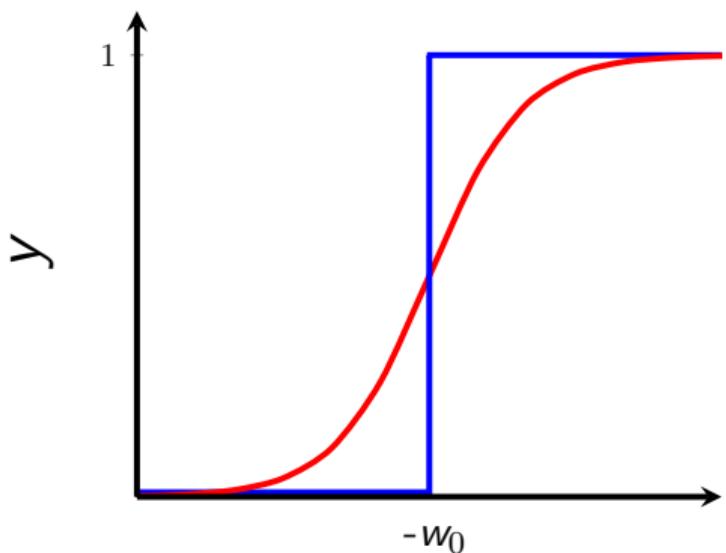
$$z = \sum_{i=1}^n w_i x_i$$



$$z = \sum_{i=1}^n w_i x_i$$

- sigmoid 神经元的输出函数比阶跃函数更平滑
- 下面是 sigmoid 函数的一种形式，称作为 logistic 函数

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^n w_i x_i)}}$$

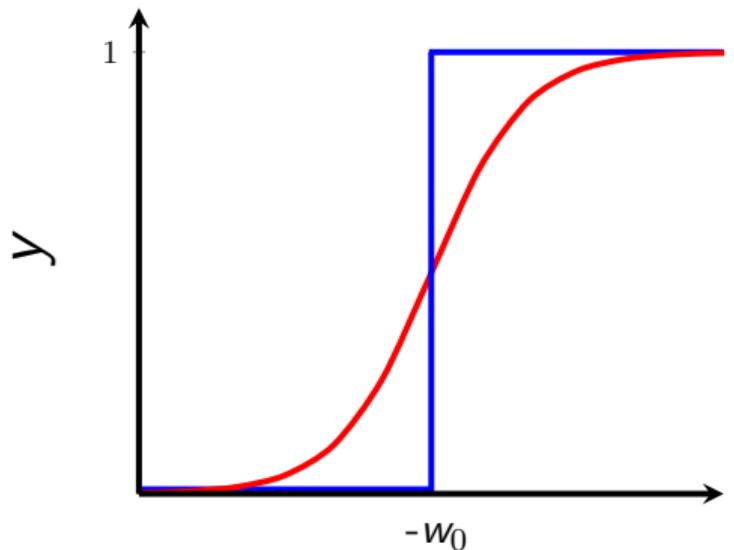


- sigmoid 神经元的输出函数比阶跃函数更平滑
- 下面是 sigmoid 函数的一种形式，称作为 logistic 函数

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^n w_i x_i)}}$$

- 在阈值 $(-w_0)$ 的两侧不再有陡峭的变化

$$z = \sum_{i=1}^n w_i x_i$$

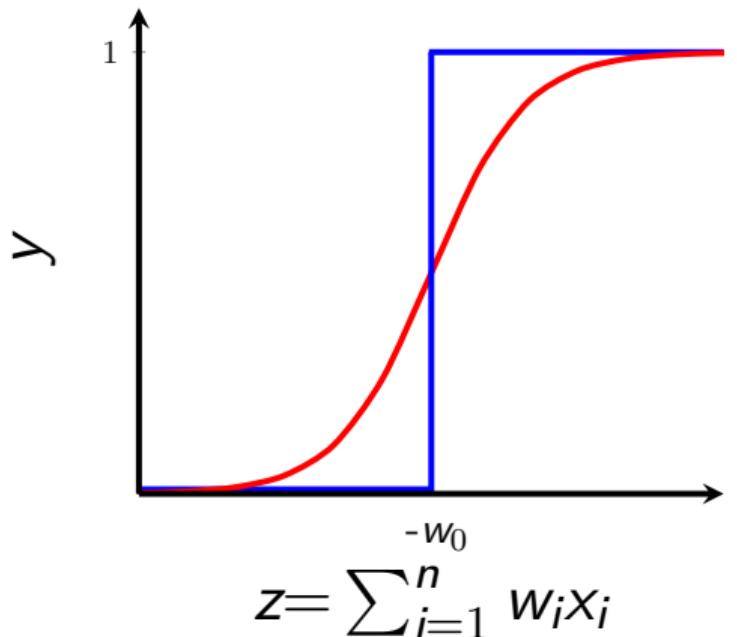


$$z = \sum_{i=1}^n w_i x_i$$

- sigmoid 神经元的输出函数比阶跃函数更平滑
- 下面是 sigmoid 函数的一种形式，称作为 logistic 函数

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^n w_i x_i)}}$$

- 在阈值 $(-w_0)$ 的两侧不再有陡峭的变化
- 输出 y 也不再是二值的，而是介于 0 和 1 之间的实数，可以被解释为一种概率



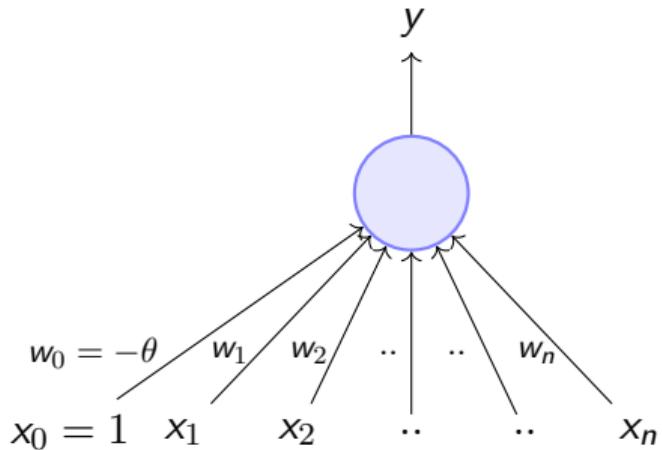
- sigmoid 神经元的输出函数比阶跃函数更平滑
- 下面是 sigmoid 函数的一种形式，称作为 logistic 函数

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^n w_i x_i)}}$$

- 在阈值 $(-w_0)$ 的两侧不再有陡峭的变化
- 输出 y 也不再是二值的，而是介于 0 和 1 之间的实数，可以被解释为一种概率
- 不同于预测『喜欢』或『不喜欢』，而是预测喜欢一部电影的概率



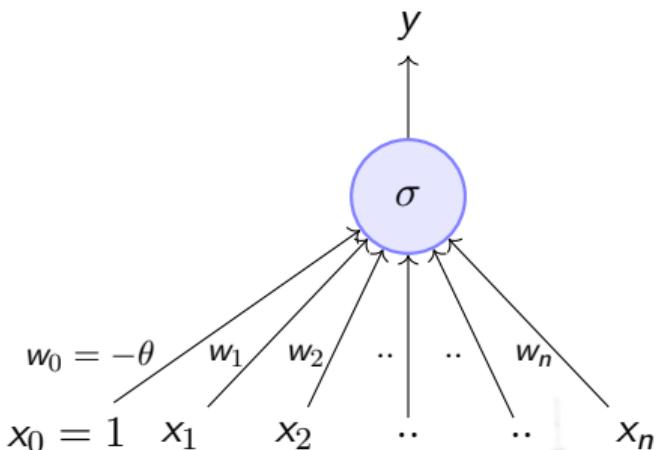
感知机



$$y = 1 \quad \text{if} \sum_{i=0}^n w_i * x_i \geq 0$$

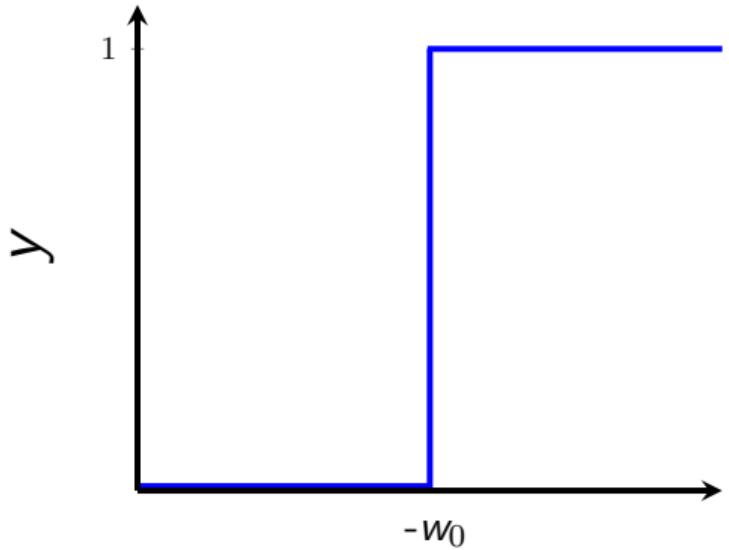
$$= 0 \quad \text{if} \sum_{i=0}^n w_i * x_i < 0$$

Sigmoid (logistic) 神经元



$$y = \frac{1}{1 + e^{-(\sum_{i=0}^n w_i x_i)}}$$

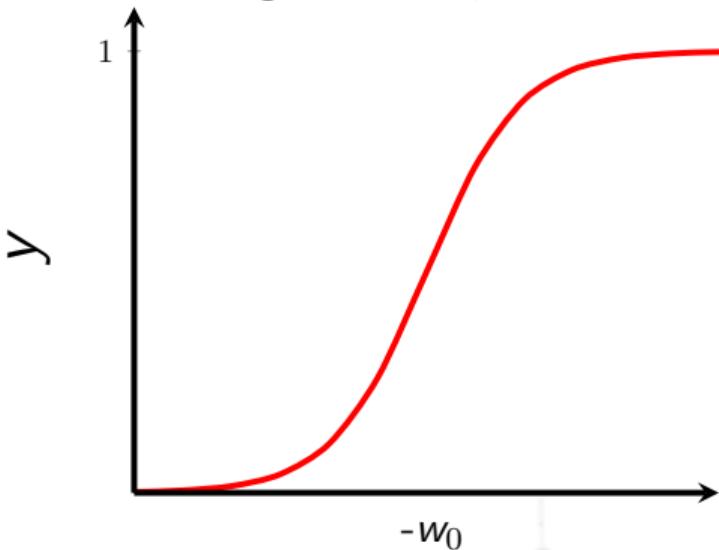
感知机



$$z = \sum_{i=1}^n w_i x_i$$

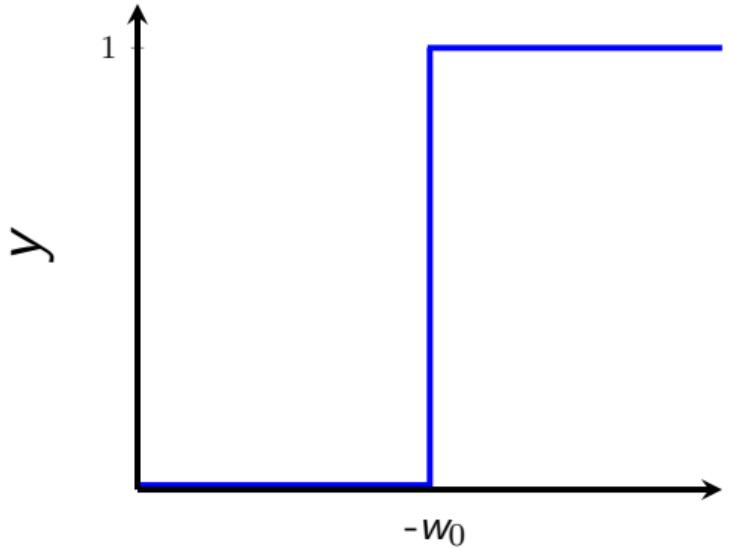
不平滑, 不连续 (在 w_0 处), 不可微

Sigmoid 神经元



$$z = \sum_{i=1}^n w_i x_i$$

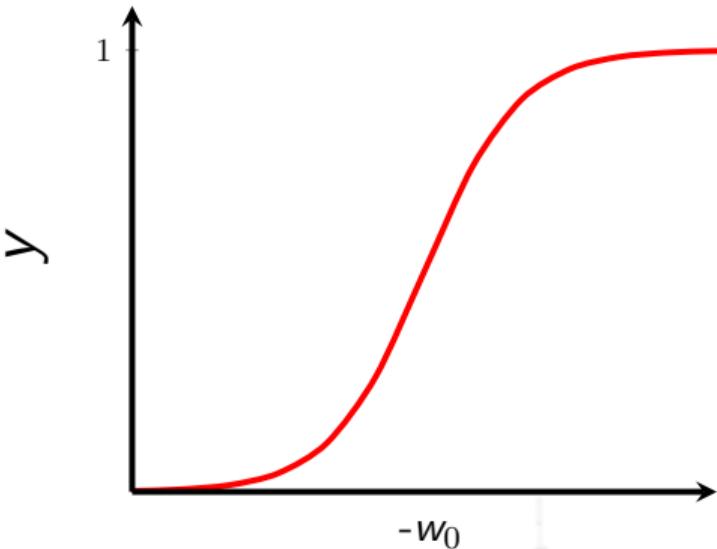
感知机



$$z = \sum_{i=1}^n w_i x_i$$

不平滑, 不连续 (在 w_0 处), 不可微

Sigmoid 神经元



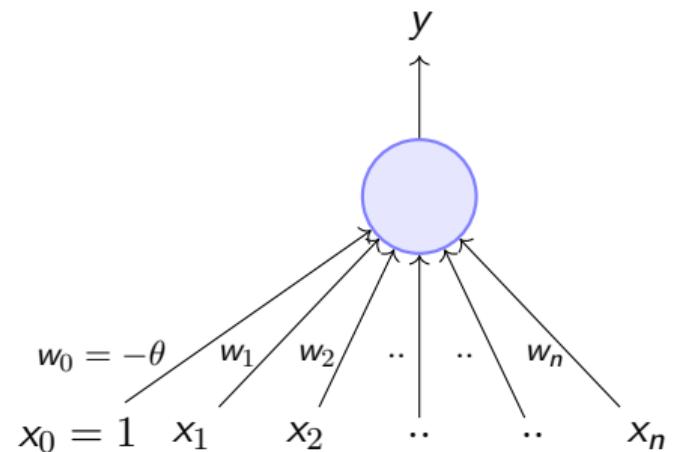
$$z = \sum_{i=1}^n w_i x_i$$

平滑, 连续, 可微

一个典型的有监督机器学习设置



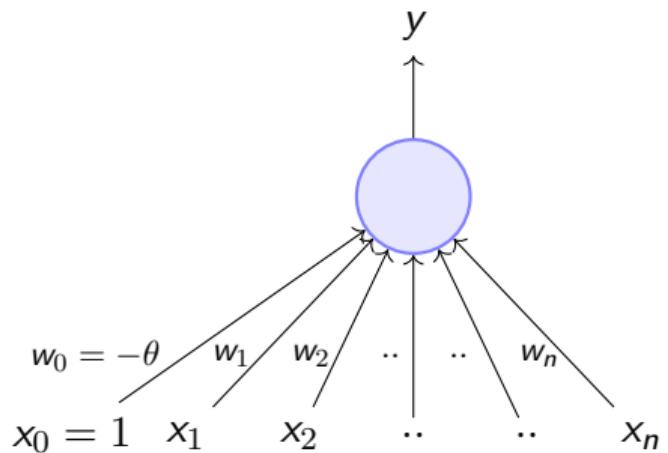
Sigmoid (logistic) 神经元



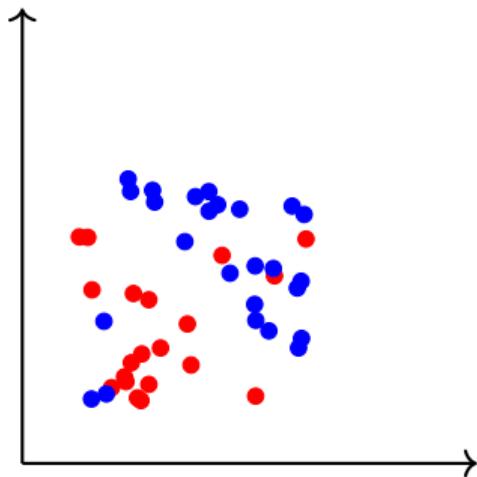
- 正如感知机学习算法用于学习感知机权重一样，我们也需要一个算法来学习 sigmoid 神经元的权重



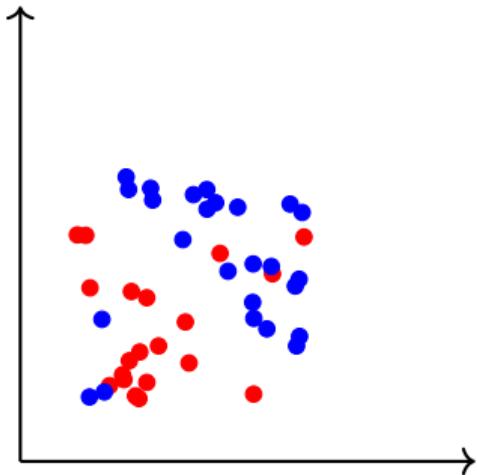
Sigmoid (logistic) 神经元



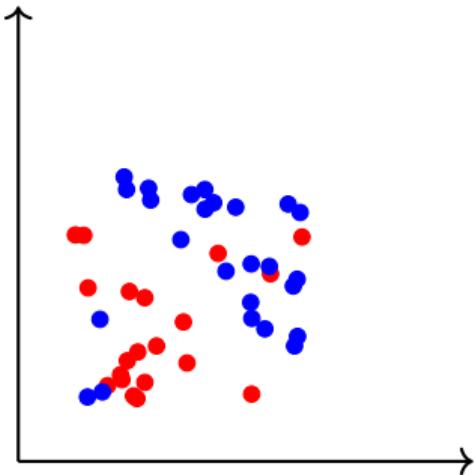
- 正如感知机学习算法用于学习感知机权重一样，我们也需要一个算法来学习 sigmoid 神经元的权重
- 在介绍学习算法之前，先看看误差 error 的概念



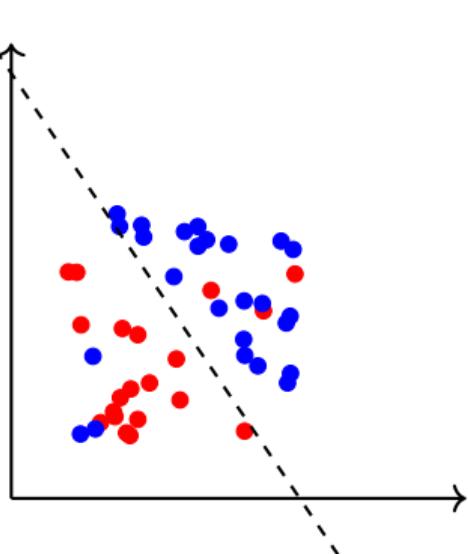
- 单个的感知机不能用于处理这样线性不可分的数据



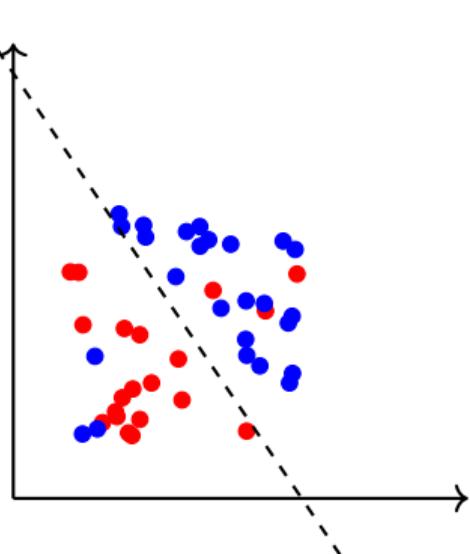
- 单个的感知机不能用于处理这样线性不可分的数据
- “不能处理” 是啥意思？



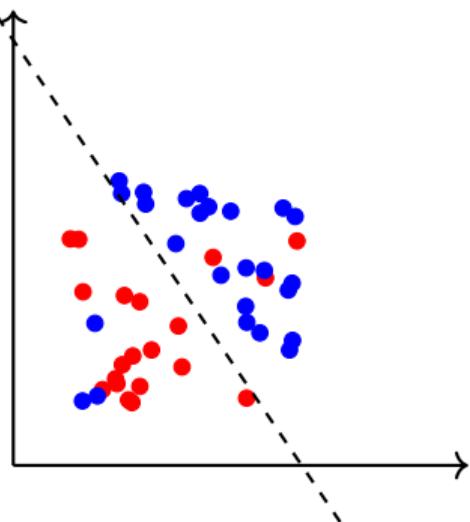
- 单个的感知机不能用于处理这样线性不可分的数据
- “不能处理” 是啥意思？
- 如果非要使用一个单个的感知机模型来处理这样的数据呢？



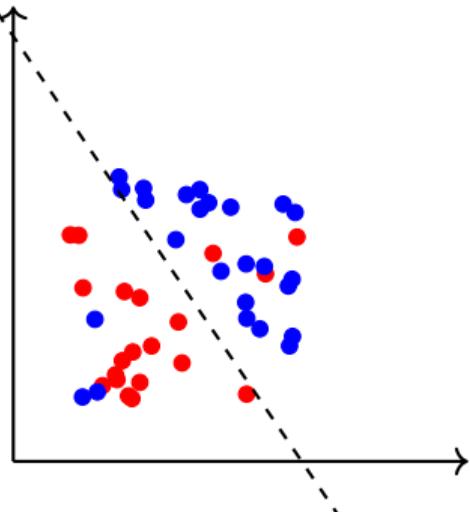
- 单个的感知机不能用于处理这样线性不可分的数据
- “不能处理” 是啥意思？
- 如果非要使用一个单个的感知机模型来处理这样的数据呢？
- 我们可能得到一条这样的直线…



- 单个的感知机不能用于处理这样线性不可分的数据
- “不能处理” 是啥意思？
- 如果非要使用一个单个的感知机模型来处理这样的数据呢？
- 我们可能得到一条这样的直线…
- 这条直线也不是太糟糕



- 单个的感知机不能用于处理这样线性不可分的数据
- “不能处理” 是啥意思？
- 如果非要使用一个单个的感知机模型来处理这样的数据呢？
- 我们可能得到一条这样的直线…
- 这条直线也不是太糟糕
- 虽然它误分了 3 个蓝色的点和 3 个红色的点，这样的错误对于大多数应用而言还是可以接受的



- 单个的感知机不能用于处理这样线性不可分的数据
- “不能处理” 是啥意思？
- 如果非要使用一个单个的感知机模型来处理这样的数据呢？
- 我们可能得到一条这样的直线…
- 这条直线也不是太糟糕
- 虽然它误分了 3 个蓝色的点和 3 个红色的点，这样的错误对于大多数应用而言还是可以接受的
- 在大多数情况，我们的目标并不是将误差降到零，相反，是为了最小化可能的误差

一个典型的有监督机器学习设置包括下面的部分：

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T w x$

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T w x$

或者是任意函数

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T w x$

或者是任意函数

- **参数 (Parameters)** : 在所有上面的例子中, w 是需要从数据中学习的参数

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T w x$

或者是任意函数

- **参数 (Parameters)** : 在所有上面的例子中, w 是需要从数据中学习的参数
- **学习算法 (Learning algorithm)** : 学习模型参数 w 的算法, 例如感知机学习算法、梯度下降算法等

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T w x$

或者是任意函数

- **参数 (Parameters)** : 在所有上面的例子中, w 是需要从数据中学习的参数
- **学习算法 (Learning algorithm)** : 学习模型参数 w 的算法, 例如感知机学习算法、梯度下降算法等
- **目标/损失/误差函数 (Objective/Loss/Error function)** : 用于指导学习算法

一个典型的有监督机器学习设置包括下面的部分：

- **数据 (Data)** : $\{x_i, y_i\}_{i=1}^n$
- **模型 (Model)** : 描述 x 和 y 之间关系的函数, 例如,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T w x$

或者是任意函数

- **参数 (Parameters)** : 在所有上面的例子中, w 是需要从数据中学习的参数
- **学习算法 (Learning algorithm)** : 学习模型参数 w 的算法, 例如感知机学习算法、梯度下降算法等
- **目标/损失/误差函数 (Objective/Loss/Error function)** : 用于指导学习算法 — 学习算法的目标是最小化损失函数



以预测电影为例





以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$

以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数



以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

- **参数:** w



以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

- **参数:** w
- **学习算法:** 梯度下降 (接下来会学到)

以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

- **参数:** w
- **学习算法:** 梯度下降 (接下来会学到)
- **目标/损失/误差函数:**

以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

- **参数:** w
- **学习算法:** 梯度下降（接下来会学到）
- **目标/损失/误差函数:** 一种可能的损失函数是

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

以预测电影为例

- **数据:** $\{x_i = \text{movie}, y_i = \text{like/dislike}\}_{i=1}^n$
- **模型:** 描述 x 和 y 之间关系的函数

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

- **参数:** w
- **学习算法:** 梯度下降（接下来会学到）
- **目标/损失/误差函数:** 一种可能的损失函数是

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

学习算法的目标是找到 w 使得上面的损失函数最小

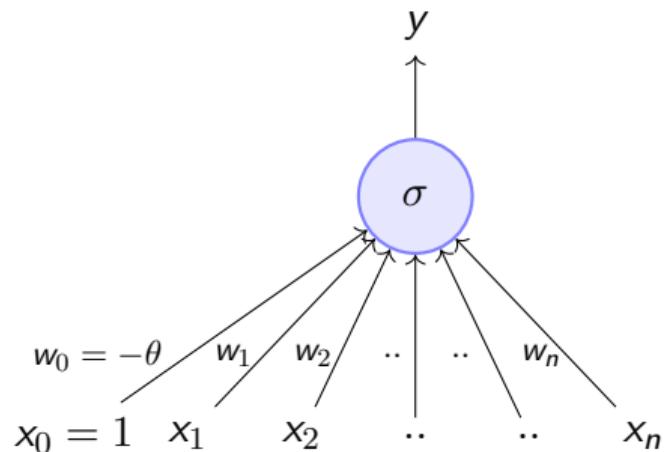


参数学习：随机猜测

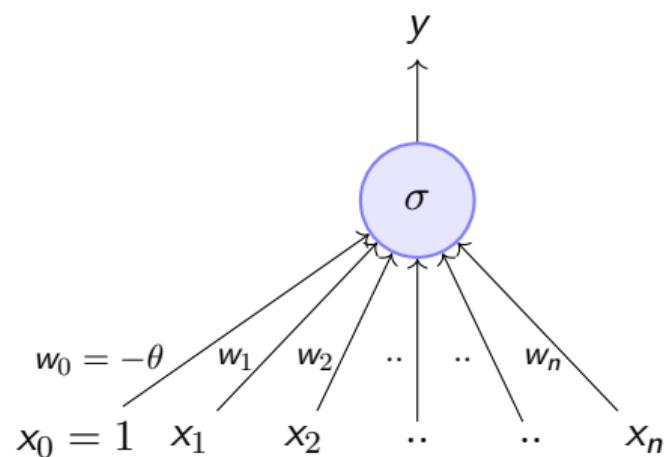




- 给定一个适当的目标函数，设计一个 算法，从数据中学习这个模型的权重

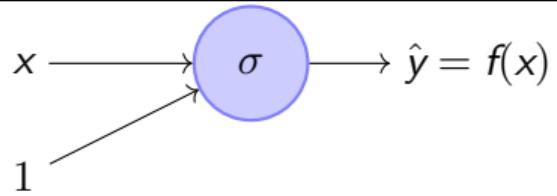


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



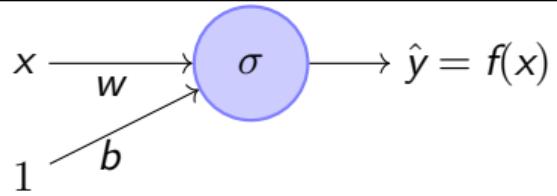
- 给定一个适当的目标函数，设计一个 算法，从 数据中学习这个模型的权重
- σ 表示 sigmoid 函数（本例中的 logistic 函数）

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



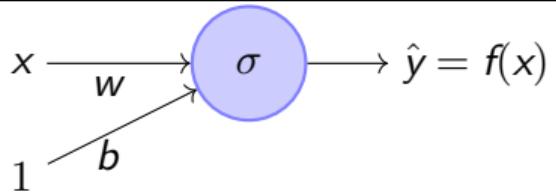
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

- 给定一个适当的目标函数，设计一个 算法，从 数据中学习这个模型的权重
- σ 表示 sigmoid 函数（本例中的 logistic 函数）
- 为了解释起来方便，考虑只有一个输入的简化模型



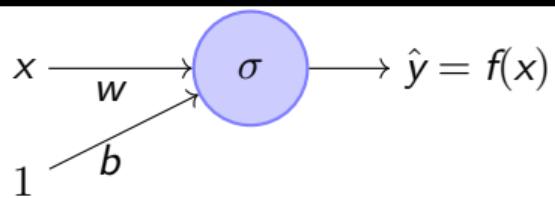
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

- 给定一个适当的目标函数，设计一个 算法，从 数据中学习这个模型的权重
- σ 表示 sigmoid 函数（本例中的 logistic 函数）
- 为了解释起来方便，考虑只有一个输入的简化 模型
- 为了和文献一致，称 w_0 为 b (bias)

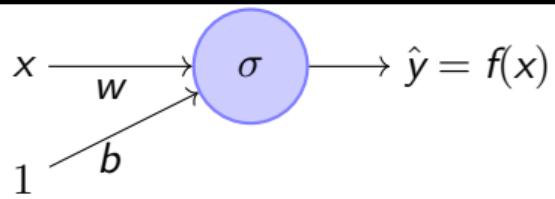


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

- 给定一个适当的目标函数，设计一个 算法，从 数据中学习这个模型的权重
- σ 表示 sigmoid 函数（本例中的 logistic 函数）
- 为了解释起来方便，考虑只有一个输入的简化 模型
- 为了和文献一致，称 w_0 为 b (bias)
- 不预测喜欢/不喜欢，而是给定 $imdbRating(x)$ ，预测 $criticsRating(y)$



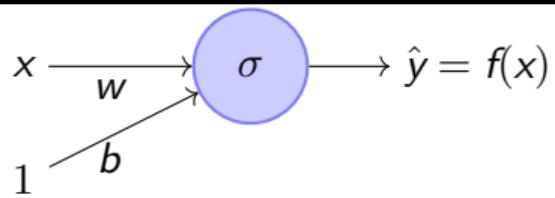
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



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

Input for training

$\{x_i, y_i\}_{i=1}^N \rightarrow N$ pairs of (x, y)



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

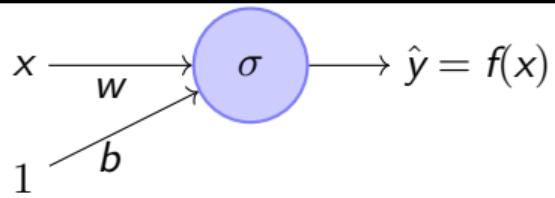
Input for training

$\{x_i, y_i\}_{i=1}^N \rightarrow N$ pairs of (x, y)

Training objective

找到 w 和 b 使得:

$$\underset{w,b}{\text{minimize}} \mathcal{L}(w, b) = \sum_{i=1}^N (y_i - f(x_i))^2$$



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

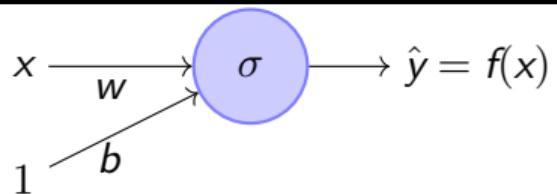
Input for training

$\{x_i, y_i\}_{i=1}^N \rightarrow N$ pairs of (x, y)

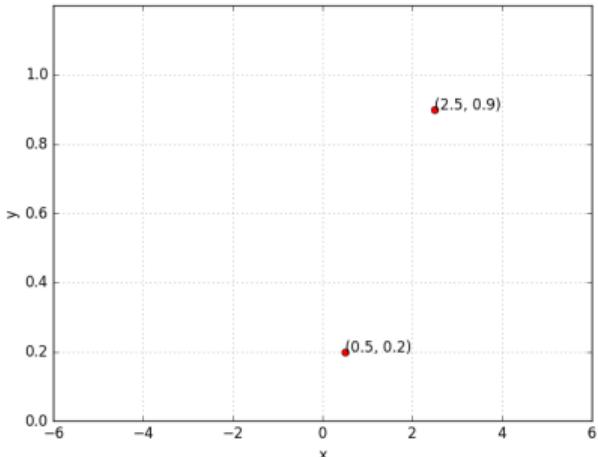
Training objective

找到 w 和 b 使得:

$$\underset{w,b}{\text{minimize}} \mathcal{L}(w, b) = \sum_{i=1}^N (y_i - f(x_i))^2$$



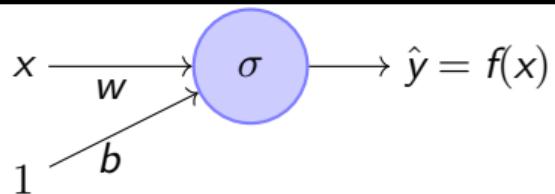
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



训练这个网络意味着：

- 假定有两个训练数据 $(x, y) = (0.5, 0.2)$ 和 $(2.5, 0.9)$

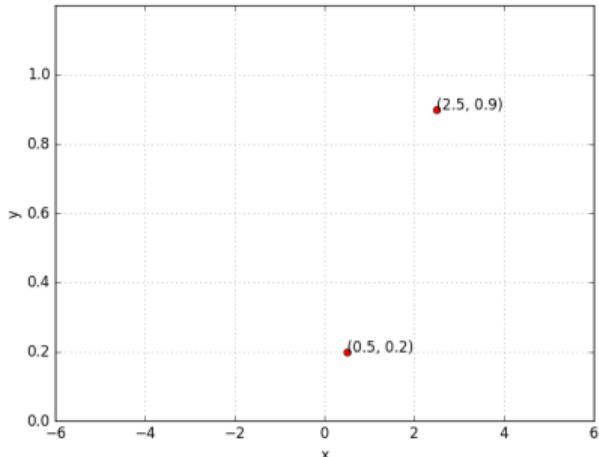


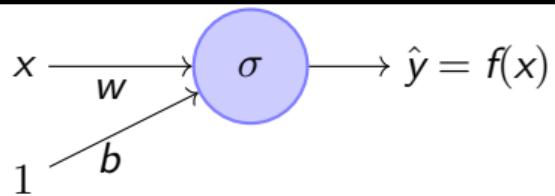


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

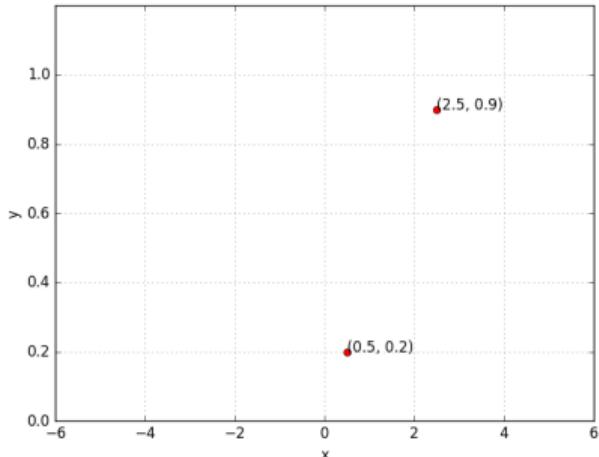
训练这个网络意味着：

- 假定有两个训练数据 $(x, y) = (0.5, 0.2)$ 和 $(2.5, 0.9)$
- 训练结束后，算法找到 w^* , b^* 使得





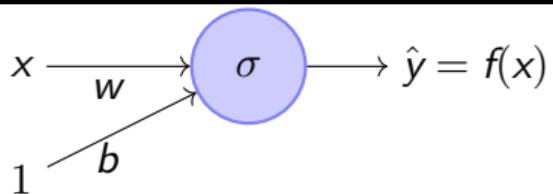
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



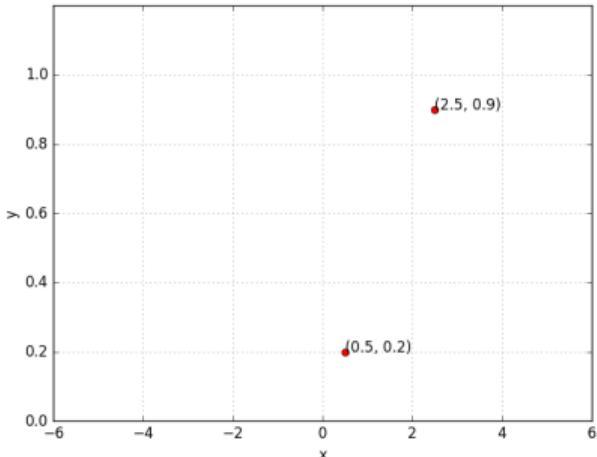
训练这个网络意味着：

- 假定有两个训练数据 $(x, y) = (0.5, 0.2)$ 和 $(2.5, 0.9)$
- 训练结束后，算法找到 w^* , b^* 使得
- $f(0.5) \rightarrow 0.2$, $f(2.5) \rightarrow 0.9$





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

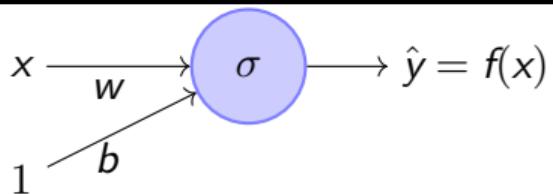


训练这个网络意味着：

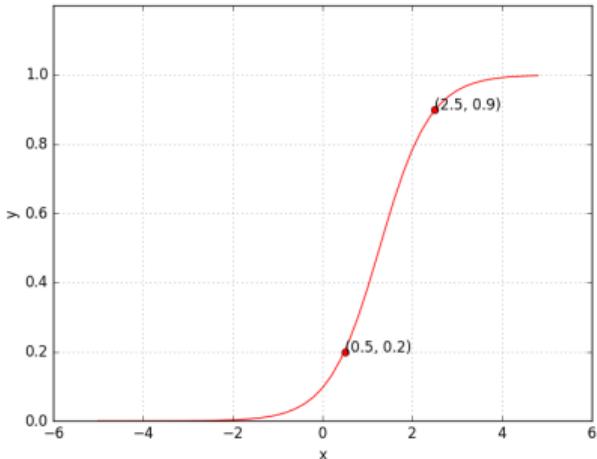
- 假定有两个训练数据 $(x, y) = (0.5, 0.2)$ 和 $(2.5, 0.9)$
- 训练结束后，算法找到 w^* , b^* 使得
- $f(0.5) \rightarrow 0.2$, $f(2.5) \rightarrow 0.9$

换句话说...

- 希望找到一个 sigmoid 函数，使得 $(0.5, 0.2)$ 和 $(2.5, 0.9)$ 在函数曲线上



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



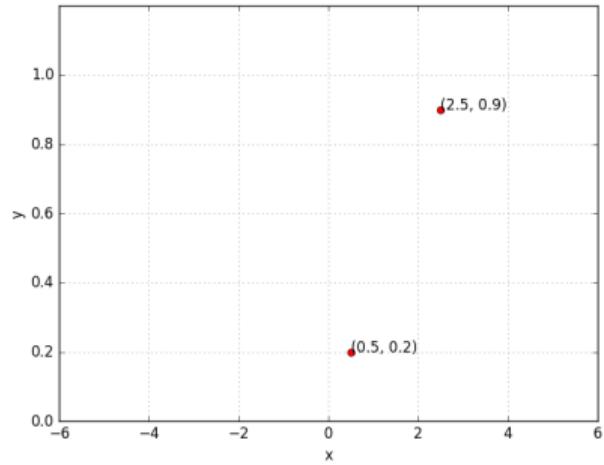
训练这个网络意味着：

- 假定有两个训练数据 $(x, y) = (0.5, 0.2)$ 和 $(2.5, 0.9)$
- 训练结束后，算法找到 w^* , b^* 使得
- $f(0.5) \rightarrow 0.2$, $f(2.5) \rightarrow 0.9$

换句话说...

- 希望找到一个 sigmoid 函数，使得 $(0.5, 0.2)$ 和 $(2.5, 0.9)$ 在函数曲线上

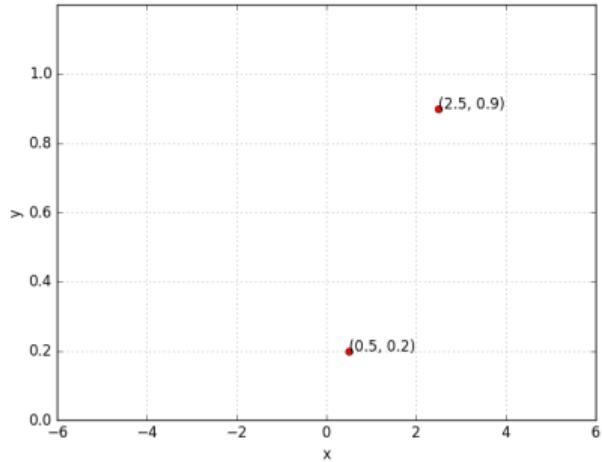
这是一个曲线拟合 (*Curve fitting*) 问题



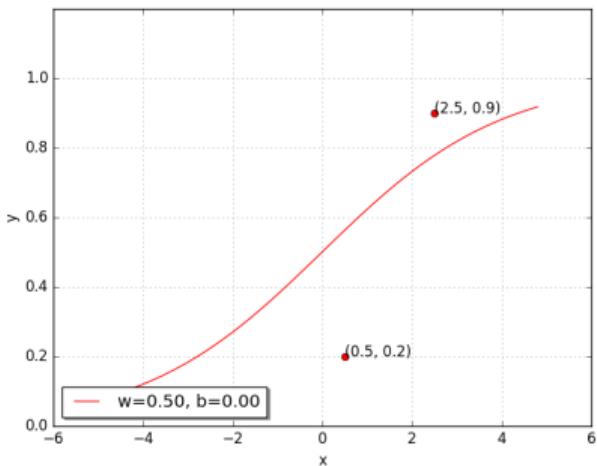
$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$



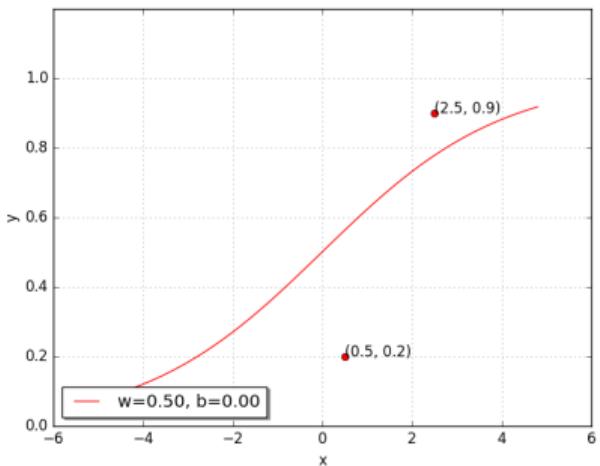
- 能否手动地找到这样的 w^* 和 b^* ?



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

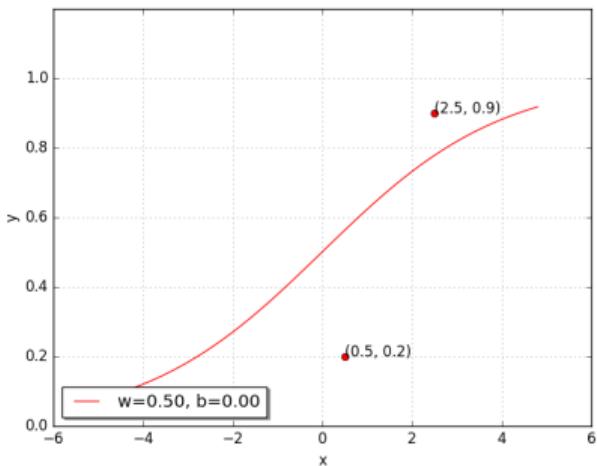


$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$



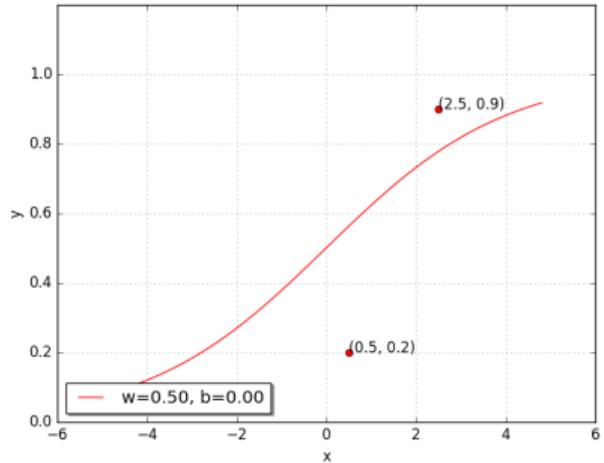
- 能否手动地找到这样的 w^* 和 b^* ?
- 随机猜测一个 $w = 0.5, b = 0$)
- 显然不是很好，但有多坏呢？

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



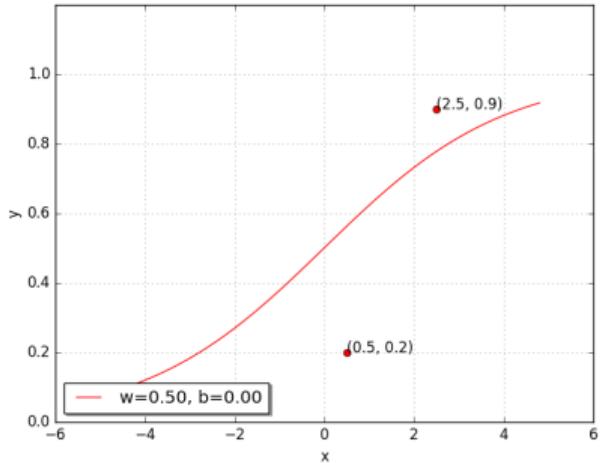
- 能否手动地找到这样的 w^* 和 b^* ?
- 随机猜测一个 $w = 0.5, b = 0$)
- 显然不是很好，但有多坏呢？
- 计算损失函数 $\mathcal{L}(w, b)$

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



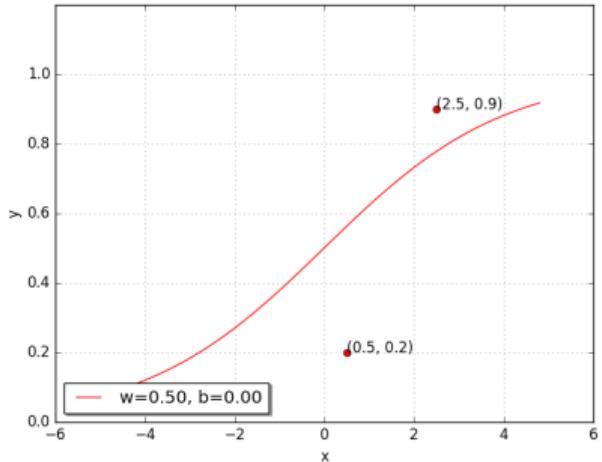
$$\mathcal{L}(w, b) = \frac{1}{2} * \sum_{i=1}^N (y_i - f(x_i))^2$$

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



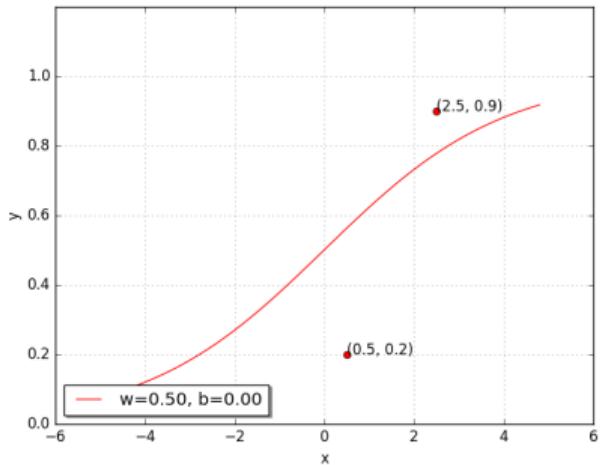
$$\begin{aligned}\mathcal{L}(w, b) &= \frac{1}{2} * \sum_{i=1}^N (y_i - f(x_i))^2 \\ &= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2\end{aligned}$$

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



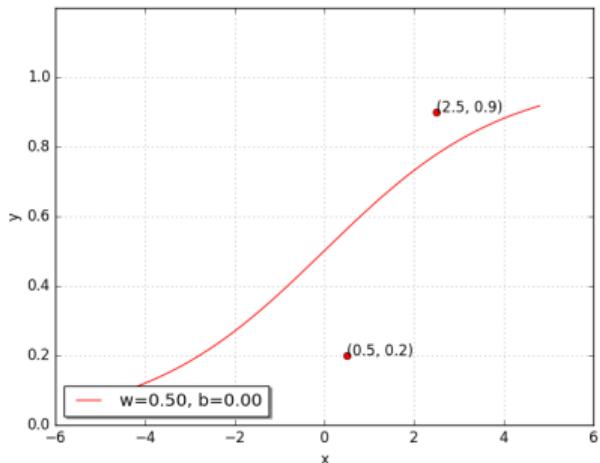
$$\begin{aligned}\mathcal{L}(w, b) &= \frac{1}{2} * \sum_{i=1}^N (y_i - f(x_i))^2 \\ &= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 \\ &= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2\end{aligned}$$

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



$$\begin{aligned}
 \mathcal{L}(w, b) &= \frac{1}{2} * \sum_{i=1}^N (y_i - f(x_i))^2 \\
 &= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 \\
 &= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2 \\
 &= 0.073
 \end{aligned}$$

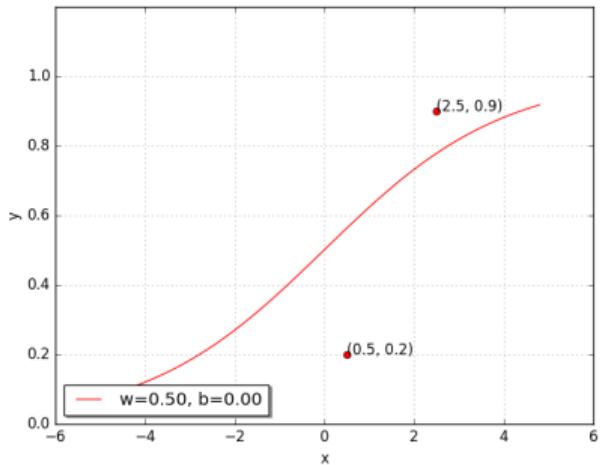
$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$



$$\begin{aligned}
 \mathcal{L}(w, b) &= \frac{1}{2} * \sum_{i=1}^N (y_i - f(x_i))^2 \\
 &= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 \\
 &= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2 \\
 &= 0.073
 \end{aligned}$$

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

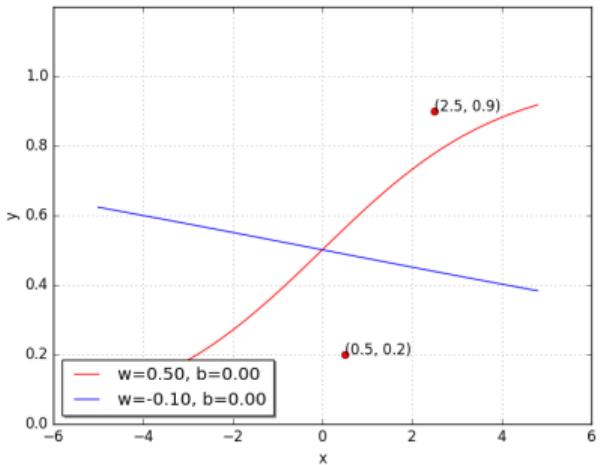
目标是希望损失函数 $\mathcal{L}(w, b)$ 尽可能接近 0



试一下其他的 w, b

w	b	$\mathcal{L}(w, b)$
0.50	0.00	0.0730

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

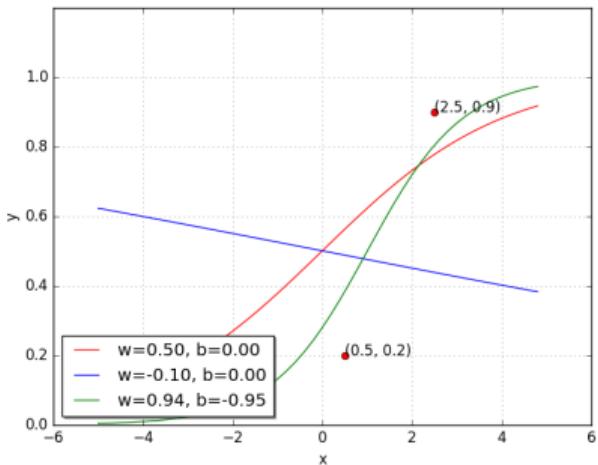


试一下其他的 w, b

w	b	$\mathcal{L}(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481

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

损失函数更大了...

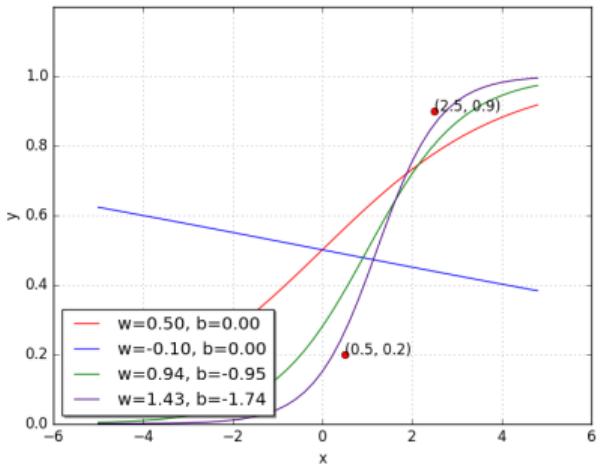


试一下其他的 w, b

w	b	$\mathcal{L}(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214

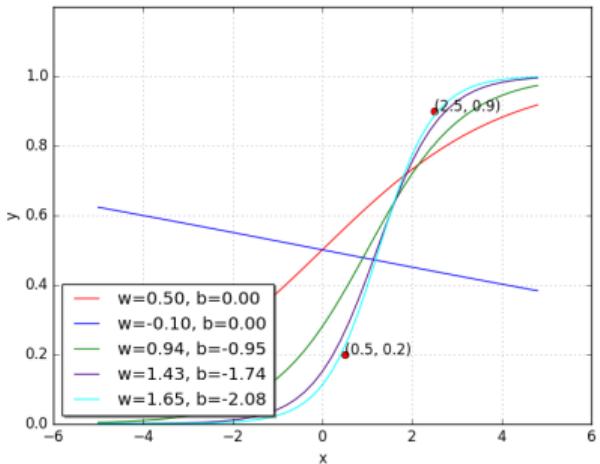
$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$

也许需要朝相反的方向来调整 w 和 b ...



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

增加 w 并且降低 b

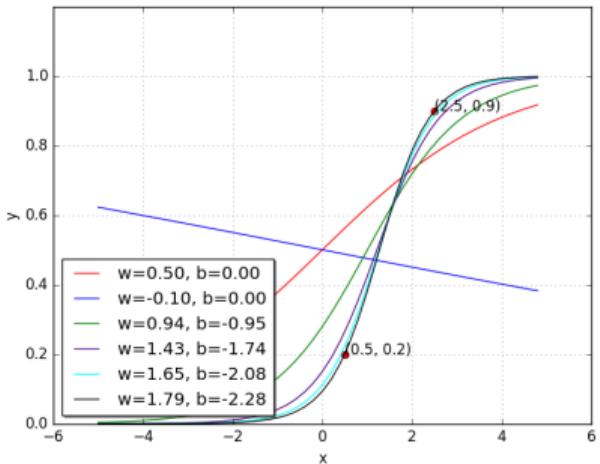


试一下其他的 w, b

w	b	$\mathcal{L}(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003

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

增加 w 并且降低 b



试一下其他的 w, b

w	b	$\mathcal{L}(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

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

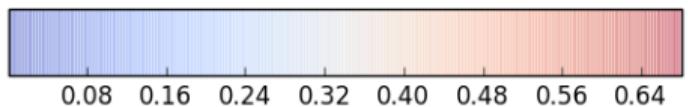
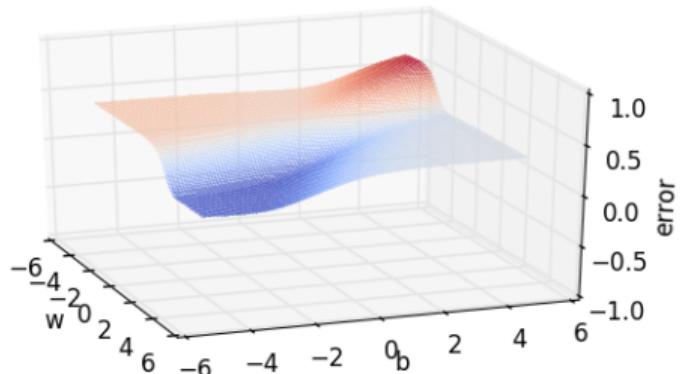
经过几次尝试，终于找到了使损失函数为 0 的 w 和 b

有没有什么比随机猜测更好的方法....



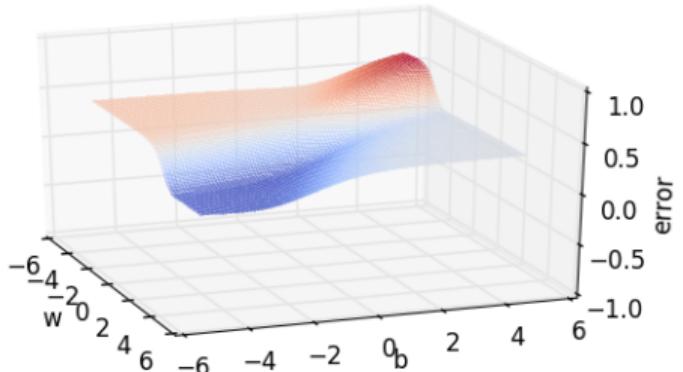
- 当只有两个训练数据和 2 个需要学习的参数时，可以容易地画出损失函数 $\mathcal{L}(w, b)$ 随参数 (w, b) 变化的图，从而可以很简单地手动找到使 $(\mathcal{L}(w, b))$ 最小的参数值

Random search on error surface

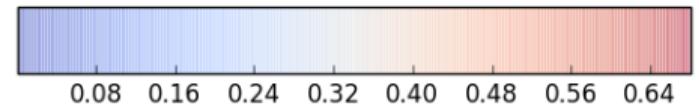


- 当只有两个训练数据和 2 个需要学习的参数时，可以容易地画出损失函数 $\mathcal{L}(w, b)$ 随参数 (w, b) 变化的图，从而可以很简单地手动找到使 $(\mathcal{L}(w, b))$ 最小的参数值

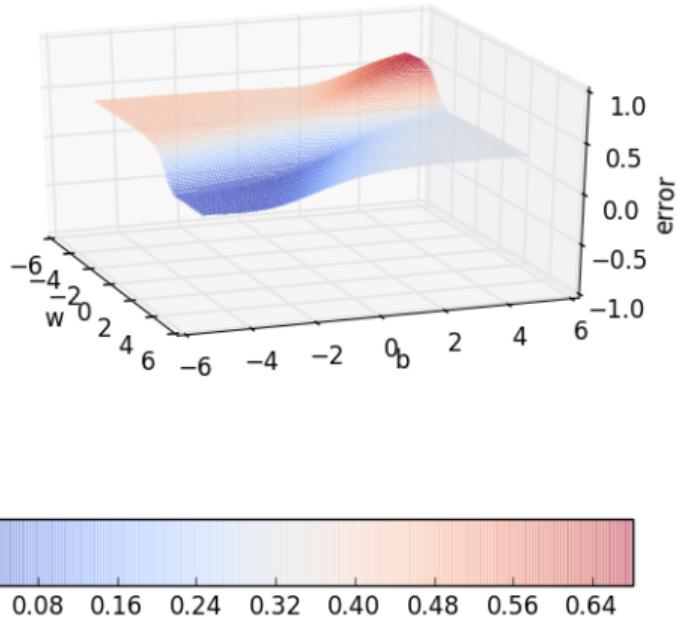
Random search on error surface



- 当只有两个训练数据和 2 个需要学习的参数时，可以容易地画出损失函数 $\mathcal{L}(w, b)$ 随参数 (w, b) 变化的图，从而可以很简单地手动找到使 $(\mathcal{L}(w, b))$ 最小的参数值
- 当有更多的训练数据和更多的参数时，手动找参数的方法变得不再可行



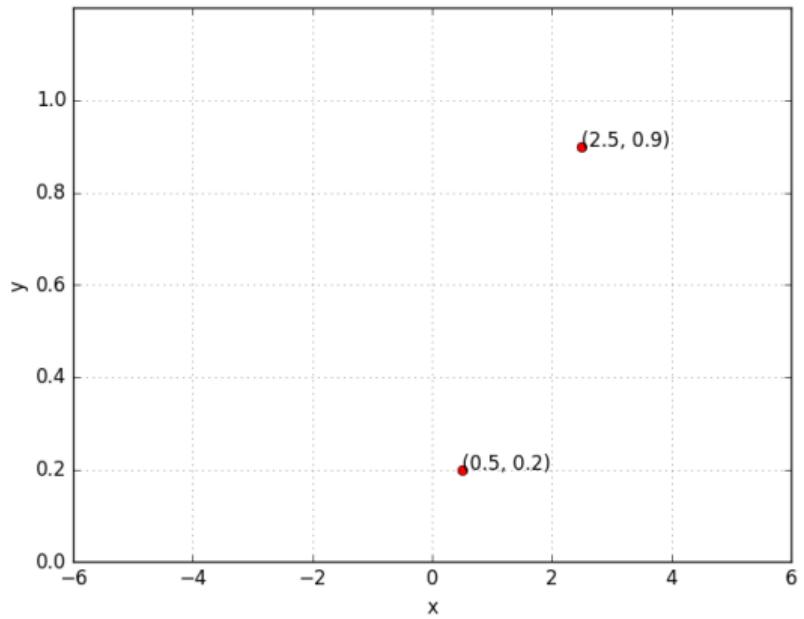
Random search on error surface



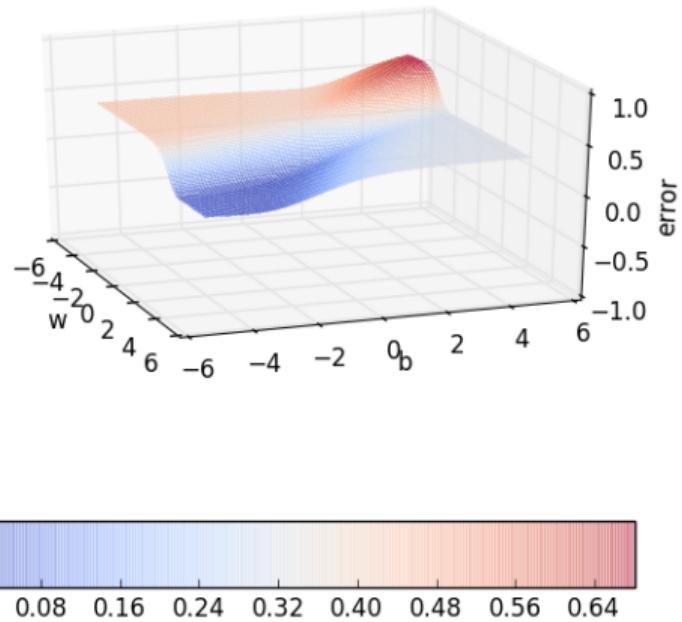
- 当只有两个训练数据和 2 个需要学习的参数时，可以容易地画出损失函数 $\mathcal{L}(w, b)$ 随参数 (w, b) 变化的图，从而可以很简单地手动找到使 $(\mathcal{L}(w, b))$ 最小的参数值
- 当有更多的训练数据和更多的参数时，手动找参数的方法变得不再可行
- 另外，我们也只能画出损失函数随参数在一个小的范围变化的图，而不是从 $(-\infty, \infty)$

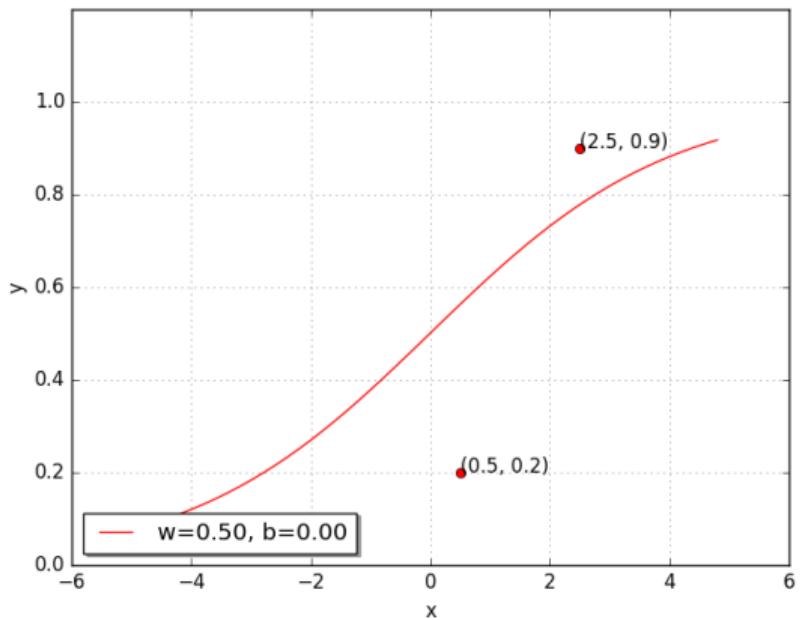


先看看上面随机猜测算法的几何解释

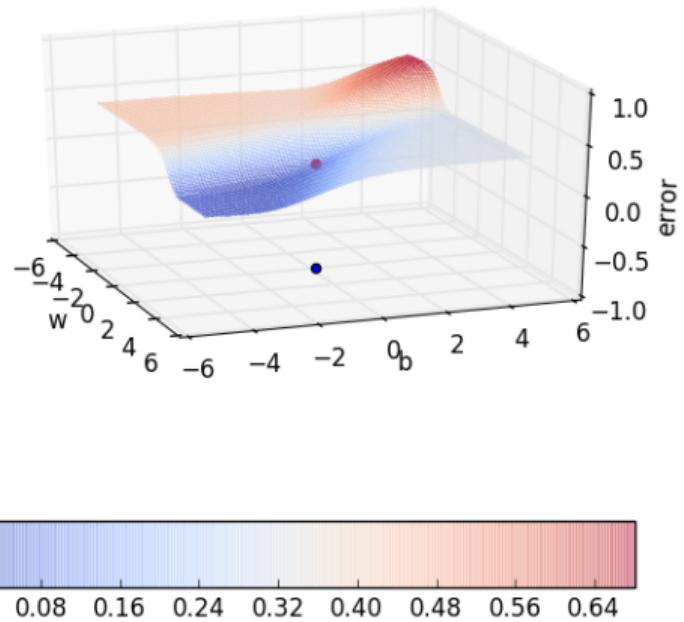


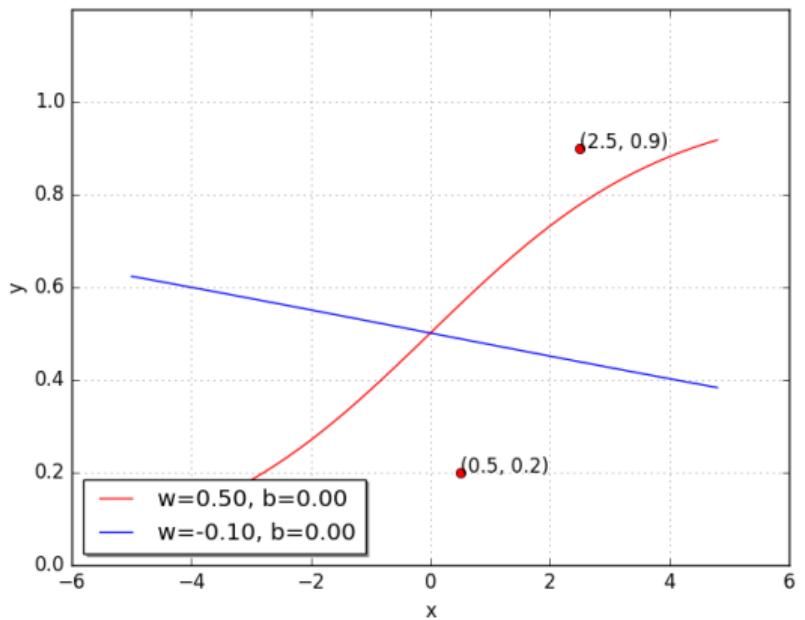
Random search on error surface



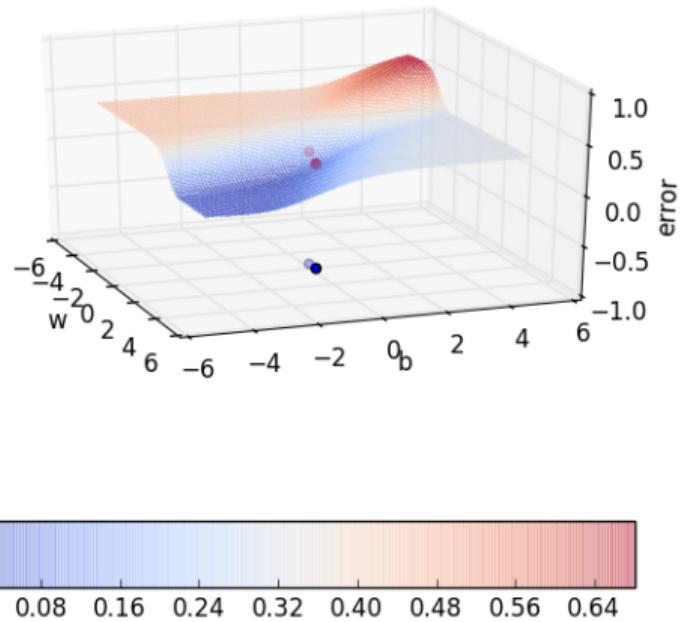


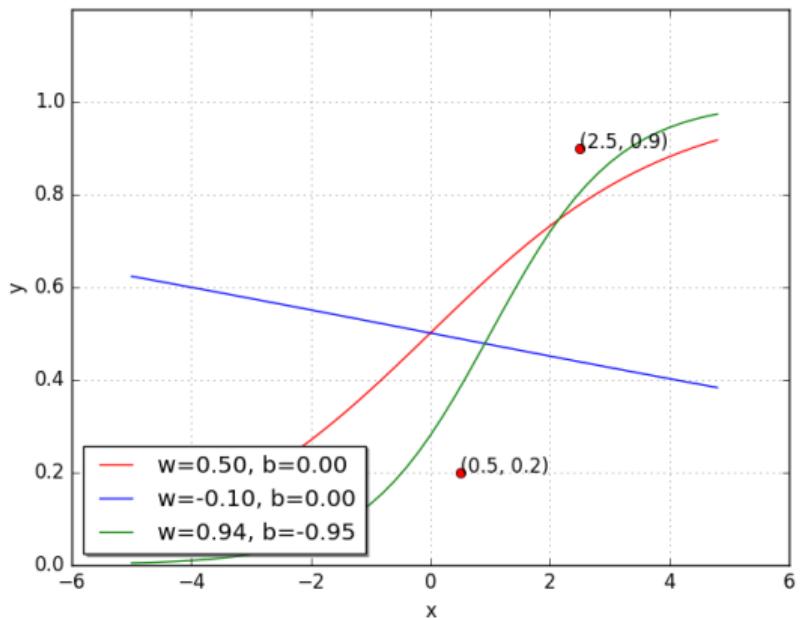
Random search on error surface



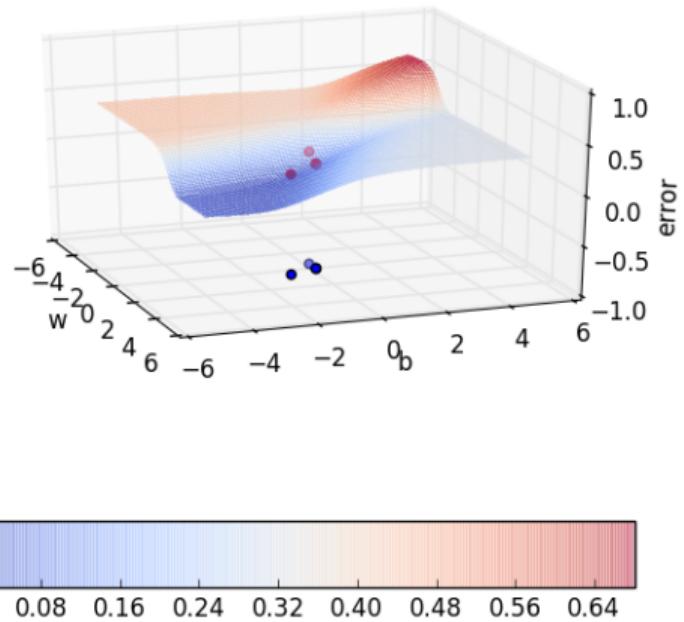


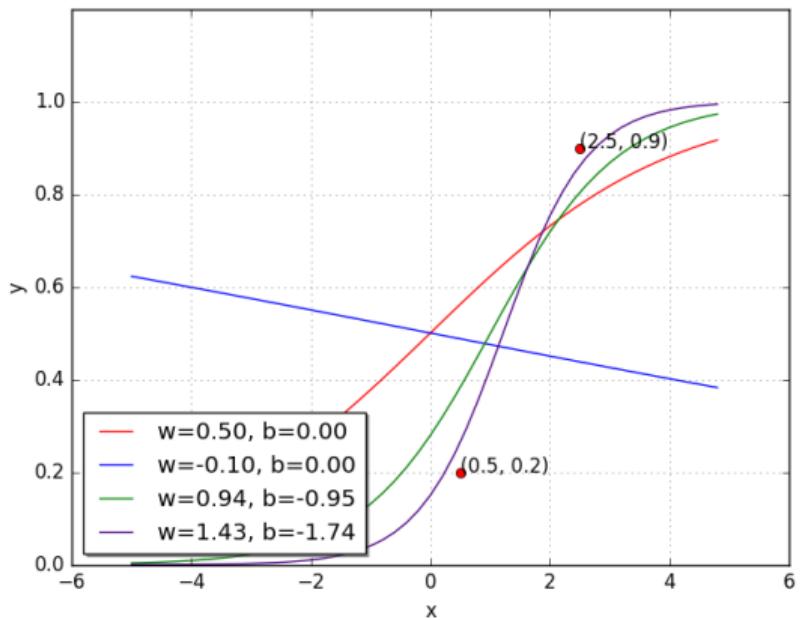
Random search on error surface



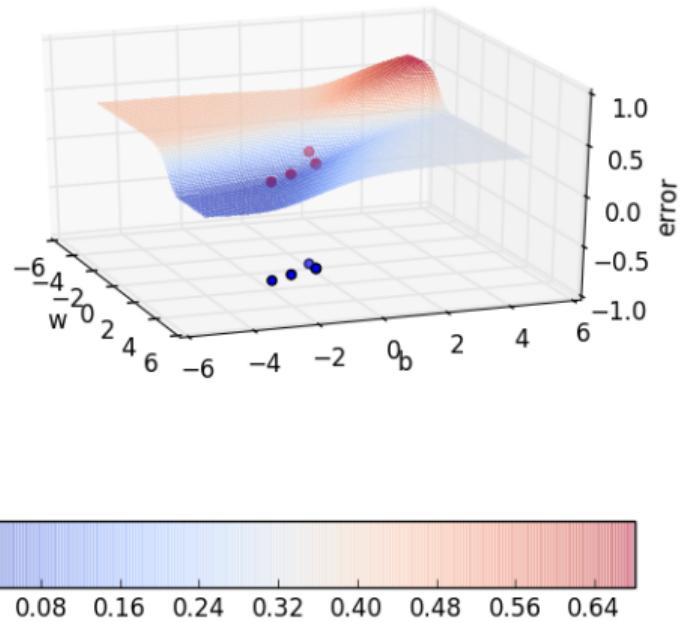


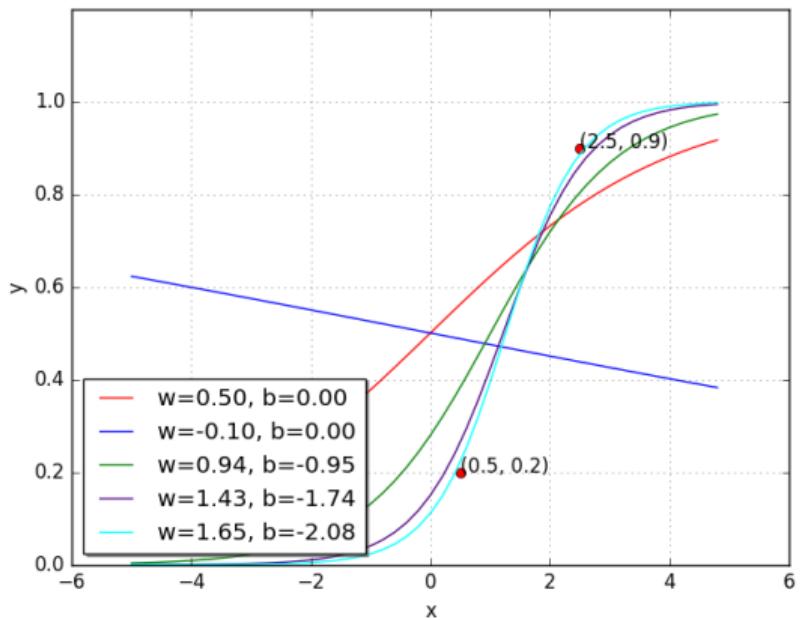
Random search on error surface



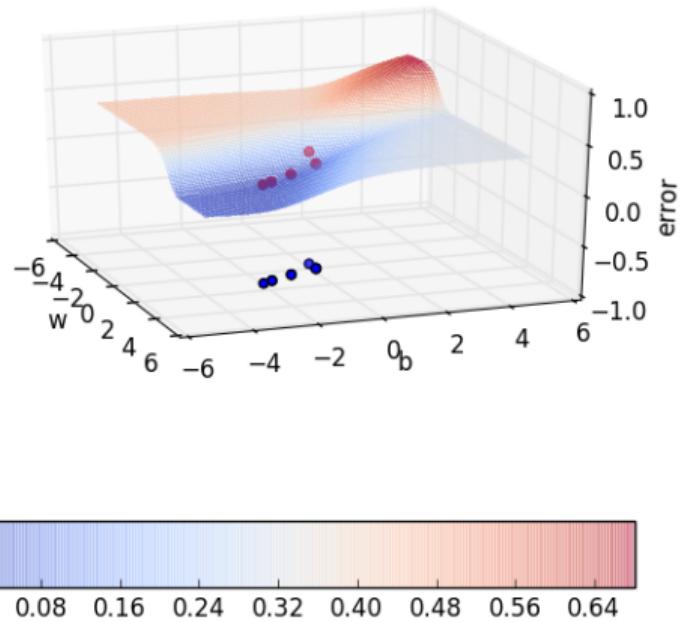


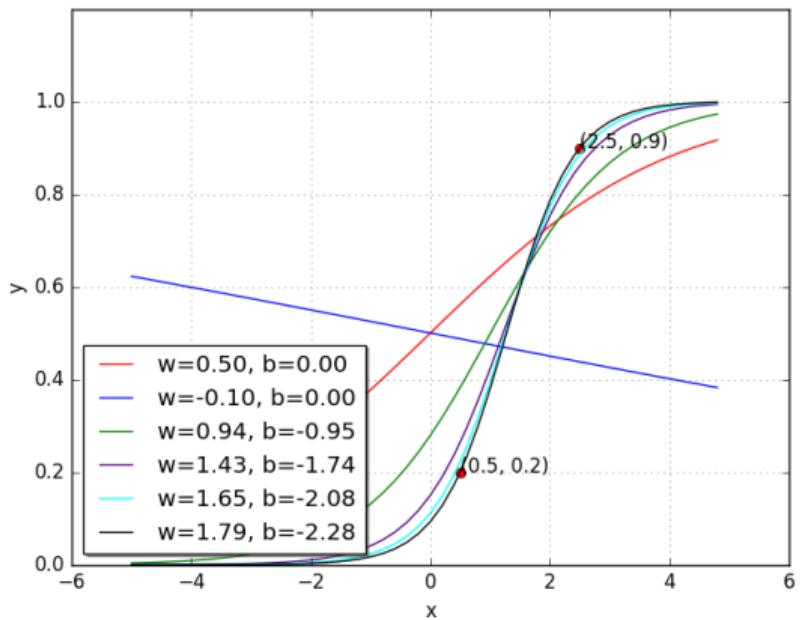
Random search on error surface



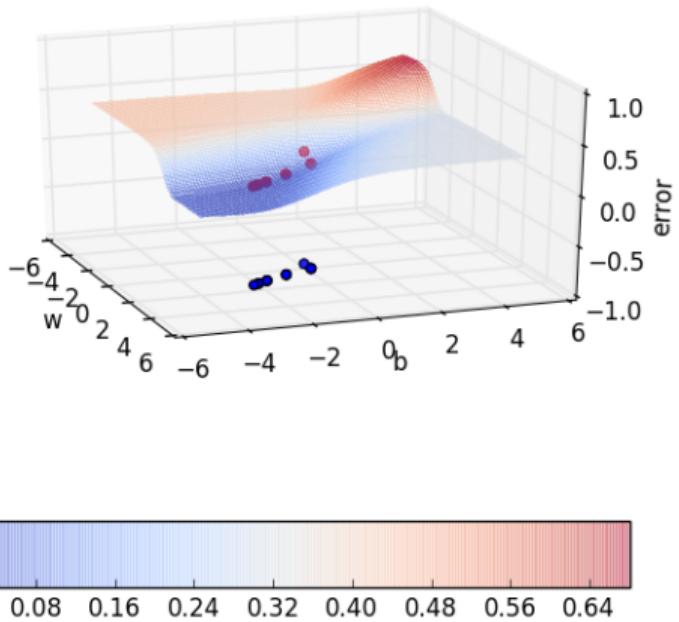


Random search on error surface





Random search on error surface



参数学习: 梯度下降 (Gradient Descent)



看看是否有比随机猜测更有效的方法



目标

找到一个更好的方法来遍历损失函数曲面从而达到迅速找到损失的最小值，而不是在损失函数上进行暴力搜索



随机初始化的参数向量

$$\theta = [w, b]$$



随机初始化的参数向量

$$\theta = [w, b]$$

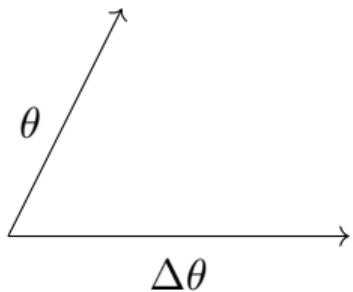
$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量



随机初始化的参数向量

$$\theta = [w, b]$$



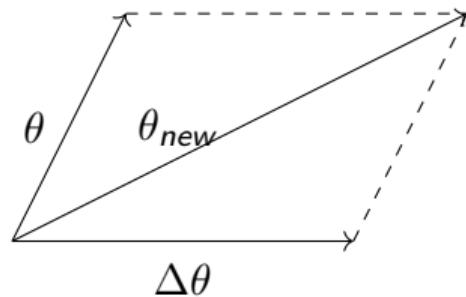
$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量



随机初始化的参数向量

$$\theta = [w, b]$$



$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量

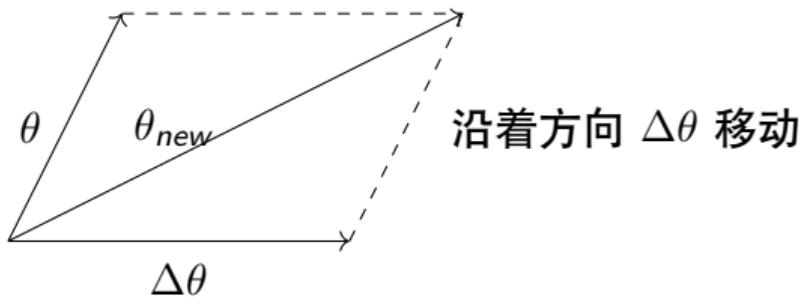


随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量



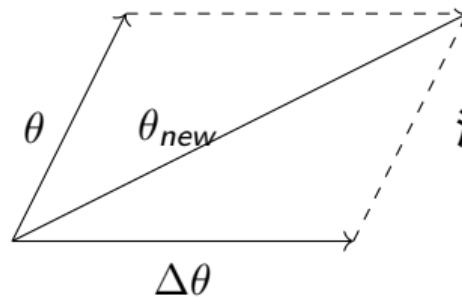


随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量



沿着方向 $\Delta\theta$ 移动

保守一点：只移动一小步
 η

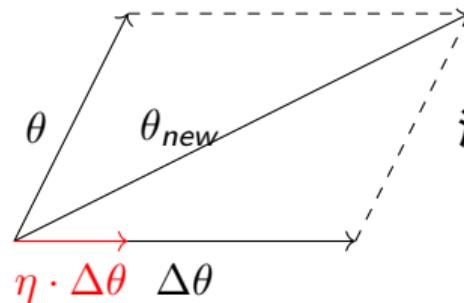


随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量



沿着方向 $\Delta\theta$ 移动

保守一点：只移动一小步
 η

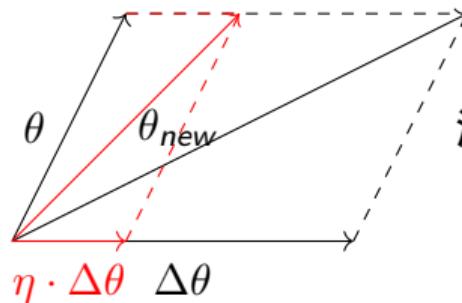


随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量



沿着方向 $\Delta\theta$ 移动

保守一点：只移动一小步
 η

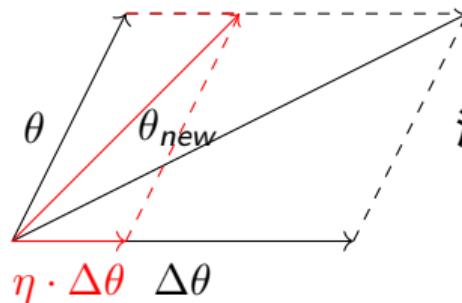
随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$



沿着方向 $\Delta\theta$ 移动

保守一点：只移动一小步
 η

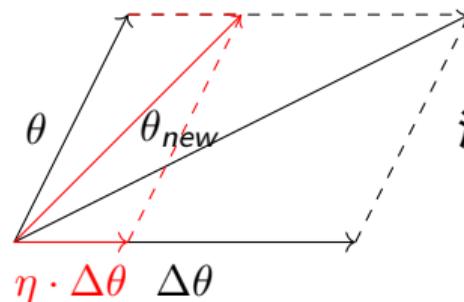
随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的
变化量

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$



沿着方向 $\Delta\theta$ 移动

保守一点：只移动一小步
 η

问题：如何计算 $\Delta\theta$?



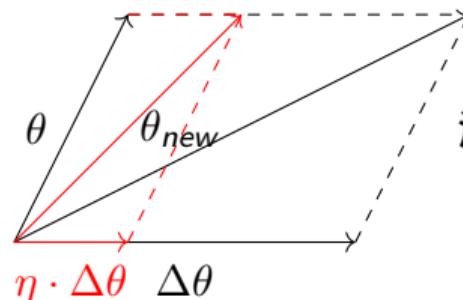
随机初始化的参数向量

$$\theta = [w, b]$$

$$\Delta\theta = [\Delta w, \Delta b]$$

参数 w 和 b 的变化量

$$\theta_{new} = \theta + \eta \cdot \Delta\theta$$



沿着方向 $\Delta\theta$ 移动

保守一点：只移动一小步
 η

问题：如何计算 $\Delta\theta$?

答案：使用泰勒展开式



为了表示方便, 让 $\Delta\theta = u$, 使用泰勒展开式, 可以得到

为了表示方便, 让 $\Delta\theta = u$, 使用泰勒展开式, 可以得到

$$\mathcal{L}(\theta + \eta u) = \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathcal{L}(\theta) u + \frac{\eta^3}{3!} * \dots + \frac{\eta^4}{4!} * \dots$$



为了表示方便, 让 $\Delta\theta = u$, 使用泰勒展开式, 可以得到

$$\begin{aligned}\mathcal{L}(\theta + \eta u) &= \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathcal{L}(\theta) u + \frac{\eta^3}{3!} * \dots + \frac{\eta^4}{4!} * \dots \\ &= \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) [\eta \text{ is typically small, so } \eta^2, \eta^3, \dots \rightarrow 0]\end{aligned}$$

为了表示方便, 让 $\Delta\theta = u$, 使用泰勒展开式, 可以得到

$$\begin{aligned}\mathcal{L}(\theta + \eta u) &= \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathcal{L}(\theta) u + \frac{\eta^3}{3!} * \dots + \frac{\eta^4}{4!} * \dots \\ &= \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) [\eta \text{ is typically small, so } \eta^2, \eta^3, \dots \rightarrow 0]\end{aligned}$$

注意: 移动 (ηu) 对降低损失函数有帮助, 只要当,

$$\mathcal{L}(\theta + \eta u) - \mathcal{L}(\theta) < 0 \text{ [i.e., 如果新的损失比先前的损失小]}$$

为了表示方便, 让 $\Delta\theta = u$, 使用泰勒展开式, 可以得到

$$\begin{aligned}\mathcal{L}(\theta + \eta u) &= \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) + \frac{\eta^2}{2!} * u^T \nabla^2 \mathcal{L}(\theta) u + \frac{\eta^3}{3!} * \dots + \frac{\eta^4}{4!} * \dots \\ &= \mathcal{L}(\theta) + \eta * u^T \nabla_{\theta} \mathcal{L}(\theta) [\eta \text{ is typically small, so } \eta^2, \eta^3, \dots \rightarrow 0]\end{aligned}$$

注意: 移动 (ηu) 对降低损失函数有帮助, 只要当,

$$\mathcal{L}(\theta + \eta u) - \mathcal{L}(\theta) < 0 \text{ [i.e., 如果新的损失比先前的损失小]}$$

这意味着,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$



因此有,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$

但是 $u^T \nabla_{\theta} \mathcal{L}(\theta)$ 最小能有多小?



因此有,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$

但是 $u^T \nabla_{\theta} \mathcal{L}(\theta)$ 最小能有多小? 继续看看....



因此有,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$

但是 $u^T \nabla_{\theta} \mathcal{L}(\theta)$ 最小能有多小? 继续看看....

让 β 是 u 和 $\nabla_{\theta} \mathcal{L}(\theta)$ 的角度, 有,

因此有,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$

但是 $u^T \nabla_{\theta} \mathcal{L}(\theta)$ 最小能有多小? 继续看看....

让 β 是 u 和 $\nabla_{\theta} \mathcal{L}(\theta)$ 的角度, 有,

$$-1 \leq \cos(\beta) = \frac{u^T \nabla_{\theta} \mathcal{L}(\theta)}{\|u\| * \|\nabla_{\theta} \mathcal{L}(\theta)\|} \leq 1$$

因此有,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$

但是 $u^T \nabla_{\theta} \mathcal{L}(\theta)$ 最小能有多小? 继续看看....

让 β 是 u 和 $\nabla_{\theta} \mathcal{L}(\theta)$ 的角度, 有,

$$-1 \leq \cos(\beta) = \frac{u^T \nabla_{\theta} \mathcal{L}(\theta)}{\|u\| * \|\nabla_{\theta} \mathcal{L}(\theta)\|} \leq 1$$

上面的不等式两边乘以 $k = \|u\| * \|\nabla_{\theta} \mathcal{L}(\theta)\|$

$$-k \leq k * \cos(\beta) = u^T \nabla_{\theta} \mathcal{L}(\theta) \leq k$$

因此有,

$$u^T \nabla_{\theta} \mathcal{L}(\theta) < 0$$

但是 $u^T \nabla_{\theta} \mathcal{L}(\theta)$ 最小能有多小? 继续看看....

让 β 是 u 和 $\nabla_{\theta} \mathcal{L}(\theta)$ 的角度, 有,

$$-1 \leq \cos(\beta) = \frac{u^T \nabla_{\theta} \mathcal{L}(\theta)}{\|u\| * \|\nabla_{\theta} \mathcal{L}(\theta)\|} \leq 1$$

上面的不等式两边乘以 $k = \|u\| * \|\nabla_{\theta} \mathcal{L}(\theta)\|$

$$-k \leq k * \cos(\beta) = u^T \nabla_{\theta} \mathcal{L}(\theta) \leq k$$

因此, 当 $\cos(\beta) = -1$ i.e., 当 β 是 180° 时,

$\mathcal{L}(\theta + \eta u) - \mathcal{L}(\theta) = u^T \nabla_{\theta} \mathcal{L}(\theta) = k * \cos(\beta)$ 将达到最小的值



梯度下降规则

- 移动的方向 u 应该是与梯度的方向相差 180°



梯度下降规则

- 移动的方向 u 应该是与梯度的方向相差 180°
- 也就是沿着梯度的反方向移动

梯度下降规则

- 移动的方向 u 应该是与梯度的方向相差 180°
- 也就是沿着梯度的反方向移动

参数更新公式

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

where, $\nabla w_t = \frac{\partial \mathcal{L}(w, b)}{\partial w}$ at $w = w_t, b = b_t$, $\nabla b = \frac{\partial \mathcal{L}(w, b)}{\partial b}$ at $w = w_t, b = b_t$

梯度下降规则

- 移动的方向 u 应该是与梯度的方向相差 180°
- 也就是沿着梯度的反方向移动

参数更新公式

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

where, $\nabla w_t = \frac{\partial \mathcal{L}(w, b)}{\partial w}$ at $w = w_t, b = b_t$, $\nabla b = \frac{\partial \mathcal{L}(w, b)}{\partial b}$ at $w = w_t, b = b_t$

因此，得到一个比随机猜测更好方法来移动 w 和 b



- 基于上面对参数更新的规则，得到梯度下降算法：



- 基于上面对参数更新的规则，得到梯度下降算法：

Algorithm 2: gradient_descent()

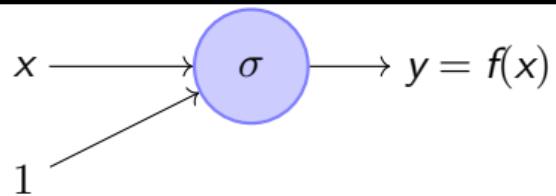
```
t ← 0;  
max_iterations ← 1000;  
while  $t < max\_iterations$  do  
     $w_{t+1} \leftarrow w_t - \eta \nabla w_t;$   
     $b_{t+1} \leftarrow b_t - \eta \nabla b_t;$   
     $t \leftarrow t + 1;$   
end
```

- 基于上面对参数更新的规则，得到梯度下降算法：

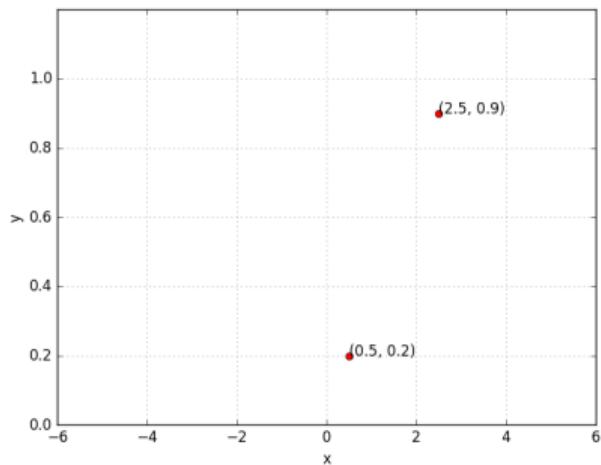
Algorithm 3: gradient_descent()

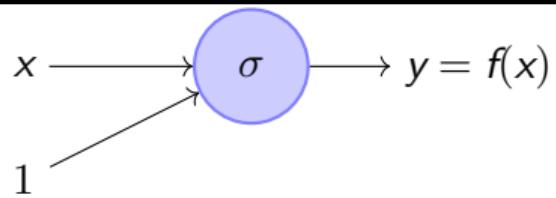
```
t ← 0;  
max_iterations ← 1000;  
while  $t < max\_iterations$  do  
     $w_{t+1} \leftarrow w_t - \eta \nabla w_t;$   
     $b_{t+1} \leftarrow b_t - \eta \nabla b_t;$   
     $t \leftarrow t + 1;$   
end
```

- 使用这个算法，首先需要计算 ∇w 和 ∇b



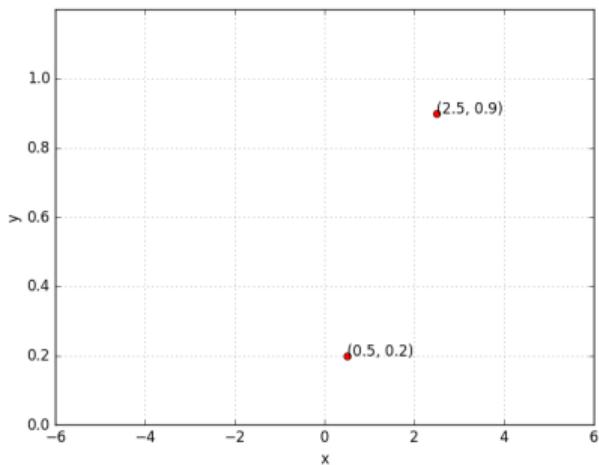
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

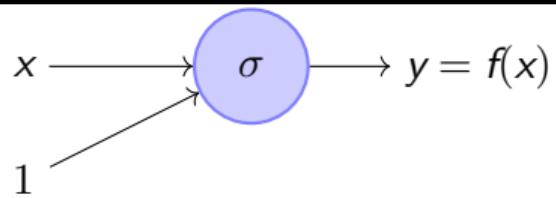




$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

假设只有一个训练数据 (x, y)

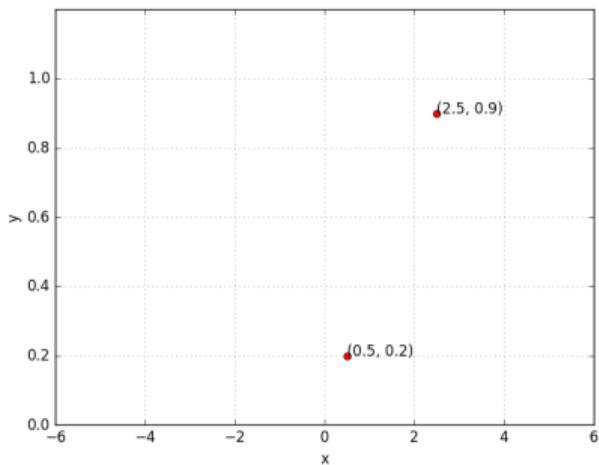


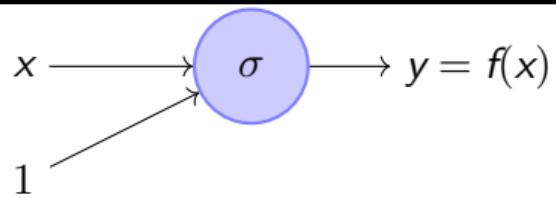


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

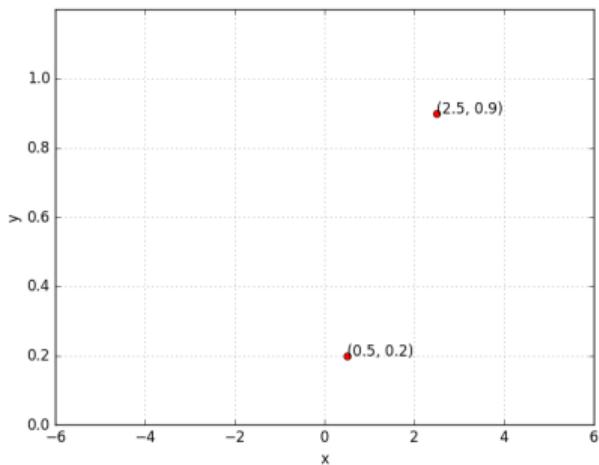
假设只有一个训练数据 (x, y)

$$\mathcal{L}(w, b) = \frac{1}{2} * (f(x) - y)^2$$





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



假设只有一个训练数据 (x, y)

$$\mathcal{L}(w, b) = \frac{1}{2} * (f(x) - y)^2$$

$$\nabla w = \frac{\partial \mathcal{L}(w, b)}{\partial w} = \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right]$$



$$\nabla w = \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right]$$



$$\begin{aligned}\nabla w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\ &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)]\end{aligned}$$



$$\begin{aligned}\nabla w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\ &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\ &= (f(x) - y) * \frac{\partial}{\partial w} (f(x))\end{aligned}$$

$$\begin{aligned}\nabla w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\&= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\&= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)\end{aligned}$$

$$\begin{aligned}\nabla w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\&= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\&= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)\end{aligned}$$

$$\frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)$$

$$\begin{aligned}
 \nabla w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\
 &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\
 &= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\
 &= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)
 \end{aligned}$$

$$\begin{aligned}
 &\frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w} (e^{-(wx+b)})
 \end{aligned}$$

$$\begin{aligned}
 \nabla_w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\
 &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\
 &= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\
 &= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)
 \end{aligned}$$

$$\begin{aligned}
 &\frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w} (e^{-(wx+b)}) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w} (-wx - b)
 \end{aligned}$$

$$\begin{aligned}
 \nabla_w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\
 &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\
 &= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\
 &= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)
 \end{aligned}$$

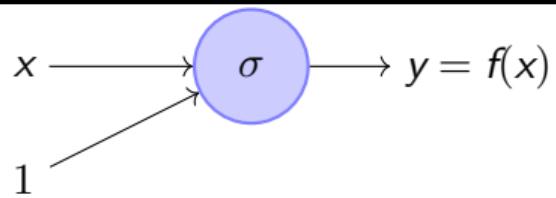
$$\begin{aligned}
 &\frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w} (e^{-(wx+b)}) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w} (-wx - b) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x)
 \end{aligned}$$

$$\begin{aligned}
 \nabla_w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\
 &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\
 &= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\
 &= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right)
 \end{aligned}$$

$$\begin{aligned}
 &\frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w} (e^{-(wx+b)}) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w} (-wx - b) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x) \\
 &= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x)
 \end{aligned}$$

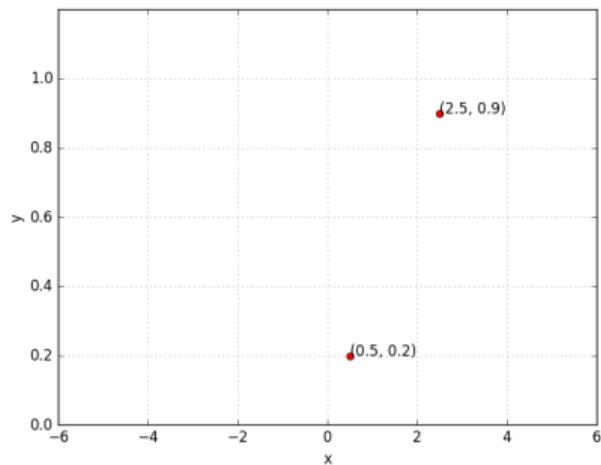
$$\begin{aligned}
 \nabla w &= \frac{\partial}{\partial w} \left[\frac{1}{2} * (f(x) - y)^2 \right] \\
 &= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w} (f(x) - y)] \\
 &= (f(x) - y) * \frac{\partial}{\partial w} (f(x)) \\
 &= (f(x) - y) * \frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right) \\
 &= (\textcolor{red}{f(x) - y}) * f(x) * (1 - f(x)) * x
 \end{aligned}$$

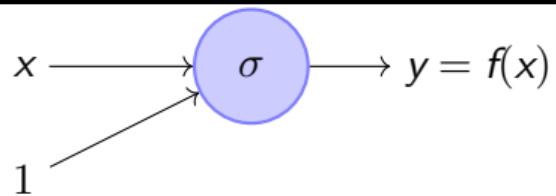
$$\begin{aligned}
 &\frac{\partial}{\partial w} \left(\frac{1}{1 + e^{-(wx+b)}} \right) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w} (e^{-(wx+b)}) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w} (-wx - b) \\
 &= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x) \\
 &= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x) \\
 &= f(x) * (1 - f(x)) * x
 \end{aligned}$$



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

因此如果只有一个训练数据 (x, y) , 可以得到,

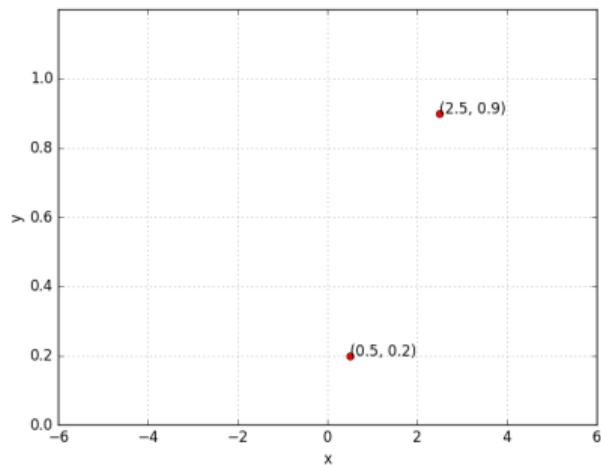


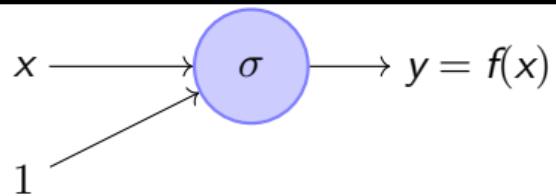


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

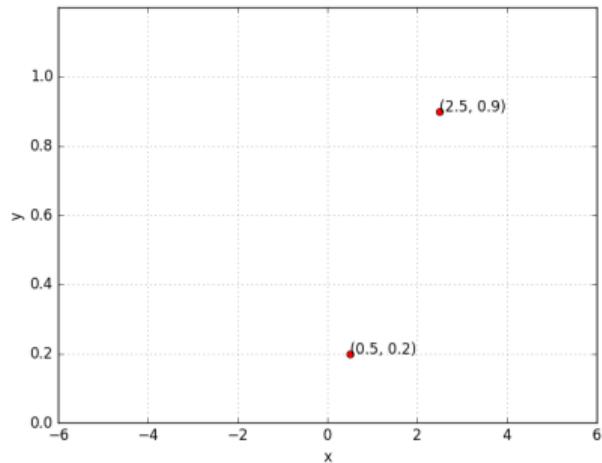
因此如果只有一个训练数据 (x, y) , 可以得到,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$





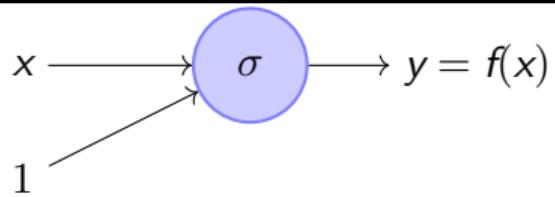
$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$



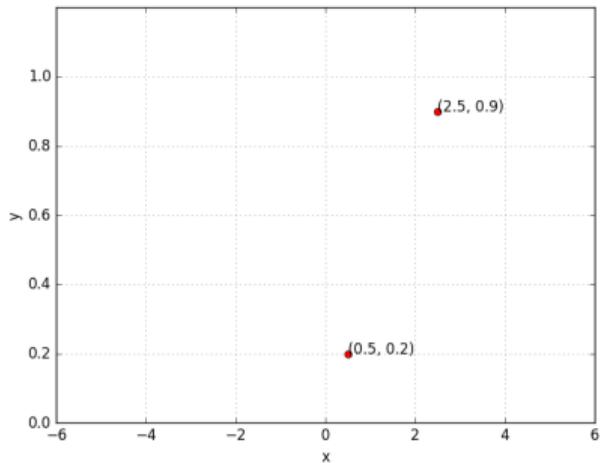
因此如果只有一个训练数据 (x, y) , 可以得到,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

如果有两个数据,



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

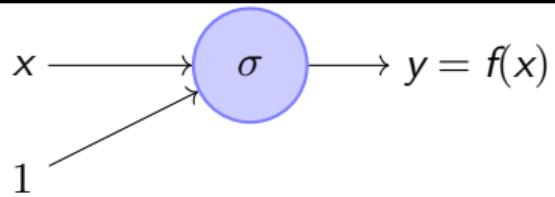


因此如果只有一个训练数据 (x, y) , 可以得到,

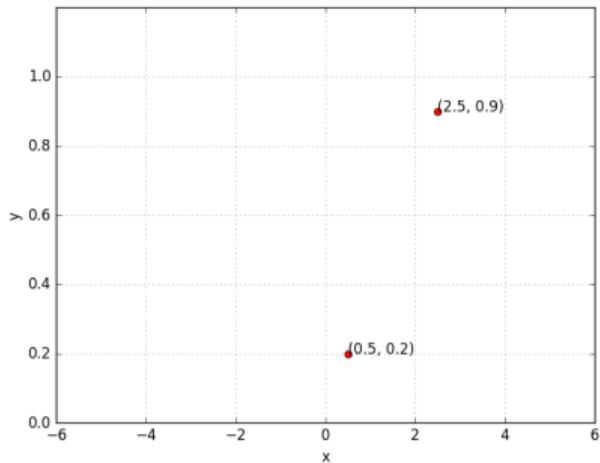
$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

如果有两个数据,

$$\nabla w = \sum_{i=1}^2 (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$



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



因此如果只有一个训练数据 (x, y) , 可以得到,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

如果有两个数据,

$$\nabla w = \sum_{i=1}^2 (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$

$$\nabla b = \sum_{i=1}^2 (f(x_i) - y_i) * f(x_i) * (1 - f(x_i))$$

```
X = [0.5, 2.5]  
Y = [0.2, 0.9]
```

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))
```

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

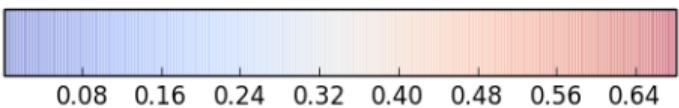
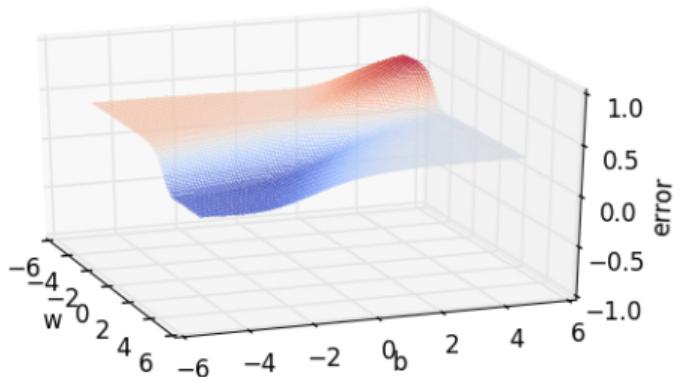
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err
```

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err
```

Random search on error surface



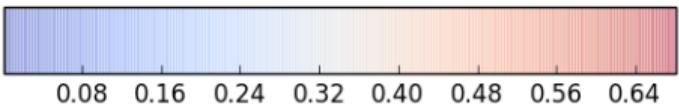
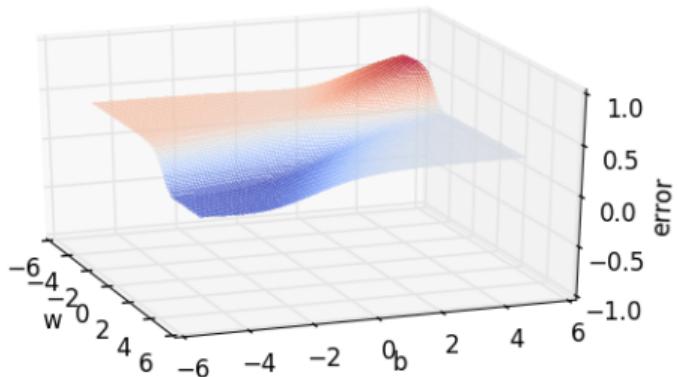
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)
```

Random search on error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

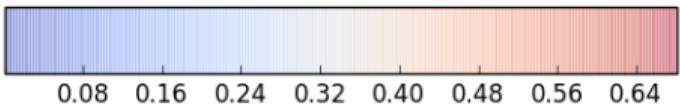
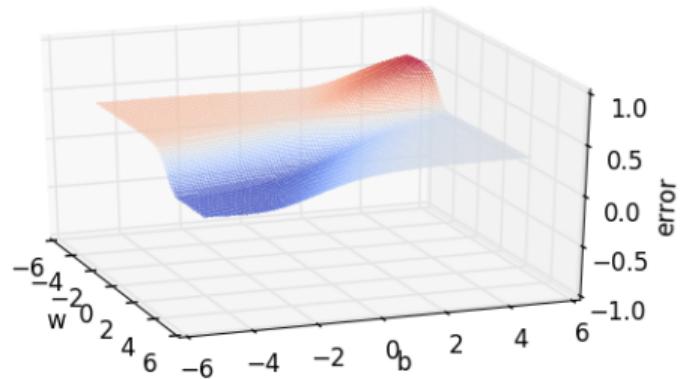
def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x
```

Random search on error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

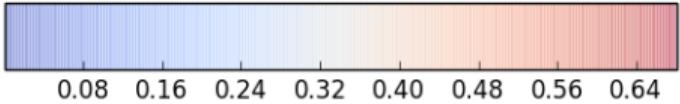
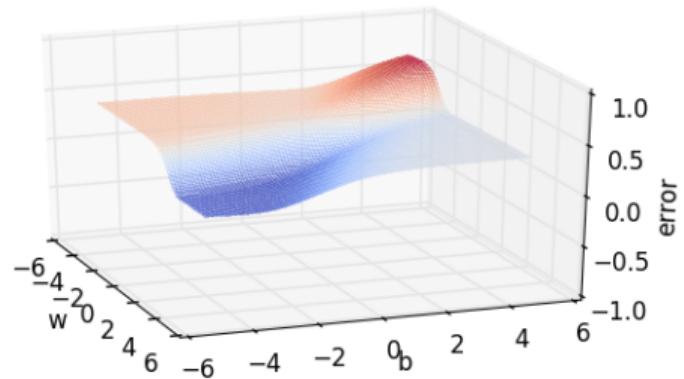
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Random search on error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

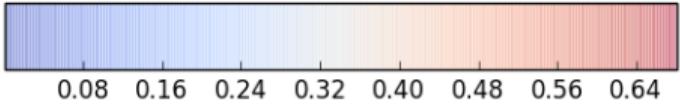
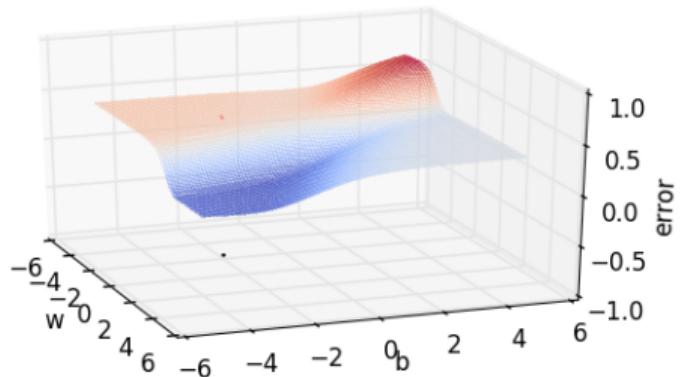
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

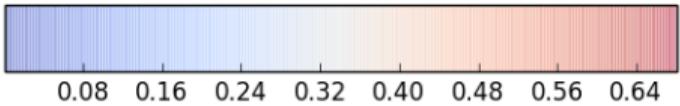
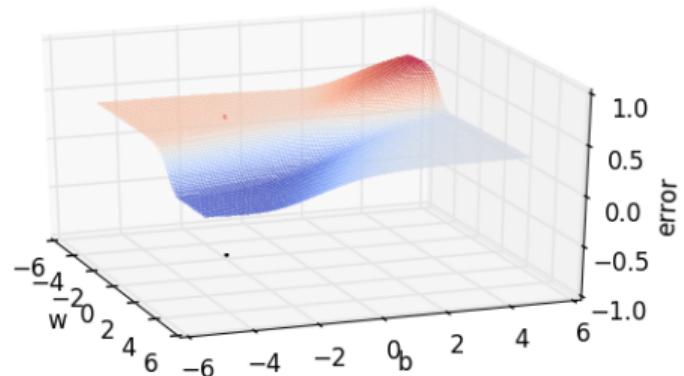
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

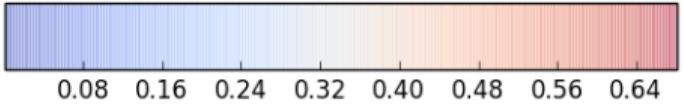
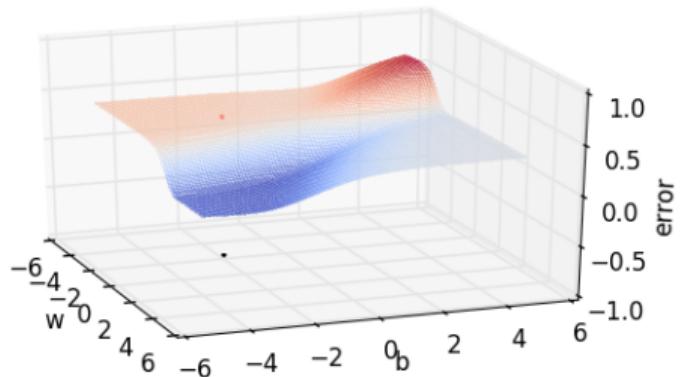
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

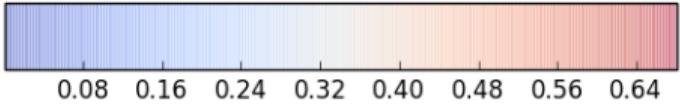
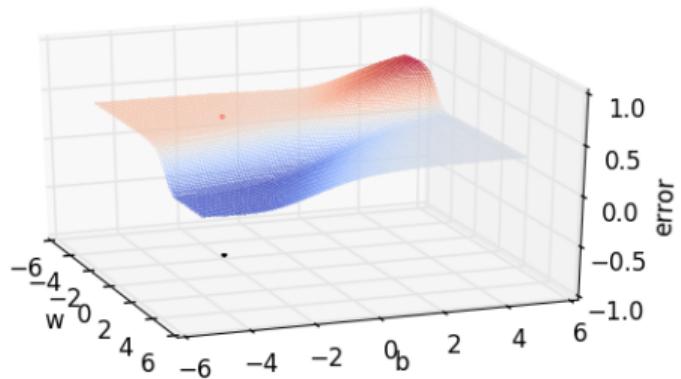
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

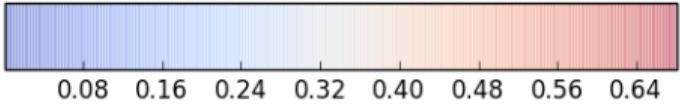
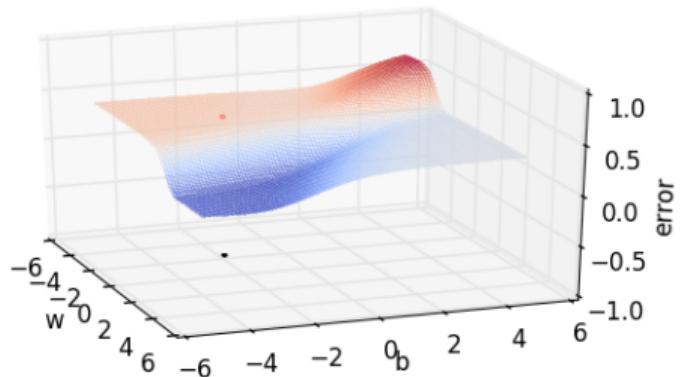
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

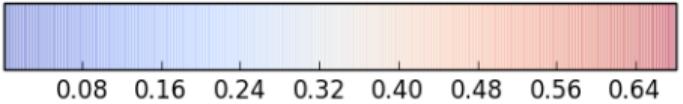
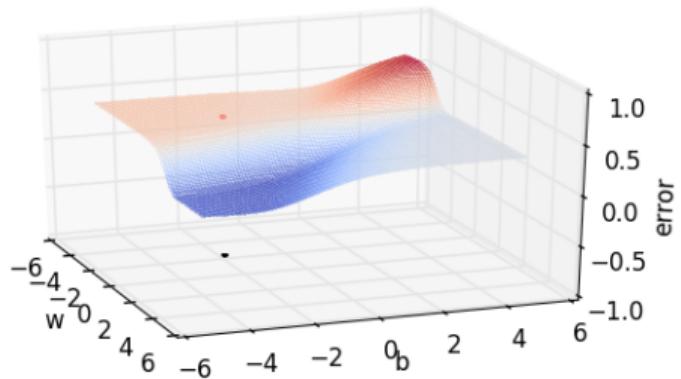
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

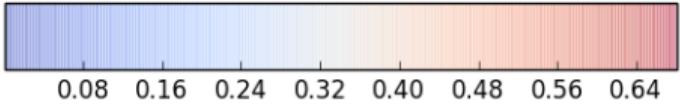
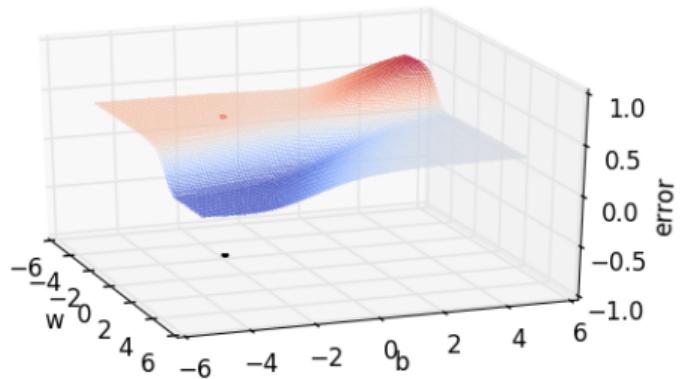
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

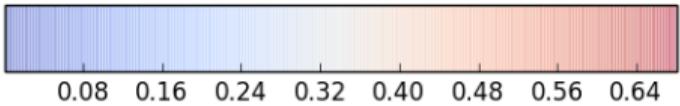
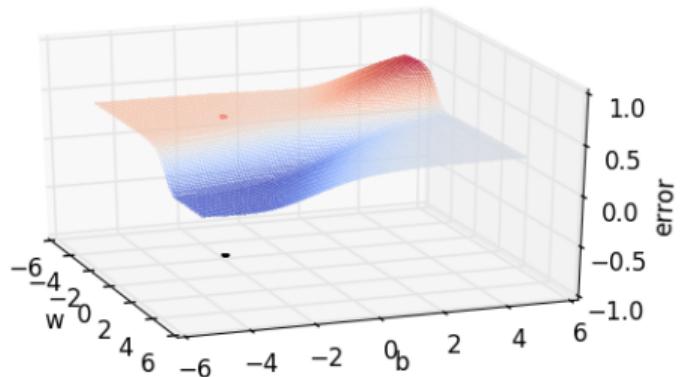
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

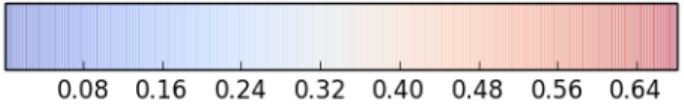
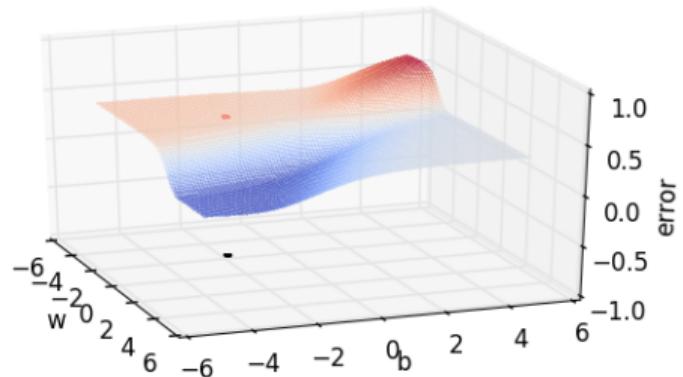
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

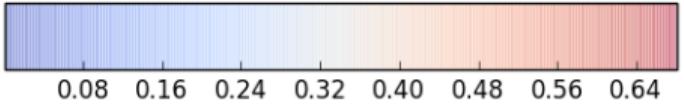
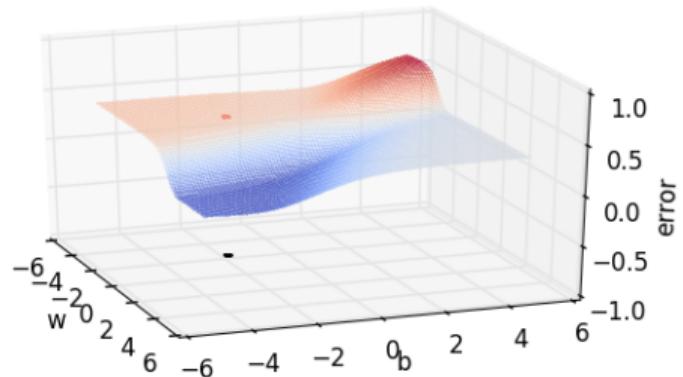
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

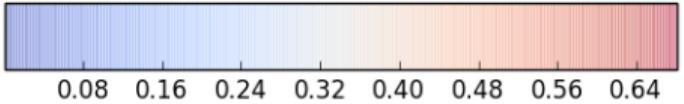
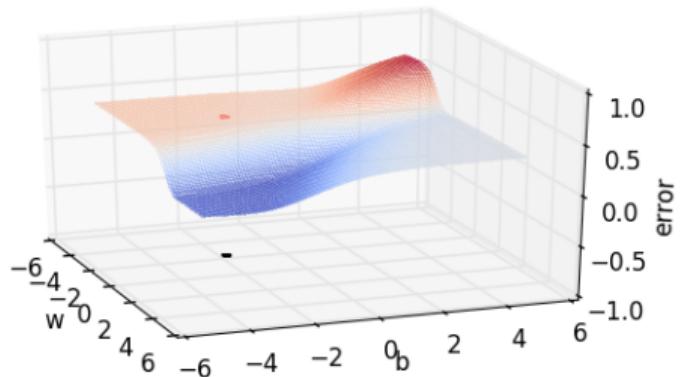
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

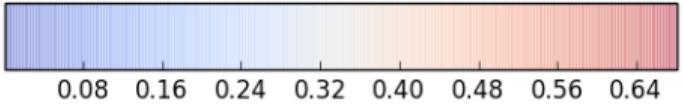
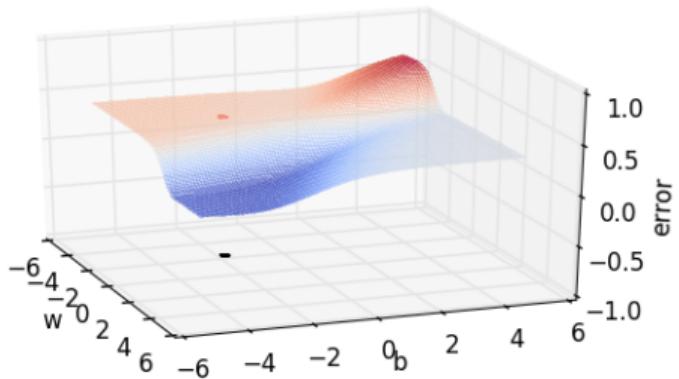
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

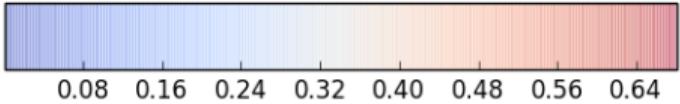
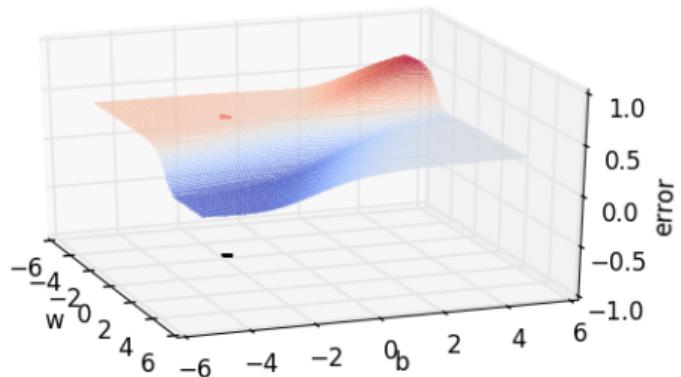
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

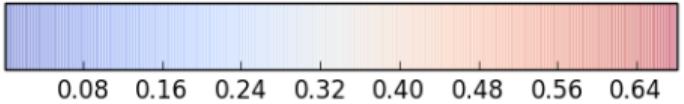
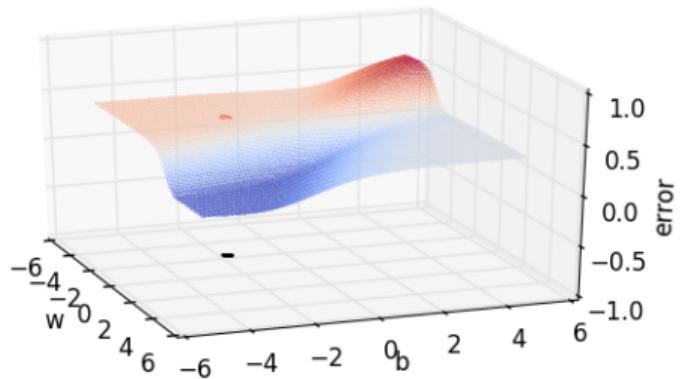
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

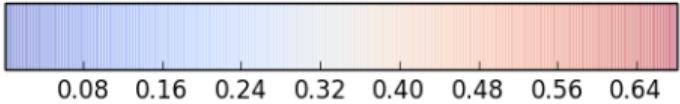
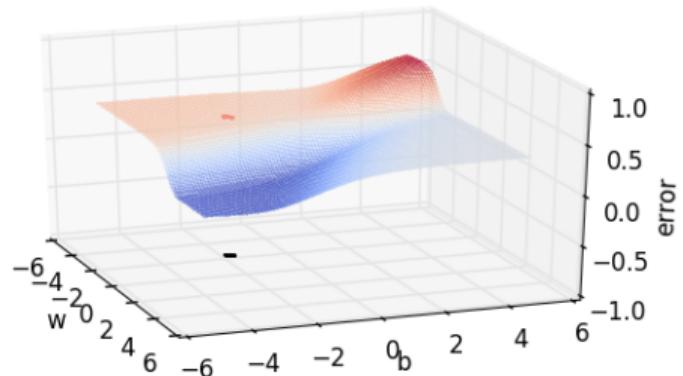
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

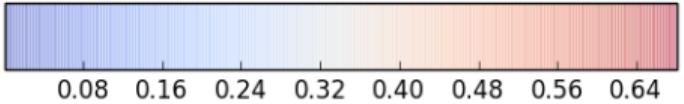
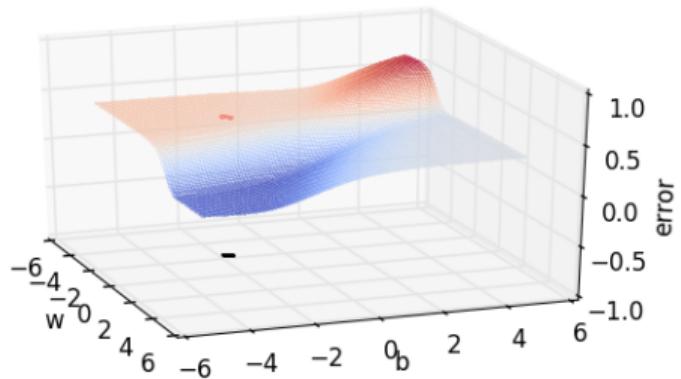
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

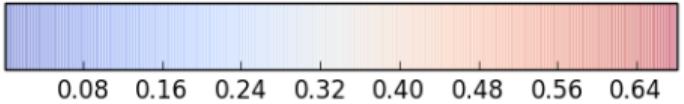
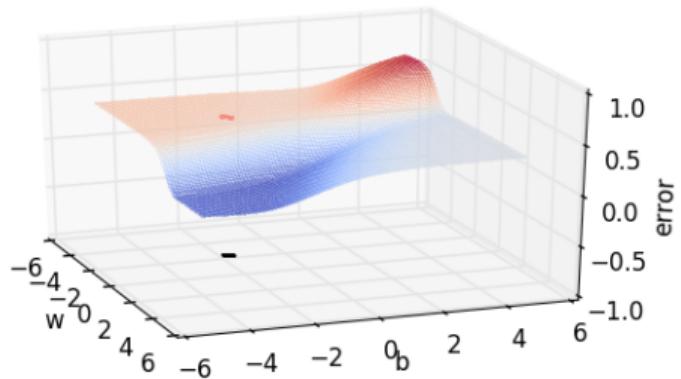
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

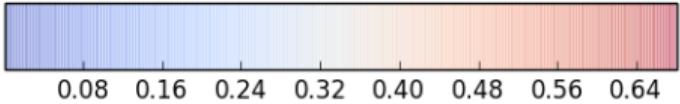
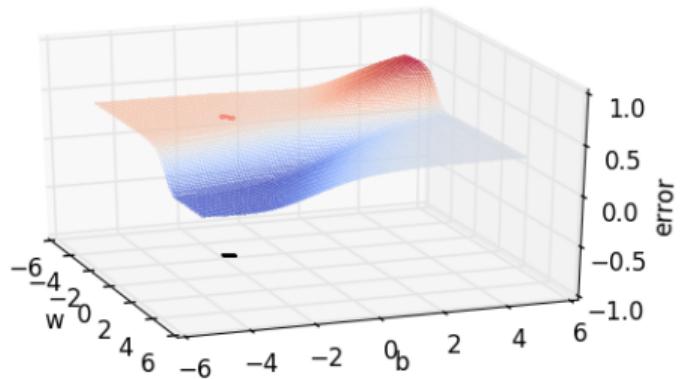
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

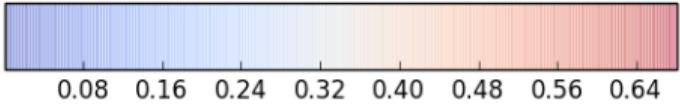
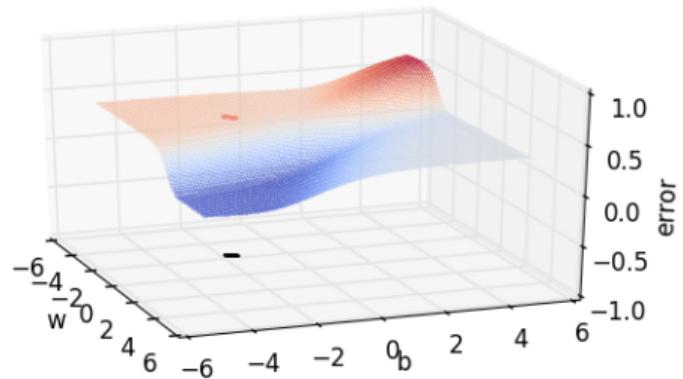
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

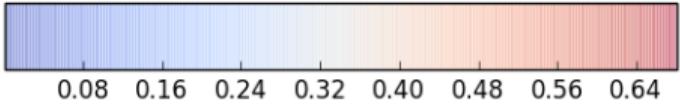
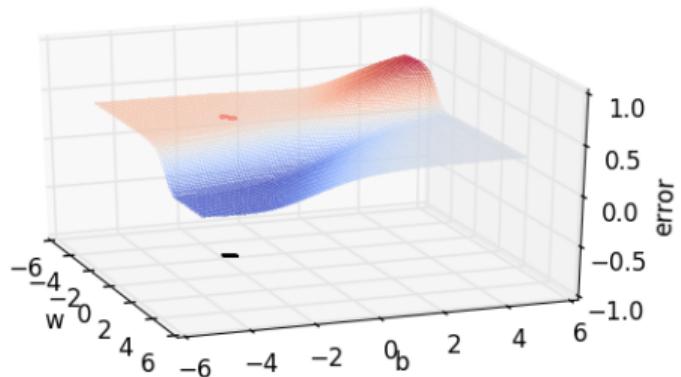
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

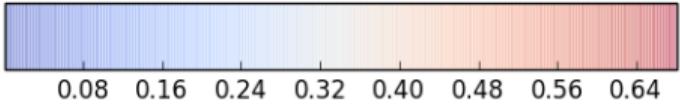
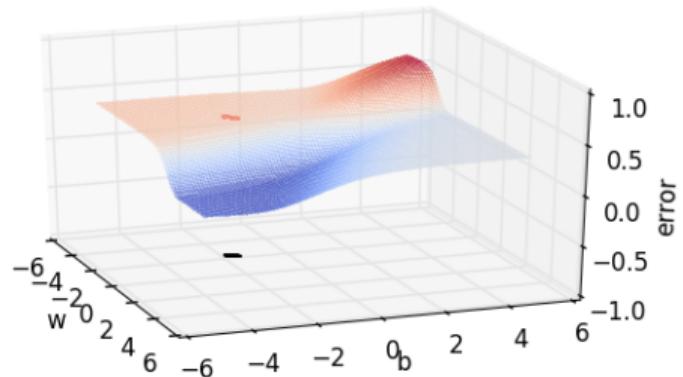
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

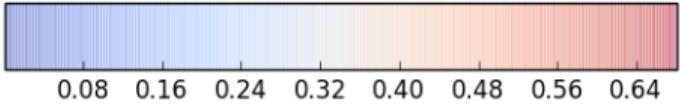
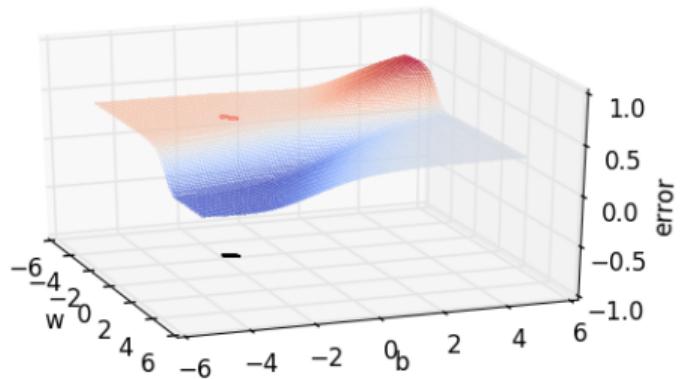
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

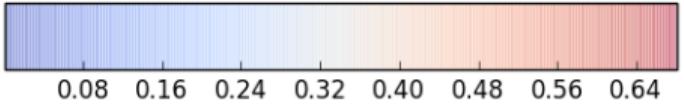
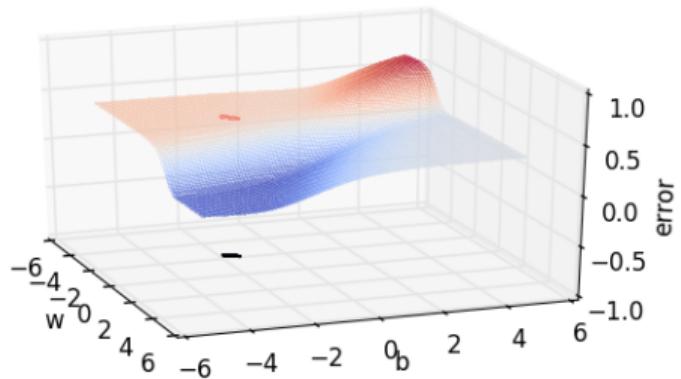
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

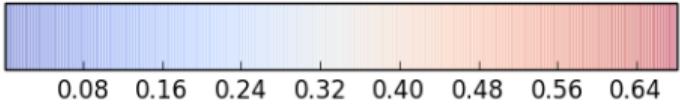
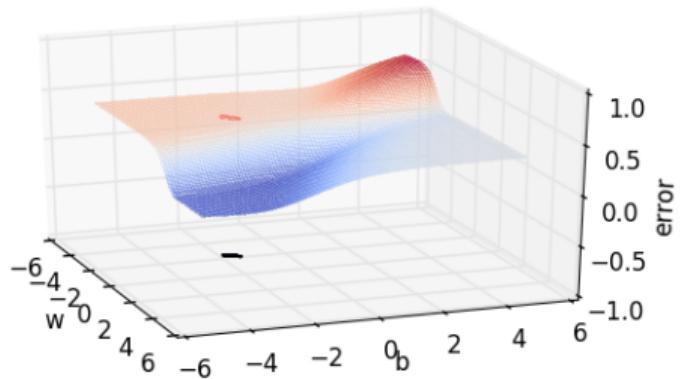
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

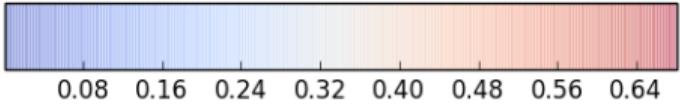
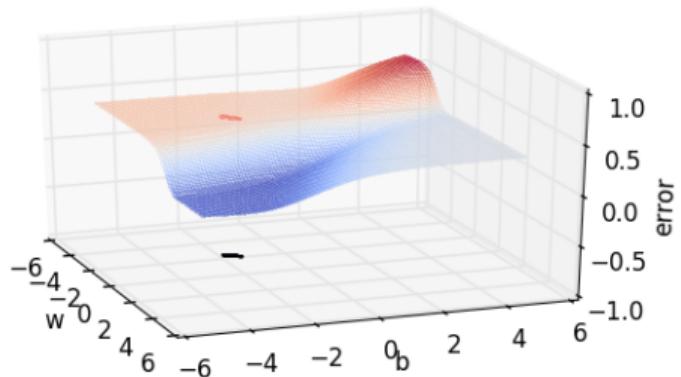
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

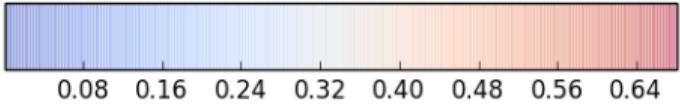
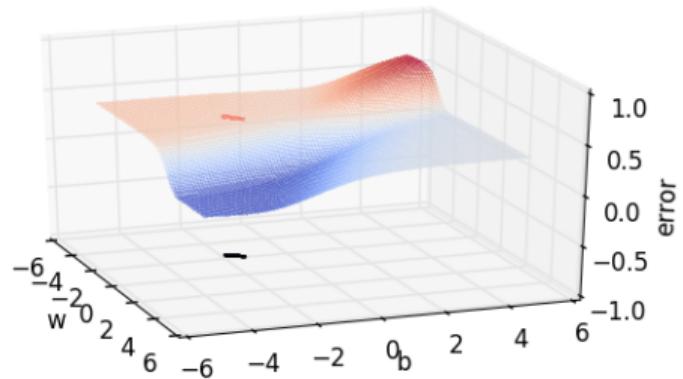
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

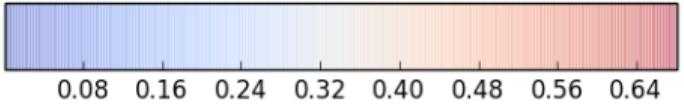
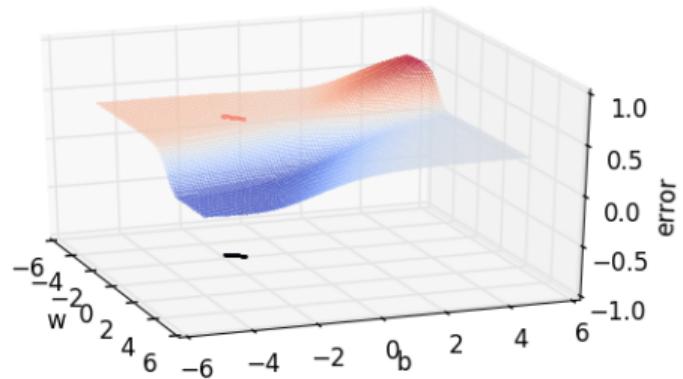
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

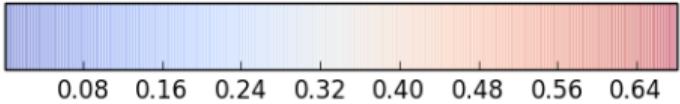
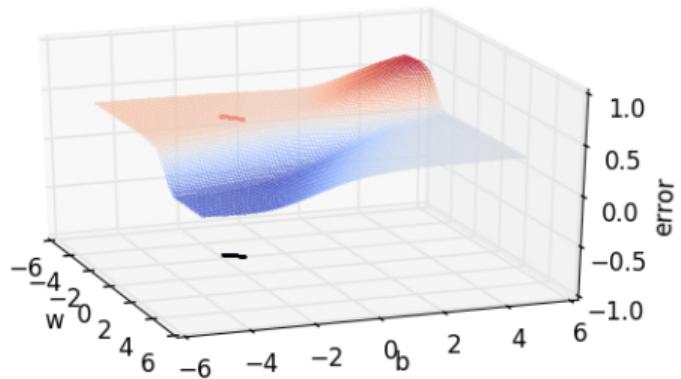
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

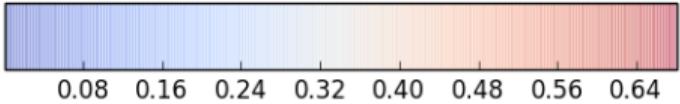
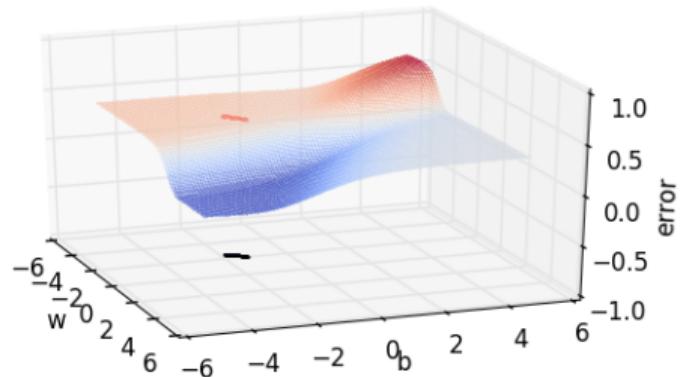
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

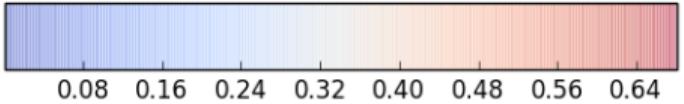
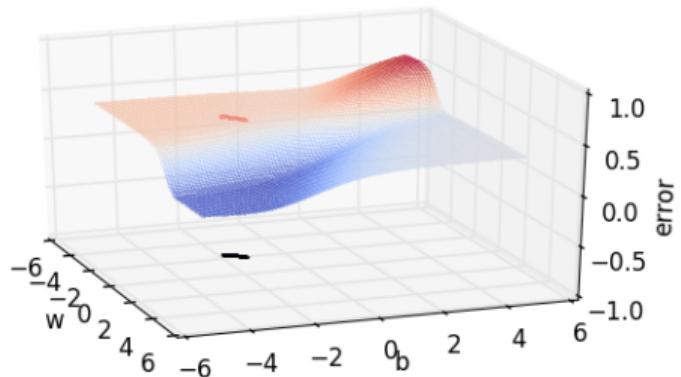
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

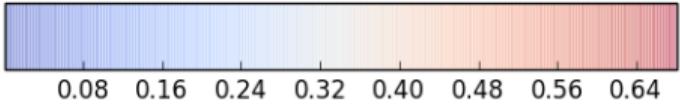
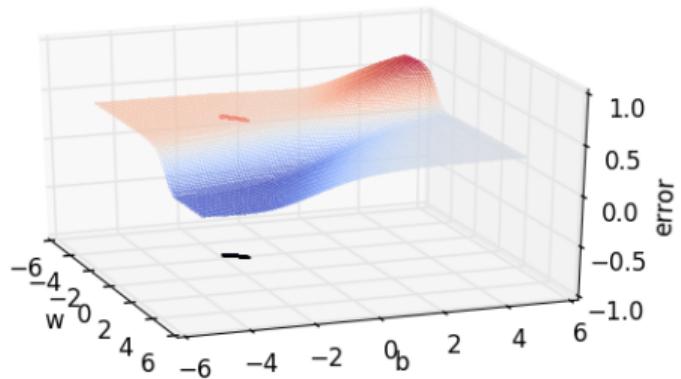
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

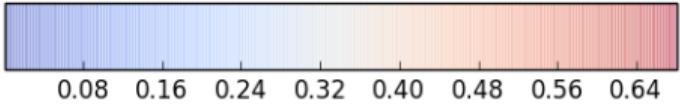
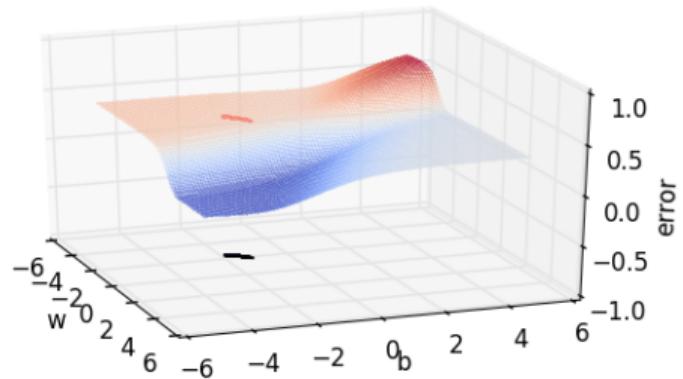
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

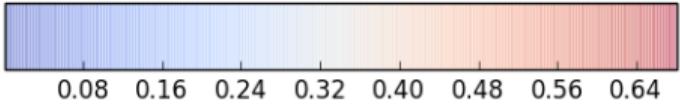
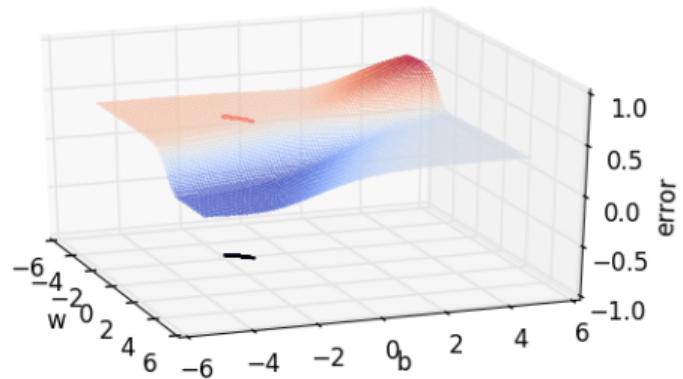
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

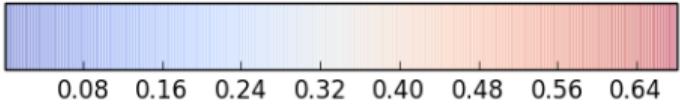
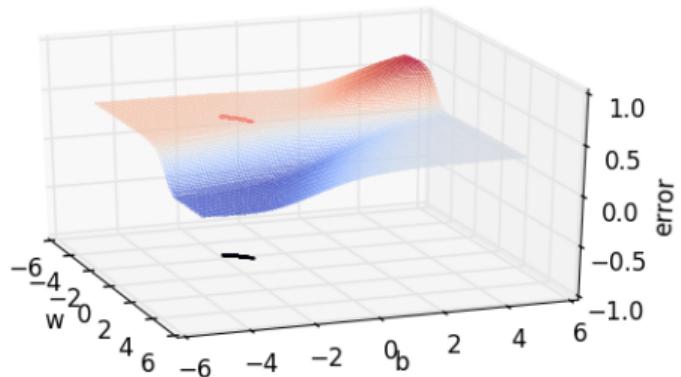
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

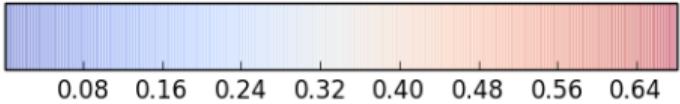
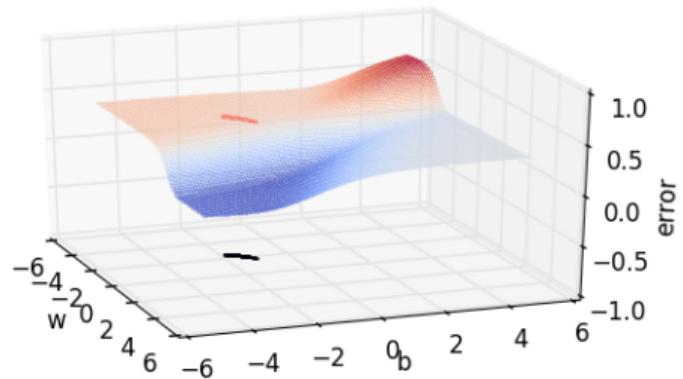
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

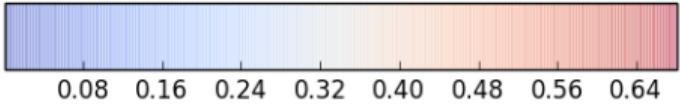
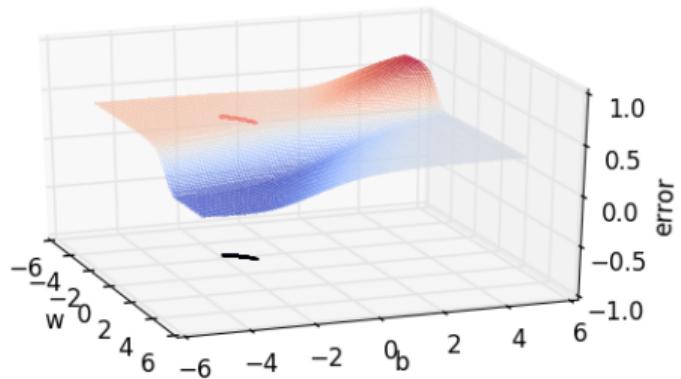
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

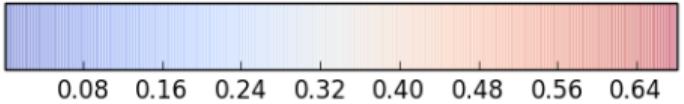
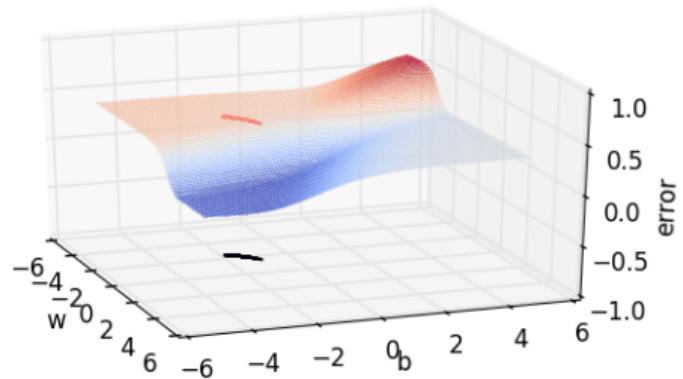
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

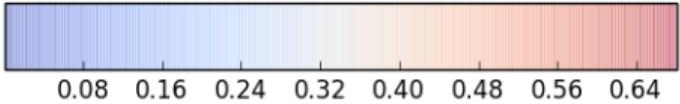
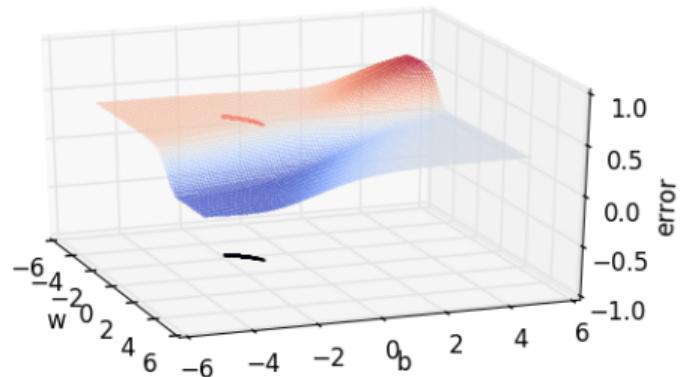
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

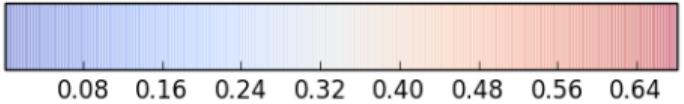
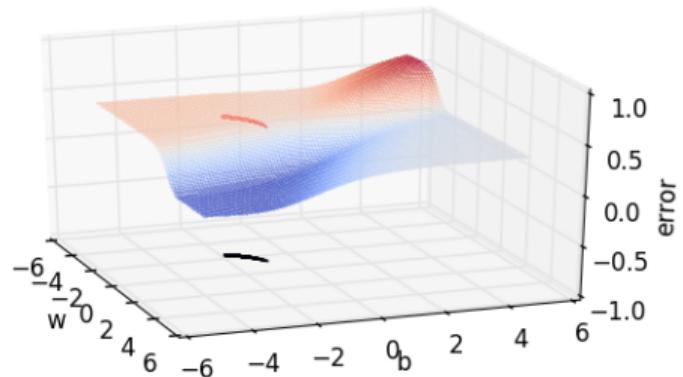
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

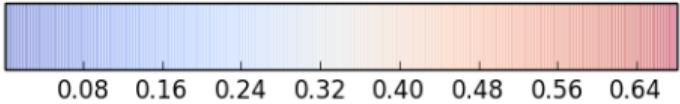
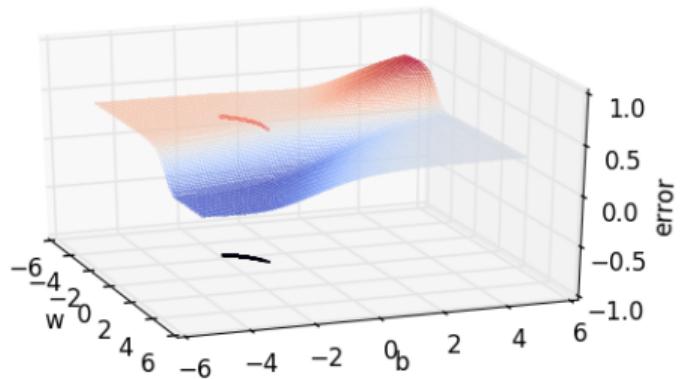
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

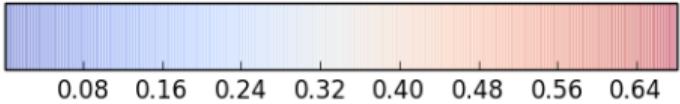
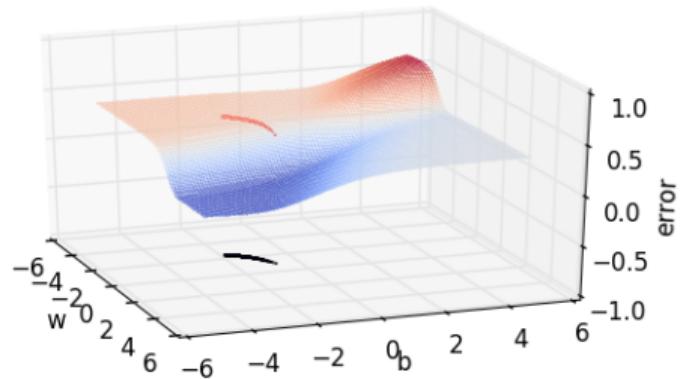
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

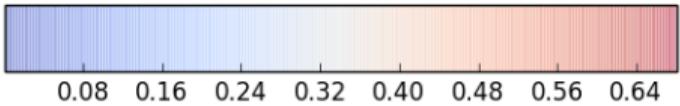
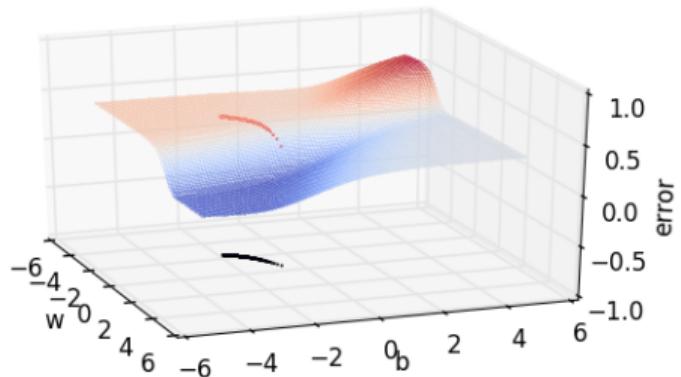
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

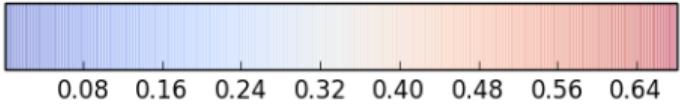
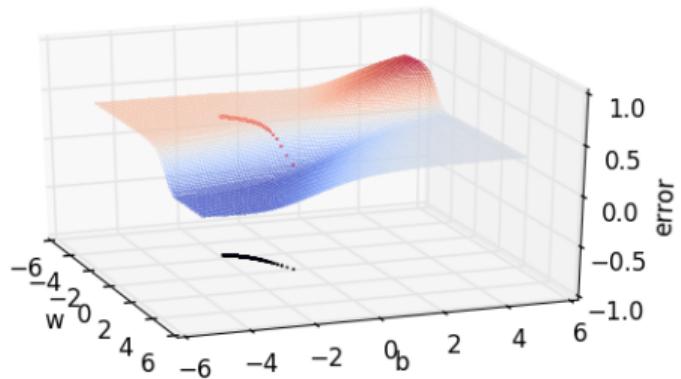
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

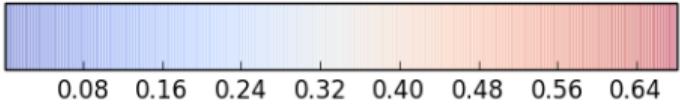
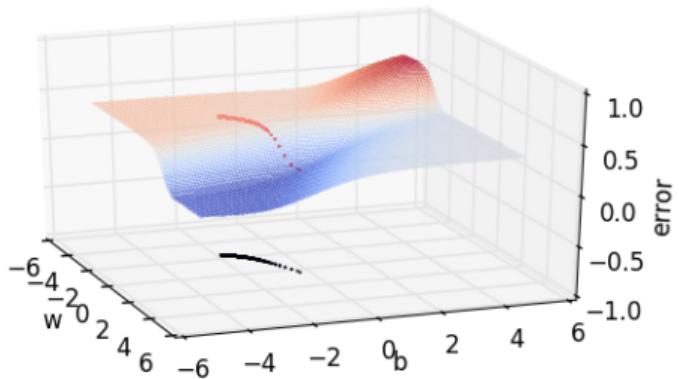
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

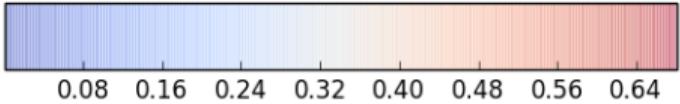
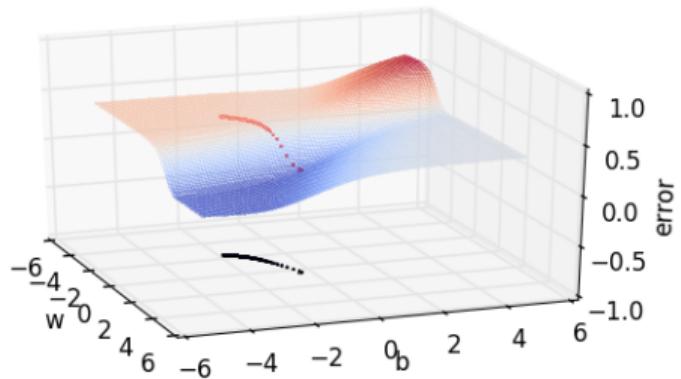
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

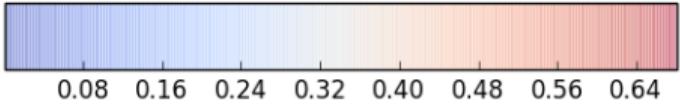
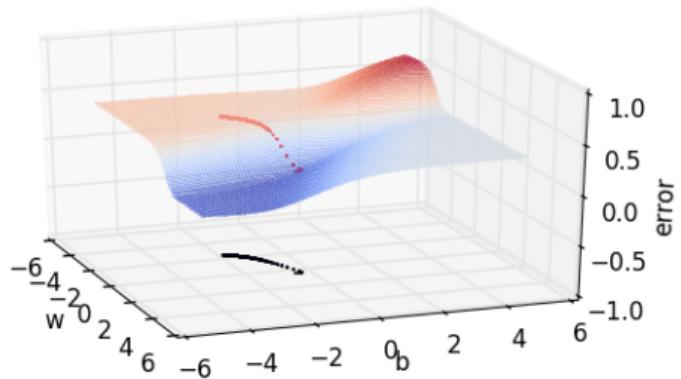
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

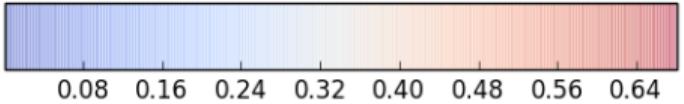
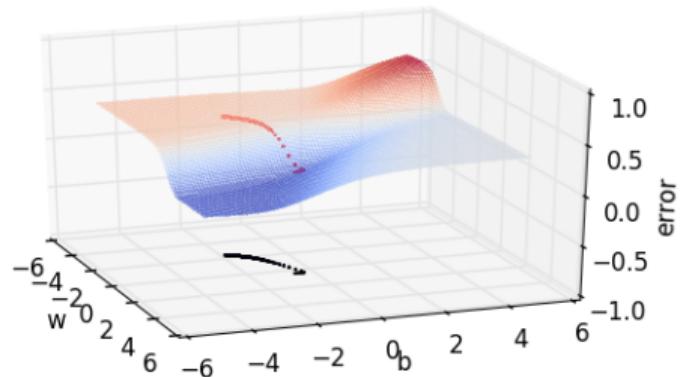
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

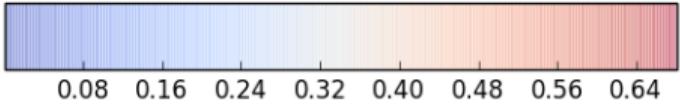
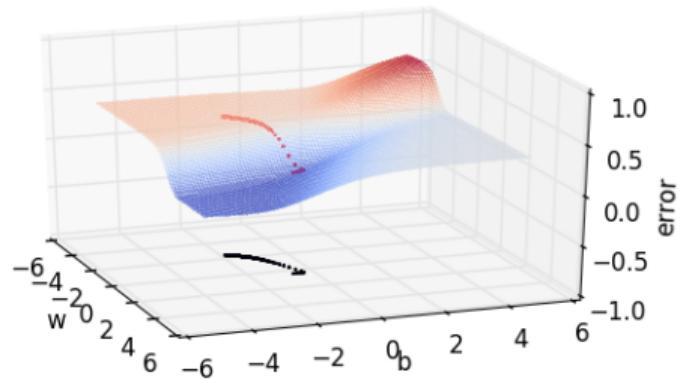
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

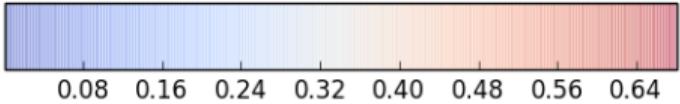
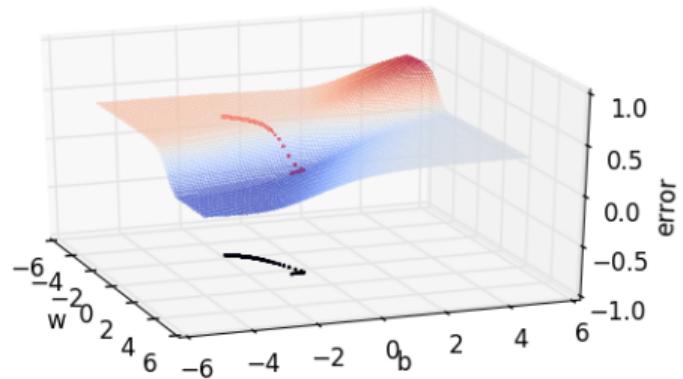
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

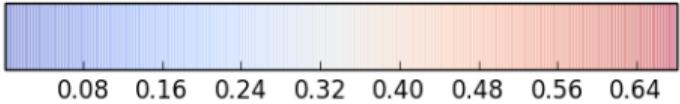
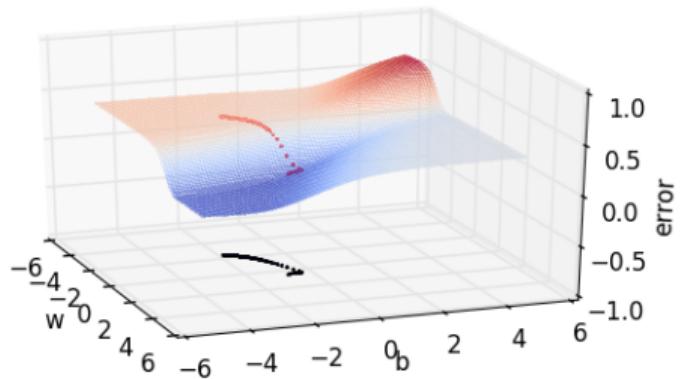
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

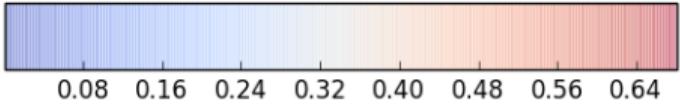
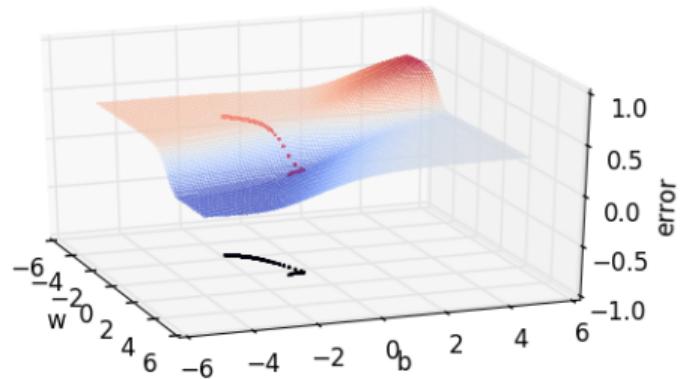
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

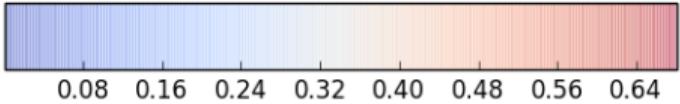
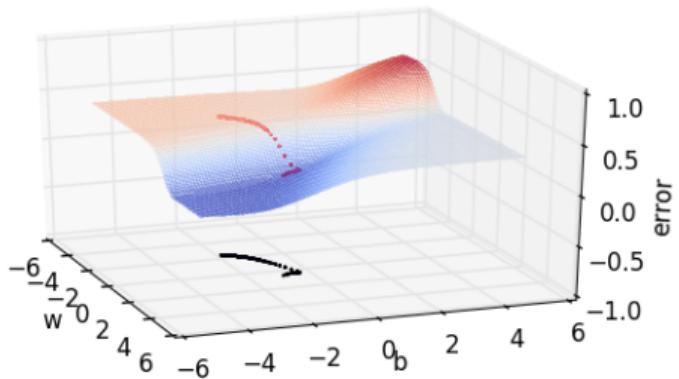
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

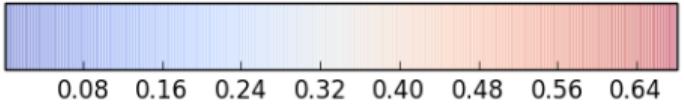
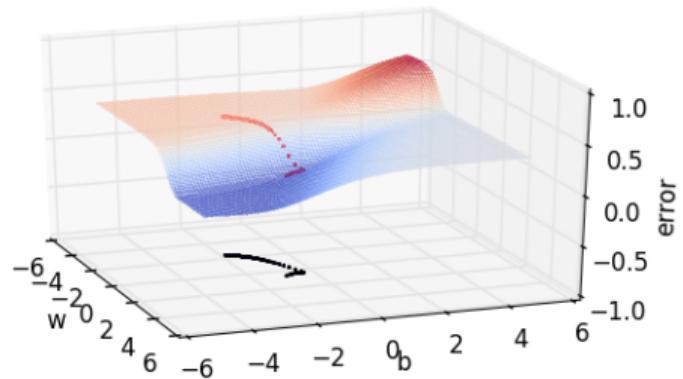
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

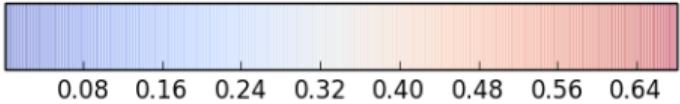
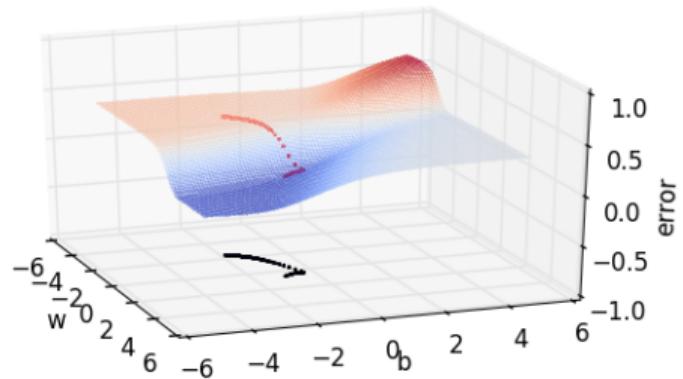
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

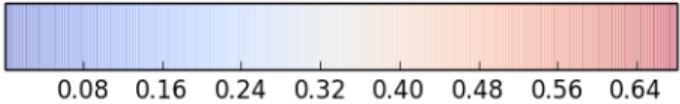
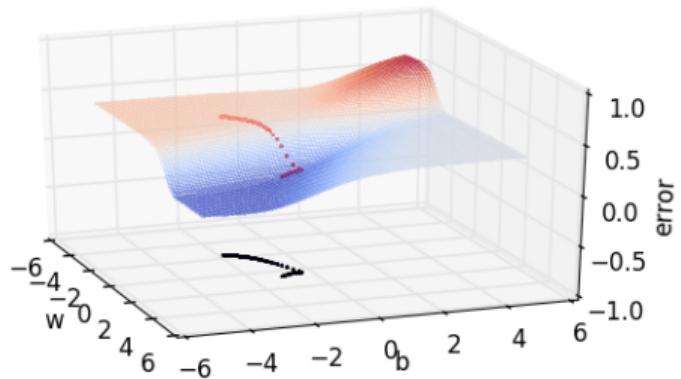
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

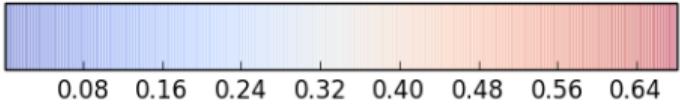
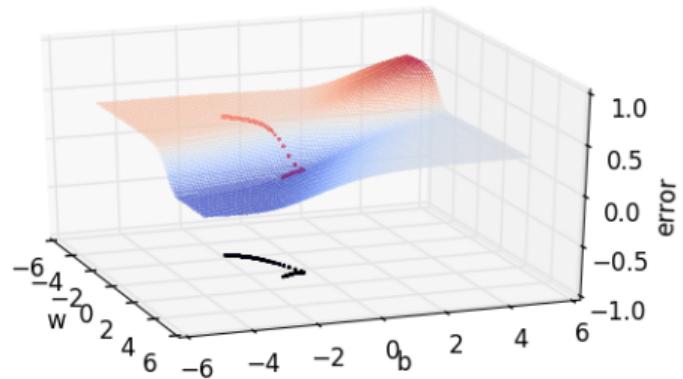
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

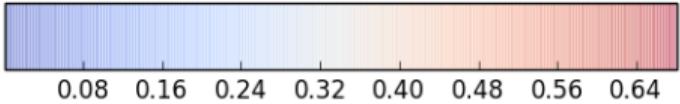
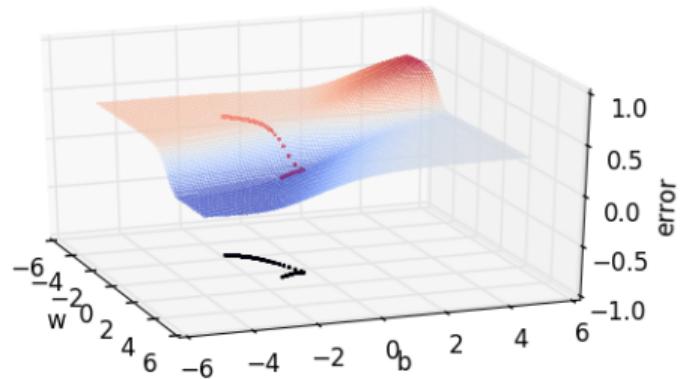
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

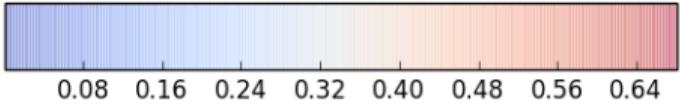
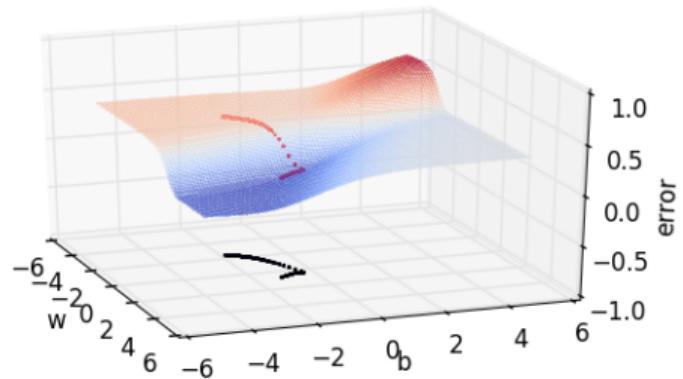
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

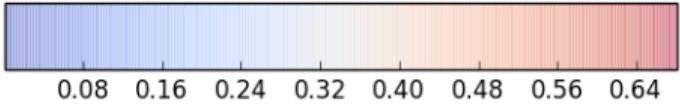
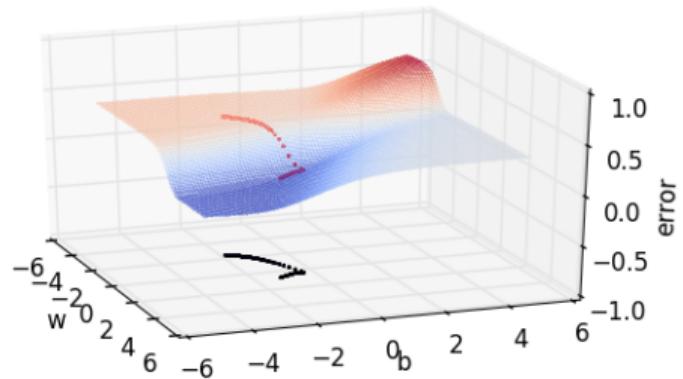
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

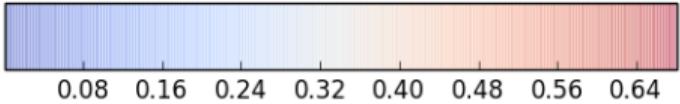
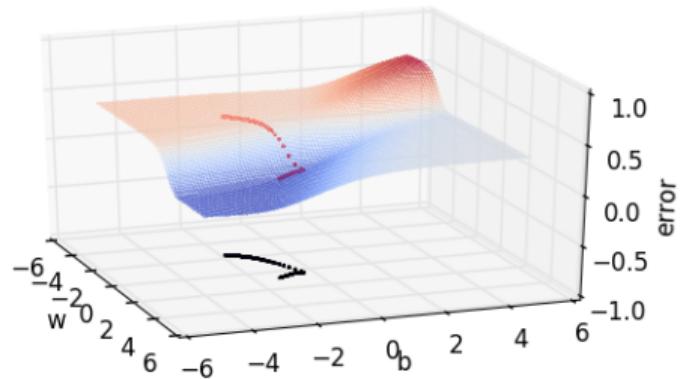
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

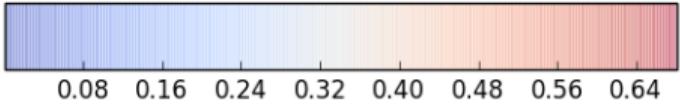
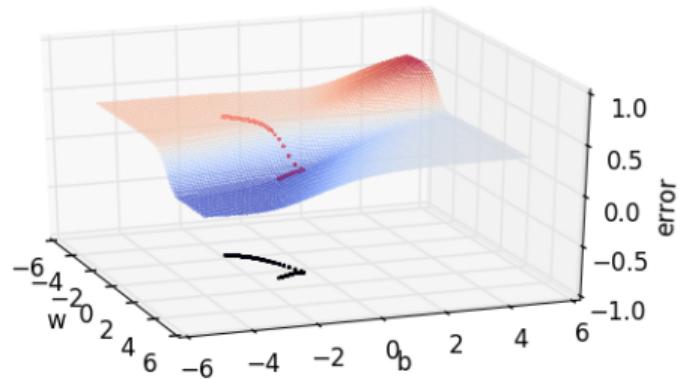
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

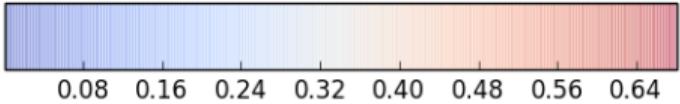
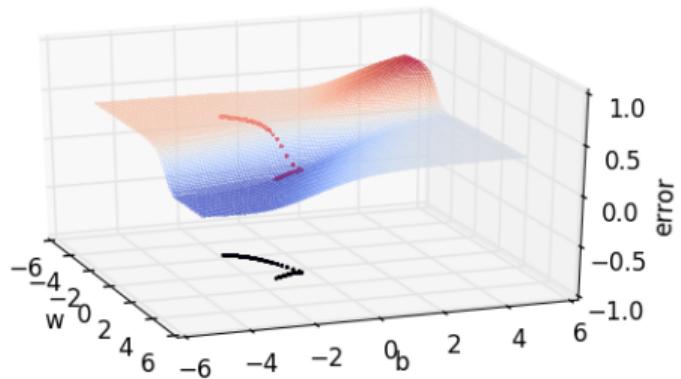
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

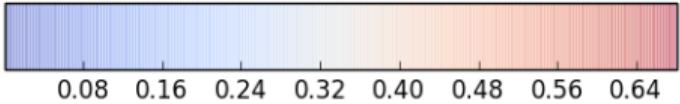
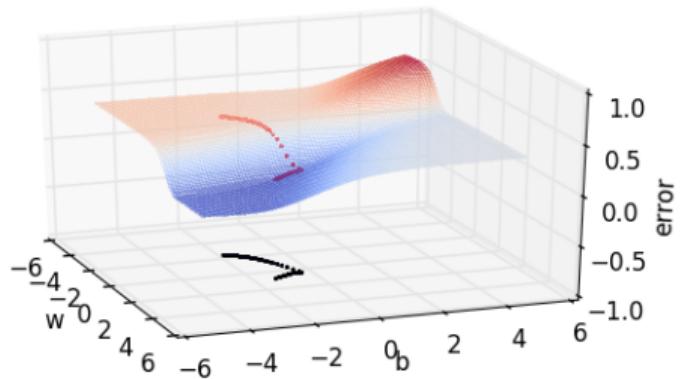
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

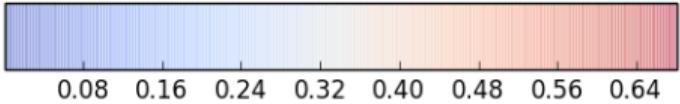
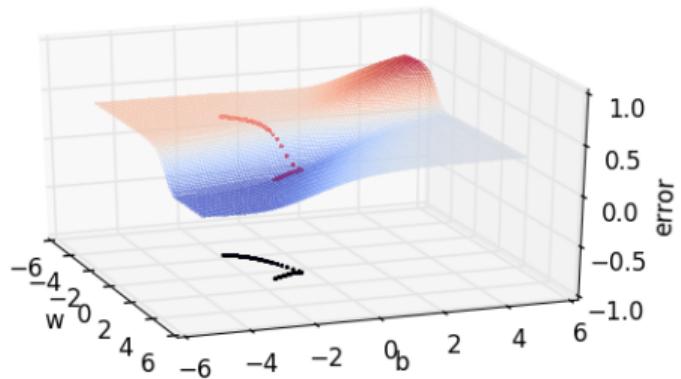
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

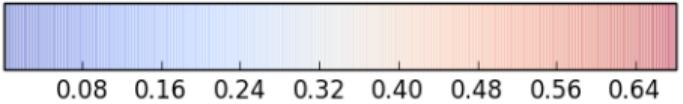
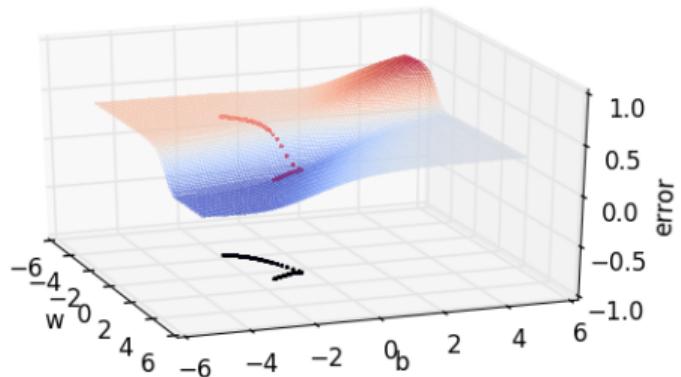
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

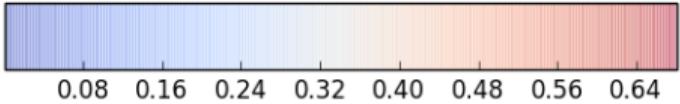
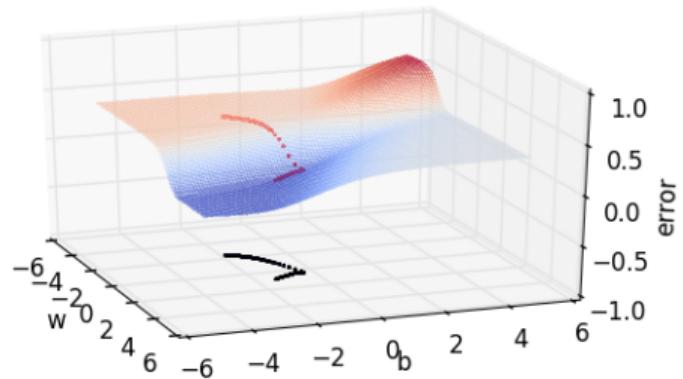
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

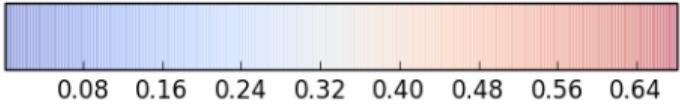
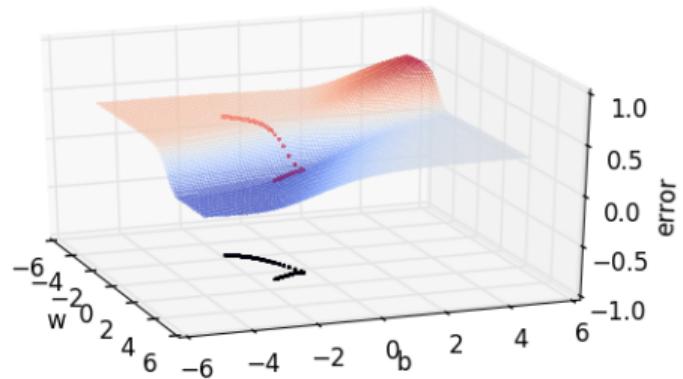
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

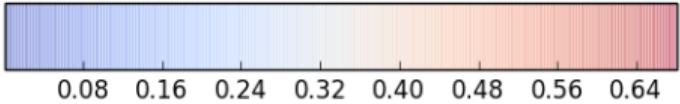
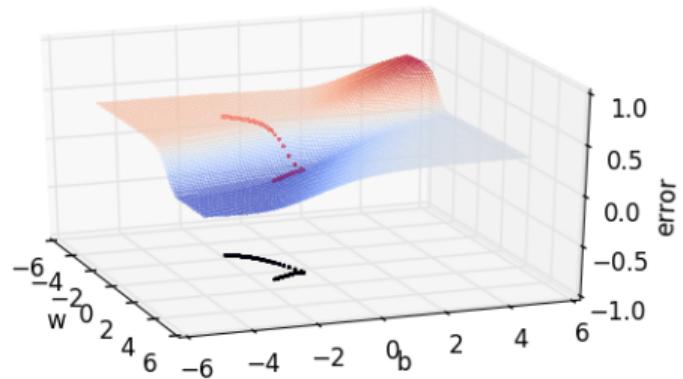
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

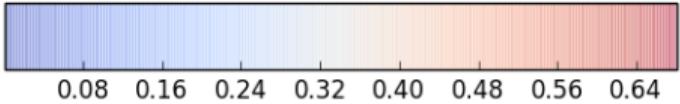
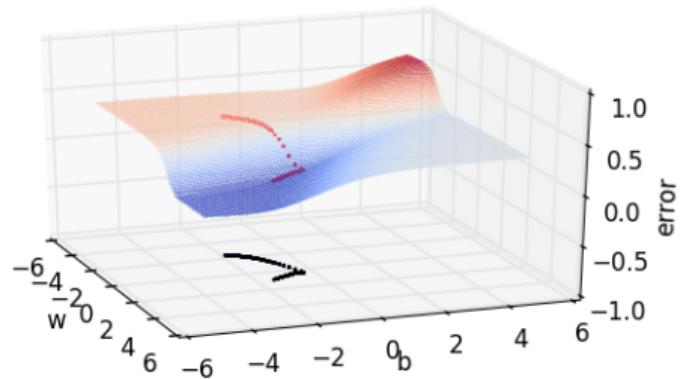
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

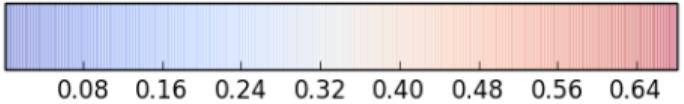
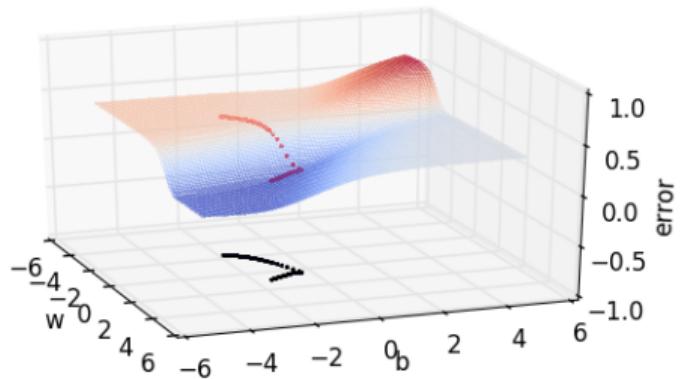
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

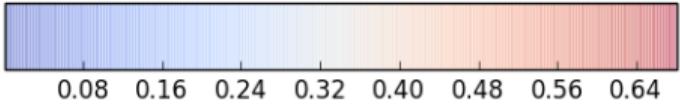
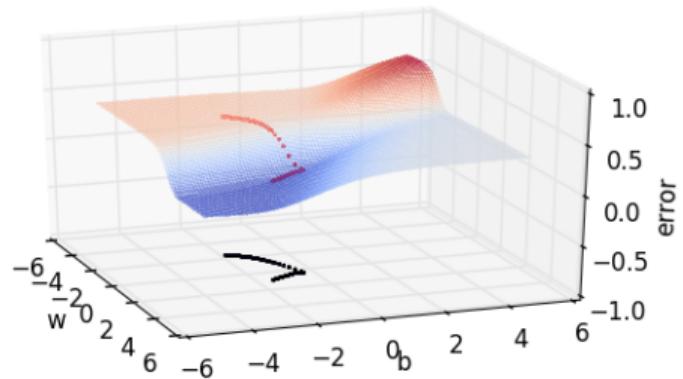
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

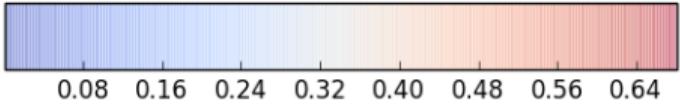
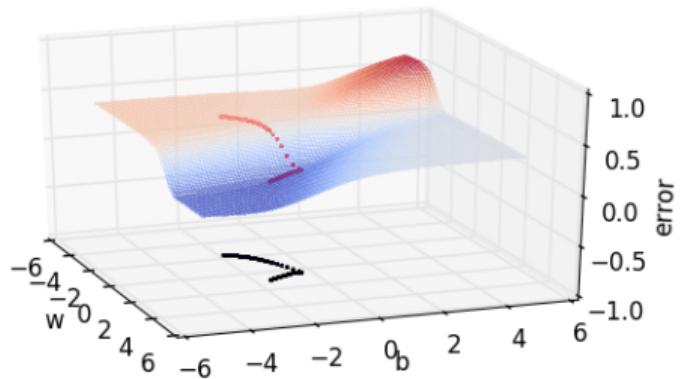
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

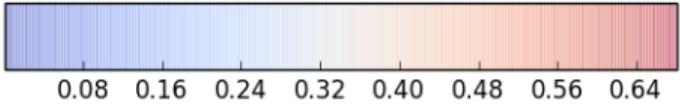
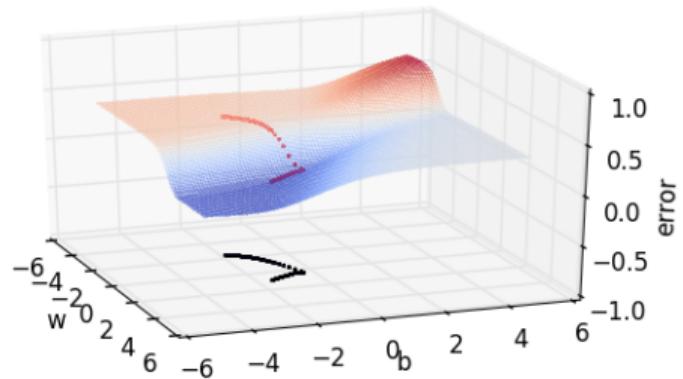
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

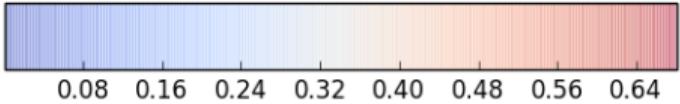
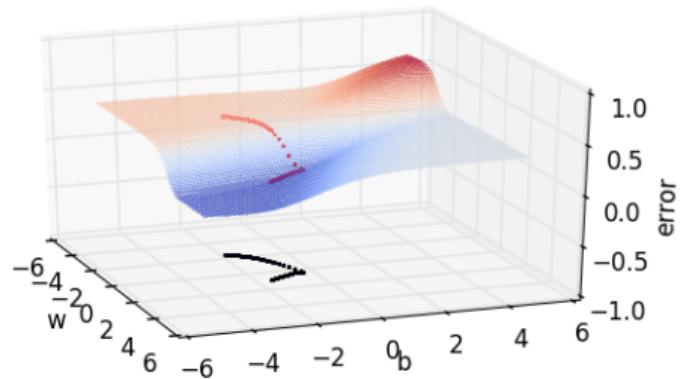
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

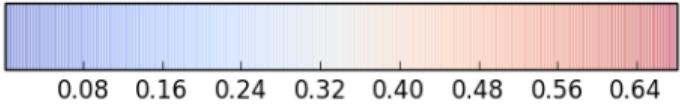
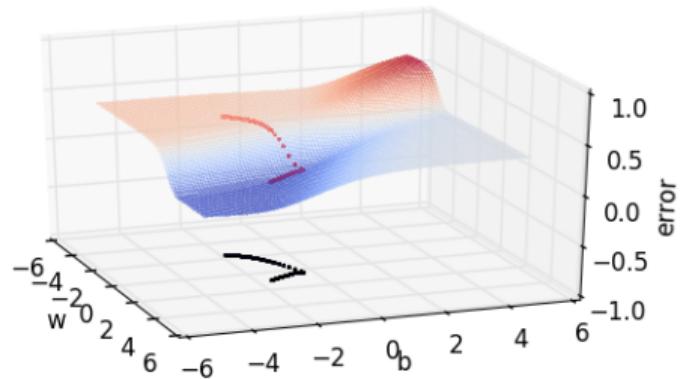
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

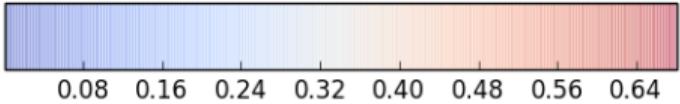
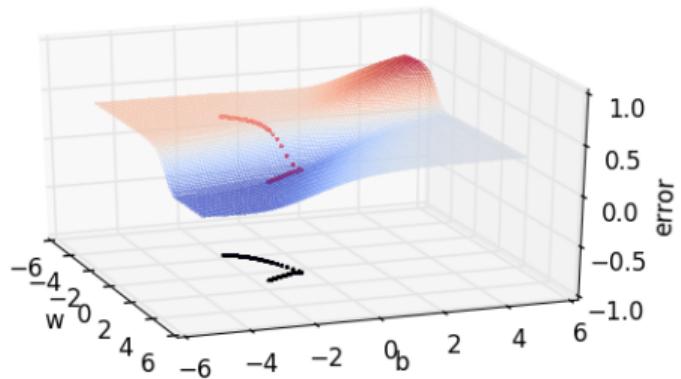
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

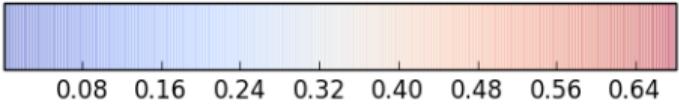
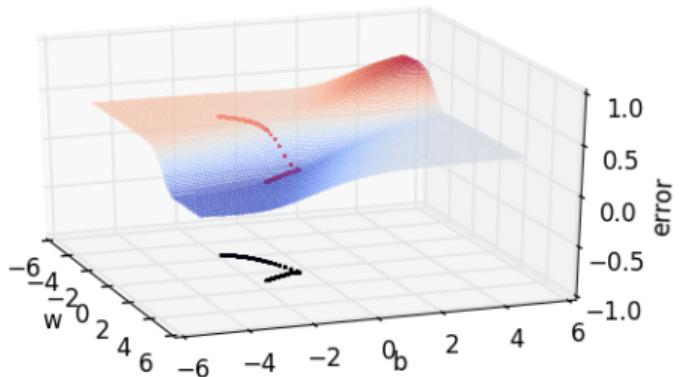
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

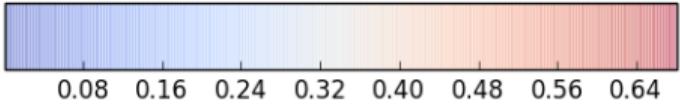
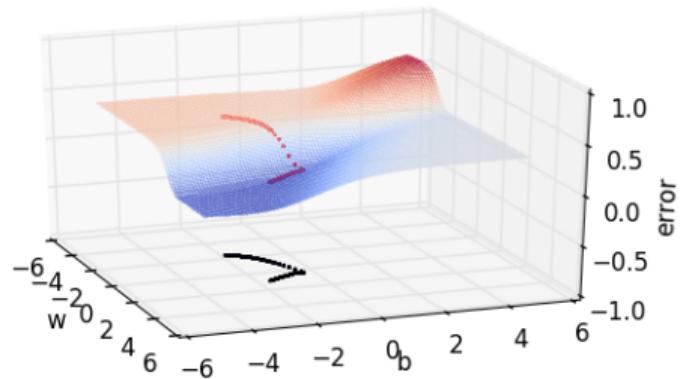
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

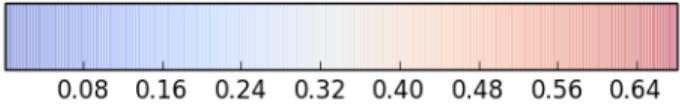
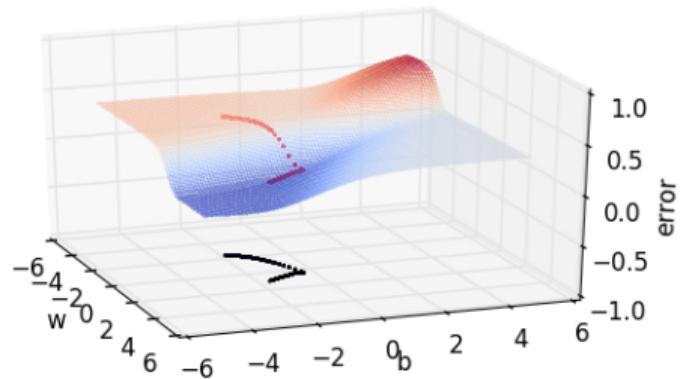
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

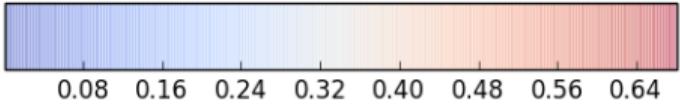
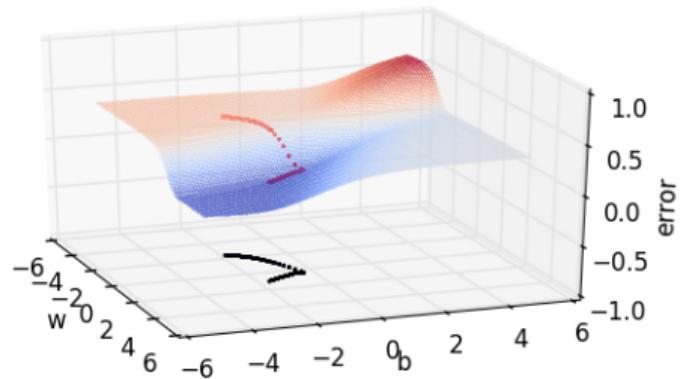
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

[X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

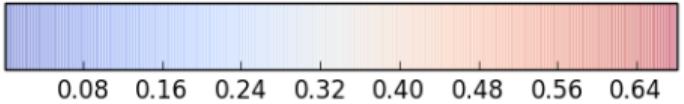
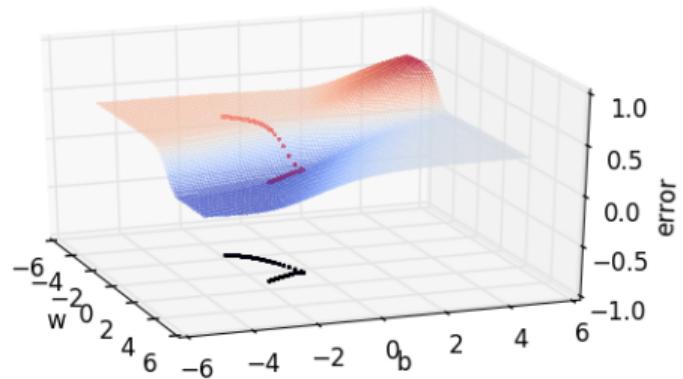
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

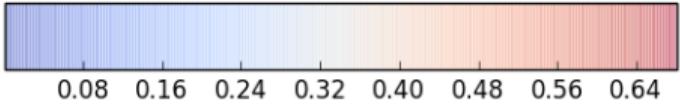
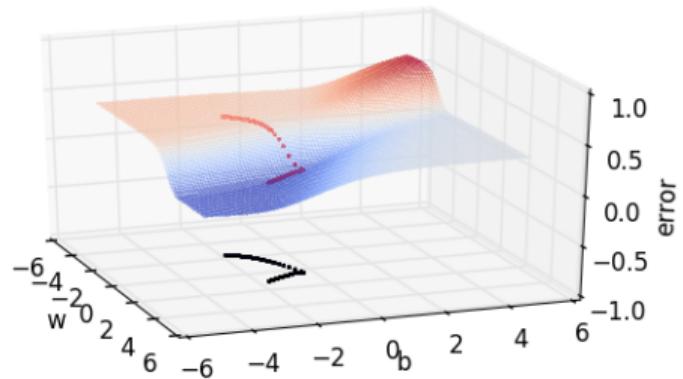
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

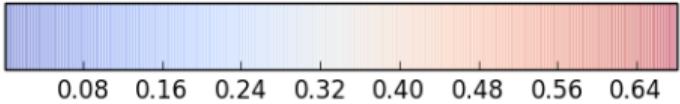
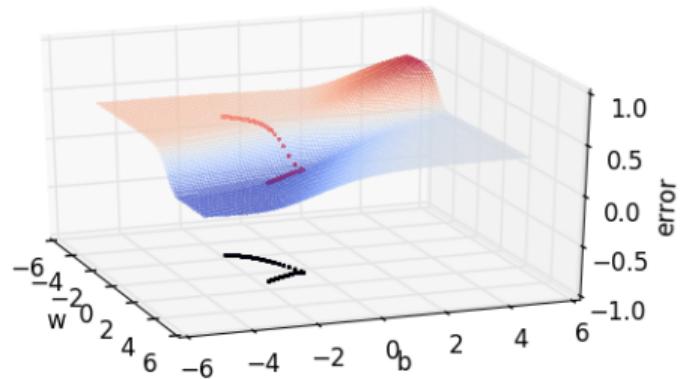
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

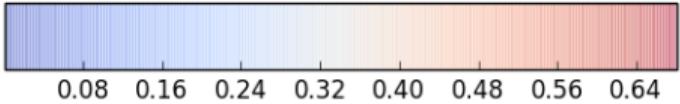
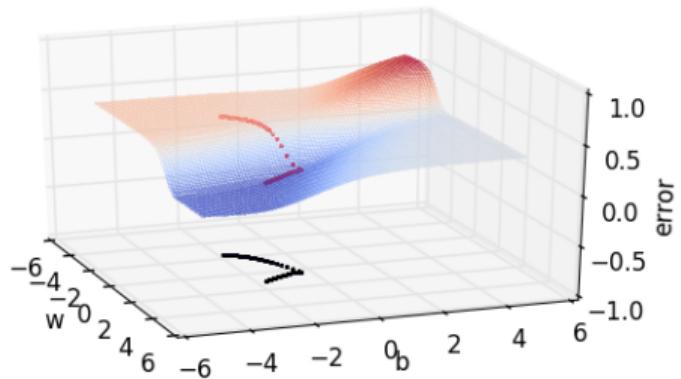
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

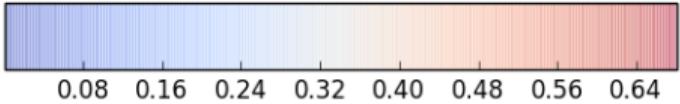
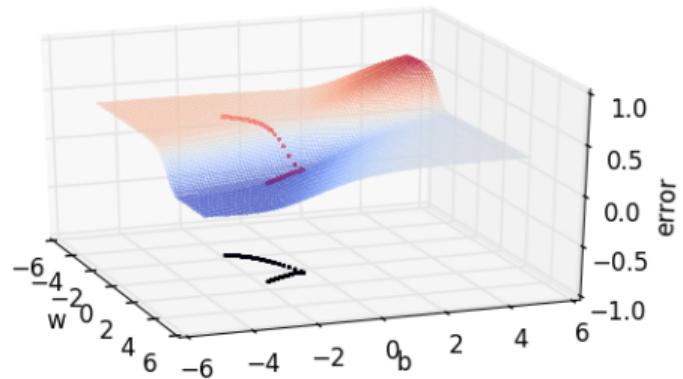
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

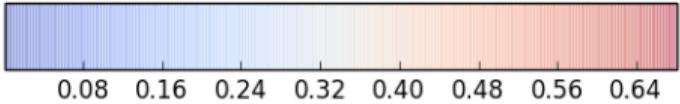
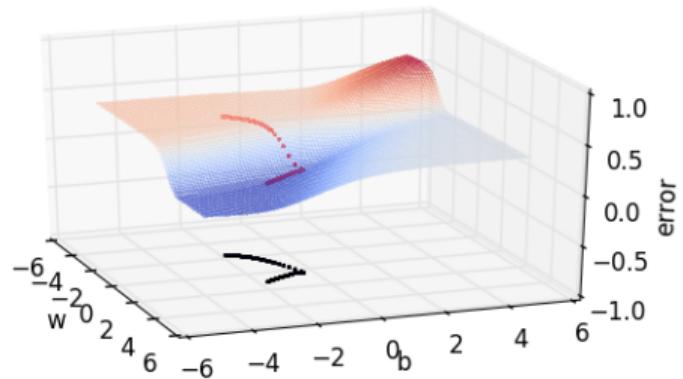
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

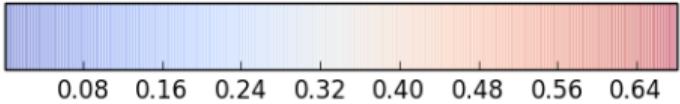
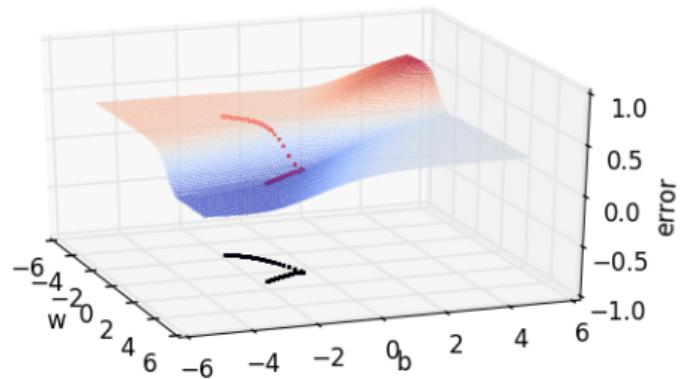
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

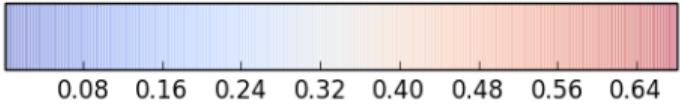
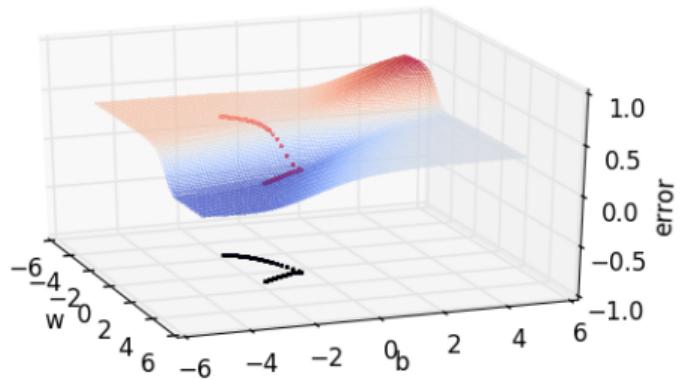
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

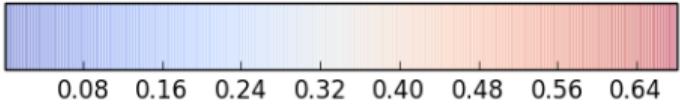
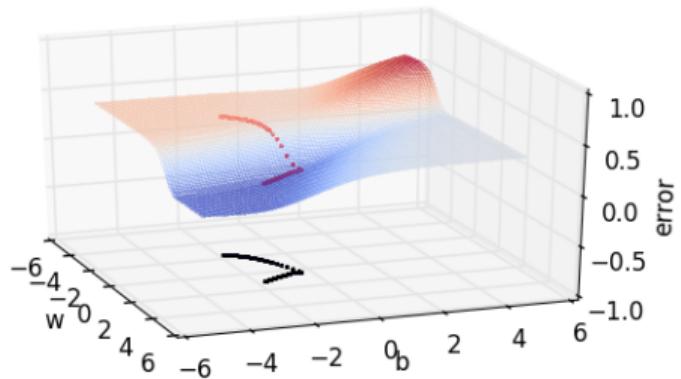
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

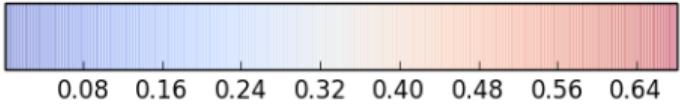
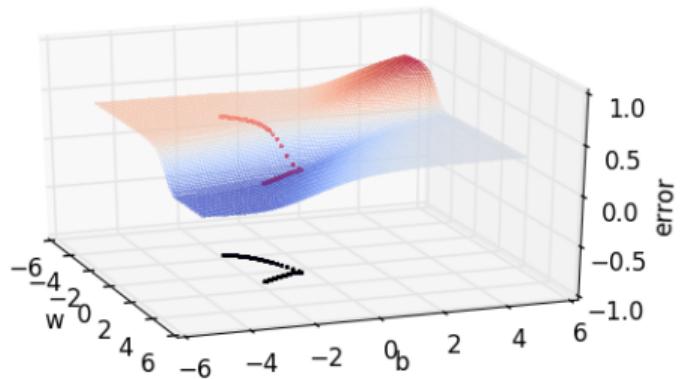
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

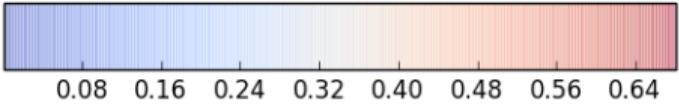
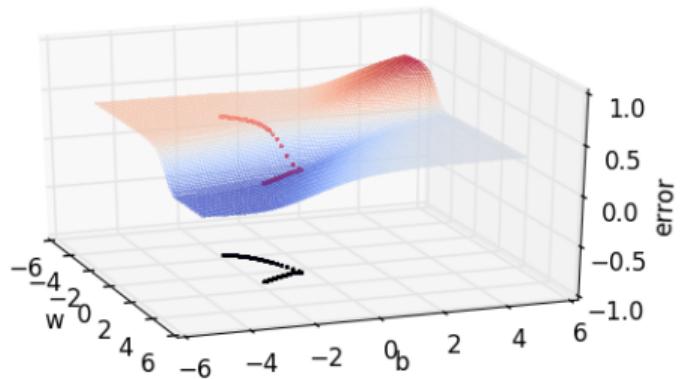
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

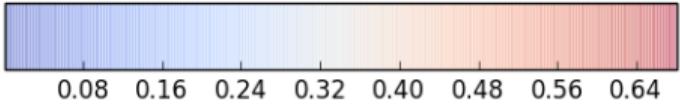
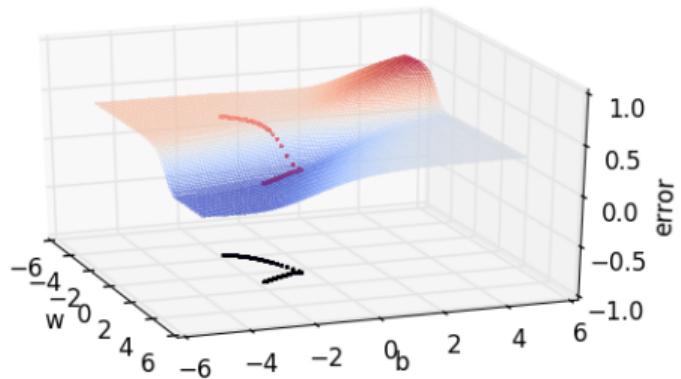
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

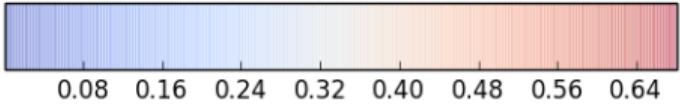
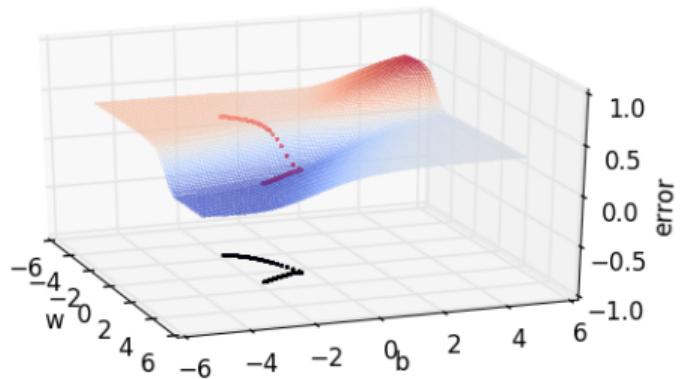
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

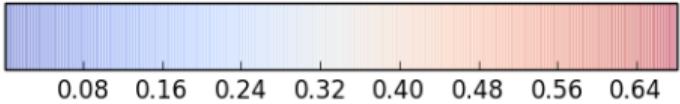
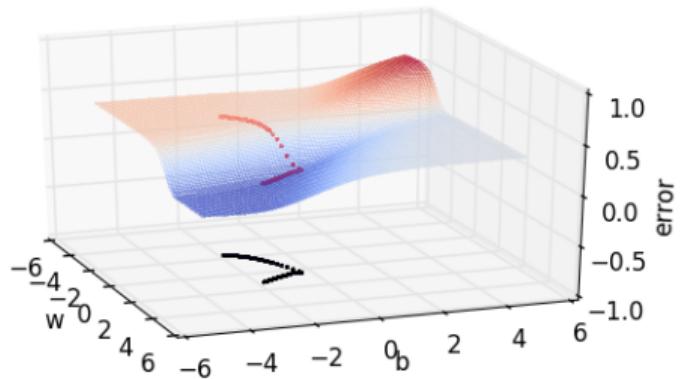
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

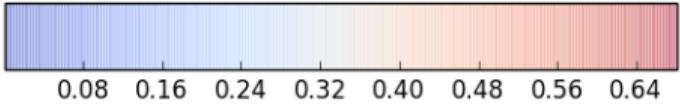
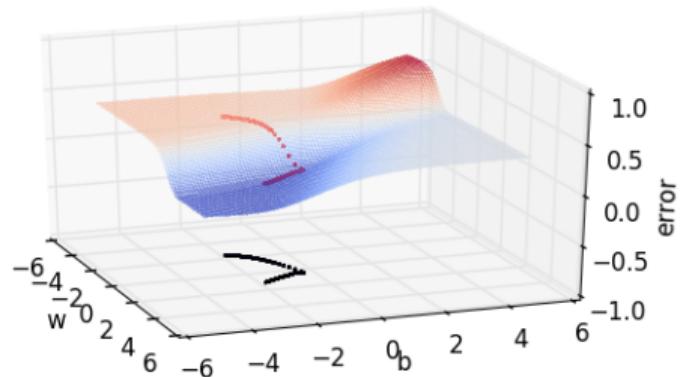
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

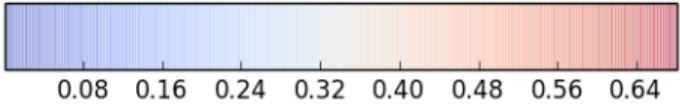
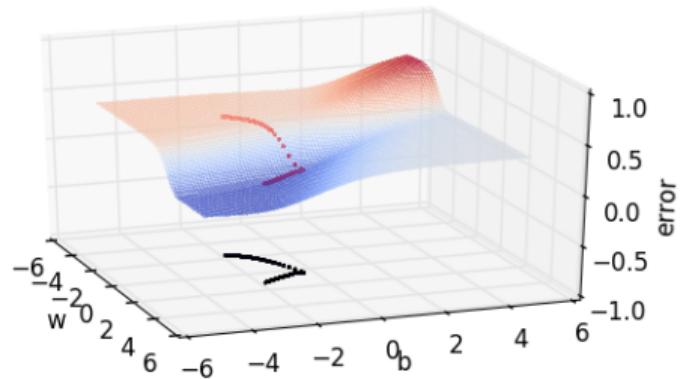
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

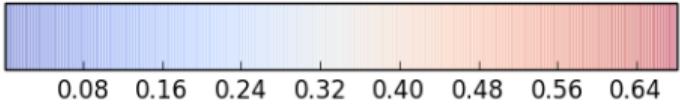
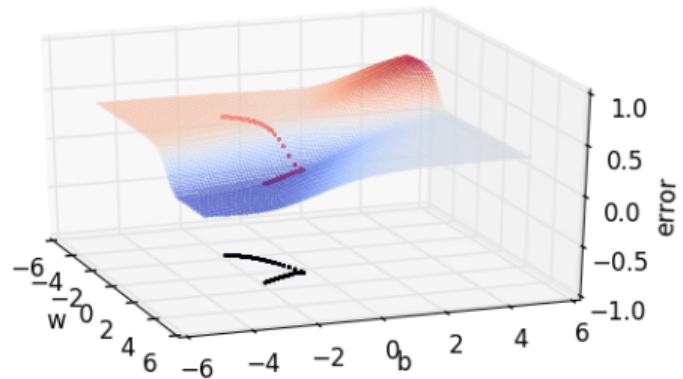
def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db

```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

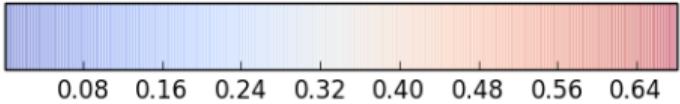
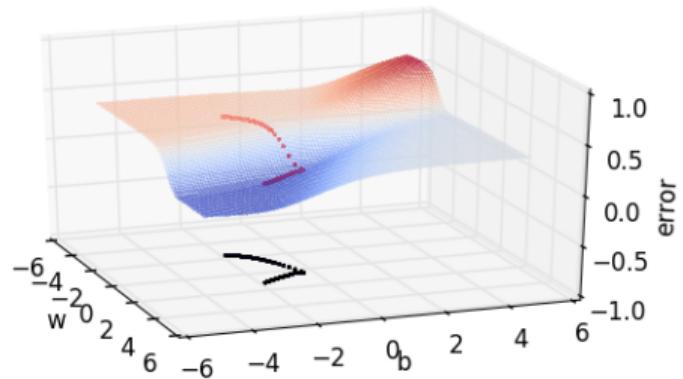
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface



```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

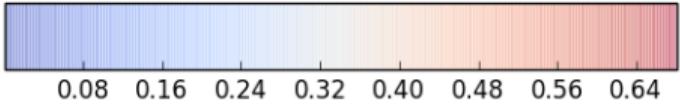
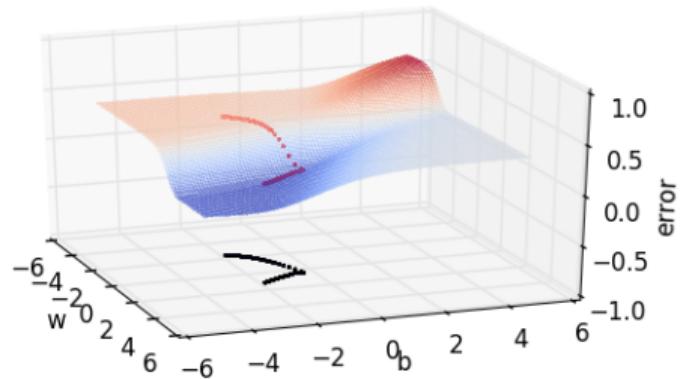
def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Gradient descent on the error surface





多层 Sigmoid 神经元网络的表示能力





多层感知机网络的表示能力

多层 Sigmoid 神经元网络的表示能力



多层感知机网络的表示能力

有一个隐含层的多层感知机网络能 精确地 (no errors) 表示 任意 boolean 函数

多层 Sigmoid 神经元网络的表示能力

多层感知机网络的表示能力

有一个隐含层的多层感知机网络能 精确地 (no errors) 表示 任意 boolean 函数

多层 Sigmoid 神经元网络的表示能力

有一个隐含层的多层 Sigmoid 神经元网络 能以期望的精度 近似 任意 连续 函数

多层感知机网络的表示能力

有一个隐含层的多层感知机网络能 精确地 (no errors) 表示 任意 boolean 函数

多层 Sigmoid 神经元网络的表示能力

有一个隐含层的多层 Sigmoid 神经元网络 能以期望的精度 近似 任意 连续 函数

换句话说，对于任意的函数 $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ ，总能找到一个具有一个隐含层的多层 Sigmoid 神经元网络，它的输出 $g(x)$ 满足 $|g(x) - f(x)| < \epsilon$!!

多层感知机网络的表示能力

有一个隐含层的多层感知机网络能 精确地 (no errors) 表示 任意 boolean 函数

多层 Sigmoid 神经元网络的表示能力

有一个隐含层的多层 Sigmoid 神经元网络 能以期望的精度 近似 任意 连续 函数

换句话说，对于任意的函数 $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ ，总能找到一个具有一个隐含层的多层 Sigmoid 神经元网络，它的输出 $g(x)$ 满足 $|g(x) - f(x)| < \epsilon$!!

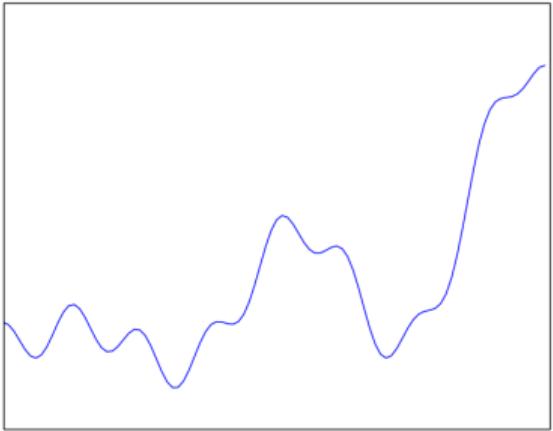
证明：参考 [Cybenko, 1989], [Hornik, 1991]

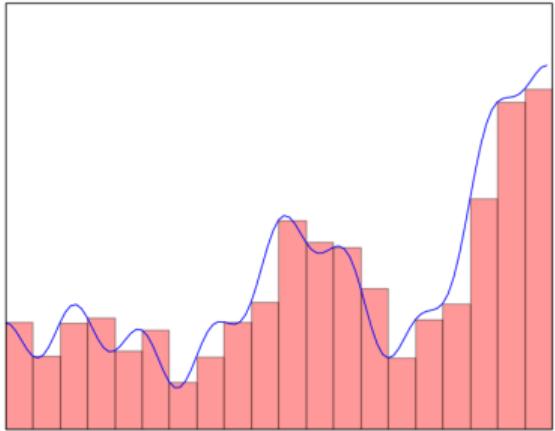
- 具体证明过程，可以阅读这个链接上的文章*
- 下面的讨论基于上述的文章

*<http://neuralnetworksanddeeplearning.com/chap4.html>

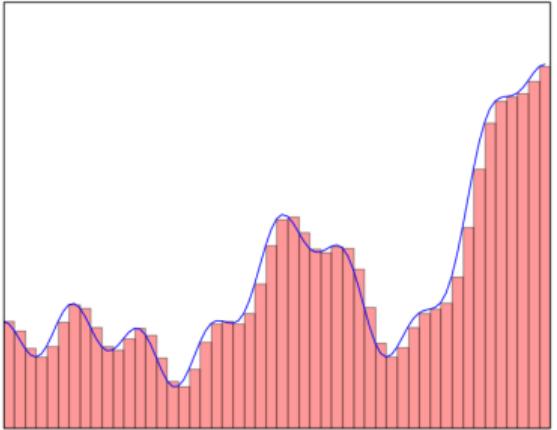


- 一个神经元网络能否被用来表示一个任意的函数（如图中所示的函数）？

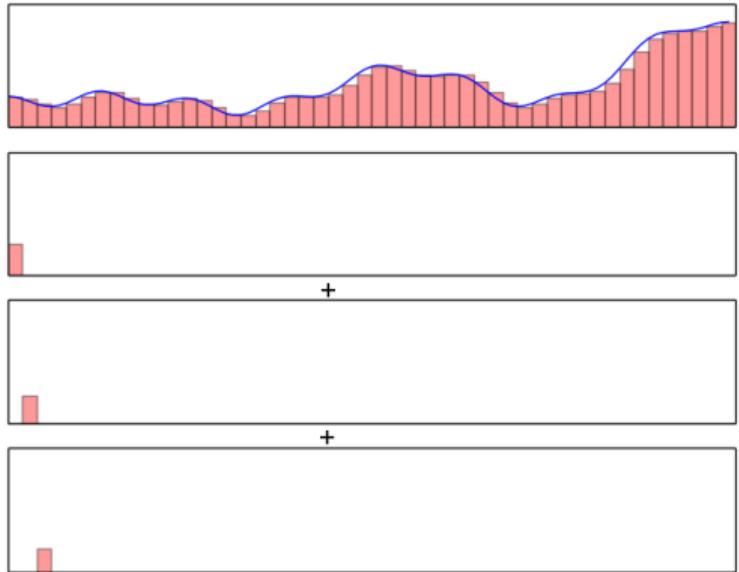




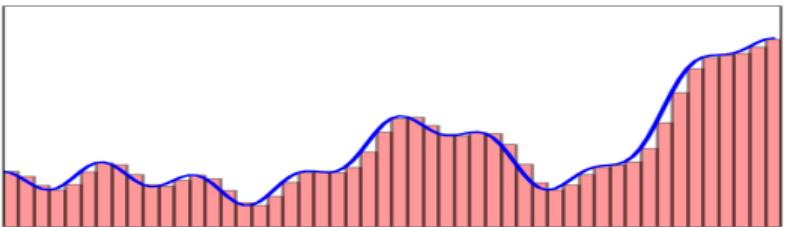
- 一个神经元网络能否被用来表示一个任意的函数（如图中所示的函数）？
- 这个任意的函数可以由几个“tower”函数近似



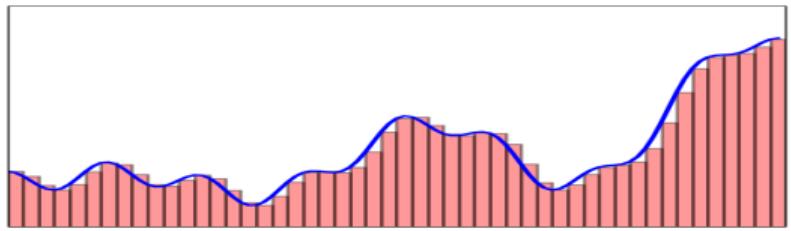
- 一个神经元网络能否被用来表示一个任意的函数（如图中所示的函数）？
- 这个任意的函数可以由几个“tower”函数近似
- 这样的“tower”函数越多，近似的效果越好



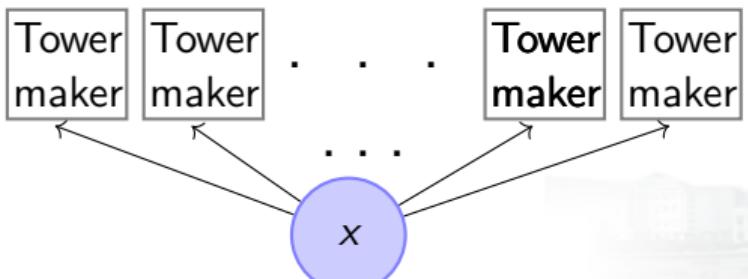
- 一个神经元网络能否被用来表示一个任意的函数（如图中所示的函数）？
- 这个任意的函数可以由几个“tower”函数近似
- 这样的“tower”函数越多，近似的效果越好
- 任意的函数可以用多个“tower”函数的求和来近似

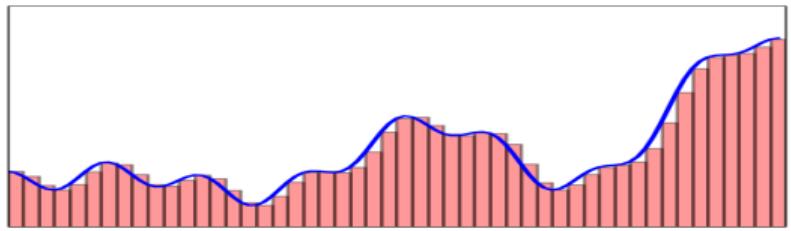


- 这些“tower”函数是非常相似的，除了它们的高度和在 x 轴的位置不一样

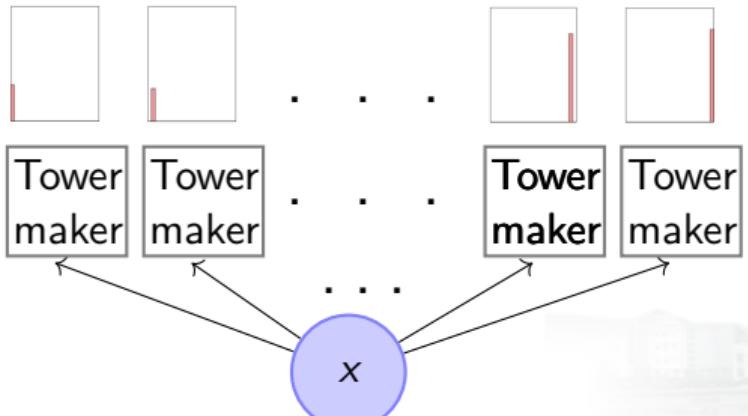


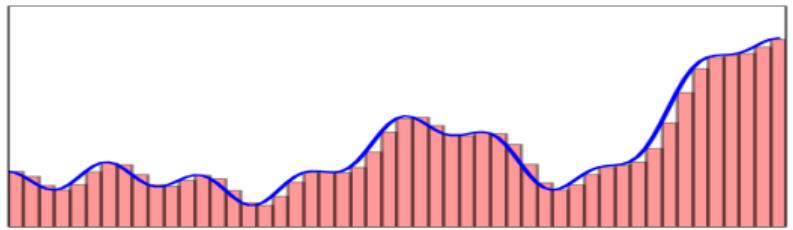
- 这些 “tower” 函数是非常相似的，除了它们的高度和在 x 轴的位置不一样
- 假设有一个黑盒，它的输入是 x ，构造这样的 tower 函数



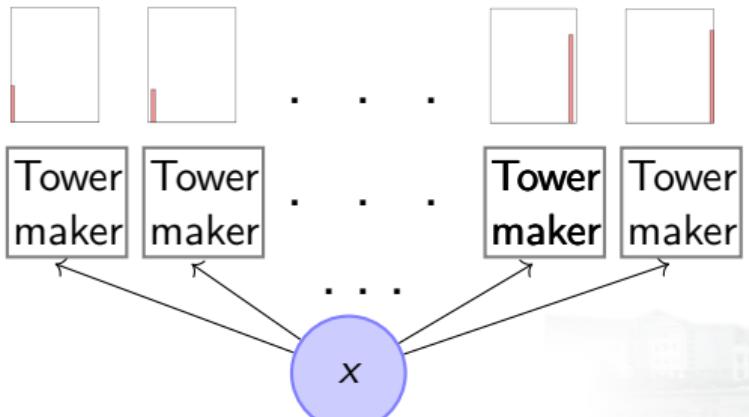


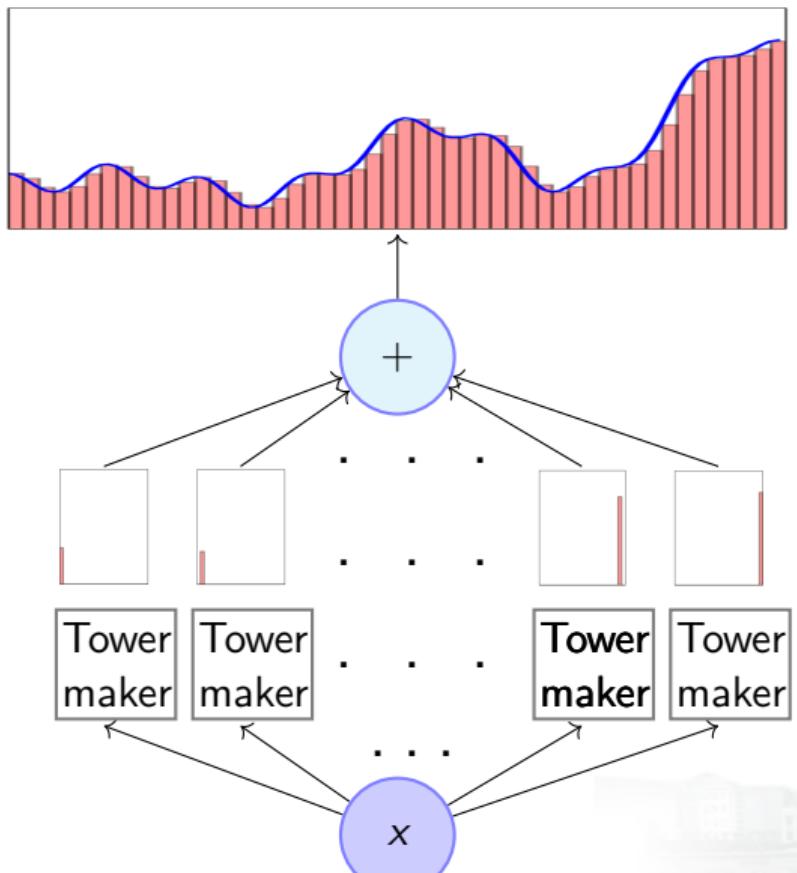
- 这些“tower”函数是非常相似的，除了它们的高度和在 x 轴的位置不一样
- 假设有一个黑盒，它的输入是 x ，构造这样的 tower 函数



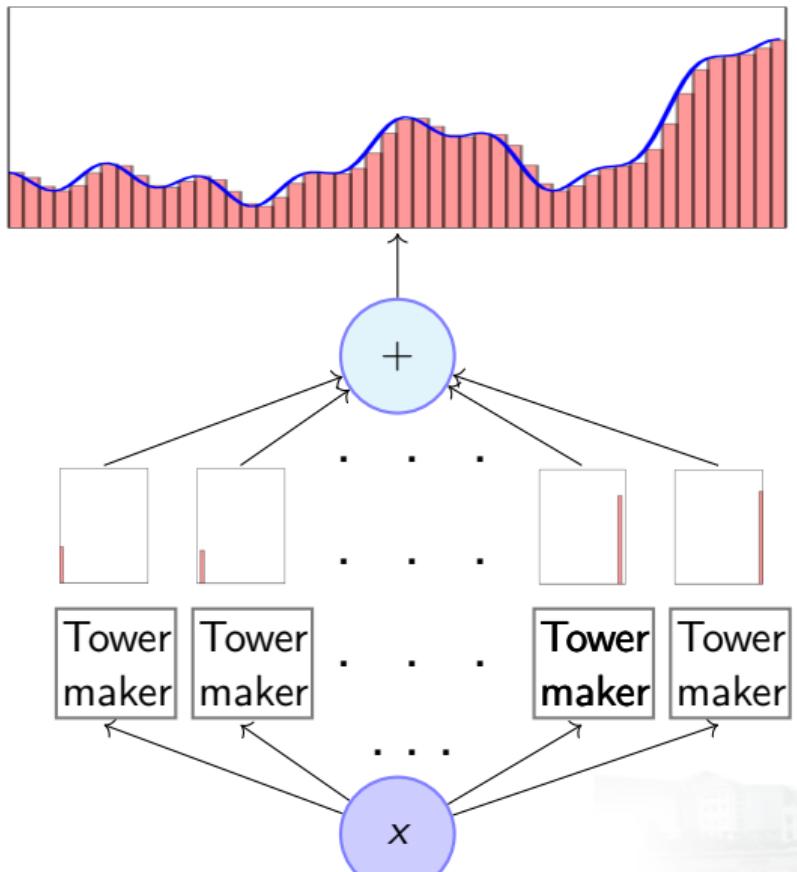


- 这些“tower”函数是非常相似的，除了它们的高度和在 x 轴的位置不一样
- 假设有一个黑盒，它的输入是 x ，构造这样的 tower 函数
- 通过把这些黑盒加起来就能构造一个简单的网络来近似任意函数

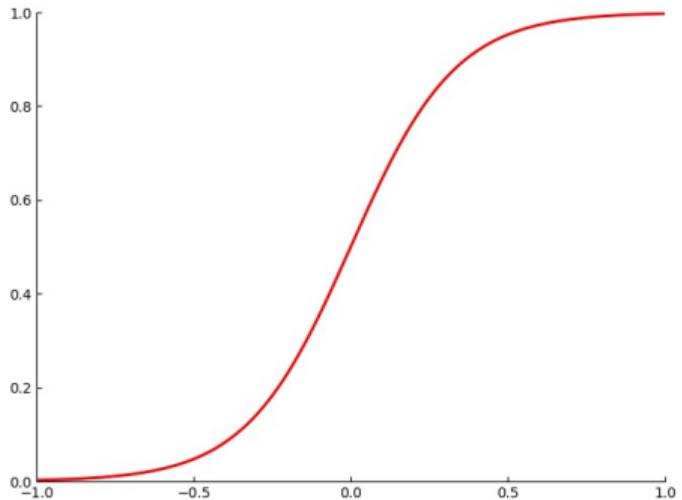




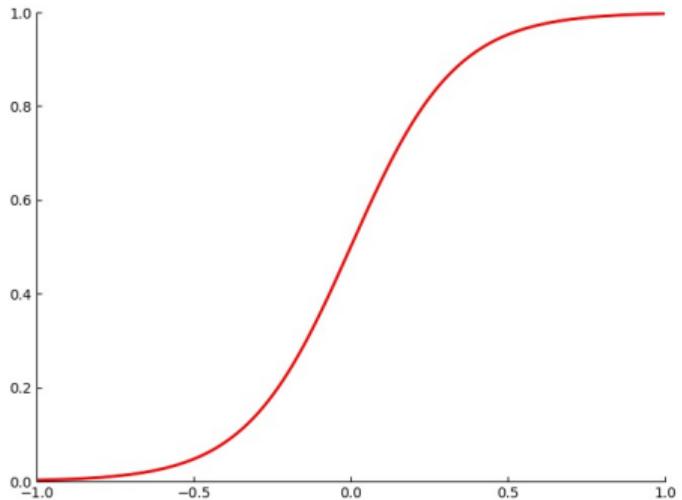
- 这些“tower”函数是非常相似的，除了它们的高度和在 x 轴的位置不一样
- 假设有一个黑盒，它的输入是 x ，构造这样的 tower 函数
- 通过把这些黑盒加起来就能构造一个简单的网络来近似任意函数



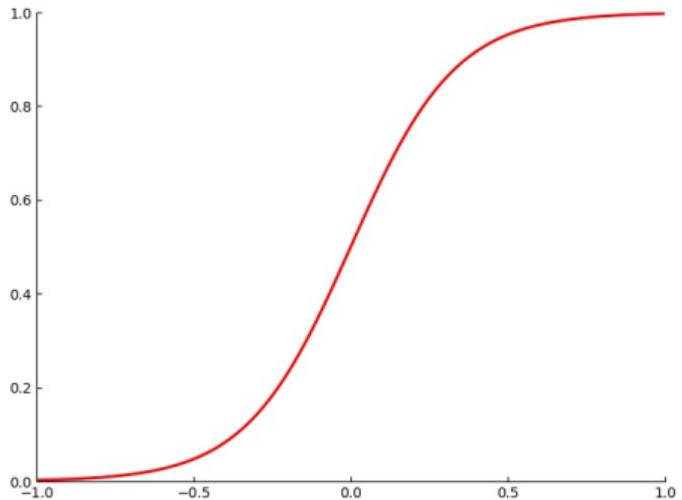
- 这些“tower”函数是非常相似的，除了它们的高度和在 x 轴的位置不一样
- 假设有一个黑盒，它的输入是 x ，构造这样的 tower 函数
- 通过把这些黑盒加起来就能构造一个简单的网络来近似任意函数
- 如何实现这样的黑盒？



- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数

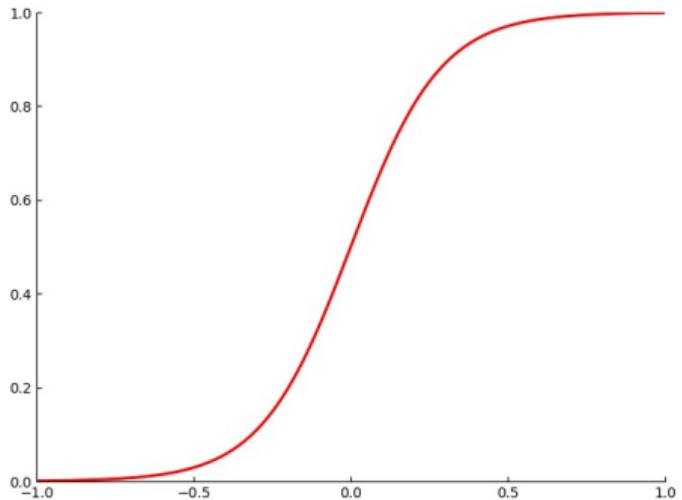


- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的



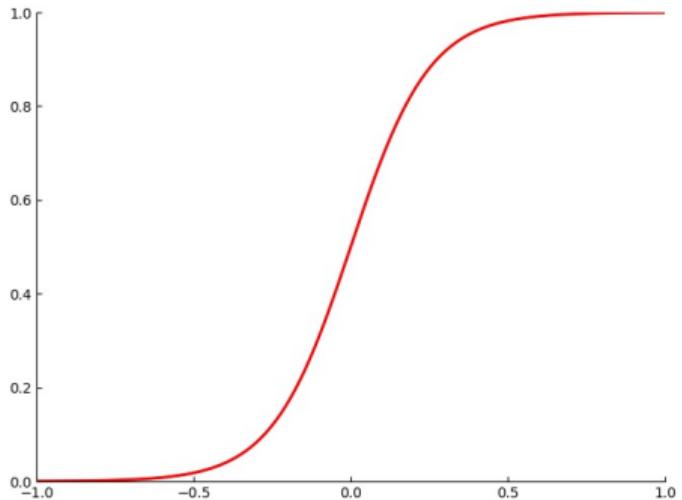
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 0, b = 0$$



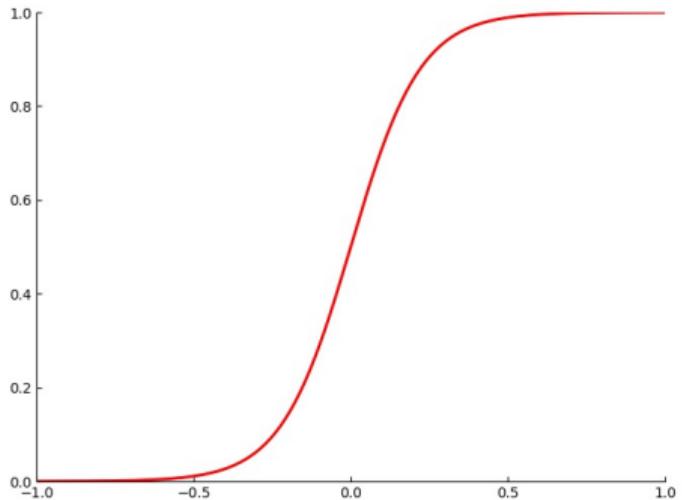
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 1, b = 0$$



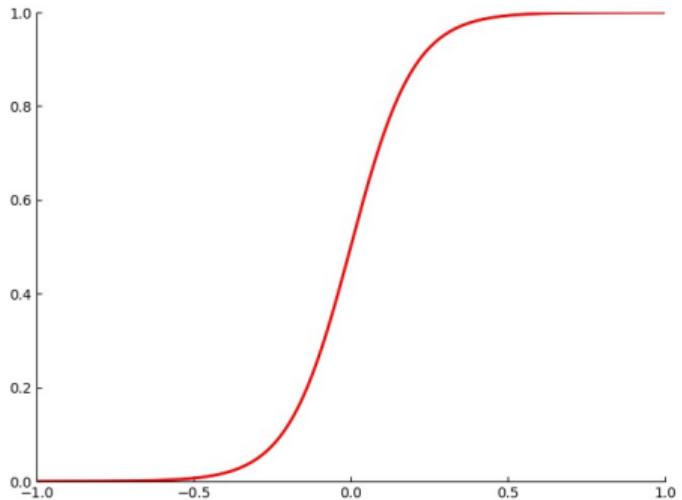
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 2, b = 0$$



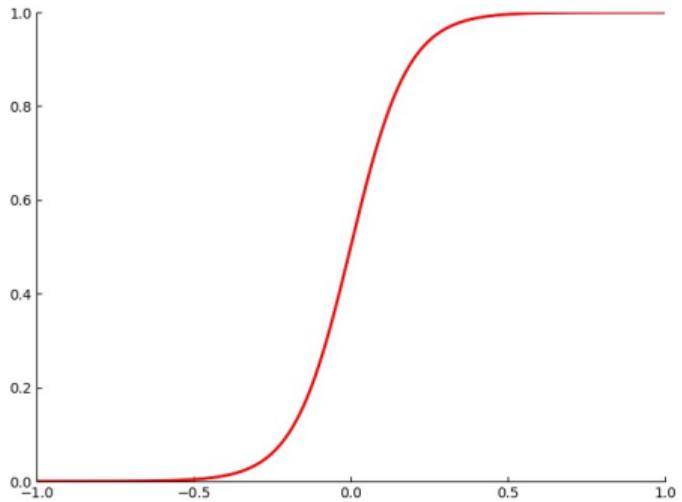
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 3, b = 0$$



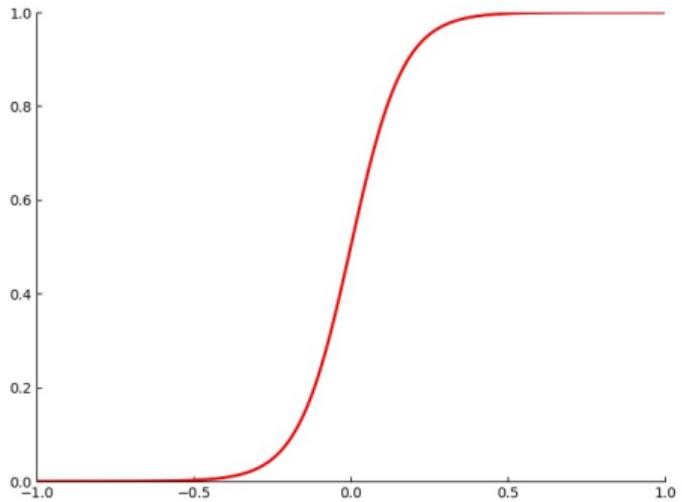
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 4, b = 0$$



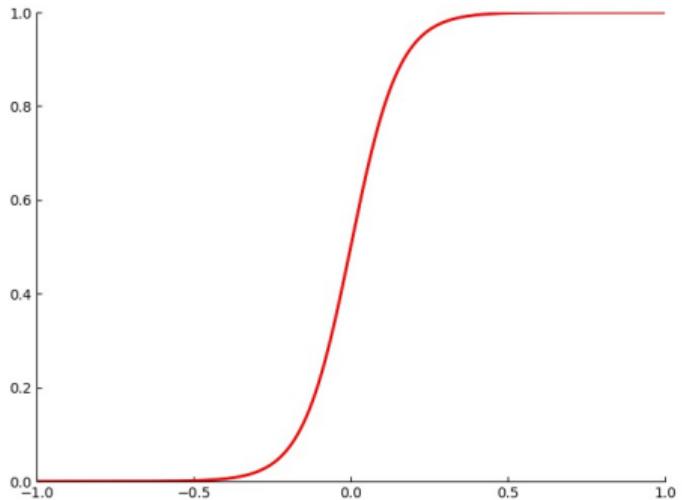
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 5, b = 0$$



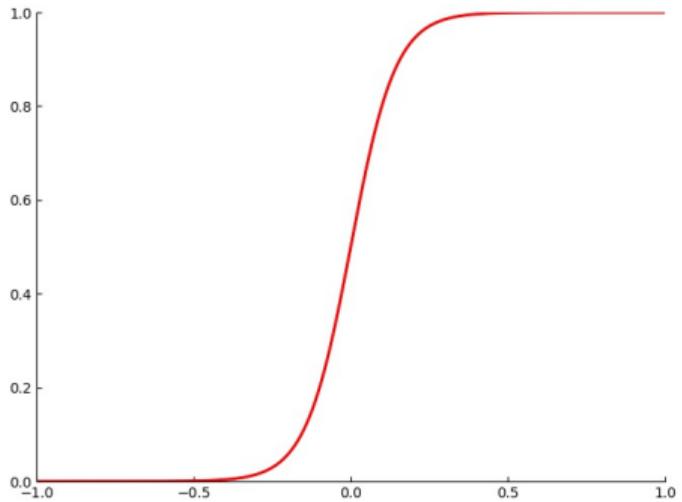
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 6, b = 0$$



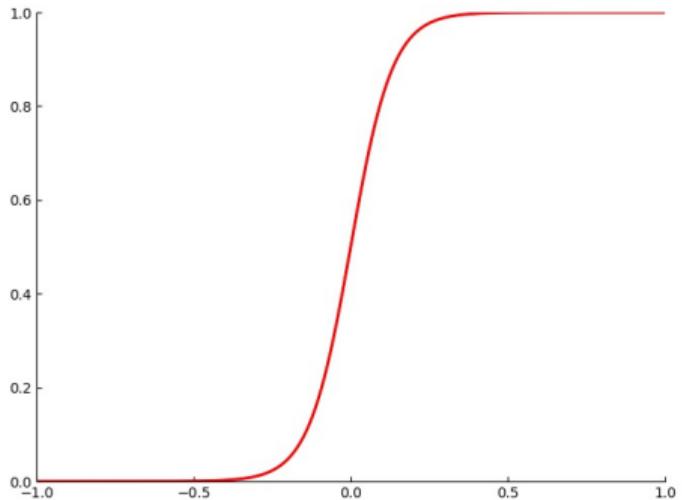
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 7, b = 0$$



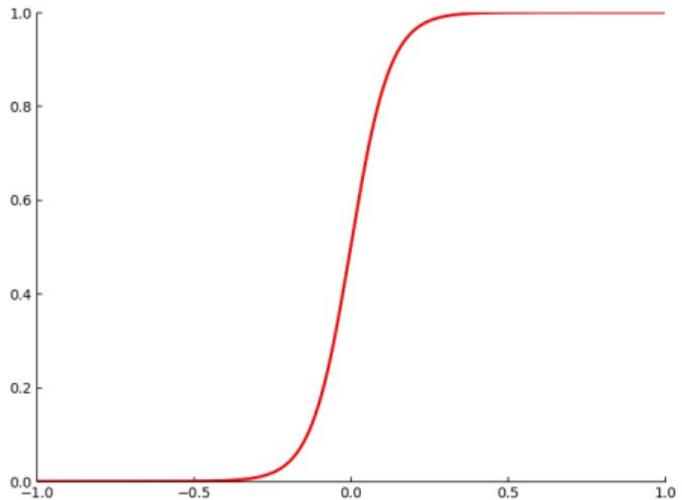
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 8, b = 0$$



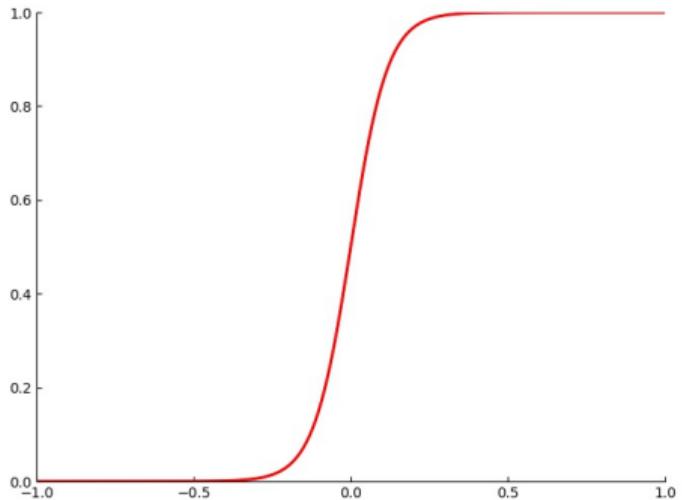
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 9, b = 0$$



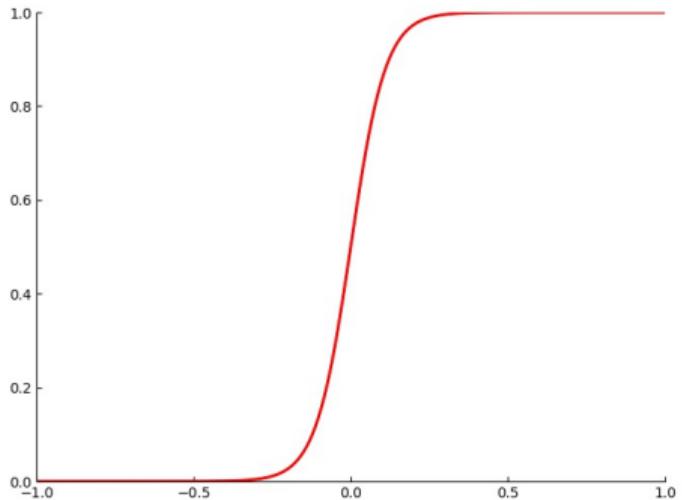
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 10, b = 0$$



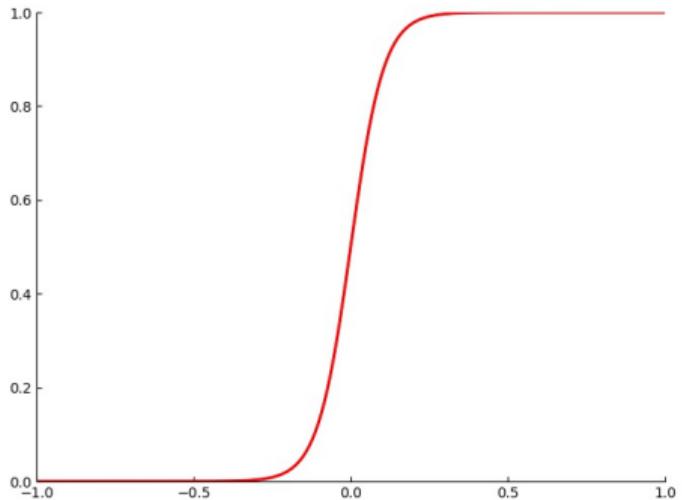
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 11, b = 0$$



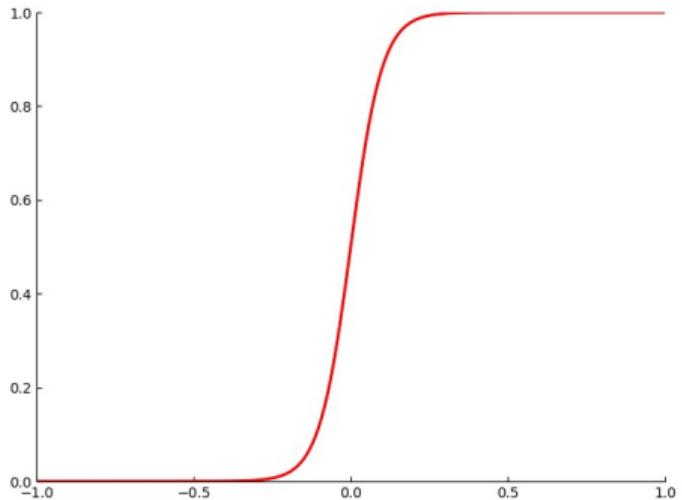
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 12, b = 0$$



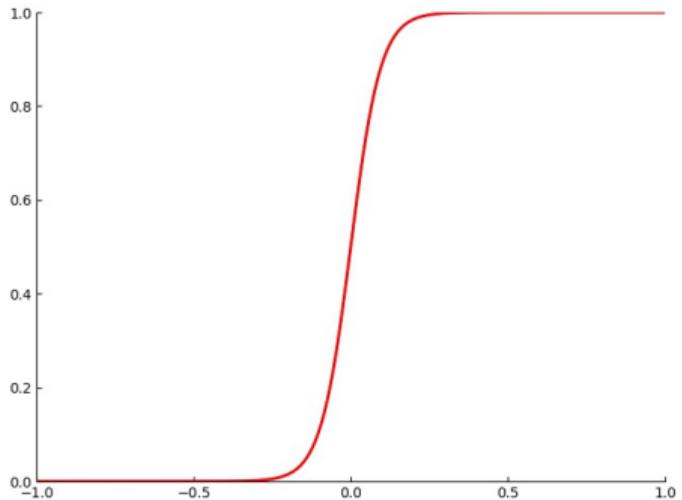
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 13, b = 0$$



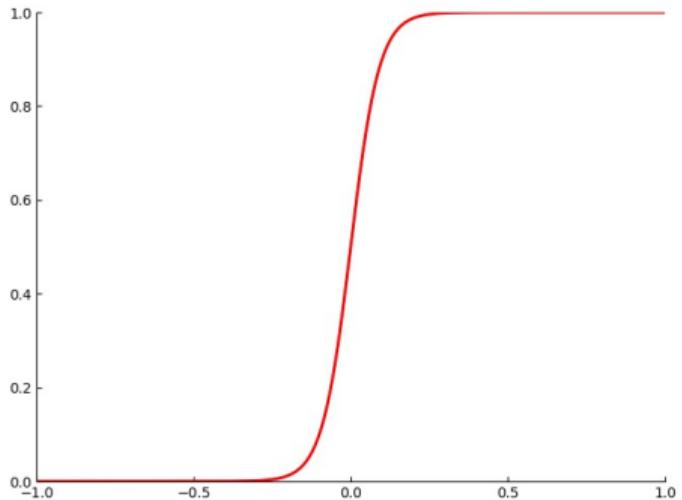
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 14, b = 0$$



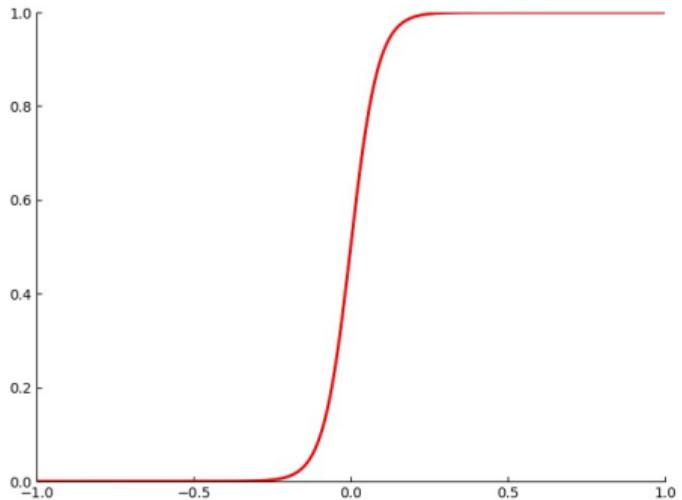
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 15, b = 0$$



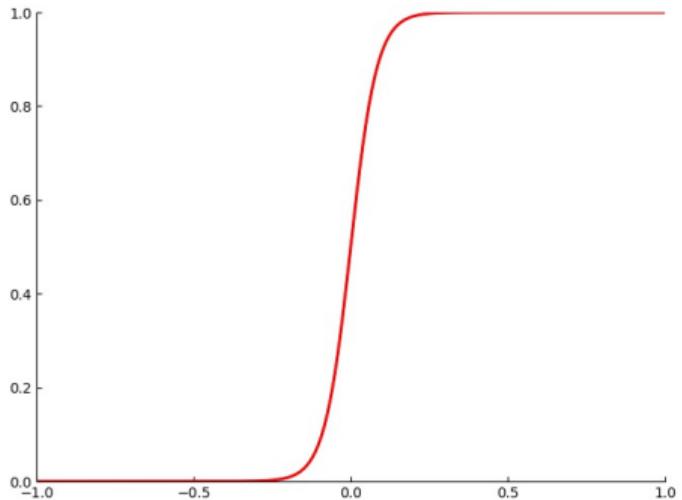
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 16, b = 0$$



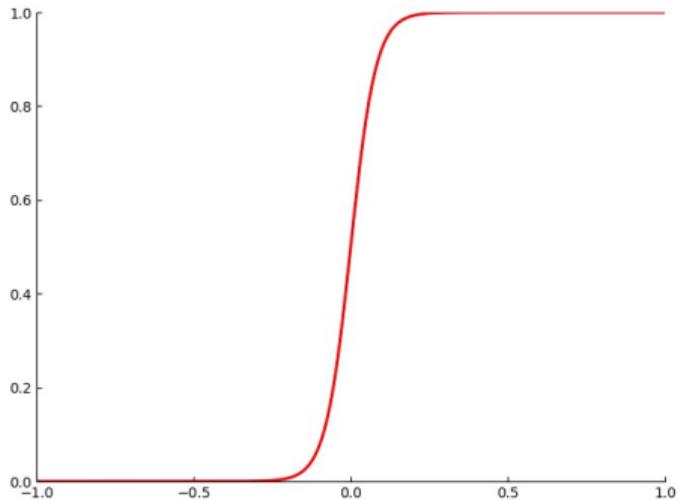
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 17, b = 0$$



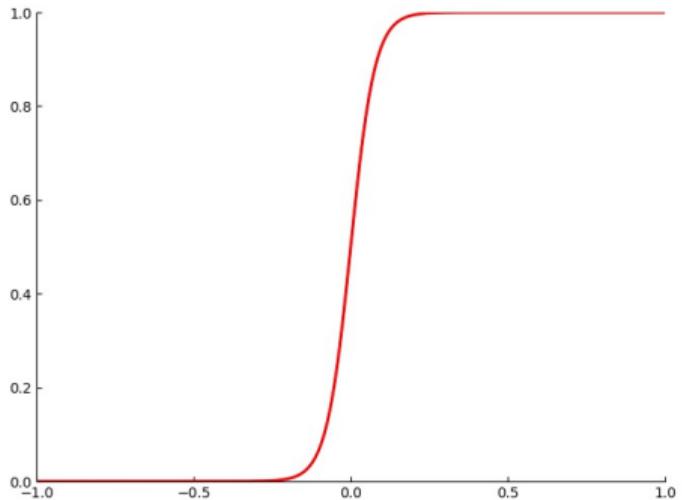
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 18, b = 0$$



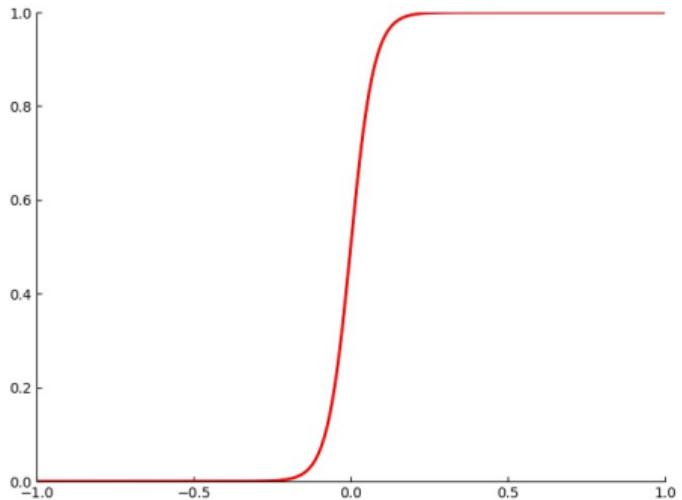
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 19, b = 0$$



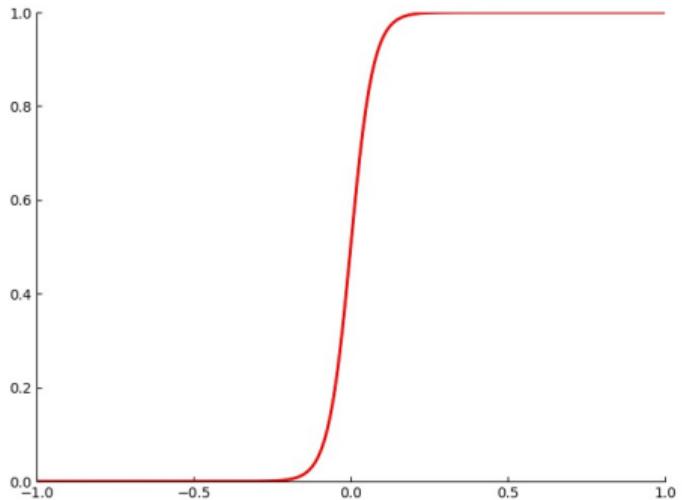
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 20, b = 0$$



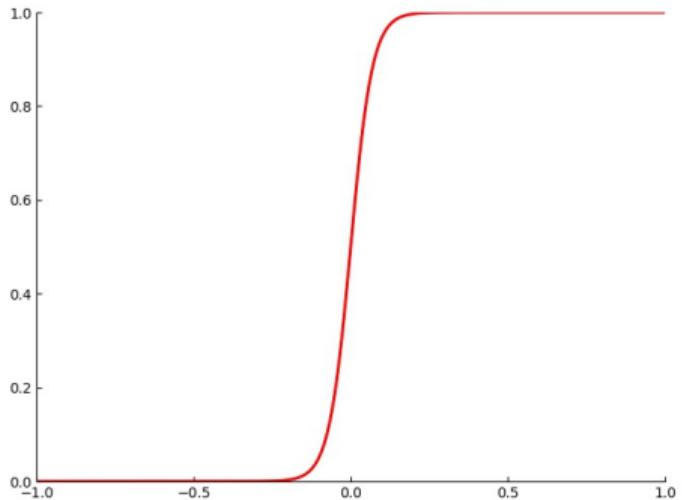
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 21, b = 0$$



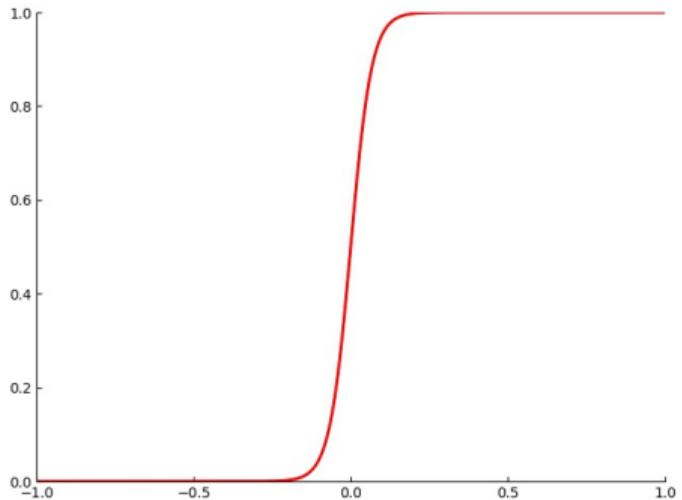
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 22, b = 0$$



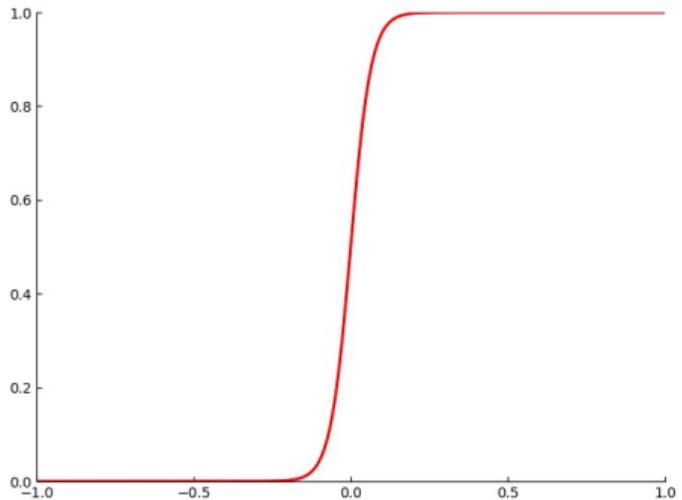
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 23, b = 0$$



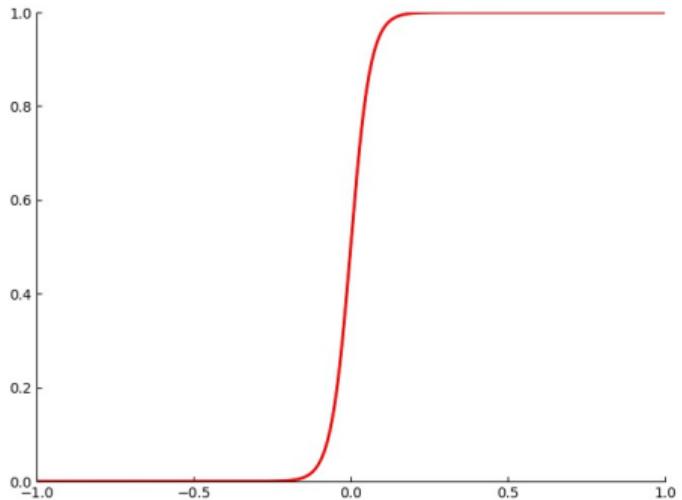
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 24, b = 0$$



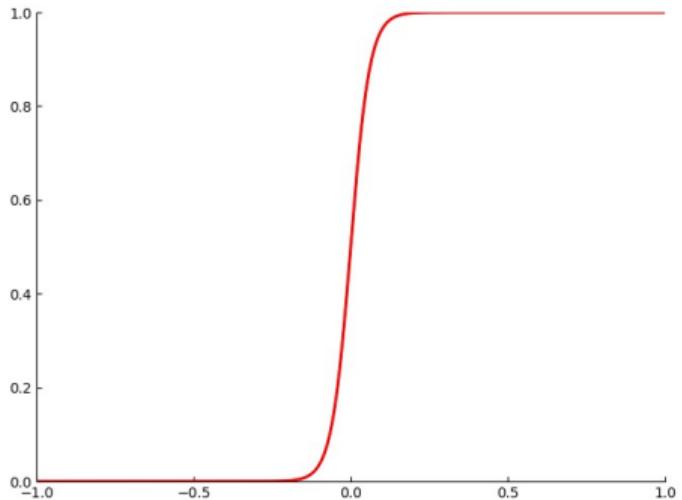
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 25, b = 0$$



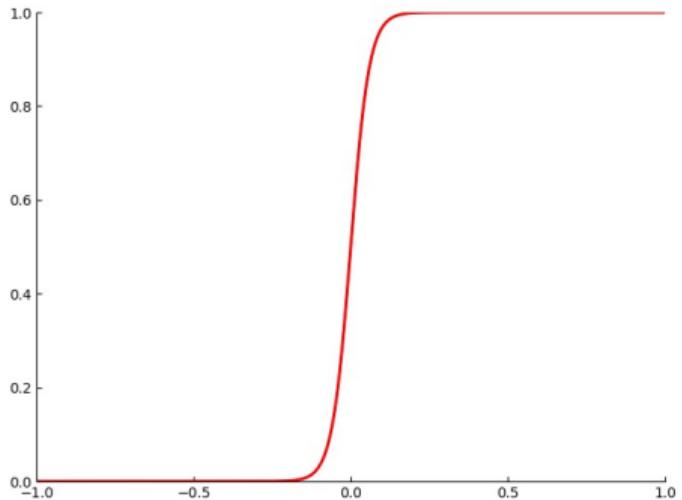
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 26, b = 0$$



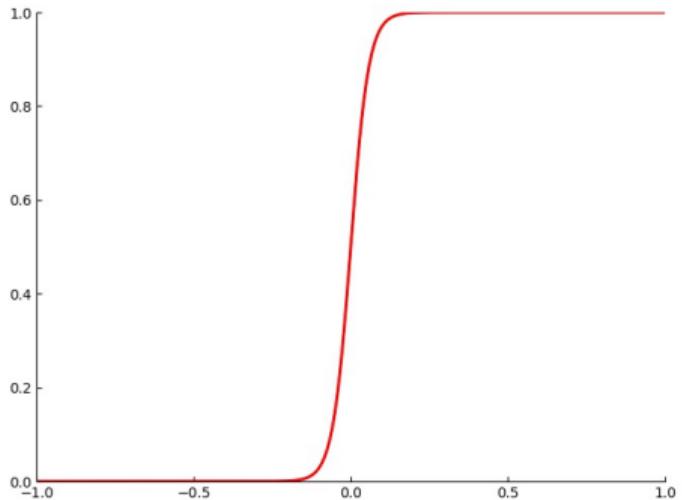
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 27, b = 0$$



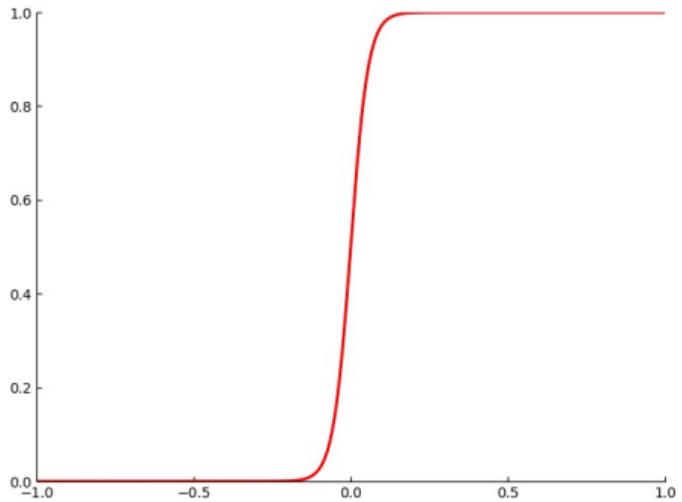
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 28, b = 0$$



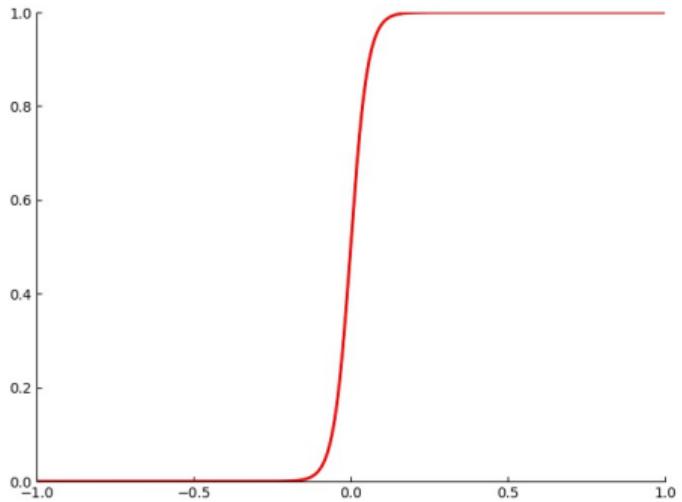
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 29, b = 0$$



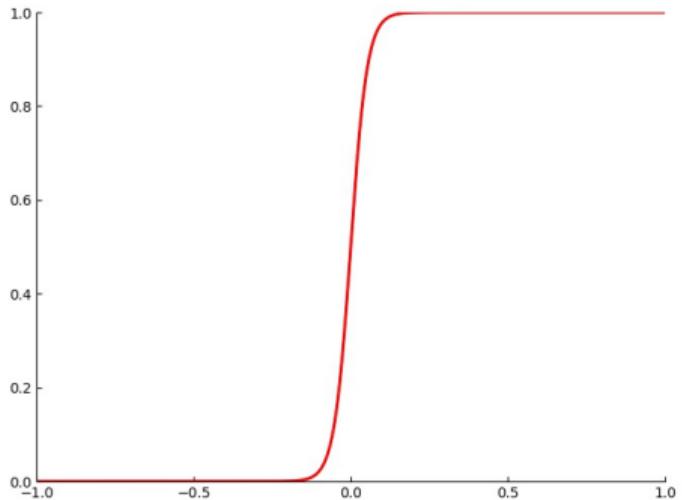
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 30, b = 0$$



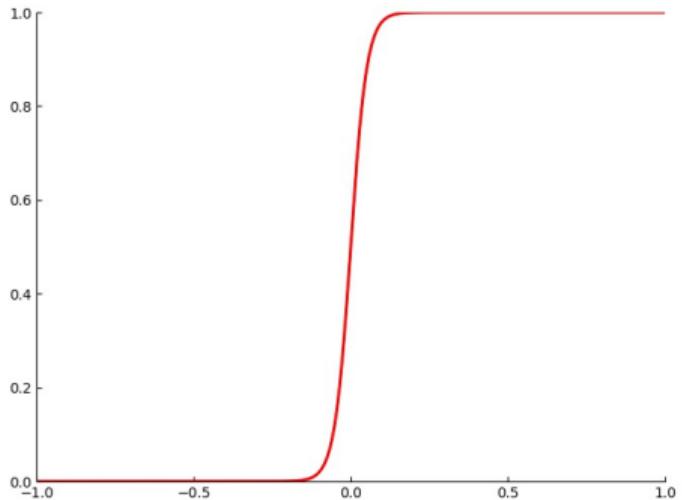
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 31, b = 0$$



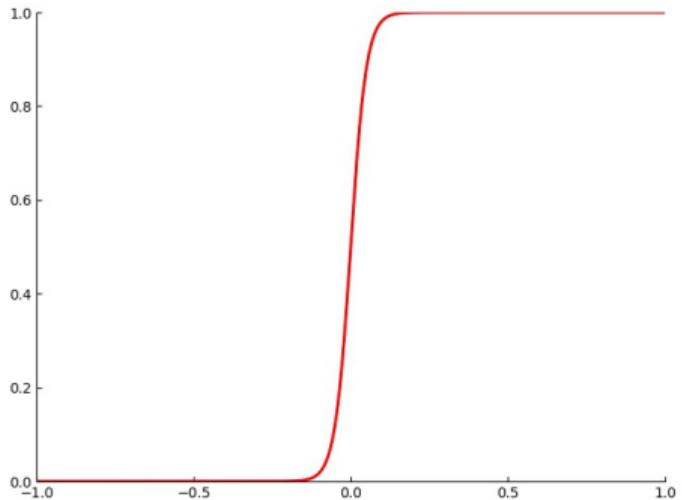
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 32, b = 0$$



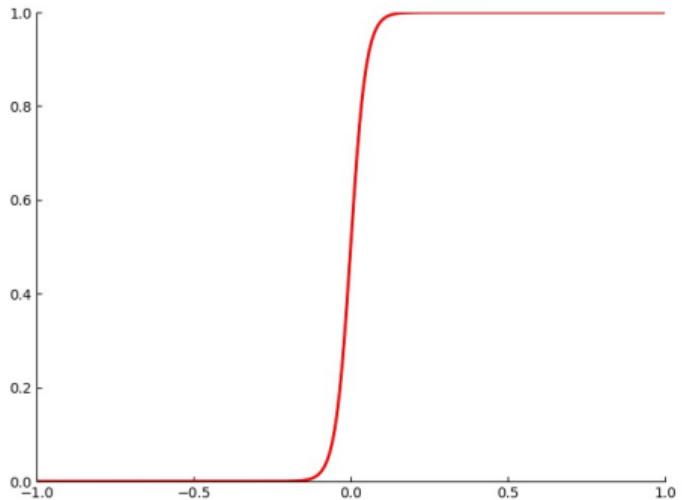
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 33, b = 0$$



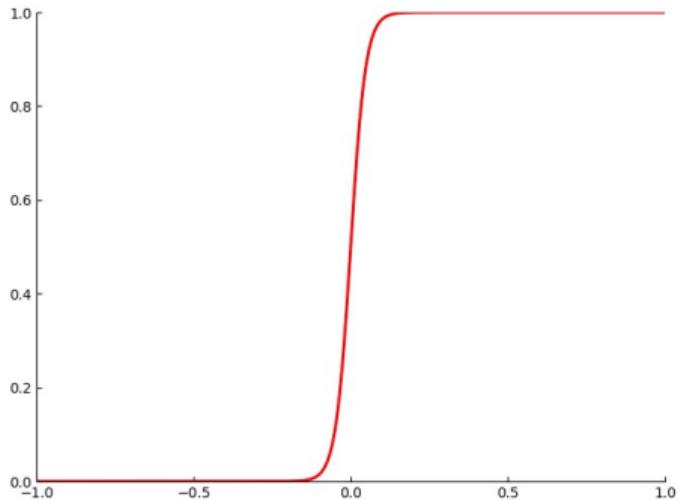
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 34, b = 0$$



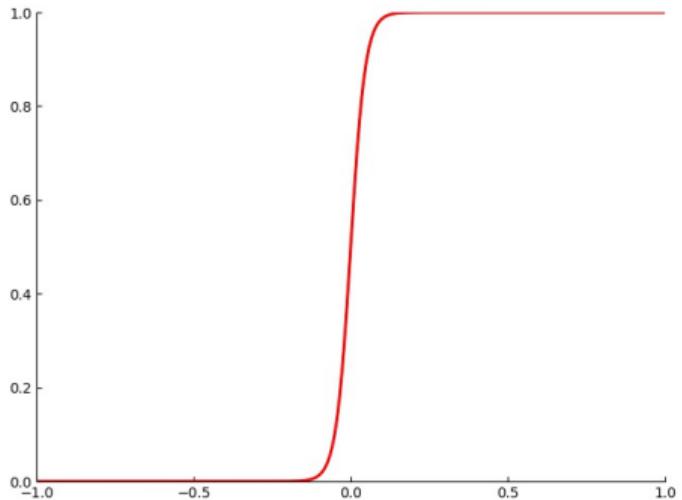
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 35, b = 0$$



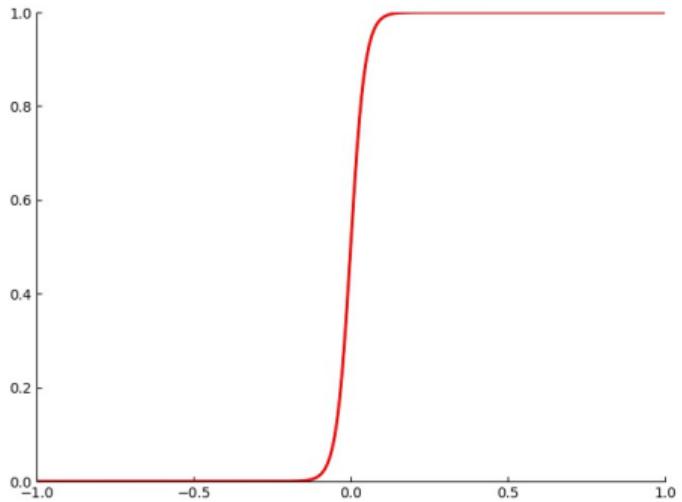
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 36, b = 0$$



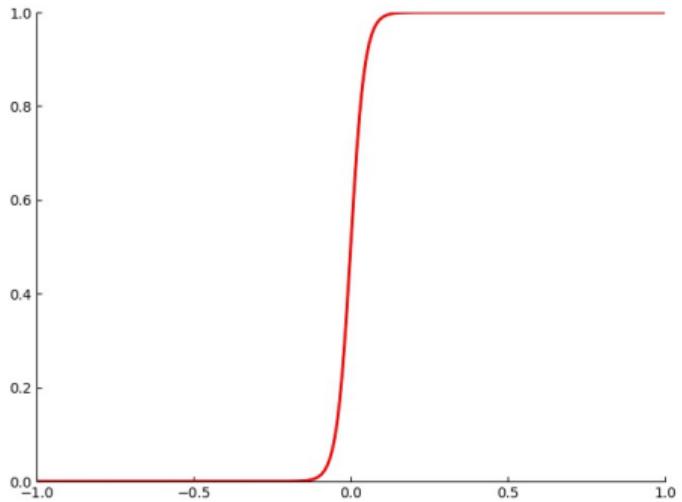
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 37, b = 0$$



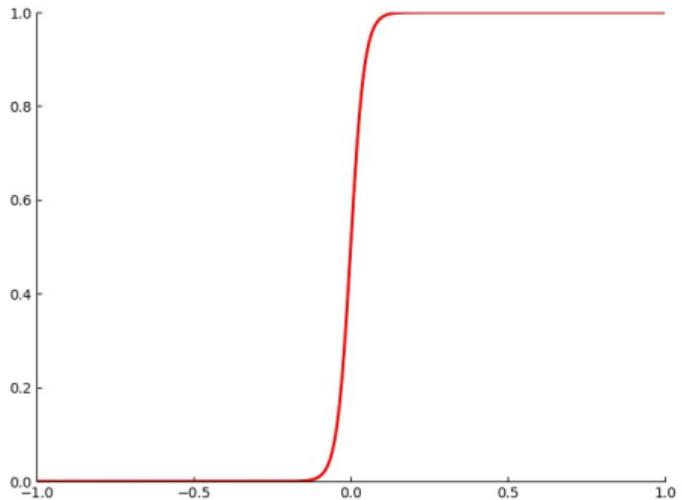
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 38, b = 0$$



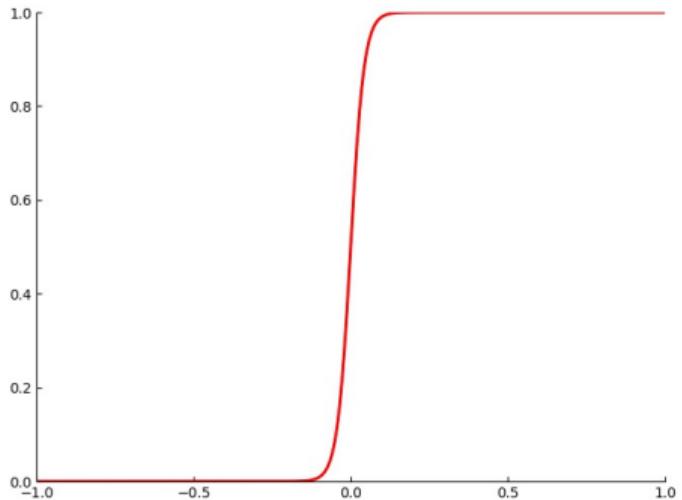
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 39, b = 0$$



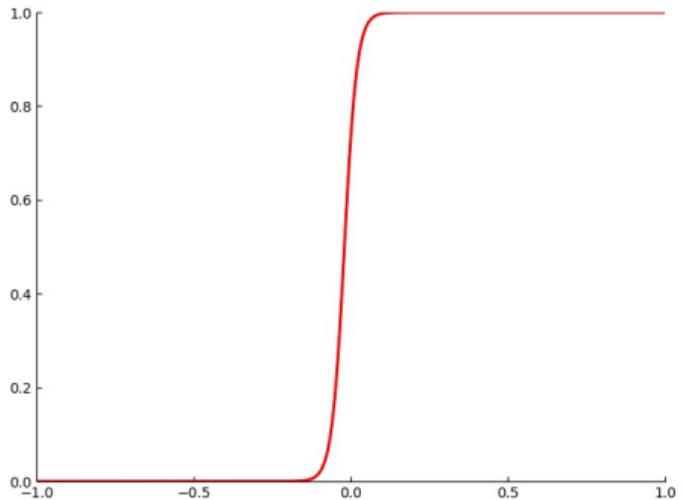
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 40, b = 0$$



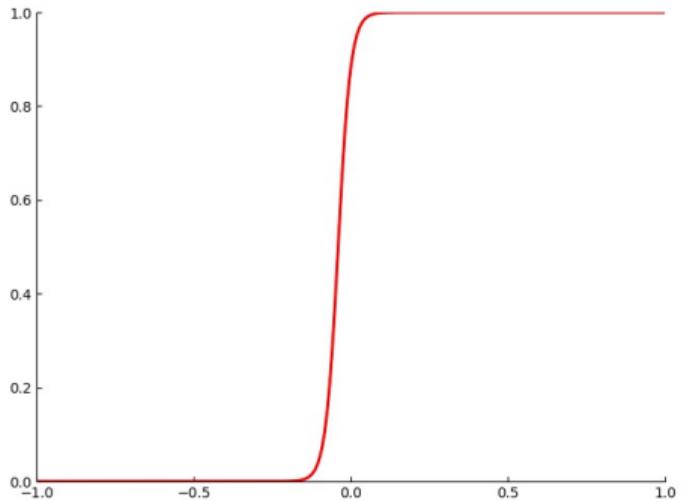
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 41, b = 0$$



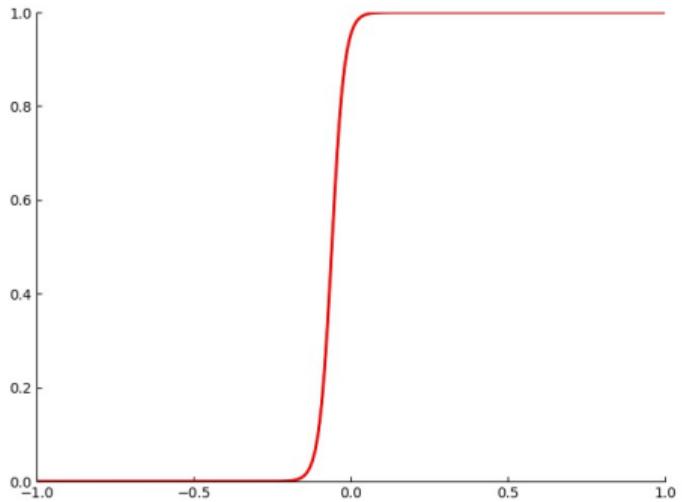
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 1$$



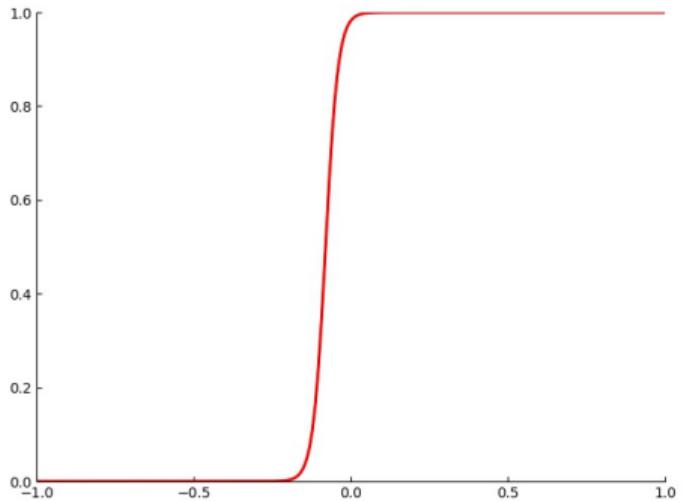
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 2$$



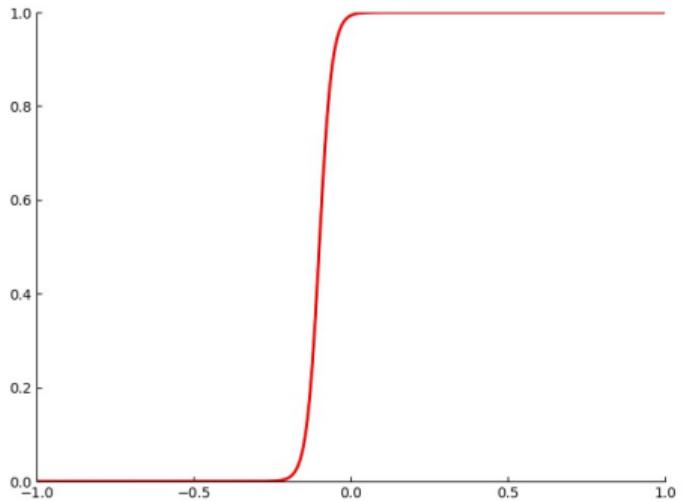
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 3$$



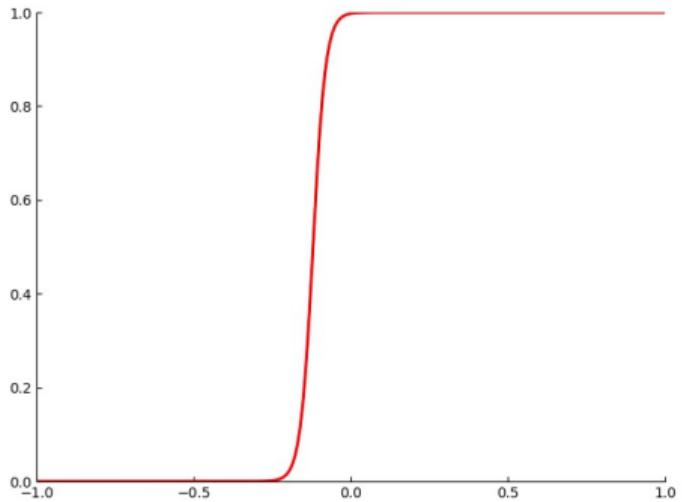
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 4$$



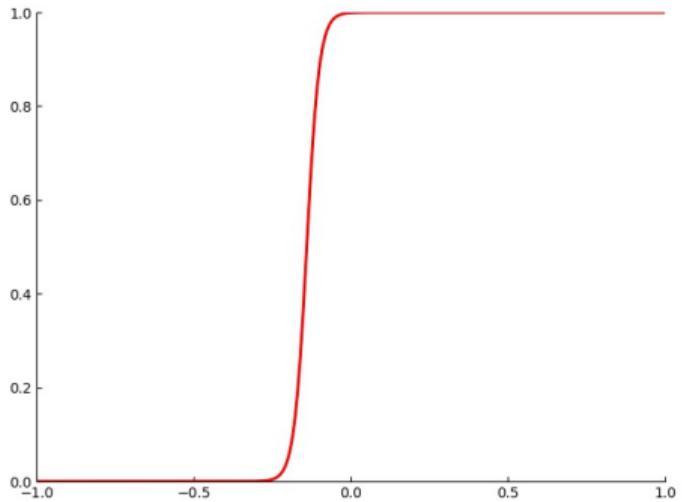
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 5$$



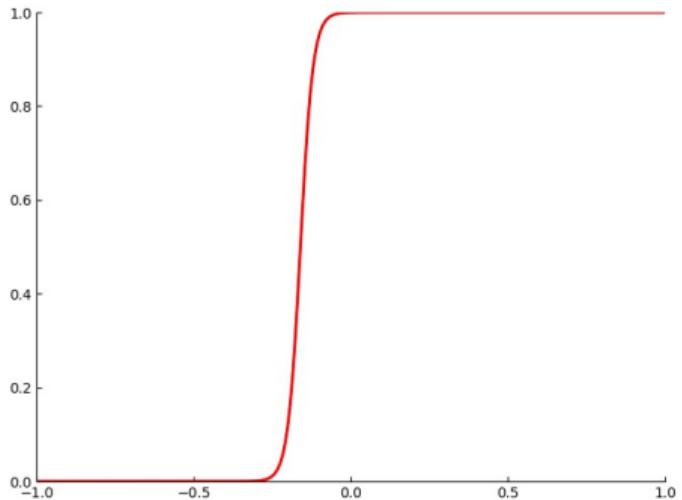
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 6$$



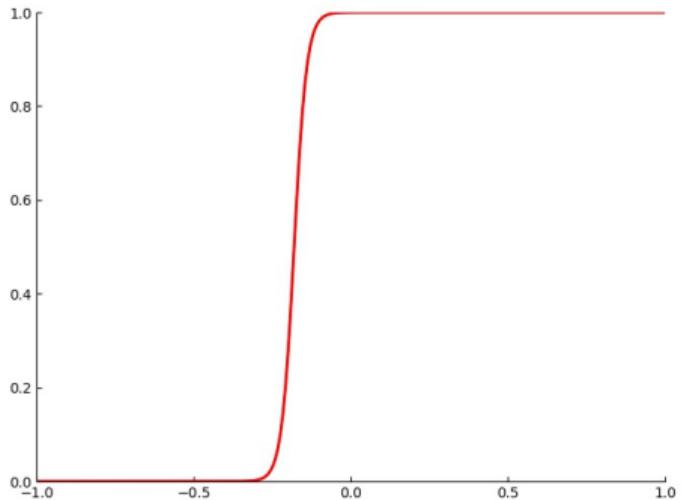
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 7$$



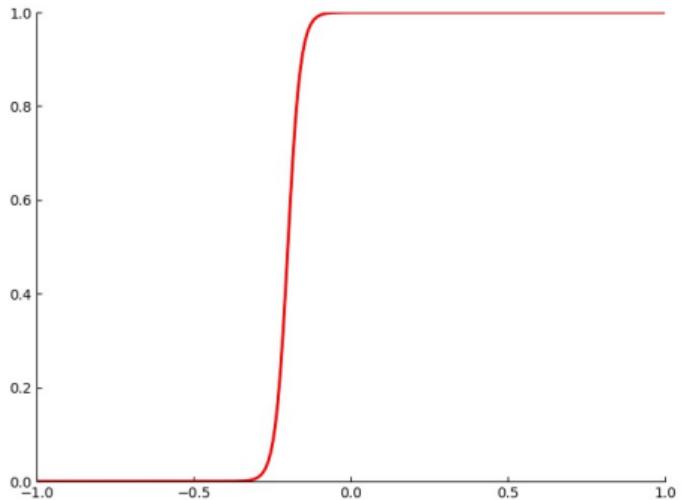
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 8$$



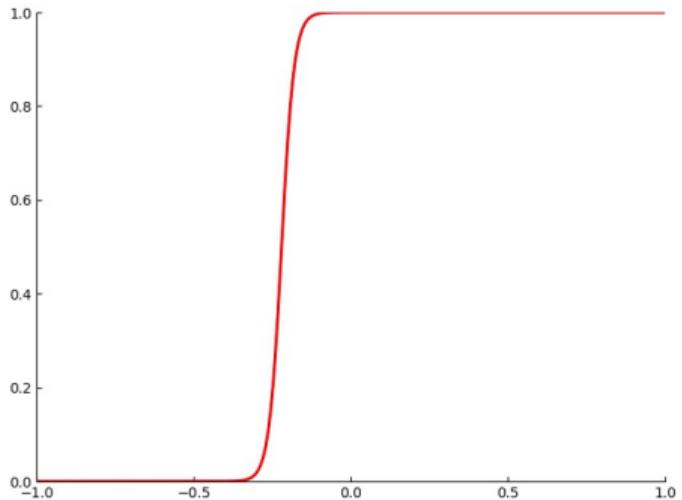
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 9$$



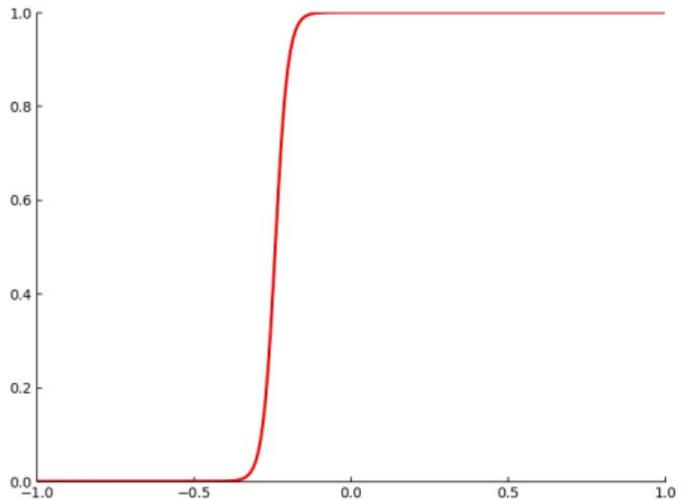
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 10$$



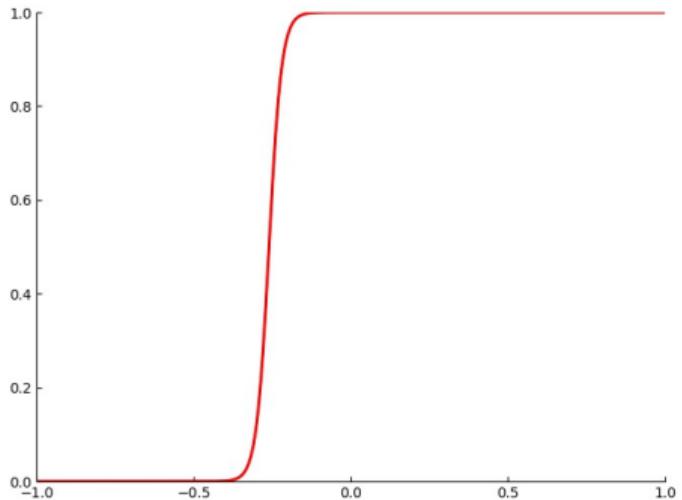
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 11$$



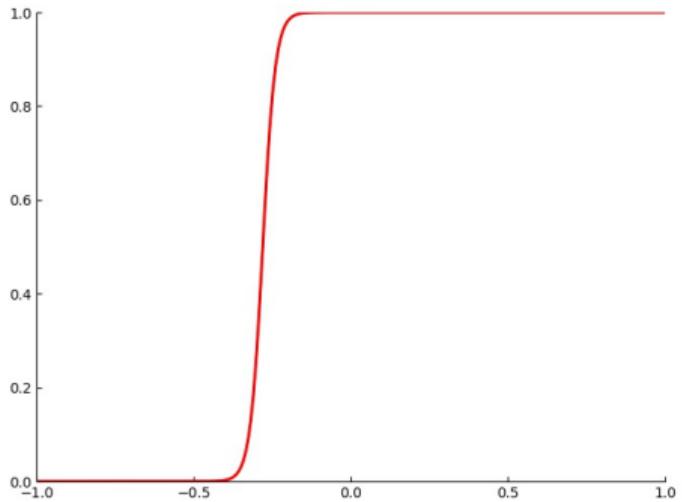
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 12$$



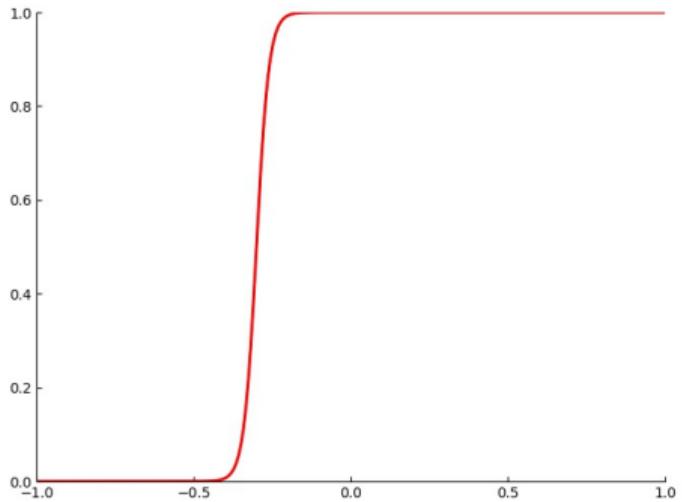
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 13$$



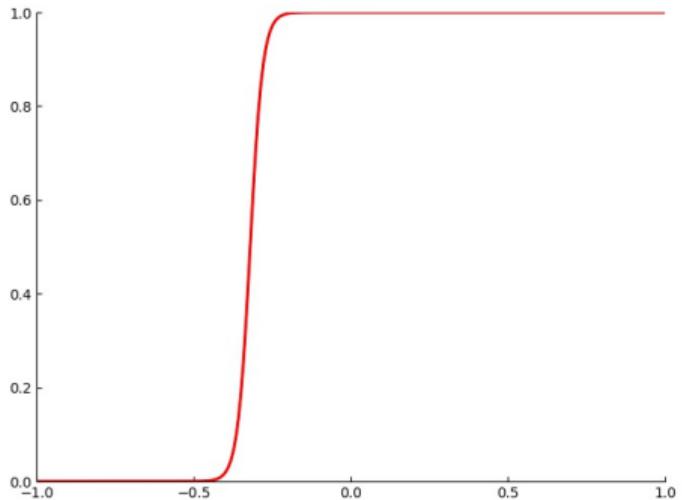
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 14$$



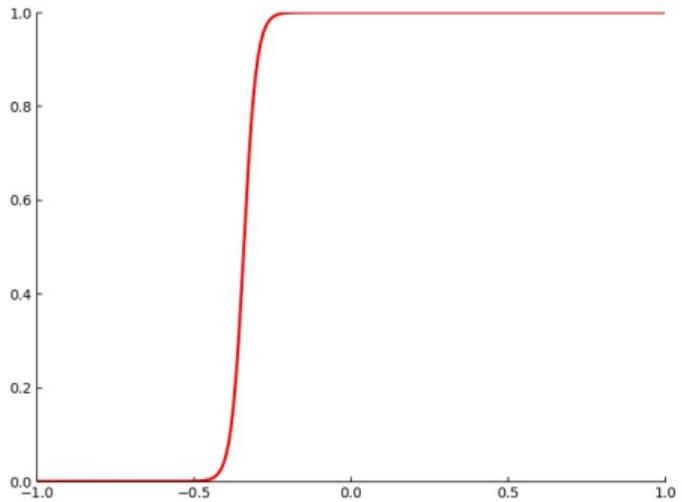
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 15$$



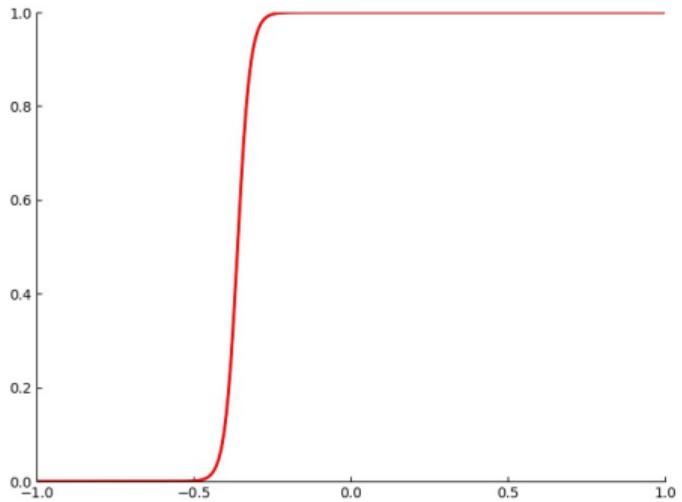
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 16$$



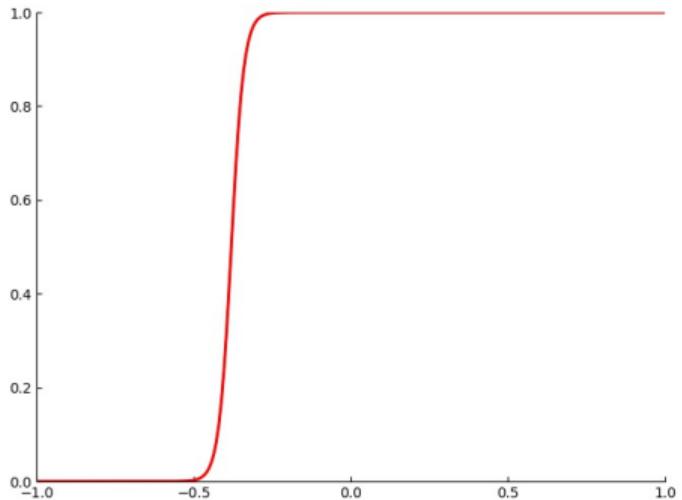
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 近一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 17$$



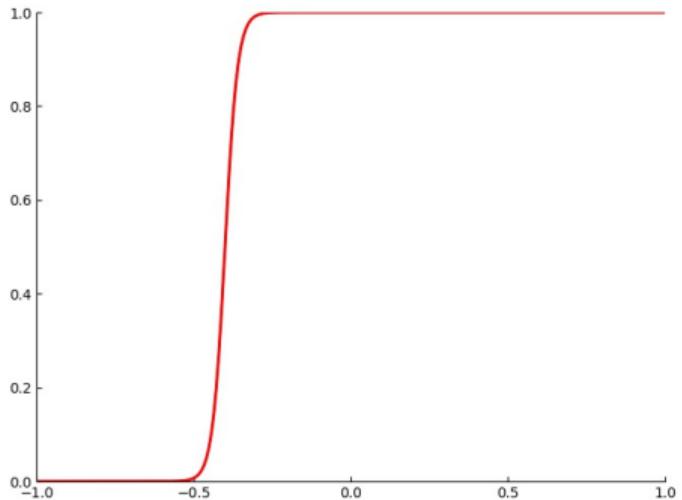
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 18$$



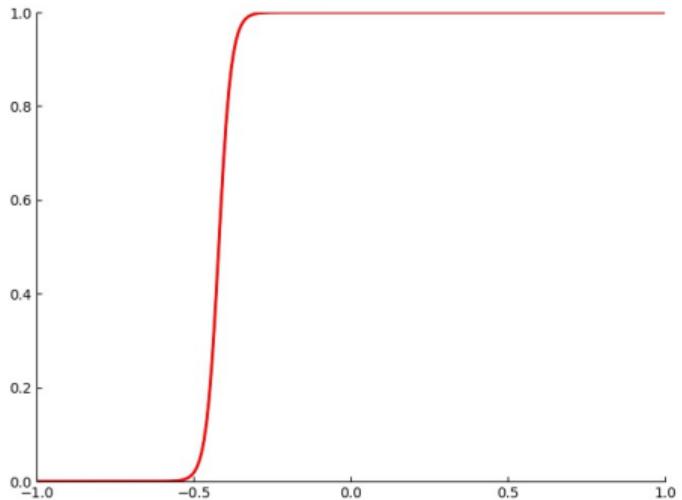
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 19$$



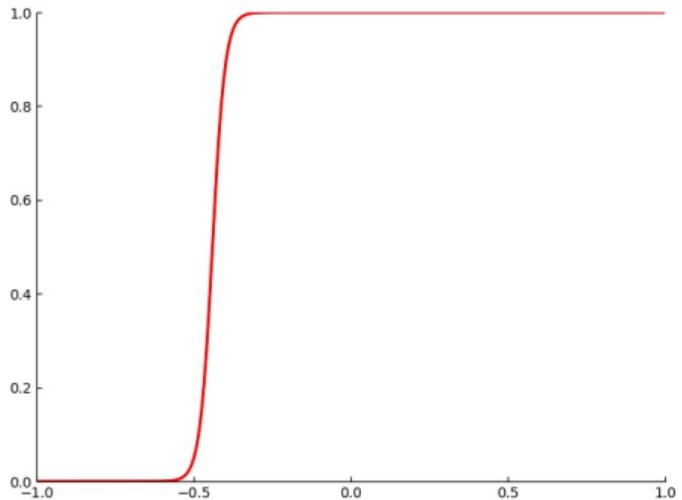
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 20$$



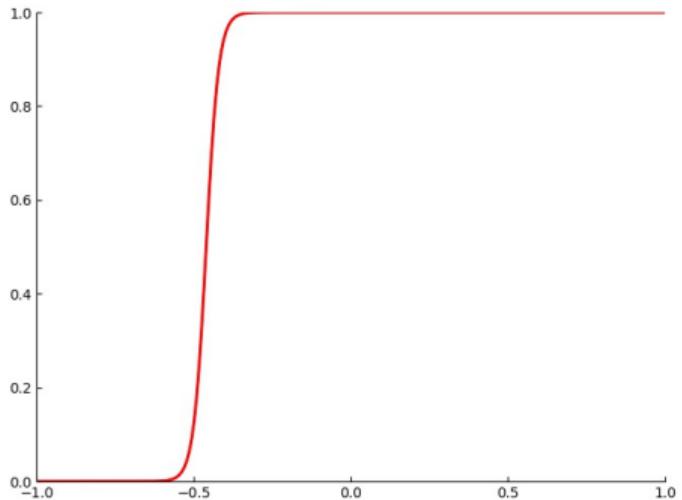
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 21$$



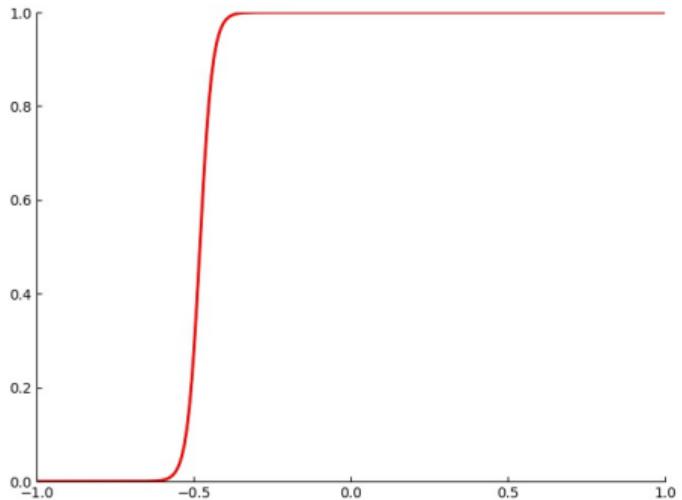
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 22$$



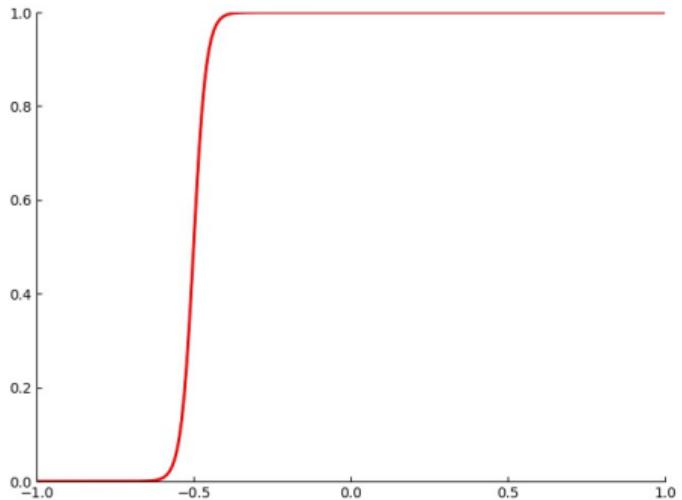
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 23$$



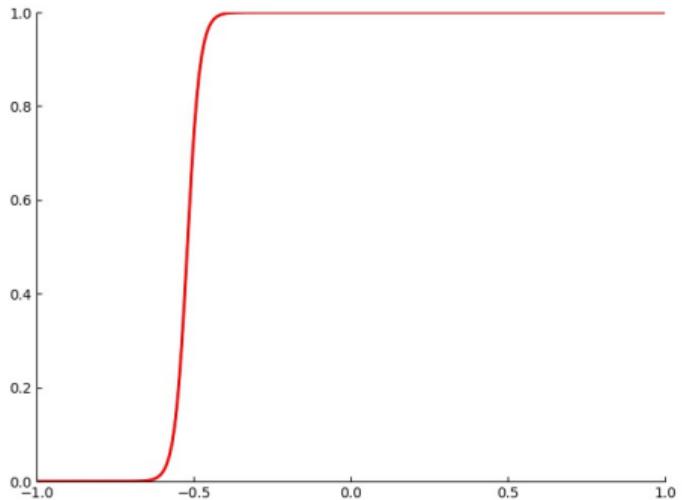
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 24$$



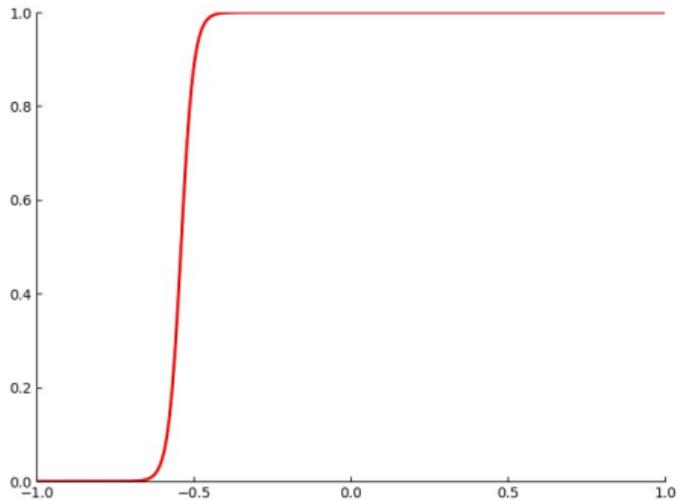
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 25$$



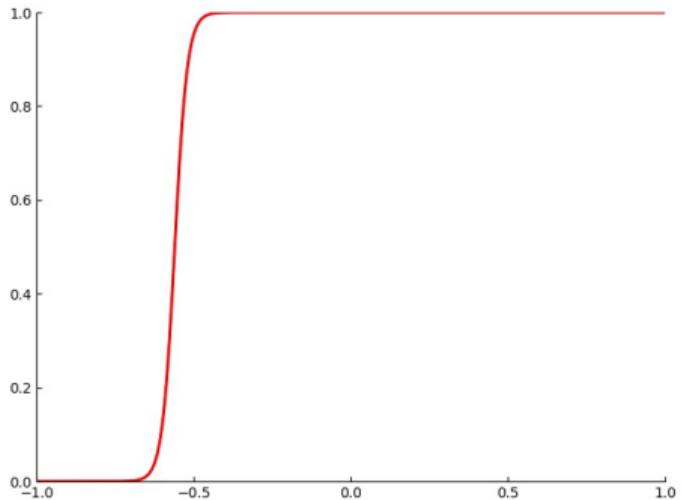
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 26$$



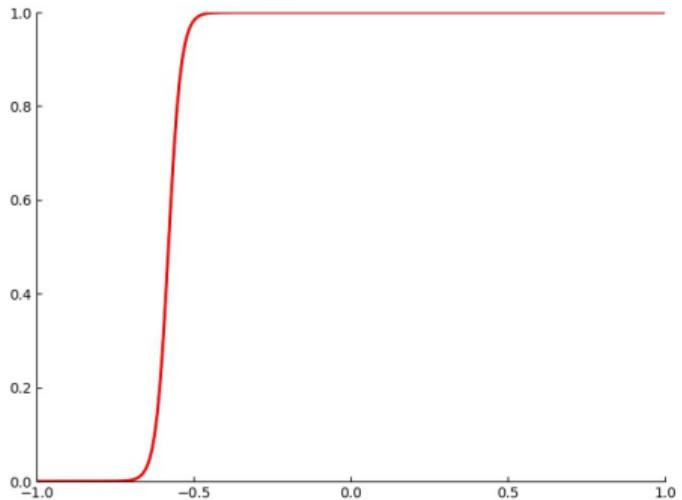
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 27$$



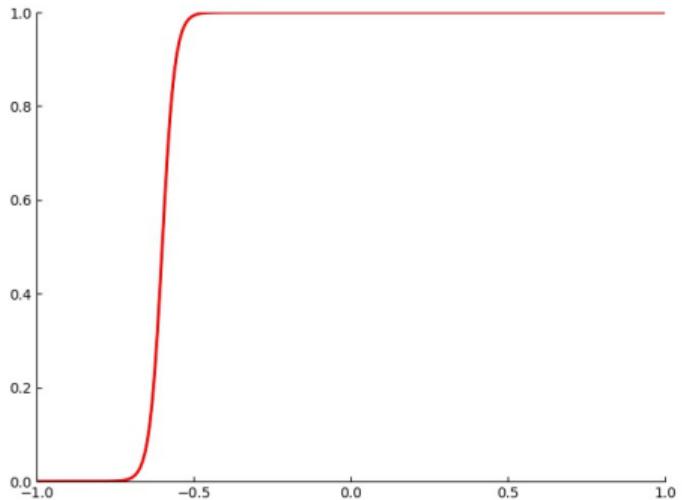
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 28$$



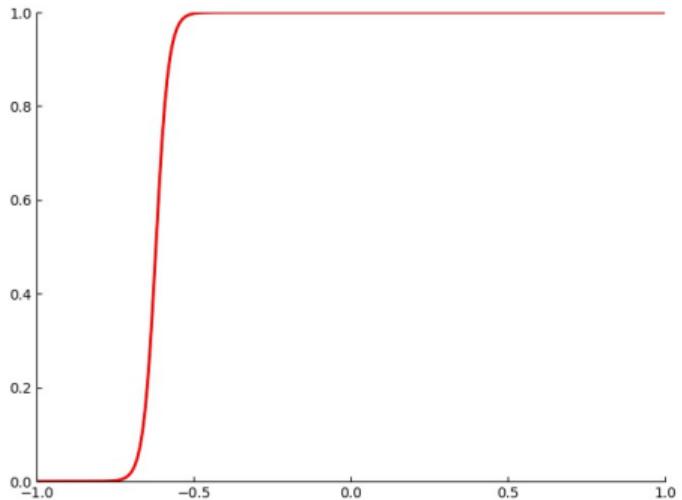
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 29$$



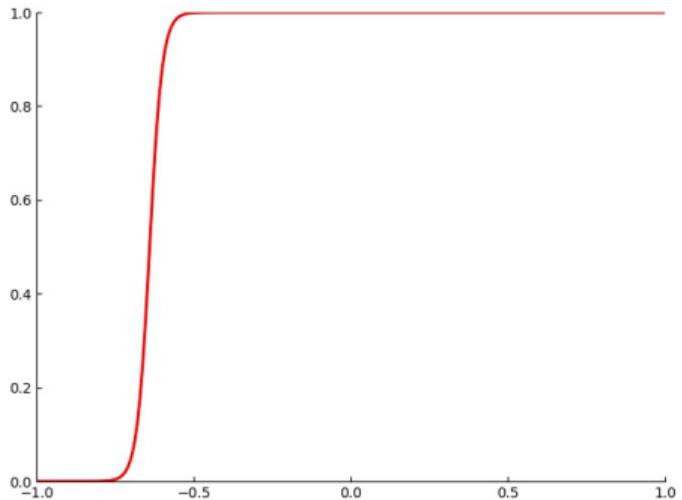
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 30$$



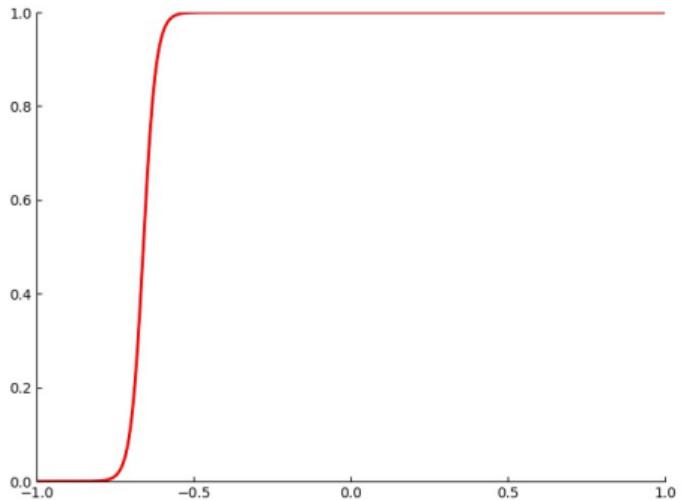
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 近一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 31$$



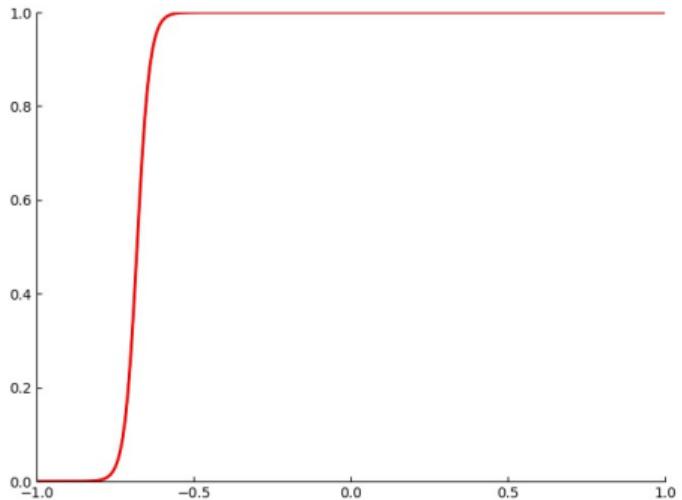
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 近一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 32$$



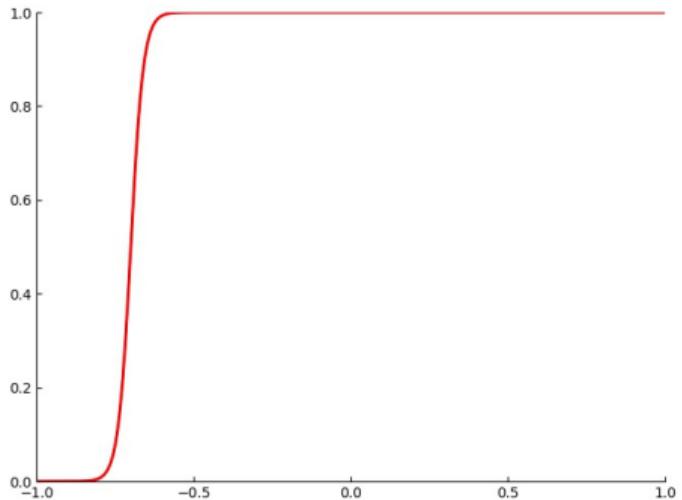
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 近一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 33$$



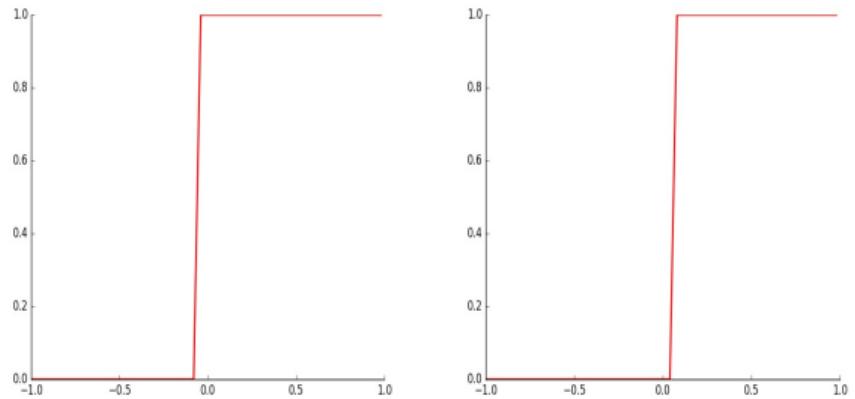
- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 34$$

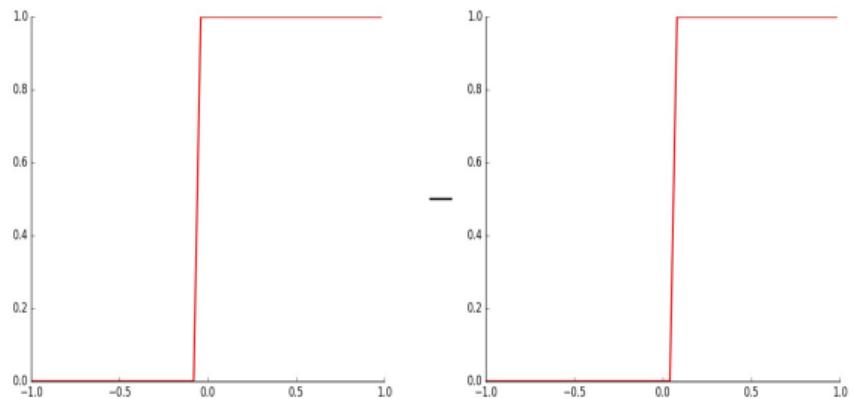


- 如果采用 logistic 函数，并且给 w 赋一个很大的值，logistic 函数将变成一个 step 函数
- 看看改变 w 的值，logistic 函数是如何变化的
- 进一步，通过改变 b 的值，可以调整 logistic 函数在 x 轴上的位置

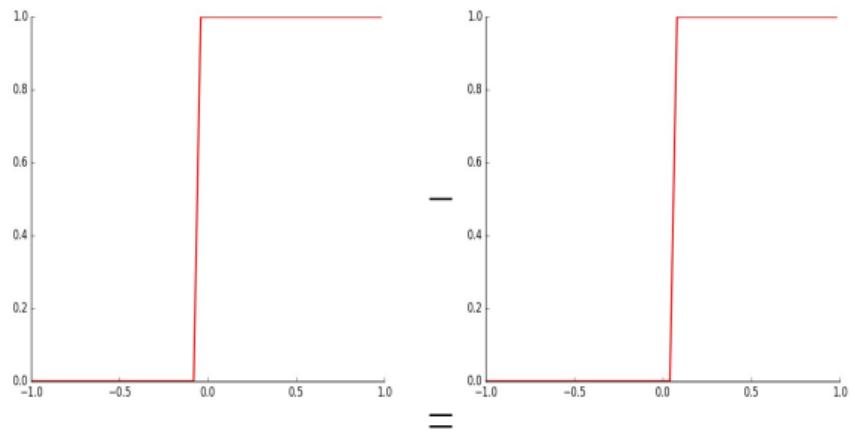
$$\sigma(x) = \frac{1}{1+e^{-(wx+b)}} \quad w = 50, b = 35$$



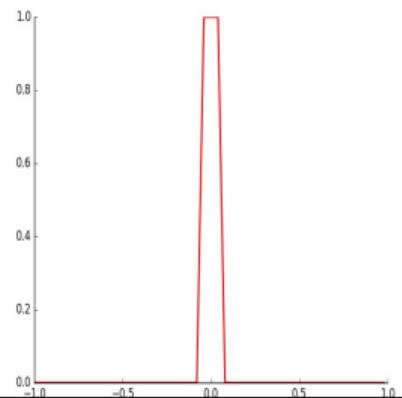
- 现在看看，如果让两个这样的 logistic 函数（不同的 b ）相减，会得到什么？

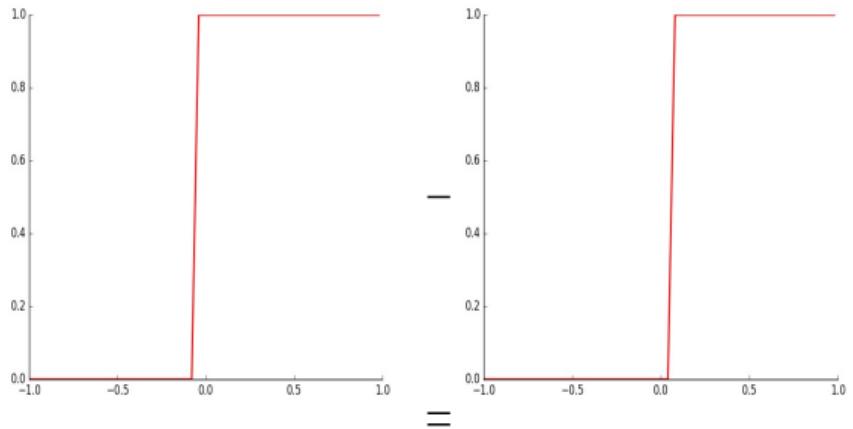


- 现在看看，如果让两个这样的 logistic 函数（不同的 b ）相减，会得到什么？

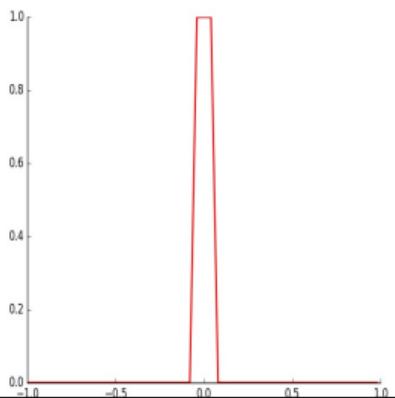


- 现在看看，如果让两个这样的 logistic 函数（不同的 b ）相减，会得到什么？



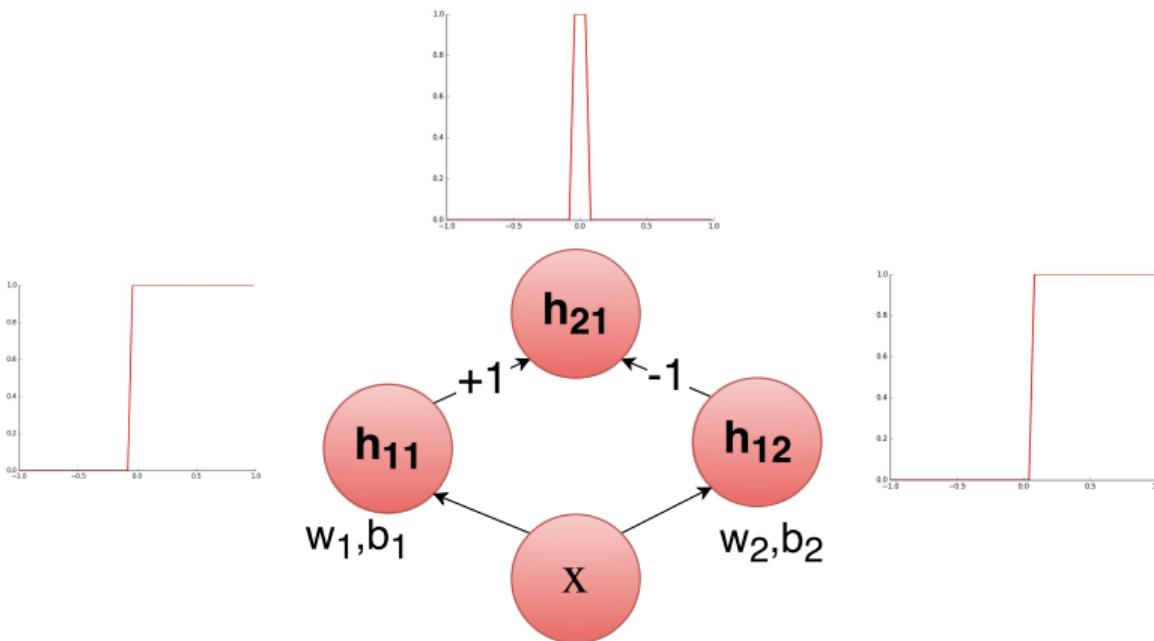


- 现在看看，如果让两个这样的 logistic 函数（不同的 b ）相减，会得到什么？
- 是不是得到了一个 tower 函数？





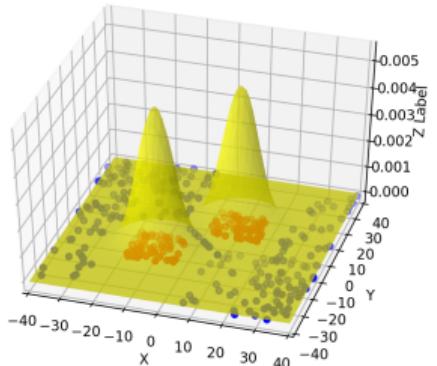
- 是否可以设计一个神经网络来实现让两个 logistic 函数相减呢 ?



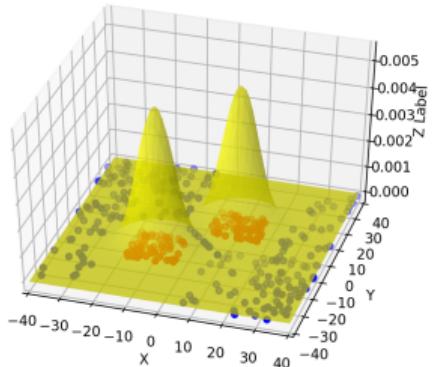
- 如果有多个输入该怎么办？

- 如果有多个输入该怎么办？
- 假定我们要预测在海底的一个特定位置是否有石油 (Yes/No)

- 如果有多个输入该怎么办？
- 假定我们要预测在海底的一个特定位置是否有石油 (Yes/No)
- 进一步假定预测是基于两个因素：含盐量 Salinity (x_1) 和压力 Pressure (x_2)



- 如果有多个输入该怎么办?
- 假定我们要预测在海底的一个特定位置是否有石油 (Yes/No)
- 进一步假定预测是基于两个因素: 含盐量 Salinity (x_1) 和压力 Pressure (x_2)
- 给定一些训练数据, $y(\text{oil}|\text{no-oil})$ 是以 x_1 和 x_2 为自变量的一个复杂函数

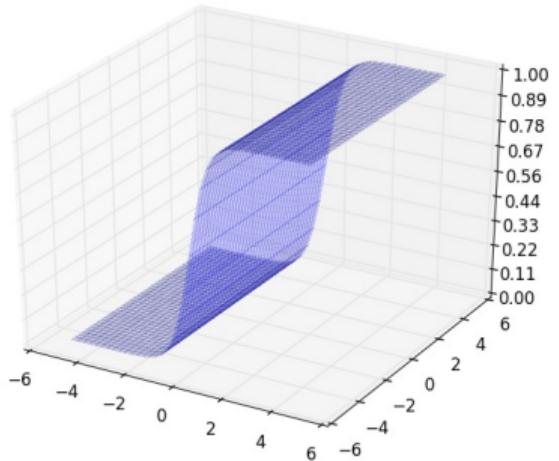


- 如果有多个输入该怎么办？
- 假定我们要预测在海底的一个特定位置是否有石油 (Yes/No)
- 进一步假定预测是基于两个因素：含盐量 Salinity (x_1) 和压力 Pressure (x_2)
- 给定一些训练数据， $y(\text{oil}|\text{no-oil})$ 是以 x_1 和 x_2 为自变量的一个复杂函数
- 如何设计一个神经网络来近似这个函数？



- 这是一个 2-D sigmoid 函数

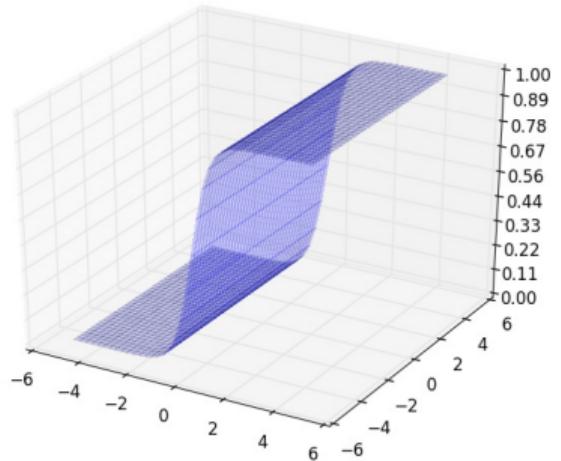
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$





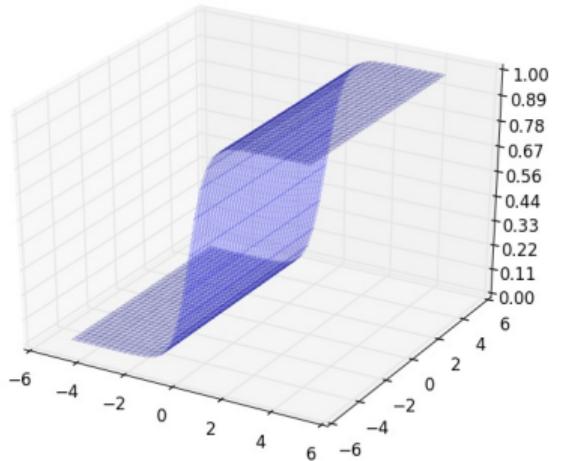
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?



$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

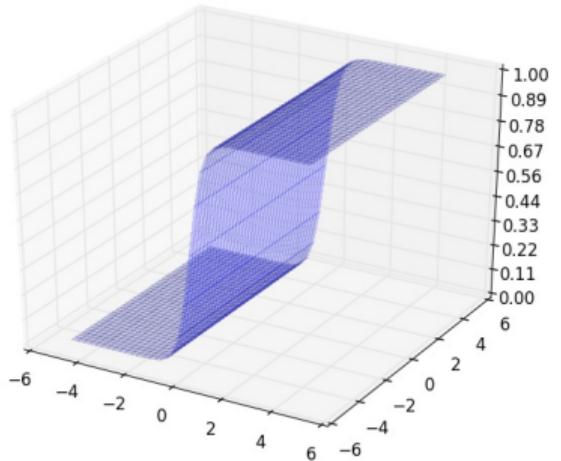
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 2, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

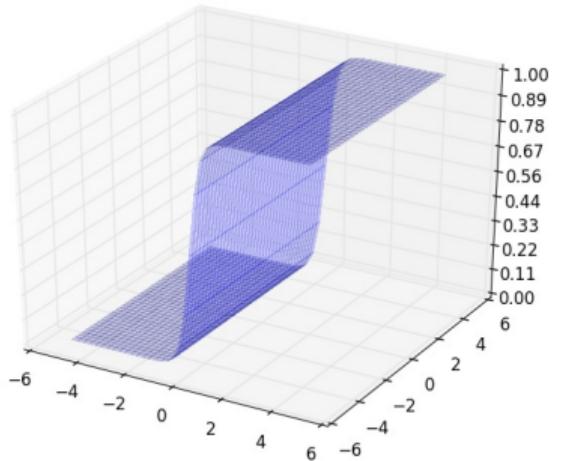
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 3, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

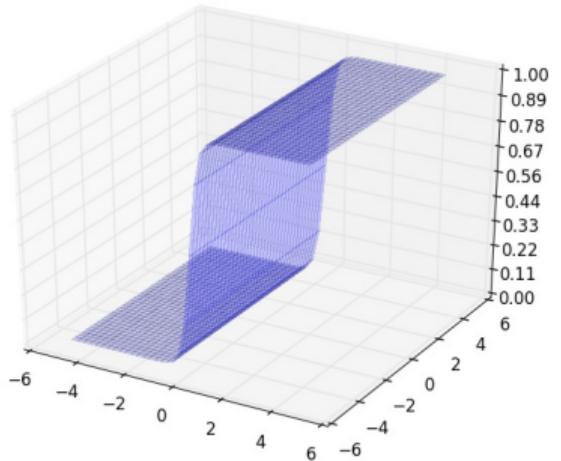
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 4, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

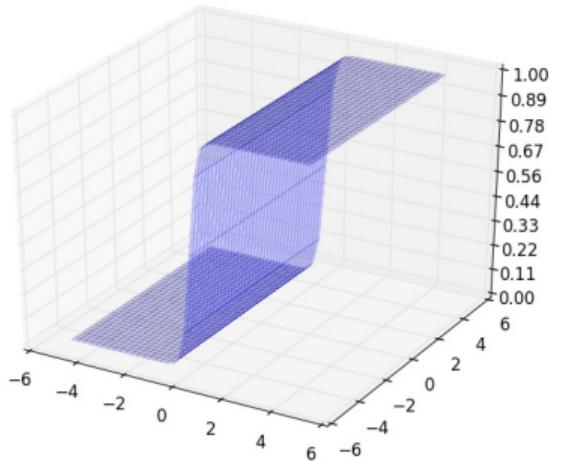
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 5, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

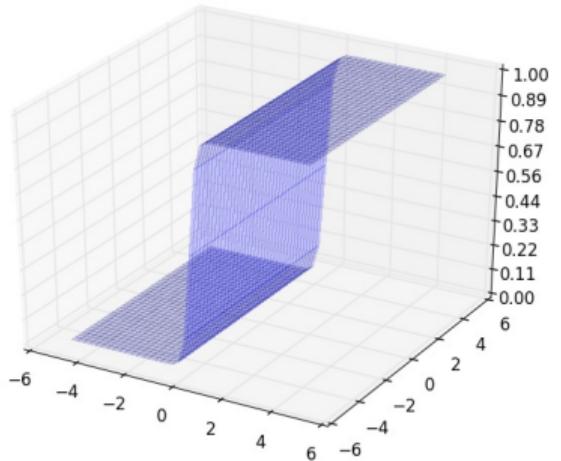
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 6, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

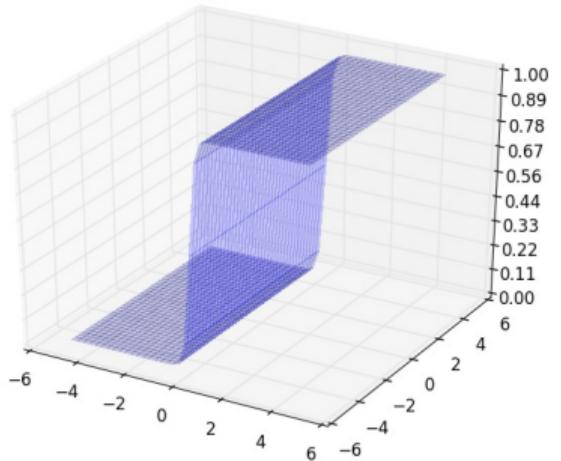
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 7, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

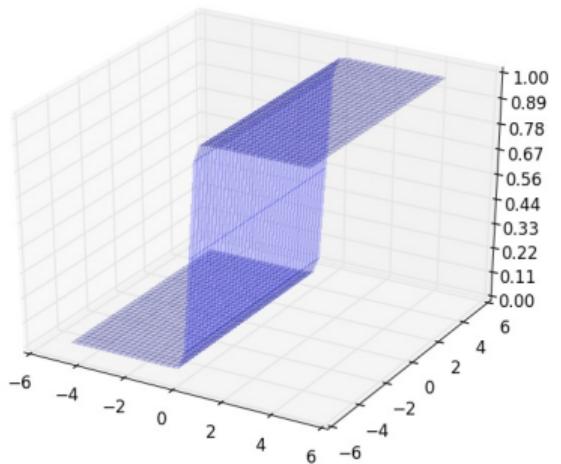
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 8, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

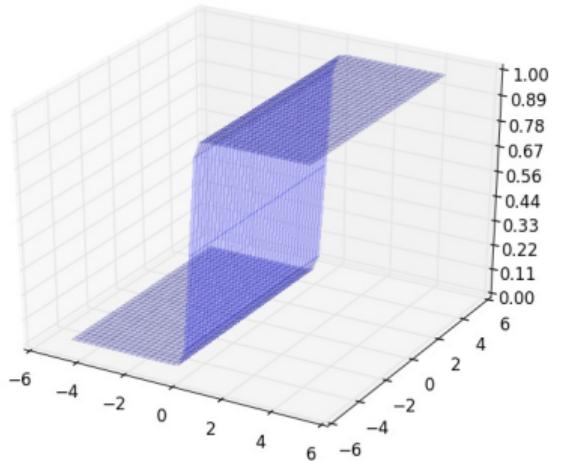
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 9, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

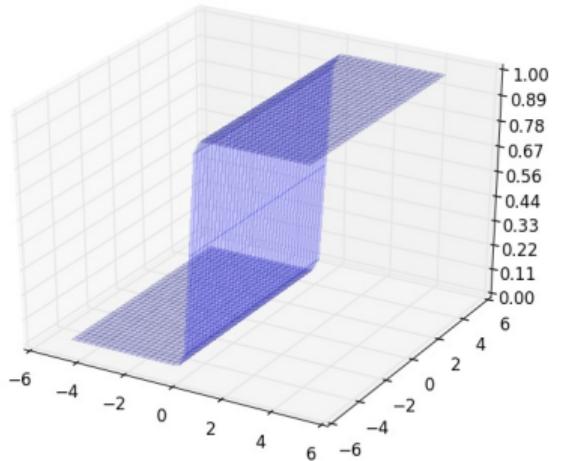
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 10, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

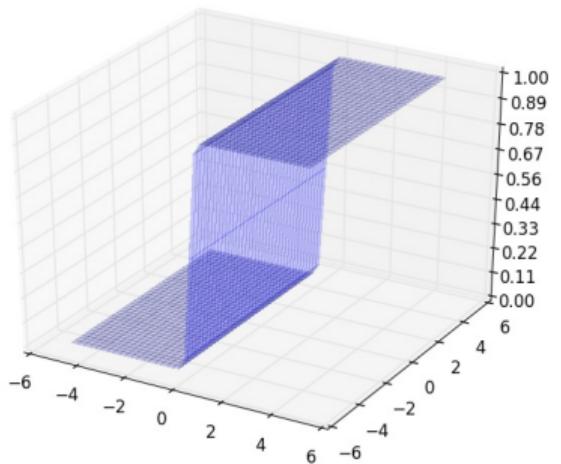
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 11, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

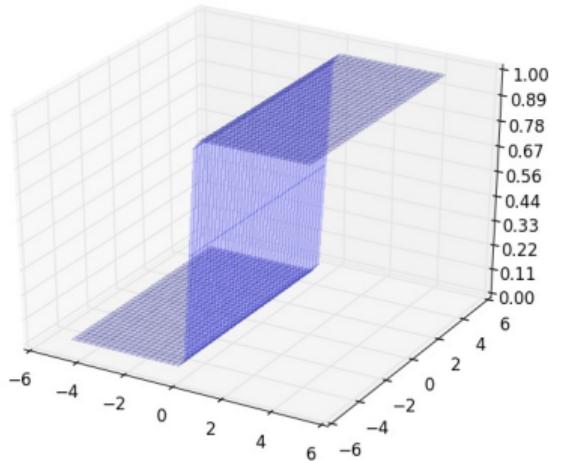
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 12, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

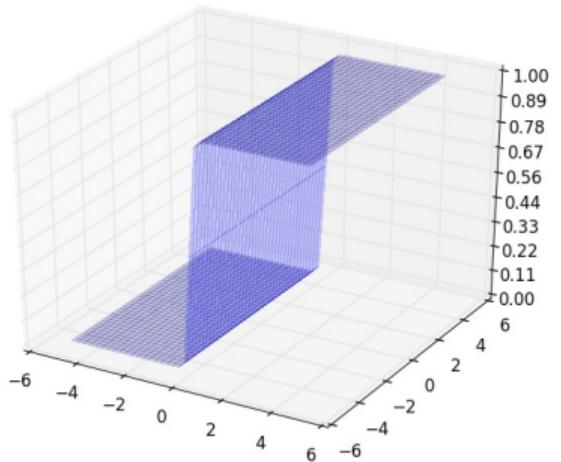
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 13, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

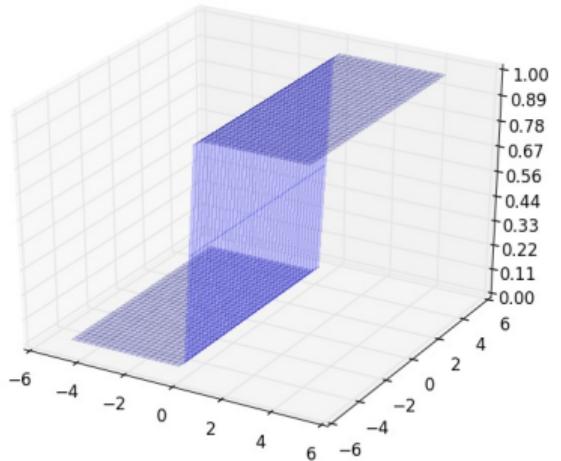
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 14, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

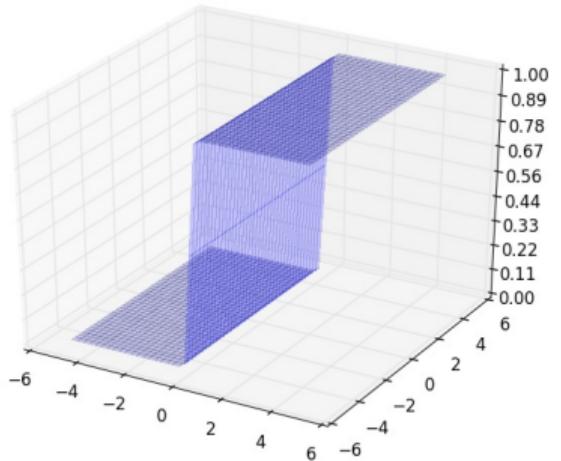
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 15, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

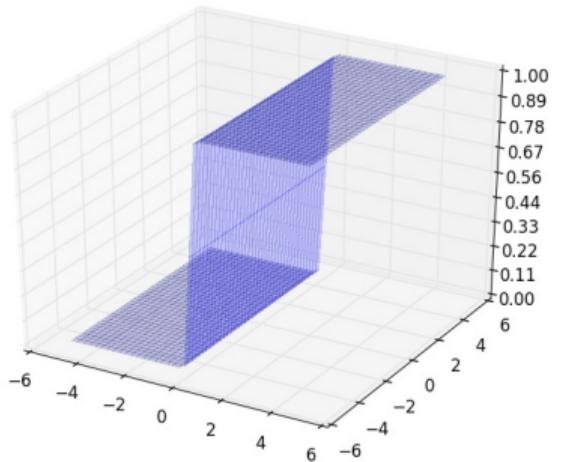
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 16, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

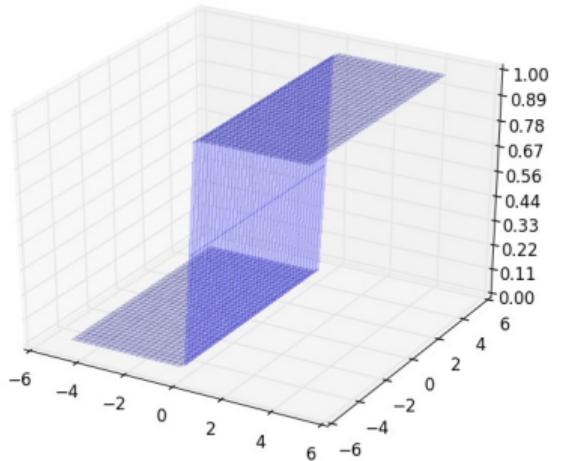
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 17, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

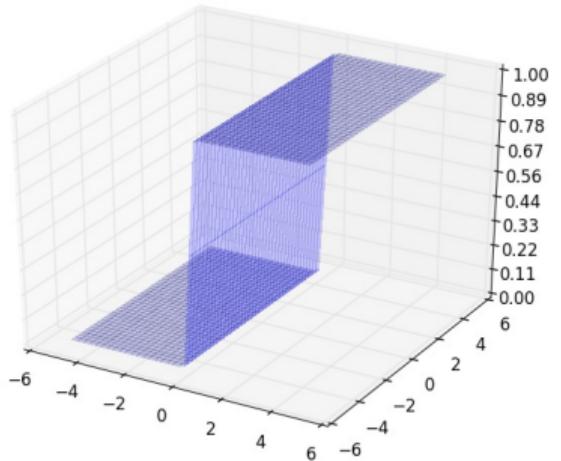
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 18, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

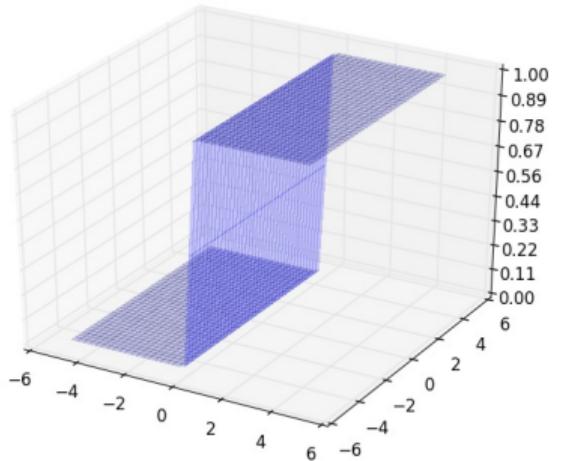
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 19, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

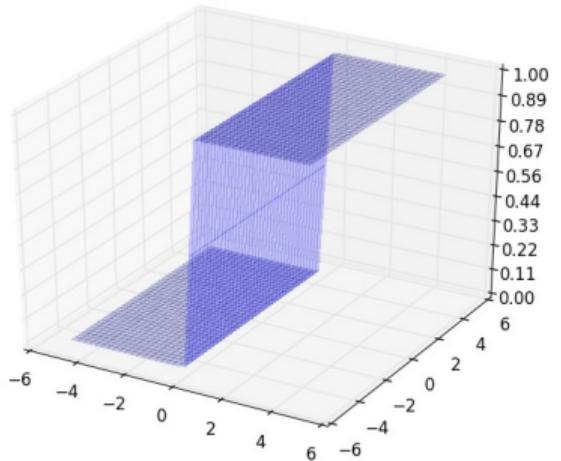
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 20, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

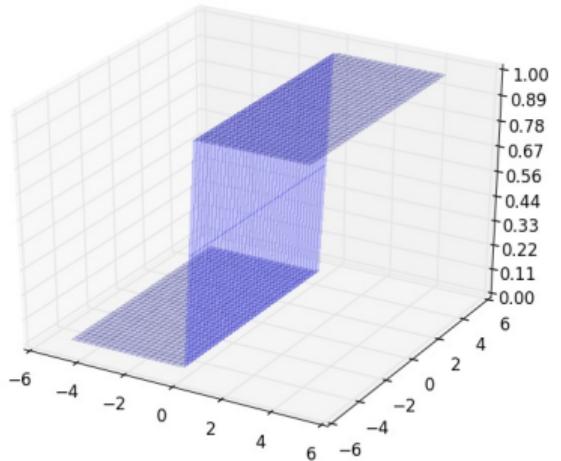
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 21, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

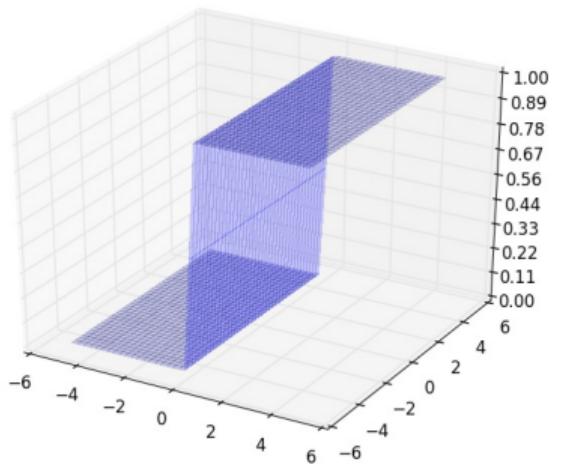
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 22, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

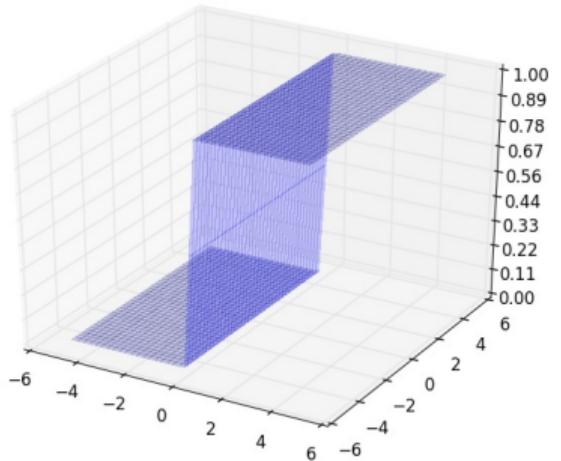
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



$$w_1 = 23, w_2 = 0, b = 0$$

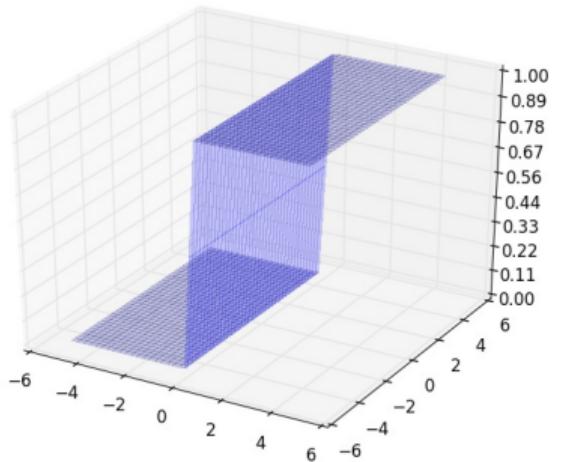
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?



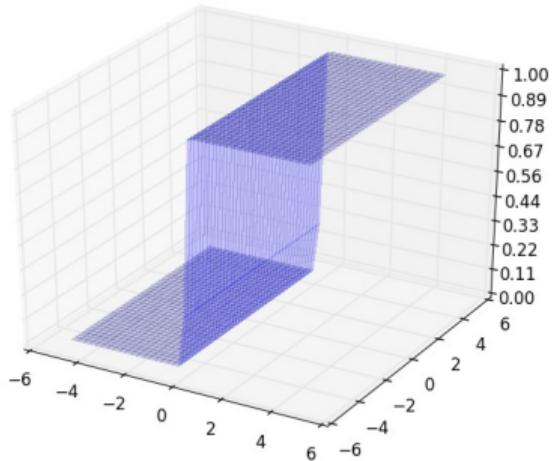
$$w_1 = 24, w_2 = 0, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

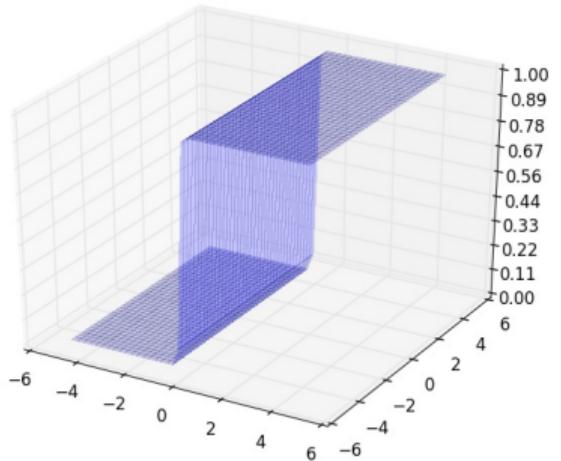


- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

$$w_1 = 25, w_2 = 0, b = 5$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

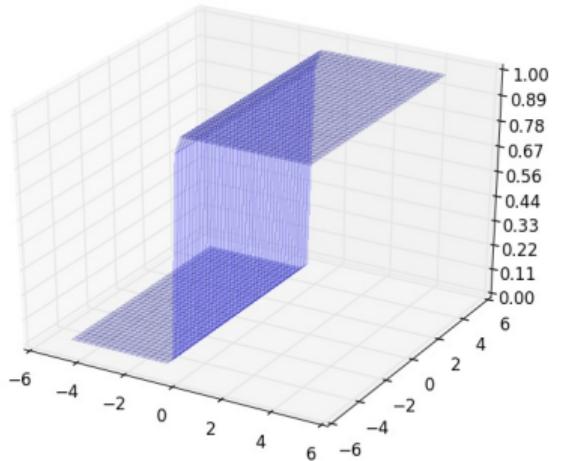
- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数？
- 首先，让 w_2 为 0，调整 w_1 是否能得到一个 2-D 的 step 函数？
- 如果改变 b 又会发生什么？



$$w_1 = 25, w_2 = 0, b = 10$$

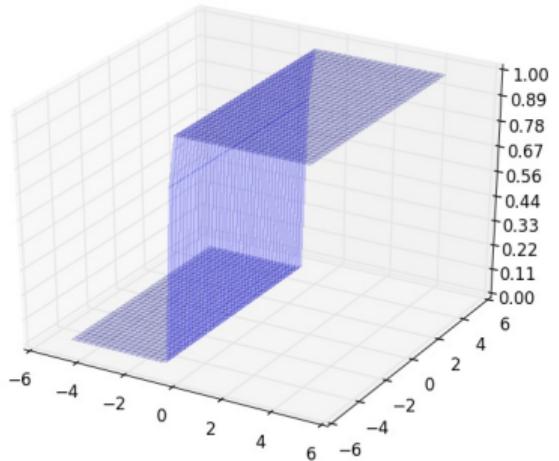
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数？
- 首先，让 w_2 为 0，调整 w_1 是否能得到一个 2-D 的 step 函数？
- 如果改变 b 又会发生什么？



$$w_1 = 25, w_2 = 0, b = 15$$

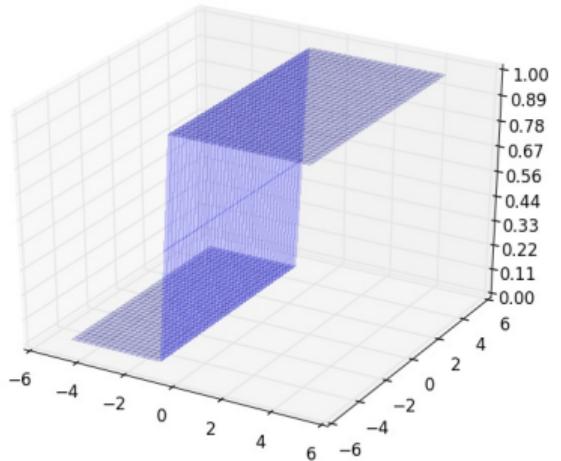
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

$$w_1 = 25, w_2 = 0, b = 20$$

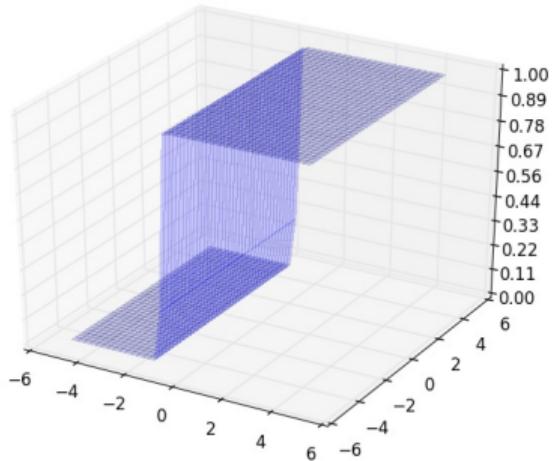
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

$$w_1 = 25, w_2 = 0, b = 25$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

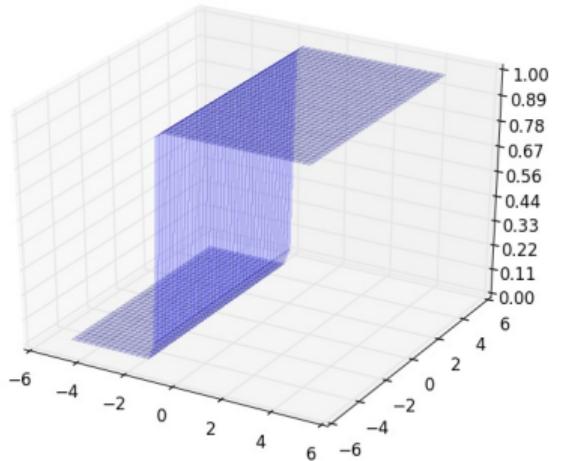


- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

$$w_1 = 25, w_2 = 0, b = 30$$

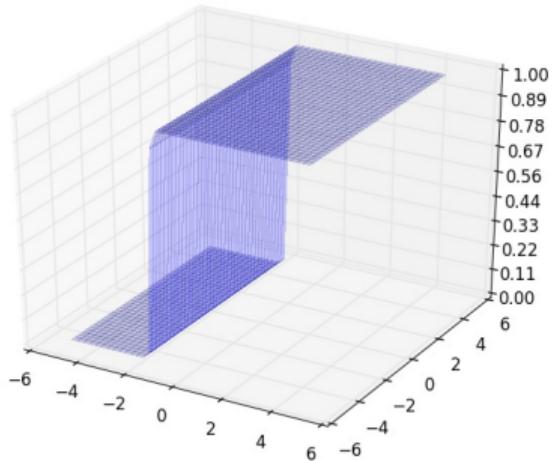
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数？
- 首先，让 w_2 为 0，调整 w_1 是否能得到一个 2-D 的 step 函数？
- 如果改变 b 又会发生什么？



$$w_1 = 25, w_2 = 0, b = 35$$

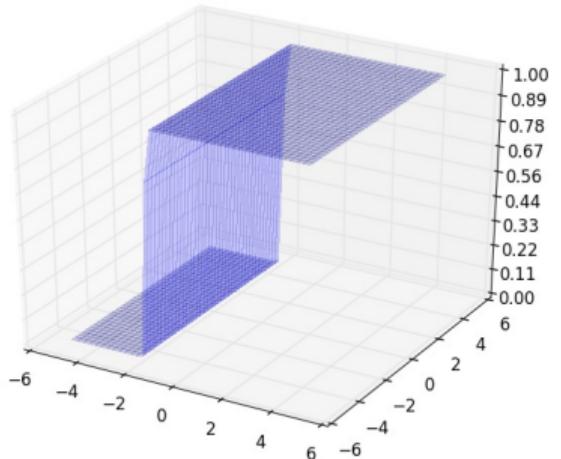
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

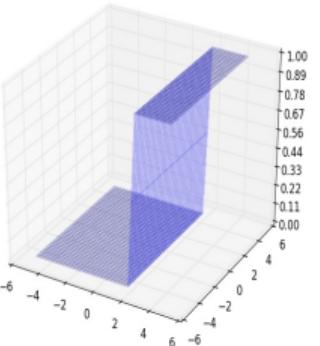
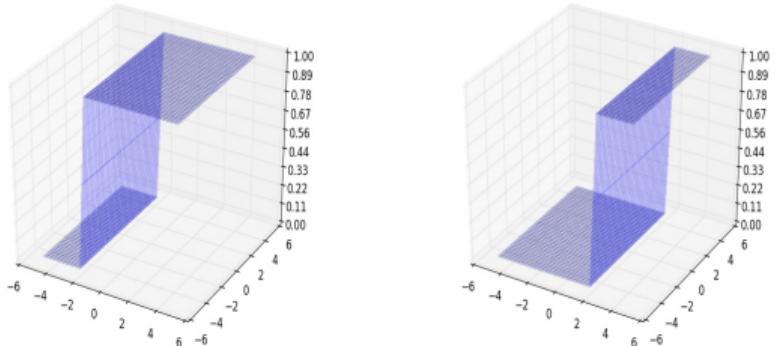
$$w_1 = 25, w_2 = 0, b = 40$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



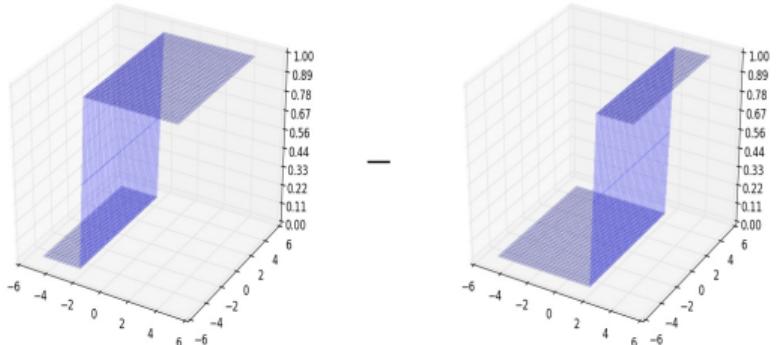
$$w_1 = 25, w_2 = 0, b = 45$$

- 这是一个 2-D sigmoid 函数
- 如何得到一个 tower 函数 ?
- 首先, 让 w_2 为 0 , 调整 w_1 是否能得到一个 2-D 的 step 函数 ?
- 如果改变 b 又会发生什么?

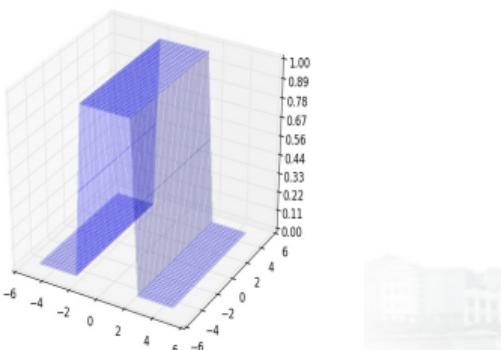
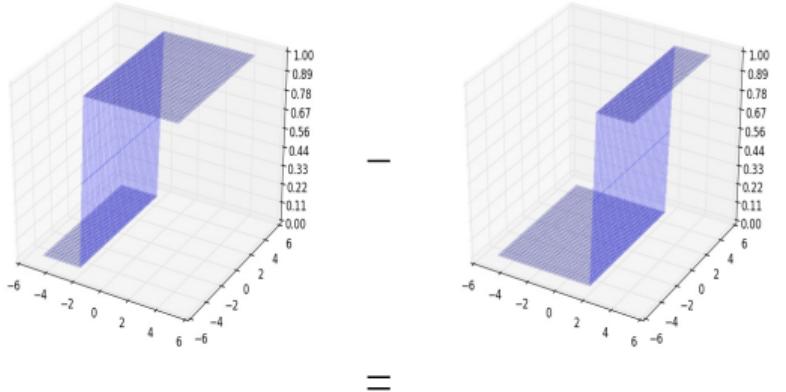


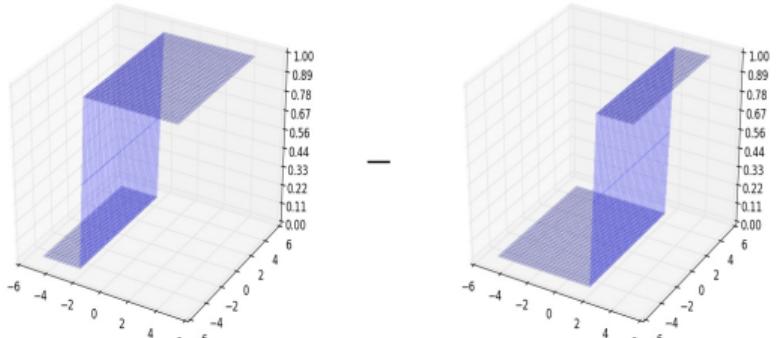
- 如果让两个这样的 Step 函数 (不同的 b) 相减, 会得到什么?

- 如果让两个这样的 Step 函数 (不同的 b) 相减, 会得到什么?

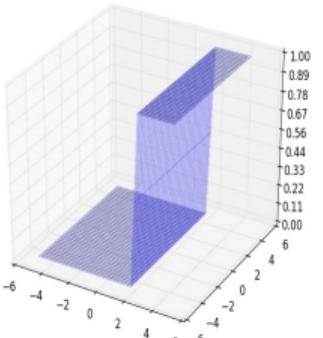


- 如果让两个这样的 Step 函数 (不同的 b) 相减, 会得到什么?

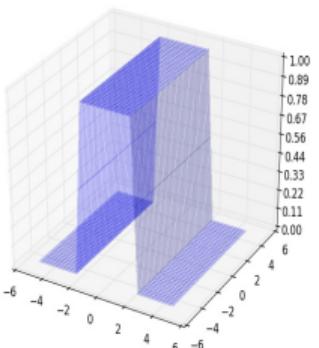




—



—

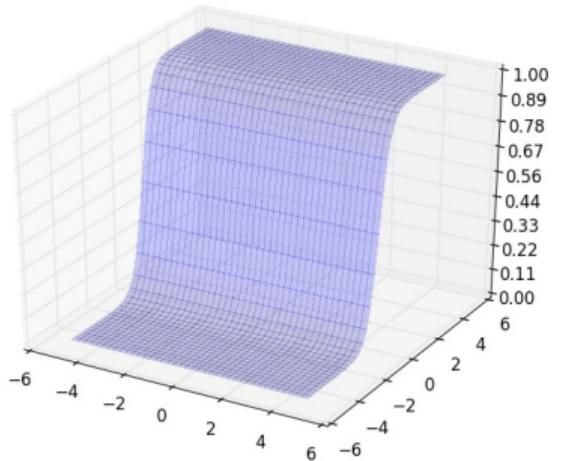


- 如果让两个这样的 Step 函数 (不同的 b) 相减, 会得到什么 ?
- 我们得不到一个 tower 函数 (或者说, 我们只能得到一个缺少两个侧面的 tower 函数)



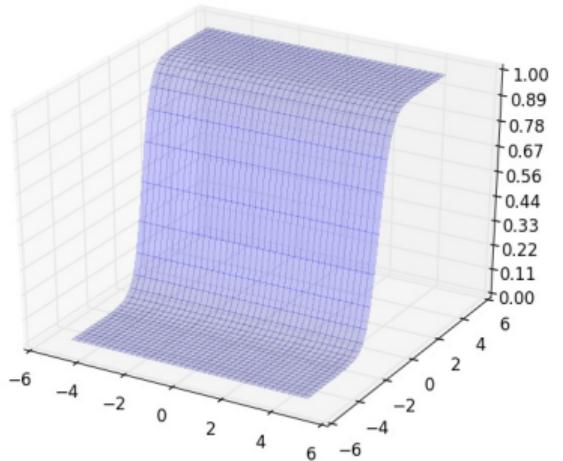
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



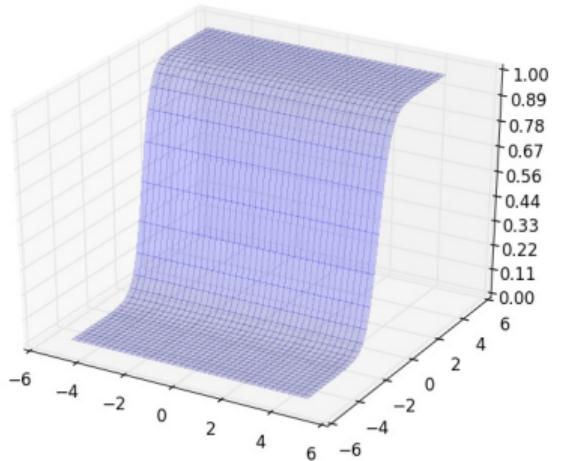
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

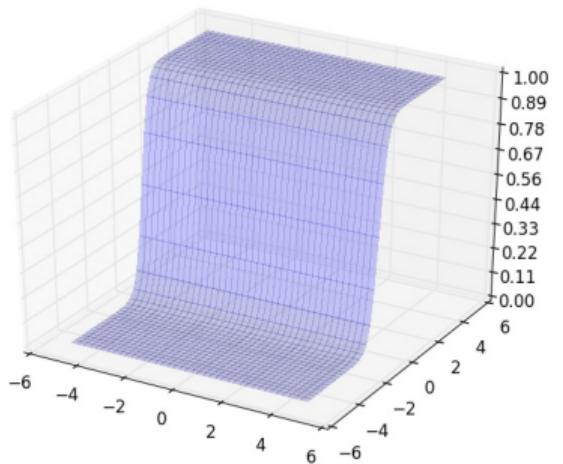
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 2, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

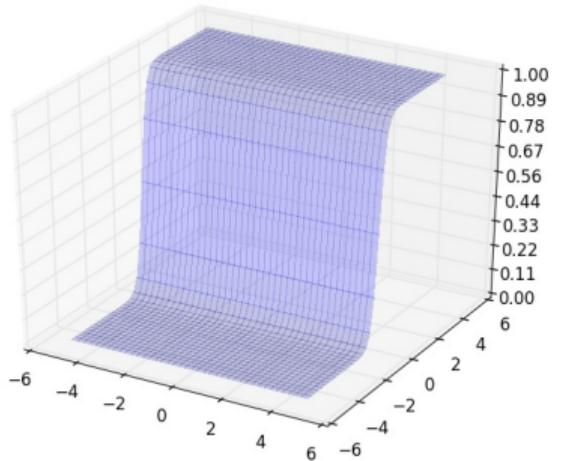
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 3, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

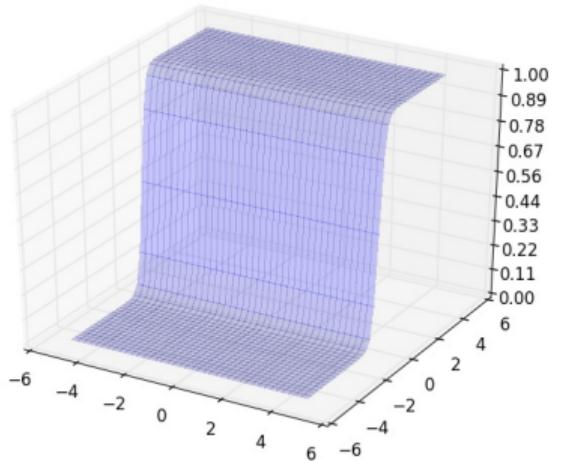
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 4, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

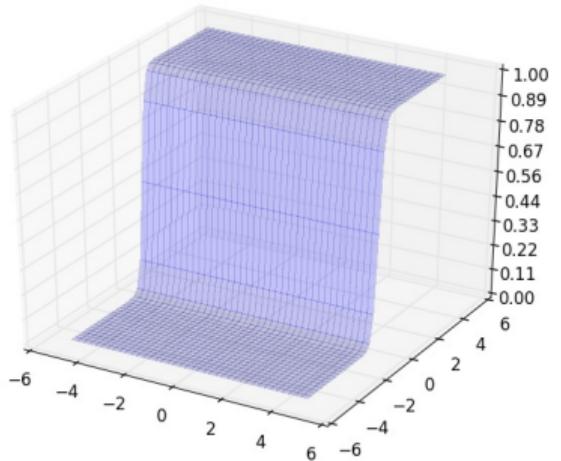
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 5, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

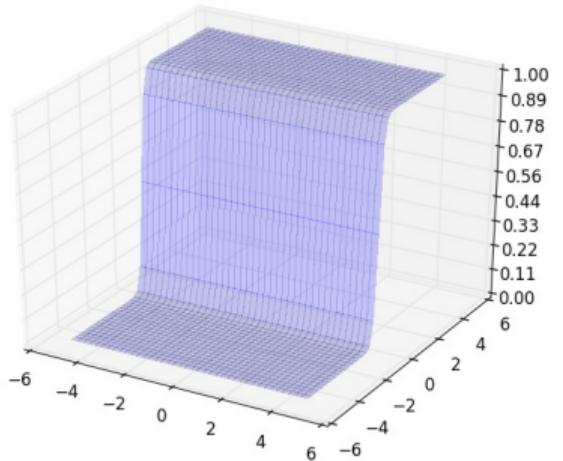
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 6, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

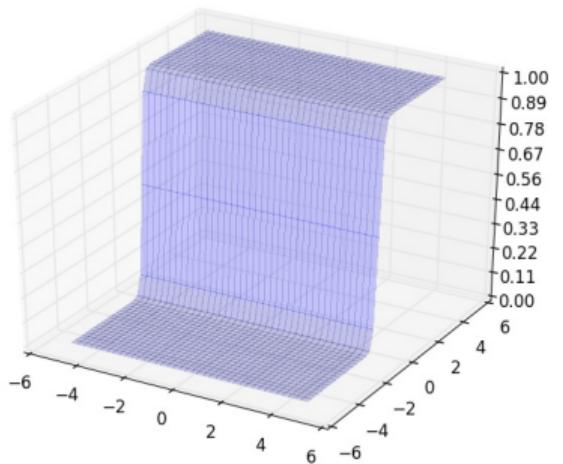
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 7, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

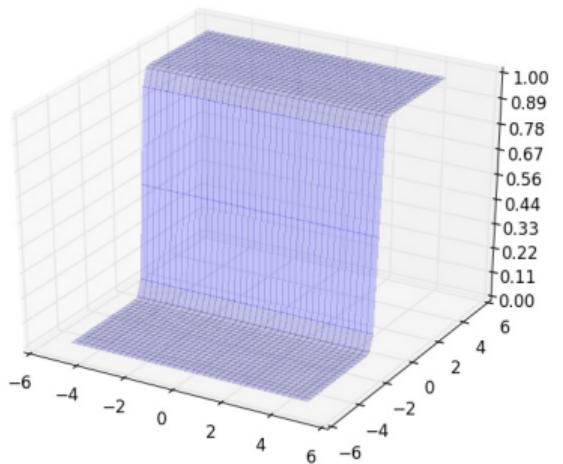
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 8, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

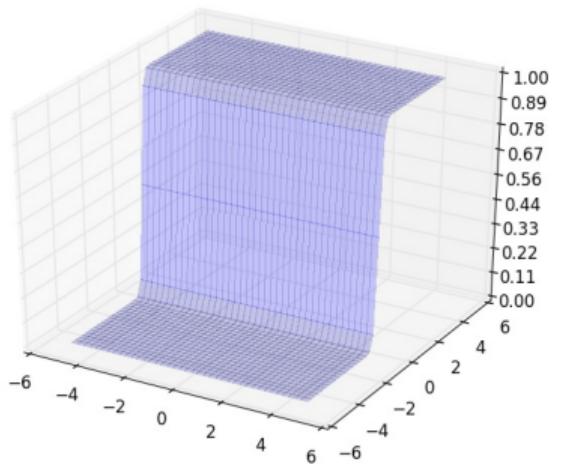
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 9, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

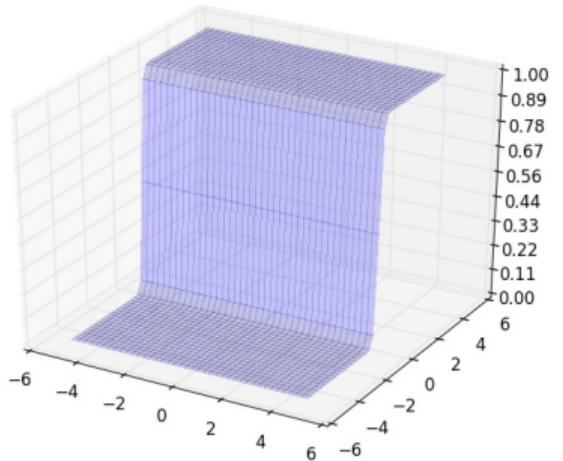
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 10, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

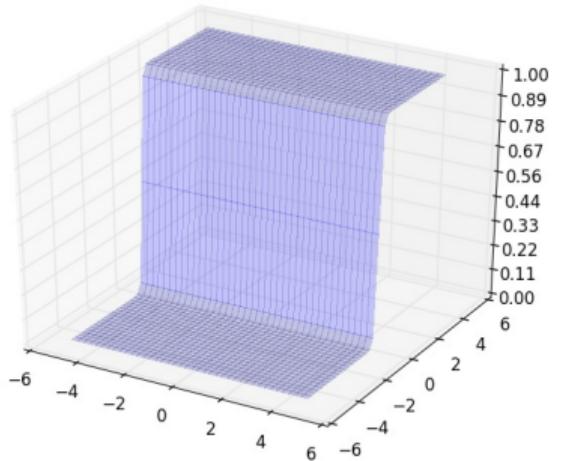
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 11, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

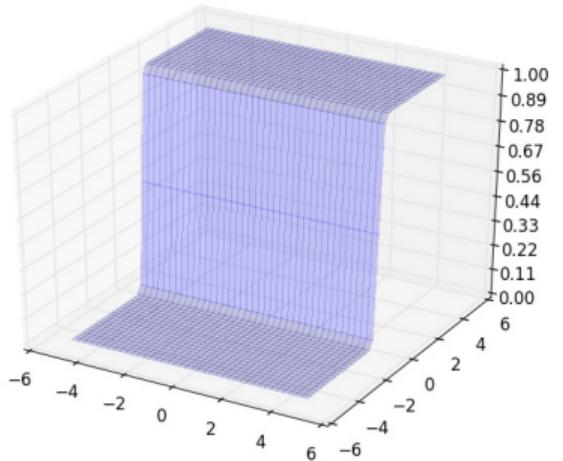
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 12, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

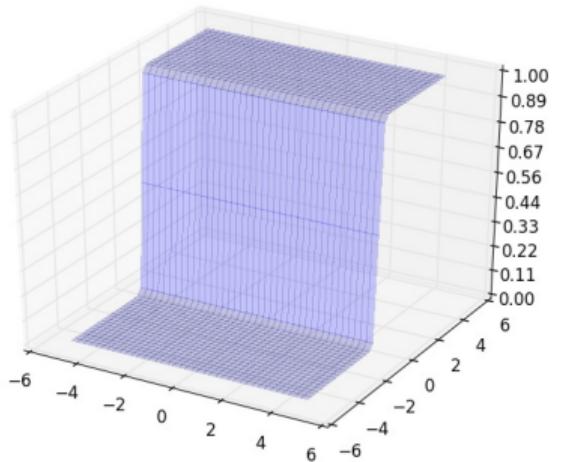
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 13, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

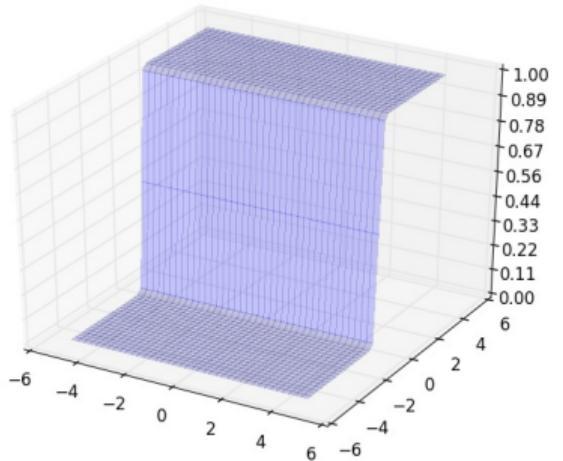
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 14, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

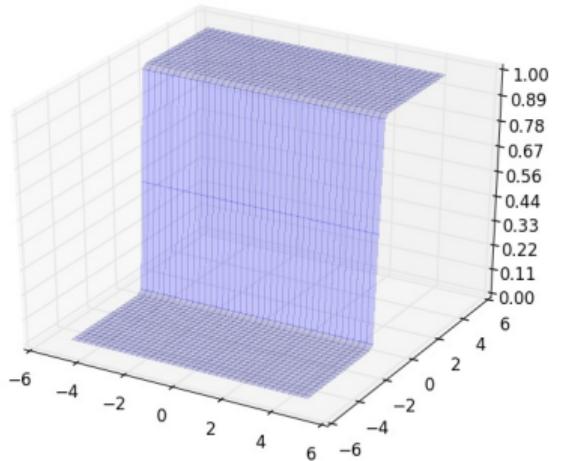
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 15, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

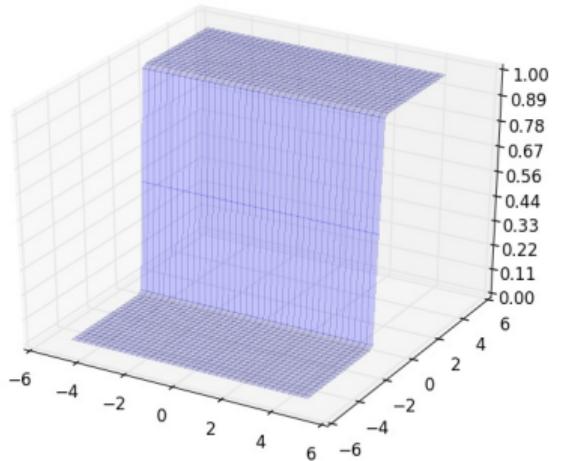
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 16, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)

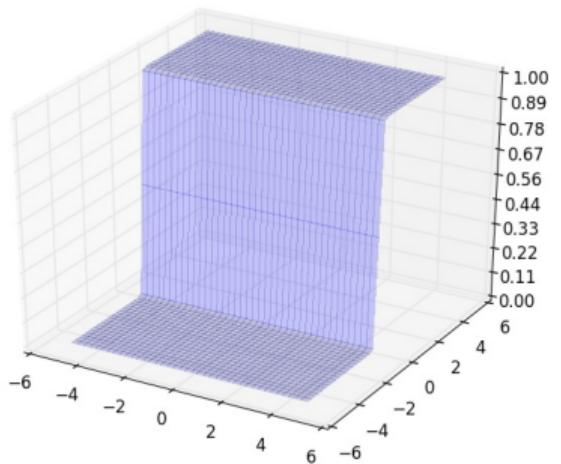


$$w_1 = 0, w_2 = 17, b = 0$$



$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

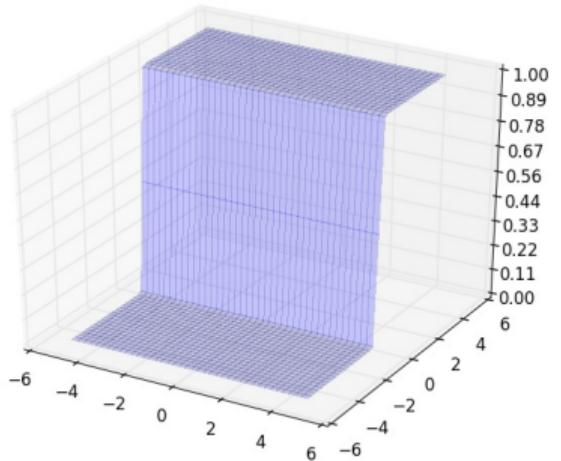
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 18, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

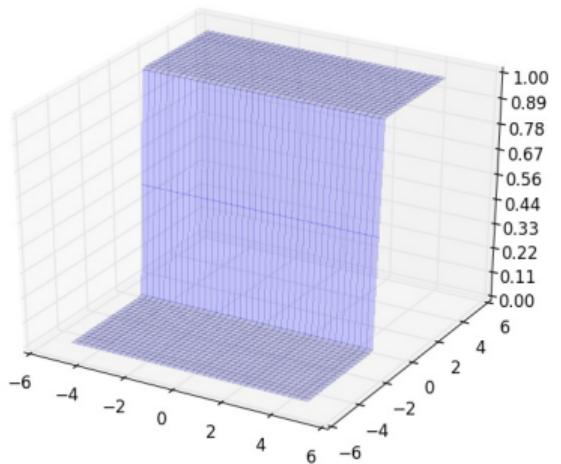
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 19, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

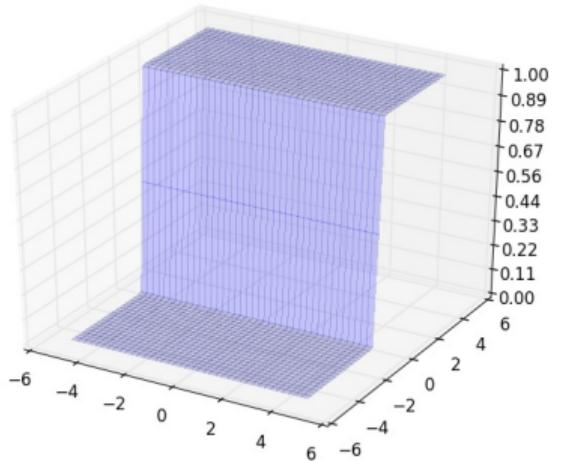
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 20, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

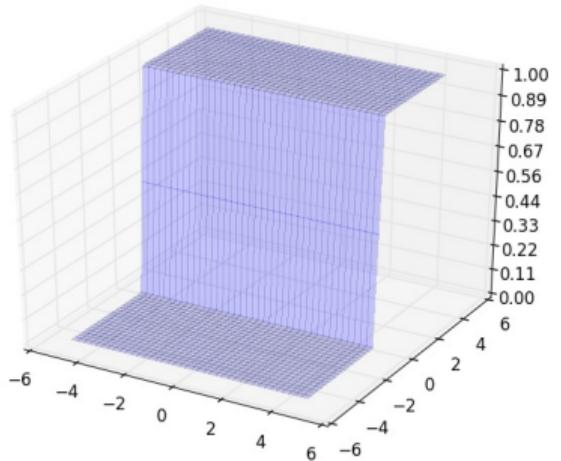
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 21, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

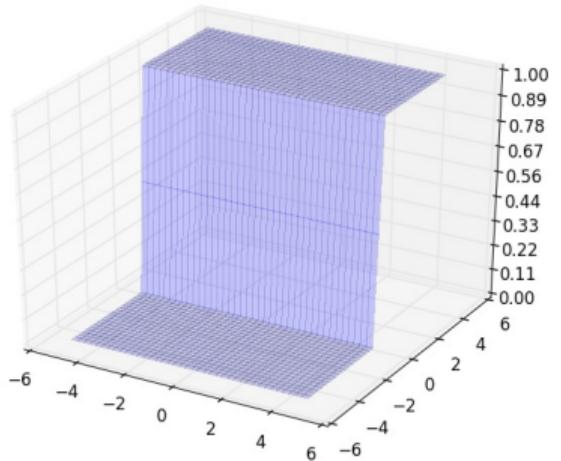
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 22, b = 0$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)

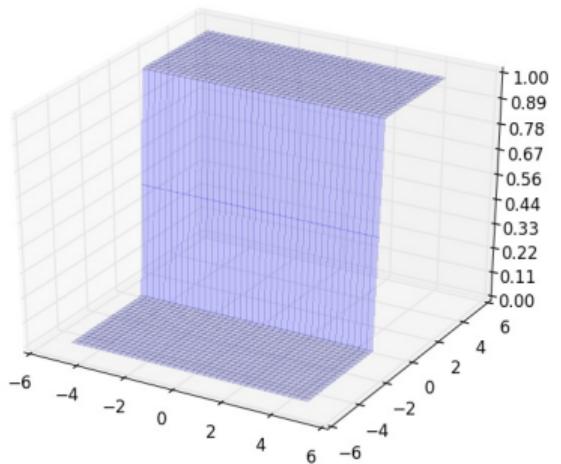


$$w_1 = 0, w_2 = 23, b = 0$$



$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

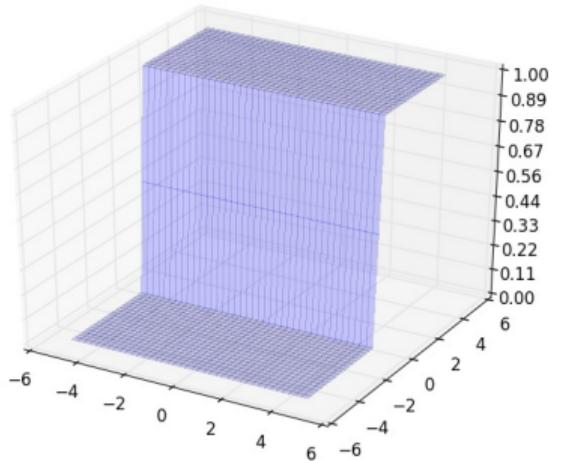
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)



$$w_1 = 0, w_2 = 24, b = 0$$

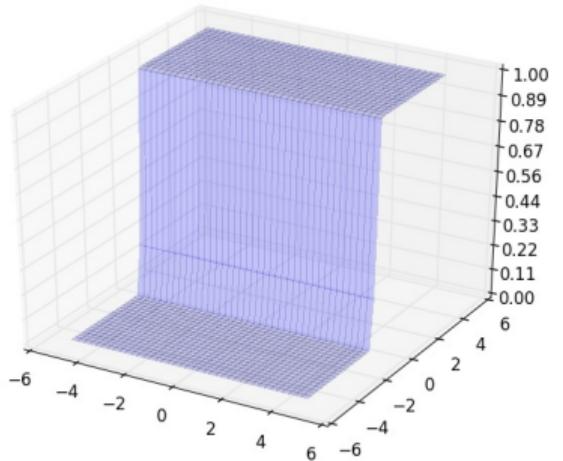
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

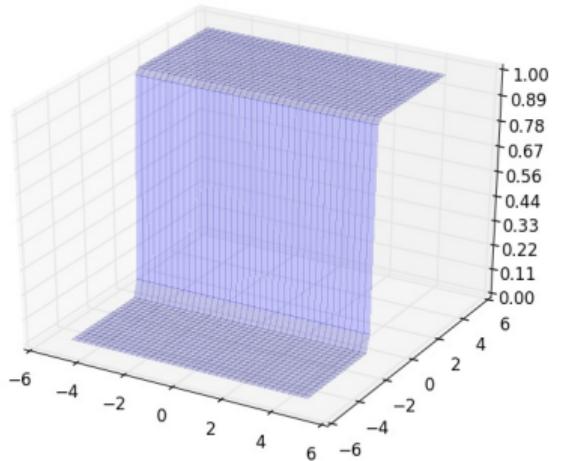
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 5$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

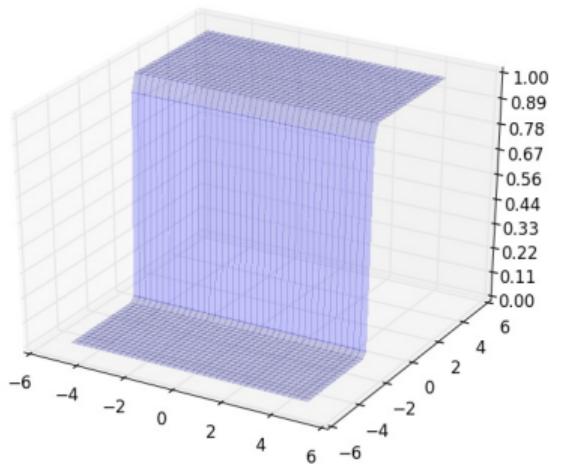
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 10$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

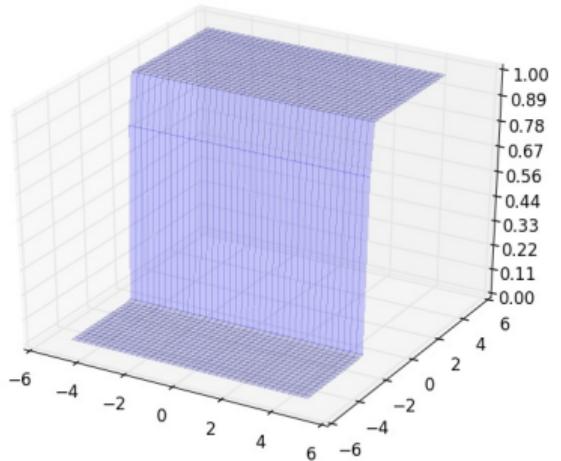
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 15$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

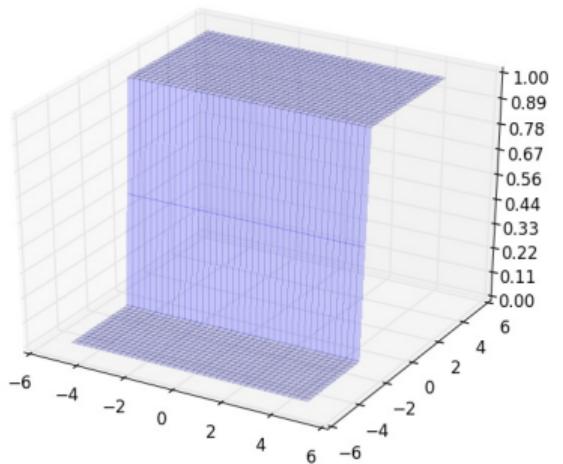
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 20$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

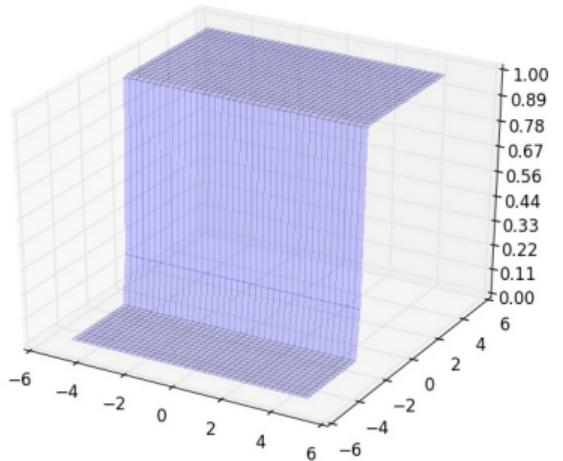
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 25$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

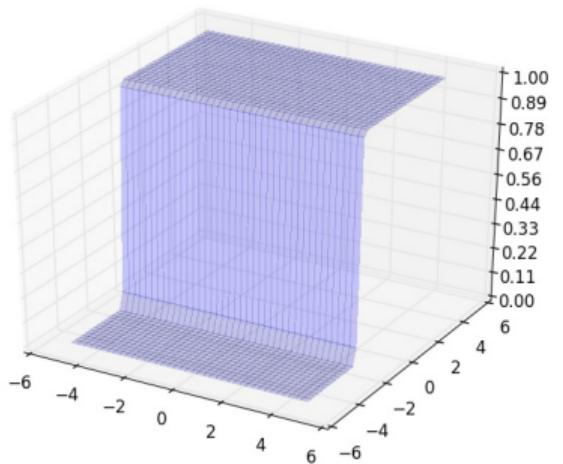
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 30$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

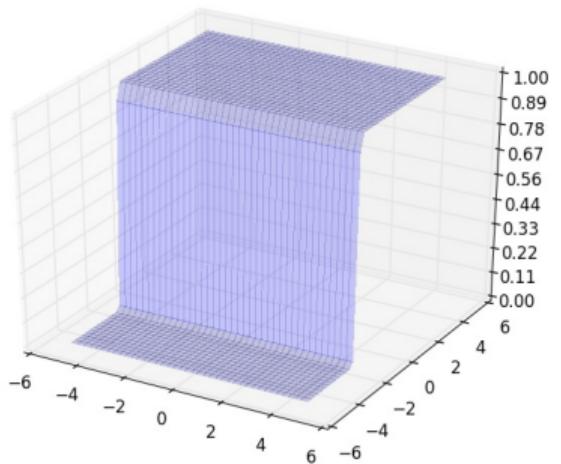
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 35$$

$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

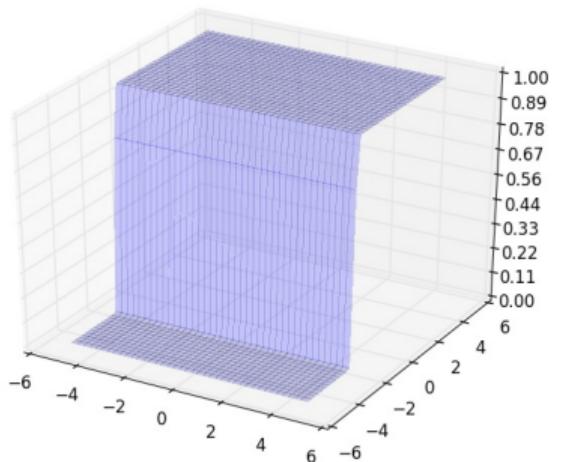
- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值



$$w_1 = 0, w_2 = 25, b = 40$$

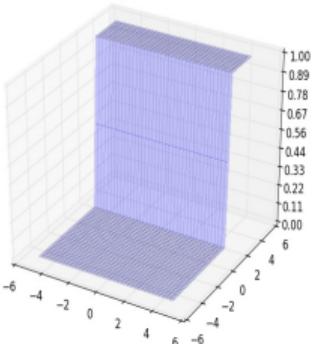
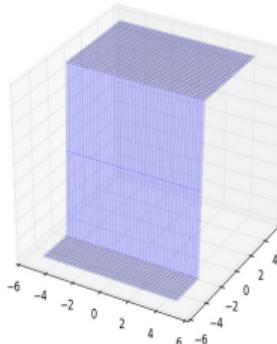
$$y = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$

- 让 w_1 为 0, 调整 w_2 得到一个 2-D 的 step 函数 (不同的方向)
- 改变 b 的值

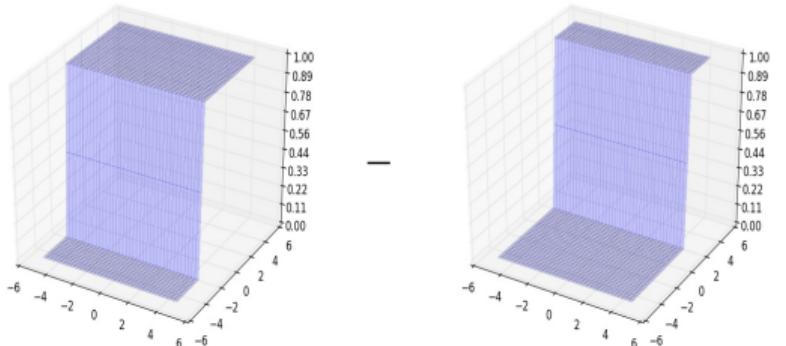


$$w_1 = 0, w_2 = 25, b = 45$$

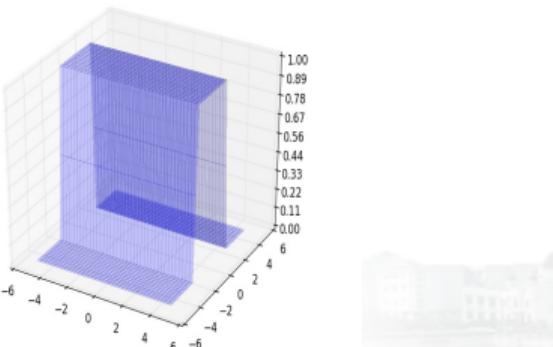
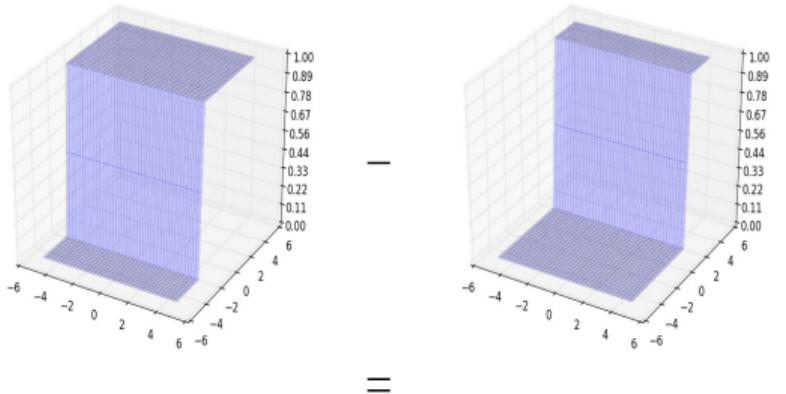
- 又一次, 如果让两个这样的 Step 函数(不同的 b) 相减, 会得到什么?

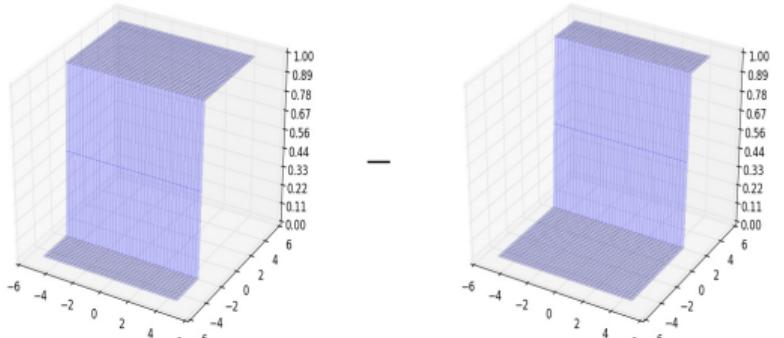


- 又一次, 如果让两个这样的 Step 函数(不同的 b) 相减, 会得到什么?



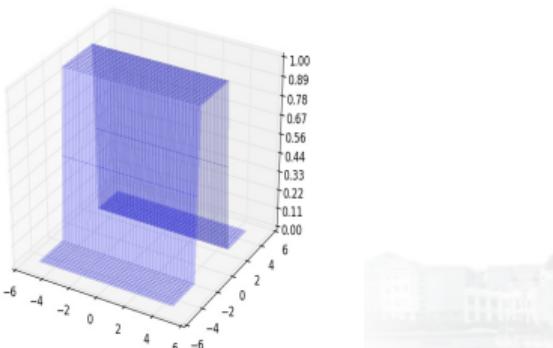
- 又一次, 如果让两个这样的 Step 函数(不同的 b) 相减, 会得到什么?

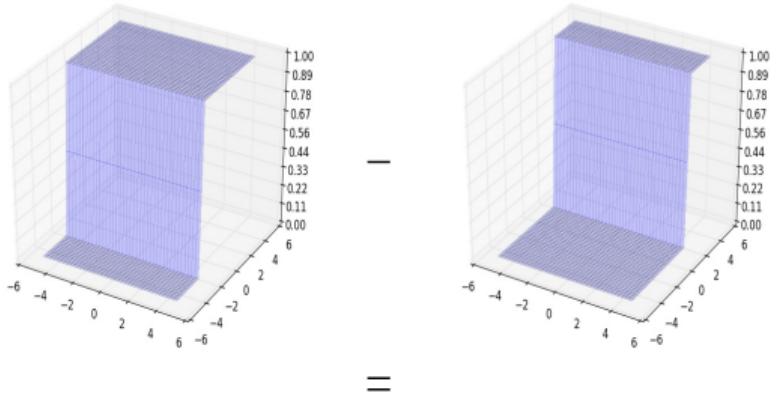




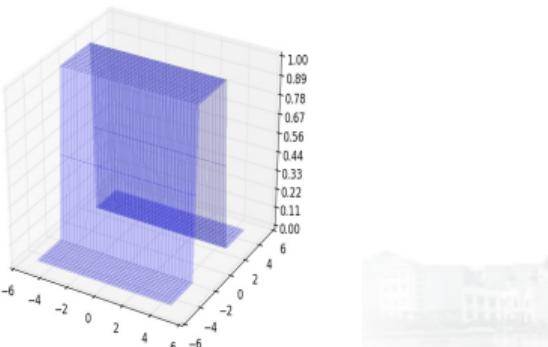
—

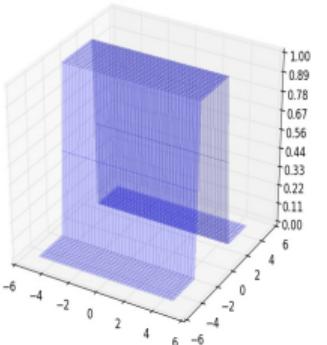
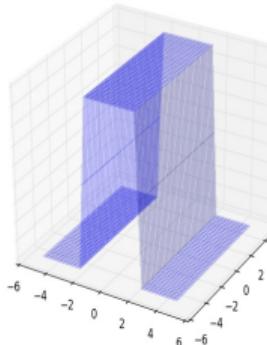
- 又一次, 如果让两个这样的 Step 函数 (不同的 b) 相减, 会得到什么?
- 我们仍然得不到一个 tower 函数 (或者说, 我们只能得到一个缺少两个侧面的 tower 函数)



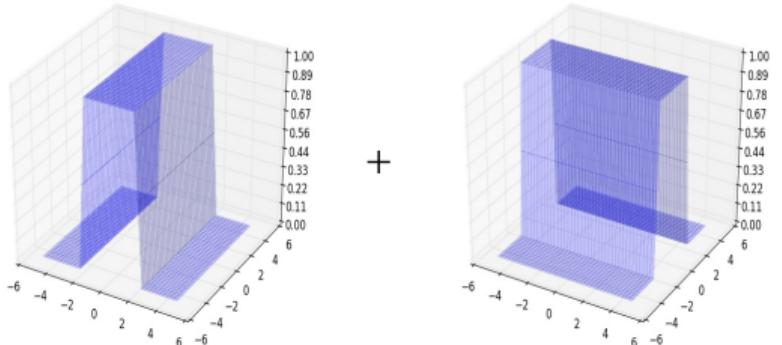


- 又一次, 如果让两个这样的 Step 函数 (不同的 b) 相减, 会得到什么?
- 我们仍然得不到一个 tower 函数 (或者说, 我们只能得到一个缺少两个侧面的 tower 函数)
- 注意到, 这个不完整的 tower 函数和之前得到的那个不完整的 tower 函数的方向不同

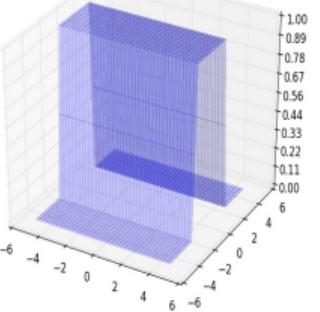




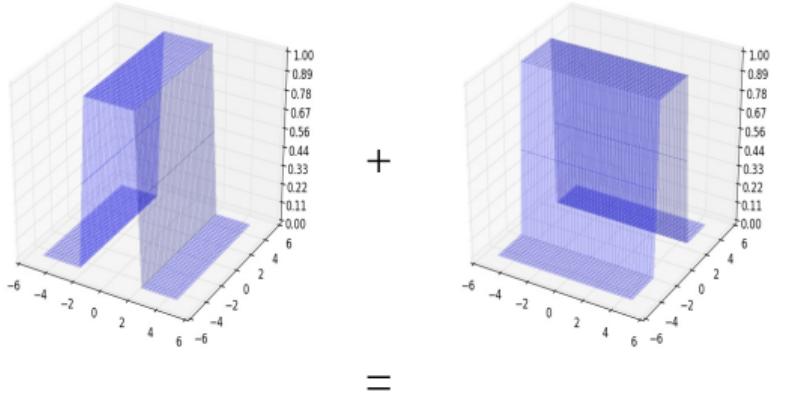
- 如果将这两个不完整的 tower 函数相加呢？



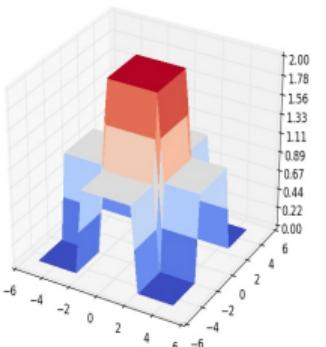
- 如果将这两个不完整的 tower 函数相加呢？

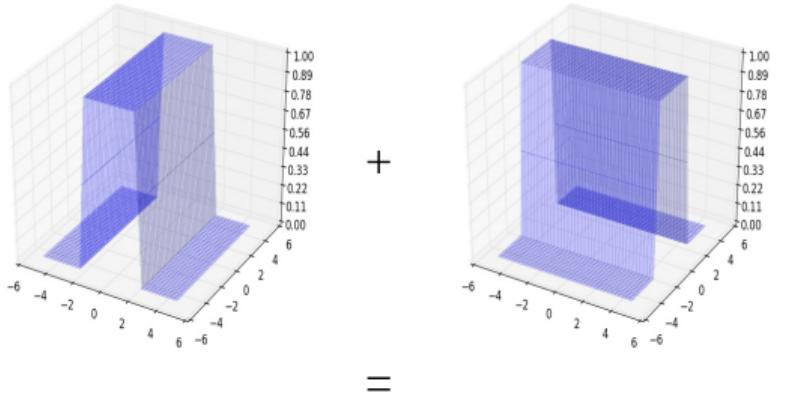


- 如果将这两个不完整的 tower 函数相加呢？

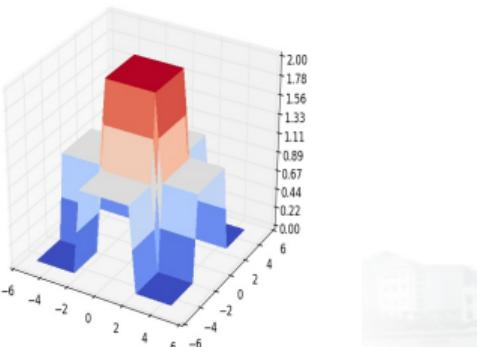


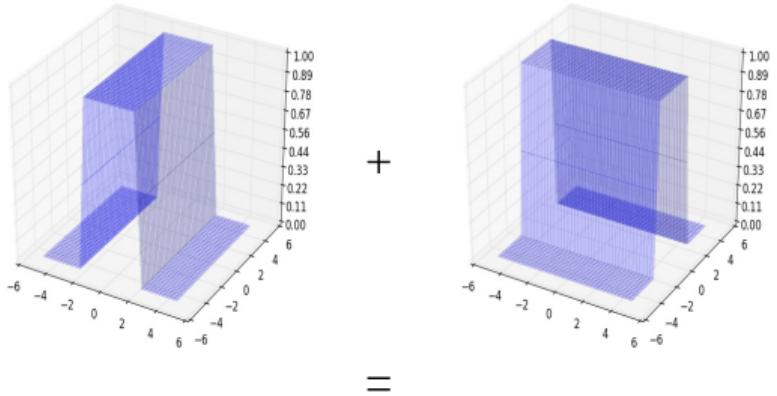
=



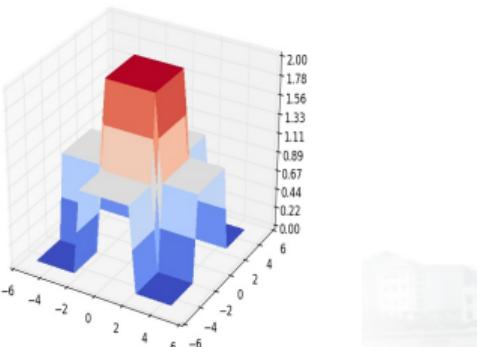


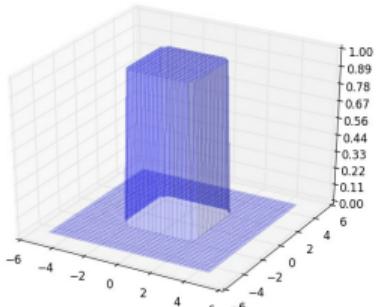
- 如果将这两个不完整的 tower 函数相加呢？
- 就能得到一个在高空架起来的的 tower 函数了



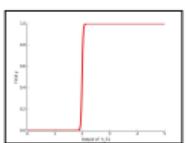


- 如果将这两个不完整的 tower 函数相加呢？
- 就能得到一个在高空架起来的的 tower 函数了
- 把这个函数传给另一个 sigmoid 神经元就能得到一个理想的 tower 函数了

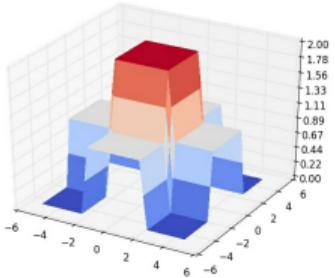


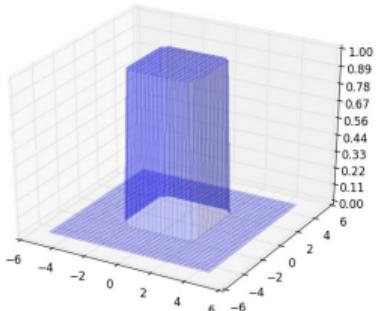


h_{31} is passed through a sigmoid function with the following characteristics

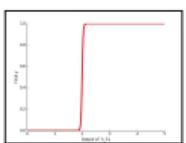


- 如果将这两个不完整的 tower 函数相加呢？
- 就能得到一个在高空架起来的的 tower 函数了
- 把这个函数传给另一个 sigmoid 神经元就能得到一个理想的 tower 函数了

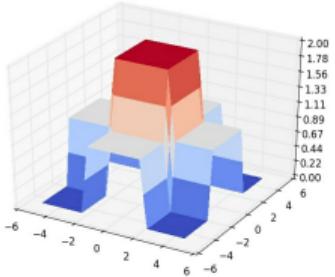




h_{31} is passed through a sigmoid function with the following characteristics

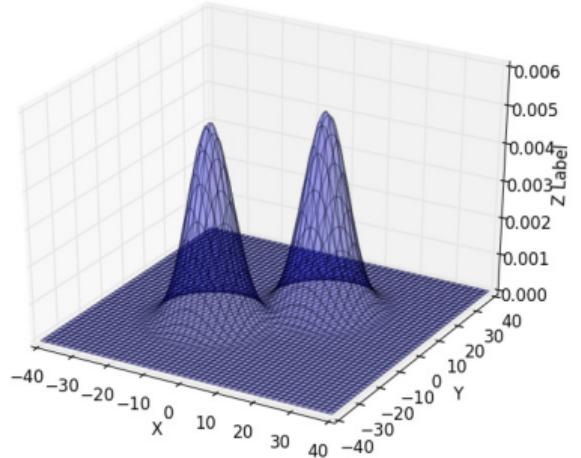


- 如果将这两个不完整的 tower 函数相加呢？
- 就能得到一个在高空架起来的的 tower 函数了
- 把这个函数传给另一个 sigmoid 神经元就能得到一个理想的 tower 函数了
- 现在，可以使用这些 tower 函数近似任意函数了



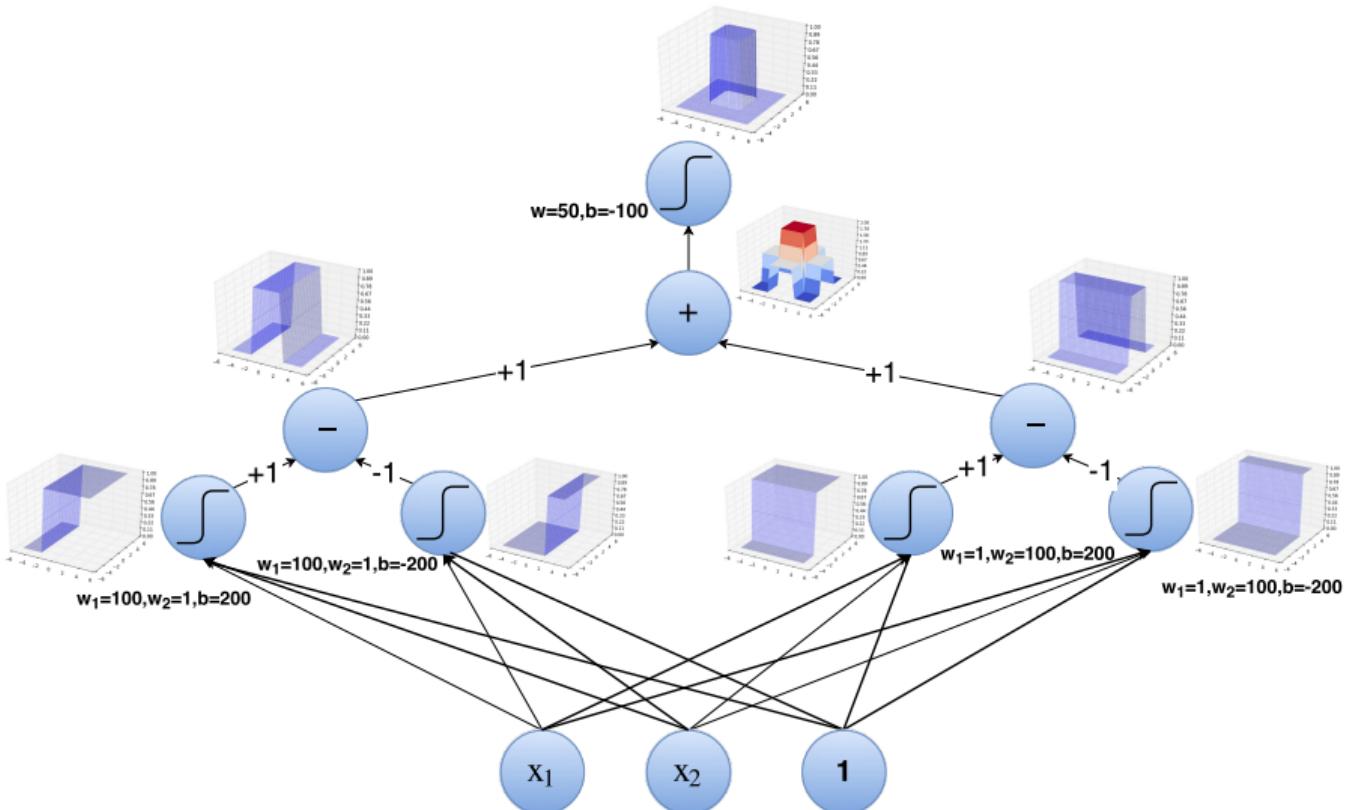


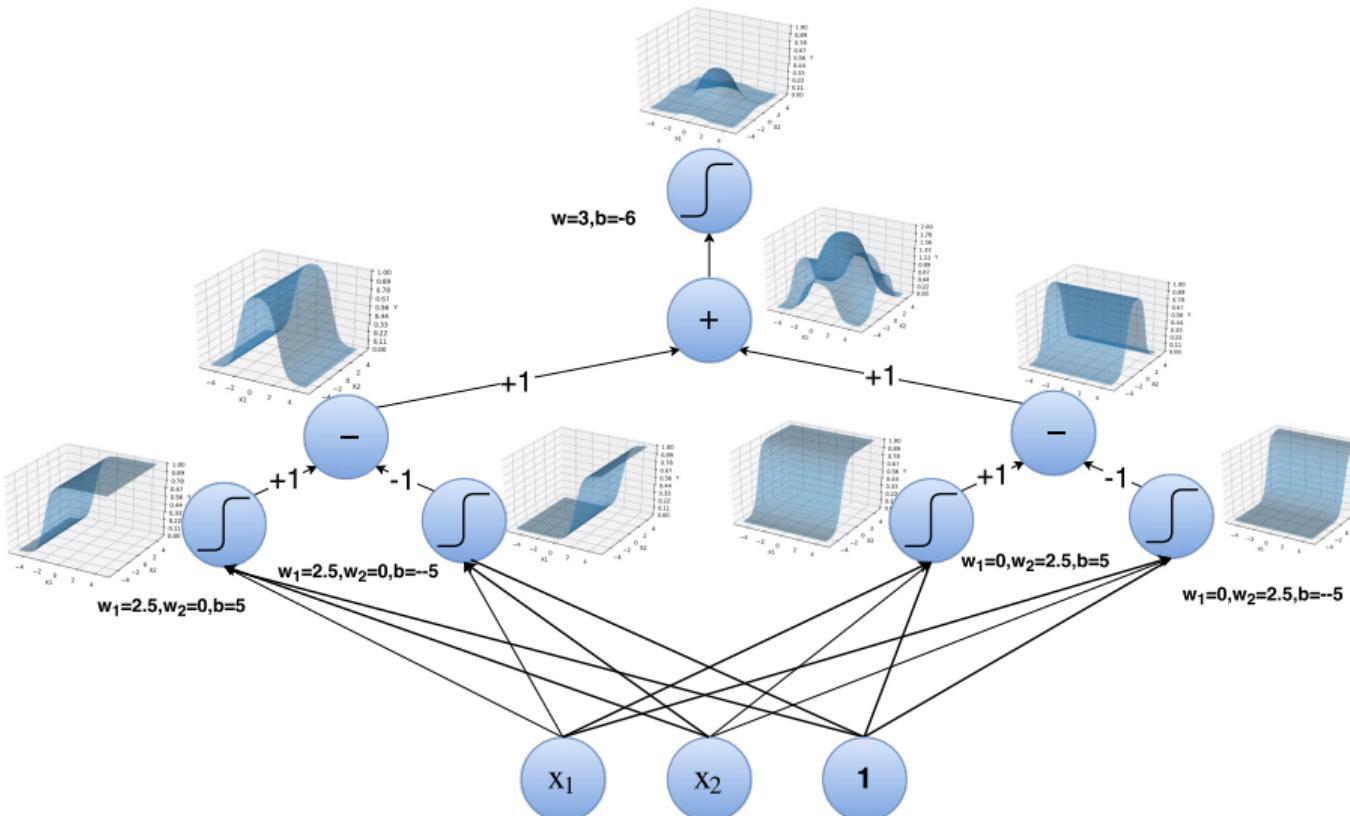
- 例如，可以使用一组 tower 函数来近似下面的这个函数





- 如何设计一个神经网络来表示构建一个 3-D tower 函数的整个过程 ?





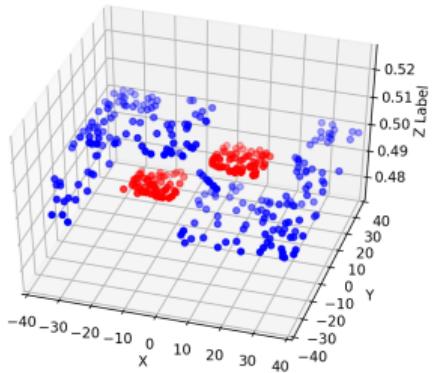
思考

- 对于 1-D 的输入，需要 2 个神经元来构建一个 tower 函数
- 对于 2-D 的输入，需要 4 个神经元来构建一个 tower 函数
- 对于 n -D 的输入，需要多少个神经元？

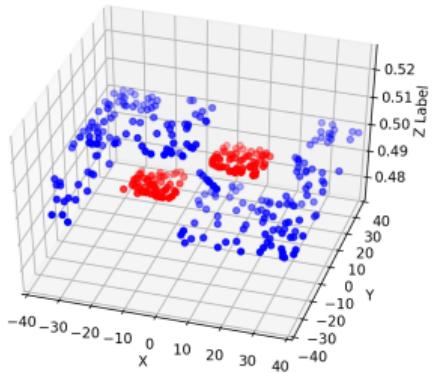


回顾

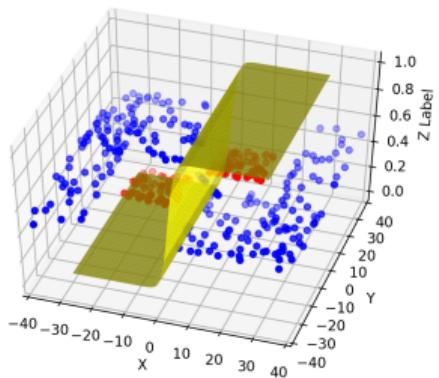
- 我们为什么关心近似任意的函数？
- 我们能否将所有这些与分类问题联系起来？



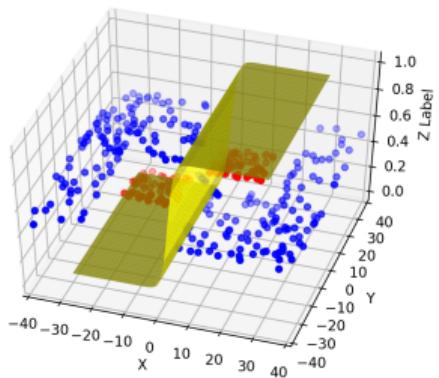
- 将蓝色的点和红色的点分开



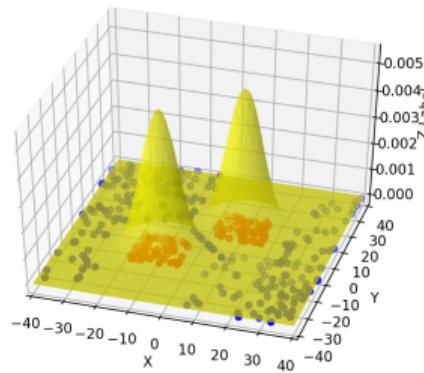
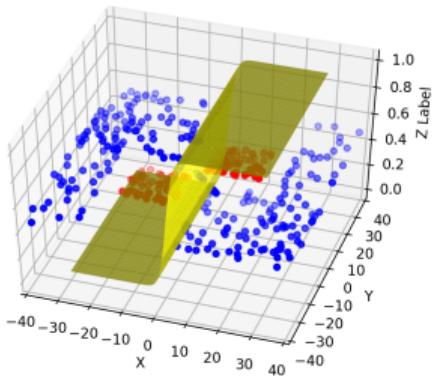
- 将蓝色的点和红色的点分开
- 如果使用一个 sigmoid 神经元来近似 $x = [x_1, x_2]$ 和 y 的关系



- 将蓝色的点和红色的点分开
- 如果使用一个 sigmoid 神经元来近似 $x = [x_1, x_2]$ 和 y 的关系

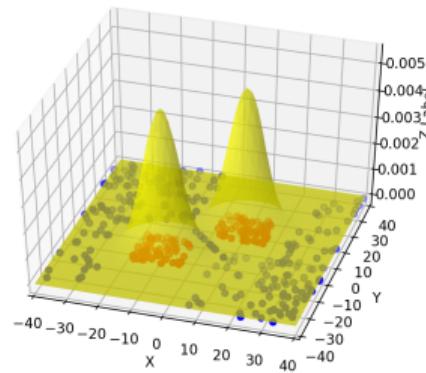
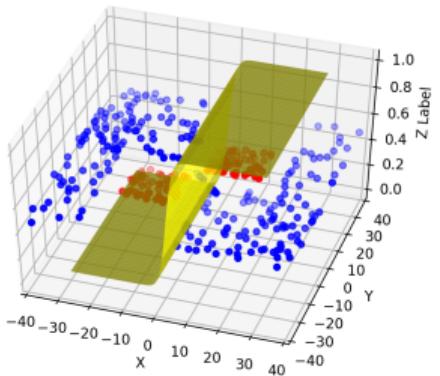


- 将蓝色的点和红色的点分开
- 如果使用一个 sigmoid 神经元来近似 $x = [x_1, x_2]$ 和 y 的关系
- 显然，会有错误（一些蓝色的点被分成红色的，一些红色的点被分成蓝色的）



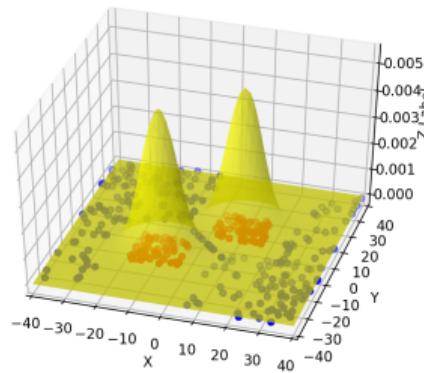
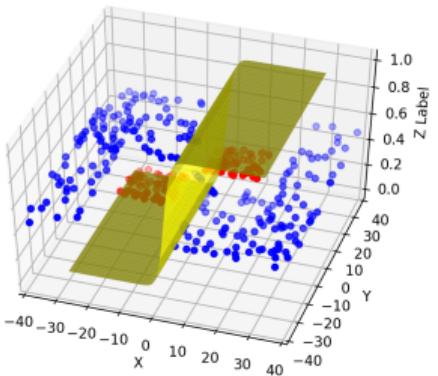
- 将蓝色的点和红色的点分开
- 如果使用一个 sigmoid 神经元来近似 $x = [x_1, x_2]$ 和 y 的关系
- 显然，会有错误（一些蓝色的点被分成红色的，一些红色的点被分成蓝色的）

■ 这才是我们想要的



- 将蓝色的点和红色的点分开
- 如果使用一个 sigmoid 神经元来近似 $x = [x_1, x_2]$ 和 y 的关系
- 显然，会有错误（一些蓝色的点被分成红色的，一些红色的点被分成蓝色的）

- 这才是我们想要的
- 一个具有两个隐含层的神经网络通过将多个 tower 函数相加，能够近似这个函数



- 将蓝色的点和红色的点分开
- 如果使用一个 sigmoid 神经元来近似 $x = [x_1, x_2]$ 和 y 的关系
- 显然，会有错误（一些蓝色的点被分成红色的，一些红色的点被分成蓝色的）

- 这才是我们想要的
- 一个具有两个隐含层的神经网络通过将多个 tower 函数相加，能够近似这个函数
- 这意味着，这样的神经网络能够精确地将红色的点和蓝色的点分开