
COGS 181, Fall 2017

Neural Networks and Deep Learning

Lecture 5: Logistic Regression

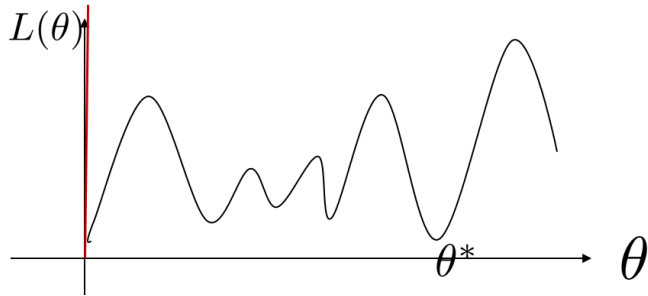
Three key things you are learning in this class:

Representation: With better and better understanding of the underlining statistics about the data and methods.

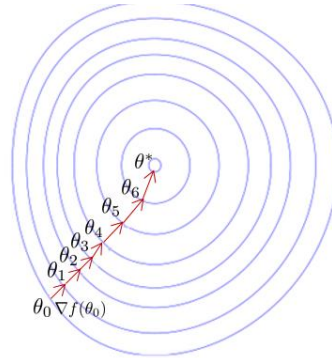
Evaluation: The ideal strategy is always to aim at your target directly (take non-stop flight as opposed to having multiple stops).

Optimization: Based on the chosen representation and evaluation, you pick a strategy (mathematical/statistical) to achieve your goal.

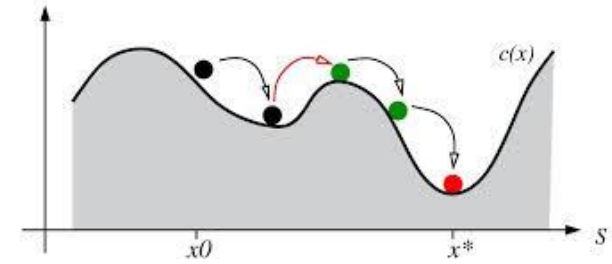
Optimization



exhaustive search



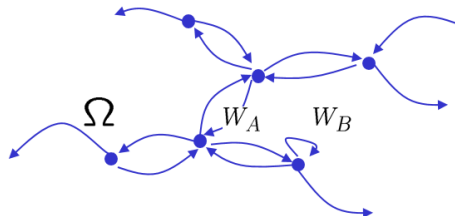
gradient descent



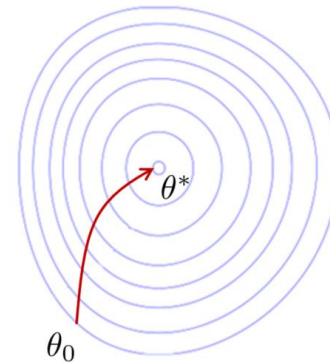
simulated annealing

Markov Chain: $MC = (\Omega, K, \nu)$

To design transition kernel: $p \bullet K = p$



Markov chain Monte Carlo



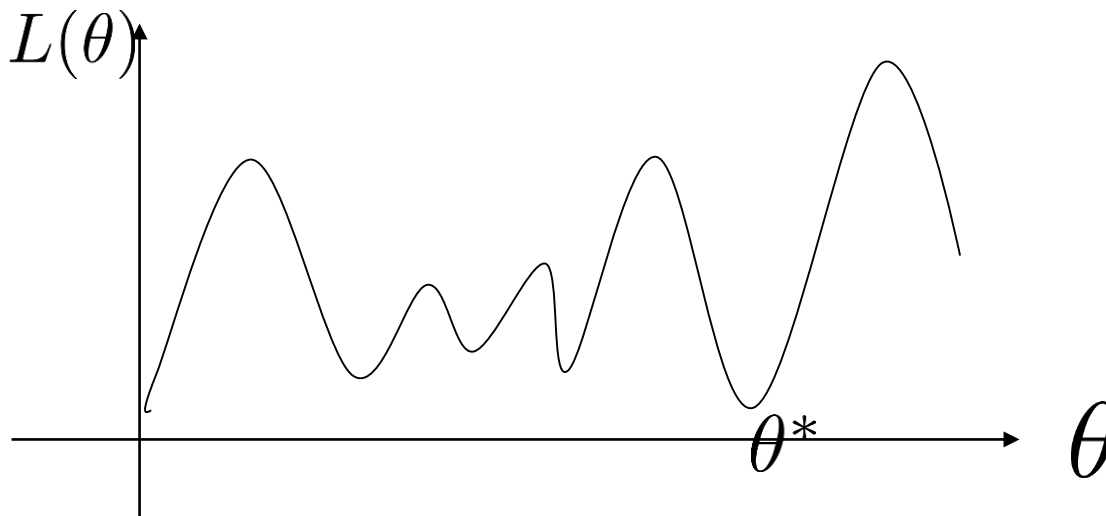
Newton's method

Optimization: argmin

$$\theta^* = \arg \min_{\theta} L(\theta)$$

The operator $\arg \min$ defines the optimal value (in the argument of function $L()$) θ^* that minimizes $L(\theta)$

$\arg \min L(\theta)$ doesn't return the value of $L(\theta)$

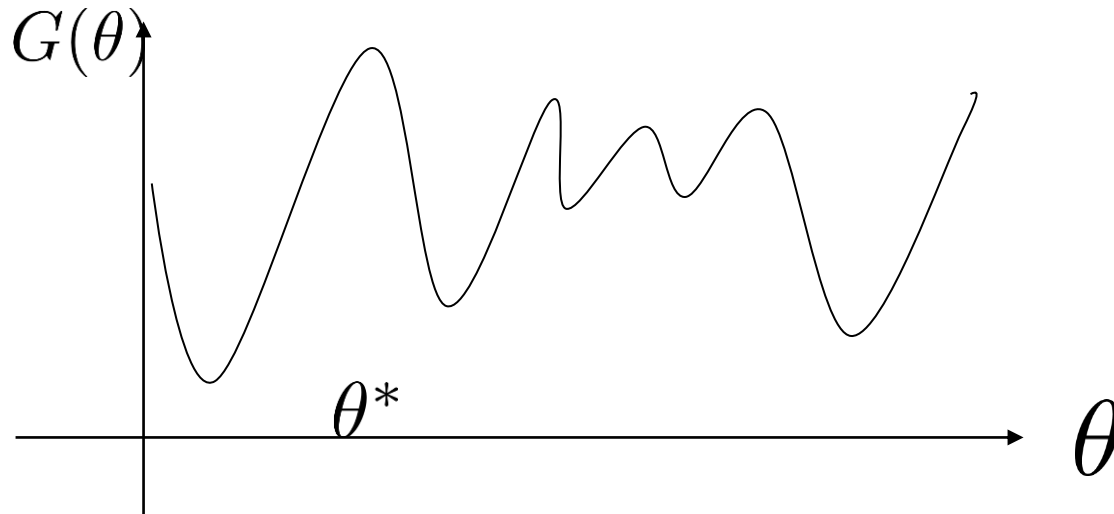


Optimization: argmax

$$\theta^* = \arg \max_{\theta} G(\theta)$$

The operator $\arg \max$ defines the optimal value (in the argument of function $G()$) θ^* that maximizes $G(\theta)$

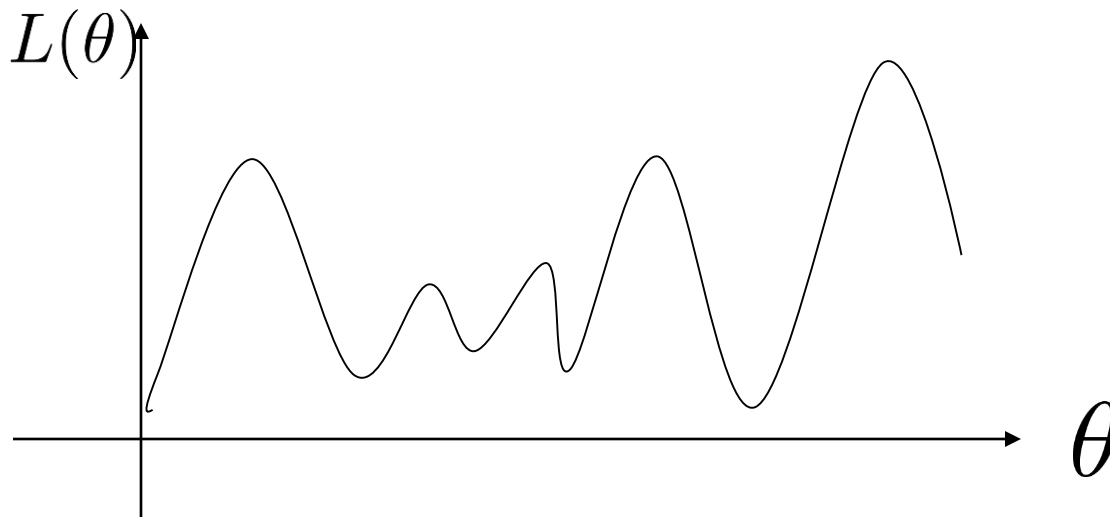
$\arg \max G(\theta)$ doesn't return the value of $G(\theta)$



Learning

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad \mathbf{x}_i = (x_{i1}, \dots, x_{im}) \quad y \in \{-1, +1\}$$

$$\theta^* = \arg \min_{\theta} L(\theta)$$



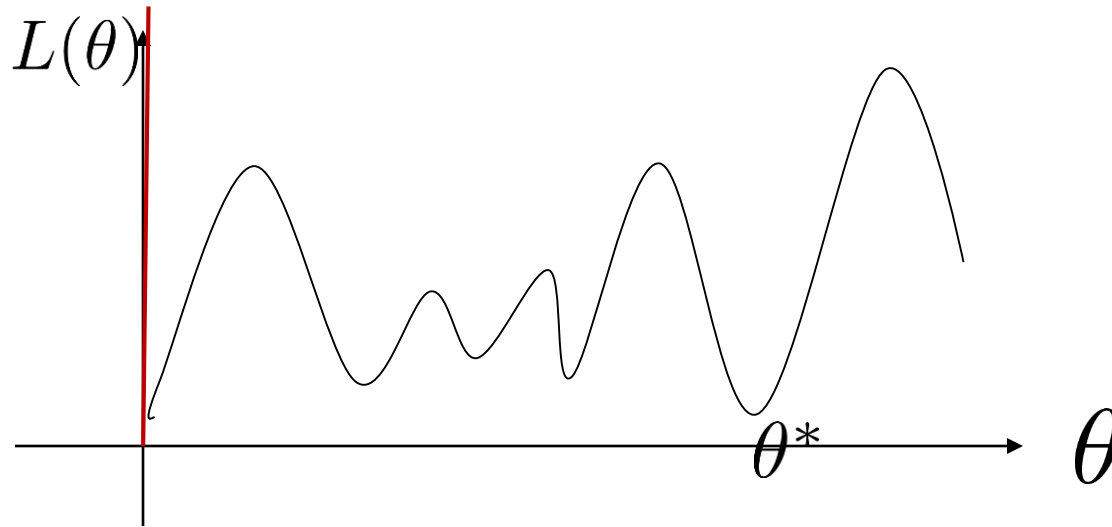
Given a θ , you can always evaluate the $L(\theta)$.

But you don't know which θ^* results in the minimum value of $L(\theta)$.

Optimization: exhaustive search

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad \mathbf{x}_i = (x_{i1}, \dots, x_{im}) \quad y \in \{-1, +1\}$$

$$\theta^* = \arg \min_{\theta} L(\theta)$$



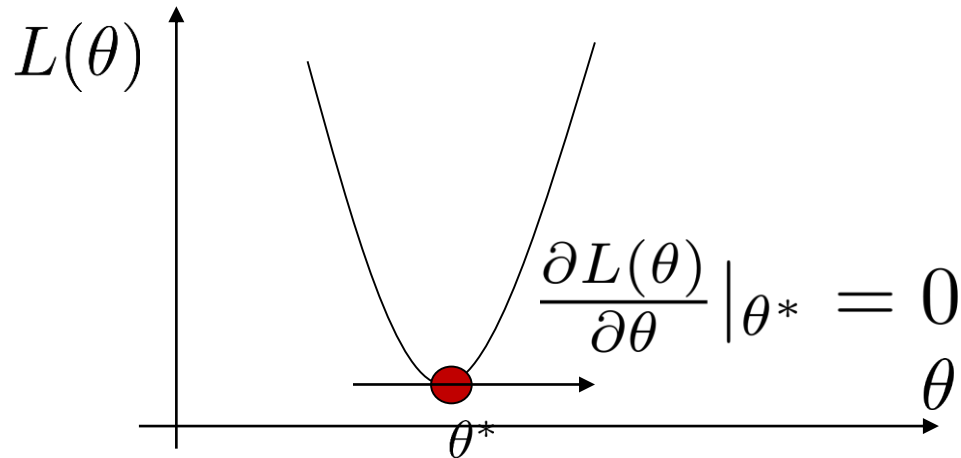
If you don't make any assumption about $L(\theta)$, a straight-forward way is to perform exhaustive search: searching for all possible θ .

You are guaranteed to find the optimal θ^* ,
but with an extremely high computational cost.

Optimization: closed form solution

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad \mathbf{x}_i = (x_{i1}, \dots, x_{im}) \quad y \in \{-1, +1\}$$

$$\theta^* = \arg \min_{\theta} L(\theta)$$



If you $L(\theta)$ is convex and everywhere differentiable:

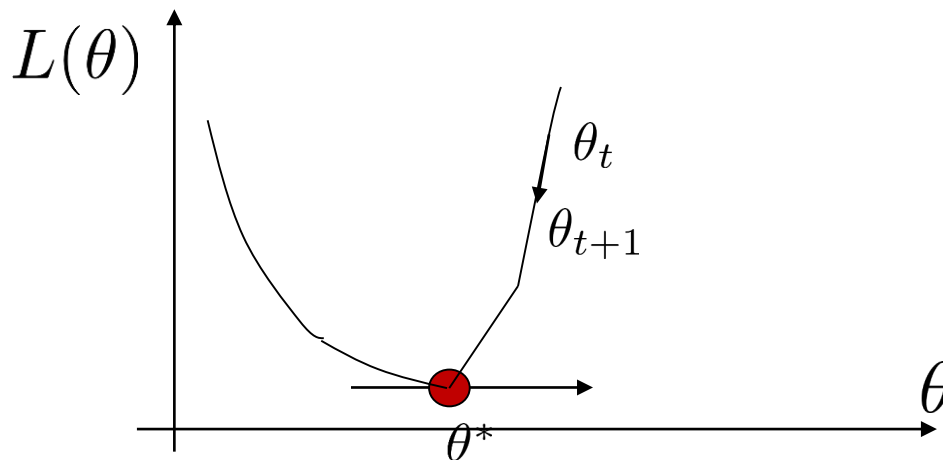
$$\text{Set: } \frac{\partial L(\theta)}{\partial \theta} = 0$$

$$\theta^* \text{ is the solution to } \frac{\partial L(\theta)}{\partial \theta} \big|_{\theta^*} = 0$$

Optimization: gradient descent

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad \mathbf{x}_i = (x_{i1}, \dots, x_{im}) \quad y \in \{-1, +1\}$$

$$\theta^* = \arg \min_{\theta} L(\theta)$$



If you $L(\theta)$ is convex but NOT everywhere differentiable:

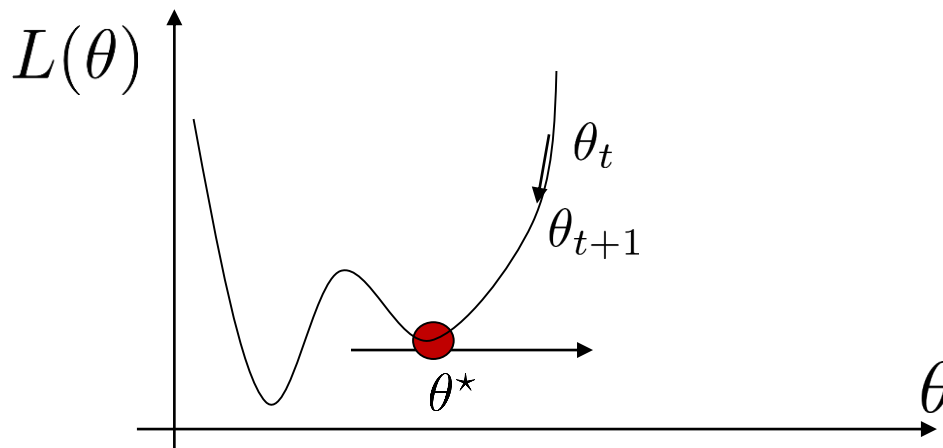
Search for θ^* such that $\frac{\partial L(\theta)}{\partial \theta} \big|_{\theta^*} = 0$

using gradient descent $\theta_{t+1} = \theta_t - \lambda_t \frac{\partial L(\theta)}{\partial \theta}$

Optimization: gradient descent

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad \mathbf{x}_i = (x_{i1}, \dots, x_{im}) \quad y \in \{-1, +1\}$$

$$\theta^* = \arg \min_{\theta} L(\theta)$$



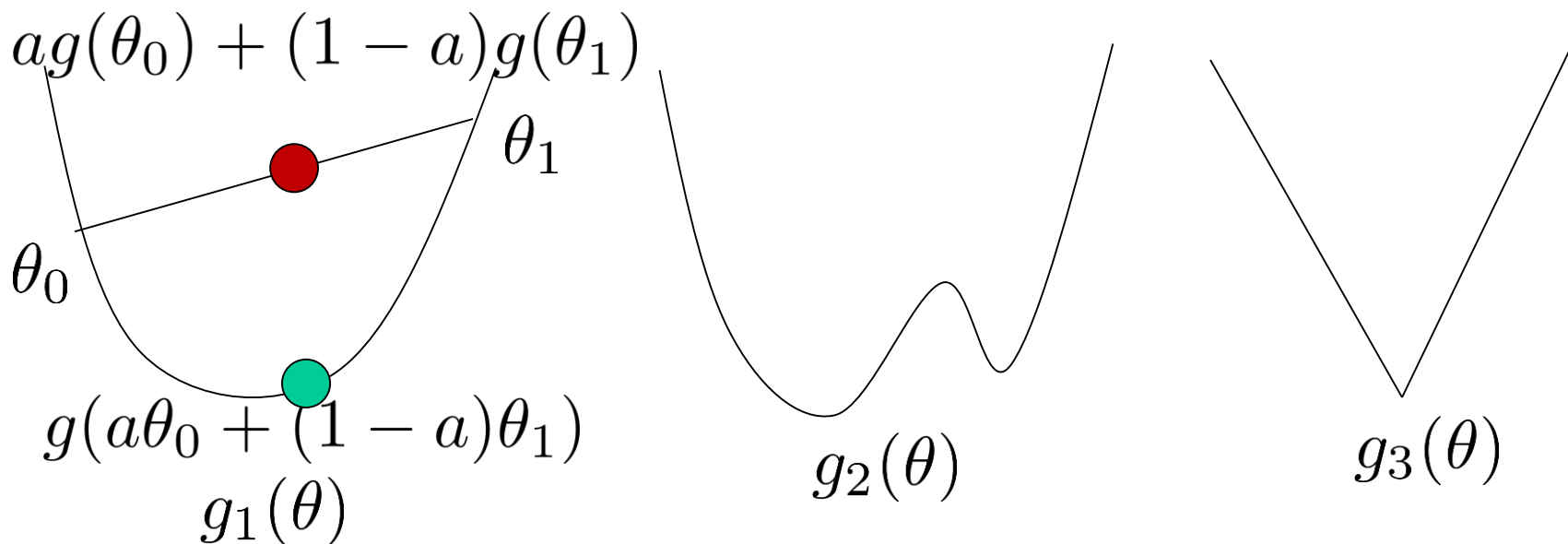
If you $L(\theta)$ is non-convex:

Search for θ^* such that $\frac{\partial L(\theta)}{\partial \theta} \big|_{\theta^*} = 0$

using gradient descent $\theta_{t+1} = \theta_t - \lambda_t \frac{\partial L(\theta)}{\partial \theta}$

Convex functions

$$\theta^* = \arg \min_{\theta} g(\theta)$$



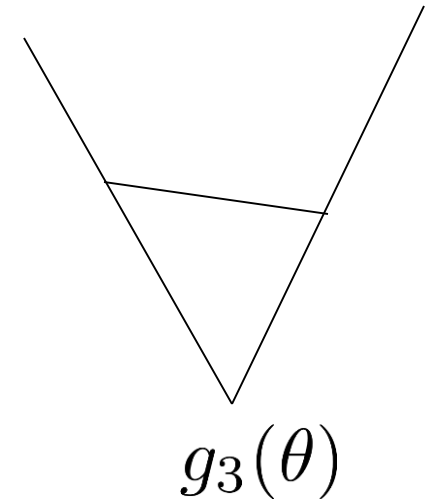
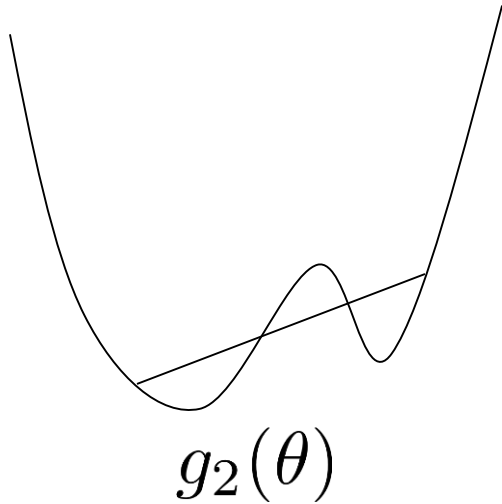
Definition:

$$\forall \theta_0, \theta_1, a \in [0, 1]$$

$$ag(\theta_0) + (1-a)g(\theta_1) \geq g(a\theta_0 + (1-a)\theta_1)$$

Convex functions

$$\theta^* = \arg \min_{\theta} g(\theta)$$



$$\forall \theta_0, \theta_1, a \in [0, 1]$$

$$ag(\theta_0) + (1 - a)g(\theta_1) \geq g(a\theta_0 + (1 - a)\theta_1)$$

Alternatively (for differentiable function):

$$g(\theta') \geq g(\theta) + \langle \nabla g(\theta), \theta' - \theta \rangle$$

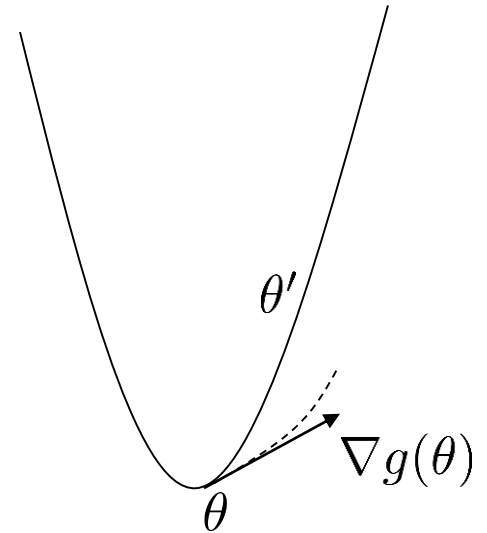
Strongly convex

Strongly convex: with parameter $\lambda > 0$

$$\langle \nabla g(\theta_0) - \nabla g(\theta_1), \theta_0 - \theta_1 \rangle \geq \lambda \|\theta_0 - \theta_1\|_2^2$$

Equivalently:

$$g(\theta_1) \geq g(\theta_0) + \langle \nabla g(\theta_0), \theta_1 - \theta_0 \rangle + \frac{\lambda}{2} \|\theta_1 - \theta_0\|_2^2$$



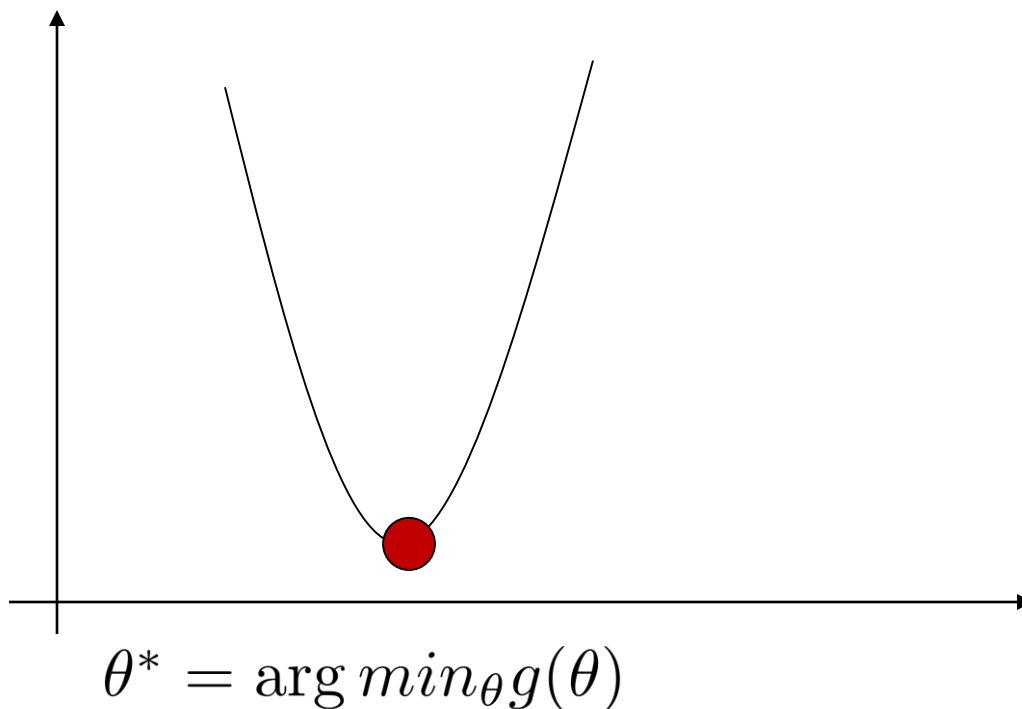
A strongly convex function is convex.

$$\theta^* = \arg \min_{\theta} f(\theta) = \arg_{\theta} [f'(\theta) \equiv 0]$$

A convex function is not necessarily strongly convex.

$$2 \times (\theta - 3) = 0 \rightarrow \theta = 3$$

Convex function: differentiable



$$g(\theta) = (\theta - 3)^2 + 4$$

$$\frac{dg(\theta)}{d\theta} = 2 \times (\theta - 3) \qquad \frac{dg(\theta)}{d\theta} = 0$$

$$2 \times (\theta - 3) = 0 \rightarrow \theta = 3$$

General approaches for optimization

- Exhaustive search
- Gradient descent
- Coordinate descent
- Newton's method
- Line search
- Stochastic computing
- Stochastic sampling (Markov chain Monte Carlo)
-

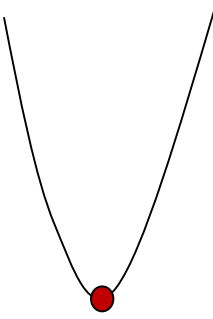
Parameters

Throughout the quarter in the class, we use either

W and θ

to denote the underlying model parameters that we want to learn.

Quadratic function: least square estimation


$$S_{training} = \{(x_i, y_i), i = 1..n\} \quad y_i \in \mathcal{R}$$

Obtain/train: $f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$

$$W = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

$$W^* = \arg \min_W \sum_i (\mathbf{x}_i^T \cdot W - y_i)^2$$

Let: $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T \quad Y = (y_1, \dots, y_n)^T$

$$\mathbf{x}_i = \begin{pmatrix} 1 \\ x_i \\ x_i^2 \end{pmatrix}$$

$$W^* = \arg \min_W = \arg \min_W g(W) = (XW - Y)^T (XW - Y)$$

$$g(W) = W^T X^T X W - W^T X^T Y - Y^T X W + Y^T Y$$

$$\frac{dg(W)}{dW} = 2X^T X W - 2X^T Y = 0$$

$$W^* = (X^T X)^{-1} X^T Y$$

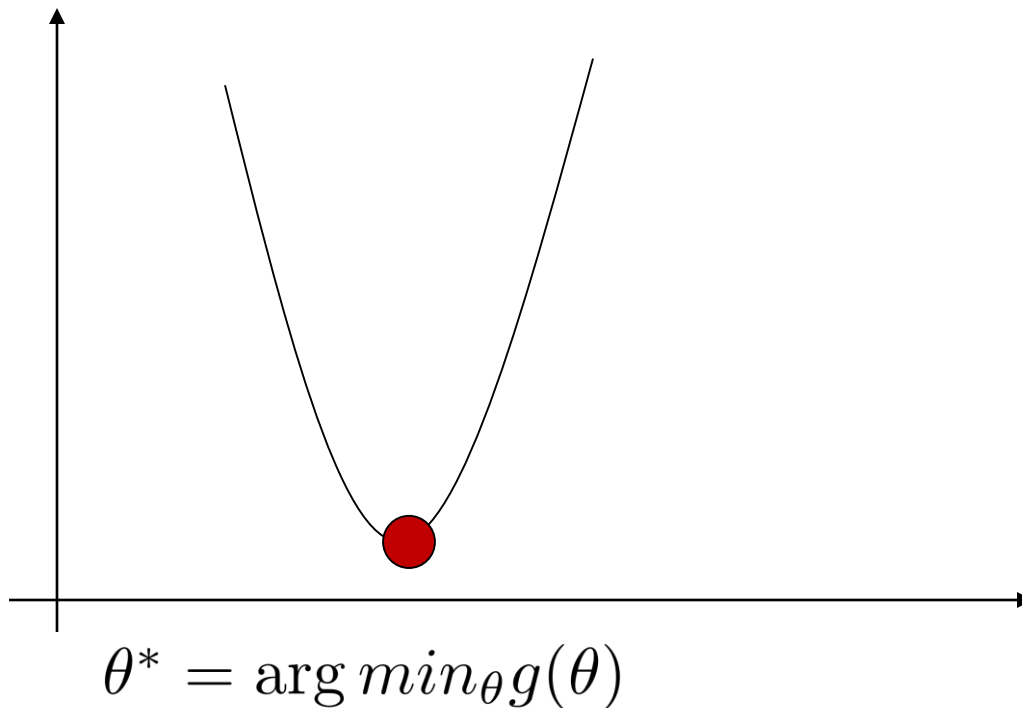
In matlab, you can simply call $X \setminus Y$ or $\text{pinv}(X) * Y$

In matlab

```
X=rand(1,20);  
Y=10.0+X*3.0+randn(1,20)/2.0;  
plot(X,Y,'.');  
X(2,:)=1.0;
```

```
c=inv(X*X')*X*Y'  
hold on;  
plot([0,1],[c(2),c(2)+c(1)],'r');
```

Convex function: differentiable



$$g(\theta) = (\theta - 3)^2 + 4$$

$$\frac{df(\theta)}{d\theta} = 2 \times (\theta - 3) \qquad \frac{df(\theta)}{d\theta} = 0$$

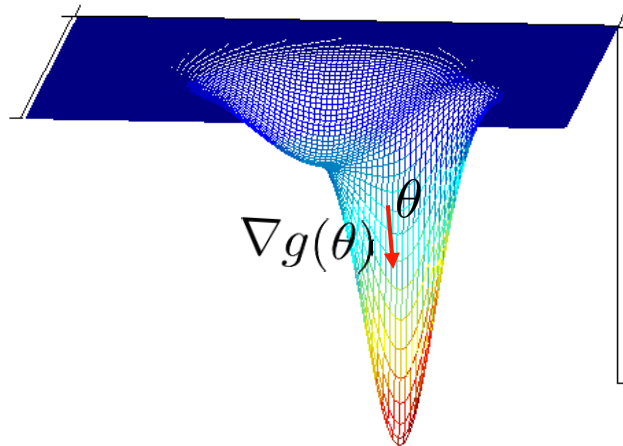
$$2 \times (\theta - 3) = 0 \rightarrow \theta = 3$$

Convex and differentiable: gradient descent

Definition:

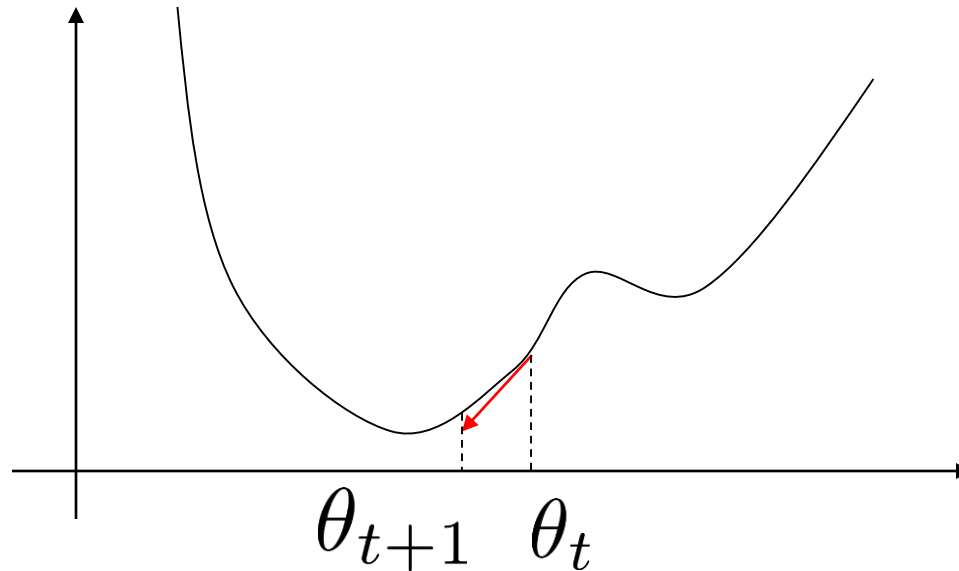
1. x^* is a globally optimal solution for $\theta^* \in \Omega$ and $g(\theta^*) \leq g(\theta) \forall \theta \in \Omega$
2. x^* is a locally optimal solution if there is a neighborhood \mathcal{N} around x such that $\theta^* \in \Omega$, $g(\theta^*) \leq g(\theta)$, $\forall \theta \in \mathcal{N} \cap \Omega$.

Gradient: $\nabla g(\theta) = [\frac{\partial g}{\partial \theta_i}]_{i=1, \dots, d}$ (its a vector!)



Hessian: $\nabla^2 g(\theta) = [\frac{\partial^2 g}{\partial \theta_i \partial \theta_j}]_{i,j=1, \dots, d}$ (its a matrix!)

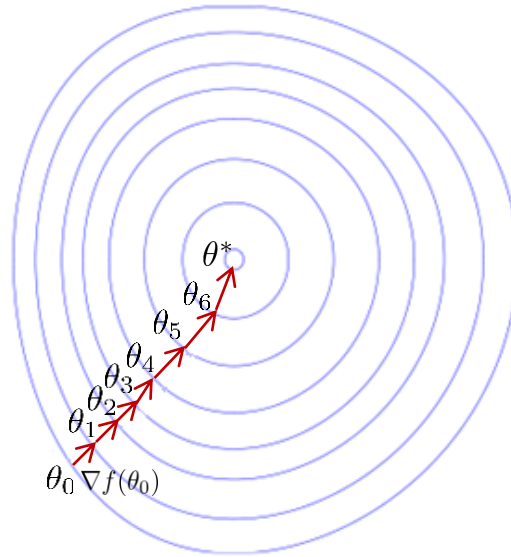
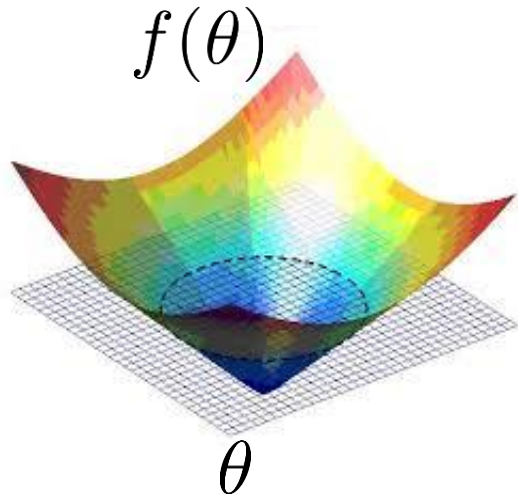
Gradient descent (ascent)



Gradient Method as a Line Search Method \rightarrow Descent Direction

- (a) Pick a direction v
- (b) Pick a step size λ
- (c) $\theta_{t+1} = \theta_t - \lambda \times v$ such that function decreases
- (d) Repeat

Gradient descent



$$\theta_{t+1} \leftarrow \theta_t - \lambda_t \nabla f(\theta_t) \quad \lambda_t : \text{stepsize}$$

Theorem:

For a continuous differentiable function f on a neighborhood of θ_0 , if $v^T \nabla f(\theta) < 0$, then there exists $T > 0$ such that $f(\theta_0 + tv) < f(\theta_0), \forall t \in (0, T]$.

L1 Loss

$$S_{training} = \{(x_i, y_i), i = 1..n\} \quad y_i \in \mathcal{R}$$

$$\text{Obtain/train: } f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$$

$$W = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

$$W^* = \arg \min_W \sum_{i=1}^n |\mathbf{x}_i^T \cdot W - y_i|$$

$$\mathbf{x}_i = \begin{pmatrix} 1 \\ x_i \\ x_i^2 \end{pmatrix}$$

$$\begin{aligned} \frac{\partial |f(w)|}{\partial w} &= \begin{cases} \frac{\partial |f(w)|}{\partial w} & \text{if } f(w) \geq 0 \\ -\frac{\partial |f(w)|}{\partial w} & \text{otherwise} \end{cases} \\ &= \text{sign}(f(w)) \cdot \frac{\partial f(w)}{\partial w} \end{aligned}$$

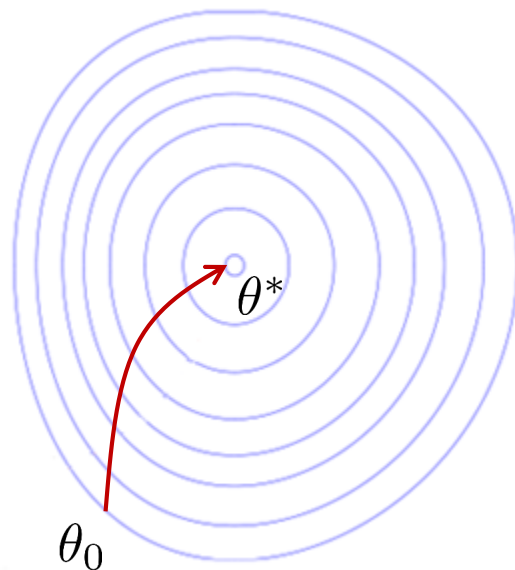
$$L(W) = \sum_{i=1}^n |\mathbf{x}_i^T W - y_i|$$

$$\frac{\partial L(W)}{\partial W} = \sum_{i=1}^n \text{sign}(\mathbf{x}_i^T W - y_i) \cdot \mathbf{x}_i$$

$$W_{t+1} = W_t - \lambda_t \frac{\partial L(W)}{\partial W}$$

Newton's method

$$\theta_{t+1} \leftarrow \theta_t - [\nabla^2 g(\theta_t)]^{-1} \nabla g(\theta_t)$$



A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information to take a more direct route.

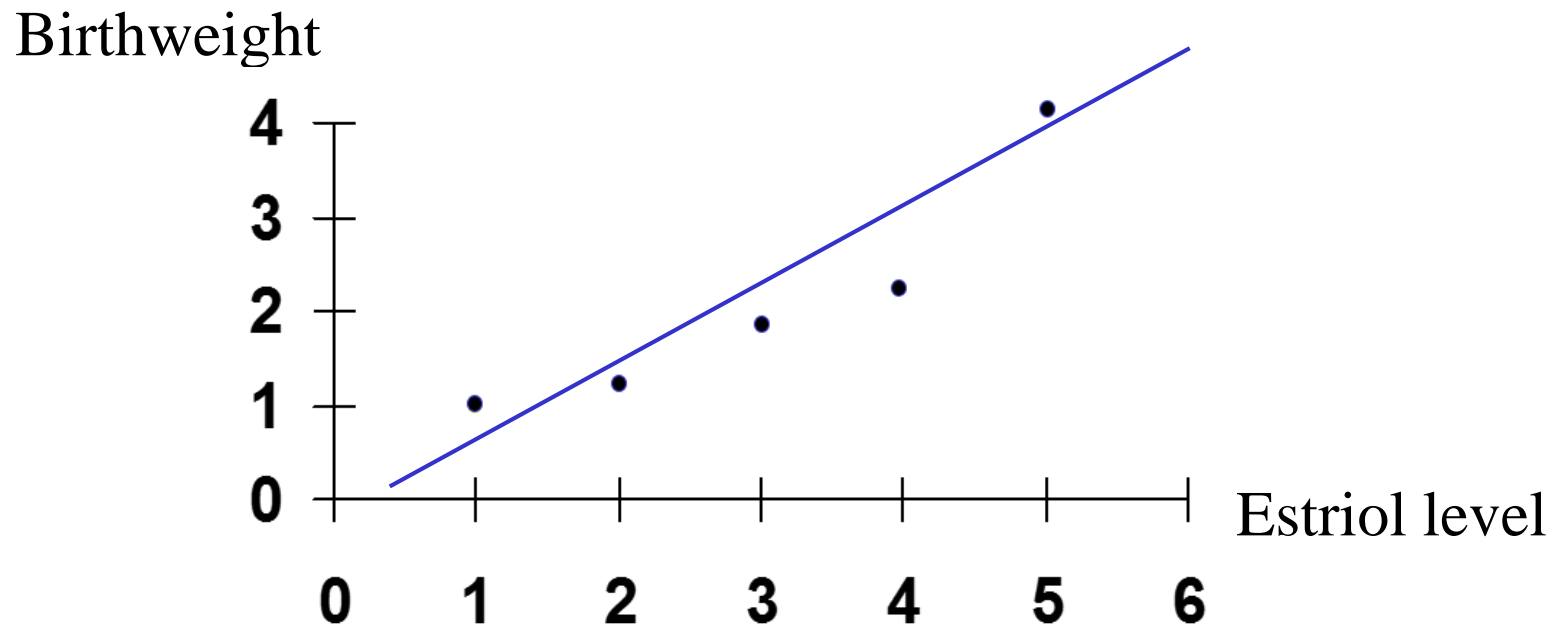
An example

- What is the relationship between Mother's Estriol level & Birthweight using the following data?

<u>Estriol</u>	<u>Birthweight</u>
(mg/24h)	(g/1000)
1	1
3	1.9
2	1.05
5	4.1
4	2.1

Scatterplot

Birthweight vs. Estriol level



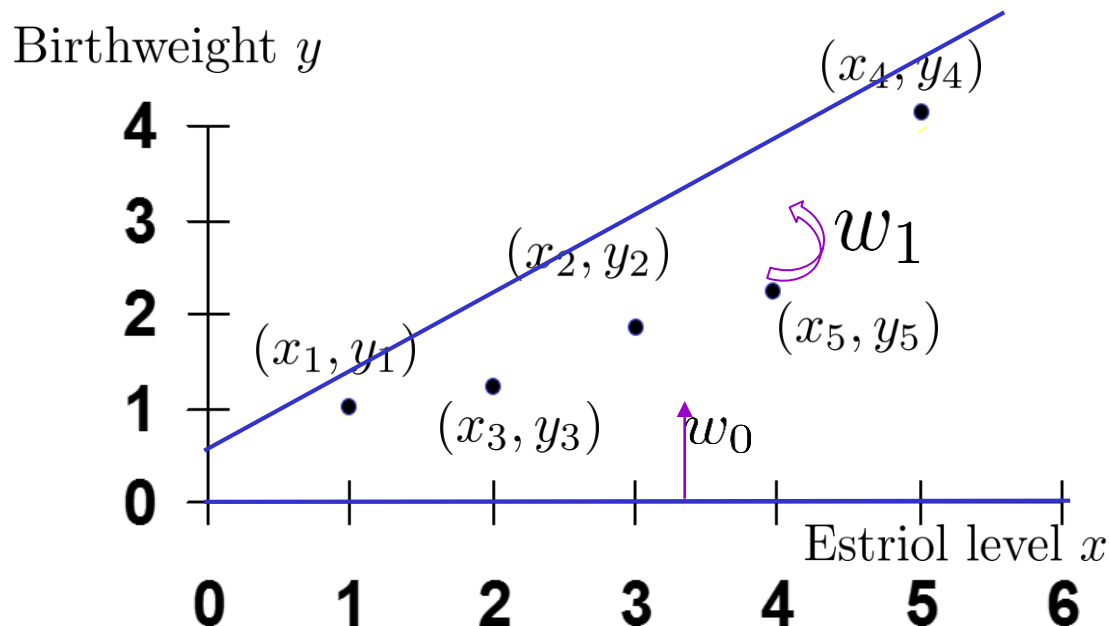
Univariate linear regression: the simplest case

$$S_{training} = \{(x_i, y_i), i = 1..n\}$$

Input: $x_i, x_i \in R$

Model parameter: $\mathbf{w} = (w_0, w_1), w_0, w_1 \in R$

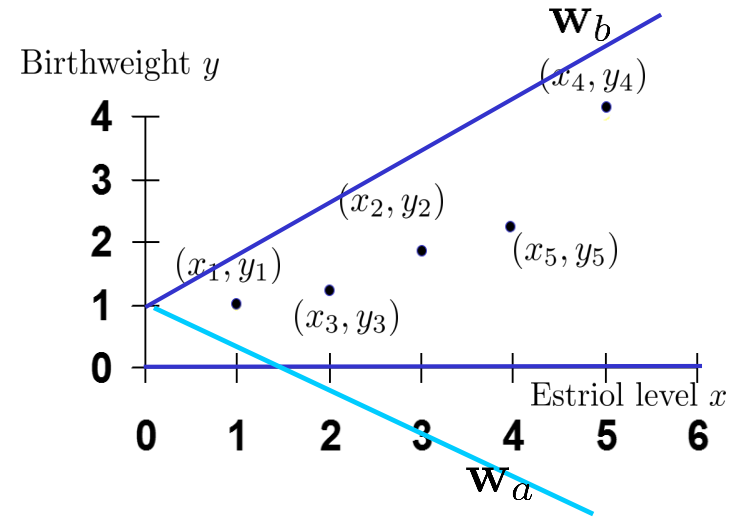
Output: $y_i = w_0 + w_1 x, \quad y_i \in R$



An example

- Training data

<u>Estriol</u> (mg/24h)	<u>Birthweight</u> (g/1000)
1	1
3	1.9
2	1.05
5	4.1
4	2.1



$$S_{training} = \{(x_i, y_i), i = 1..n\} = \{(1, 1), (3, 1.9), (2, 1.05), (5, 4.1), (4, 2.1)\}$$

$$\text{If } \mathbf{w}_a = (w_0, w_1) = (1, -0.5)$$

$$e_{training}(\mathbf{w}_a) = \frac{1}{5} \sum_{i=1}^5 (y_i - (1 - 0.5x_i))^2 = 9.62$$

$$\text{If } \mathbf{w}_b = (w_0, w_1) = (1, 0.5)$$

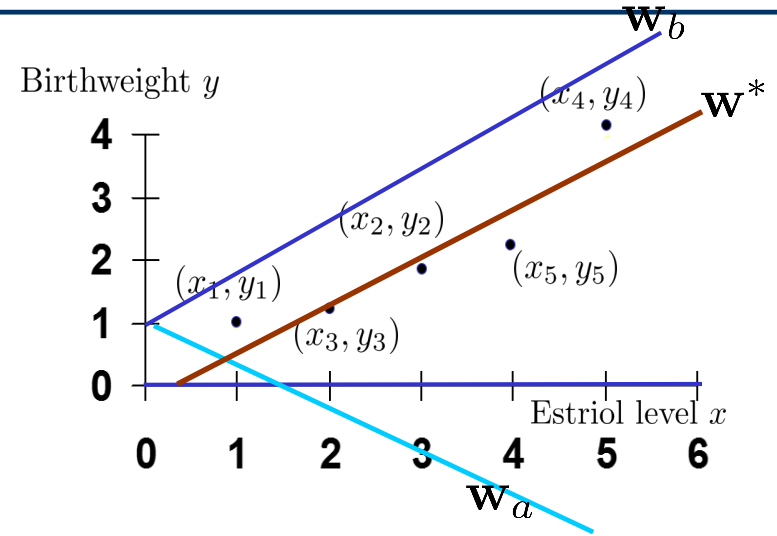
$$e_{training}(\mathbf{w}_b) = \frac{1}{5} \sum_{i=1}^5 (y_i - (1 + 0.5x_i))^2 = 0.54$$

\mathbf{w}_b is better than \mathbf{w}_a since $e_{training}(\mathbf{w}_b) < e_{training}(\mathbf{w}_a)$

An example

- **Training data**

<u>Estriol</u> (mg/24h)	<u>Birthweight</u> (g/1000)
1	1
3	1.9
2	1.05
5	4.1
4	2.1



$$S_{training} = \{(x_i, y_i), i = 1..n\} = \{(1, 1), (3, 1.9), (2, 1.05), (5, 4.1), (4, 2.1)\}$$

Our goal is find the optimal $\mathbf{w}^* = (w_0, w_1)^*$:

$$\mathbf{w}^* = \arg_{\mathbf{w}} \min e_{training}(\mathbf{w}) = \frac{1}{5} \sum_{i=1}^5 (y_i - (w_0 + w_1 * x_i))^2$$

Least square estimation

$$S_{training} = \{(x_i, y_i), i = 1..n\} \quad y_i \in \mathcal{R}$$

$$\text{Obtain/train: } f(x, \mathbf{w}) = w_0 + w_1 x \quad W = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

$$W^* = \arg \min_W \sum_i (\mathbf{x}_i^T \cdot W - y_i)^2 \quad \mathbf{x}_i = \begin{pmatrix} 1 \\ x_i \end{pmatrix}$$

$$\text{Let: } \mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T \quad Y = (y_1, \dots, y_n)^T$$

$$W^* = \arg \min_W = \arg \min_W g(W) = (X \cdot W - Y)^T (X \cdot W - Y)$$

$$g(W) = W^T X^T X W - W^T X^T Y - Y^T X W + Y^T Y$$

$$\frac{dg(W)}{dW} = 2X^T X W - 2X^T Y = 0$$

$$W^* = (X^T X)^{-1} X^T Y$$

In matlab, you can simply call $X \setminus Y$ or $\text{pinv}(X) * Y$

Linear regression algorithm

- Step 1: Select a model.
 - $y = w_0 + w_1x$ (For us right now – not always)
- Step 2: Put data into matrix form.
- Step 3: Solve for the weights (\mathbf{w}) in Matlab.
- Step 4: Report the model.
- Step 5: Visualize the model using model predictions.

Linear Regression Algorithm

- Step 1: Select a model.
 - $y = w_0 + w_1x$ (For us right now – not always)
- Step 2: Put data into matrix form.
 - $\mathbf{y} = \mathbf{A}\mathbf{w}$;
- Step 3: Solve for the weights (\mathbf{w}).
- Step 4: Report the model.
- Step 5: Visualize the model using model predictions.

Step 2: Put Data Into Matrix Form

$$S_{training} = \{(x_i, y_i), i = 1..n\} = \{(1, 1), (3, 1.9), (2, 1.05), (5, 4.1), (4, 2.1)\}$$

Basic Equations

$$y = w_0 + w_1 x$$

$$1 = w_0 + w_1 \times 1$$

$$1.9 = w_0 + w_1 \times 3$$

$$1.05 = w_0 + w_1 \times 2$$

$$4.1 = w_0 + w_1 \times 5$$

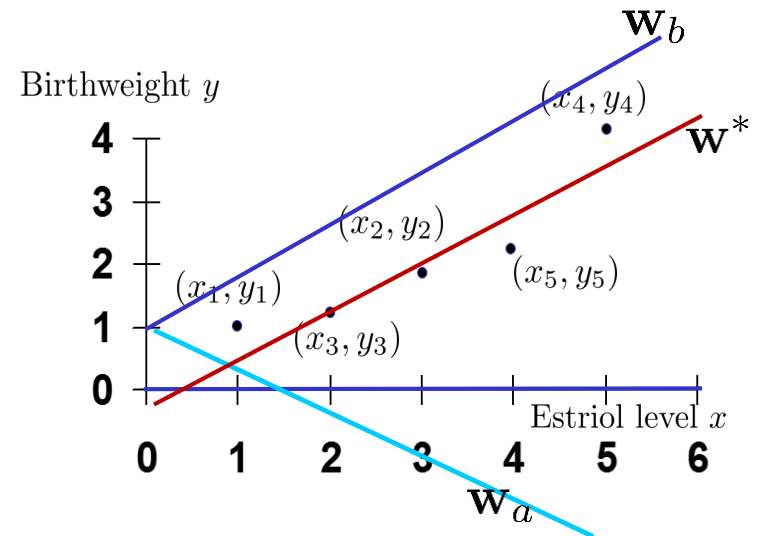
$$2.1 = w_0 + w_1 \times 4$$

Matrix Form

$$Y = X \cdot W$$

$$Y = X \cdot W$$

$$\begin{pmatrix} 1 \\ 1.9 \\ 1.05 \\ 4.1 \\ 2.1 \end{pmatrix} = \begin{pmatrix} 1, 1 \\ 1, 3 \\ 1, 2 \\ 1, 5 \\ 1, 4 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$



In matlab, you can simply call
 $X \setminus Y$ or $\text{pinv}(X) * Y$

$$\begin{pmatrix} -0.145 \\ 0.725 \end{pmatrix}$$

$$e_{training}(\mathbf{w}_a) = 9.62$$

$$e_{training}(\mathbf{w}_b) = 0.54$$

$$e_{training}(\mathbf{w}^*) = 0.21$$

Linear Regression Algorithm

- Step 1: Select a model.
 - $y = w_0 + w_1x$ (For us right now – not always)
- Step 2: Put data into matrix form.
 - $y = Aw$;
- Step 3: Solve for the weights (w) in Matlab.
 - $w = A \backslash y$; or $w = \text{pinv}(A) * y$;
- Step 4: Report the model.
- Step 5: Visualize the model using model predictions.

Linear Regression Algorithm

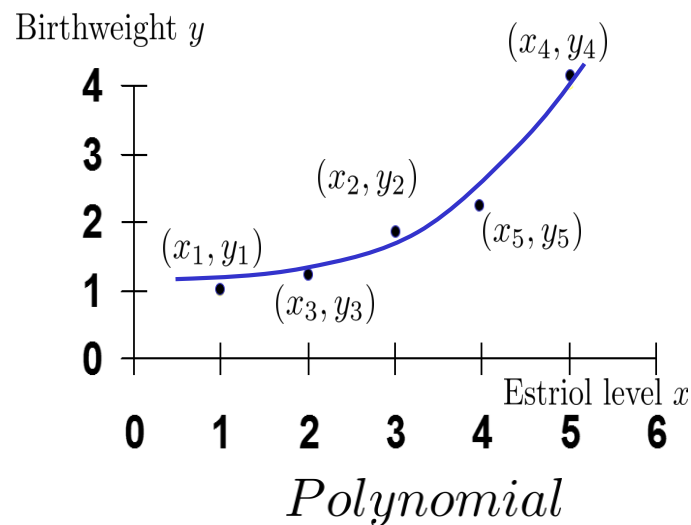
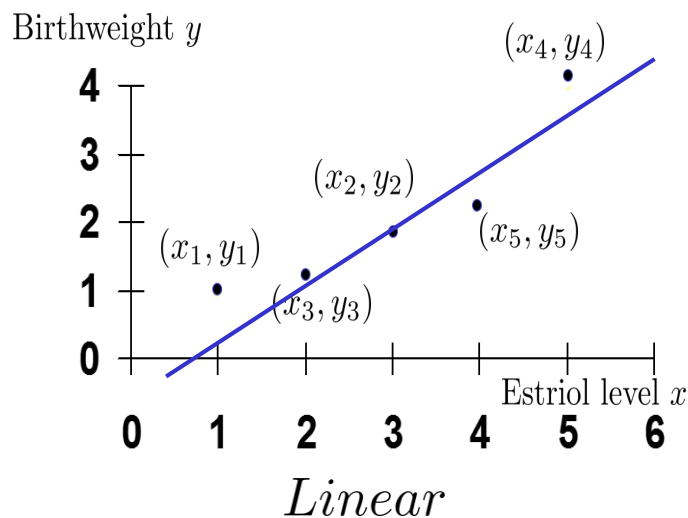
- Step 1: Select a model.
 - $y = w_0 + w_1x$ (For us right now – not always)
- Step 2: Put data into matrix form.
 - $y = Aw$;
- Step 3: Solve for the weights (**w**) in Matlab.
 - $w = A \backslash y$; or $w = \text{pinv}(A) * y$;
- Step 4: Report the model.
 - Fill in the **w** values in our model (step 1) and write out again.
 - $y = \mathbf{-0.145} + \mathbf{0.725x}$
- Step 5: Visualize the model using model predictions.

Univariate linear regression

Why is this supervised? How is it labeled?

We are treating the variable we are plotting on the y-axis as a label – as a truth that we want to estimate and predict for new data.

There are different types of regressors, leading to different levels of complexity.



More Complex Linear Combinations

- Polynomial Linear Regression
- Multivariate Linear Regression

Polynomial Linear Regression

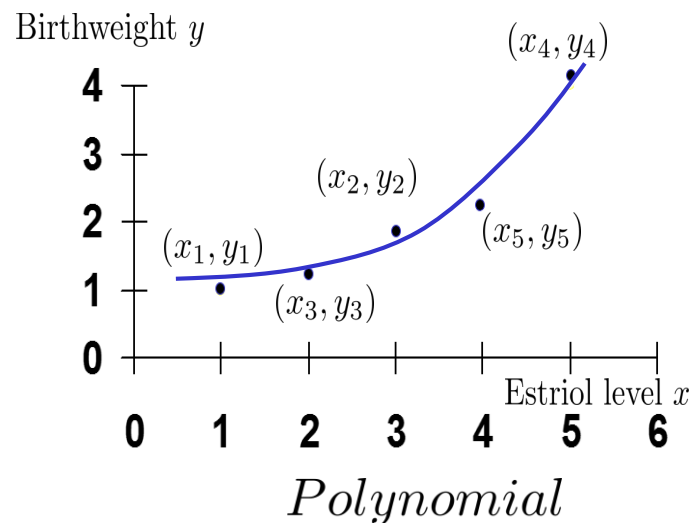
$$S_{training} = \{(x_i, y_i), i = 1..n\}$$

Input: $x, x \in R$

Model parameter: $\mathbf{w} = (w_0, w_1, \dots, w_d), w_i \in R$

$$\text{Output: } y = w_0 + w_1x_1 + w_2x_1^2 + \dots + w_qx_m^q$$

The combination of terms has your input variable raised to an addition power for each subsequent term.



Put Data Into Matrix Form

$$S_{training} = \{(x_i, y_i), i = 1..n\} = \{(1, 1), (3, 1.9), (2, 1.05), (5, 4.1), (4, 2.1)\}$$

Basic Equations Matrix Form

$$y = w_0 + w_1x + w_2x^2 \quad Y = X \cdot W$$

$$1 = w_0 + w_1 \times 1 + w_2 \times 1$$

$$1.9 = w_0 + w_1 \times 3 + w_2 \times 9$$

$$1.05 = w_0 + w_1 \times 2 + w_2 \times 4$$

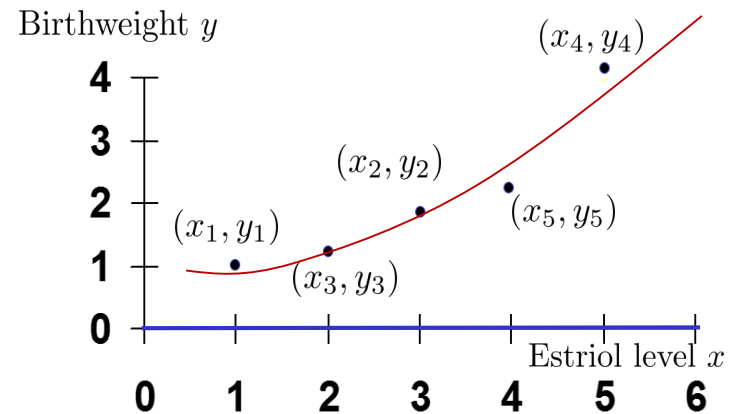
$$4.1 = w_0 + w_1 \times 5 + w_2 \times 25$$

$$2.1 = w_0 + w_1 \times 4 + w_2 \times 16$$

$$Y = X \cdot W$$

$$\begin{pmatrix} 1 \\ 1.9 \\ 1.05 \\ 4.1 \\ 2.1 \end{pmatrix} = \begin{pmatrix} 1, 1, 1 \\ 1, 3, 9 \\ 1, 2, 4 \\ 1, 5, 25 \\ 1, 4, 16 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

$$e_{training}(\mathbf{w}^*) = 0.063$$



In matlab, you can simply call
 $X \setminus Y$ or $\text{pinv}(X) * Y$

$$\begin{pmatrix} 1.48 \\ -0.67 \\ 0.2321 \end{pmatrix}$$

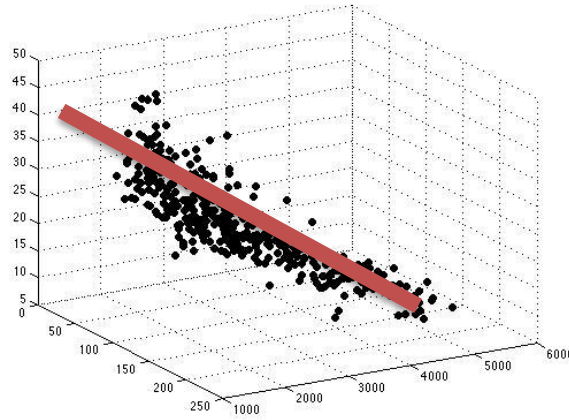
$$e_{training}(\mathbf{w}_a) = 9.62$$

$$e_{training}(\mathbf{w}_b) = 0.54$$

$$e_{training}(\mathbf{w}^*) = 0.21$$

previous linear model

Multi-variate Linear Regression



Input: $\mathbf{x} = (x_1, \dots, x_m)$, $x_i \in R$

Model parameter: $\mathbf{w} = (w_0, w_1, \dots, w_m)$, $w_i \in R$

Output: $y = w_0 + w_1x_1 + w_2x_2 + \dots + w_mx_m$

We apply do the least square fitting method again.

Step 2: Put Data Into Matrix Form

$$\begin{aligned} S_{training} &= \{((x_{i1}, x_{i2}), y_i), i = 1..n\} \\ &= \{((1, 0.5), 1), ((3, 0.9), 1.9), ((2, 1.0), 1.05), ((5, 6.7), 4.1), ((4, 2.5), 2.1)\} \end{aligned}$$

Basic Equations

$$y = w_0 + w_1x_1 + w_2x_2$$

$$1 = w_0 + w_1 \times 1 + w_2 \times 0.5$$

$$1.9 = w_0 + w_1 \times 3 + w_2 \times 0.9$$

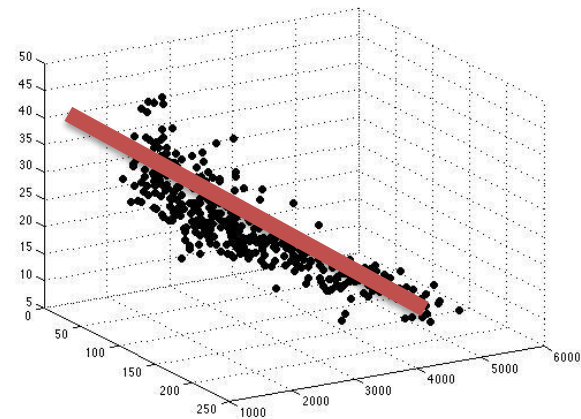
$$1.05 = w_0 + w_1 \times 2 + w_2 \times 1.0$$

$$4.1 = w_0 + w_1 \times 5 + w_2 \times 6.7$$

$$2.1 = w_0 + w_1 \times 4 + w_2 \times 2.5$$

Matrix Form

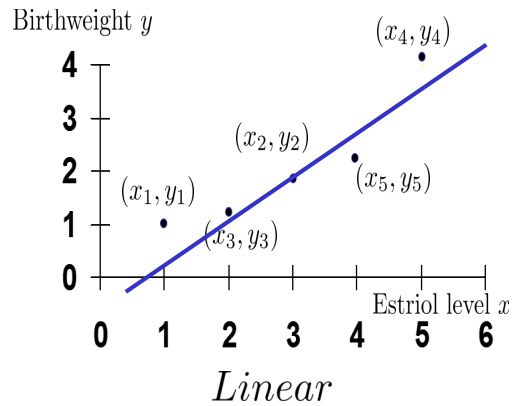
$$Y = X \cdot W$$



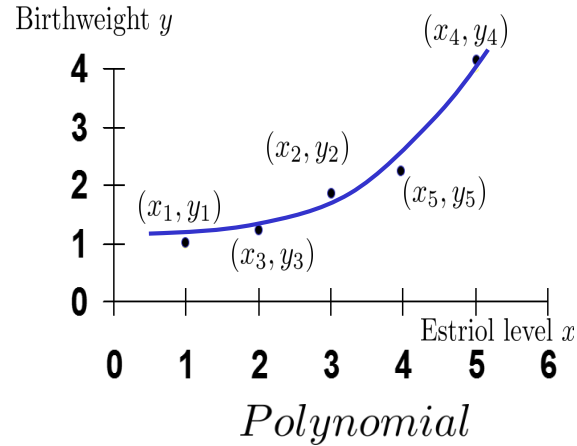
In matlab, you can simply call
 $X \setminus Y$ or $\text{pinv}(X) * Y$

$$\begin{pmatrix} 1 \\ 1.9 \\ 1.05 \\ 4.1 \\ 2.1 \end{pmatrix} = \begin{pmatrix} 1, 1, 0.5 \\ 1, 3, 0.9 \\ 1, 2, 1.0 \\ 1, 5, 6.7 \\ 1, 4, 2.5 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} \quad \begin{pmatrix} 0.482 \\ 0.2552 \\ 0.338 \end{pmatrix}$$

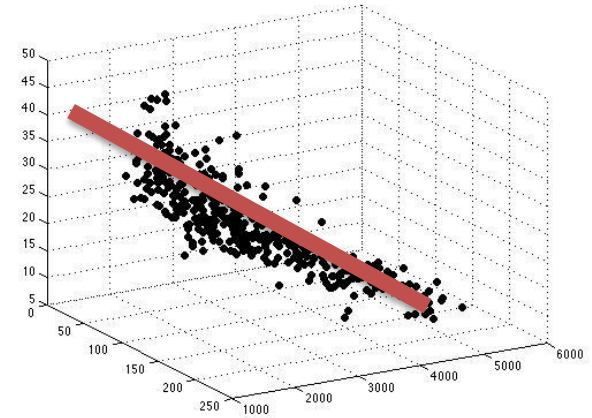
Select your model



$$e_{training}(\mathbf{w}^*) = 0.21$$



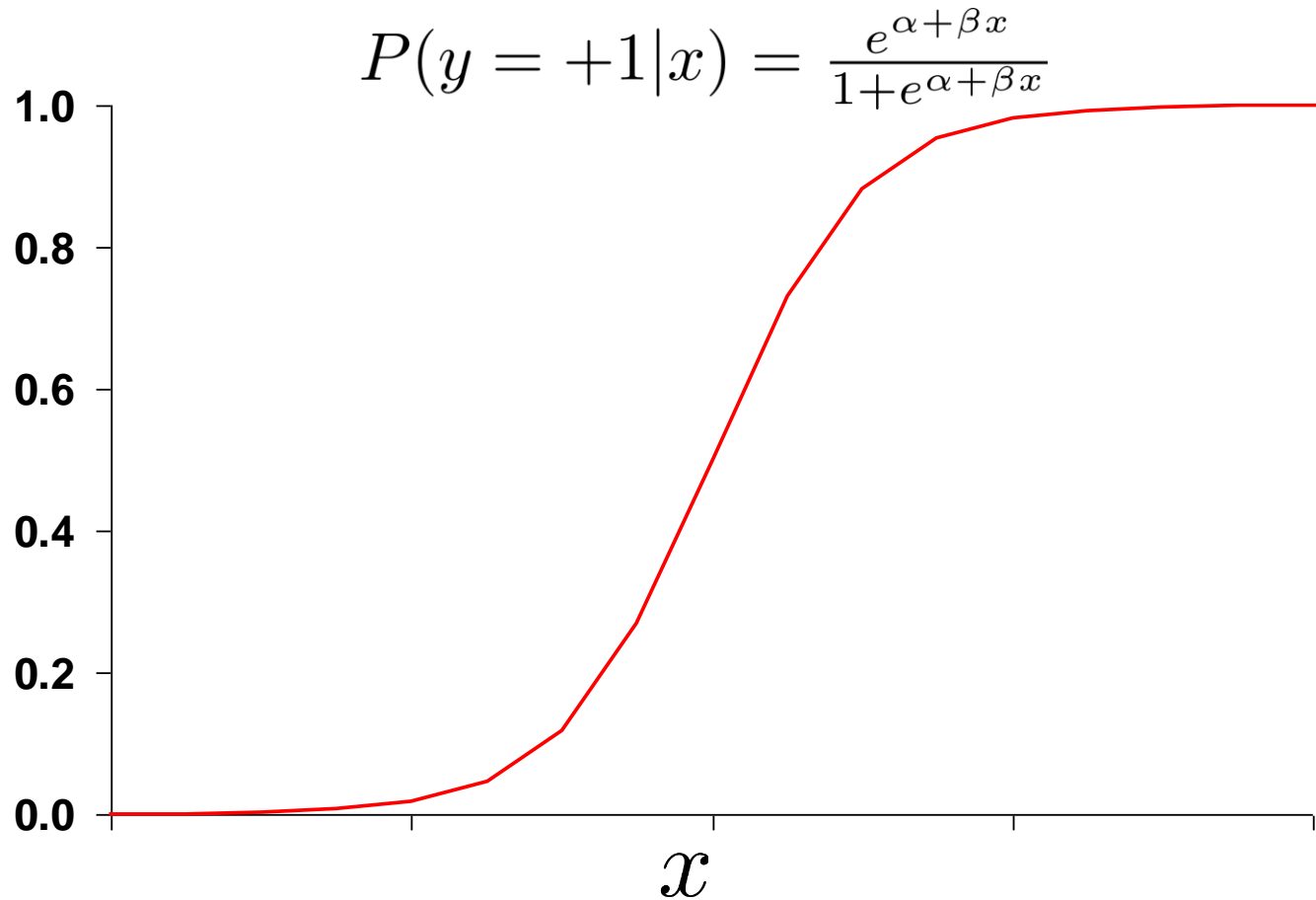
$$e_{training}(\mathbf{w}^*) = 0.063$$



$$e_{training}(\mathbf{w}^*) = 0.052$$

The logistic function

Probability of
being positive



The logistic function

$$p(y = 1|x) = \frac{e^{\alpha + \beta x}}{1 + e^{\alpha + \beta x}} = \frac{1}{1 + e^{-(\alpha + \beta x)}}$$

$$\ln\left(\frac{p(y=1|x)}{1-p(y|x)}\right) = \alpha + \beta x \quad \text{logit of } p(y = 1|x)$$

Advantages of the logit

- Simple transformation of $P(y|x)$
- Linear relationship with x
- Can be continuous (Logit between $-\infty$ to $+\infty$)
- Known binomial distribution (P between 0 and 1)
- Directly related to the notion of odds of disease

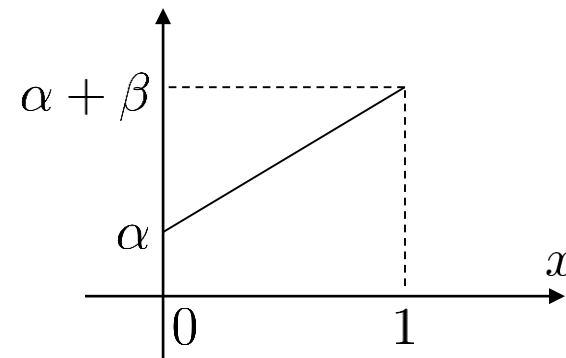
$$\ln\left(\frac{p}{1-p}\right) = \alpha + \beta x \quad \frac{p}{1-p} = e^{\alpha + \beta x}$$

The logistic function

$$\text{Logit of } P(y = +1|x): \ln\left(\frac{P(y=+1|x)}{1-P(y=+1|x)}\right) = \alpha + \beta x$$

Advantages of the logit:

1. Simple transformation of $p(y|x)$
2. Linear relationship with x
3. Can be continuous (Logit between $-\infty$ to $+\infty$)
4. Known binomial distribution (P between 0 and 1)



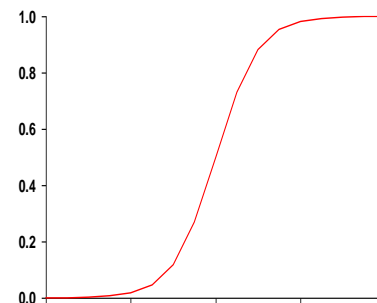
The logistic function

Logit of $P(y = +1|x)$: $\ln\left(\frac{P(y=+1|x)}{1-P(y=+1|x)}\right) = \alpha + \beta x$

$$P(y = +1|x) = \frac{e^{\alpha + \beta x}}{1 + e^{\alpha + \beta x}} = \frac{1}{1 + e^{-(\alpha + \beta x)}}$$

$$P(y = -1|x) = \frac{1}{1 + e^{\alpha + \beta x}}$$

$$P(y = +1|x) + P(y = -1|x) = 1$$



An agnostic notation: $P(y|x) = \frac{1}{1 + e^{-y(\alpha + \beta x)}}$

Training a logistic regression classifier

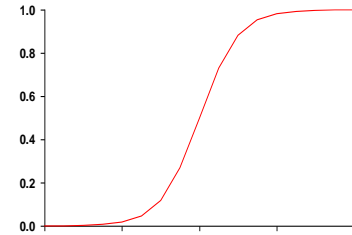
$$S_{training} = \{(x_i, y_i), i = 1..n\} \quad \begin{array}{l} x_i \in R, i = 1..n \\ y_i \in \{-1, +1\}, i = 1..n \end{array}$$

$$S_{training} = \{(-1.1, -1), (3.2, +1), (2.5, -1), (5.0, +1), (4.3, +1)\}$$

Train a classifier $f(x)$:

If $f(x)$ is a logistic regression classifier:

$$f(x) = \begin{cases} +1 & \text{if } \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$



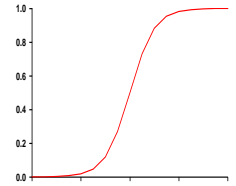
$$e_{training} = \frac{1}{n} \sum_{i=1}^n \mathbf{1}(y_i \neq f(x_i))$$

Training a logistic regression classifier

$$S_{training} = \{(-1.1, -1), (3.2, +1), (2.5, -1), (5.0, +1), (4.3, +1)\} \quad \begin{array}{l} x_i \in R, i = 1..n \\ y_i \in \{-1, +1\}, i = 1..n \end{array}$$

Train a logistic regression classifier $f(x)$:

$$P(y = +1|x) = \frac{1}{1+e^{-(\alpha+\beta x)}} \quad f(x) = \begin{cases} +1 & \text{if } \frac{1}{1+e^{-(\alpha+\beta x)}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$



$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n [P(y_i = +1|x_i)]^{0.5+y_i/2} \times [P(y_i = -1|x_i)]^{0.5-y_i/2}$$

$$P(y_i|x_i) = \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} - \sum_{i=1}^n \ln\left(\frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}\right) = \arg \min_{(\alpha, \beta)} \sum_{i=1}^n \ln(1+e^{-y_i(\alpha+\beta x_i)})$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} [\ln(1+e^{(\alpha-1.1\beta)}) + \ln(1+e^{-(\alpha+3.2\beta)}) +$$

$$\ln(1+e^{(\alpha+2.5\beta)}) + \ln(1+e^{-(\alpha+5.0\beta)}) + \ln(1+e^{-(\alpha+4.3\beta)})]$$

Training a logistic regression classifier

We want to maximize the probability for a given training set. Suppose we have two points $\{(x_1, y_1 = +1), (x_2, y_2 = -1)\}$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^2 [P(y_i | x_i)]$$

where

$$P(y_i | x_i) = [P(y_i = +1 | x_i)]^{0.5+y_i/2} \times [P(y_i = -1 | x_i)]^{0.5-y_i/2}$$

In the example given above:

$$P(y_1 | x_1) = [P(y_1 = +1 | x_1)]^{0.5+1/2} \times [P(y_1 = -1 | x_1)]^{0.5-1/2} = [P(y_1 = +1 | x_1)] \times 1, \text{ since } y_1 = +1$$

$$P(y_2 | x_2) = [P(y_2 = +1 | x_2)]^{0.5-1/2} \times [P(y_2 = -1 | x_2)]^{0.5+1/2} = 1 \times [P(y_2 = -1 | x_2)], \text{ since } y_2 = -1$$

Training a logistic regression classifier

We want to maximize the probability for a given training set. Suppose we have two points $\{(x_1, y_1 = +1), (x_2, y_2 = -1)\}$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^2 [P(y_i | x_i)]$$

where

$$P(y_i | x_i) = [P(y_i = +1 | x_i)]^{0.5+y_i/2} \times [P(y_i = -1 | x_i)]^{0.5-y_i/2}$$

We then have the form:

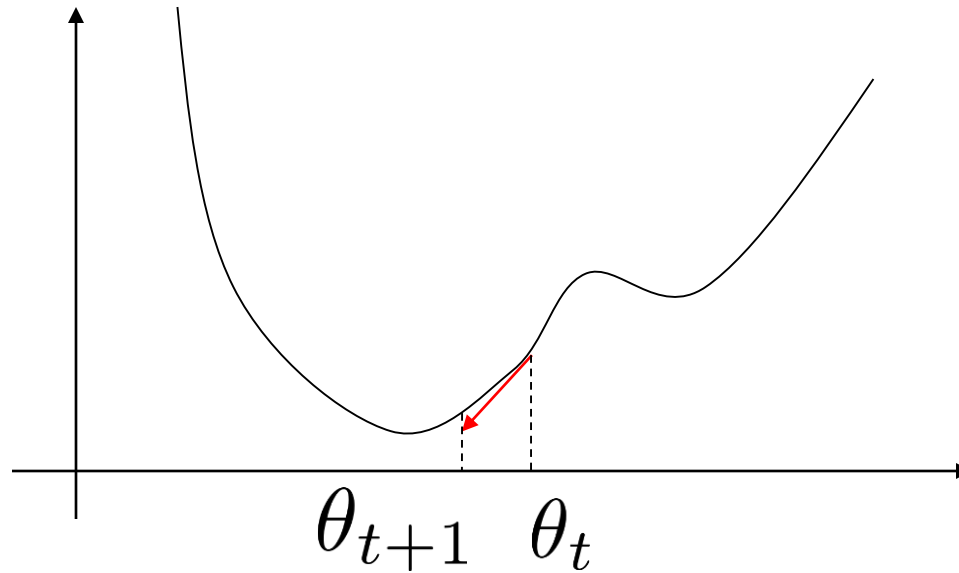
$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n [P(y_i = +1 | x_i)]^{0.5+y_i/2} \times [P(y_i = -1 | x_i)]^{0.5-y_i/2},$$

which can be simplified to:

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1 + e^{-y_i(\alpha + \beta x_i)}}$$

if we have: $P(y|x) = \frac{1}{1 + e^{-y(\alpha + \beta x)}}$

Gradient descent (ascent)



Gradient Descent Direction

- (a) Pick a direction $\nabla g(x_t)$
- (b) Pick a step size λ_t
- (c) $\theta_{t+1} = \theta_t - \lambda_t \times \nabla g(\theta_t)$ such that function decreases
- (d) Repeat

Training a logistic regression classifier

$$P(y_i|x_i) = \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

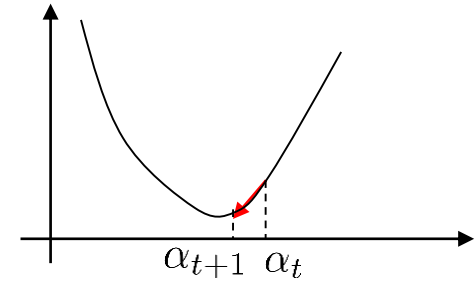
$$x_i \in R, i = 1..n$$

$$y_i \in \{-1, +1\}, i = 1..n$$

Train a logistic regression classifier $f(x)$:

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} g(\alpha, \beta)$$

$$g(\alpha, \beta) = \sum_{i=1}^n \ln(1 + e^{-y_i(\alpha+\beta x_i)})$$



$$\nabla_{\alpha} g(\alpha, \beta) = \sum_i \frac{-y_i e^{-y_i(\alpha+\beta x_i)}}{1+e^{-y_i(\alpha+\beta x_i)}} = \sum_i -y_i(1 - P(y_i|x_i))$$

$$\nabla_{\beta} g(\alpha, \beta) = \sum_i \frac{-y_i x_i e^{-y_i(\alpha+\beta x_i)}}{1+e^{-y_i(\alpha+\beta x_i)}} = \sum_i -y_i x_i(1 - P(y_i|x_i))$$

$$\alpha_{t+1} = \alpha_t - \lambda_t \times \nabla_{\alpha} g(\alpha_t, \beta_t)$$

$$\beta_{t+1} = \beta_t - \lambda_t \times \nabla_{\beta} g(\alpha_t, \beta_t)$$

Sometimes we also use $\{0, 1\}$ for y

$$S_{training} = \{(-1.1, 0), (3.2, 1), (2.5, 0), (5.0, 1), (4.3, 1)\}$$

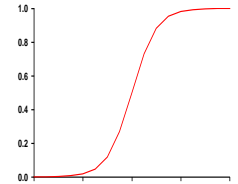
$$x_i \in R, i = 1..n$$

$$y_i \in \{0, 1\}, i = 1..n$$

Train a logistic regression classifier $f(x)$:

$$P(y = 1|x) = \frac{1}{1+e^{-(\alpha+\beta x)}} \quad f(x) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(\alpha+\beta x)}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$P(y = 0|x) = \frac{e^{-(\alpha+\beta x)}}{1+e^{-(\alpha+\beta x)}} = \frac{1}{1+e^{(\alpha+\beta x)}}$$



$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n [P(y_i = 1|x_i)]^{y_i} \times [P(y_i = 0|x_i)]^{1-y_i}$$

$$P(y_i|x_i) = \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

Sometimes we also use $\{0, 1\}$ for y

$$S_{training} = \{(-1.1, 0), (3.2, 1), (2.5, 0), (5.0, 1), (4.3, 1)\}$$

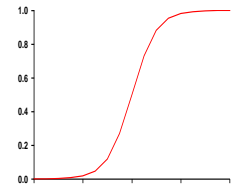
$$x_i \in R, i = 1..n$$

$$y_i \in \{0, 1\}, i = 1..n$$

Train a logistic regression classifier $f(x)$:

$$P(y = 1|x) = \frac{1}{1+e^{-(\alpha+\beta x)}} \quad f(x) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(\alpha+\beta x)}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$P(y = 0|x) = \frac{e^{-(\alpha+\beta x)}}{1+e^{-(\alpha+\beta x)}} = \frac{1}{1+e^{(\alpha+\beta x)}}$$



$$P(y_i|x_i) = \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} - \sum_{i=1}^n \ln\left(\frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}\right) = \arg \min_{(\alpha, \beta)} \sum_{i=1}^n \ln(1+e^{-(2y_i-1)(\alpha+\beta x_i)})$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} [\ln(1+e^{(\alpha-1.1\beta)}) + \ln(1+e^{-(\alpha+3.2\beta)}) +$$

$$\ln(1+e^{(\alpha+2.5\beta)}) + \ln(1+e^{-(\alpha+5.0\beta)}) + \ln(1+e^{-(\alpha+4.3\beta)})]$$

Multivariate input

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}$$

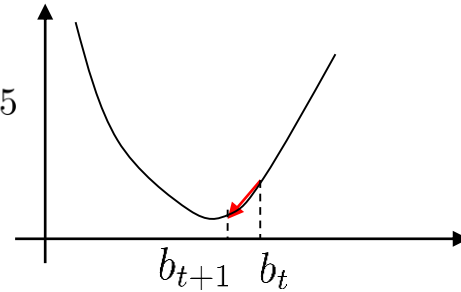
$$\mathbf{x}_i \in R^m, i = 1..n$$

$$y_i \in \{-1, +1\}, i = 1..n$$

Train a logistic regression classifier $f(\mathbf{x})$:

$$(b, \mathbf{w})^* = \arg \min_{(b, \mathbf{w})} g(b, \mathbf{w}) \quad f(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(b+\mathbf{w}^T\mathbf{x})}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$

$$g(b, \mathbf{w}) = \sum_{i=1}^n \ln(1 + e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)})$$



$$\nabla_b g(b, \mathbf{w}) = \sum_i \frac{-y_i e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -y_i(1 - P(y_i|\mathbf{x}_i))$$

$$\nabla_{\mathbf{w}} g(b, \mathbf{w}) = \sum_i \frac{-y_i \mathbf{x}_i e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -y_i \mathbf{x}_i(1 - P(y_i|\mathbf{x}_i))$$

$$b_{t+1} = b_t - \lambda_t \times \nabla_b g(b_t, \mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla_{\mathbf{w}} g(b_t, \mathbf{w}_t)$$

Multivariate input and $y \in \{0, 1\}$

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}}$$

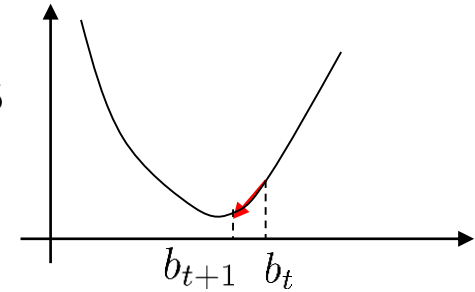
$$\mathbf{x}_i \in R^m, i = 1..n$$

$$y_i \in \{0, 1\}, i = 1..n$$

Train a logistic regression classifier $f(\mathbf{x})$:

$$(b, \mathbf{w})^* = \arg \min_{(b, \mathbf{w})} g(b, \mathbf{w}) \quad f(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(b+\mathbf{w}^T\mathbf{x})}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$g(b, \mathbf{w}) = \sum_{i=1}^n \ln(1 + e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)})$$



$$\nabla_b g(b, \mathbf{w}) = \sum_i \frac{-(2y_i-1)e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -(2y_i-1)(1-P(y_i|\mathbf{x}_i))$$

$$\nabla_{\mathbf{w}} g(b, \mathbf{w}) = \sum_i \frac{-(2y_i-1)\mathbf{x}_i e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -(2y_i-1)\mathbf{x}_i(1-P(y_i|\mathbf{x}_i))$$

$$b_{t+1} = b_t - \lambda_t \times \nabla_b g(b_t, \mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla_{\mathbf{w}} g(b_t, \mathbf{w}_t)$$

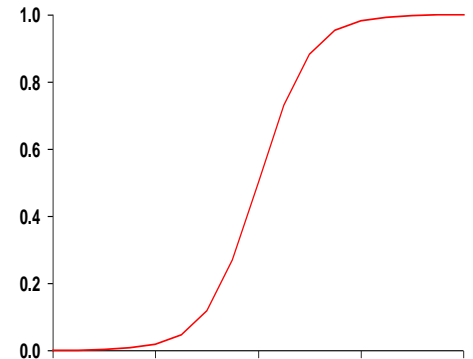
Logistic regression classifier

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}$$

$$\mathbf{x} \in R^m$$

$$y \in \{-1, +1\}$$

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(b+\mathbf{w}^T\mathbf{x})}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$



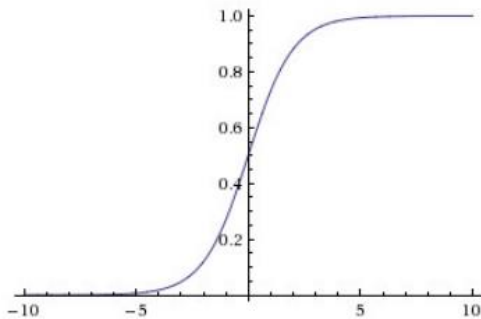
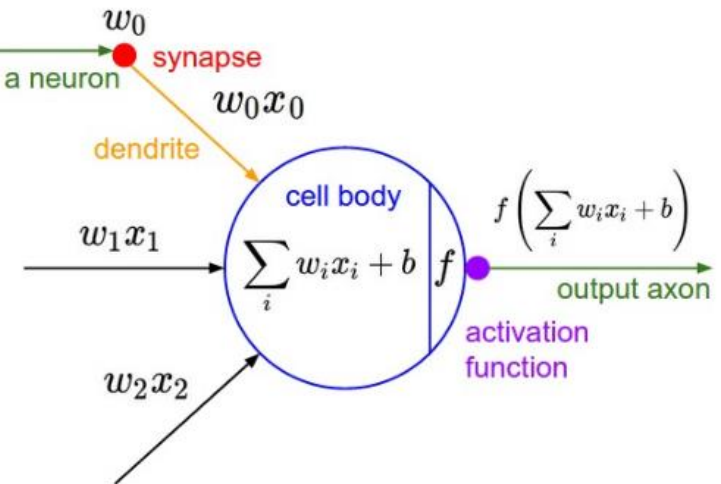
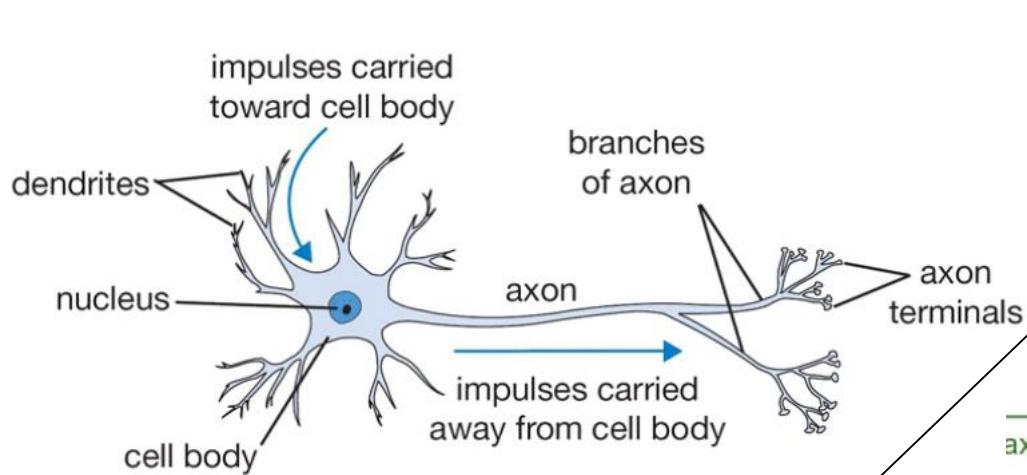
Pros:

1. It is well-normalized.
2. Easy to turn into probability.
3. Easy to implement.

Cons:

1. Indirect loss function.
2. Dependent on good feature set.
3. Weak on feature selection.

Perceptron



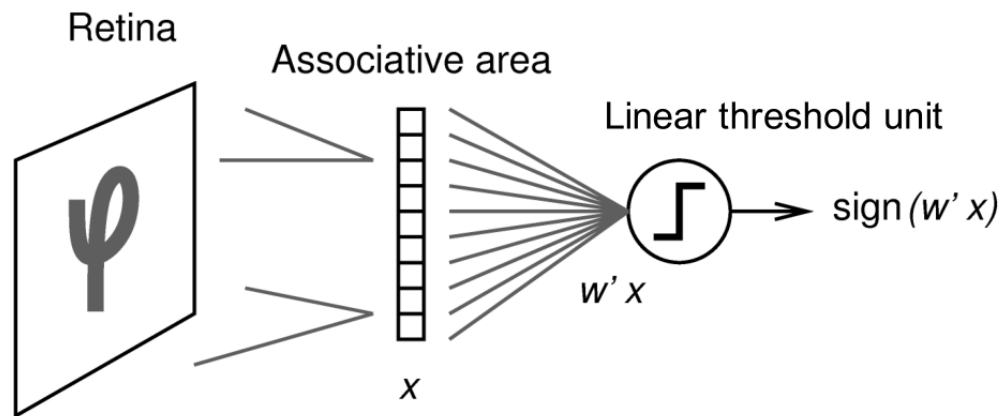
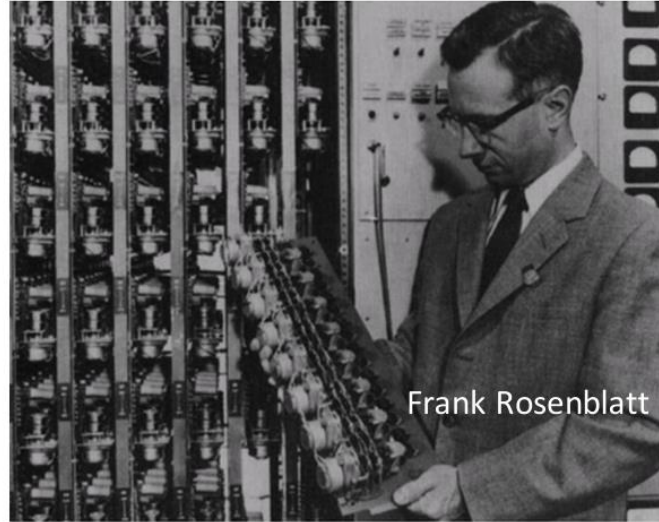
Sigmoid function
$$f(x) = \frac{1}{1+e^{-x}}$$

Perceptron

Let's look at a very simple form.

Slides modified from Jake Olson's lecture.

Perceptron

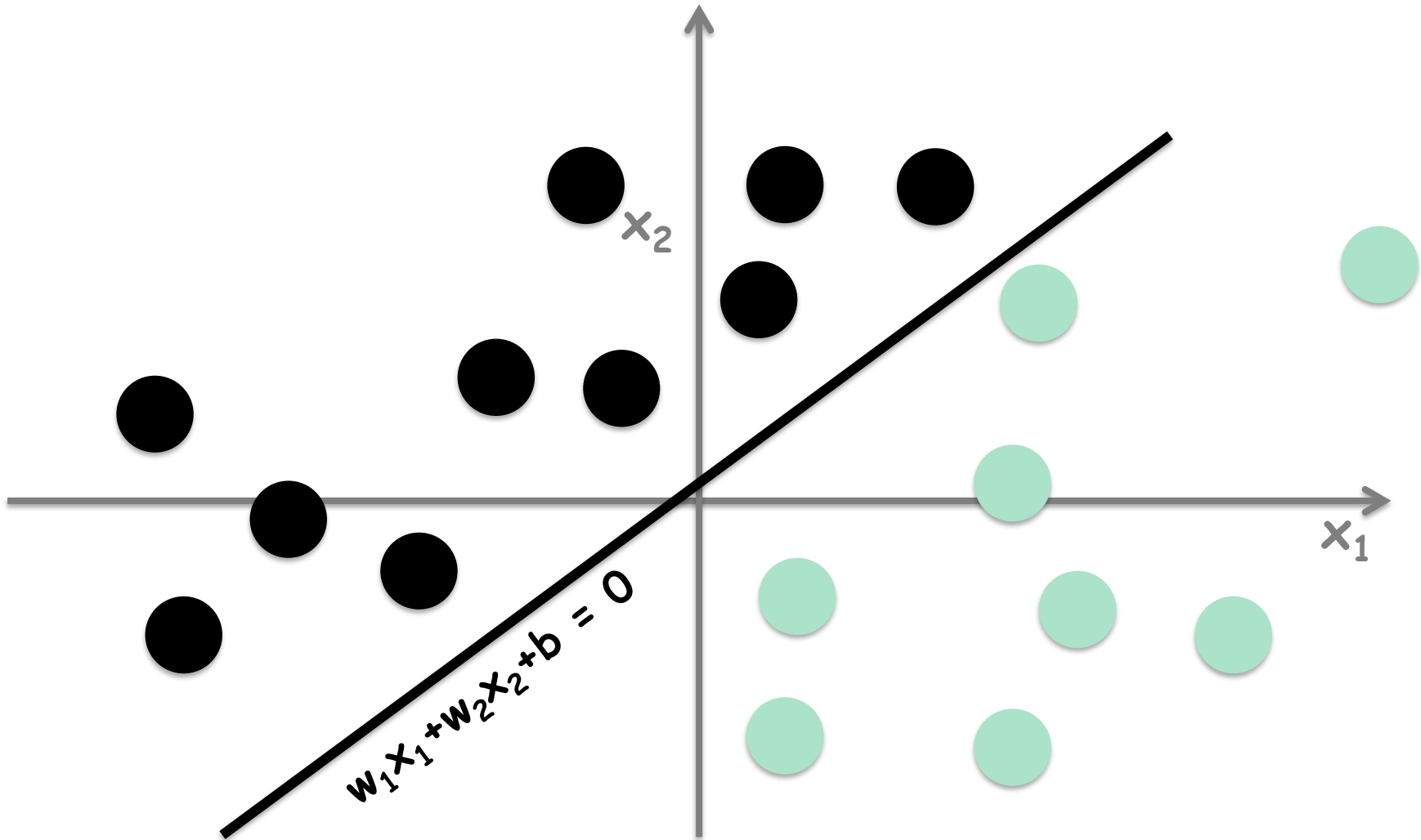


Supervised learning of the weights w using the Perceptron algorithm.

Linear classifier

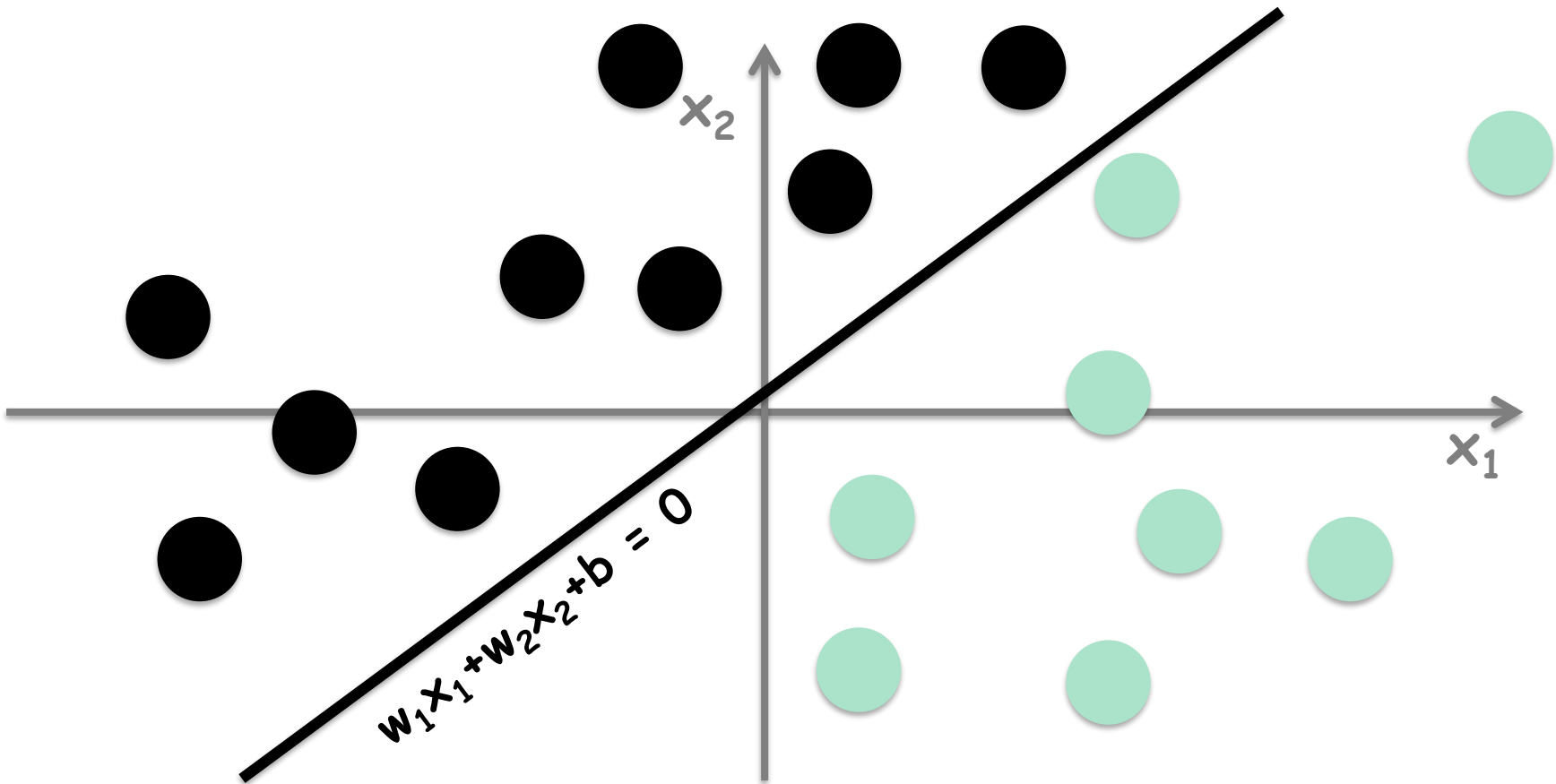
Find: $\arg \min_{\mathbf{w}} \sum_{i=1}^n \mathbf{1}(y_i \neq \text{sign}(w_1 x_{i1} + w_2 x_{i2} + b))$

Linear decision boundary: $w_1 x_1 + w_2 x_2 + b = 0$



Linear classifier

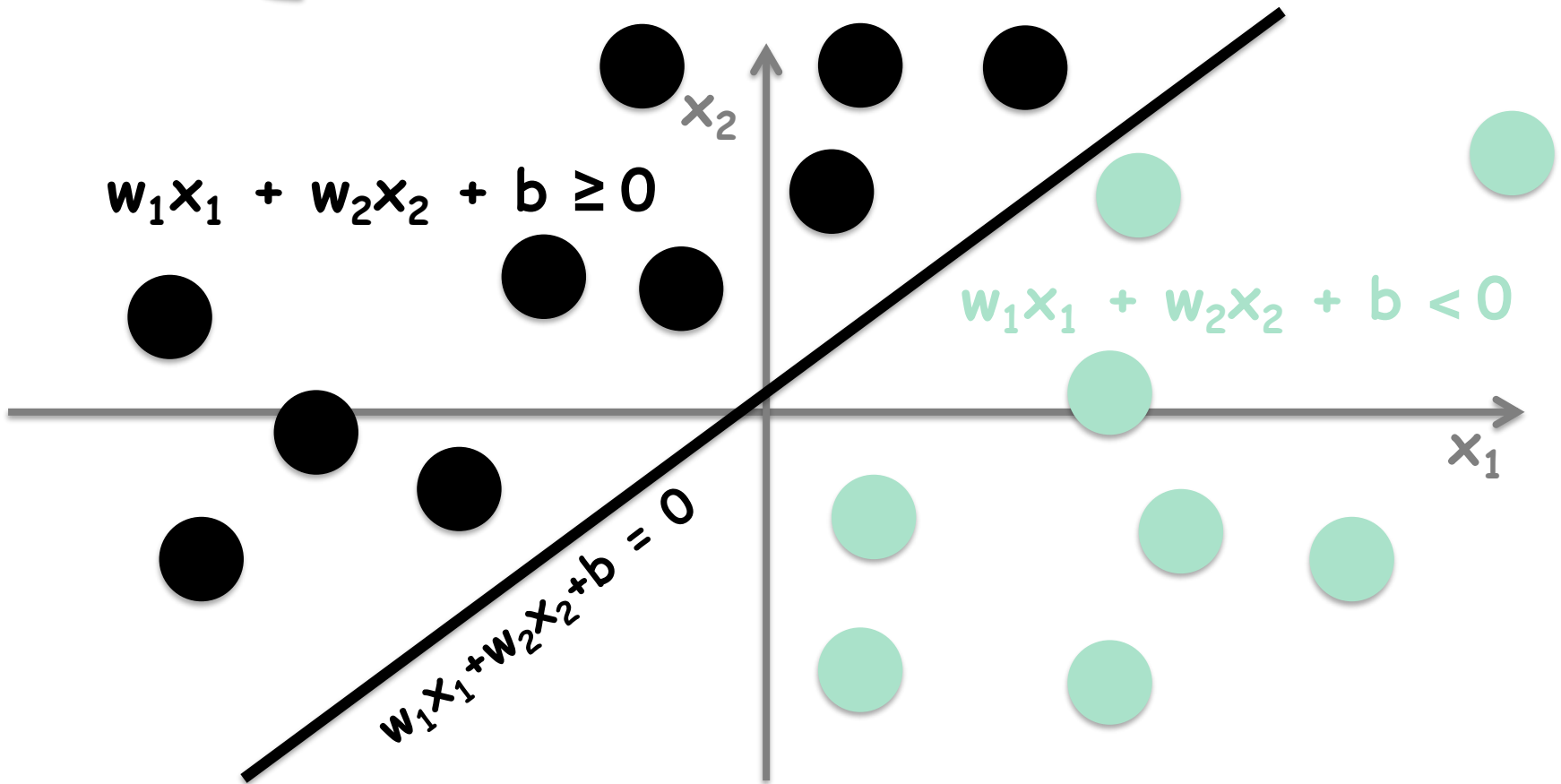
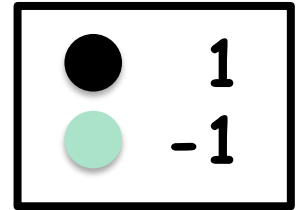
Linear decision boundary: $w_1x_1 + w_2x_2 + b = 0$



Linear classifier

Linear decision boundary and a classification

$$f(x) = \begin{cases} 1 & w_1x_1 + w_2x_2 + b \geq 0 \\ -1 & w_1x_1 + w_2x_2 + b < 0 \end{cases}$$



Perceptron Learning Algorithm

- Initialize the weights (however you choose)
 - $w_1x_1 + w_2x_2 + b$ (initialize w_1 , w_2 , and b)
- Step 1: Choose a data point.
- Step 2: Compute the model output for the datapoint.
- Step 3: Compare model output to the target output.
 - If correct classification, go to Step 5!
 - If not, go to Step 4.
- Step 4: Update weights using perceptron learning rule. Start over on Step 1 with the first data point.

Or

$$W_{new} = W_{old} + (target_i - output_i)x_i$$
$$b_{new} = b_{old} + (target_i - output_i)$$

$$W_{new} = W_{old} + \lambda(target_i - output_i)x_i$$
$$b_{new} = b_{old} + \lambda(target_i - output_i)$$

- Step 5: Go to the next data point. If you have gone through them all, you have found the solution!

Initialize the weights

- Choose randomly – but start the weights small!

- We'll choose:

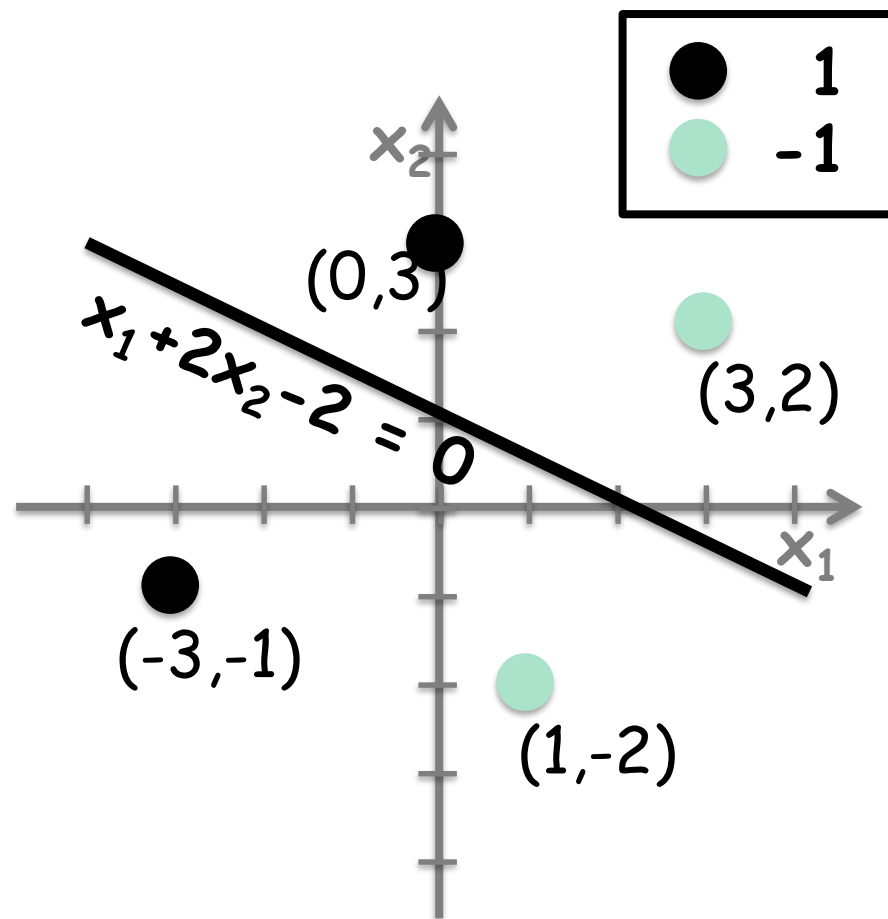
- $w_1 = 1$
- $w_2 = 2$
- $b = -2$

- Generally:

- $w_1x_1 + w_2x_2 + b = 0$

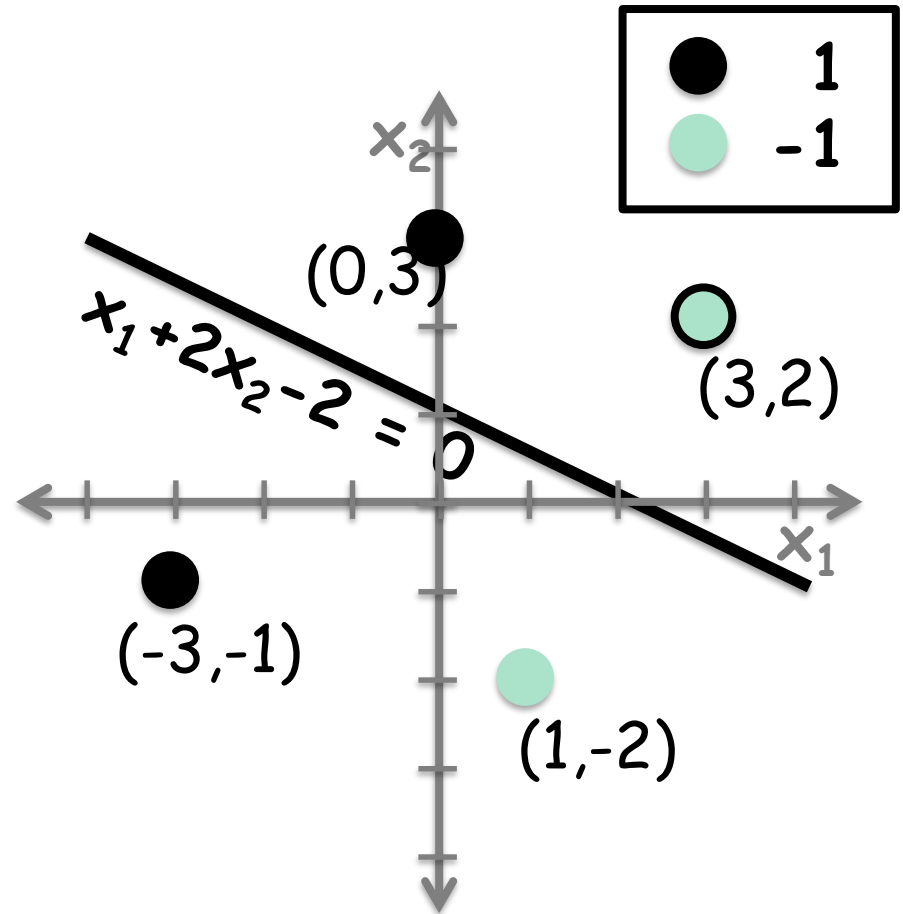
- Applying our values:

- $x_1 + 2x_2 - 2 = 0$



Step 1: Choose a point

- Choose randomly (or sequentially), but remember which you choose!



Step 2: Compute the model output

Inputs

$$\mathbf{x}_1 = 3$$

$$\mathbf{x}_2 = 2$$

$$\text{output} = 1$$

Use the inputs and the linear boundary equation to find the “net activity.”

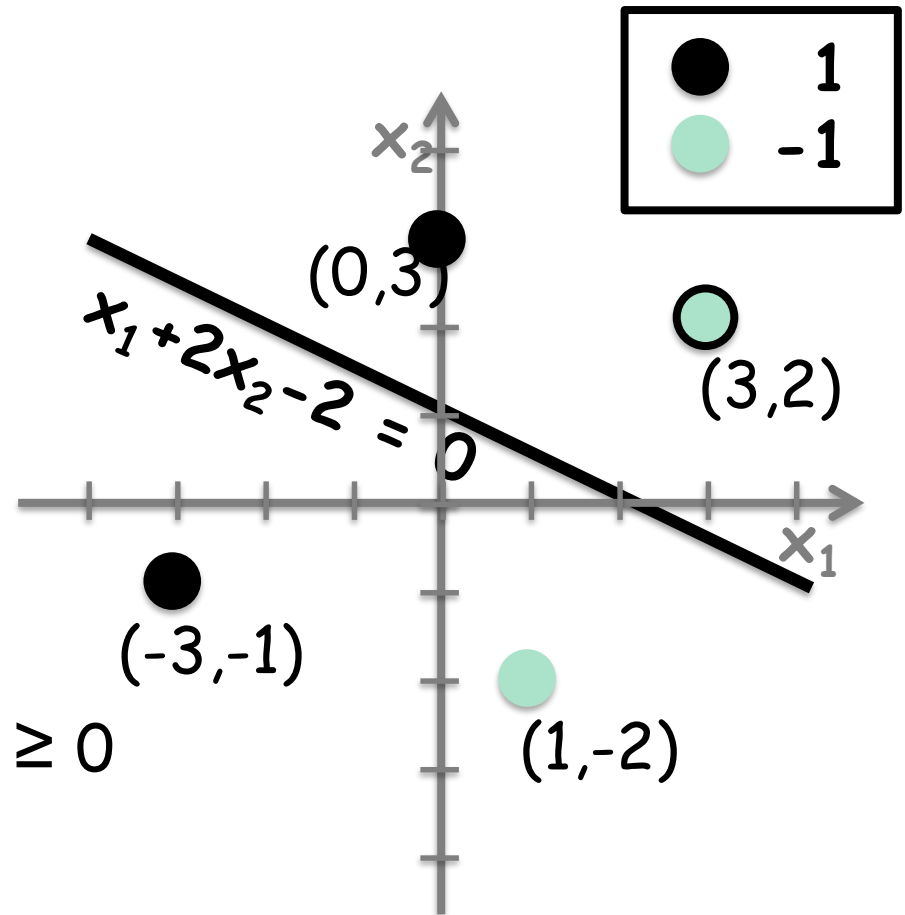
$$\text{net} = \mathbf{x}_1 + 2\mathbf{x}_2 - 2$$

$$\text{net} = (3) + 2(2) - 2$$

$$\text{net} = 5$$

Find the model output using the classification (step) *activation function*.

$$\text{output} = f(\text{net}) = \begin{cases} 1 & \text{net} \geq 0 \\ -1 & \text{net} < 0 \end{cases}$$



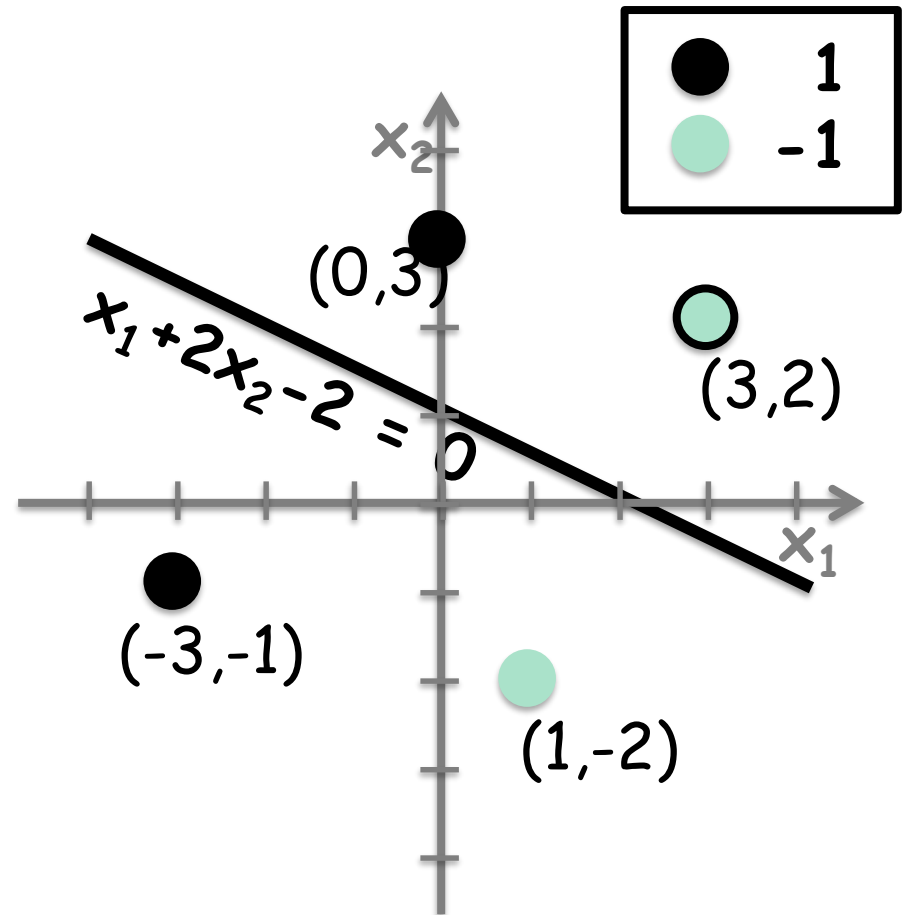
Step 3: Compare the model output and target values

Does the model output equal the target value?

If yes, do another point until you have done them all!

If no, update the weights.

output = 1
target = -1
failure :(



Step 4: Update weights using perceptron learning rule.

- We need to improve! How?
 - Move the line to a better spot!

- How?

$$w_{new} = w_{old} + (target_i - output_i) x_i$$

$$b_{new} = b_{old} + (target_i - output_i)$$

For this example:

$$w_1 = 1 + (-1 - 1) * 3 = -5$$

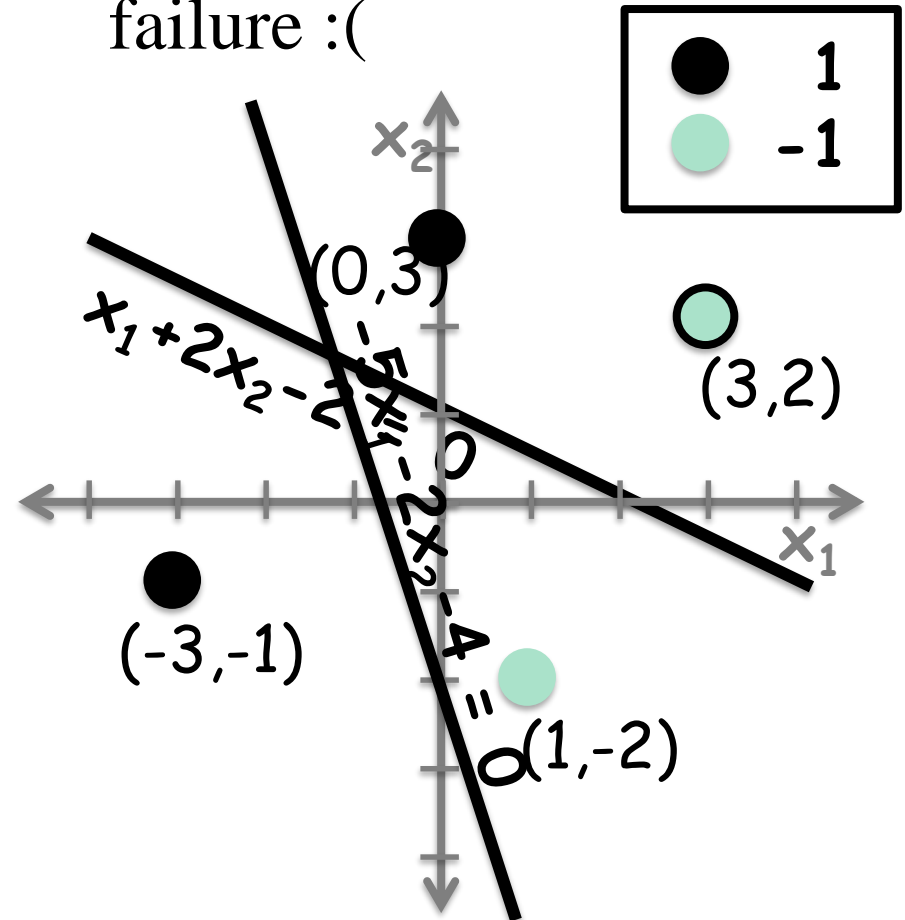
$$w_2 = 2 + (-1 - 1) * 2 = -2$$

$$b = -2 + (-1 - 1) = -4$$

output = 1

target = -1

failure :(



Perceptron Learning Algorithm

- Initialize the weights (however you choose)
 - $w_1x_1 + w_2x_2 + b$ (initialize w_1 , w_2 , and b)
- Step 1: Choose a data point.
- Step 2: Compute the model output for the datapoint.
- Step 3: Compare model output to the target output.
 - If correct classification, go to Step 5!
 - If not, go to Step 4.
- Step 4: Update weights using perceptron learning rule. Start over on Step 1 with the first data point.

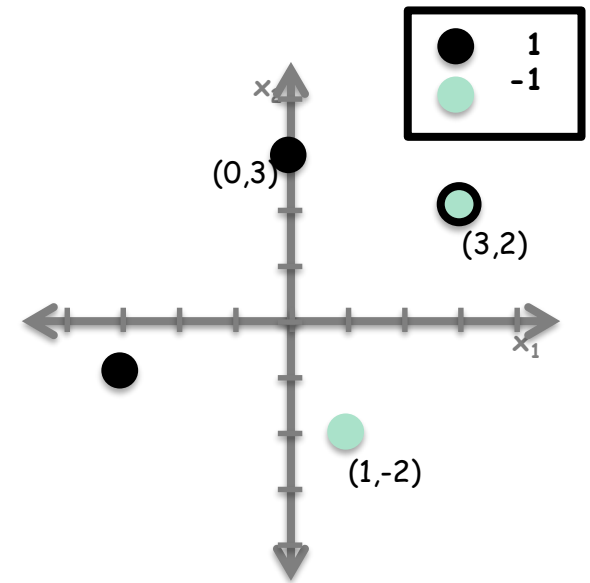
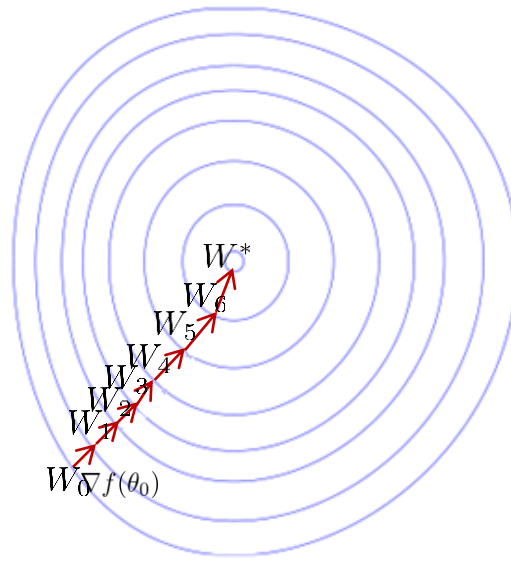
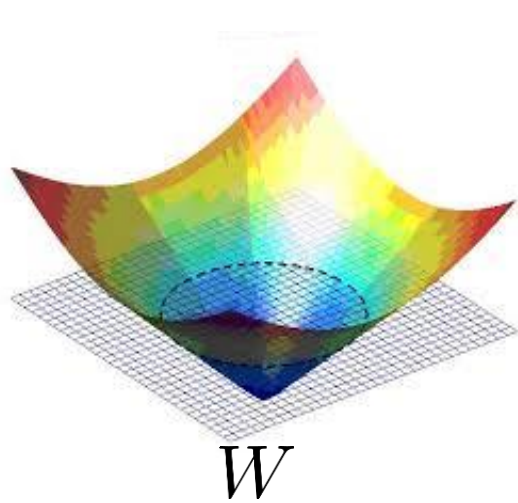
Or

$$W_{new} = W_{old} + (target_i - output_i)x_i$$
$$b_{new} = b_{old} + (target_i - output_i)$$

$$W_{new} = W_{old} + \lambda(target_i - output_i)x_i$$
$$b_{new} = b_{old} + \lambda(target_i - output_i)$$

- Step 5: Go to the next data point. If you have gone through them all, you have found the solution!

Perceptron training is a special gradient descent algorithm



$$W_{t+1} \leftarrow W_t - \nabla f(W_t)$$

$$f(W_t) = (f(\mathbf{x}_i) - y_i)\mathbf{x}_i$$

The XOR problem that kills the Perceptron algorithm

The XOR operator: \oplus

Positive (+1) \oplus Negative (-1) = Negative

Positive (+1) \oplus Positive (+1) = Positive

Negative (-1) \oplus Positive (+1) = Negative

Negative (-1) \oplus Negative (-1) = Positive

