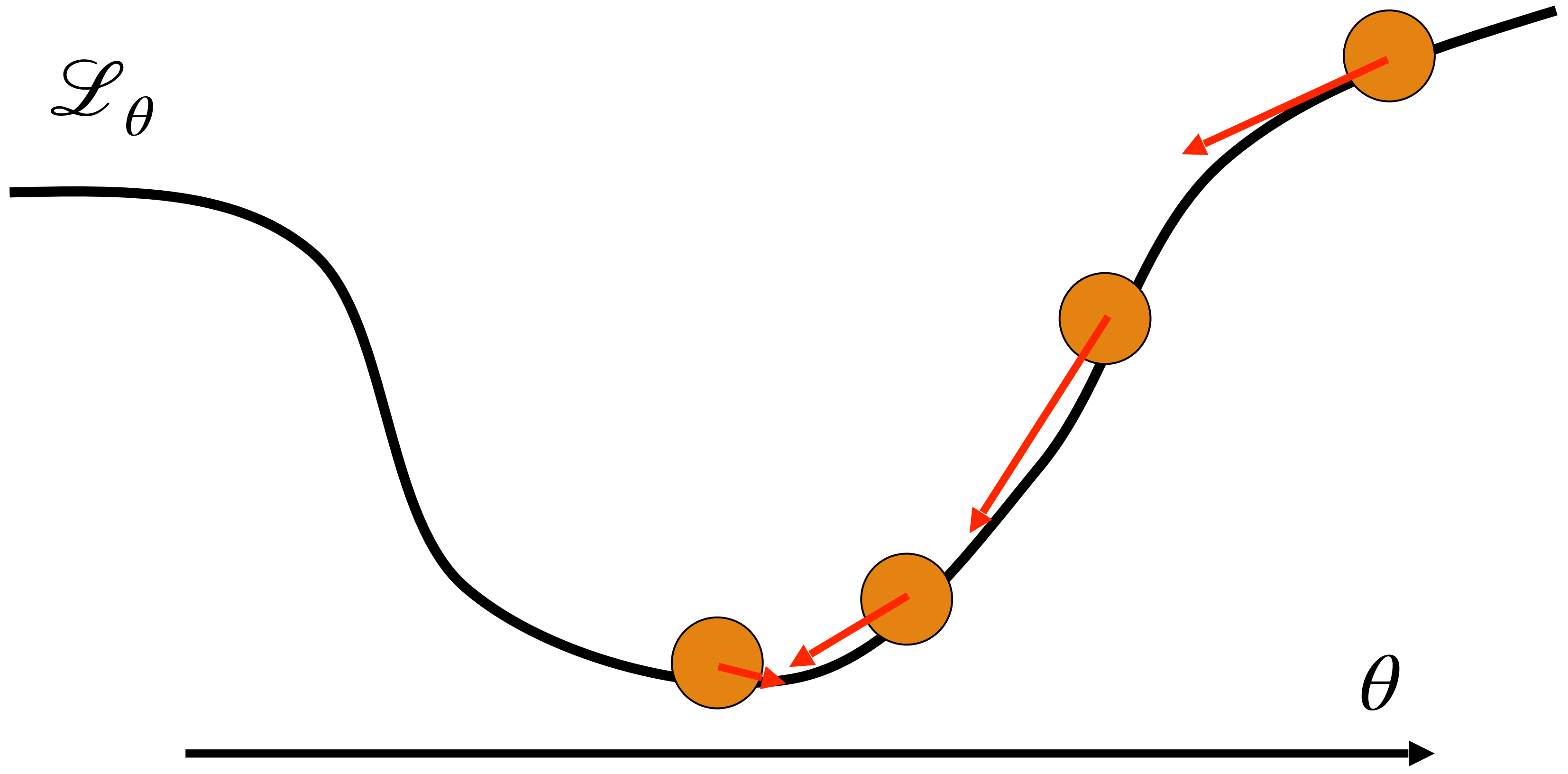
Gradient Descent

- Initialize θ
- Do
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta$
- While $||\alpha \nabla_\theta \mathscr{L}_\theta|| \leq \epsilon$
- Learning rate: α
  - Can change in each iteration



Gradient for the MSE loss

- MSE: $\mathscr{L}_\theta = \frac{1}{m}\sum_j (\epsilon^{(j)})^2 = \frac{1}{m}\sum_j (y^{(j)} - \theta^\intercal x^{(j)})^2$
- $\partial_{\theta_i}\mathscr{L}_\theta = \frac{1}{m}\sum_j \partial_{\theta_i}(\epsilon^{(j)})^2 = \frac{1}{m}\sum_j 2\epsilon^{(j)}\partial_{\theta_i}\epsilon^{(j)}$
  - $\partial_{\theta_i}(y^{(j)} - \theta^\intercal x^{(j)}) = -\partial_{\theta_i}\theta_i x_i^{(j)} + 0$ in the other terms $= x_i^{(j)}$
  - $\partial_{\theta_i}\mathscr{L}_\theta = -\frac{2}{m}\sum_j \epsilon^{(j)} x_i^{(j)} = -\frac{2}{m}(y - \theta^\intercal X)X_i^\intercal$

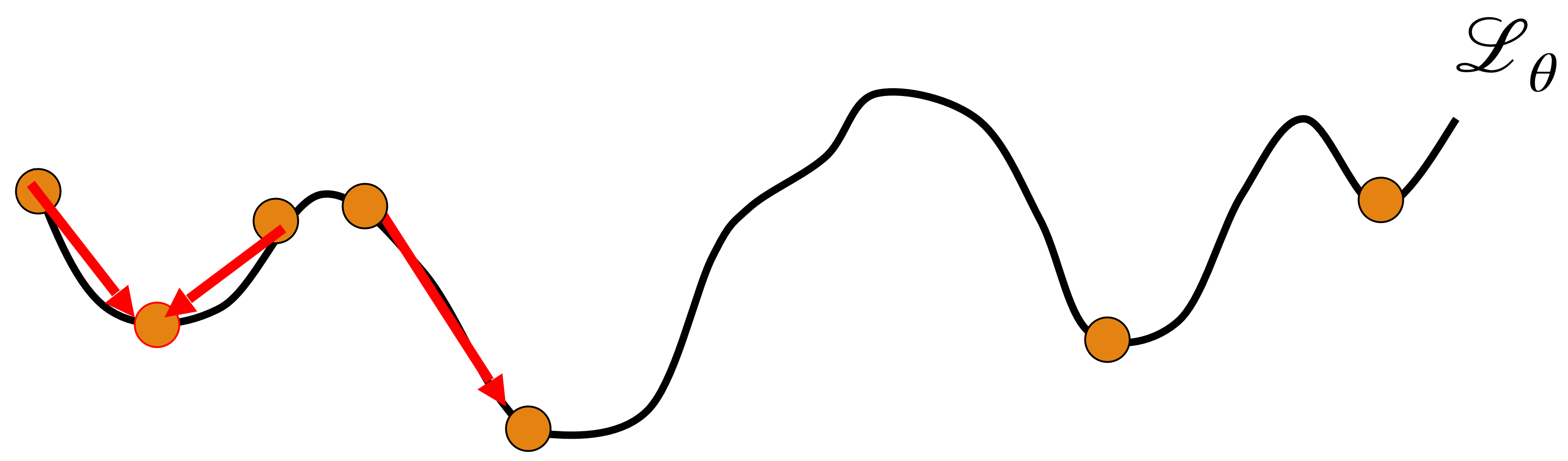
error

- $\nabla_\theta \mathscr{L}_\theta = -\frac{2}{m}(y - \theta^\intercal X)X^\intercal$ sensitivity to θ
- Can also be seen directly from

$$\mathscr{L}_\theta = \frac{1}{m}(y - \theta^\intercal X)(y - \theta^\intercal X)^\intercal = \frac{1}{m}(\theta^\intercal X X^\intercal \theta - 2yX^\intercal\theta + yy^\intercal)$$

Gradient Descent — further considerations

- GD is a very general algorithm
  - We'll use it often
  - Much of the engine for recent advances in ML
- Issues:



  - Can get stuck in local minima
    - Worse — can get stuck in saddle points, $\nabla_\theta \mathscr{L}_\theta = 0$ with improvement direction
  - Can be slow to converge, sensitive to initialization
  - How to choose step size / learning rate?
    - Constant? 1/iteration? Line search? Newton's method?

Newton's method

- Given black-box $f(z)$, how to find a root $f(z) = 0$?
- Initialize some z
- Repeat:
  - Evaluate $f(z)$ and $\partial_z f(z)$ to find tangent to f at z: $f'(z') = (z' - z)\partial_z f(z) + f(z)$
  - Update z to the root of f': $z \leftarrow z - \dfrac{f(z)}{\partial_z f(z)}$
- Considerations:
  - May not converge, sometimes unstable
  - Usually converges quickly for nice, smooth, locally quadratic functions

Newton's method for gradient descent

- We want to find a (local) minimum $f(\theta) = \nabla_\theta \mathscr{L}_\theta = 0$
- Initialize some θ
- Repeat:
  - Evaluate gradient $g = \nabla_\theta \mathscr{L}_\theta$ and Hessian $H = \nabla^2_\theta \mathscr{L}_\theta$
  - Update $\theta \leftarrow \theta - H^{-1} g$
- Considerations:
  - Update step may be too large for highly non-convex losses
  - Computational complexity to invert H: $O(n^3)$

Gradient Descent: complexity

- Assume $\mathcal{L}_\theta(\mathcal{D}) = \frac{1}{m}\sum_j \ell_\theta(x^{(j)}, y^{(j)})$
  - MSE: $\ell_\theta(x, y) = (y - \theta^\intercal x)^2$
- Computing $\nabla_\theta \mathcal{L}_\theta = \frac{1}{m}\sum_j \nabla_\theta \ell_\theta^{(j)}$: usually $O(mn)$
  - What if we use really large datasets? ("big data")
  - What if we learn from data streams? (more data keeps coming in...)

Stochastic / Online Gradient Descent

- Estimate $\nabla_\theta \mathscr{L}_\theta$ fast on a sample of data points
- For each data point:

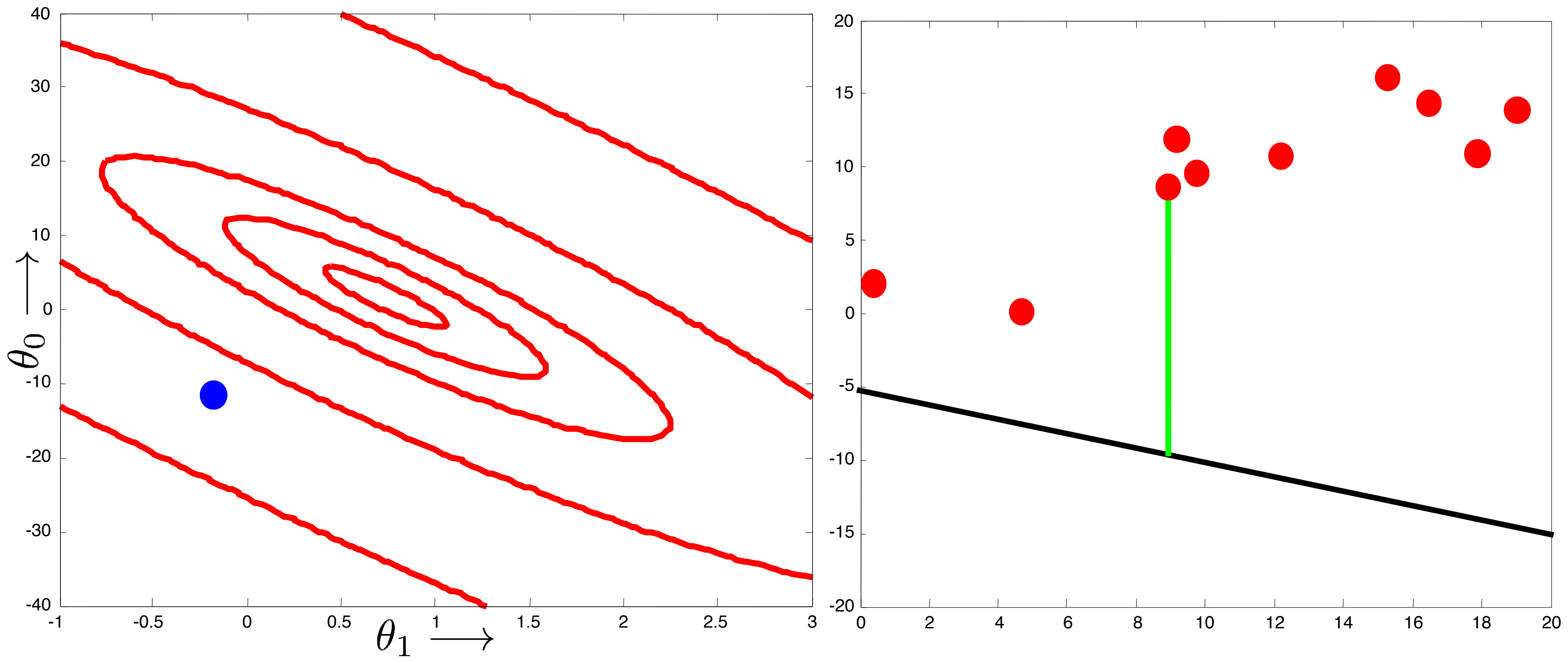
$$\nabla_\theta \mathscr{L}_\theta(x^{(j)}, y^{(j)}) = \nabla_\theta (y^{(j)} - \theta^\intercal x^{(j)})^2 = -2(y^{(j)} - \theta^\intercal x^{(j)})(x^{(j)})^\intercal$$

- This is an unbiased estimator of the gradient, i.e. in expectation

$$\mathbb{E}_{j \sim \mathrm{Uniform}(1,\ldots,m)}[\nabla_\theta \mathscr{L}_\theta^{(j)}] = \frac{1}{m} \sum_j \nabla_\theta \mathscr{L}_\theta^{(j)} = \nabla_\theta \mathscr{L}_\theta(\mathscr{D})$$

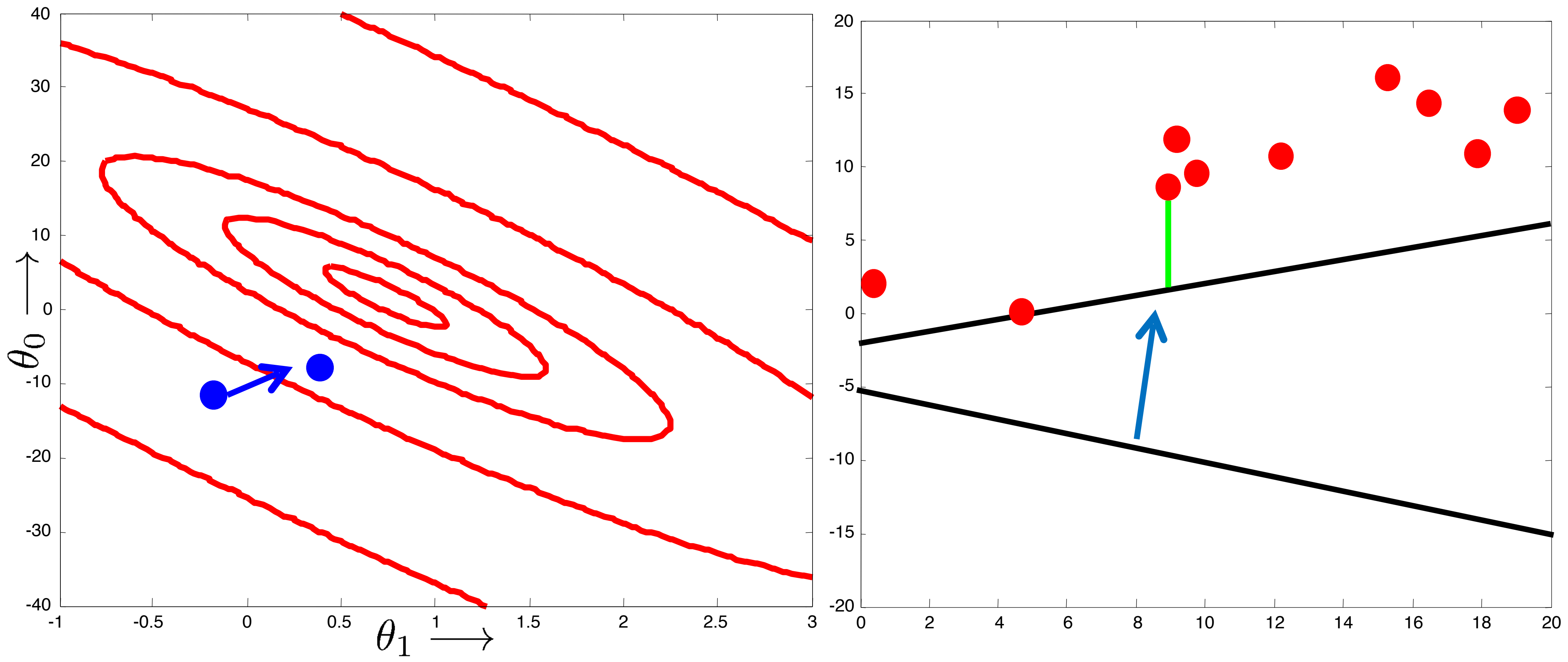
- $\nabla_\theta \mathscr{L}_\theta(\mathscr{D})$ is already a noisy unbiased estimator of true gradient $\mathbb{E}_{x,y \sim p}[\nabla_\theta \mathscr{L}_\theta(x, y)]$
  - SGD is even more noisy

Stochastic Gradient Descent



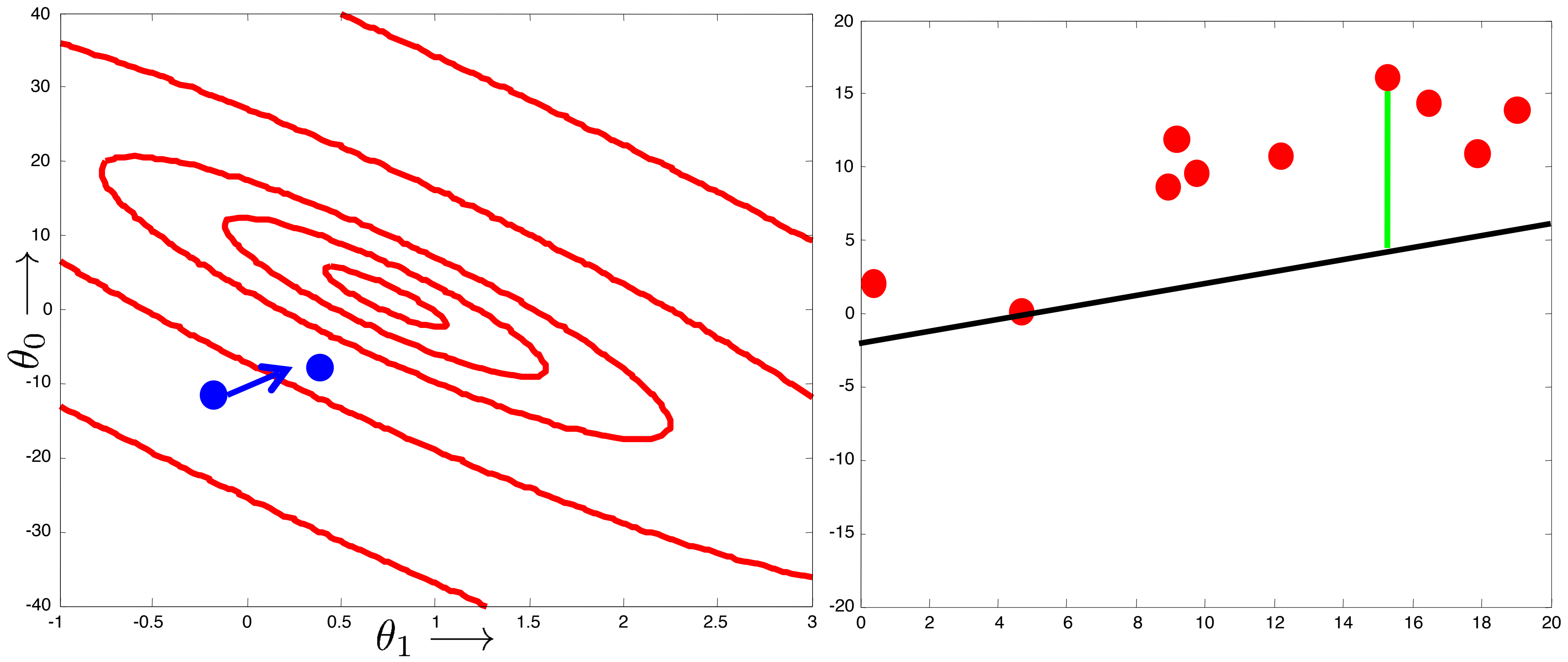
- Initialize θ
- Repeat:
  - Sample $j \sim \mathrm{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



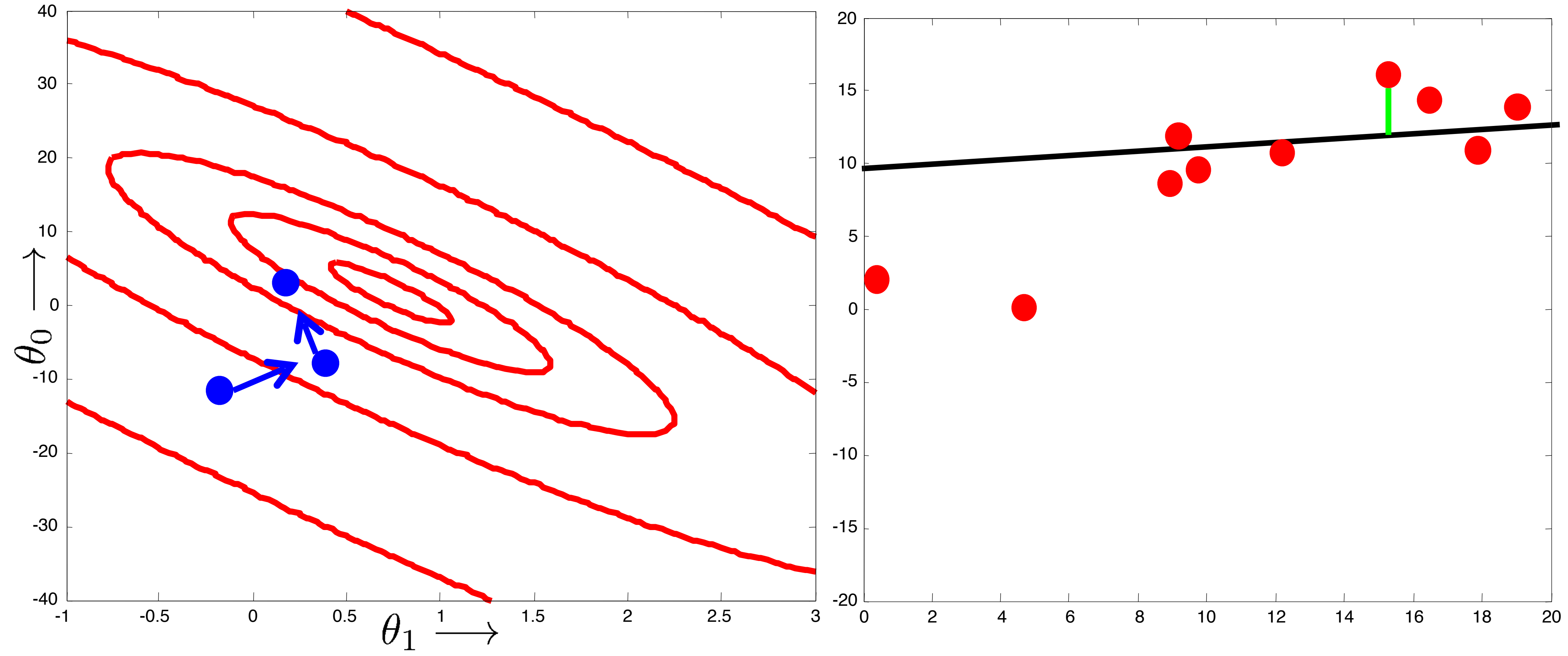
- Initialize θ
- Repeat:
  - Sample $j \sim \mathrm{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



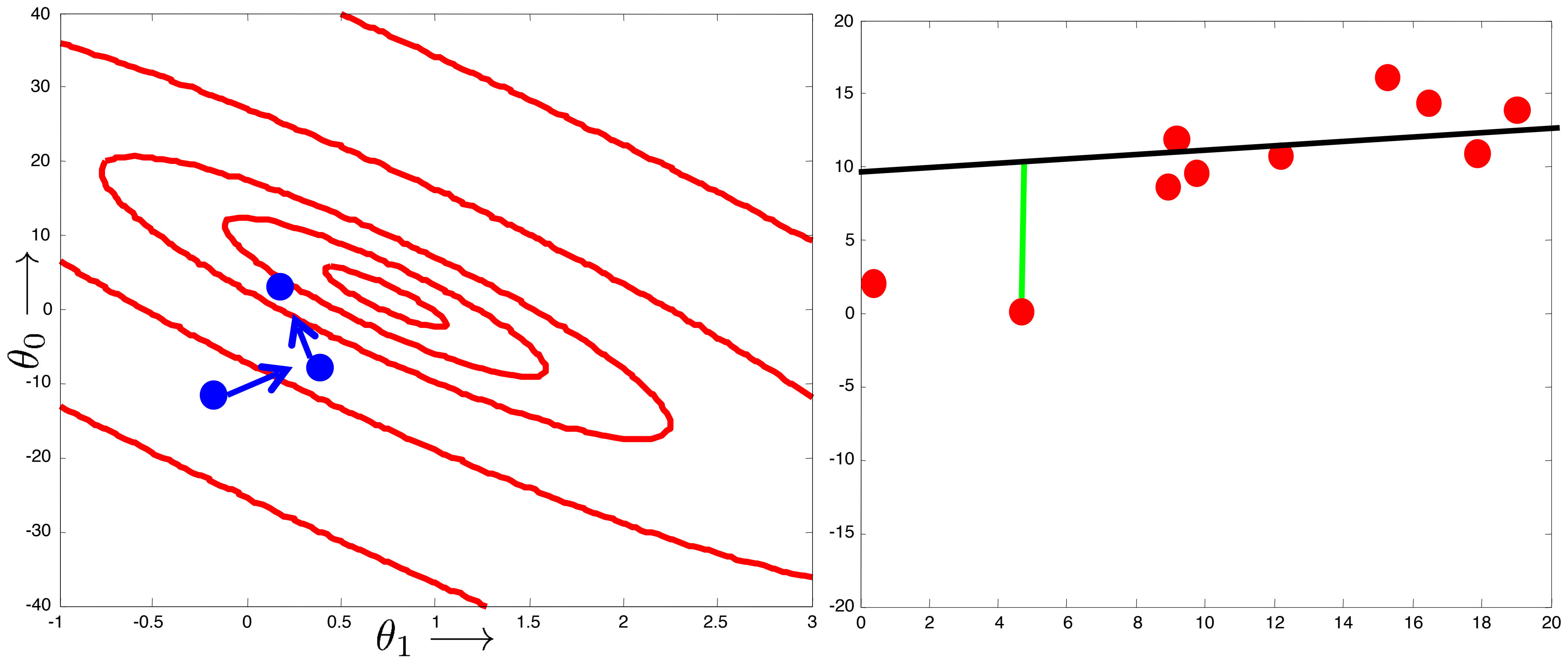
- Initialize θ
- Repeat:
  - Sample $j \sim \text{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



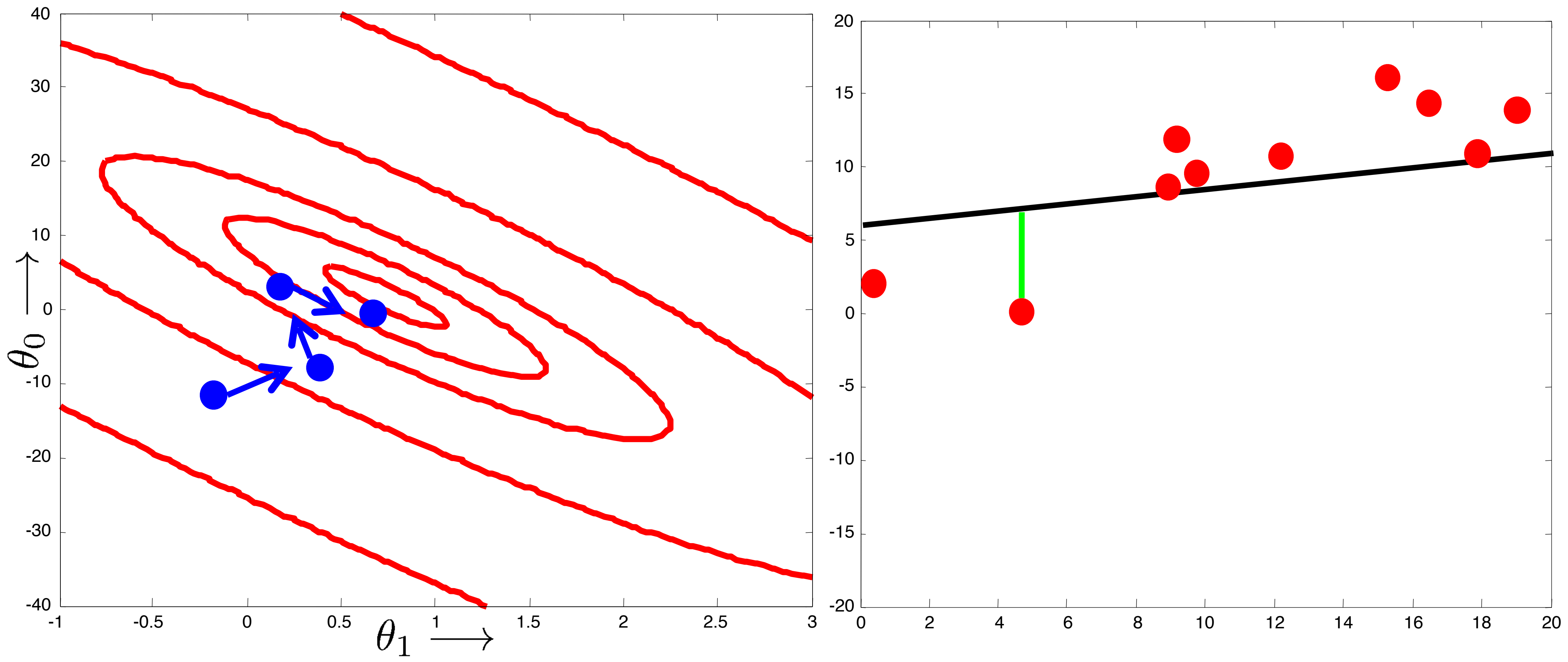
- Initialize θ
- Repeat:
  - Sample $j \sim \text{Uniform}(1, \ldots, m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



- Initialize θ
- Repeat:
  - Sample $j \sim \mathrm{Uniform}(1, \ldots, m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent

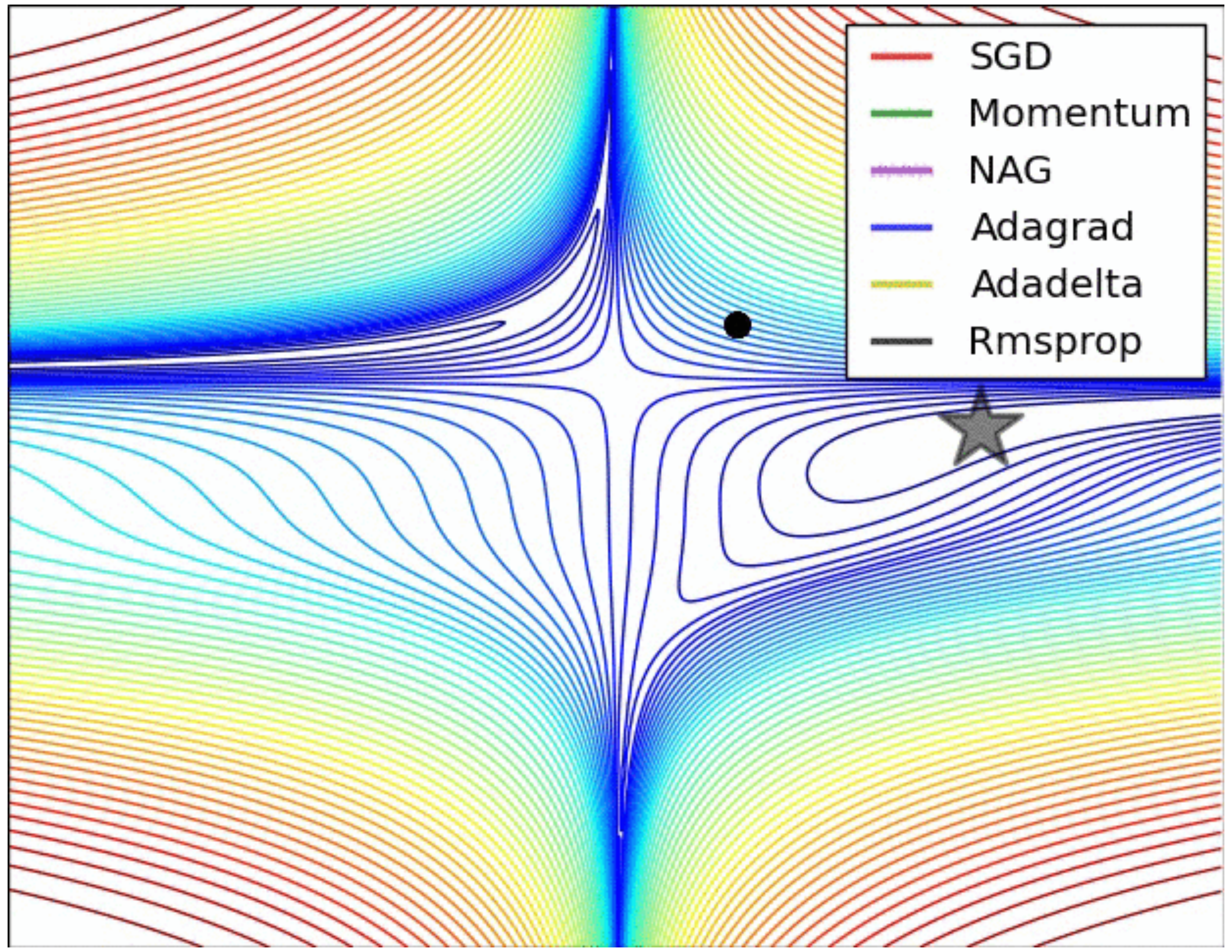


- Initialize θ
- Repeat:
  - Sample $j \sim \mathrm{Uniform}(1, \ldots, m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent: considerations

- Benefits:
  - Each gradient step is faster
  - Don't wait for all data with same θ, improve θ "early and often"
  - Arguably the most important optimization algorithm nowadays
- Drawbacks:
  - May not actually descend on training loss
  - Stopping conditions may be harder to evaluate
- Mini-batch updates: draw $b \ll m$ data points
  - $\operatorname{var} \nabla_\theta \mathscr{L}_\theta(\text{batch}) = \operatorname{var} \frac{1}{b} \sum_{j \in \text{batch}} \nabla_\theta \mathscr{L}_\theta^{(j)} = \frac{1}{b} \operatorname{var} \nabla_\theta \mathscr{L}_\theta(\text{point})$
  - Variance increases the smaller the batch size
    - Generally bad, but can help overcome local minima / saddle points

Advanced gradient-based methods



- Momentum
  - Gradient is like velocity in parameter space
    - Previous gradients still carry momentum
  - Smoothens SGD path
  - Effectively averages gradients over steps, reduces variance
- Preconditioning
  - Scale and rotate loss landscape to make it nicer
  - E.g., multiply by inverse Hessian (as in Newton's method)