Training

Karel Zimmermann

http://cmp.felk.cvut.cz/~zimmerk/

Vision for Robotics and Autonomous Systems
https://cyber.felk.cvut.cz/vras/

Center for Machine Perception
https://cmp.felk.cvut.cz

Department for Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

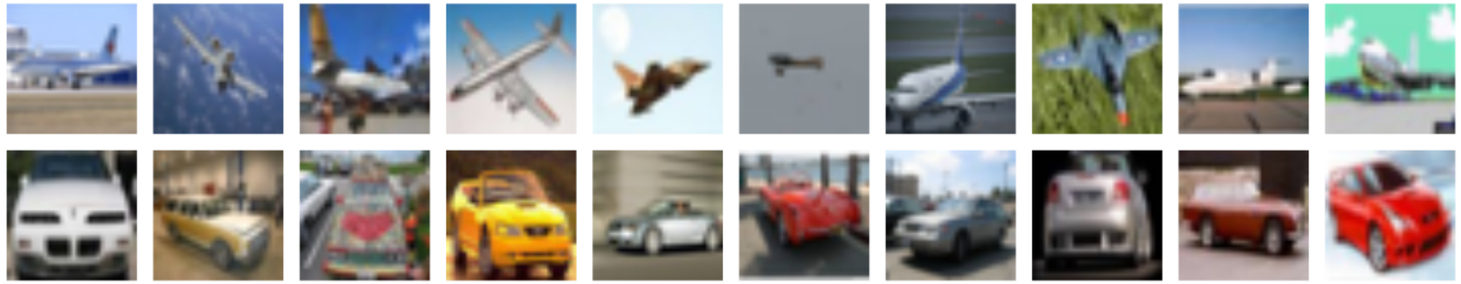
Linear classifier and neuron

Labels

RGB images

$+1$

-1



def classify():

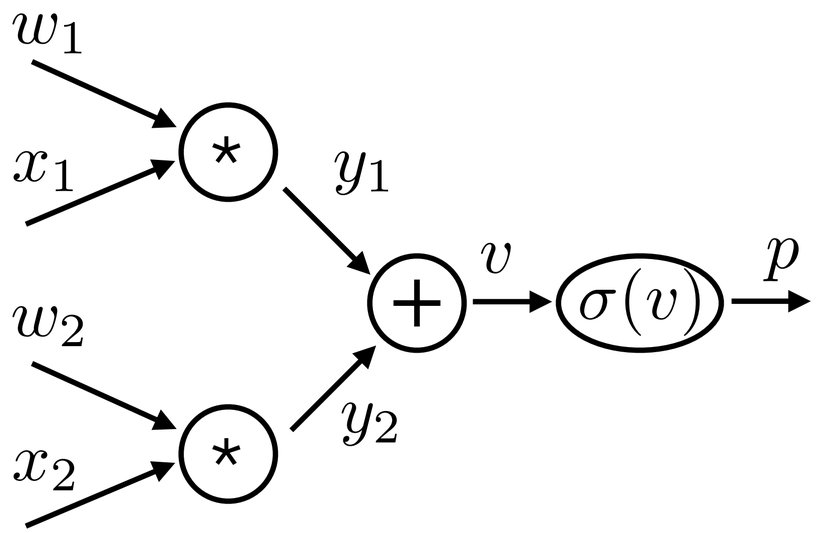
 # Linear classifier

$\mathbf{x} = \text{vec}($  $)$

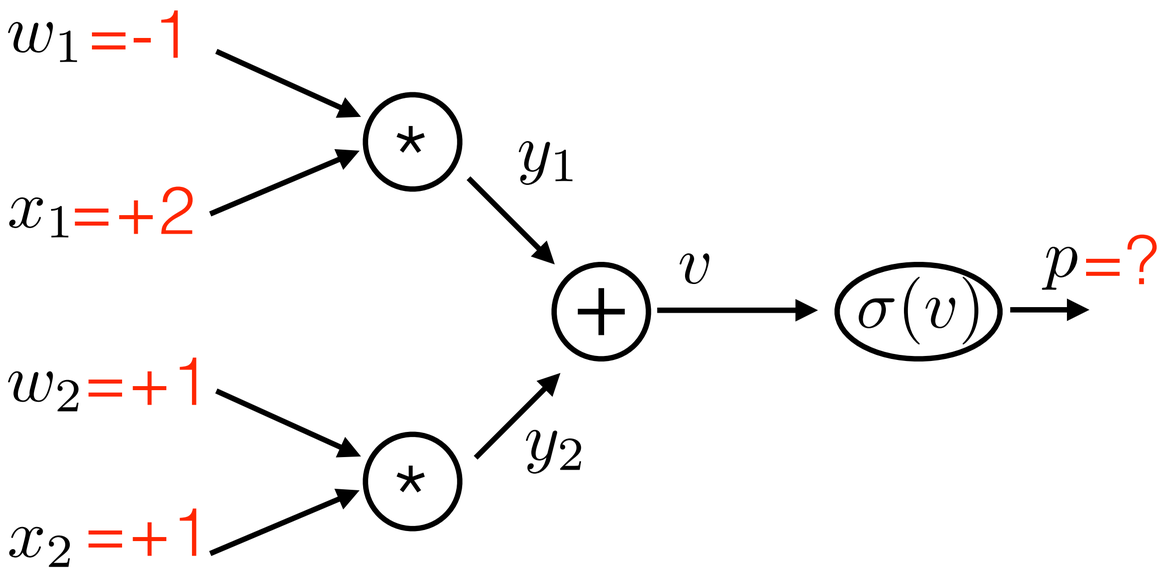
$p = \sigma\left(\mathbf{w}^\top \mathbf{x}\right)$

return p

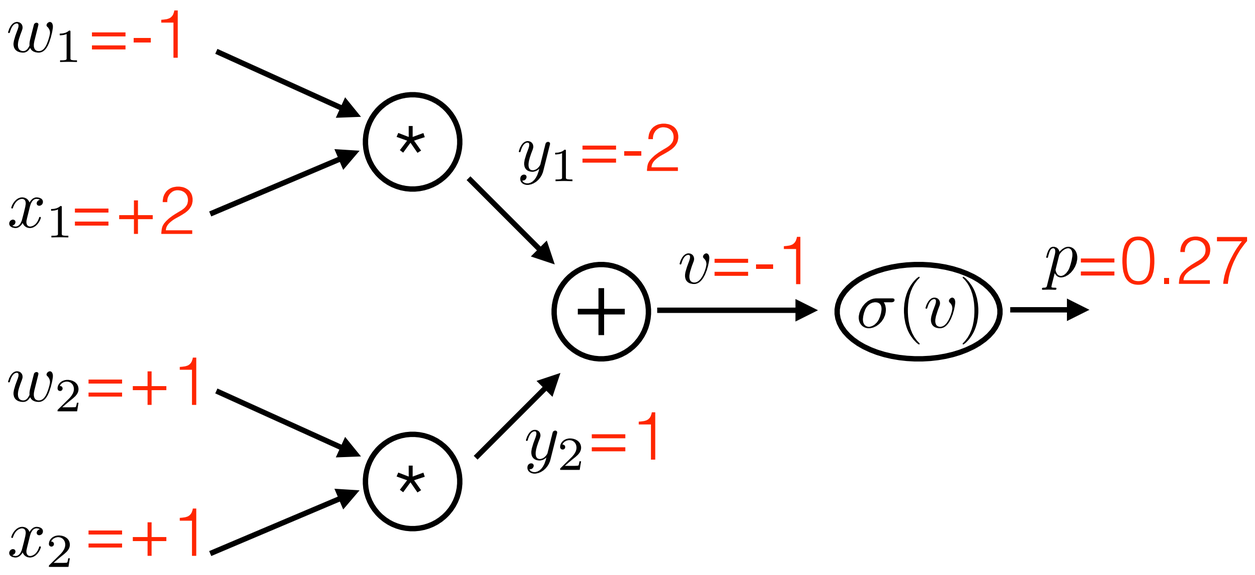
Computational graph of linear classifier



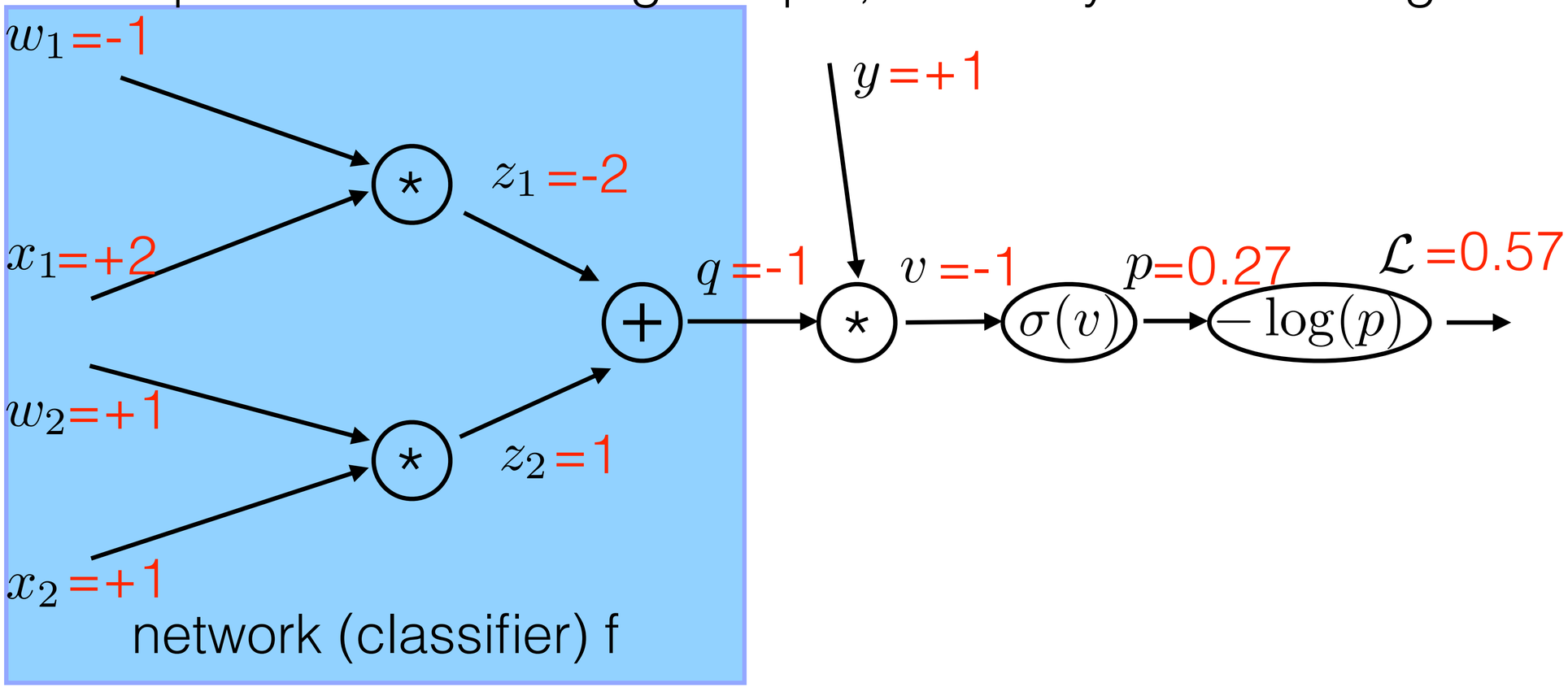
Example I: given trained neuron, and input, what is output?



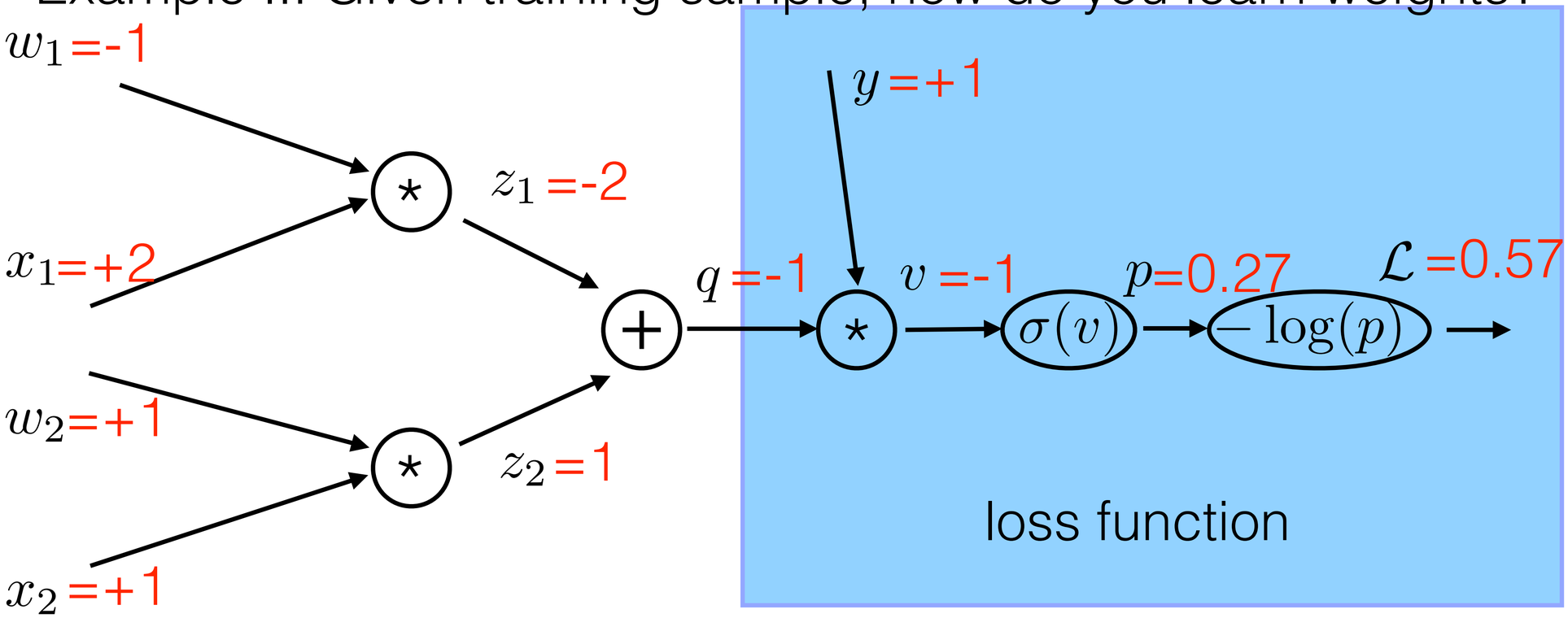
Example I: given trained classifier, and input, what is output?



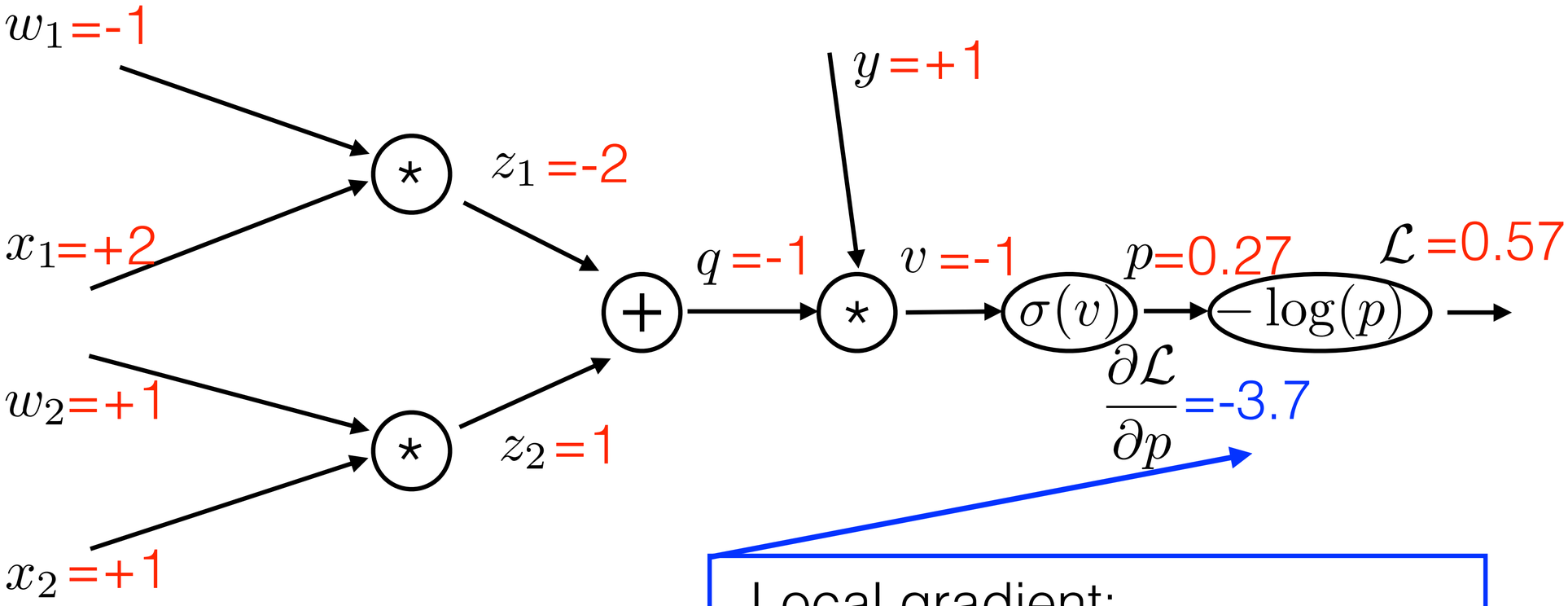
Example II: Given training sample, how do you learn weights?



Example II: Given training sample, how do you learn weights?



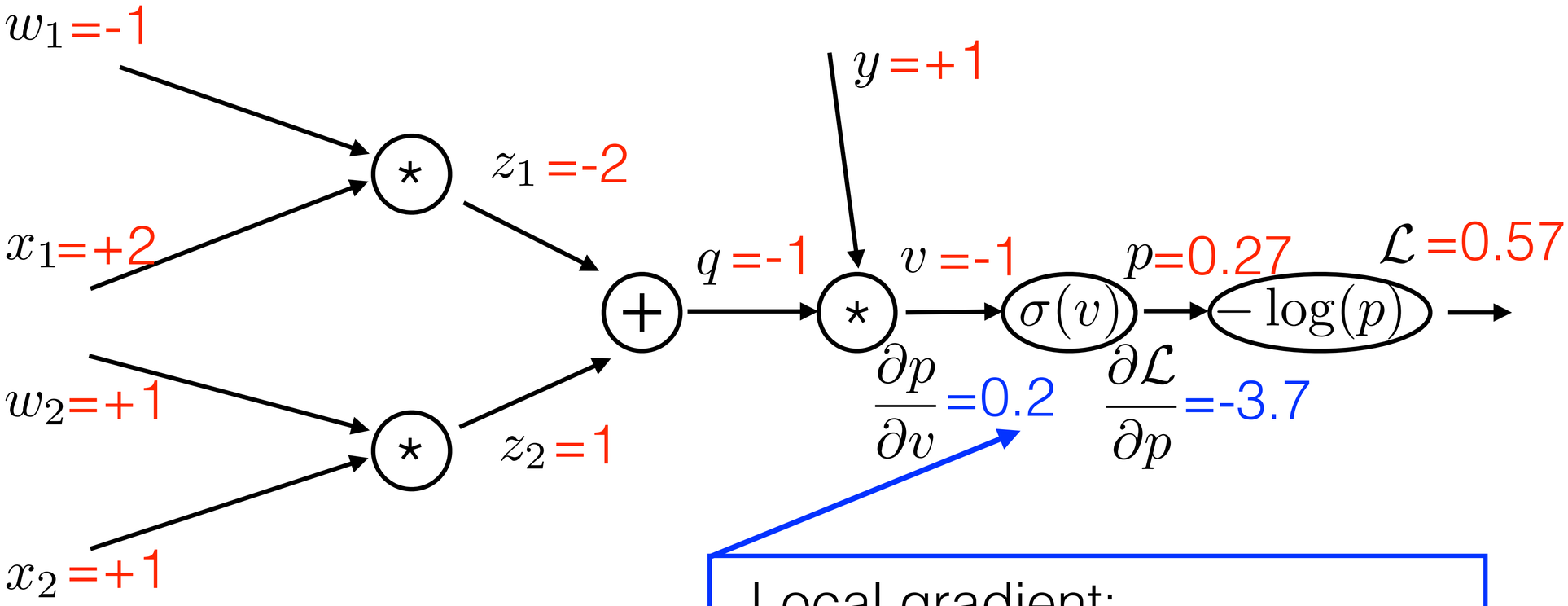
Example II: Given training sample, how do you learn weights?



Local gradient:

$$\frac{\partial \mathcal{L}}{\partial p} = \frac{\partial(-\log(p))}{\partial p} = -\frac{1}{p}$$

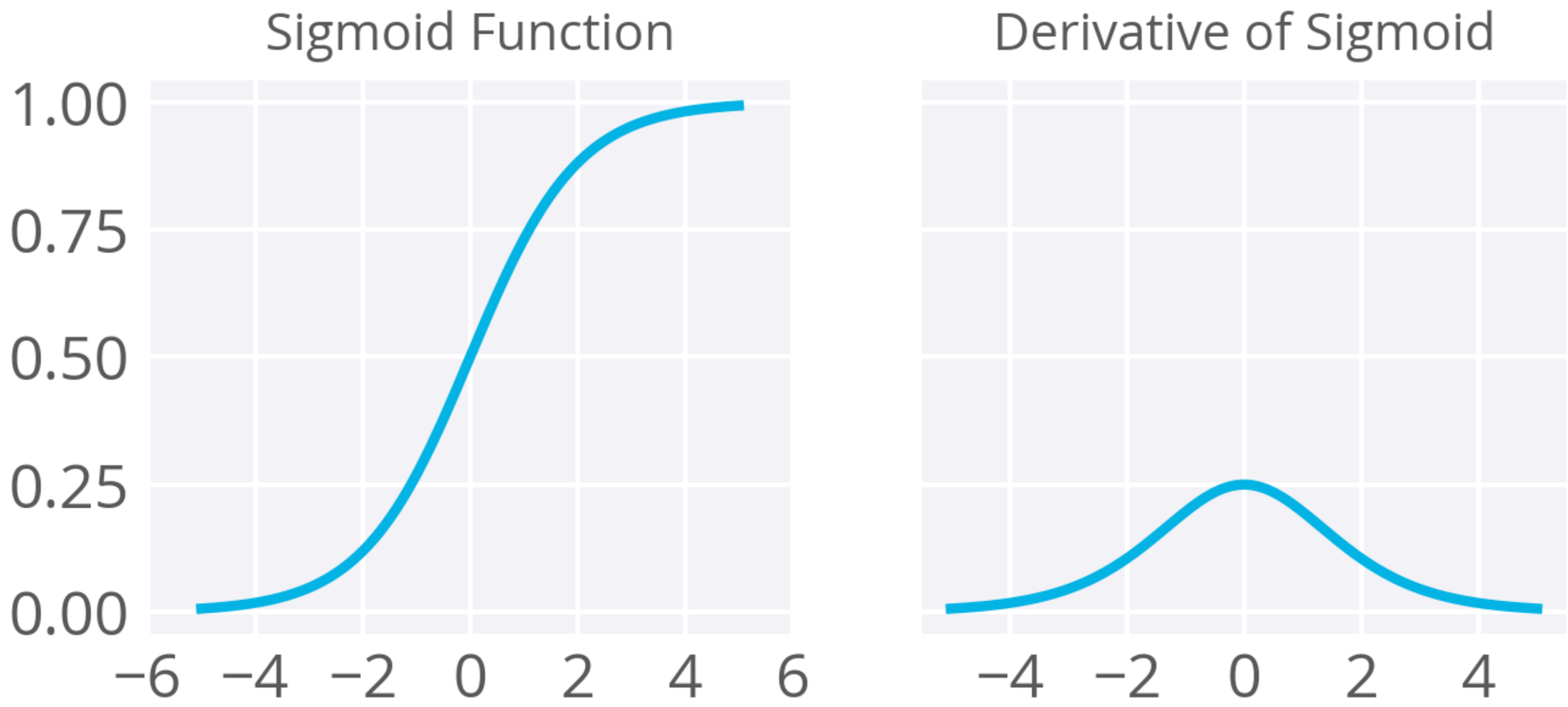
Example II: Given training sample, how do you learn weights?



Local gradient:

$$\frac{\partial p}{\partial v} = \frac{\partial \sigma(v)}{\partial v} = \sigma(v)(1-\sigma(v))$$

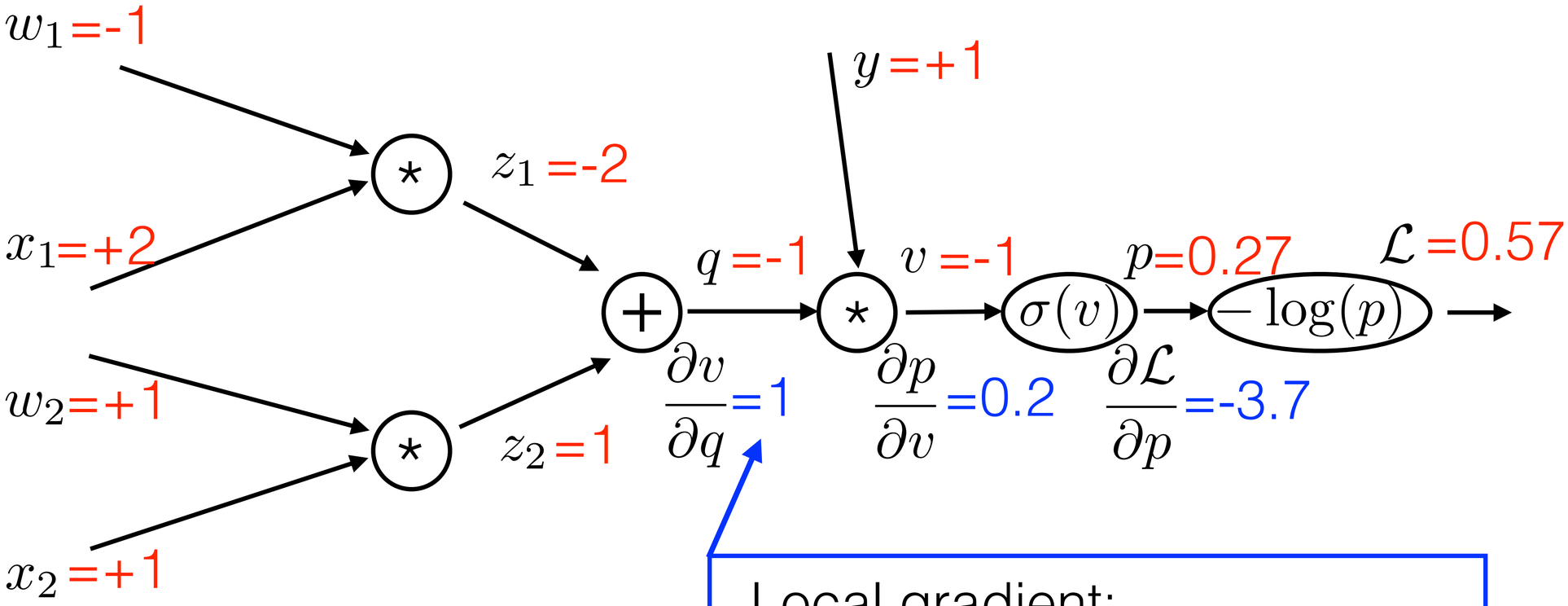
Example II: Given training sample, how do you learn weights?



Local gradient:

$$\frac{\partial p}{\partial v} = \frac{\partial \sigma(v)}{\partial v} = \sigma(v)(1 - \sigma(v))$$

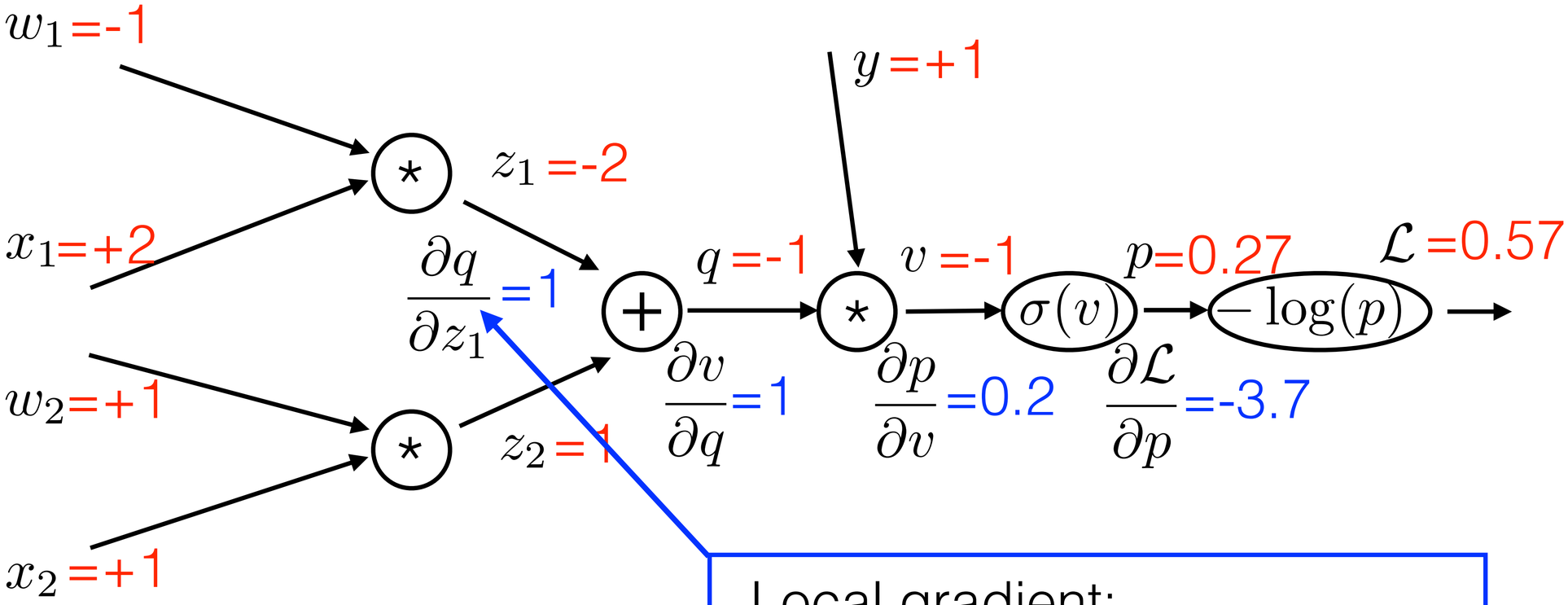
Example II: Given training sample, how do you learn weights?



Local gradient:

$$\frac{\partial v}{\partial q} = \frac{\partial (yq)}{\partial q} = y$$

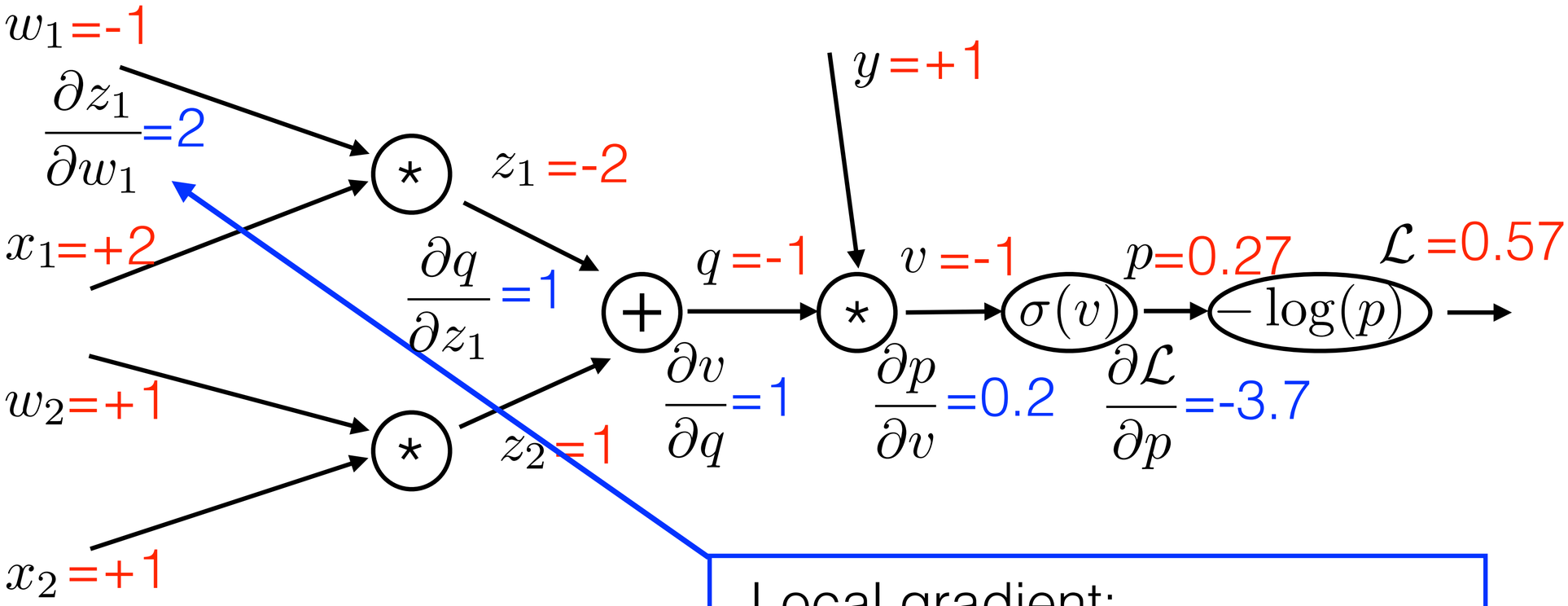
Example II: Given training sample, how do you learn weights?



Local gradient:

$$\frac{\partial q}{\partial z_1} = \frac{\partial (z_1 + z_2)}{\partial z_1} = 1$$

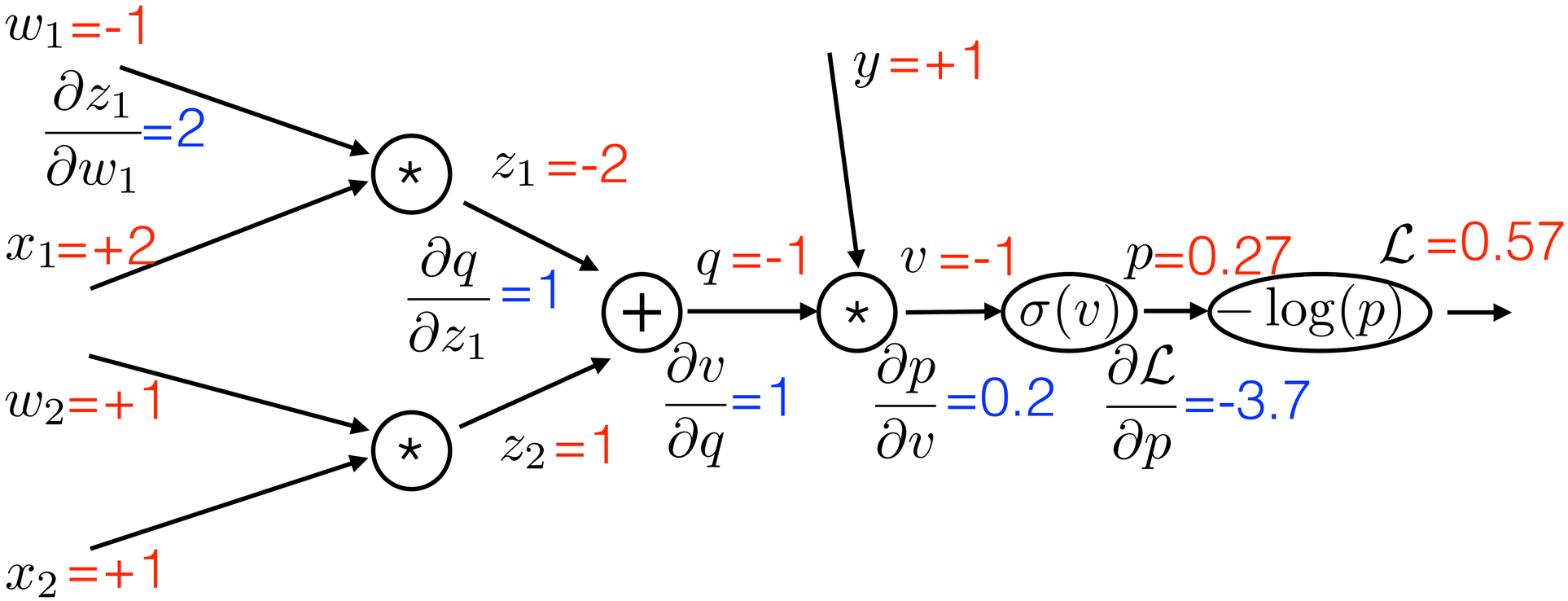
Example II: Given training sample, how do you learn weights?



Local gradient:

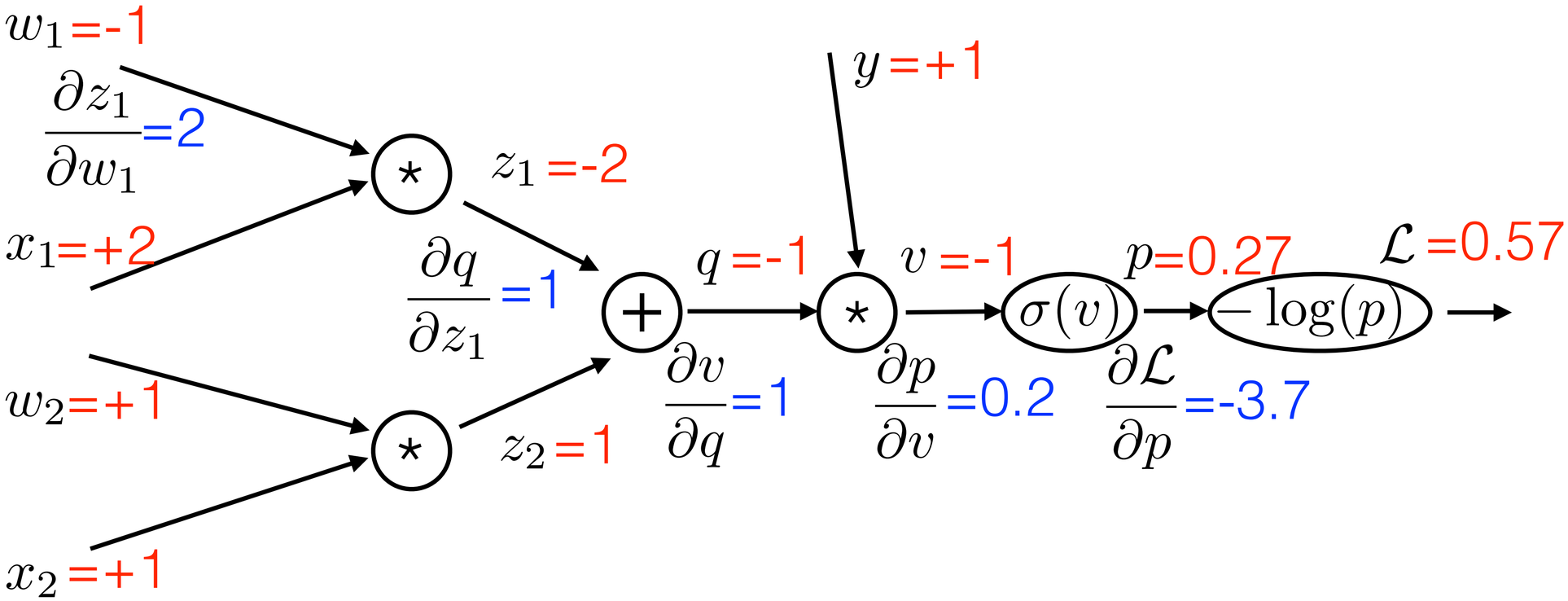
$$\frac{\partial z_1}{\partial w_1} = \frac{\partial (w_1 x_1)}{\partial w_1} = x_1$$

Example II: Given training sample, how do you learn weights?



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1} = -3.7*0.2*1*1*2 = -1.48$$

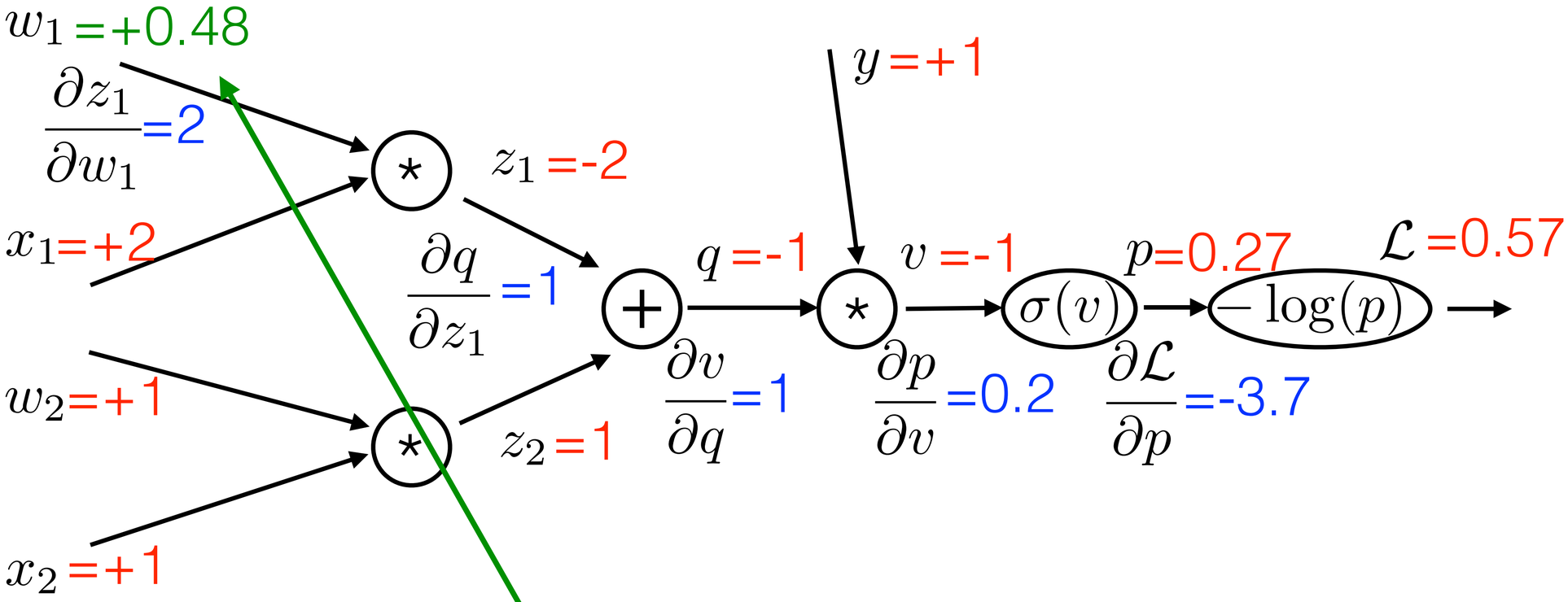
Example II: Given training sample, how do you learn weights?



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1} = -3.7*0.2*1*1*2 = -1.48$$

$$w_1 = w_1 - \alpha\frac{\partial \mathcal{L}}{\partial w1} = +0.48$$

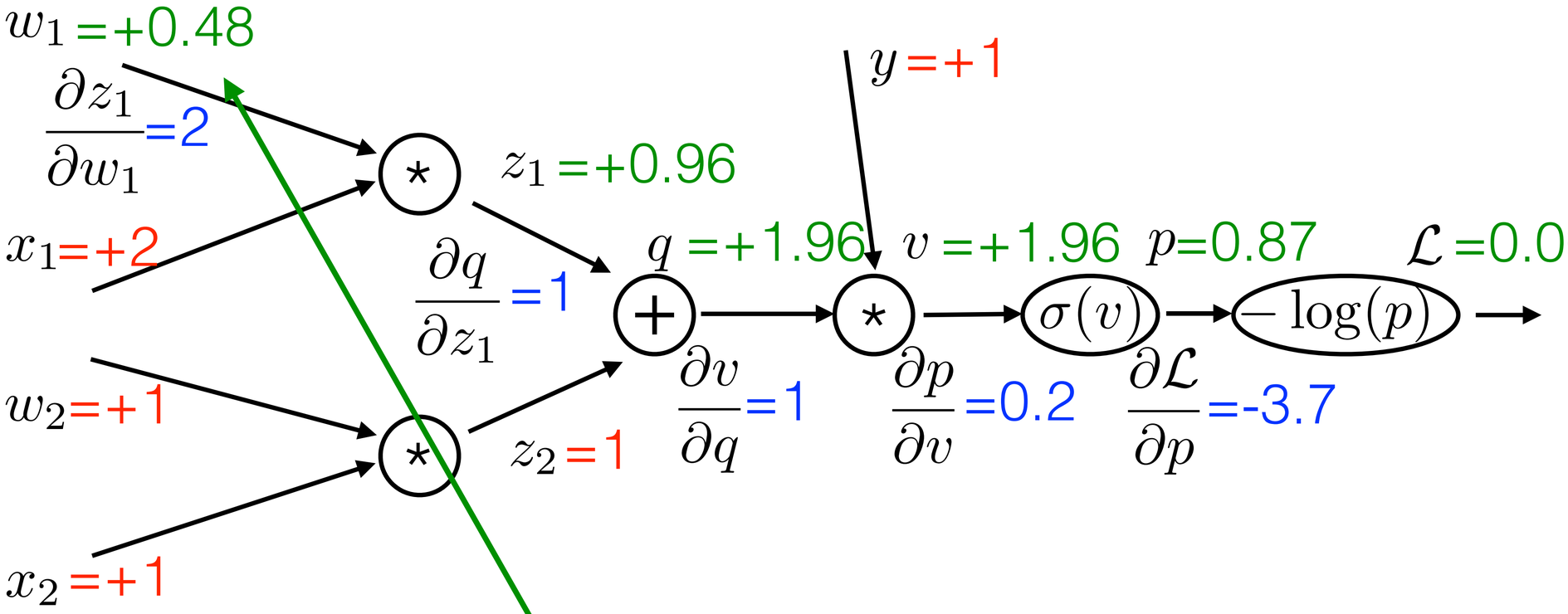
Example II: Given training sample, how do you learn weights?



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1} = -3.7*0.2*1*1*2 = -1.48$$

$$w_1 = w_1 - \alpha\frac{\partial \mathcal{L}}{\partial w1} = +0.48$$

Example II: Given training sample, how do you learn weights?



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1} = -3.7*0.2*1*1*2 = -1.48$$

$$w_1 = w_1 - \alpha\frac{\partial \mathcal{L}}{\partial w1} = +0.48$$

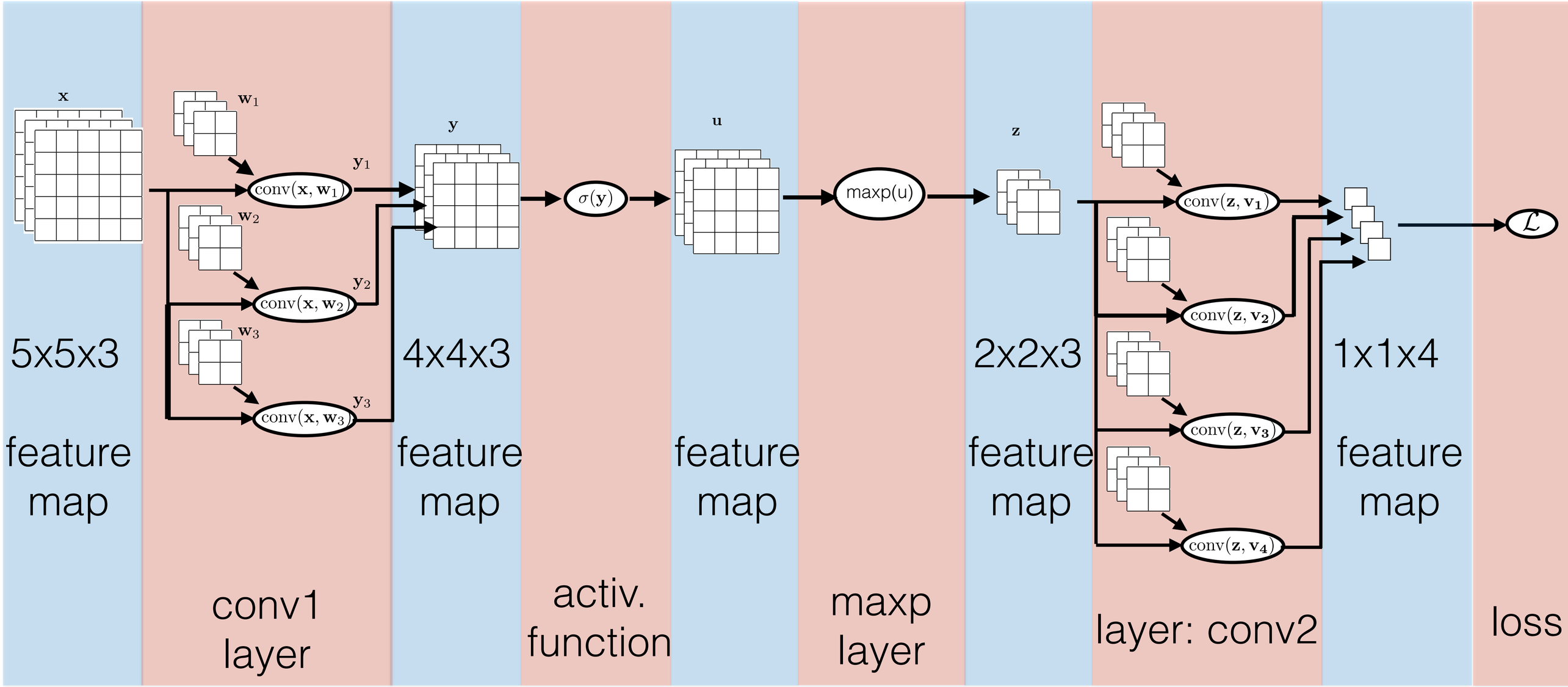
Prequel

"Close-to-Zero and Close-to-infinity gradient are actually bad!!!"

- achieving close-to-zero gradient usually means that learning got stuck (and not that you achieved the global optimum)
- achieving close-to-infinity gradient usually means that learning diverges (and not that you found direction in which you can find the global optimum)
- there are many saddle points and few local minima (surprisingly none of it is a huge problem)
- sharp local minimum is bad (sensitive to noise)
- we can avoid many problems by:
  - advanced gradient optimization method (today's lecture)
  - suitable initialization (next lecture)
  - optimization-friendly network structure (lecture after the next lecture)

Learning as gradient minimization

- Let us denote whole network including loss layer as $f(\mathbf{x}; \mathbf{w})$



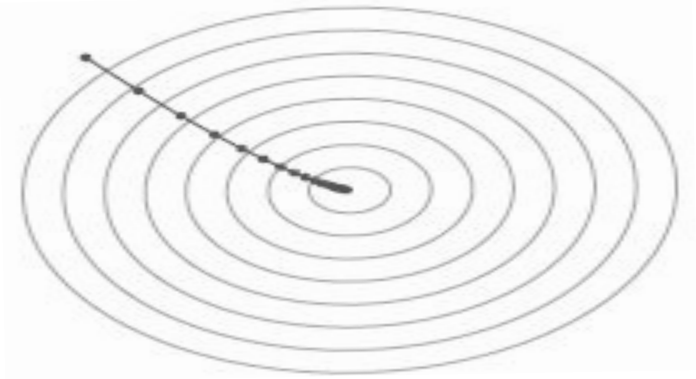
Learning as gradient minimization

1. Initialize weights $\mathbf{w}_0$ and $k = 1$
2. Plug $\mathbf{x}_i$ to input and estimate $\left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$ by backprop
3. Estimate gradient over whole training set

$$\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}} = \frac{1}{N}\sum_{i=1}^{N} \left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

4. Update weights

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



Learning as gradient minimization

1. Initialize weights $\mathbf{w}_0$ and $k = 1$
2. Plug $\mathbf{x}_i$ to input and estimate $\left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$ by backprop
3. Estimate gradient over whole training set

$$\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}} = \frac{1}{N}\sum_{i=1}^{N} \left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

4. Update weights

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

- Whole training set is typically huge =>
  Gradient estimation would be time-consuming =>
  We instead estimate “stochastic gradient over minibatch”

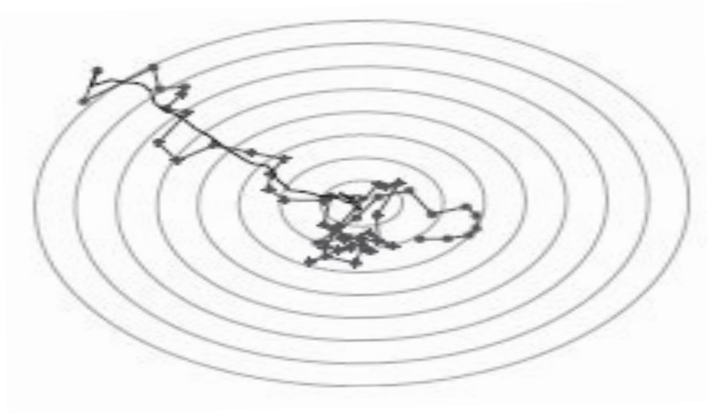
Learning as gradient minimization

1. Initialize weights $\mathbf{w}_0$ and $k = 1$
2. Plug $\mathbf{x}_i$ to input and estimate $\left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$ by backprop
3. Estimate gradient over random mini-batch

$$\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}} = \frac{1}{|\mathrm{MB}|} \sum_{i \in \mathrm{MB}} \left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

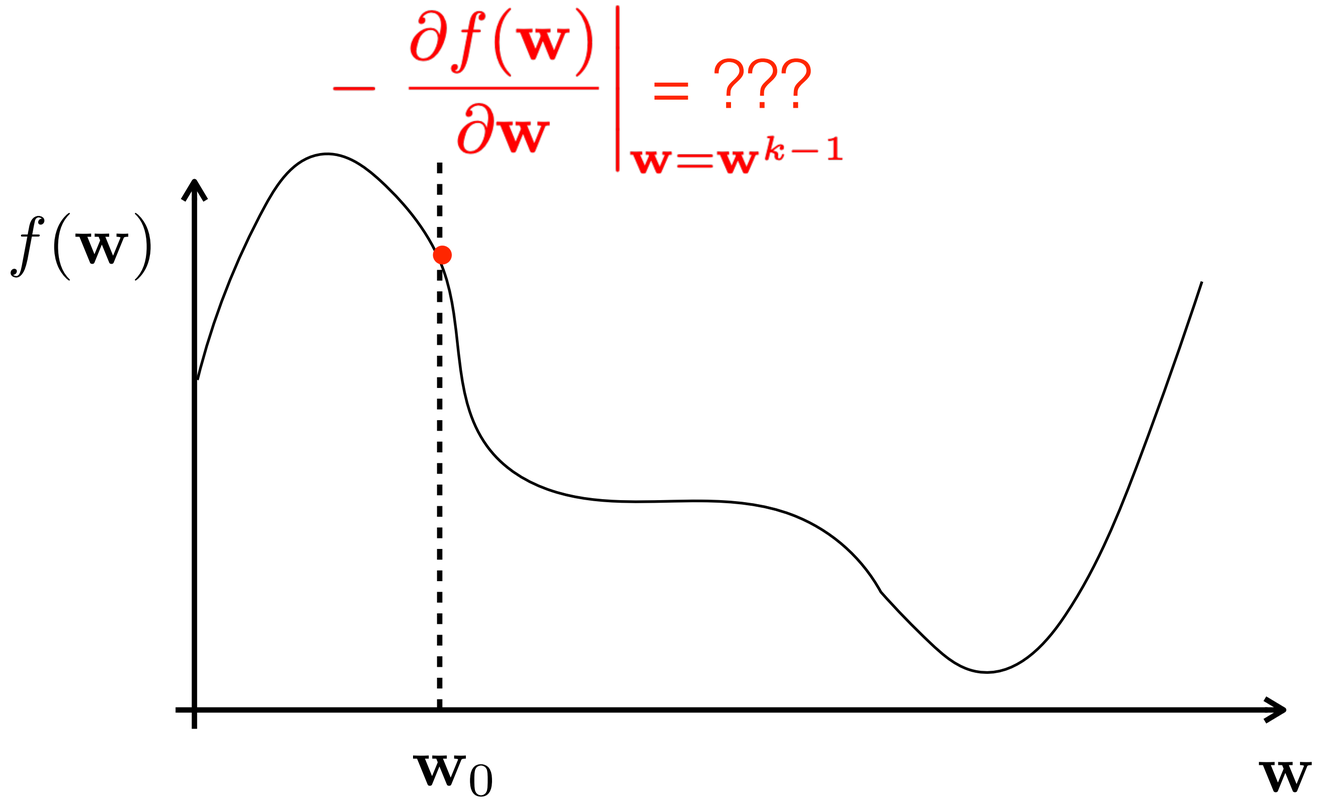
4. Update weights

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



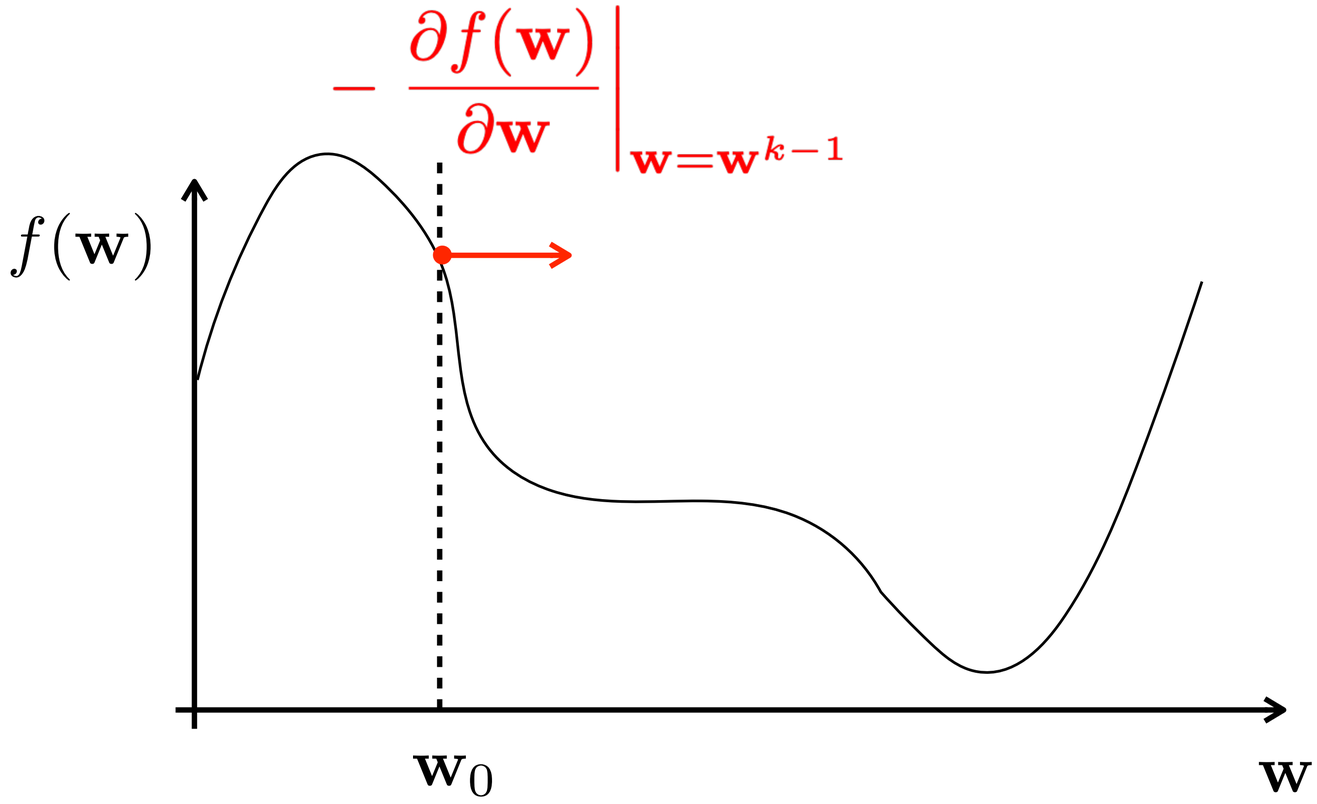
Stochastic Gradient Descent (SGD) drawbacks

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



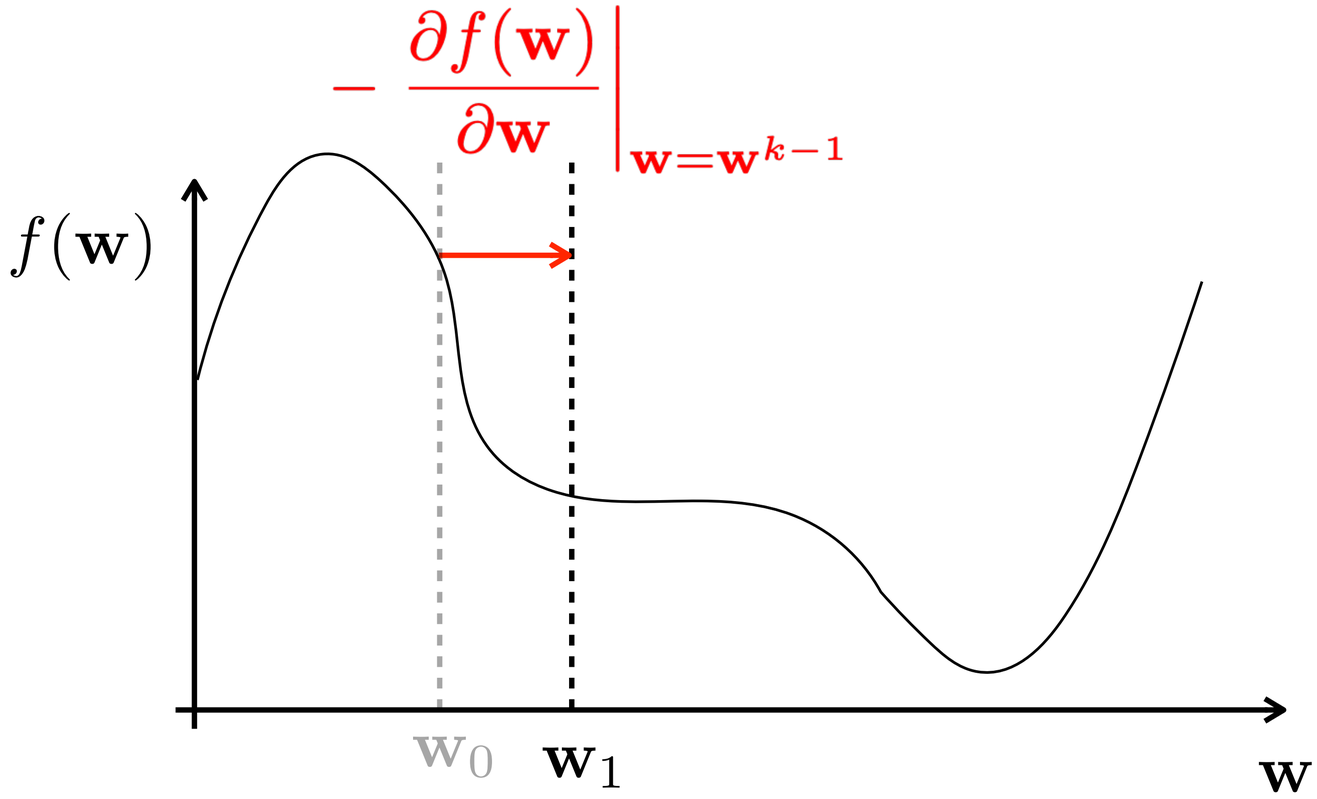
Stochastic Gradient Descent (SGD) drawbacks

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



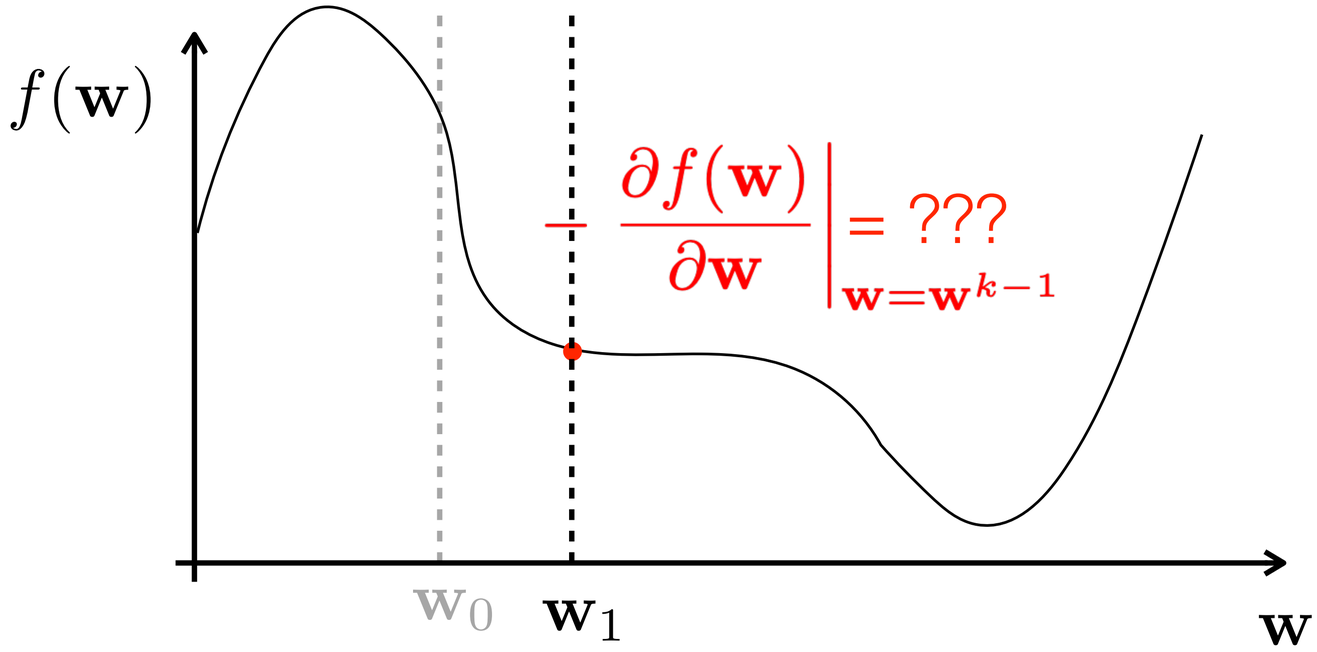
SGD drawbacks

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



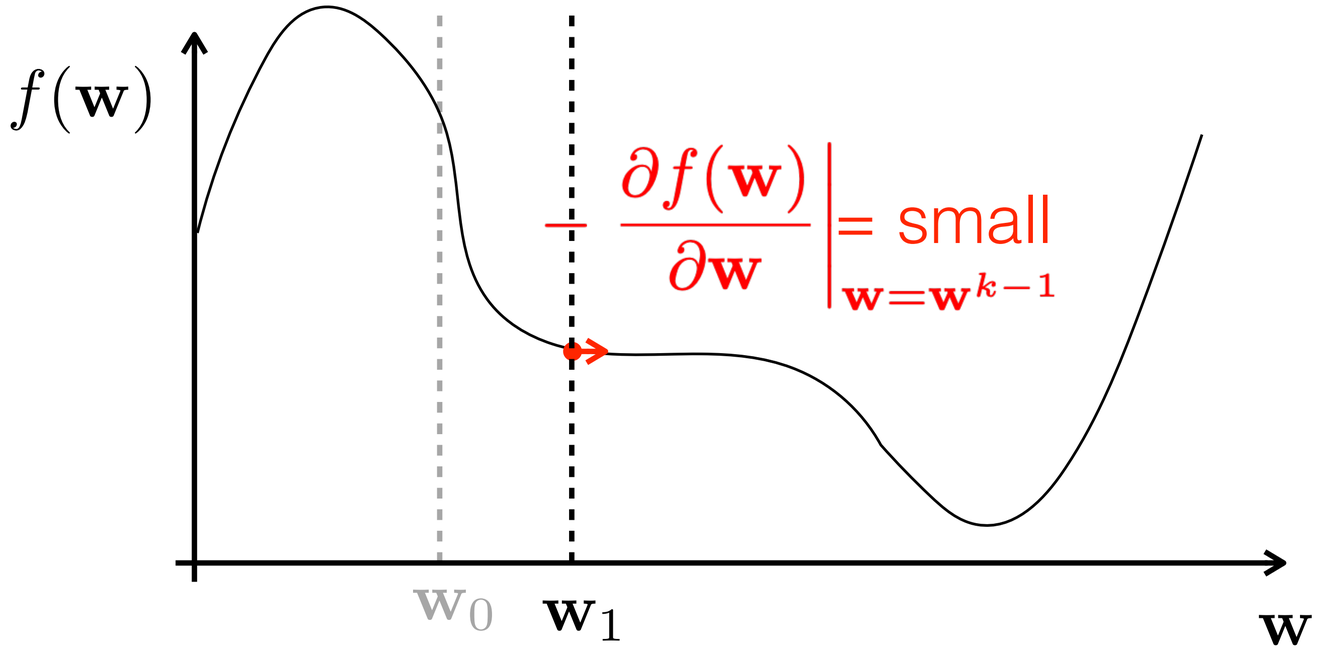
SGD drawbacks

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



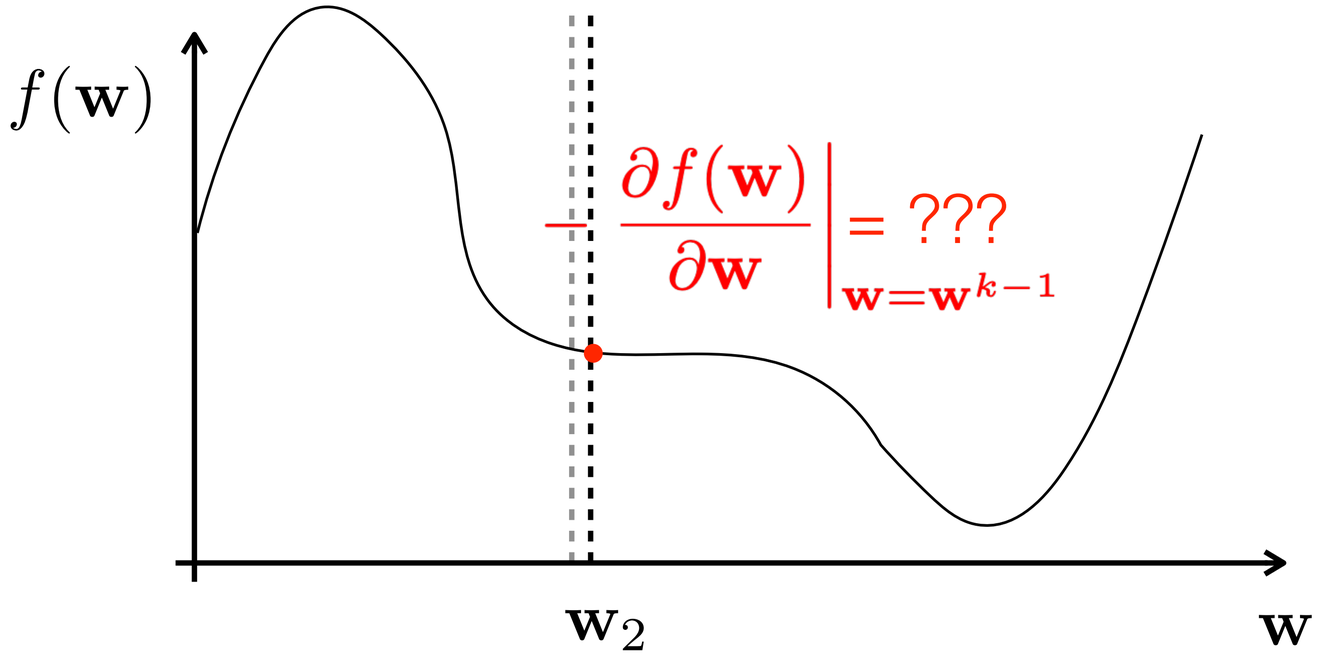
SGD drawbacks

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



SGD drawbacks

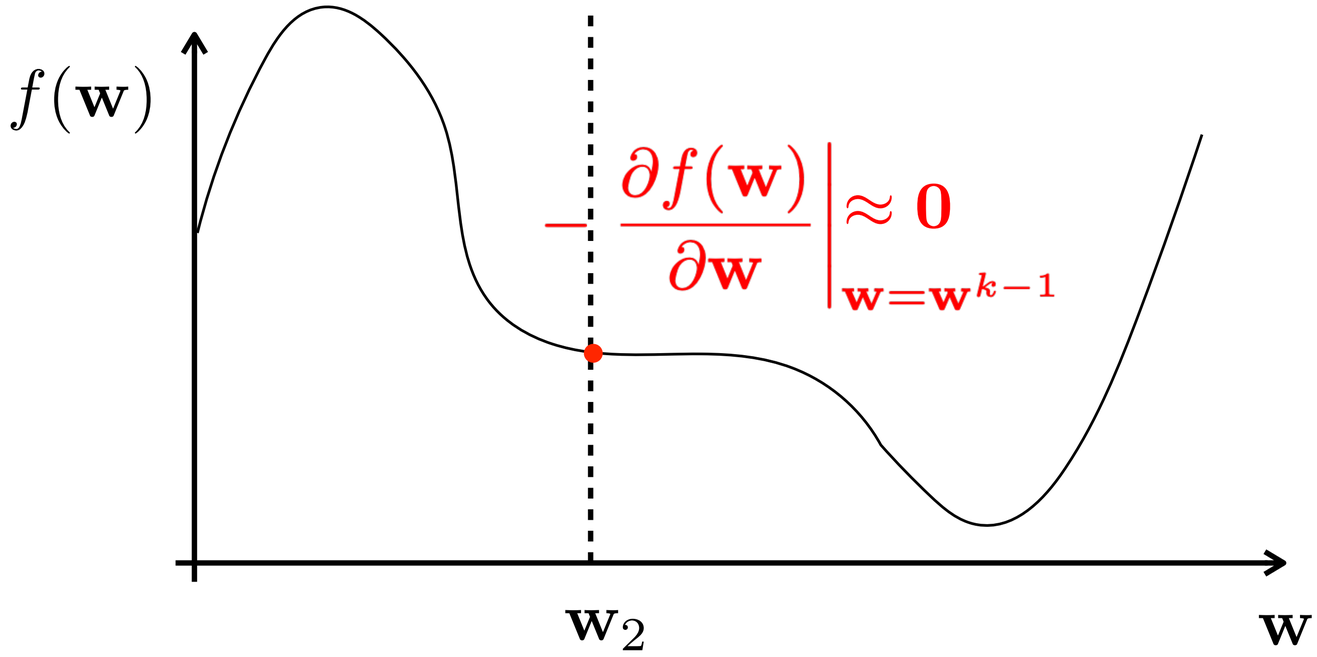
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



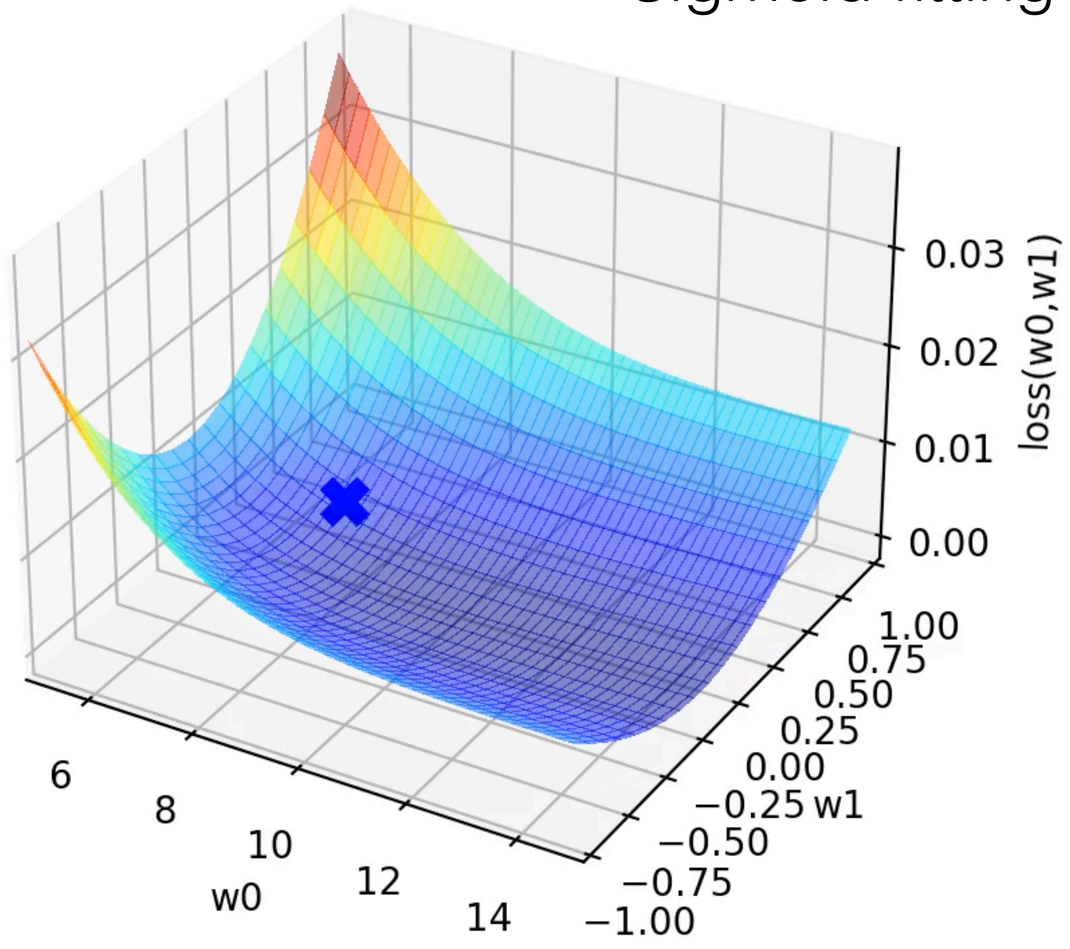
SGD drawbacks

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

- SGD gets stuck easily on a flat landscape

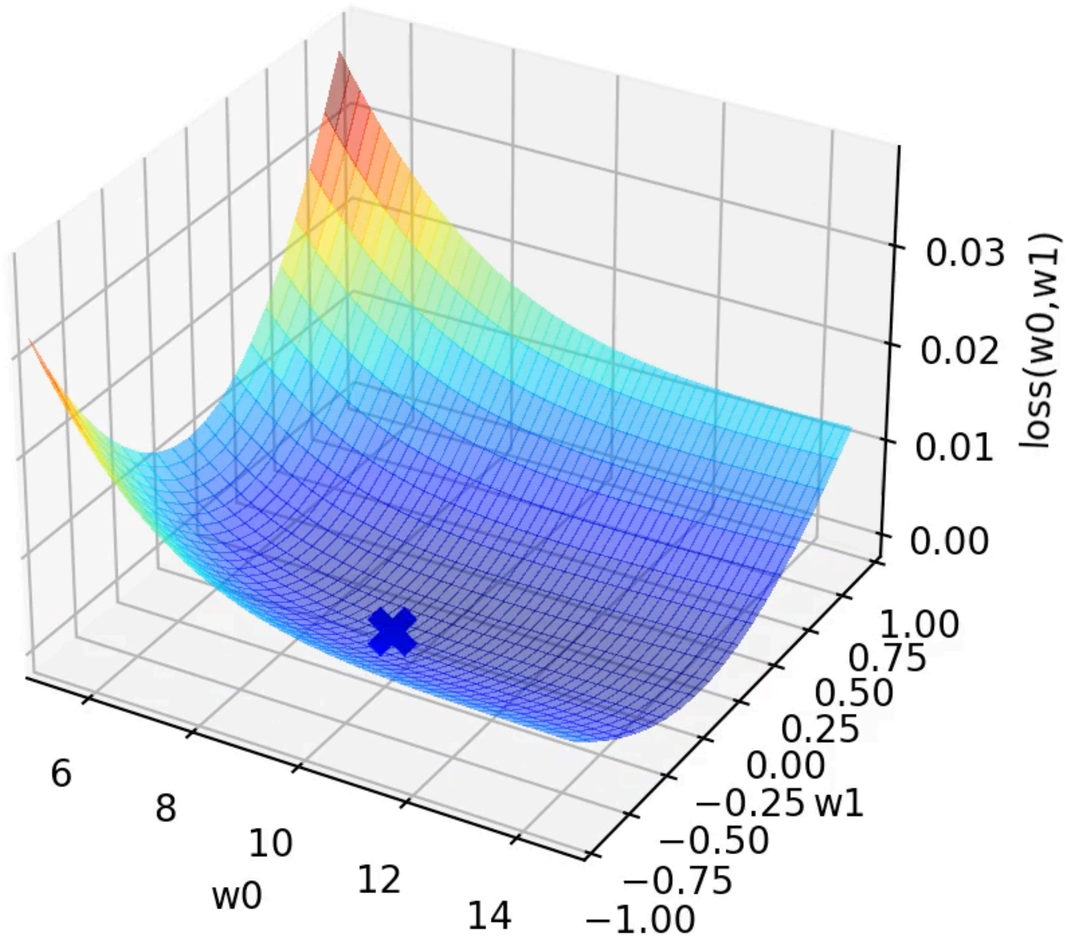


Sigmoid fitting problem from labs



LSQ

$\alpha = 50$

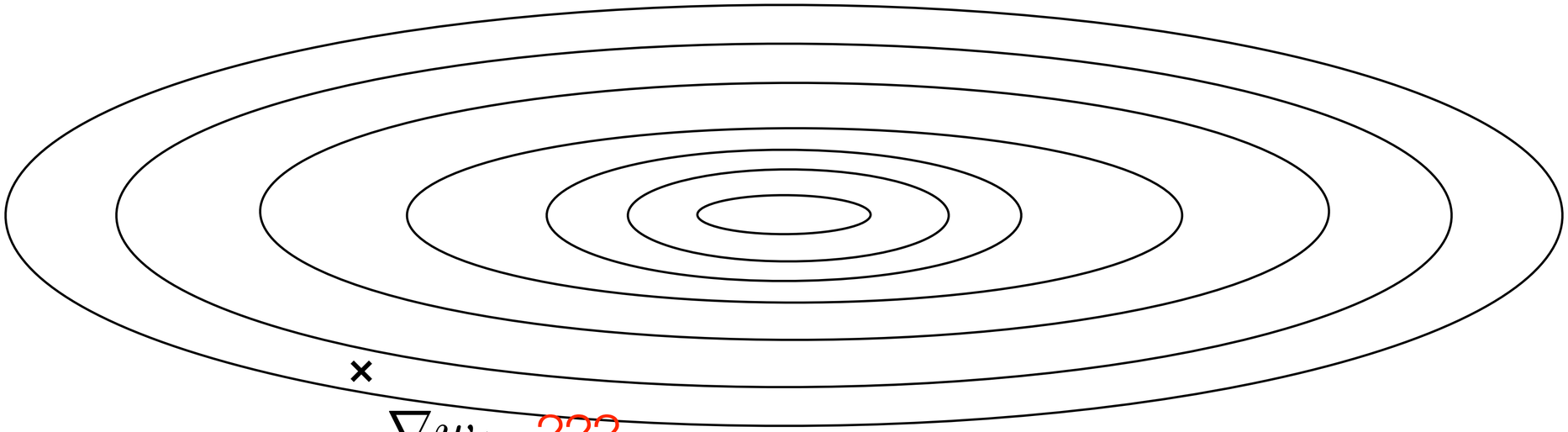


LSQ

$\alpha = 80$

SGD in 2 dimensional weights

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



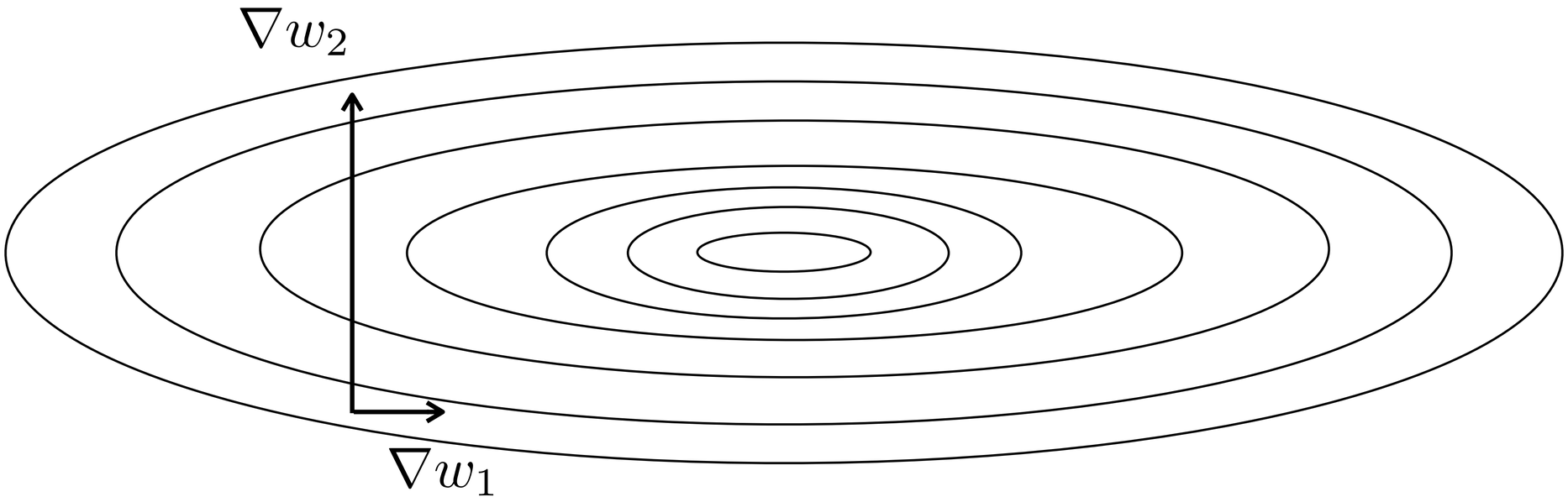
∇w_1=???

∇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 in 2 dimensional weights

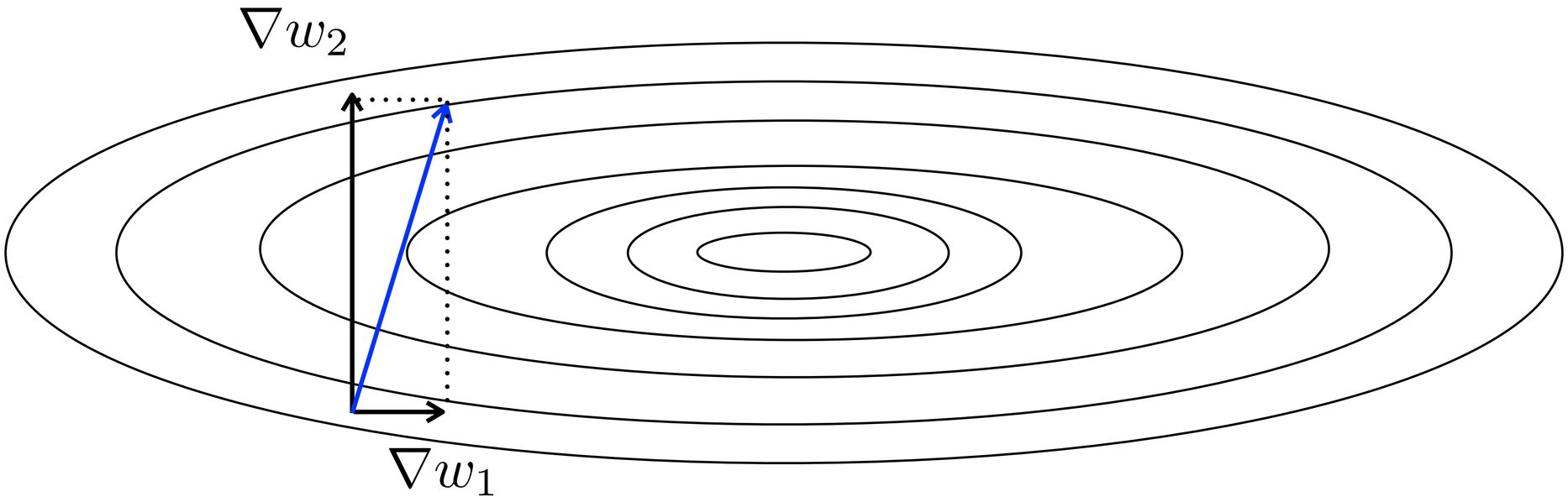
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD in 2 dimensional weights

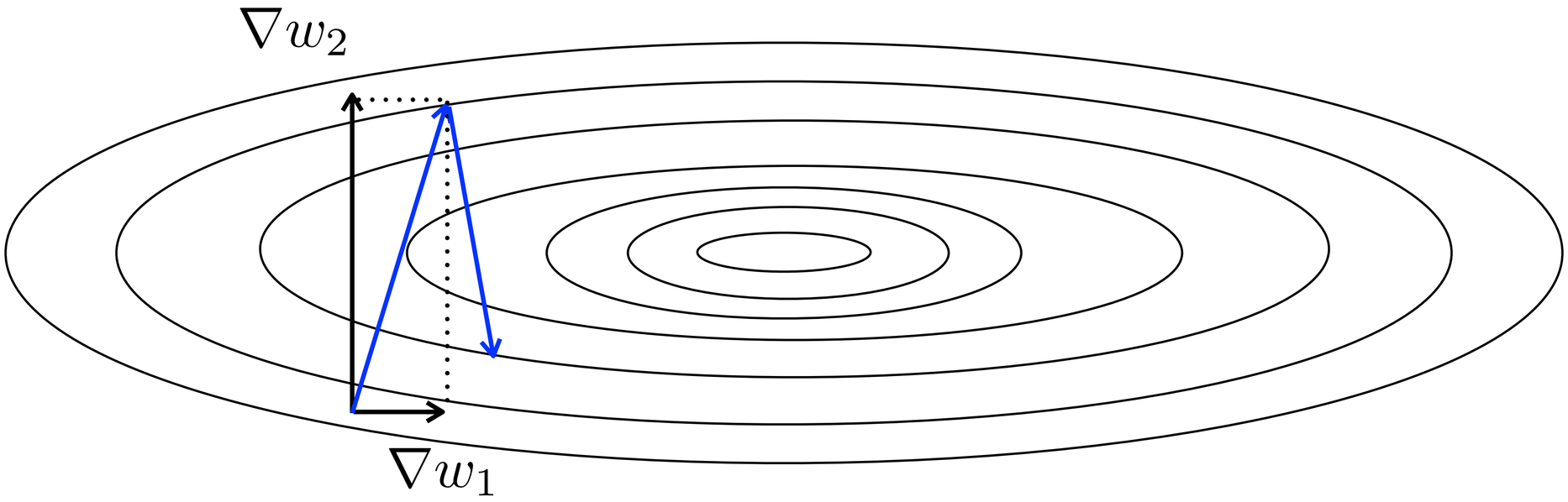
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD in 2 dimensional weights

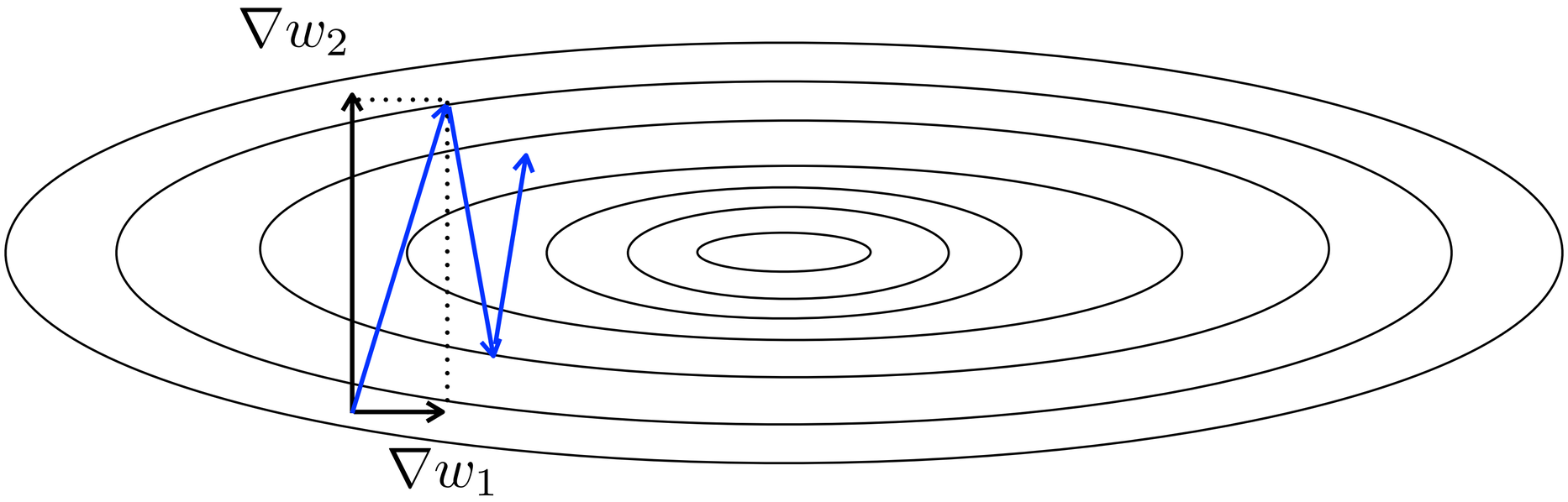
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD in 2 dimensional weights

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

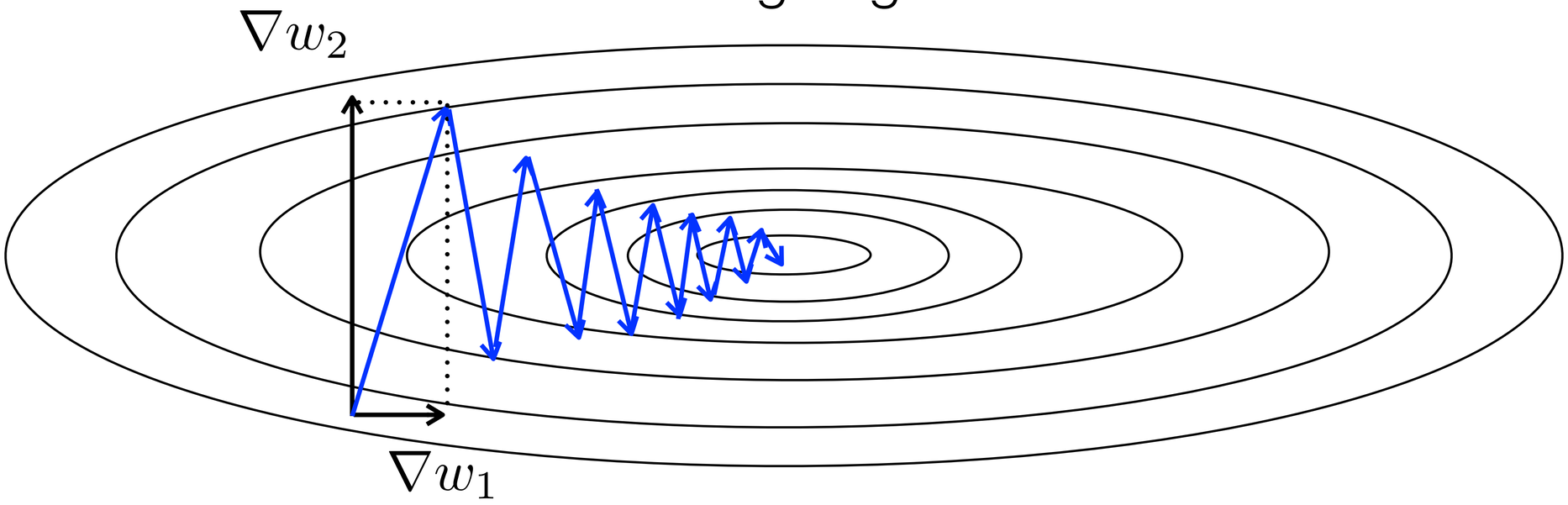


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD in 2 dimensional weights

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

Undesired zig-zag behaviour

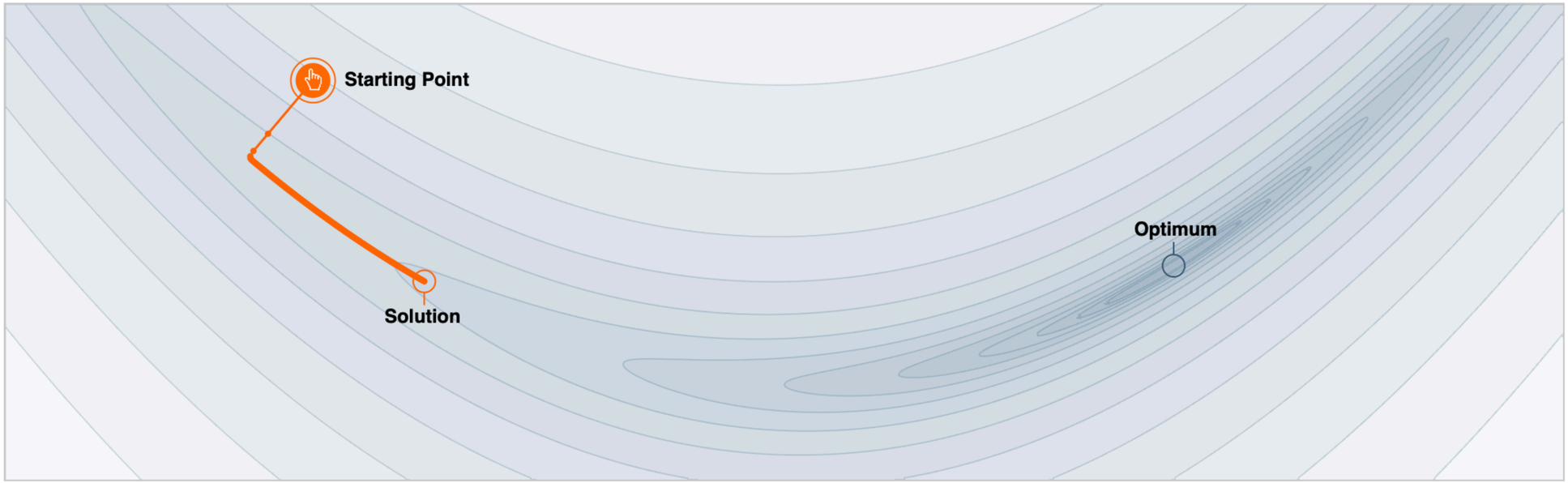


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks - in 2D

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

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

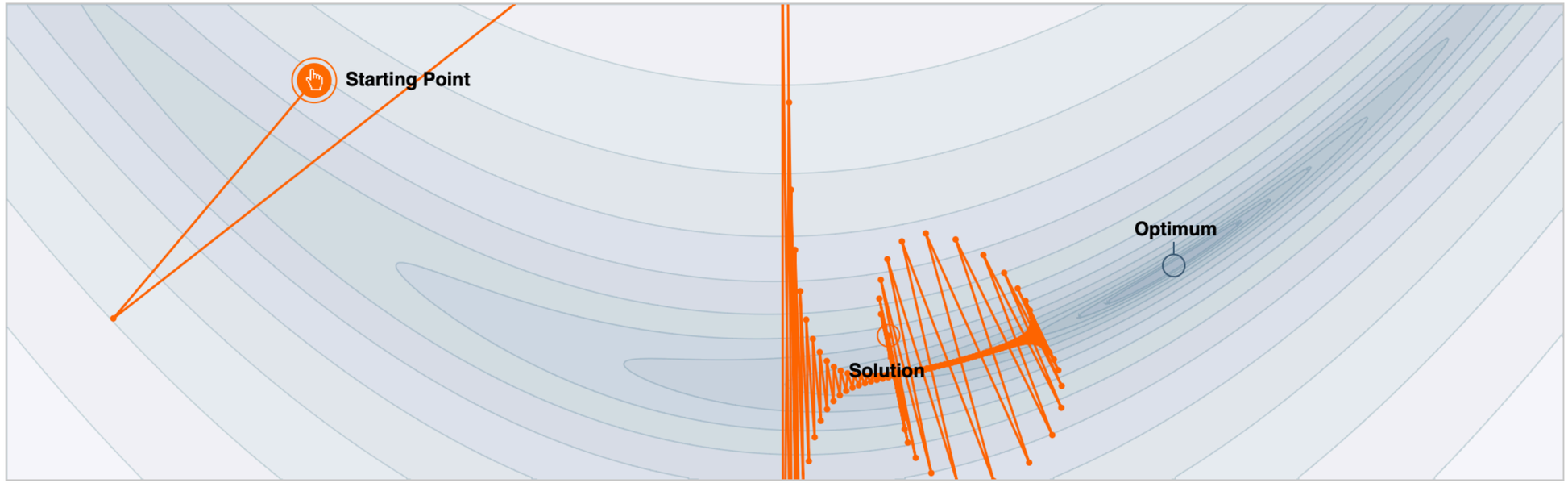


https://distill.pub/2017/momentum/

SGD drawbacks - in 2D

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

$\alpha =$ 5e-3

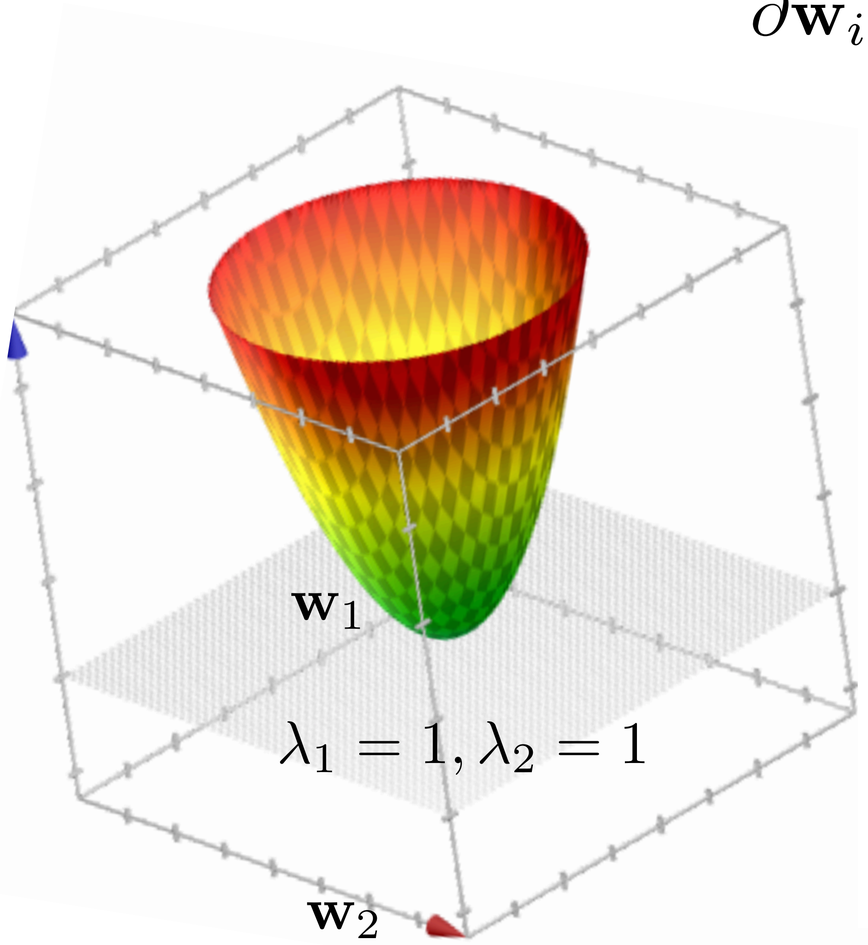


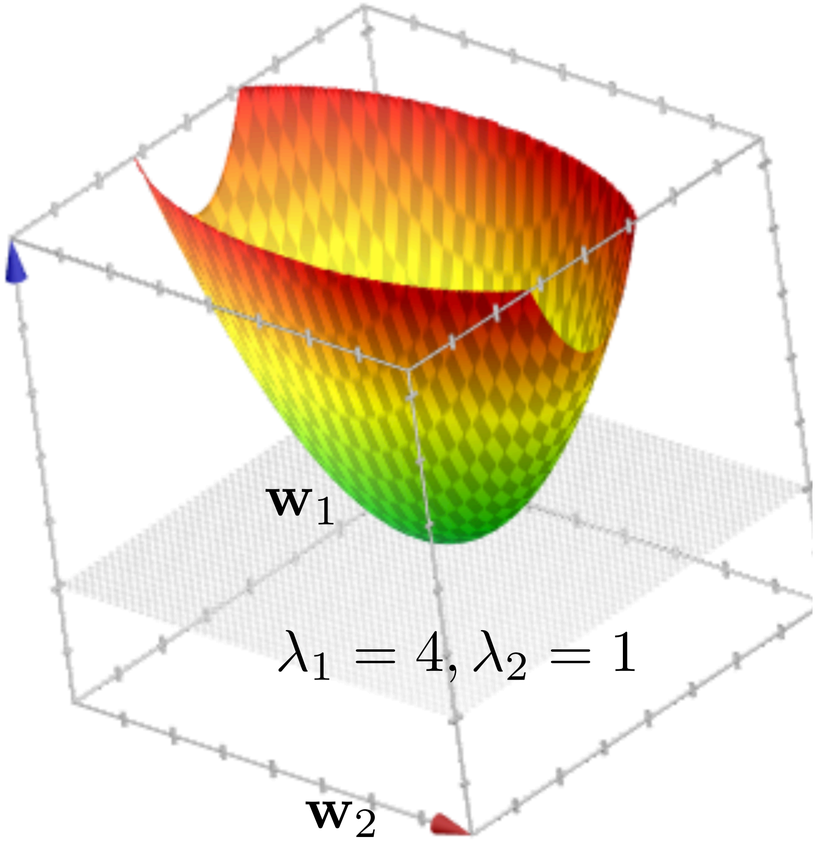
https://distill.pub/2017/momentum/

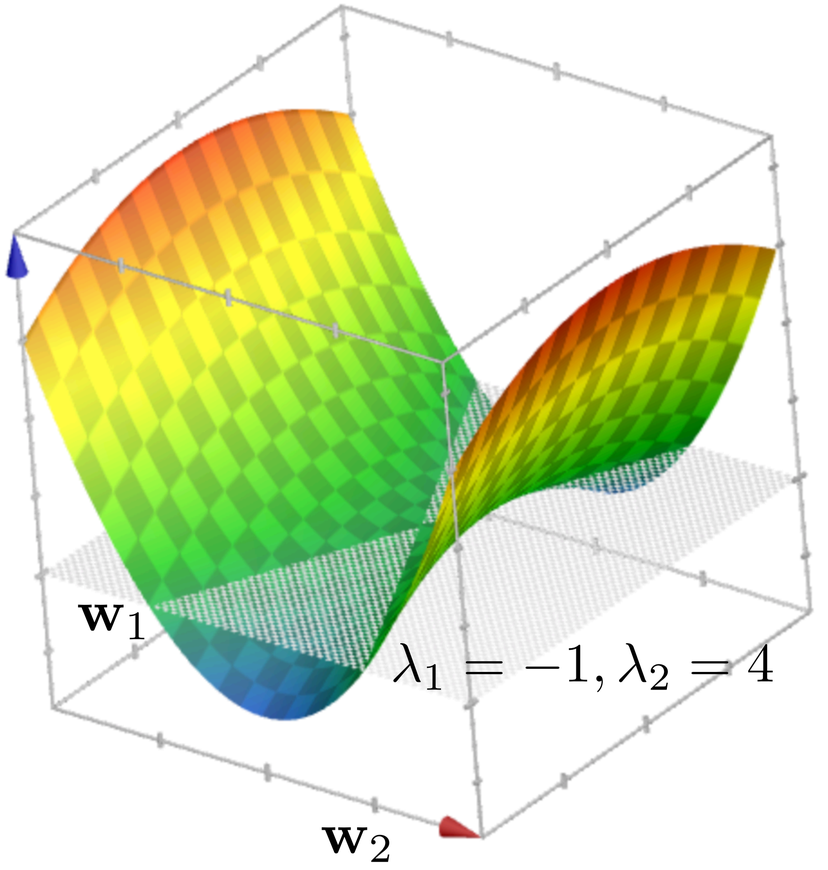
SGD on quadric

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 on quadric

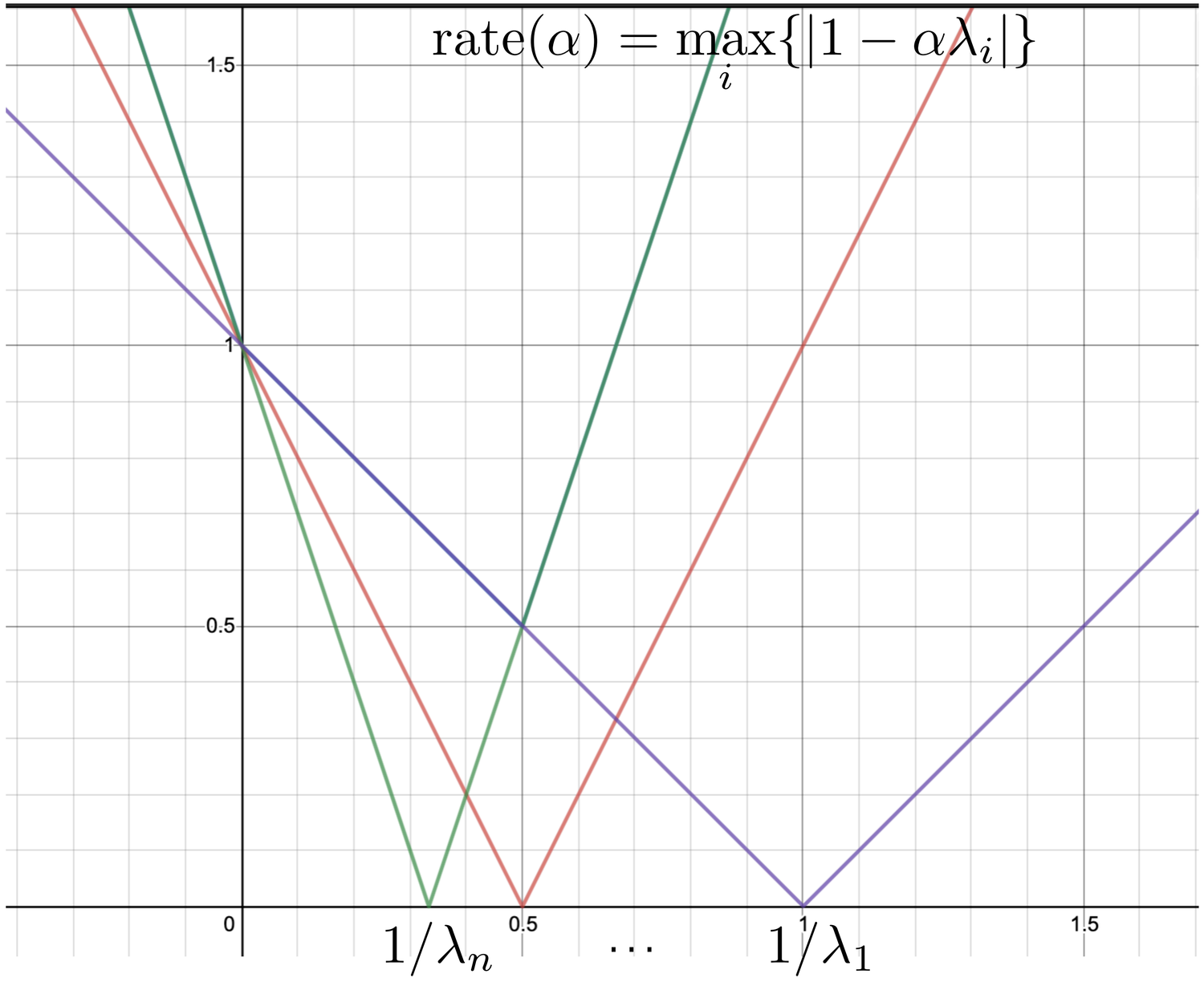
Criterion: $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathtt{A}\mathbf{w}$ with $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

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

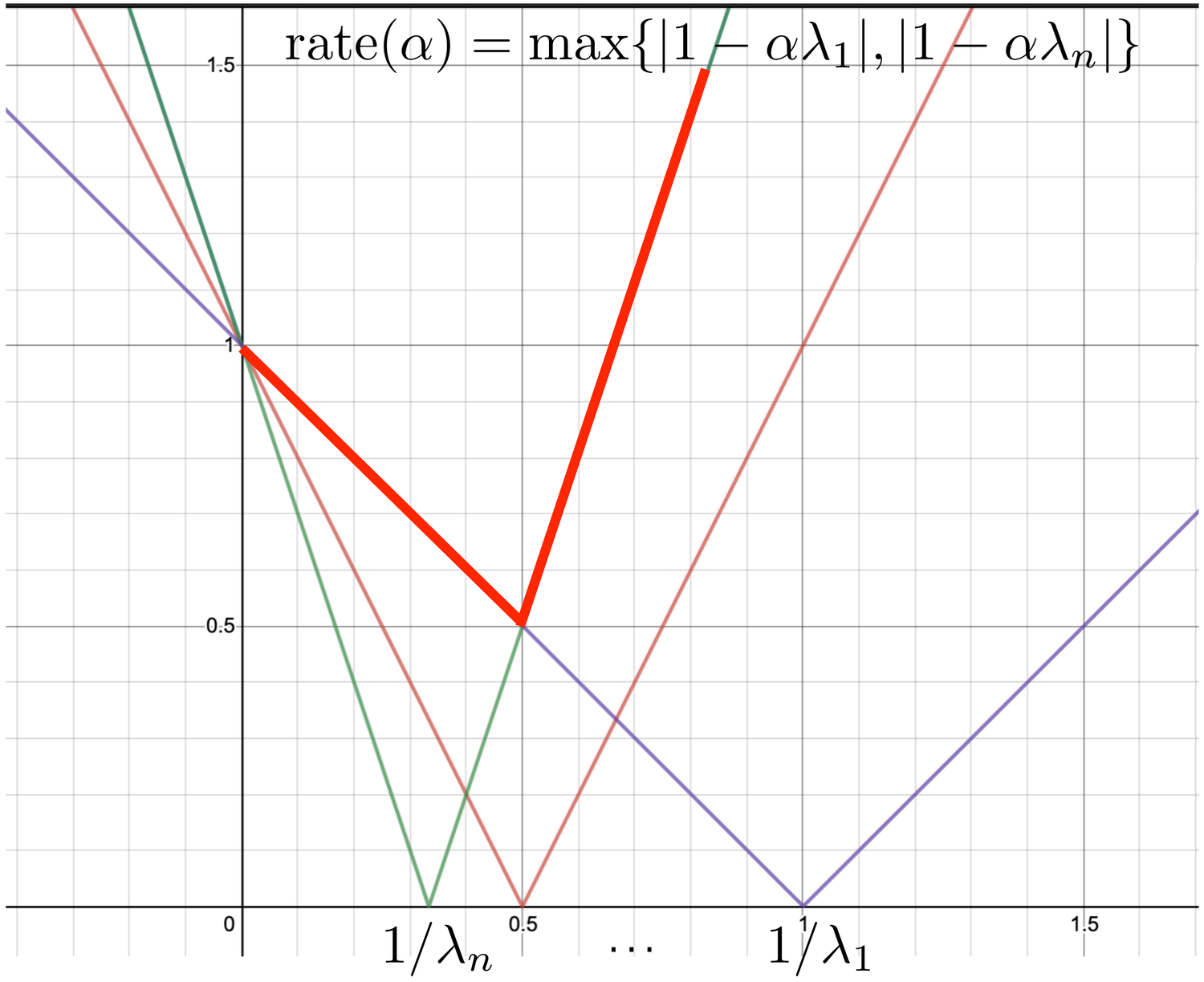
SGD after k iterations: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

- 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 $\mathtt{A} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$

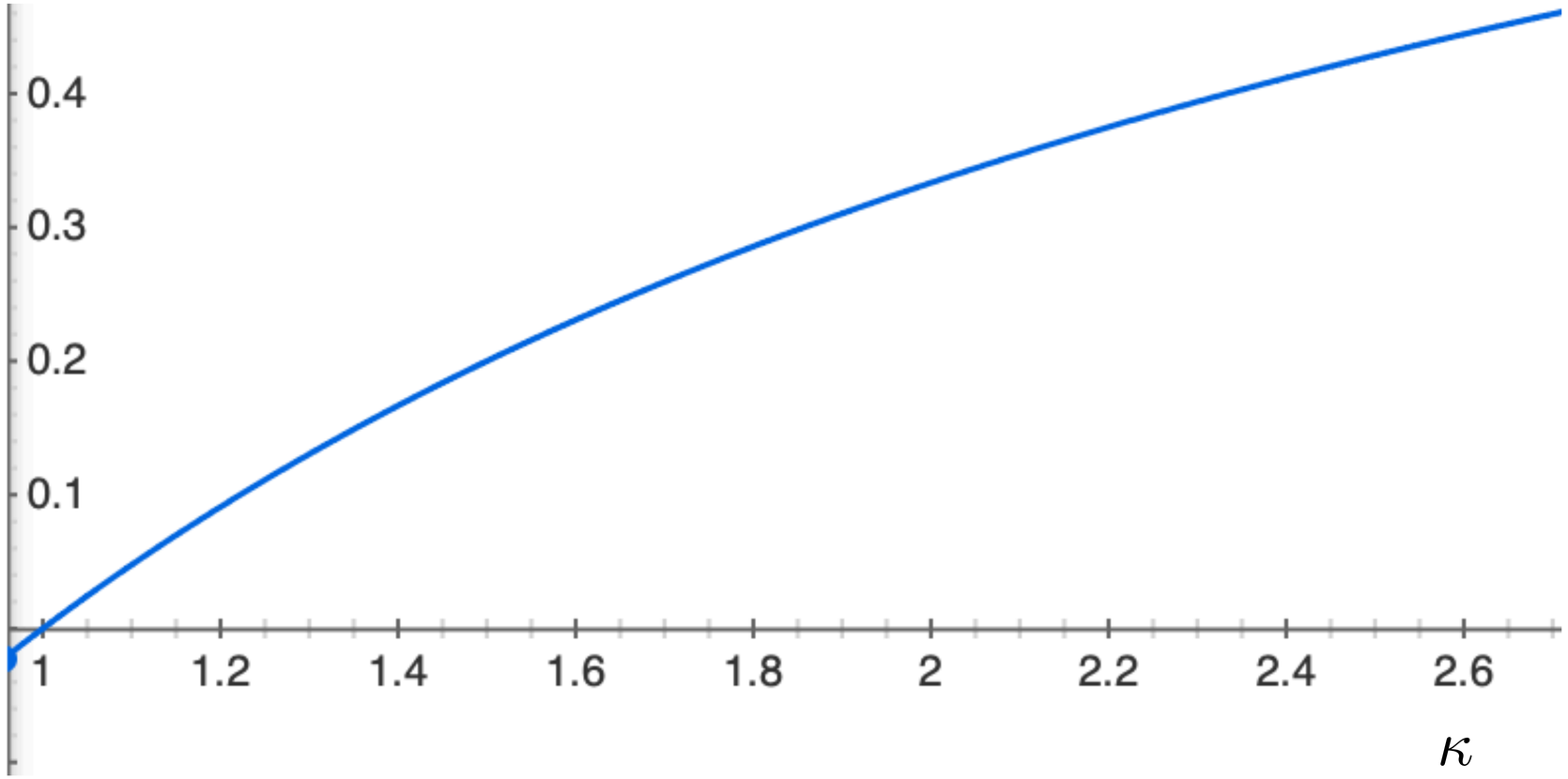
Gradient: $\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}_i}\right|_{\mathbf{w}_i = \mathbf{w}_i^{k-1}} = \lambda_i \mathbf{w}_i^{k-1}$

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

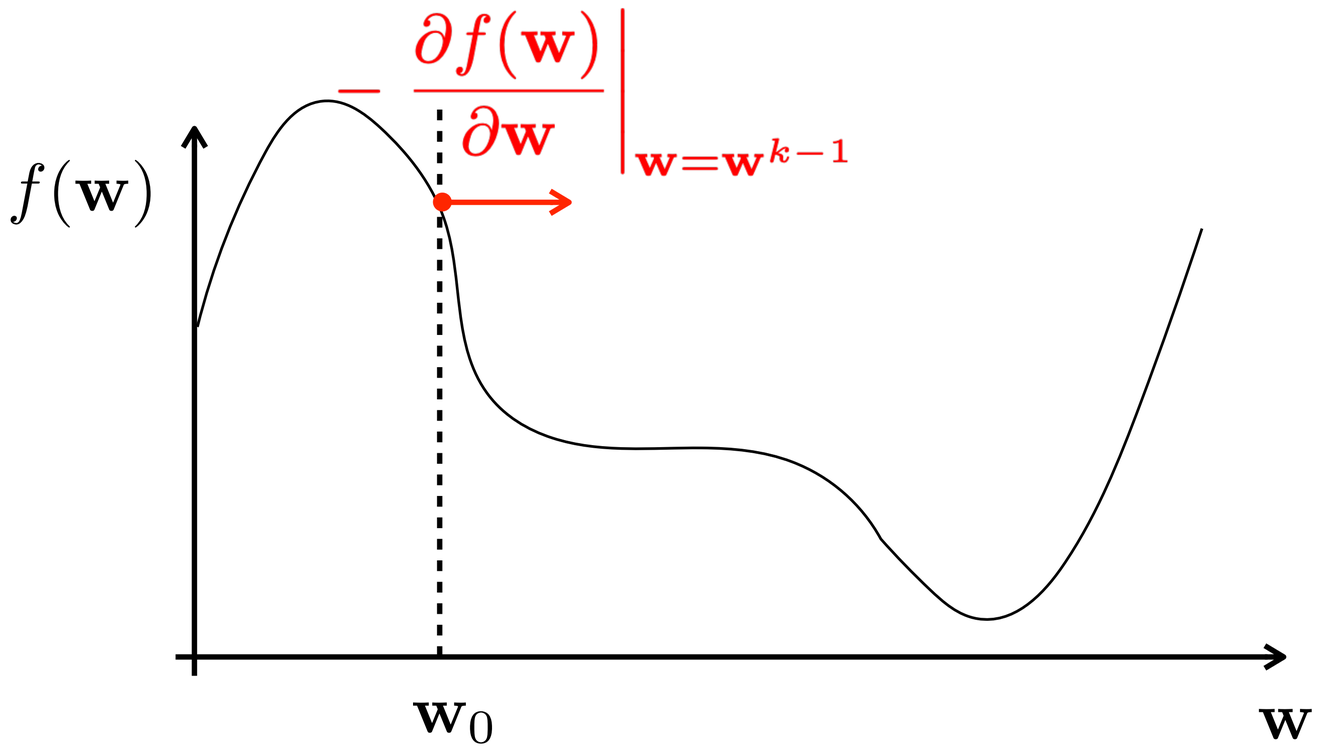
$$\frac{\kappa - 1}{\kappa + 1}$$



SGD + momentum

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

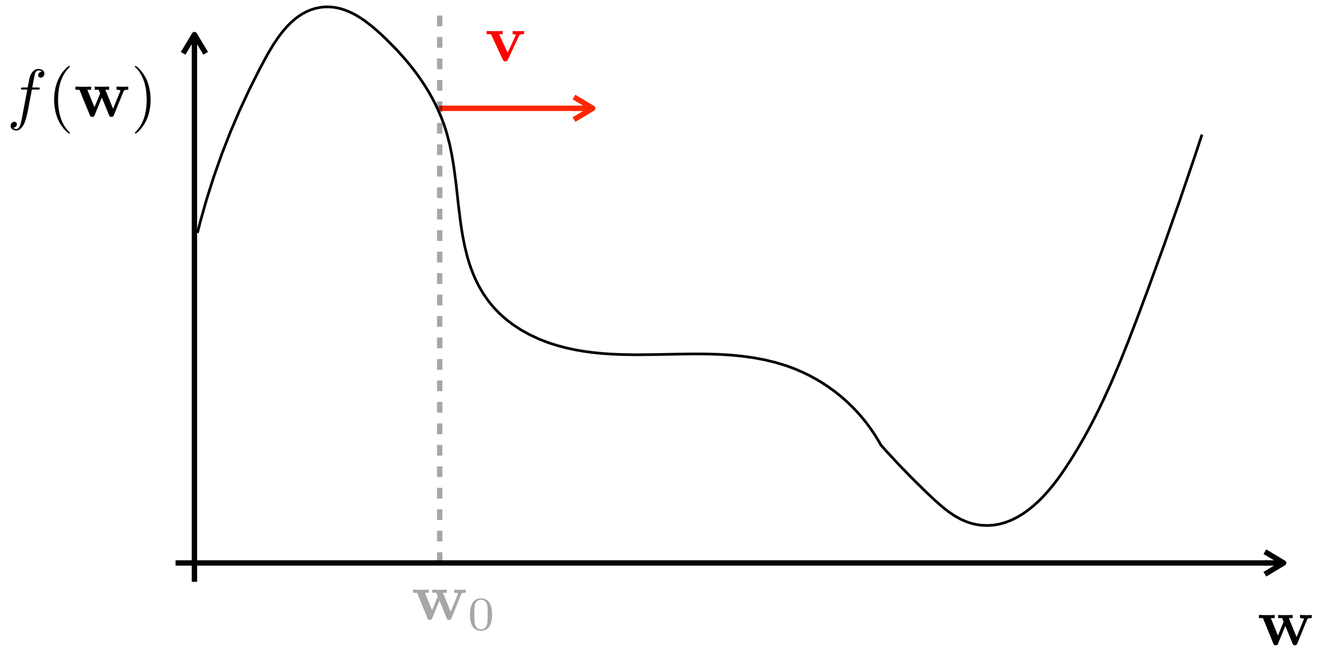
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

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

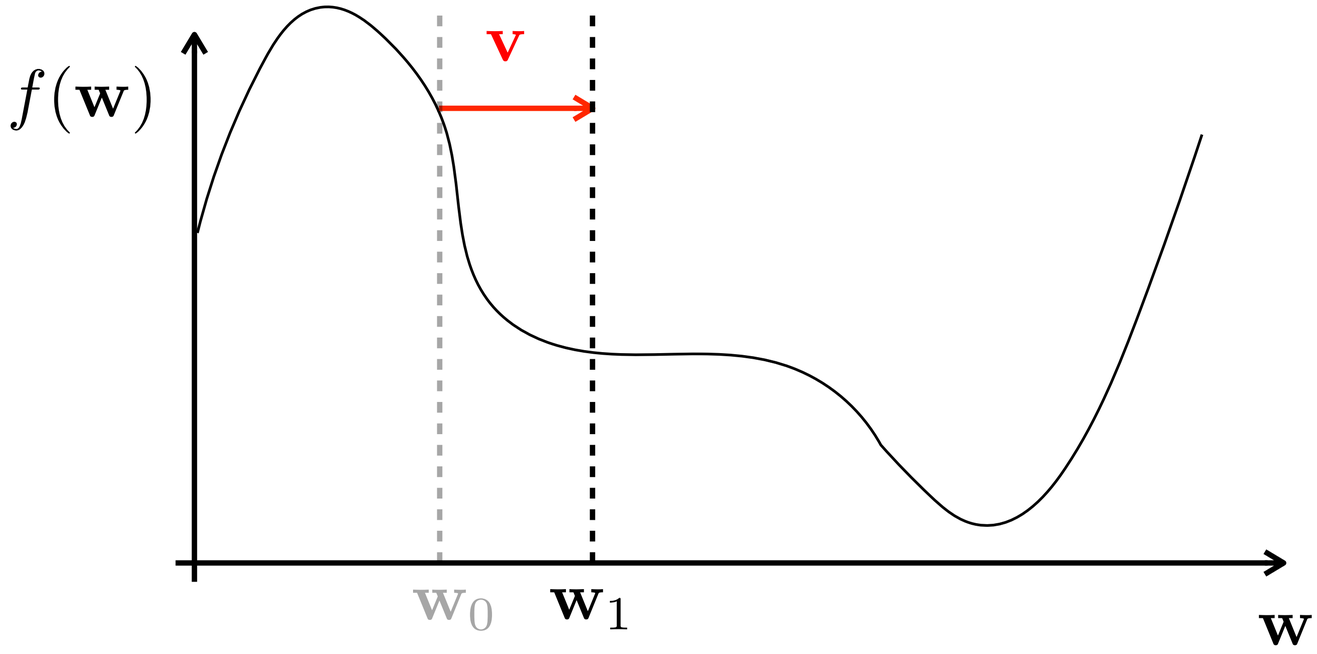
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

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

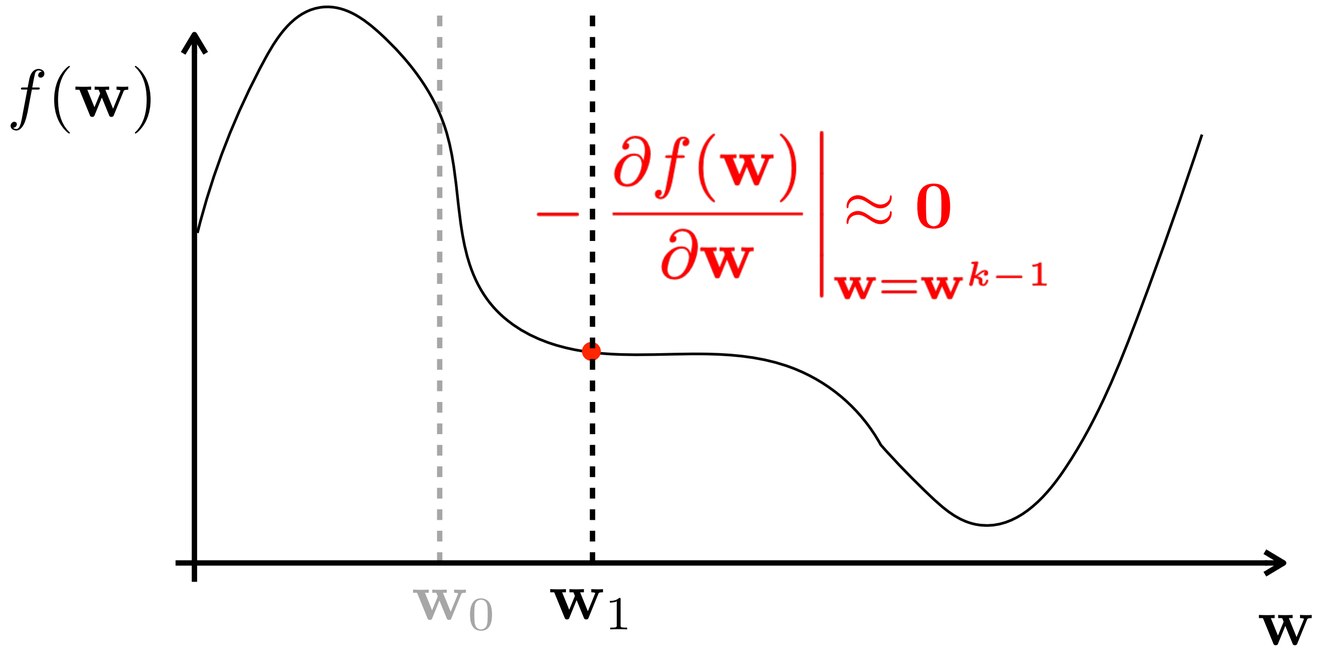
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

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

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

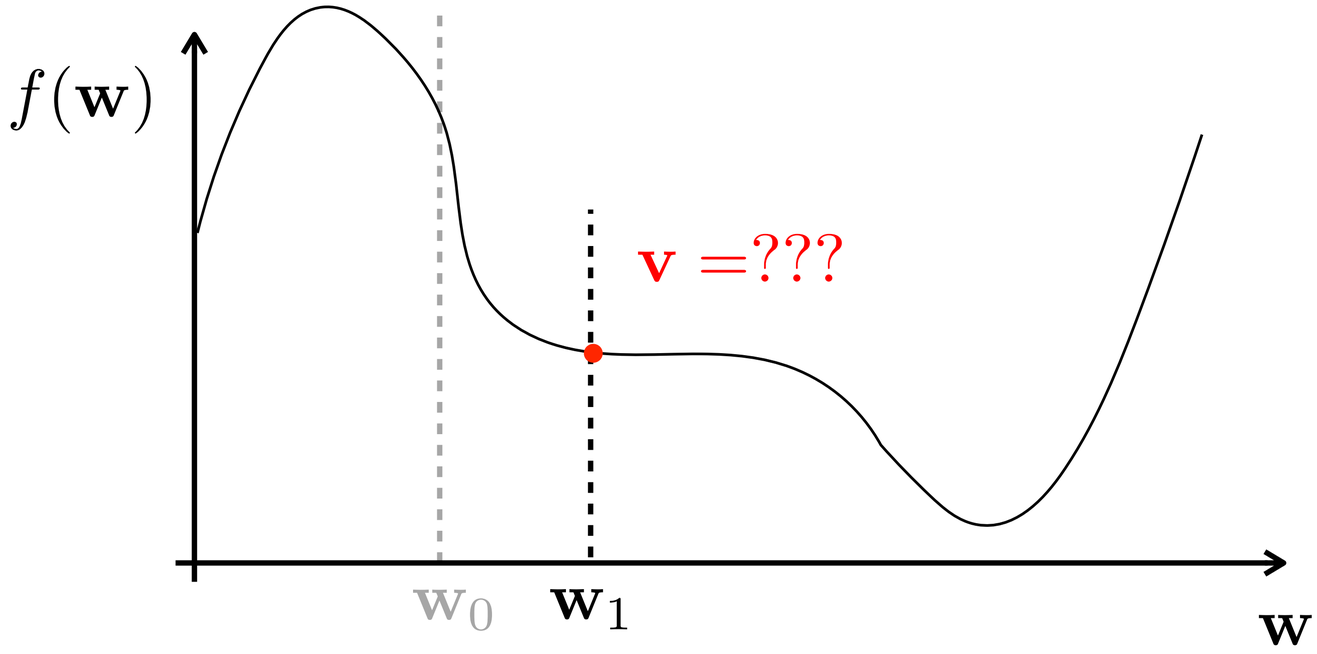


SGD + momentum

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

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

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

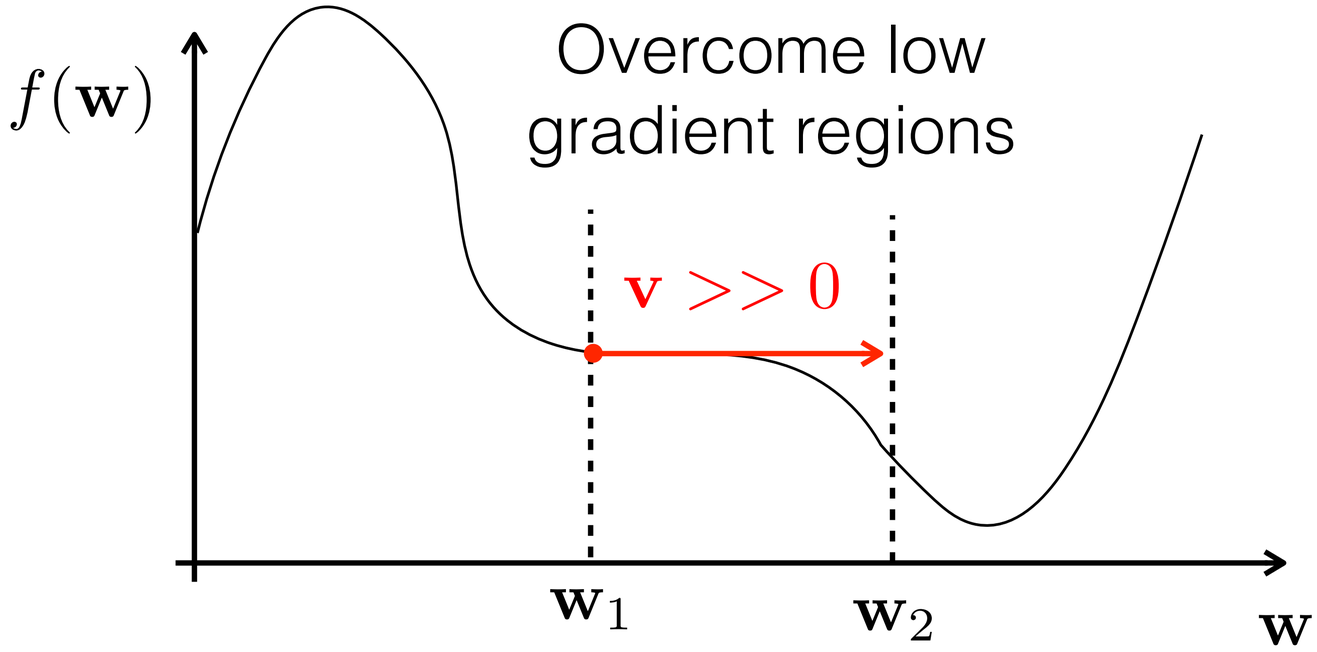


SGD + momentum

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

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

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

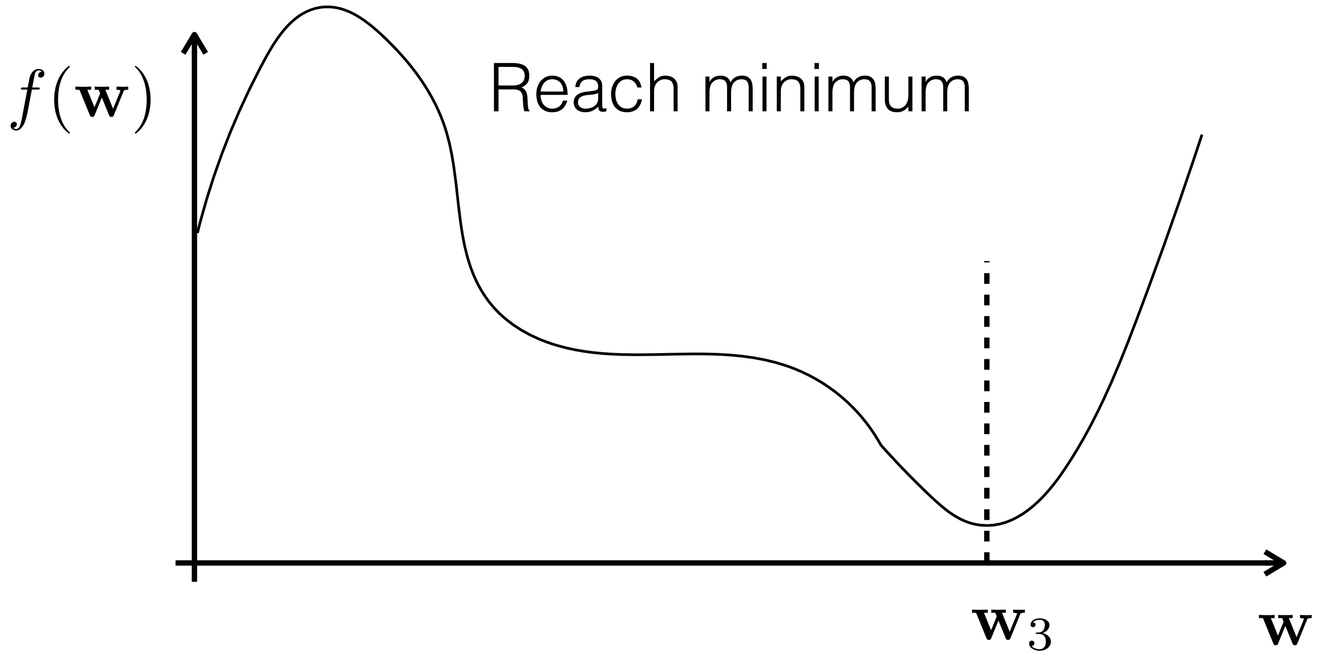


SGD + momentum

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

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

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

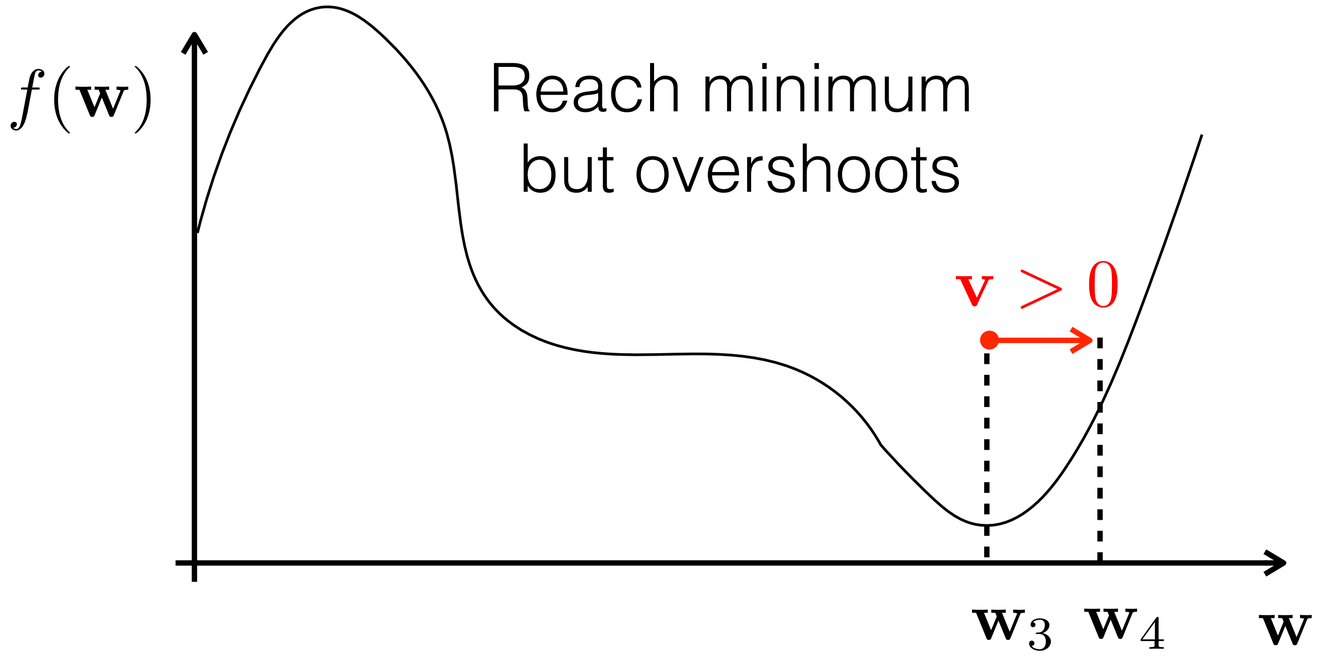


SGD + momentum

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

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

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

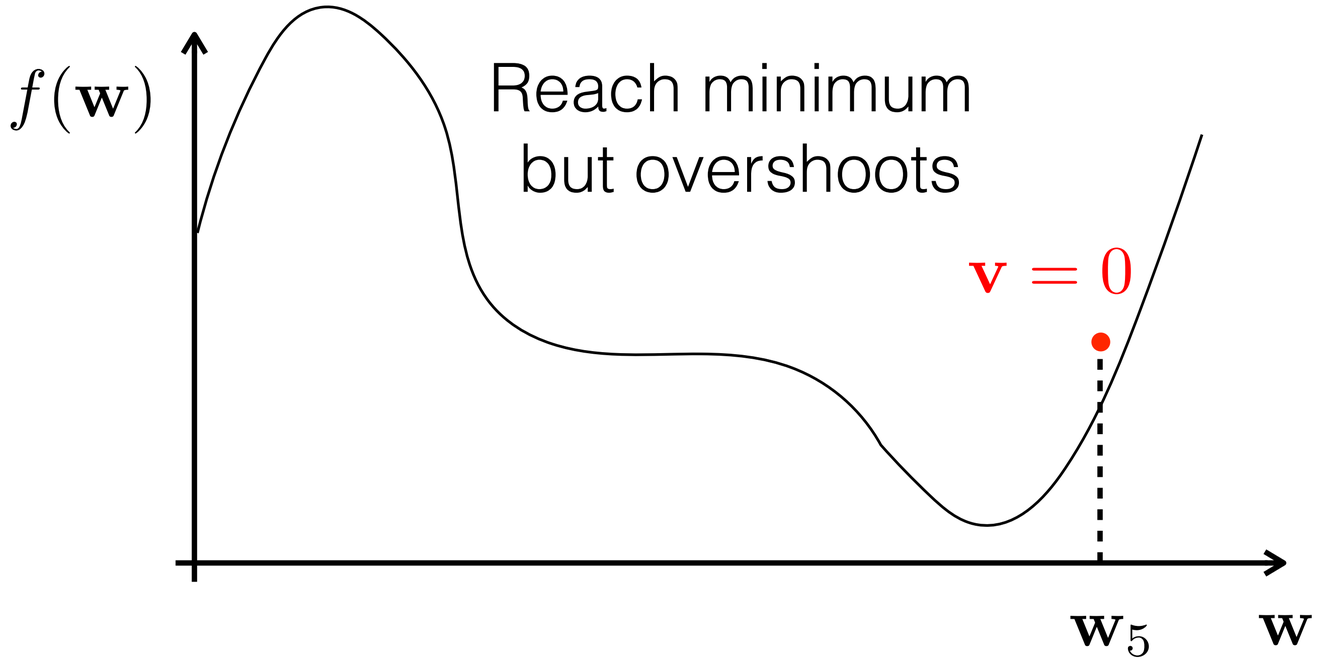


SGD + momentum

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

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

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

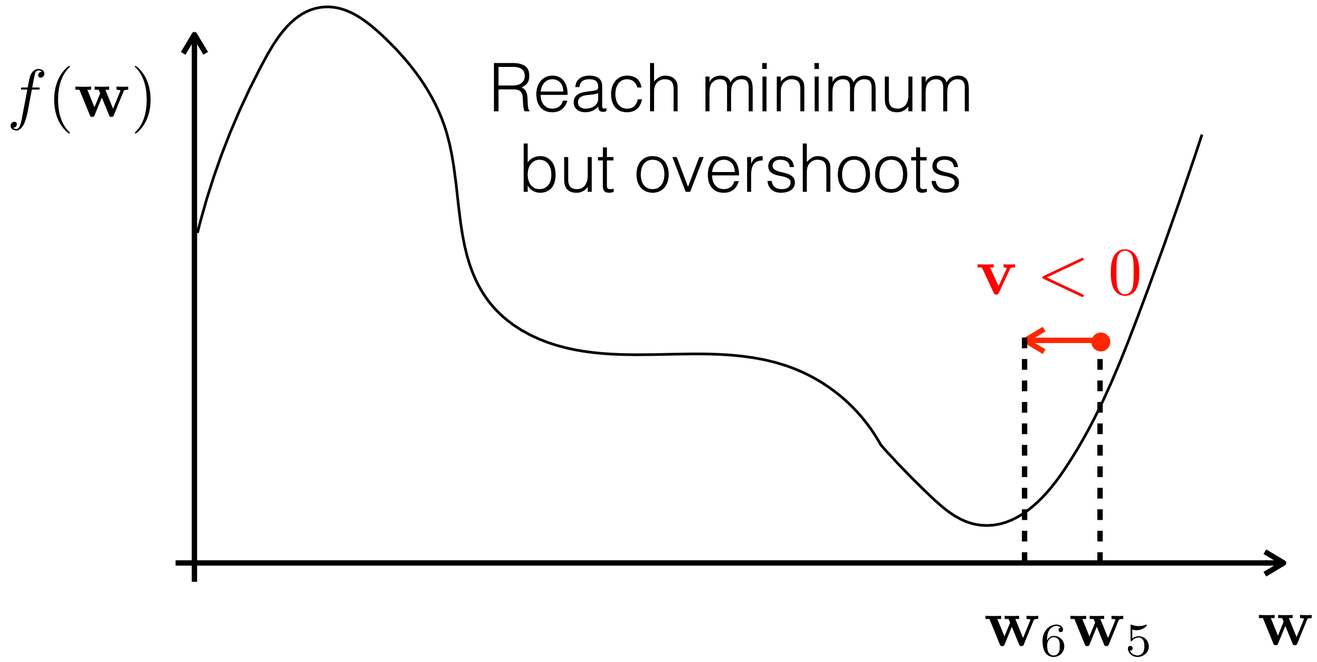


SGD + momentum

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

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

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

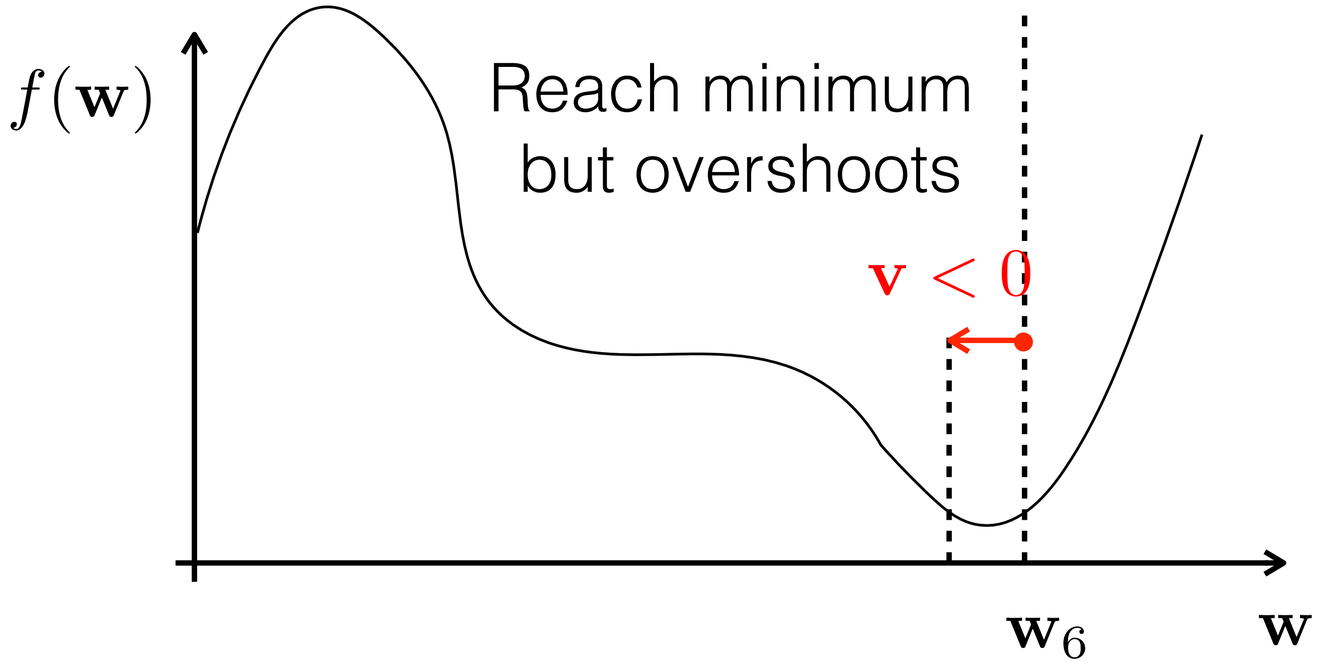


SGD + momentum

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

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

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

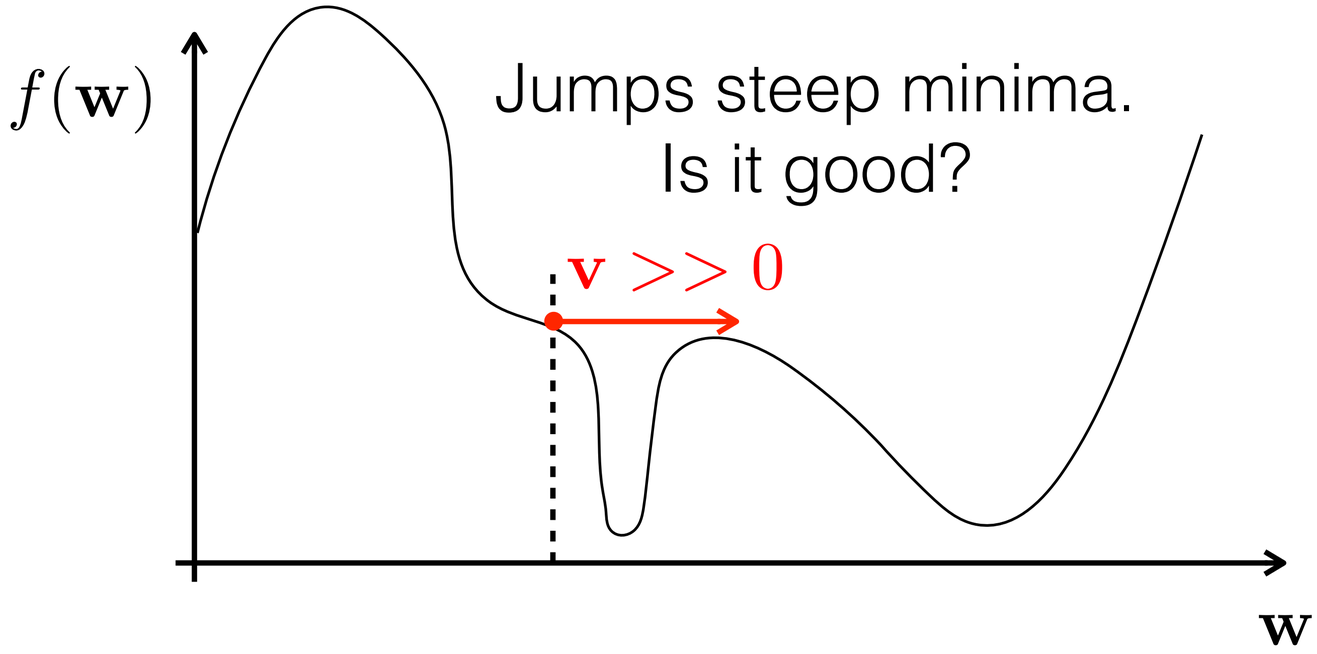


SGD + momentum

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

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

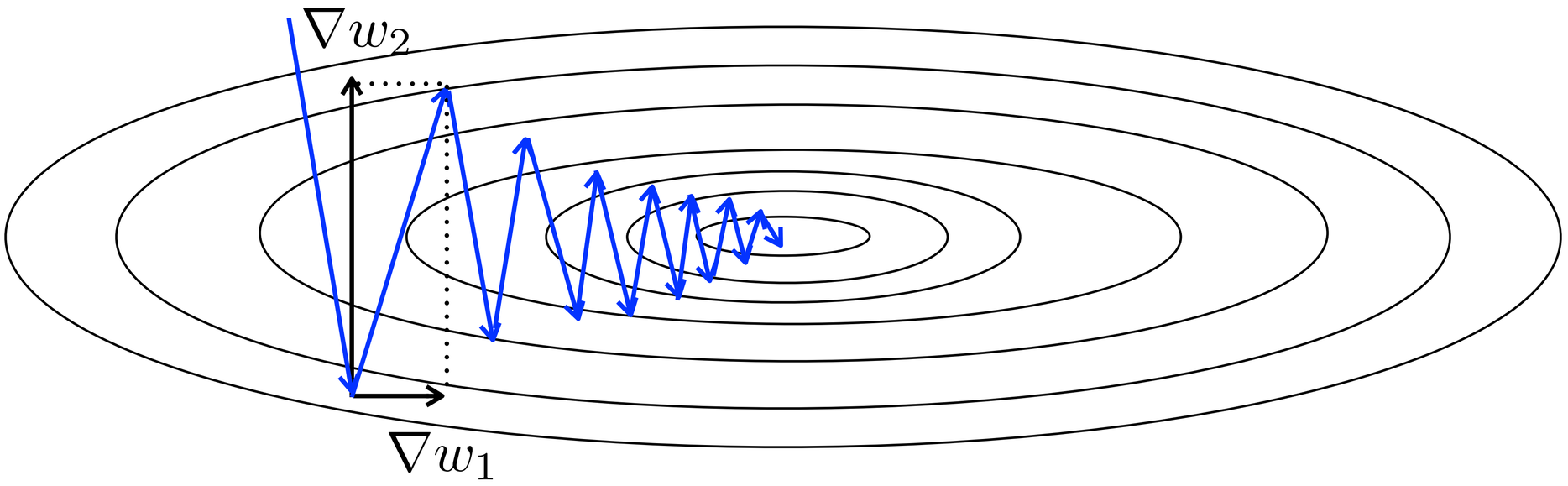
- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β



“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

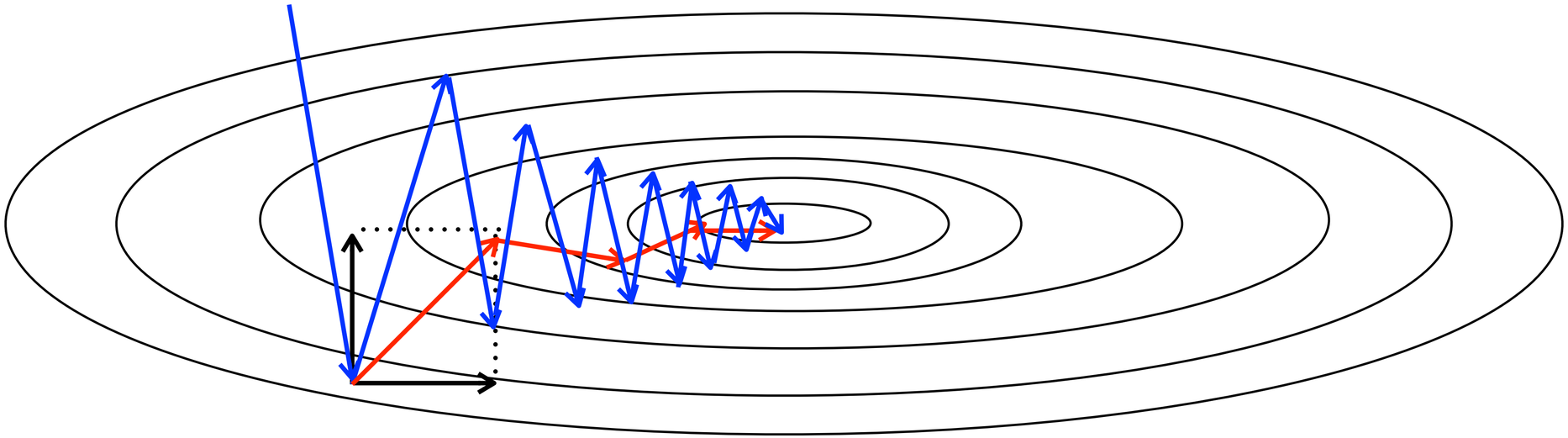


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

“SGD” vs “SGD + momentum” in 2D

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

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



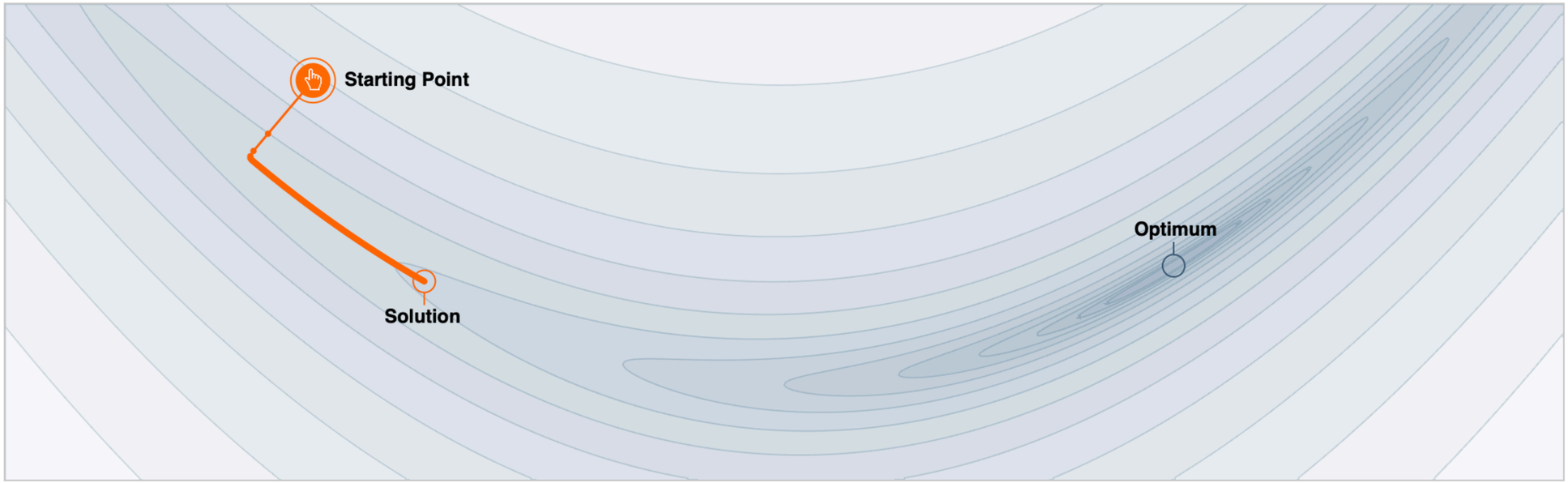
Momentum suppresses this problem partially by averaging element-wise gradients

“SGD” vs “SGD + momentum” in 2D

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

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

$\alpha = 1\text{e-}3$ $\beta = 0$



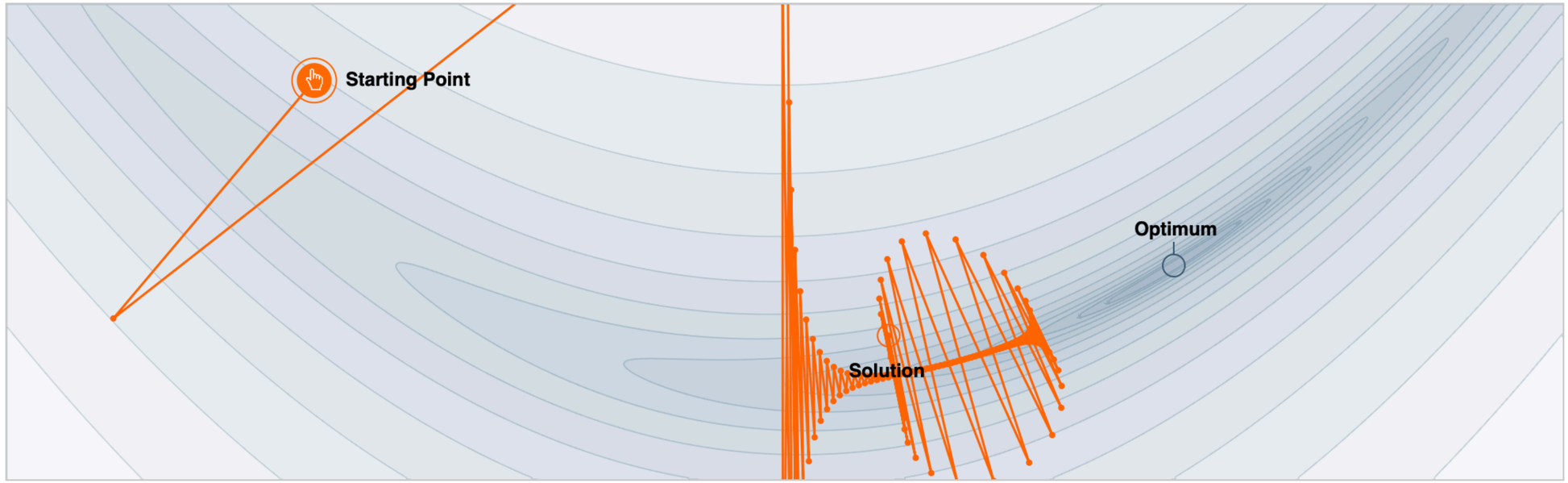
https://distill.pub/2017/momentum/

“SGD” vs “SGD + momentum” in 2D

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

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

$\alpha = 5\text{e-}3$ $\beta = 0$



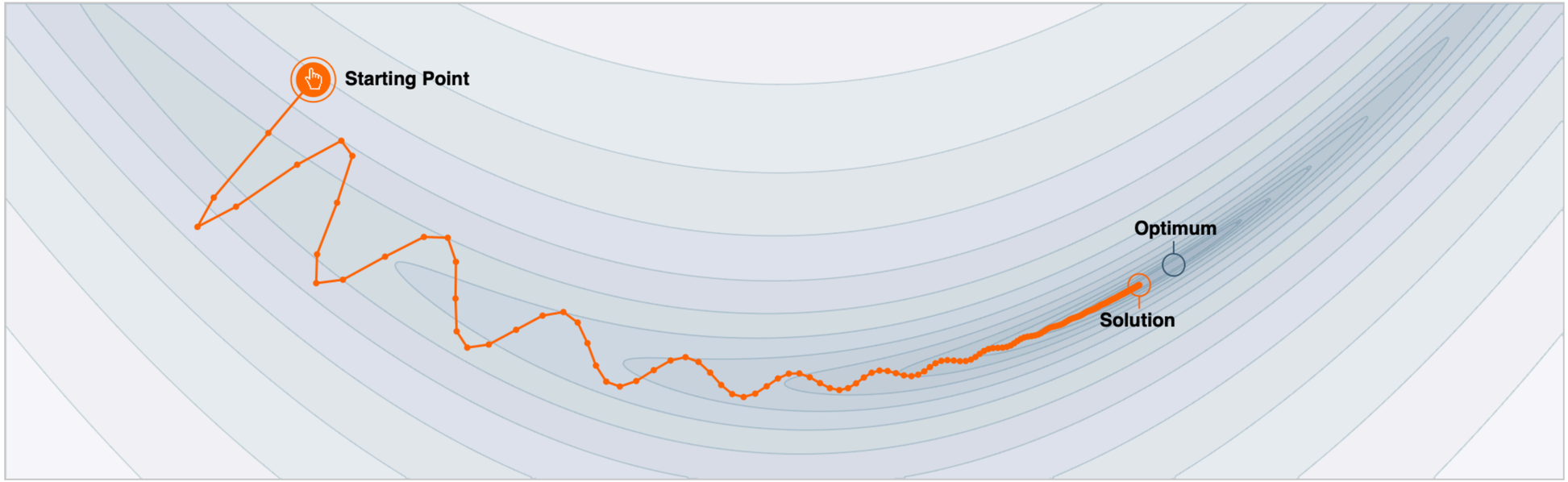
https://distill.pub/2017/momentum/

“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 + 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:

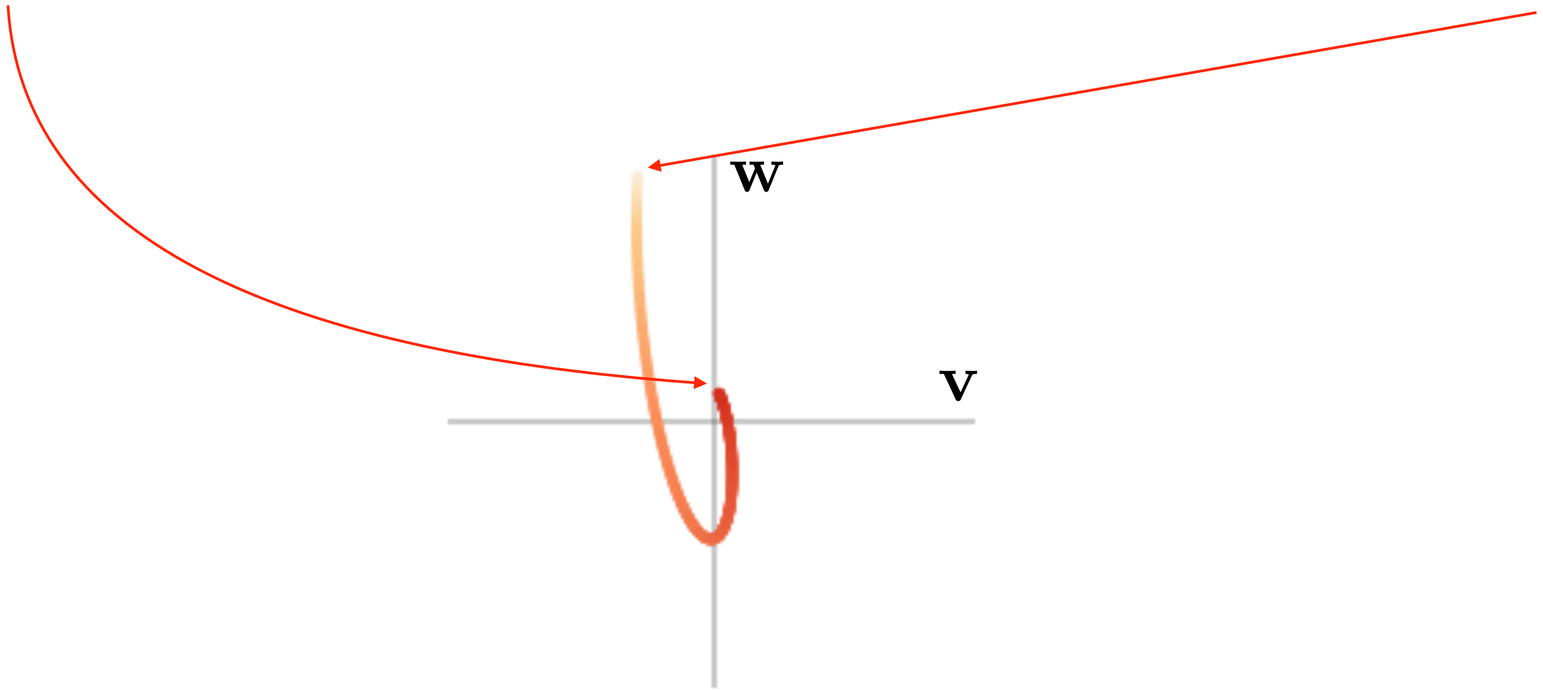
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

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

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

SGD + momentum on quadric

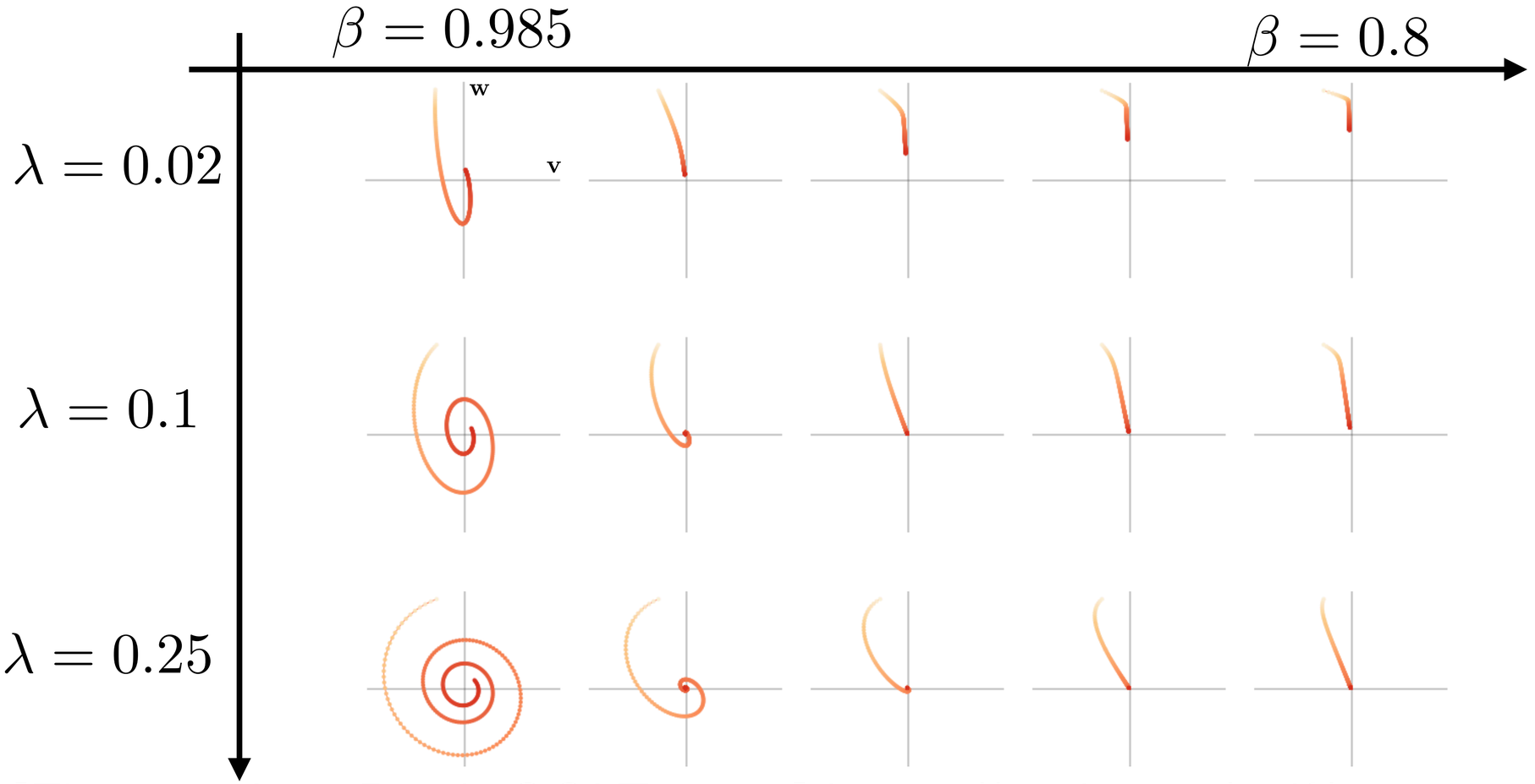
$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k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



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

SGD + momentum on quadric

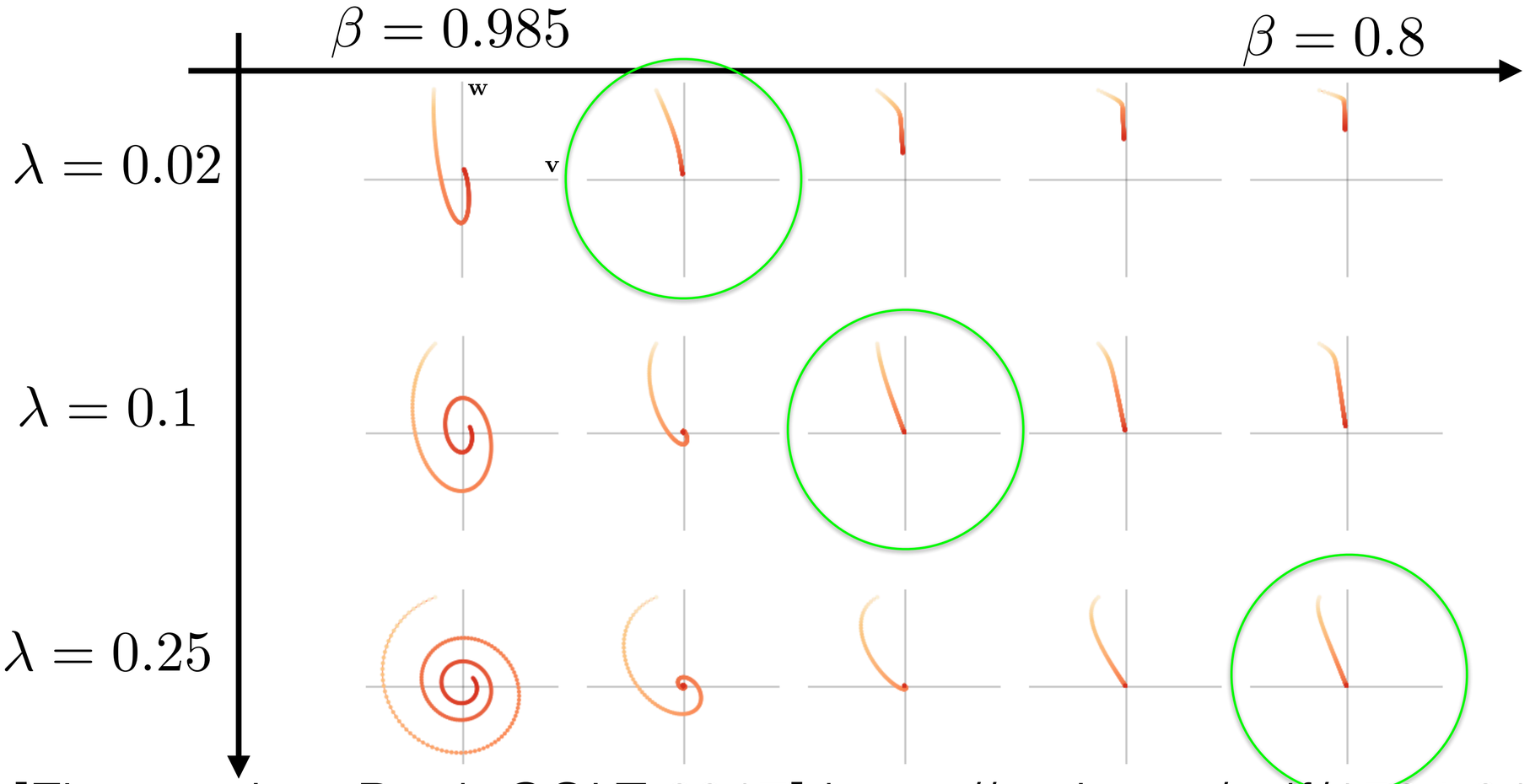
$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k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



[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:

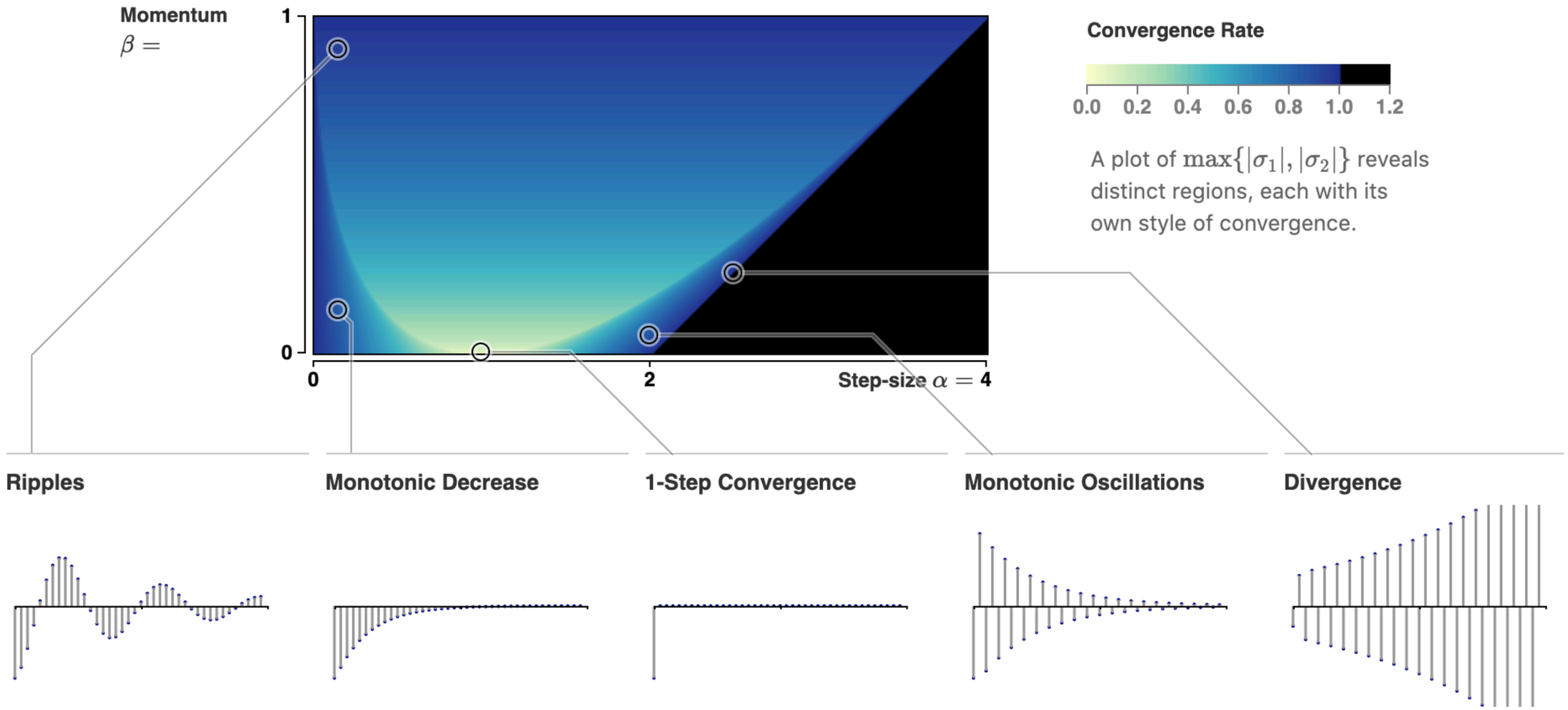
$$\begin{bmatrix} \mathbf{v}_i^k \\ \mathbf{w}_i^k \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix} \begin{bmatrix} \mathbf{v}_i^{k-1} \\ \mathbf{w}_i^{k-1} \end{bmatrix} = \begin{bmatrix} \beta & \lambda_i \\ -\alpha\beta & 1-\alpha\lambda_i \end{bmatrix}^k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

- 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:

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

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

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

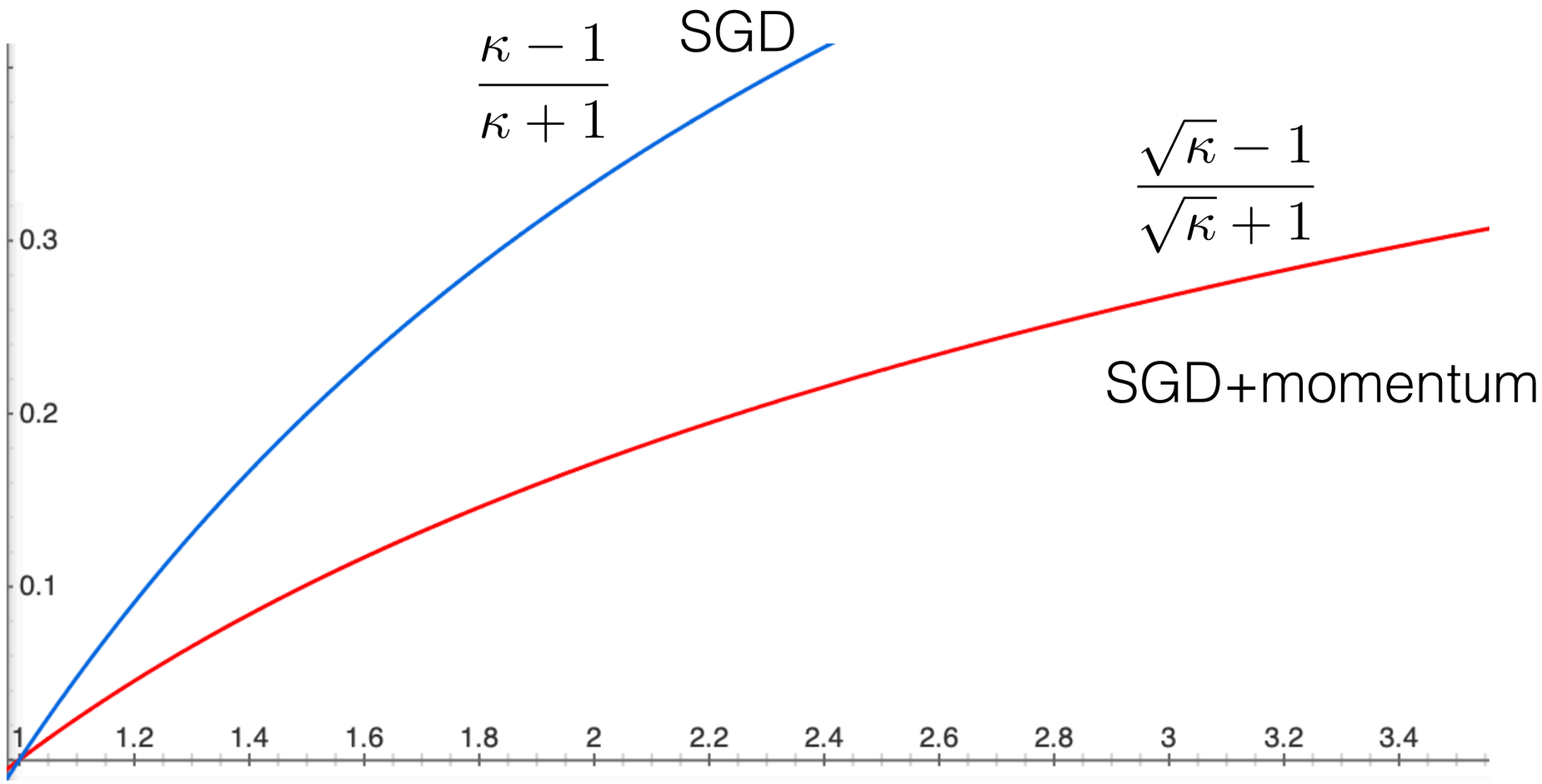
- Optimal convergence rate: $\mathrm{rate}(\alpha^*, \beta^*) = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$

[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

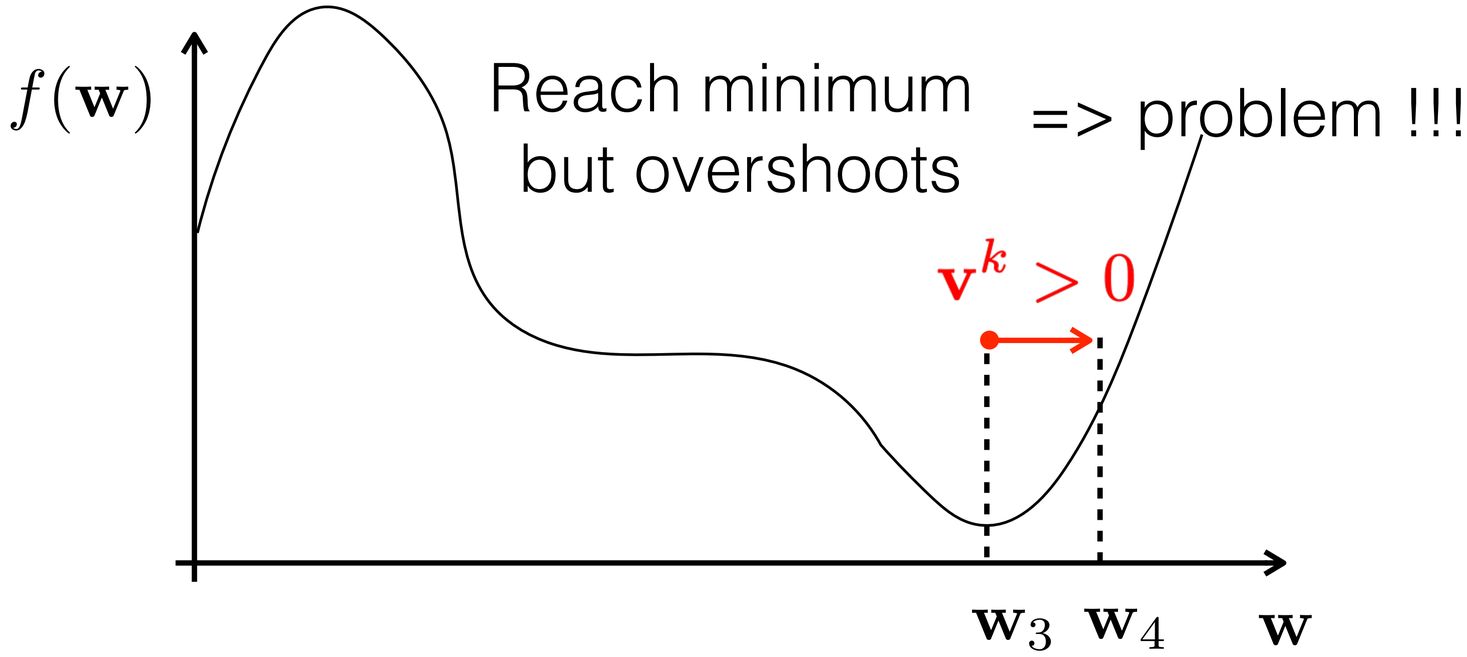
```
# initialise
import torch.nn as nn
import torch.optim as optim

# 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

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

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

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

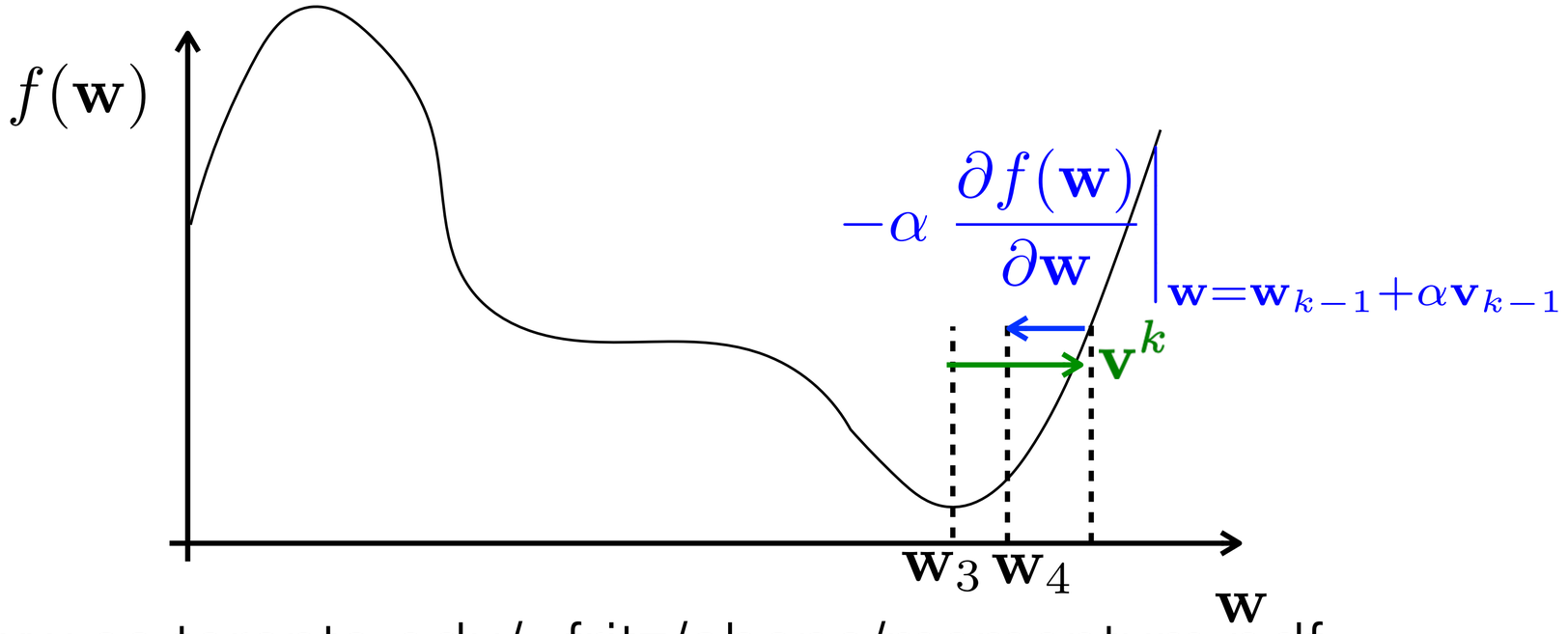


SGD with Nesterov momentum

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

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

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



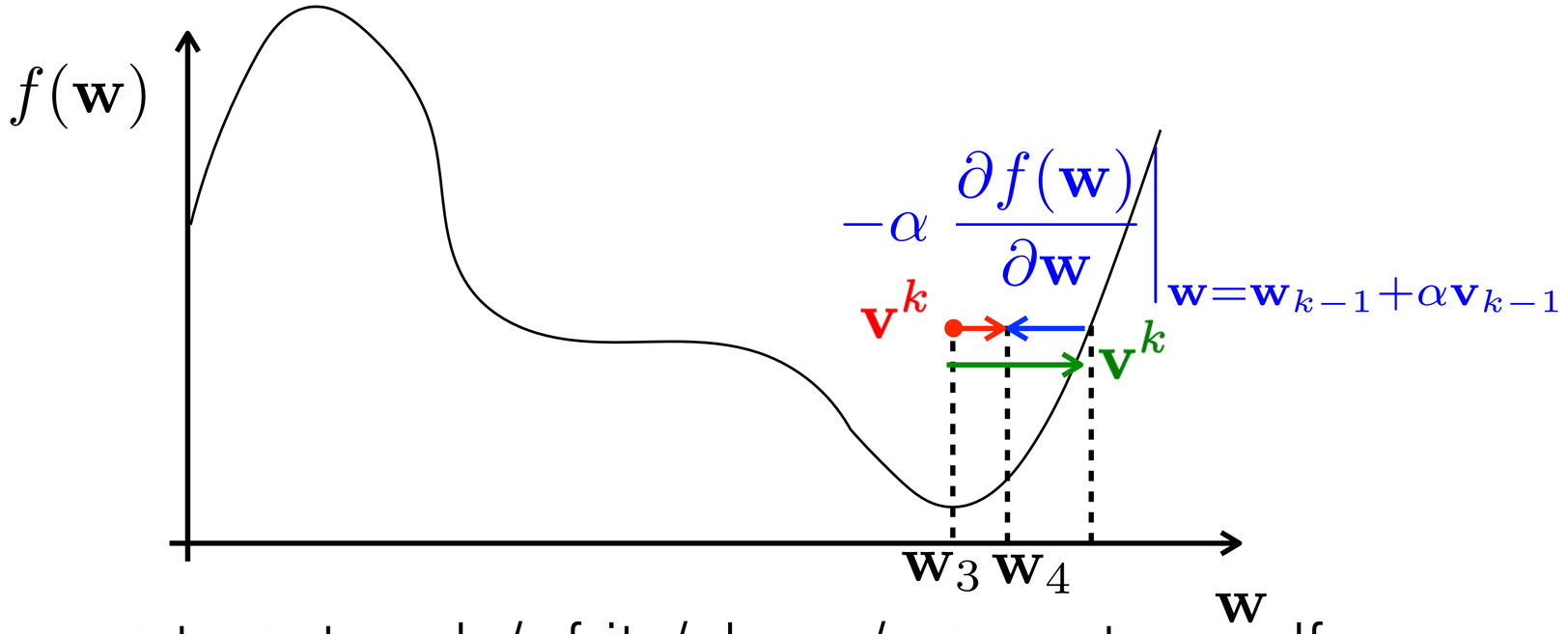
http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

SGD with Nesterov momentum

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

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

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

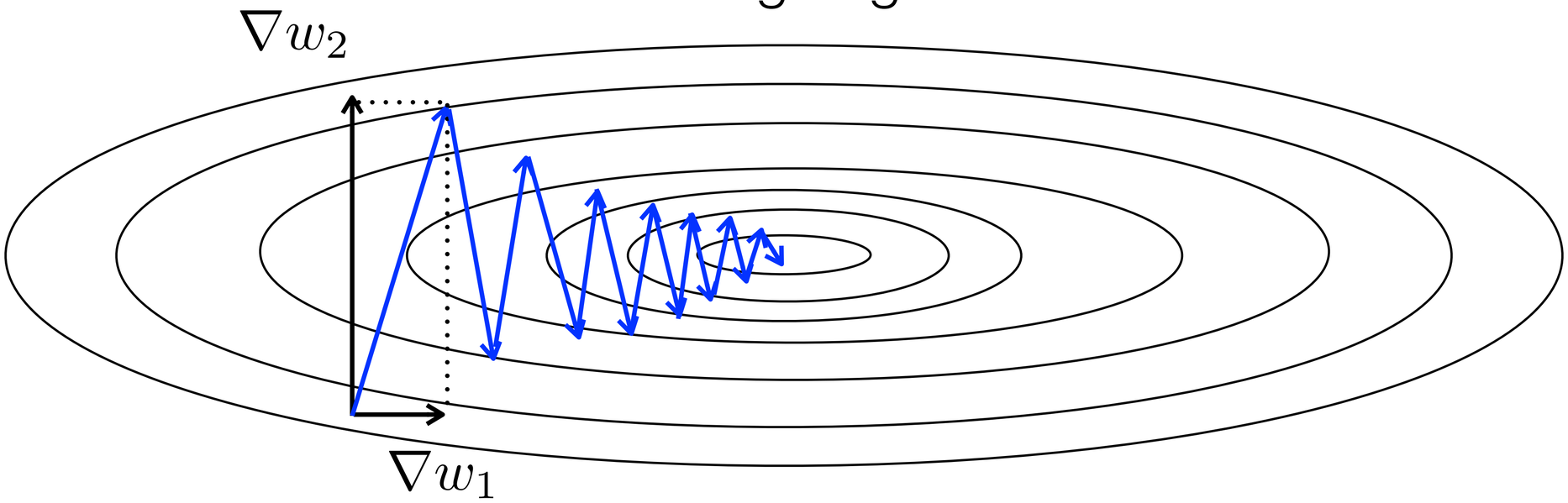


http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

Beyond first order methods

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

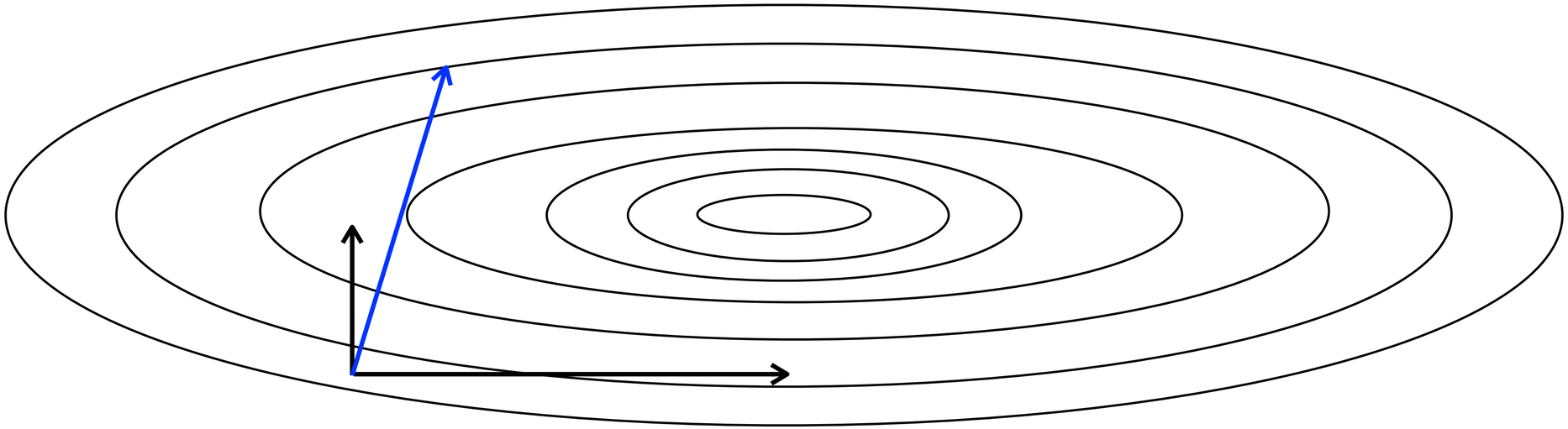
Undesired zig-zag behaviour



Momentum helps, but the zig-zag behaviour remains.

Full Newton Method

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



Hessian $H = \left. \frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$ adjust 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: $\mathbf{w}_i^k = (1 - \alpha\lambda_i)^k \mathbf{w}_i^0$

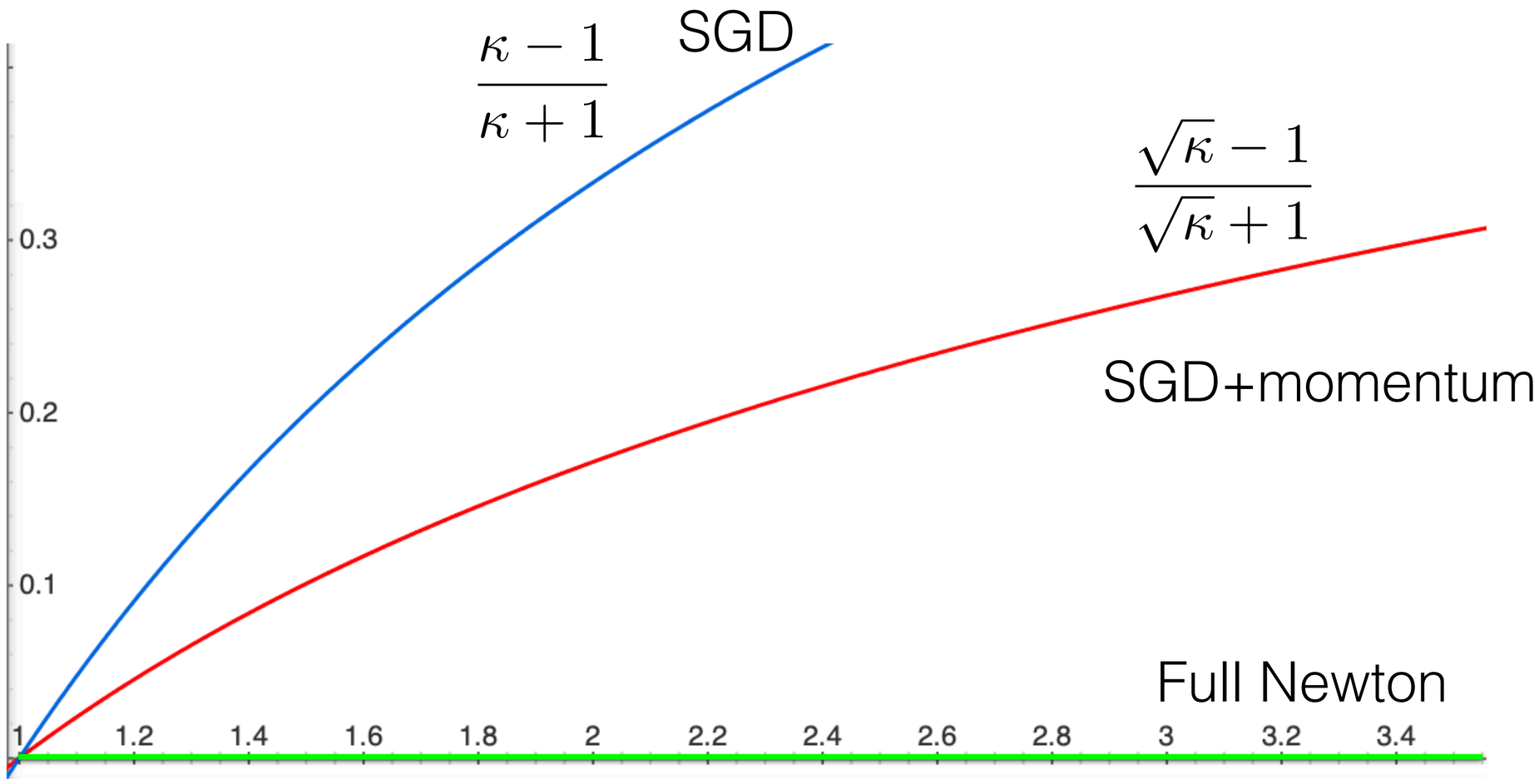
Full Newton after k iterations: $\mathbf{w}_i^k = (1 - \alpha)^k \mathbf{w}_i^0$

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

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

SGD + momentum on quadric

Convergence rate

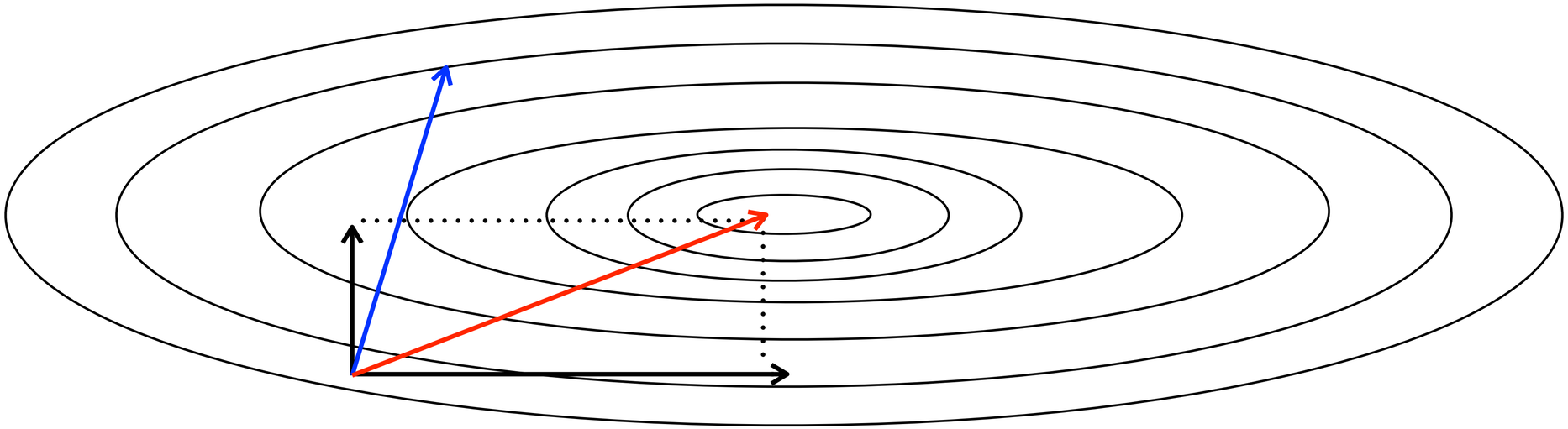



```
torch.optim.SGD(params, lr=0.001, momentum=0.9)
```

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)

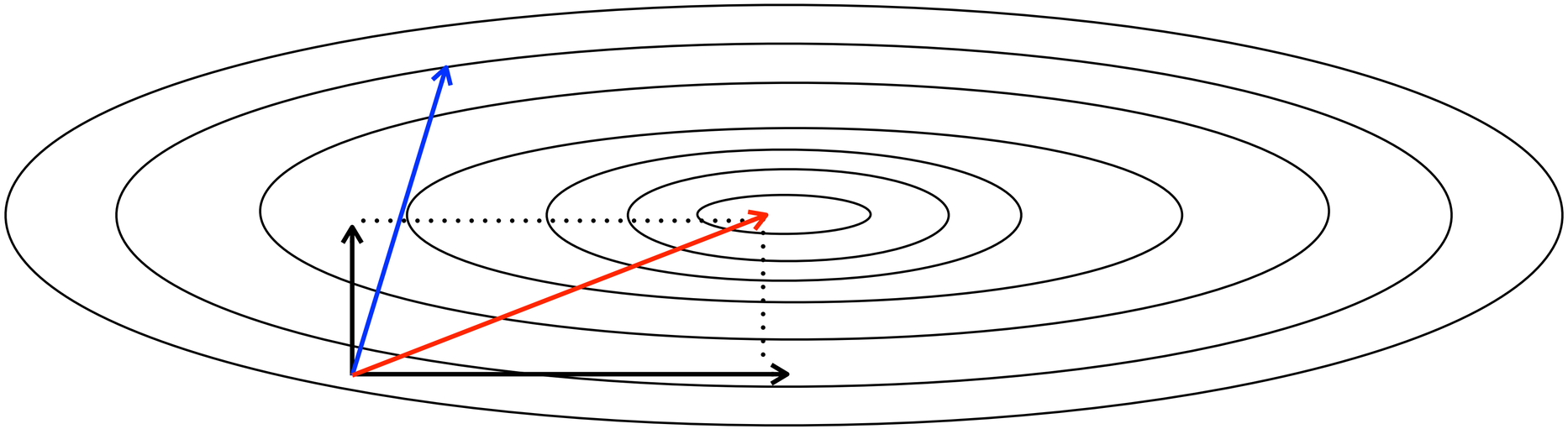


- Why not to use Hessian?

Full Newton Method

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

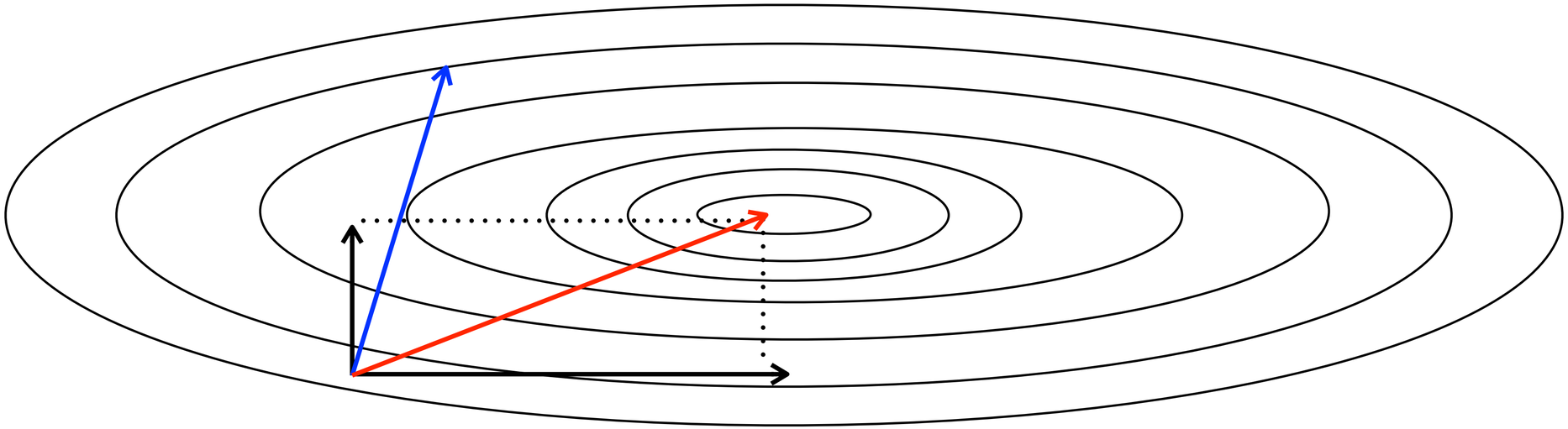
Convergence rate for convex quadratic form is zero (converges within one step)



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)

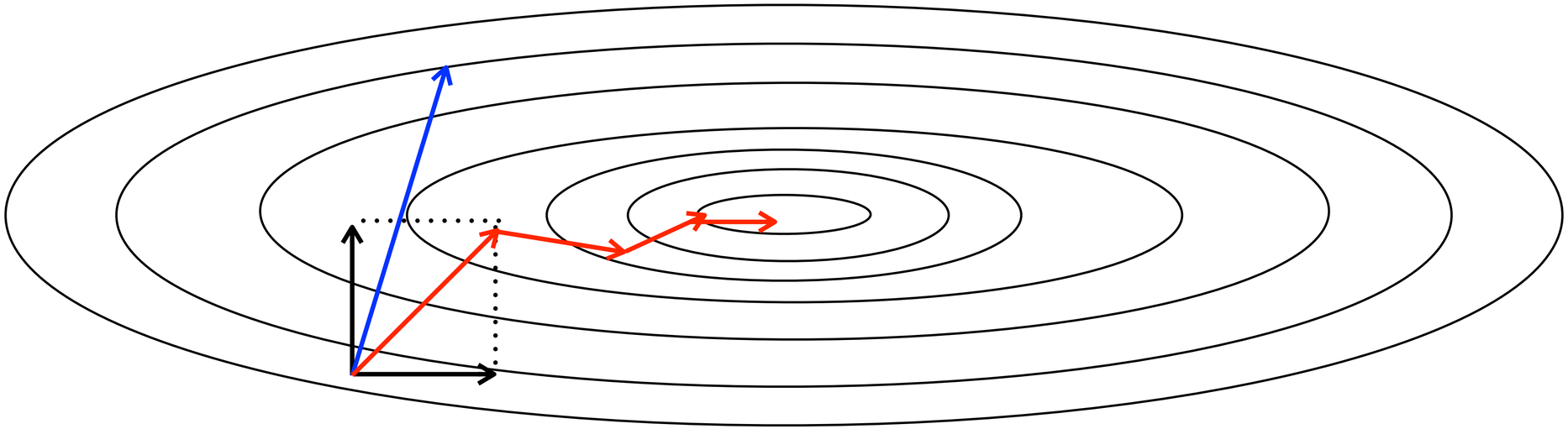


What does the Hessian actually do?

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

Full Newton method - approximation

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



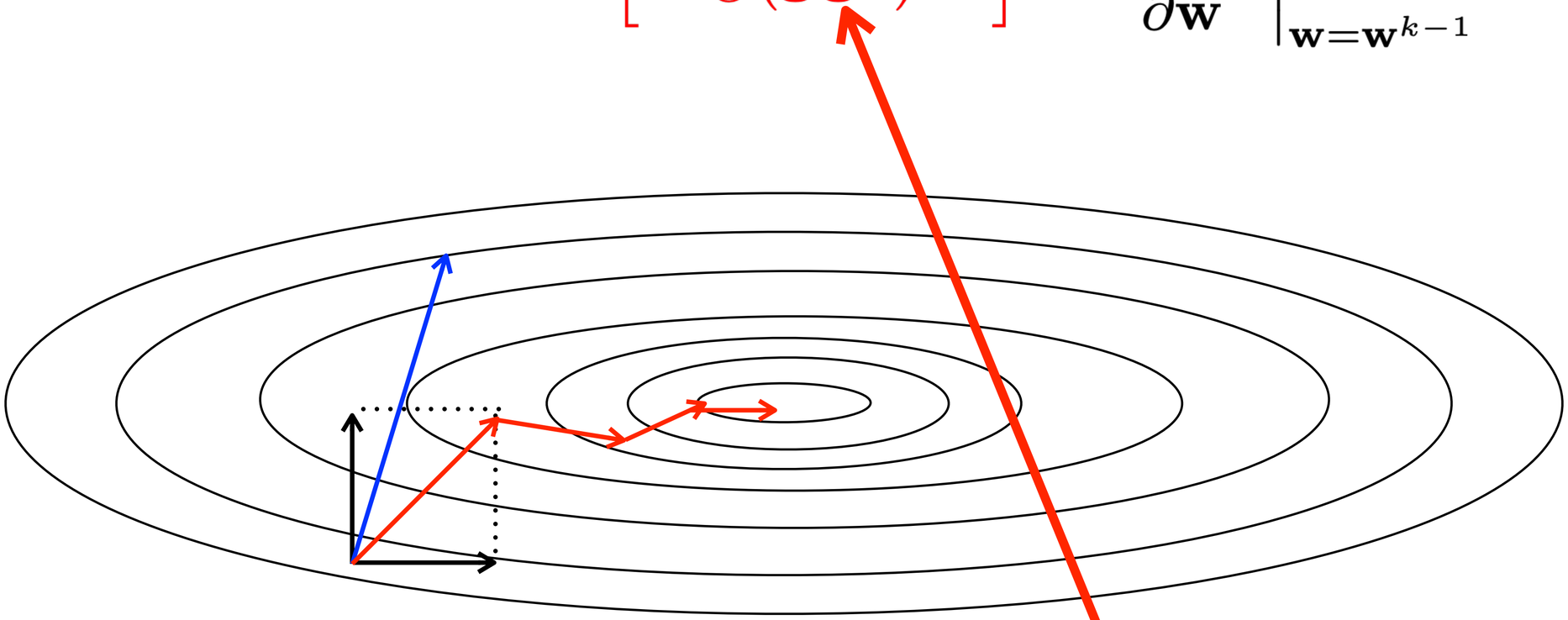
- Approximate Hessian as $\hat{H} \approx \operatorname{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}$

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

Fisher information matrix

Full Newton method - approximation

$$\mathbf{w}^k \approx \mathbf{w}^{k-1} - \alpha \left[\operatorname{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}\right]^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



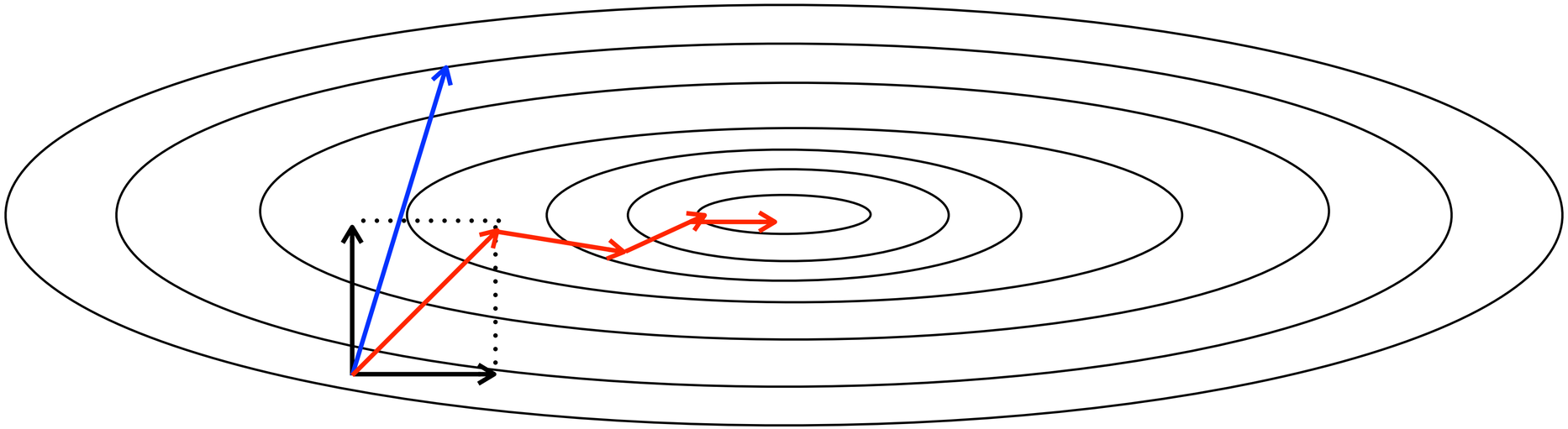
- Approximate Hessian as $\hat{H} \approx \operatorname{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}$

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

Full Newton method - approximation

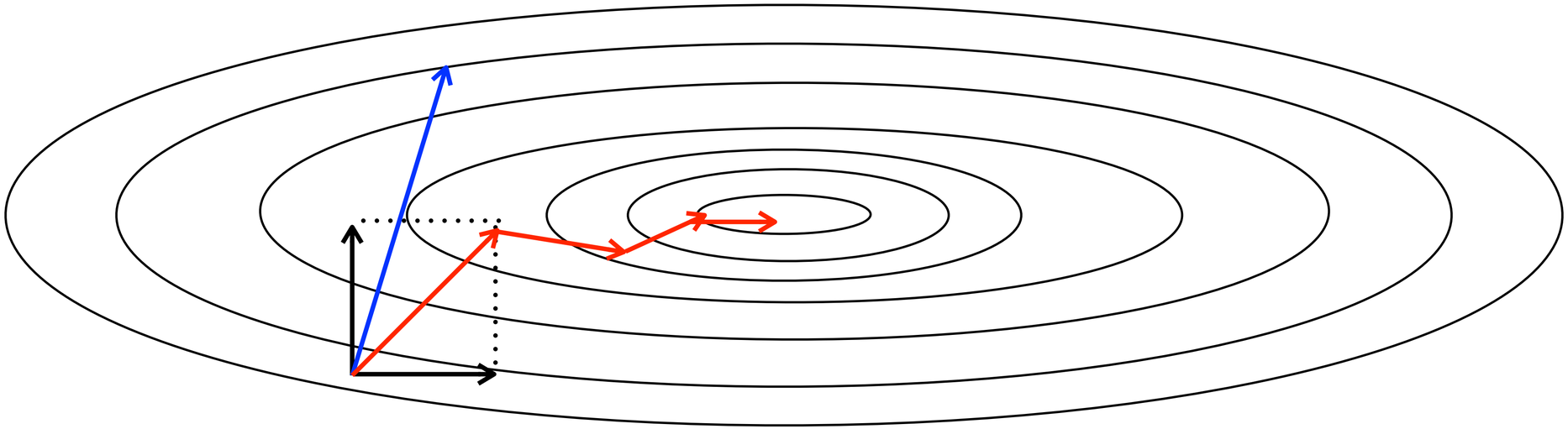
$$\mathbf{w}^k \approx \mathbf{w}^{k-1} - \alpha \left[\operatorname{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}\right]^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

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



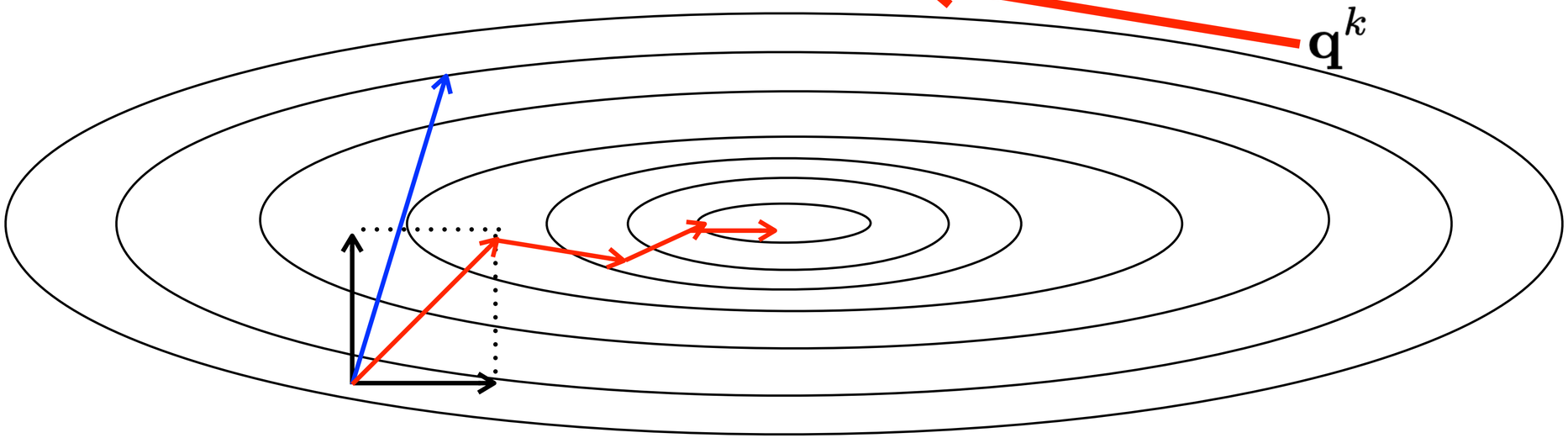
Full Newton method - approximation

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



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 $\mathbf{g},\ \mathbf{g}^2$

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

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

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 $\mathbf{g},\ \mathbf{g}^2$

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

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

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

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

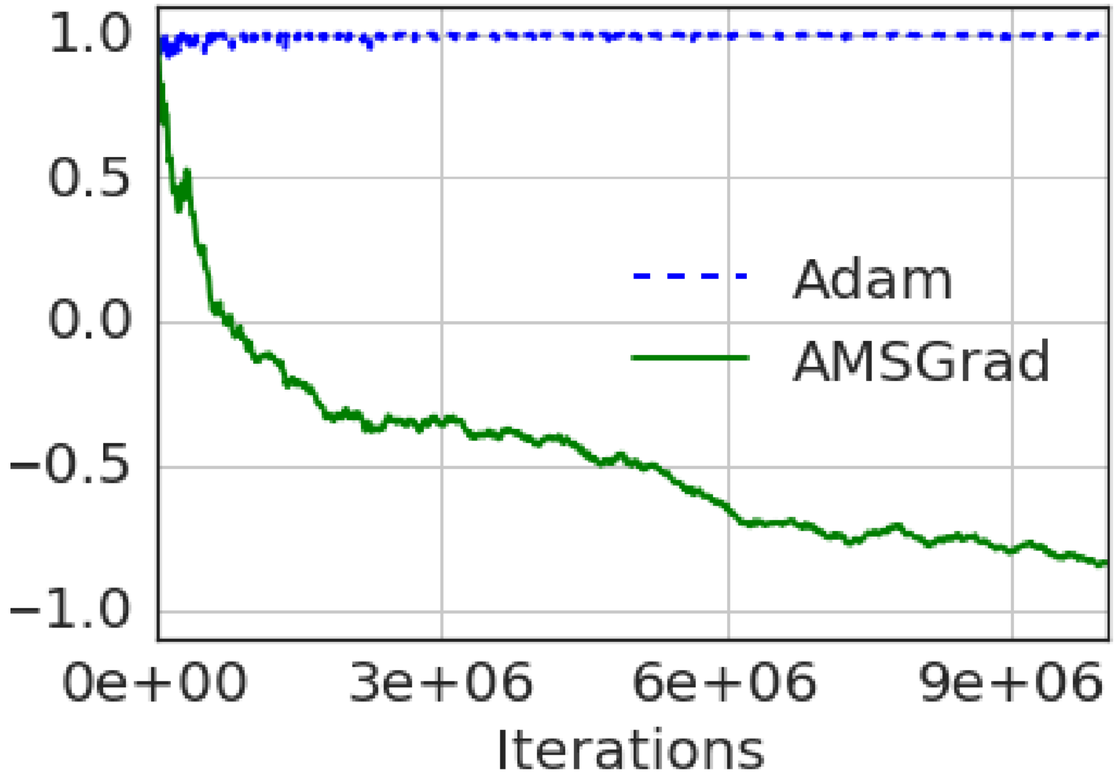
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\hat{\mathbf{q}}^k} + \epsilon} \odot \hat{\mathbf{v}}^k$$

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)
```

AMSgrad

For $\beta_1 < \sqrt{\beta_2}$ there always exists a stochastic optimization problem, where Adam fails.



[Reddi ICLR 2018]
https://openreview.net/pdf?id=ryQu7f-RZ

Adam

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

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

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

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

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

[Kingma ICLR 2015]

AMSgrad

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

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

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

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

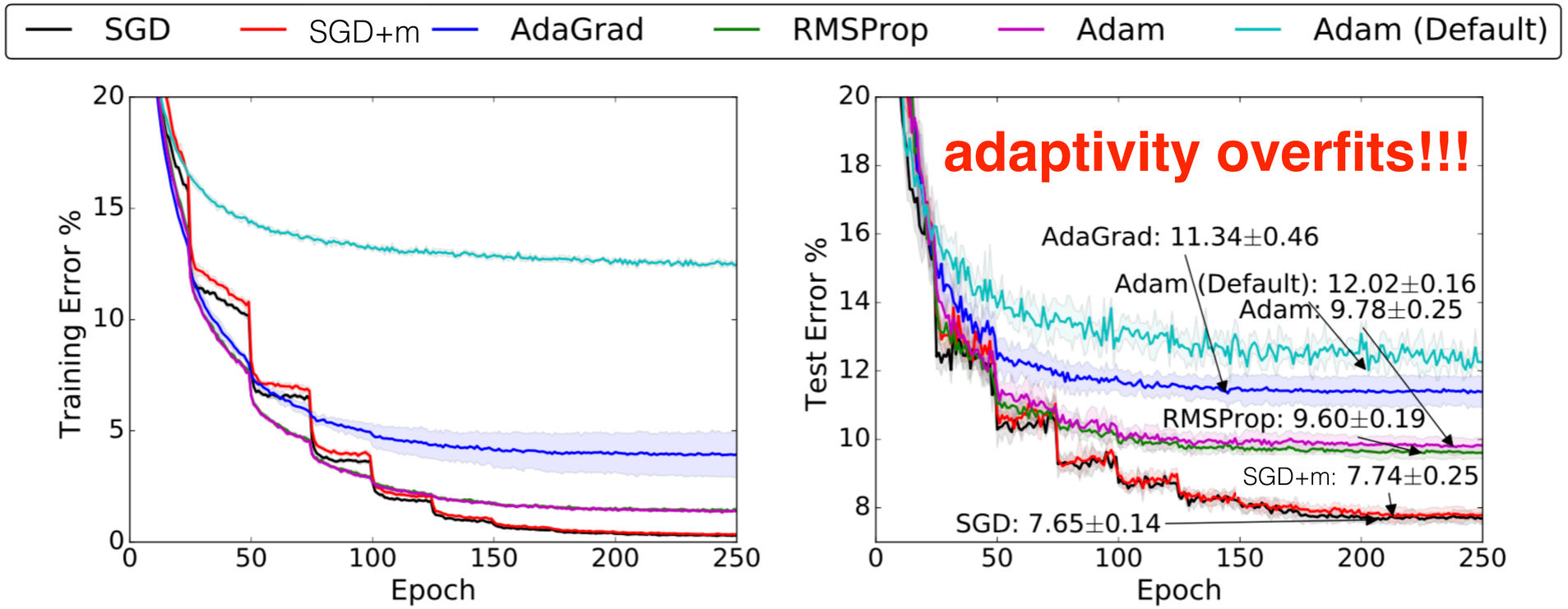
$$\hat{\mathbf{q}}^k = \max\{\hat{\mathbf{q}}^k, \hat{\mathbf{q}}^{k-1}\}$$

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

[Reddi ICLR 2018]
https://openreview.net/pdf?id=ryQu7f-RZ

Summary

- However careful setting of other hyper-parameters makes not-adaptive method work similarly or better https://arxiv.org/pdf/1705.08292.pdf [NIPS 2017]



Summary

- Adam or AMSgrad 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)
```

- There is a whole family of Quasi-Newton methods, which make use of advanced Hessian approximations (L-BFGS). However, their applicability for huge state-of-the-art networks is discussable.
https://arxiv.org/abs/1805.02338v1

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

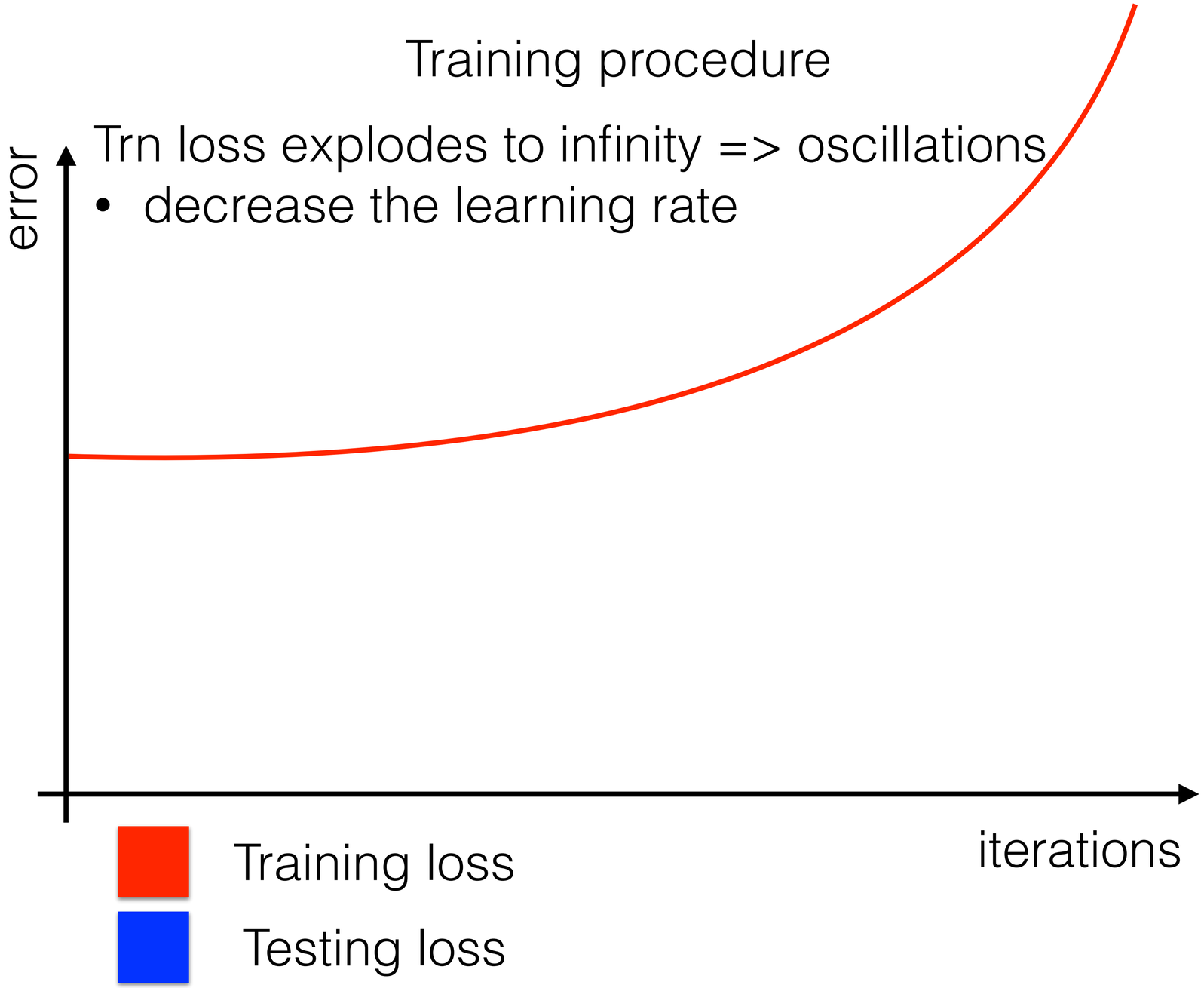
Training procedure

- 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

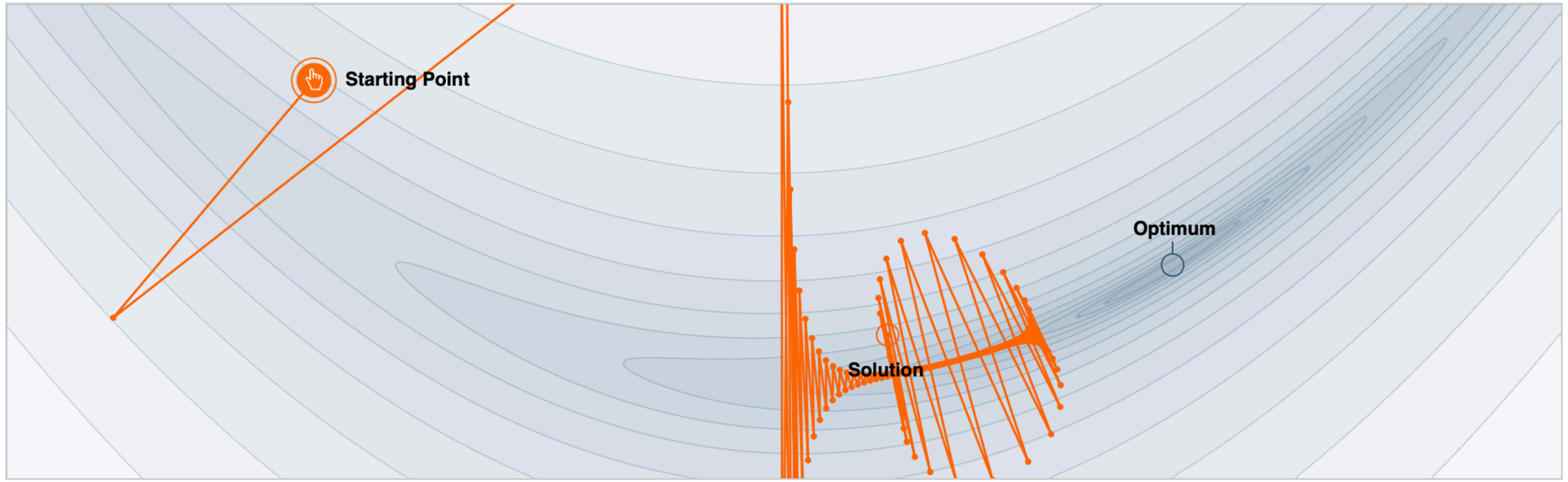
Trn loss explodes to infinity => oscillations

- decrease the learning rate



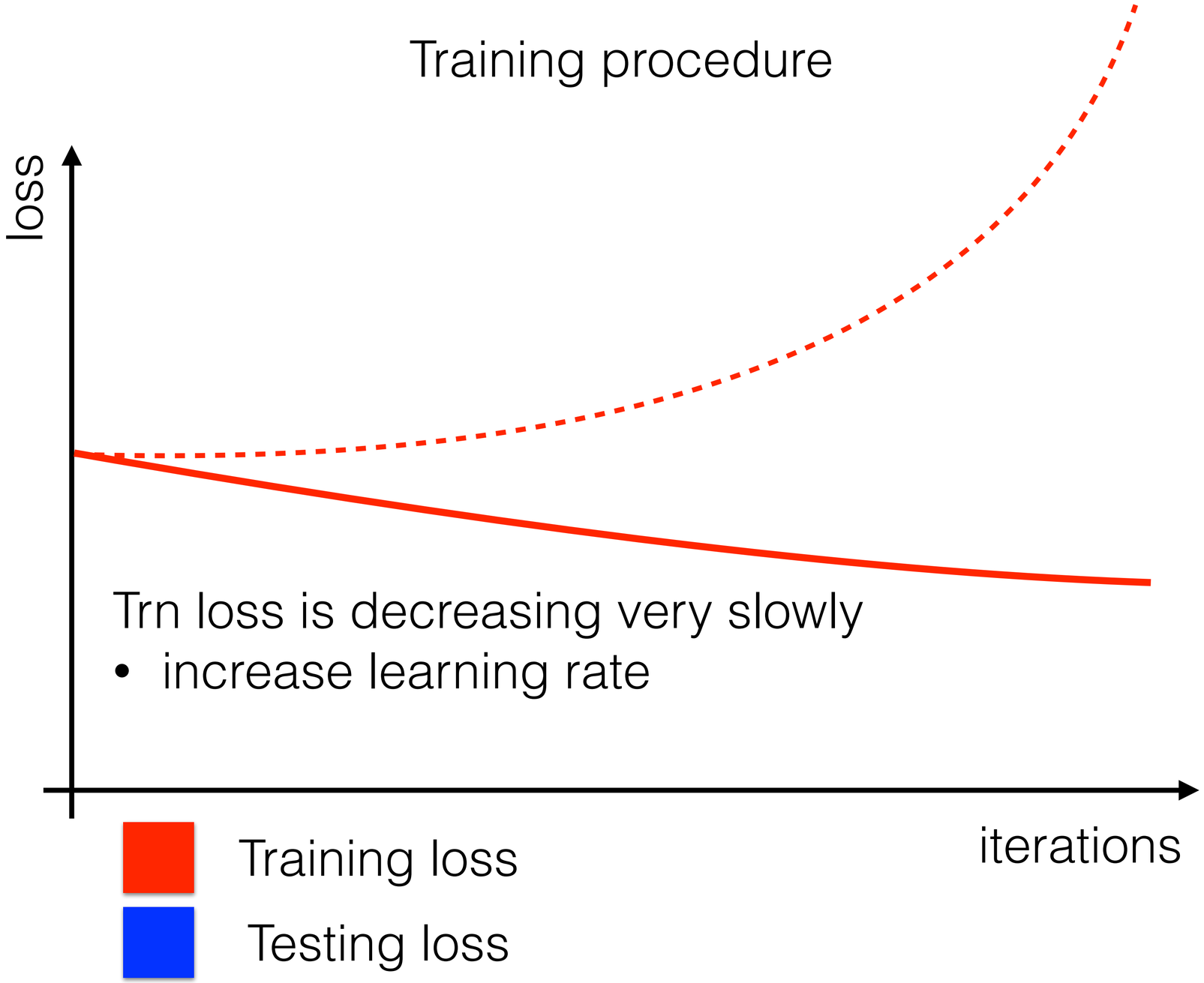
SGD drawbacks - in 2D

$\alpha = 5\text{e-}3$



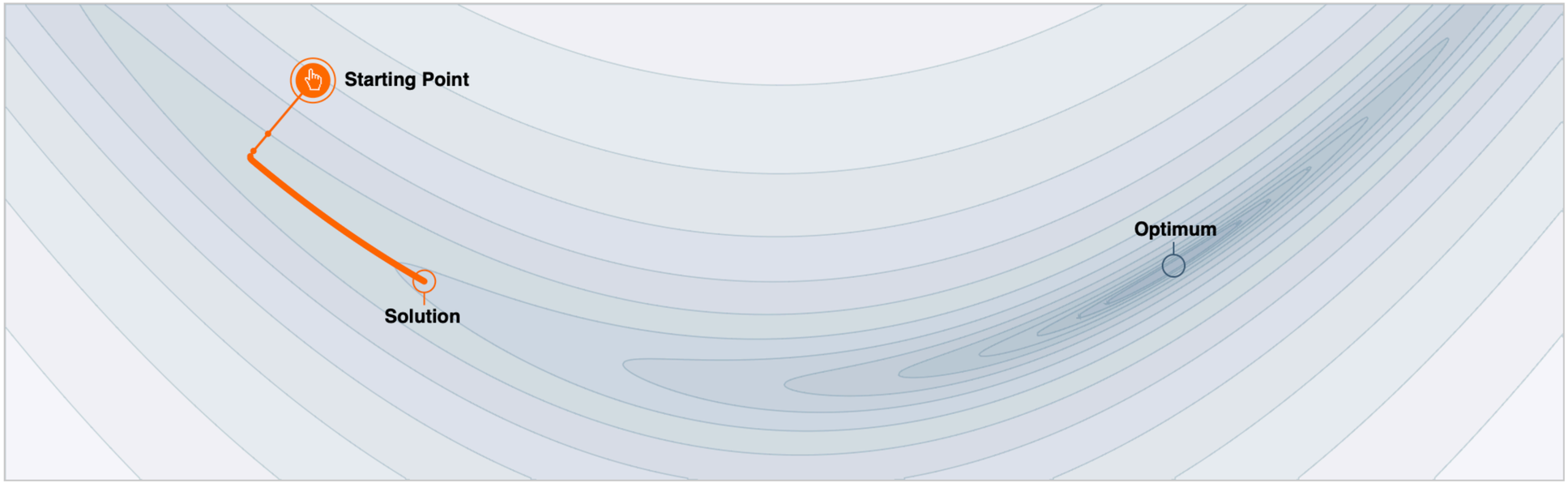
https://distill.pub/2017/momentum/

Training procedure



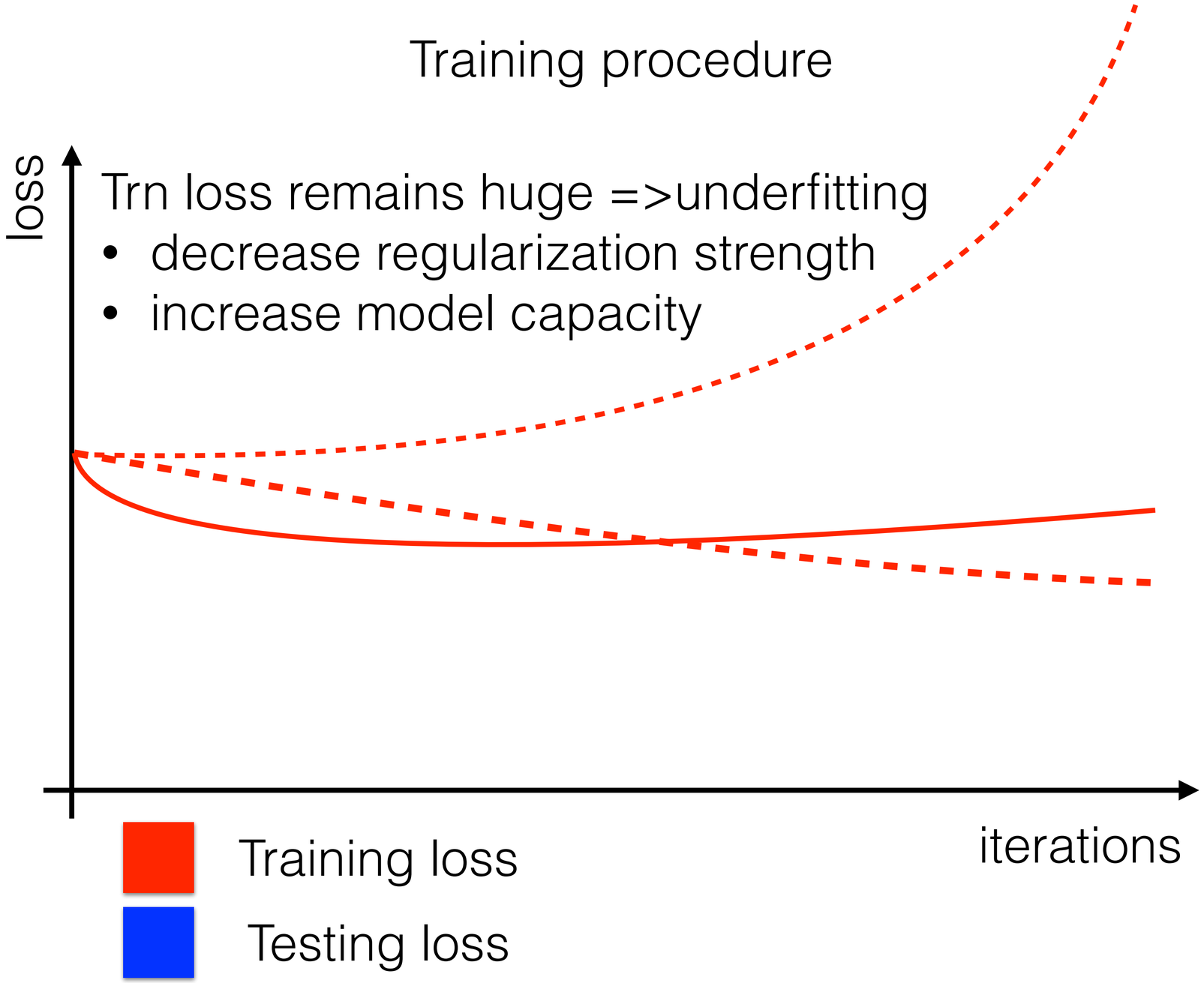
SGD drawbacks - in 2D

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

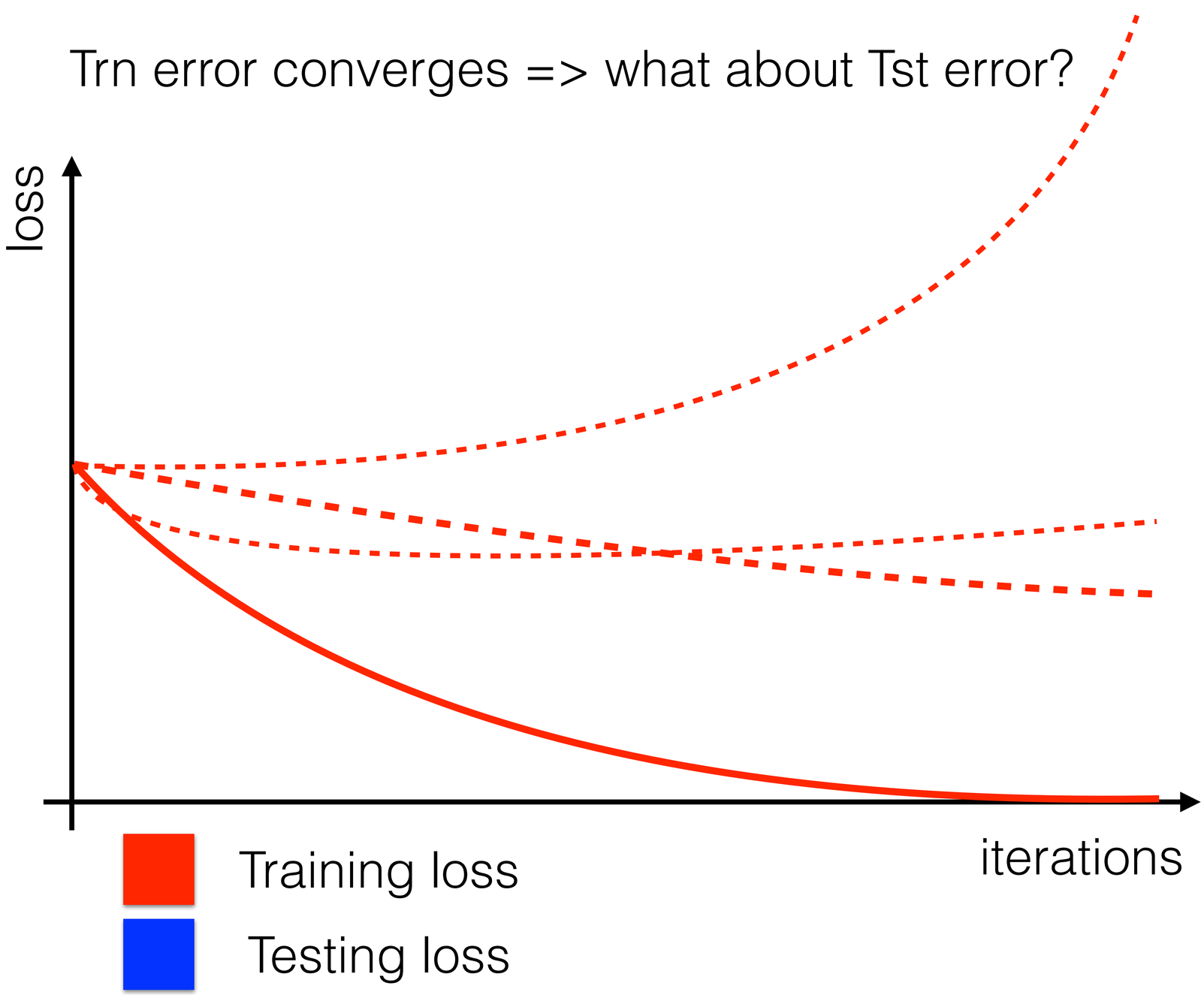


https://distill.pub/2017/momentum/

Training procedure

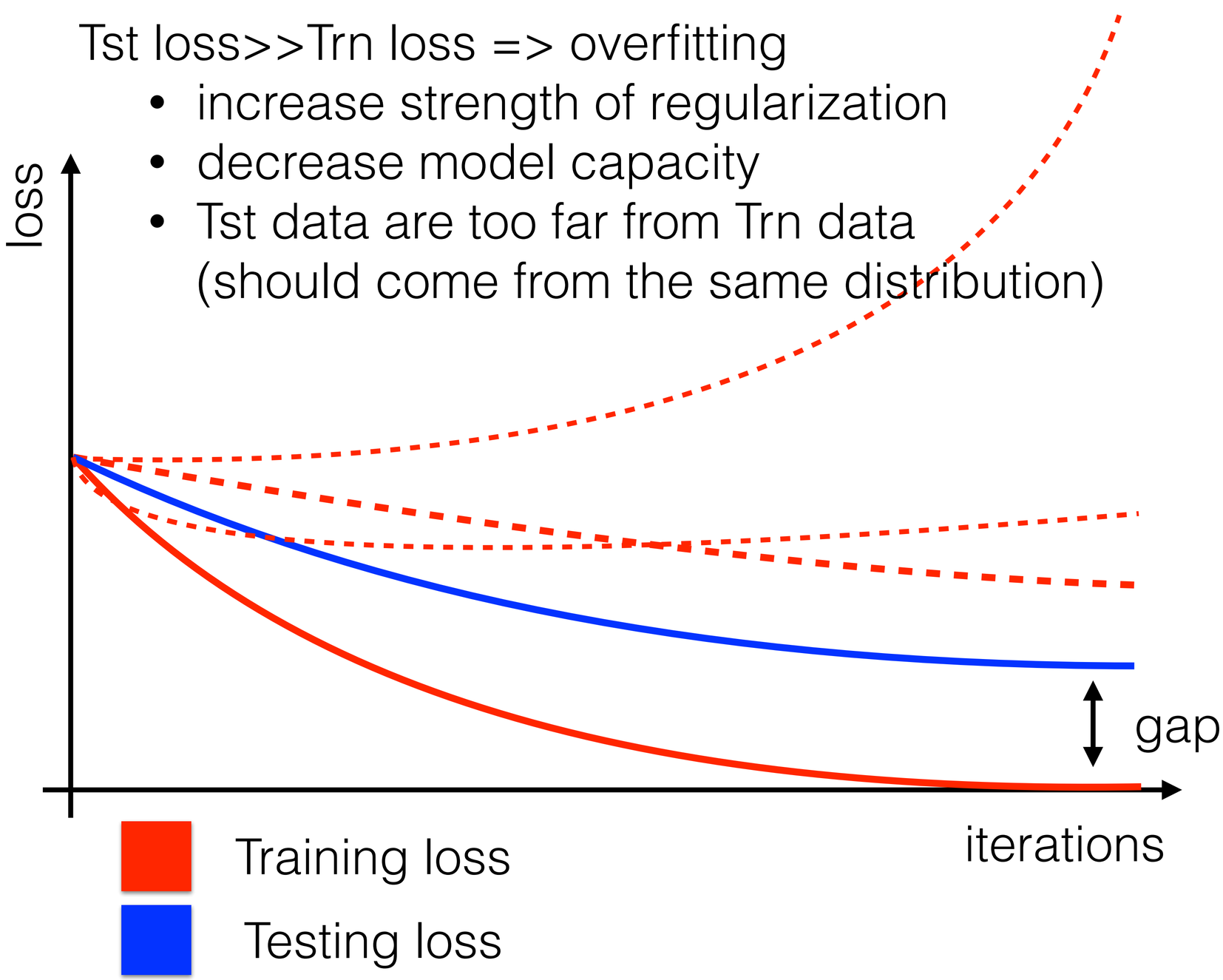


Trn error converges => what about Tst error?



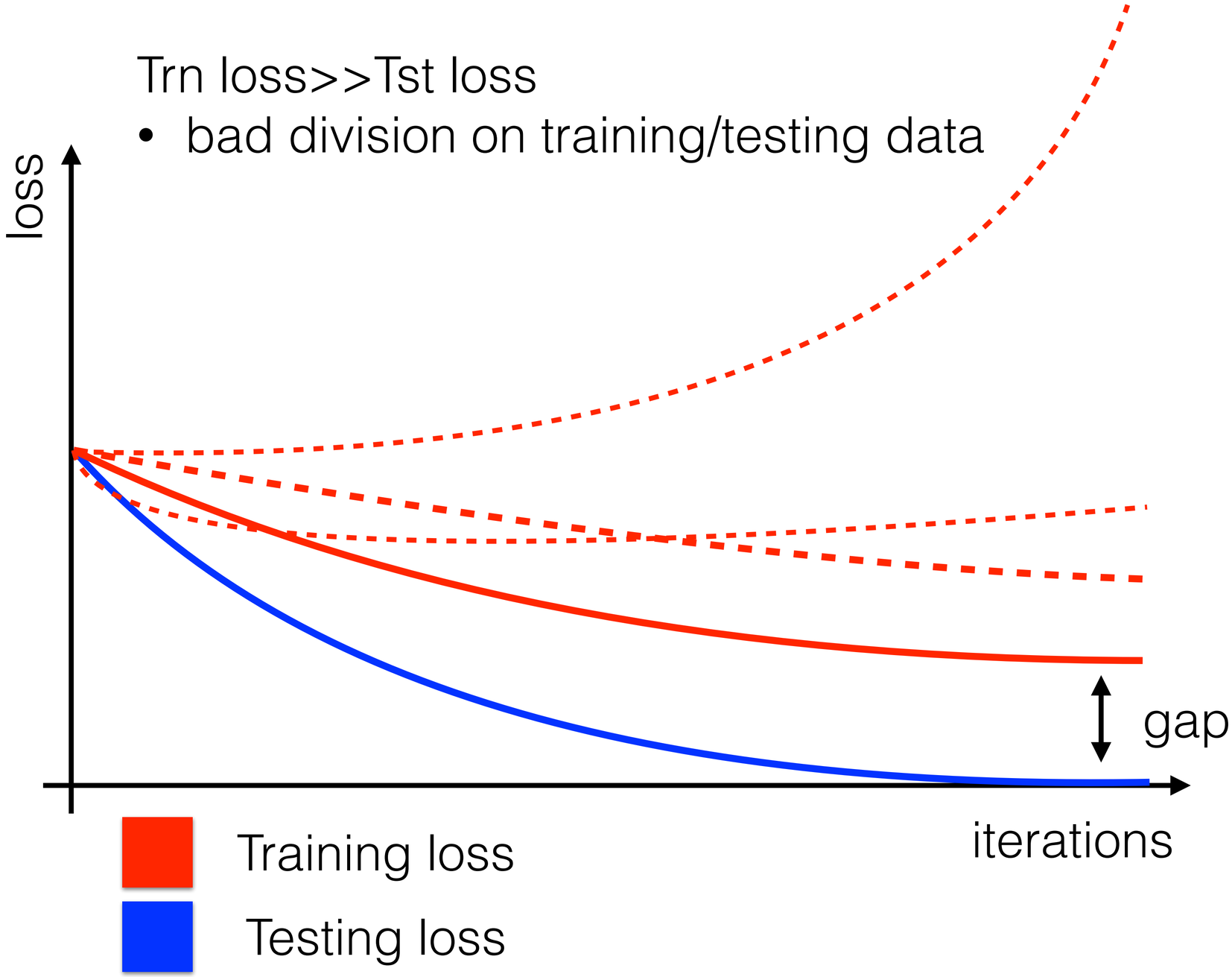
Tst loss>>Trn loss => overfitting

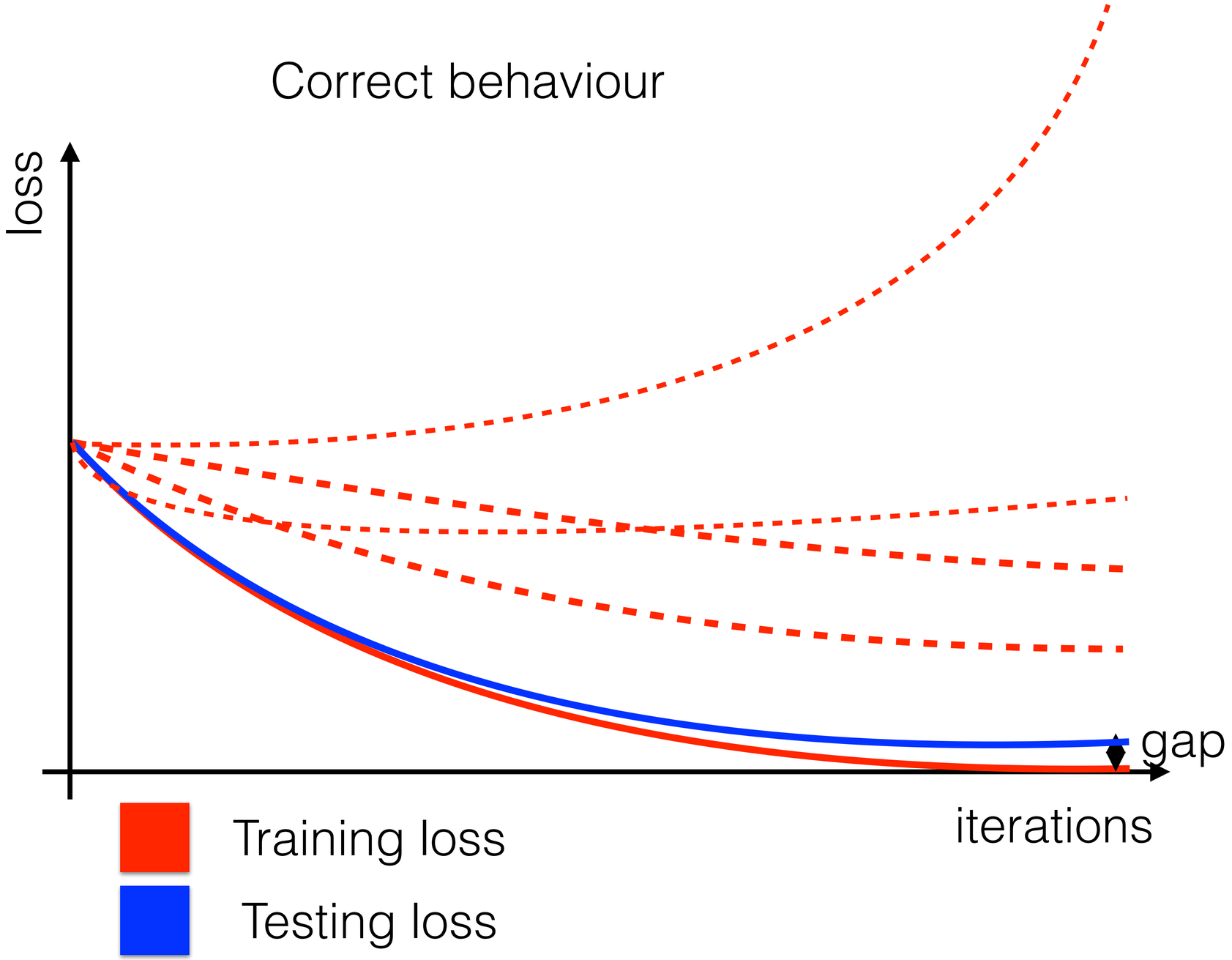
- increase strength of regularization
- decrease model capacity
- Tst data are too far from Trn data
  (should come from the same distribution)



Trn loss>>Tst loss

- bad division on training/testing data




Correct behaviour
loss
gap
iterations
Training loss
Testing loss

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

How to report classifier quality?

Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




CLS CARS



CLS BGGD:



Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




CLS CARS



CLS BGGD:



Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




CLS CARS



CLS BGGD:


false negative (FN) … classifier **falsely** indicates positive class (e.g. car) as a **negative** class => missed danger

Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




CLS CARS



CLS BGGD:



false negative (FN) … classifier **falsely** indicates positive class (e.g. car) as a **negative** class => missed danger

false positive (FP) … classifier **falsely** indicates negative class (e.g. background) as a **positive** class => false alarm

Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




CLS CARS



CLS BGGD:



false negative (FN) … classifier **falsely** indicates positive class (e.g. car) as a **negative** class => missed danger

false positive (FP) … classifier **falsely** indicates negative class (e.g. background) as a **positive** class => false alarm

true positive (TP) … classifier correctly indicate ground **truth** positive class (e.g. car) as a **positive** class => correctly found danger

Binary classifier testing presence of potentially dangerous case:

GT CARS


GT BKGD:




CLS CARS



CLS BGGD:


false negative (FN) … classifier **falsely** indicates positive class (e.g. car) as a **negative** class => missed danger

false positive (FP) … classifier **falsely** indicates negative class (e.g. background) as a **positive** class => false alarm

true positive (TP) … classifier correctly indicate ground **truth** positive class (e.g. car) as a **positive** class => correctly found danger

true negative (TN) … classifier correctly indicate ground **truth** negative class (e.g. car) as a **negative** class => correctly found safety

Binary classifier testing presence of potentially dangerous case:

GT CARS


CLS CARS



GT BKGD:




CLS BGGD:



false negative (FN) =1

false positive (FP) =2

true positive (TP) =1

true negative (TN) =2

$$\text{Precision (P)} = \frac{TP}{TP + FP} = \frac{1}{1 + 2} = 1/3$$

$$\text{Recall (R)} = \frac{TP}{TP + FN} = \frac{1}{1 + 1} = 1/2$$

Oracle: Precision = Recall = 1

Binary classifier testing presence of potentially dangerous case:

GT CARS

GT BKGD:

CLS CARS

CLS BGGD:

0.9 0.5 0.1 -0.1 -0.4 -0.6

false negative (FN) =1

false positive (FP) =2

true positive (TP) =1

true negative (TN) =2

$$\text{Precision (P)} = \frac{TP}{TP + FP} = \frac{1}{1 + 2} = 1/3$$

$$\text{Recall (R)} = \frac{TP}{TP + FN} = \frac{1}{1 + 1} = 1/2$$

Oracle: Precision = Recall = 1

Binary classifier testing presence of potentially dangerous case:

GT CARS

GT BKGD:

CLS CARS

CLS BGGD:

0.9 0.5 0.1 -0.1 -0.4 -0.6

false negative (FN) =0

false positive (FP) =2

true positive (TP) =2

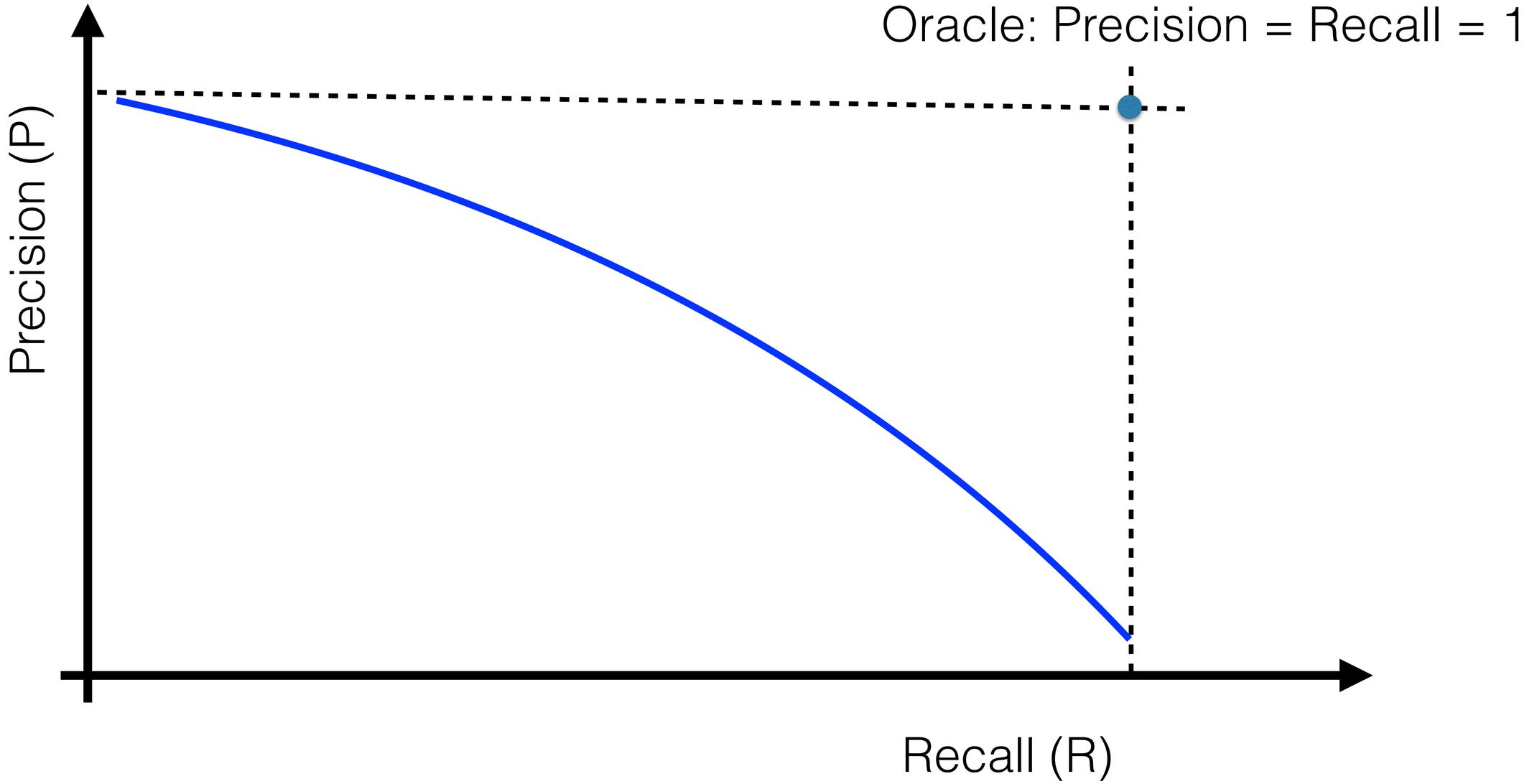
true negative (TN) =2

$$\text{Precision (P)} = \frac{TP}{TP + FP} = \frac{2}{2 + 2} = 1/2$$

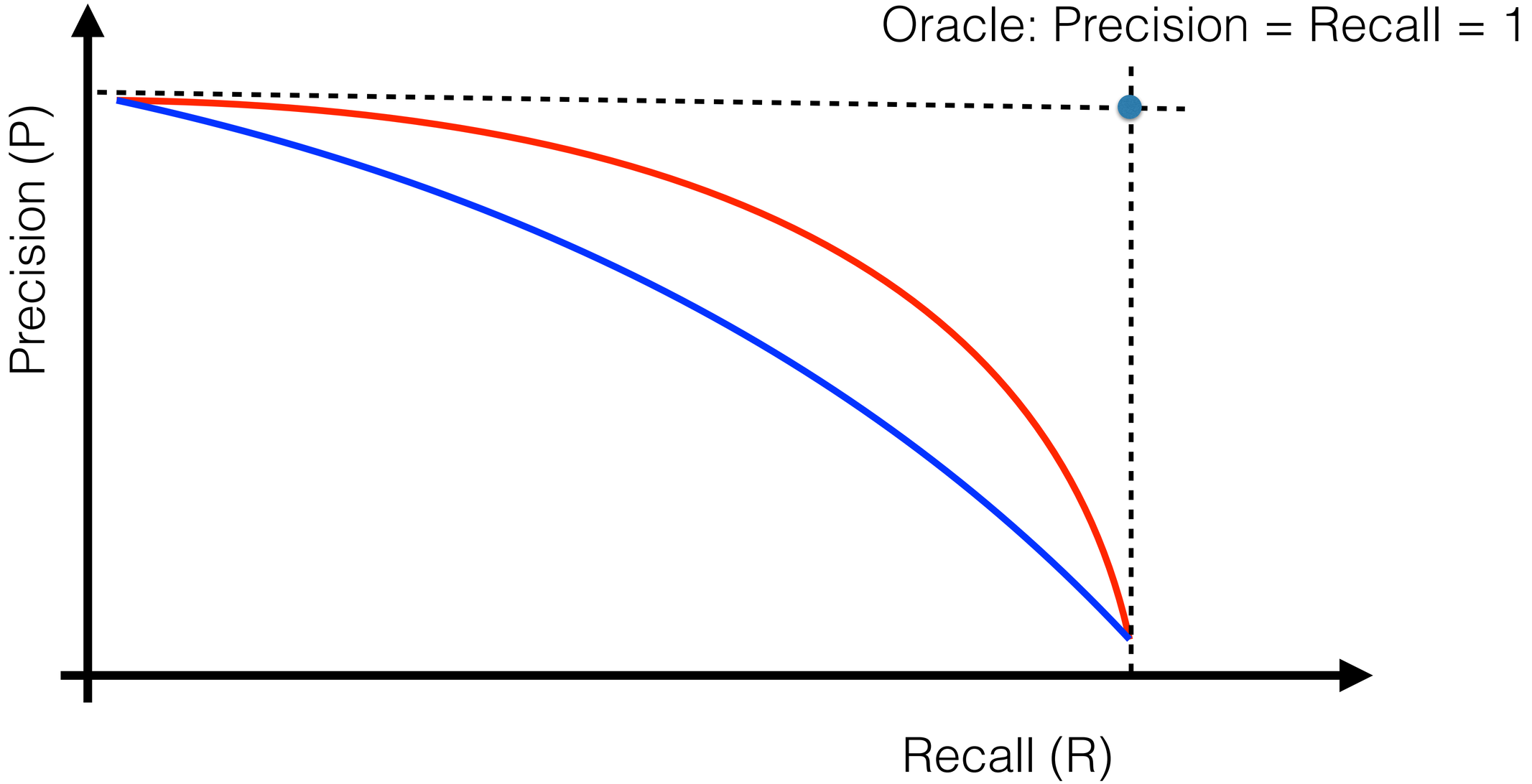
$$\text{Recall (R)} = \frac{TP}{TP + FN} = \frac{2}{2 + 0} = 1$$

Oracle: Precision = Recall = 1

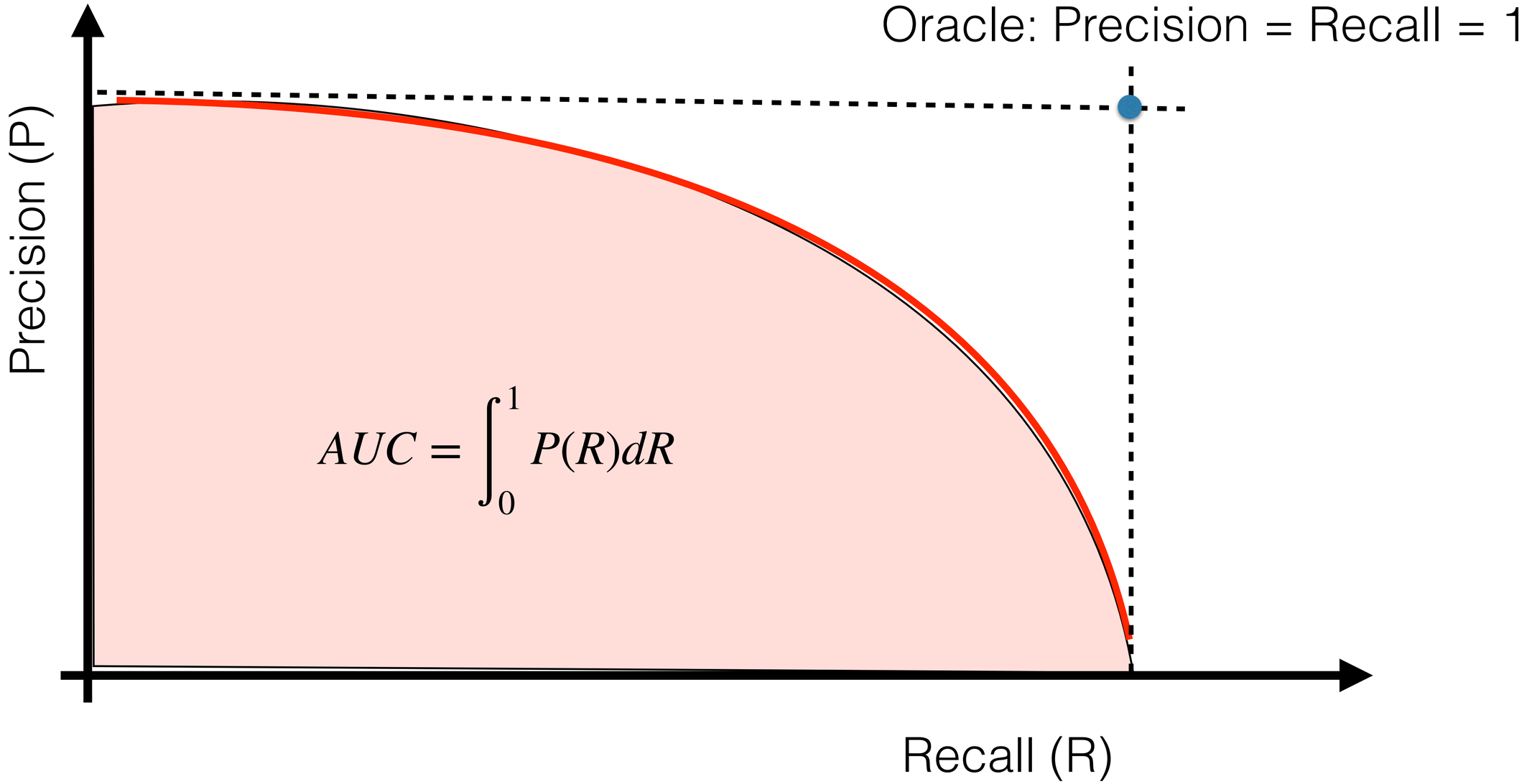
Smoothed Precision-Recall curve



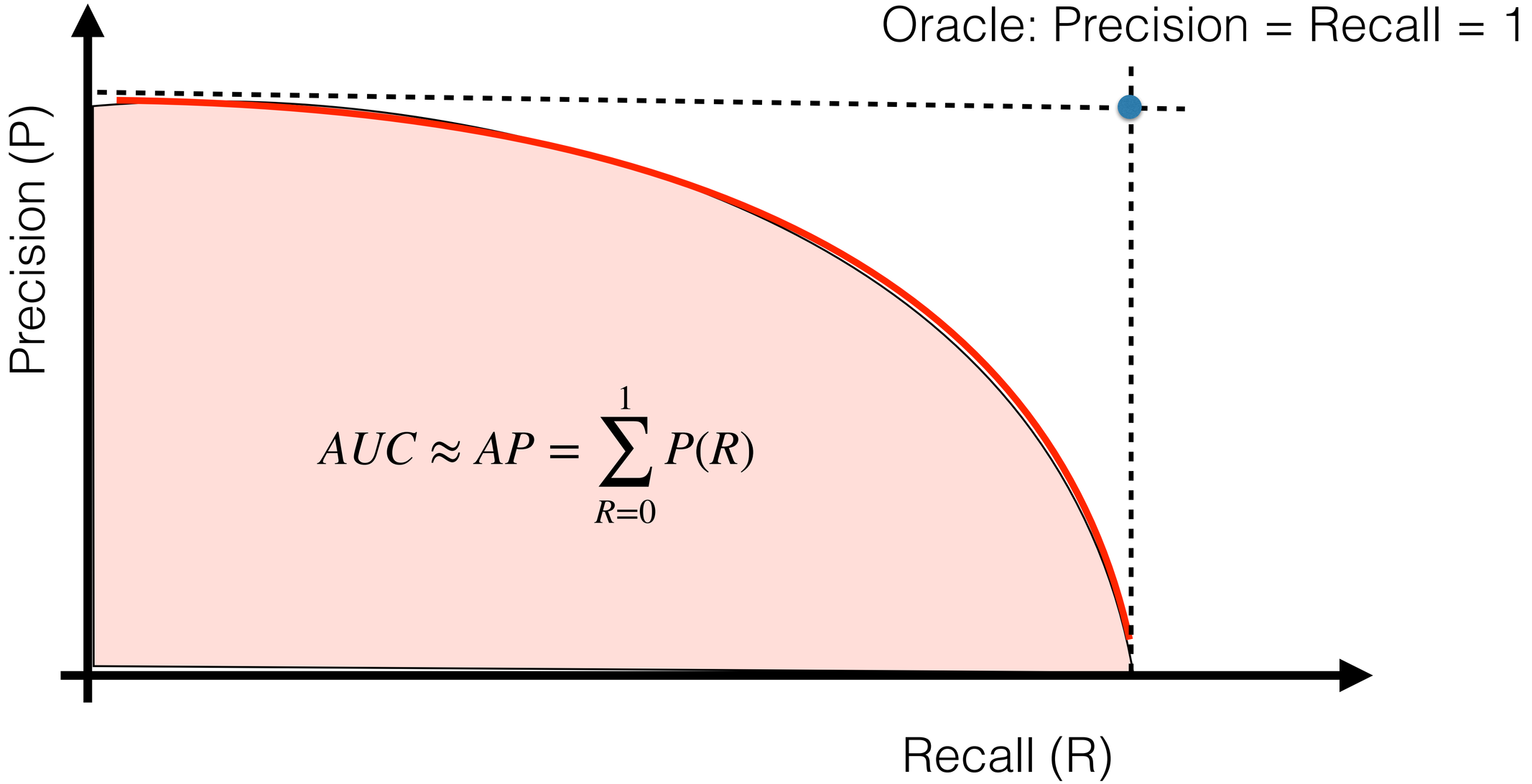
Smoothed Precision-Recall curve



Smoothed Precision-Recall curve



Smoothed Precision-Recall curve



Binary classifier testing presence of potentially dangerous case:

GT CARS

GT BKGD:

CLS CARS

CLS BGGD:

0.9 0.5 0.1 -0.1 -0.4 -0.6

false negative (FN) =0

false positive (FP) =2

true positive (TP) =1

true negative (TN) =2

$$\text{Precision (P)} = \frac{TP}{TP + FP} = \frac{1}{1 + 2} = 1/3$$

$$\text{Recall (R)} = \frac{TP}{TP + FN} = \frac{1}{1 + 2} = 1/2$$

$$\text{IoU} = \frac{TP}{TP + FP + FN} = \frac{1}{1 + 2 + 1} = 1/4$$