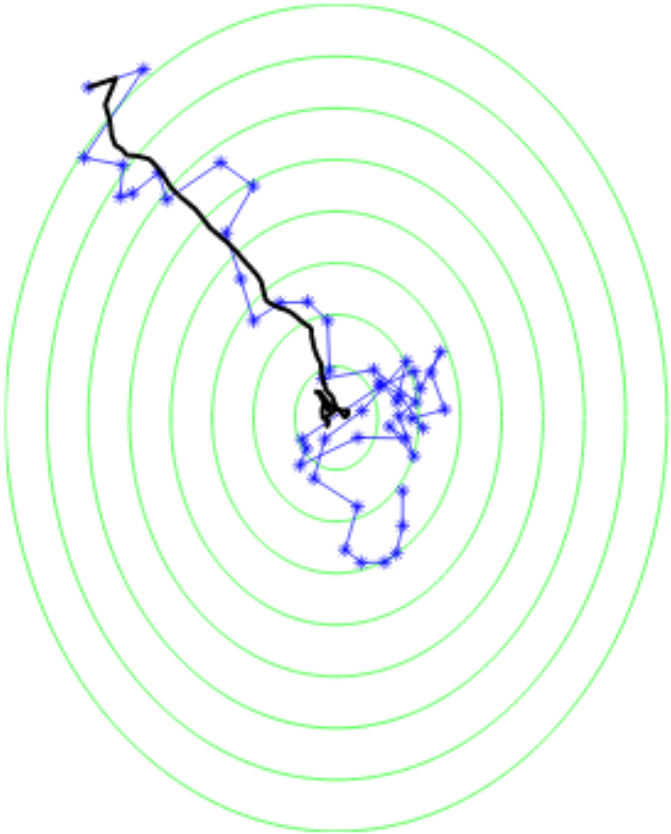
Optimization in Machine Learning

First order methods: SGD



Learning goals

- SGD
- Stochasticity
- Convergence
- Batch size

STOCHASTIC GRADIENT DESCENT

NB: We use g instead of f as objective, bc. f is used as model in ML.

$g : \mathbb{R}^d \to \mathbb{R}$ objective, g **average over functions**:

$$g(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} g_i(\mathbf{x}), \qquad g \text{ and } g_i \text{ smooth}$$

Stochastic gradient descent (SGD) approximates the gradient

$$\begin{aligned} \nabla_{\mathbf{x}}\, g(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \quad &:= \quad \boldsymbol{d} \quad \text{by} \\ \frac{1}{|J|} \sum_{i \in J} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \quad &:= \quad \hat{\boldsymbol{d}}, \end{aligned}$$

with random subset $J \subset \{1, 2, ..., n\}$ of gradients called **mini-batch**.
This is done e.g. when computing the true gradient is **expensive**.

STOCHASTIC GRADIENT DESCENT

Algorithm 1 Basic SGD pseudo code

1: Initialize $\mathbf{x}^{[0]}$, $t = 0$
2: **while** stopping criterion not met **do**
3: Randomly shuffle data and partition into minibatches $J_1, \ldots, J_K$ of size m
4: **for** $k \in \{1, \ldots, K\}$ **do**
5: $t \leftarrow t + 1$
6: Compute gradient estimate with J_k: $\hat{\boldsymbol{d}}^{[t]} \leftarrow \frac{1}{m} \sum_{i \in J_k} \nabla_{\mathbf{x}} g_i(\mathbf{x}^{[t-1]})$
7: Apply update: $\mathbf{x}^{[t]} \leftarrow \mathbf{x}^{[t-1]} - \alpha \cdot \hat{\boldsymbol{d}}^{[t]}$
8: **end for**
9: **end while**

- Instead of drawing batches randomly we might want to go through the g_i sequentially (unless g_i are sorted in any way)
- Updates are computed faster, but also more stochastic:
  - In the simplest case, batch-size $m := |J_k|$ is set to $m = 1$
  - If n is a billion, computation of update is a billion times faster
  - **But** (we will see later): Convergence rates suffer from stochasticity!

SGD IN ML

In ML, we perform ERM:

$$\bar{\mathcal{R}}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)}_{g_i(\theta)}$$

- for a data set

$$\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$$

- a loss function $L(y, f(\mathbf{x}))$, e.g. L2 loss $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$,
- and a model class f, e.g. the linear model $f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) = \boldsymbol{\theta}^\top \mathbf{x}$.

SGD IN ML

For large data sets, computing the exact gradient

$$\boldsymbol{d} = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right) \quad \text{(also: score)}$$

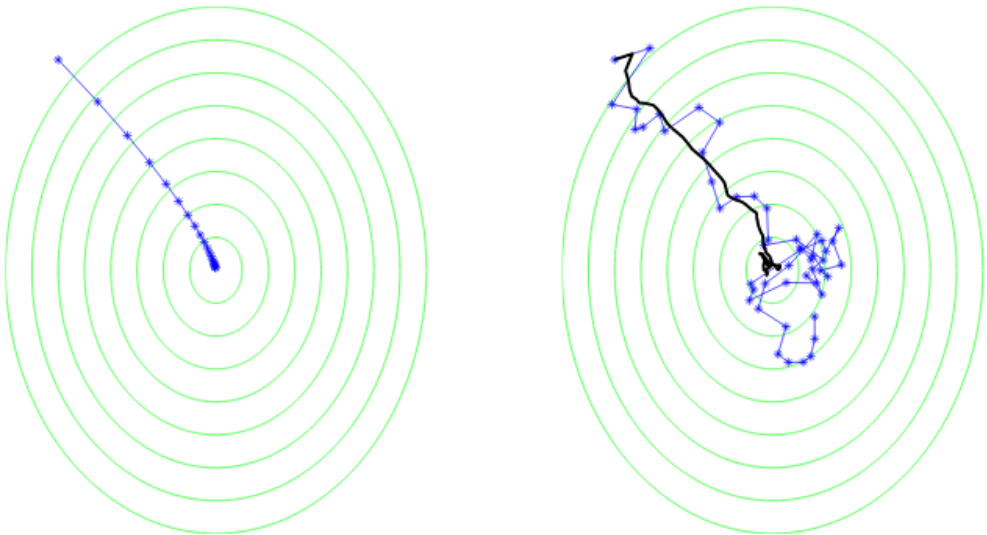
may be expensive or infeasible to compute and is approximated by

$$\hat{\boldsymbol{d}} = \frac{1}{m} \sum_{i \in J} \nabla_{\boldsymbol{\theta}} L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right),$$

for $J \subset 1, 2, \ldots, n$ random subset.

NB: Often, the max. size of J technically limited by how much data fits in to the memory.

STOCHASTICITY OF SGD



Source: Shalev-Shwartz, Ben-David. Understanding machine learning. Cambridge University Press, 2014.

Minimize $g(x_1, x_2) = 1.25(x_1 + 6)^2 + (x_2 - 8)^2)$.

Left: GD. Right: SGD, the black line depicts the averaged value of $\mathbf{x}$.

STOCHASTICITY OF SGD

Assume batch size $m = 1$ (statements also apply for higher batch size).

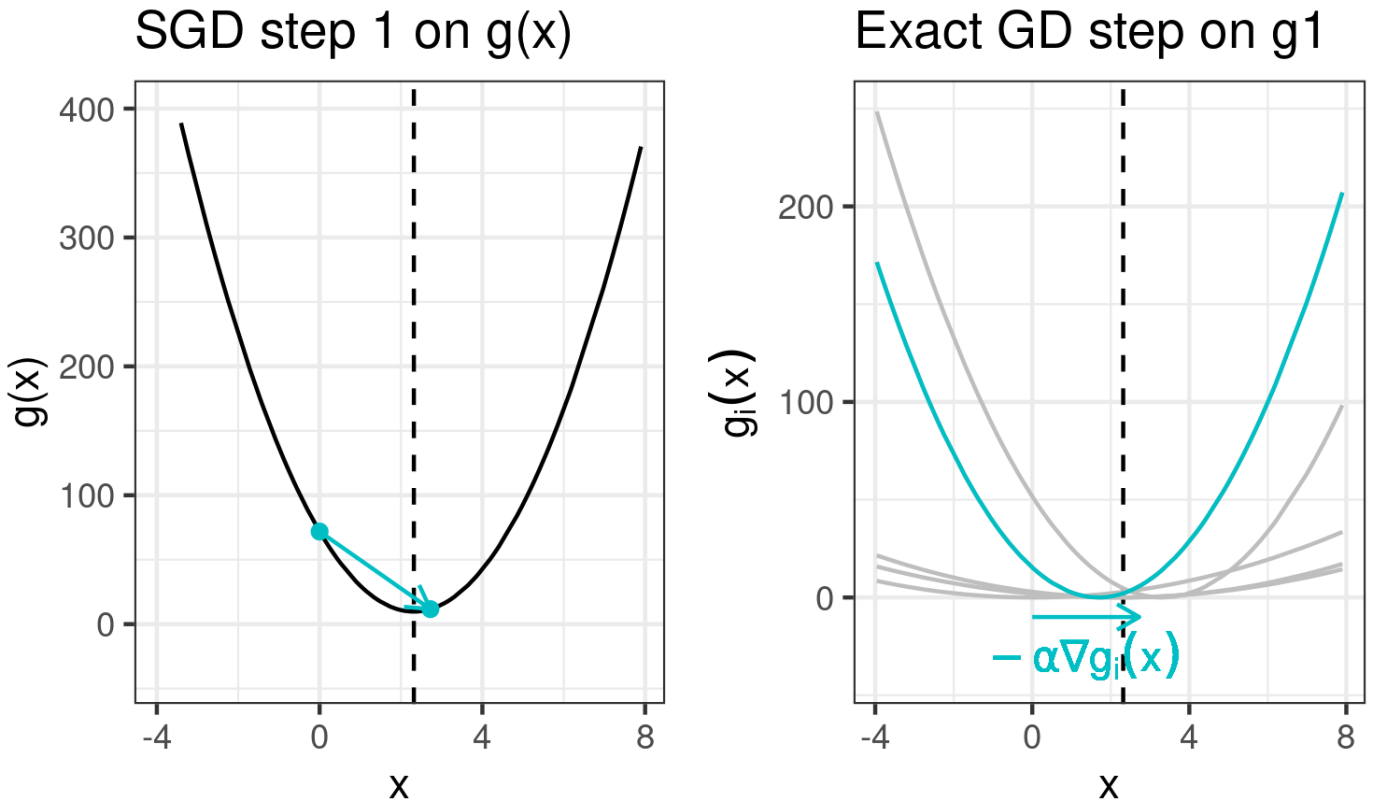
- **(Possibly) suboptimal direction:** Approx. gradient $\hat{\boldsymbol{d}} = \nabla_{\mathbf{x}} g_i(\mathbf{x})$ might point in suboptimal (possibly not even descent!) direction
- **Unbiased estimate:** If J drawn i.i.d., approx. gradient $\hat{\boldsymbol{d}}$ is an unbiased estimate of gradient $\boldsymbol{d} = \nabla_{\mathbf{x}} g(\mathbf{x}) = \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x})$:

$$\begin{aligned} \mathbb{E}_i \left[\nabla_{\mathbf{x}}\, g_i(\mathbf{x})\right] &= \sum_{i=1}^{n} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \cdot \mathbb{P}(i = i) = \sum_{i=1}^{n} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \cdot \frac{1}{n} \\ &= \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) = \nabla_{\mathbf{x}}\, g(\mathbf{x}). \end{aligned}$$

Conclusion: SGD might perform single suboptimal moves, but moves in “right direction” **on average**.

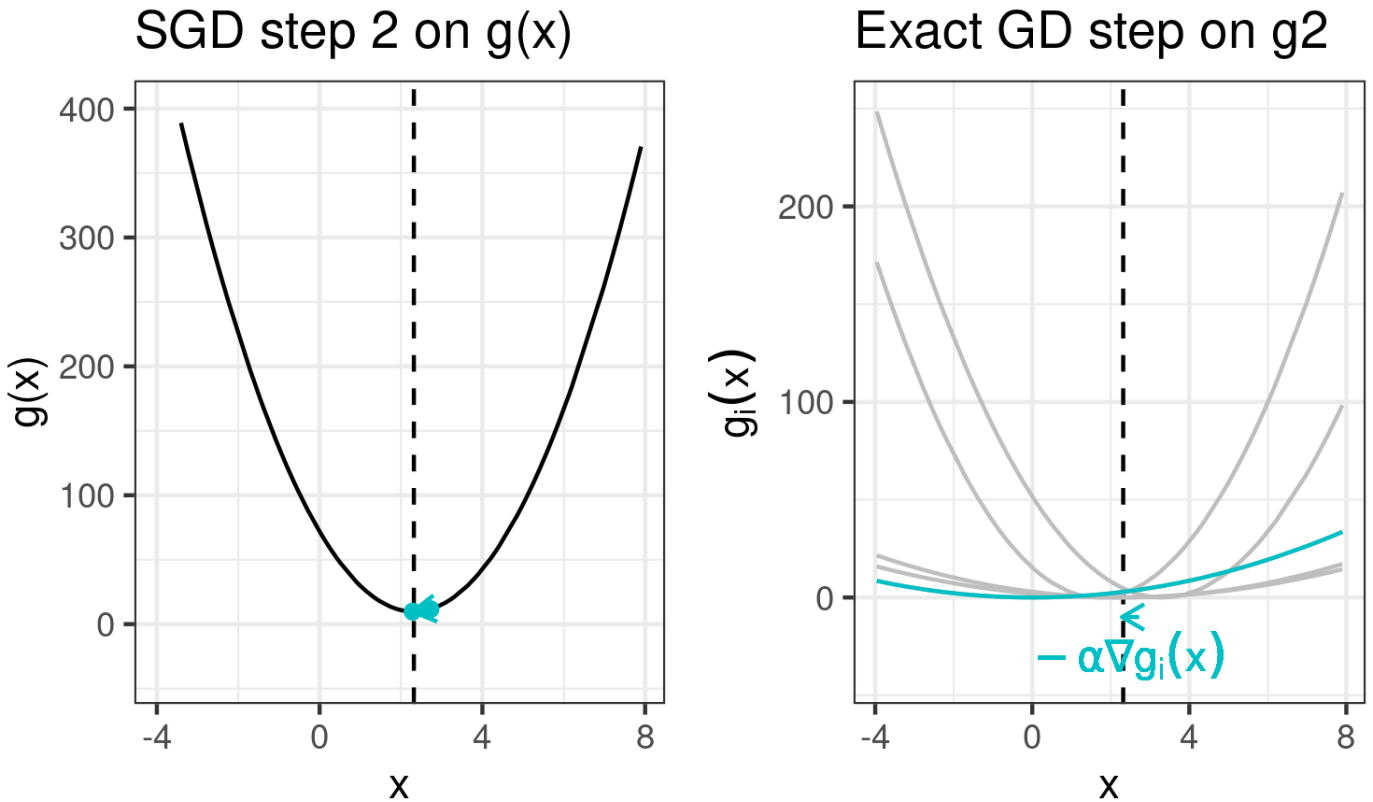
ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. We run SGD with $m = 1$.



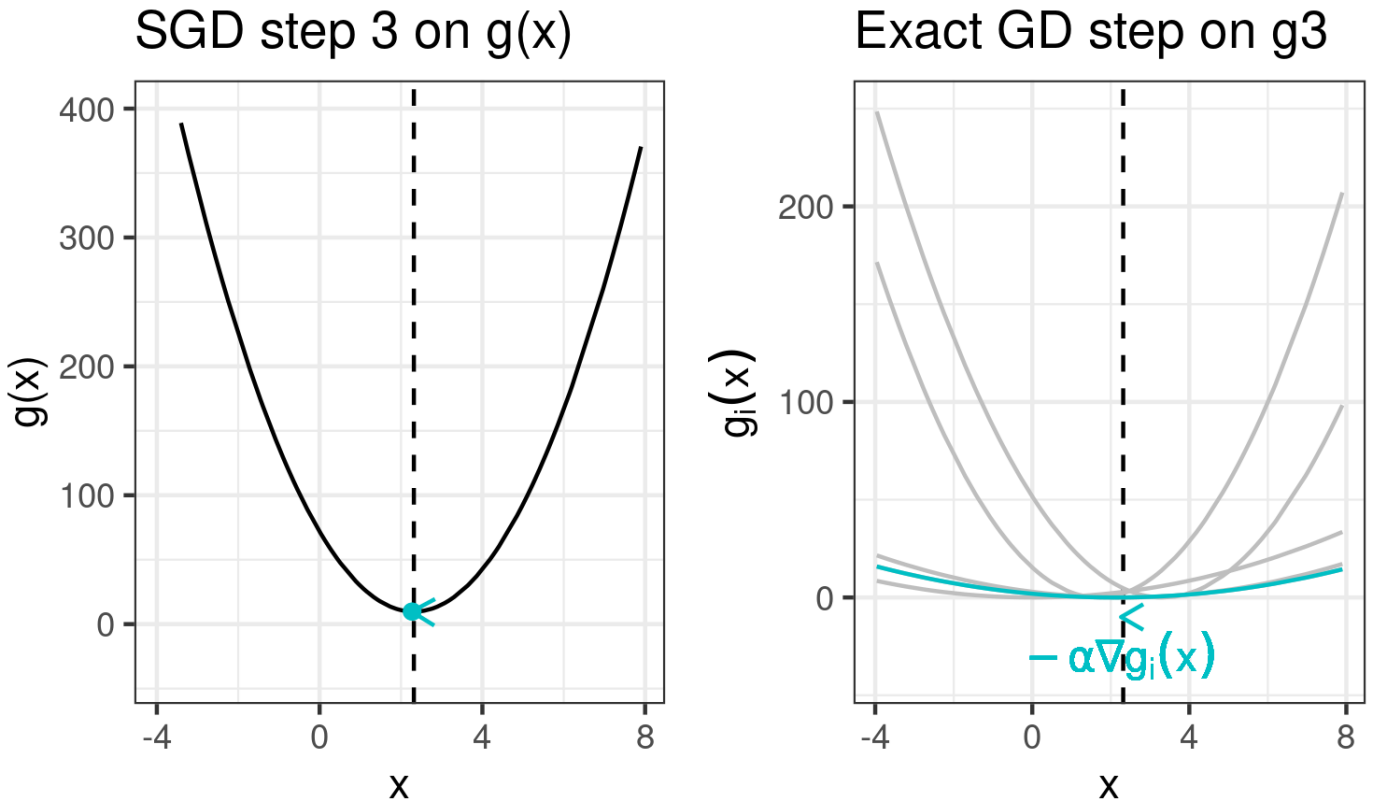
ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. We run SGD with $m = 1$.



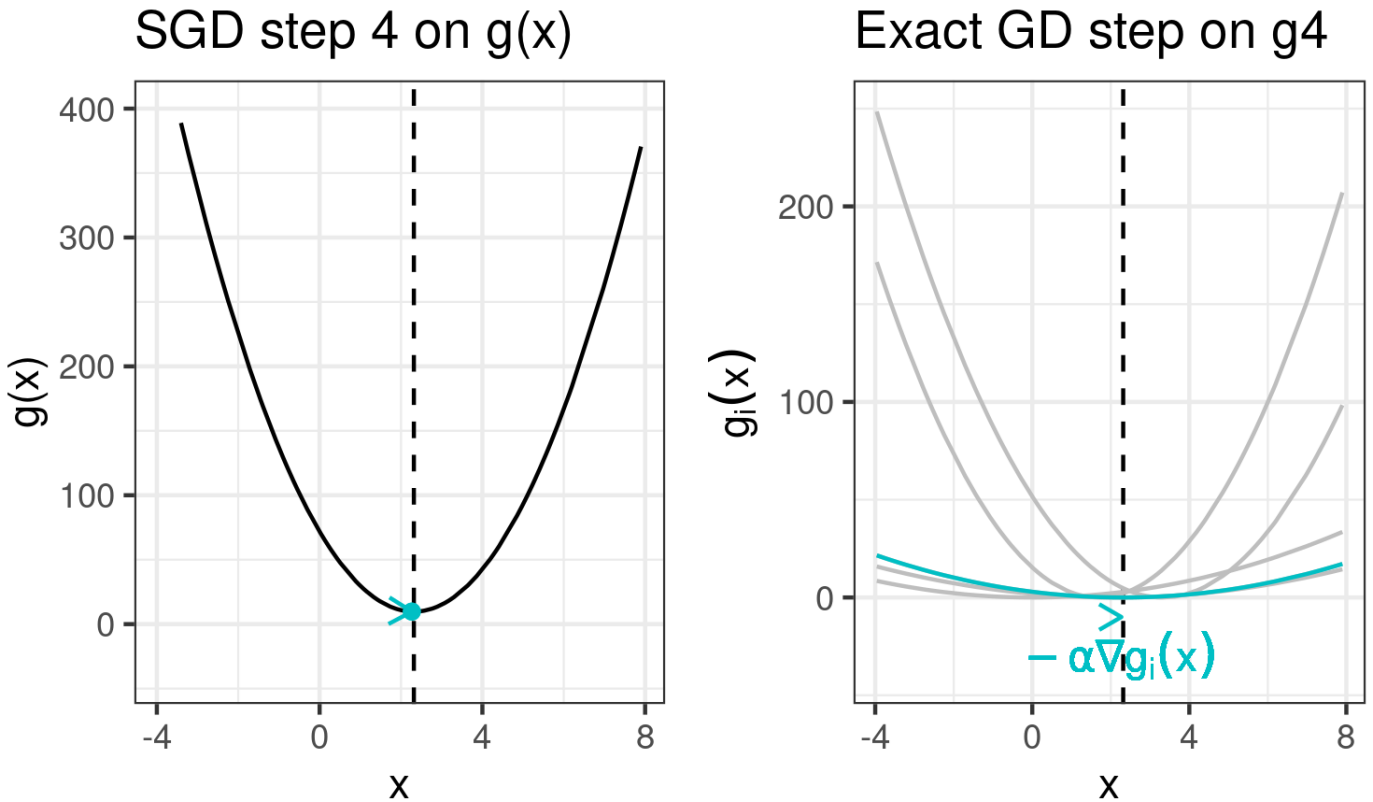
ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. We run SGD with $m = 1$.



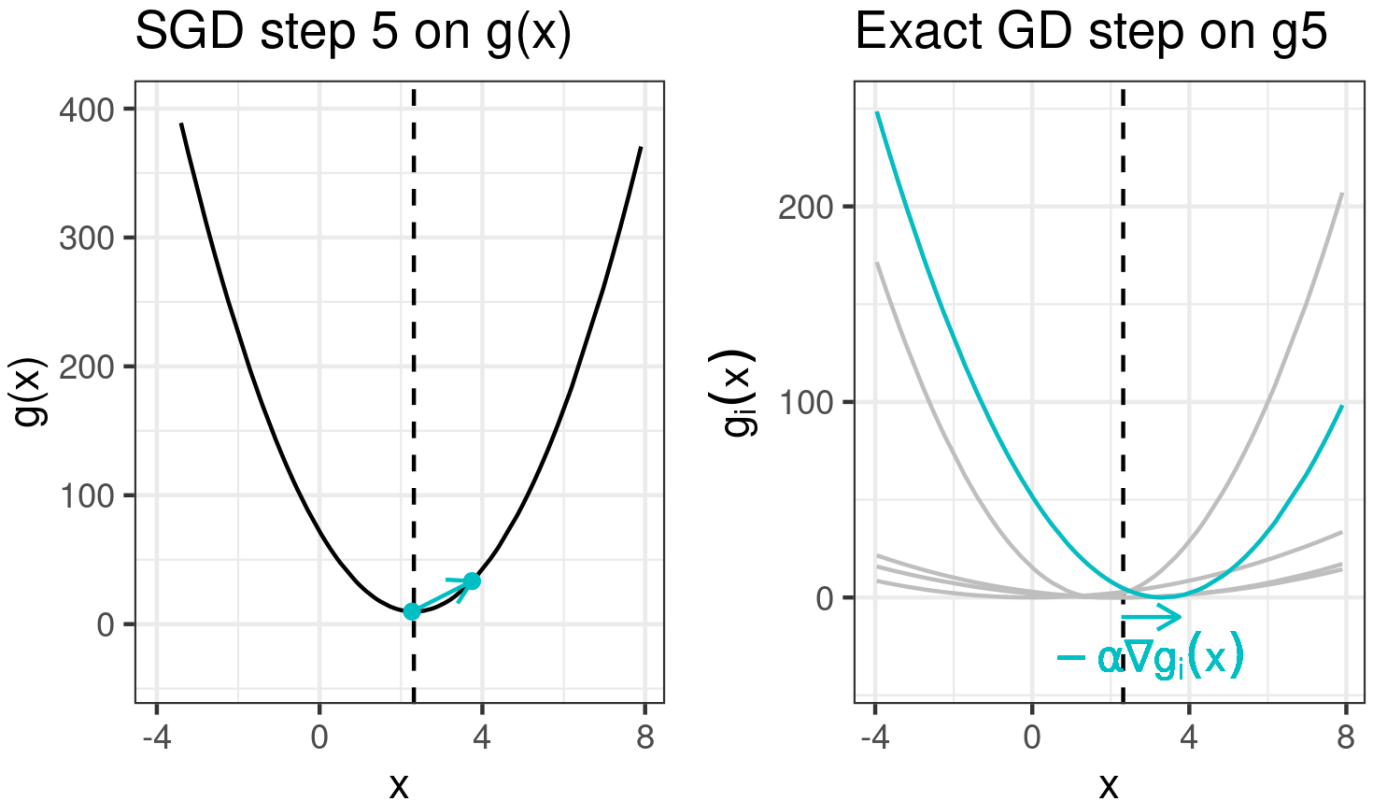
ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. We run SGD with $m = 1$.



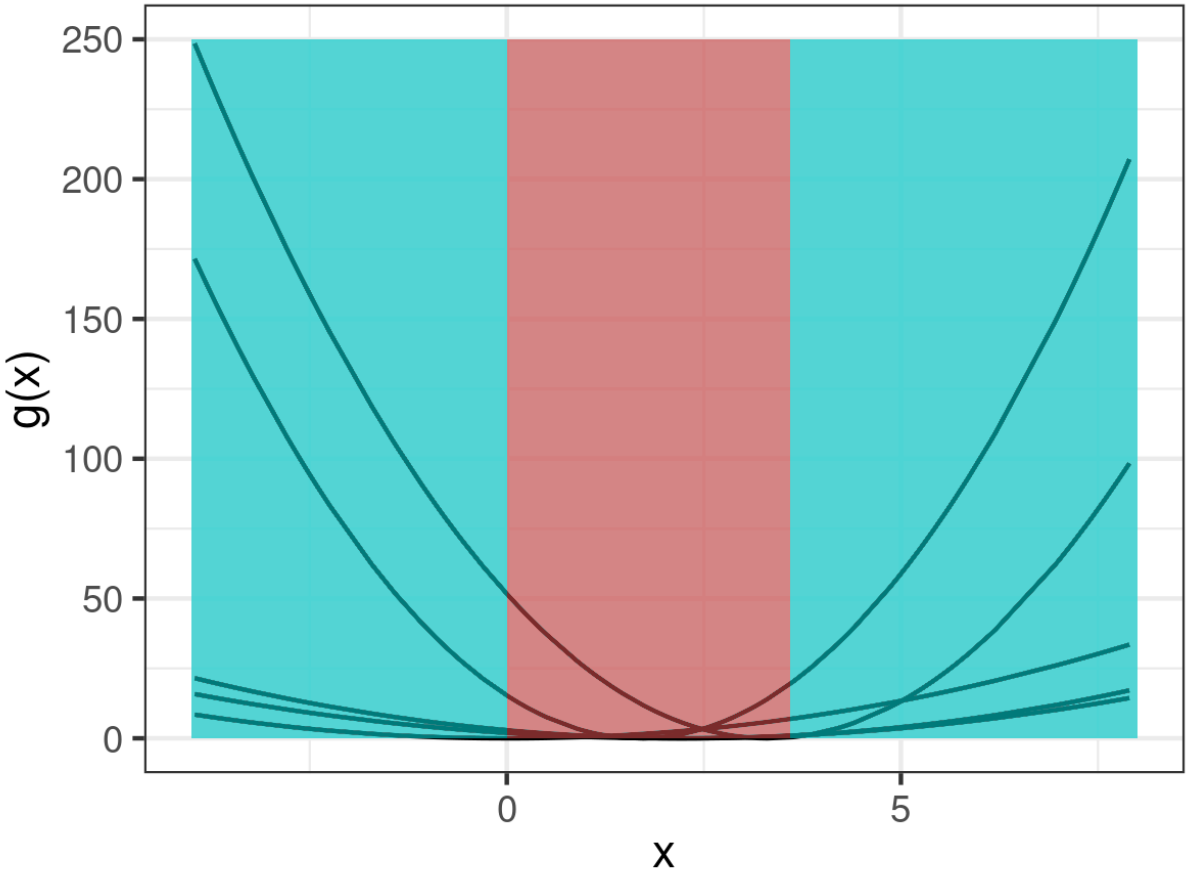
ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. We run SGD with $m = 1$.



In iteration 5, SGD performs a suboptimal move away from the optimum.

ERRATIC BEHAVIOR OF SGD



x in **blue area**: $-\nabla g_i(\mathbf{x})$ will point towards optimum, no matter which i we sample.

x in **red area** (confusion area): We might sample i for which $-\nabla g_i(\mathbf{x})$ points away from optimum and perform suboptimal moves close to optimum.

ERRATIC BEHAVIOR OF SGD

- At $\mathbf{x}$, "confusion" is captured variance of gradients

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla_{\mathbf{x}}\, g_i(\mathbf{x}) - \nabla_{\mathbf{x}}\, g(\mathbf{x})\|^2$$

- If this variance is 0, the next step will be in the right direction (independent of the sampled i)
- If the term is small, the next step will likely go in the right direction
- If the term is large, the step will likely go in the wrong direction (middle of confusion area, where $\mathbf{x}^*$ lives)

CONVERGENCE OF SGD

As a consequence, SGD has worse convergence properties than GD. However, this can be controlled via **increasing the batch size** or **reducing the step size**.

The larger the batch size m

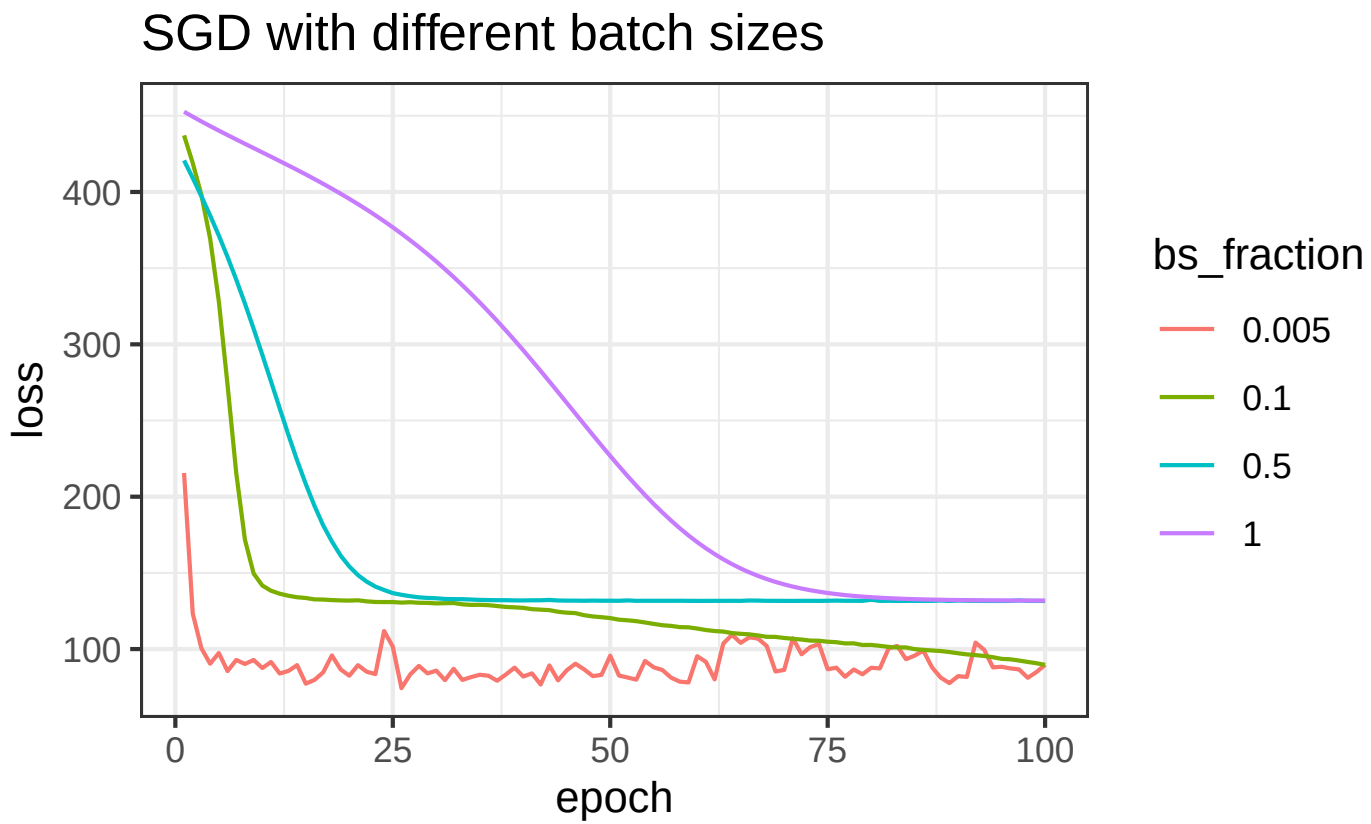
- the better the approximation to $\nabla_{\mathbf{x}} g(\mathbf{x})$
- the lower the variance
- the lower the risk of performing steps in the wrong direction

The smaller the step size α

- the smaller a step in a potentially wrong direction
- the lower the effect of high variance

As maximum batch size is usually limited by computational resources (memory), choosing the step size is crucial.

EFFECT OF BATCH SIZE



SGD to optimize an NN with batch size $\in \{0.5\%, 10\%, 50\%\}$ of the training data. The higher the batch size, the lower the variance.