Gradient Descent for L2 Penalized Logistic Regr.

$$\min_{w \in \mathbb{R}^M} \underbrace{\frac{1}{2}\lambda w^T w - \sum_{n=1}^{N} \log \text{BernPMF}(t_n | \sigma(w^T \phi(x_n)))}_{\mathcal{L}(w)}$$

input: initial $w \in \mathbb{R}$

input: step size $s_0 \in \mathbb{R}_+$

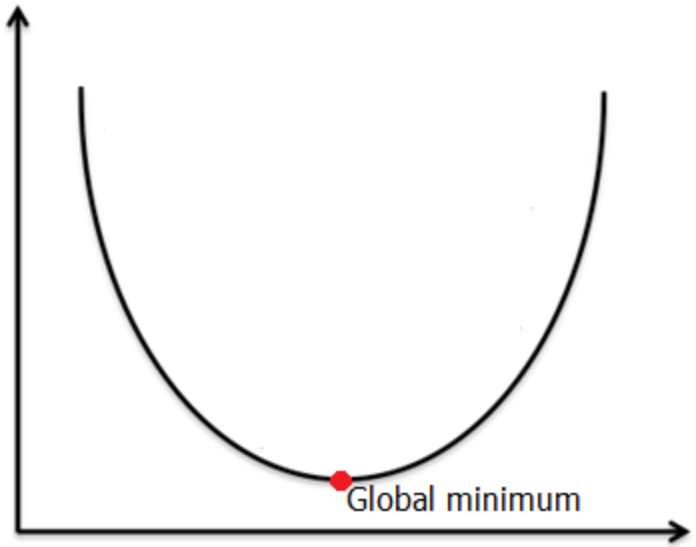
while not converged:

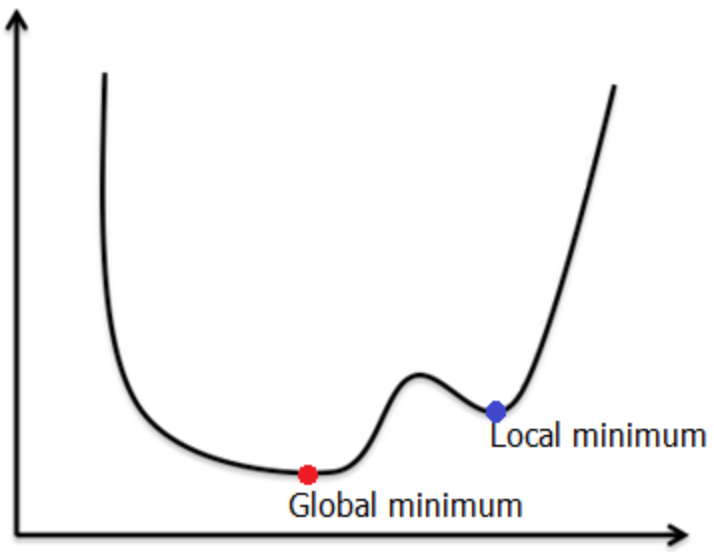
$$w \leftarrow w - s_0 \nabla_w \mathcal{L}(w)$$

You need to specify:

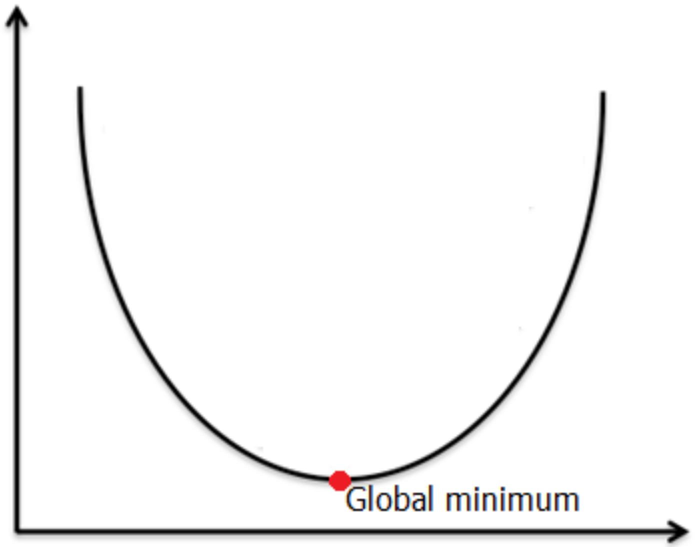
- Max. num iterations *T*
- Step size *s*
- Convergence threshold *d*

Will gradient descent always find same solution?

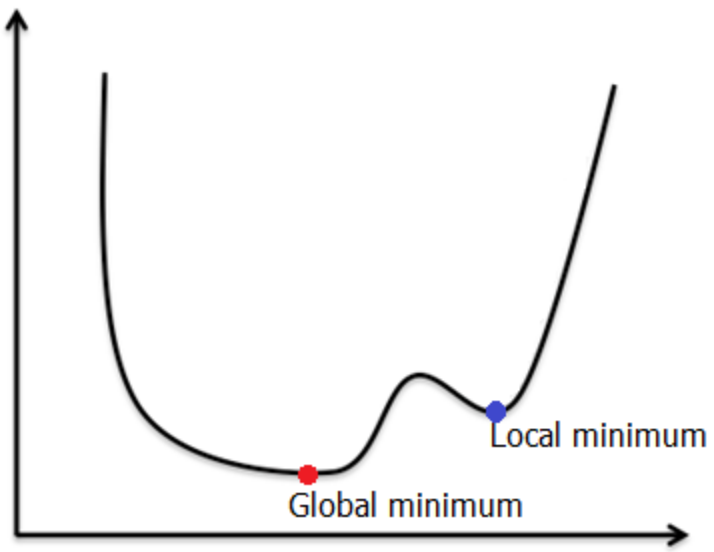




Will gradient descent always find same solution?

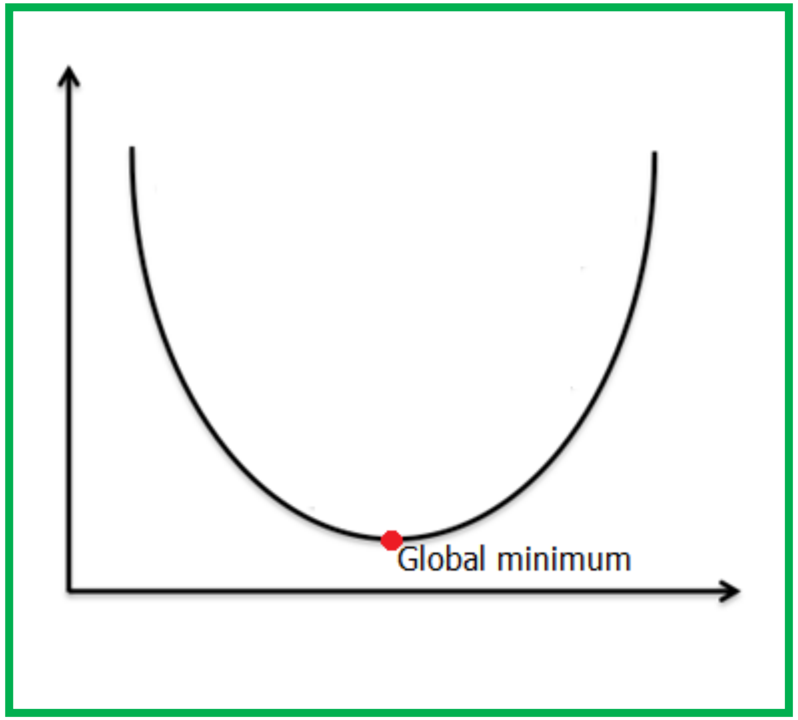


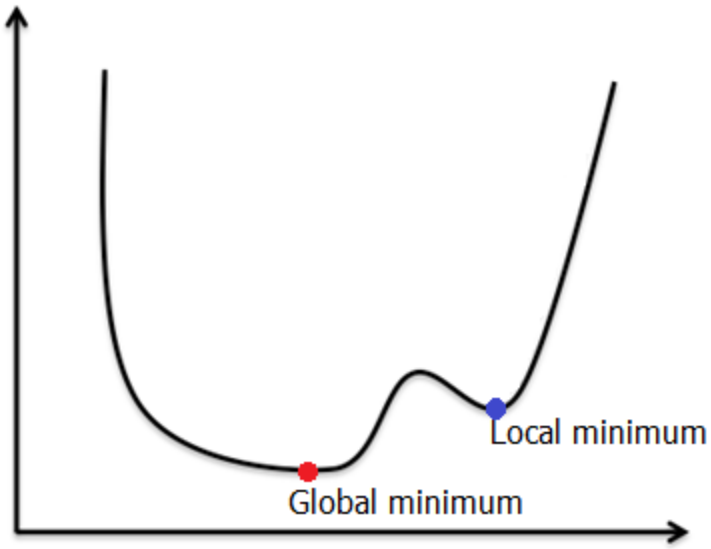
Yes, if loss looks like this



Not if multiple local minima exist

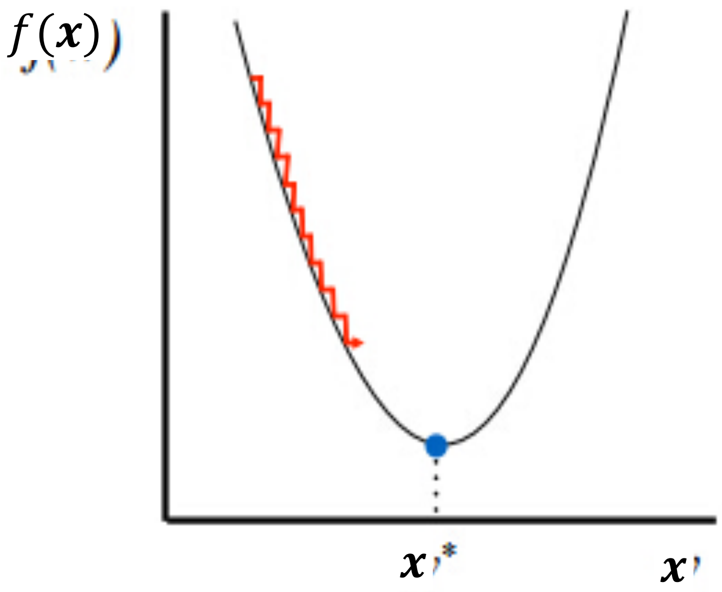
Loss for logistic regression is convex!



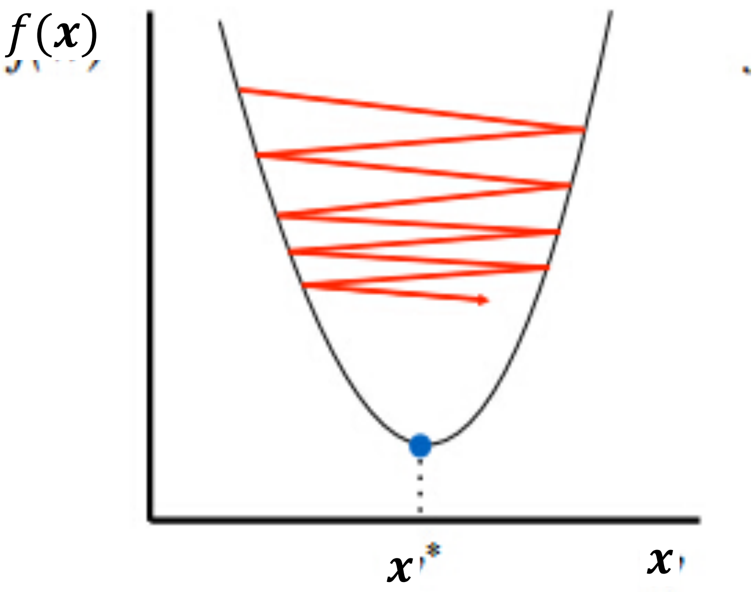


Intuition: 1D gradient descent

Choosing good step size matters!



Too small: converge very slowly



Too big: overshoot and even diverge

Log likelihood vs iterations

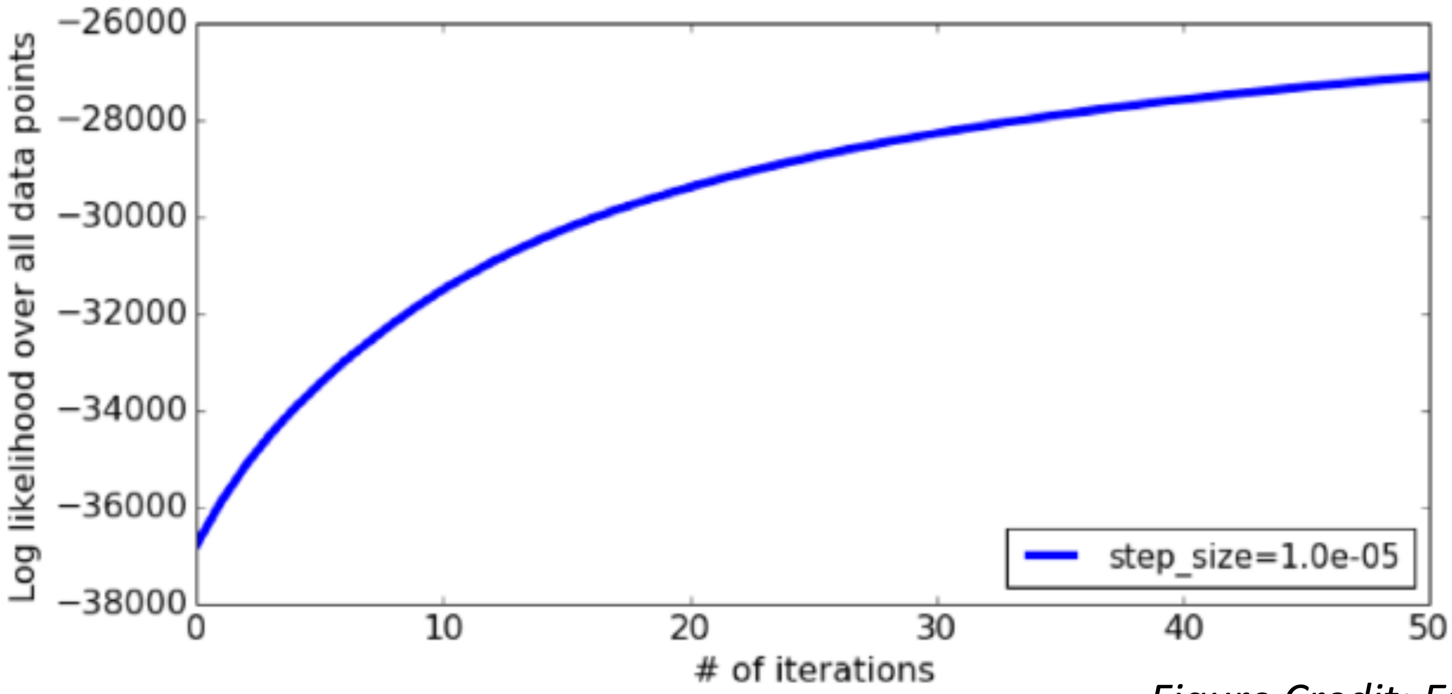


Figure Credit: Emily Fox (UW)

Maximizing likelihood: Higher is better!
(could multiply by -1 and minimize instead)

If step size is too small

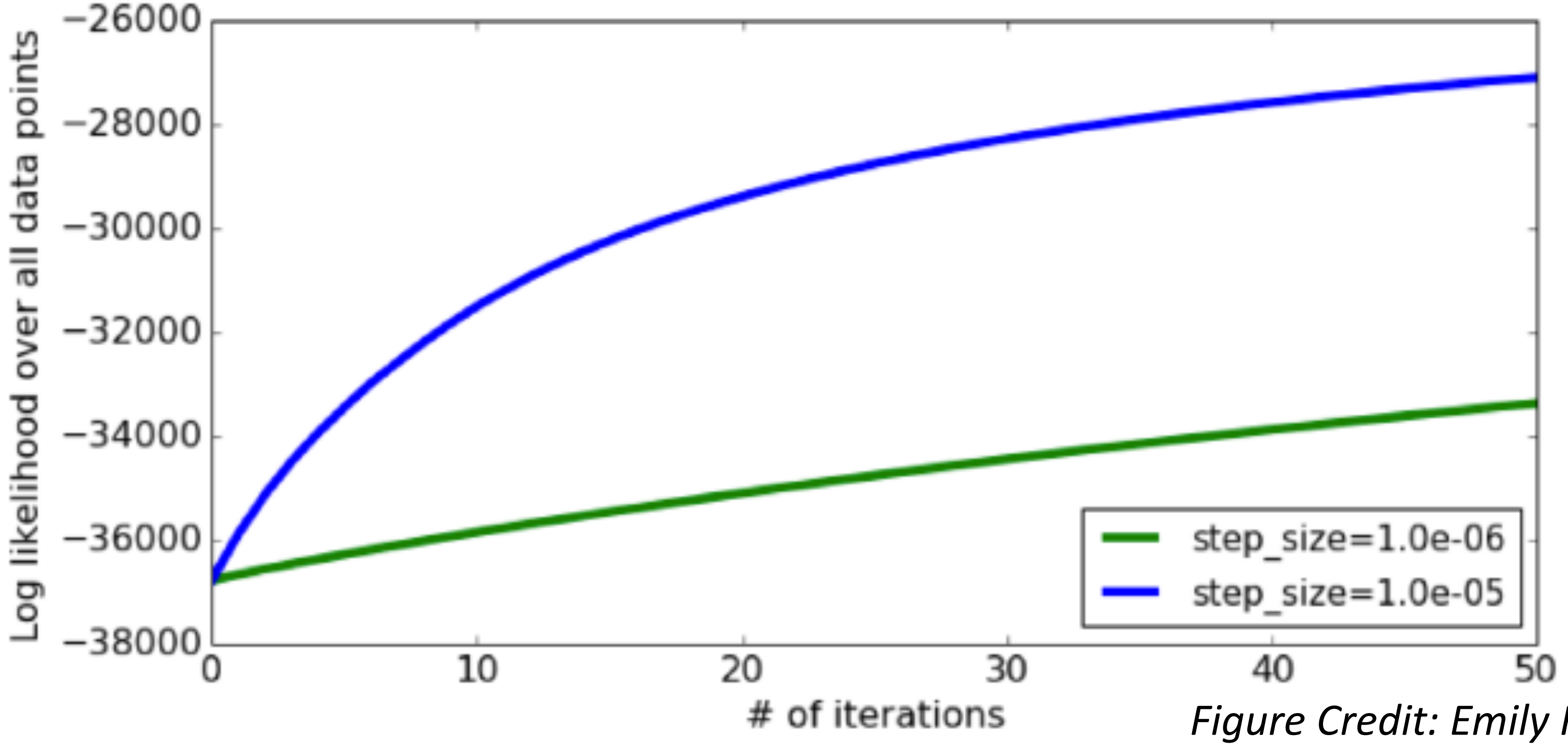


Figure Credit: Emily Fox (UW)

If step size is large

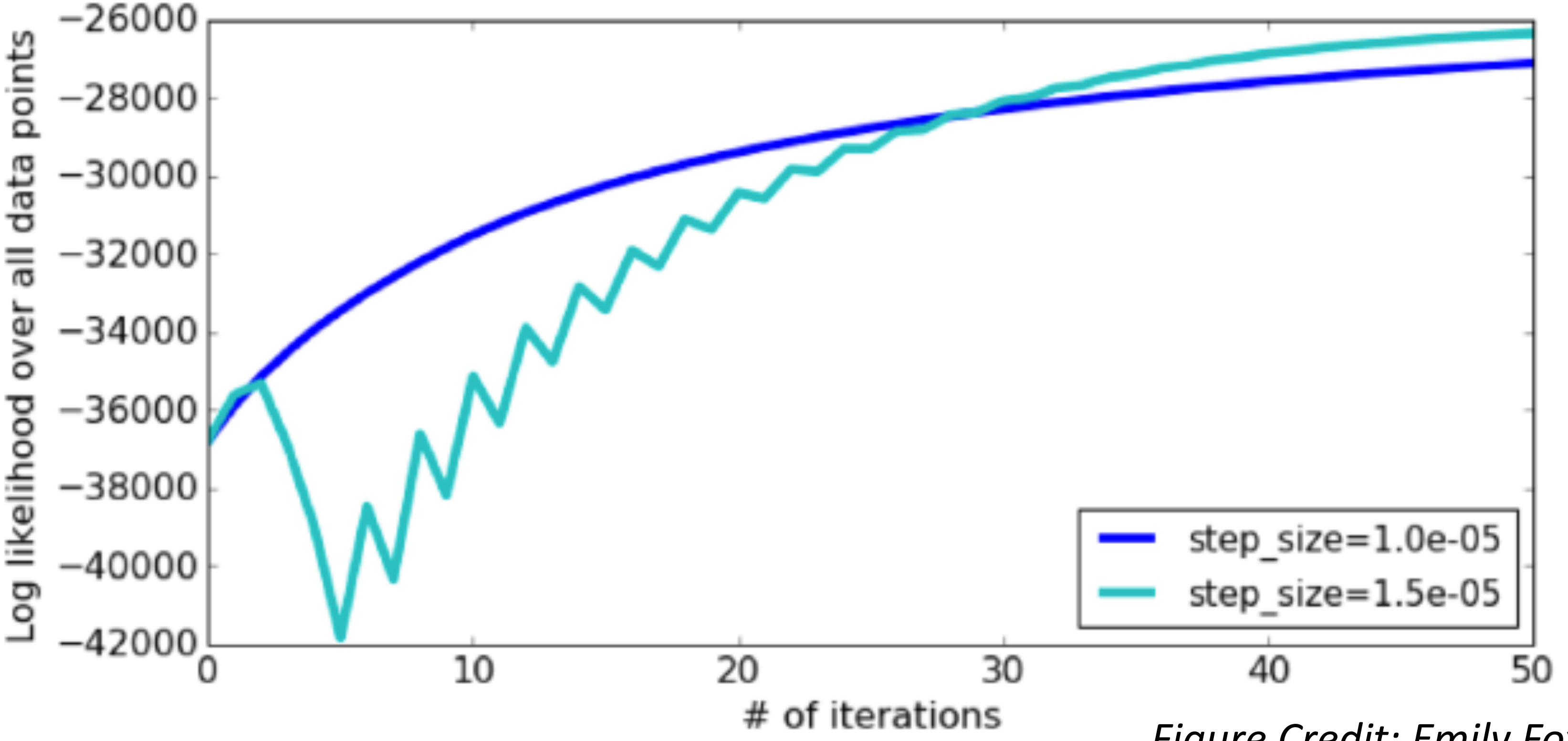


Figure Credit: Emily Fox (UW)

If step size is too large

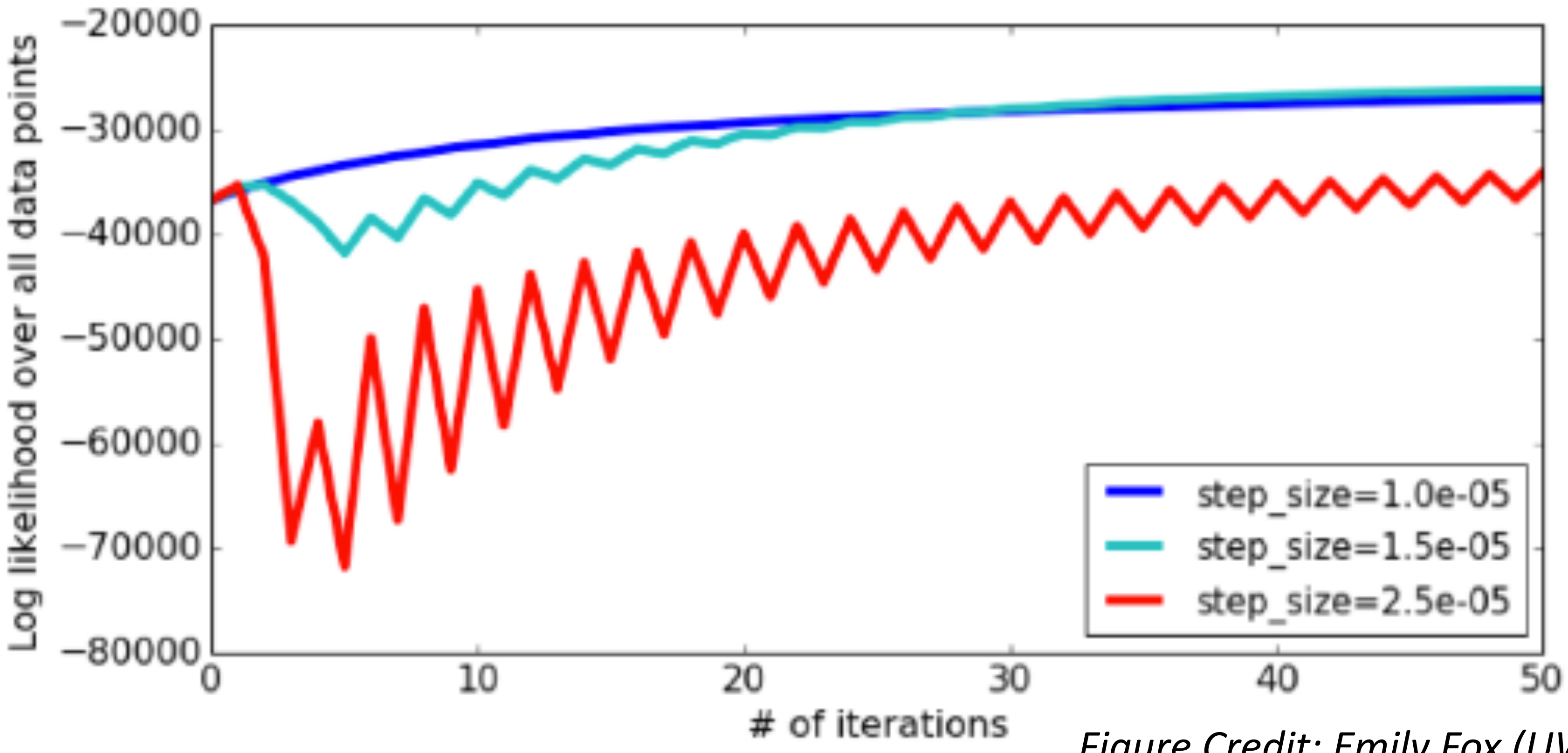


Figure Credit: Emily Fox (UW)

If step size is way too large

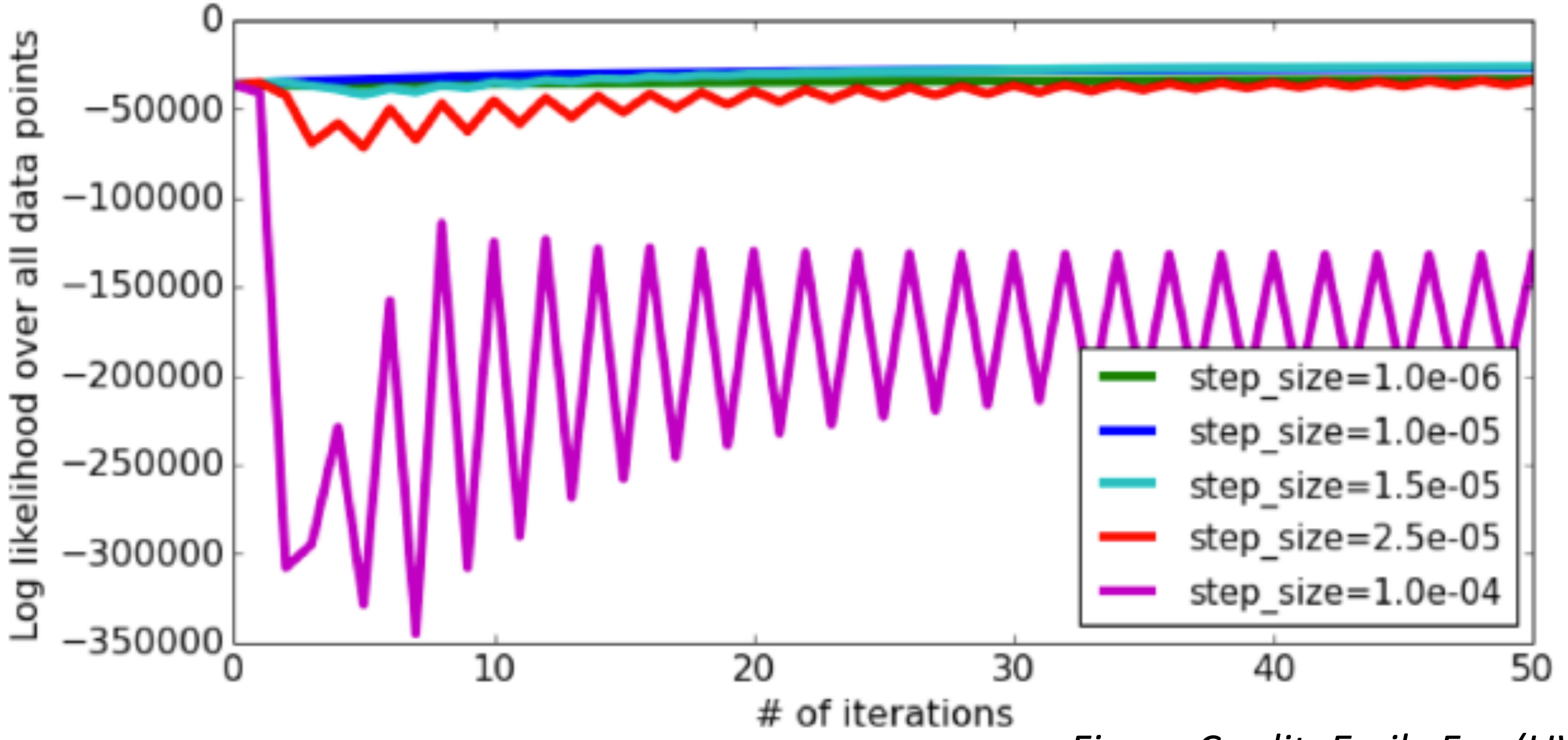


Figure Credit: Emily Fox (UW)

Rule for picking step sizes

- Never try just one!
- Usually: Want largest step size that doesn't diverge
- Try several values (exponentially spaced) until
  - Find one clearly too small
  - Find one clearly too large (unhelpful oscillation / divergence)
- Always make trace plots!
  - Show the loss, norm of gradient, and parameter values versus epoch

- Smarter choices for step size:
  - Decaying methods
  - Search methods
  - Second-order methods

Decaying step sizes

$$\textbf{input: } \text{initial } w \in \mathbb{R}$$
$$\textbf{input: } \text{initial step size } s_0 \in \mathbb{R}_+$$
$$\text{while not converged:}$$
$$w \leftarrow w - s_t \nabla_w \mathcal{L}(w)$$
$$s_t \leftarrow \text{decay}(s_0, t)$$
$$t \leftarrow t + 1$$

Linear decay $\frac{s_0}{kt}$

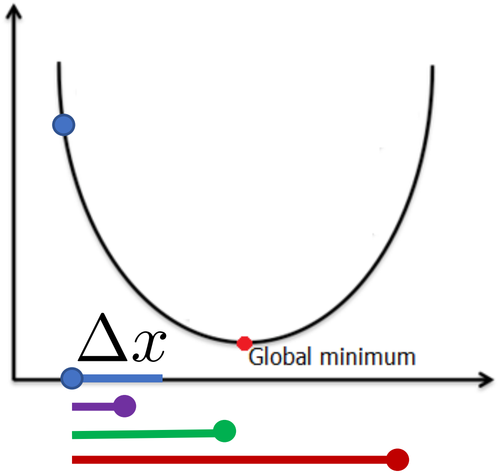
Exponential decay $s_0 e^{-kt}$

Often helpful, but hard to get right!

Searching for good step size

Goal: $$\min_x f(x)$$

Step Direction: $$\Delta x = -\nabla_x f(x)$$



Possible step lengths

Exact Line Search: Expensive but gold standard

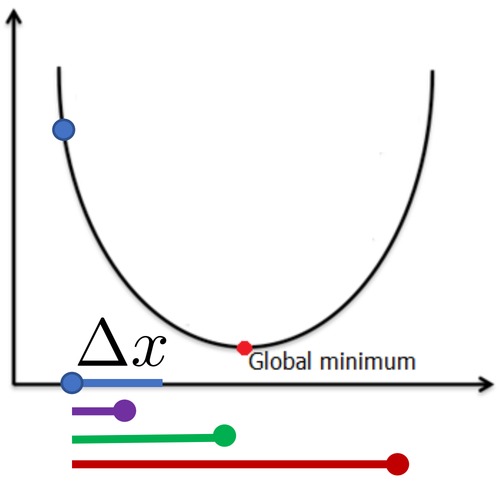
Search for the best scalar s >= 0, such that:

$$s^* = \arg\min_{s \geq 0} f(x + s\Delta x)$$

Searching for good step size

Goal: $$\min_x f(x)$$

Step Direction: $$\Delta x = -\nabla_x f(x)$$



Possible step lengths

Backtracking Line Search: More Efficient!

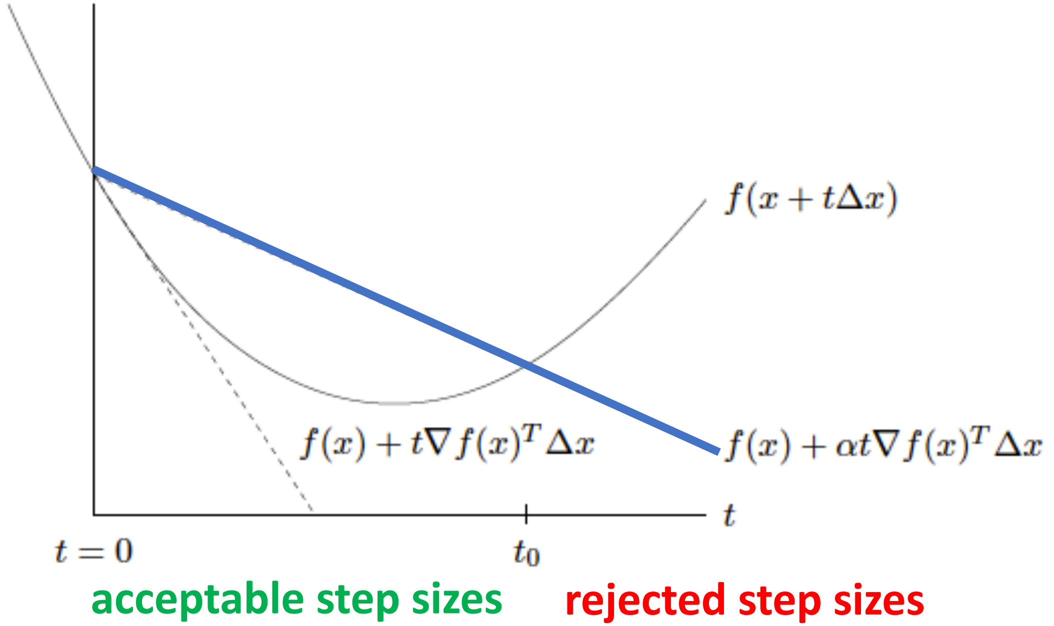
$s = 1$

while reduced slope linear extrapolation $\hat{f}(x + s\Delta x) < f(x + s\Delta x)$:

$\quad s \leftarrow 0.9 \cdot s$

Backtracking line search

Python : scipy.optimize.line_search



Linear extrapolation with reduced slope by factor alpha

Figure 9.1 *Backtracking line search.* The curve shows f, restricted to the line over which we search. The lower dashed line shows the linear extrapolation of f, and the upper dashed line has a slope a factor of α smaller. The backtracking condition is that f lies below the upper dashed line, *i.e.*, $0 \leq t \leq t_0$.

$s = 1$

while reduced slope linear extrapolation $\hat{f}(x + s\Delta x) < f(x + s\Delta x)$:

$s \leftarrow 0.9 \cdot s$

More resources on step sizes!

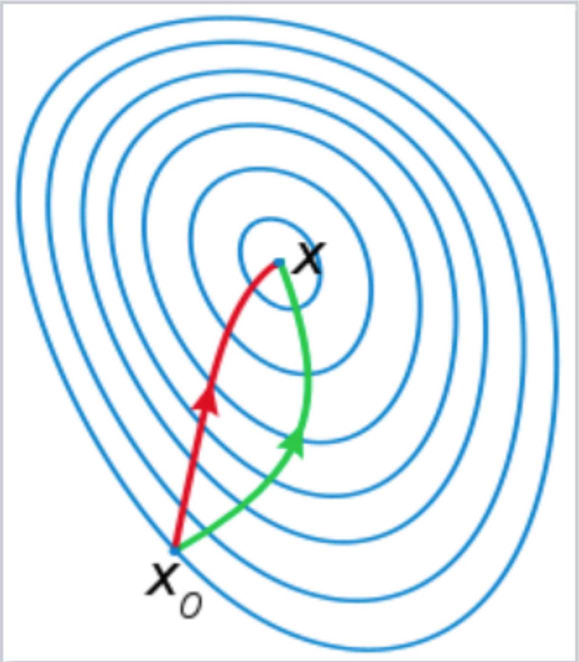
Online Textbook: Convex Optimization

http://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

Convex Optimization
Stephen Boyd and Lieven Vandenberghe

Cambridge University Press

2nd order methods for gradient descent
Big Idea: 2nd deriv. can help!

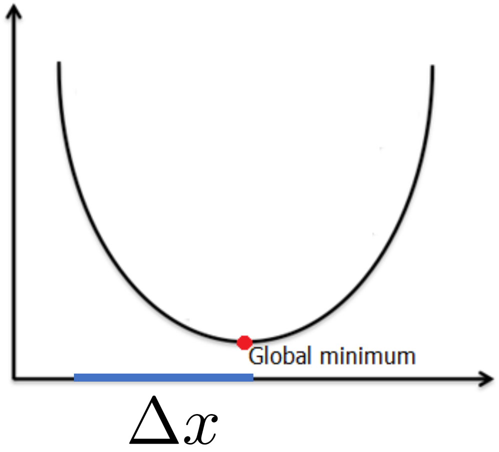


A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information (i.e. the second derivative) to take a more direct route.

Newton's method:
Use second-derivative to rescale step size!

Goal: $$\min_x f(x)$$

Step Direction: $$\Delta x = -\frac{f'(x_n)}{f''(x_n)}$$

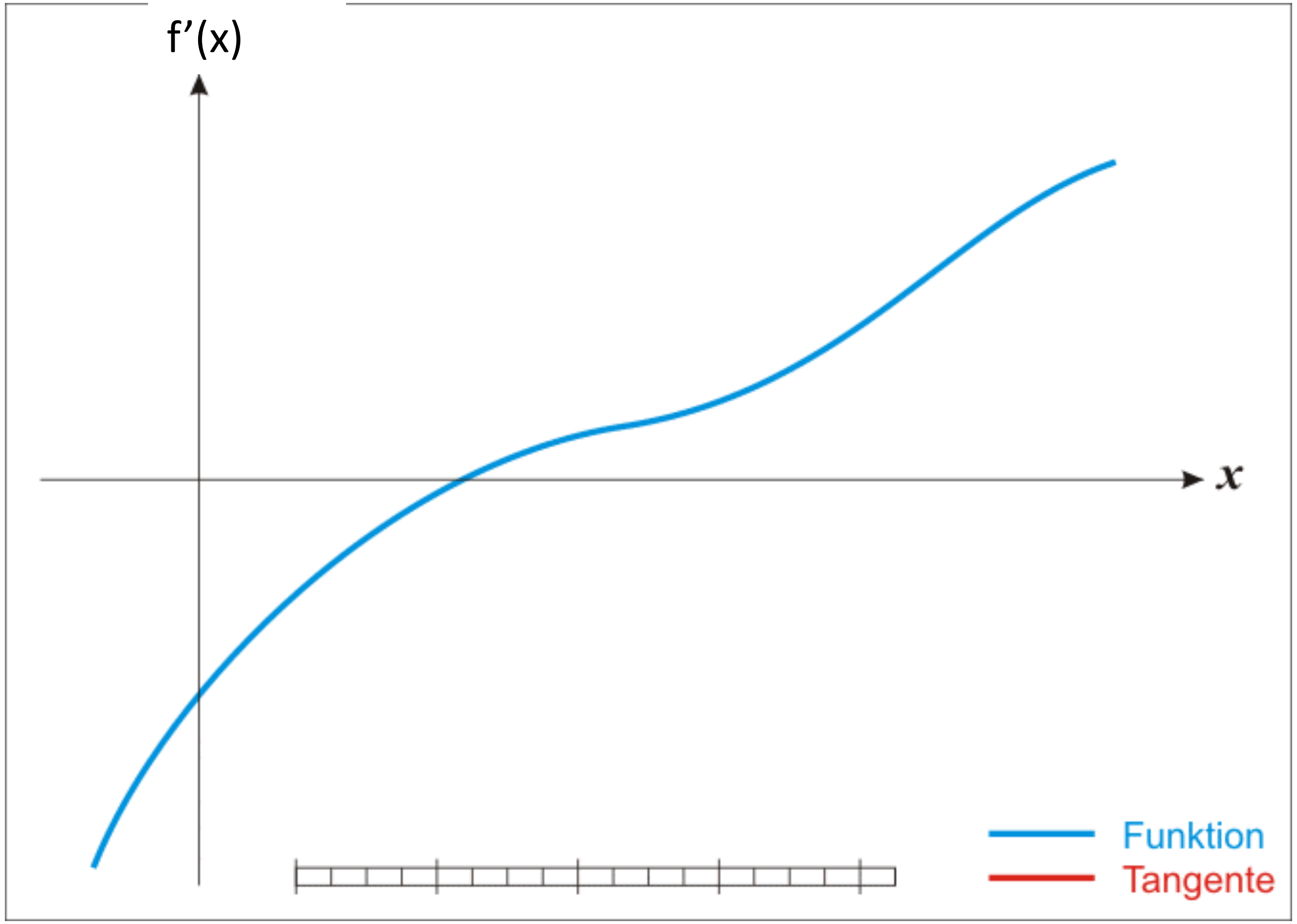


Will step directly to minimum if f is quadratic!

$$\Delta x = -H(x)^{-1}\nabla_x f(x)$$

In high dimensions, need the Hessian matrix

Animation of Newton's method



$$\Delta x = -\frac{f'(x_n)}{f''(x_n)}$$

To optimize, we want to find zeros of first derivative!

L-BFGS: gold standard approximate 2nd order GD

Python : scipy.optimize.fmin_l_bfgs_b

L-BFGS : Limited Memory Broyden–Fletcher–Goldfarb–Shanno (BFGS)

- Provide loss and gradient functions
- Approximates the Hessian via recent history of gradient steps

$$\Delta x = -H(x)^{-1} \nabla_x f(x)$$

In high dimensions, need the Hessian matrix
But this is quadratic in length of x , **expensive**

$$\Delta x = -\hat{H}(x)^{-1} \nabla_x f(x)$$

Instead, use low-rank approximation