# Optimization

Mladen Kolar (mkolar@chicagobooth.edu)

# Optimization in Machine Learning

Often we have some conceptual problem that we want to solve.

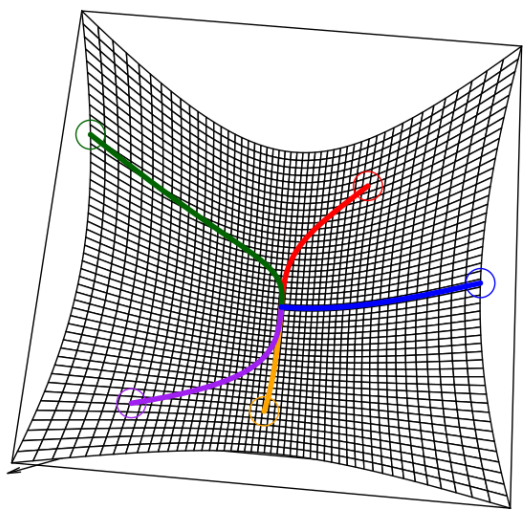
We translate this problem into an optimization problem of the form

$$\min_{x \in D} \; f(x)$$

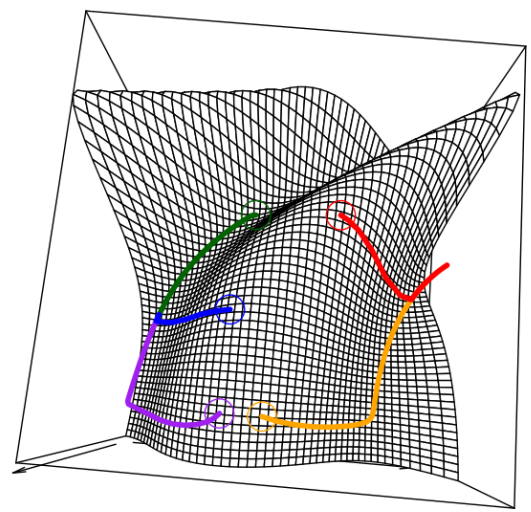
What are examples from our lectures?

# Local minima are global minima

For convex optimization problems, local minima are global minima



Convex



Nonconvex

# Minimization procedures

Derivative free minimization

- require $f(x)$ only
- useful when computing $\nabla f(x)$ is very expensive

Gradient descent and other first order methods

- require $f(x)$ and $\nabla f(x)$
- useful when dealing with large amounts of data
- many iterations to get high accuracy solution, but each iteration is cheap (We do not care about high accuracy. Why?)

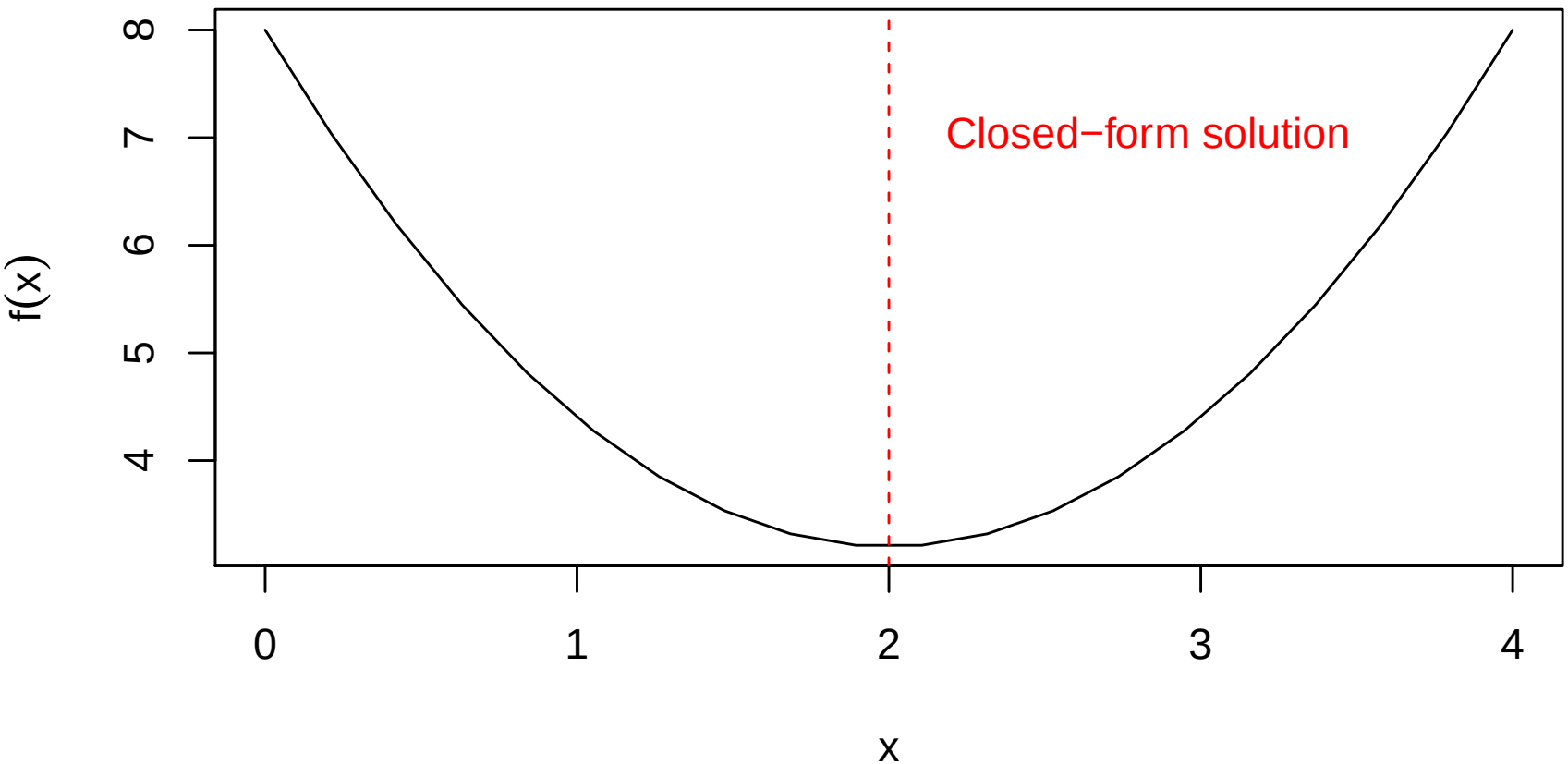
Second order methods (Newton's method)

- require $f(x)$, $\nabla f(x)$, and $\Delta f(x)$
- few iteration to find minimum, but each iteration can be expensive
- R packages (that I have seen) implement this

# Gradient descent

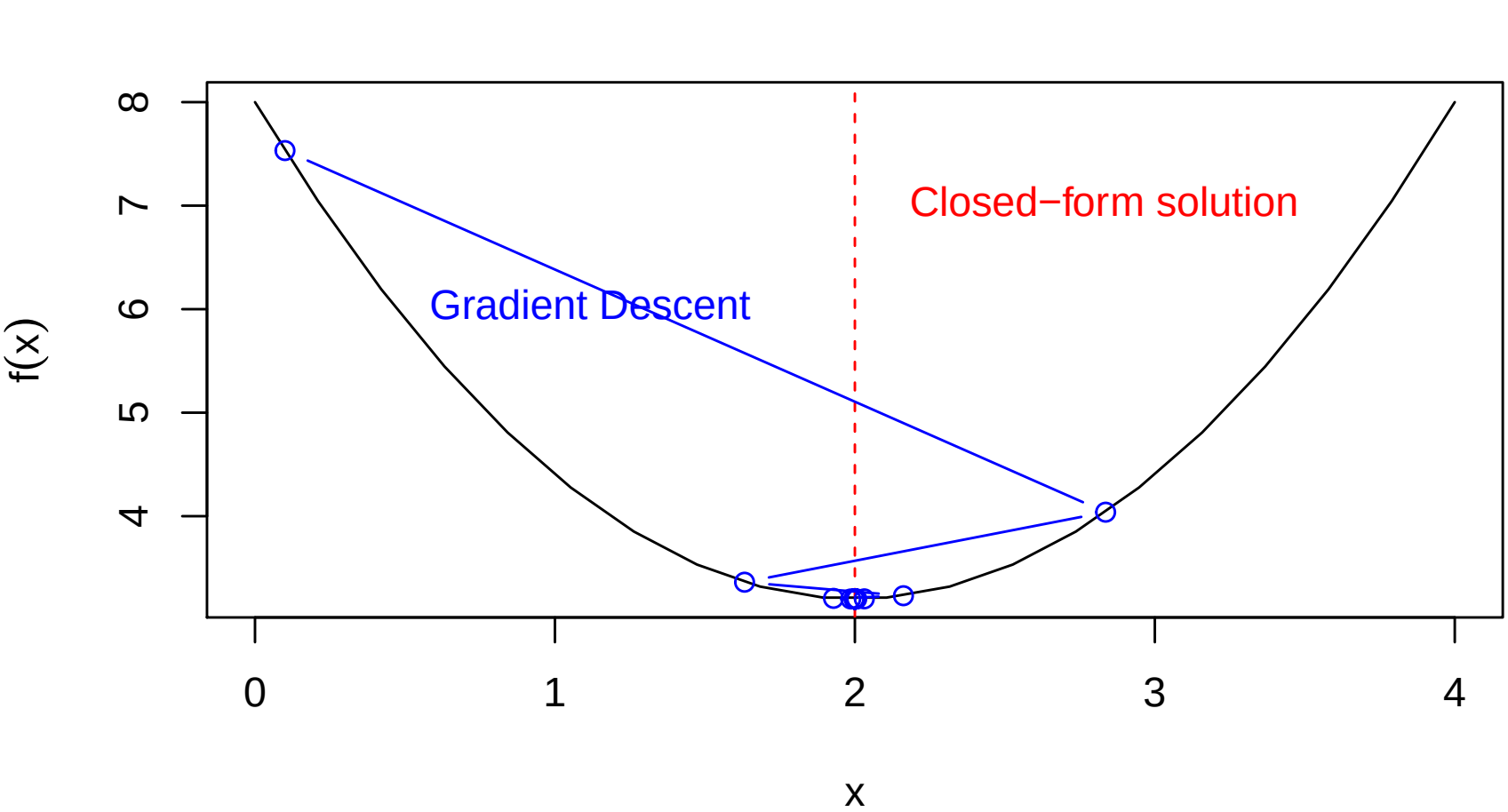
Suppose that we want to find $x$ that minimizes the following function

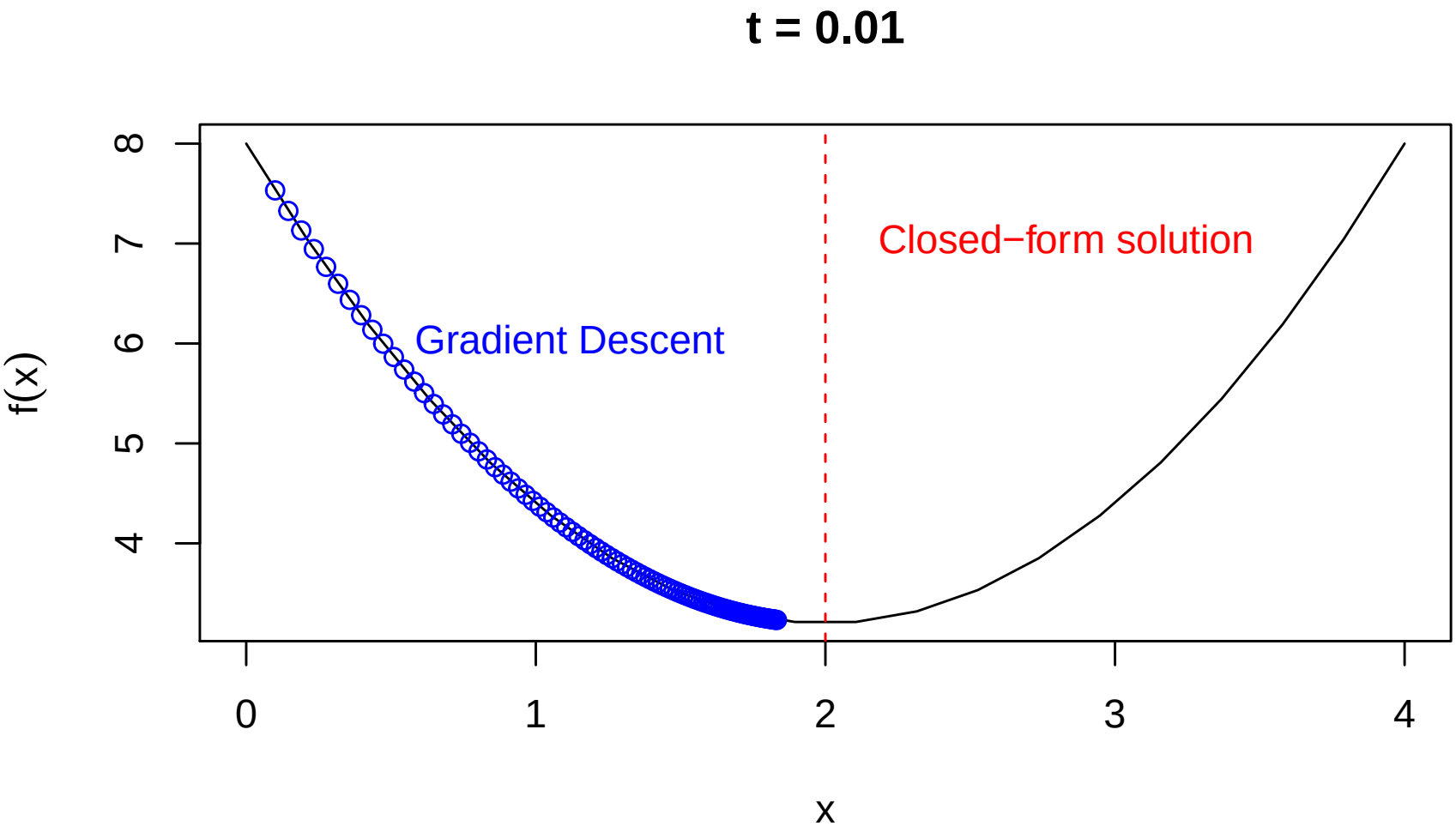
$$f(x) = 1.2(x-2)^2 + 3.2$$

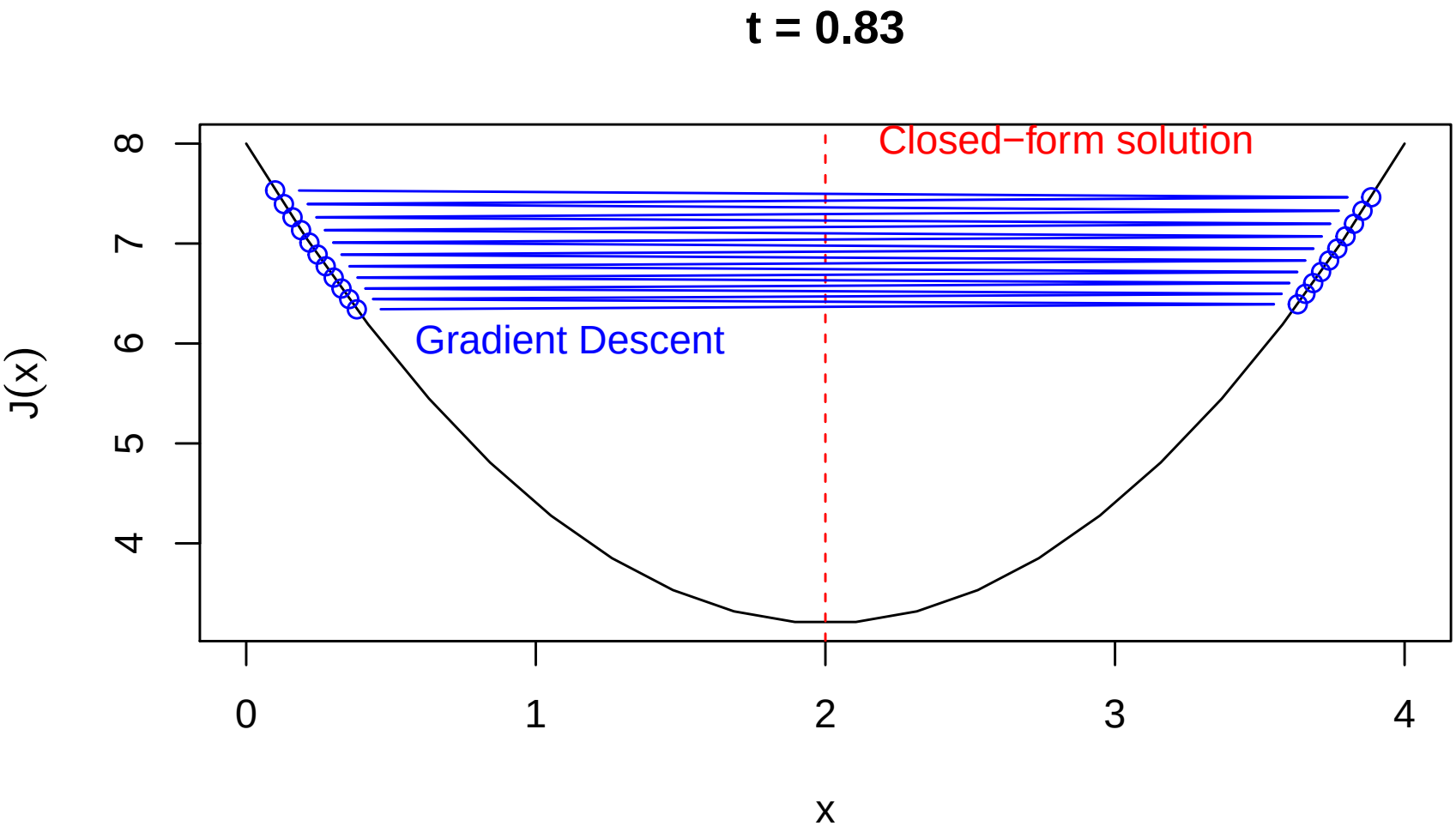



In general, we cannot find a closed form solution, but can compute $\partial f(x)$.

Set $x^0 = 0$. Iteratively update $x^{(k+1)} = x^{(k)} - t \cdot \nabla f(x^{(k)})$.
$t$ is the step size. Also known as the learning rate.




t = 0.01
Closed−form solution
Gradient Descent
f(x)
4
5
6
7
8
0
1
2
3
4
x


t = 0.83
Closed−form solution
Gradient Descent
J(x)
4
5
6
7
8
0
1
2
3
4
x

# Gradient descent

Consider unconstrained, smooth convex optimization

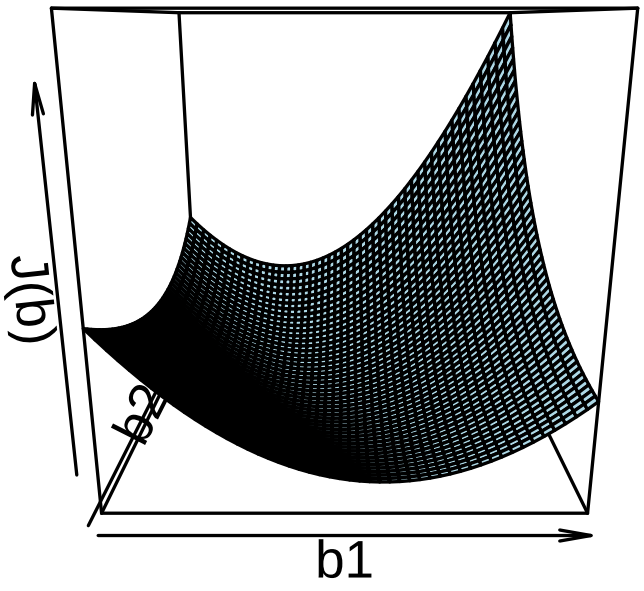
$$\min_x \; f(x)$$

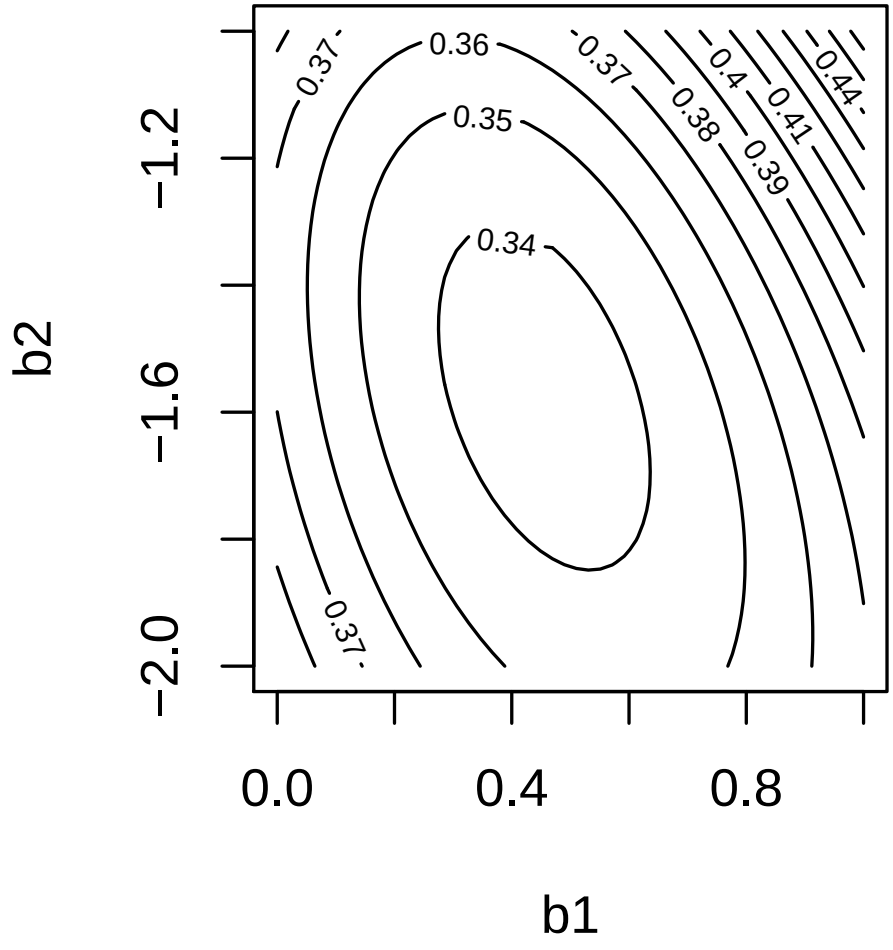
Gradient descent: choose initial point $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f\left(x^{(k)}\right), \quad k = 1, 2, 3, \ldots$$

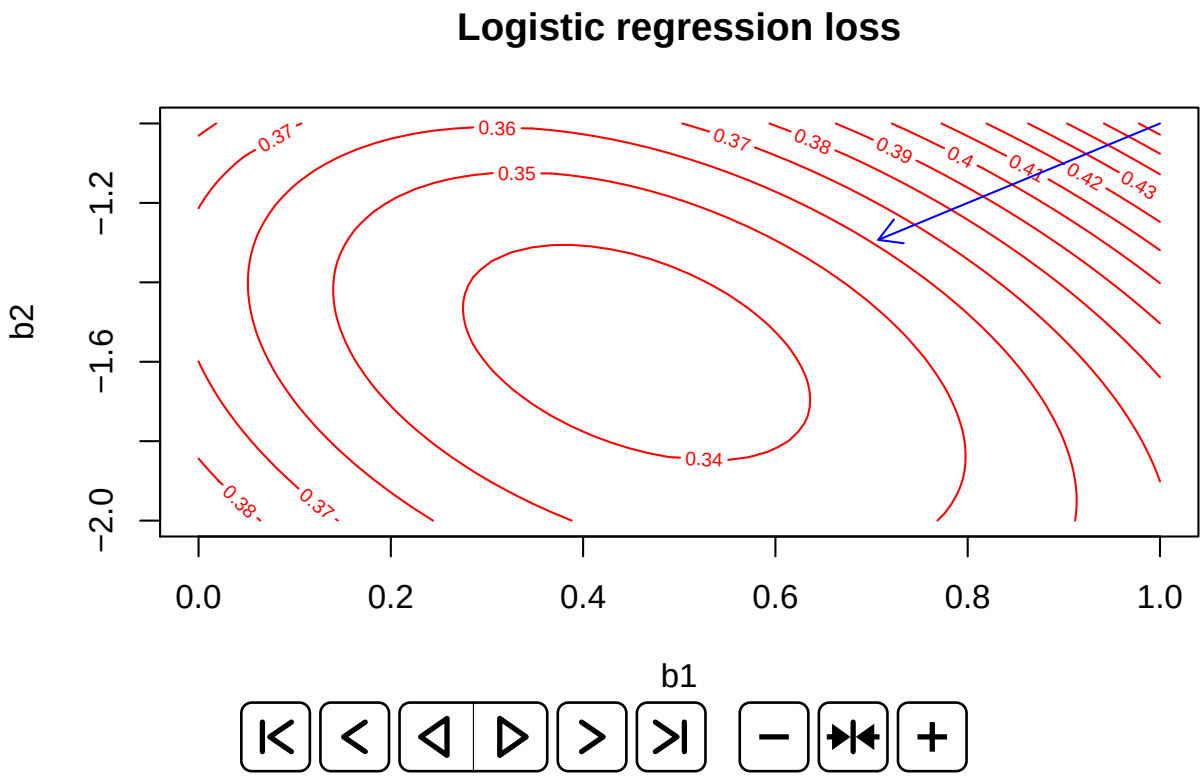
Stop at some point

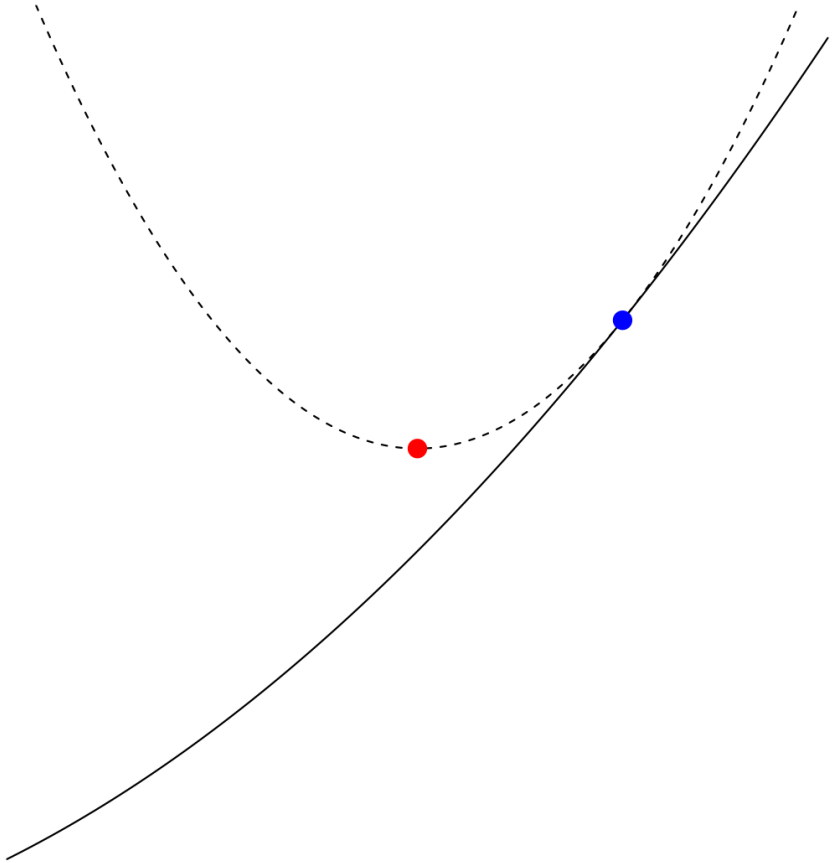
# Gradient descent



# Gradient descent





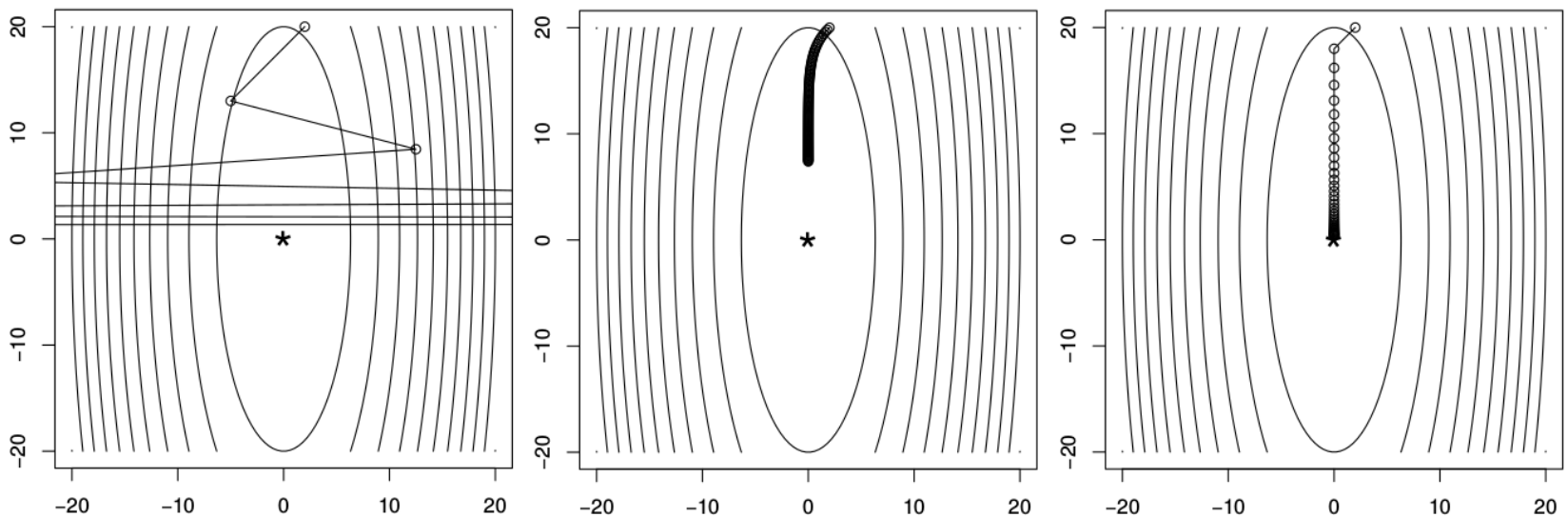
Blue point is $x$, red point is

$$x^+ = \underset{y}{\operatorname{argmin}} \; f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} \|y - x\|_2^2$$

Choosing an appropriate step size is extremely important!

Fixed step size

- our iterates can diverge if the step is too big
- if the step size is too small, very slow convergence



# Backtracking line search DETAILS!

One way to adaptively choose the step size is to use backtracking line search:

- First fix parameters $0 < \beta < 1$ and $0 < \alpha \leq 1/2$
- At each iteration, start with $t = 1$, and while

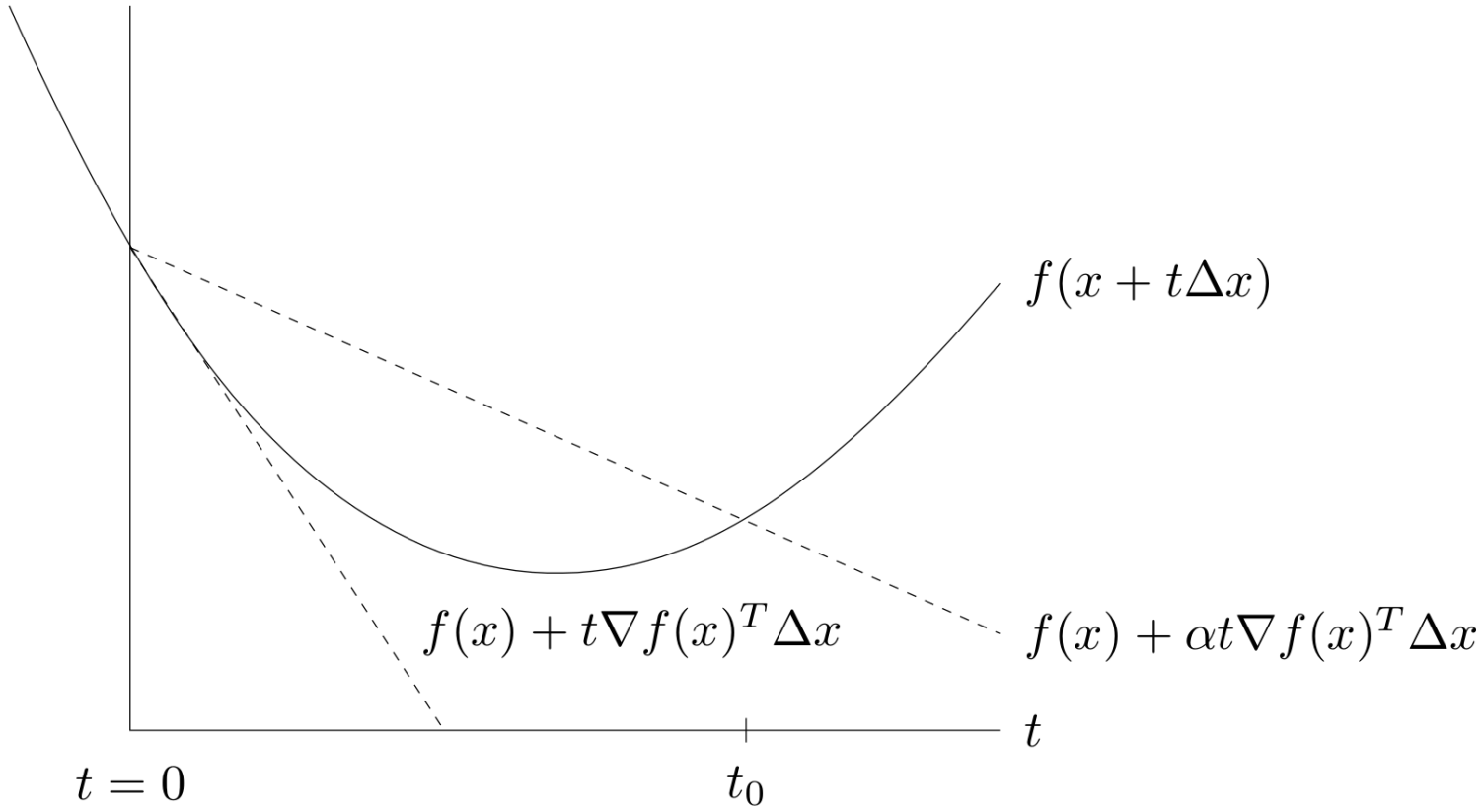
$$f(x - t\nabla f(x)) > f(x) - \alpha t \|\nabla f(x)\|_2^2$$

shrink $t = \beta t$. Else perform gradient descent update

$$x^+ = x - t\nabla f(x)$$

Simple and tends to work well in practice (further simplification: just take $\alpha = 1/2$)

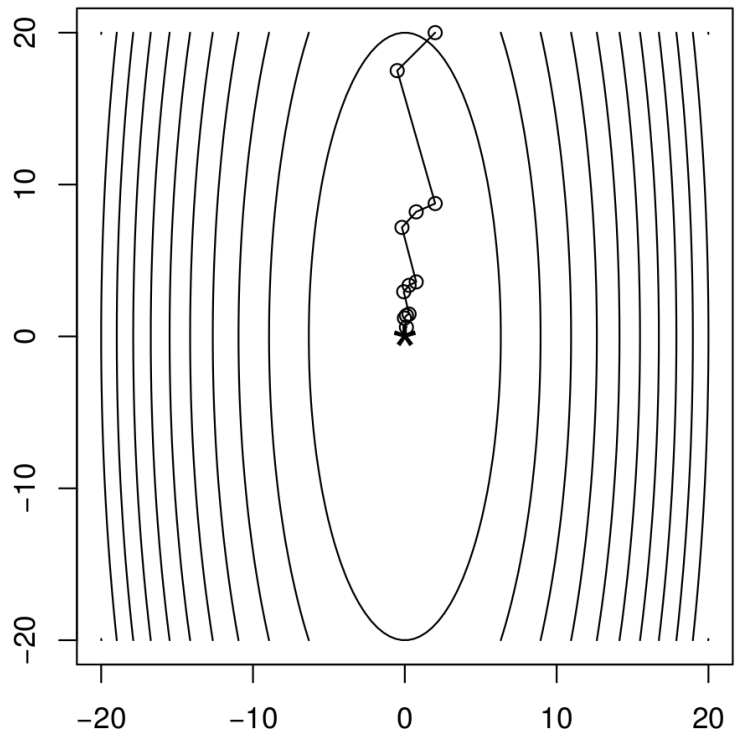
# Backtracking line search DETAILS!




For us $\Delta x = -\nabla f(x)$

# Backtracking line search

Backtracking picks up roughly the right step size (12 outer steps, 40 steps total):




Here $\alpha = \beta = 0.5$

# Practicalities

Stopping rule: stop when $\|\nabla f(x)\|_2$ is small

- Recall $\nabla f(x^*) = 0$ at solution $x^*$
- If $f$ is strongly convex with parameter $m$, then

$$\|\nabla f(x)\|_2 \leq \sqrt{2m\epsilon} \implies f(x) - f^* \leq \epsilon$$

Pros and cons of gradient descent:

- Pro: simple idea, and each iteration is cheap
- Pro: very fast for well-conditioned, strongly convex problems
- Con: often slow, because interesting problems aren't strongly convex or well-conditioned
- Con: can't handle nondifferentiable functions

# Example: gradient descent for logistic regression

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$ for $i = 1, \ldots, n$, consider the logistic regression loss:

$$f(\beta) = \sum_{i=1}^{n} \underbrace{\left(-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta))\right)}_{f_i(\beta)} = \sum_{i=1}^{n} f_i(\beta)$$

The gradient is

$$\nabla f(\beta) = \sum_{i=1}^{n} (y_i - p_i(\beta)) x_i$$

where

$$\begin{aligned} p_i(\beta) &= P(Y = 1 \mid x_i, \beta) \\ &= \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta)), \quad i = 1, \ldots, n. \end{aligned}$$

# Example: gradient descent for logistic regression

Initialize $\beta = (0, \ldots, 0)$
**repeat**
  Let $g = (0, \ldots, 0)$ be the gradient vector
  **for** *i = 1:n* **do**
    $p_i = \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta))$
    $\text{error}_i = p_i - y_i$
    $g = g + \text{error}_i \cdot x_i$
  **end**
  $\beta = \beta - \eta \cdot g/n$
**until** *until convergence*;

Note that algorithm uses all observations to compute the gradient.
This approach is called batch gradient descent.

The gradient computation $\nabla f(\beta) = \sum_{i=1}^{n}(y_i - p_i(\beta))x_i$ is doable when $n$ is moderate, but not when $n \approx 500$ million.

- One batch update costs $O(np)$
- One stochastic update costs $O(p)$

So clearly, for example, 10K stochastic steps are much more affordable.

Also, we often take fixed step size for stochastic updates to be $\approx n$ what we use for batch updates. (Why?)

# Stochastic gradient descent for logistic regression

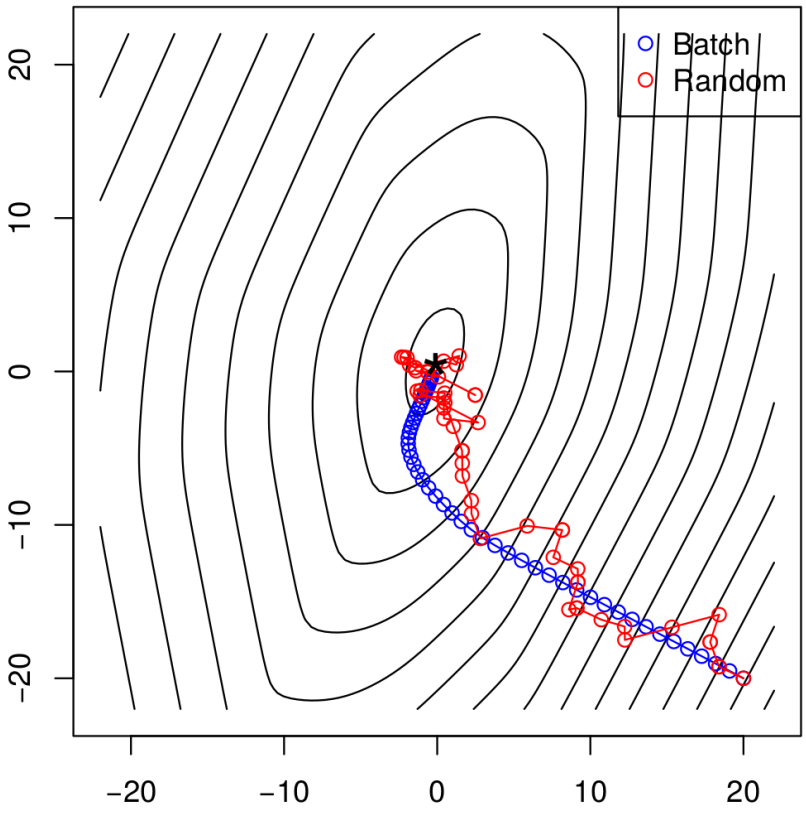
Initialize $\beta = (0, \ldots, 0)$
**repeat**
  Pick observation $i$
  $p_i = \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta))$
  $\text{error}_i = p_i - y_i$
  $\beta = \beta - \eta(\text{error}_i \cdot x_i)$
**until** *until convergence*;

Coefficient $\beta$ is updated after each observation.

In practice, mini batch is often used.

- Compute gradient based on a small subset of observations
- Make update to coefficient vector

The "classic picture":




Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

# Example: Perceptron Revisited

Perceptron update:

$$\beta^{t+1} = \beta^t + \mathbb{1}\left\{y_t(x_t^T \beta^t) \leq 0\right\} \cdot y_t x_t$$

Batch hinge loss minimization update:

$$\beta^{t+1} = \beta^t + \eta \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left\{y_i(x_i^T \beta^t) \leq 0\right\} \cdot y_i x_i$$

Perceptron can be thought of as SGD for hinge loss minimization with step size constant $\eta = 1$ and no regularization.

# Avoiding overfitting using early stopping

Common practice is to stop gradient descent before it reaches optimum

One monitors peformance of the current solution on the validation data

If the performance starts to deteriorate, stop the descent

Why should this work?