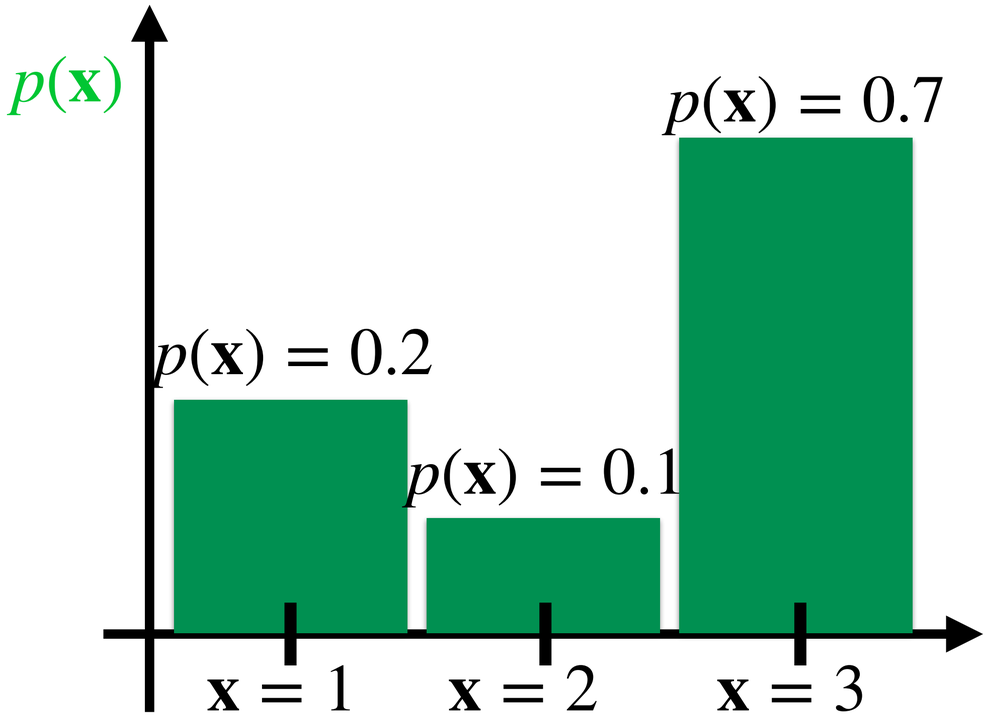
Why is learning prone to fail?

Optimization issues: SGD+momentum and its convergence rate, AdamW, Newton method, BFGS, diminishing/exploding gradient, oscillations, double descent

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

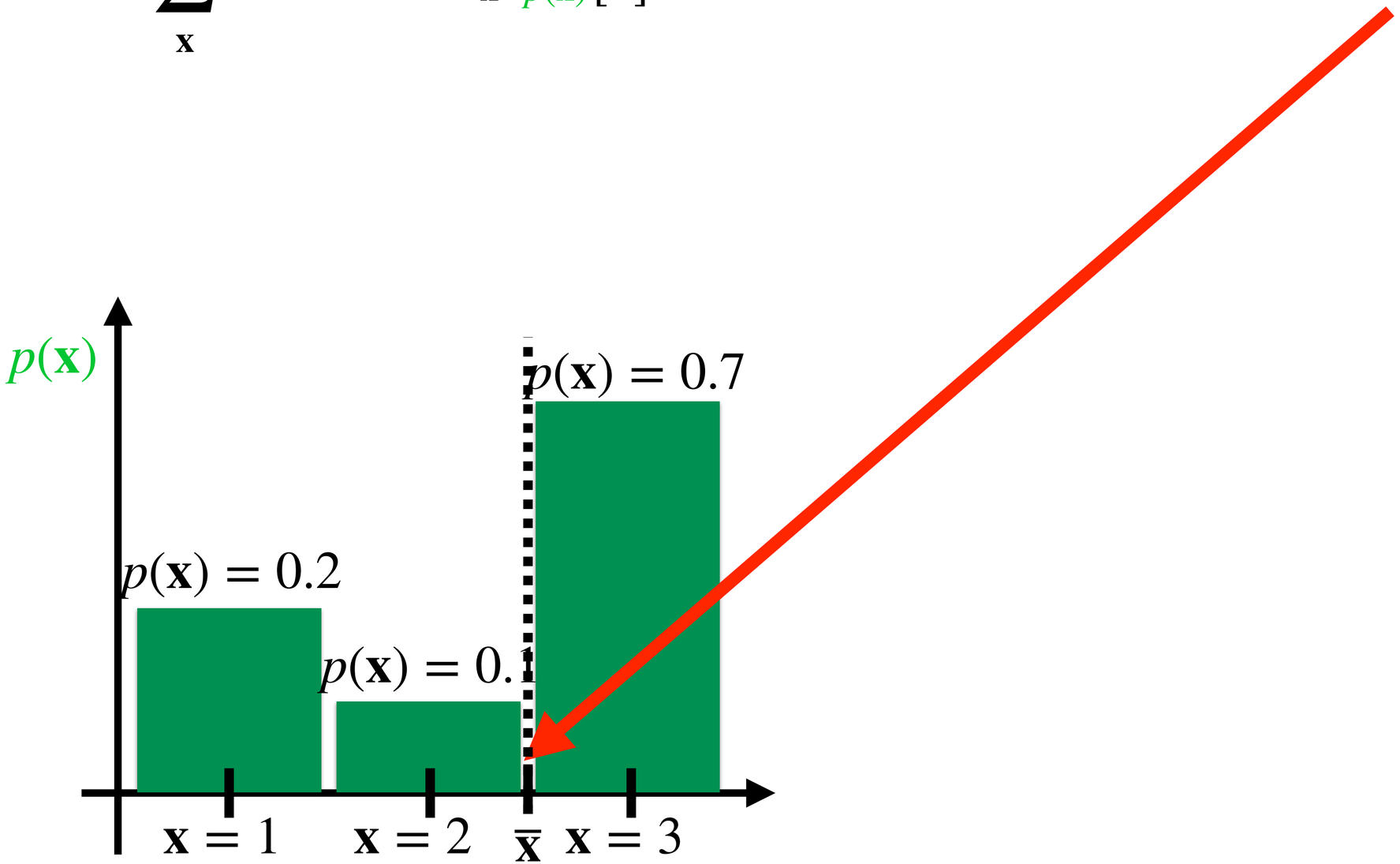
Prerequisites: Mean and average

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



Prerequisites: Mean and average

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

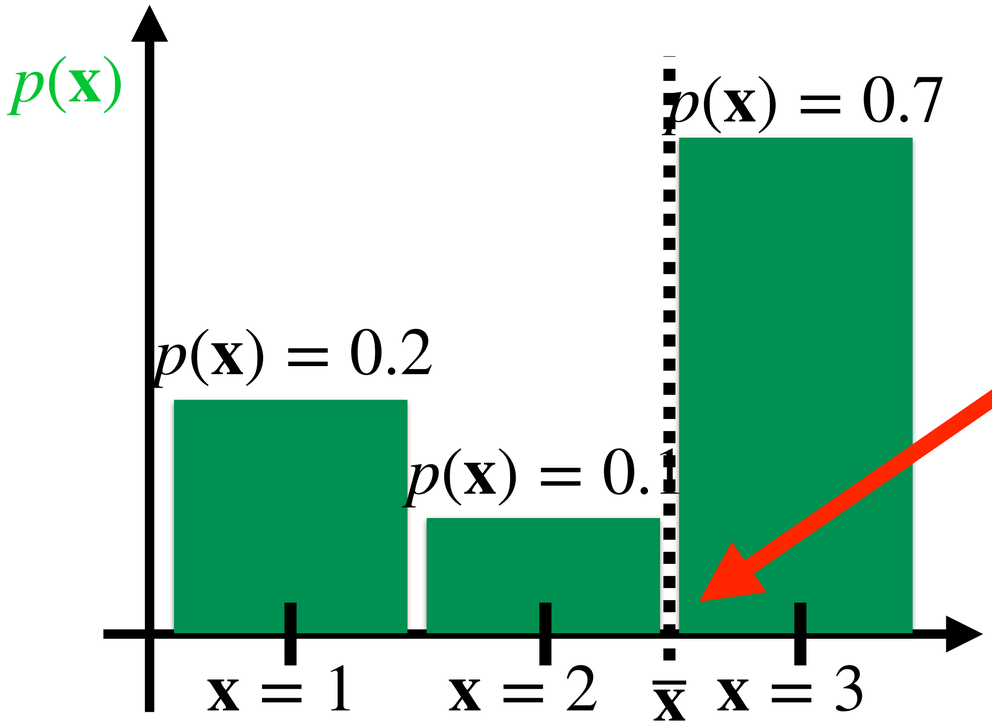


Prerequisites: Mean and average

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

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

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



Prerequisites: Mean and average

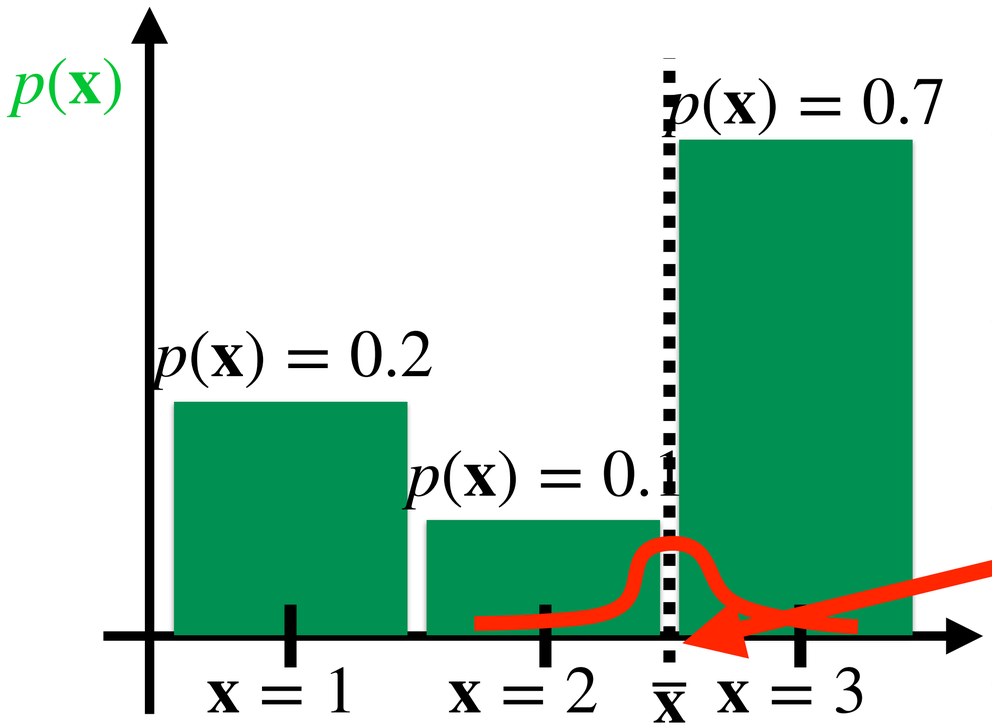
$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\mathbf{x}] = 0.2 \cdot 1 \; + \; 0.1 \cdot 2 \; + \; 0.7 \cdot 3 = 2.5$$

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

For $N \to \infty$

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

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



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

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

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

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

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

Prerequisites: Mean and average

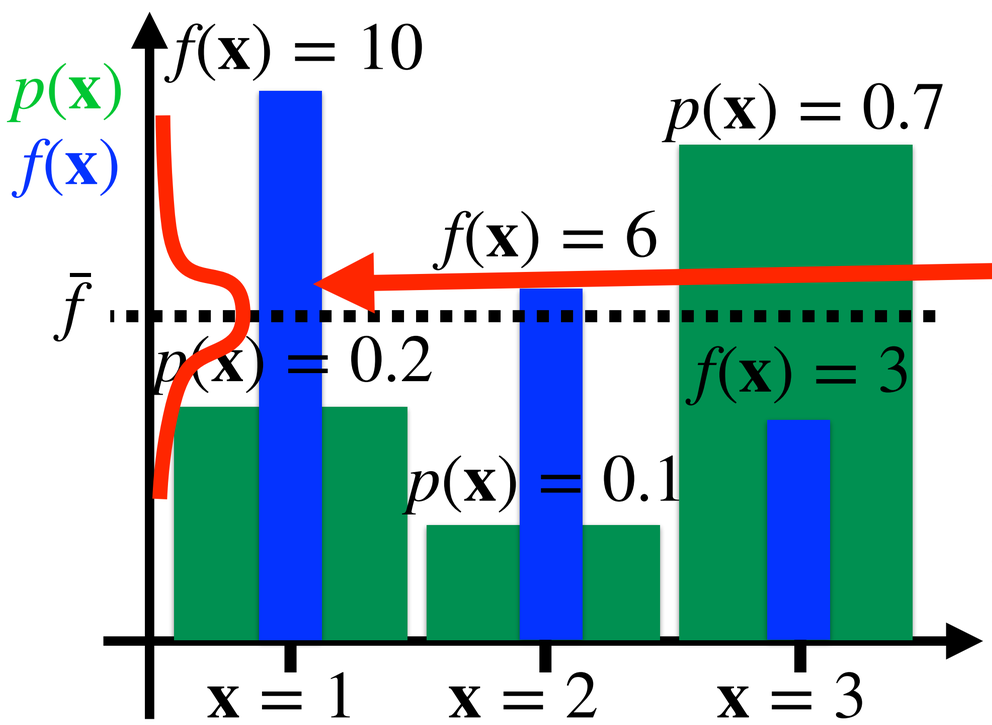
$$\bar{f} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot f(\mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[f(\mathbf{x})\right] = 0.2 \cdot 10 \;+\; 0.1 \cdot 6 \;+\; 0.7 \cdot 3 = 4.7$$

$$\approx \frac{1}{N}\sum_{i} f(\mathbf{x}_i) = \frac{1}{10}(10 + 10 + 6 + 3 + 3 + 3 + 3 + 3 + 3 + 3) = 4.7$$

For $N \to \infty$

$$\mathcal{N}(\bar{f}_i;\; \bar{f}, \frac{\sigma_f^2}{\sqrt{N}})$$

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



$$\bar{f}_1 = \frac{1}{10}(10 + 10 + 6 + 6 + 6 + 3 + 3 + 3 + 3 + 3) = 5.3$$

$$\bar{f}_2 = \frac{1}{10}(10 + 6 + 6 + 3 + 3 + 3 + 3 + 3 + 3 + 3) = 4.3$$

$$\bar{f}_3 = \frac{1}{10}(10 + 6 + 6 + 6 + 3 + 3 + 3 + 3 + 3 + 3) = 4.6$$

$$\bar{f}_4 = \frac{1}{10}(10 + 6 + 6 + 6 + 6 + 3 + 3 + 3 + 3 + 3) = 4.9$$

$$\bar{f}_5 = \frac{1}{10}(3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3) = 3.0$$

Batches

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

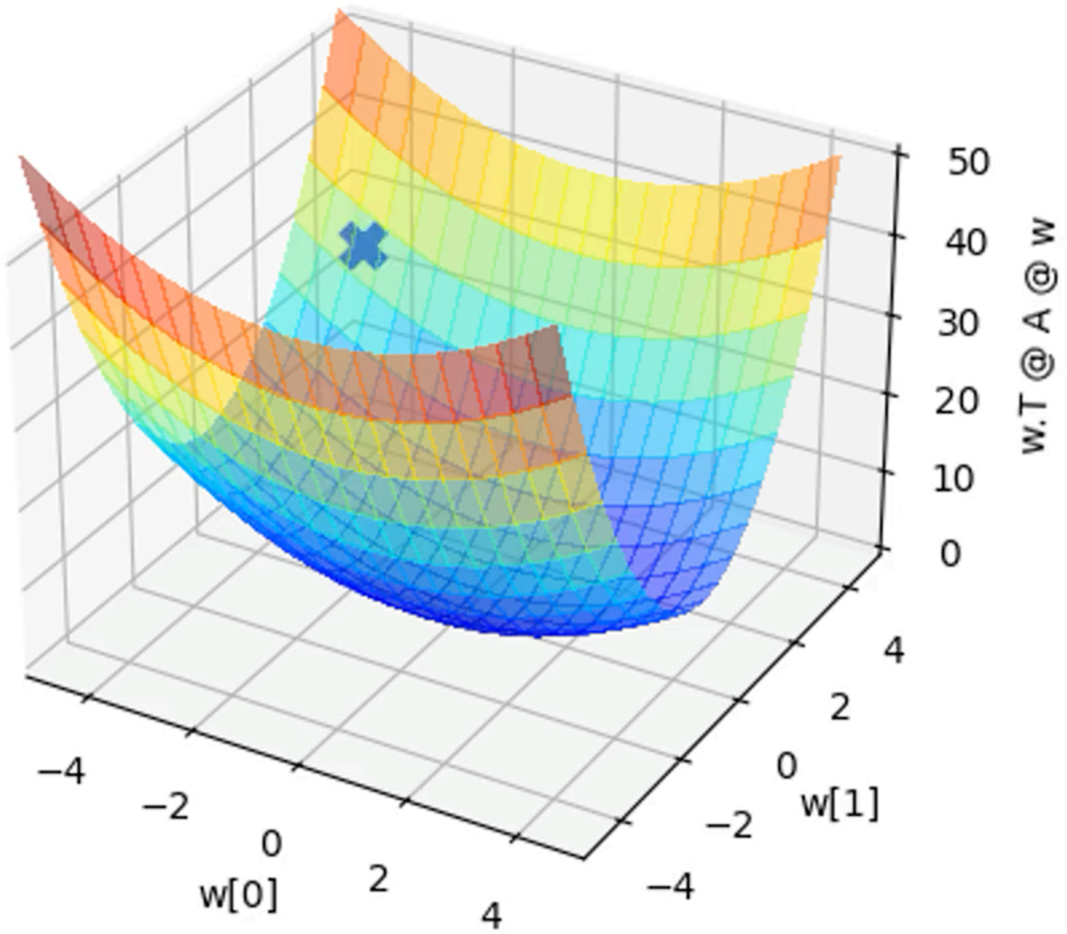
$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A} (\mathbf{w} - \mathbf{w}_i)$$

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

Batches

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

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

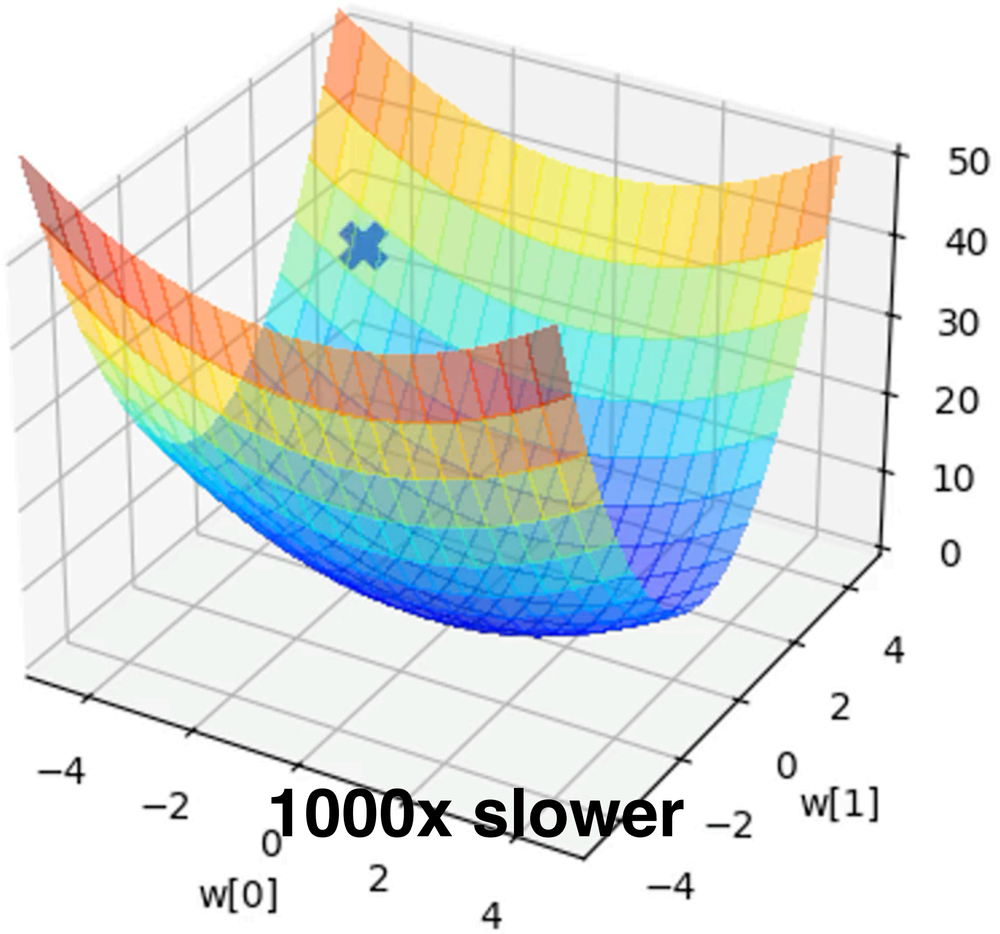


Batches

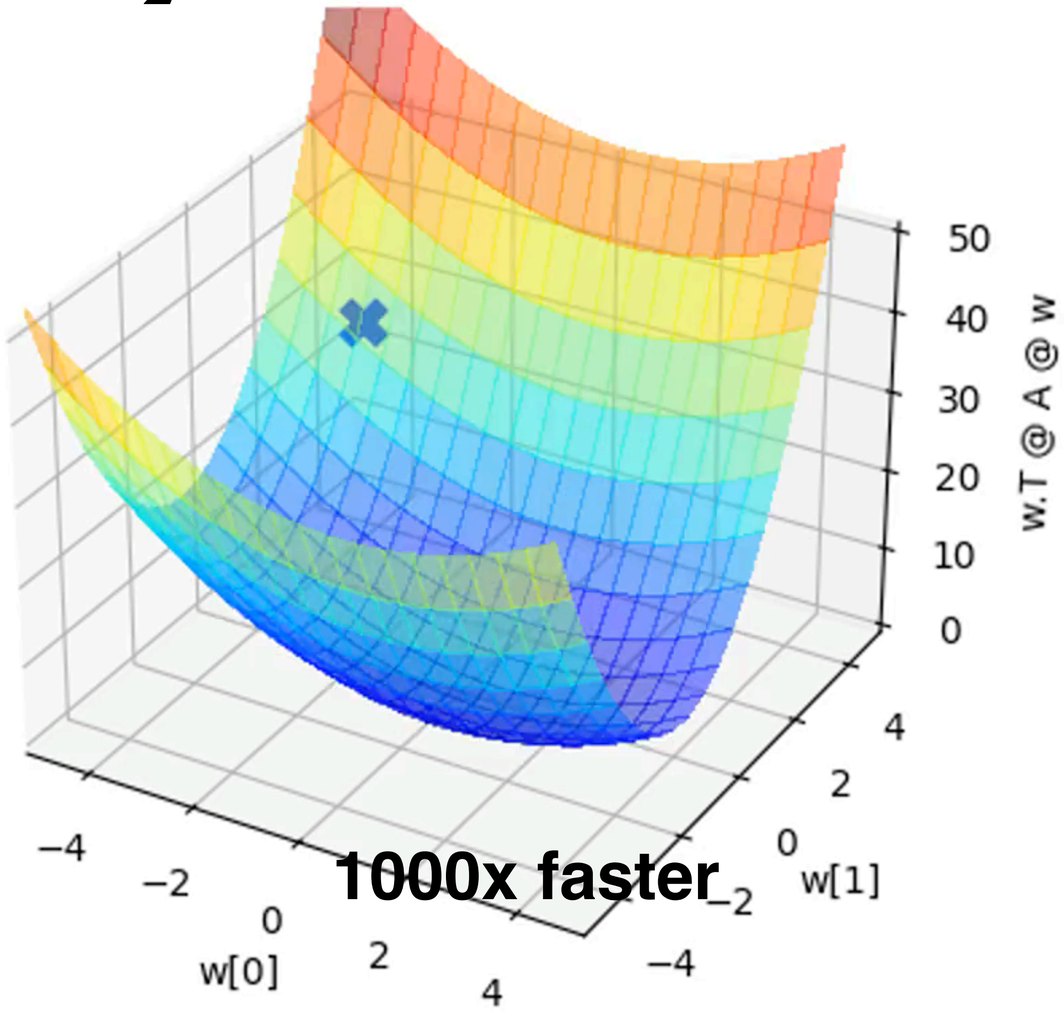
$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A} (\mathbf{w} - \mathbf{w}_i)$$

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

$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A}, \quad i = \text{rand}(1,1000)$$



1000x slower



1000x faster

Batches

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} D_{KL}\big(p_{\text{data}}(\mathbf{x}, y) \parallel p(\mathbf{x}, y \mid \mathbf{w})\big) = \arg\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\big[-\log p(y \mid \mathbf{x}, \mathbf{w})\big]$$

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

True gradient
(we do not have acces to)

Full trn set gradient
(time-consuming estimation)

Batches

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} D_{KL}\big(p_{\text{data}}(\mathbf{x}, y) \parallel p(\mathbf{x}, y \mid \mathbf{w})\big) = \arg\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\big[-\log p(y \mid \mathbf{x}, \mathbf{w})\big]$$

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

True gradient
(we do not have acces to)

Full trn set gradient
(time-consuming estimation)

Batch gradient
(N<<M)

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

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

- 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
  - computations is $100\times$ slower !!!
- 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 ?

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

- Large N => more accurate gradient with sub-linear returns.
- Runtime in multicore architectures is similar for small $N = 1,2,...$
- 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: $N \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$ or anything else that works ;-)

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

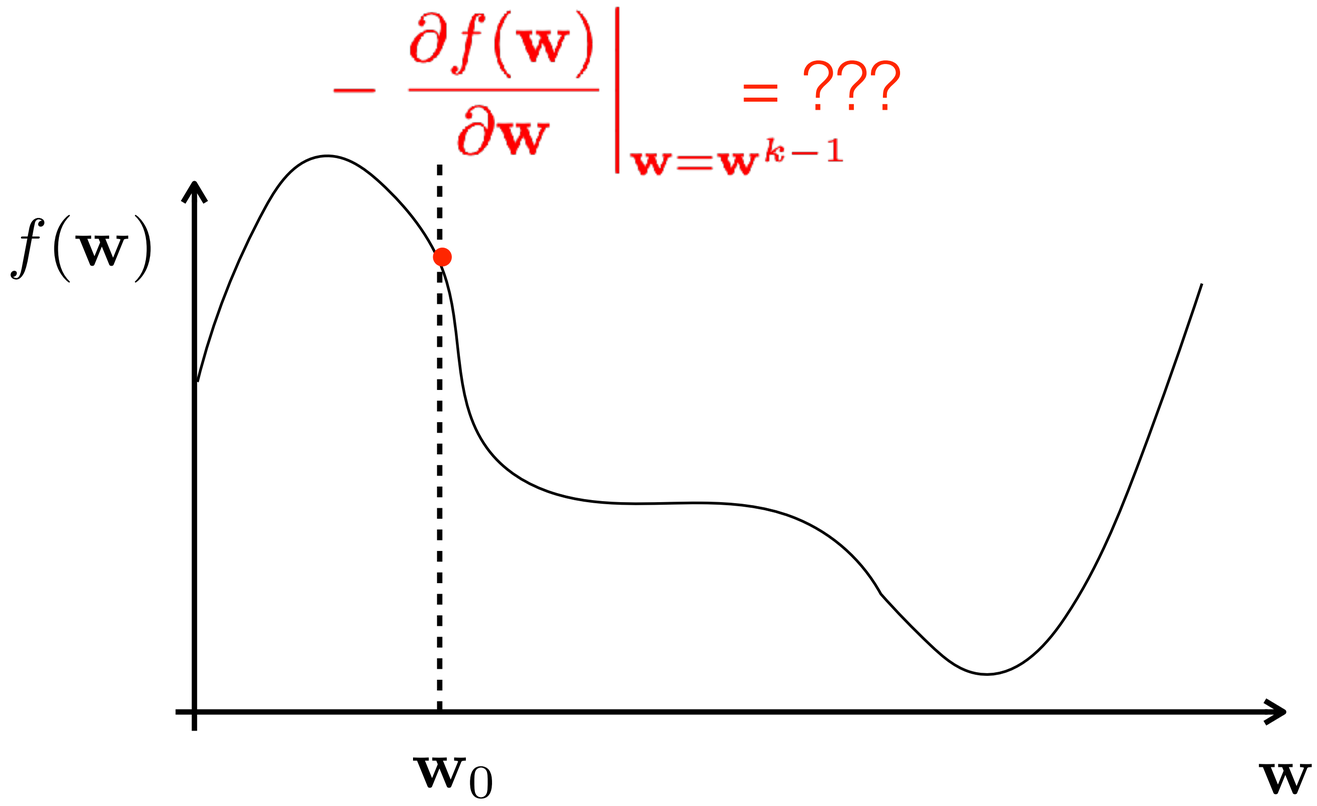
$1 < N <$ trn_size..... often called minibatch learning

Stochastic Gradient Descent (SGD) = GD over minibatches

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

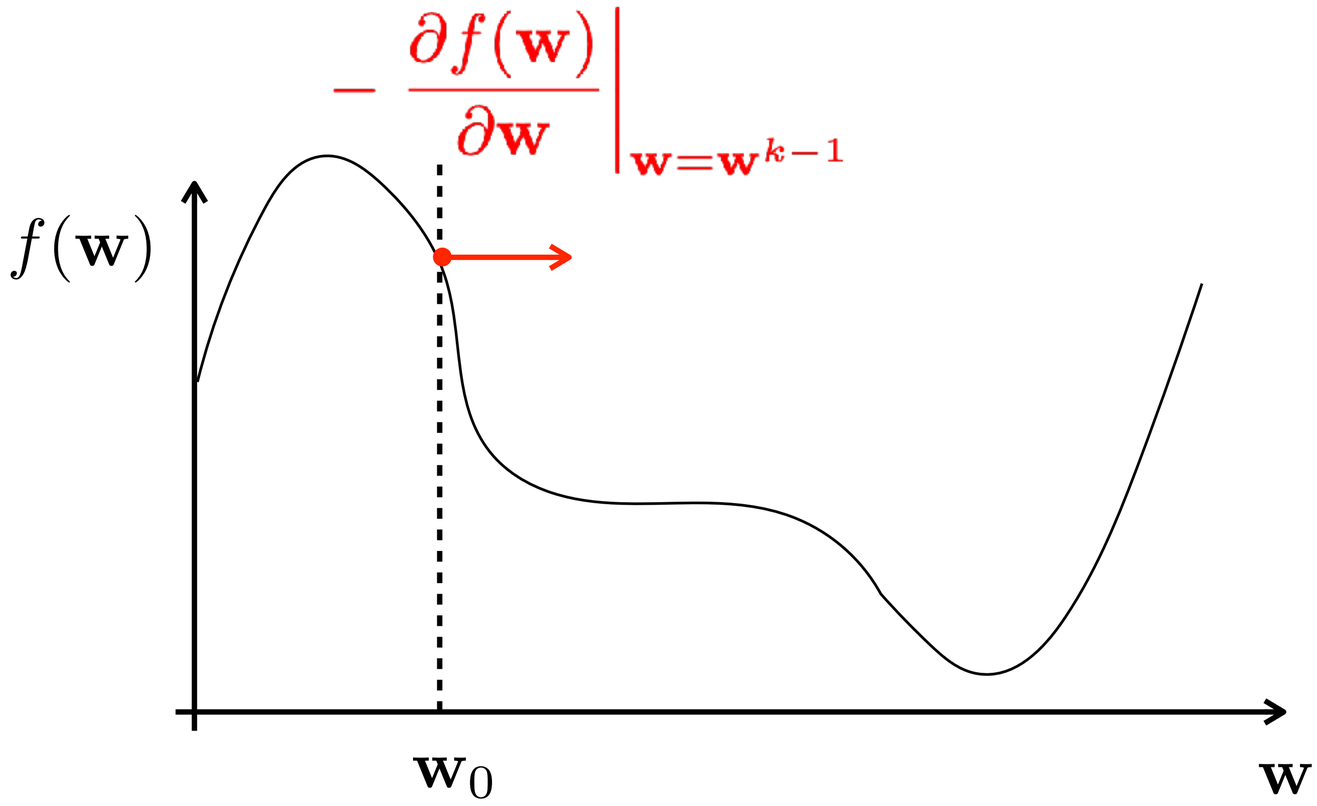
Stochastic Gradient Descent (SGD) = GD over minibatches

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



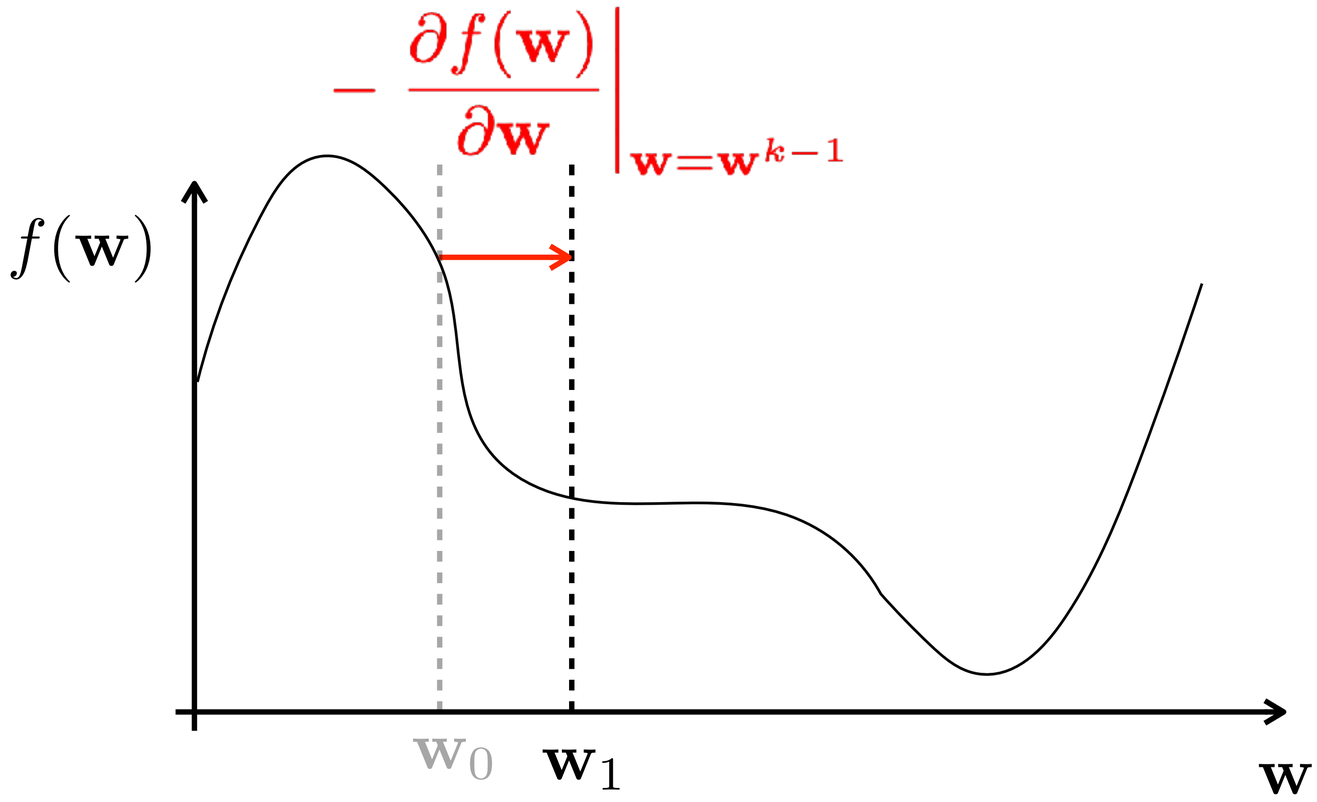
SGD drawbacks: can get stuck on flat regions

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



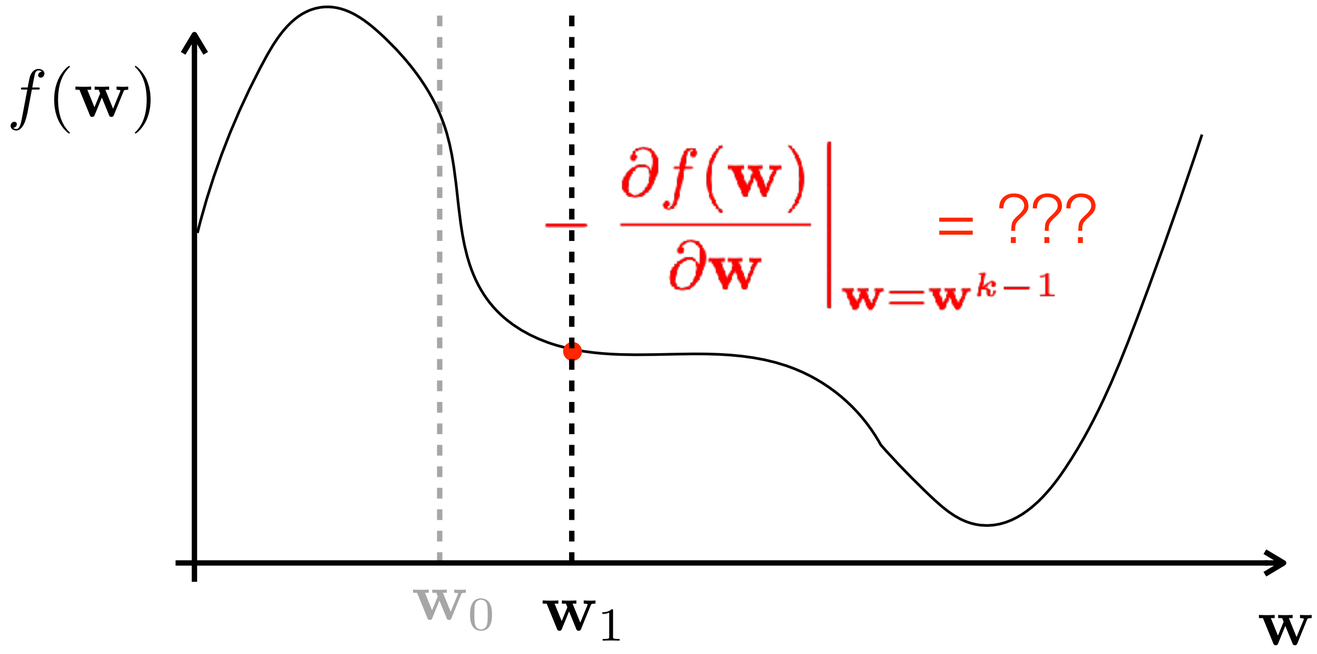
SGD drawbacks: can get stuck on flat regions

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



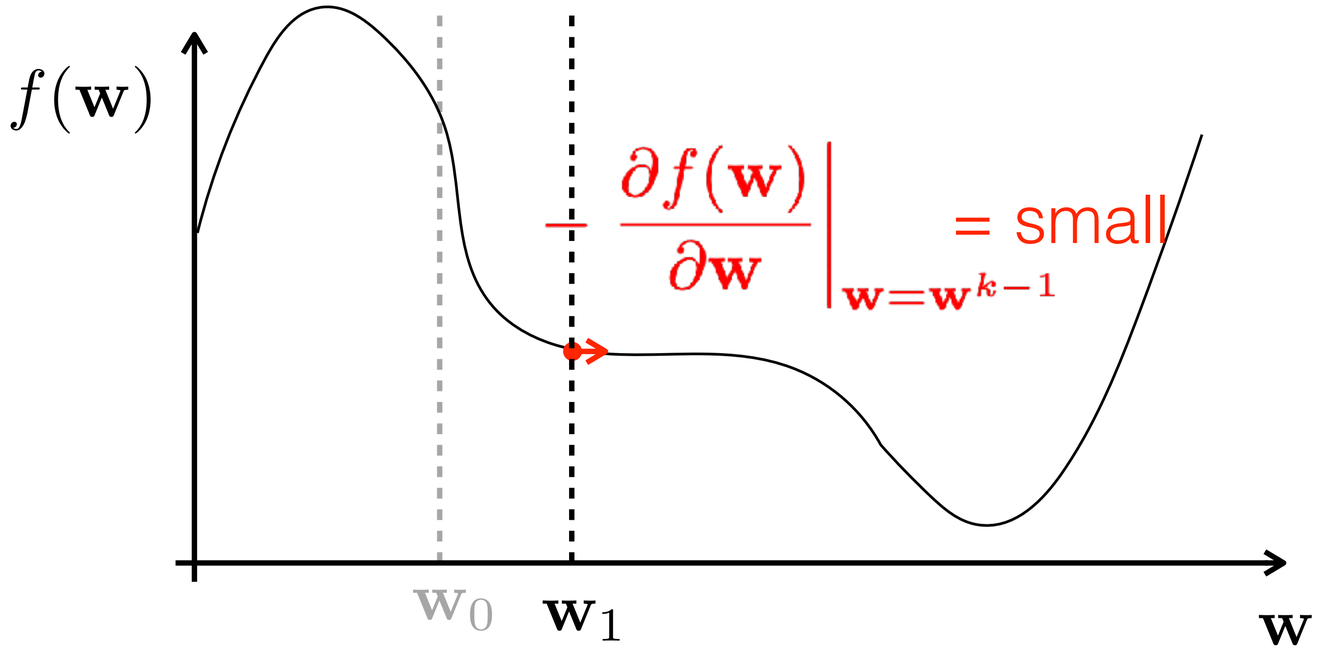
SGD drawbacks: can get stuck on flat regions

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



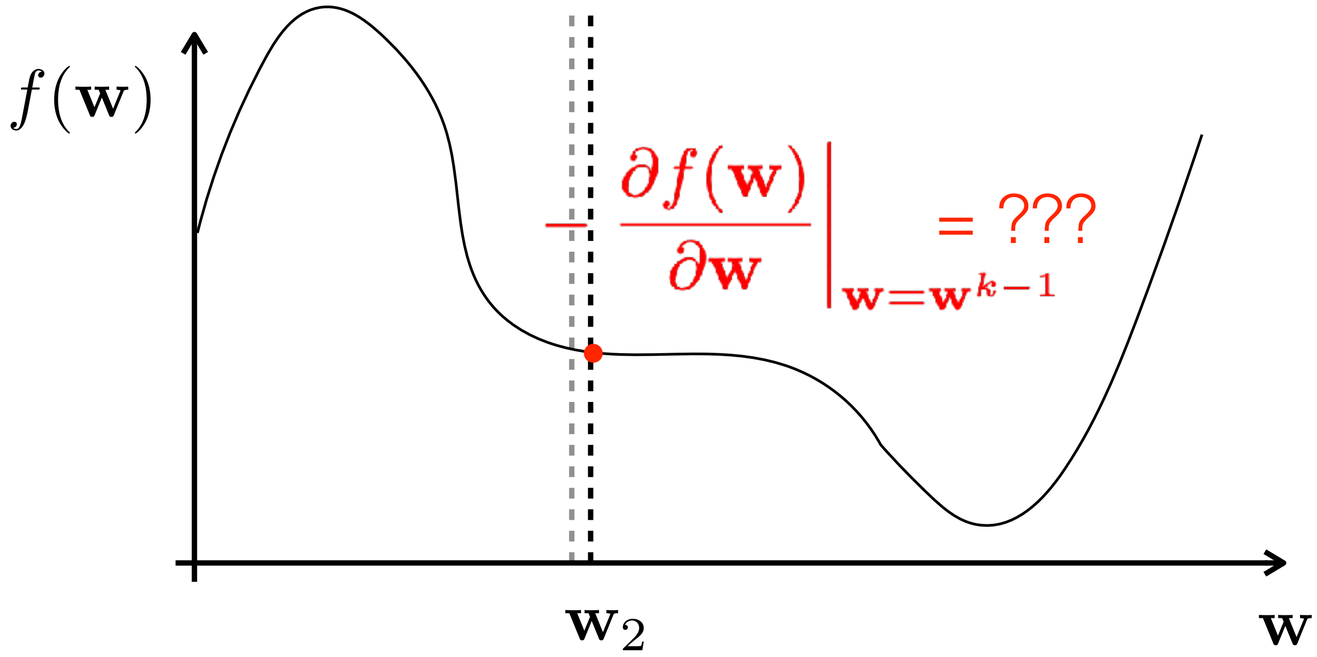
SGD drawbacks: can get stuck on flat regions

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



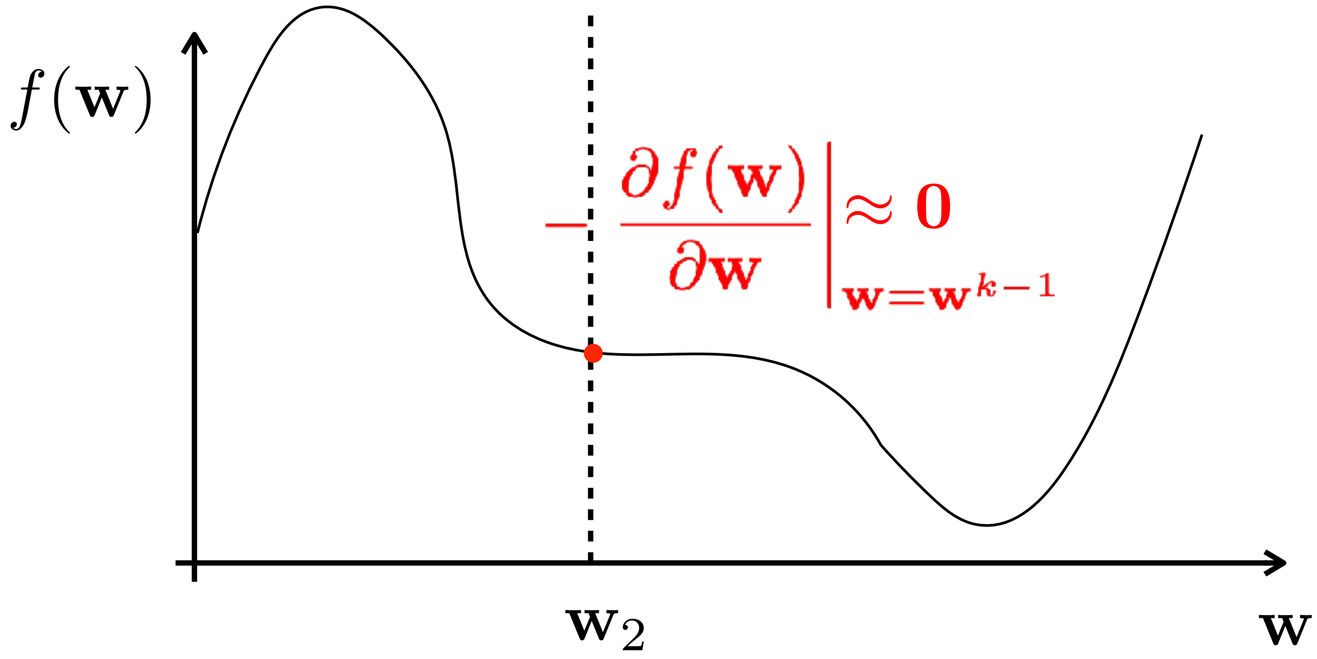
SGD drawbacks: can get stuck on flat regions

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

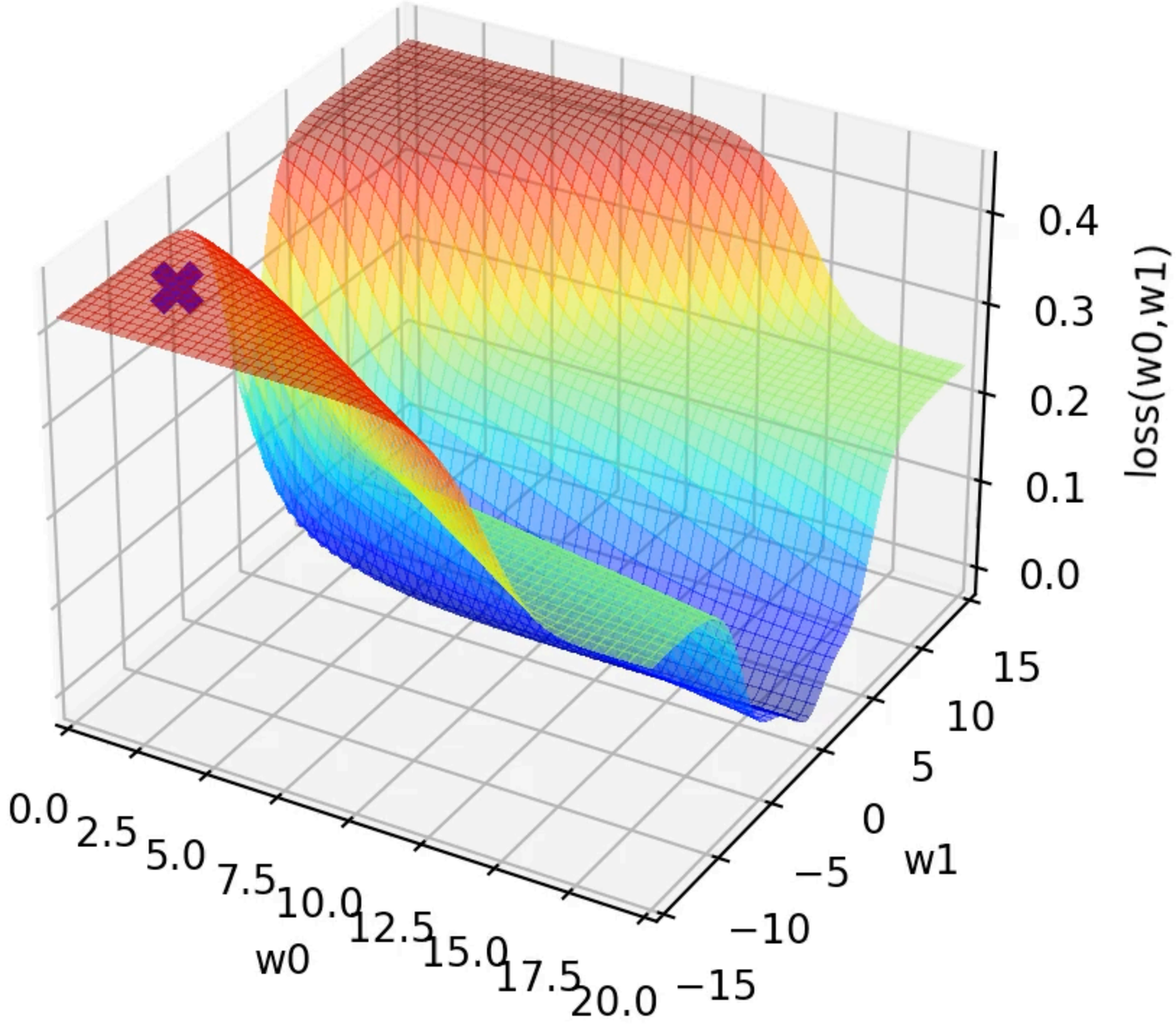


SGD drawbacks: can get stuck on flat regions

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



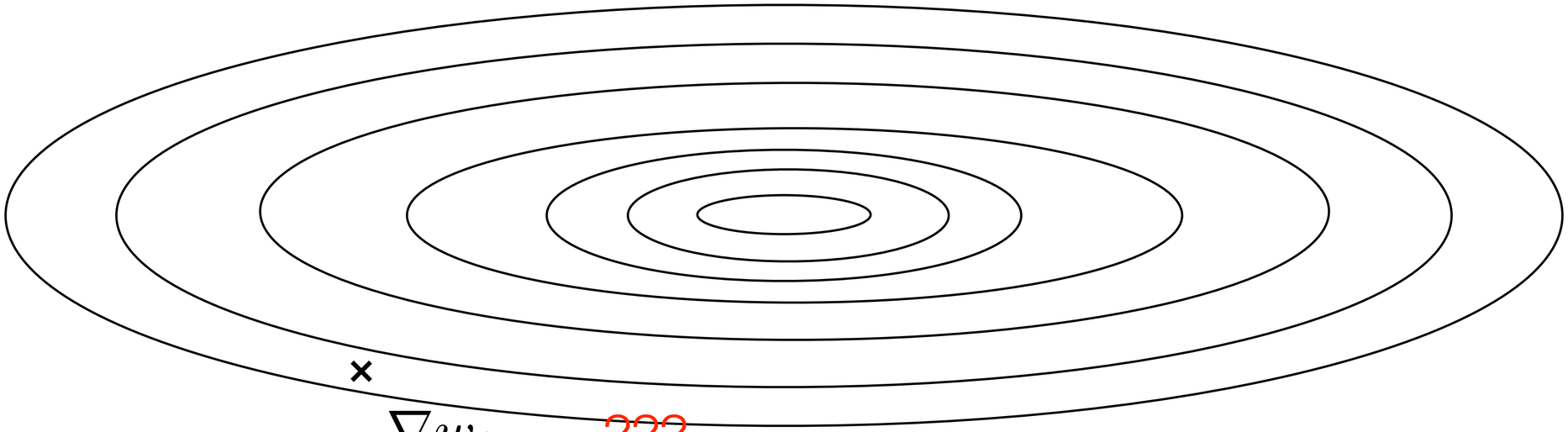
SGD drawbacks: Sigmoid fitting problem from labs



close-to-zero gradient plateaus

SGD drawbacks: oscillates

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



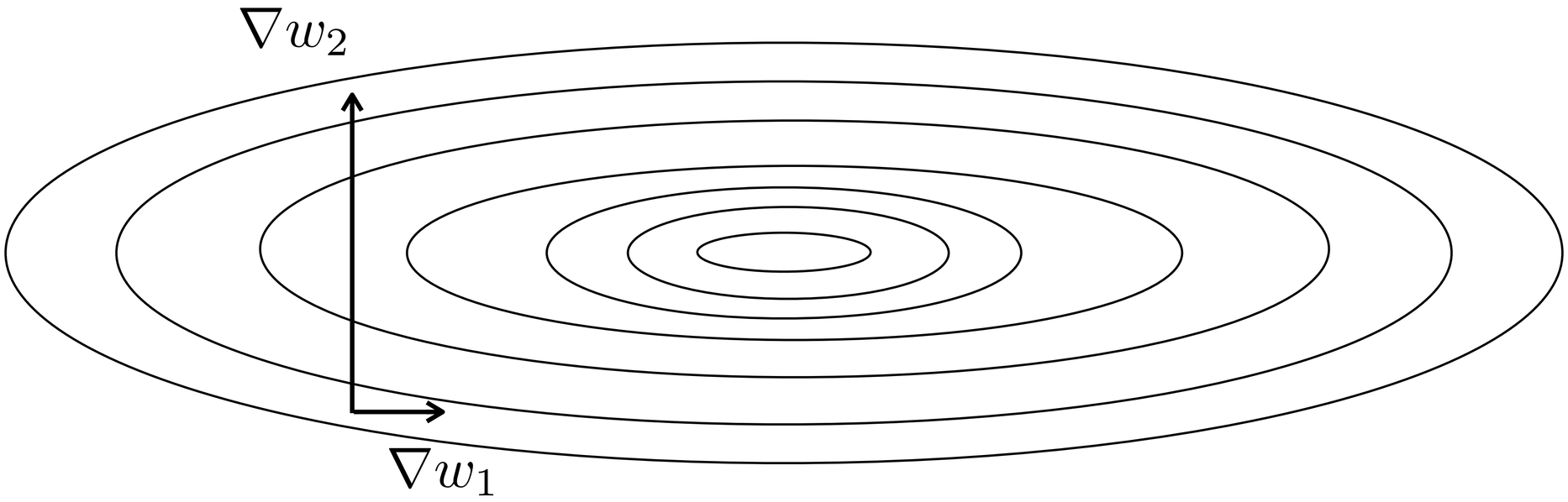
∇w_1 =???

∇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

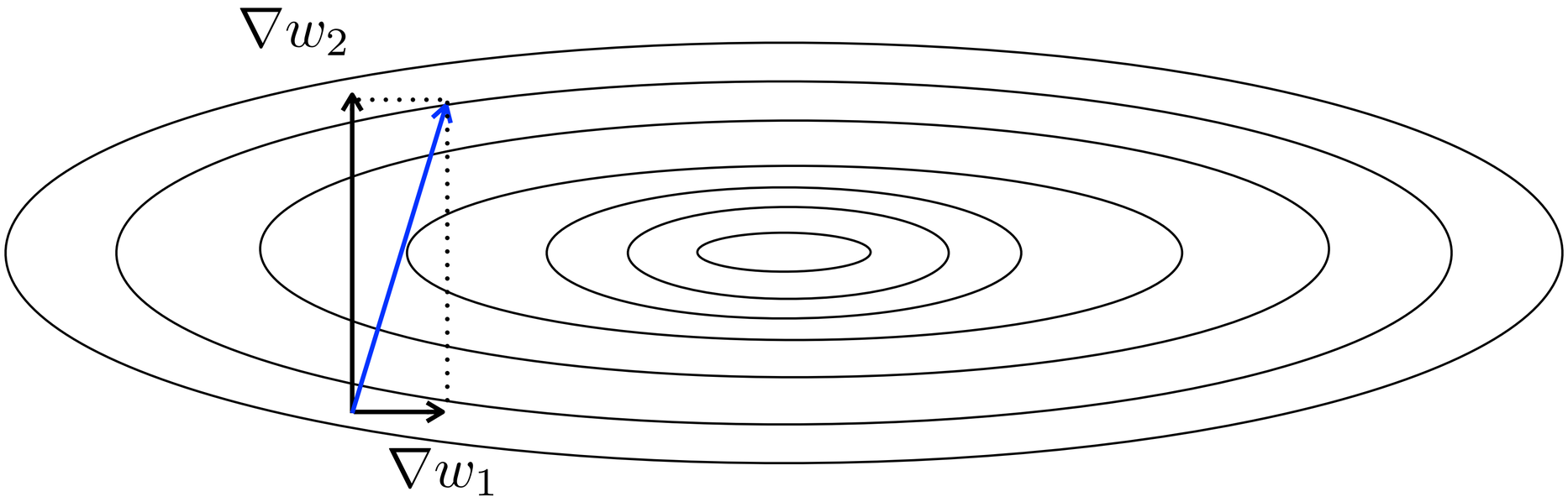
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD drawbacks: oscillates

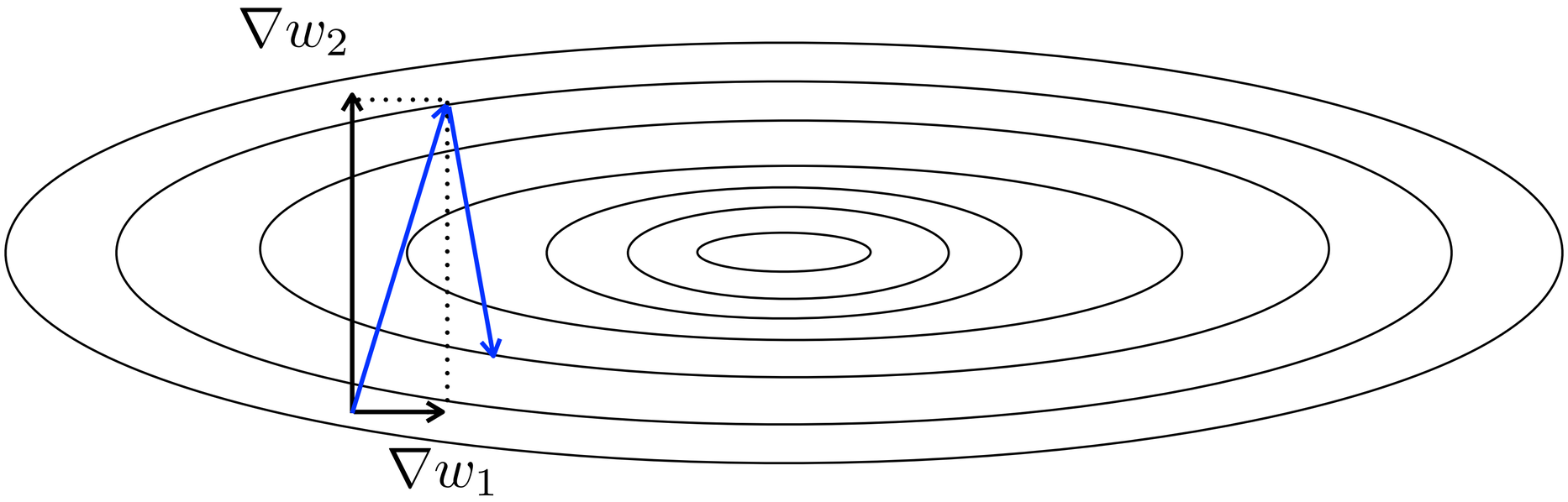
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD drawbacks: oscillates

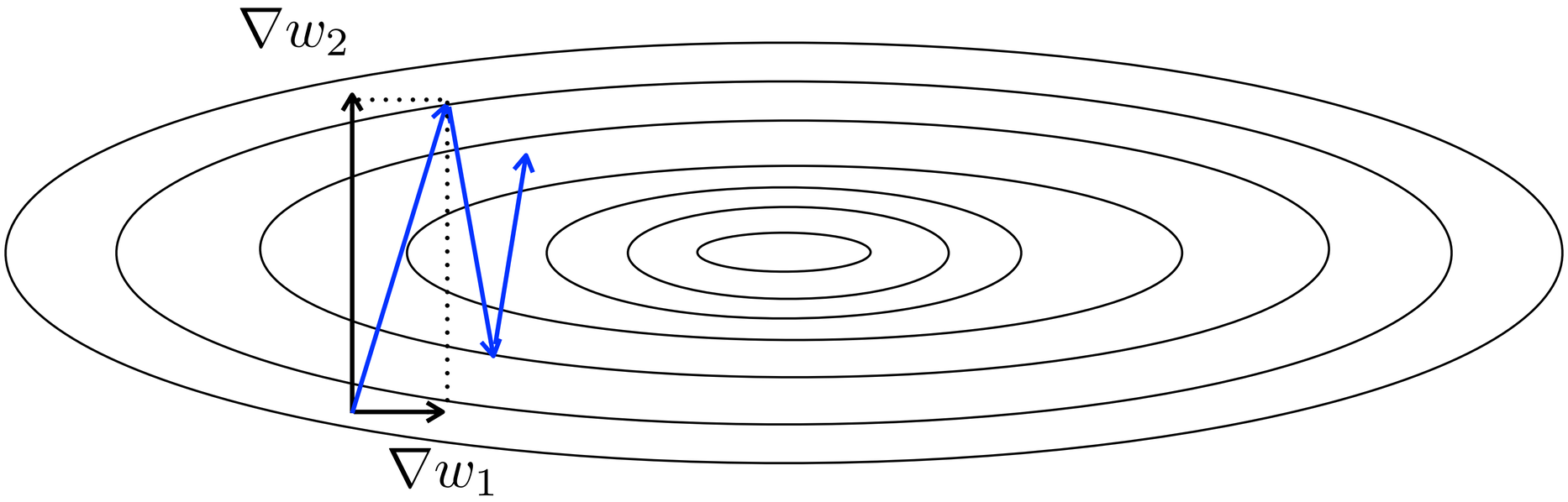
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD drawbacks: oscillates

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

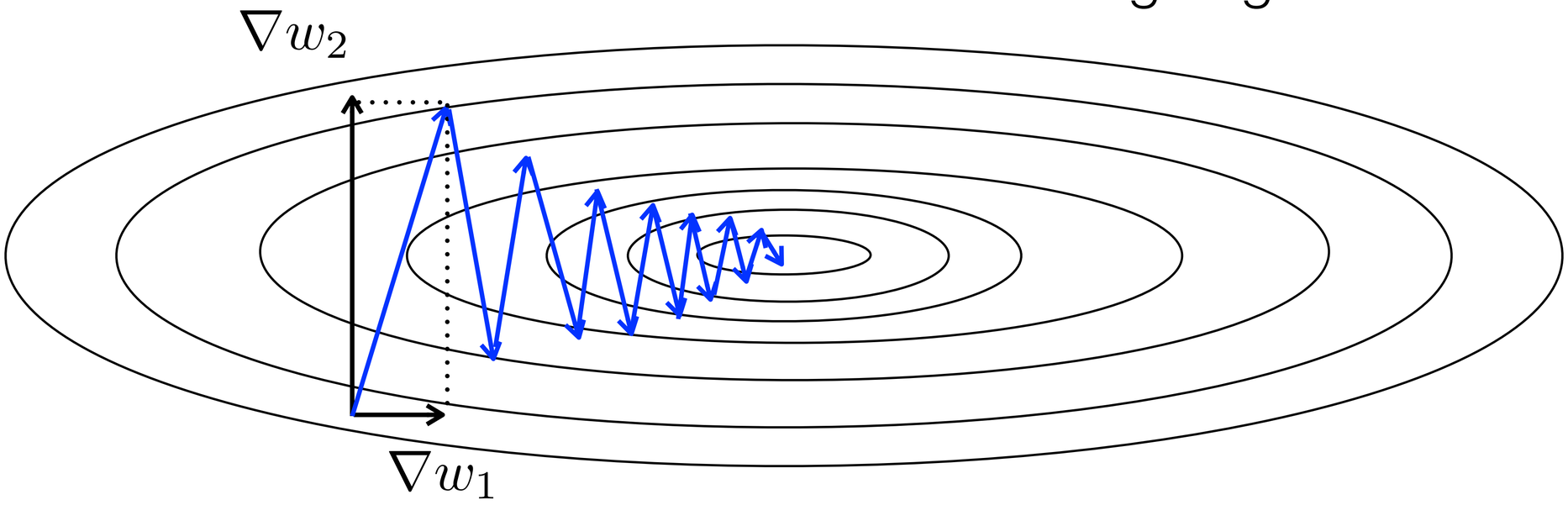


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: oscillates

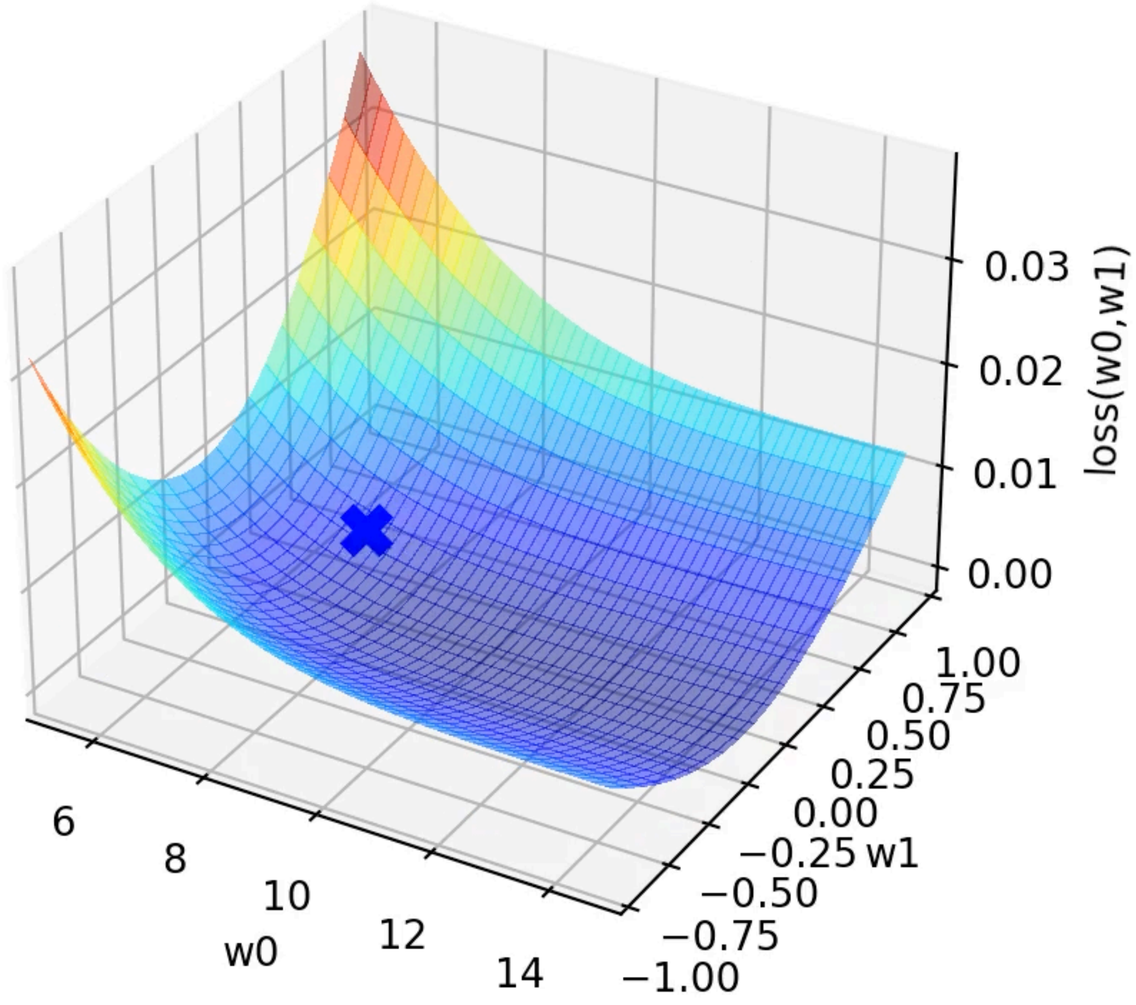
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Undesired zig-zag behaviour

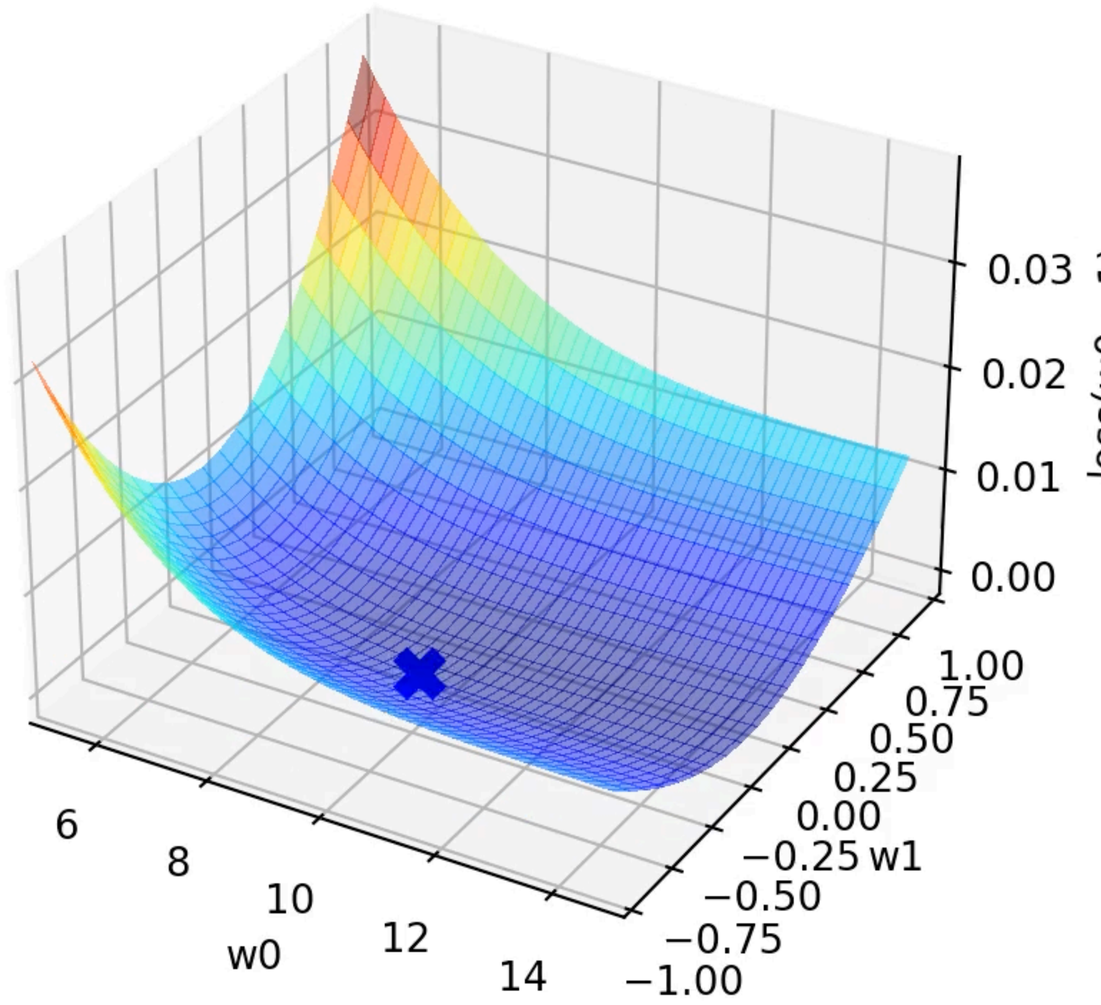


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks: Sigmoid fitting problem from labs



small learning rate

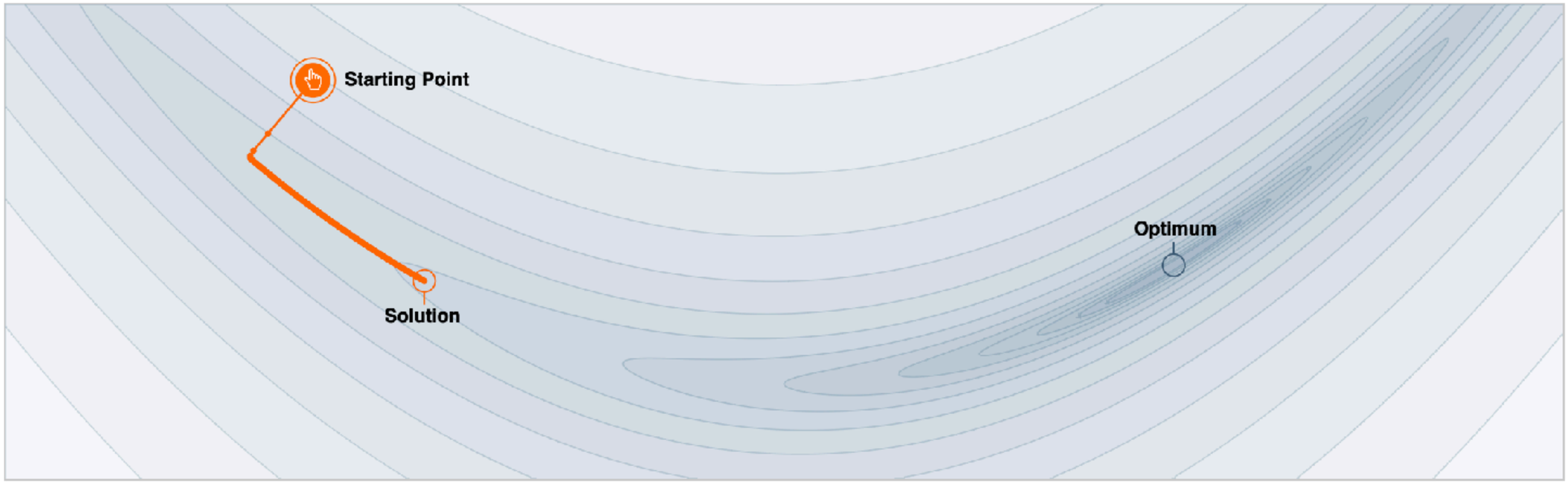


big learning rate

SGD drawbacks: oscillates

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

$\alpha = 1e\text{-}3$

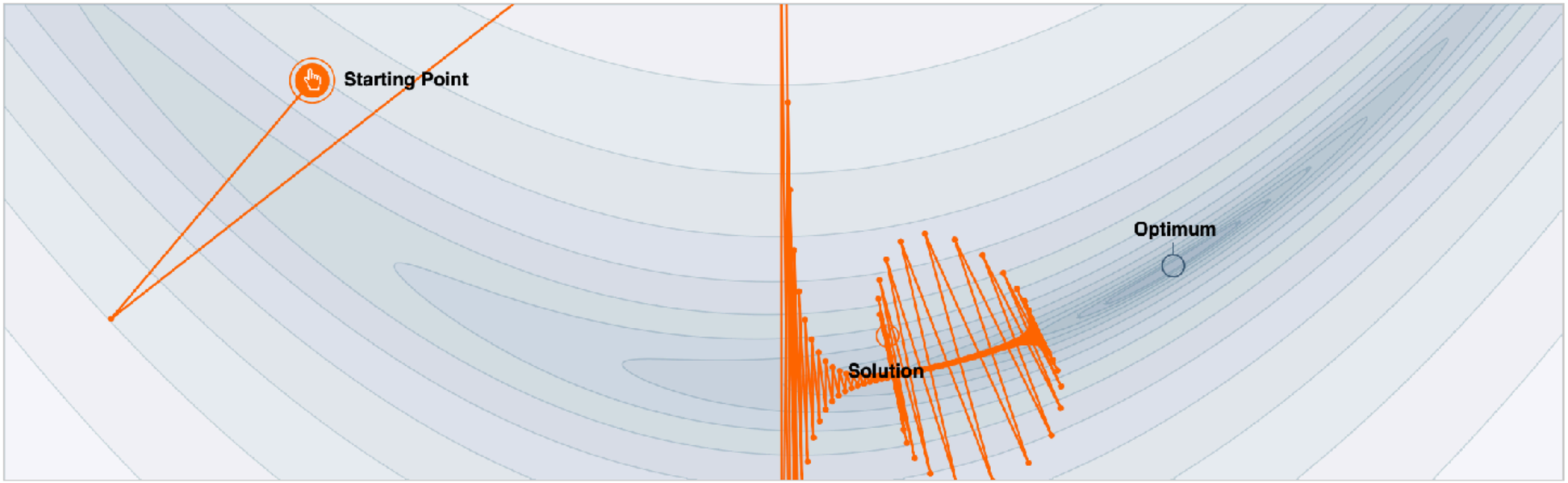


https://distill.pub/2017/momentum/

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/

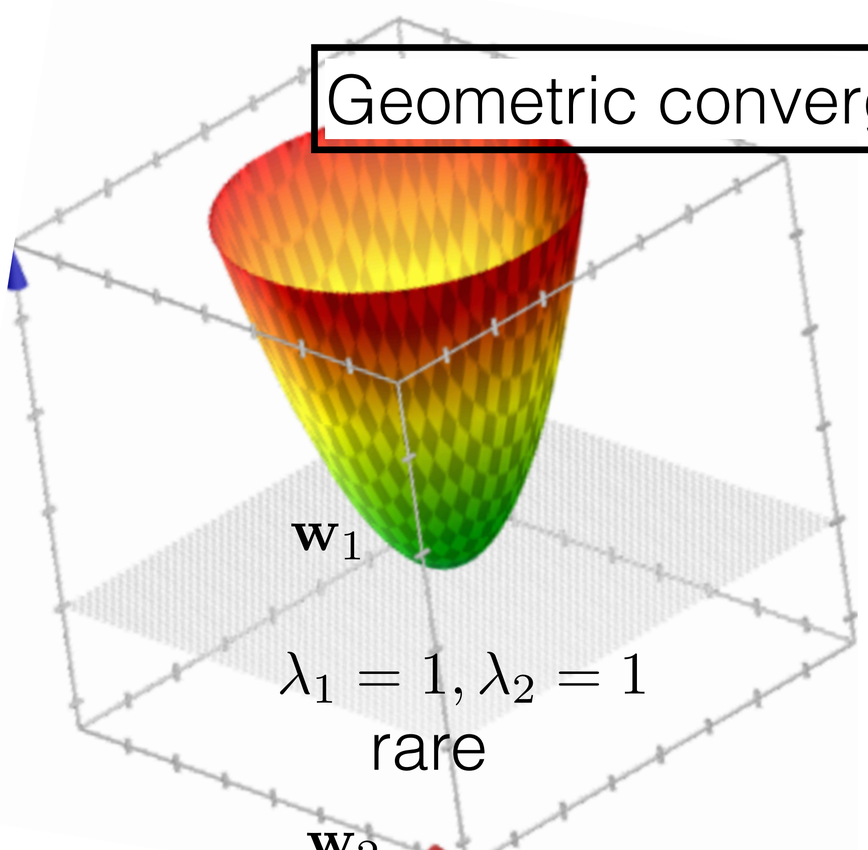
SGD on quadric

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

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

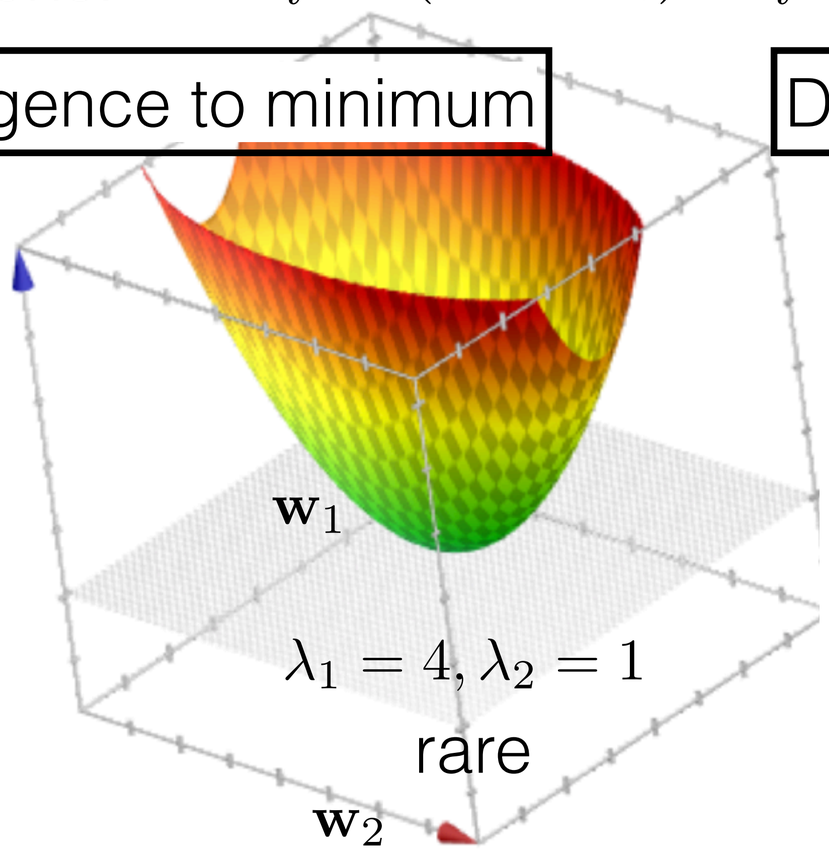
SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

Geometric convergence to minimum



$\lambda_1 = 1, \lambda_2 = 1$

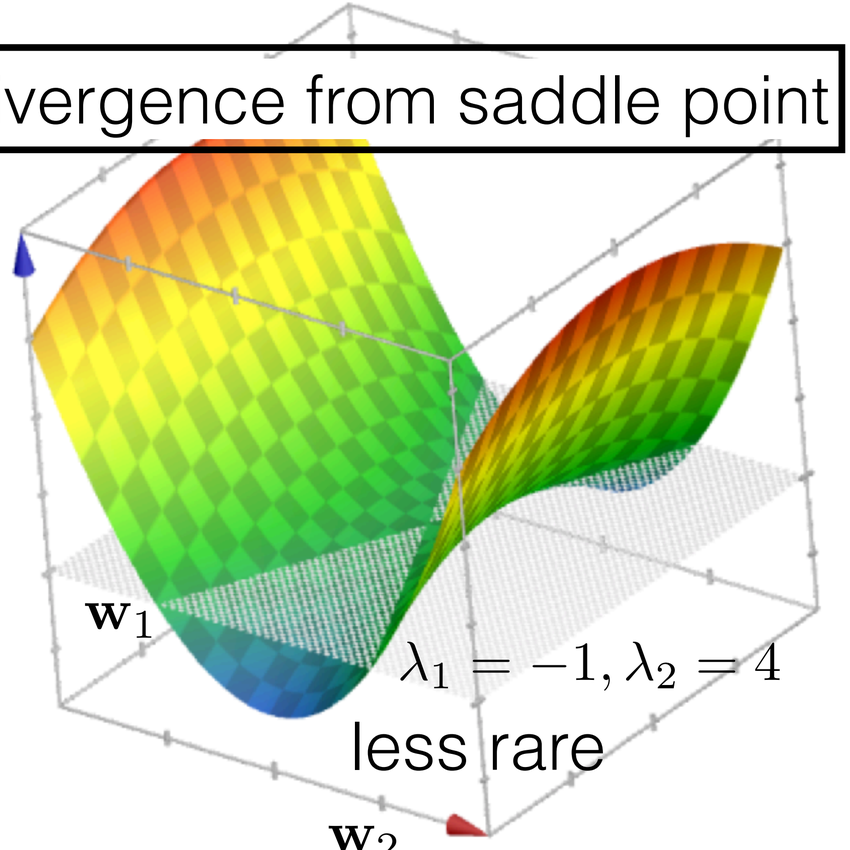
rare



$\lambda_1 = 4, \lambda_2 = 1$

rare

Divergence from saddle point



$\lambda_1 = -1, \lambda_2 = 4$

less rare

SGD on quadric

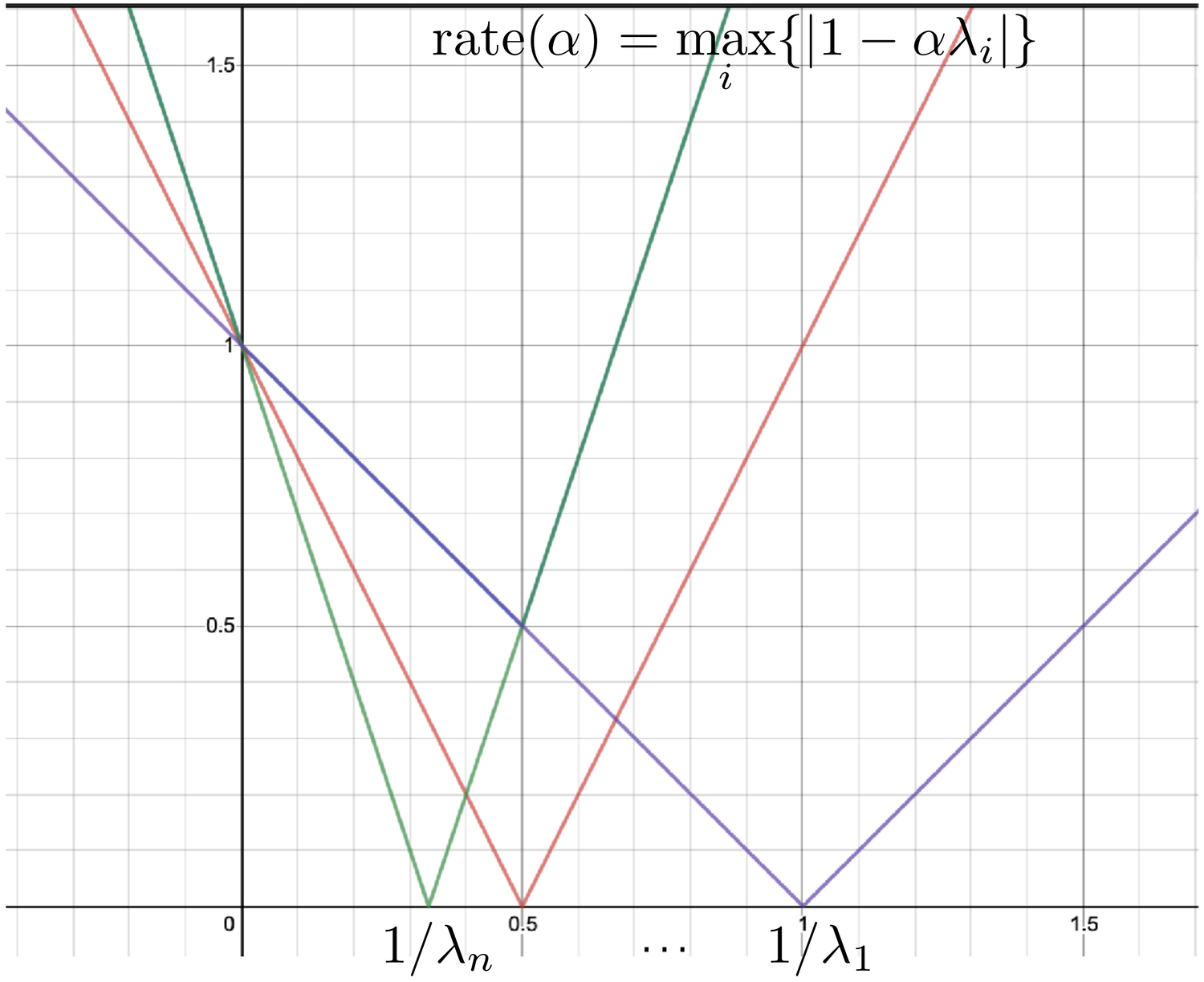
Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

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

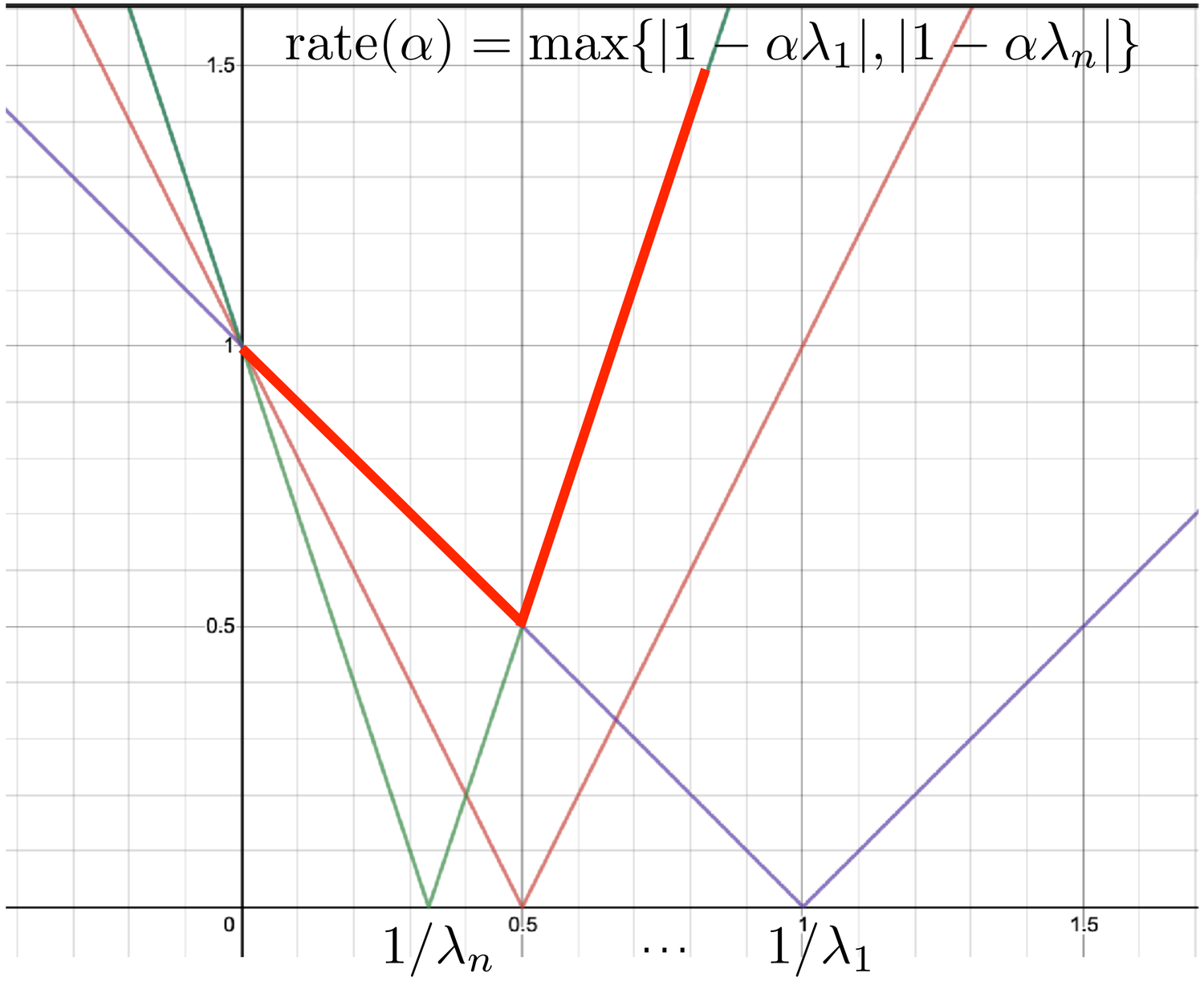
SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

- Converges for: $|1 - \alpha\lambda_i| < 1, \ \forall i$
- 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: $|1 - \alpha\lambda_i| < 1, \ \forall i$
- 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: $|1 - \alpha\lambda_i| < 1, \ \forall i$
- 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 $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

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: $|1 - \alpha\lambda_i| < 1, \ \forall i$
- with convergence rate: $\mathrm{rate}(\alpha) = \max\{|1 - \alpha\lambda_1|, |1 - \alpha\lambda_n|\}$

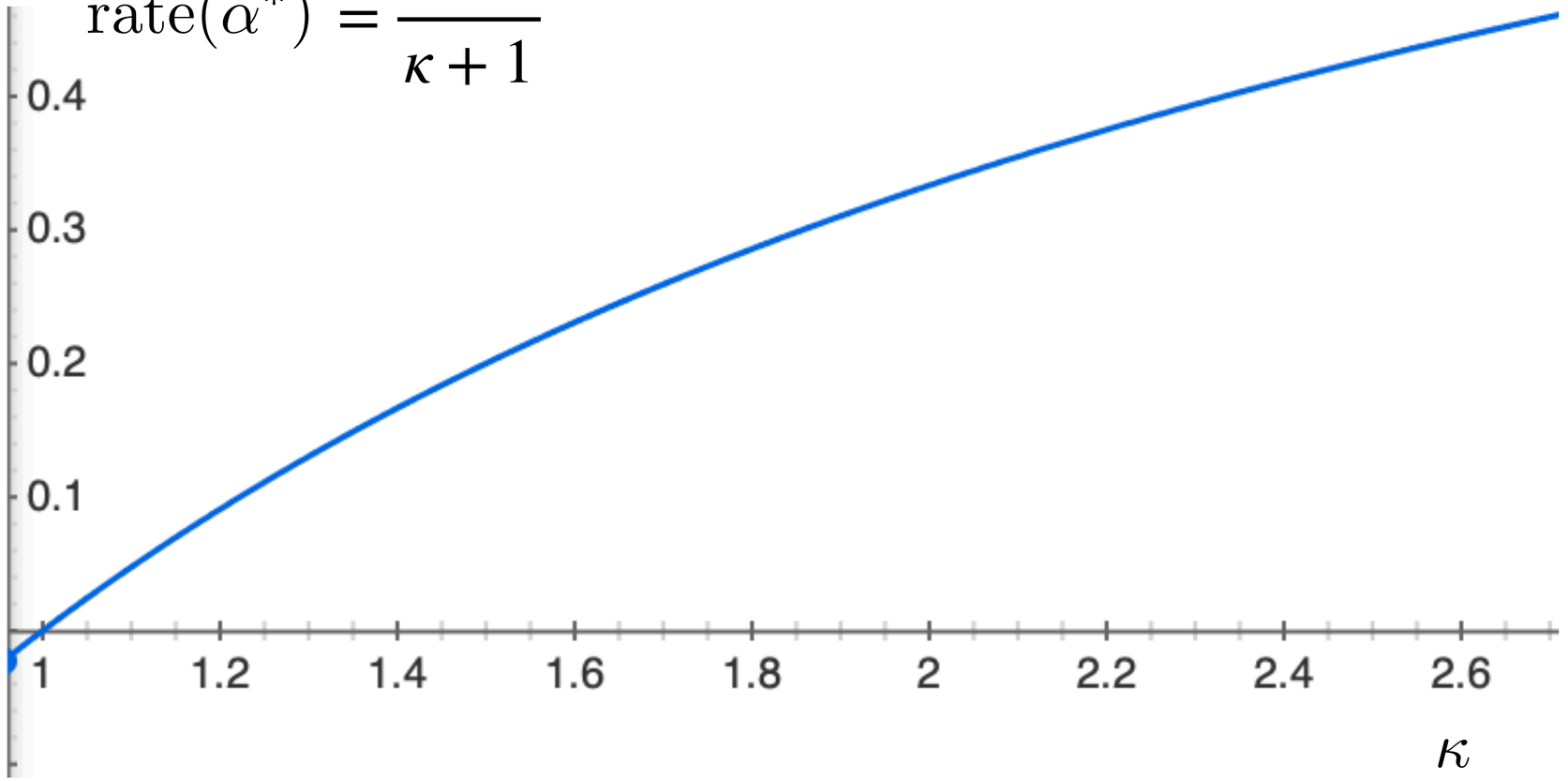
- Optimal learning rate: $\alpha^* = \arg\min_\alpha \big(\mathrm{rate}(\alpha)\big) = \frac{2}{\lambda_1 + \lambda_n}$

- Optimal conv. rate: $\mathrm{rate}(\alpha^*) = \min_\alpha \big(\mathrm{rate}(\alpha)\big) = \frac{\frac{\lambda_n}{\lambda_1} - 1}{\frac{\lambda_n}{\lambda_1} + 1} = \frac{\kappa - 1}{\kappa + 1}$

where $\kappa = \frac{\lambda_n}{\lambda_1}$ is condition number

SGD on quadric

$$\text{rate}(\alpha^*) = \frac{\kappa - 1}{\kappa + 1}$$



Summary SGD

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

Advantages?

- SGD is **faster** in term of computation time than GD
- SGD **does not get stuck in saddle-points** as easy as GD
- SGD yields **better generalization** due to inherent noise (similar to BN).

Drawbacks?

- SGD **noisier** than GD (especially for small mini-batches)
- SGD and GD **get stuck** on **flat** regions
- SGD and GD **oscillates** for $\kappa > 1$ with $\mathrm{rate}(\alpha^*) = \frac{\kappa - 1}{\kappa + 1}$

SGD + momentum

friction

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

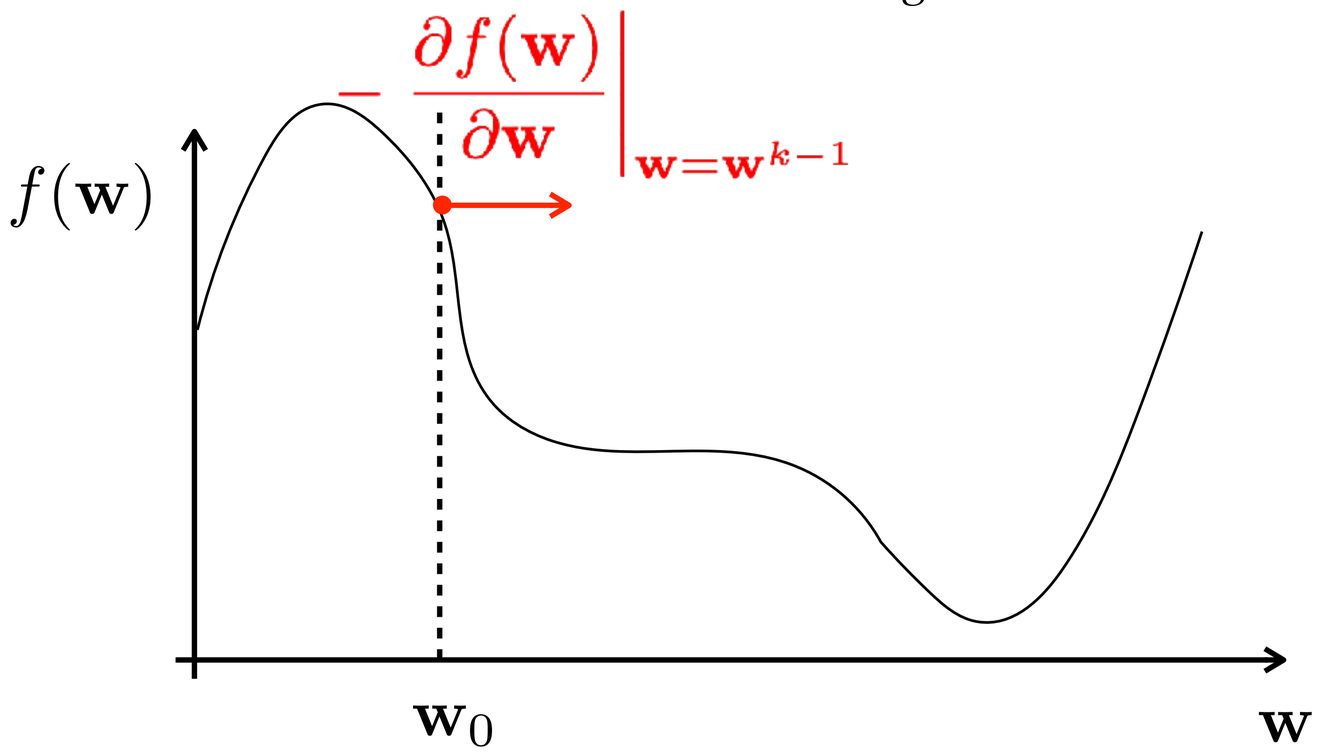
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

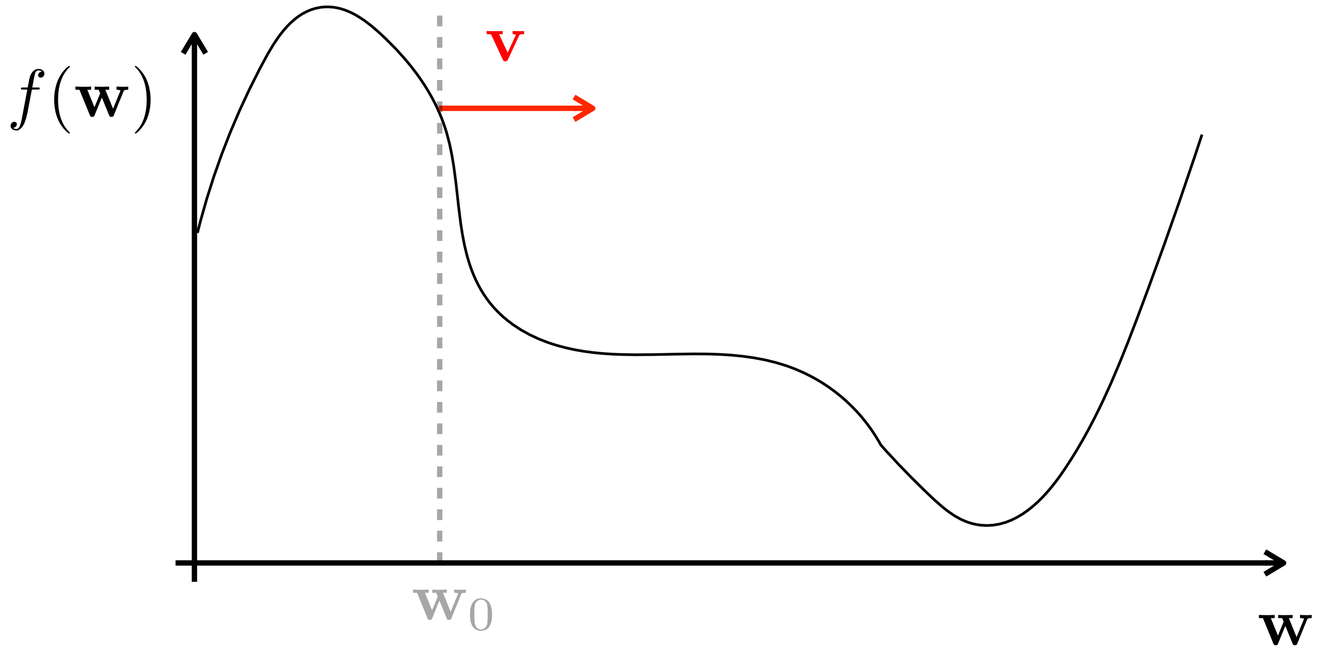
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

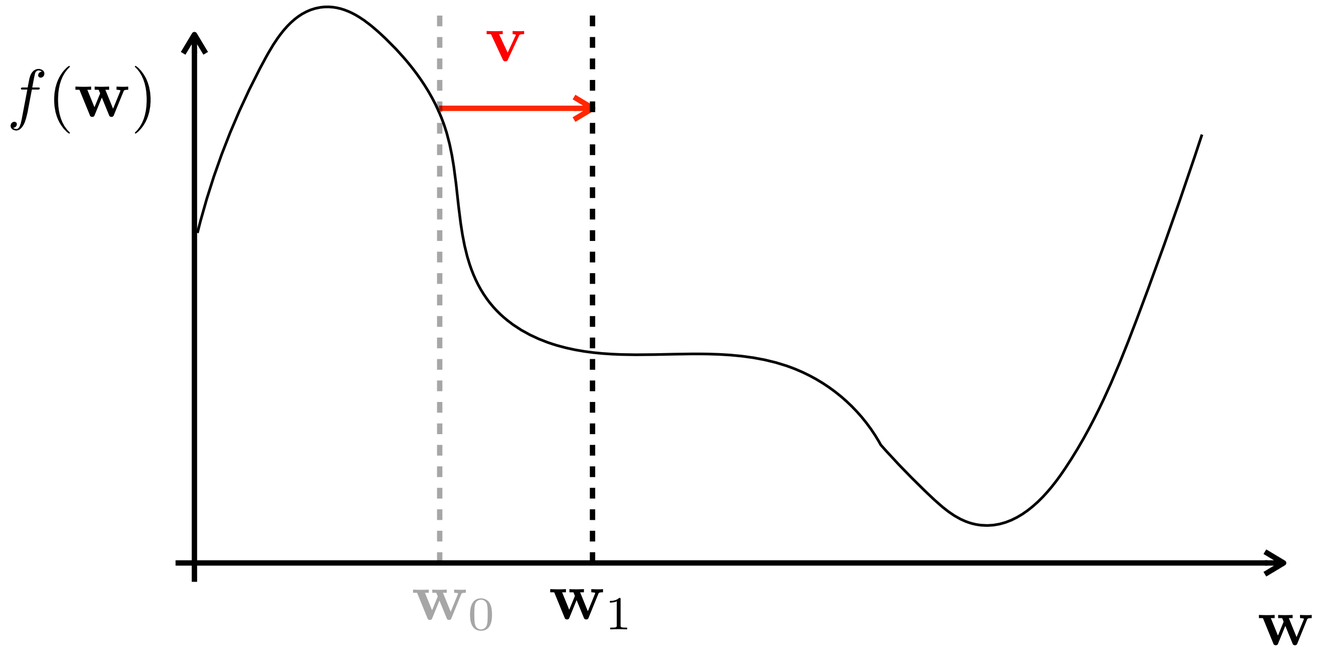
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

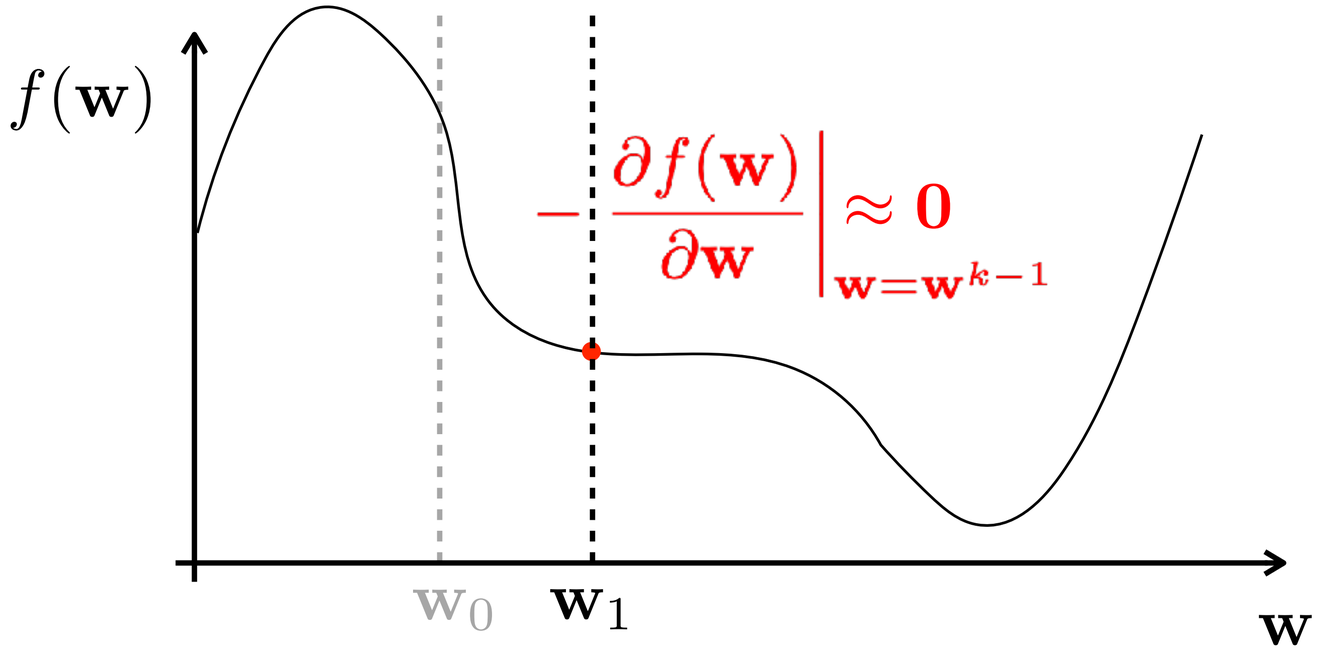
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

ODE solver:
(Euler integrator)

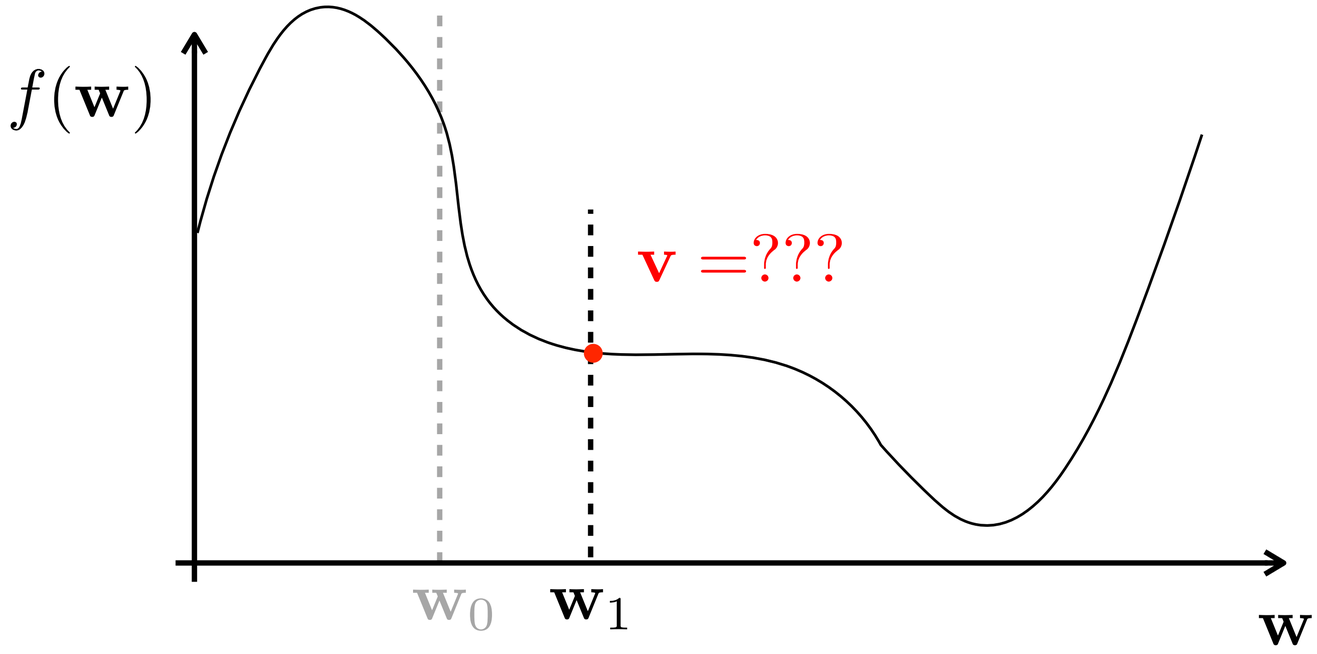
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

position

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

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

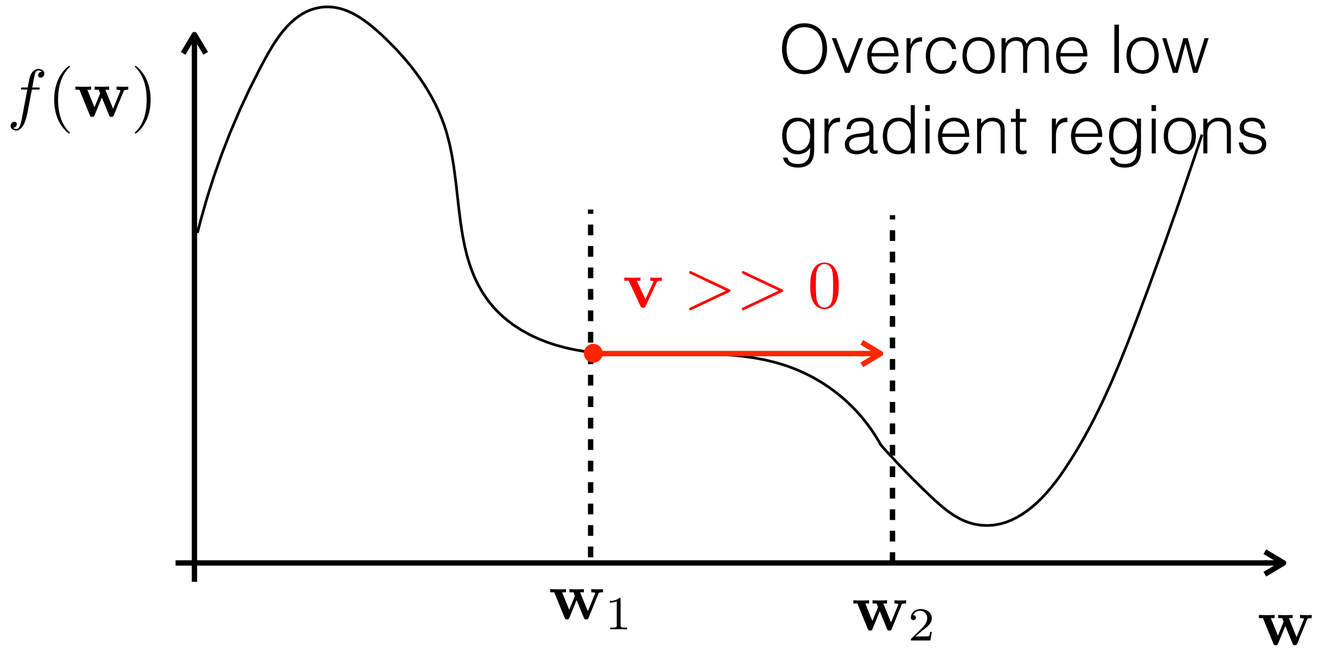
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

ODE solver:
(Euler integrator)

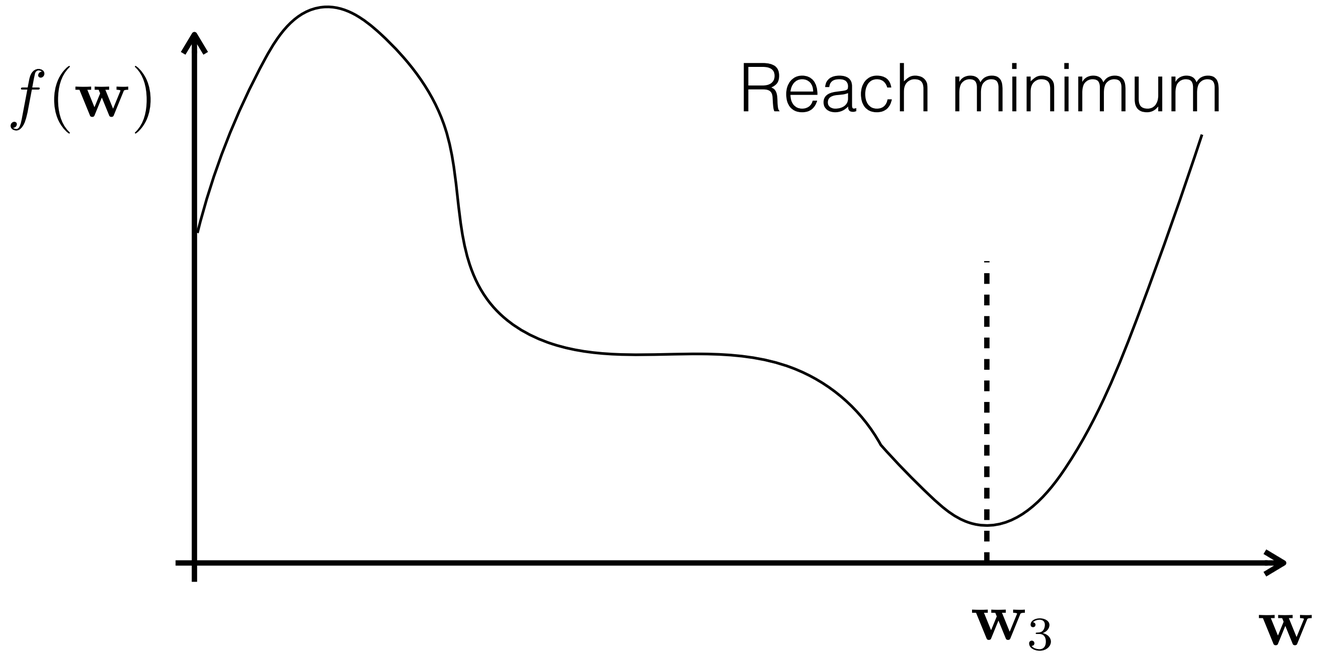
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

position

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

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

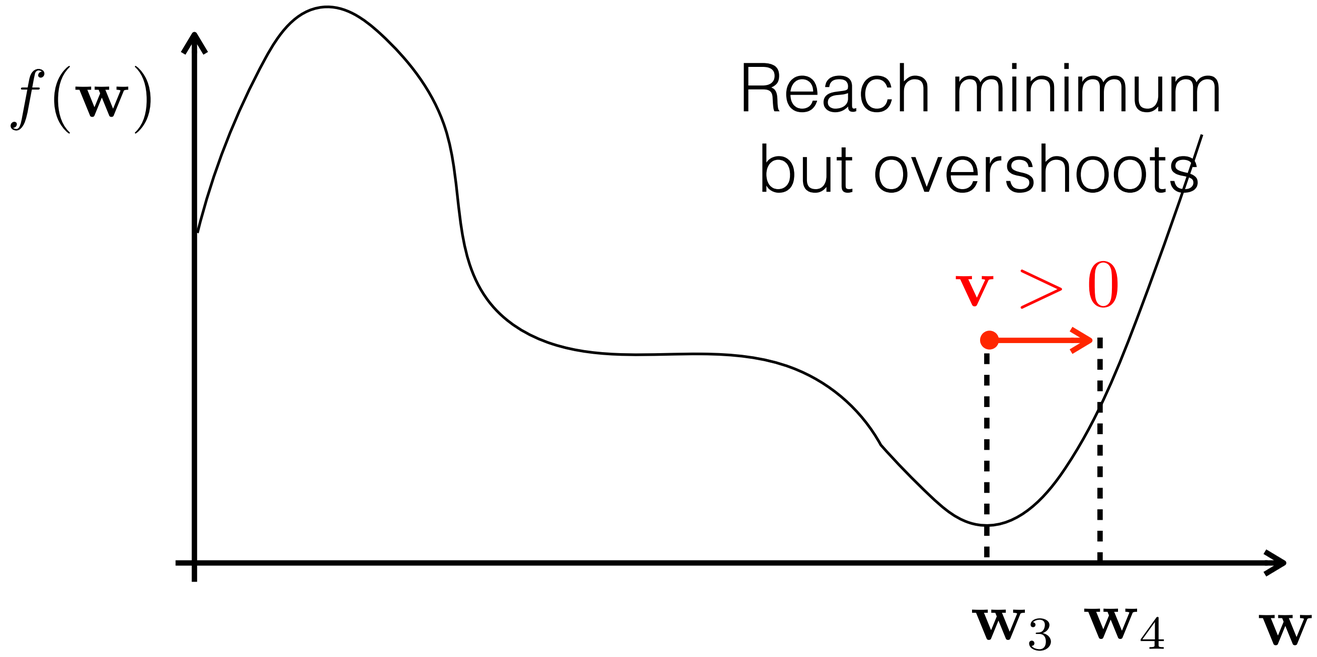
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

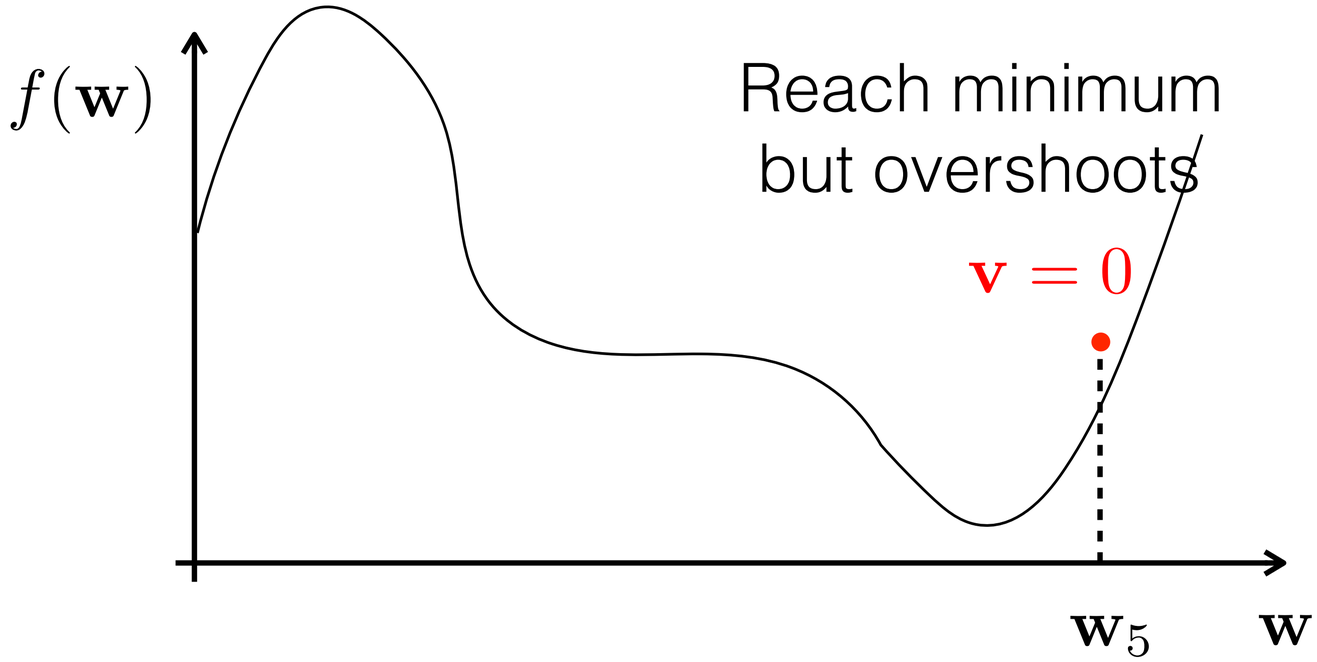
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

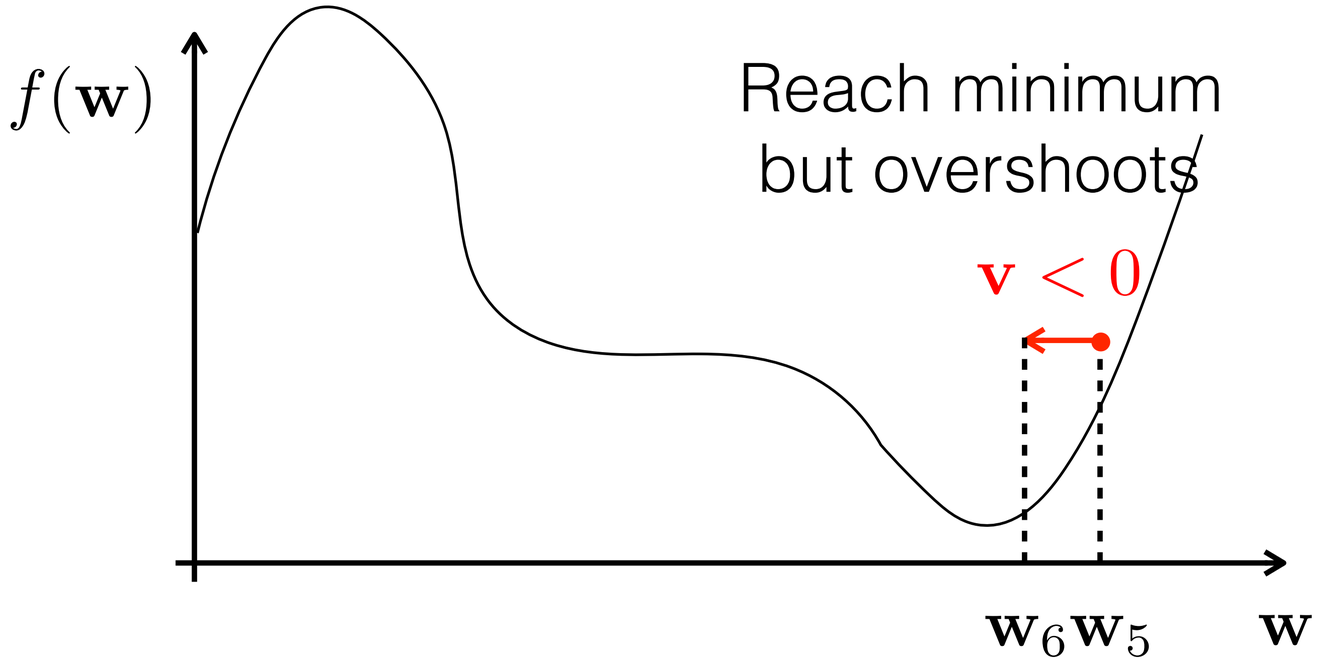
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

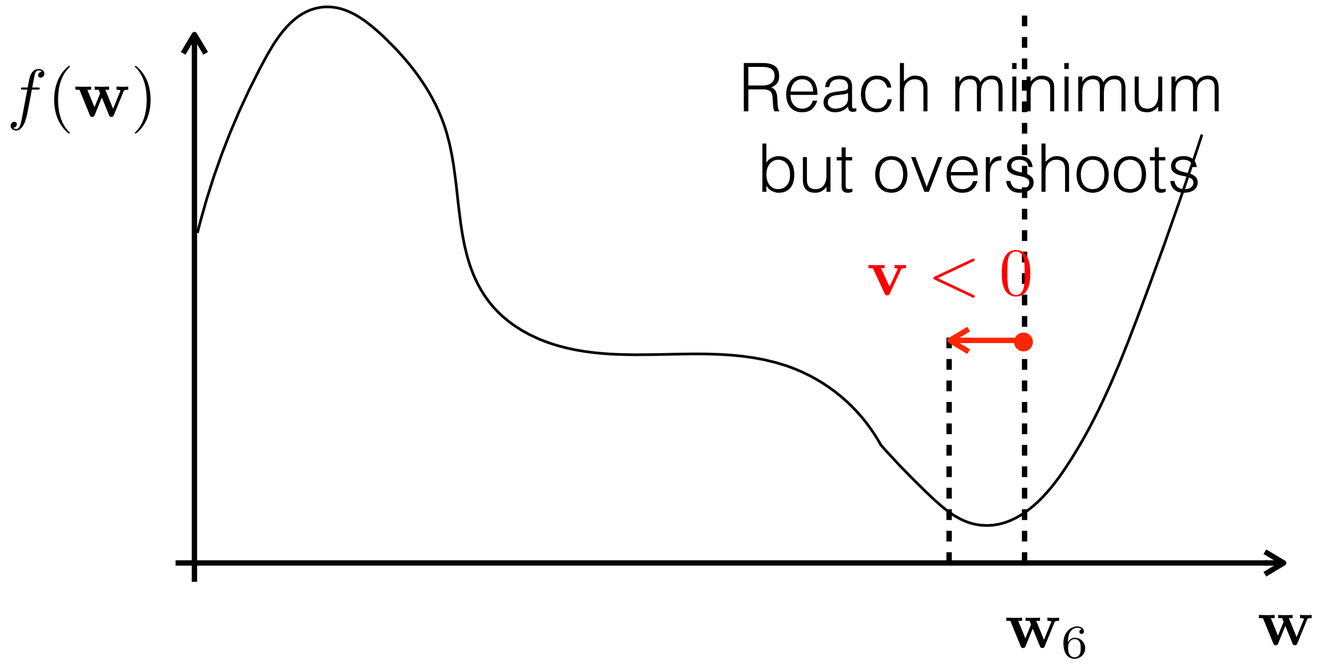
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)



SGD + momentum

friction

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

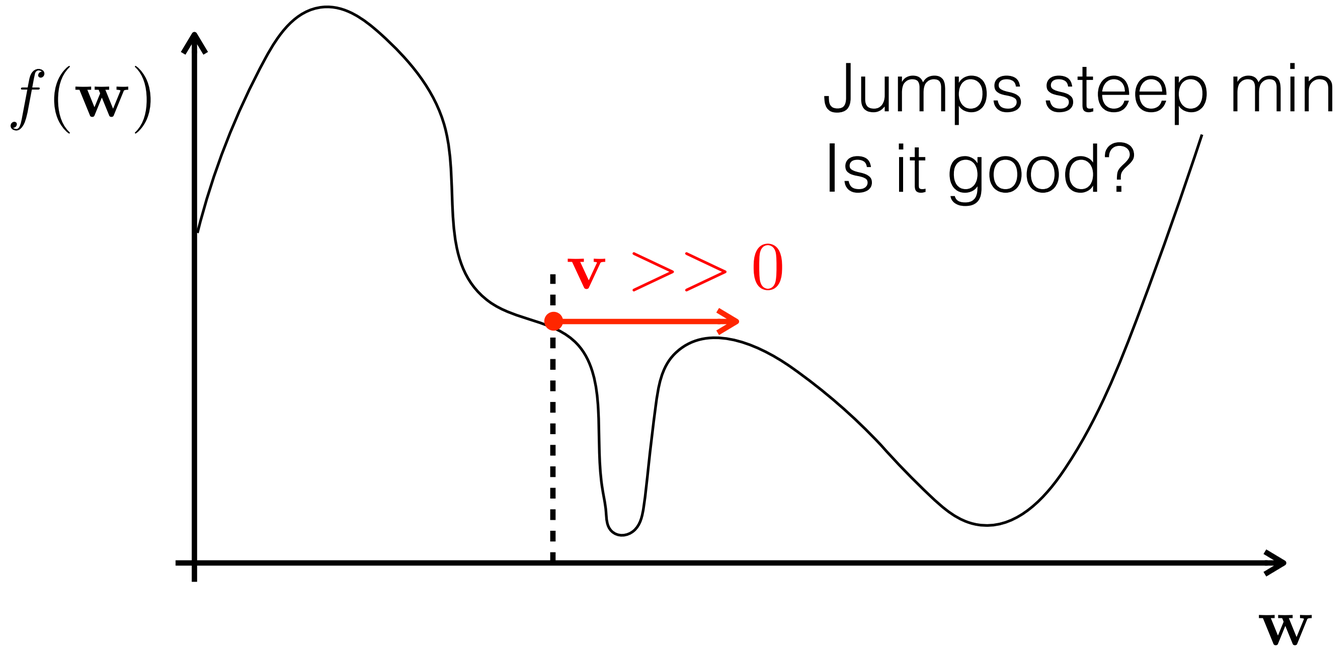
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

ODE solver:
(Euler integrator)

position

velocity

acceleration
(F_g projected on terrain slope)

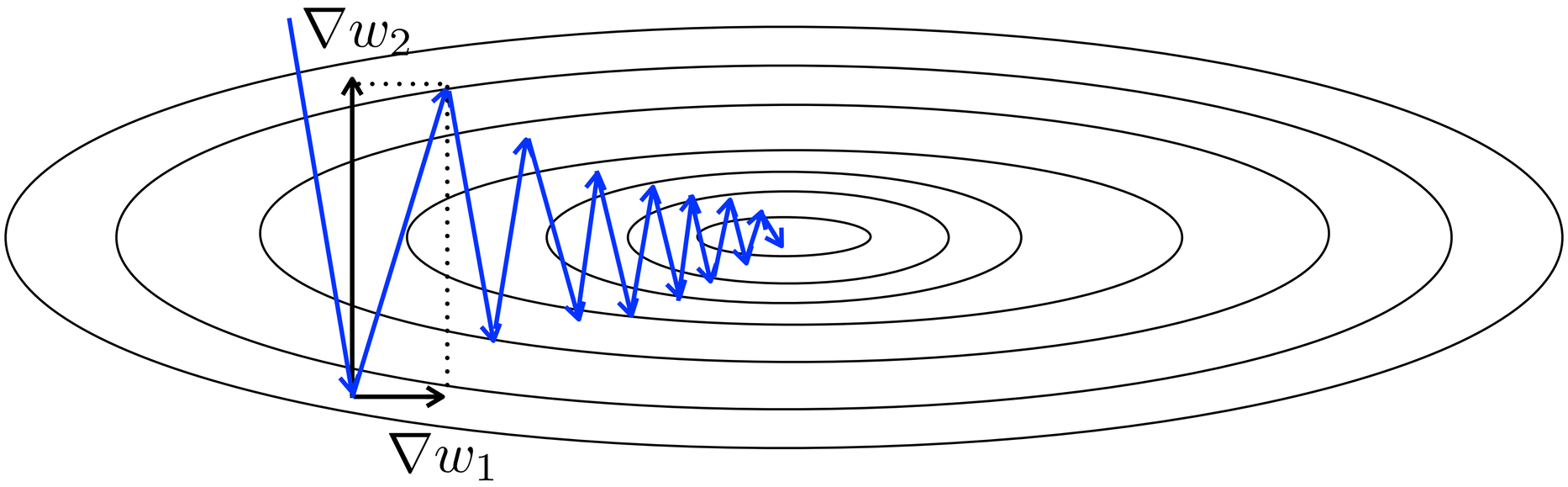


Jumps steep minima.
Is it good?

“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

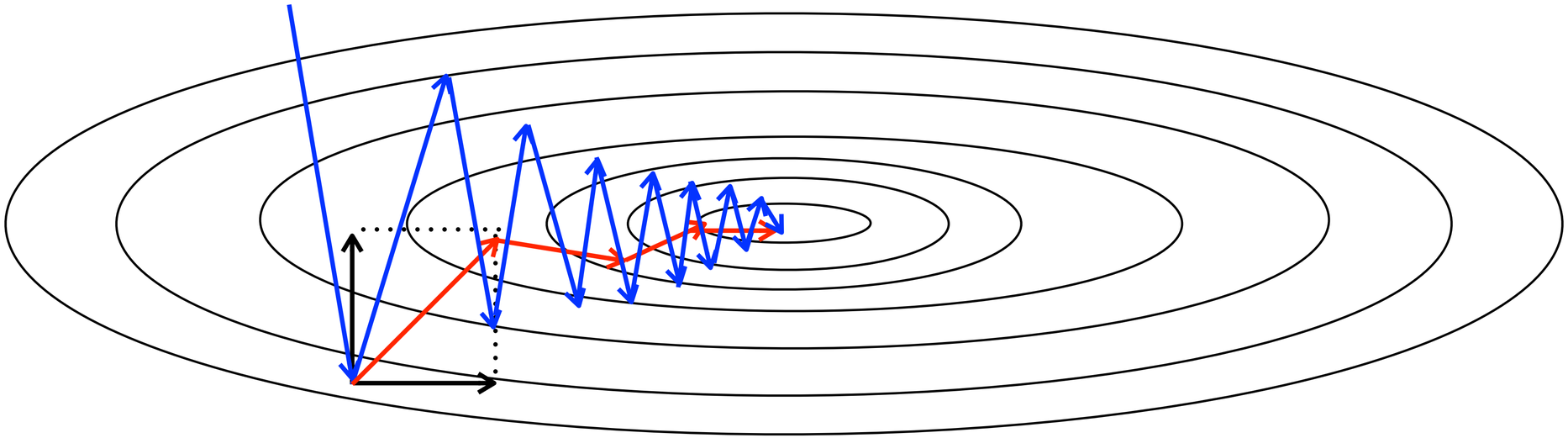


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

“SGD” vs “SGD + momentum” in 2D

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

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



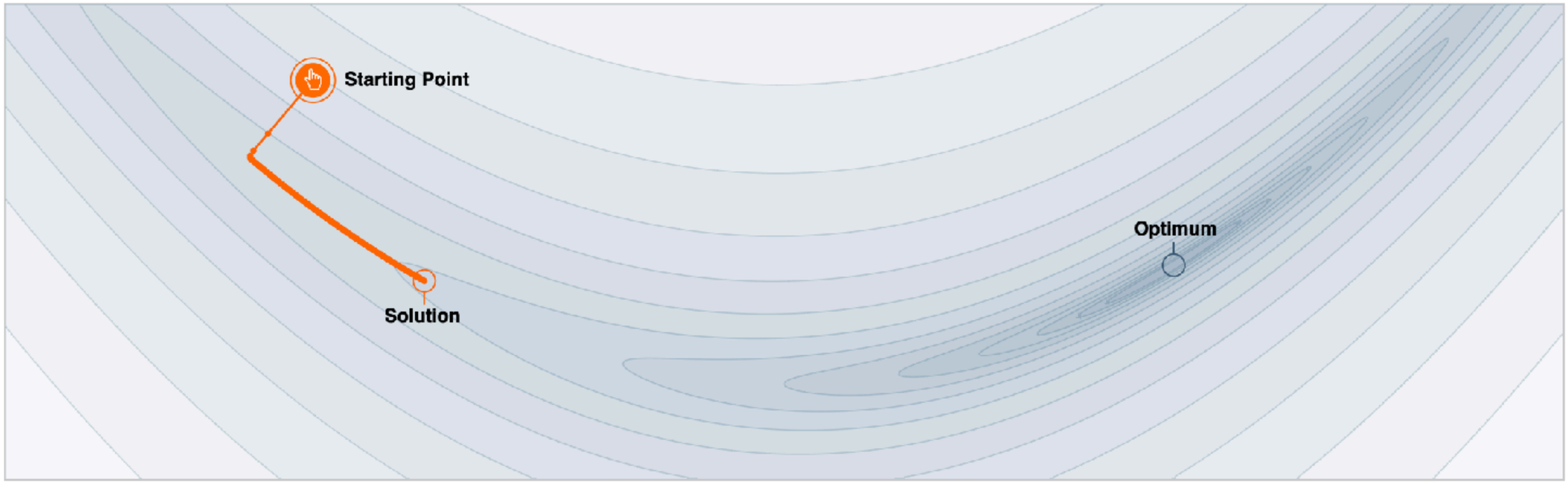
Momentum suppresses this problem partially by averaging element-wise gradients

“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



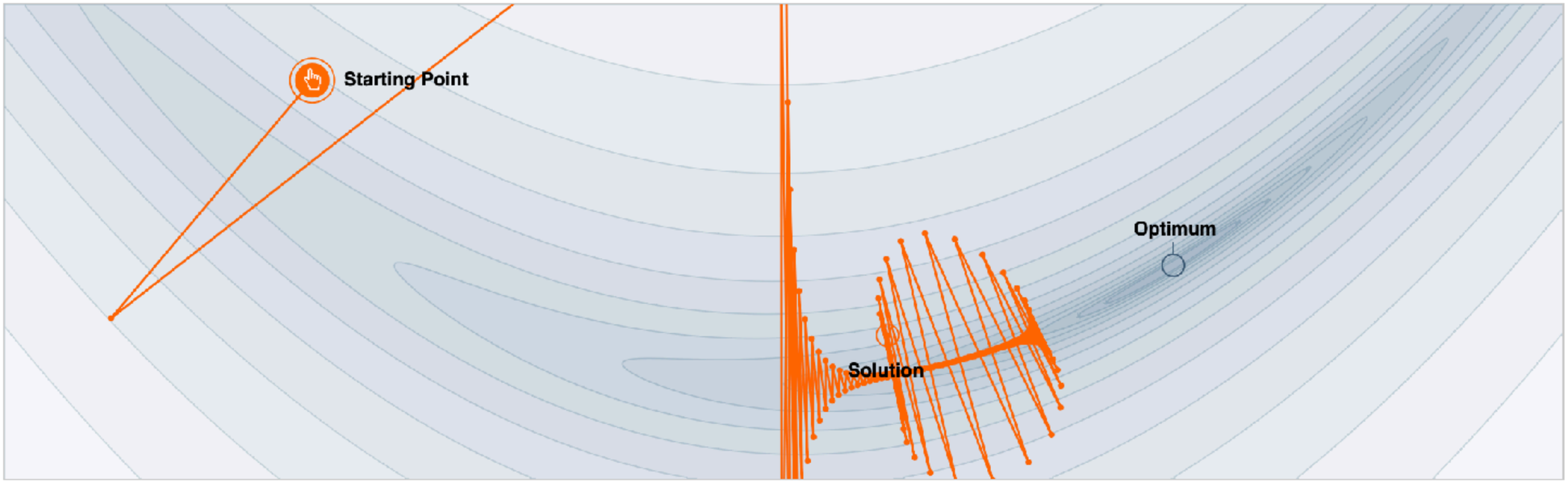
https://distill.pub/2017/momentum/

“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



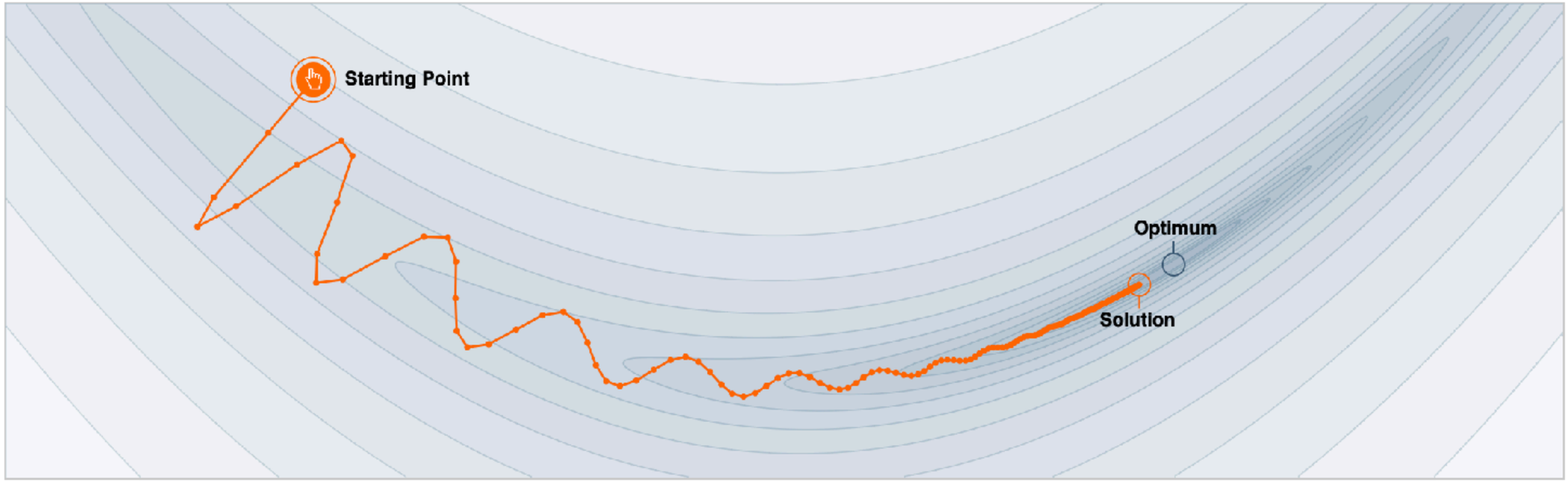
https://distill.pub/2017/momentum/

“SGD” vs “SGD + momentum” in 2D

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

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

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



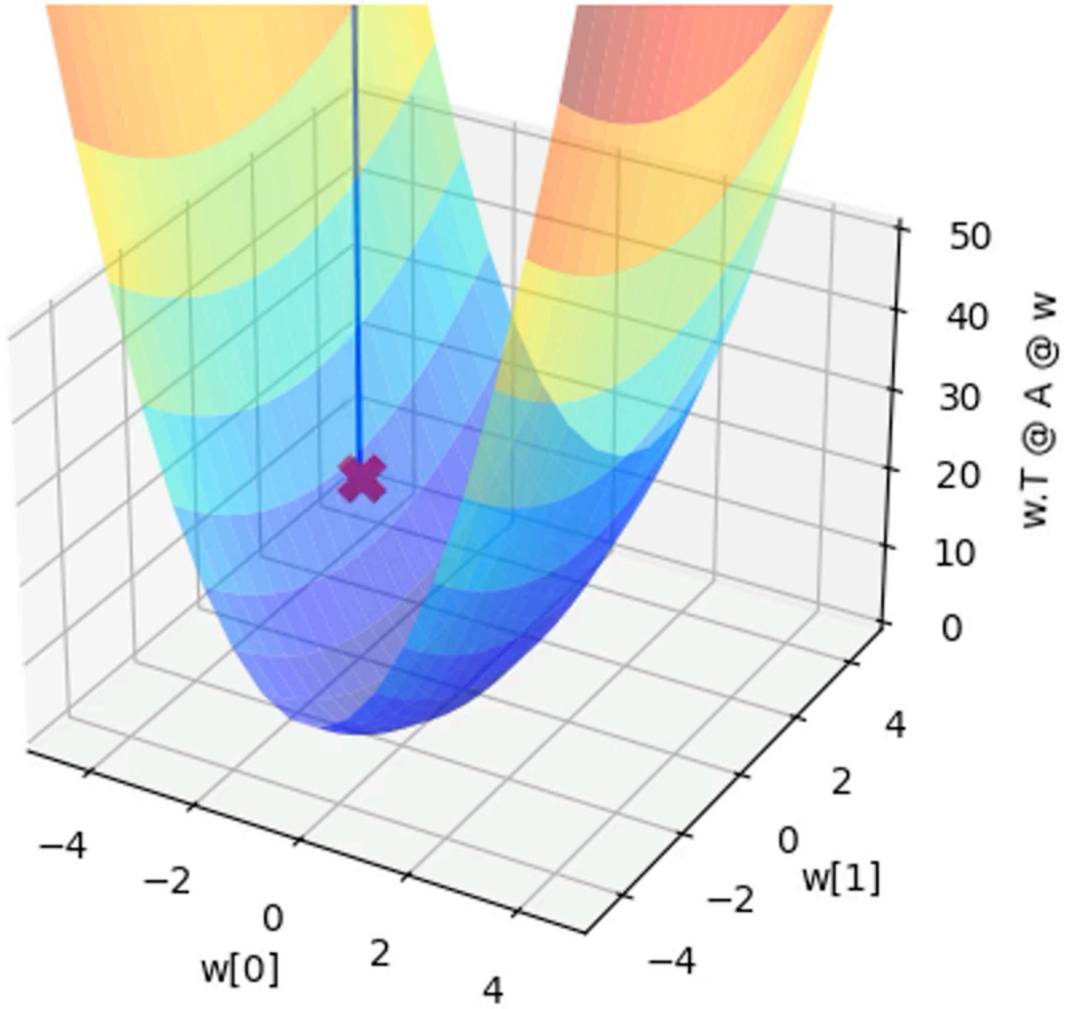
https://distill.pub/2017/momentum/

SGD vs SGD momentum

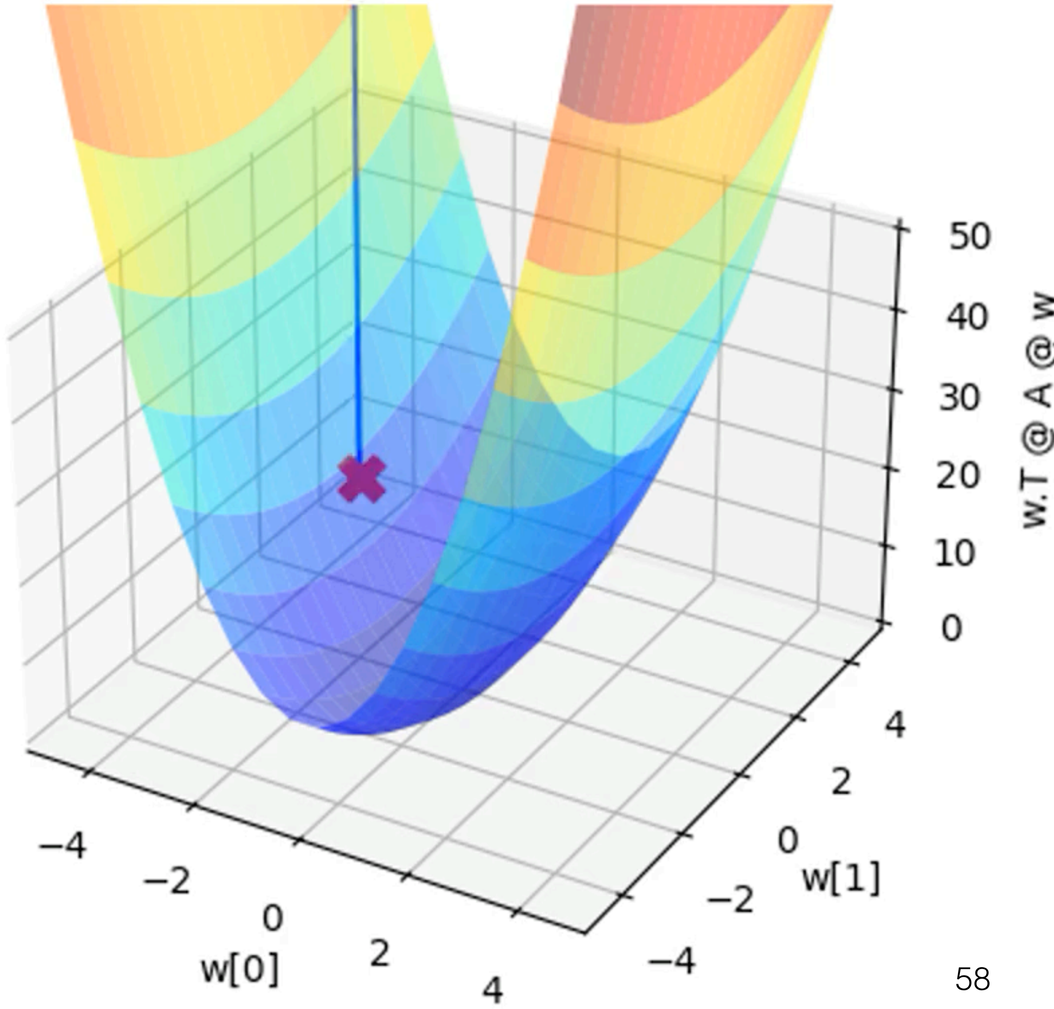
$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A} (\mathbf{w} - \mathbf{w}_i)$$

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

SGD



SGD + momentum

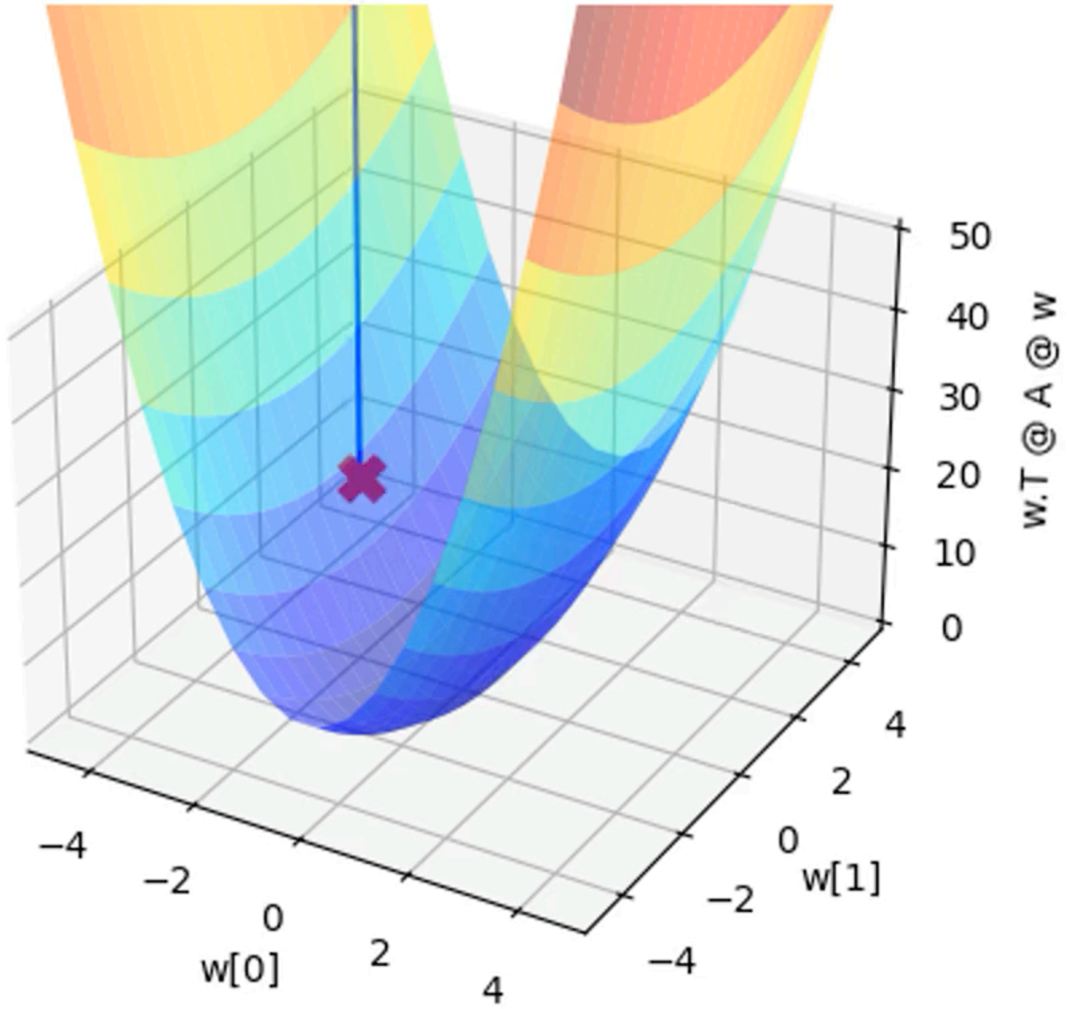


lr=0.1

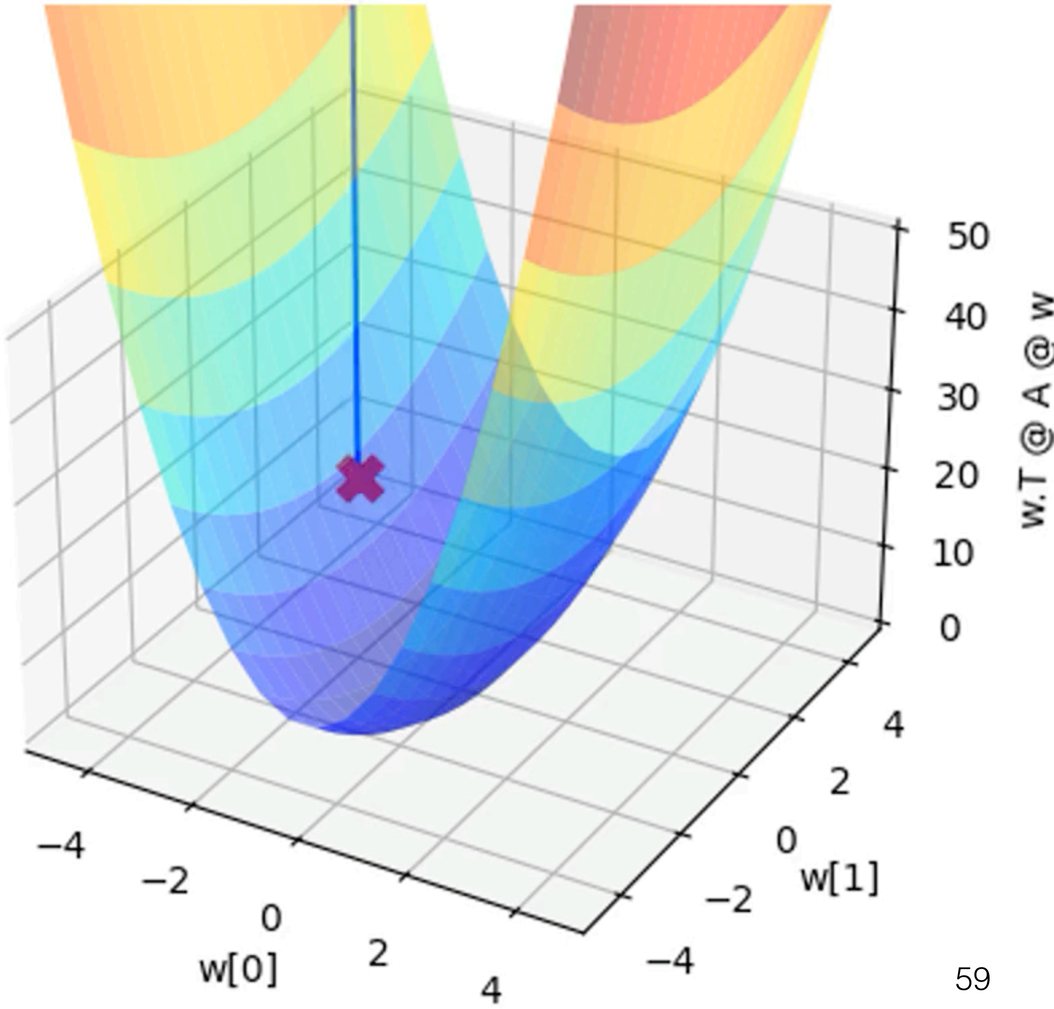
SGD vs SGD momentum

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

SGD



SGD + momentum



lr=0.2

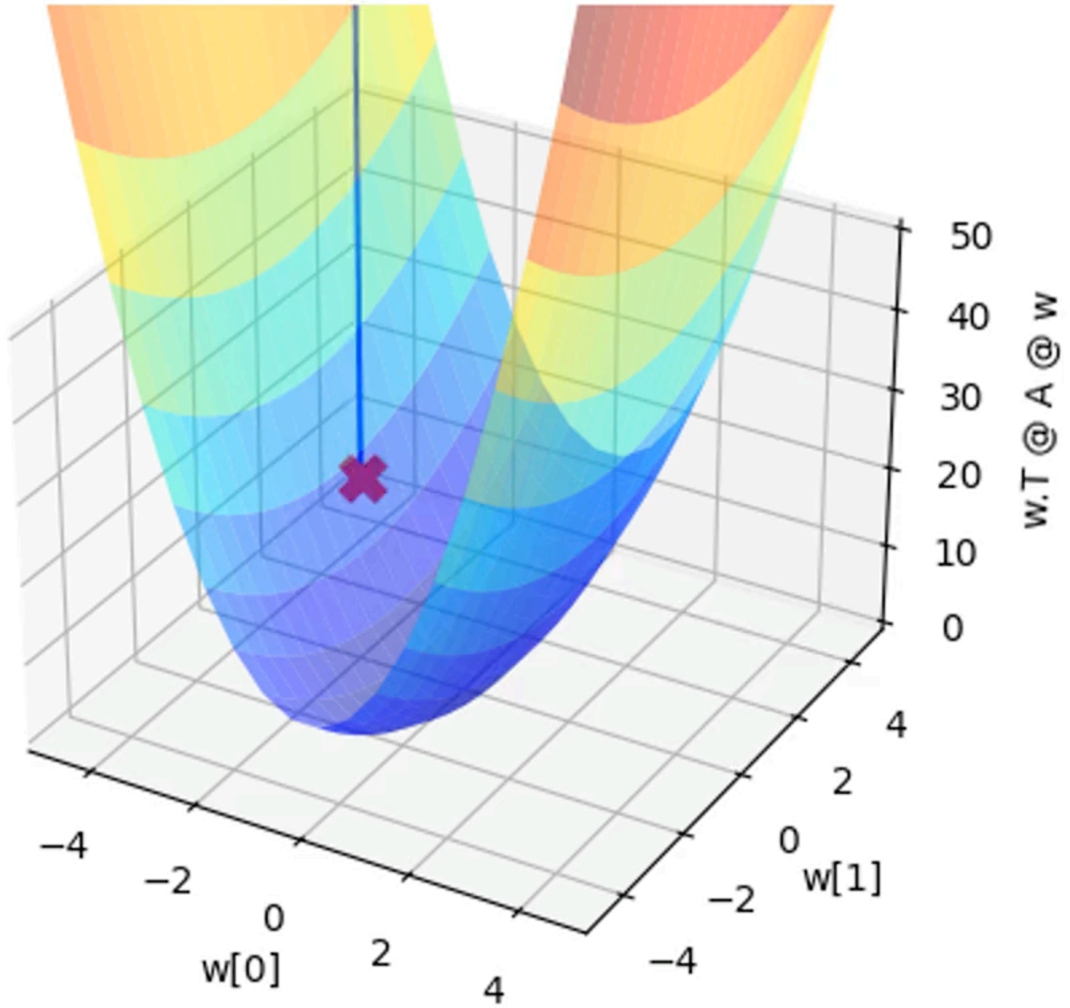
SGD vs SGD momentum

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

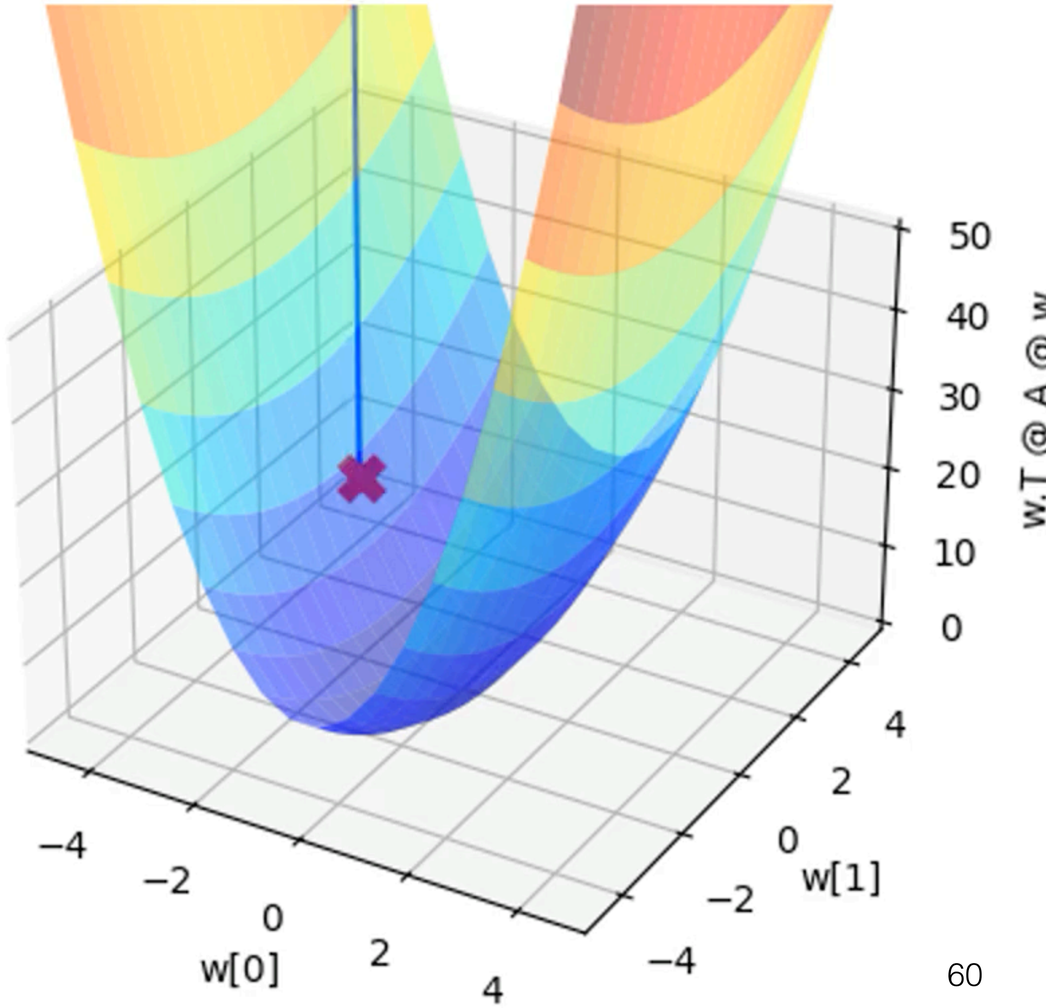
$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A}$$

SGD

SGD + momentum



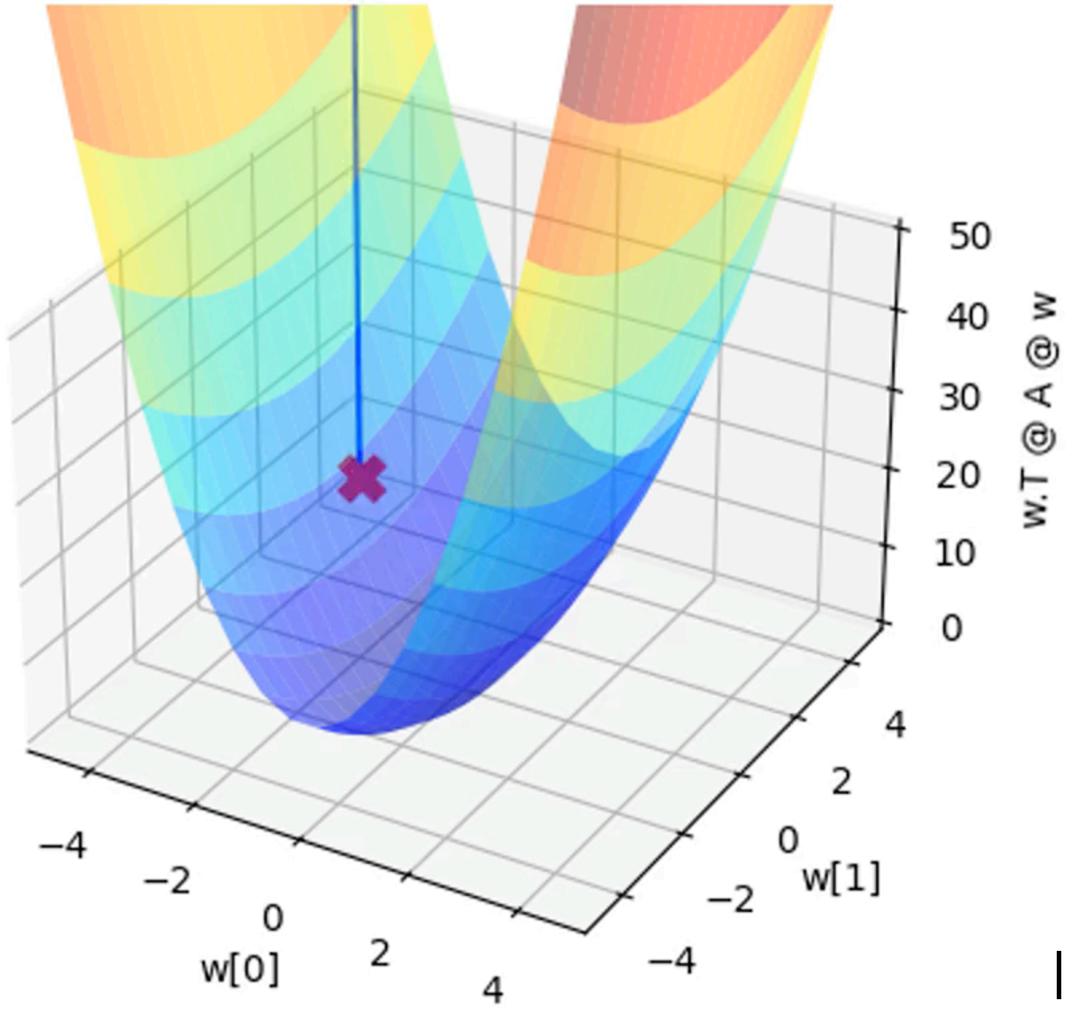
lr=0.25



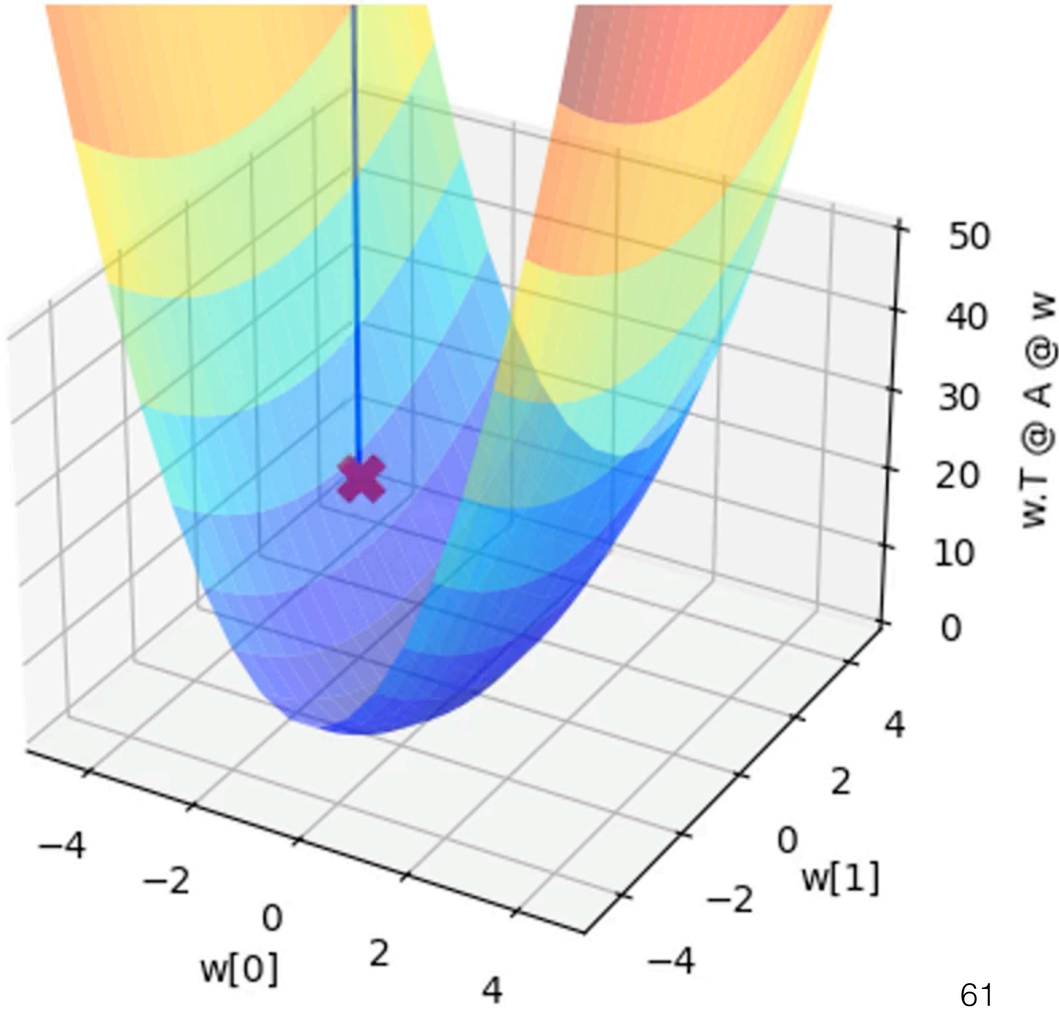
SGD vs SGD momentum

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

SGD



SGD + momentum

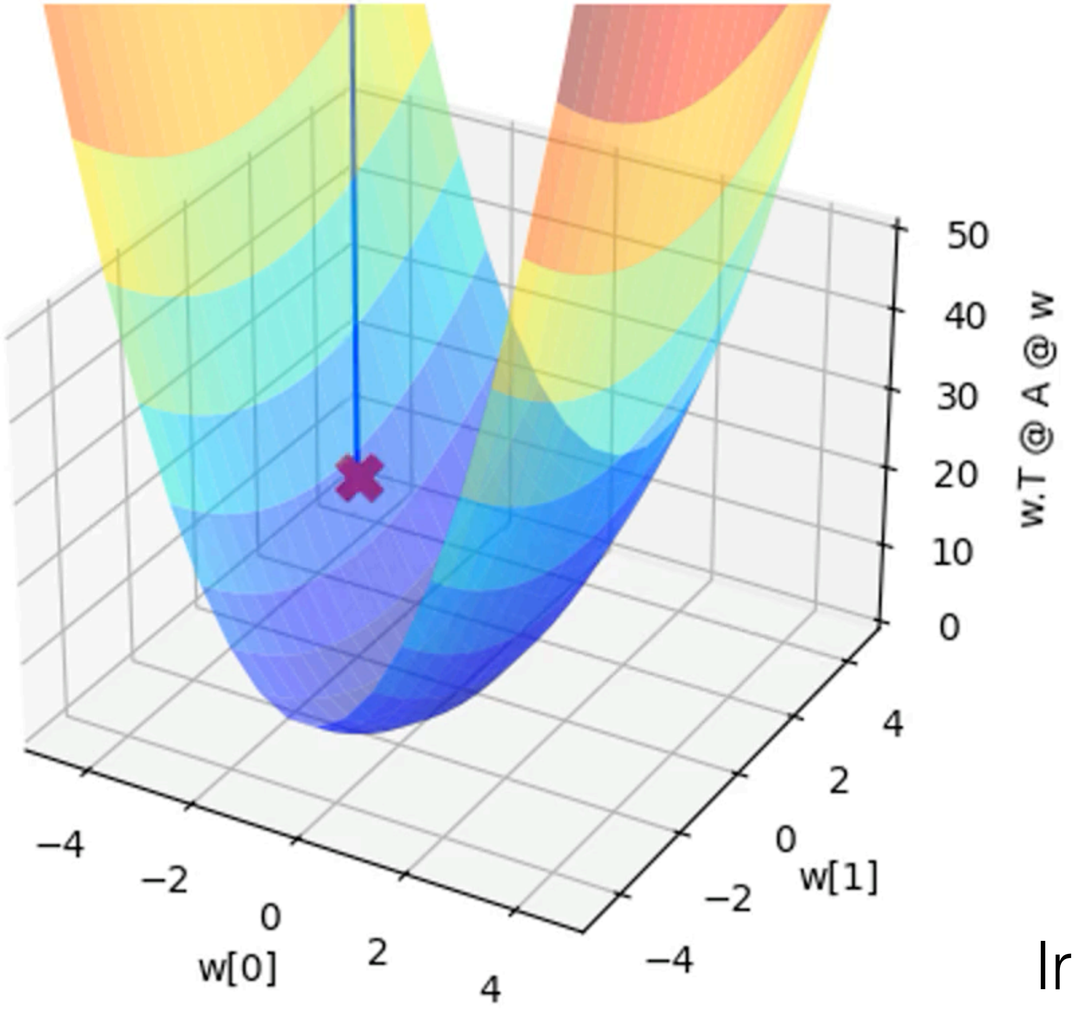


lr=0.3

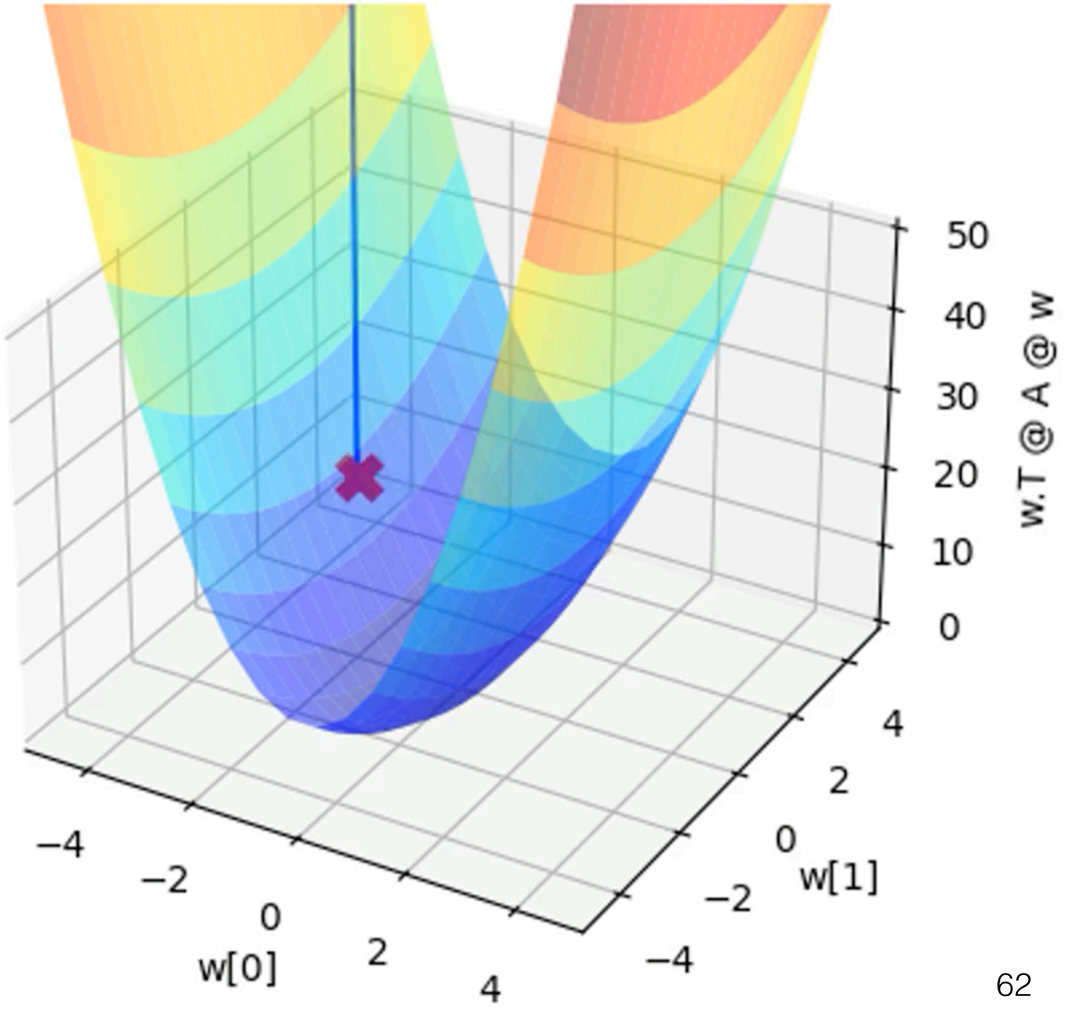
SGD vs SGD momentum

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

SGD



SGD + momentum

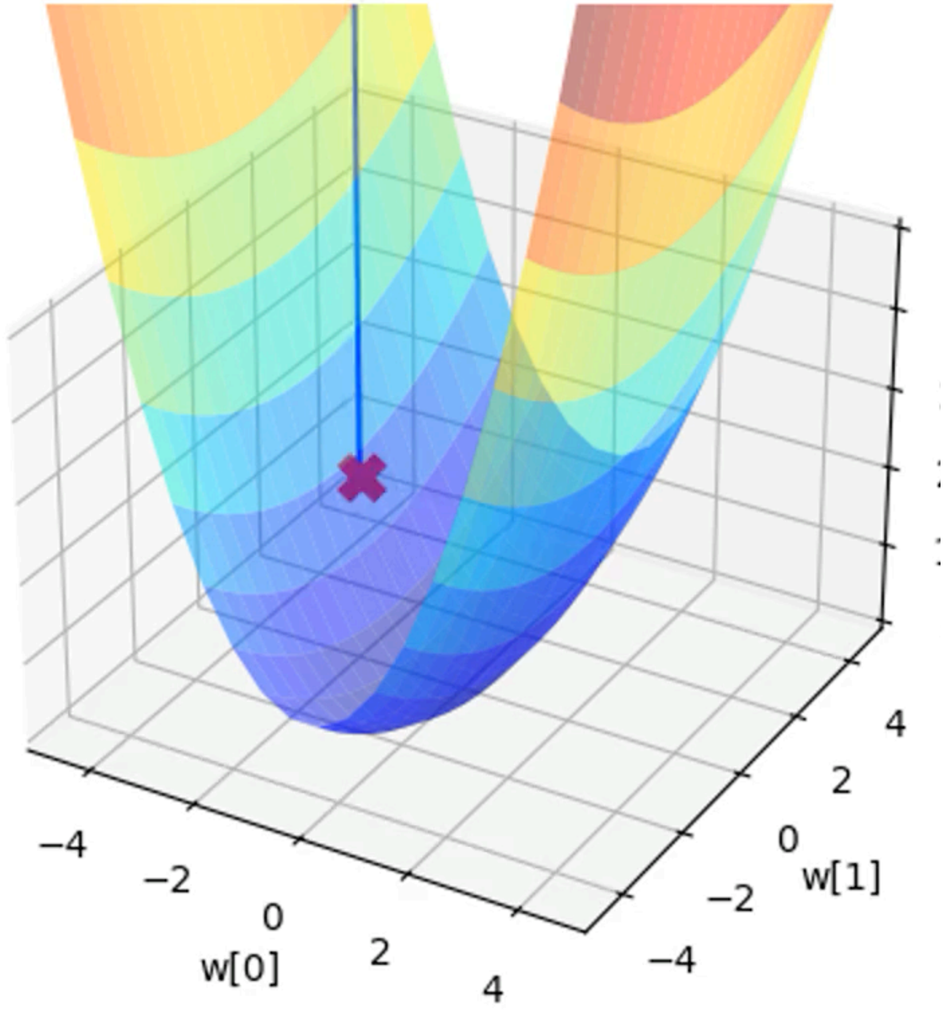


lr=0.4

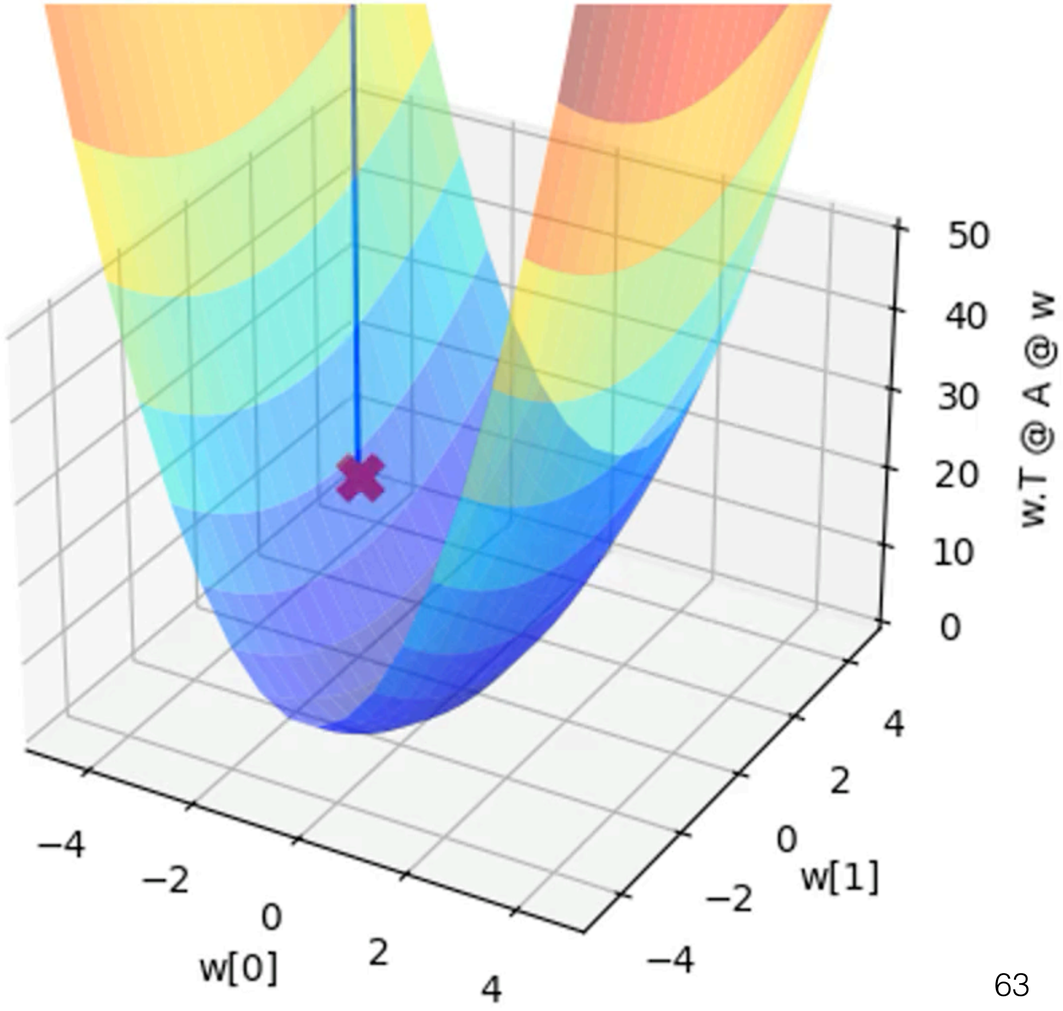
SGD vs SGD momentum

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

SGD



SGD + momentum

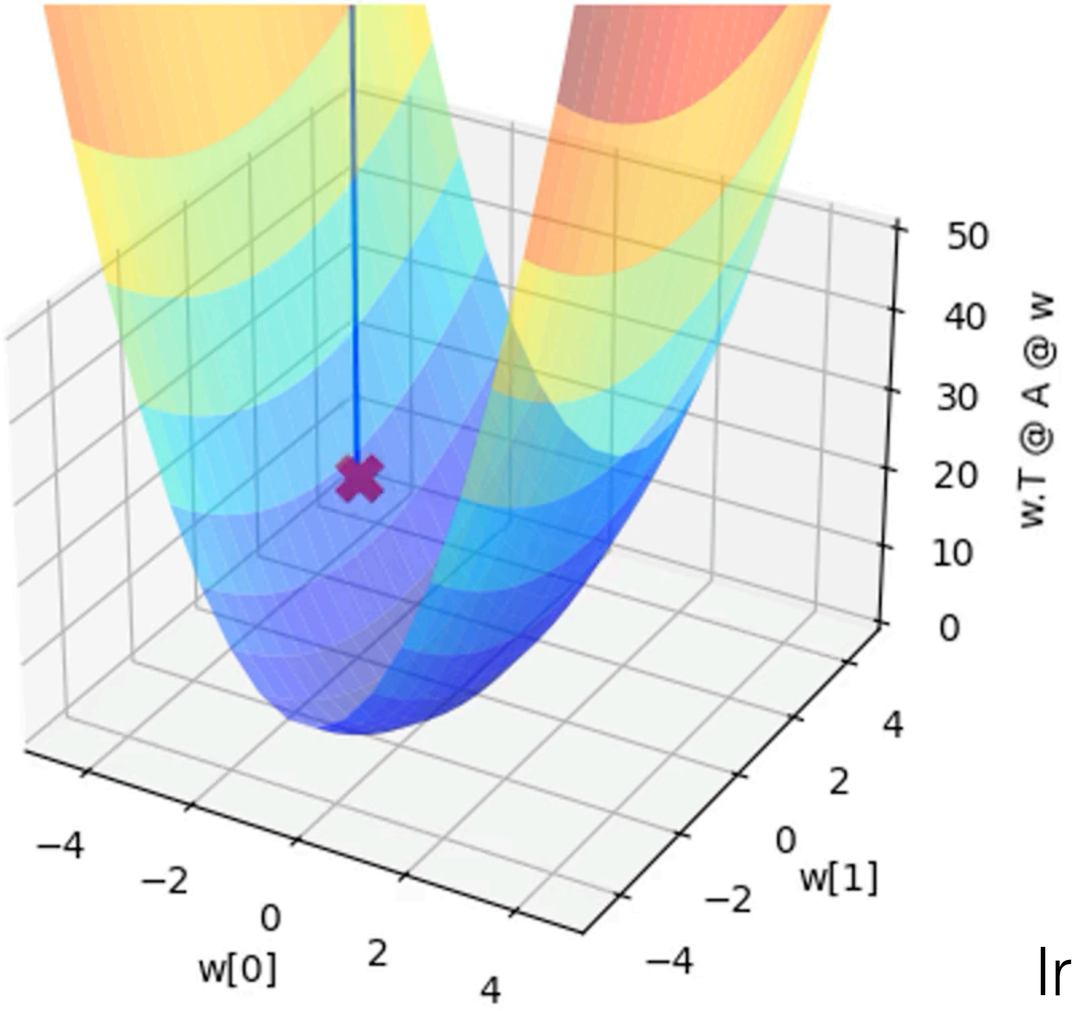


lr=2.0

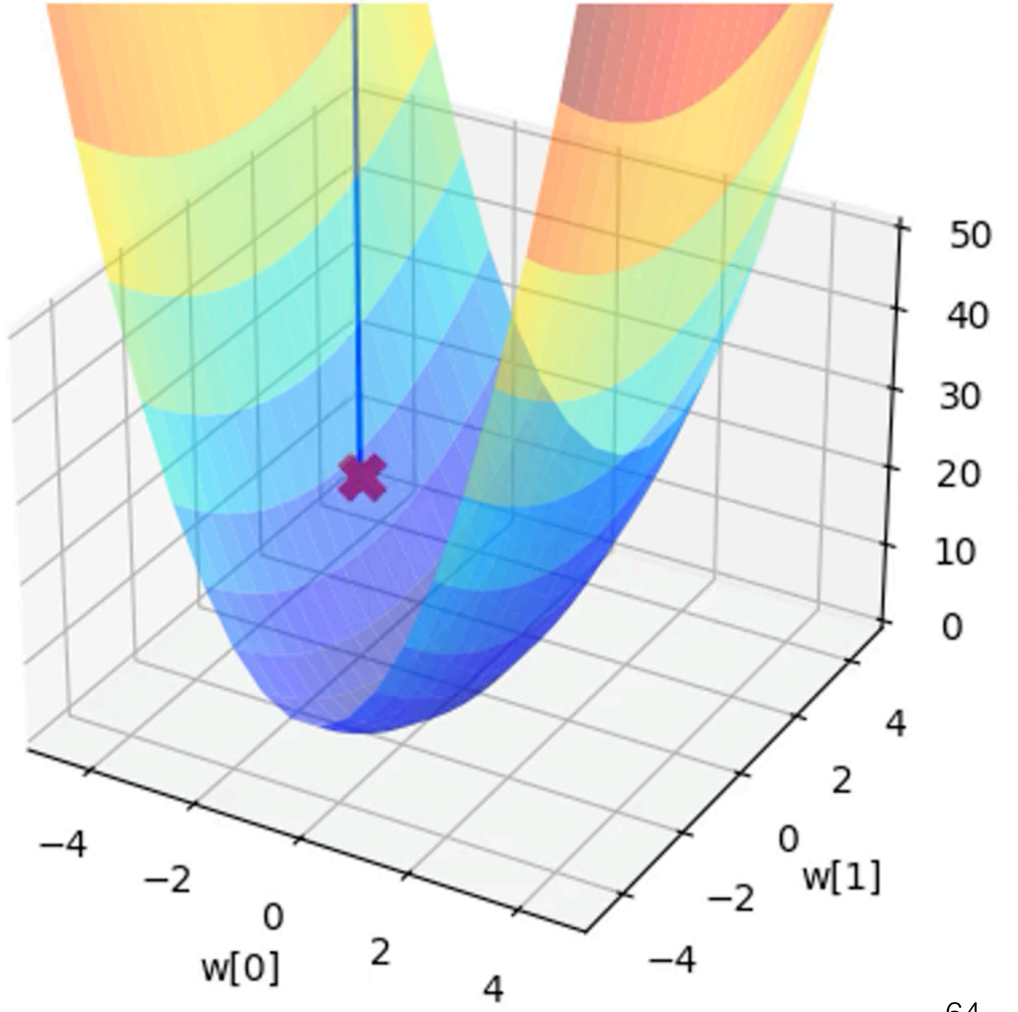
SGD vs SGD momentum

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

SGD



SGD + momentum

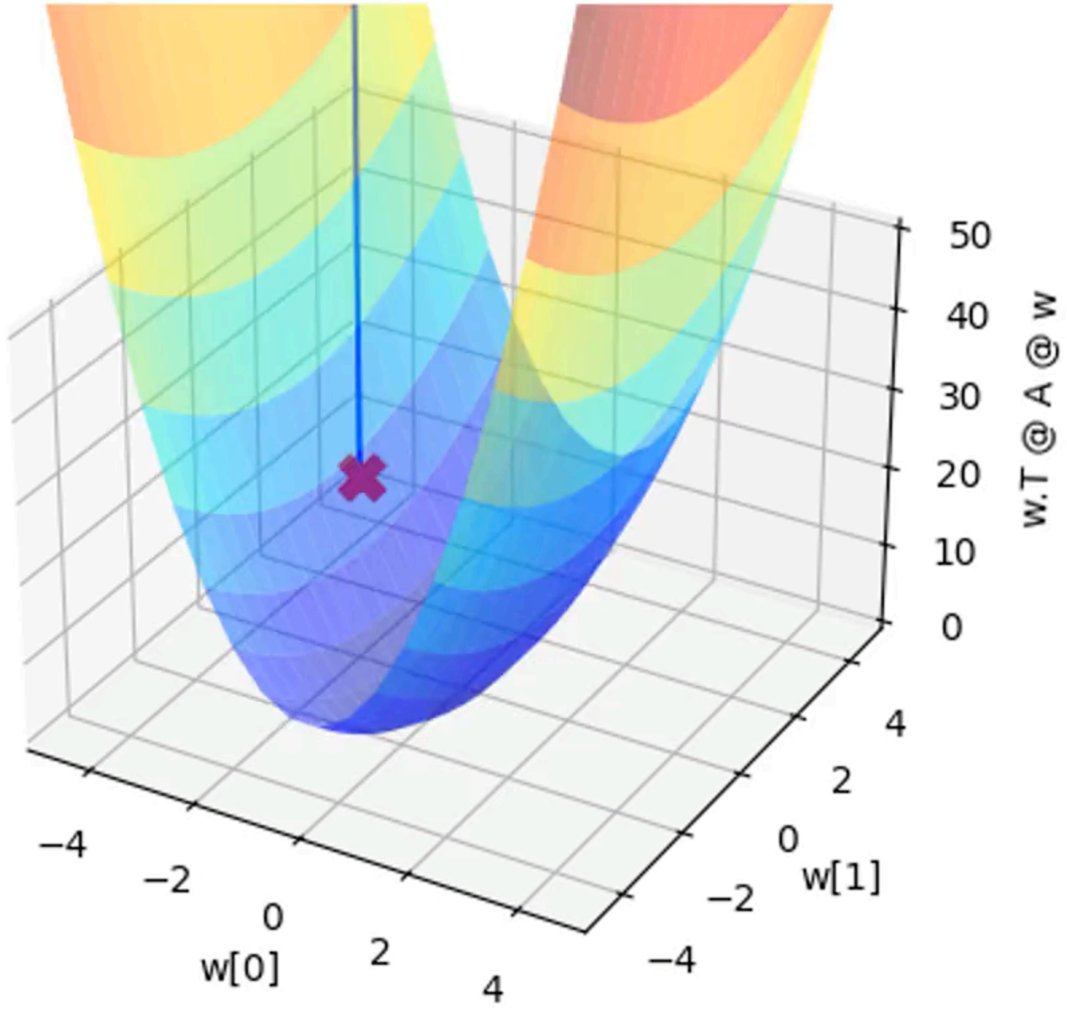


lr=10

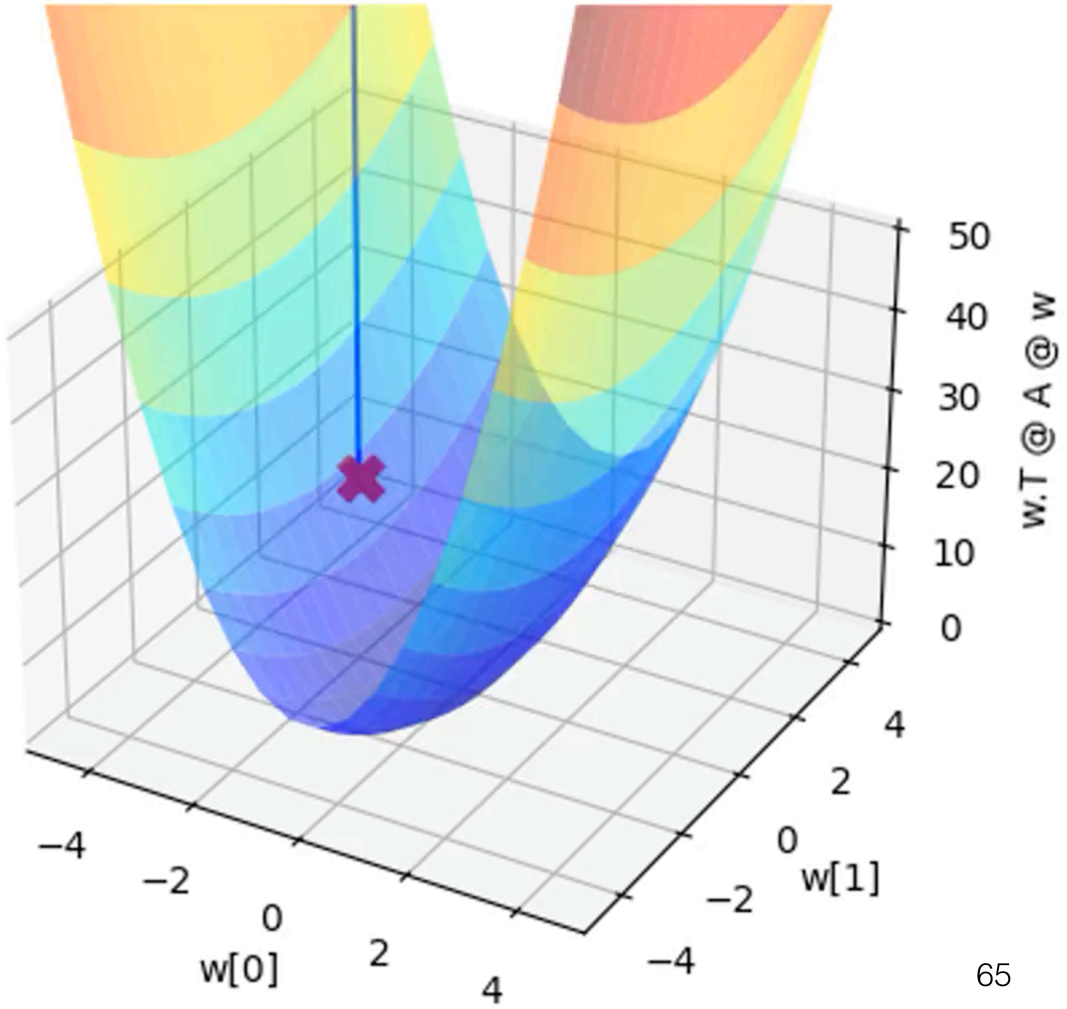
Batches

$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A} (\mathbf{w} - \mathbf{w}_i) \qquad \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A}, \quad i = \text{rand}(1,1000)$$

SGD



SGD + momentum



lr=0.2

“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:

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

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

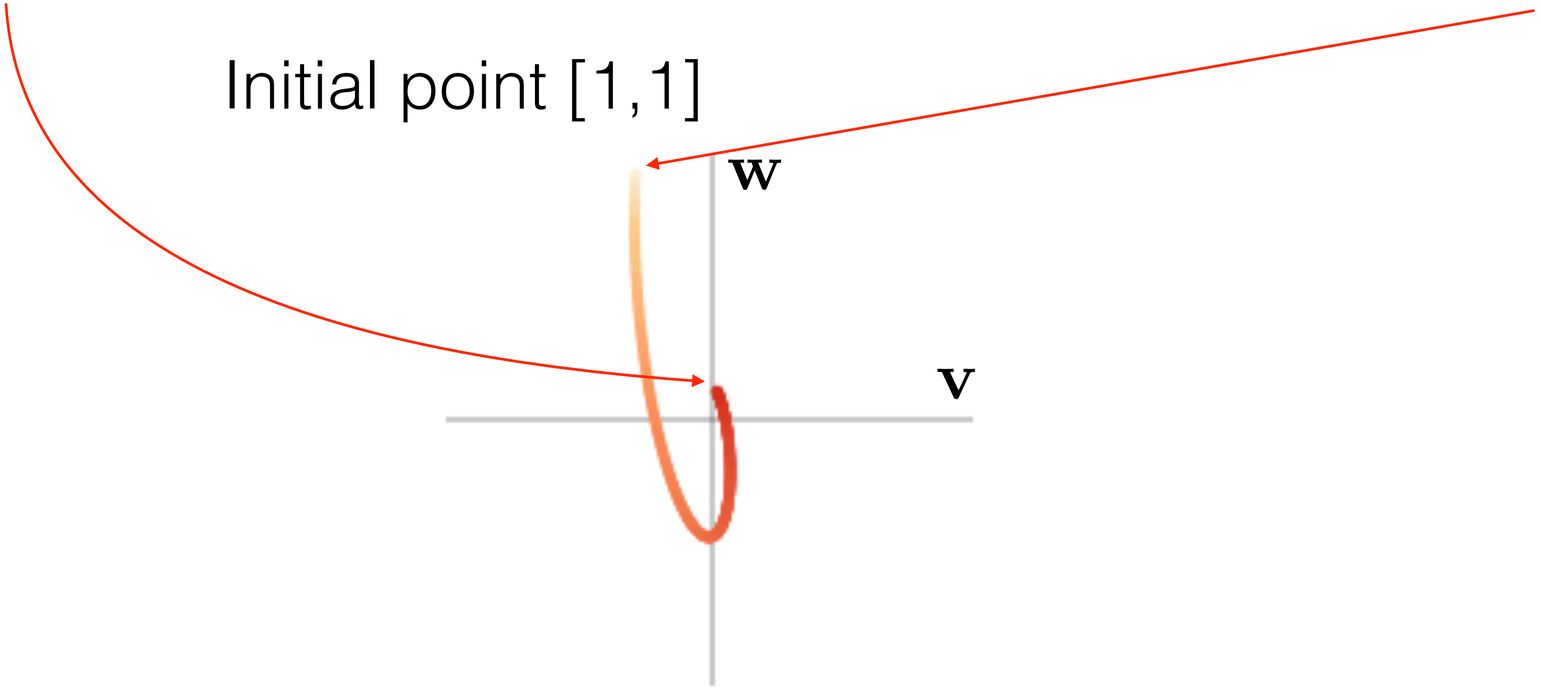
[Flammarion, Bach COLT 2017]
https://arxiv.org/pdf/1504.01577.pdf
https://distill.pub/2017/momentum/

“SGD + momentum” on quadric

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

Final point

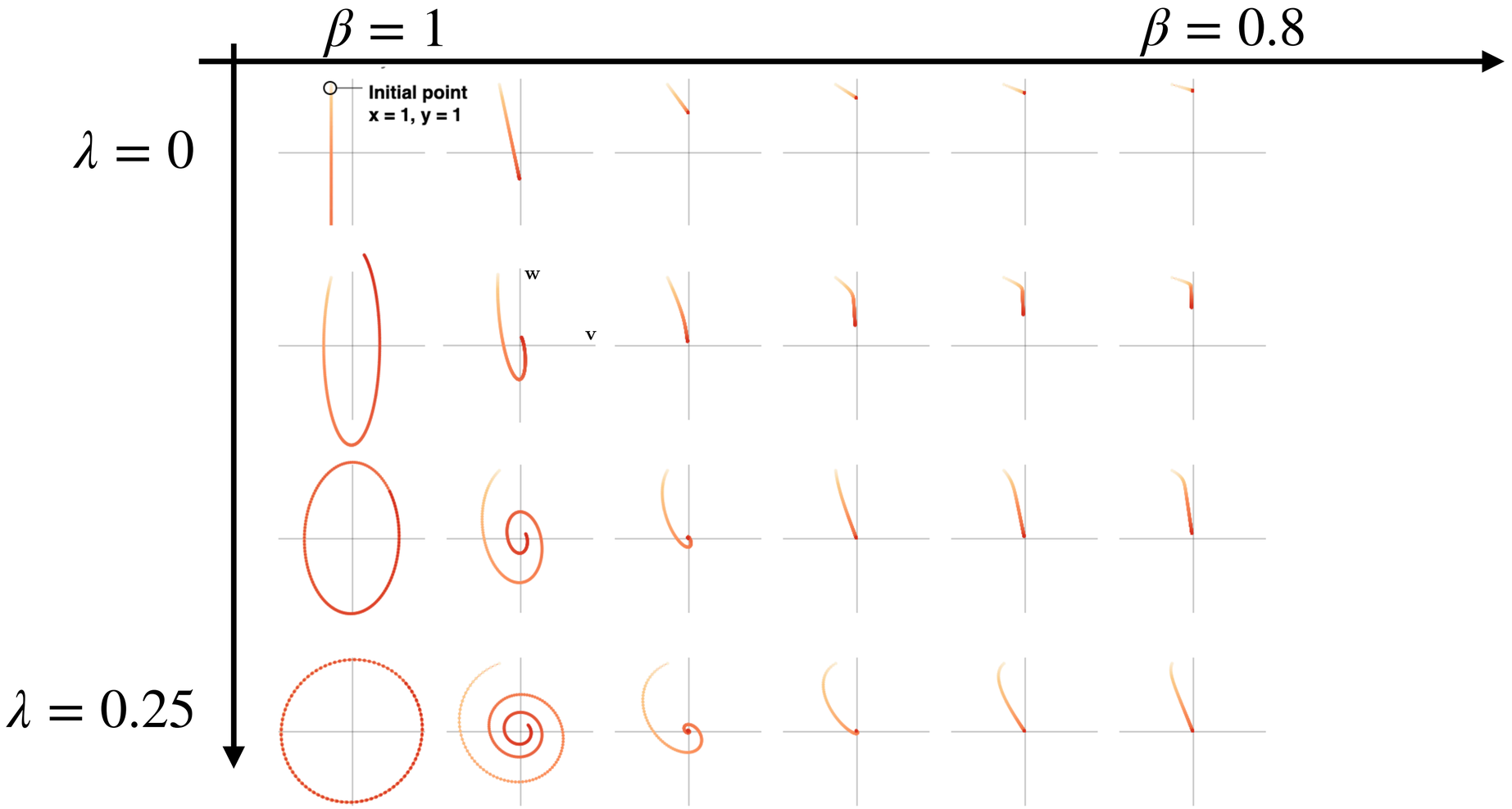
Initial point [1,1]



[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:

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

$$\sigma_1 = \frac{1}{2}\left(1 - \alpha\lambda + \beta + \sqrt{(-\alpha\lambda + \beta + 1)^2 - 4\beta}\right)$$

$$\sigma_2 = \frac{1}{2}\left(1 - \alpha\lambda + \beta - \sqrt{(-\alpha\lambda + \beta + 1)^2 - 4\beta}\right)$$

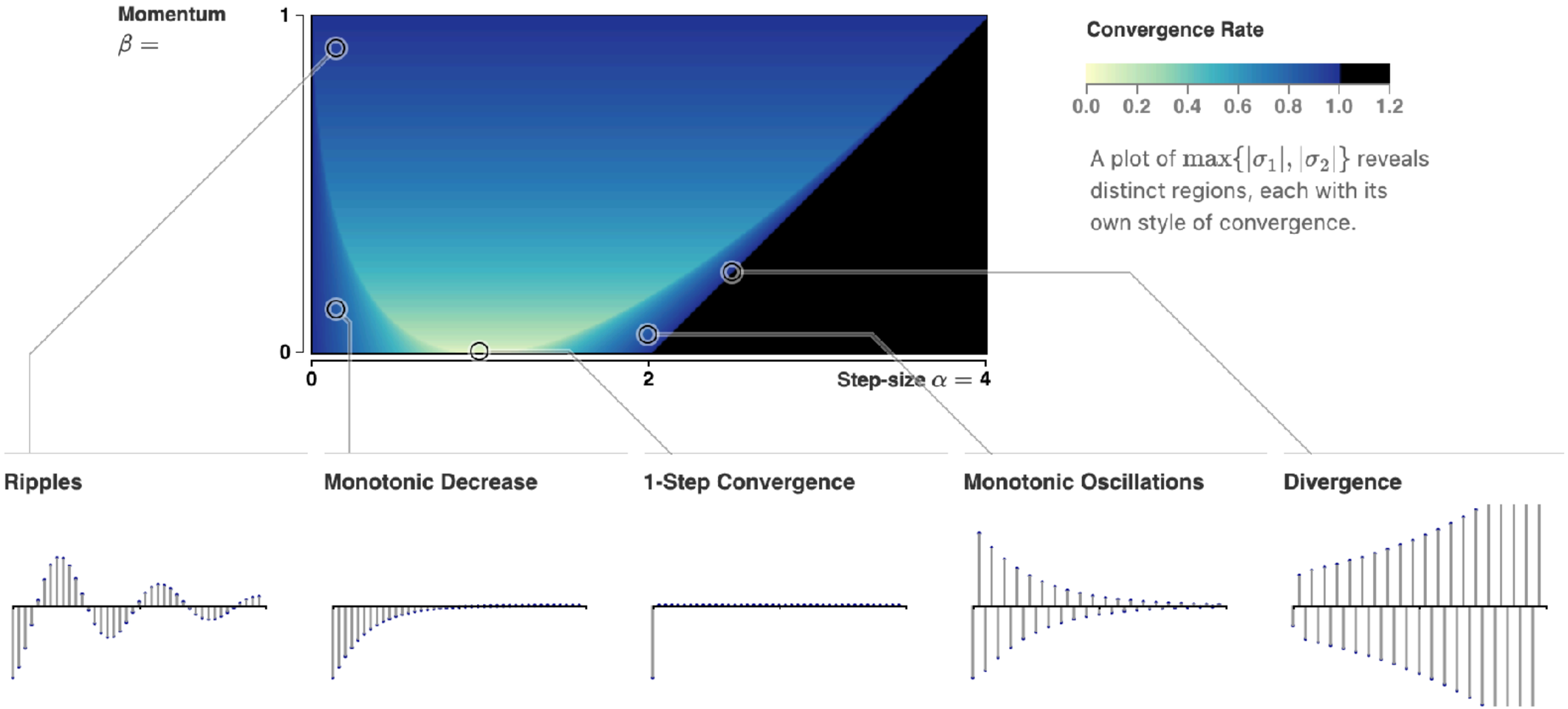
Eigenvalues of 2x2 matrix

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

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

“SGD + momentum” on quadric

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



[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: $\mathrm{rate}(\alpha, \beta) = \max_i \{|\sigma_1(\alpha, \beta, \lambda_i)|, |\sigma_2(\alpha, \beta, \lambda_i)|\}$
- Optimal parameters:

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

[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: $\mathrm{rate}(\alpha, \beta) = \max_i \{|\sigma_1(\alpha, \beta, \lambda_i)|, |\sigma_2(\alpha, \beta, \lambda_i)|\}$
- Optimal parameters:

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

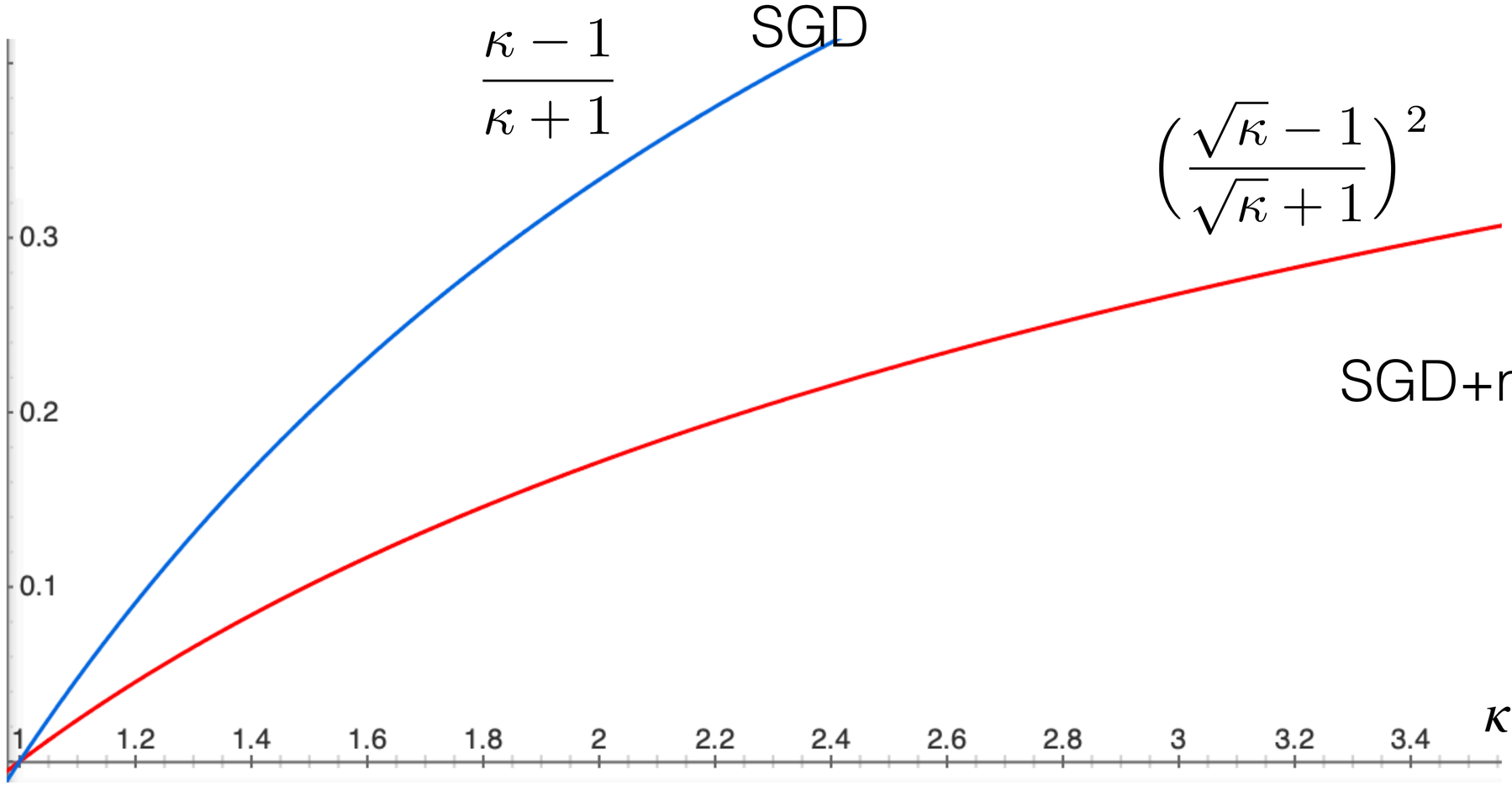
- Optimal convergence rate: $\mathrm{rate}(\alpha^*, \beta^*) = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^2$

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

“SGD + momentum” on quadric

Convergence rate

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

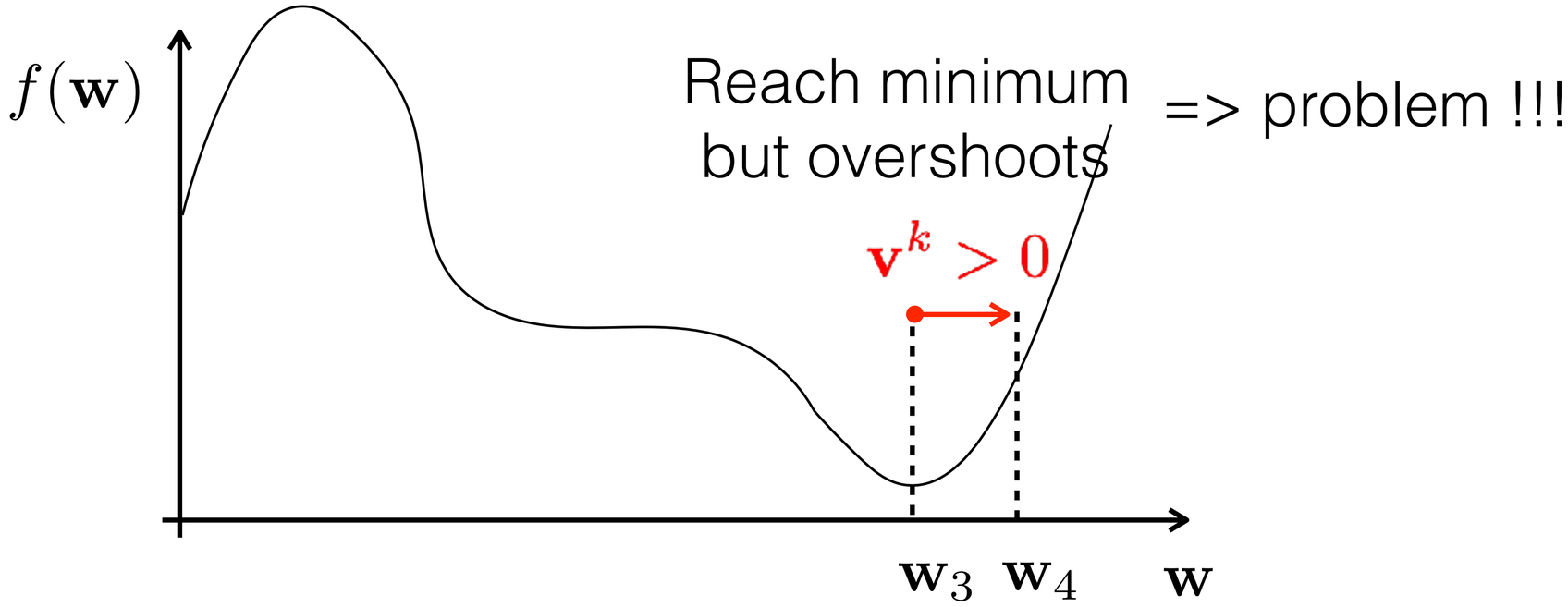



```
torch.optim.SGD(params, lr=0.001, momentum=0.9)
```

SGD + momentum - drawback

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

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

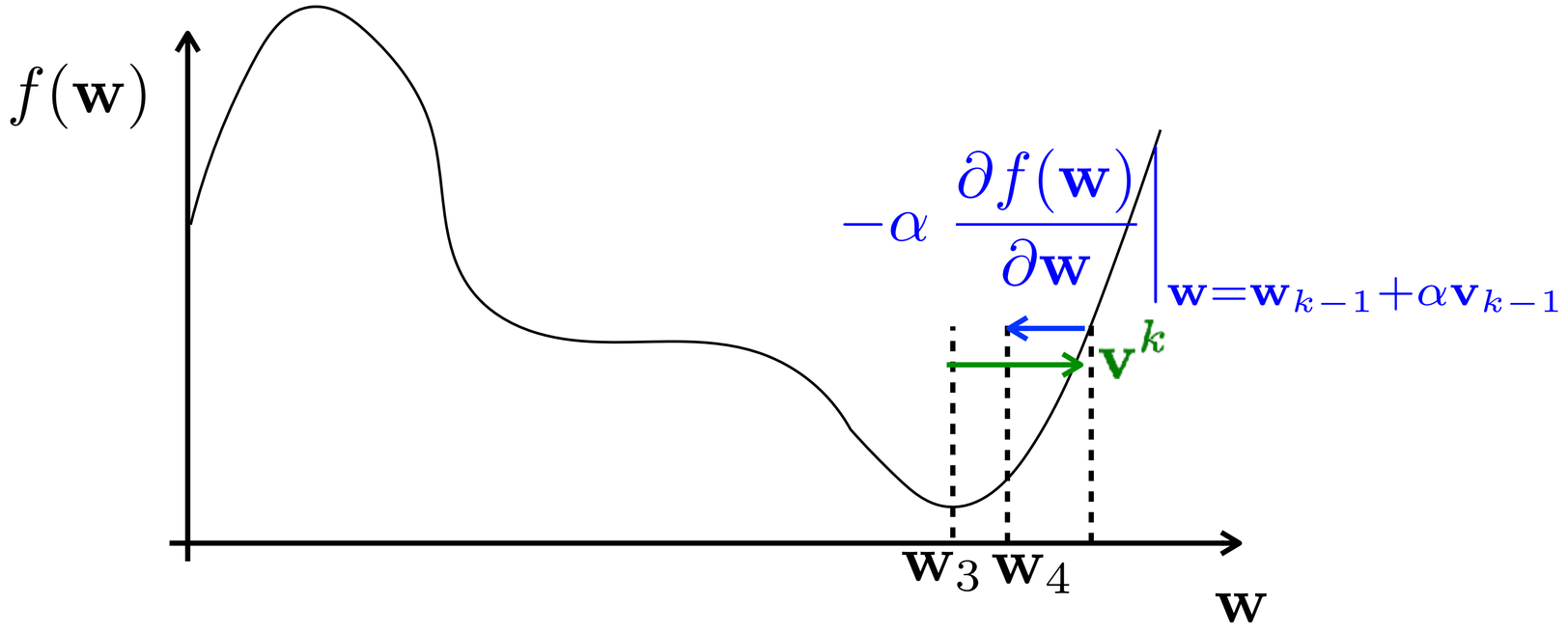


SGD with Nesterov momentum

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

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

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



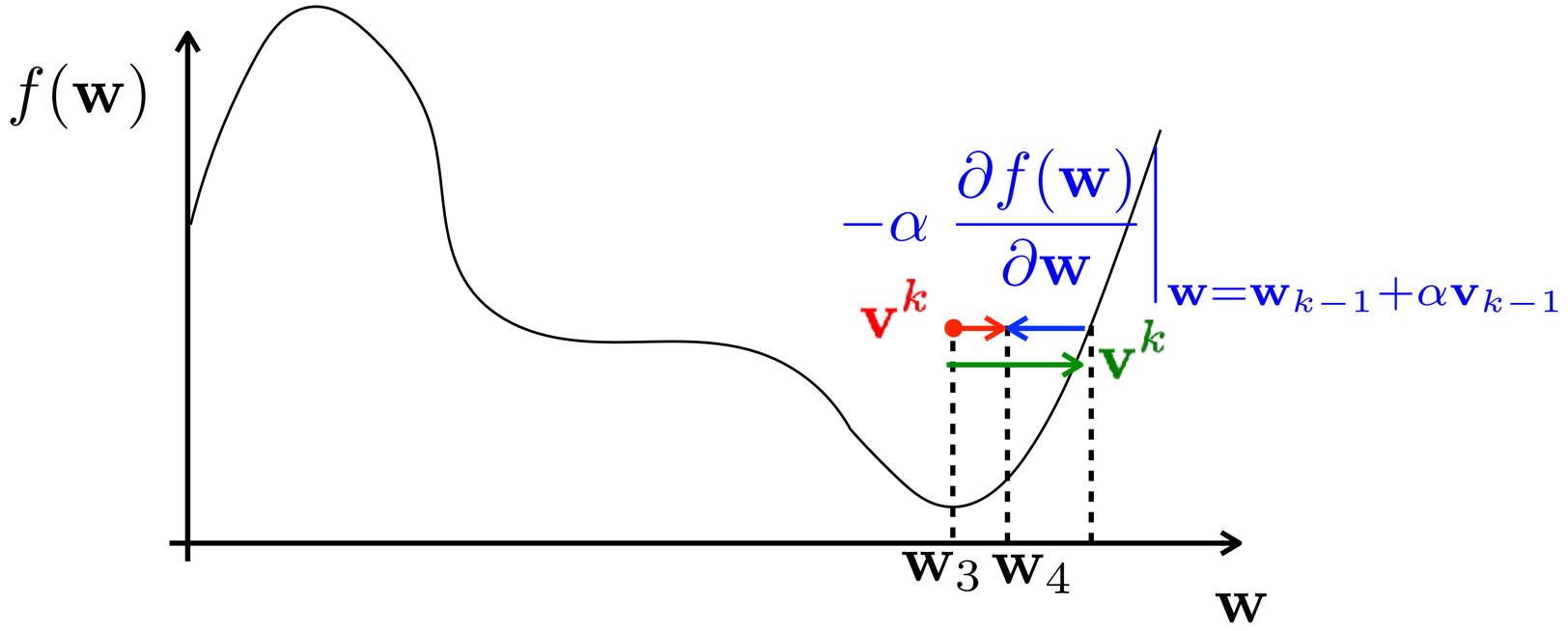
http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

SGD with Nesterov momentum

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

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

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



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

“SGD + momentum” on quadric

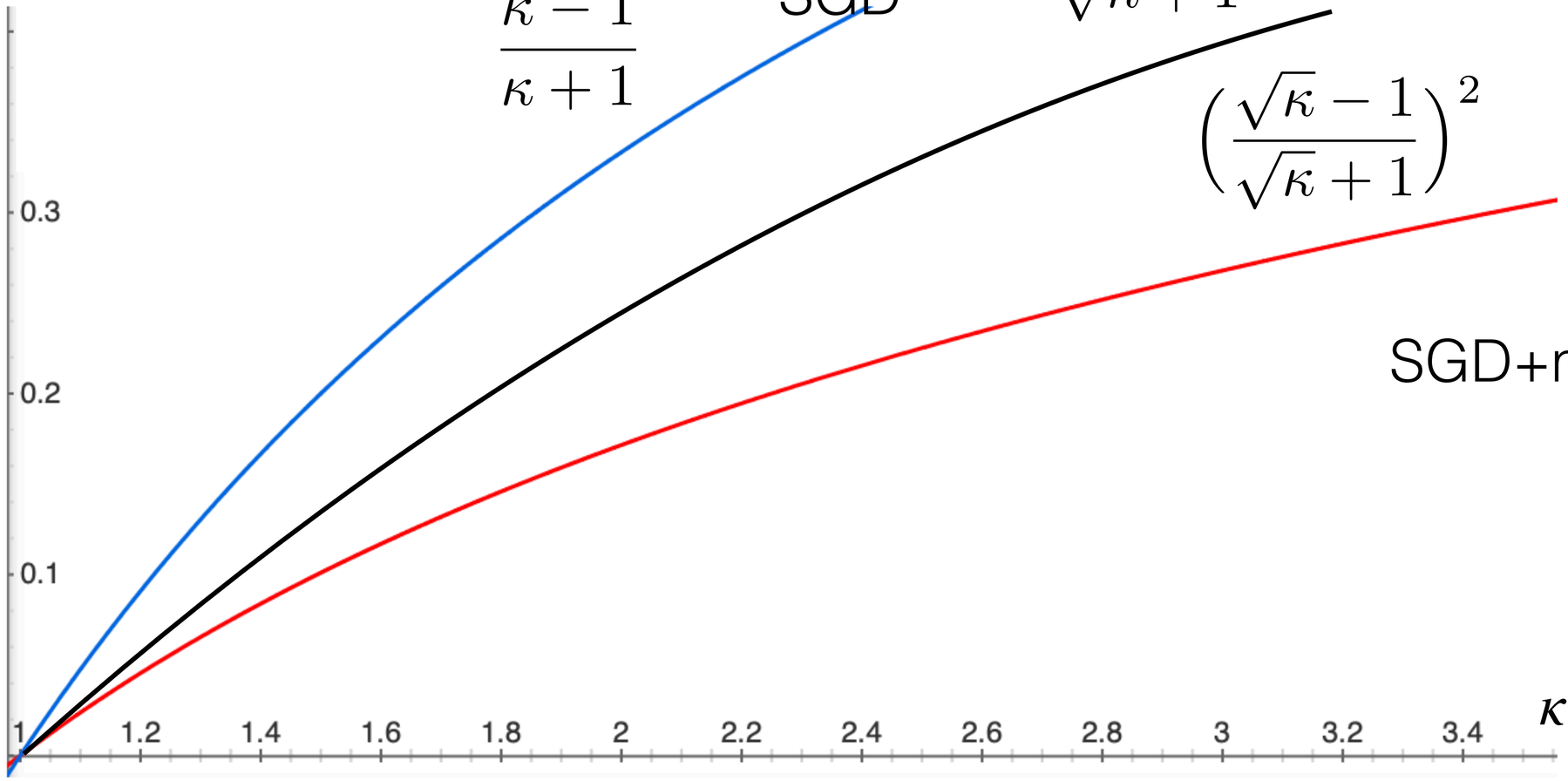
$$\text{rate}(\alpha^*, \beta^*) =$$

Convergence rate

SGD $\frac{\kappa - 1}{\kappa + 1}$

Nesterov SGD+momentum $\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$

SGD+momentum $\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^2$



Summary "SGD with momentum" (SGDM)

friction

ODE solver:
(Euler integrator)

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

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

position

velocity

acceleration
(F_g projected on terrain slope)

Advantages?

- SGDM tends to **overcome sharp minima** and saddle-points due to **momentum**
- SGDM **suppress oscillations** by **averaging out** positive and negative **gradient**
- SGDM is **less sensitive** to large **learning rates**
- SGDM **converges faster** than GD for $\kappa > 1$

Guidelines:

- SGD $\alpha^* = \frac{2}{\lambda_1 + \lambda_n}$

poor condition ($\lambda_1 \approx 0, \lambda_n \approx 100$):

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

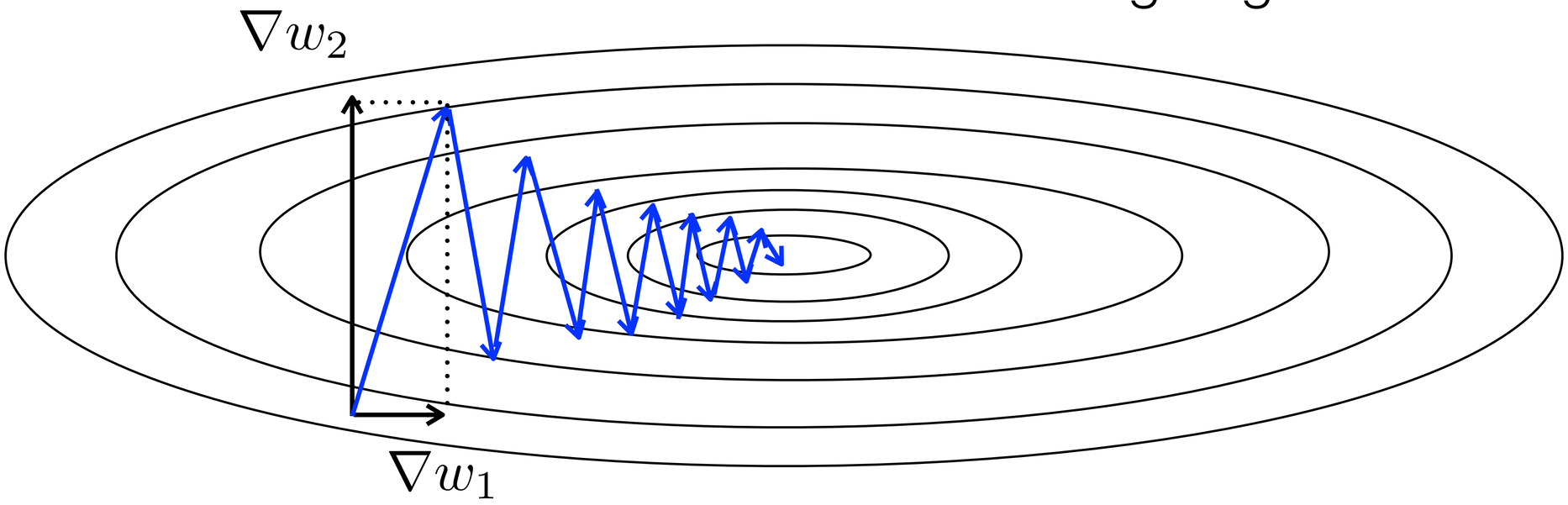
$\Rightarrow \beta^* \approx 1$

$\Rightarrow \alpha^* \approx 2\alpha^*_{\text{SGD}}$

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

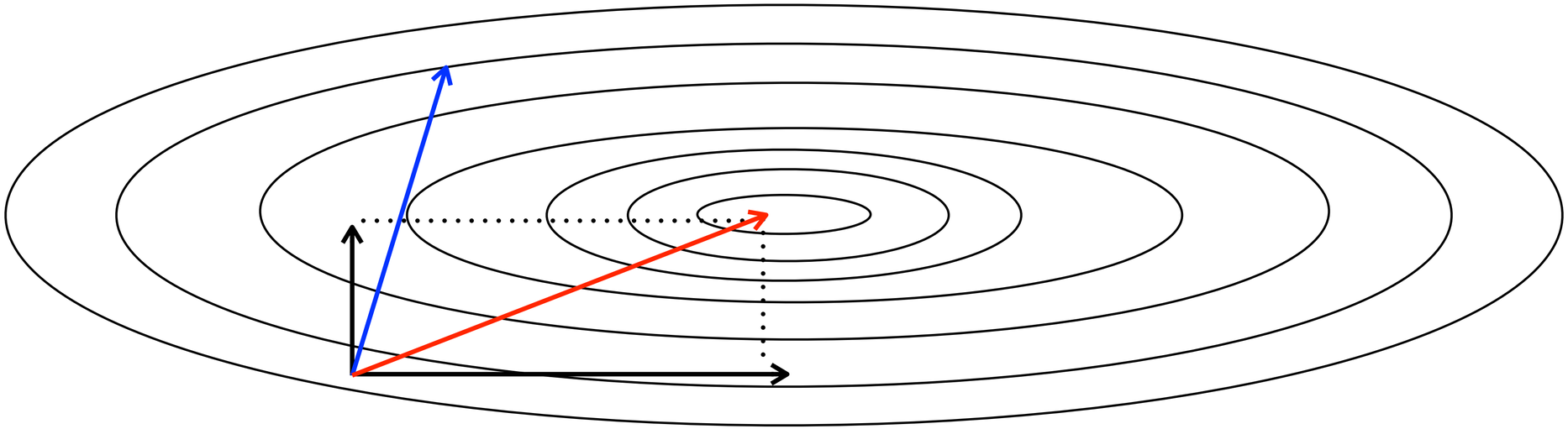


Momentum helps, but the zig-zag behaviour remains.

Newton Method

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

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



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

Convergence rate of Newton method on quadric case study

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

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: $\mathbf{H} = \left.\frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$

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

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

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

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

SGD + momentum on quadric

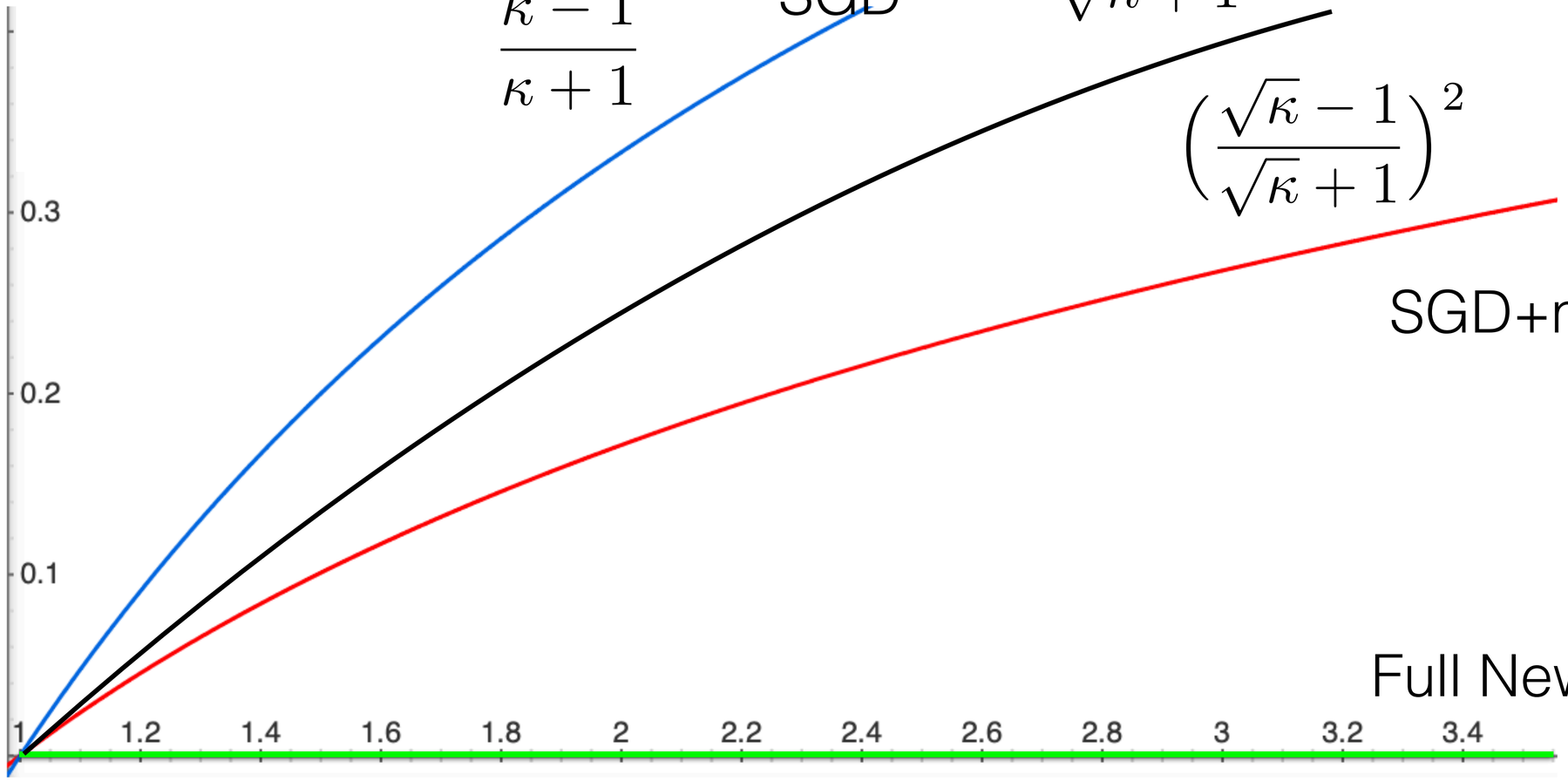
Convergence rate

$\frac{\kappa - 1}{\kappa + 1}$ SGD

$\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$ Nesterov SGD+momentum

$\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^2$ SGD+momentum

Full Newton



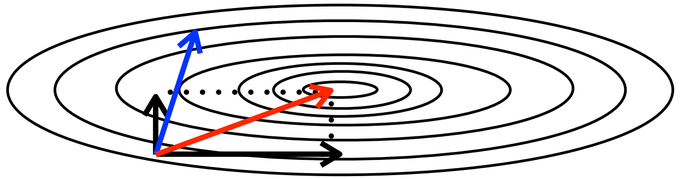
Full Newton Method

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

Why not to use Hessian?

- Hessian has $M \times M$ elements for M-dimensional $\mathbf{w}$ => **memory-consuming**
- Inverse of Hessian is $\mathcal{O}(M^3)$ => **time-consuming**
- **Accurate** estimate of H^{-1} would require significantly **larger minibatches**
- Hessian gives sense only on **convex landscapes**

What does the Hessian actually do?

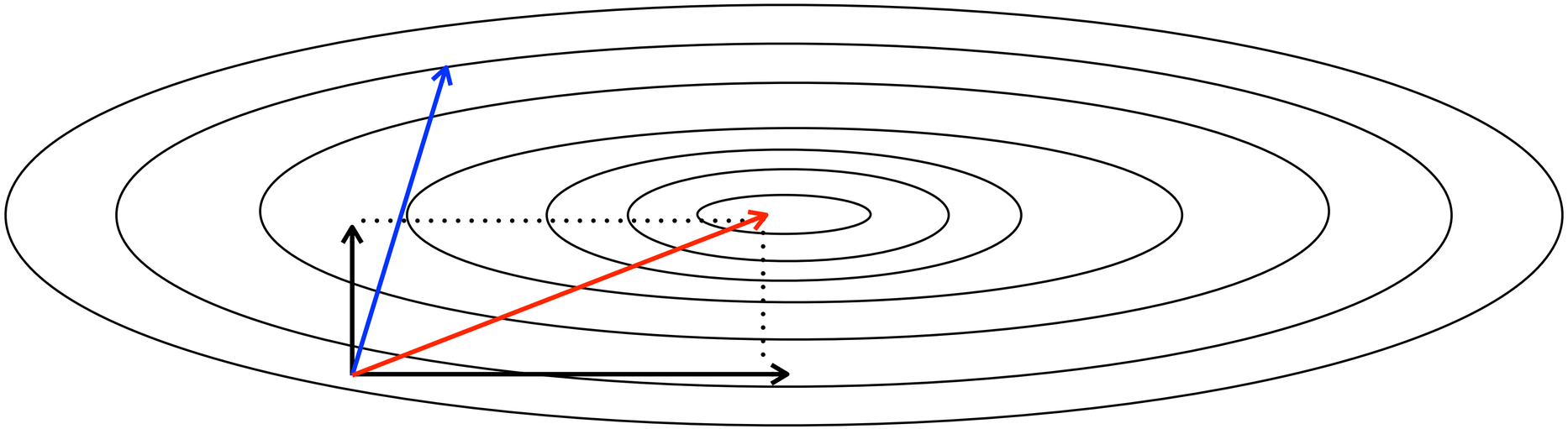


- It **slows down** each **component by** the amount corresponding to its **eigenvalue** (i.e. eigenvalue encodes steepness of the quadric in particular dimension)
- The faster the change the shorter the step

Full Newton Method

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

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



What does the Hessian actually do?

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

Full Newton Method

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

$\mathbf{g}$

$$\hat{H} = \begin{bmatrix} \sqrt{\mathbf{g}_1^2} & 0 & \dots & 0 \\ 0 & \sqrt{\mathbf{g}_2^2} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \sqrt{\mathbf{g}_n^2} \end{bmatrix}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \hat{\mathbf{H}}^{-1} \cdot \mathbf{g}$$

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

Full Newton method - approximation

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

Exponential averaging (momentum on $\mathbf{g}^2$)

$\mathbf{q}^k$

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

RMSprop

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

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

Exponential averaging (momentum on $\mathbf{g}^2$)

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

Adam = Adaptive + momentum in $\mathbf{g},\ \mathbf{g}^2$

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

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

$\mathbf{v}^k$

Exponential averaging (momentum on $\mathbf{g}$)

Adam = Adaptive + momentum in $\mathbf{g},\ \mathbf{g}^2$

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

Exponential averaging (momentum on $\mathbf{g}$)

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

Exponential averaging (momentum on $\mathbf{g}^2$)

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

$\mathbf{v}^k, \mathbf{q}^k$ initialized in zero => slow start

[Kingma ICLR 2015]

Adam = Adaptive + momentum in $\mathbf{g},\ \mathbf{g}^2$

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

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

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

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

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

Exponential averaging (momentum on $\mathbf{g}$)

Exponential averaging (momentum on $\mathbf{g}^2$)

Correction of the slow start due to zero init.

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)
```

AdamW = Adam + Weight-decay

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

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

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

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

$$\hat{\mathbf{w}}^{k-1} = \mathbf{w}^{k-1} - \alpha\lambda\mathbf{w}^{k-1}$$

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

Exponential averaging (momentum on $\mathbf{g}$)

Exponential averaging (momentum on $\mathbf{g}^2$)

Correction of the slow start due to zero init.

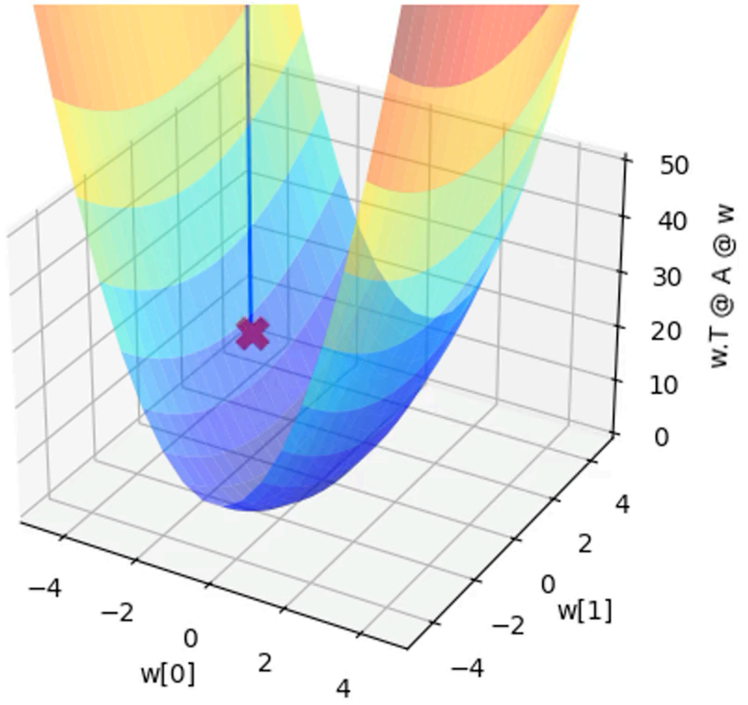
Decoupled weight-decay

```
torch.optim.AdamW(params, lr=0.001, betas=(0.9, 0.999),
                  eps=1e-08, weight_decay=0.01, …
```

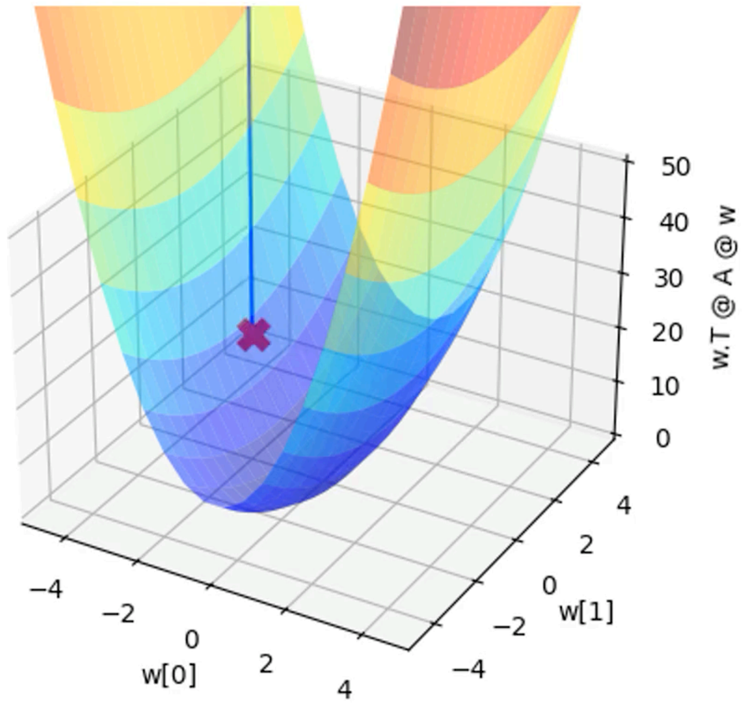
SGD vs SGD momentum

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

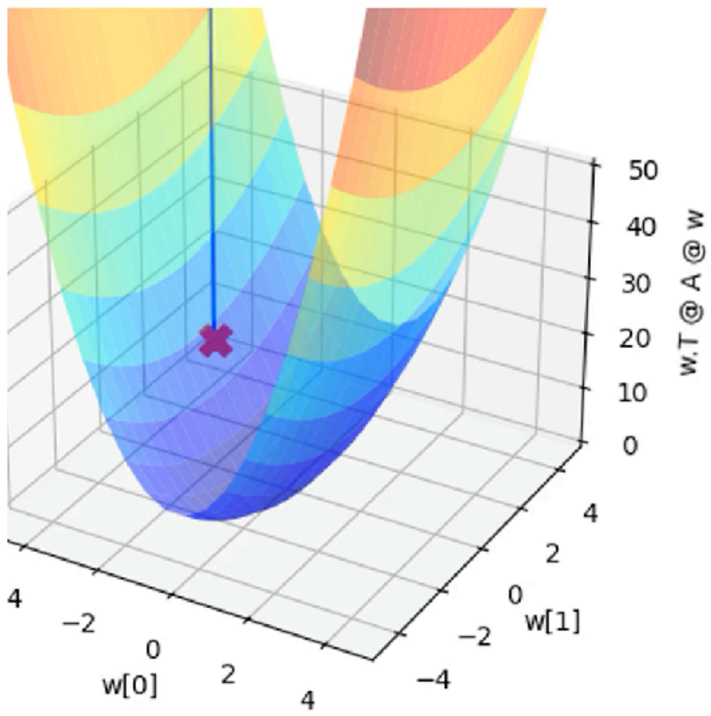
SGD



SGD + momentum



Adam

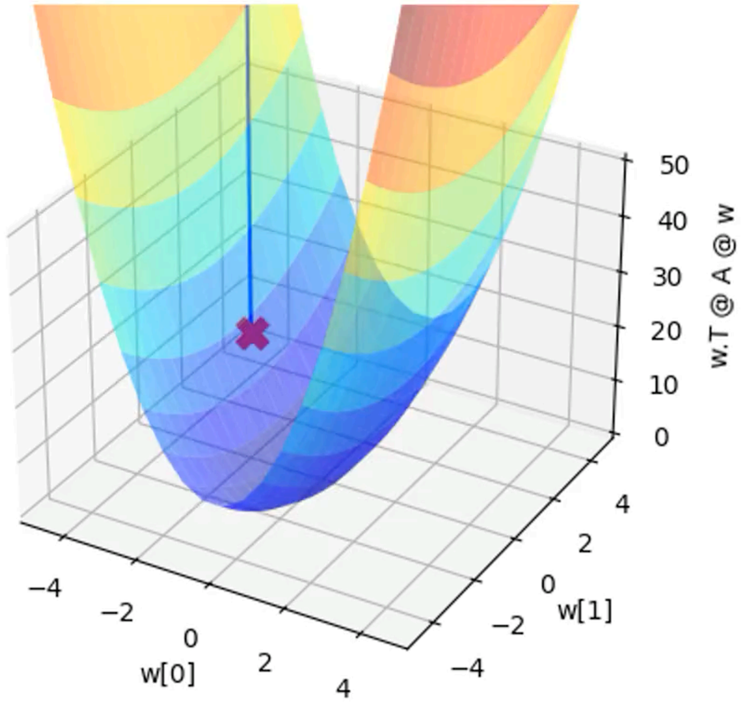


lr=0.25

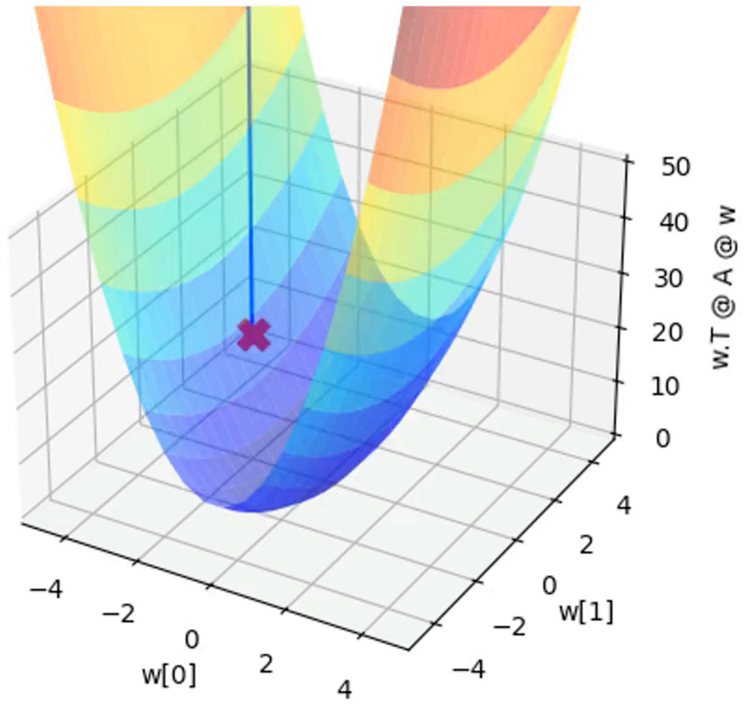
SGD vs SGD momentum

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

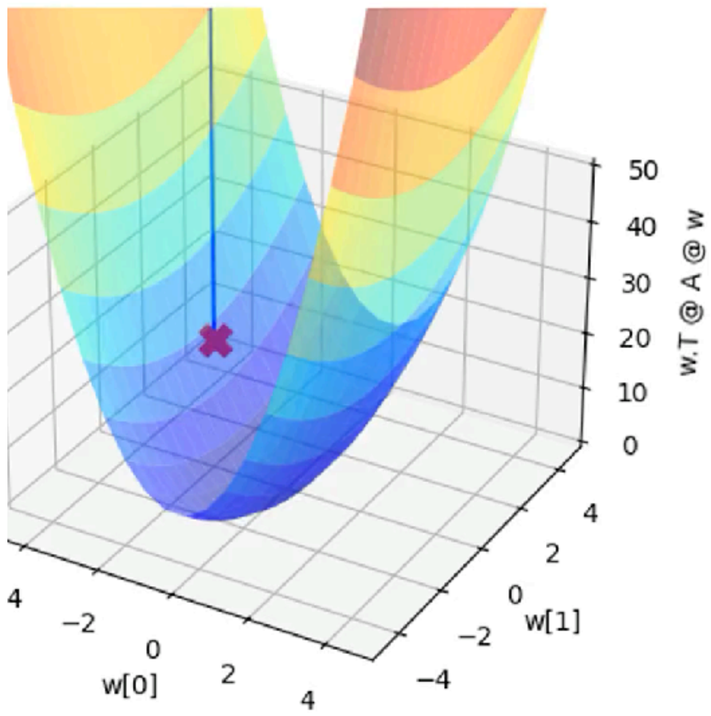
SGD



SGD + momentum



Adam



lr=0.2

Summary

- **AdamW** is the most **popular** choice, since it is that **insensitive** to the choice **hyper-parameters**.
- PyTorch of all previously mentioned implementations available:

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

- One more hack: If something is too big, you can always … ???
  - Gradient clipping
- Anything **more complex** than AdamW typically **suffers** from diminishing improvements in **iteration quality** while **increasing** in computational **time.** 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?

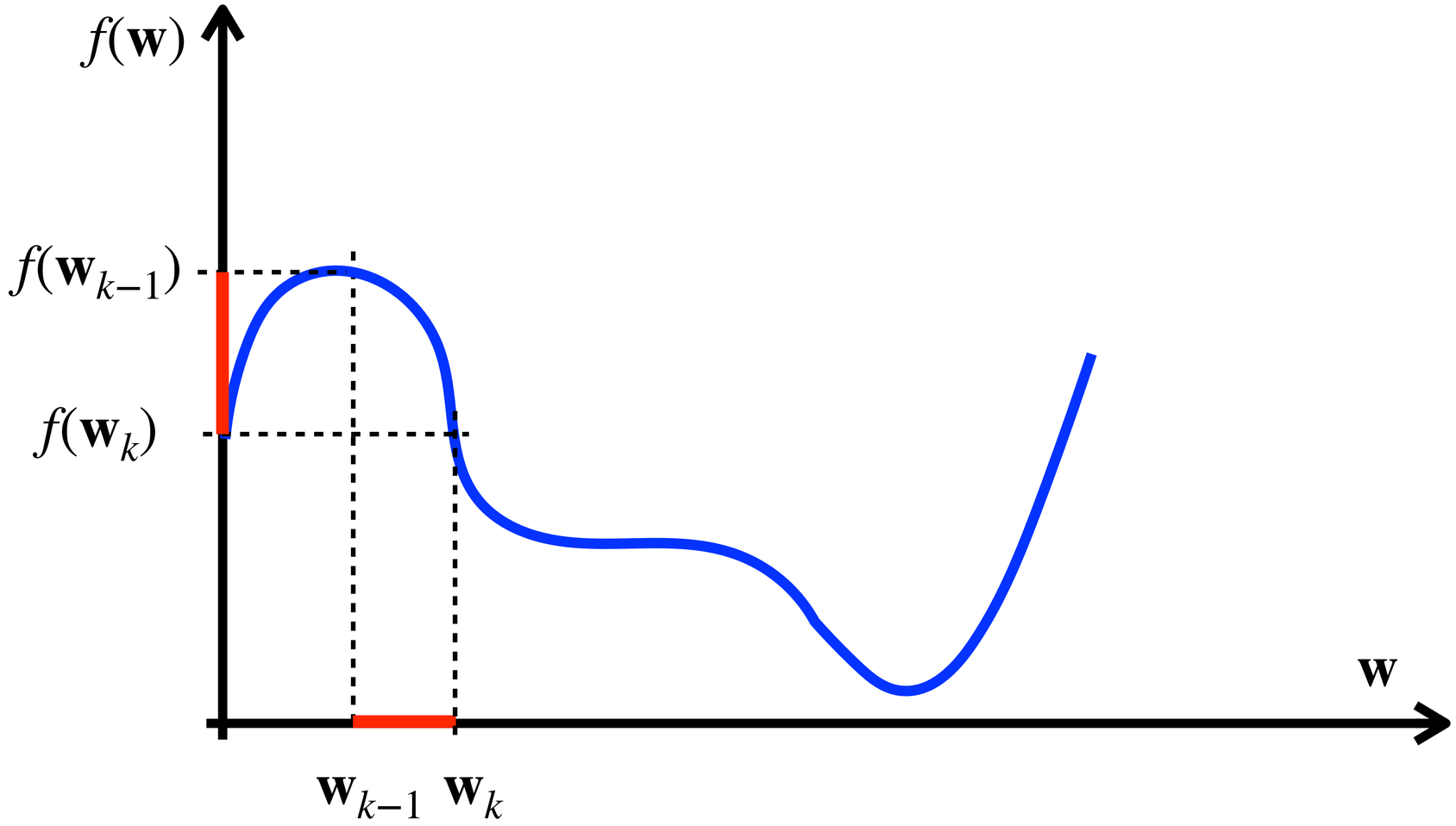
Broyden **F**letcher **G**oldfarb

BFGS: Where does the name came from?

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

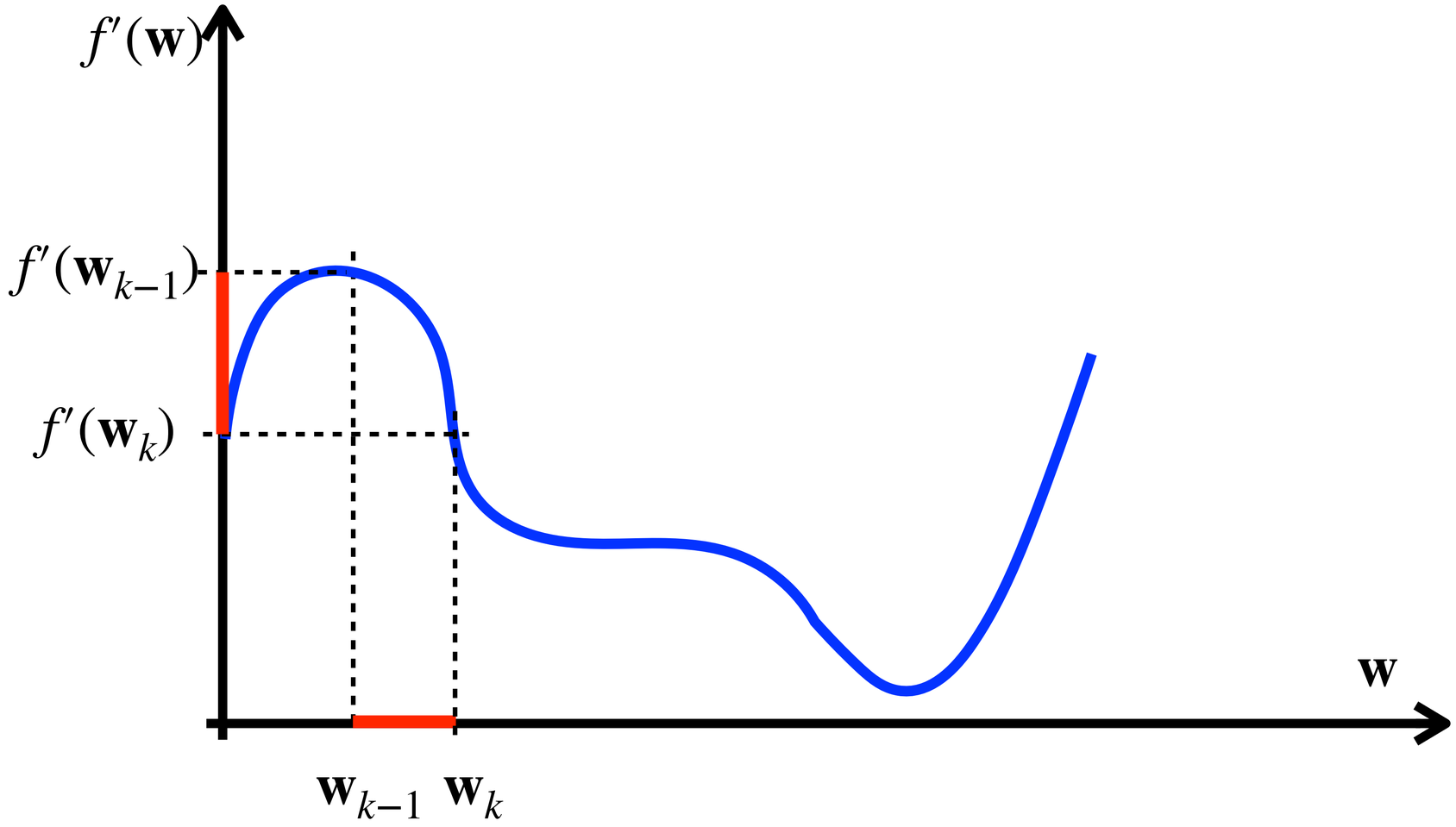
Broyden **F**letcher **G**oldfarb **S**hanno

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



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

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

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

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

$$\text{subject to : } \hat{\mathbf{H}}_k = \hat{\mathbf{H}}_k^\top$$ ..... 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 **first order** methods suffers from **slow** convergence:

Efficient for:

- **Energy minimization** (sum of squares functions + additional non-lin terms)

  $E(x) = \sum_i \|x_i - y_i\|^2 + \lambda \sum_i \|\nabla x_i\|_1$ TV methods: denoise, deblur, reconstr.
- **NLS with ill-conditioned jacobian** (better than LM on non-lin least-squares)

Applications:

- **Computer graphics** (mesh fitting with structural priors, cloth dynamic)
- **Computer vision** (optical flow, froreground/background segmentation)
- **Structural design** (opt structure/material bridges, buildings, aerospace compon.)
- **Process optimization** (chemistry)
- **Tuning control system** (automotive, aerospace, aircrafts, powerplants)

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 : $\hat{H} = \begin{bmatrix} \overline{g}_1 & 0 & \dots & 0 \\ 0 & \overline{g}_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \overline{g}_n \end{bmatrix} \quad \Rightarrow \mathbf{x} = \hat{H}^{-1}\mathbf{g}$

Full Newton

BFGS

Adam

- Optimization vs Learning

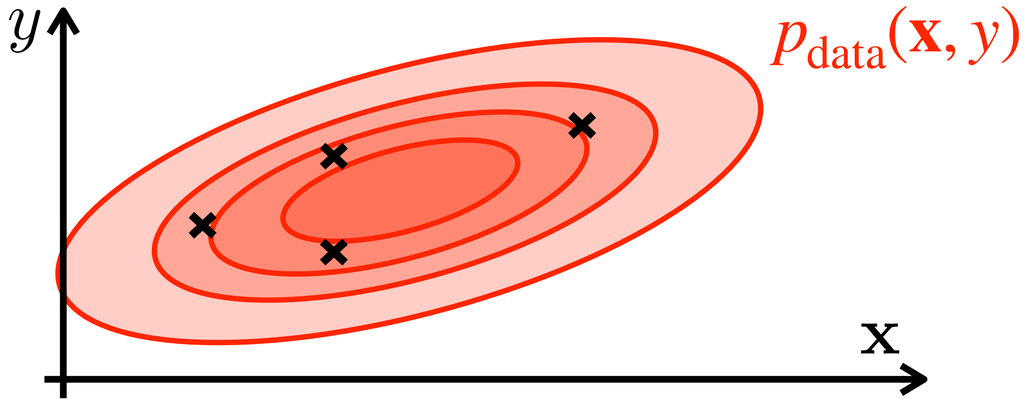
$$J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\log p(\mathbf{x}, y \mid \mathbf{w})\right]$$

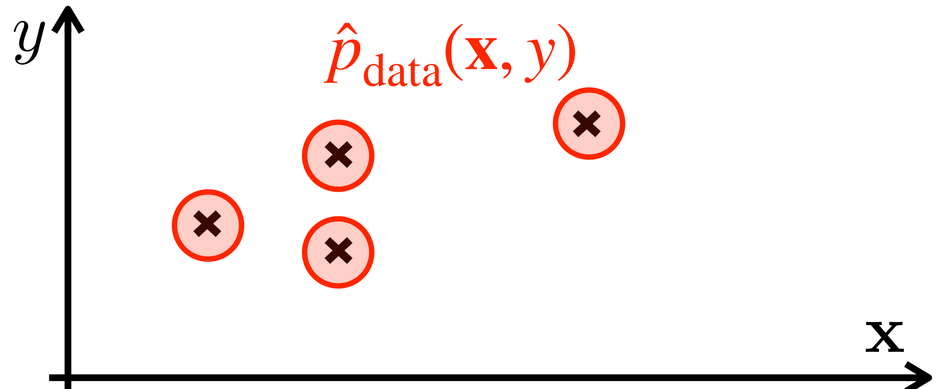
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\nabla_{\mathbf{w}} \log\left(p(\mathbf{x}, y \mid \mathbf{w})\right)\right]$$

vs.

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

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



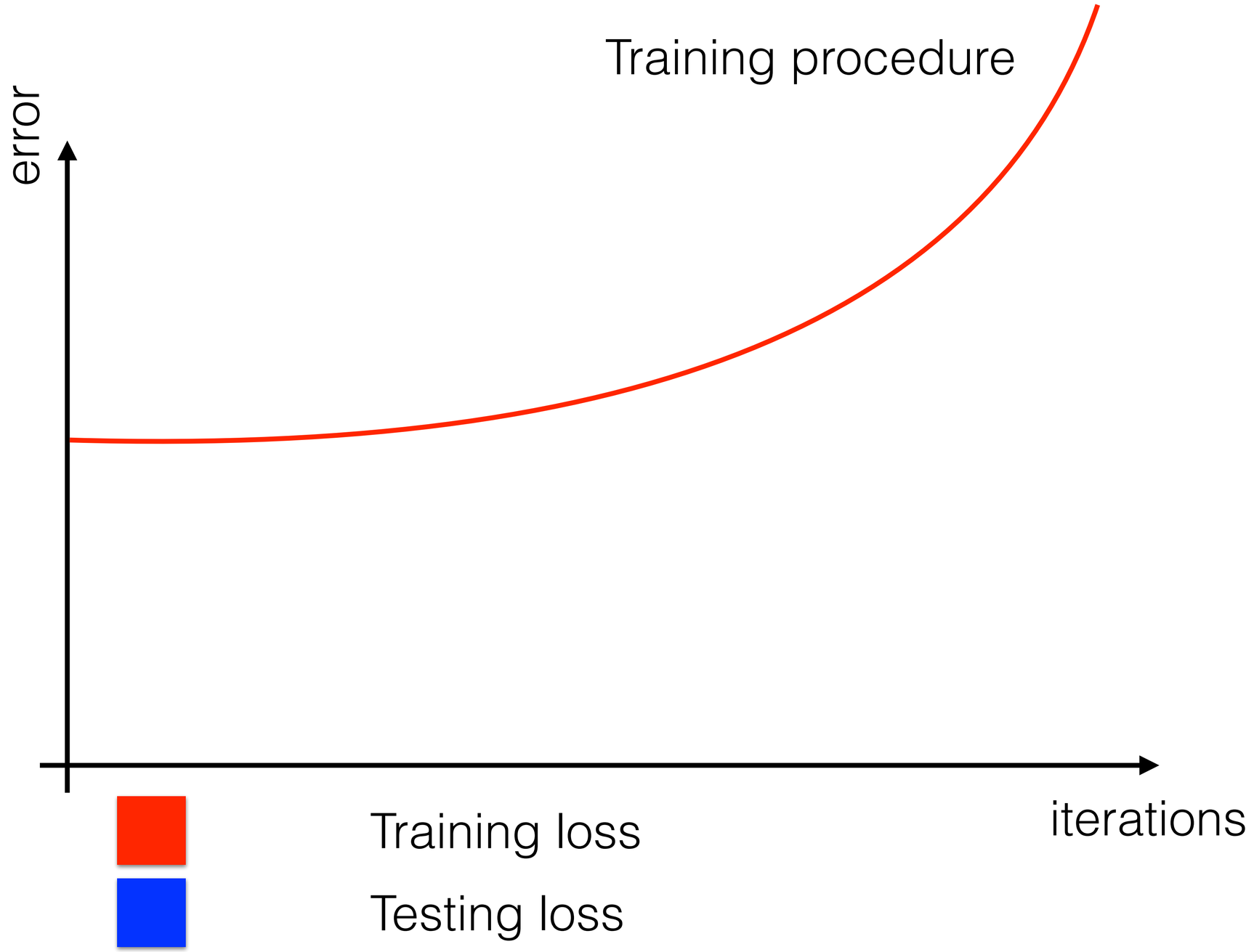


Epilog

- Stochasticity advantage:
  - **reduces** computational **time** and **memory** due to mini-batches
  - works as an **regularizer** that helps in larger models due to inherent noise
  - **avoids** getting stuck in **saddle points** due to stachasticity + geom.div.concave
- Stochasticity drawback:
  - makes estimation of **gradient** and **Hessian inaccurate**
    (Standard deviation $\approx 1/\sqrt{N}$ => suffers from sub-linear returns)
  - **smaller batches N + higher dimensions M prevents** advanced **Hessian approx** methods (LBFGS) =>Adam is typically used
- Momentum advantage:
  - **jumps** over **sharp minima**
  - **avoid getting stuck** in **flat regions**
  - **suppresses oscillations**
- Adaptivity advantage:
  - **suppresses oscillations** even more
- Adaptivity drawback:
  - **increases chance** to reach **sharp minimum**

Training procedure

- Choose:
  - Network **architecture** (ideally re-use pre-trained net)
  - Weight **initialization** (Xavier)
  - **Loss** + regularization
  - **Hyper-parameters** (learning rate, momentums, weight-decay, …)
- 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)

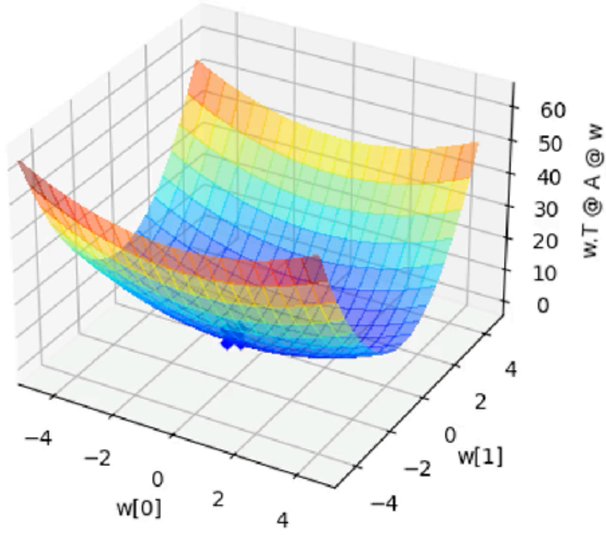

Training procedure
error
iterations
Training loss
Testing loss

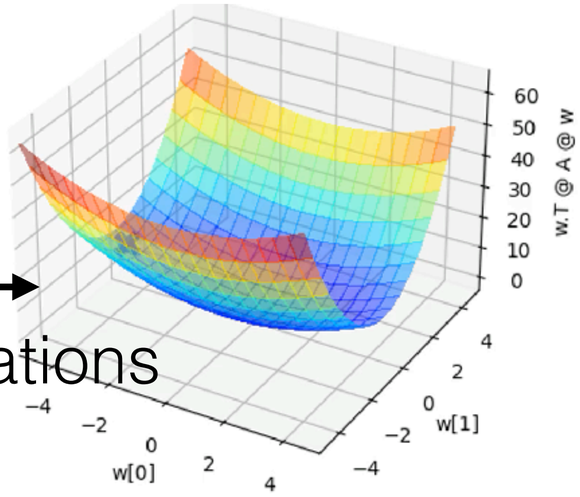
Training procedure

error

Trn loss explodes to infinity => **oscillations**

- decrease the learning rate

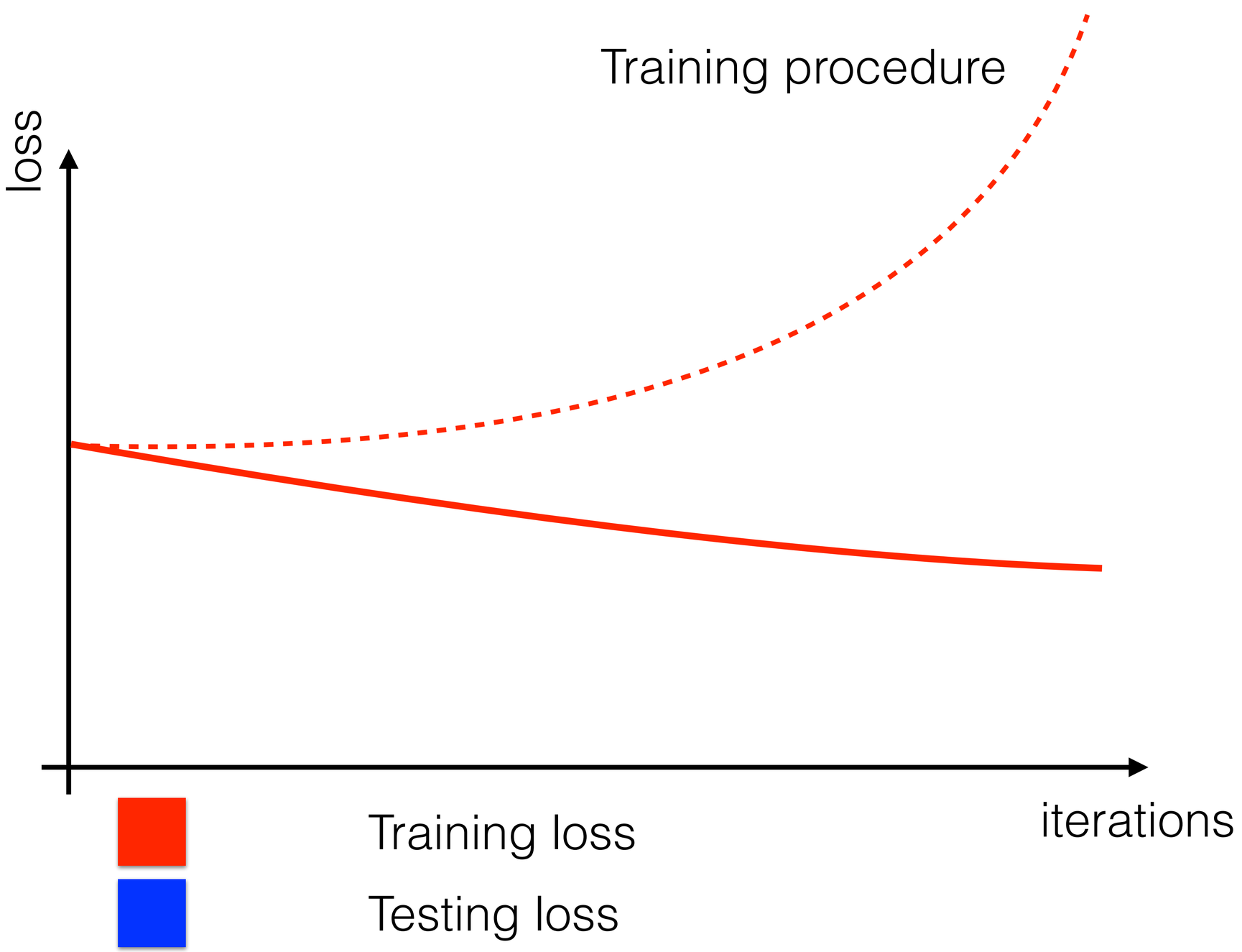




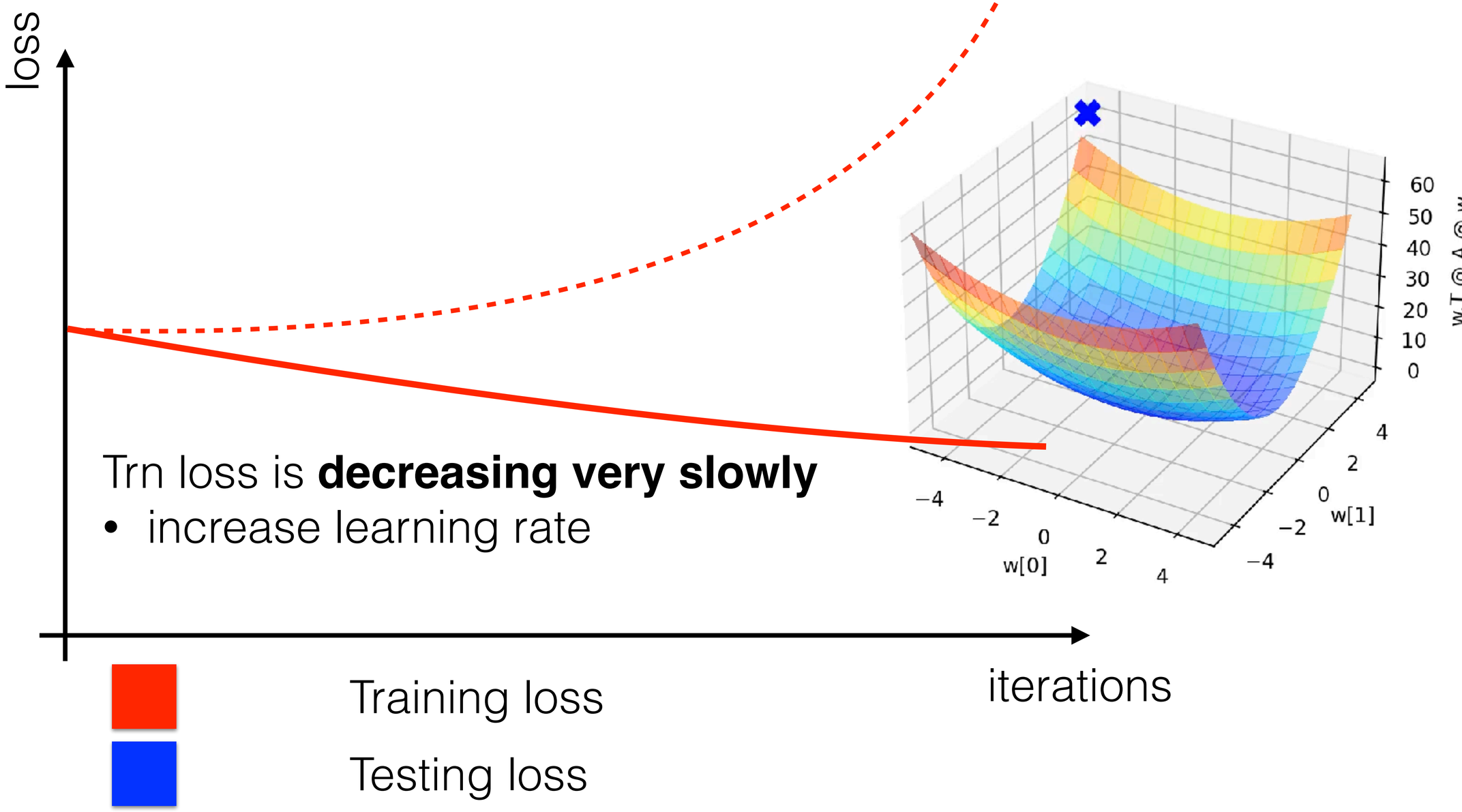
iterations

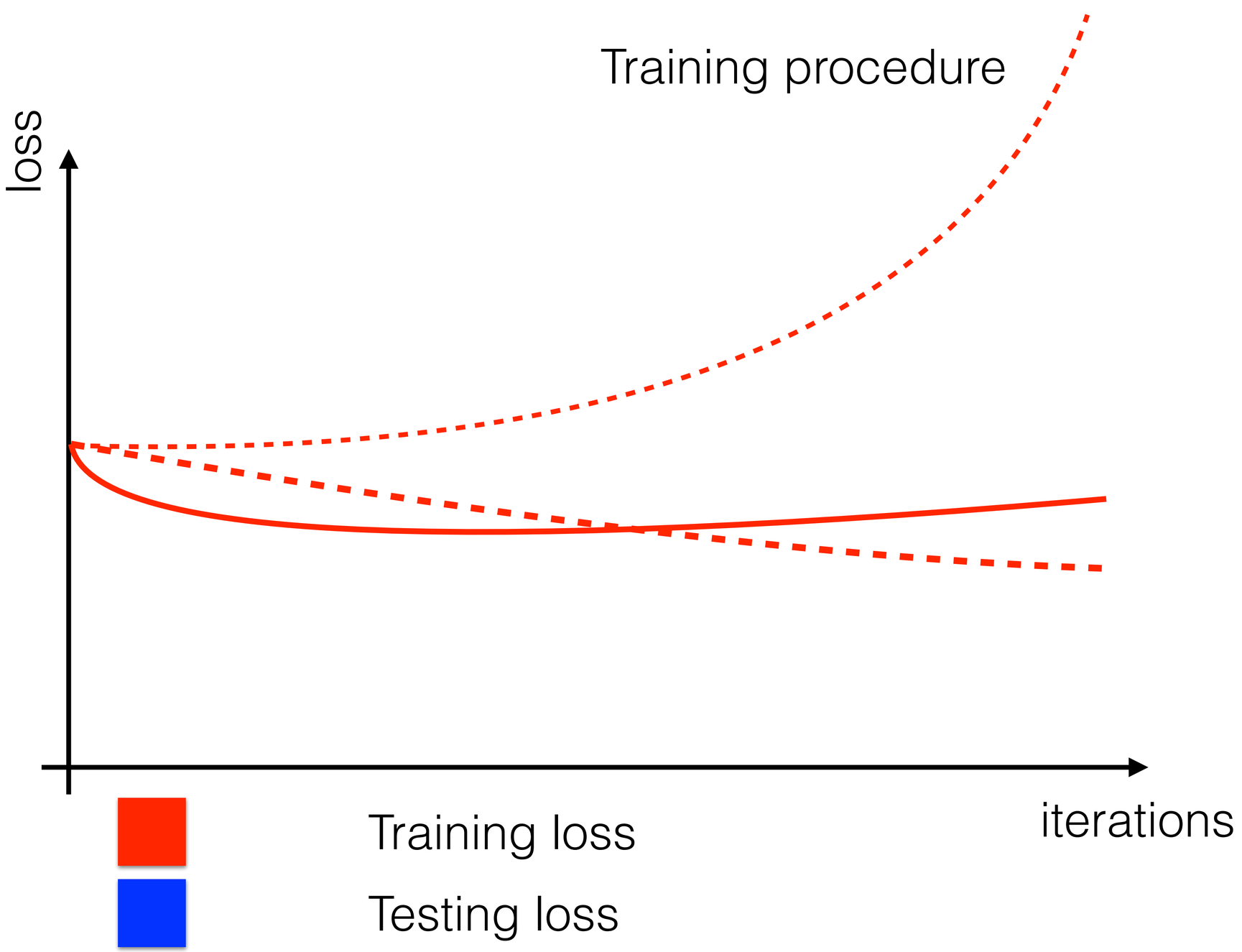
Training loss

Testing loss


Training procedure
loss
iterations
Training loss
Testing loss

Training procedure



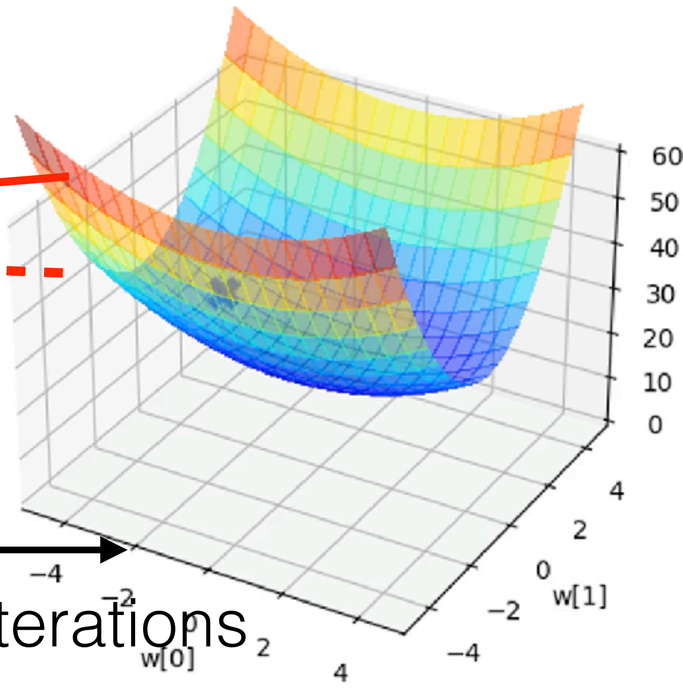

Training procedure
loss
iterations
Training loss
Testing loss

Training procedure

loss

Trn loss **remains huge** =>**underfitting**

- decrease regularization strength
- increase model capacity

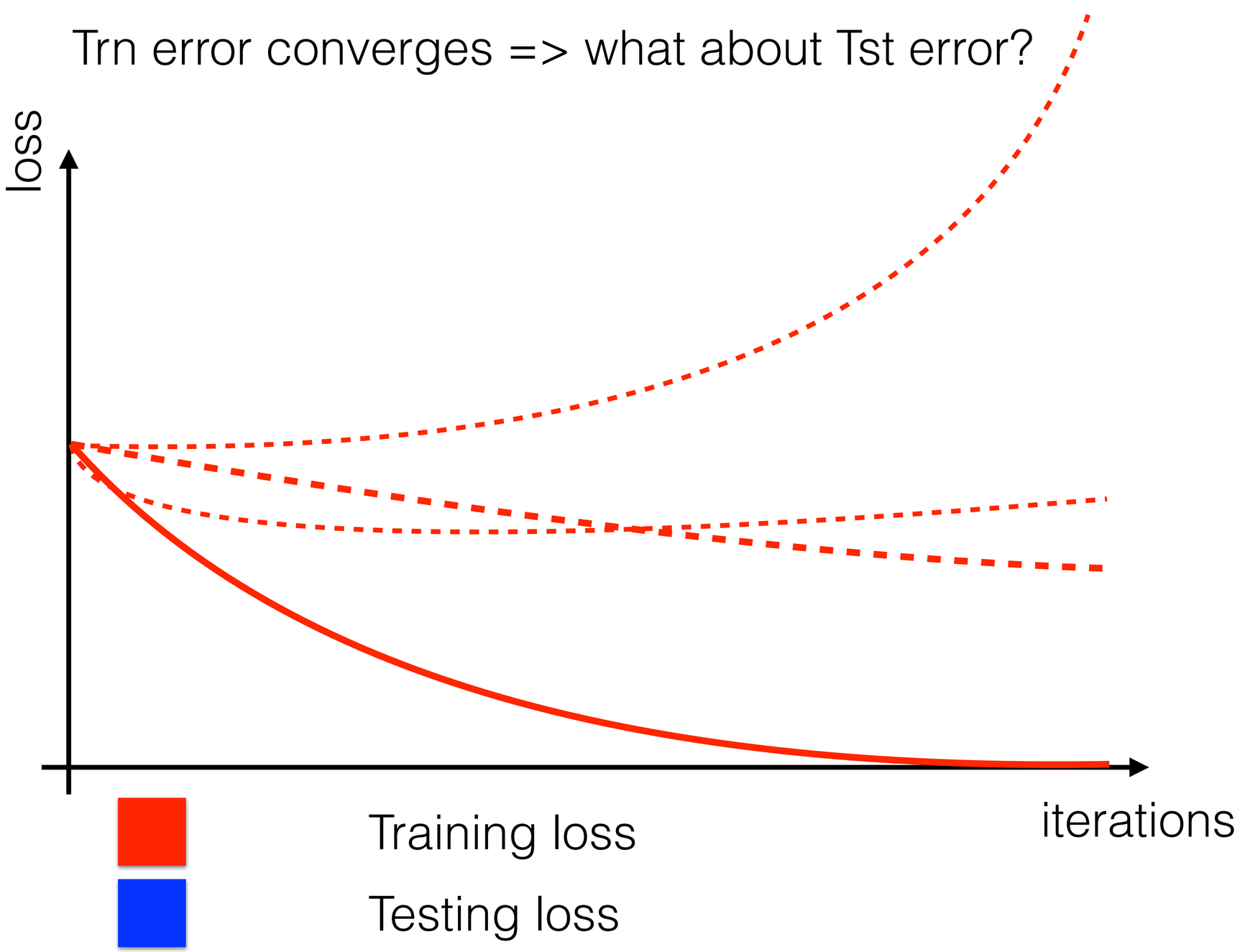


iterations

Training loss

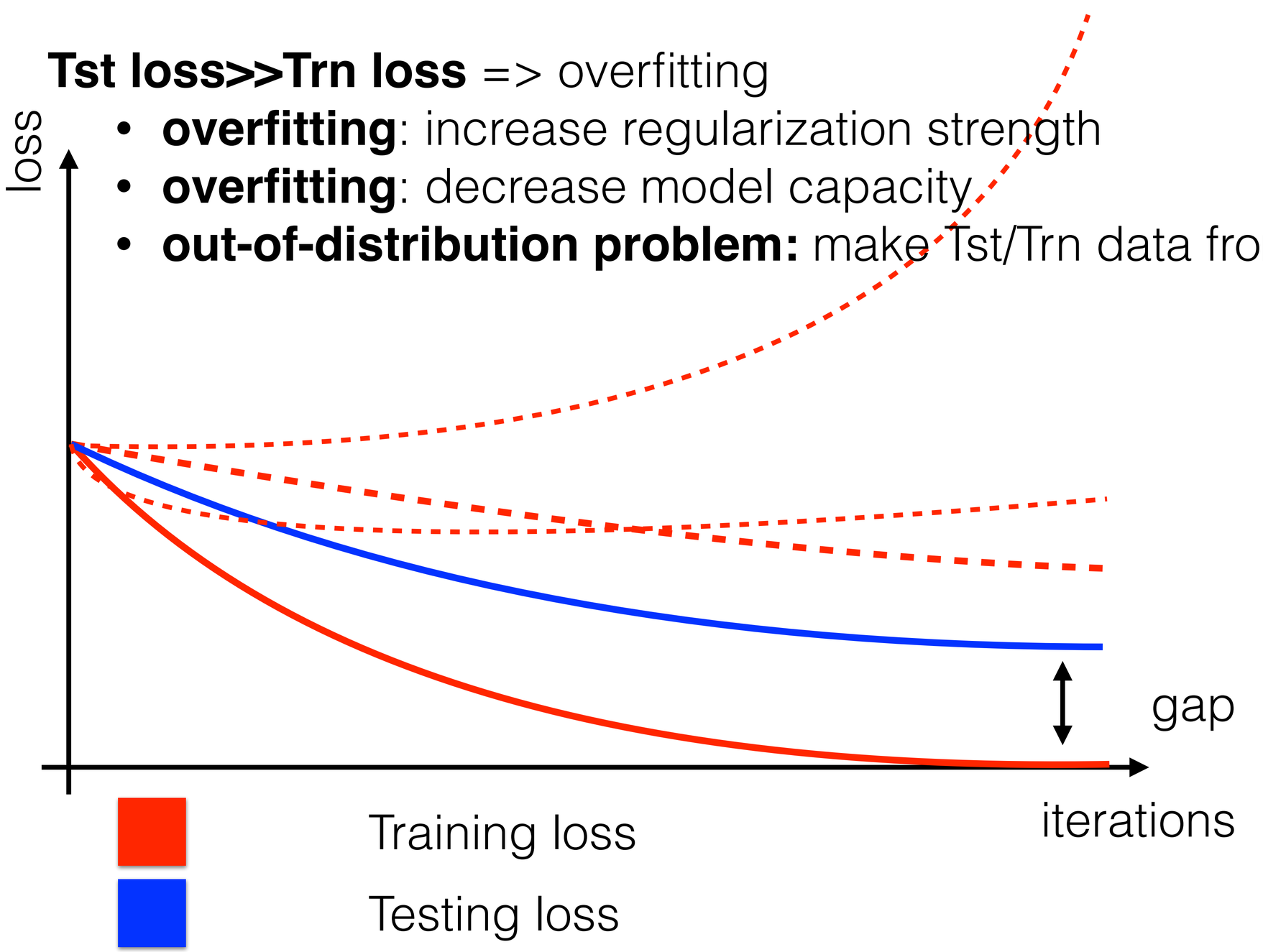
Testing loss

Trn error converges => what about Tst error?



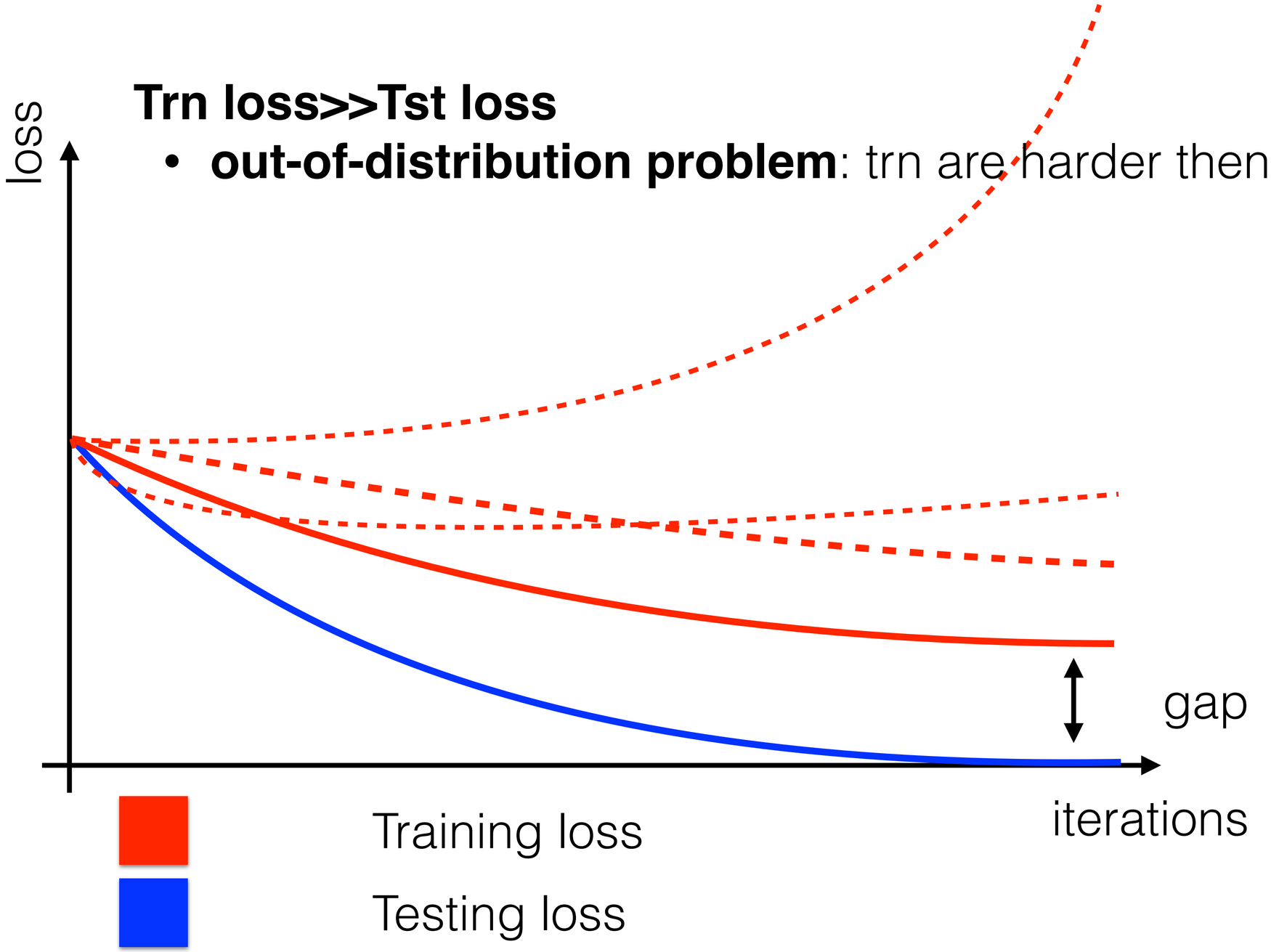
Tst loss>>Trn loss => overfitting

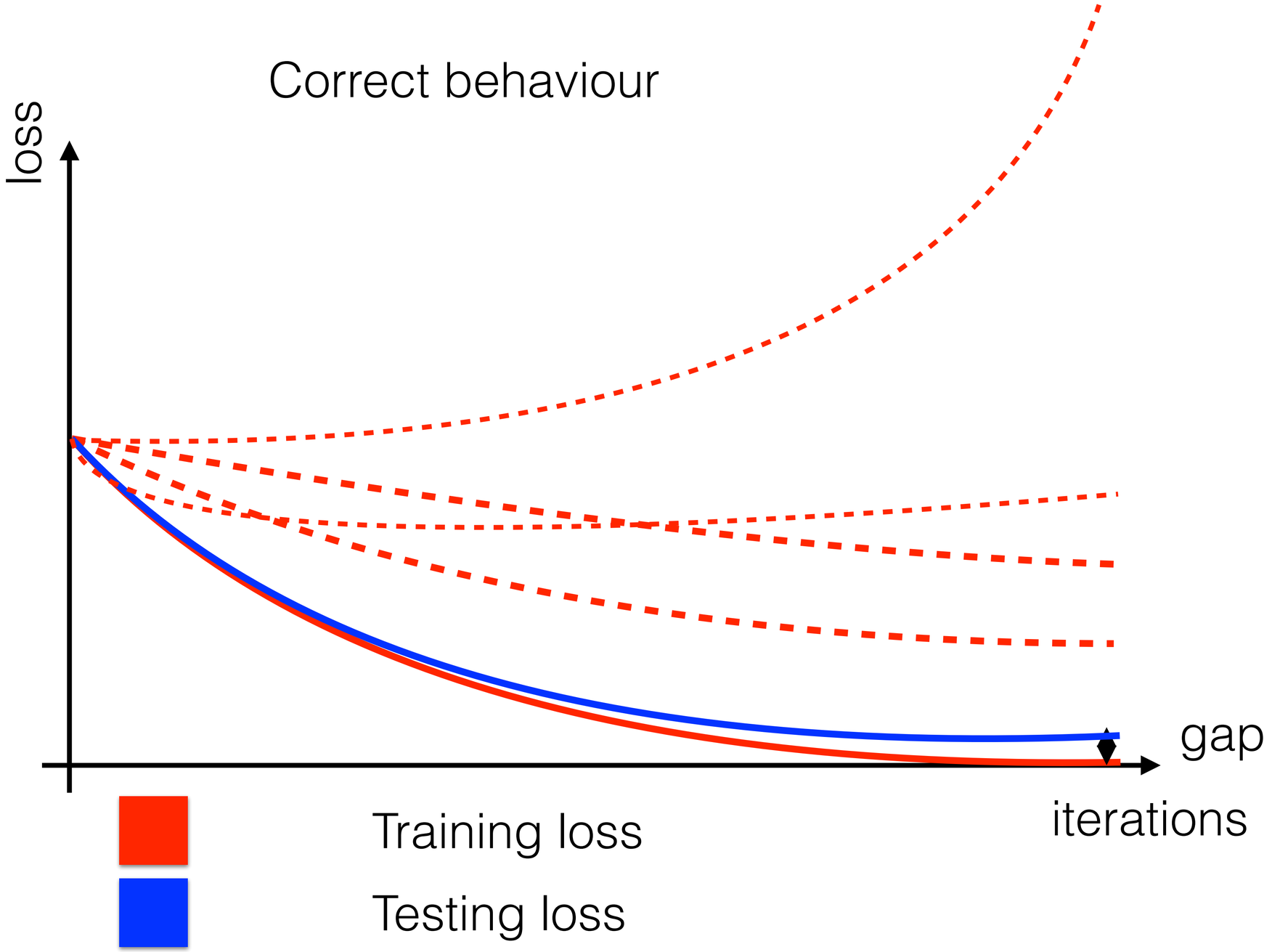
- **overfitting**: increase regularization strength
- **overfitting**: decrease model capacity
- **out-of-distribution problem:** make Tst/Trn data from same distr.



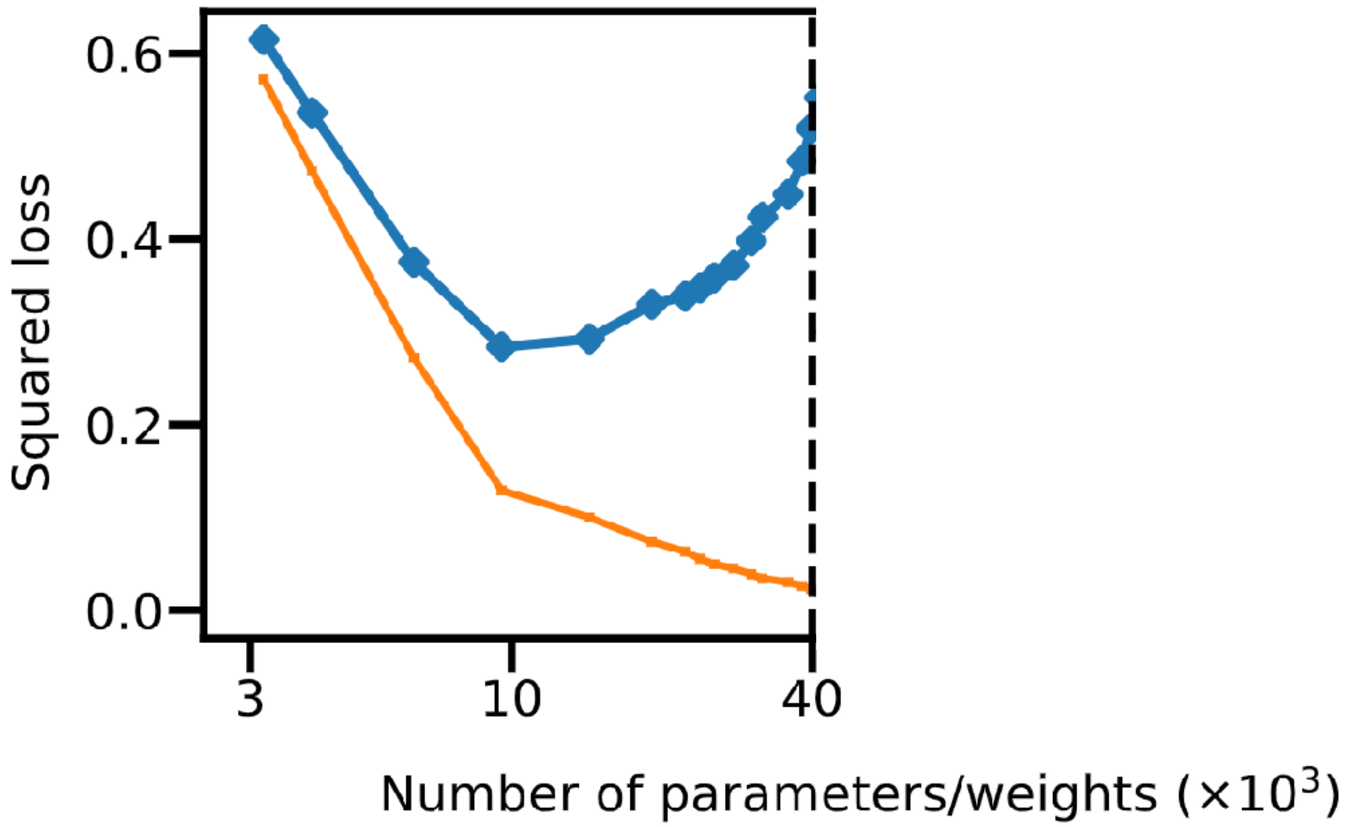
Trn loss>>Tst loss

- **out-of-distribution problem**: trn are harder then tst




Correct behaviour
loss
gap
iterations
Training loss
Testing loss

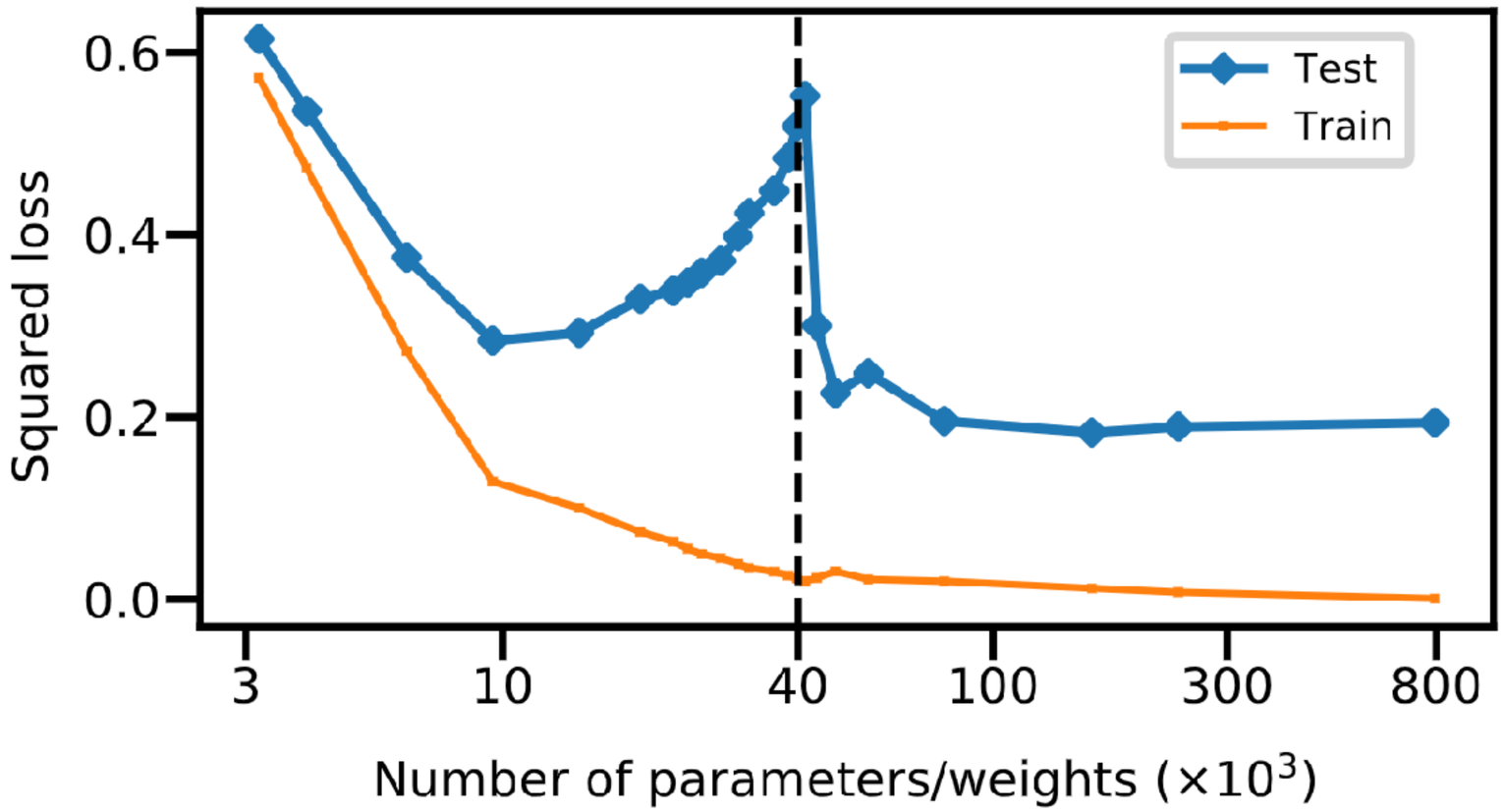
Double descent
[Belkin-DoubleDescent-2019]



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

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

Double descent
[Belkin-DoubleDescent-2019]

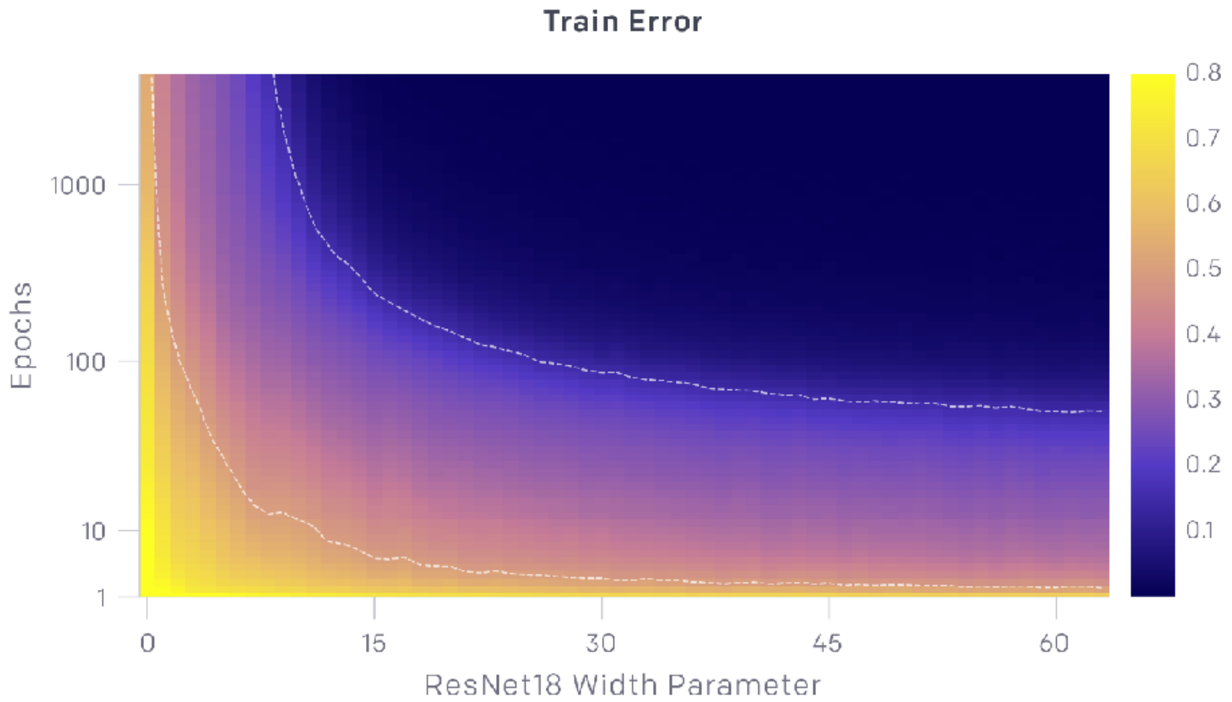


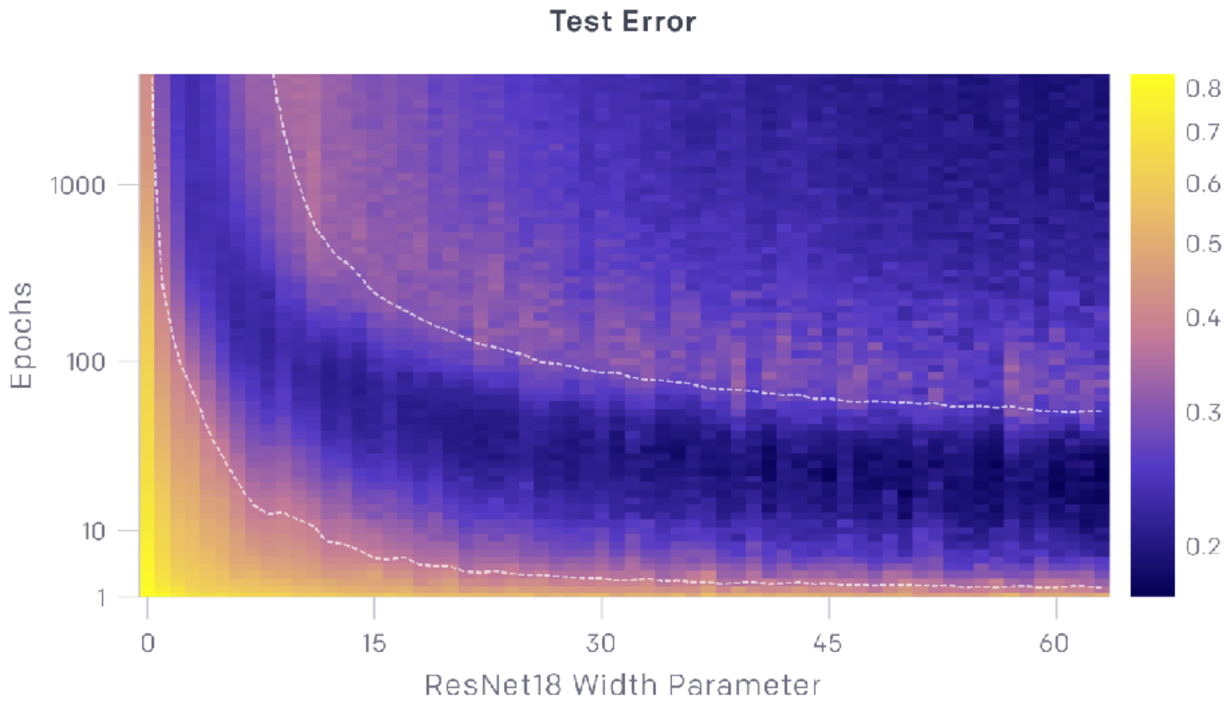
Statistical wisdom: “Too large models are worse since they overfit.”

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/