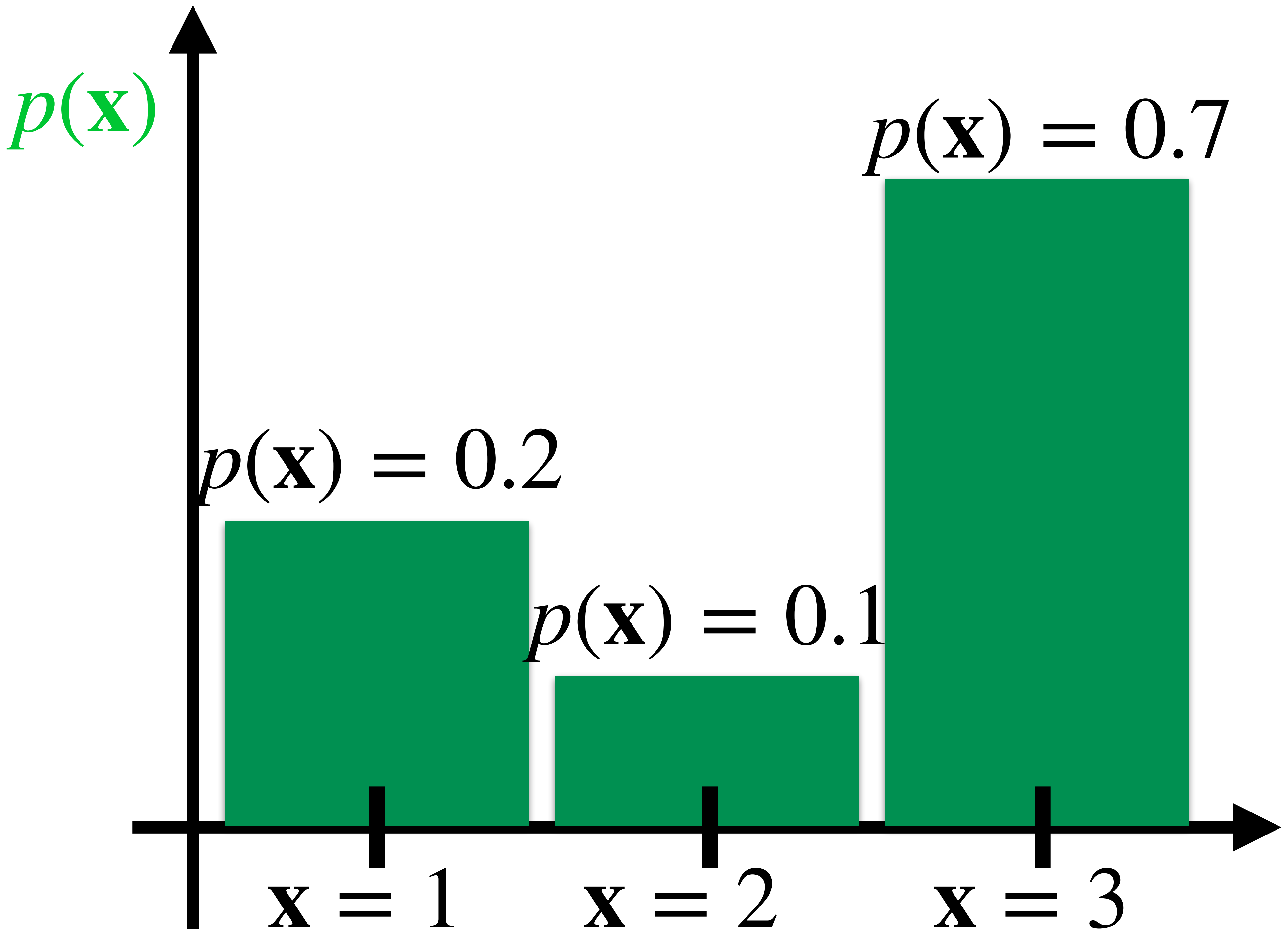
Why is learning prone to fail?

Optimization issues: SGD+momentum and its convergence rate, Adagrad, Adam, diminishing/exploding gradient, oscillations

Karel Zimmermann
Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

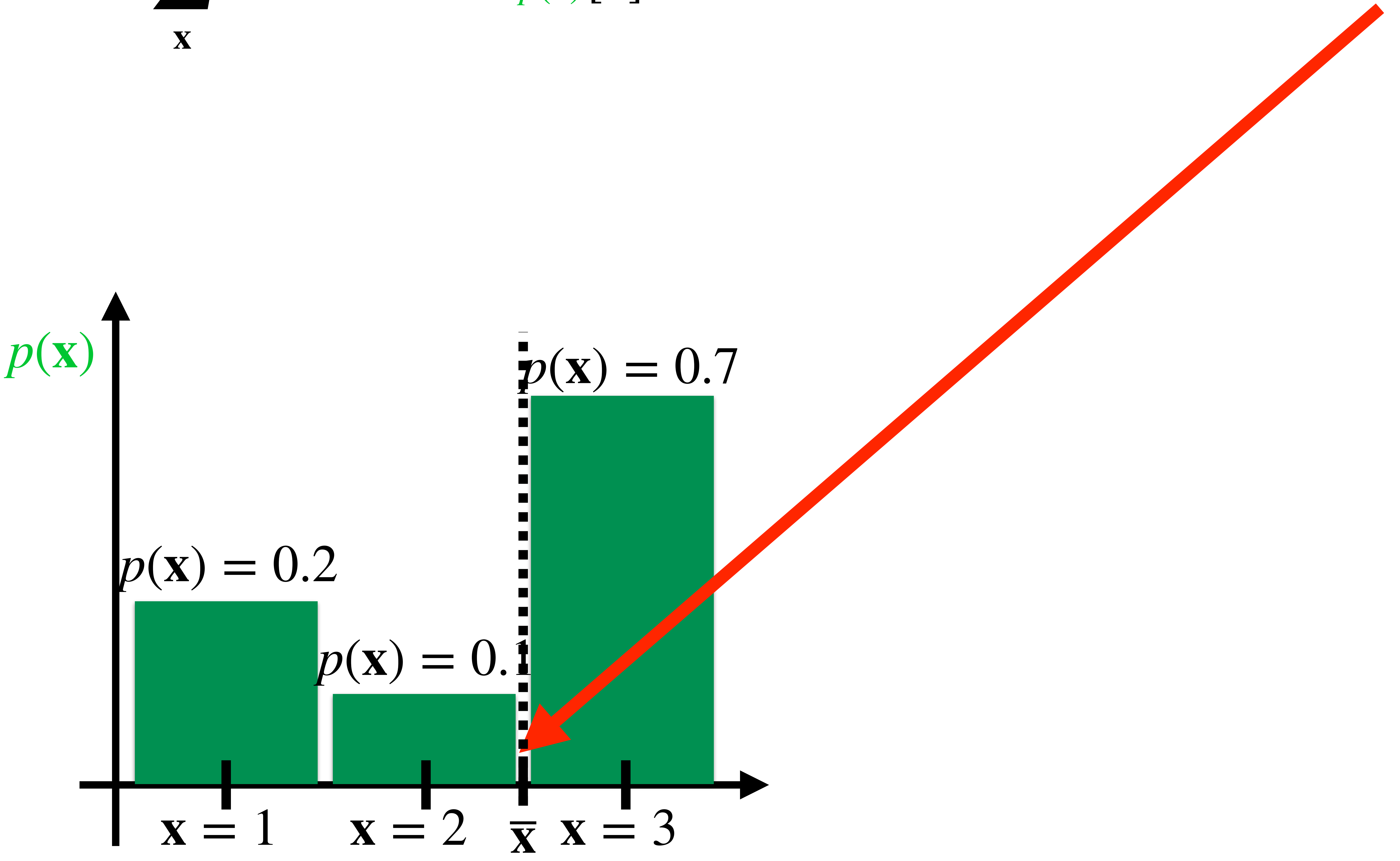
Prerequisites: Mean and average

$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}[\mathbf{x}] = ??$$



Prerequisites: Mean and average

$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}\left[\mathbf{x}\right] = 0.2 \cdot 1 \ + \ 0.1 \cdot 2 \ + \ 0.7 \cdot 3 = 2.5$$

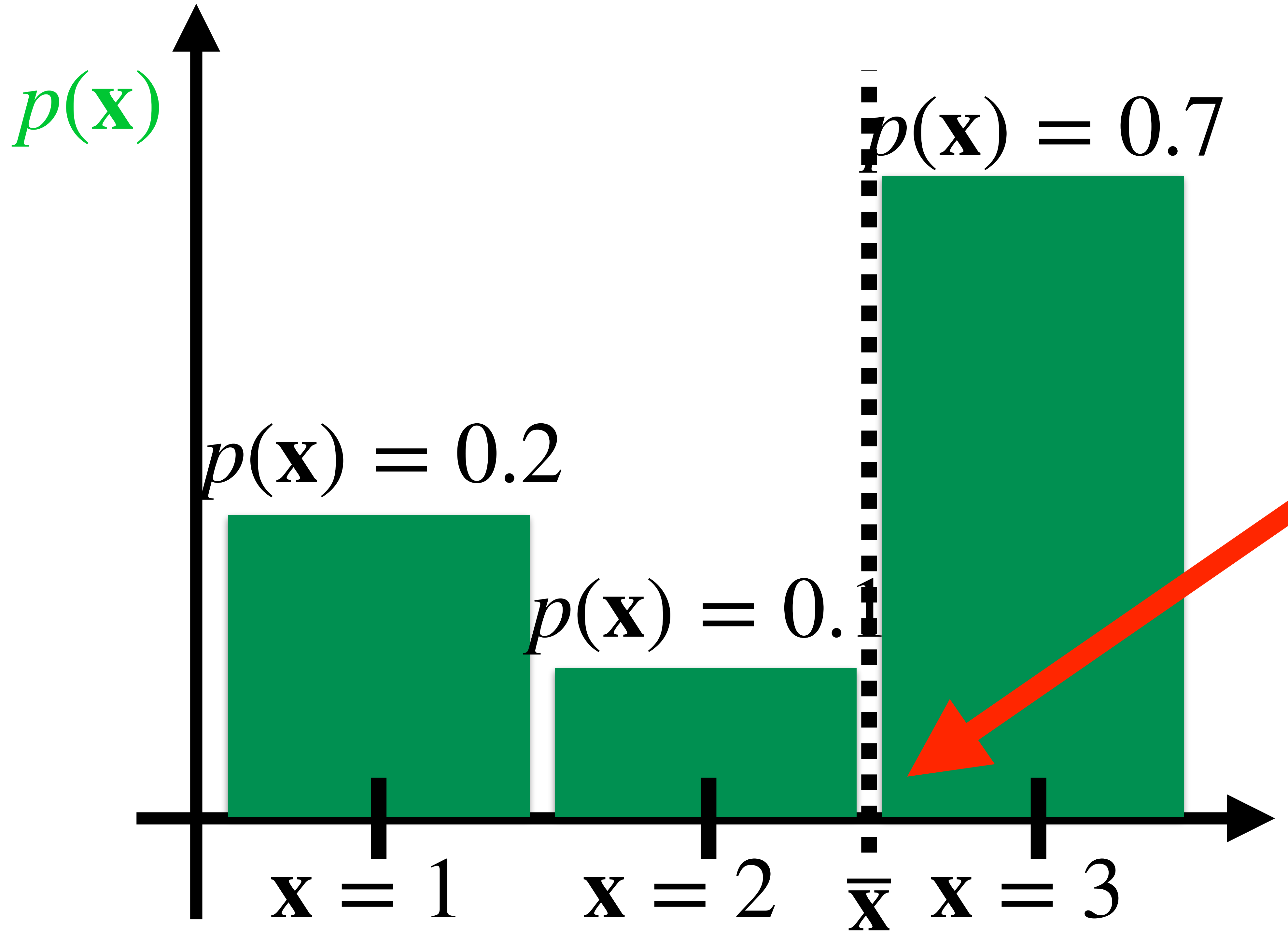


Prerequisites: Mean and average

$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}\left[\mathbf{x}\right] = 0.2 \cdot 1 \ + \ 0.1 \cdot 2 \ + \ 0.7 \cdot 3 = 2.5$$

$$\approx \frac{1}{N} \sum_{i} \mathbf{x}_i \ = \frac{1}{10}(1 + 1 + 2 + 3 + 3 + 3 + 3 + 3 + 3 + 3) = 2.5$$

where $\mathbf{x}_i \sim p$



Prerequisites: Mean and average

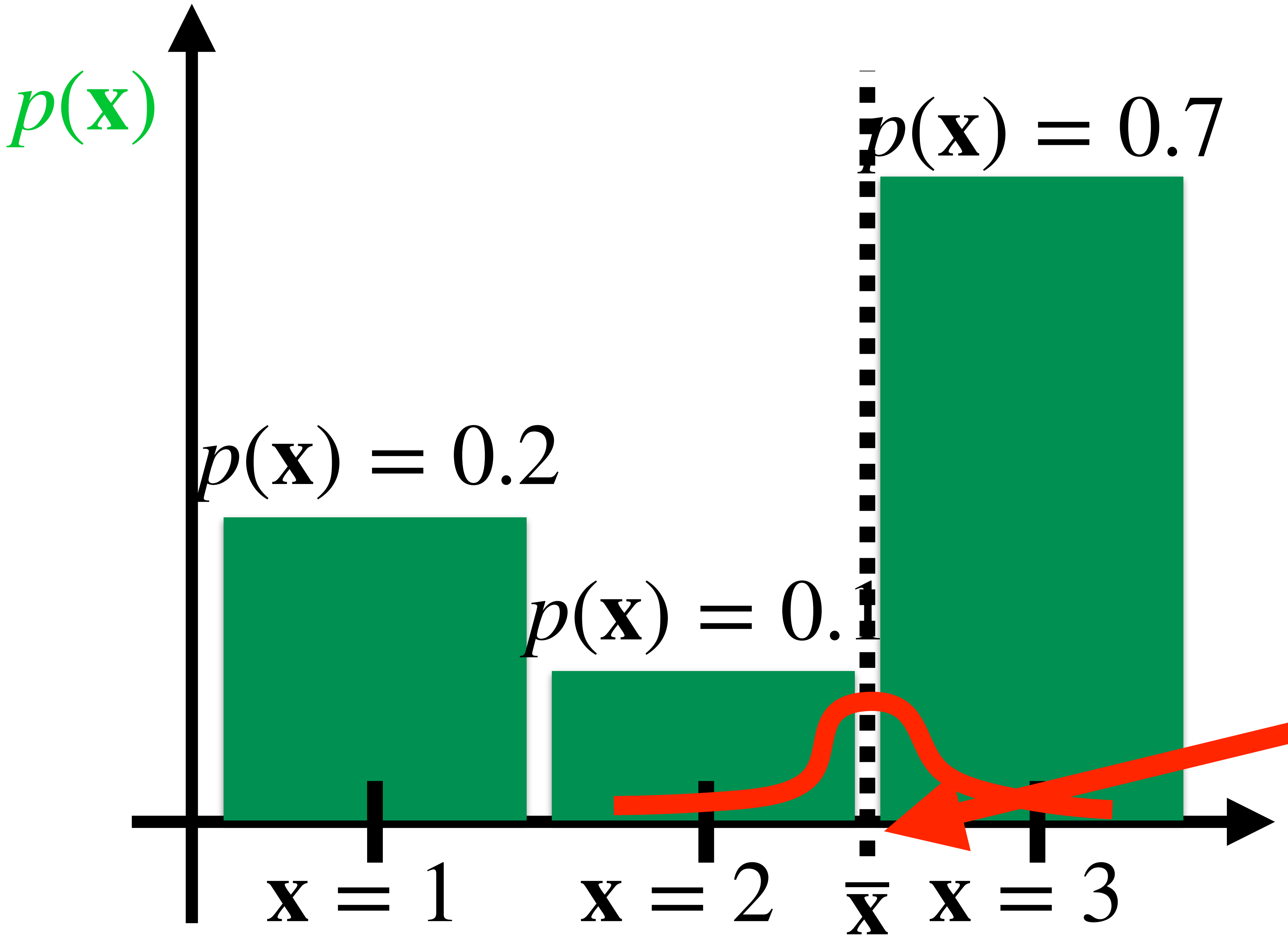
$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}[\mathbf{x}] = 0.2 \cdot 1 \ + \ 0.1 \cdot 2 \ + \ 0.7 \cdot 3 = 2.5$$

$$\approx \frac{1}{N} \sum_i \mathbf{x}_i \ = \frac{1}{10}(1 + 1 + 2 + 3 + 3 + 3 + 3 + 3 + 3 + 3) = 2.5$$

where $\mathbf{x}_i \sim p$

For $N \to \infty$

$$\mathcal{N}(\bar{\mathbf{x}}_i; \ \ \bar{\mathbf{x}}, \frac{\sigma^2_{\mathbf{x}}}{\sqrt{N}})$$



$$\bar{\mathbf{x}}_1 = \frac{1}{10}(1 + 1 + 1 + 1 + 3 + 3 + 3 + 3 + 3 + 3) = 2.2$$

$$\bar{\mathbf{x}}_2 = \frac{1}{10}(3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3) = 3.0$$

$$\bar{\mathbf{x}}_3 = \frac{1}{10}(2 + 2 + 2 + 2 + 3 + 3 + 3 + 3 + 3 + 3) = 2.6$$

$$\bar{\mathbf{x}}_4 = \frac{1}{10}(1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 2 + 2) = 1.2$$

$$\bar{\mathbf{x}}_5 = \frac{1}{10}(1 + 1 + 1 + 1 + 1 + 3 + 3 + 3 + 3 + 3) = 2.0$$

Prerequisites: Learning vs optimization

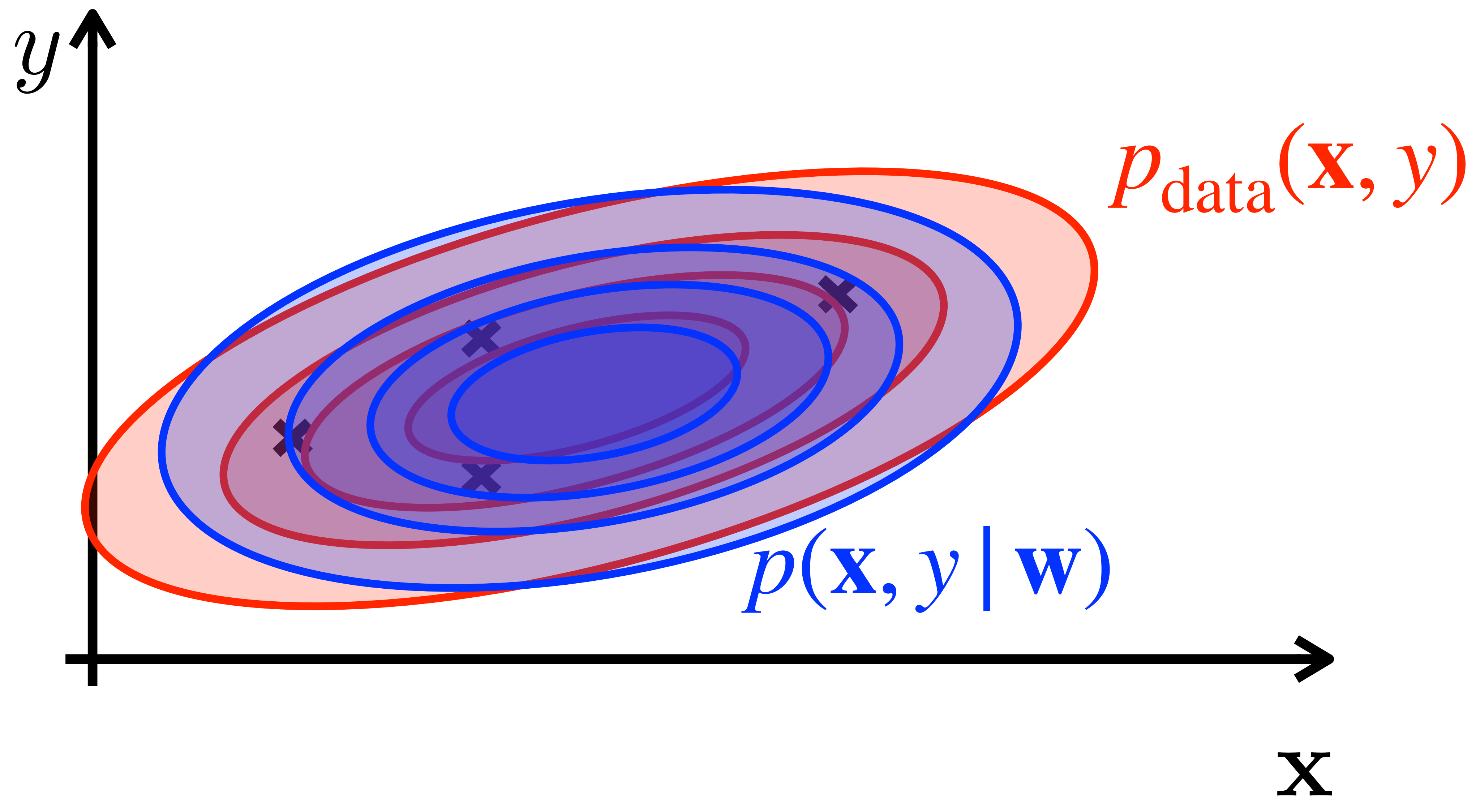
$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} D_{KL}\big(p_{\text{data}}(\mathbf{x}, y) \,\|\, p(\mathbf{x}, y \,|\, \mathbf{w})\big) = \int_{(\mathbf{x},y)} p_{\text{data}}(\mathbf{x}, y) \cdot \log \frac{p_{\text{data}}(\mathbf{x}, y)}{p(\mathbf{x}, y \,|\, \mathbf{w})}$$

$$= \arg\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\log \frac{p_{data}(\mathbf{x}, y)}{p(\mathbf{x}, y \,|\, \mathbf{w})}\right] = \arg\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[-\log p(\mathbf{x}, y \,|\, \mathbf{w})\right]$$

$$= \arg\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[-\log p(y \,|\, \mathbf{x}, \mathbf{w})\right] \approx \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{(\mathbf{x}_i, y_i) \sim p_{\text{data}}(\mathbf{x}, y)} \left[-\log p(y_i \,|\, \mathbf{x}_i, \mathbf{w})\right]$$

True criterium we want to maximize:

$$J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[-\log p(y \,|\, \mathbf{x}, \mathbf{w})\right]$$



True gradient:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[-\nabla_{\mathbf{w}} \log\big(p(y \,|\, \mathbf{x}, \mathbf{w})\big)\right]$$

$$\approx \frac{1}{N} \sum_i -\nabla_{\mathbf{w}} \log\big(p(y_i \,|\, \mathbf{x}_i, \mathbf{w})\big)$$

$|\mathcal{T}| < \infty$: Learning from a finite dataset

- Is the minibatch gradient unbiased estimate of the true gradient if $|\mathcal{T}| < \infty$?
- I want to maximize $J(\mathbf{w})$
- I recycle samples from $\mathcal{T}$ => Criterion $\hat{J}(\mathbf{w})$ is biased by the training set
- This causes overfitting, requires strong priors and data augmentation !!!!

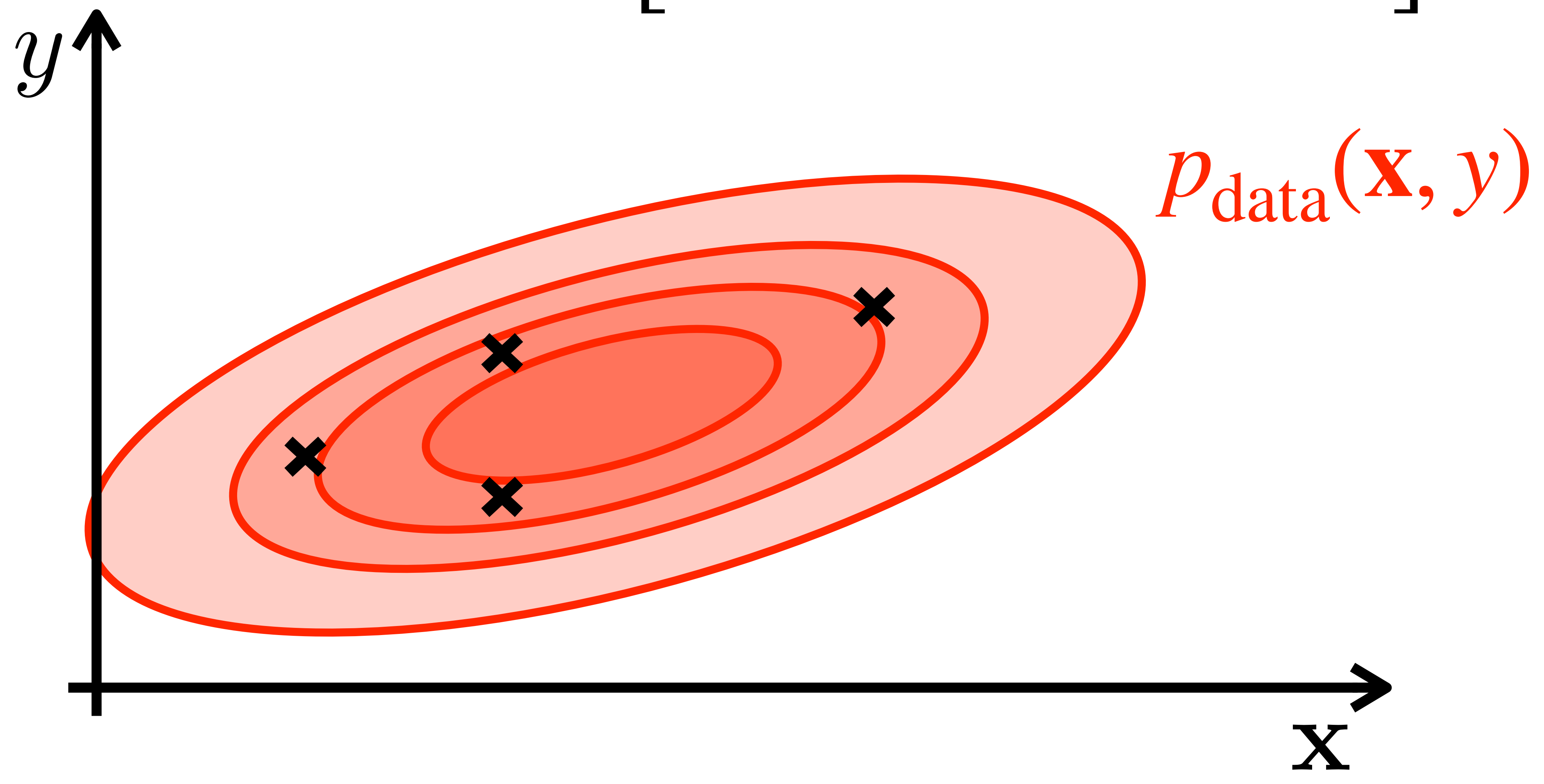
$$J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\log p(\mathbf{x}, y \,|\, \mathbf{w})\right]$$

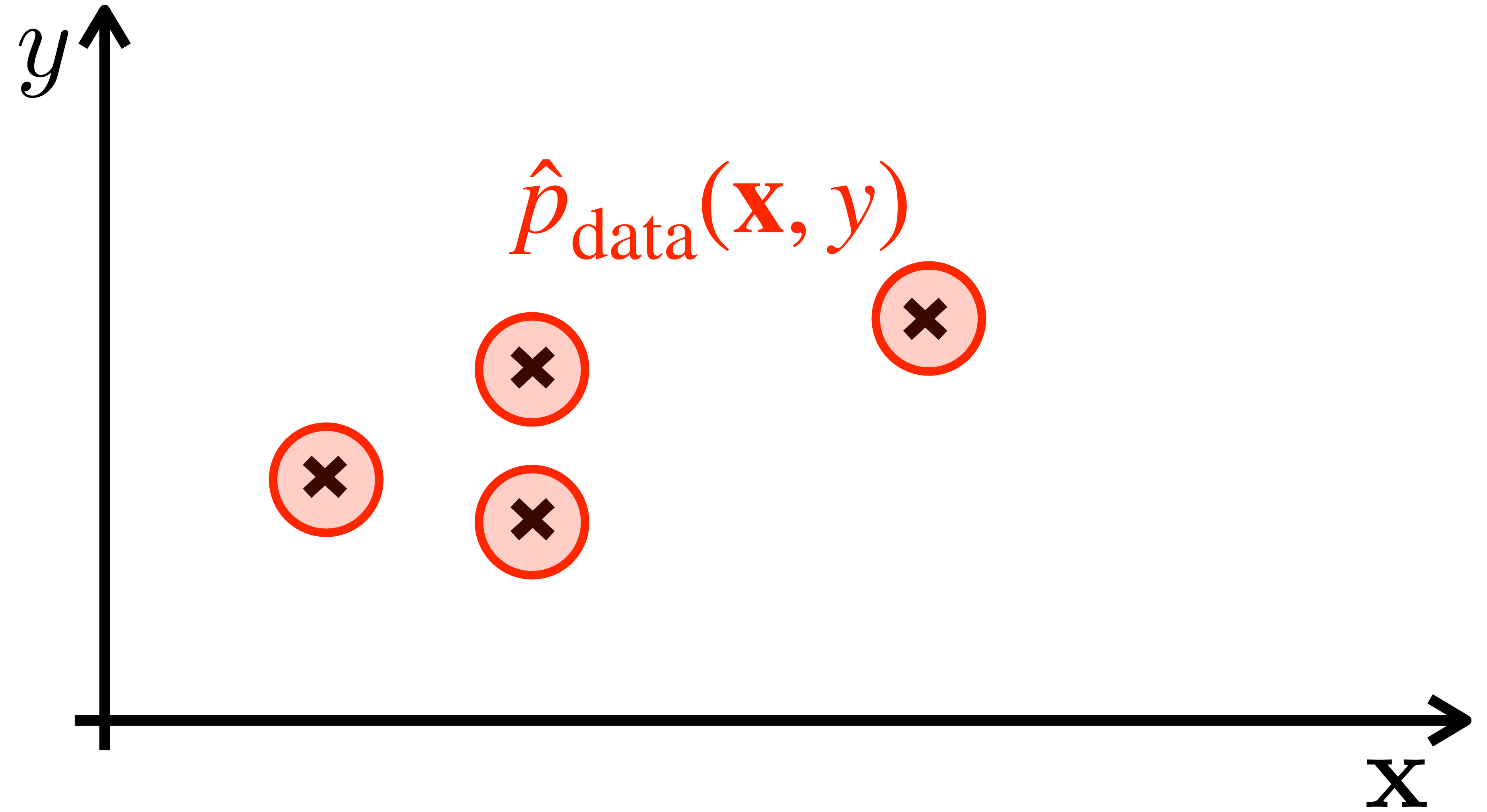
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \,|\, \mathbf{w})\right)\right]$$

vs.

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim \hat{p}_{\text{data}}}\left[\log p(\mathbf{x}, y \,|\, \mathbf{w})\right]$$

$$\nabla_{\mathbf{w}} \hat{J}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \,|\, \mathbf{w})\right)\right]$$





$|\mathcal{T}| = \infty$: Learning from an finite dataset

- Some datasets grows faster than we can learn from them:
  - Colorizing images (1074 imgs/sec uploaded to Instagram)
  - Autonomous cars (predicting vertical acceleration from images)
  - Youtube videos (predicting keywords in comments from video)
- The learning bottleneck stems from computational limitations (not from trn size).
- We always learn from a new “not-yet-seen” mini batch. => Gradient is unbiased estimate => Perform SGD on the true generalization error.

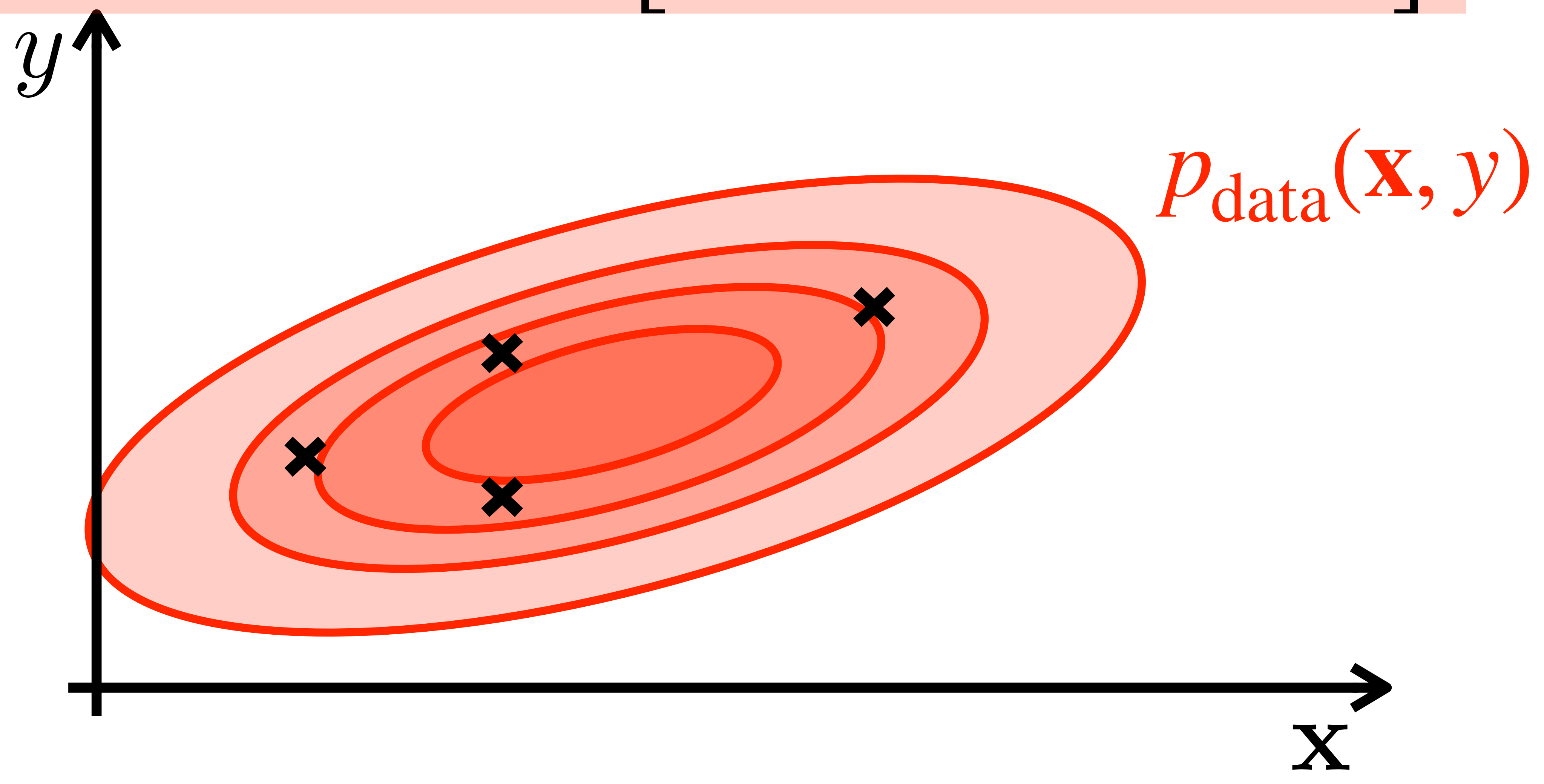
$$J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\log p(\mathbf{x}, y \,|\, \mathbf{w})\right]$$

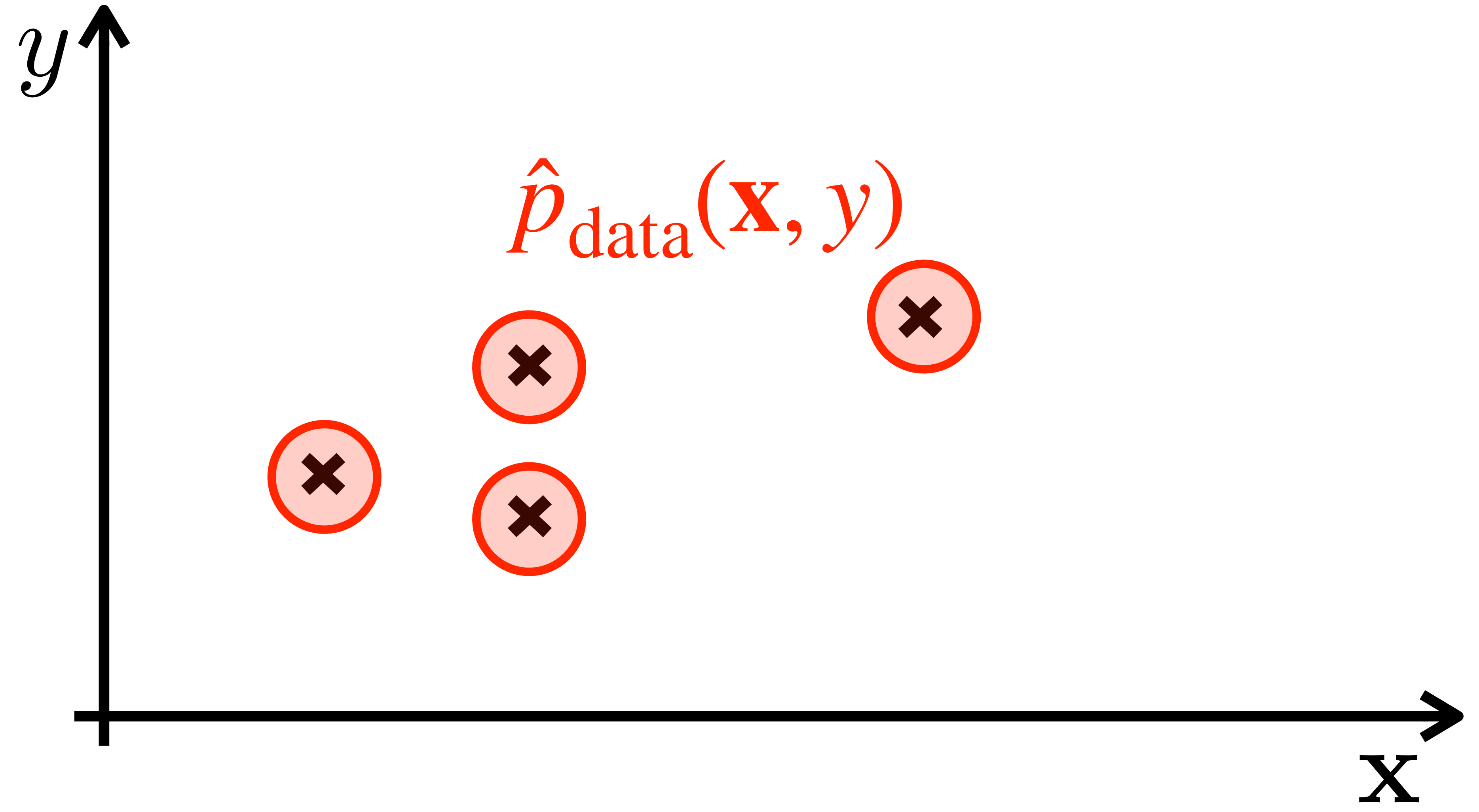
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \,|\, \mathbf{w})\right)\right]$$

vs.

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim \hat{p}_{\text{data}}}\left[\log p(\mathbf{x}, y \,|\, \mathbf{w})\right]$$

$$\nabla_{\mathbf{w}} \hat{J}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \,|\, \mathbf{w})\right)\right]$$





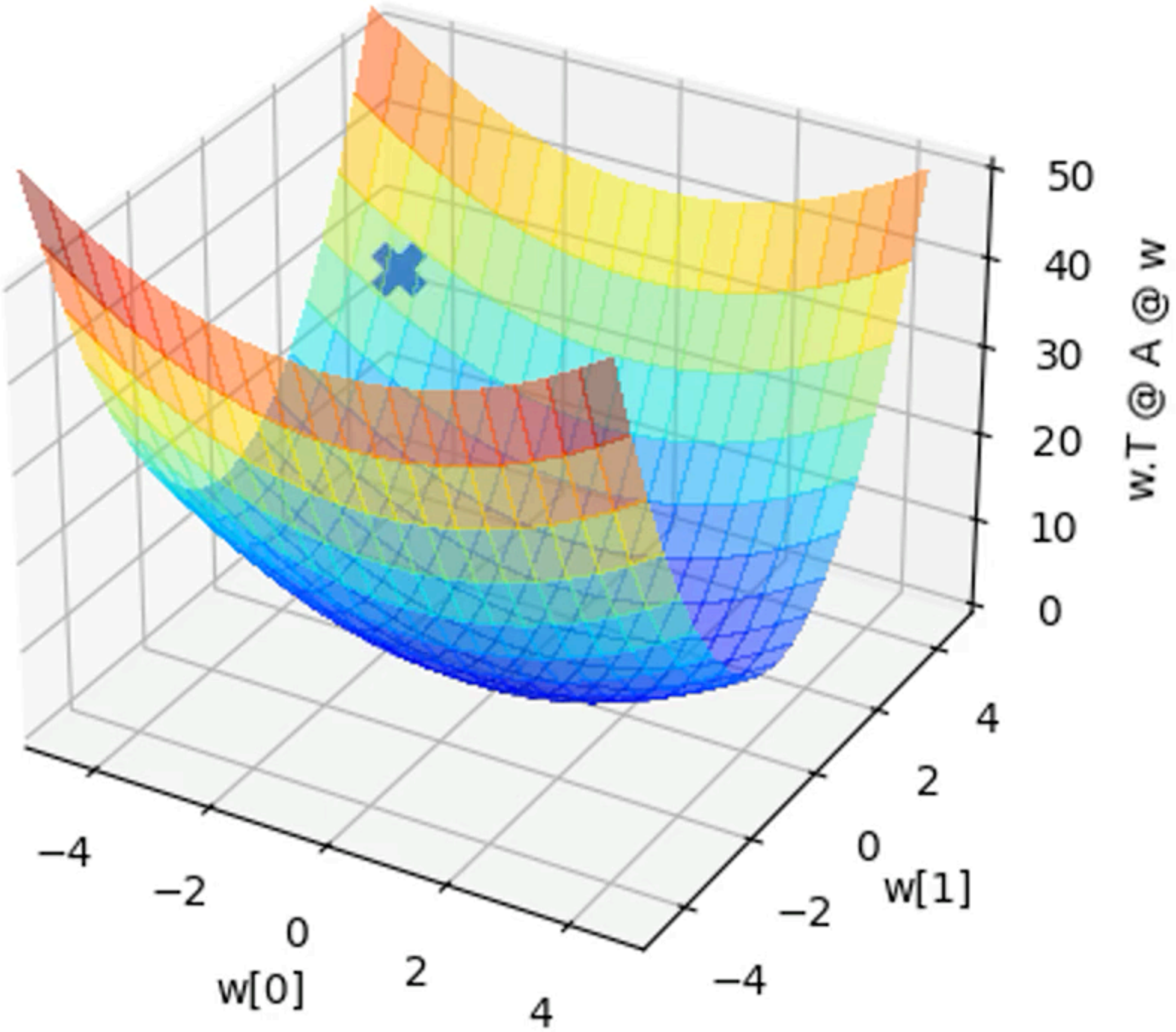
What is the disadvantage of GD on close-to-infinite dataset?

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Batches

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{A}\mathbf{w}$$

$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000}\sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A}(\mathbf{w} - \mathbf{w}_i)$$



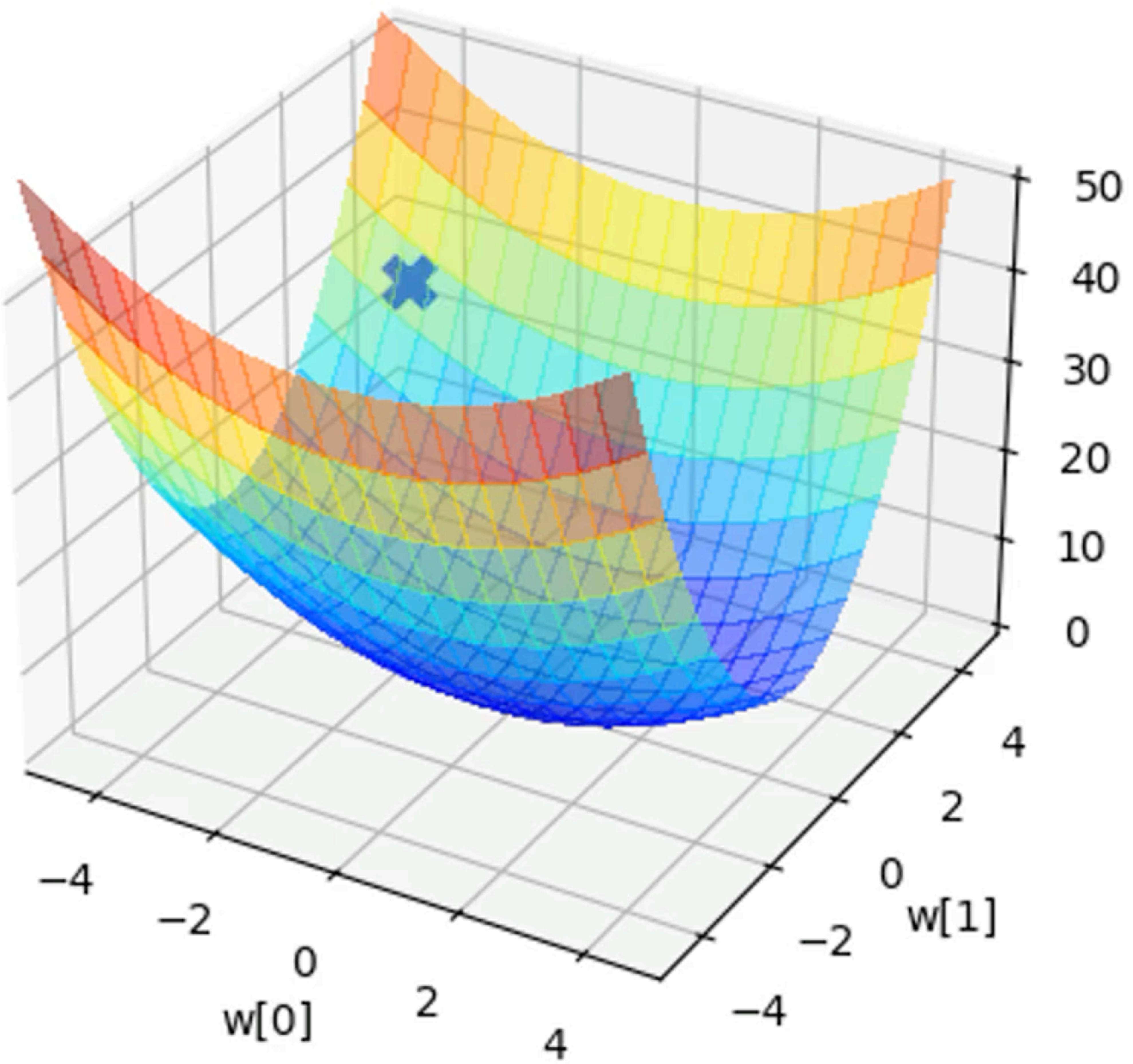
What makes it different?

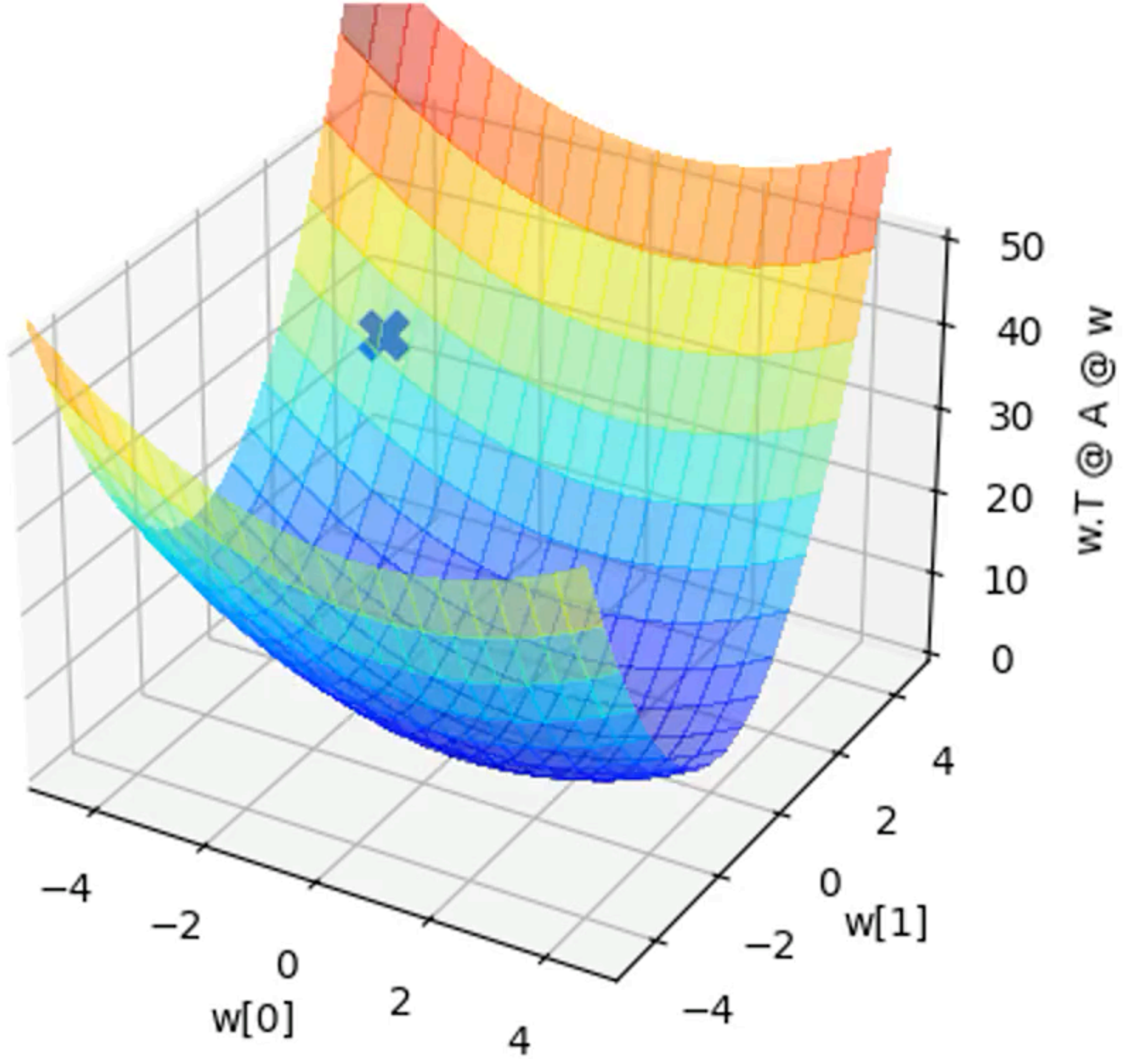
Gradient computation 1000x slower

Batches

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{A}\mathbf{w}$$

$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A}(\mathbf{w} - \mathbf{w}_i)$$





Does is worth to estimate the gradient from the full training set?

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\big(p(y\,|\,\mathbf{x},\mathbf{w})\big)\right] \approx \frac{1}{N}\sum_i \nabla_{\mathbf{w}} \log\big(p(y_i\,|\,\mathbf{x}_i,\mathbf{w})\big)$$

- Standard error of the mean estimated from N samples is $\sigma/\sqrt{N}$, where σ^2 is true variance of input samples.
- "Estimate of the gradient" based on $N = 10000$ vs $N = 100$
  - standard error is $10\times$ better
  - number of computations is $100\times$ higher !!!
- Using the large training set for estimating the gradient may suffers diminishing returns.
- Convergence in the number of computations vs number of iterations.

How should I choose N?

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\big(p(\mathbf{x}, y \,|\, \mathbf{w})\big)\right] \approx \frac{1}{N}\sum_i \nabla_{\mathbf{w}} \log\big(p(\mathbf{x}_i, y_i \,|\, \mathbf{w})\big)$$

- Large N => more accurate gradient with sub-linear returns.
- Runtime in multicore architectures is similar for small $N = 1,2,\ldots$
- Amount of required memory is linear in N (limiting factor for the most state-of-the-art hardware)
- GPU achieves better runtime with "power of 2" batch sizes.
- Small batches yields regularization.

Answer: $N \in \{4, 8, 16, 32, 64, 128, 256\}$ or anything else that works ;-)

$N = 1$ ………………… often called online learning

$1 < N <$ trn_size….. often called minibatch learning

SGD (Stochastic Gradient Descent) = GD over minibatches

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Drawbacks?

- Get stuck on flat regions
- Oscillates
- Noisy for small mini-batches

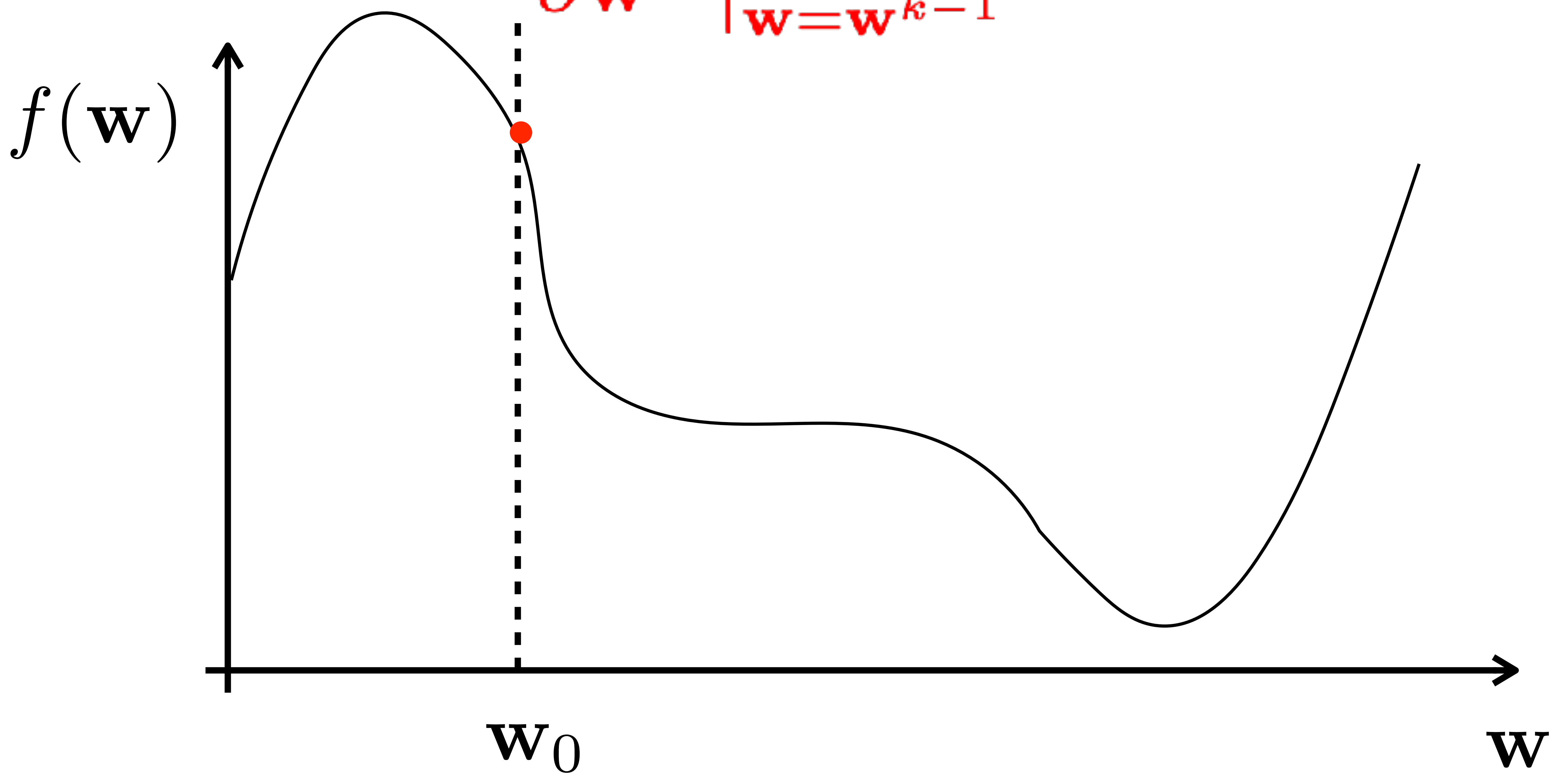
Advantages?

- Does not get stuck in saddle-points
- SGD is faster than GD

Stochastic Gradient Descent (SGD) = GD over minibatches

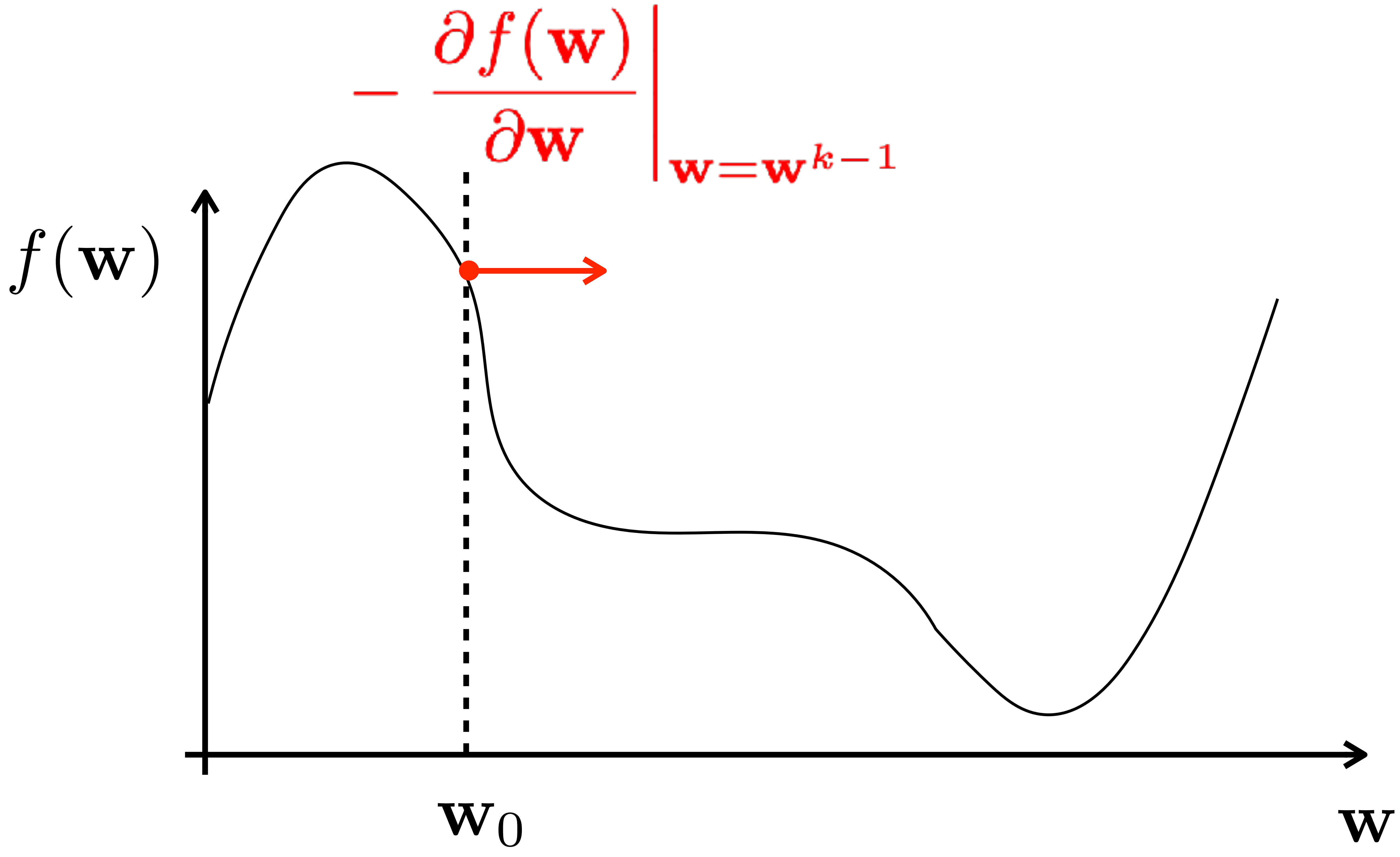
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$- \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}} = ???$$



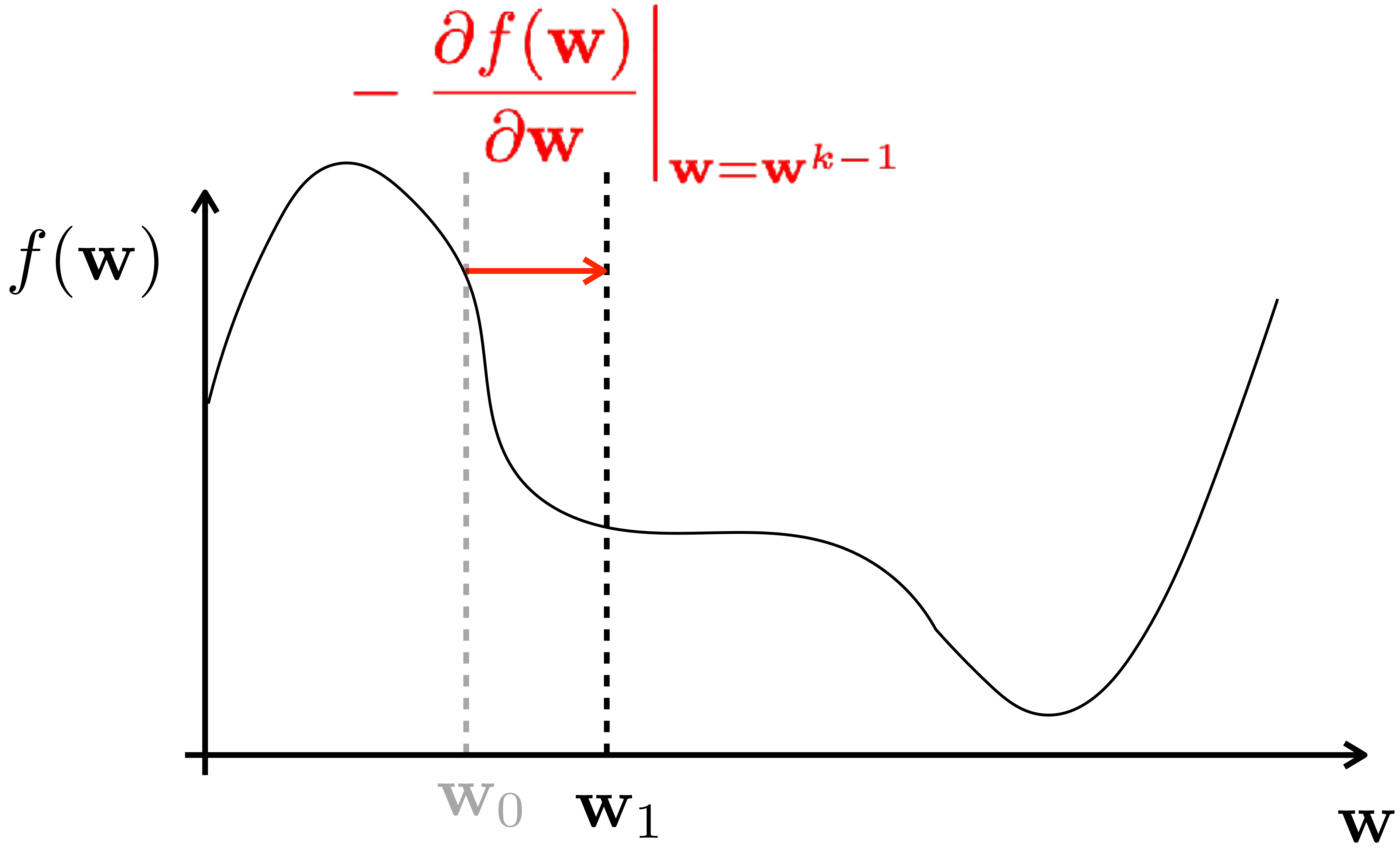
SGD drawbacks: can get stuck on flat regions

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



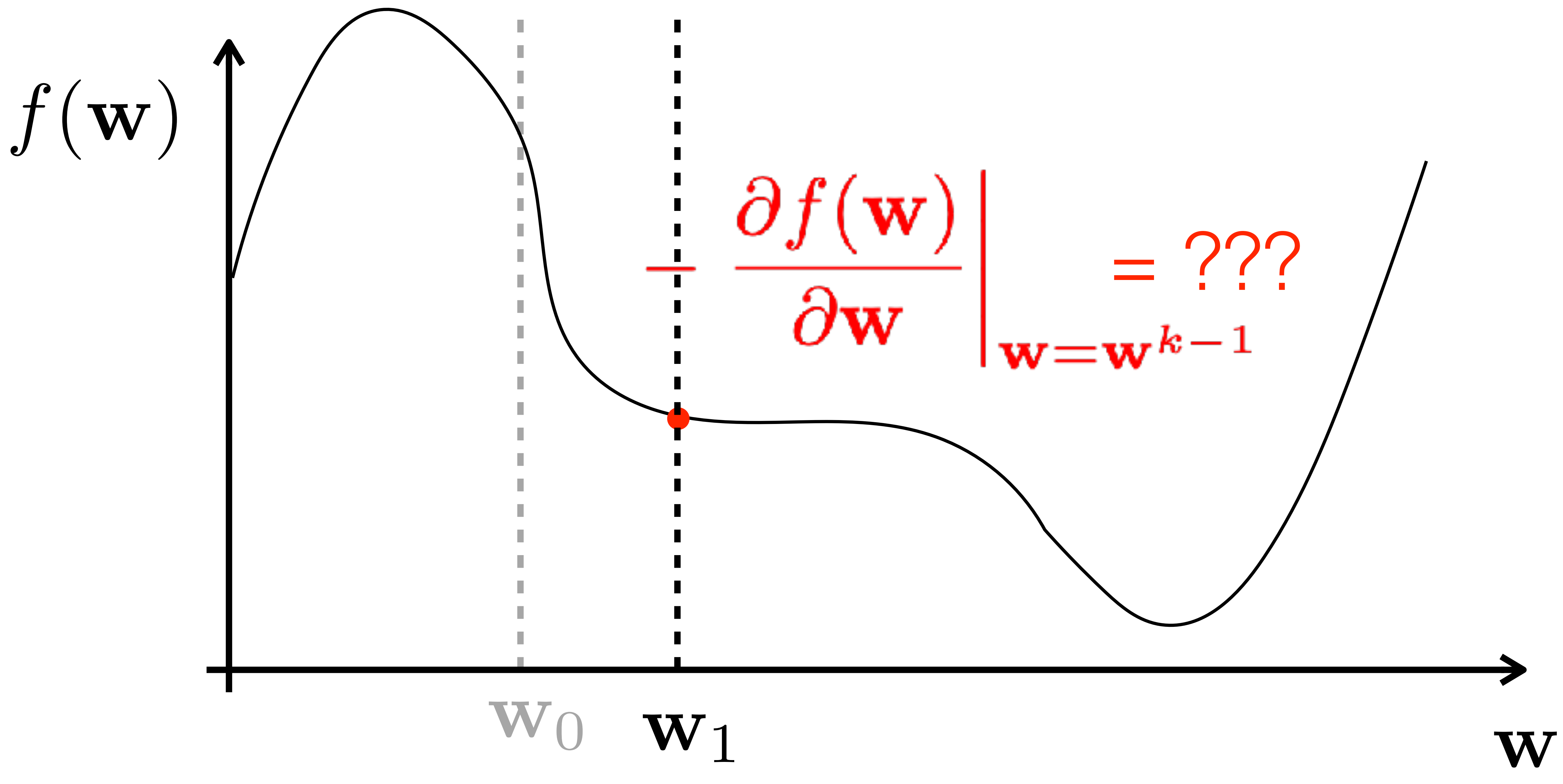
SGD drawbacks: can get stuck on flat regions

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



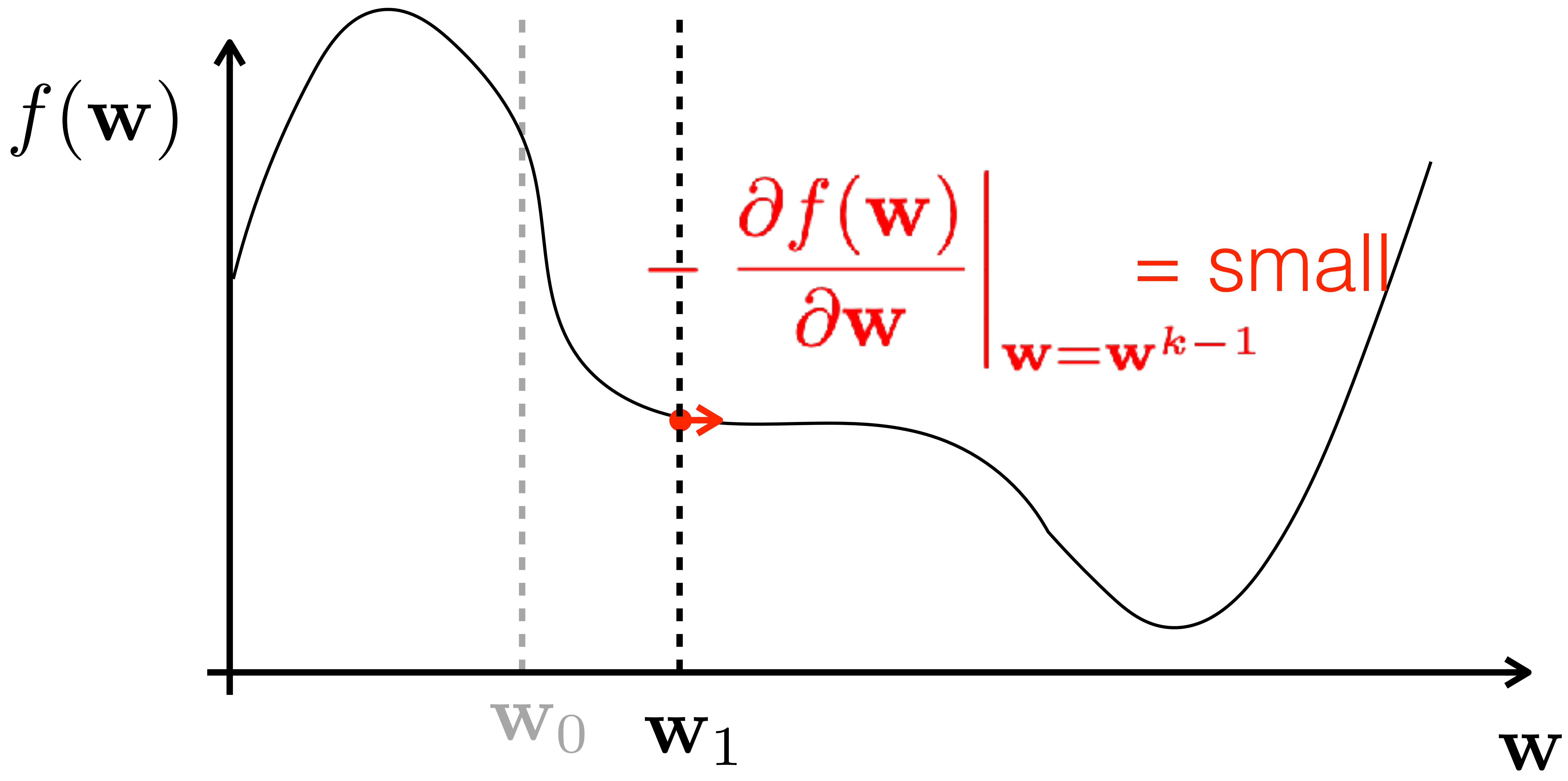
SGD drawbacks: can get stuck on flat regions

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



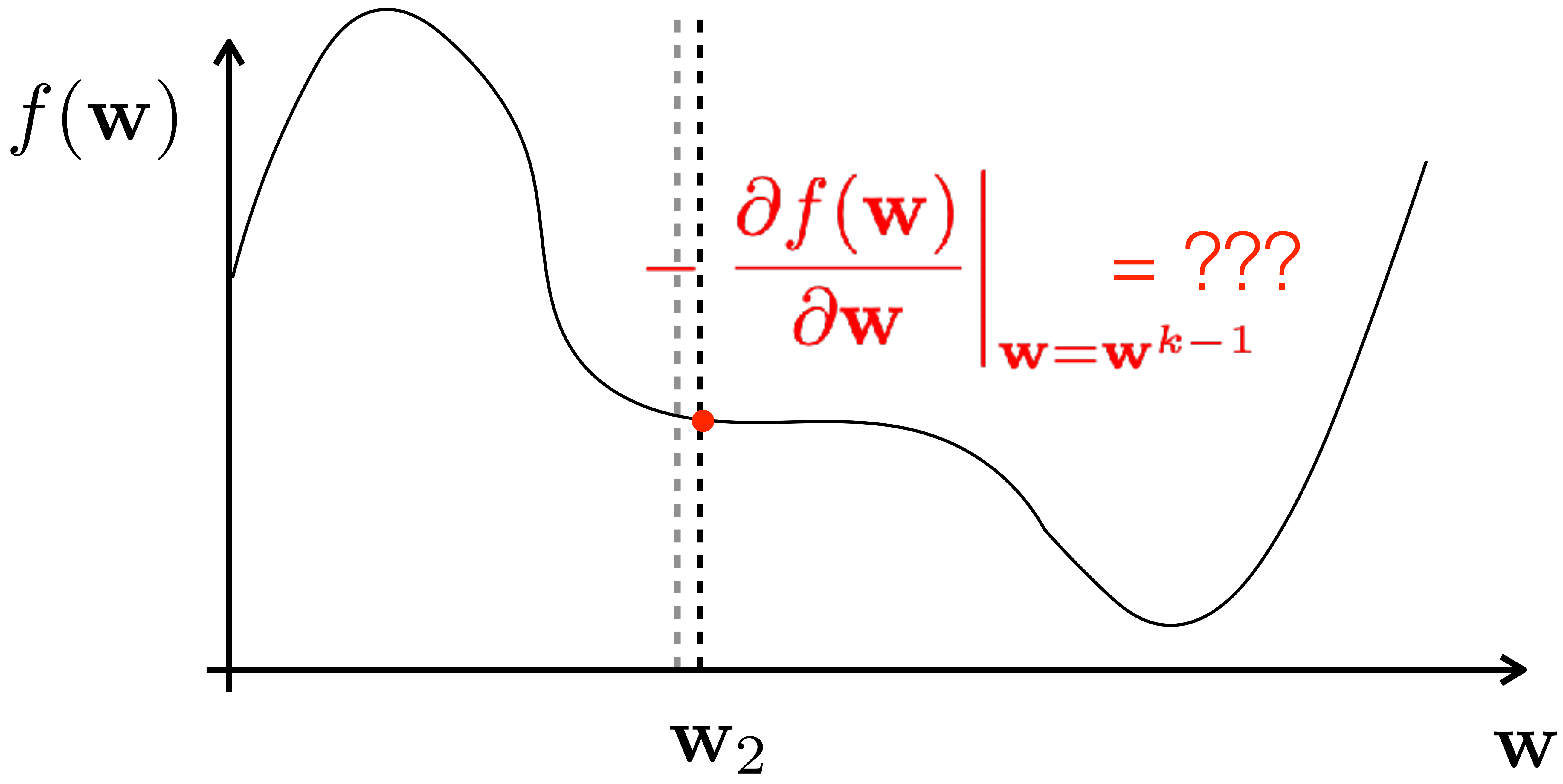
SGD drawbacks: can get stuck on flat regions

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



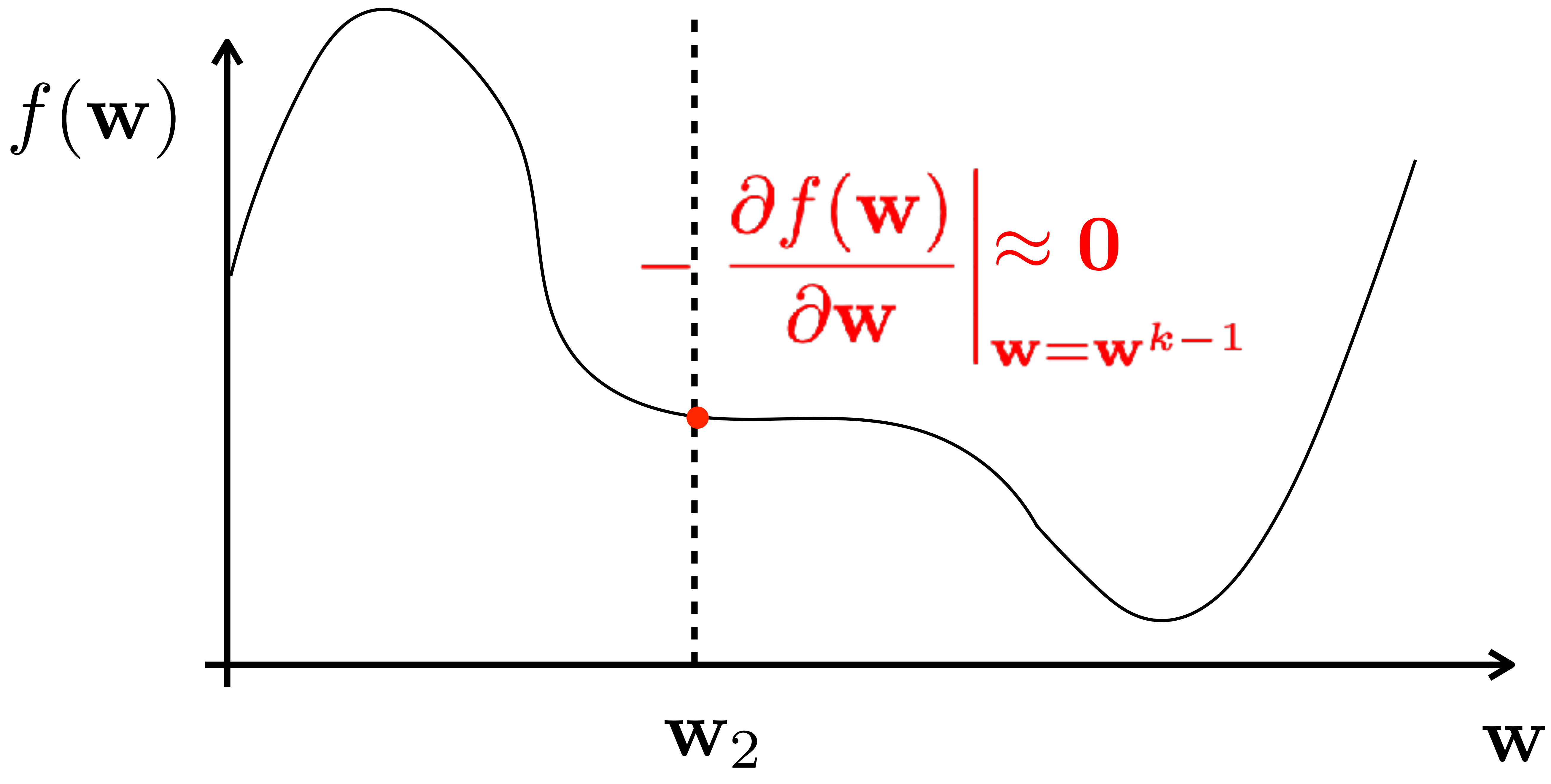
SGD drawbacks: can get stuck on flat regions

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

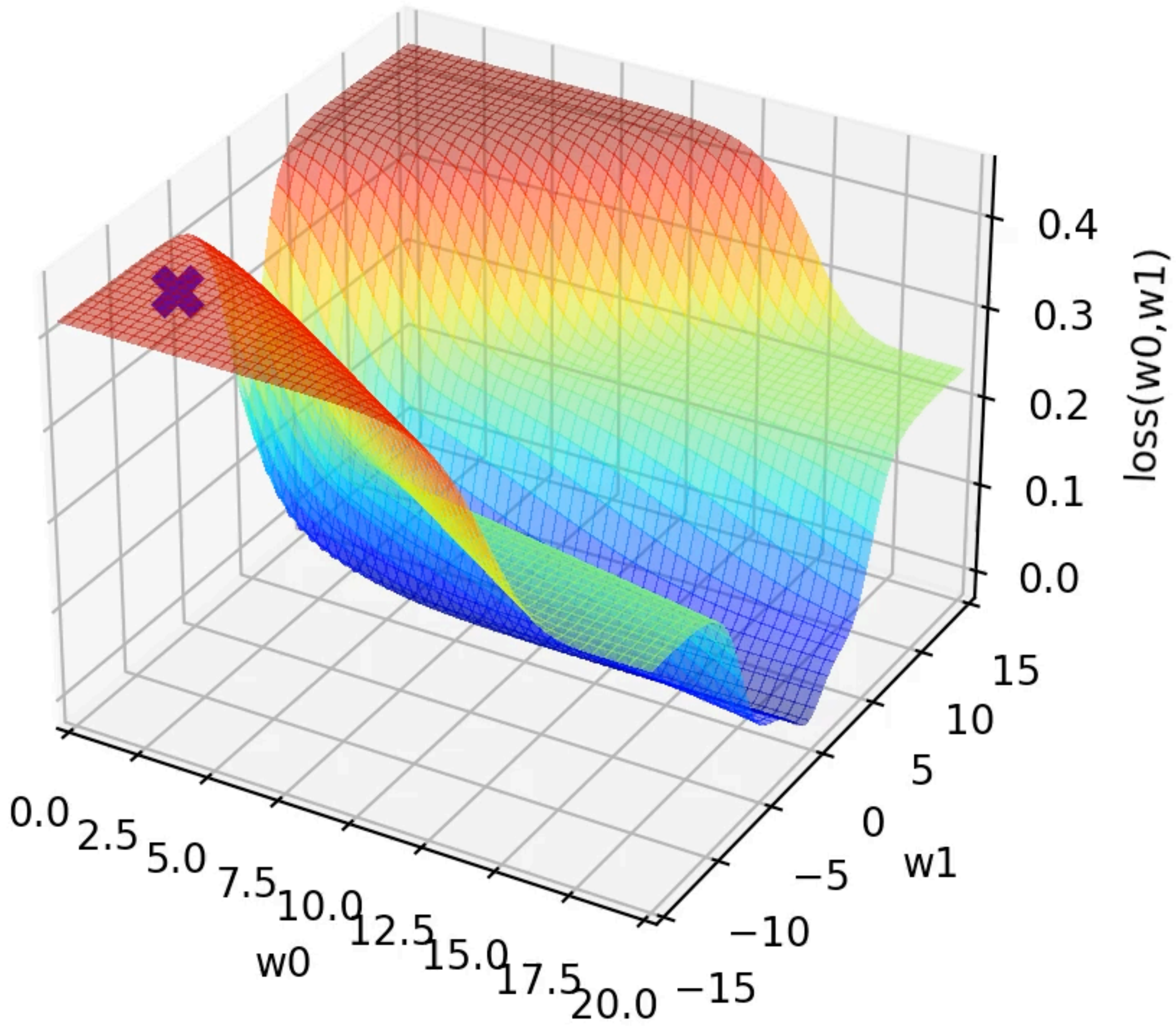


SGD drawbacks: can get stuck on flat regions

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



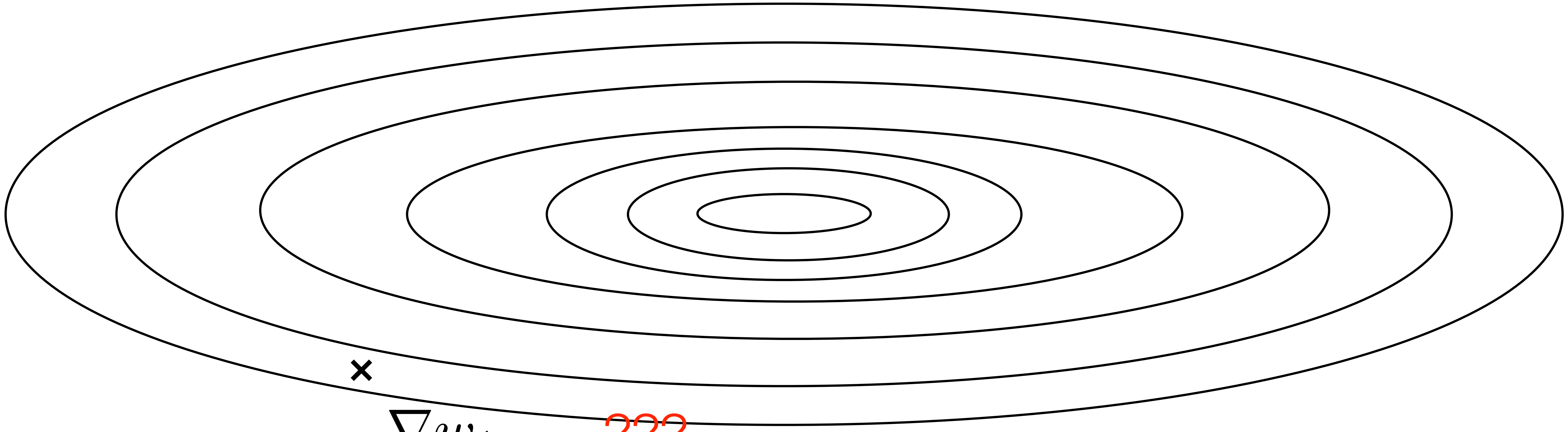
SGD drawbacks: Sigmoid fitting problem from labs



close-to-zero gradient plateaus

SGD drawbacks: oscillates

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



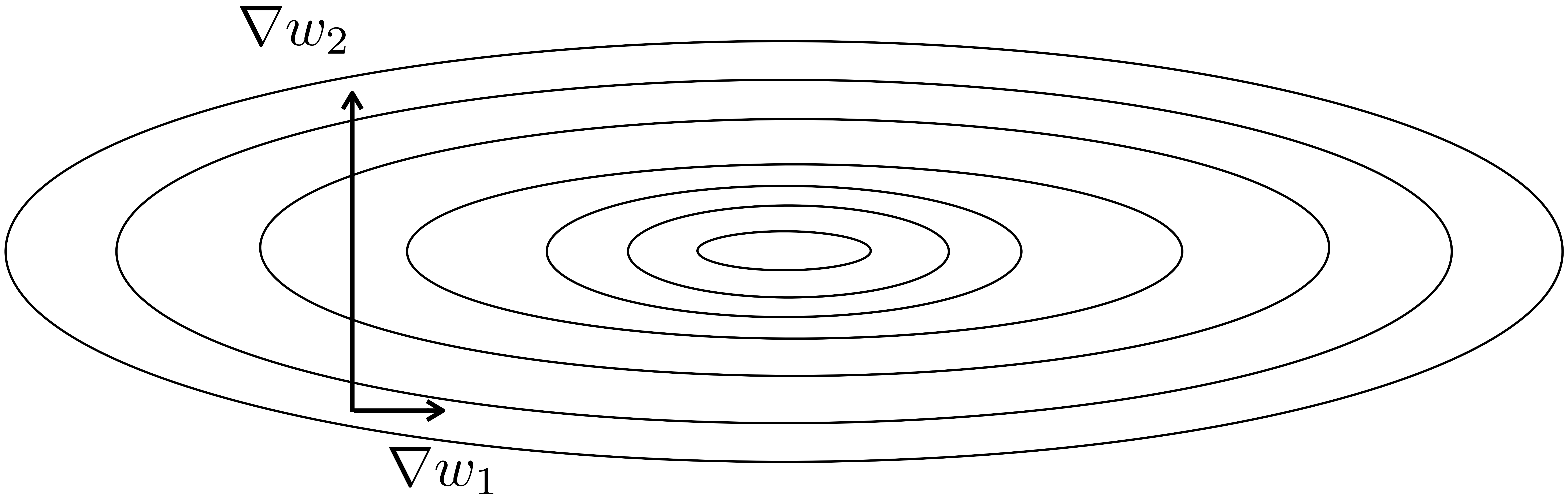
∇w_1 =???

∇w_2 =???

$$[\nabla w_1, \nabla w_2] = - \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: oscillates

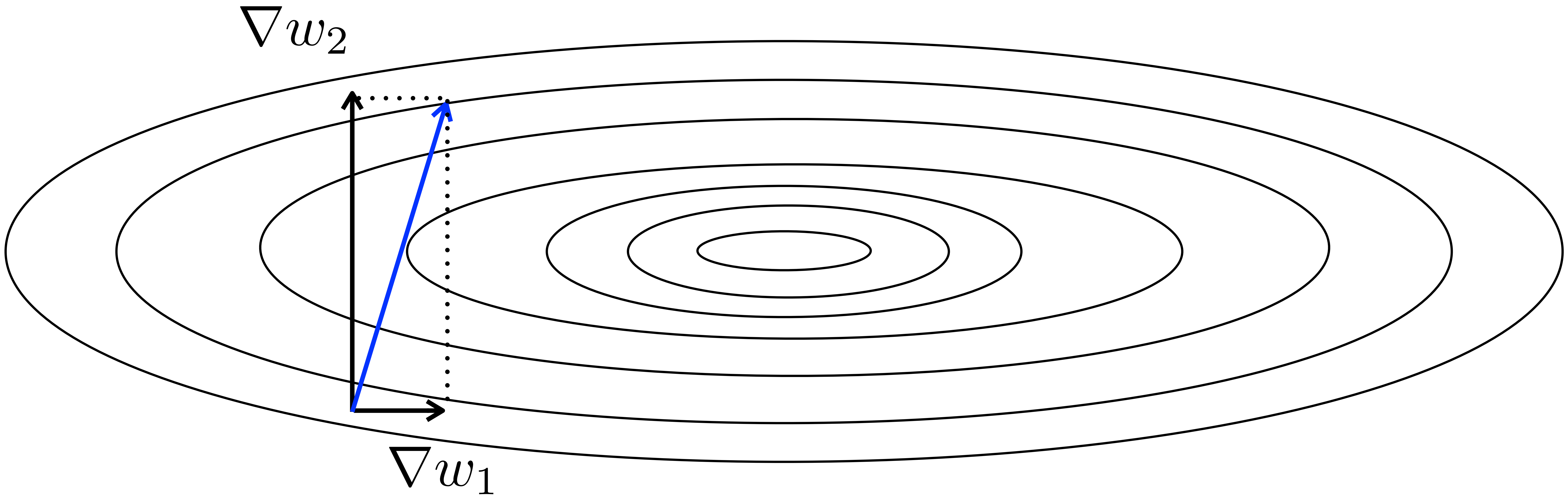
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: oscillates

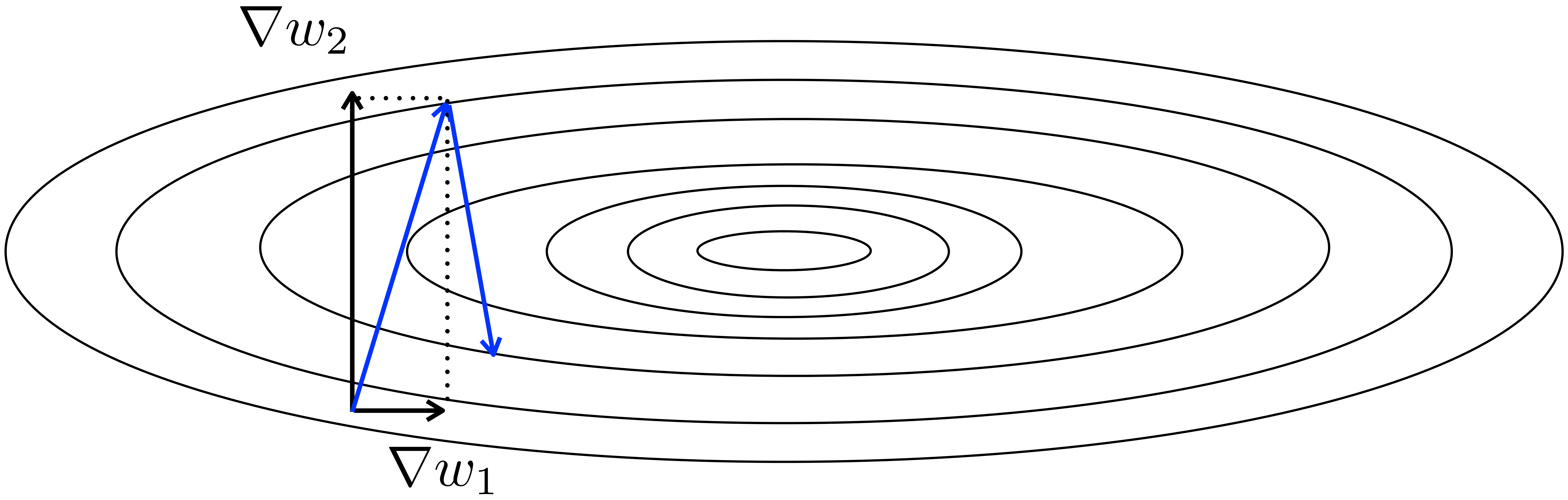
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: oscillates

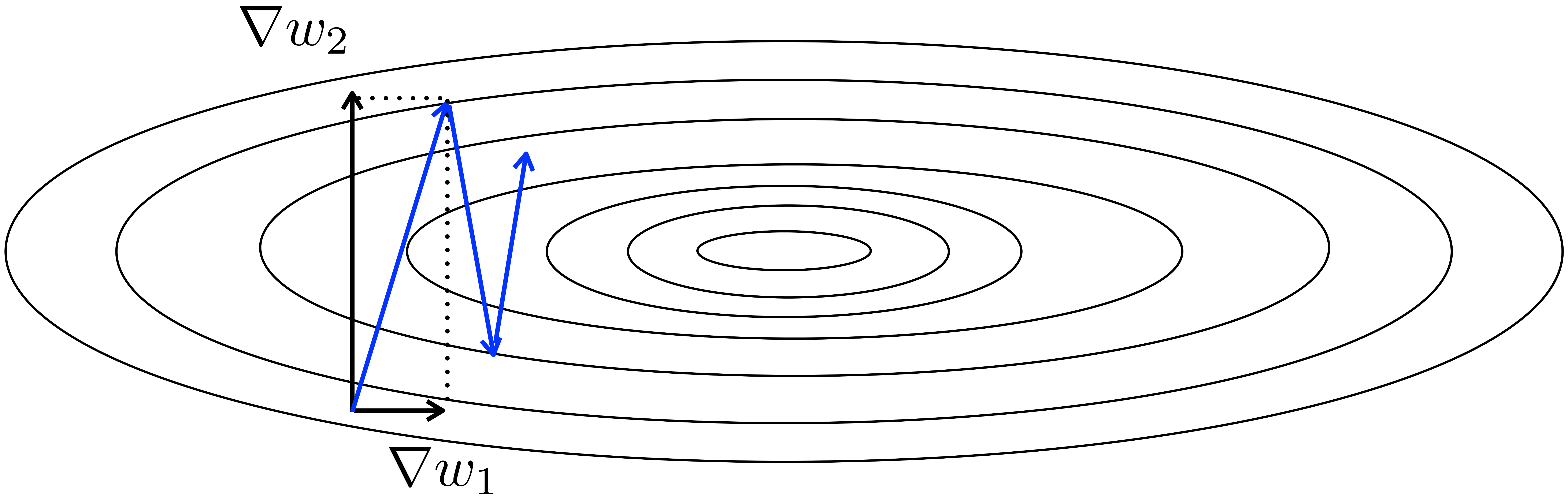
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



$$[\nabla w_1, \nabla w_2] = -\left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: oscillates

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

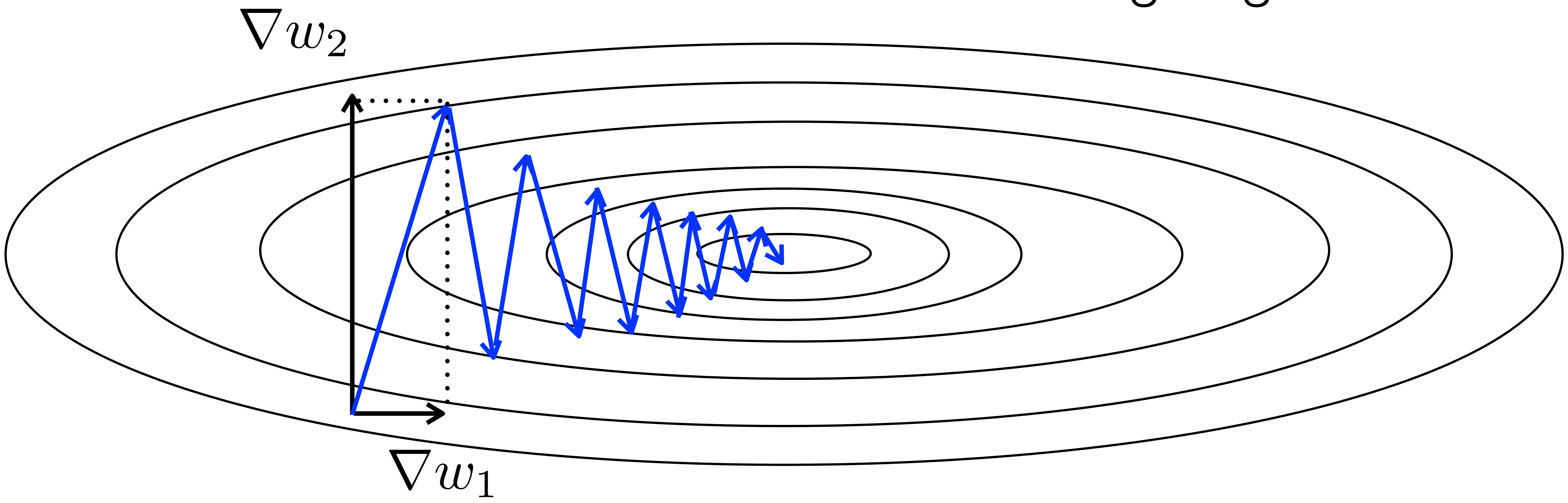


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: oscillates

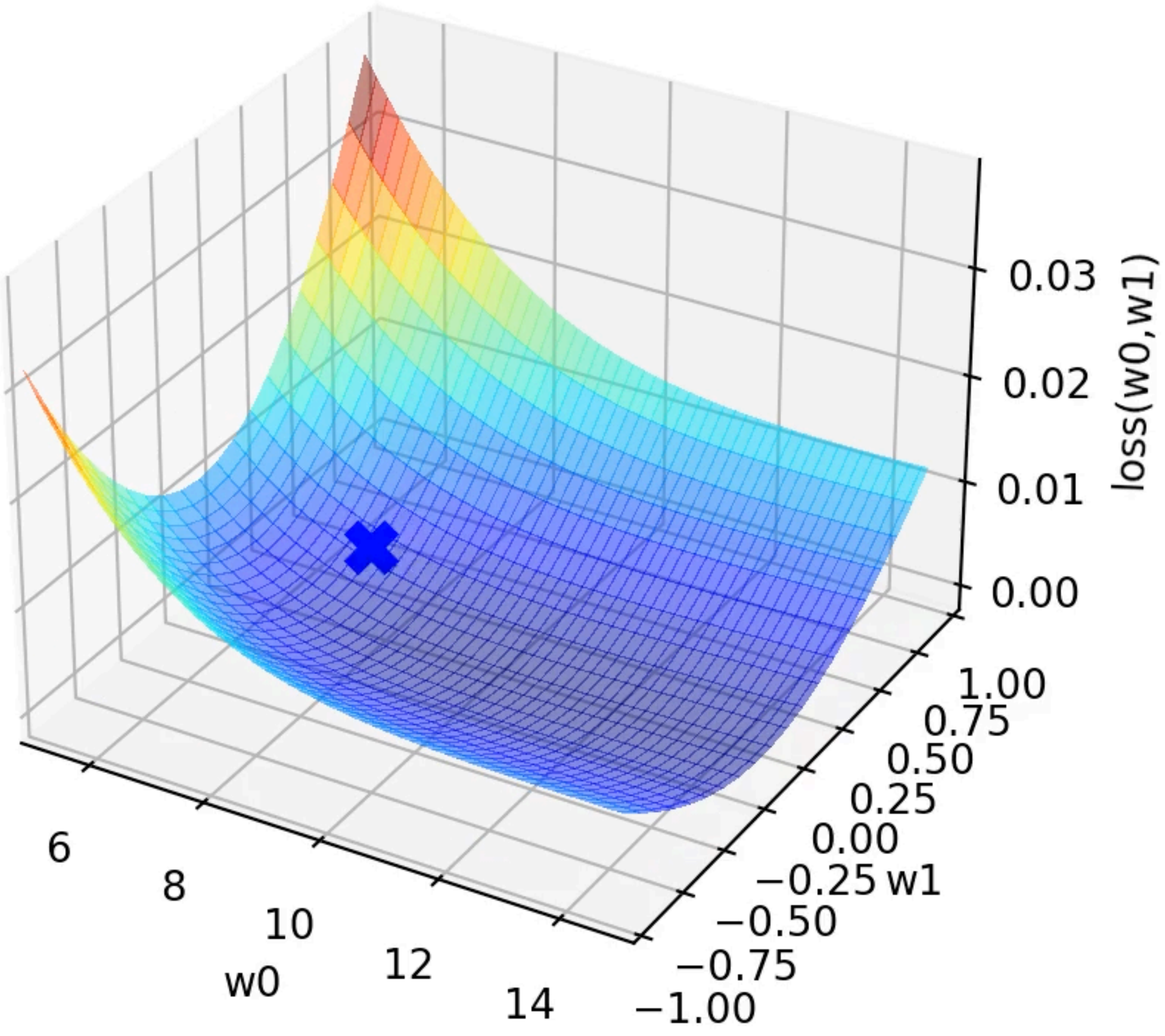
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Undesired zig-zag behaviour

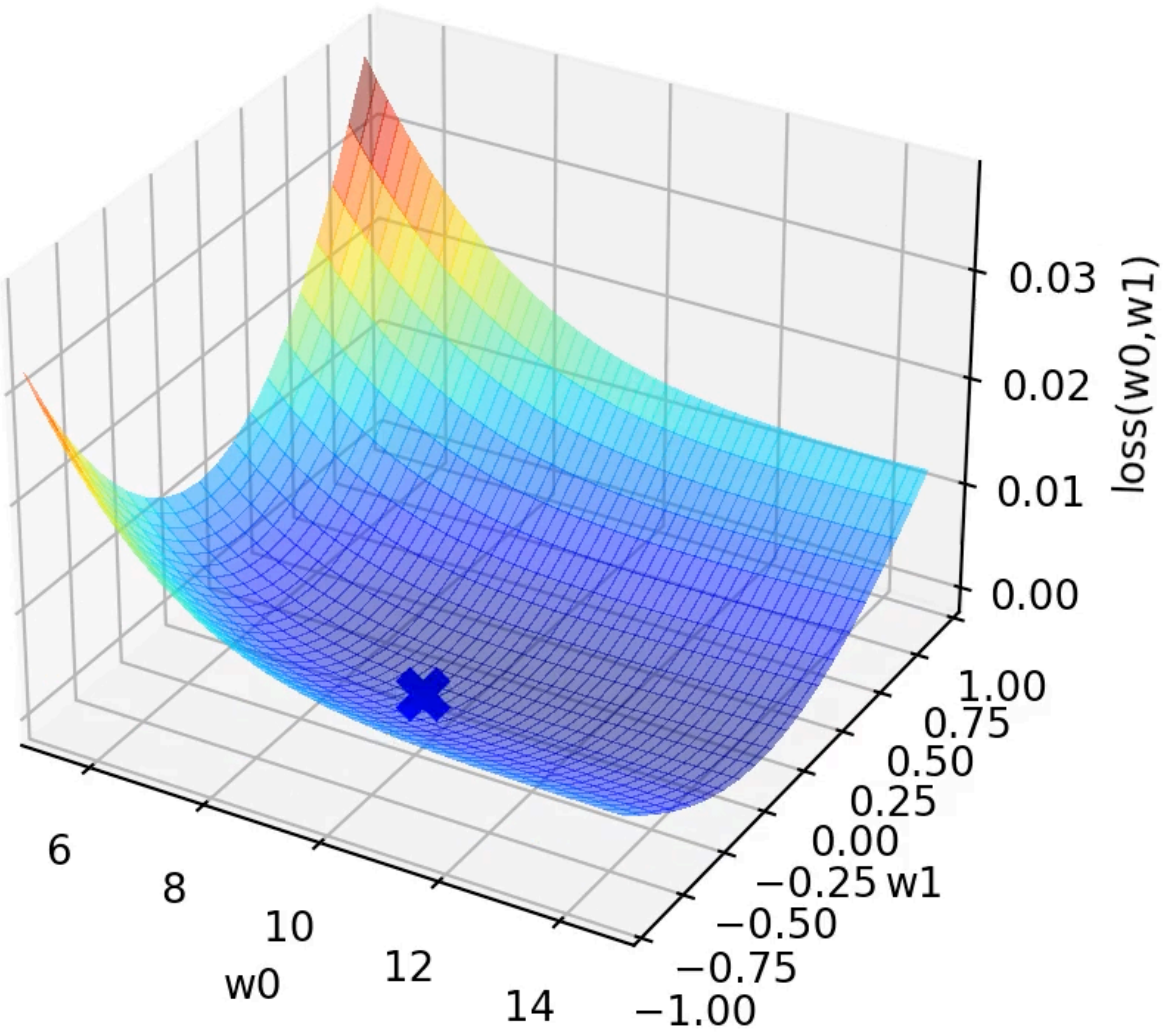


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: Sigmoid fitting problem from labs



small learning rate

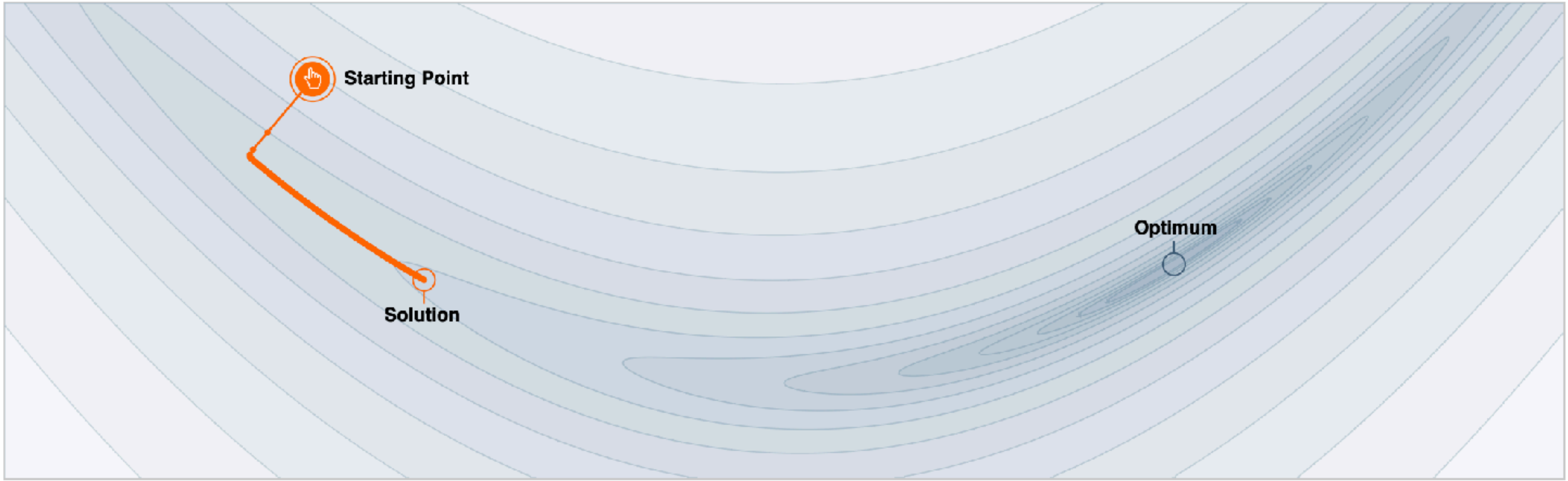


big learning rate

SGD drawbacks: oscillates

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$\alpha =$ 1e-3

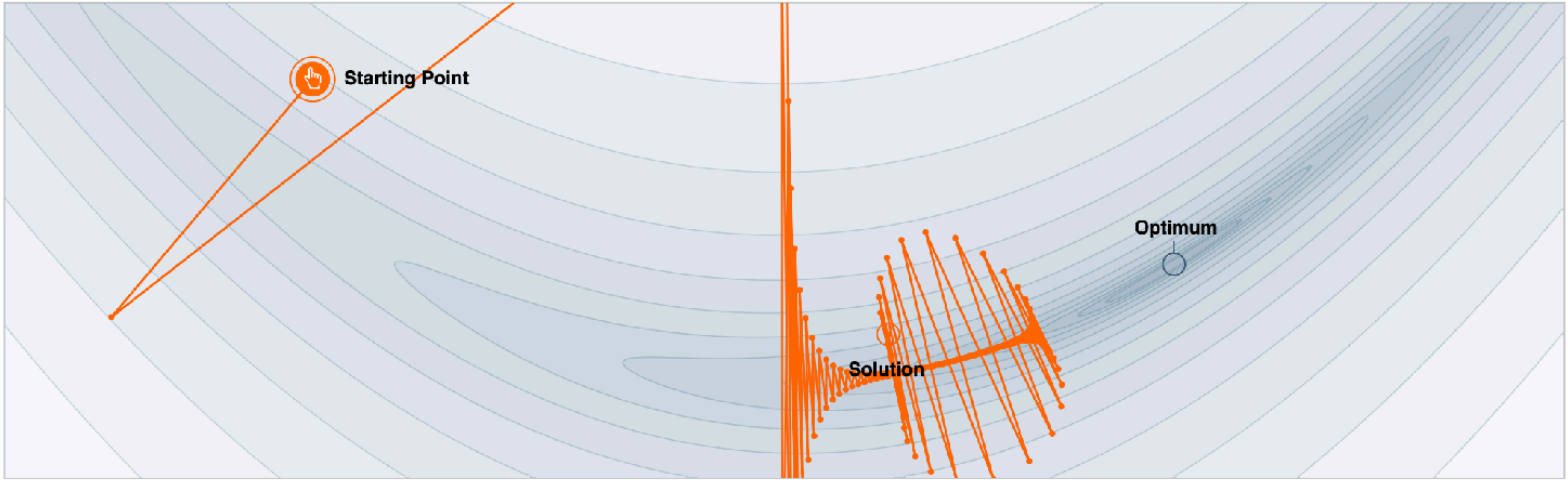


https://distill.pub/2017/momentum/

SGD drawbacks: oscillates

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$\alpha =$ 5e-3



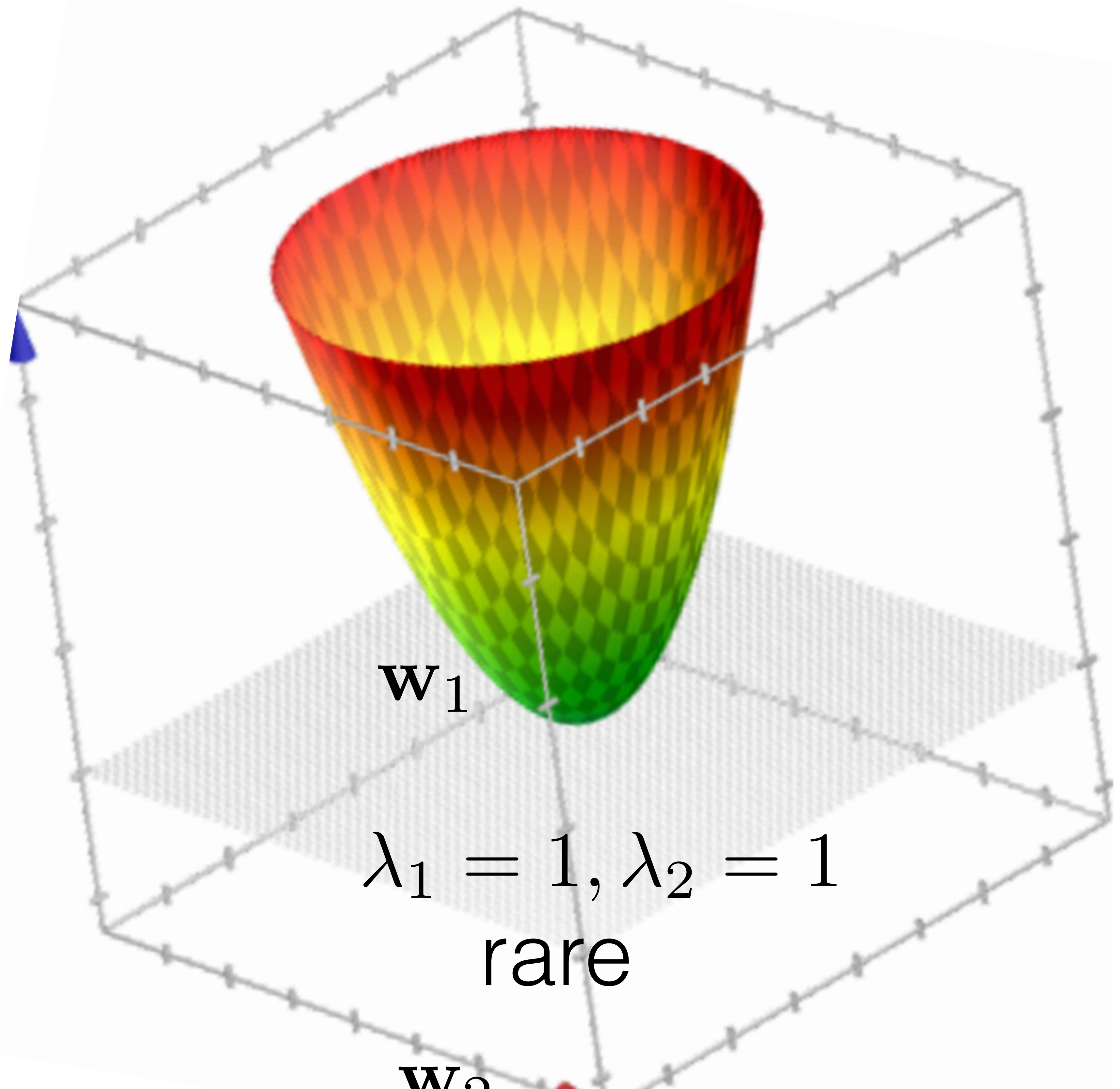
https://distill.pub/2017/momentum/

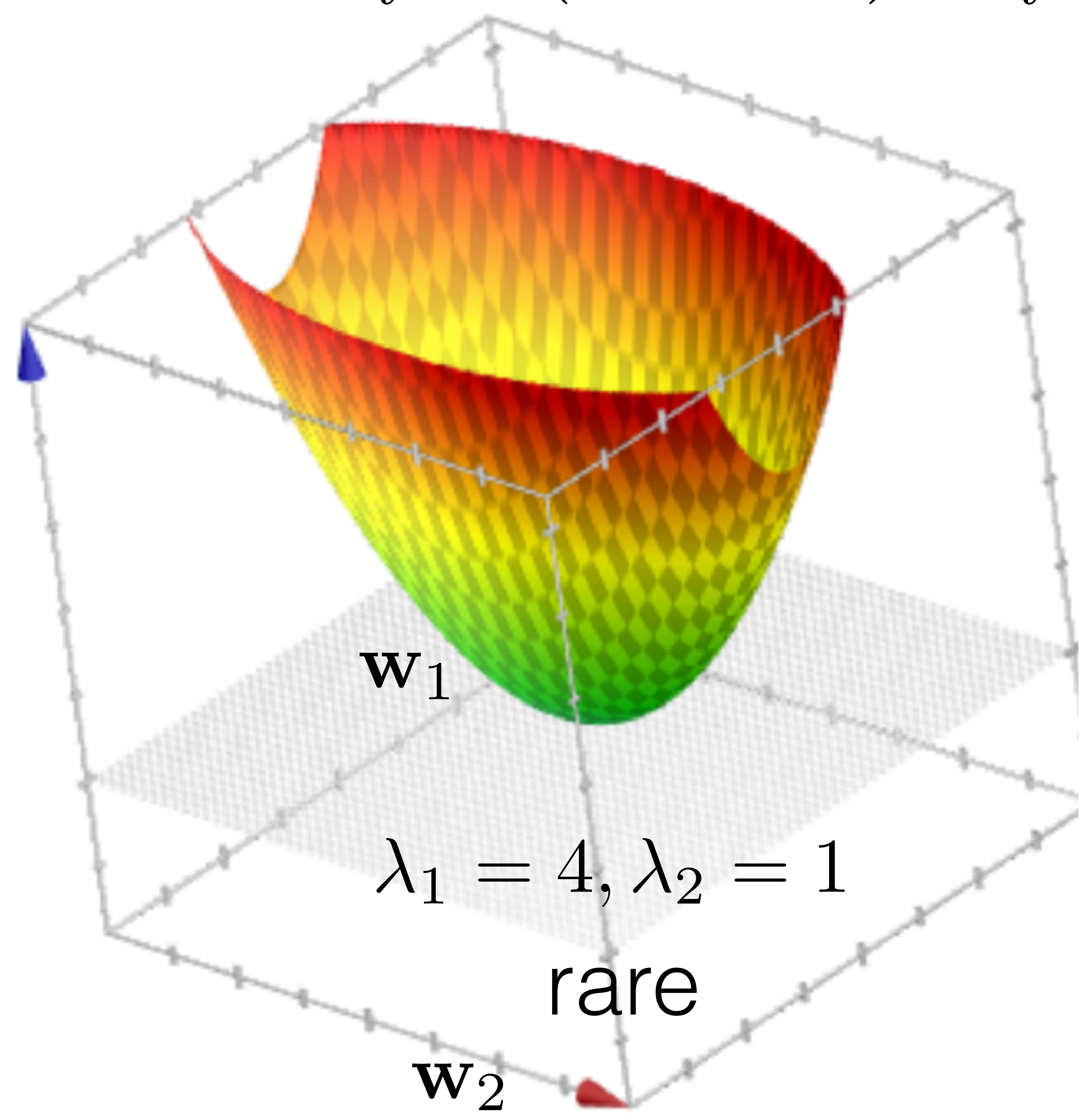
SGD on quadric

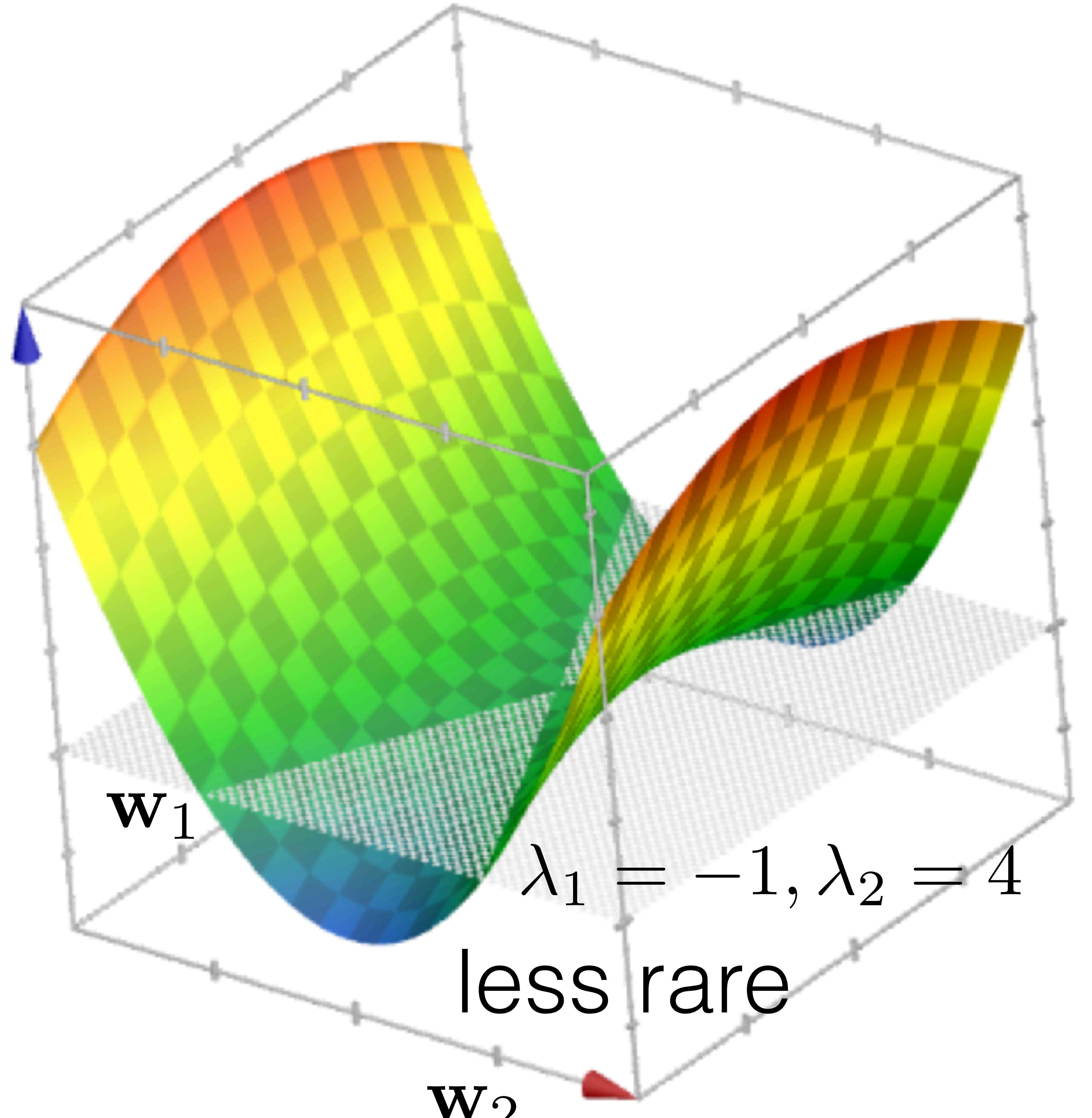
Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \dots, \lambda_n])$

Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i^{k-1}$

SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$







SGD on quadric

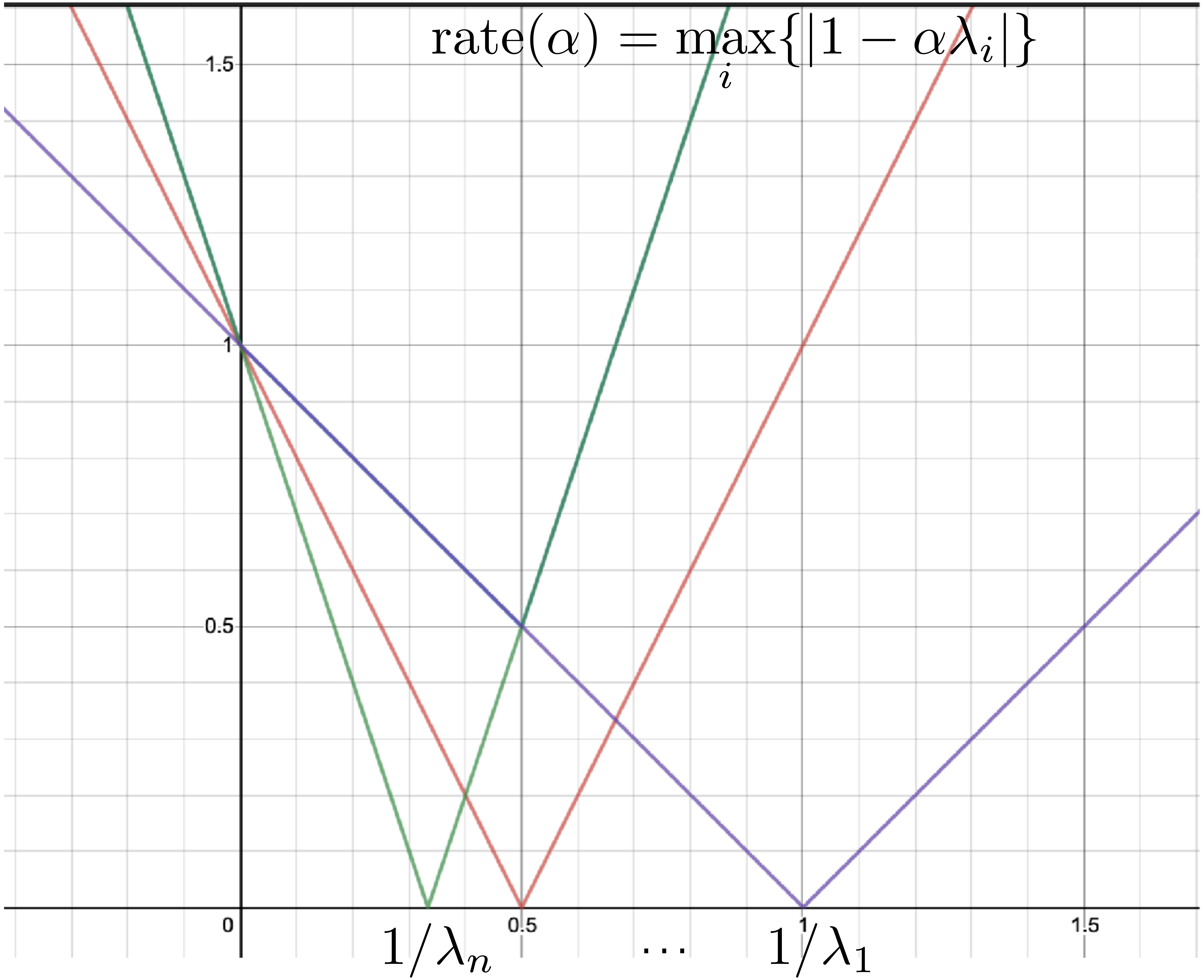
Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i^{k-1}$

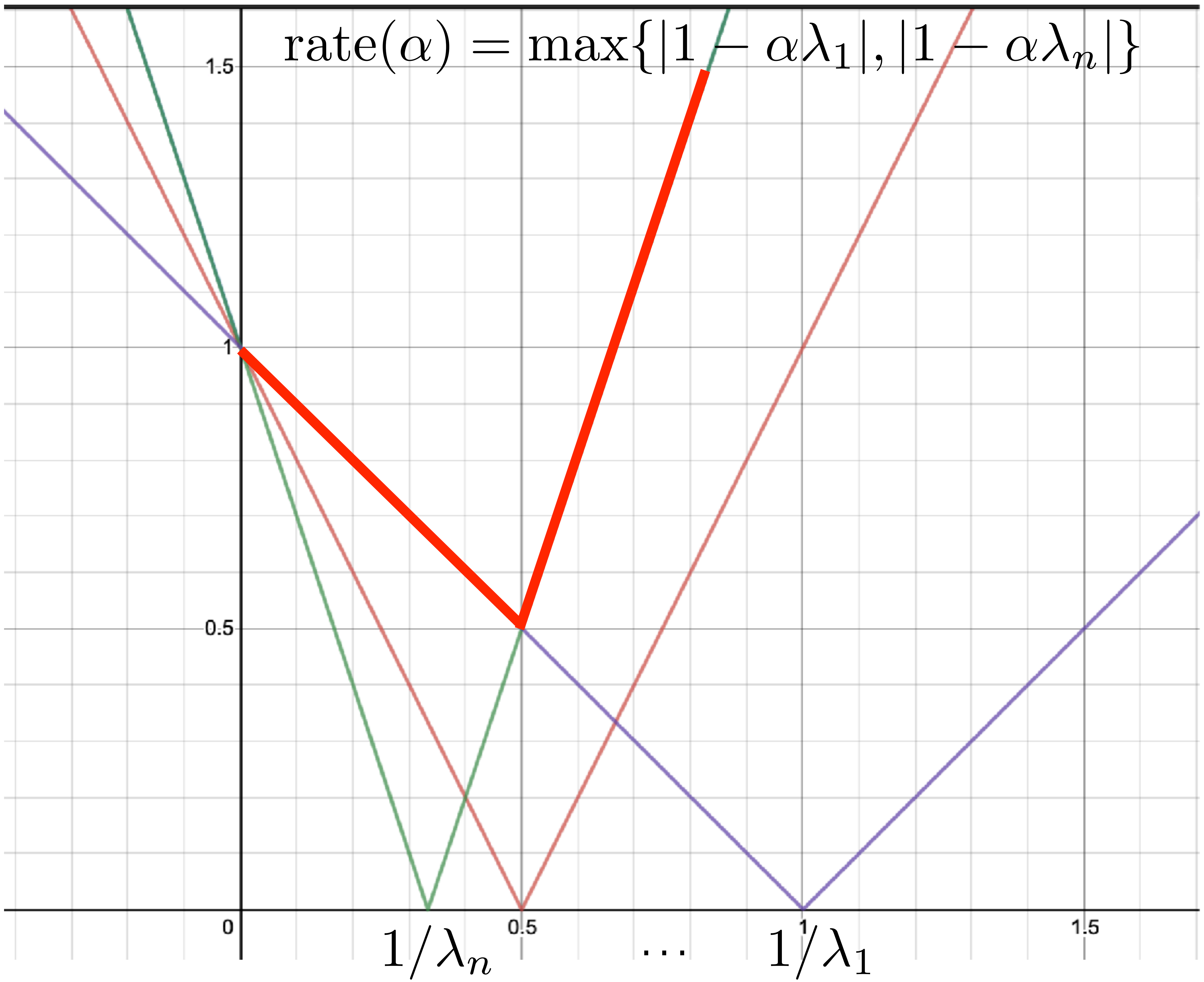
SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

- Converges for: $0 < \alpha\lambda_i < 2$
- with convergence rate: $\mathrm{rate}(\alpha) = \max_i\{|1 - \alpha\lambda_i|\}$

SGD on quadric



SGD on quadric



SGD on quadric

Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i^{k-1}$

SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

- Converges for: $0 < \alpha\lambda_i < 2$
- with convergence rate: $\mathrm{rate}(\alpha) = \max\{|1 - \alpha\lambda_1|, |1 - \alpha\lambda_n|\}$

SGD on quadric

Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i^{k-1}$

SGD after k iterations: $\mathbf{w}_i^k = (1-\alpha\lambda_i)^k \mathbf{w}_i^0$

- Converges for: $0 < \alpha\lambda_i < 2$
- with convergence rate: $\mathrm{rate}(\alpha) = \max\{|1-\alpha\lambda_1|, |1-\alpha\lambda_n|\}$

- Optimal learning rate: $\alpha^* = \arg\min_{\alpha}\big(\mathrm{rate}(\alpha)\big) = \frac{2}{\lambda_1 + \lambda_n}$

SGD on quadric

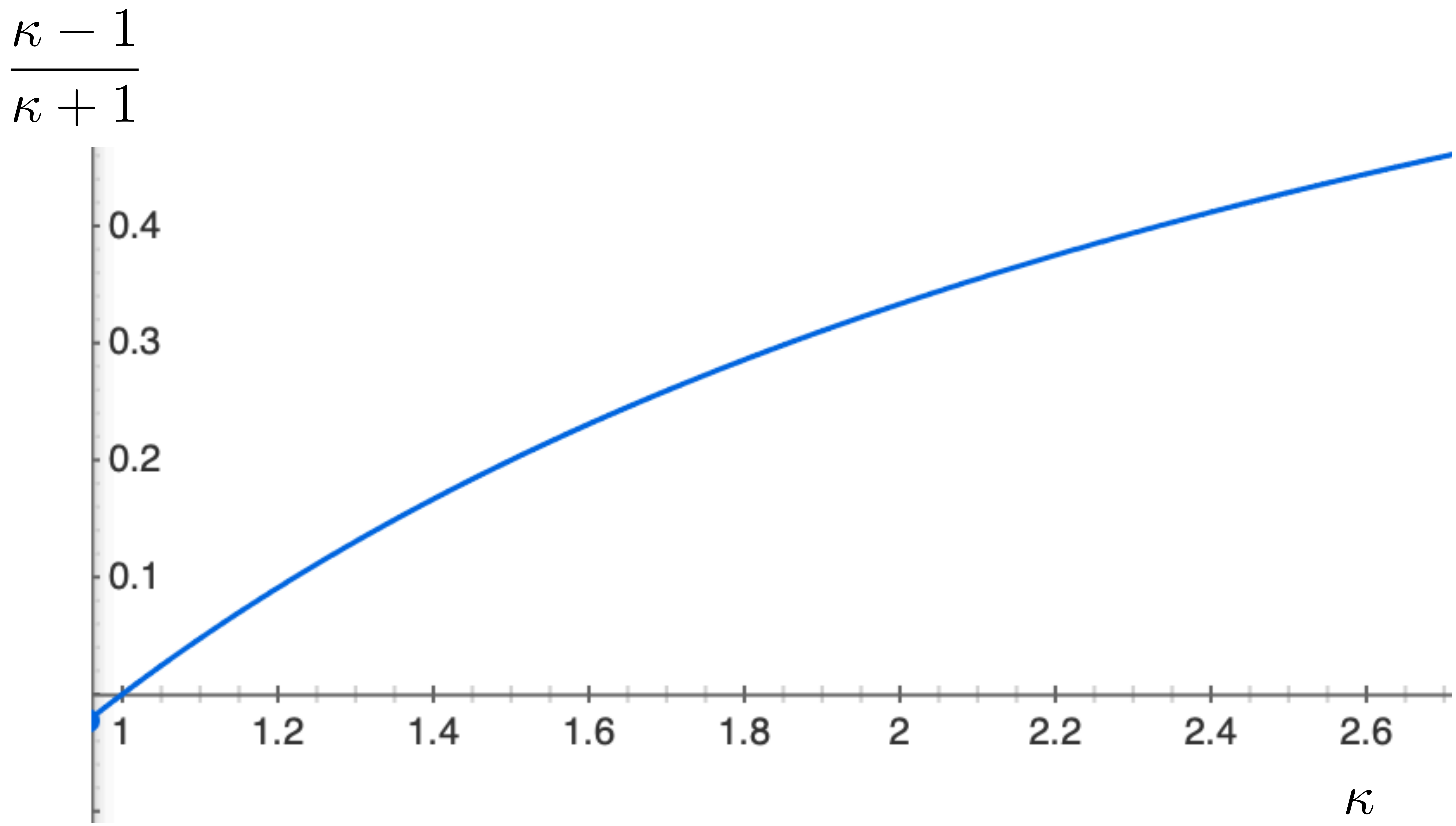
Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \dots, \lambda_n])$

Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i=\mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i^{k-1}$

SGD after k iterations: $\mathbf{w}_i^k = (1-\alpha\lambda_i)^k \mathbf{w}_i^0$

- Converges for: $0 < \alpha\lambda_i < 2$
- with convergence rate: $\mathrm{rate}(\alpha) = \max\{|1-\alpha\lambda_1|, |1-\alpha\lambda_n|\}$
- Optimal learning rate: $\alpha^* = \arg\min_\alpha \big(\mathrm{rate}(\alpha)\big) = \frac{2}{\lambda_1 + \lambda_n}$
- Optimal conv. rate: $\mathrm{rate}(\alpha^*) = \min_\alpha \big(\mathrm{rate}(\alpha)\big) = \frac{\frac{\lambda_n}{\lambda_1} - 1}{\frac{\lambda_n}{\lambda_1} + 1}$
  where $\kappa = \frac{\lambda_n}{\lambda_1}$ is condition number

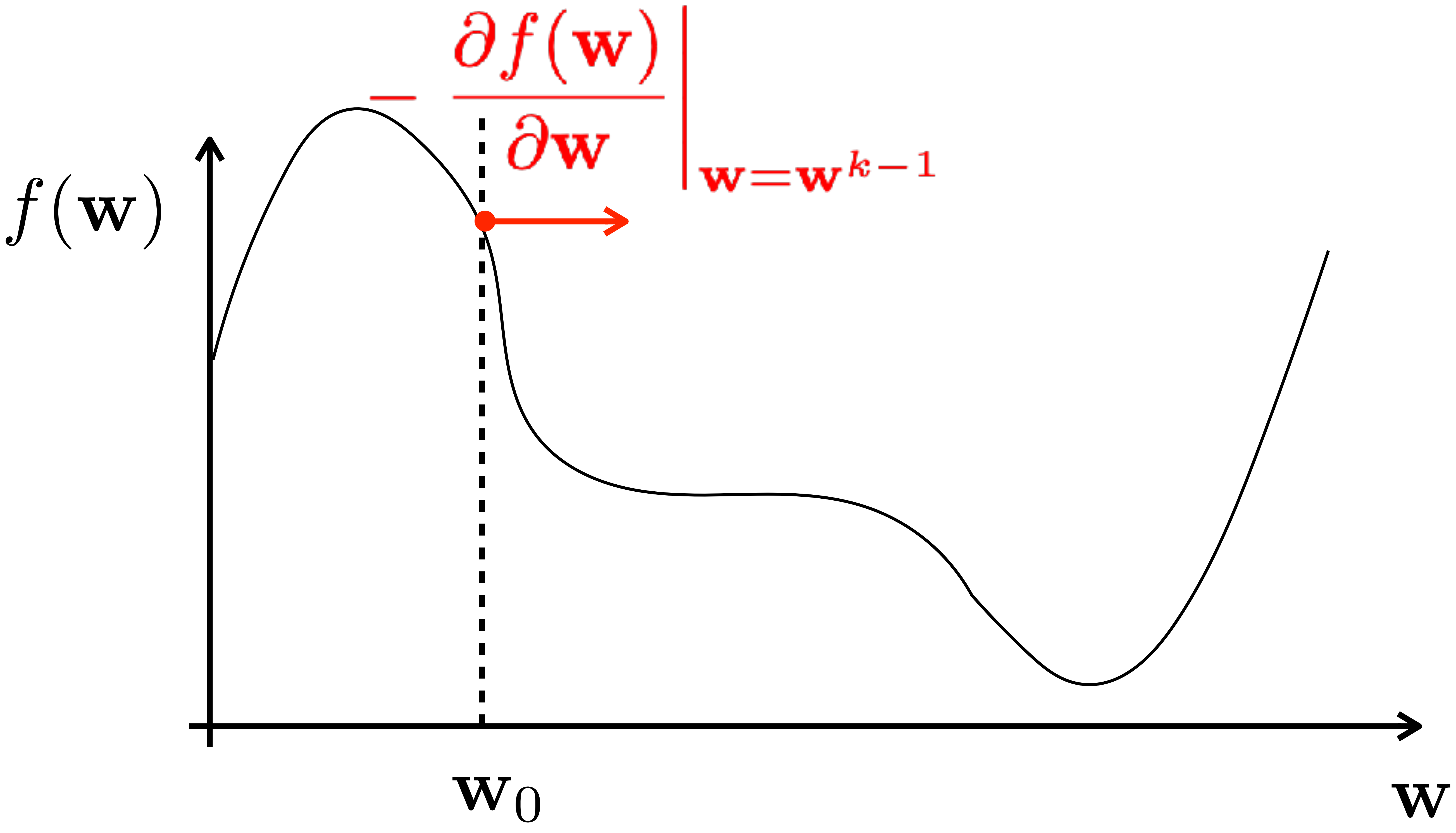
SGD on quadric



SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

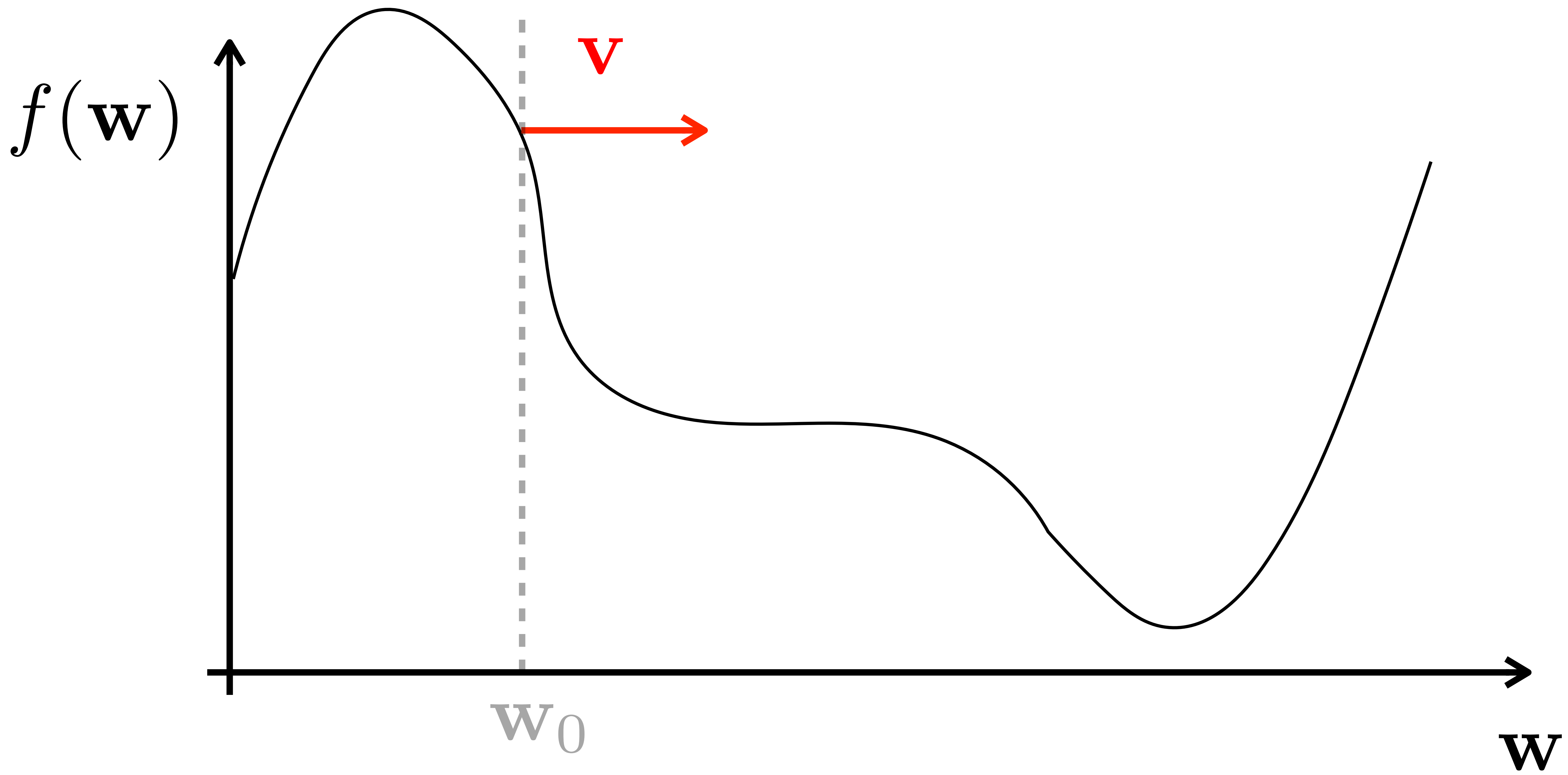
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

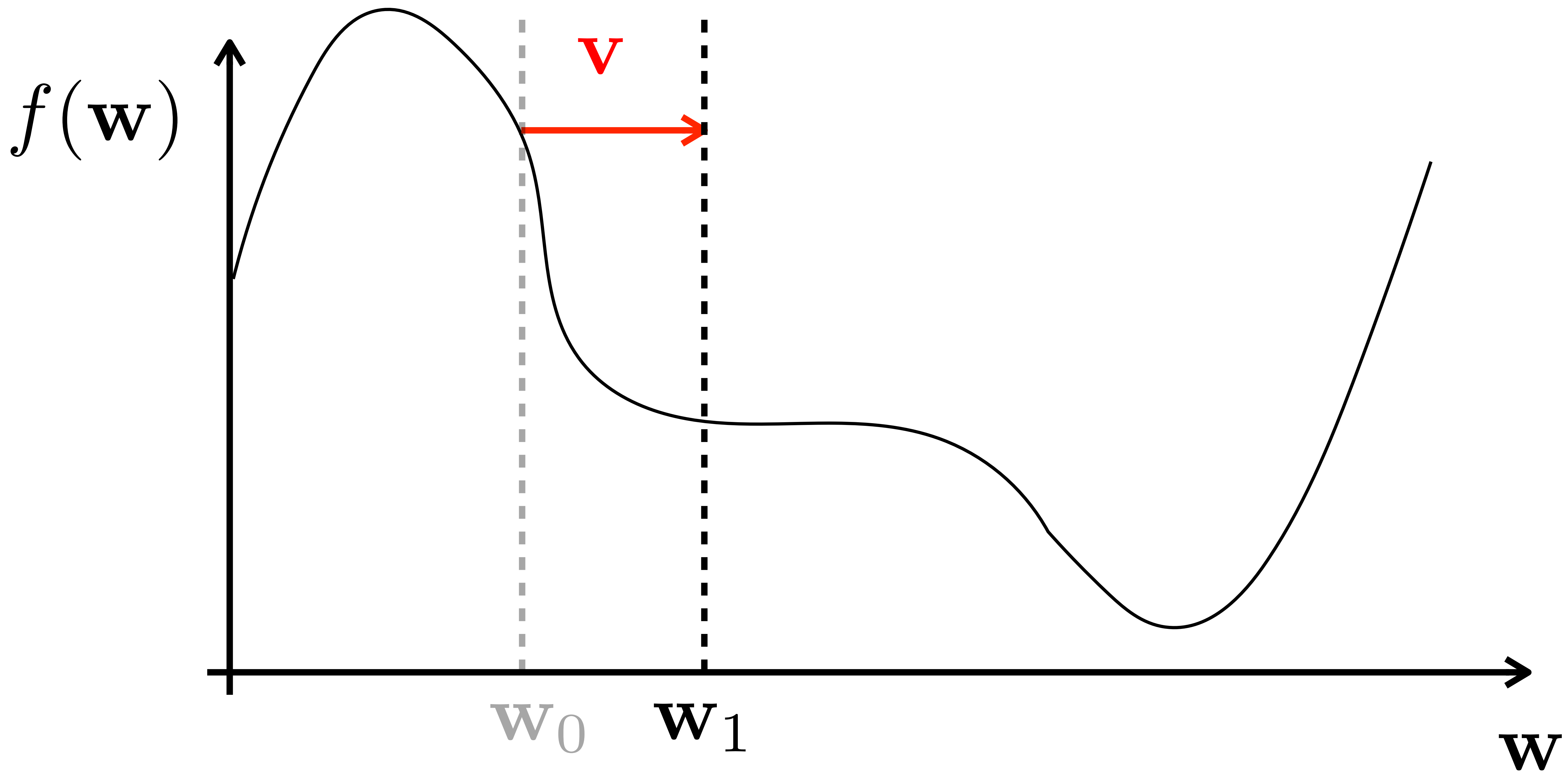
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

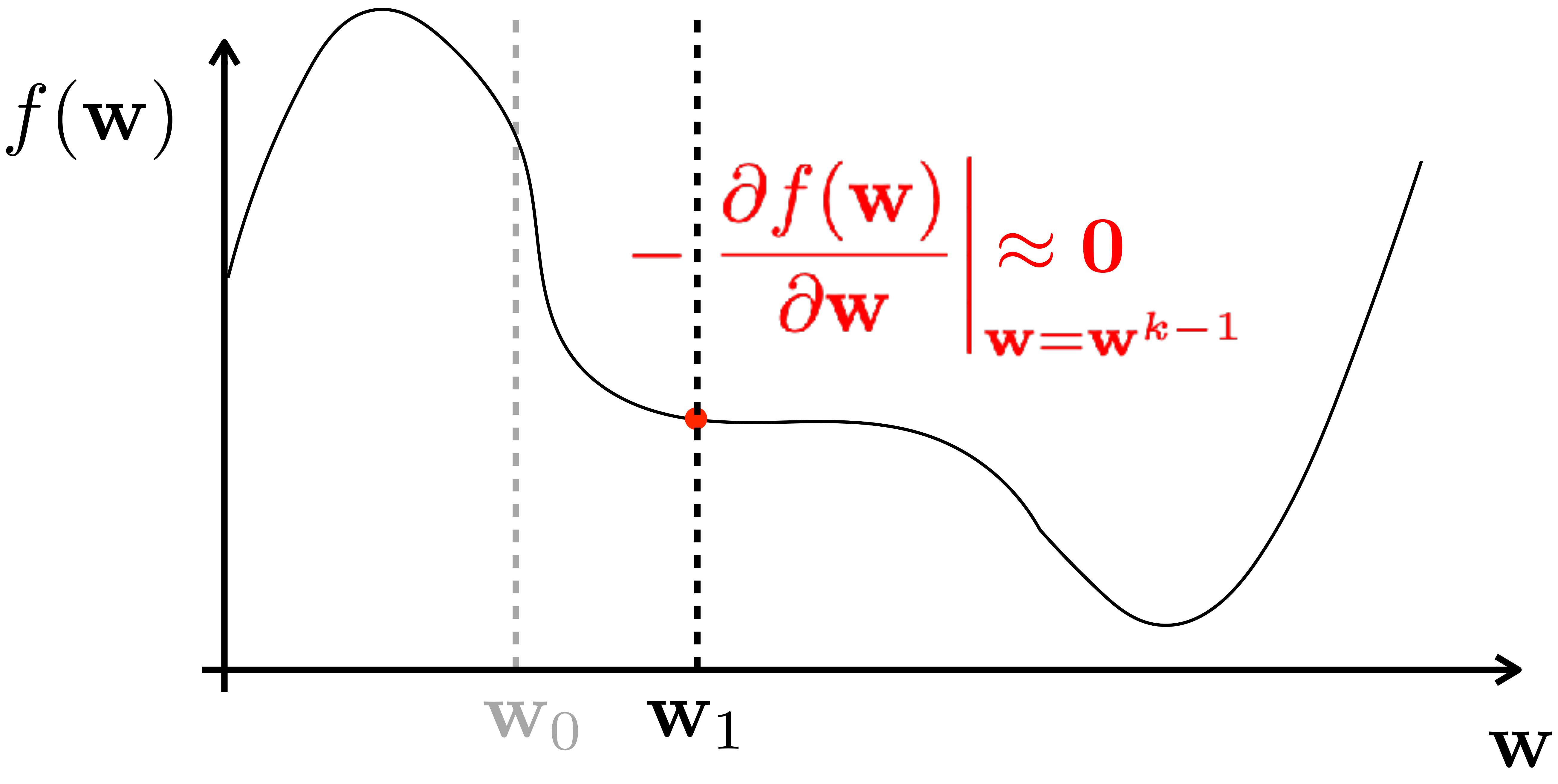
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

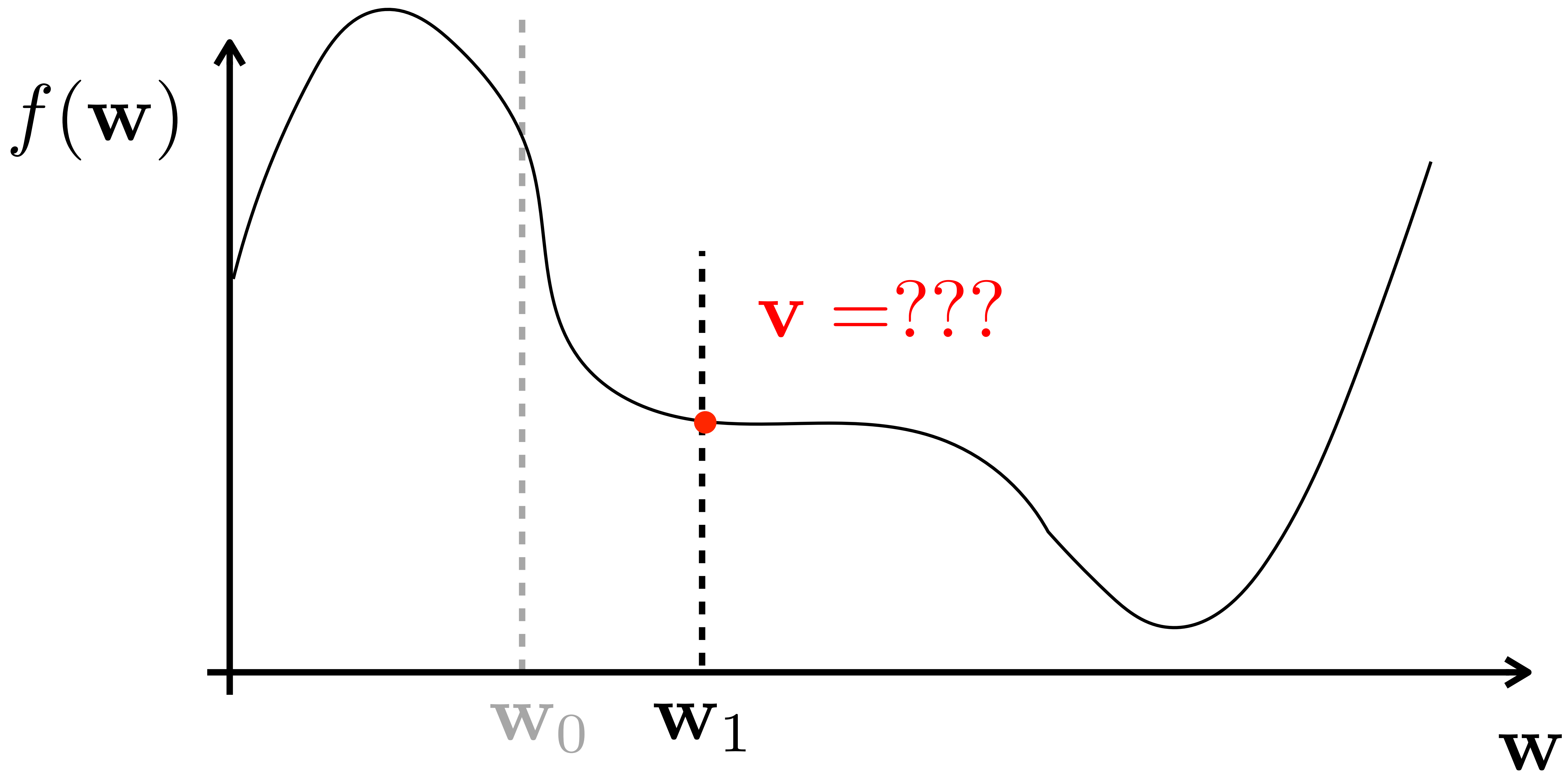


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

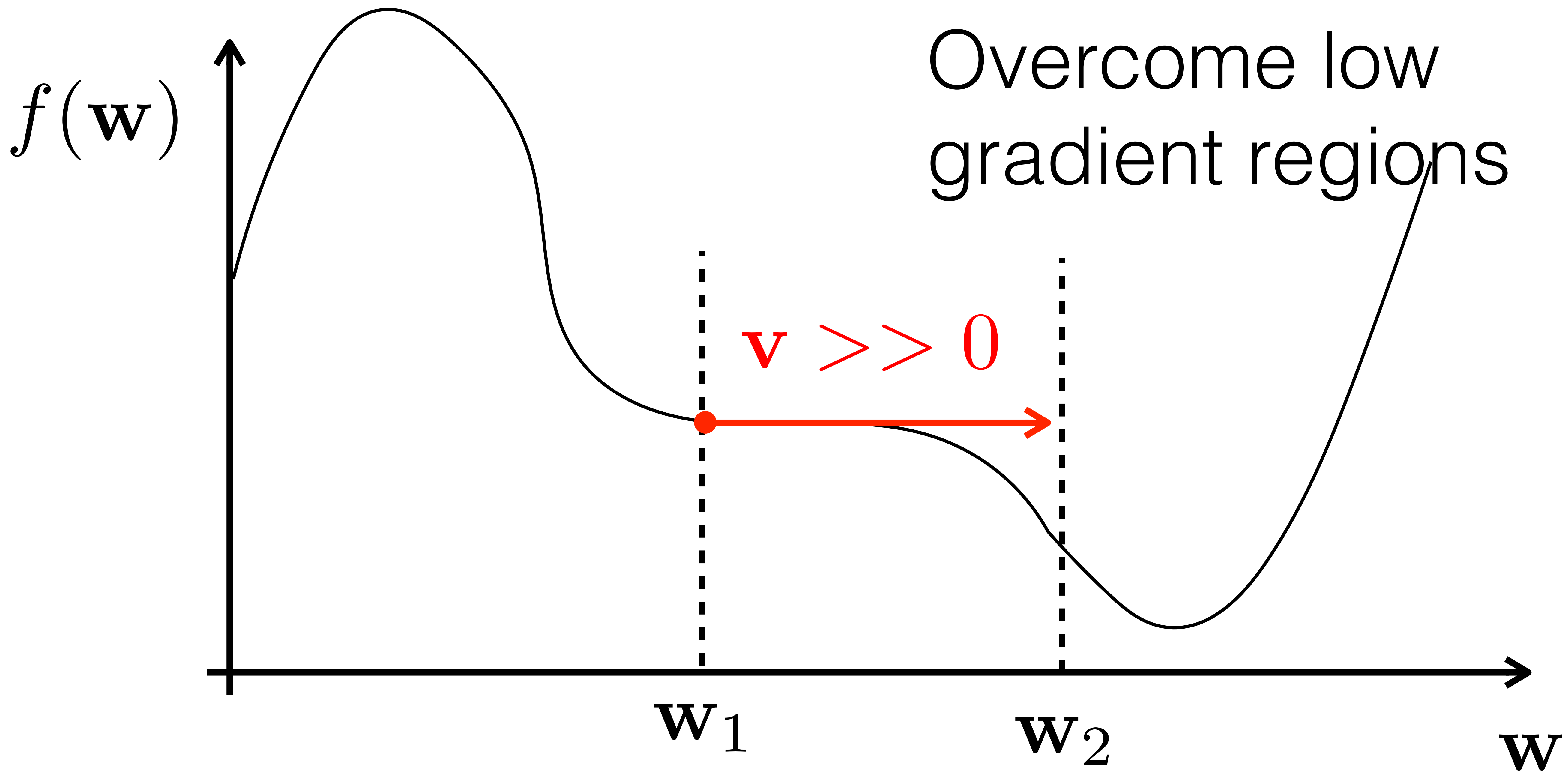


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

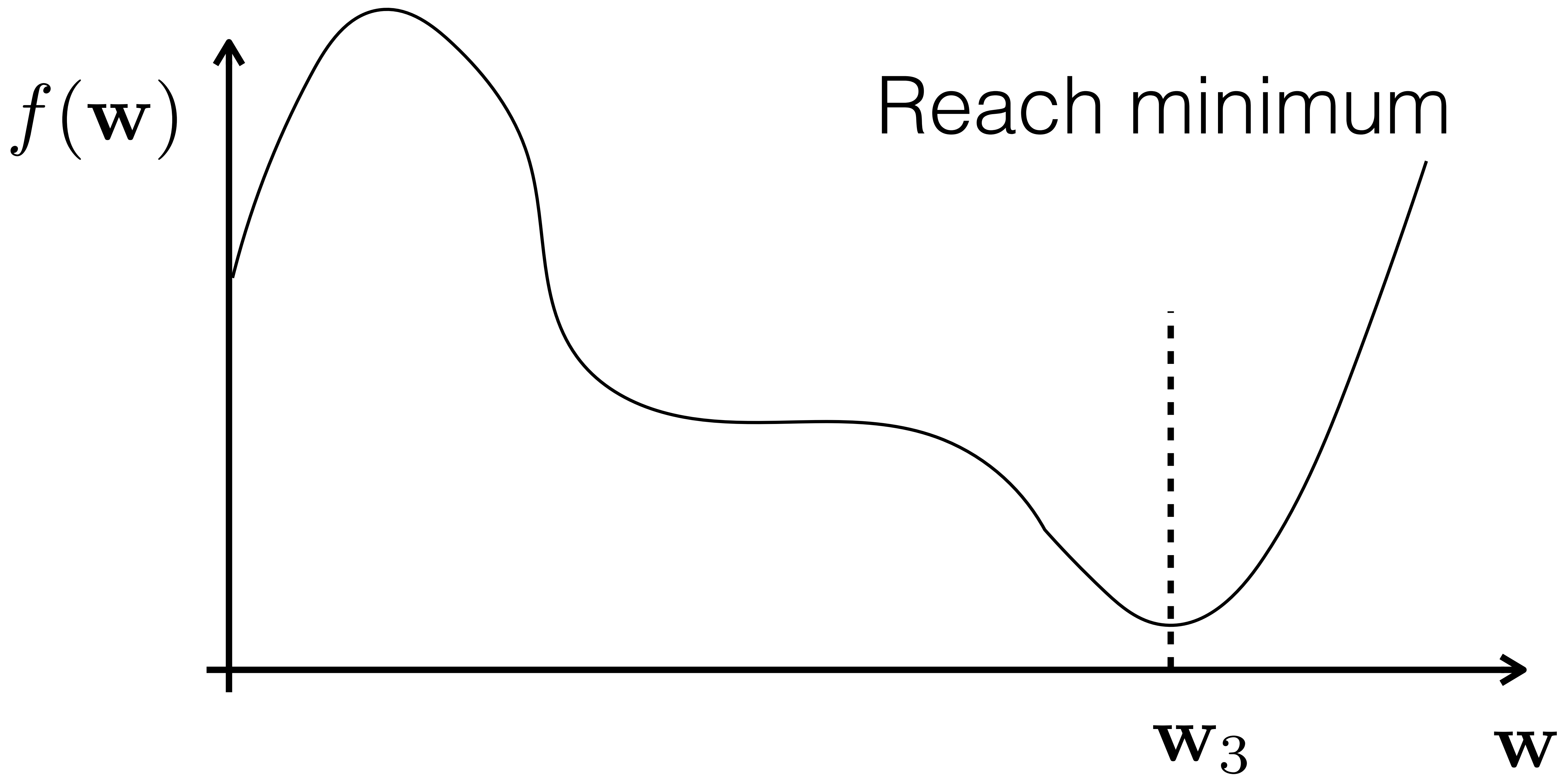


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

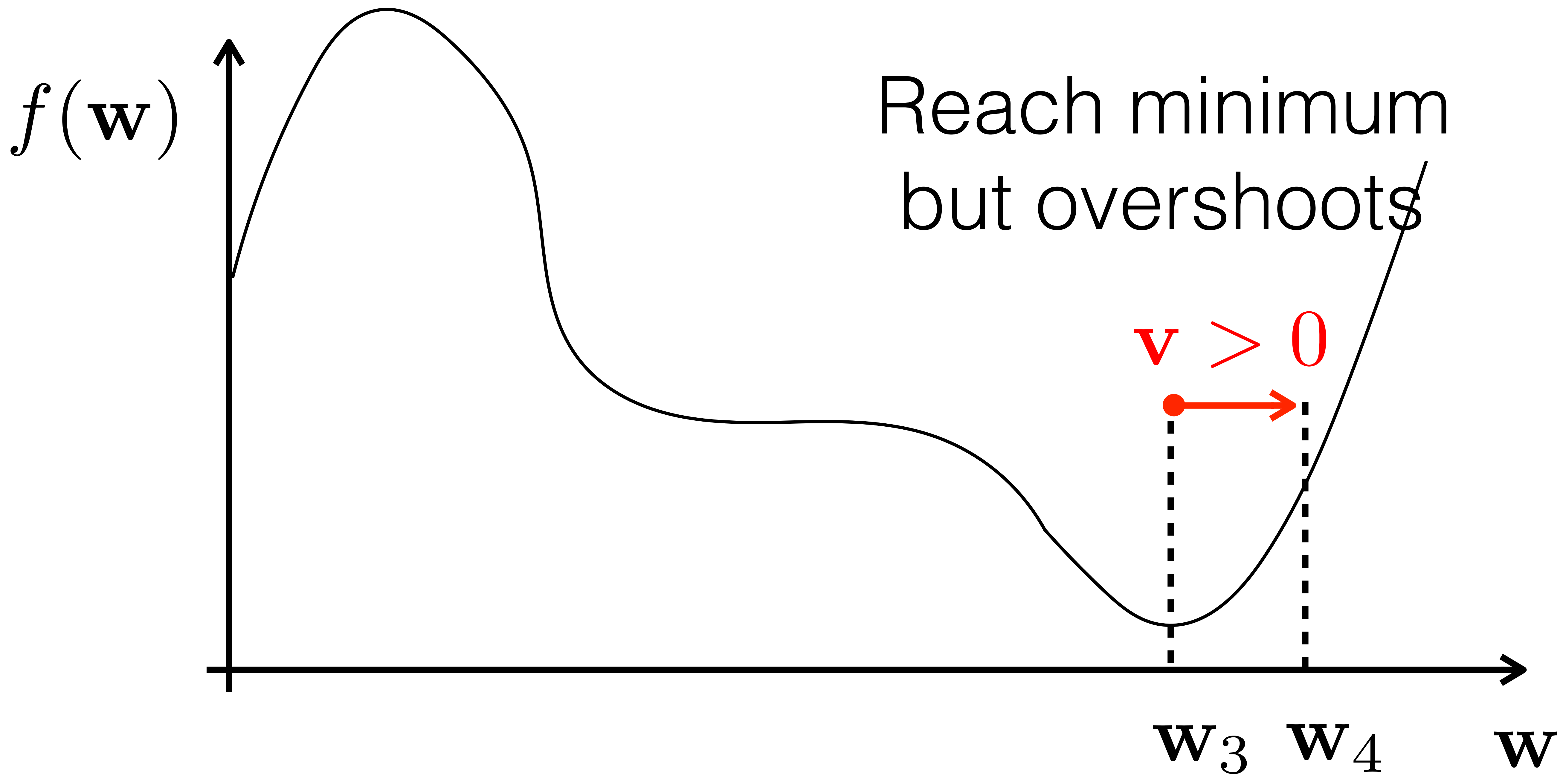


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

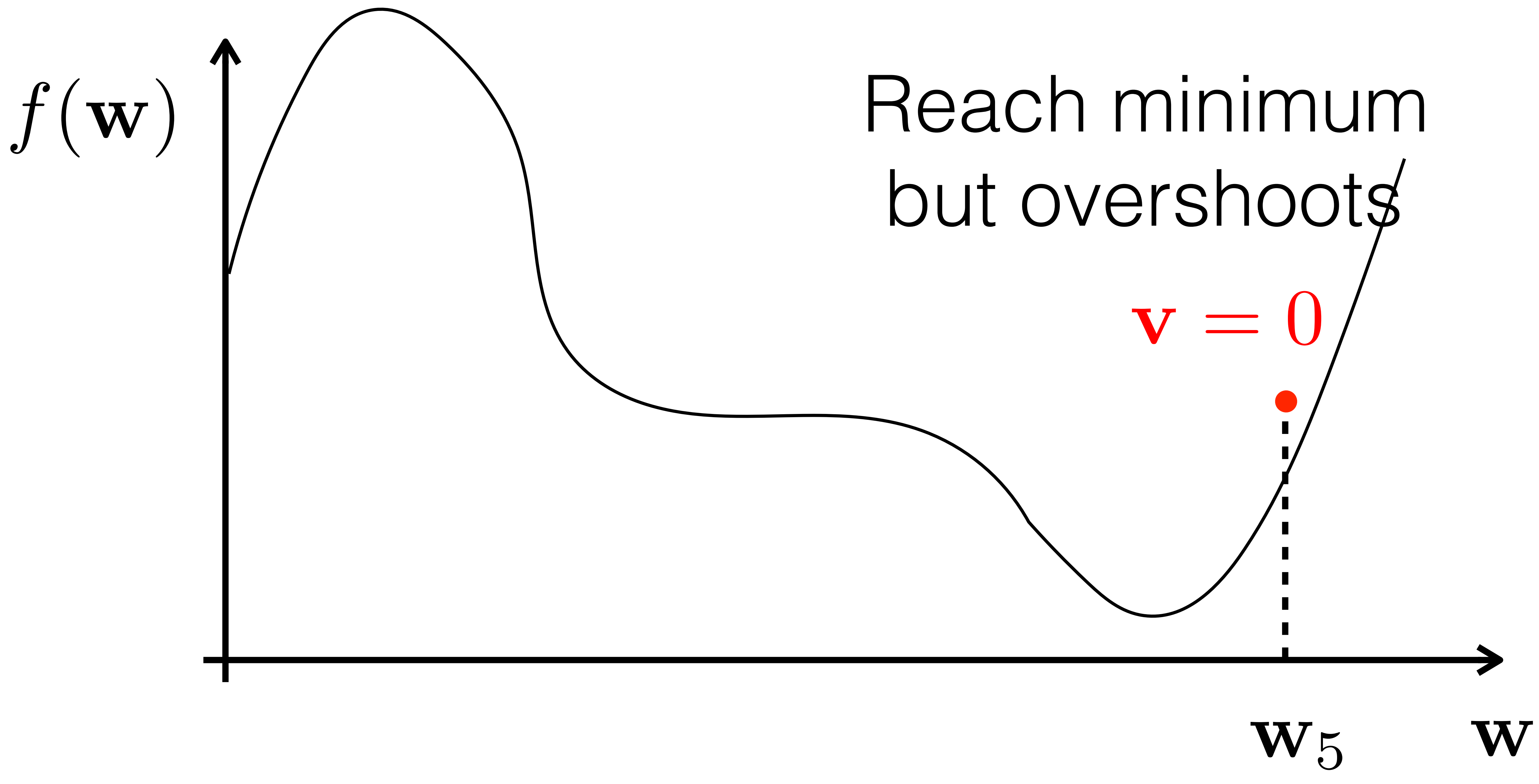


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

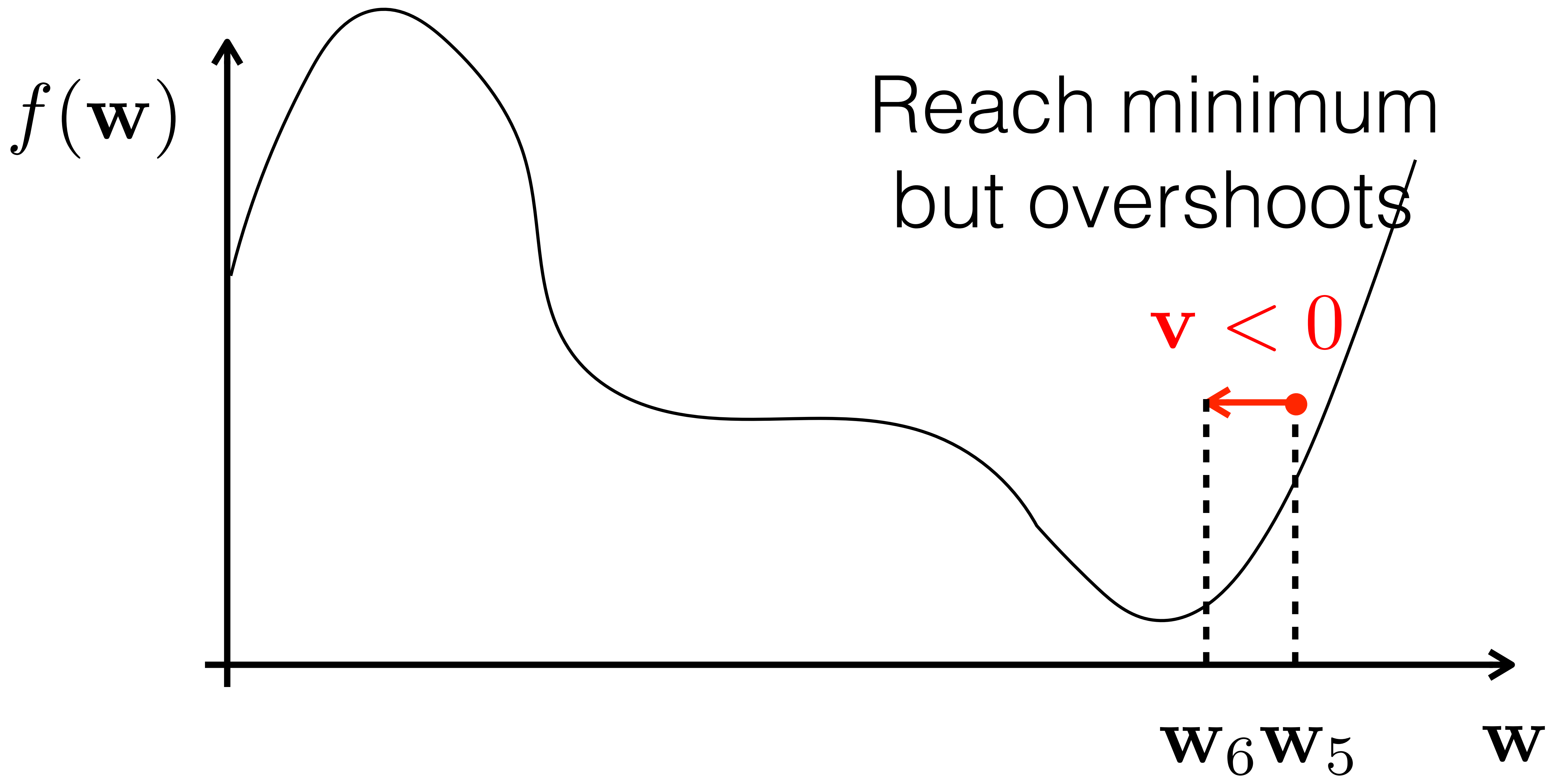


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

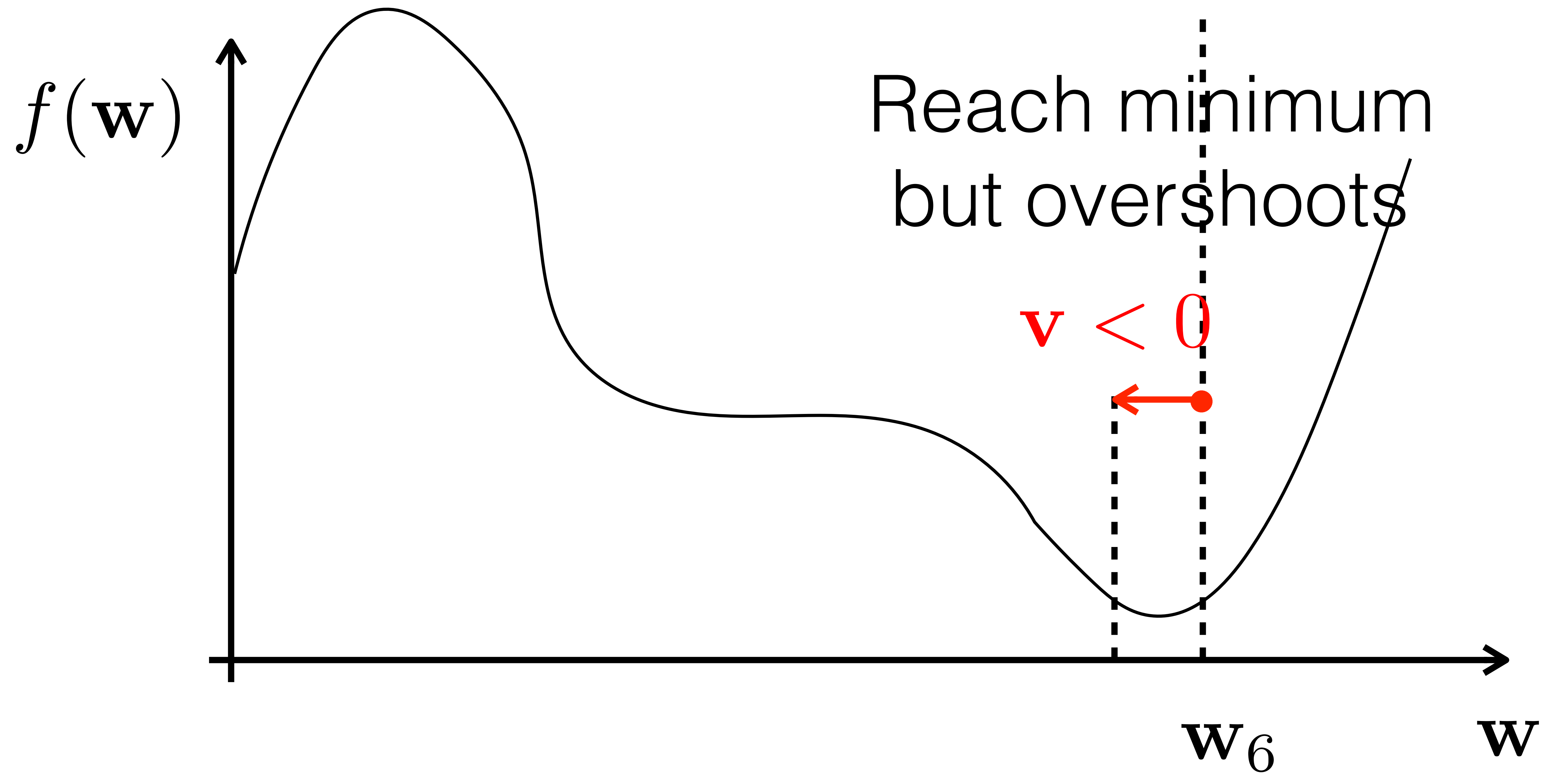


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β

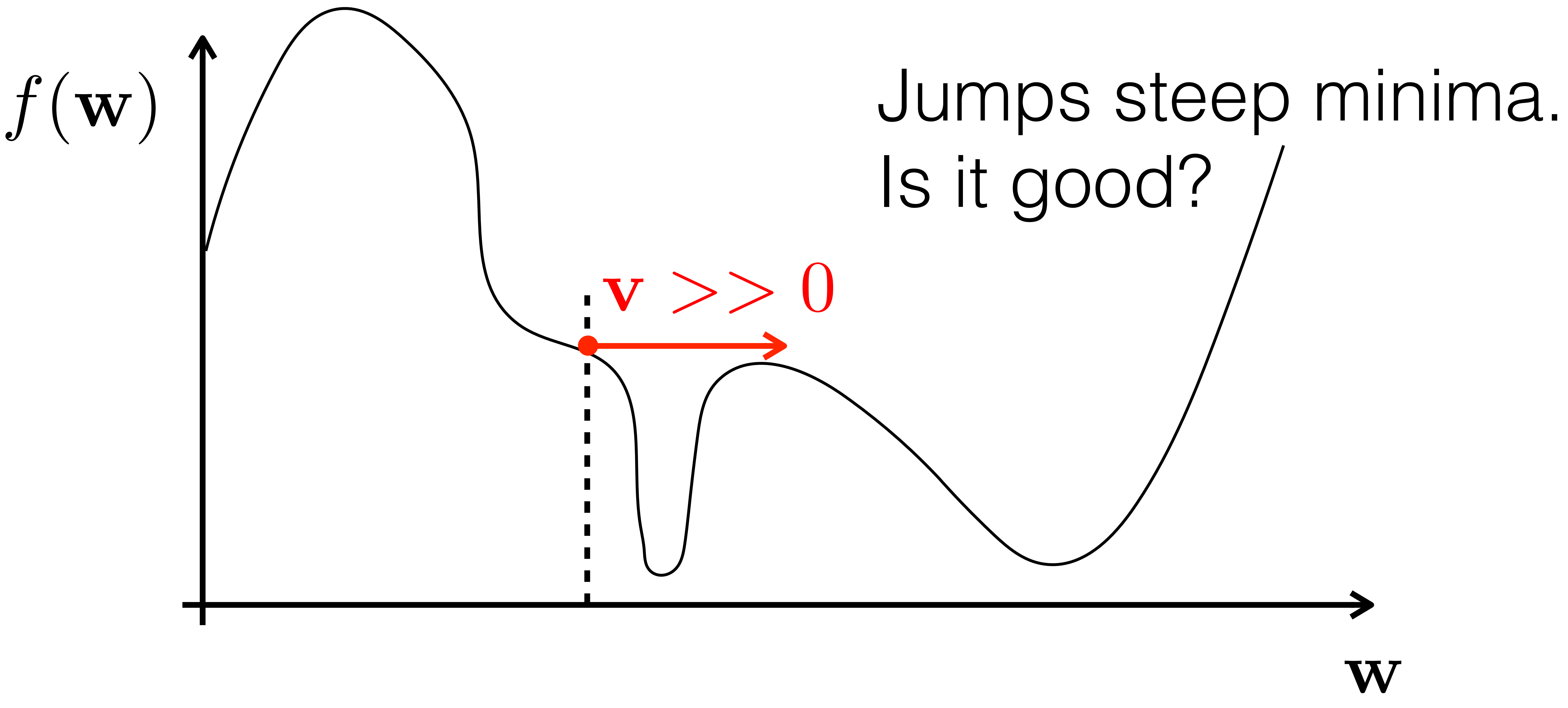


SGD + momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

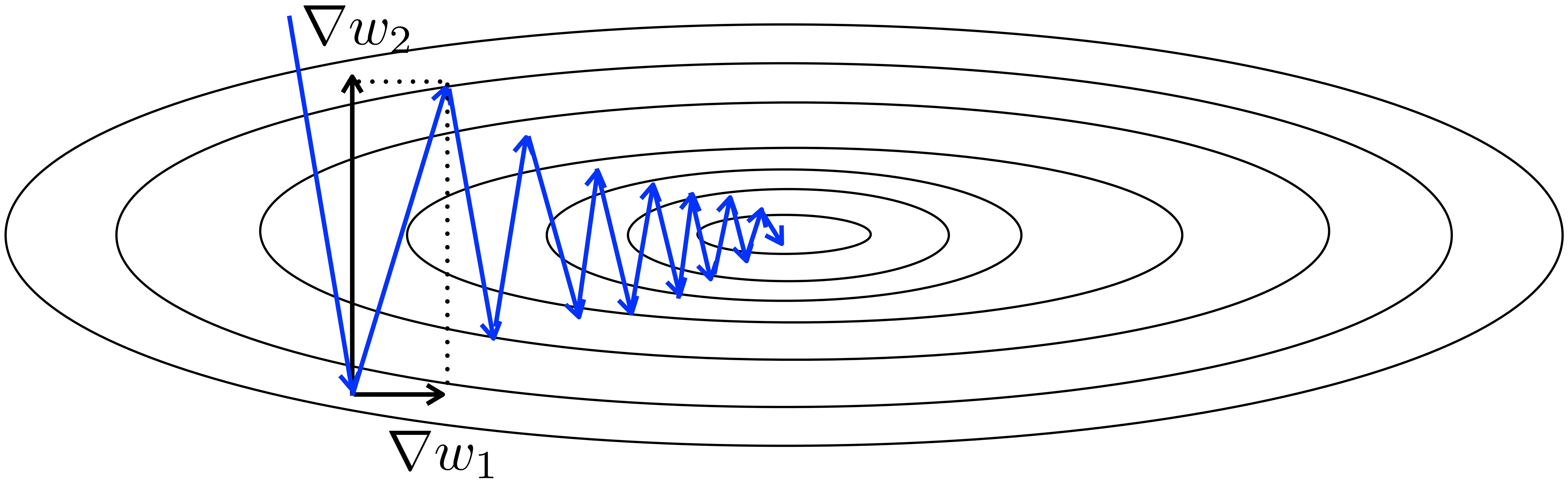
- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β



"SGD" vs "SGD + momentum" in 2D

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Undesired zig-zag behaviour

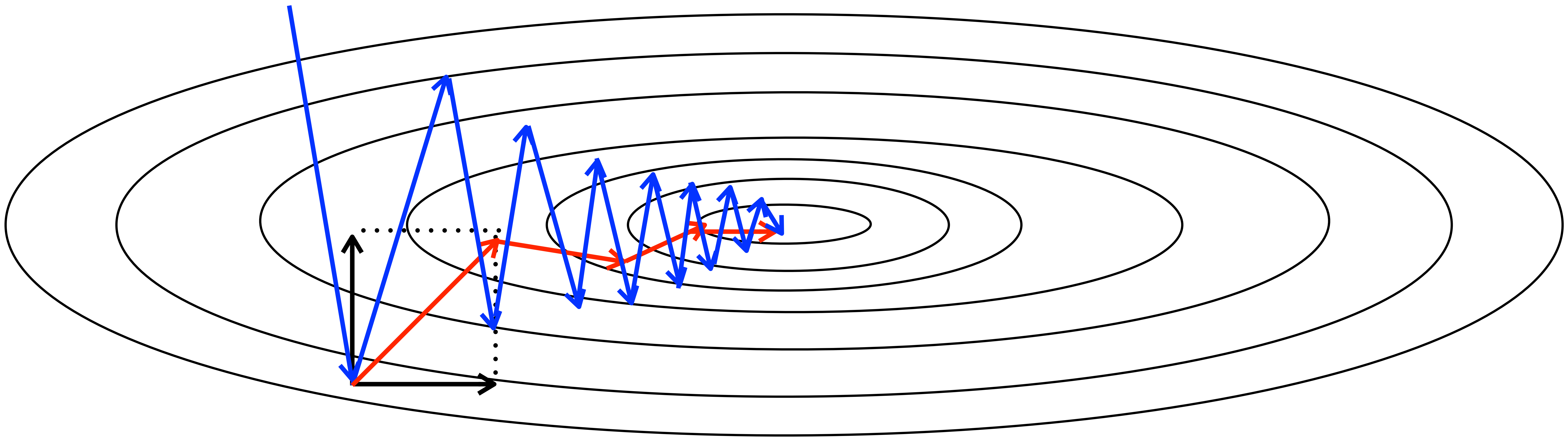


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

"SGD" vs "SGD + momentum" in 2D

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



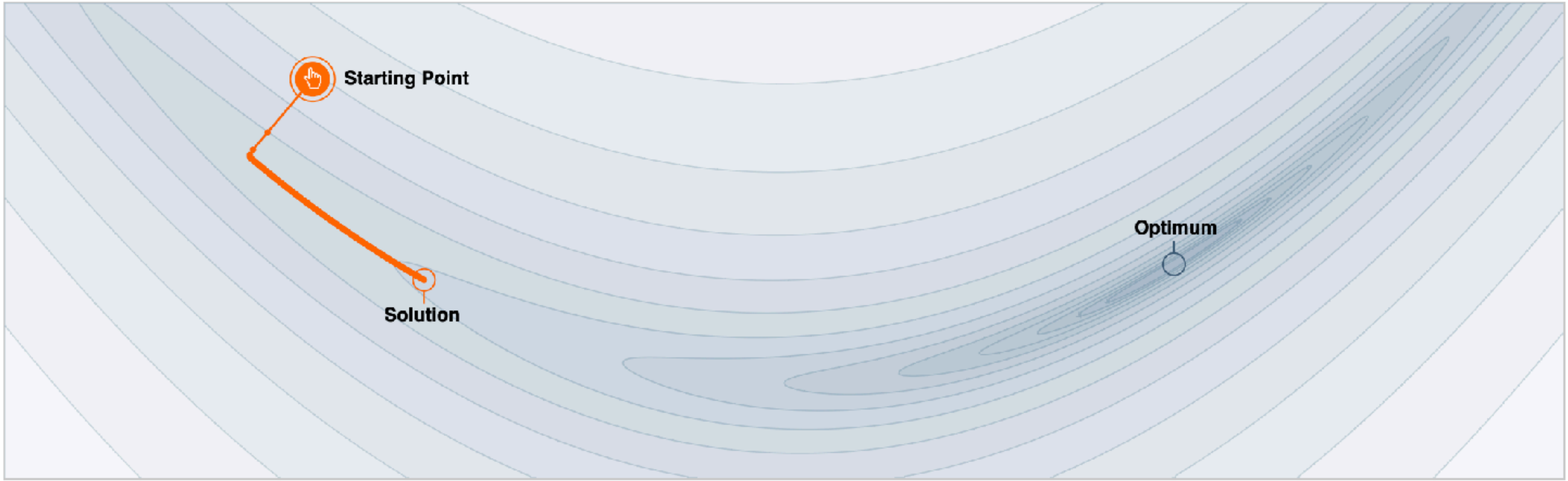
Momentum suppresses this problem partially by averaging element-wise gradients

“SGD” vs “SGD + momentum” in 2D

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

$\alpha =$ 1e-3 $\beta =$ 0



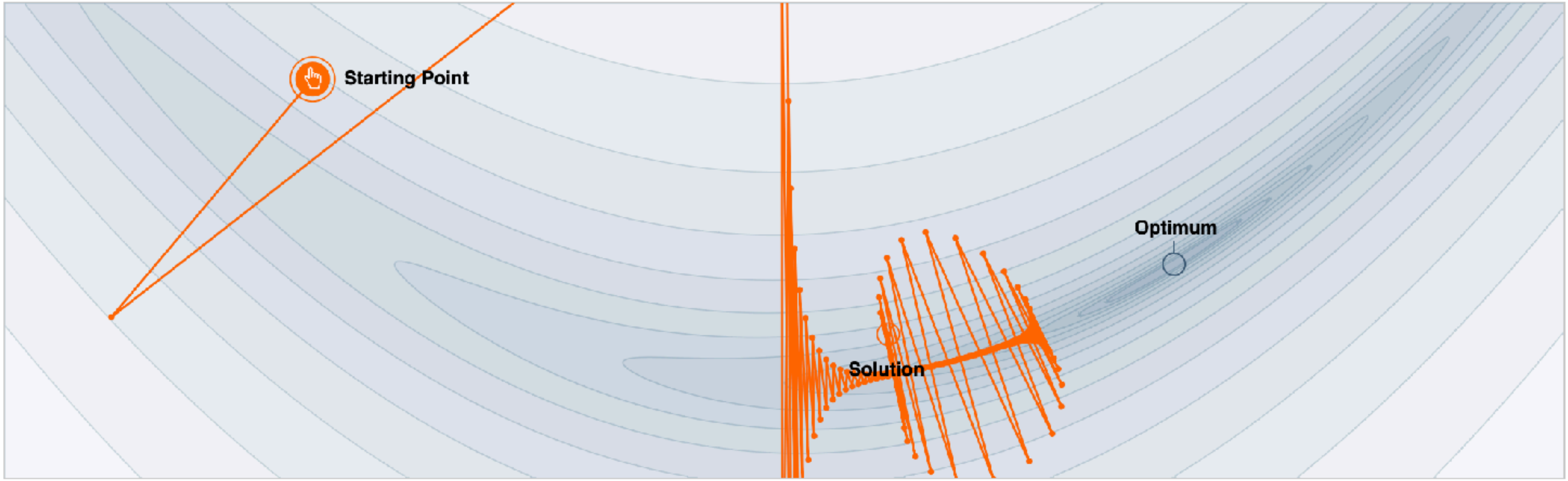
https://distill.pub/2017/momentum/

“SGD” vs “SGD + momentum” in 2D

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

$\alpha =$ 5e-3 $\beta =$ 0



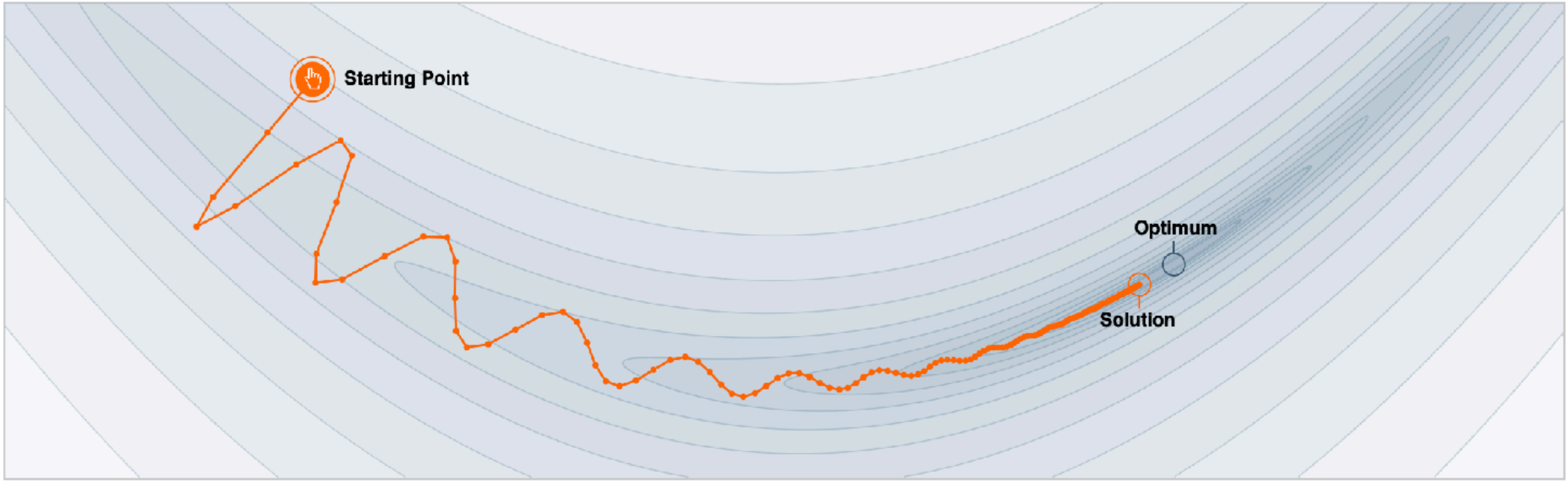
https://distill.pub/2017/momentum/

"SGD" vs "SGD + momentum" in 2D

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

$\alpha =$ 1e-3 $\beta =$ 0.9



https://distill.pub/2017/momentum/

"SGD + momentum" on quadric

Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \dots, \lambda_n])$

Gradient: $\frac{\partial f(\mathbf{w}_i)}{\partial \mathbf{w}_i} = \lambda_i \mathbf{w}_i$

SGD+momentum after k iterations:

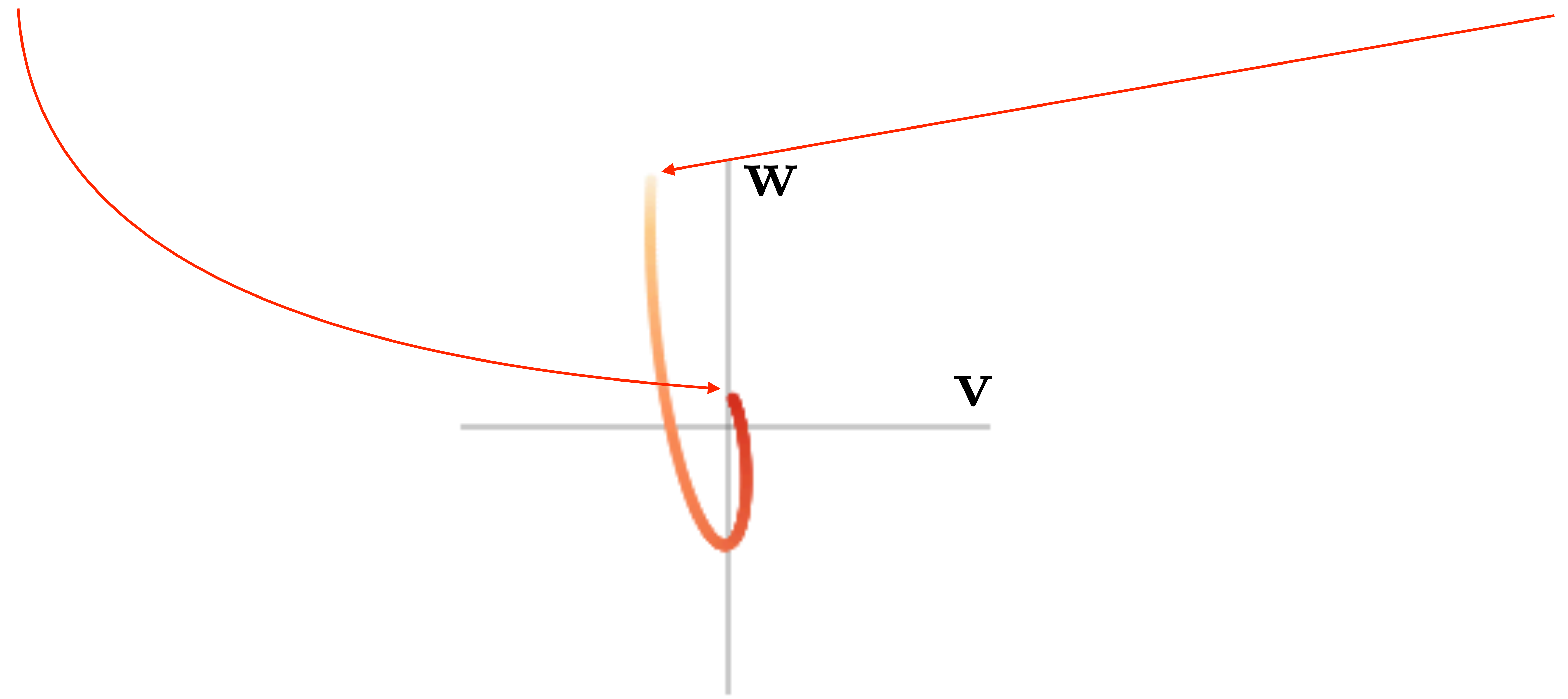
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

$$\begin{bmatrix}\mathbf{v}_i^k \\ \mathbf{w}_i^k\end{bmatrix} = \begin{bmatrix}\beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i\end{bmatrix}\begin{bmatrix}\mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1}\end{bmatrix} = \begin{bmatrix}\beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i\end{bmatrix}^k \begin{bmatrix}\mathbf{v}_i^0 \\ \mathbf{w}_i^0\end{bmatrix}$$

[Flammarion, Bach COLT 2017]
https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

"SGD + momentum" on quadric

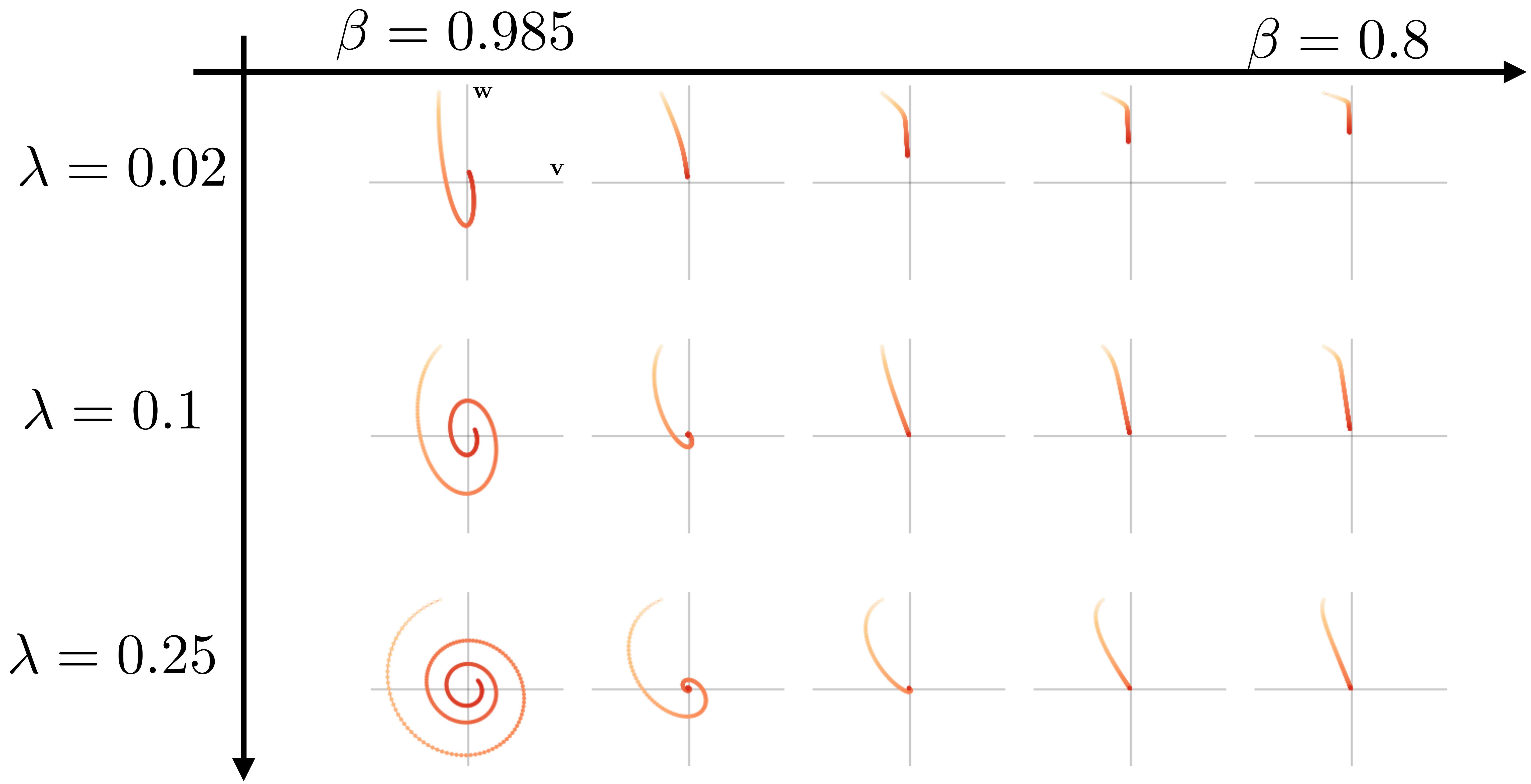
$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k \begin{bmatrix} \mathbf{v}_i^0 \\ \mathbf{w}_i^0 \end{bmatrix}$$



[Flammarion, Bach COLT 2017]
https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

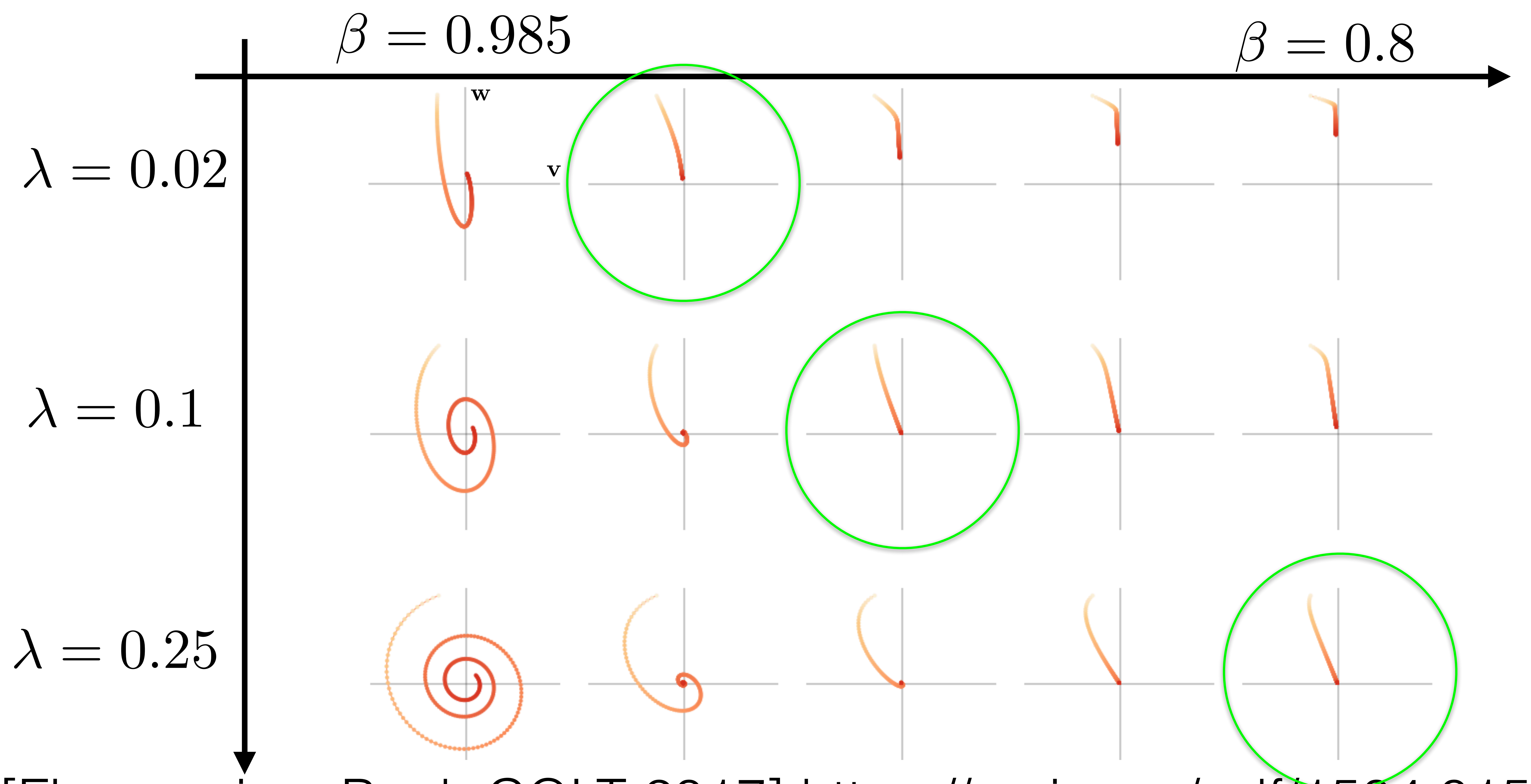
$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k \begin{bmatrix} \mathbf{v}_i^0 \\ \mathbf{w}_i^0 \end{bmatrix}$$



[Flammarion, Bach COLT 2017] https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

"SGD + momentum" on quadric

$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k \begin{bmatrix} \mathbf{v}_i^0 \\ \mathbf{w}_i^0 \end{bmatrix}$$



[Flammarion, Bach COLT 2017] https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

Gradient: $\frac{\partial f(\mathbf{w}_i)}{\partial \mathbf{w}_i} = \lambda_i \mathbf{w}_i$

SGD+momentum after k iterations:

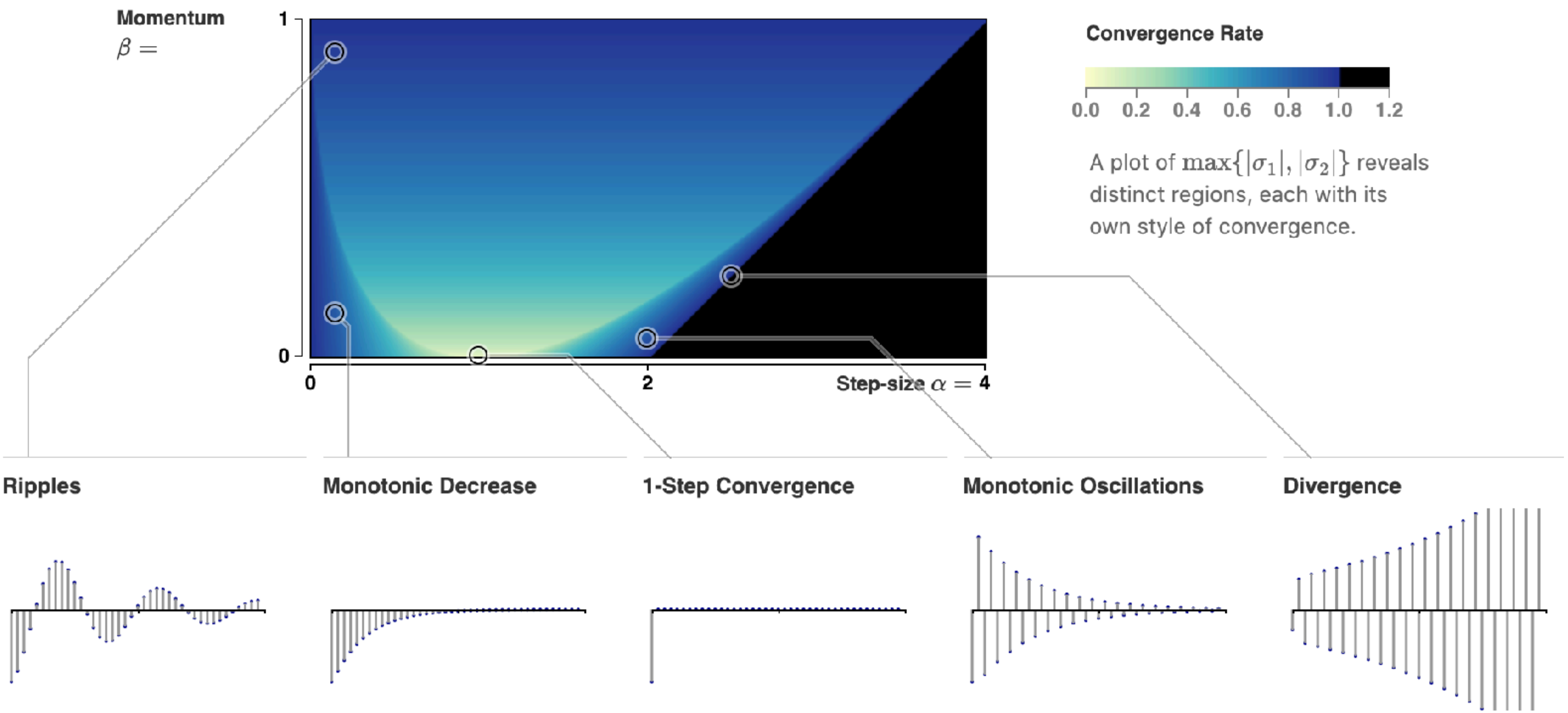
$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k \begin{bmatrix} \mathbf{v}_i^0 \\ \mathbf{w}_i^0 \end{bmatrix}$$

- Converg. rate: $\mathrm{rate}_i(\alpha, \beta) = \max\{|\sigma_1(\alpha, \beta, \lambda_i)|, |\sigma_2(\alpha, \beta, \lambda_i)|\}$

[Flammarion, Bach COLT 2017] https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

$$\text{rate}_i(\alpha, \beta) = \max\{|\sigma_1(\alpha, \beta, \lambda_i)|, |\sigma_2(\alpha, \beta, \lambda_i)|\}$$



[Flammarion, Bach COLT 2017] https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

Gradient: $\frac{\partial f(\mathbf{w}_i)}{\partial \mathbf{w}_i} = \lambda_i \mathbf{w}_i$

SGD+momentum after k iterations:

$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k \begin{bmatrix} \mathbf{v}_i^0 \\ \mathbf{w}_i^0 \end{bmatrix}$$

- Converg. rate: $\mathrm{rate}_i(\alpha, \beta) = \max\{|\sigma_1(\alpha, \beta, \lambda_i)|, |\sigma_2(\alpha, \beta, \lambda_i)|\}$
- Optimal parameters:

$$\alpha^* = \Big(\frac{2}{\sqrt{\lambda_1} + \sqrt{\lambda_n}}\Big)^2 \qquad \beta^* = \Big(\frac{\sqrt{\lambda_n} - \sqrt{\lambda_1}}{\sqrt{\lambda_n} + \sqrt{\lambda_1}}\Big)^2$$

[Flammarion, Bach COLT 2017] https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

Criterion: $$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \dots, \lambda_n])$

Gradient: $$\frac{\partial f(\mathbf{w}_i)}{\partial \mathbf{w}_i} = \lambda_i \mathbf{w}_i$$

SGD+momentum after k iterations:

$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k \begin{bmatrix} \mathbf{v}_i^0 \\ \mathbf{w}_i^0 \end{bmatrix}$$

- Converg. rate: $\mathrm{rate}_i(\alpha, \beta) = \max\{|\sigma_1(\alpha, \beta, \lambda_i)|, |\sigma_2(\alpha, \beta, \lambda_i)|\}$
- Optimal parameters:

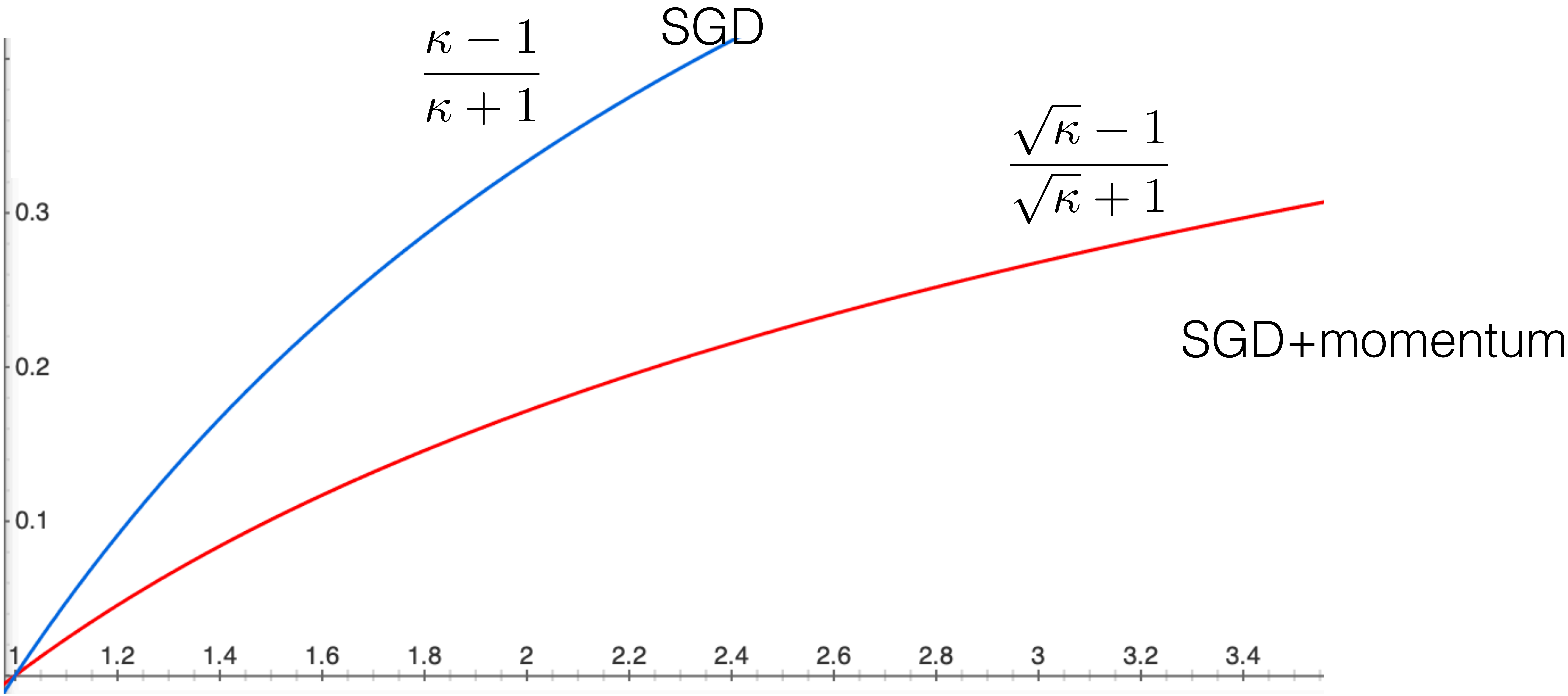
$$\alpha^* = \left(\frac{2}{\sqrt{\lambda_1} + \sqrt{\lambda_n}}\right)^2 \qquad \beta^* = \left(\frac{\sqrt{\lambda_n} - \sqrt{\lambda_1}}{\sqrt{\lambda_n} + \sqrt{\lambda_1}}\right)^2$$

- Optimal convergence rate: $$\mathrm{rate}(\alpha^*, \beta^*) = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$$

[Flammarion, Bach COLT 2017] https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

Convergence rate




```
torch.optim.SGD(params, lr=0.001, momentum=0.9)
```

PyTorch

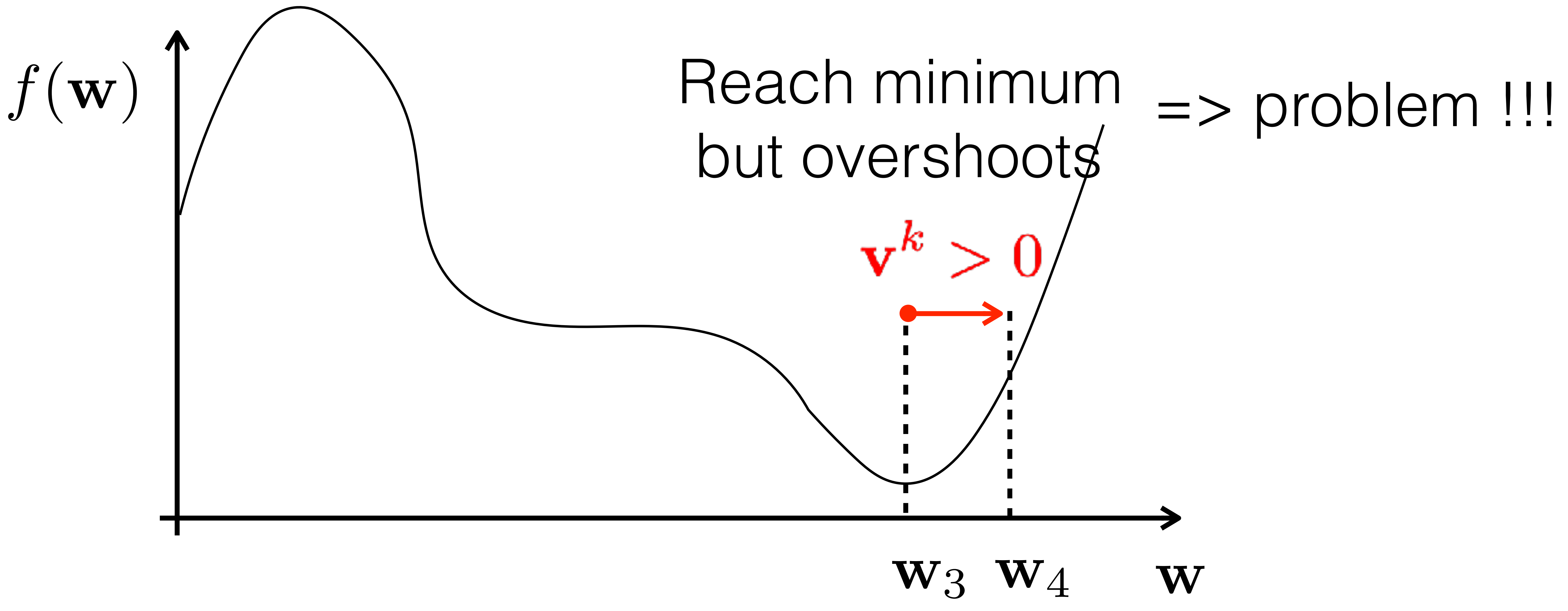
```
# initialise
import torch.nn as nn
import torch.optim as optim

# initialize optimizer
optimizer = optim.SGD(conv_net.parameters(), lr=1e-2)

# define ConvNet model
conv_net = …

# define criterion function
loss = loss_fn(conv_net(images), labels)

# compute gradient
loss.backward()

# update weights of the model
optimizer.step()
```

SGD + momentum - drawback

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff $\rho = 0.95$

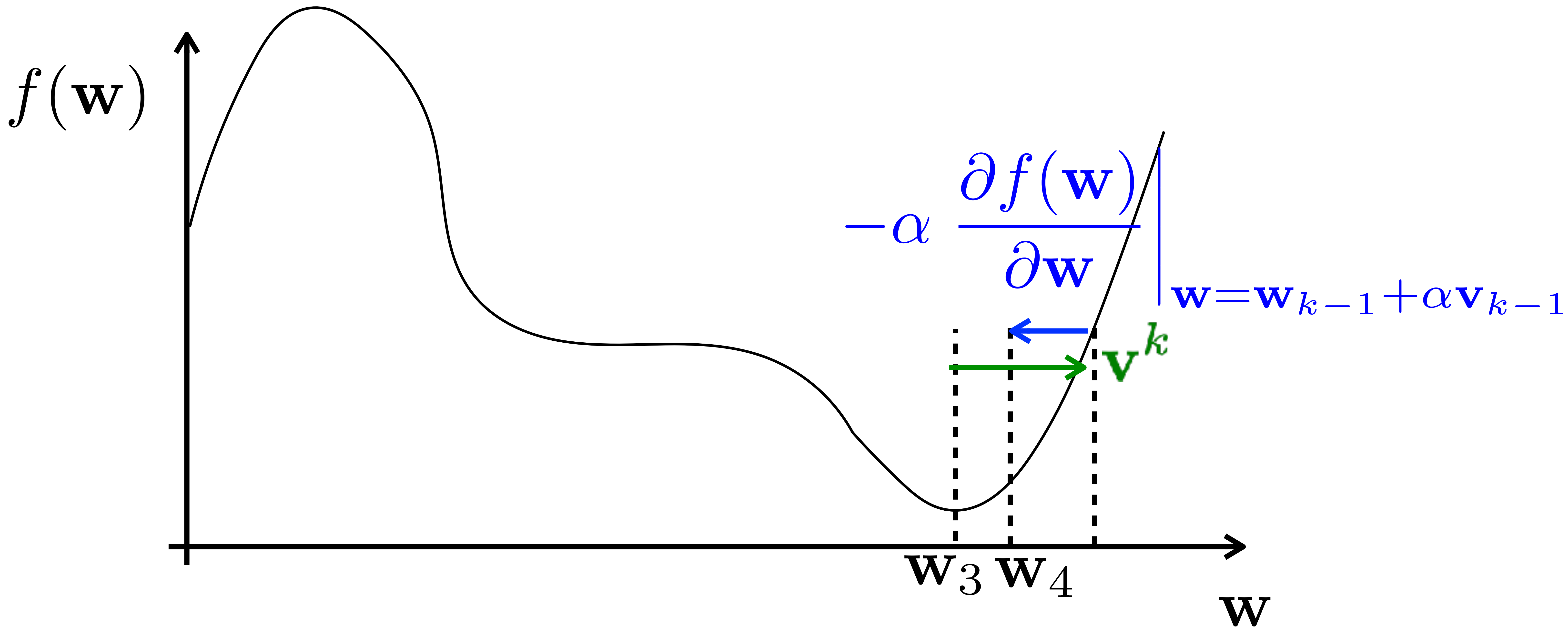


SGD with Nesterov momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}+\alpha\mathbf{v}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Look one step ahead and reduce velocity by future gradient
- Partially prevents overshooting

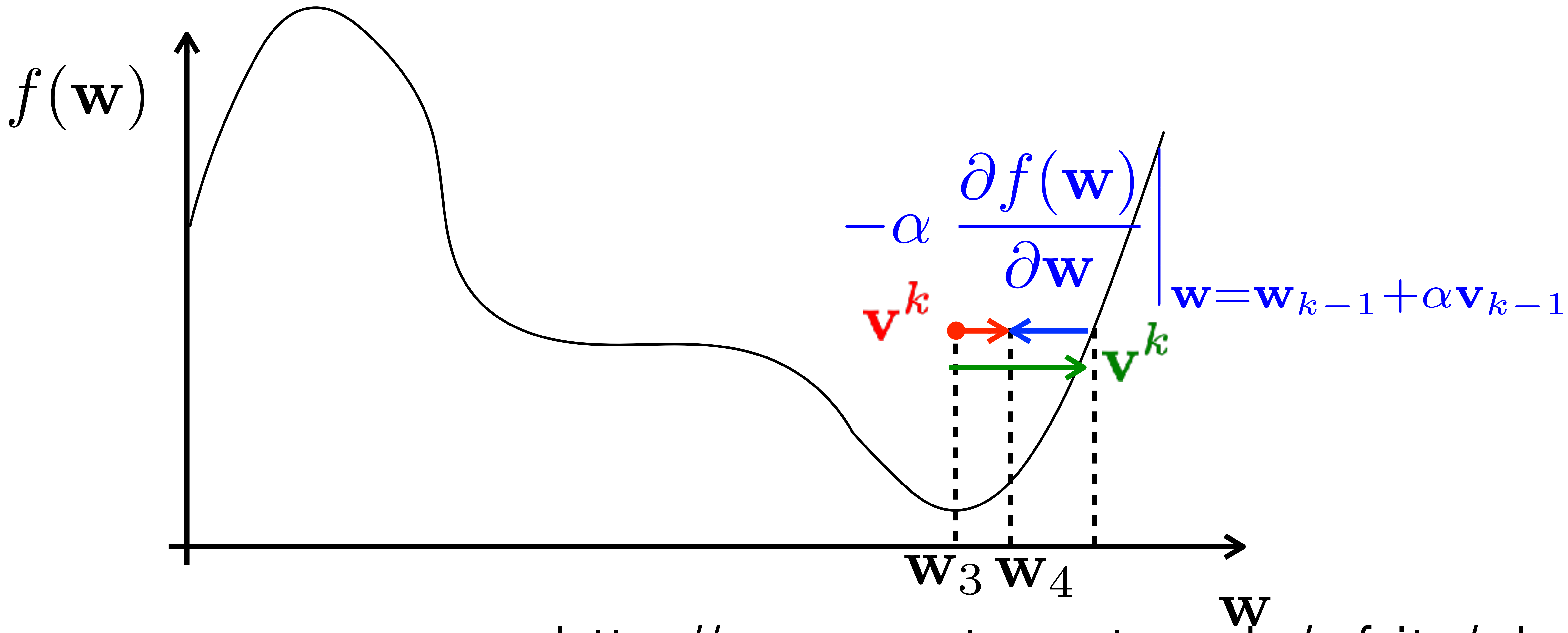


http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

SGD with Nesterov momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}+\alpha\mathbf{v}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

- Look one step ahead and reduce velocity by future gradient
- Partially prevents overshooting

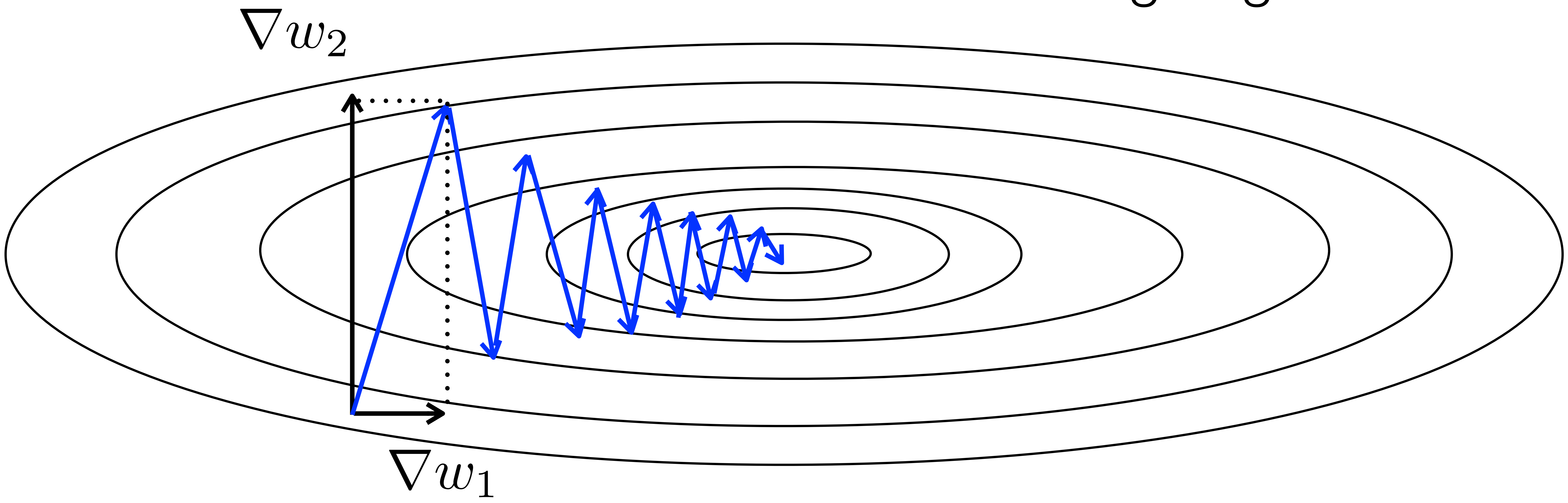


http://www.cs.toronto.edu/~fritz/absps/mon

Beyond first order methods

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Undesired zig-zag behaviour

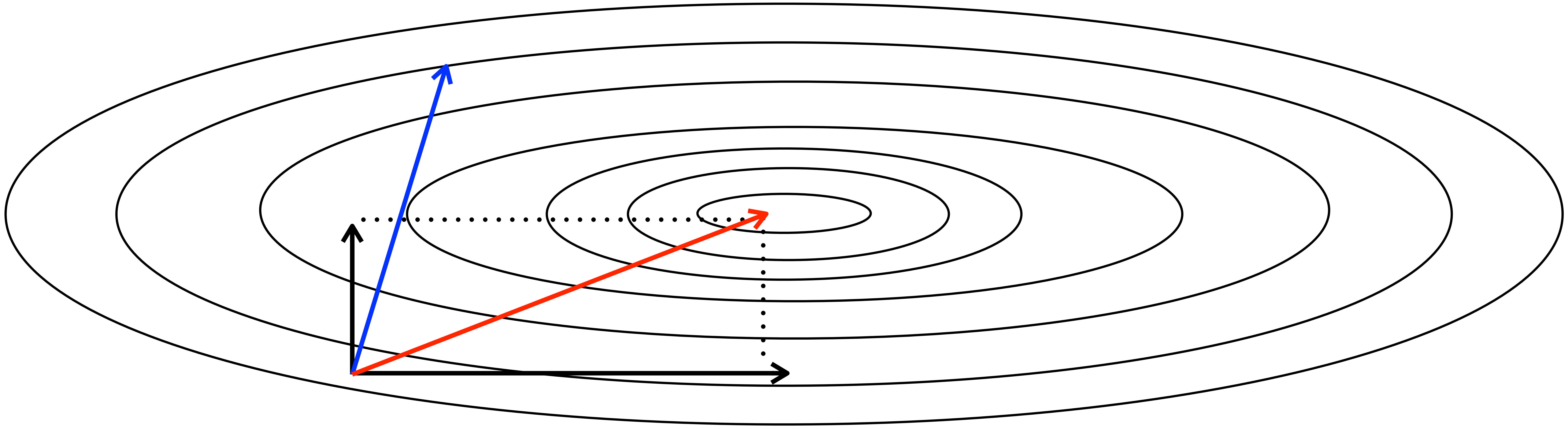


Momentum helps, but the zig-zag behaviour remains.

Full Newton Method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, H^{-1}\, \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Convergence rate for convex quadratic form is zero (converges within one step)



Hessian $H = \frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}}$ adjusts the direction of the gradient.

Convergence rate of full Newton method on quadric case study

Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i$ Hessian: $H = \left.\frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w_i}}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i$

SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

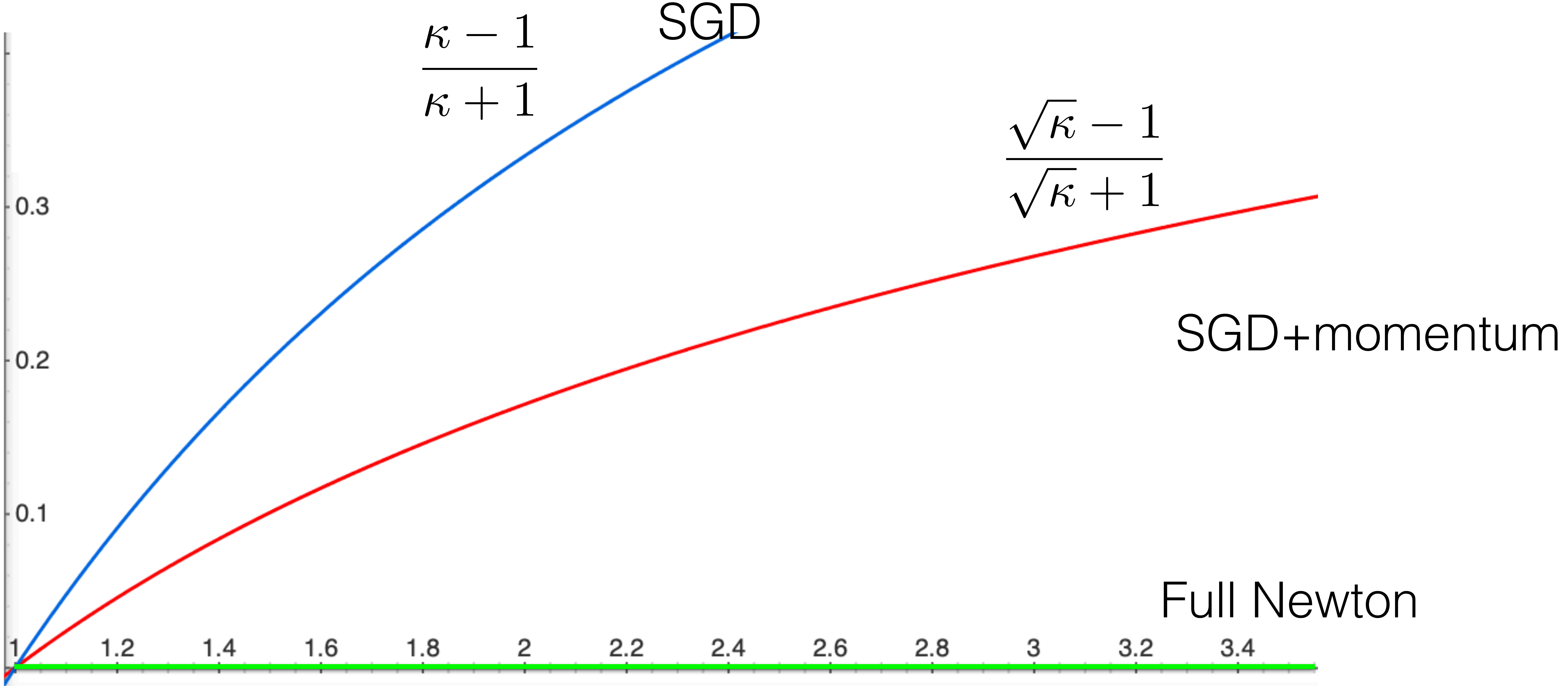
Full Newton after k iterations: $\mathbf{w}_i^k = (1 - \alpha)^k \mathbf{w}_i^0$

Optimal convergence rate: $\alpha^* = 1$

$\mathrm{rate}(\alpha^*) = 0$

SGD + momentum on quadric

Convergence rate

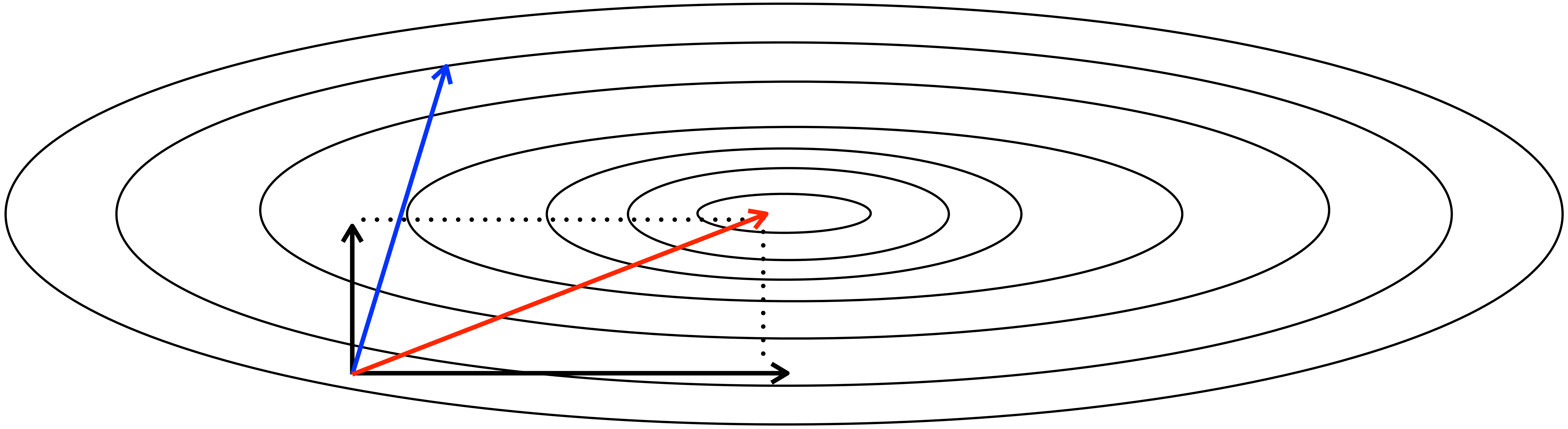


Full Newton Method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, H^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$\mathbf{g}$

Convergence rate for convex quadratic form is zero (converges within one step)



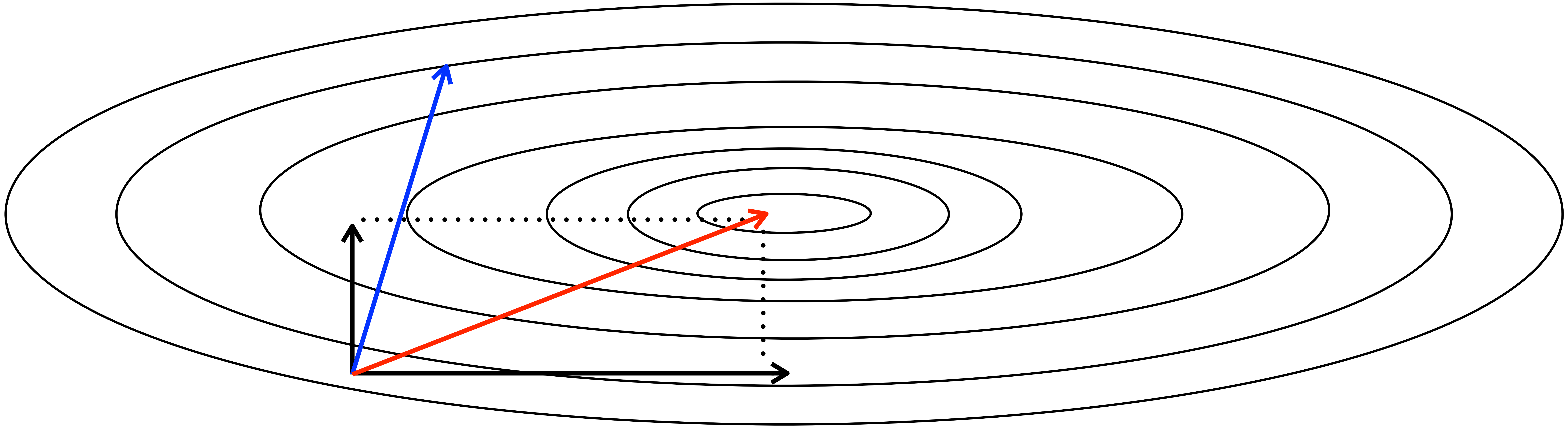
- Why not to use Hessian?
  - Hessian has $M \times M$ elements for M-dimensional parameters
  - Inverse of Hessian is $\mathcal{O}(M^3)$
  - Accurate estimate of $H^{-1} \cdot \mathbf{g}$ requiers significantly larger minibatches

Full Newton Method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, {\color{red}H^{-1}} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$\mathbf{g}$

Convergence rate for convex quadratic form is zero (converges within one step)

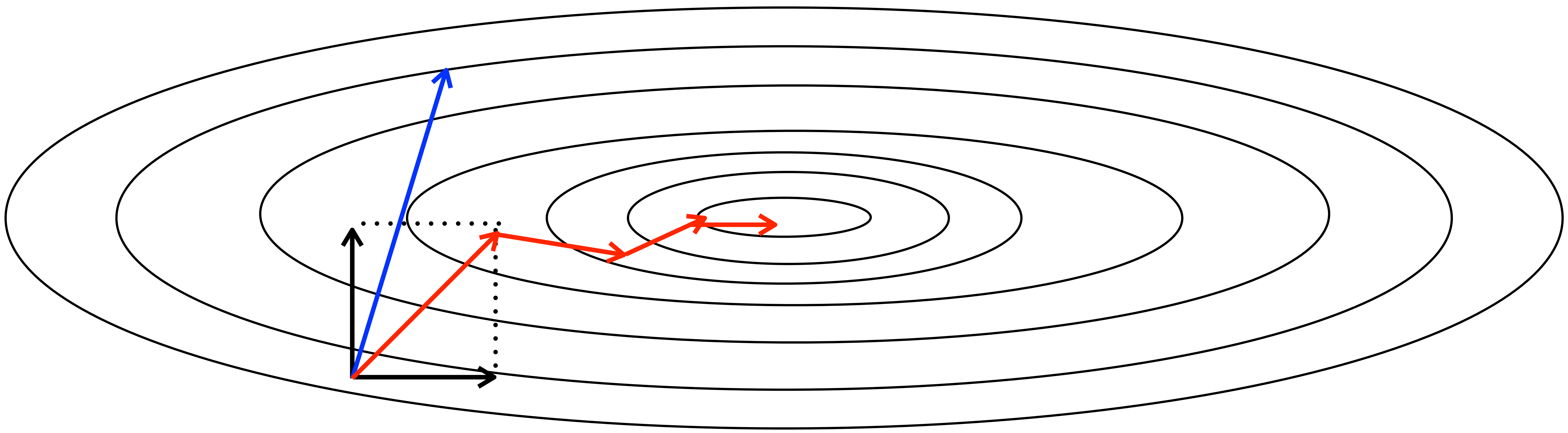


What does the Hessian actually do?

- It slows down each component by its eigenvalue
  (i.e. eigenvalue encodes steepness of the quadric in particular dimension)
- The faster the change the shorter the step

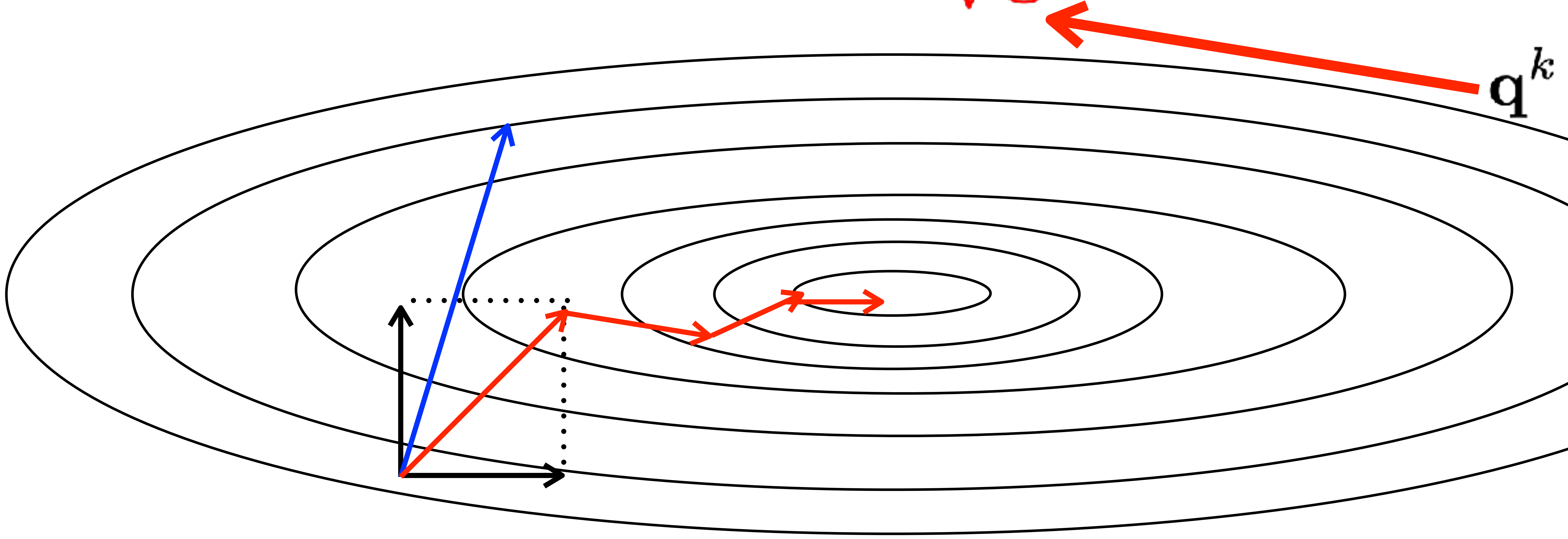
Full Newton method - approximation

$$\mathbf{w}^k \approx \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{g}^2 + \epsilon}} \odot \mathbf{g}$$



Full Newton method - approximation

$$\mathbf{w}^k \approx \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{g}^2 + \epsilon}} \odot \mathbf{g}$$



http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

RMSprop

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{g}$$

```
torch.optim.RMSprop(params, lr=0.01, alpha=0.99, eps=1e-08,
weight_decay=0, momentum=0, centered=False)
```

AdamOptimizer = AdaGrad + momentum in $\mathbf{g}$, $\mathbf{g}^2$

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{g}$$

$\mathbf{v}^k$

AdamOptimizer = AdaGrad + momentum in $\mathbf{g},\ \mathbf{g}^2$

$$\mathbf{v}^k = \beta_1 \mathbf{v}^{k-1} + (1-\beta_1)\mathbf{g}$$

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{v}^k$$

[Kingma ICLR 2015]

AdamOptimizer = AdaGrad + momentum in $\mathbf{g},\ \mathbf{g}^2$

$$\mathbf{v}^k = \beta_1 \mathbf{v}^{k-1} + (1-\beta_1)\mathbf{g}$$

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_1^k}$$

$$\hat{\mathbf{q}}^k = \frac{\mathbf{q}^k}{1-\beta_2^k}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\hat{\mathbf{q}}^k} + \epsilon} \odot \hat{\mathbf{v}}^k$$

[Kingma ICLR 2015]

AdamOptimizer = AdaGrad + momentum in $\mathbf{g}$, $\mathbf{g}^2$

$$\mathbf{v}^k = \beta_1 \mathbf{v}^{k-1} + (1-\beta_1)\mathbf{g}$$

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_1^k}$$

$$\hat{\mathbf{q}}^k = \frac{\mathbf{q}^k}{1-\beta_2^k}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\hat{\mathbf{q}}^k} + \epsilon} \odot \hat{\mathbf{v}}^k$$

Default values: $\alpha = 0.1$ $\beta_1 = 0.9$ $\beta_2 = 0.999$

```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999),
            eps=1e-08, weight_decay=0, amsgrad=False)
```

Summary

- Adam is the most popular choice, since it is not that sensitive to other hyper-parameters.

- PyTorch of all previously mentioned implementations available:

```
torch.optim.Adam(params, lr=0.001,
        betas=(0.9, 0.999),eps=1e-08,
        weight_decay=0, amsgrad=False)
```

- Anything more complex than Adam typically suffers from sub-linear returns in huge state-of-the-art networks https://arxiv.org/abs/1805.02338v1

- There is a whole family of Quasi-Newton methods, which make use of advanced Hessian approximations (L-BFGS).

```
torch.optim.LBFGS(params, lr=1, …)
```

BFGS: Where does the name came from?

Broyden

BFGS: Where does the name came from?

Broyden **F**letcher

BFGS: Where does the name came from?

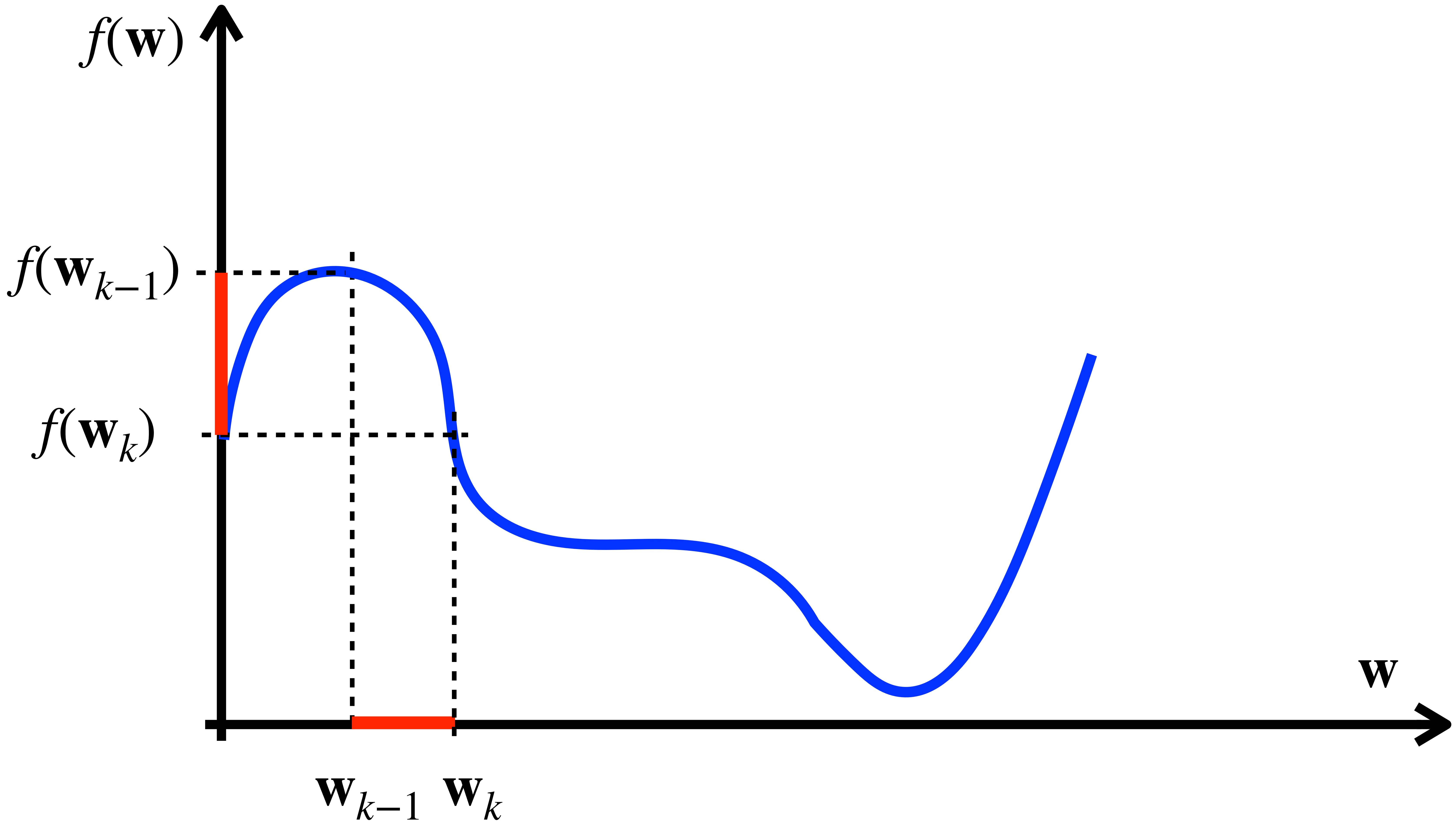
Broyden **F**letcher **G**oldfarb

BFGS: Where does the name came from?

Broyden **F**letcher **G**oldfarb **S**hanno

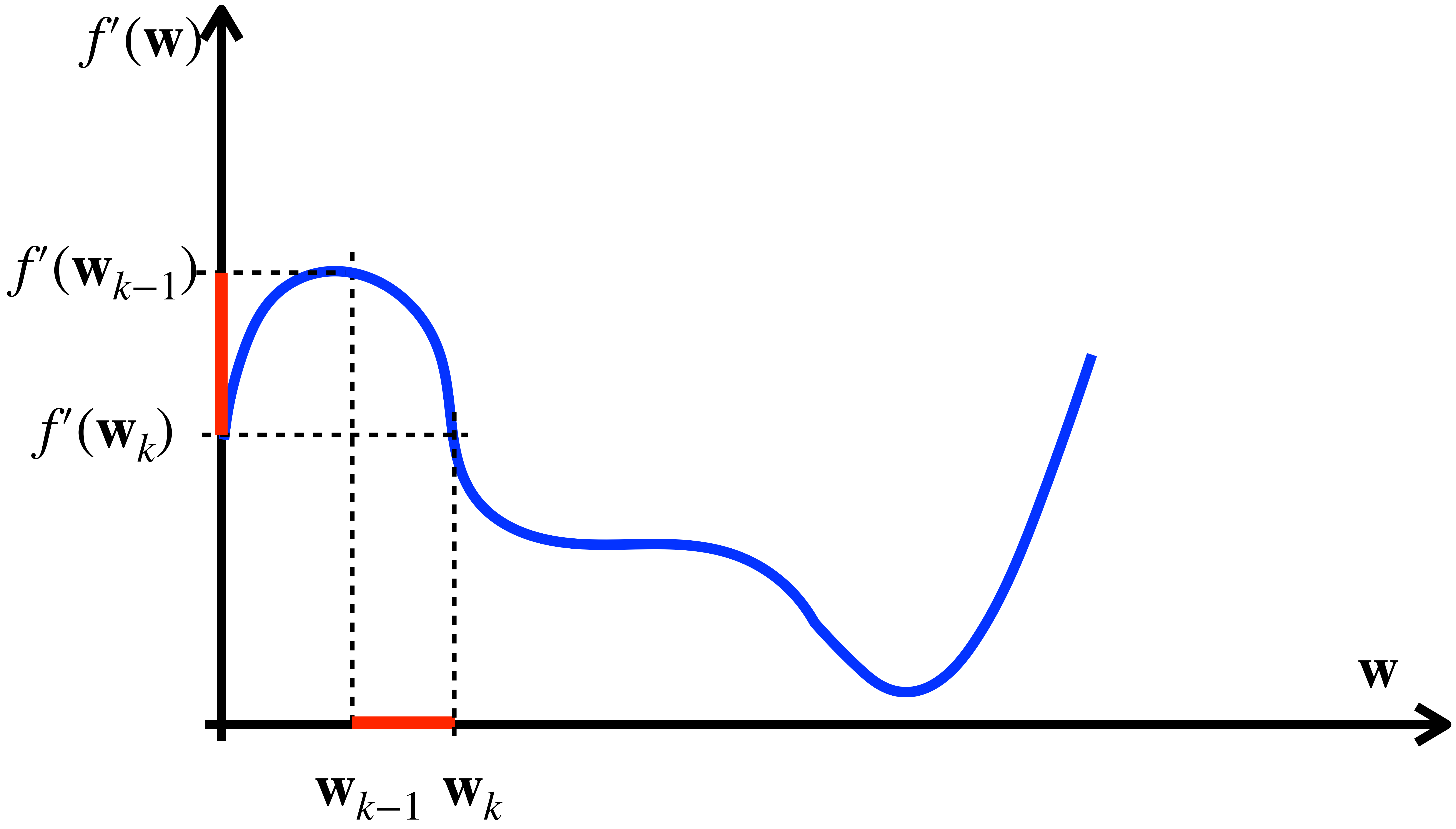
Broyden **F**letcher **G**oldfarb **S**hanno

Approximating 1D gradient: $$f'(\mathbf{w}_k) \approx \frac{f(\mathbf{w}_k) - f(\mathbf{w}_{k-1})}{\mathbf{w}_k - \mathbf{w}_{k-1}}$$



Broyden **F**letcher **G**oldfarb **S**hanno

Approximating 1D hessian: $f''(\mathbf{w}_k) \approx \dfrac{f'(\mathbf{w}_k) - f'(\mathbf{w}_{k-1})}{\mathbf{w}_k - \mathbf{w}_{k-1}}$



Broyden **F**letcher **G**oldfarb **S**hanno

Approximating 1D hessian: $f''(\mathbf{w}_k) \approx \dfrac{f'(\mathbf{w}_k) - f'(\mathbf{w}_{k-1})}{\mathbf{w}_k - \mathbf{w}_{k-1}}$

Approximating 1D Newton method (secant method):

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \frac{1}{f''(\mathbf{w}_k)} \cdot f'(\mathbf{w}_k) \approx \mathbf{w}_k - \frac{f'(\mathbf{w}_k) - f'(\mathbf{w}_{k-1})}{\mathbf{w}_k - \mathbf{w}_{k-1}} \cdot f'(\mathbf{w})$$

Broyden **F**letcher **G**oldfarb **S**hanno

Approximating 1D hessian: $f''(\mathbf{w}_k) \approx \dfrac{f'(\mathbf{w}_k) - f'(\mathbf{w}_{k-1})}{\mathbf{w}_k - \mathbf{w}_{k-1}} = \hat{\mathbf{H}}_k$

Approximating N-D hessian:

$\hat{\mathbf{H}}_k \cdot (\mathbf{w}_k - \mathbf{w}_{k-1}) = \nabla f_k - \nabla f_{k-1}$

Can I solve it?

(NxN)/2 unknowns, but only N equations

$\arg\min_{\hat{\mathbf{H}}_k} \|\hat{\mathbf{H}}_k - \hat{\mathbf{H}}_{k-1})\|_F$ .... close to previous hessian approximation

subject to : $\hat{\mathbf{H}}_k = \hat{\mathbf{H}}_k^\top$ .... symmetric

$\hat{\mathbf{H}}_k \cdot (\mathbf{w}_k - \mathbf{w}_{k-1}) = \nabla f_k - \nabla f_{k-1}$ ... approximate hessian via secan method

$$\hat{\mathbf{H}}_k^\star = \hat{\mathbf{H}}_{k-1} + \frac{(\nabla f_k - \nabla f_{k-1})(\nabla f_k - \nabla f_{k-1})^\top}{(\nabla f_k - \nabla f_{k-1})^\top \Delta \mathbf{w_k}} + \frac{\hat{\mathbf{H}}_{k-1}\Delta\mathbf{w}_k\Delta\mathbf{w}_k^\top\hat{\mathbf{H}}_{k-1}}{\Delta\mathbf{w}_k^\top\hat{\mathbf{H}}_{k-1}\Delta\mathbf{w}_k}$$ .... analytical solution

Approximating N-D Newton method (BFGS method):

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{H}_k^{-1} \cdot \nabla f_k \approx \mathbf{w}_k - \hat{\mathbf{H}}_k^{\star\,-1} \cdot \nabla f_k$$

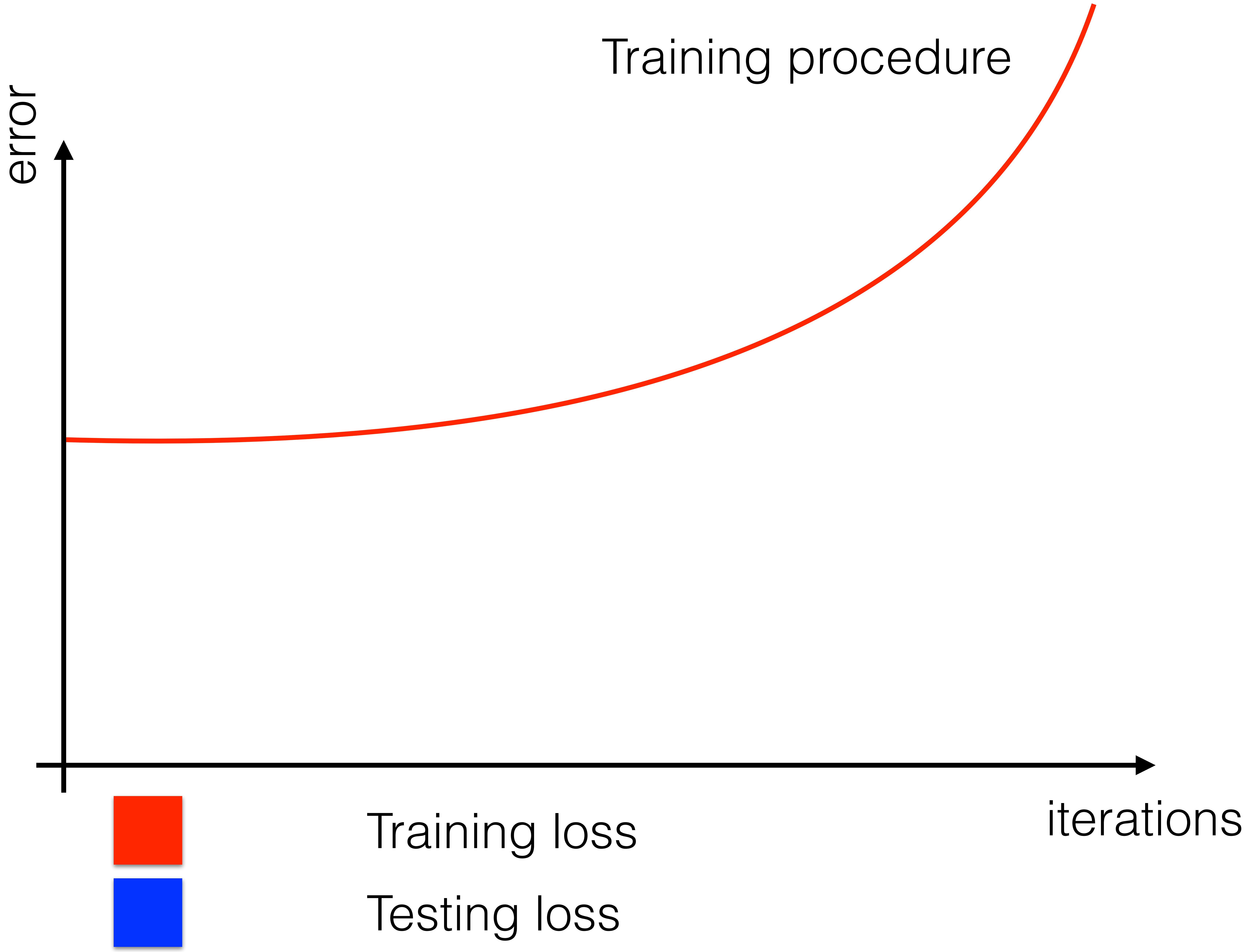
Broyden **F**letcher **G**oldfarb **S**hanno

Applications everywhere, where Hessian computation is too painful but GD suffers from slow convergence:

- Structural design (bridges, buildings, aerospace components) e.g. minimize material usage or structural properties
- Process optimization (chemistry)
- Tuning control system (automotive, aerospace, aircrafts, powerplants)
- Computer graphics (mesh fitting with structural priors)
- ….

Training procedure

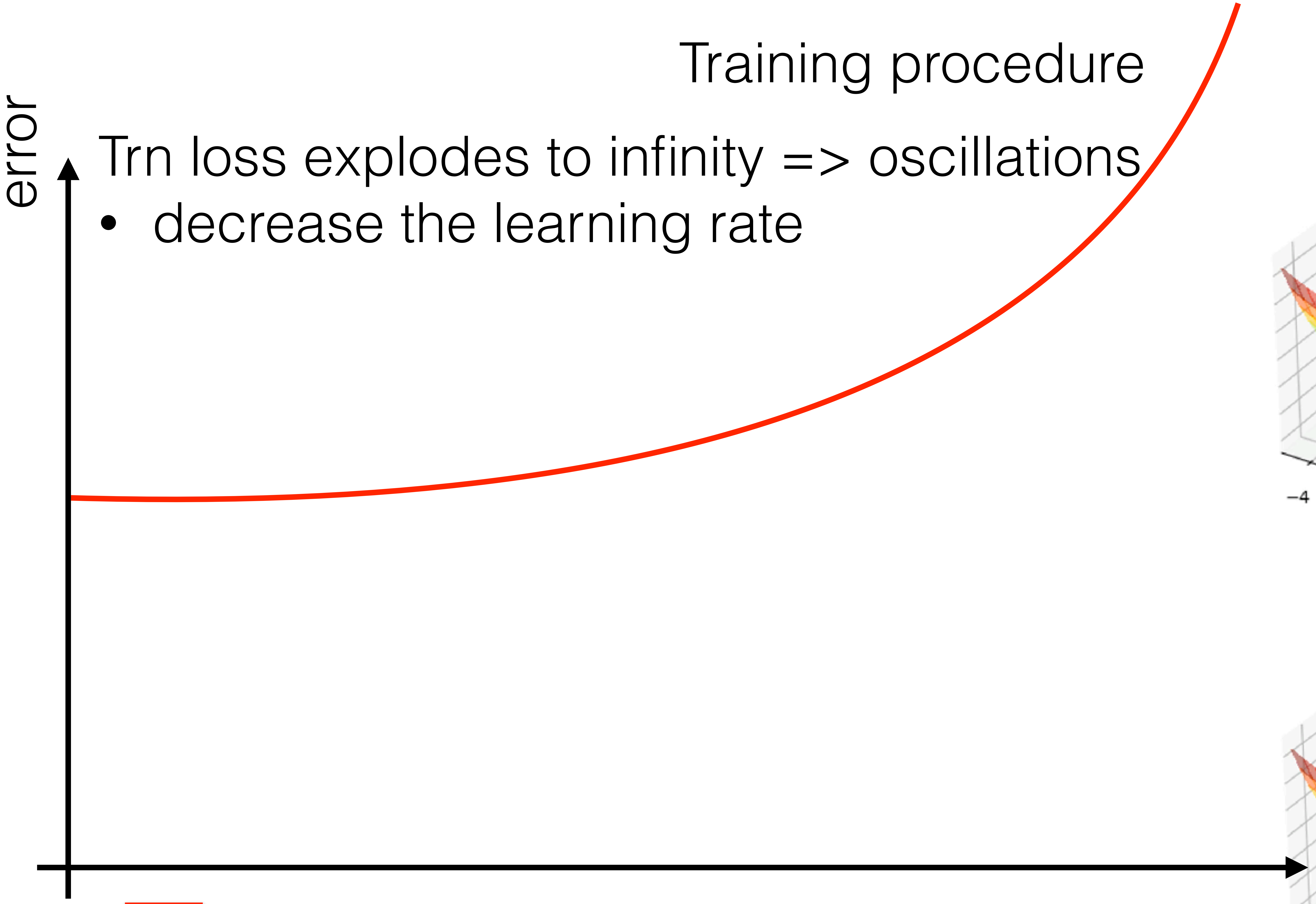
- Choose:
  - Network architecture (ideally re-use pre-trained net)
  - Weight initialization (Xavier)
  - Learning rate and other hyper-parameters.
  - Loss + regularization
- Divide data on three representative subsets:
  - Training data (the set on which the backprop is used to estimate weights)
  - Validation data (the set on which hyper-param are tuned)
  - Testing data (the set on which the error is reported)

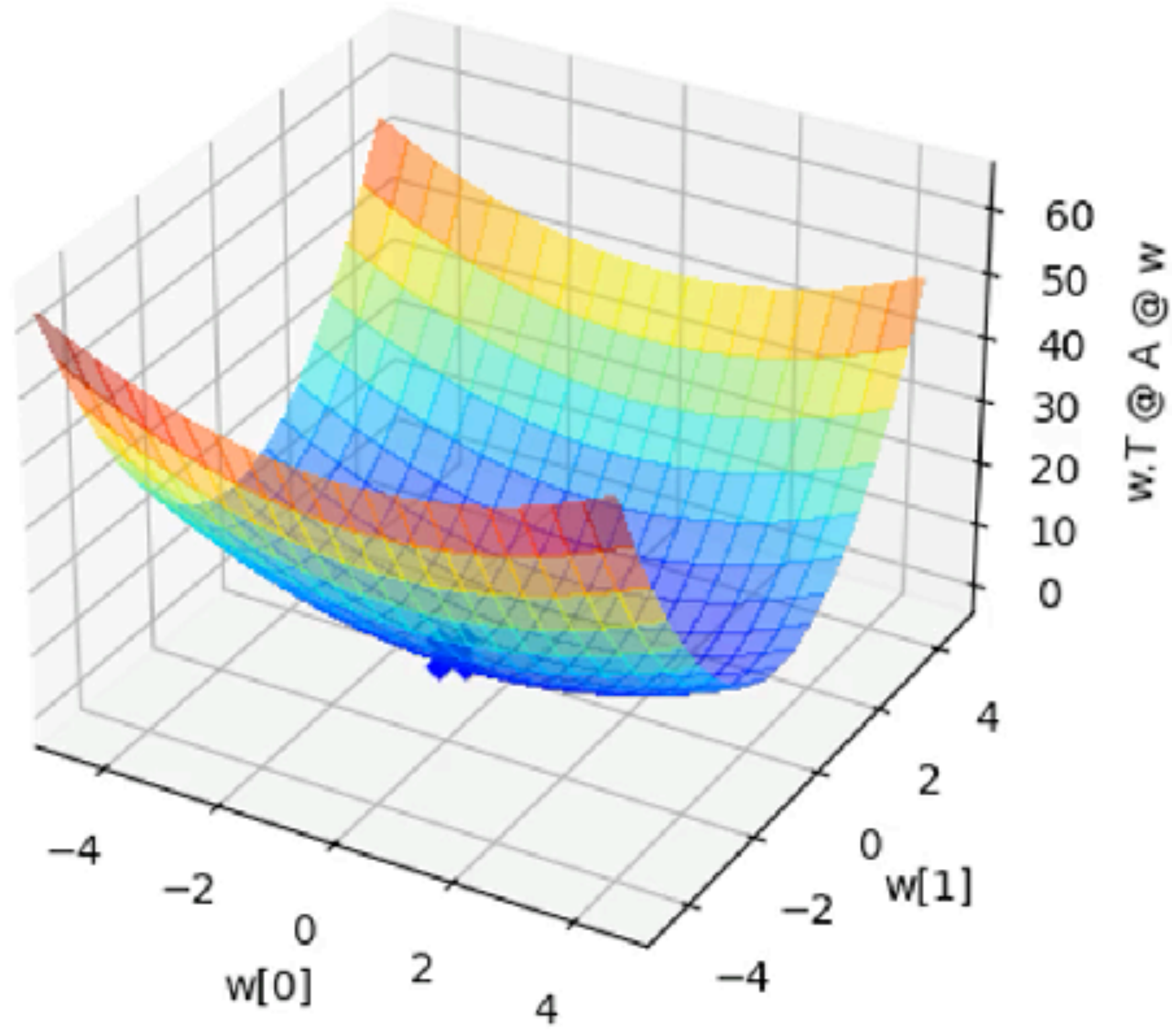

Training procedure
error
iterations
Training loss
Testing loss

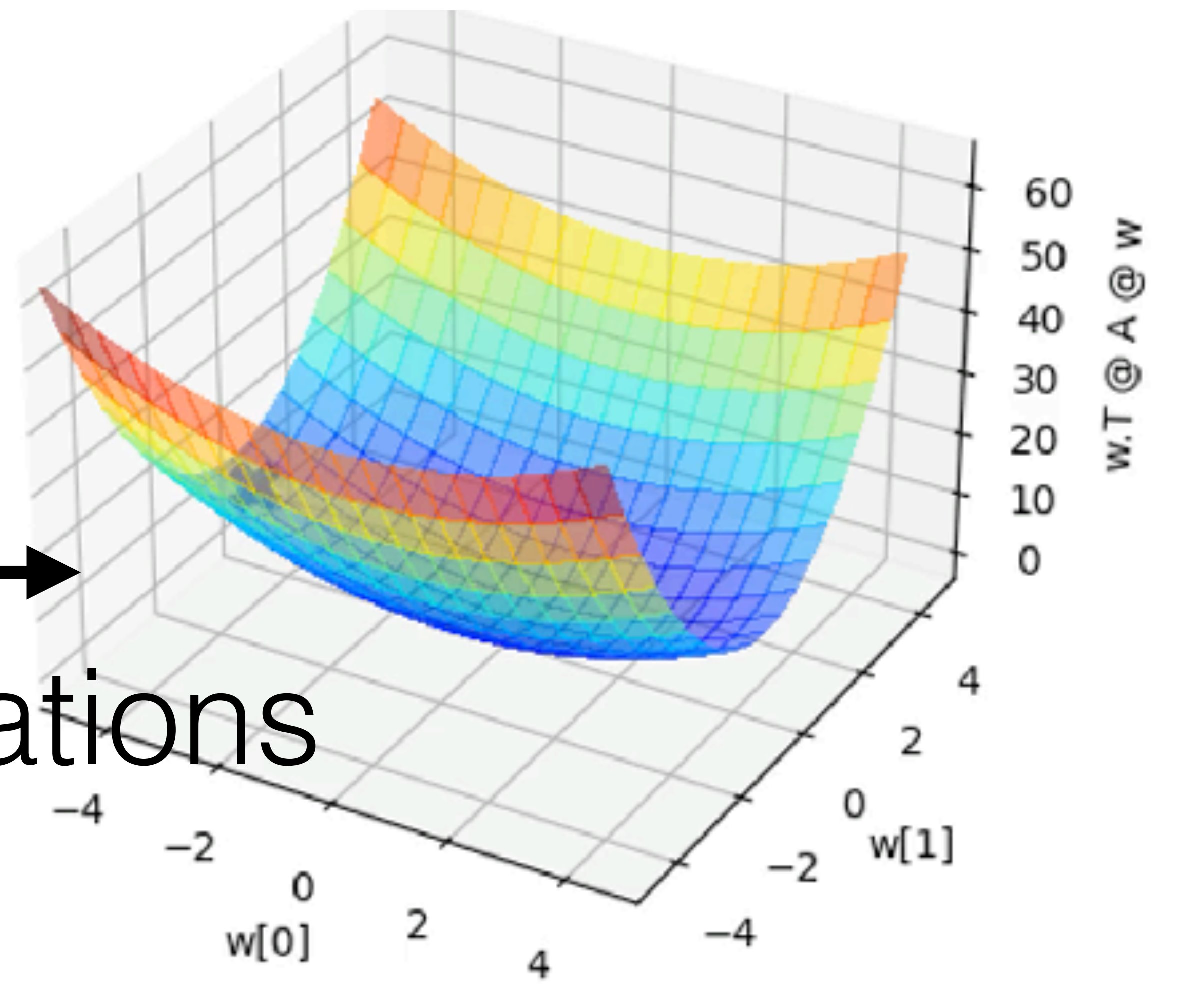
Training procedure

Trn loss explodes to infinity => oscillations

- decrease the learning rate

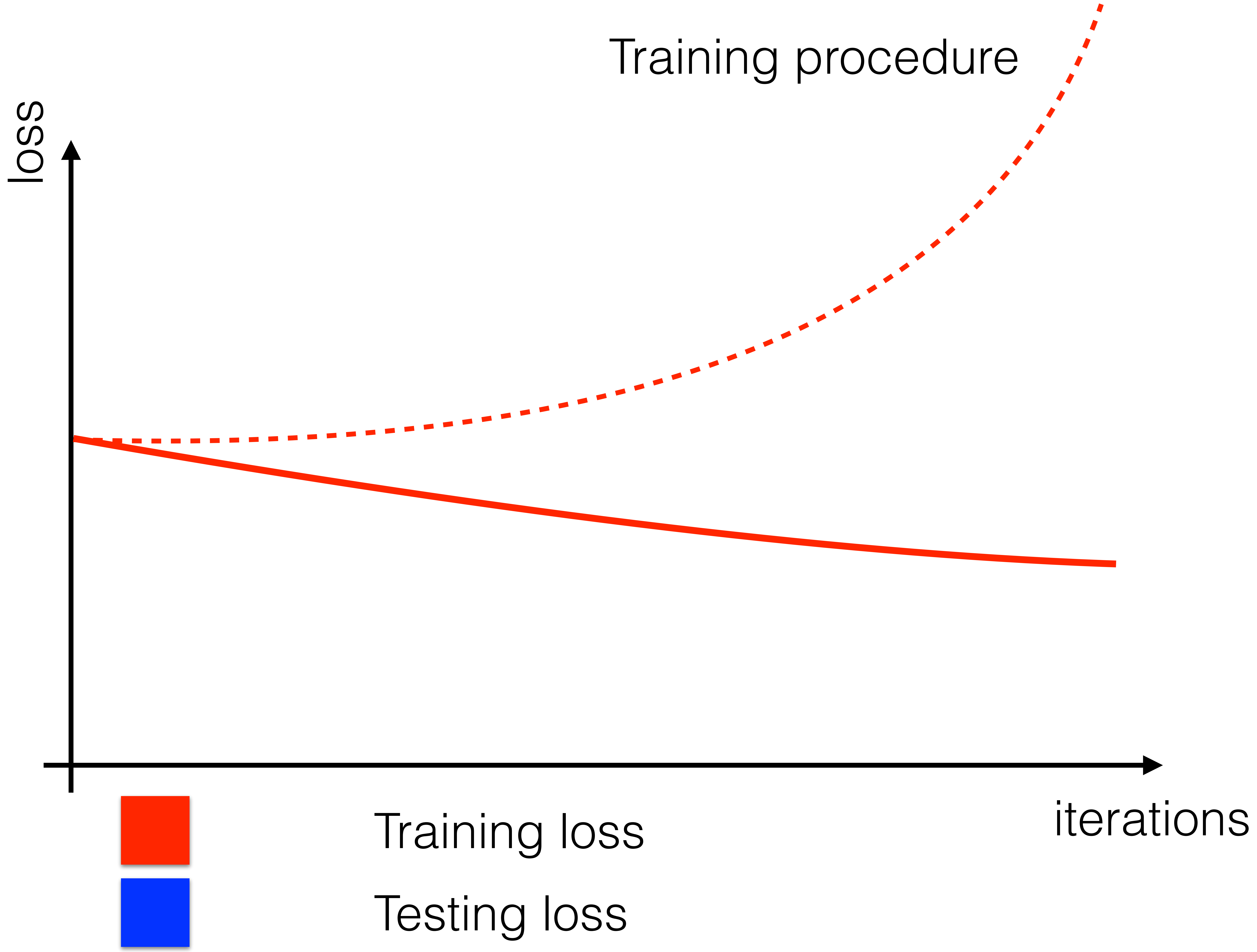




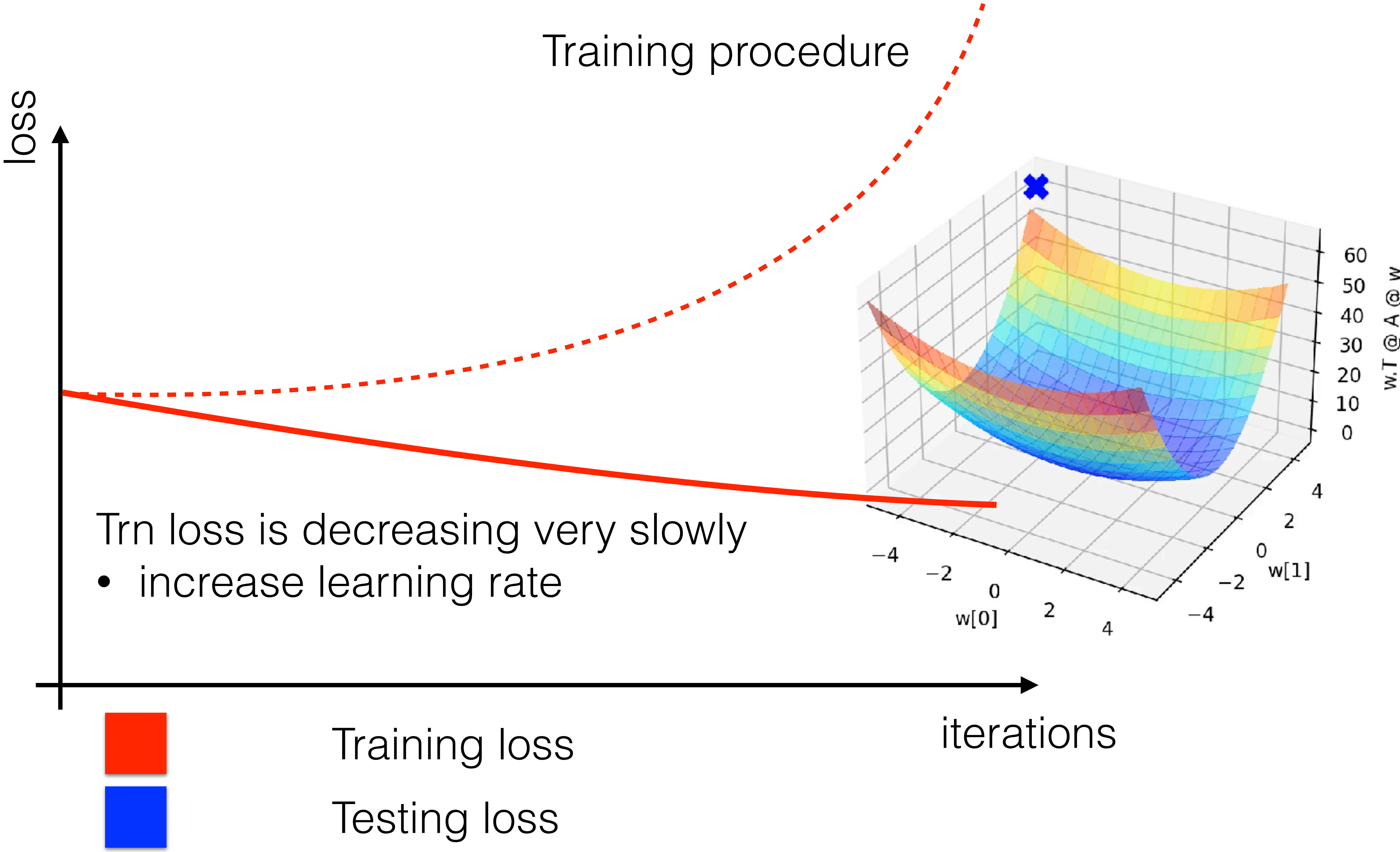


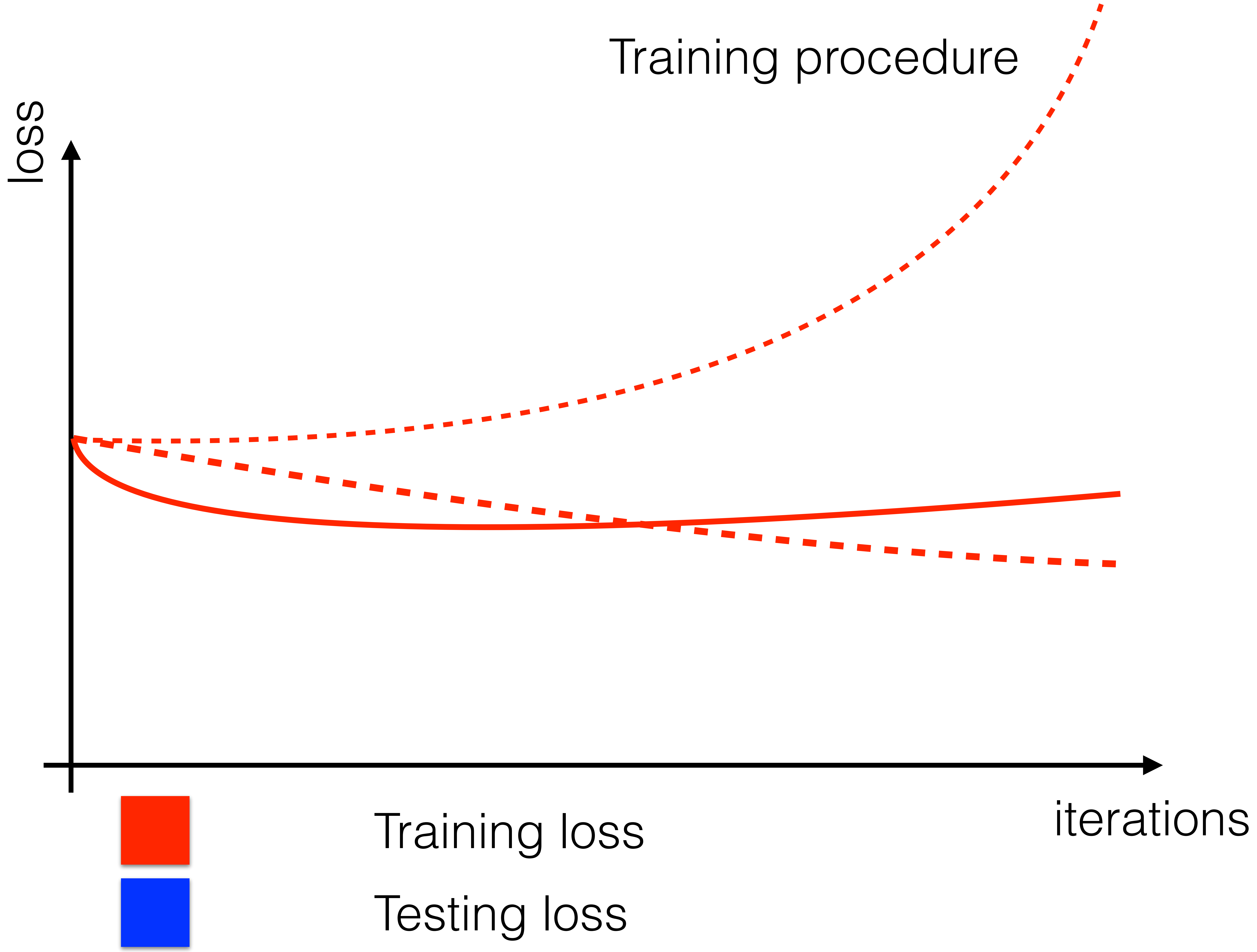
Training loss

Testing loss


Training procedure
loss
iterations
Training loss
Testing loss

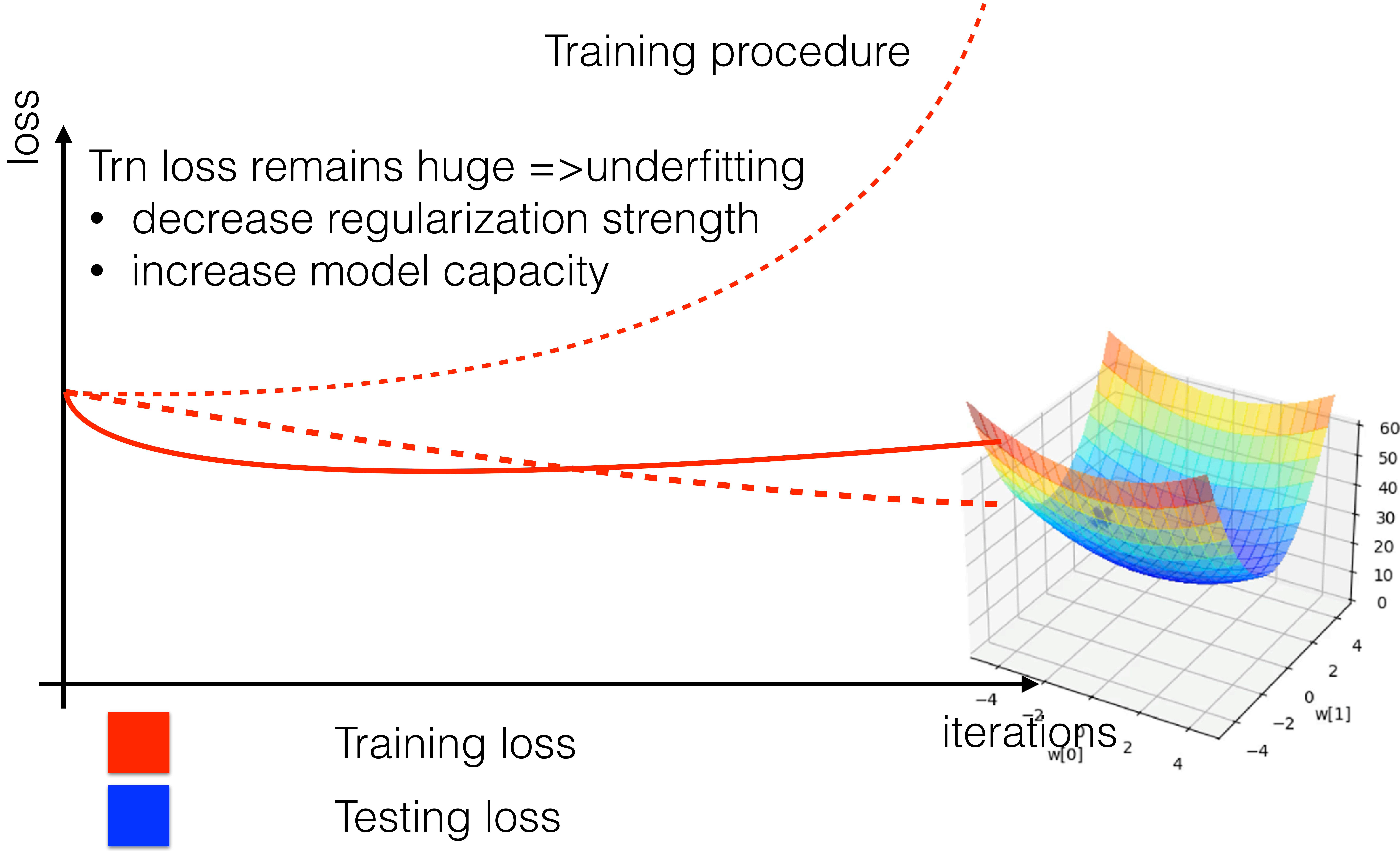
Training procedure



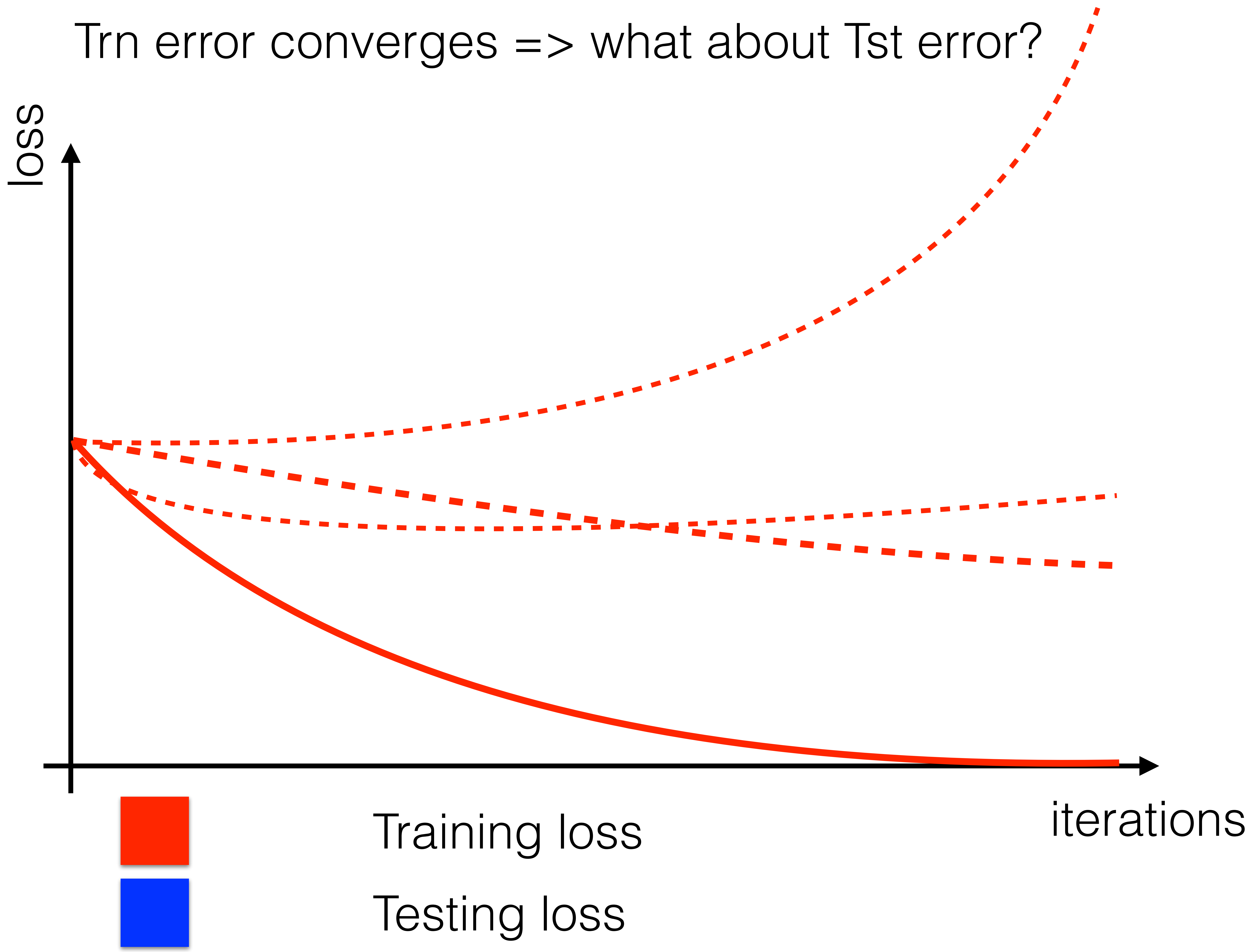

Training procedure
loss
iterations
Training loss
Testing loss

Training procedure

Trn loss remains huge =>underfitting
- decrease regularization strength
- increase model capacity

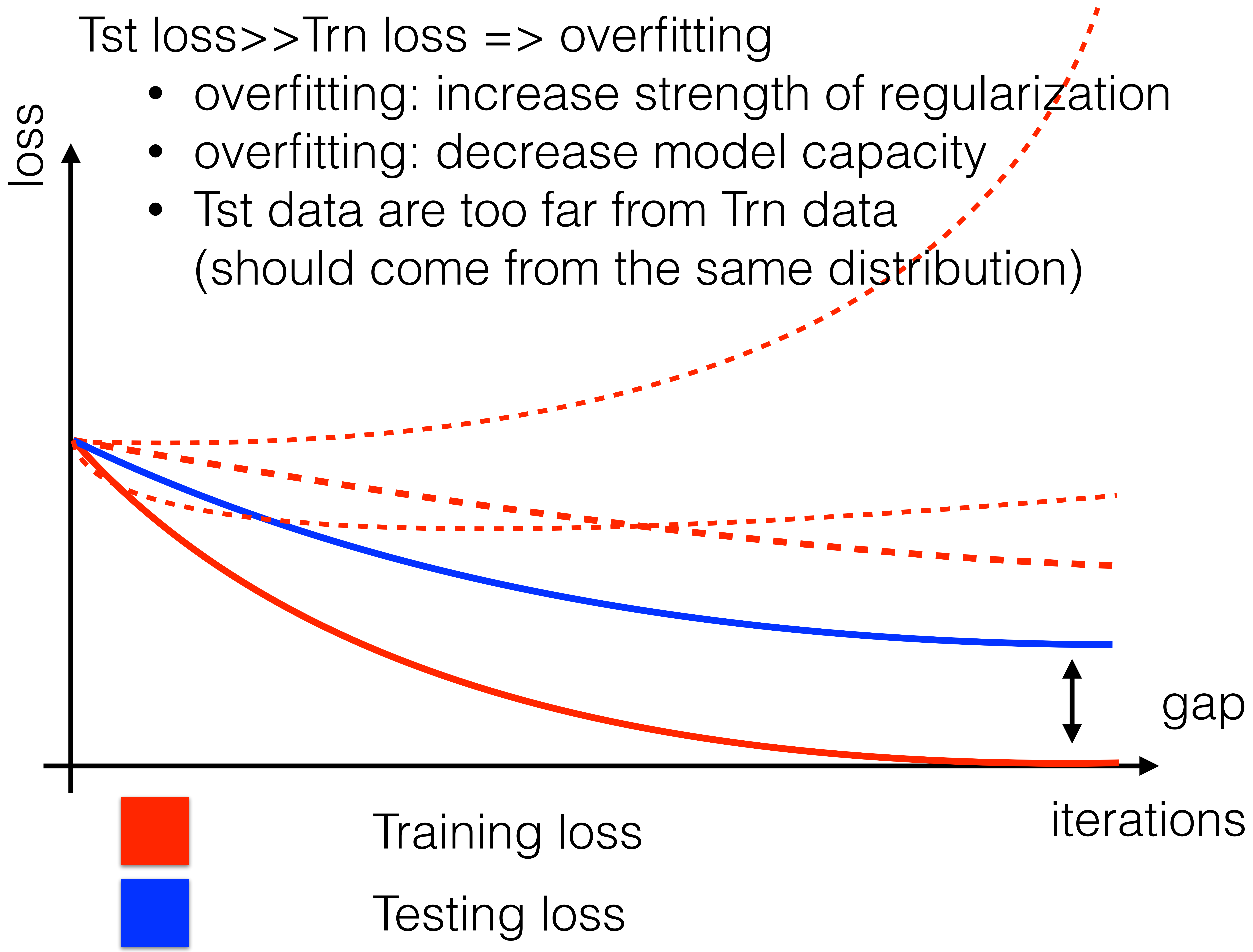


Trn error converges => what about Tst error?



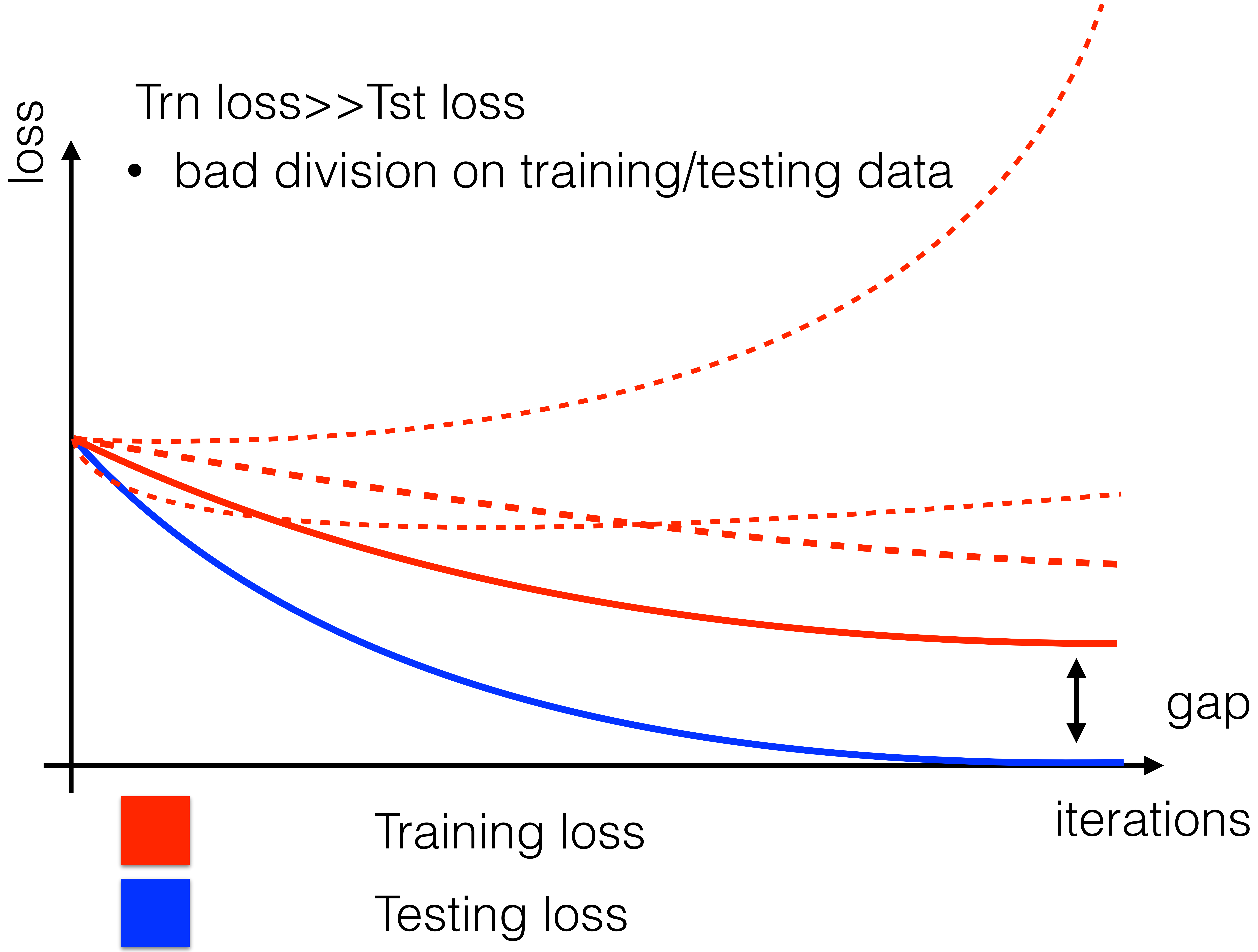
Tst loss>>Trn loss => overfitting

- overfitting: increase strength of regularization
- overfitting: decrease model capacity
- Tst data are too far from Trn data
(should come from the same distribution)

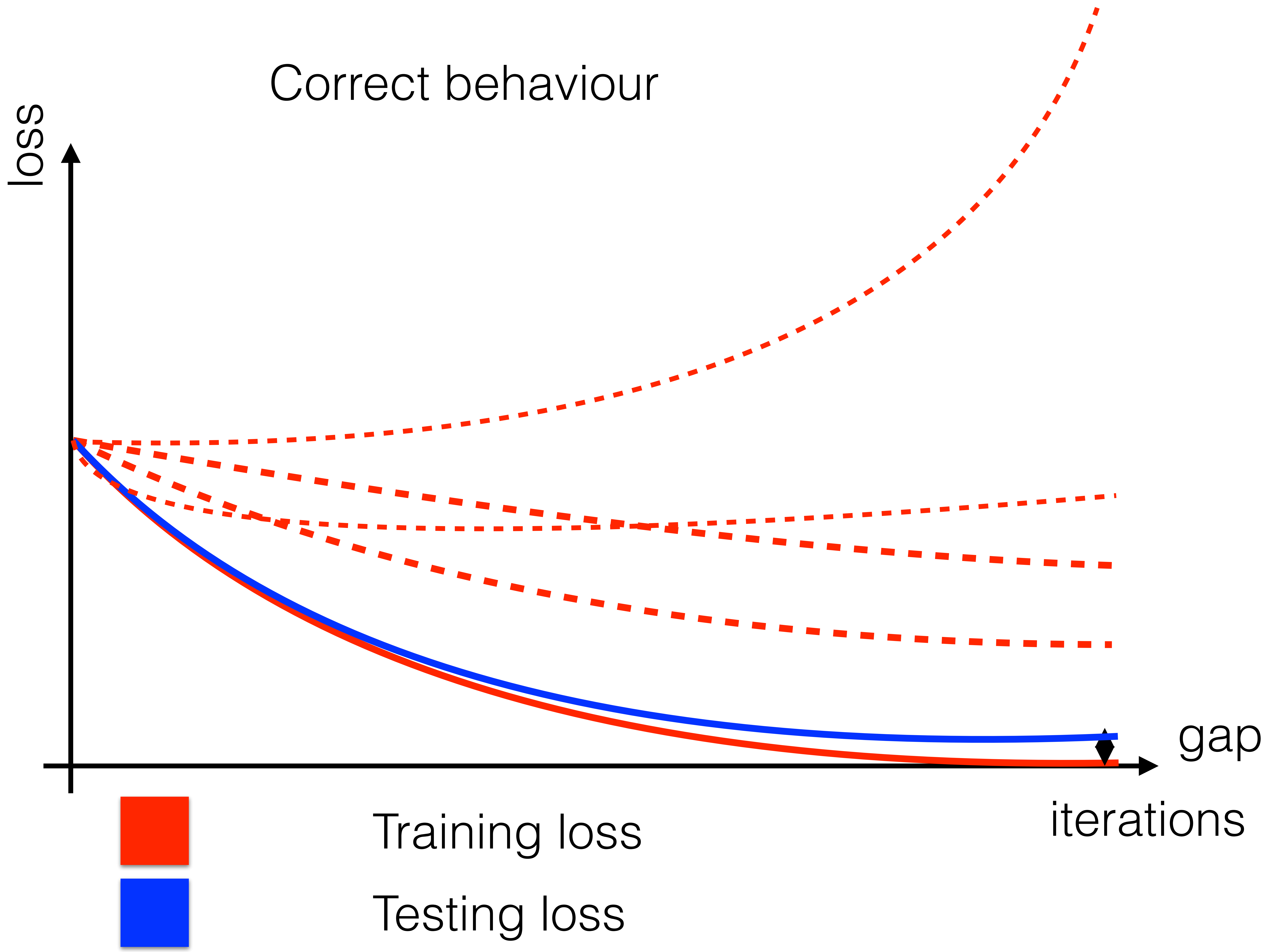


Trn loss>>Tst loss

- bad division on training/testing data



Correct behaviour



Hyper parameters tuning

- Weight initialization (Xavier)
- Trn loss is huge =>underfitting
  - decrease regularization strength
  - increase model capacity
- Trn loss explodes to infinity=> huge learning rate
  - decrease the learning rate
- Trn loss is decreasing very slowly => small learning rate
  - increase learning rate
- Tst loss>>Trn loss => overfitting
  - increase strength of regularization
  - decrease model capacity
  - Tst data are too far from Trn data
    (should come from the same distribution)
- Trn loss>>Tst loss =>bad division on training/testing data

Epilog

- Structural vs optimization issues
- Newton : $f(\mathbf{x}) \approx \frac{1}{2}\mathbf{x}^\top H\mathbf{x} + \mathbf{g}\mathbf{x} + c \quad \Rightarrow H\mathbf{x} + \mathbf{g} = 0 \quad \Rightarrow \mathbf{x} = H^{-1}\mathbf{g}$
- Adam : $\hat{H} = \begin{bmatrix} \bar{g}_1 & 0 & \dots & 0 \\ 0 & \bar{g}_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \bar{g}_n \end{bmatrix} \quad \Rightarrow \mathbf{x} = \hat{H}^{-1}\mathbf{g}$

- Optimization vs Learning

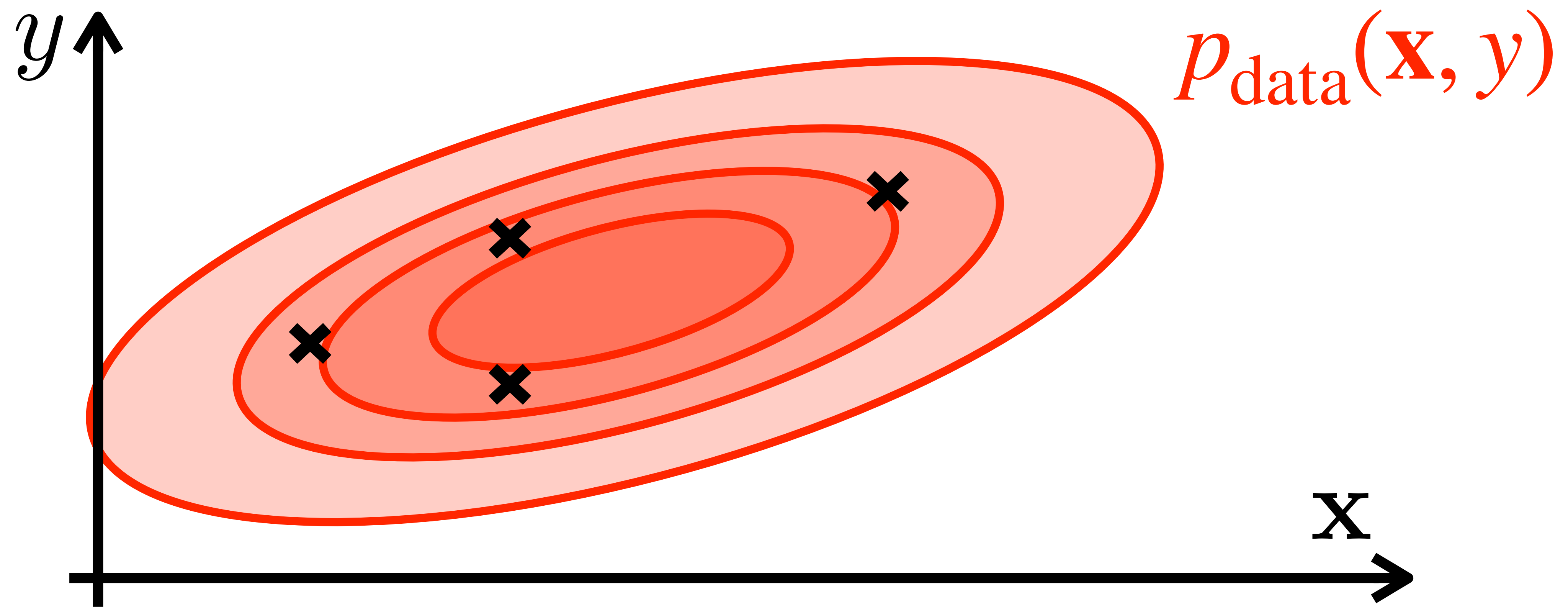
$$J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\log p(\mathbf{x}, y \mid \mathbf{w})\right]$$

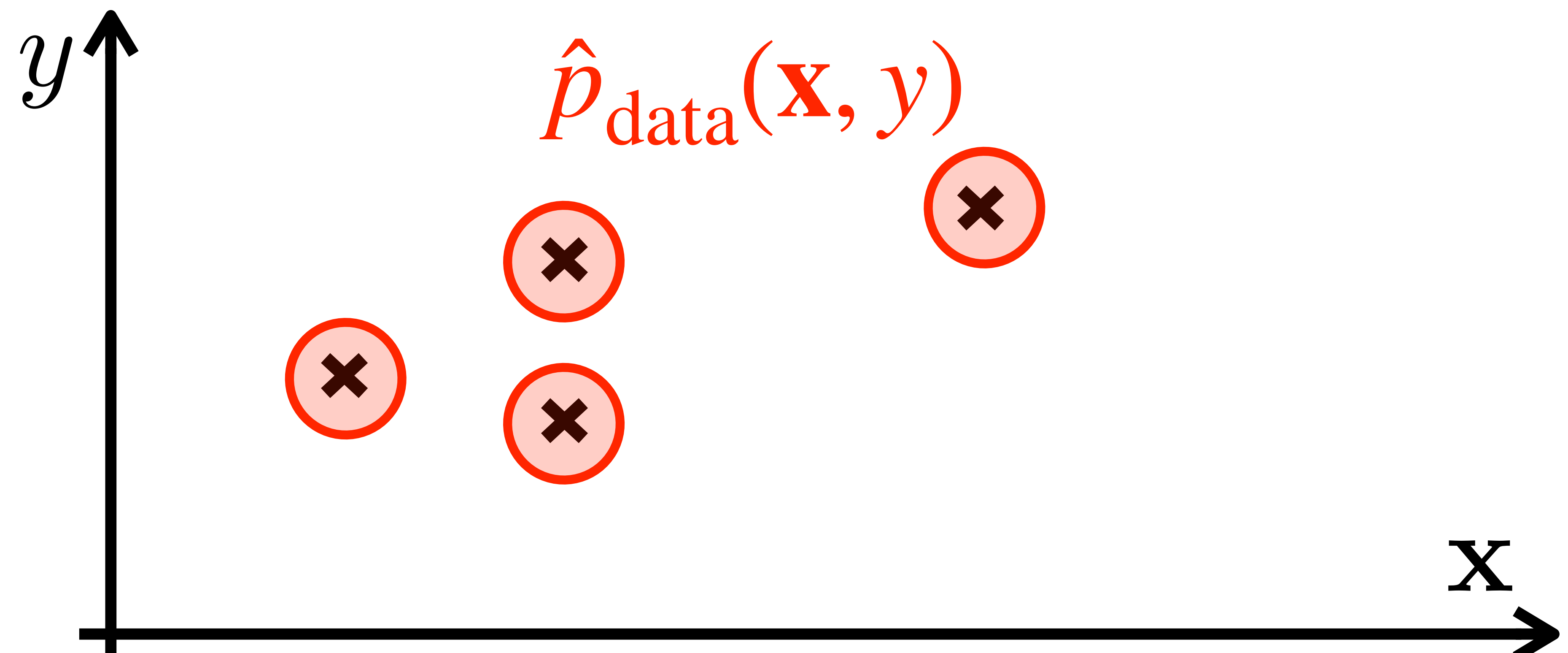
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \mid \mathbf{w})\right)\right]$$

vs.

$$\hat{J}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim \hat{p}_{\text{data}}}\left[\log p(\mathbf{x}, y \mid \mathbf{w})\right]$$

$$\nabla_{\mathbf{w}} \hat{J}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \mid \mathbf{w})\right)\right]$$

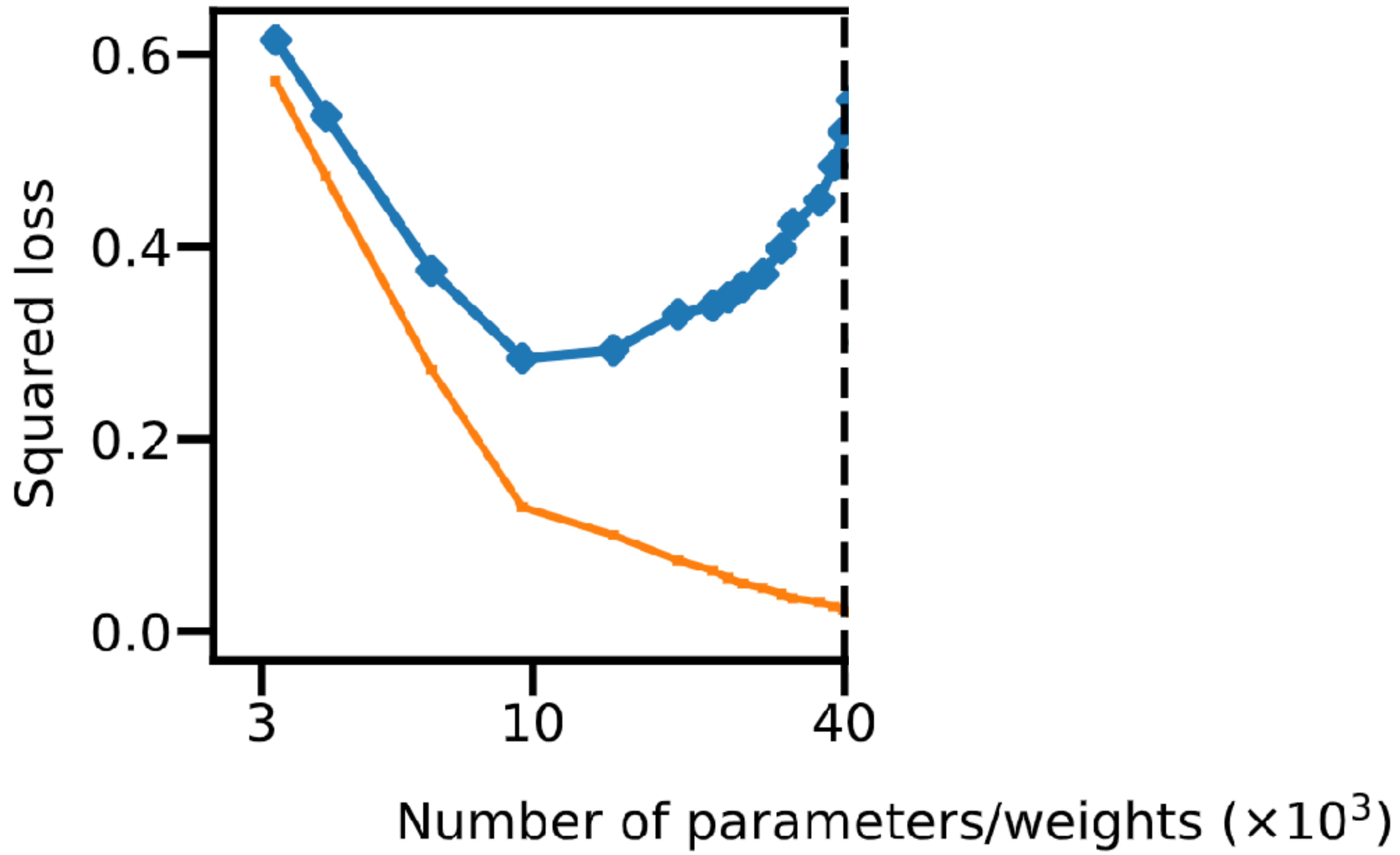




Epilog

- Stochasticity advantage:
  - reduces computational time and memory
  - works as an regularizer that helps in larger models (avoids wild hypothesis)
  - avoids getting stuck in saddle points
- Stochasticity drawback:
  - makes estimation of gradient and Hessian inaccurate (Standard deviation $\approx 1/\sqrt{N}$ => suffers from sub-linear returns)
  - smaller N +higher M avoids advanced Hessian approx methods (LBFGS) =>Adam is typically used
- Momentum advantage:
  - jumps over sharp minima
  - avoid getting stuck in flat regions
  - suppress oscillations

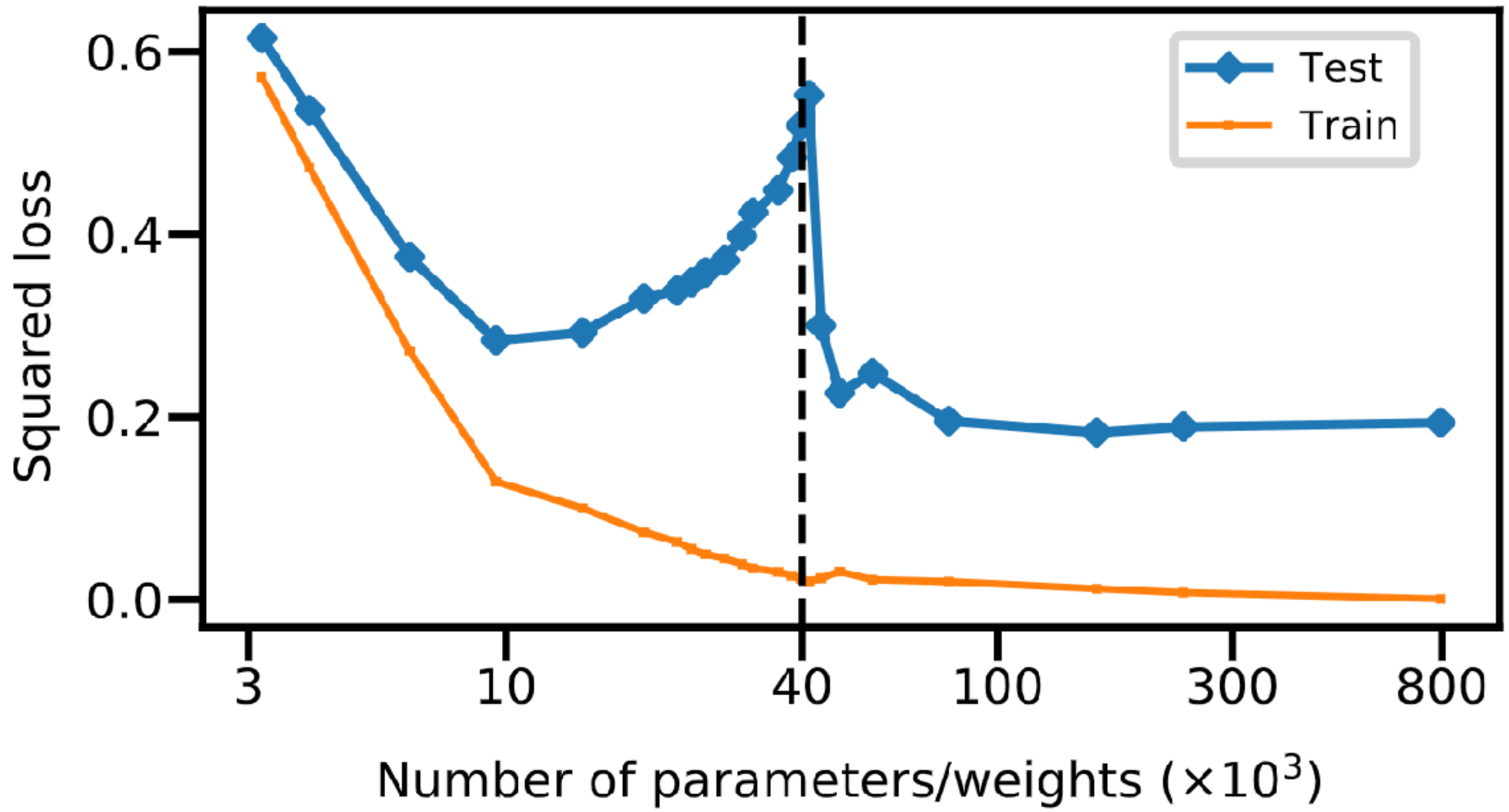
Double descent
[Belkin-DoubleDescent-2019]



Statistical wisdom: “Too large models are worse since they overfit.”

https://openai.com/blog/deep-double-descent/

Double descent
[Belkin-DoubleDescent-2019]



Statistical wisdom: “Too large models are worse since they overfit.”

Deep ML wisdom: “The larger the better”

https://openai.com/blog/deep-double-descent/

Double descent
[Belkin-DoubleDescent-2019]

https://openai.com/blog/deep-double-descent/