Learning for vision IV training & layers

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

Prequel

"The learning goes on only if the gradient is non-zero!!!"

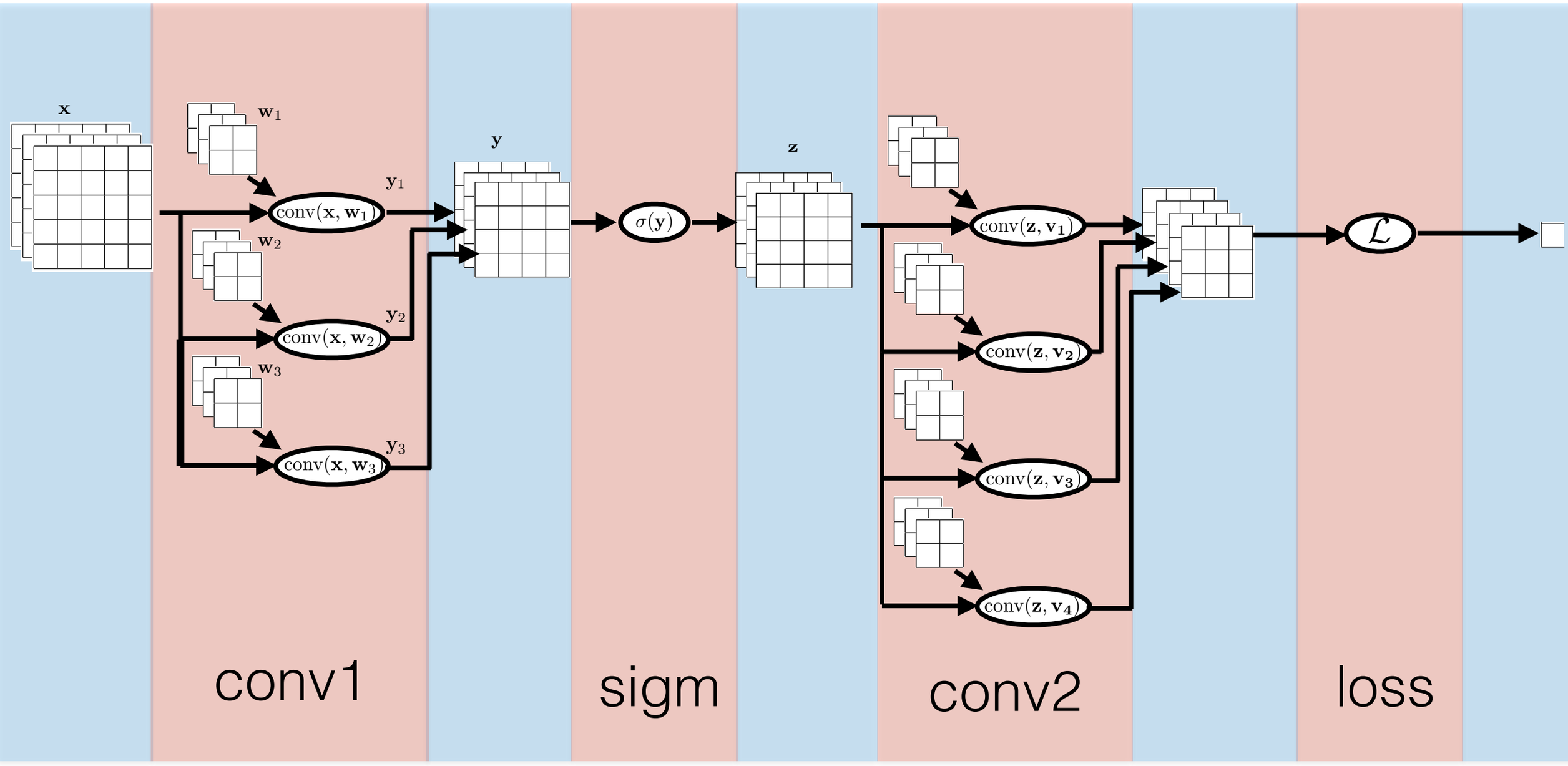
- there are many saddle points and a few local minima
- achieving zero gradient usually means that learning got stuck (and not that you achieved the global optimum)
- we can avoid "premature zero gradient" by:
  - advanced gradient optimization method
  - suitable initialization
  - optimization-friendly network structure

Outline

- Stochastic Gradient Descent (SGD)
- what happens to gradients during learning
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,
  - regularizations

Learning as gradient minimization

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



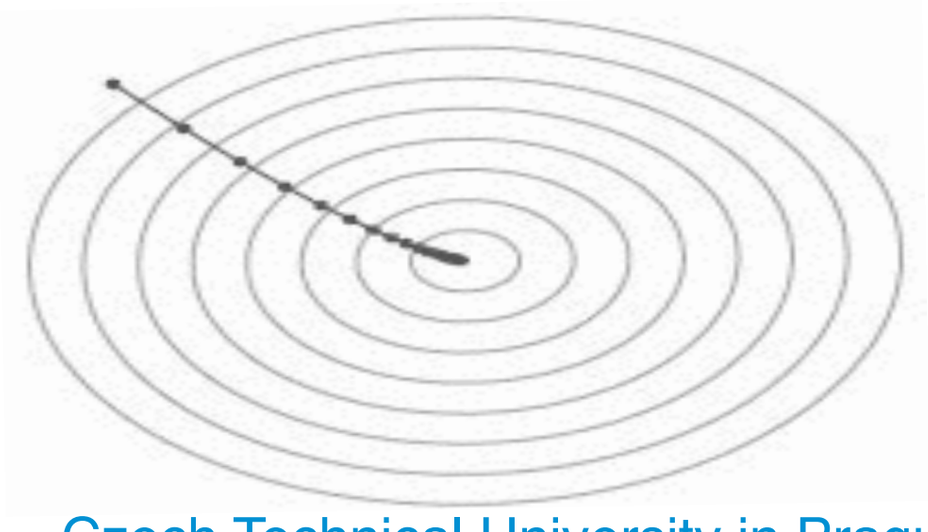
Learning as gradient minimization

1. Initialize weights $\mathbf{w}_0$ and $t = 0$
2. Plug $\mathbf{x}_i$ to input and estimate $\left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$ by backprop
3. Estimate gradient over whole training set

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

4. Update weights

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



Learning as gradient minimization

1. Initialize weights $\mathbf{w}_0$ and $t = 0$
2. Plug $\mathbf{x}_i$ to input and estimate $\left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$ by backprop
3. Estimate gradient over whole training set

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

4. Update weights

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

- Whole training set does not fit into memory => instead estimate stochastic gradient over minibatch

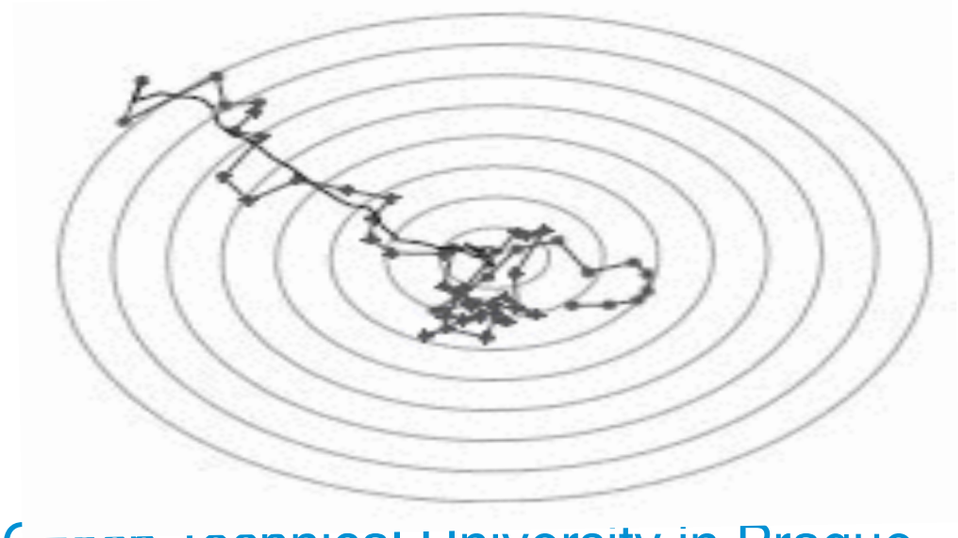
Learning as gradient minimization

1. Initialize weights $\mathbf{w}_0$ and $t = 0$
2. Plug $\mathbf{x}_i$ to input and estimate $\left.\frac{\partial f(\mathbf{x}_i; \mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$ by backprop
3. Estimate gradient over random mini-batch

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

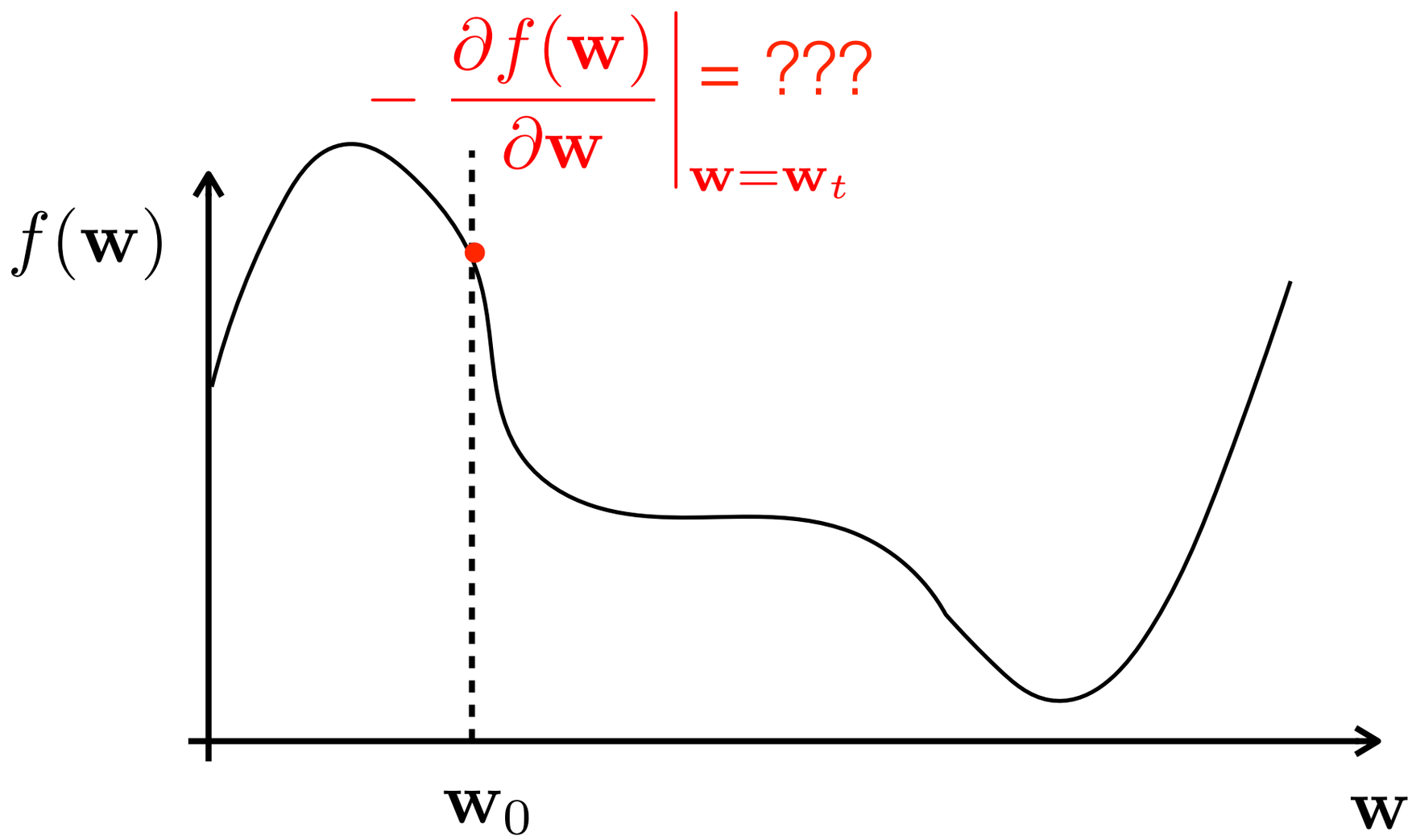
4. Update weights

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



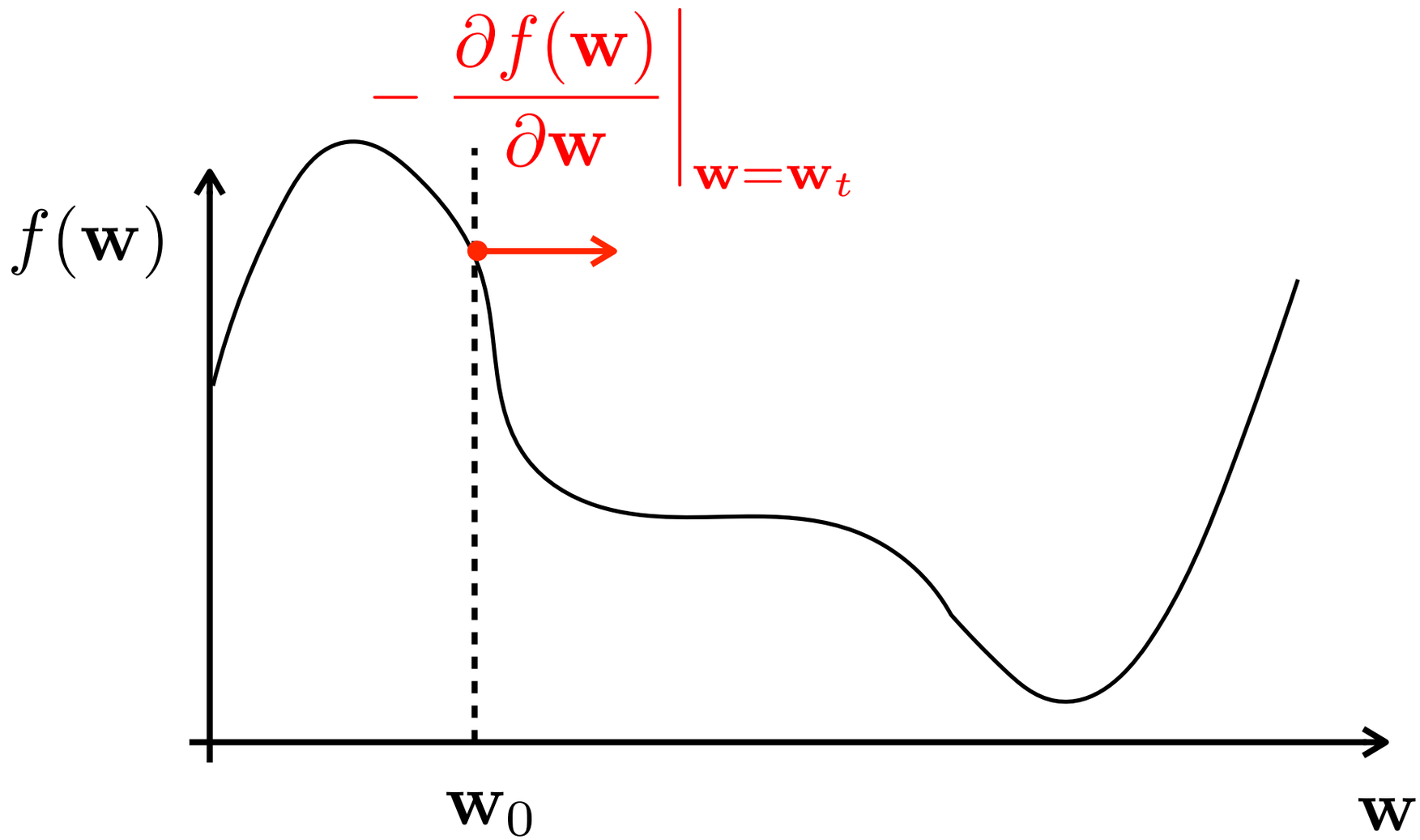
Stochastic Gradient Descent (SGD) drawbacks

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



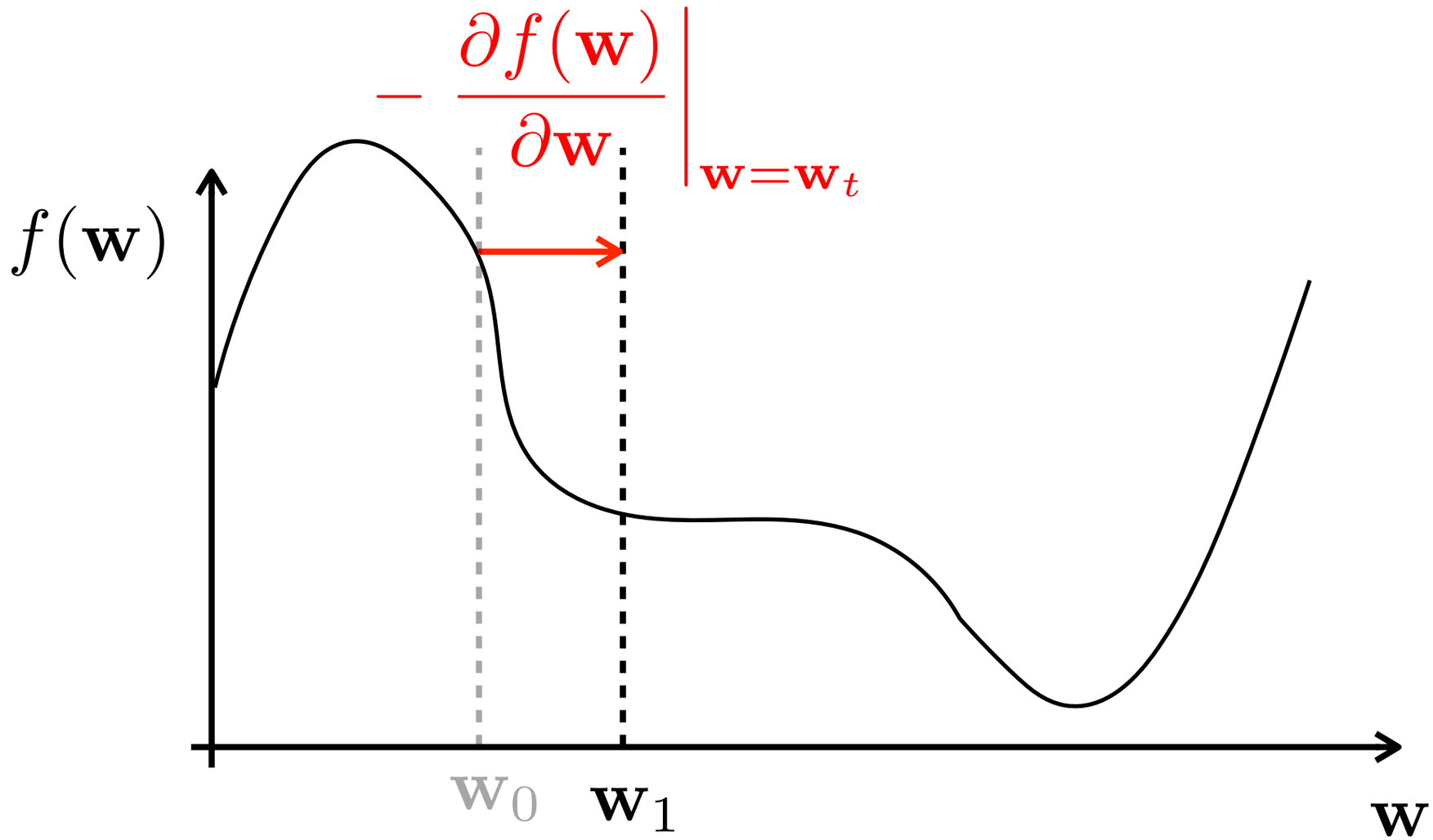
Stochastic Gradient Descent (SGD) drawbacks

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



SGD drawbacks

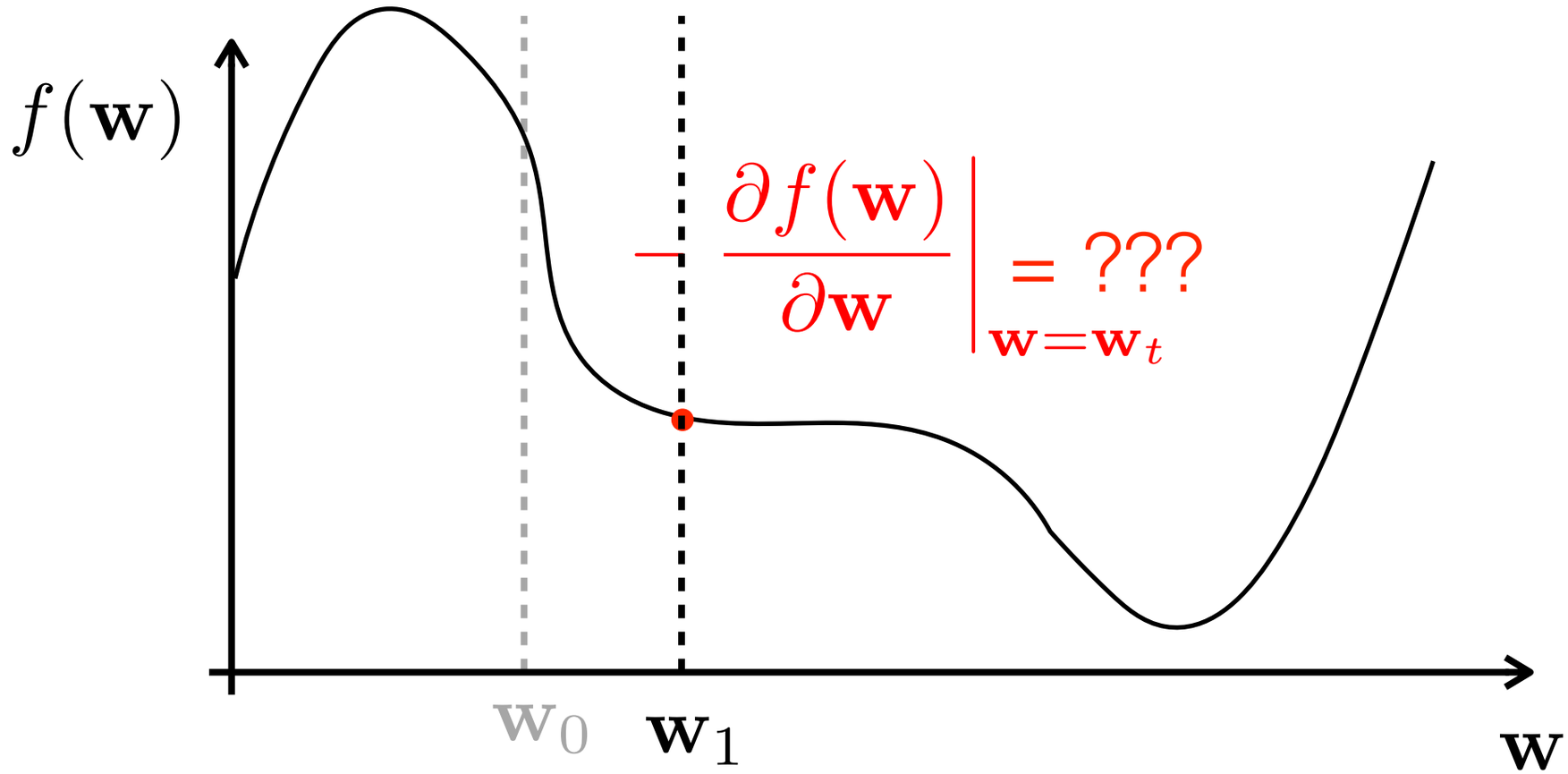
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$$



SGD drawbacks

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

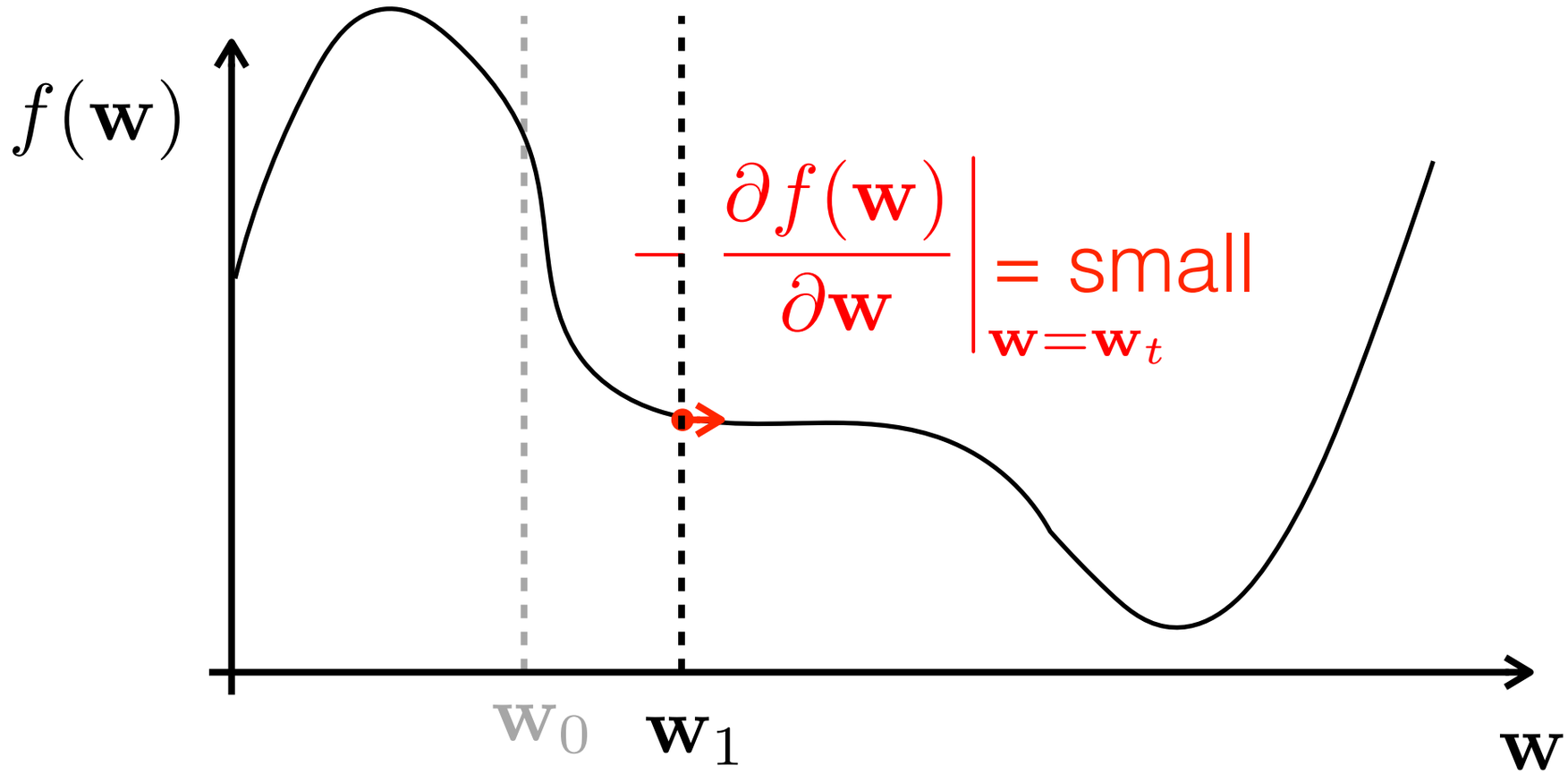
- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima



SGD drawbacks

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

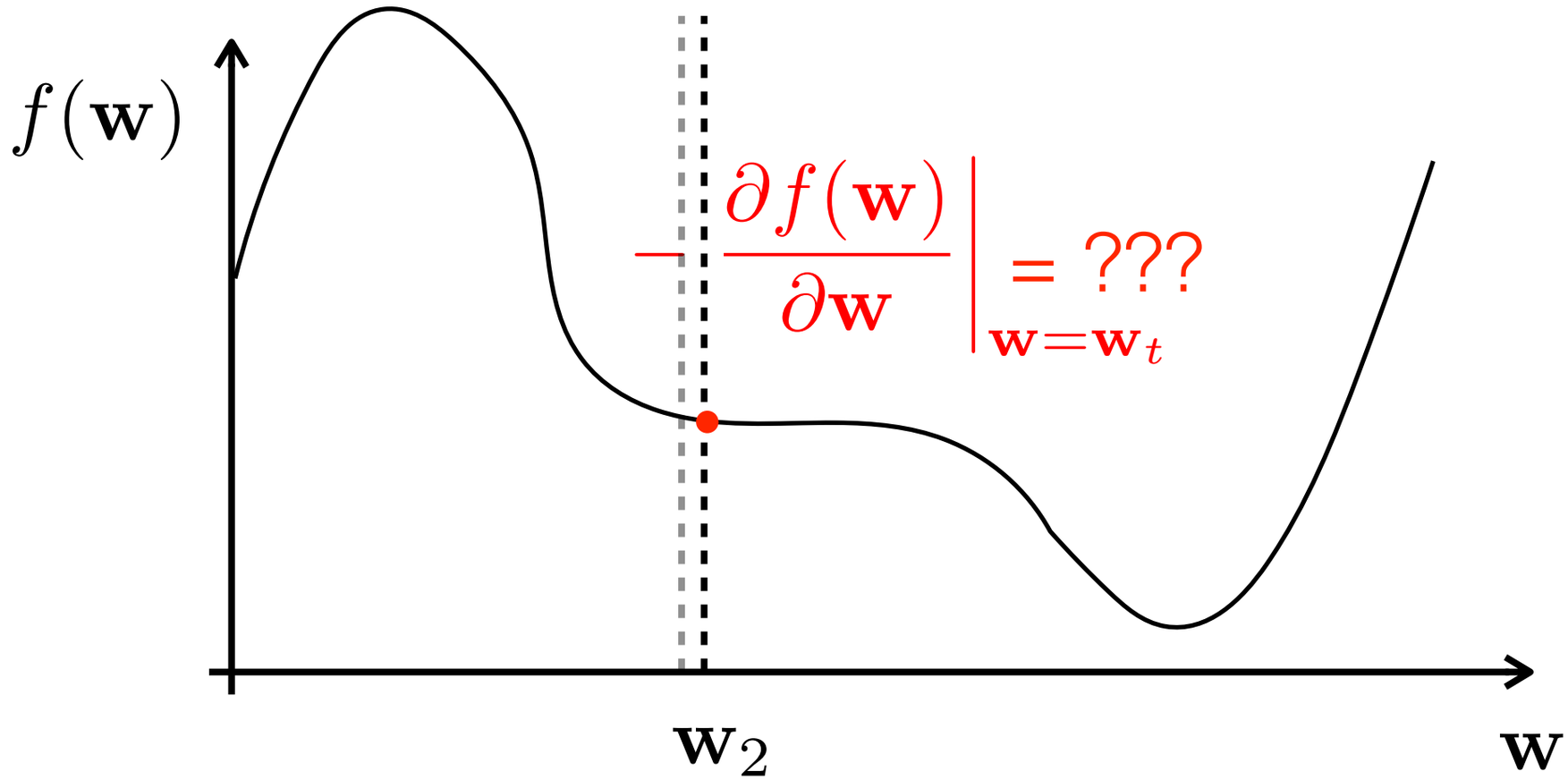
- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima



SGD drawbacks

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

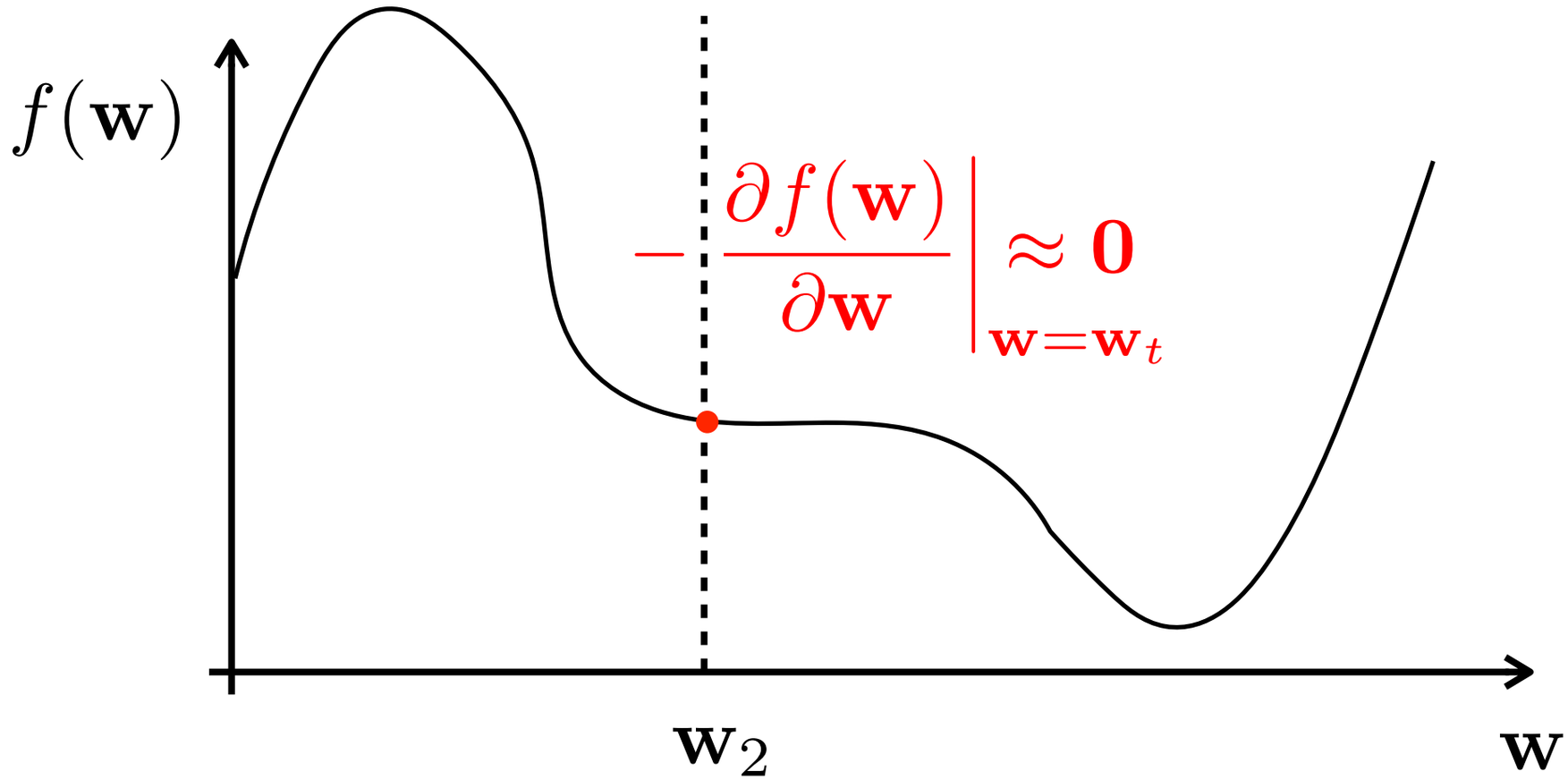
- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima



SGD drawbacks

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

- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima

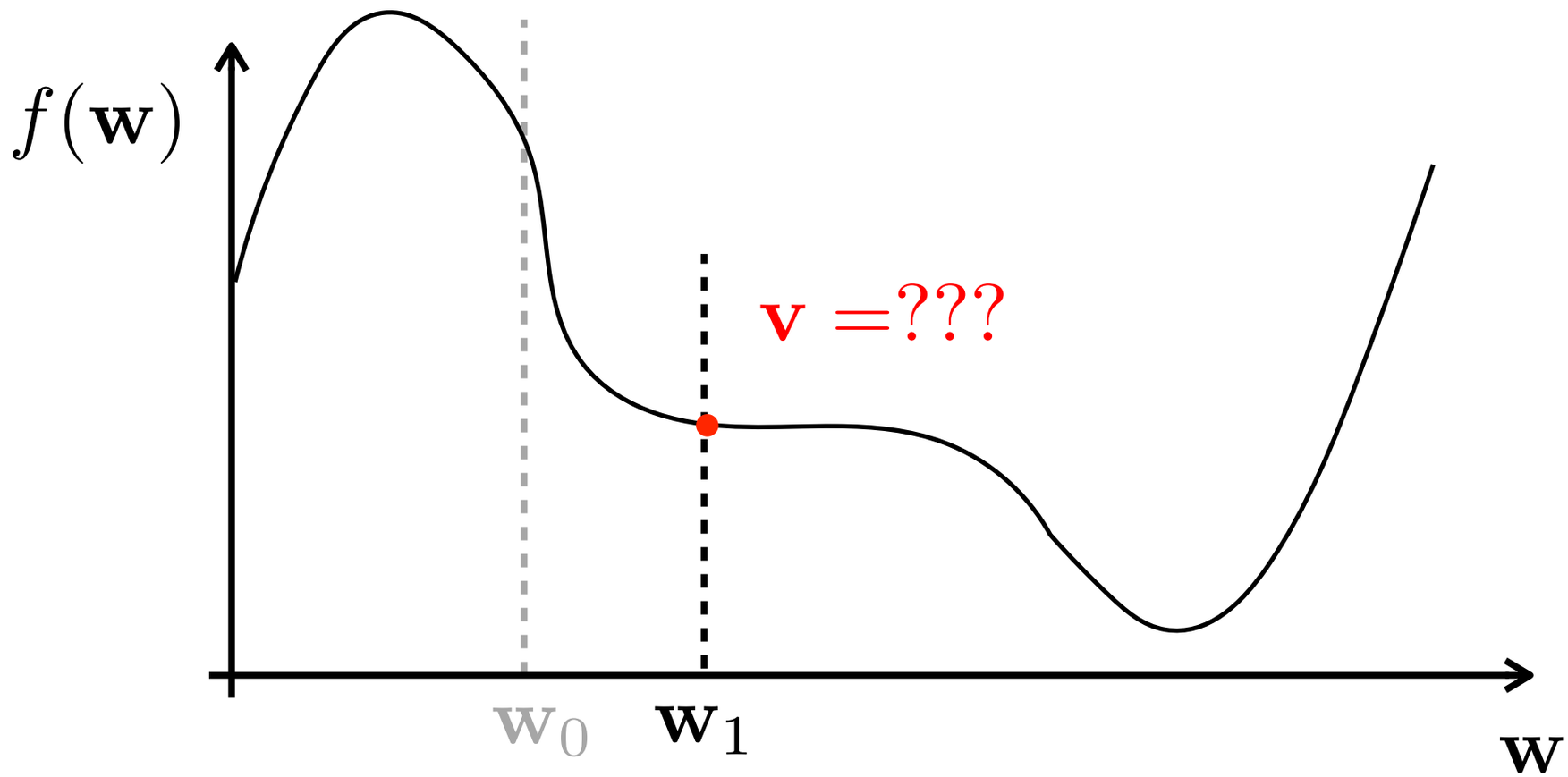


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

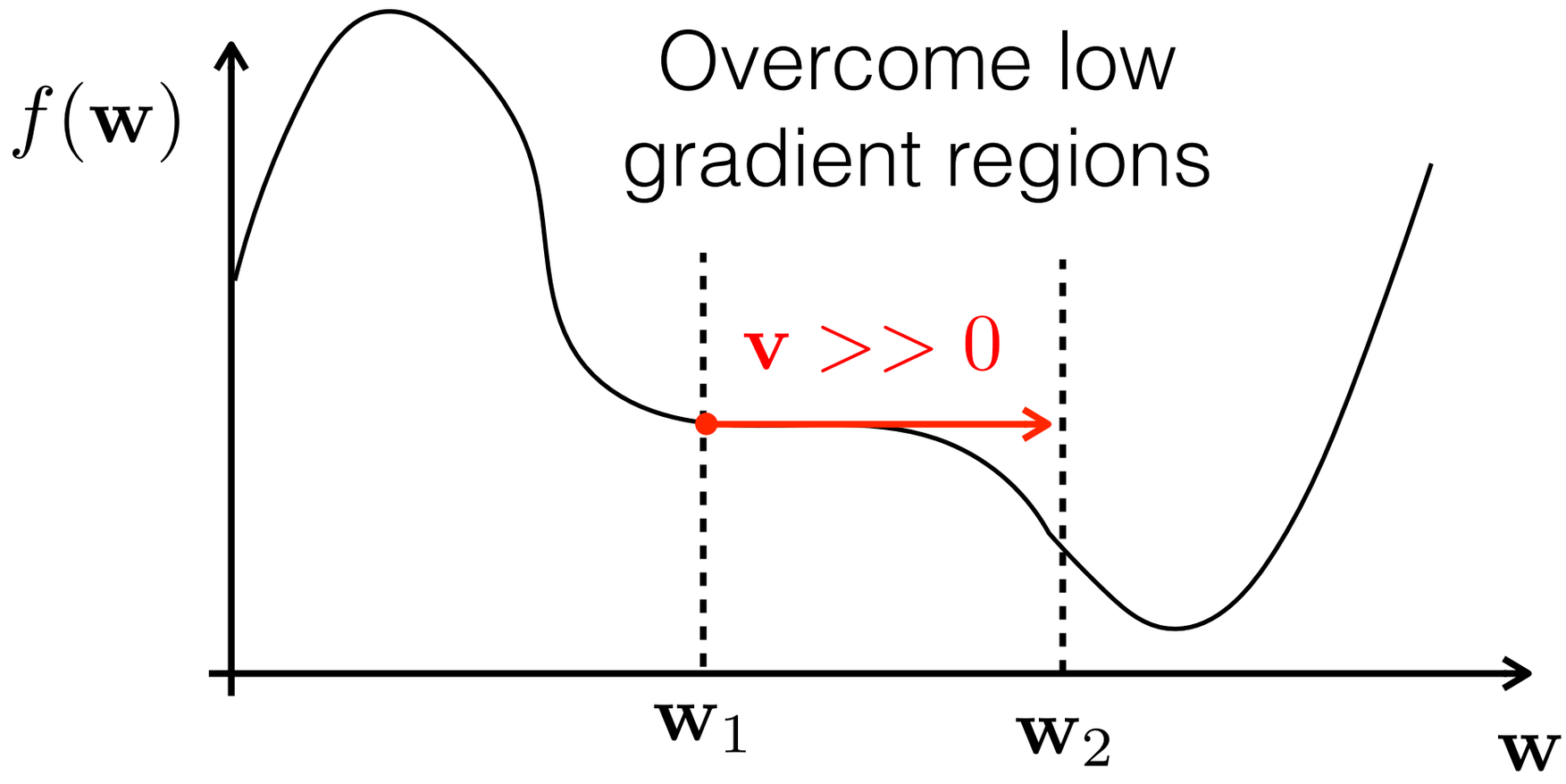


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

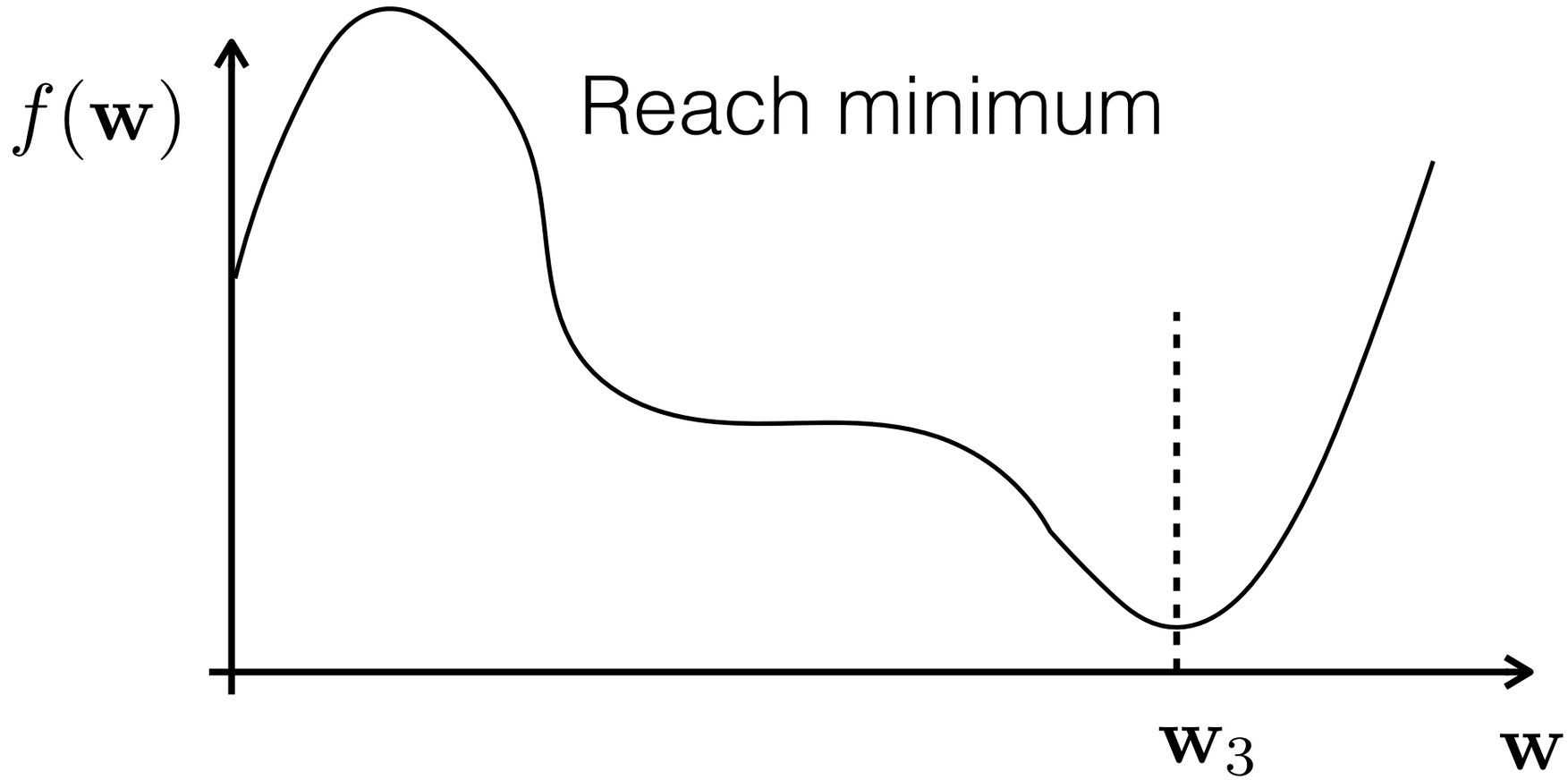


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

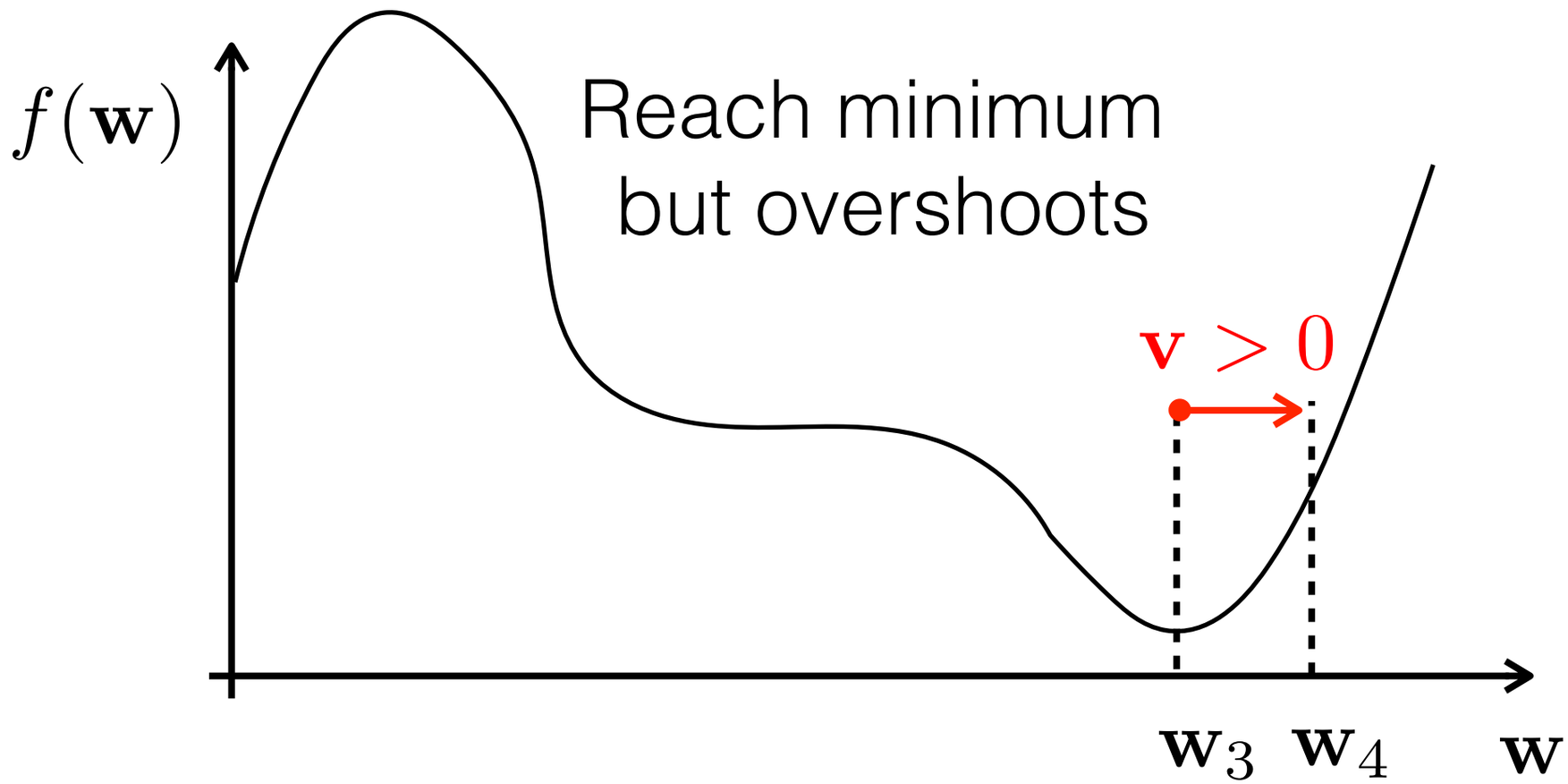


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

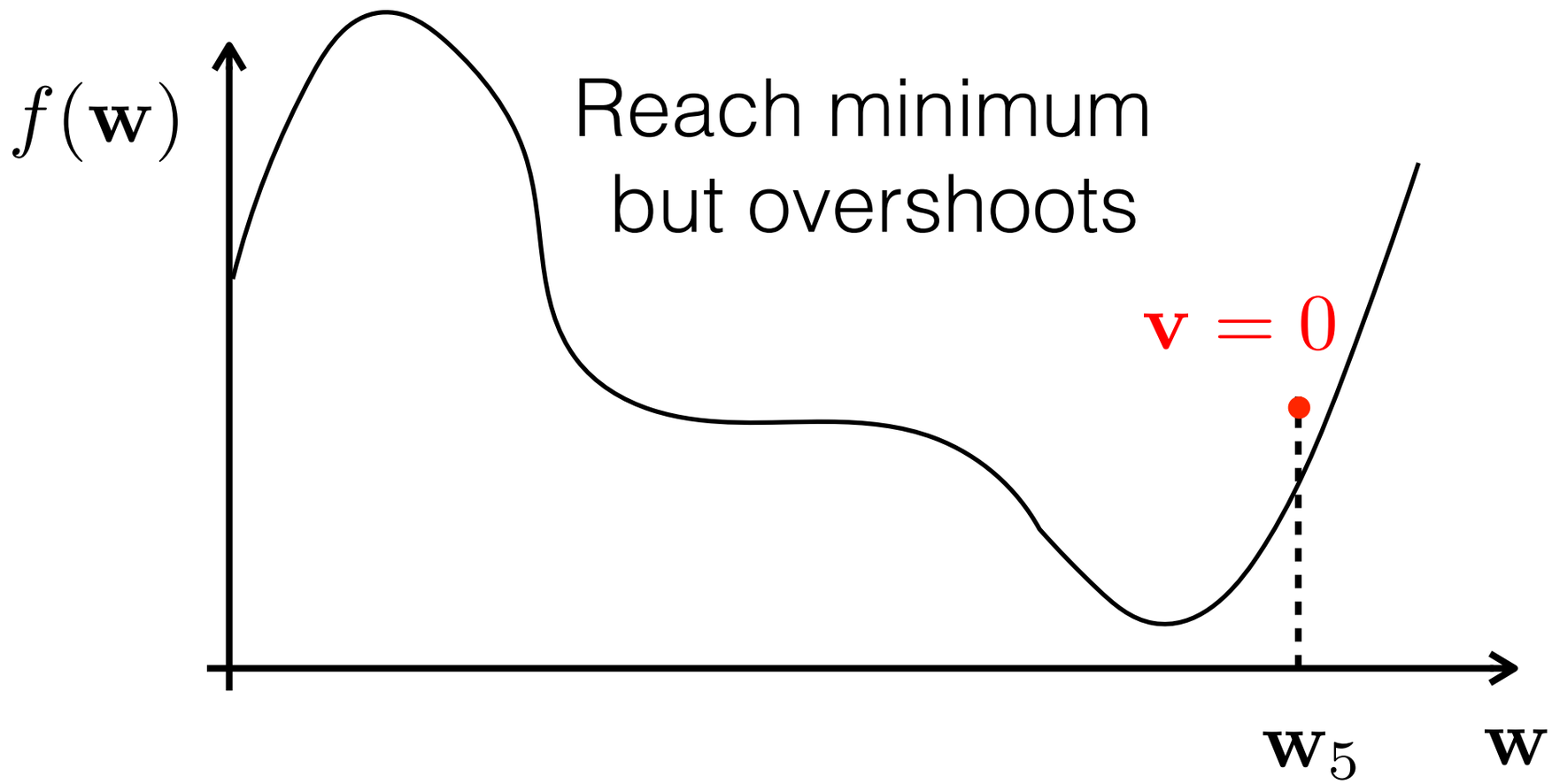


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

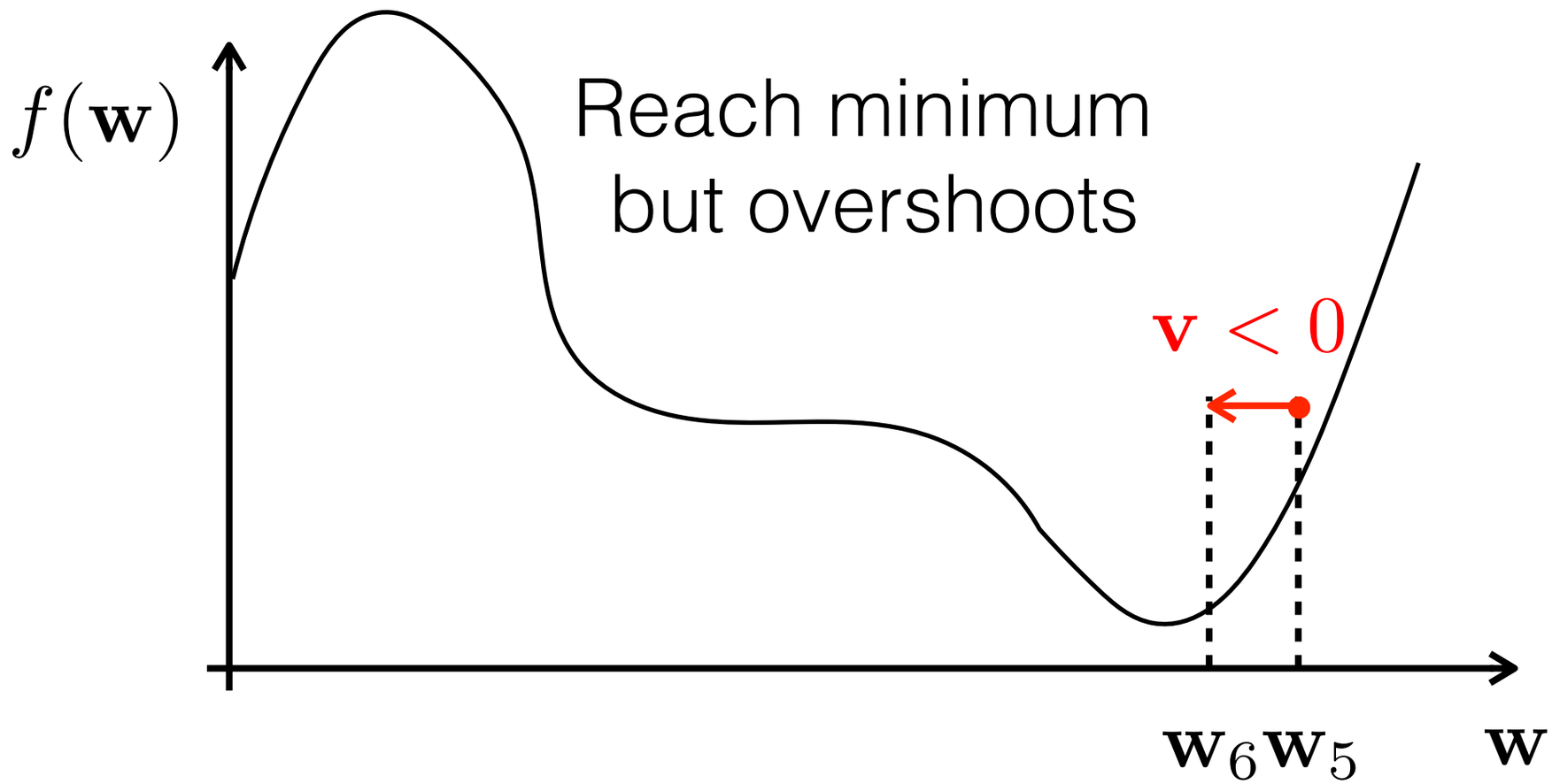


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

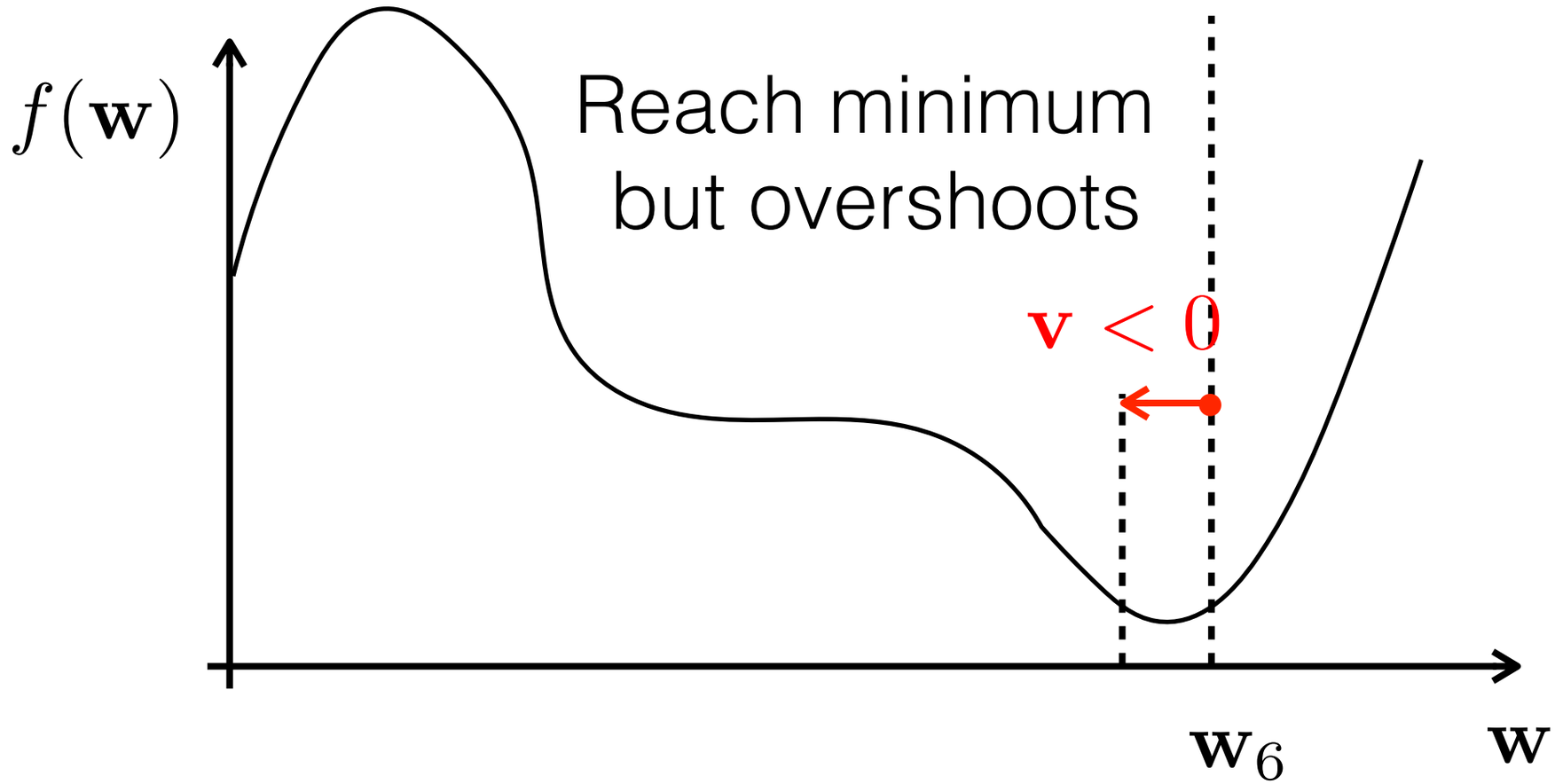


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

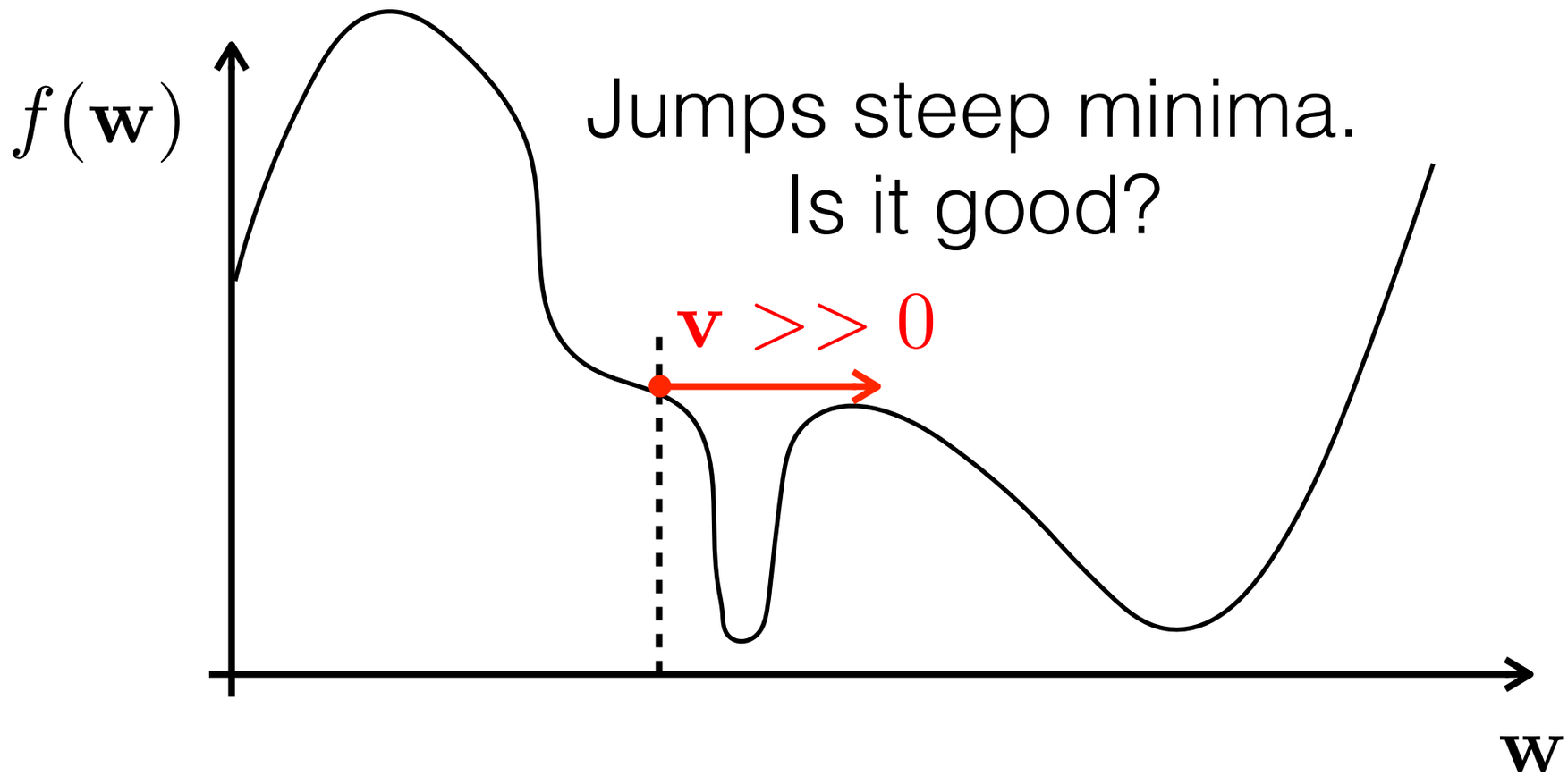


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

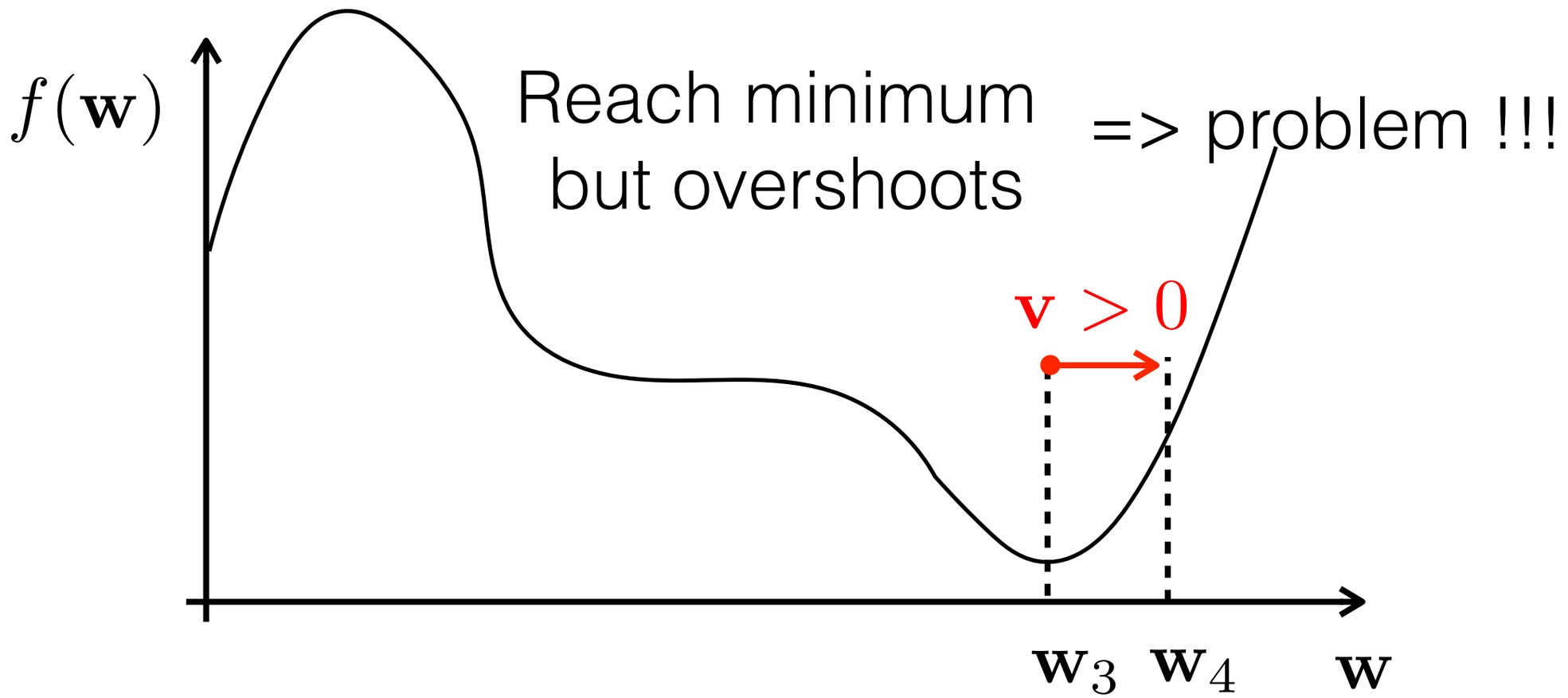


SGD + momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

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

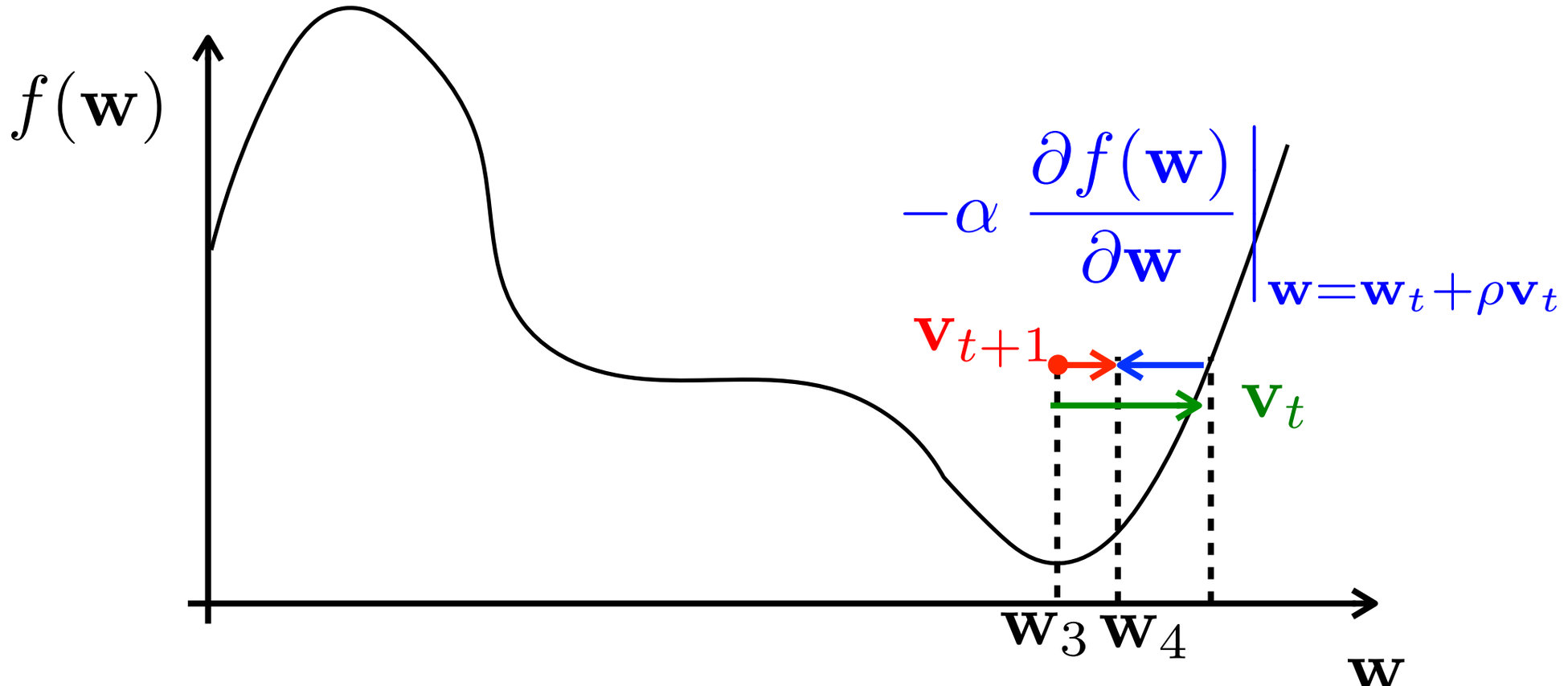


SGD with Nesterov momentum

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{v}_t$$

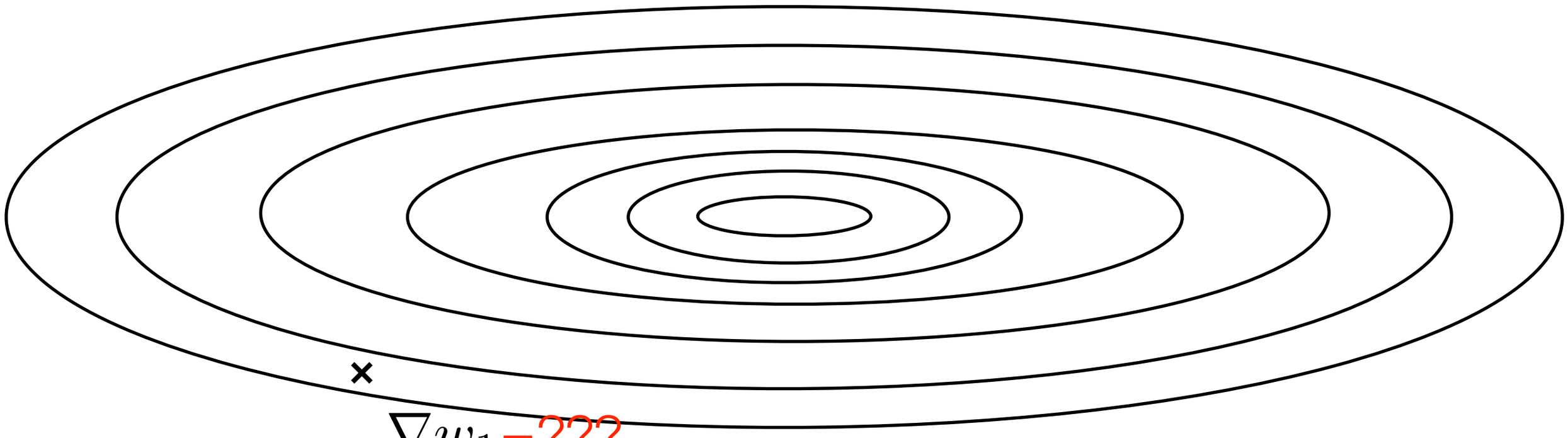
- Look one step ahead and reduce velocity by future gradient
- Partially prevents overshooting



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

SGD in 2 dimensional weights

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



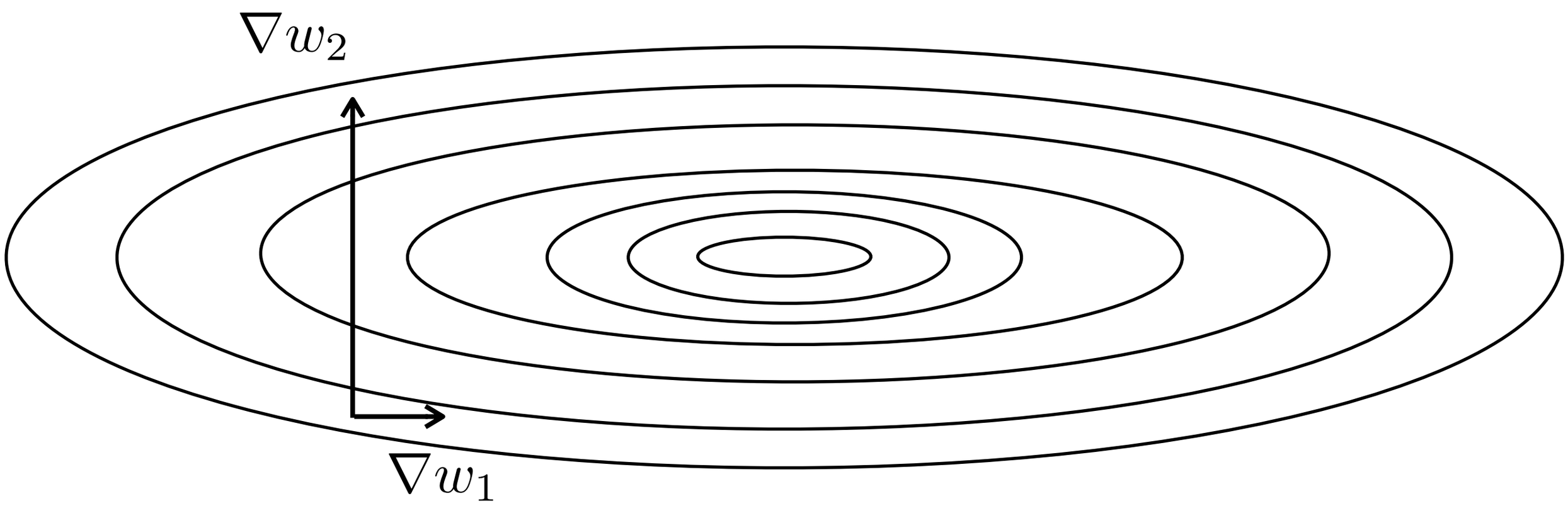
∇w_1=???

∇w_2=???

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

SGD in 2 dimensional weights

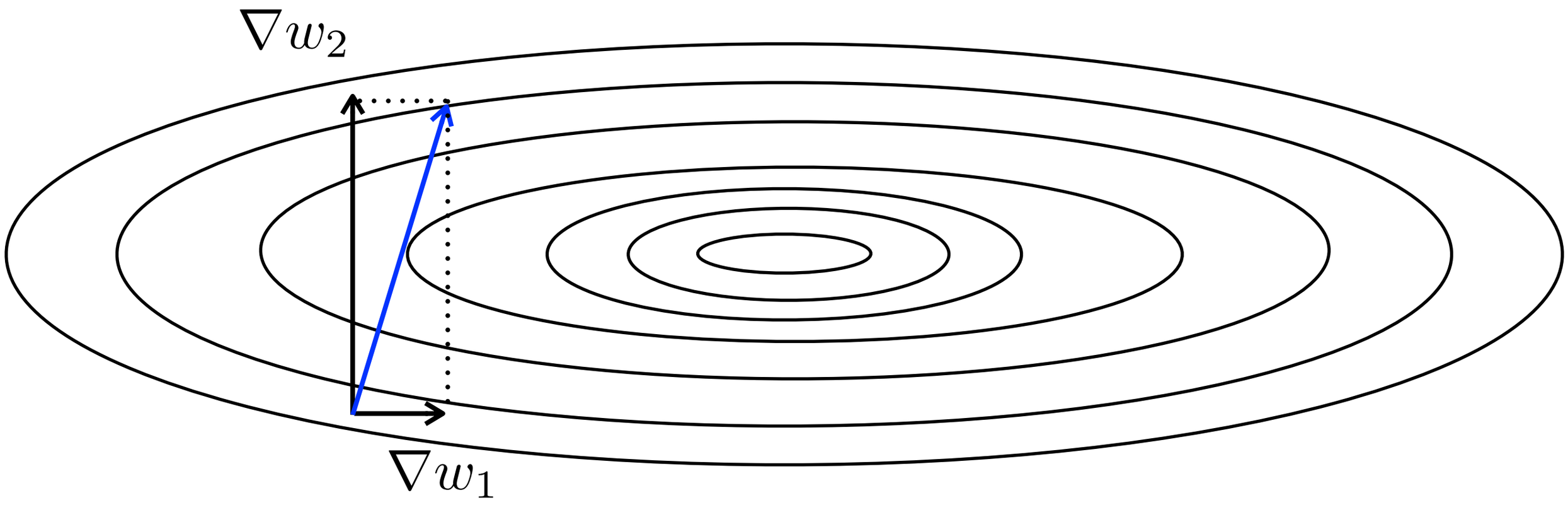
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$$



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

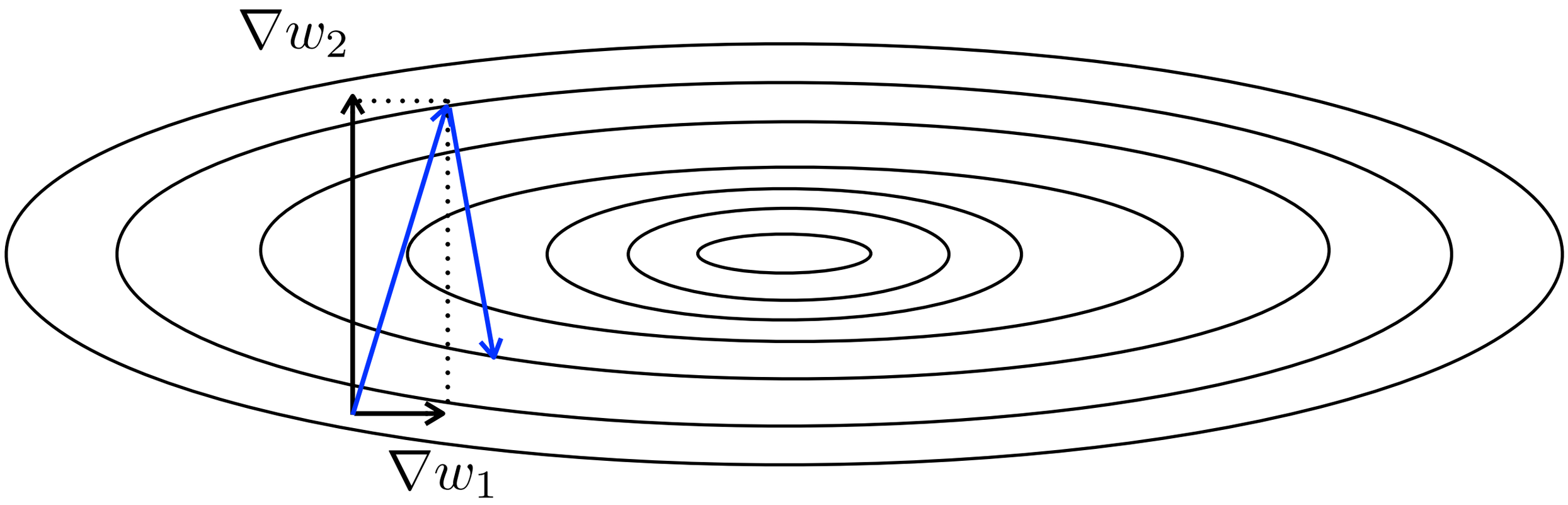
SGD in 2 dimensional weights

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



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

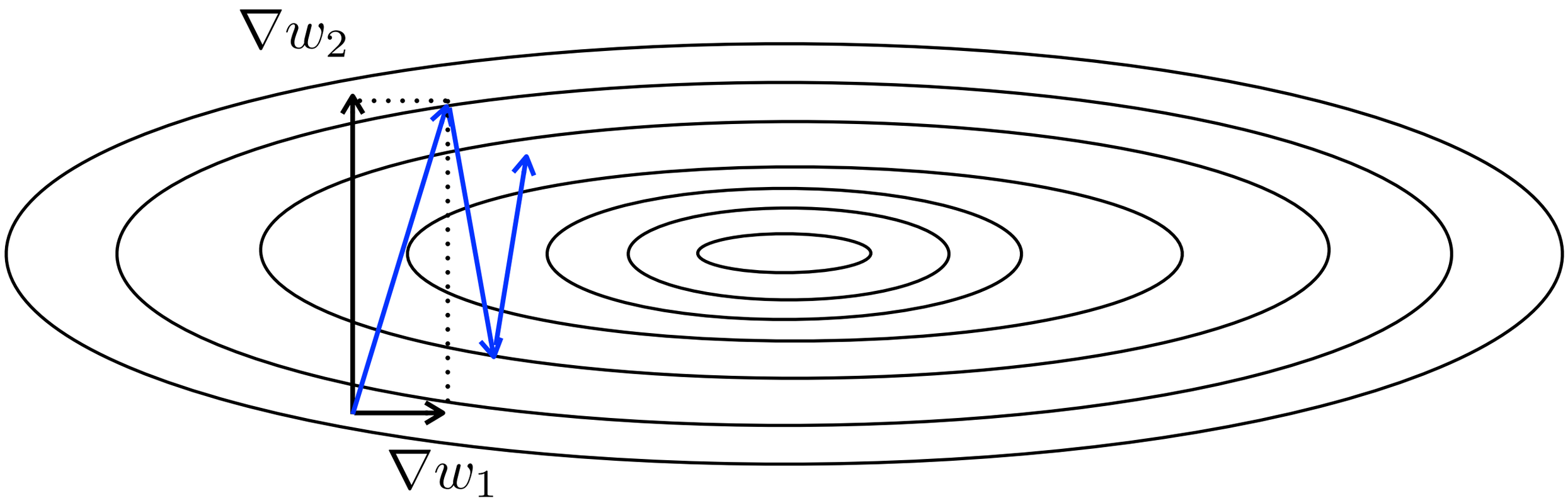
SGD in 2 dimensional weights



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

SGD in 2 dimensional weights

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

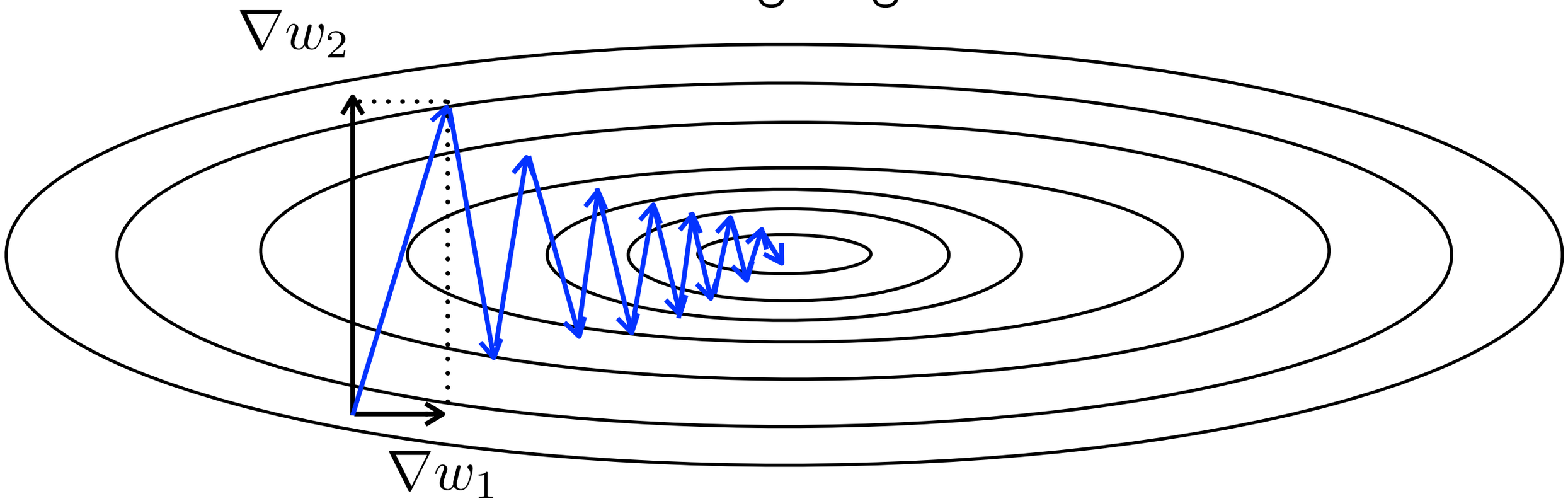


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$$

SGD in 2 dimensional weights

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

Undesired zig-zag behaviour



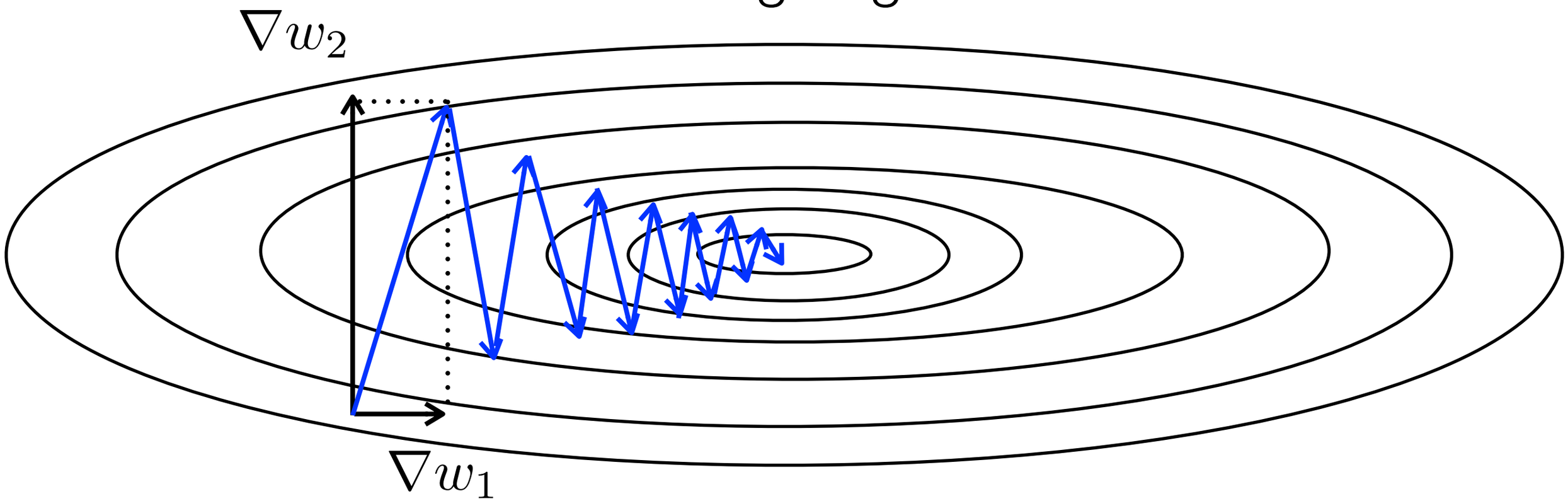
Momentum suppresses this problem partially, but not enough!

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

SGD in 2 dimensional weights

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

Undesired zig-zag behaviour

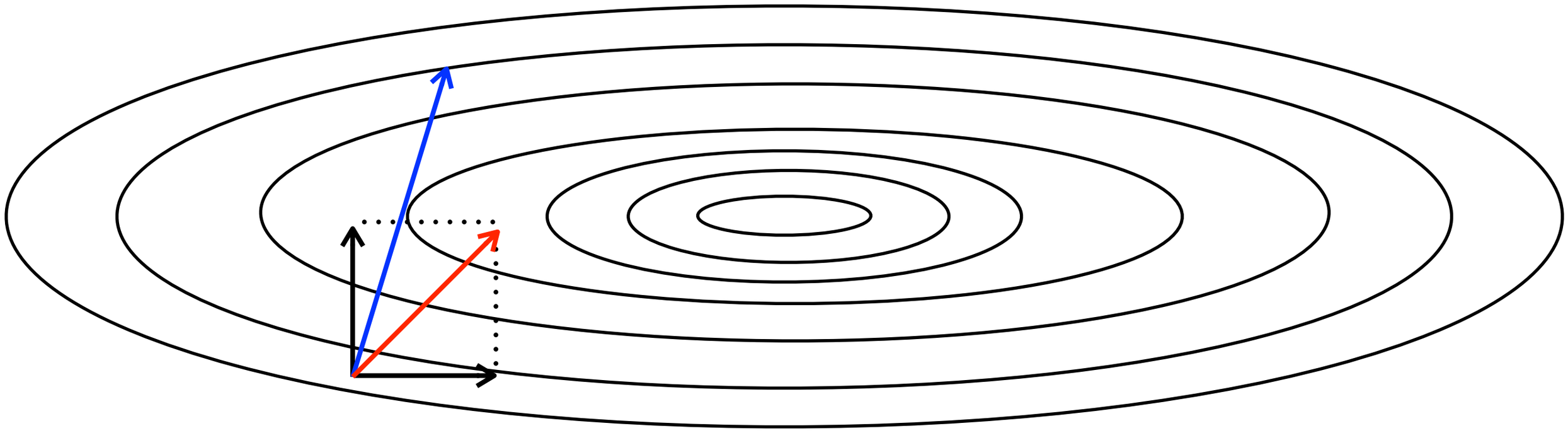


How should we scale particular dimensions?

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

Second order method

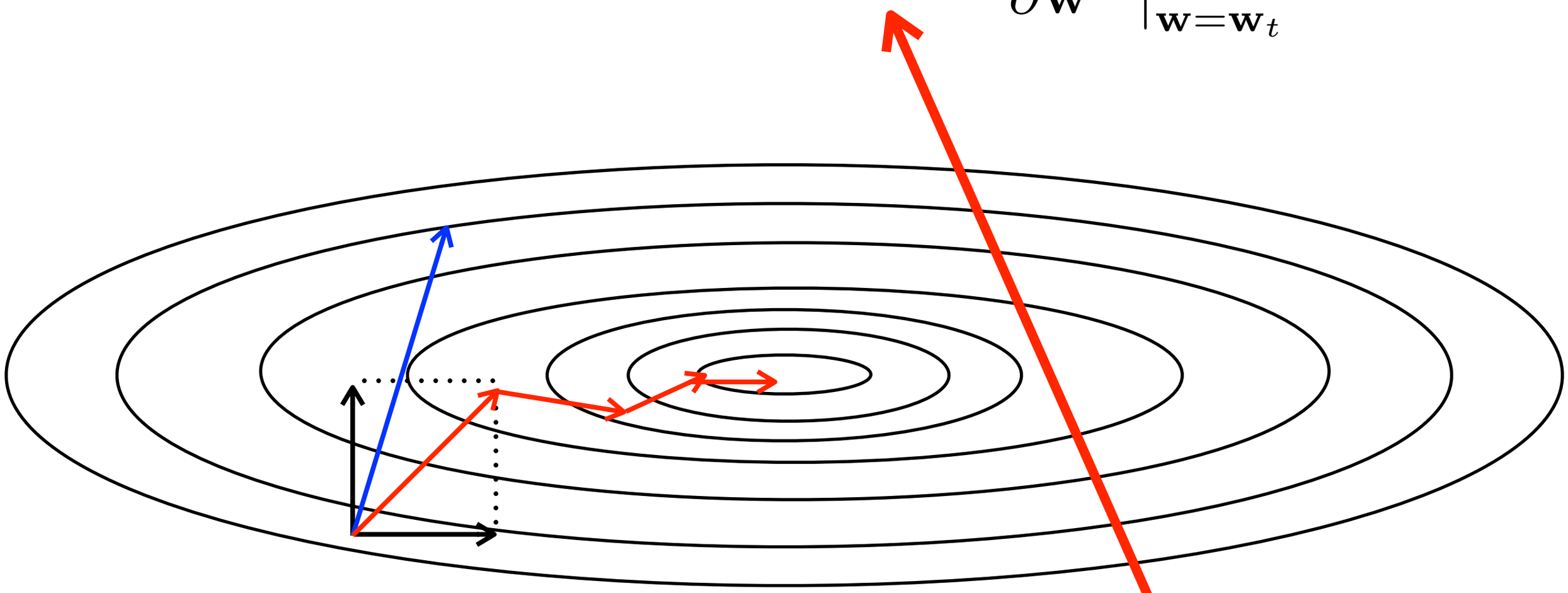
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\, H^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$$



- By the change of the gradient => Hessian $H = \left.\frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}}$
- Speed of change corresponds to eigen-values of Hessian.
- The faster the change the shorter the step

Second order method

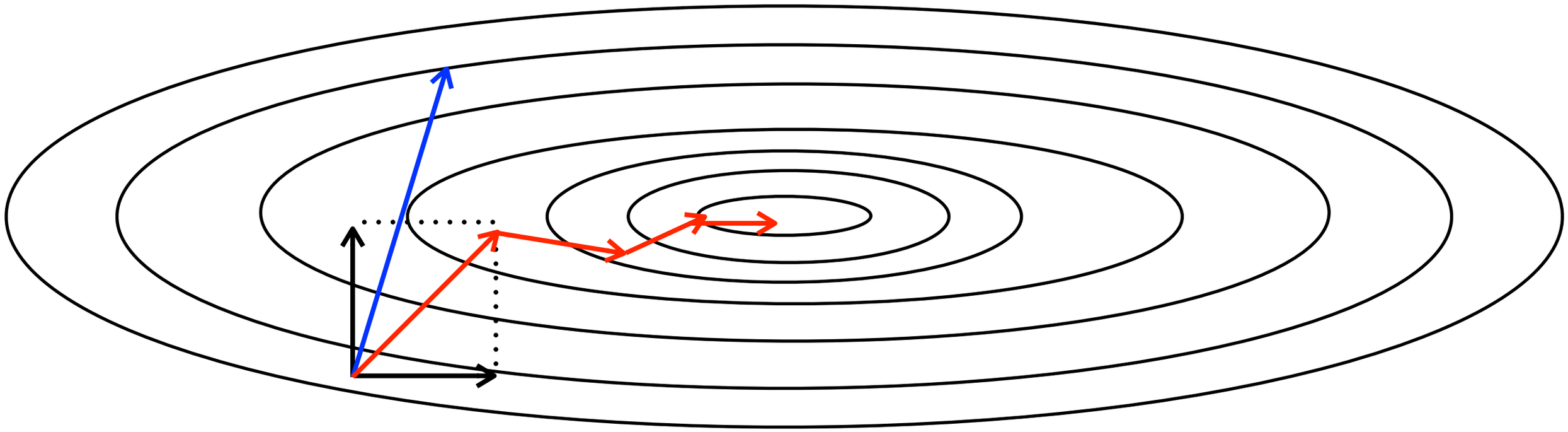
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\, H^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$$



- By the change of the gradient => Hessian $H = \left.\frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}}$
- Speed of change corresponds to eigen-values of Hessian.
- The faster the change the shorter the step

Second order method

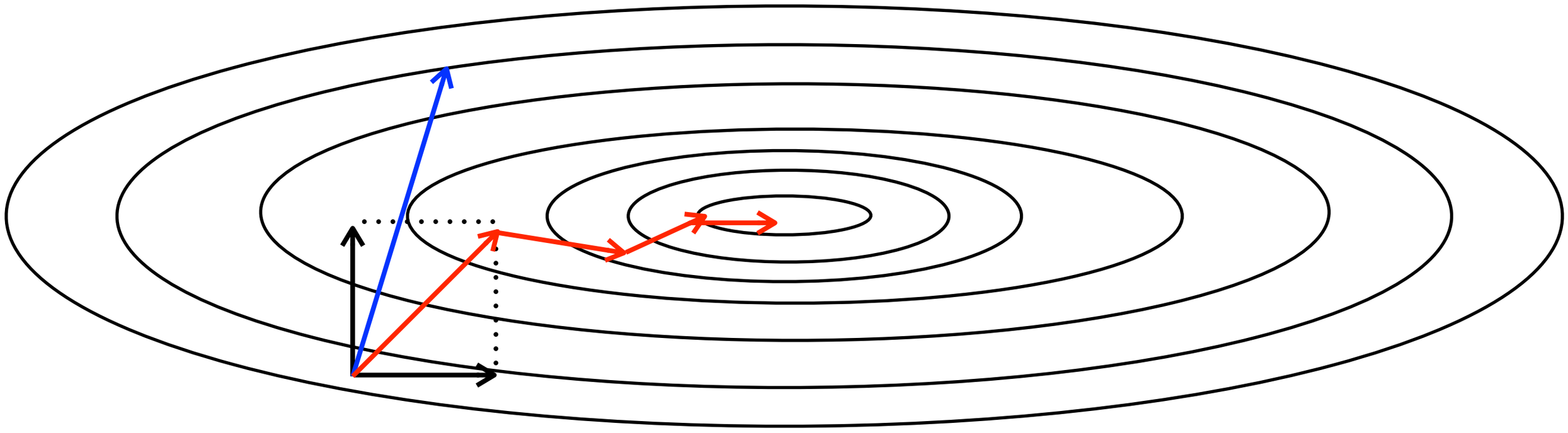
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\, H^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$$



- What is dimensionality of $H = \left.\frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$???
- $N \times N \;\Rightarrow\; \mathcal{O}(N^3)$
- Inversion is technically intractable

AdaGrad

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

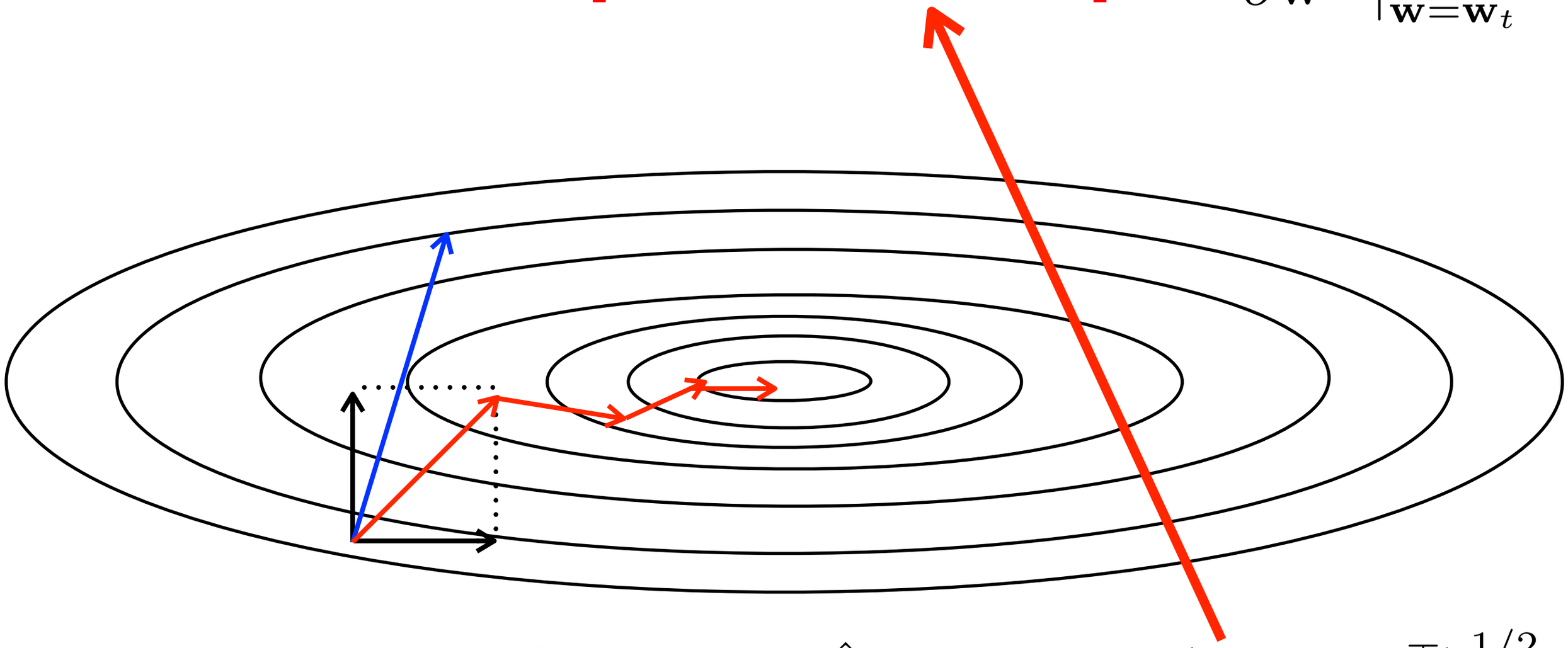


- Approximate Hessian as $\hat{H}(\mathbf{w}_t) \approx \operatorname{diag}\left(\nabla\mathbf{w}_t \nabla\mathbf{w}_t^\top\right)^{1/2}$

  where $\nabla\mathbf{w}_t = \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_t}$

AdaGrad

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \alpha \left[\mathrm{diag}\left(\nabla \mathbf{w}_t \nabla \mathbf{w}_t^\top \right)^{1/2} \right]^{-1} \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$$



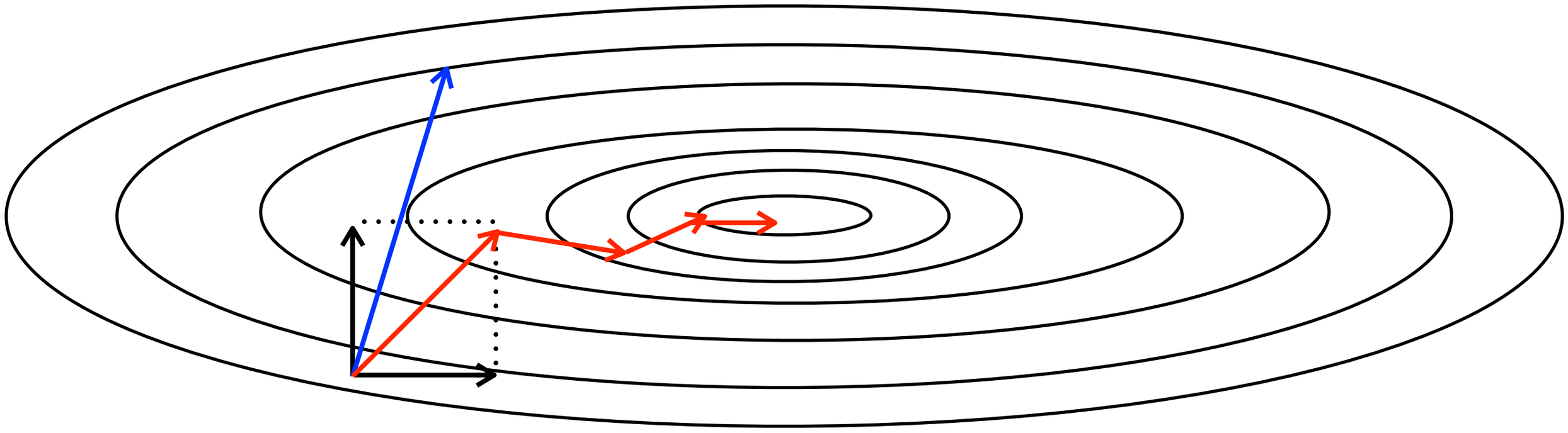
- Approximate Hessian as $\hat{H}(\mathbf{w}_t) \approx \mathrm{diag}\left(\nabla \mathbf{w}_t \nabla \mathbf{w}_t^\top \right)^{1/2}$

  where $\nabla \mathbf{w}_t = \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$

AdaGrad

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \alpha \left[\mathrm{diag}\left(\nabla \mathbf{w}_t \nabla \mathbf{w}_t^\top \right)^{1/2} \right]^{-1} \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$$

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \frac{\alpha}{\sqrt{\nabla \mathbf{w}_t^2 + \epsilon}} \odot \nabla \mathbf{w}_t$$



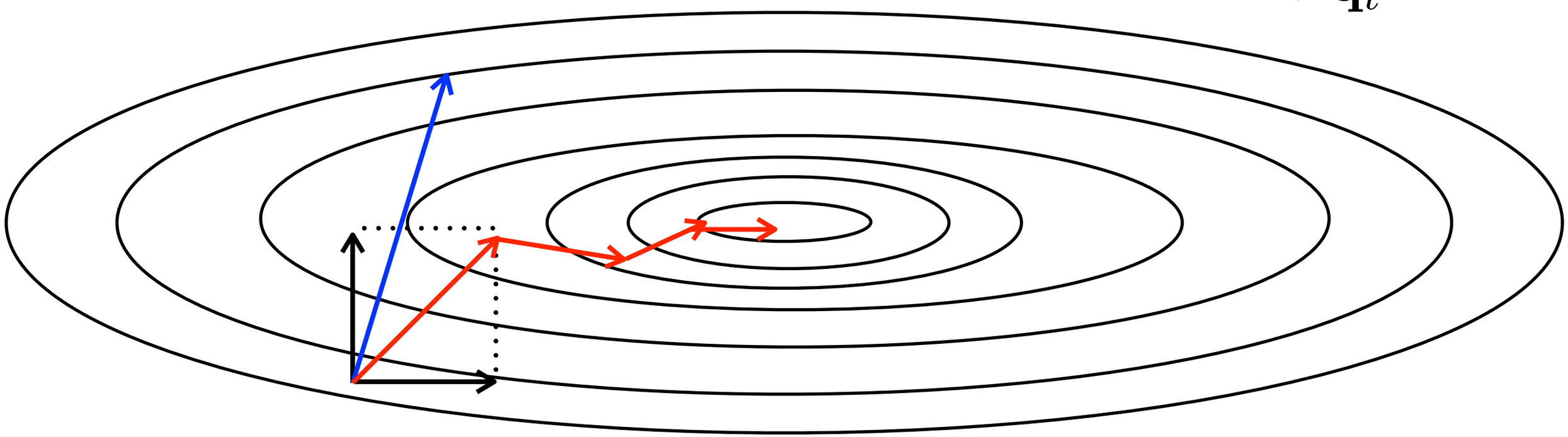
http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

AdamOptimizer = AdaGrad + momentum in $\nabla \mathbf{w}_t,\ \nabla \mathbf{w}_t^2$

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \frac{\alpha}{\sqrt{\nabla \mathbf{w}_t^2 + \epsilon}} \odot \nabla \mathbf{w}_t$$

$\mathbf{v}_t$

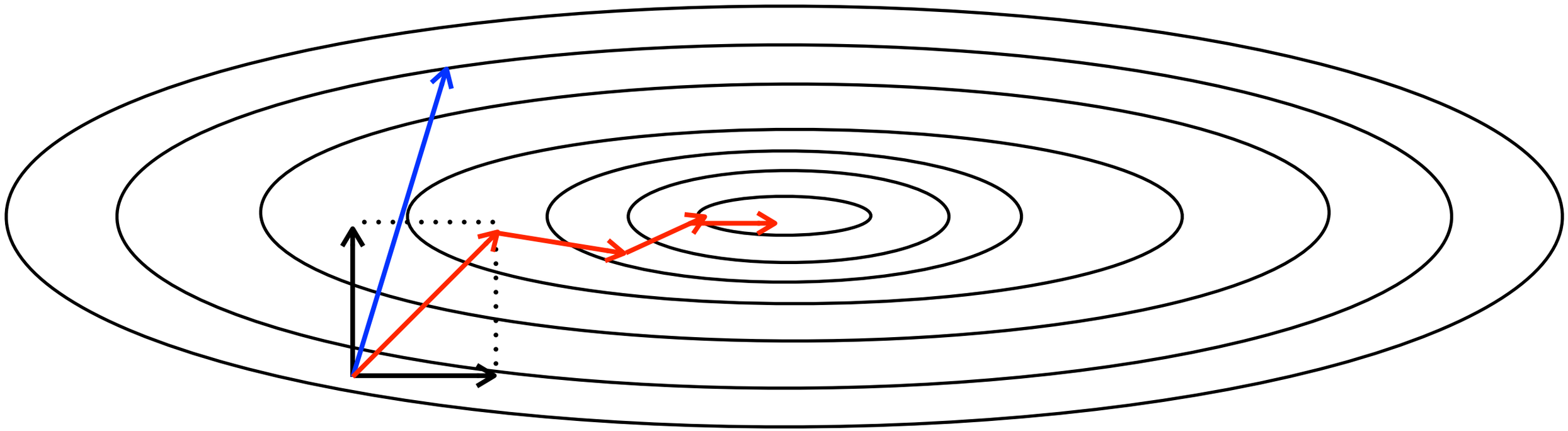
$\mathbf{q}_t$



[Kingma ICLR 2015]

AdamOptimizer = AdaGrad + momentum in $\nabla \mathbf{w}_t,\ \nabla \mathbf{w}_t^2$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{q}_t} + \epsilon} \odot \mathbf{v}_t$$



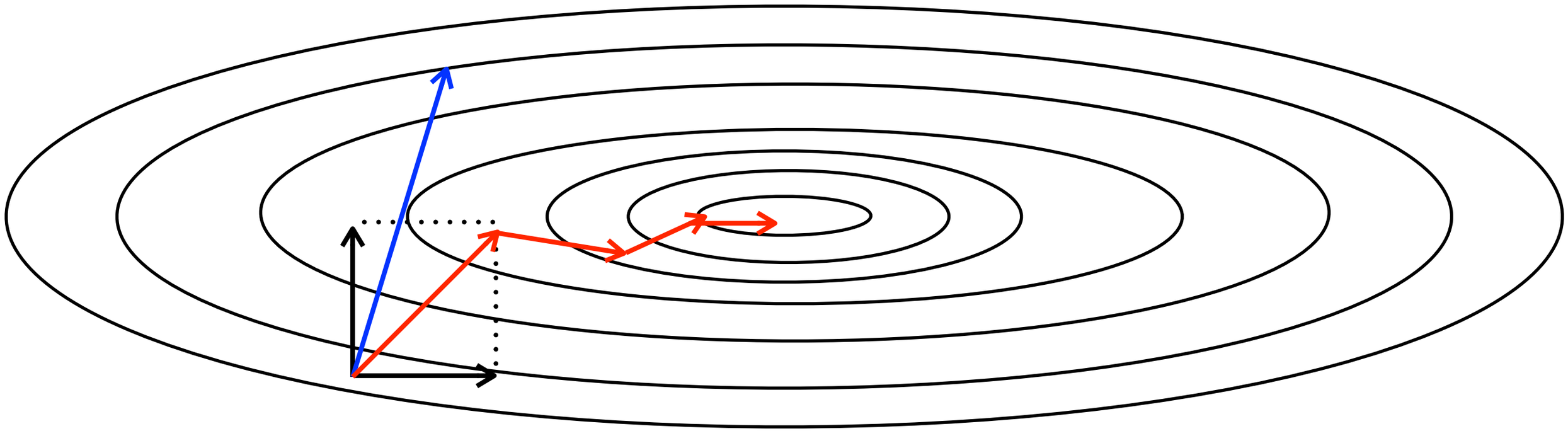
[Kingma ICLR 2015]

AdamOptimizer = AdaGrad + momentum in $\nabla \mathbf{w}_t,\ \nabla \mathbf{w}_t^2$

$$\mathbf{v}_t = \alpha \mathbf{v}_{t-1} + (1-\alpha)\nabla \mathbf{w}_t$$

$$\mathbf{q}_t = \beta \mathbf{q}_{t-1} + (1-\beta)\nabla \mathbf{w}_t^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{q}_t} + \epsilon} \odot \mathbf{v}_t$$



[Kingma ICLR 2015]

References

[Kingma ICLR 2015] https://arxiv.org/pdf/1412.6980.pdf
ADAM: A method for stochastic optimization

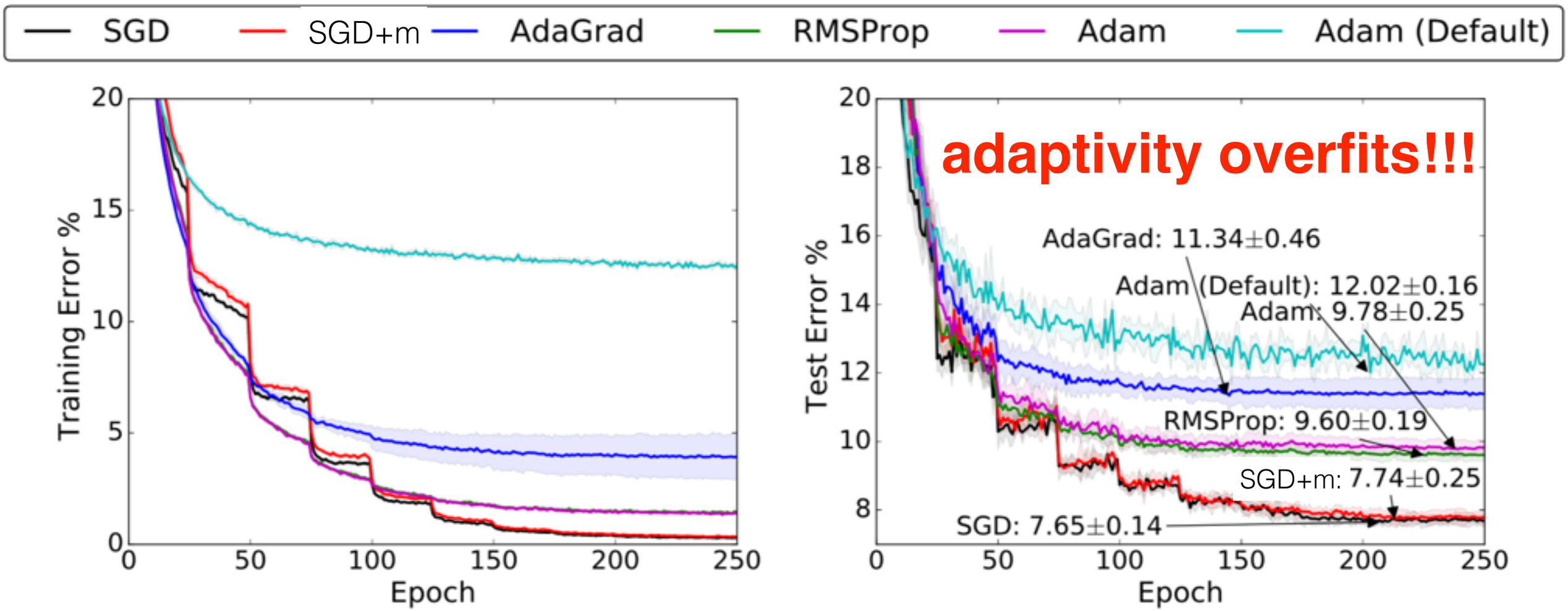
[Ruder 2017] https://arxiv.org/pdf/1609.04747.pdf
An overview of gradient descent optimization algorithm

Summary

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

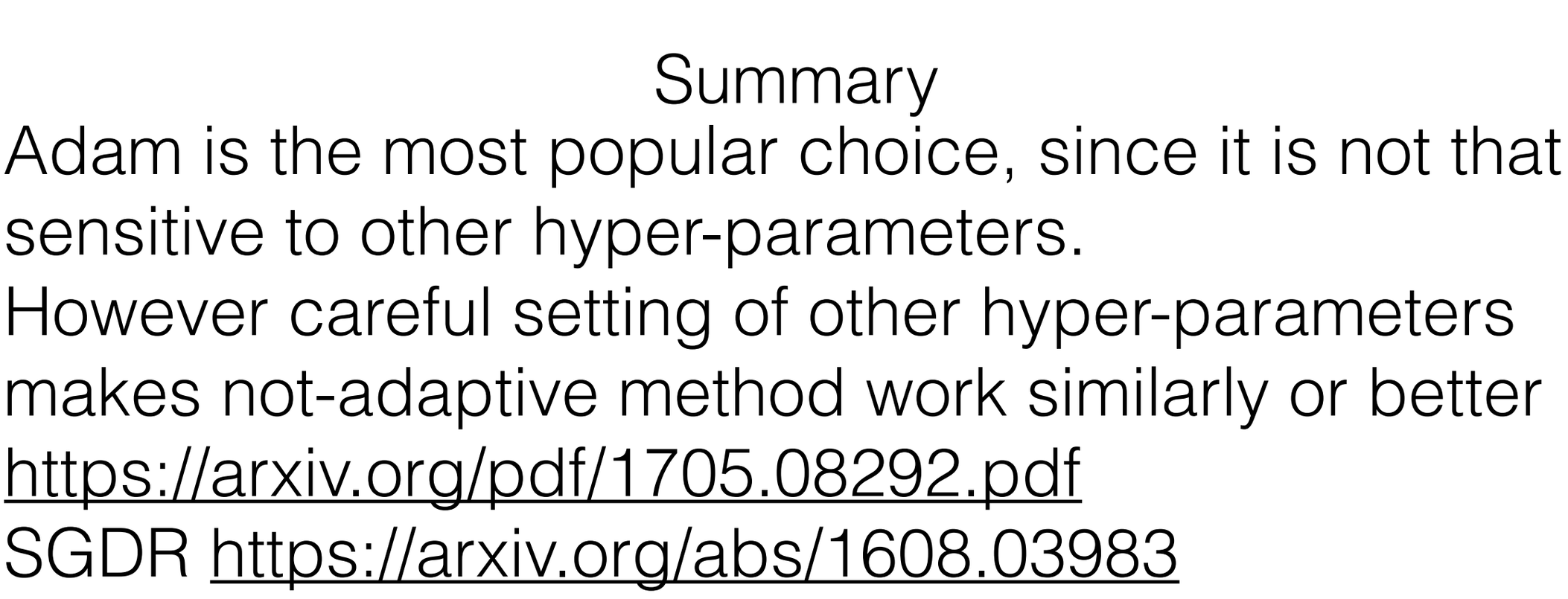
Summary

- Adam is the most popular choice, since it is not that sensitive to other hyper-parameters.
- However careful setting of other hyper-parameters makes not-adaptive method work similarly or better https://arxiv.org/pdf/1705.08292.pdf



Summary

- Adam is the most popular choice, since it is not that sensitive to other hyper-parameters.
- However careful setting of other hyper-parameters makes not-adaptive method work similarly or better https://arxiv.org/pdf/1705.08292.pdf
- SGDR https://arxiv.org/abs/1608.03983



https://medium.com/38th-street-studios/exploring-stochastic-gradient-descent-with-restarts-sgdr-fa206c38a74e

Summary

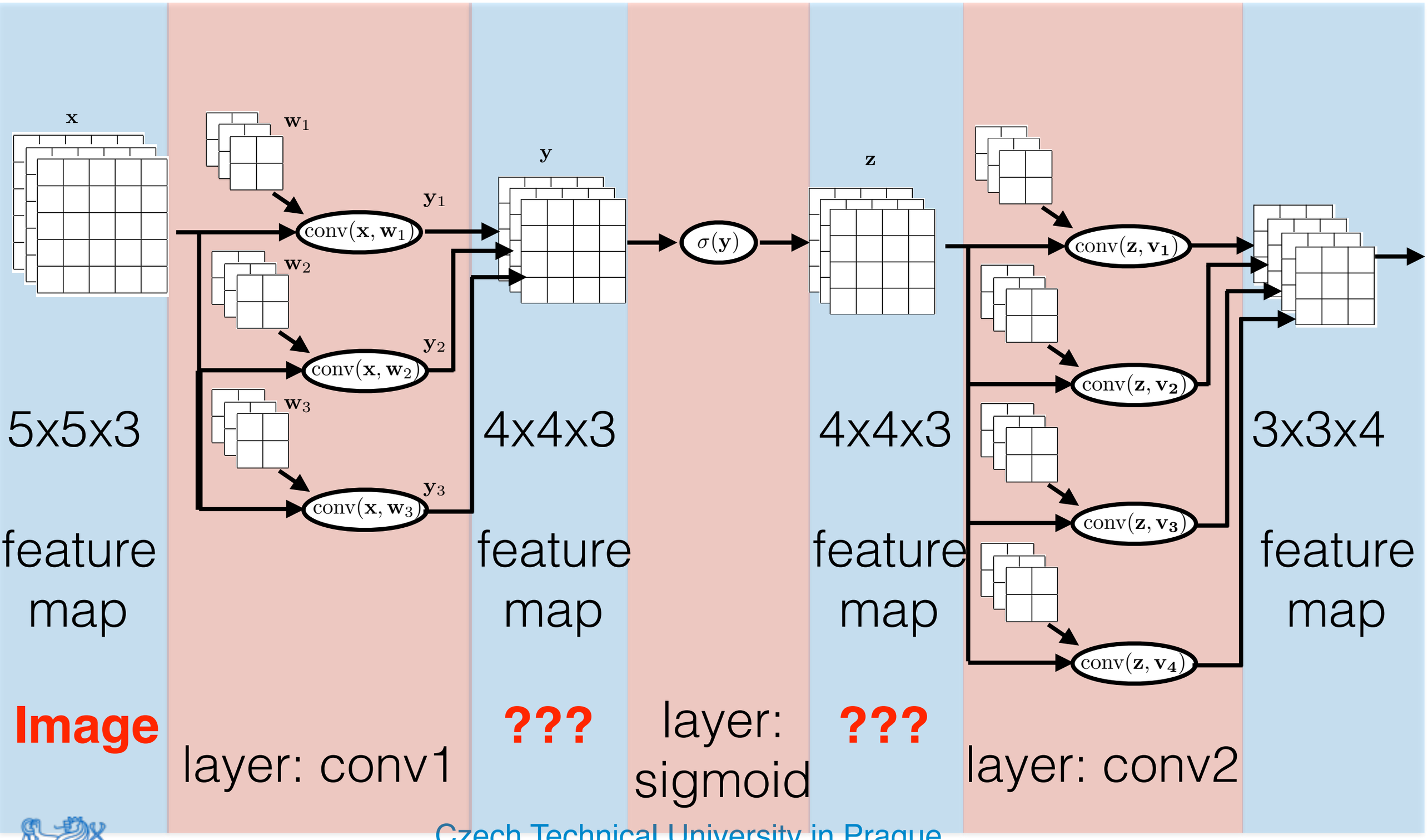
- Whatever is the gradient computation algorithm, we can easily achieve zero gradient!

Outline

- Stochastic Gradient Descent (SGD)
- what happens to gradients during learning
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,
  - regularizations

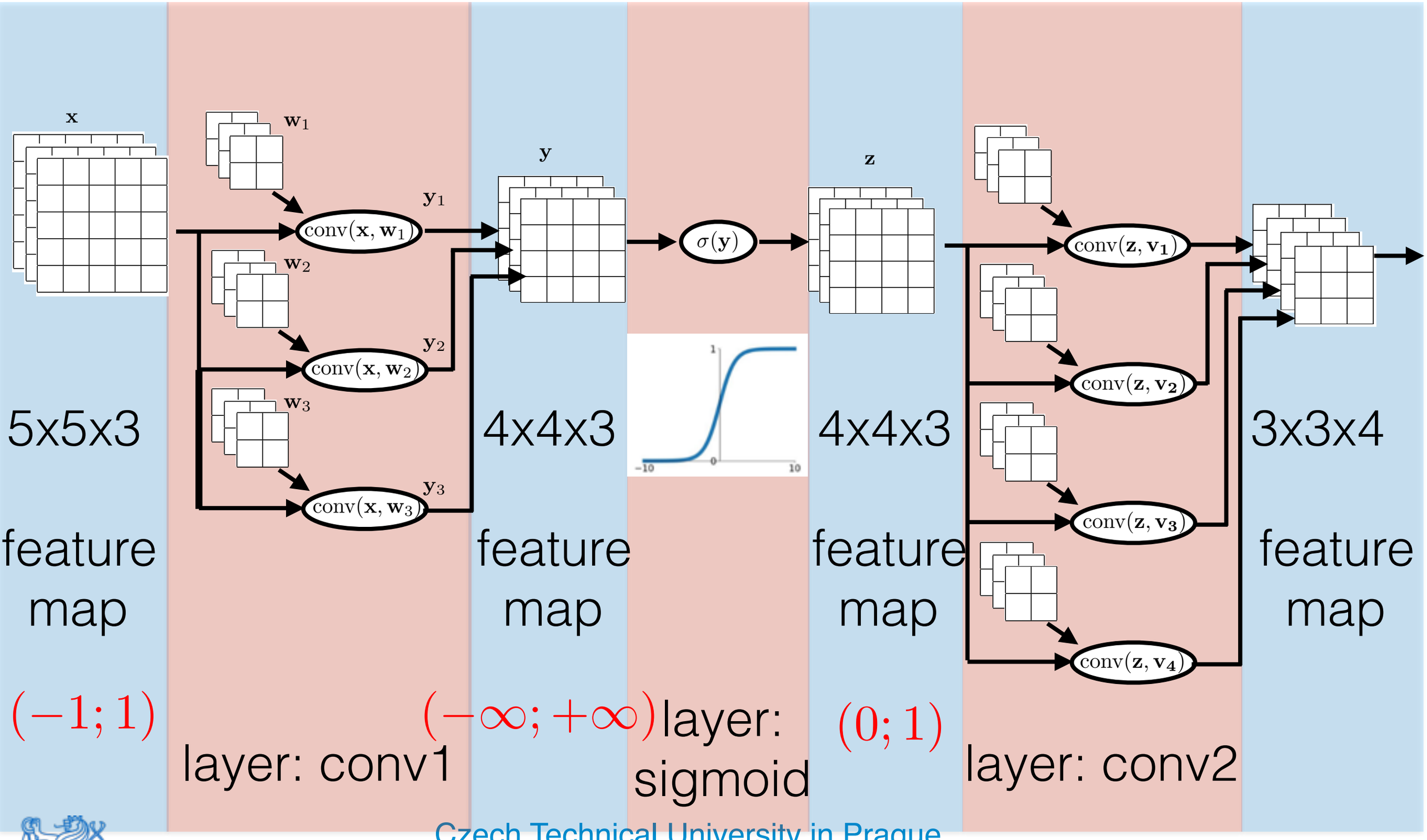
Learning

- let us plug image as input, what **values** are propagated?



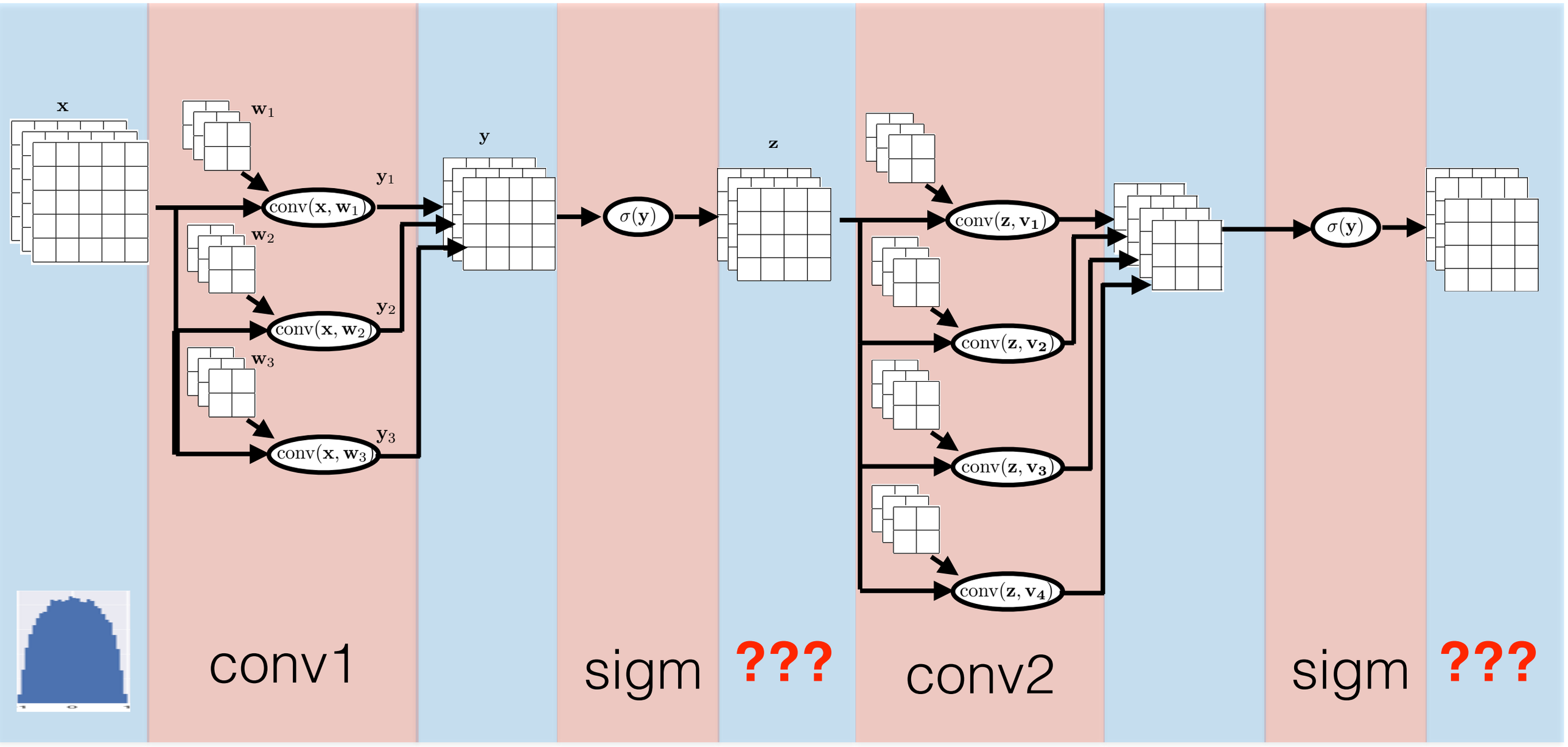
Learning

- let us plug image as input, what **values** are propagated?



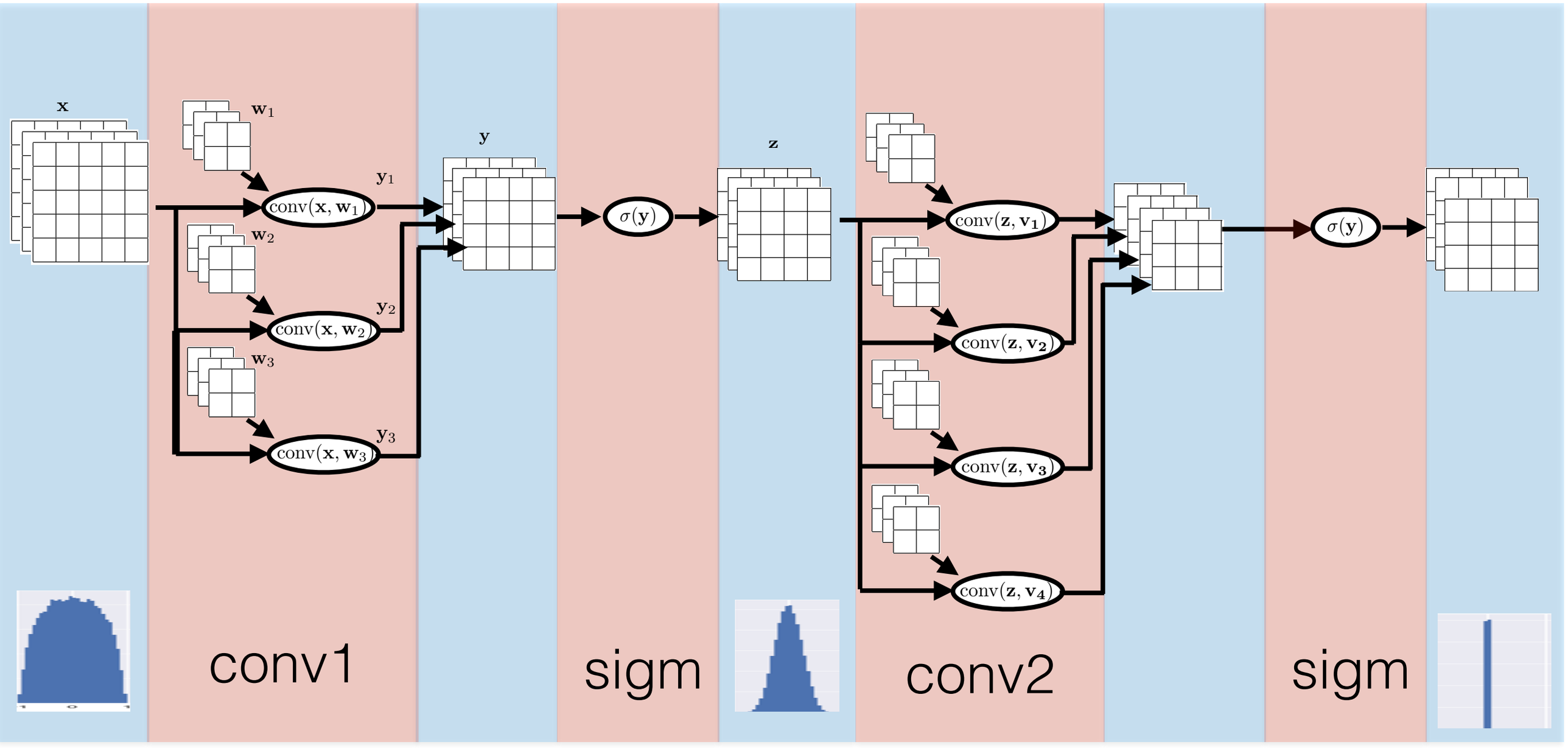
Learning

- what happens to sigm outputs when weights are **small**?



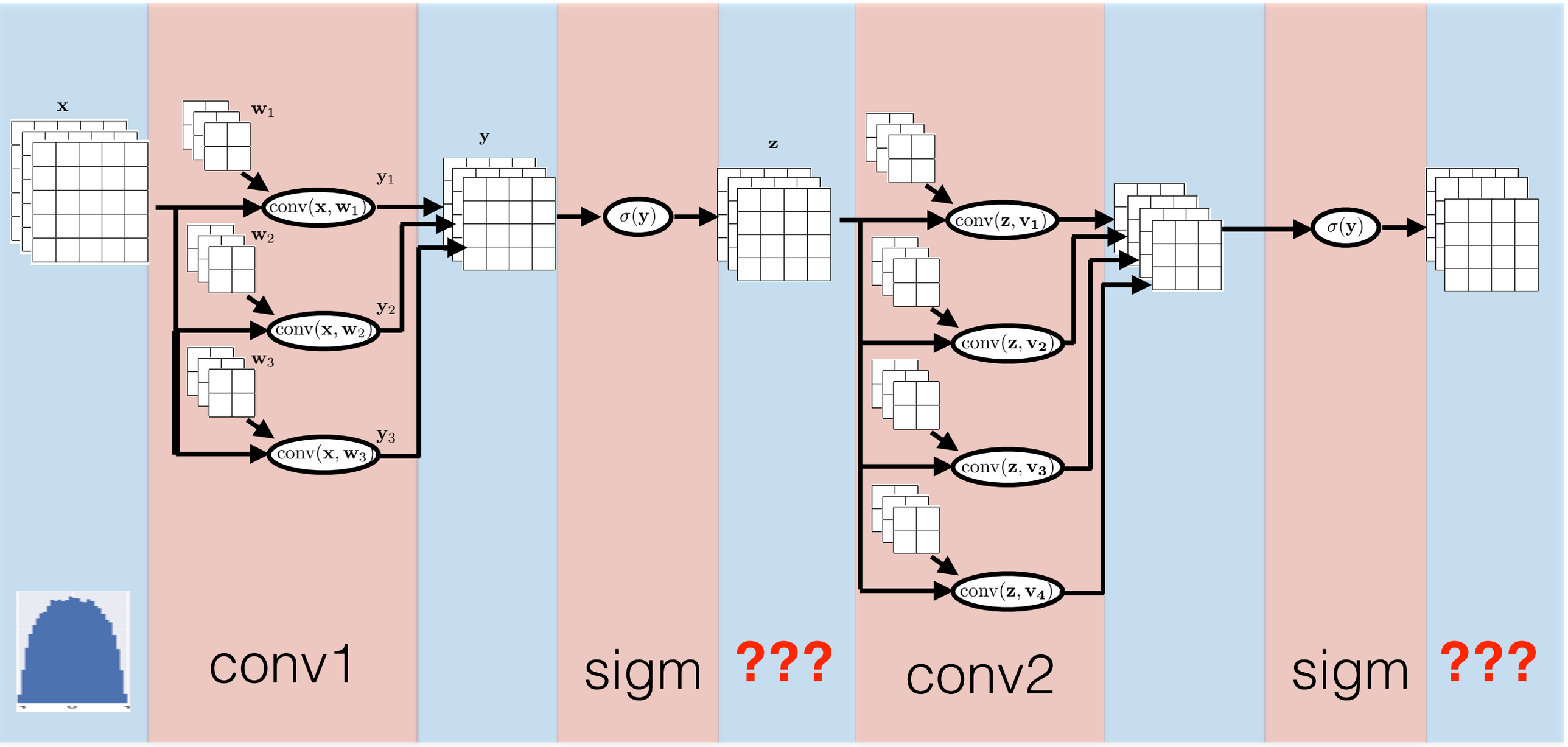
Learning

- what happens to sigm outputs when weights are **small**?



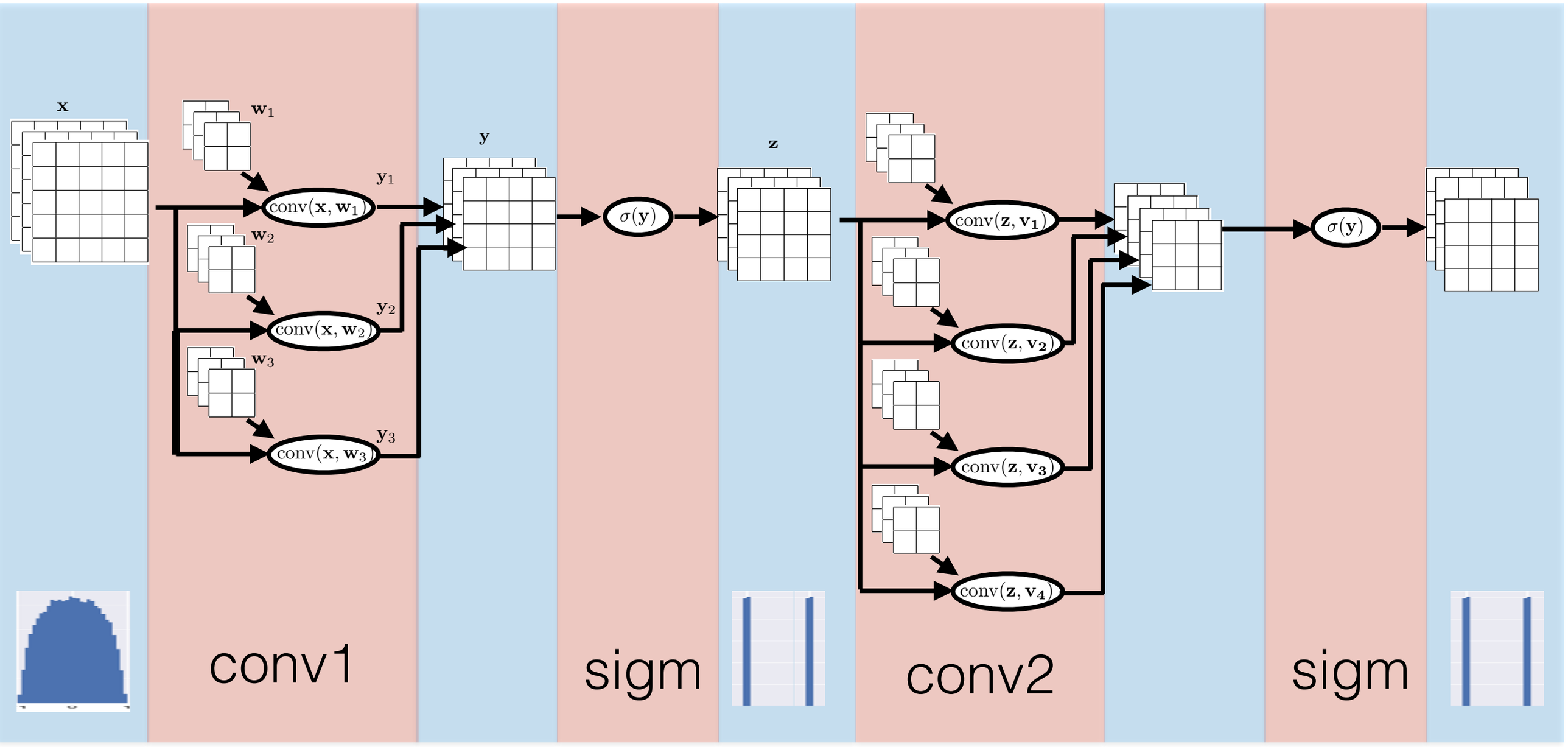
Learning

- what happens to sigm outputs when weights are **huge**?



Learning

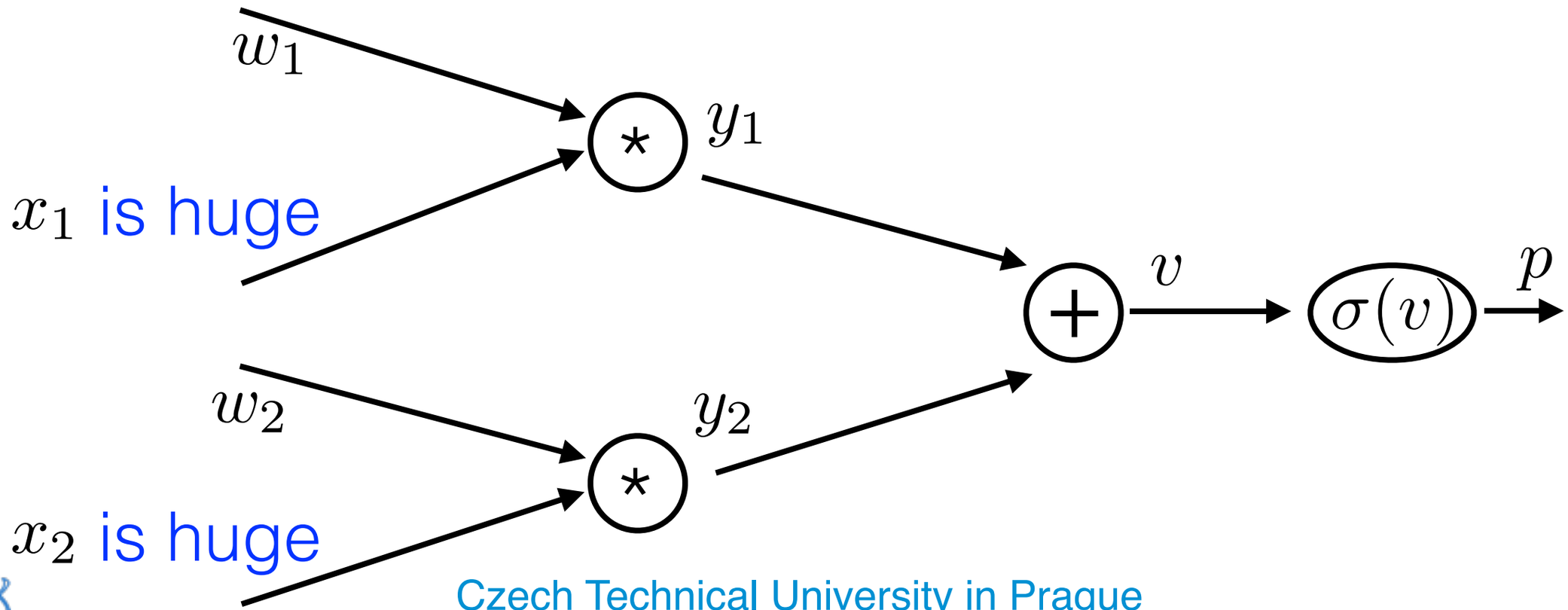
- what happens to sigm outputs when weights are **huge**?



- what happen to backprop gradient when weights are **huge**?

$$\frac{\partial p}{\partial w_1} = ?$$

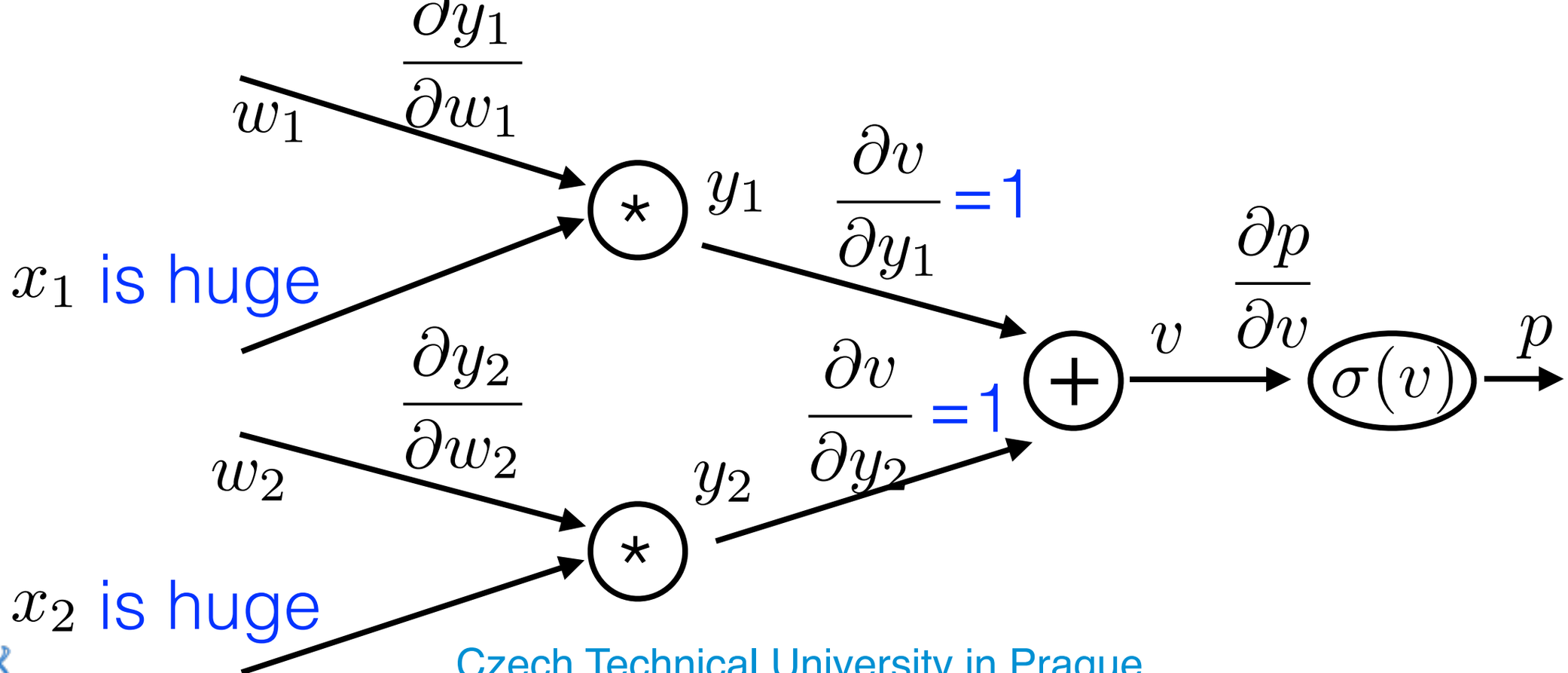
$$\frac{\partial p}{\partial w_2} = ?$$



- what happen to backprop gradient when weights are **huge**?

$\frac{\partial p}{\partial w_1}$=?

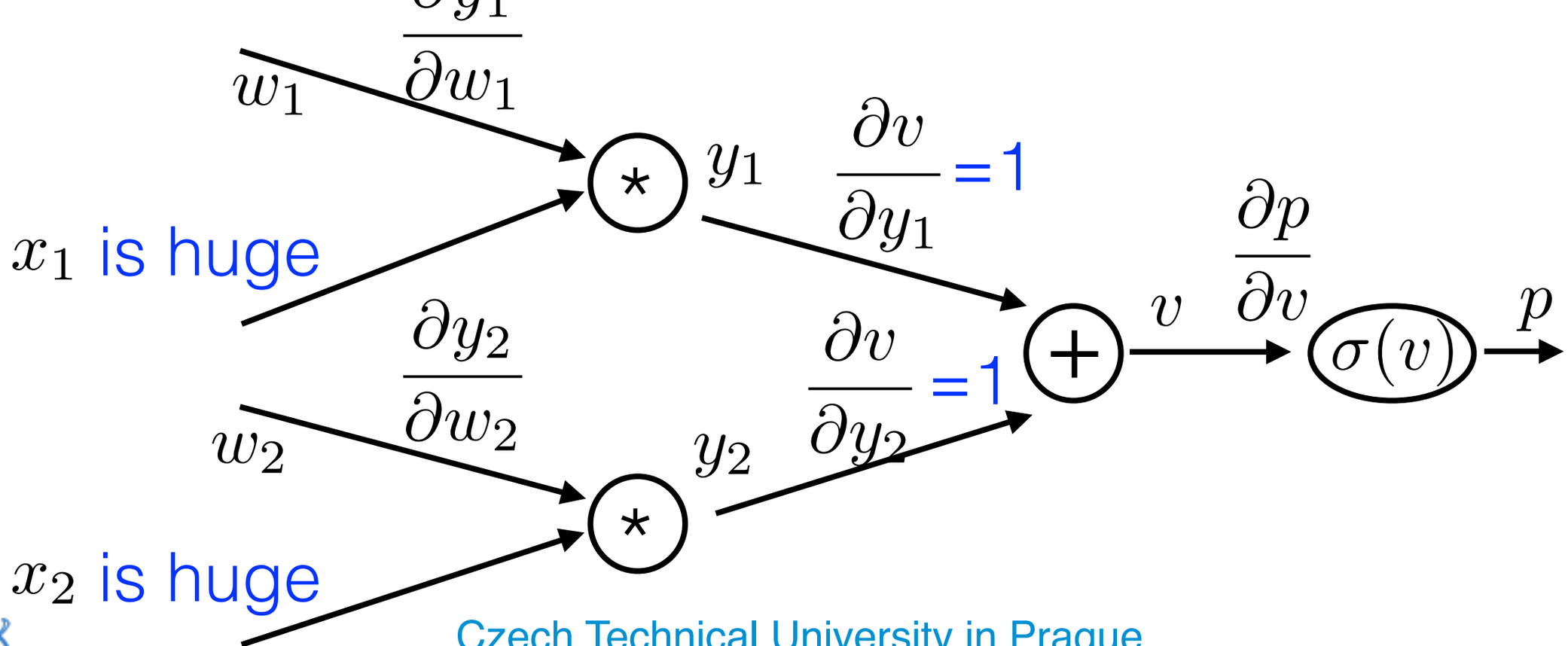
$\frac{\partial p}{\partial w_2}$=?



- what happen to backprop gradient when weights are **huge**?

$$\frac{\partial p}{\partial w_1} = \frac{\partial y_1}{\partial w_1}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} \quad =?$$

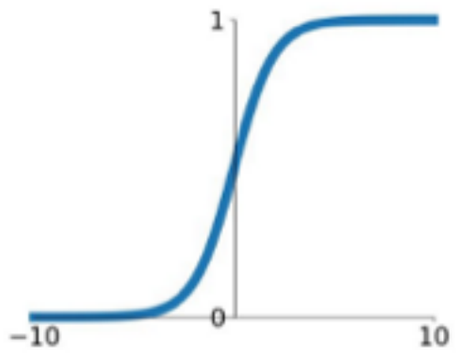
$$\frac{\partial p}{\partial w_2} = \frac{\partial y_2}{\partial w_2}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} \quad =?$$



- what happen to backprop gradient when weights are **huge**?

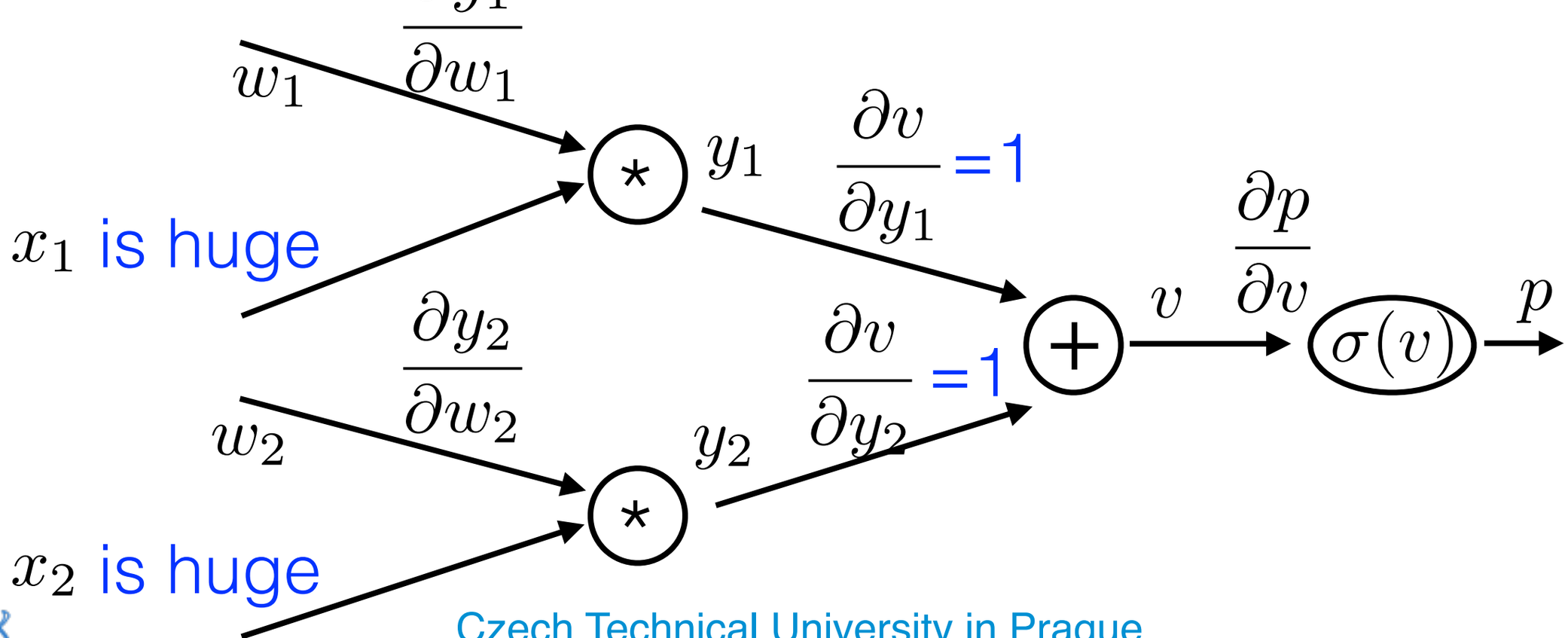
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



$$\frac{\partial p}{\partial w_1} = \frac{\partial y_1}{\partial w_1}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} \quad =?$$

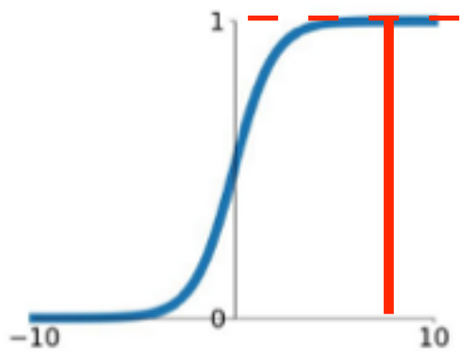
$$\frac{\partial p}{\partial w_2} = \frac{\partial y_2}{\partial w_2}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} \quad =?$$



- what happen to backprop gradient when weights are **huge**?

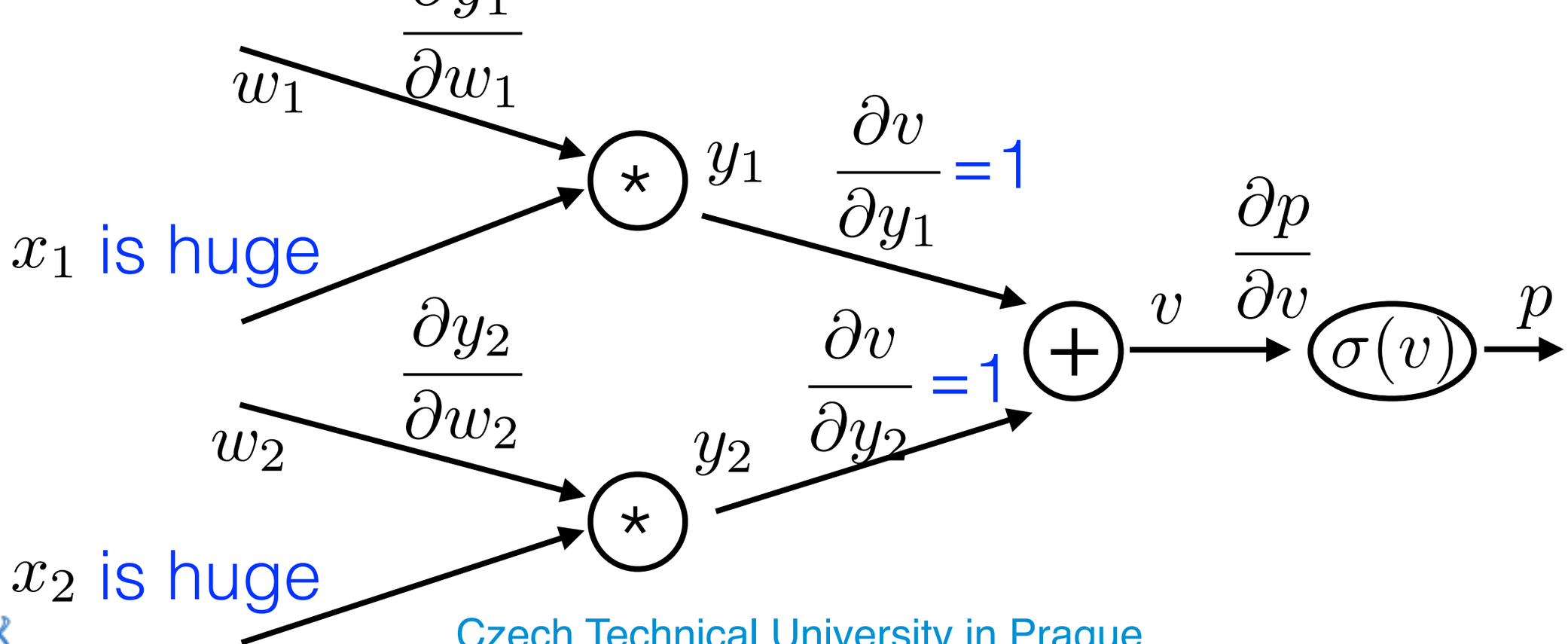
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



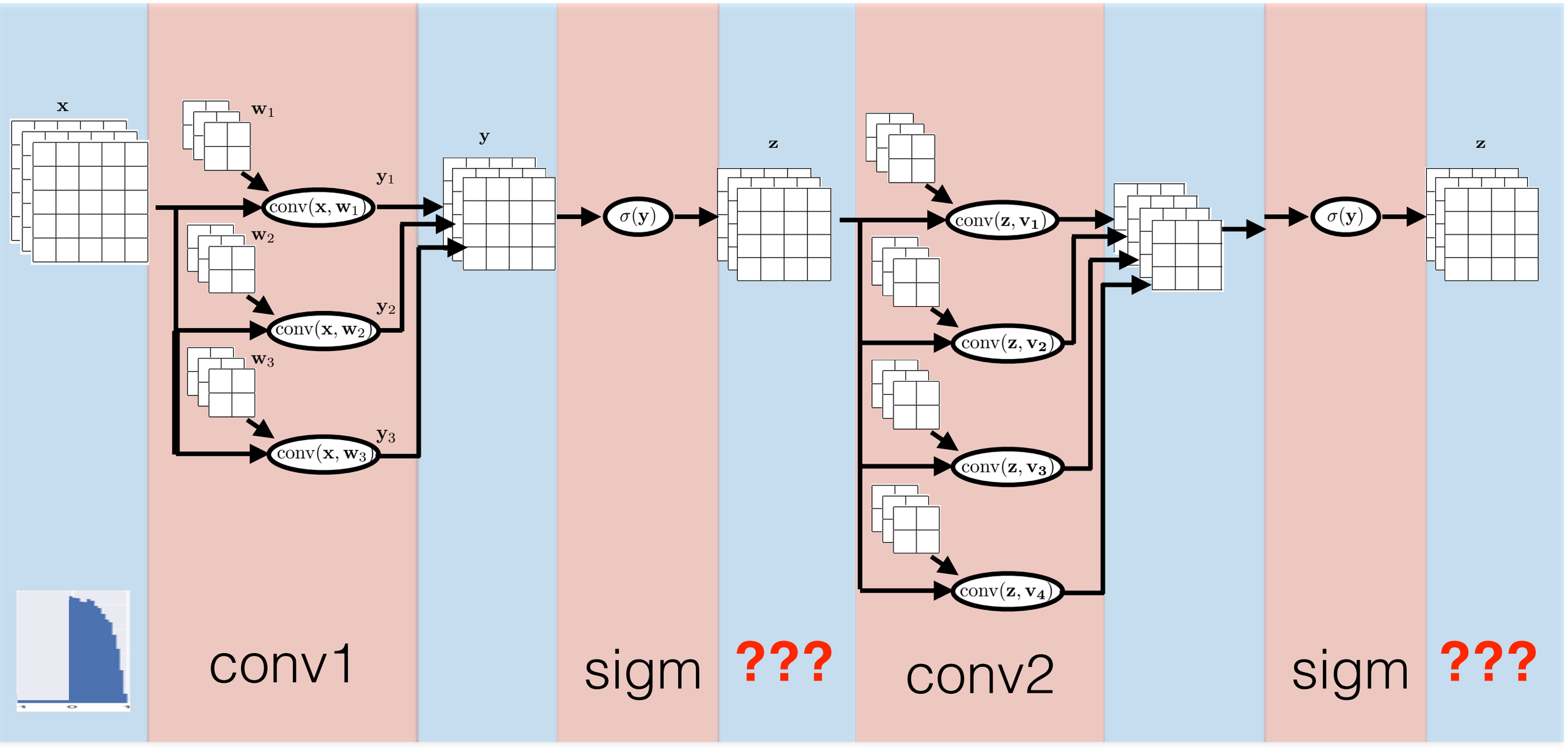
$$\frac{\partial p}{\partial v} = 0$$

$$\frac{\partial p}{\partial w_1} = \frac{\partial y_1}{\partial w_1}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} = 0 \qquad \frac{\partial p}{\partial w_2} = \frac{\partial y_2}{\partial w_2}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} = 0$$



Learning

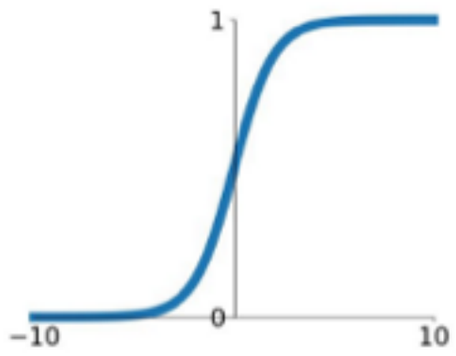
- what happens when conv input is only positive?



Learning

Sigmoid

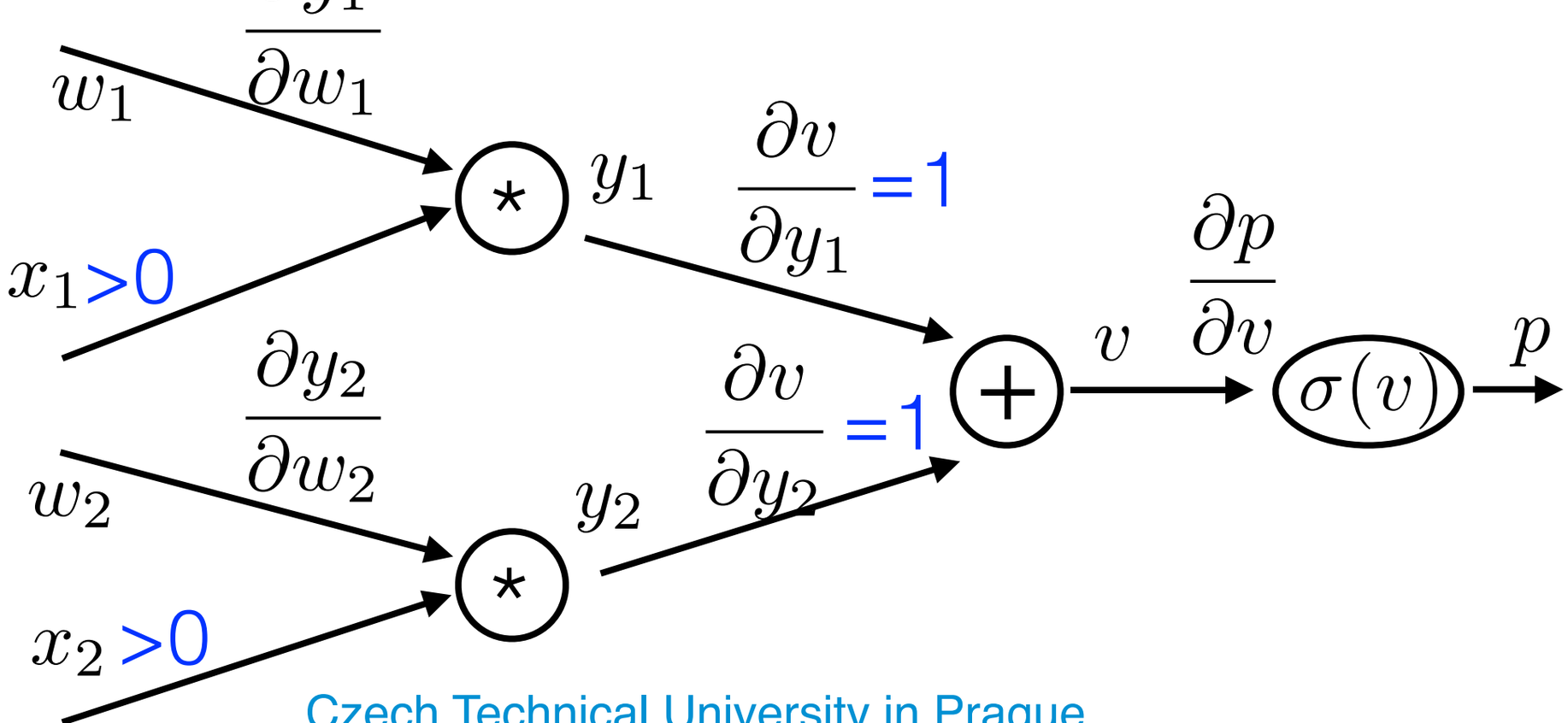
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = \frac{\partial y_1}{\partial w_1}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} \quad =?$$

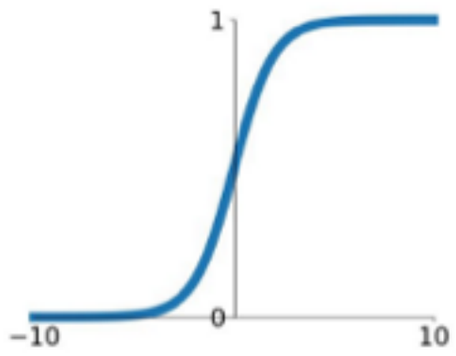
$$\frac{\partial p}{\partial w_2} = \frac{\partial y_2}{\partial w_2}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} \quad =?$$



Learning

Sigmoid

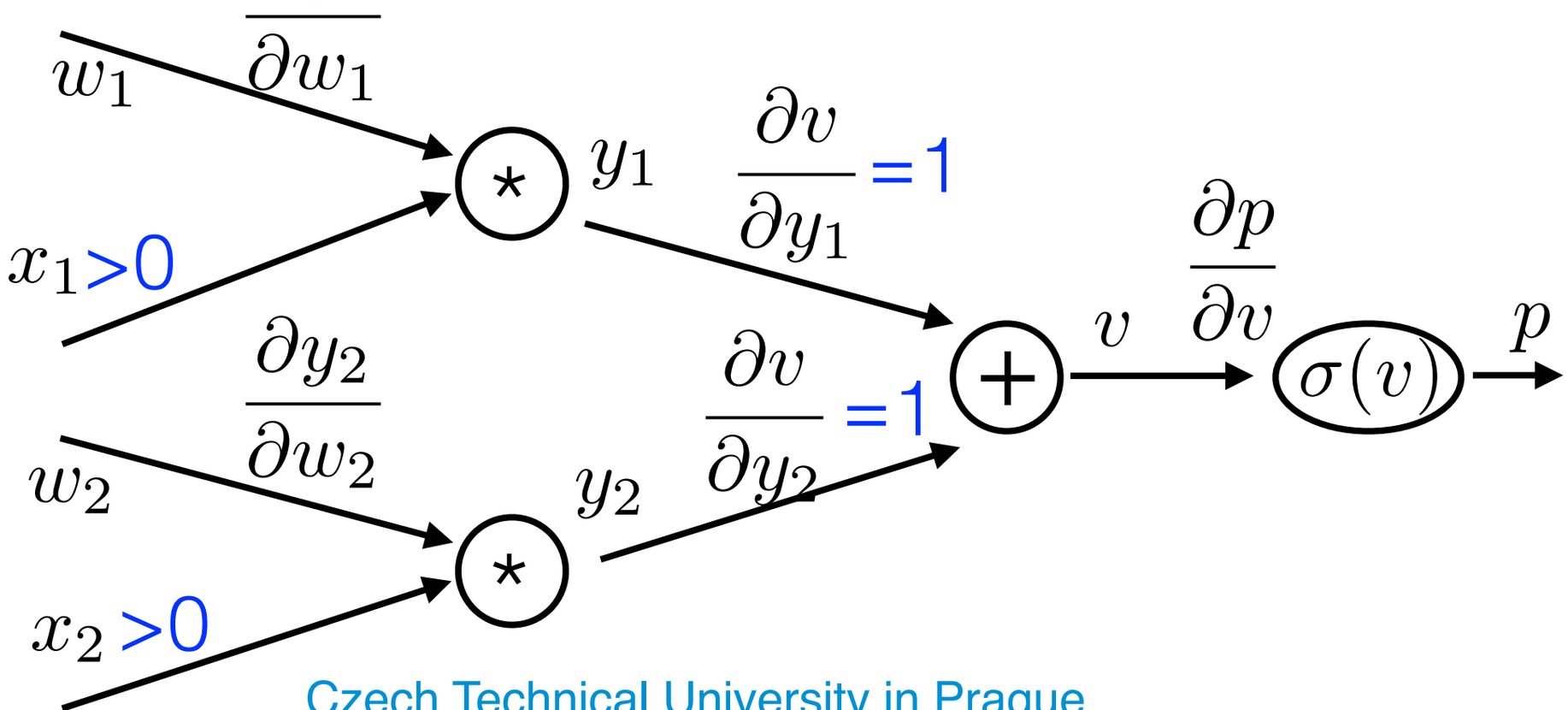
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = \frac{\partial y_1}{\partial w_1} \cdot 1 \cdot \frac{\partial p}{\partial v} \quad =?$$

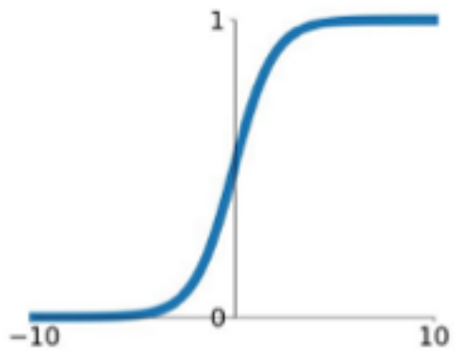
$$\frac{\partial p}{\partial w_2} = \frac{\partial y_2}{\partial w_2} \cdot 1 \cdot \frac{\partial p}{\partial v} \quad =?$$



Learning

Sigmoid

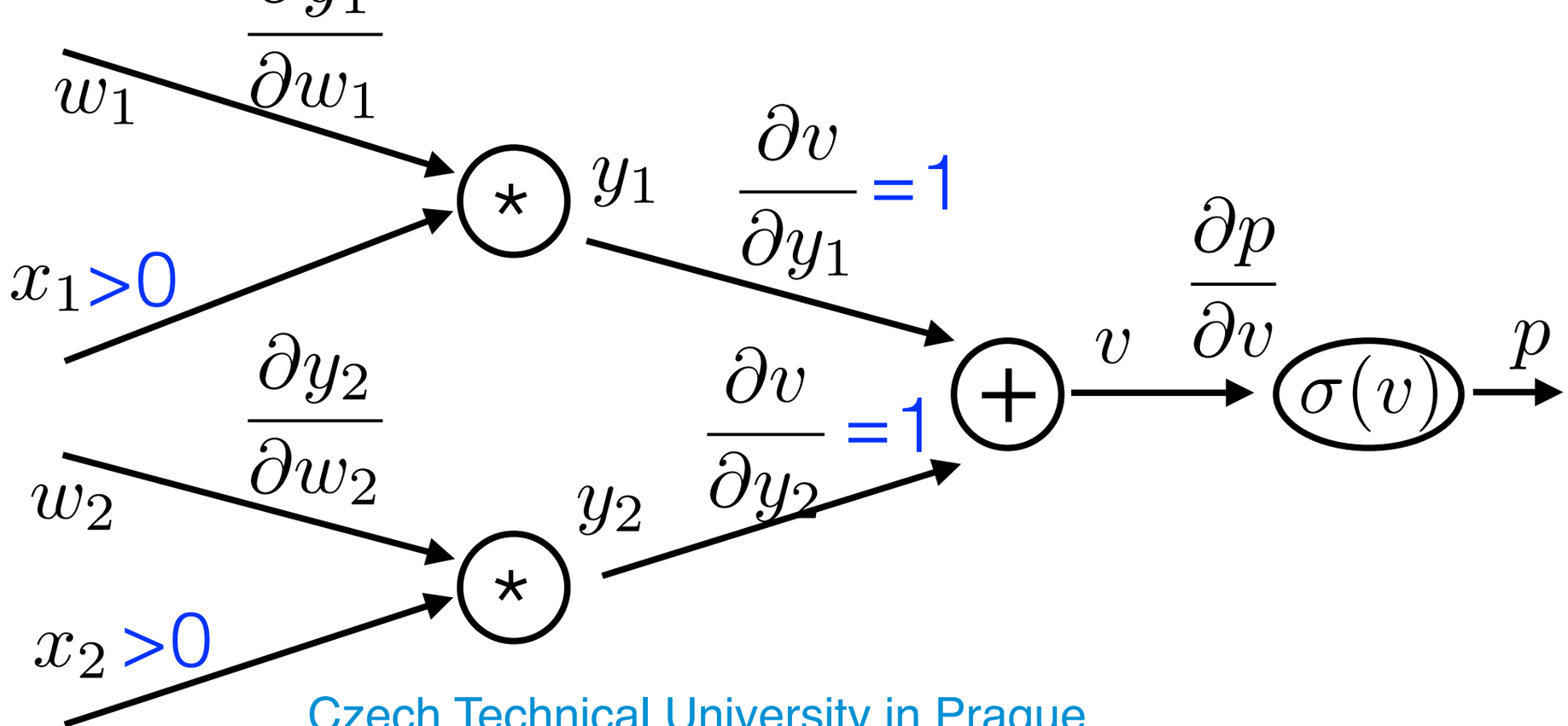
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \; =?$$

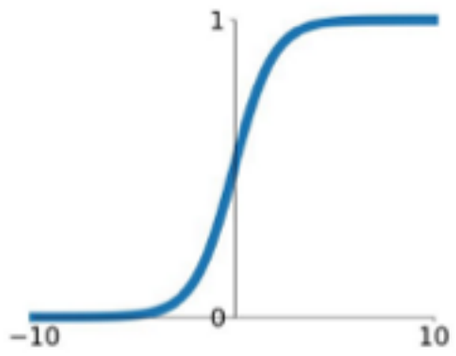
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \; =?$$



Learning

Sigmoid

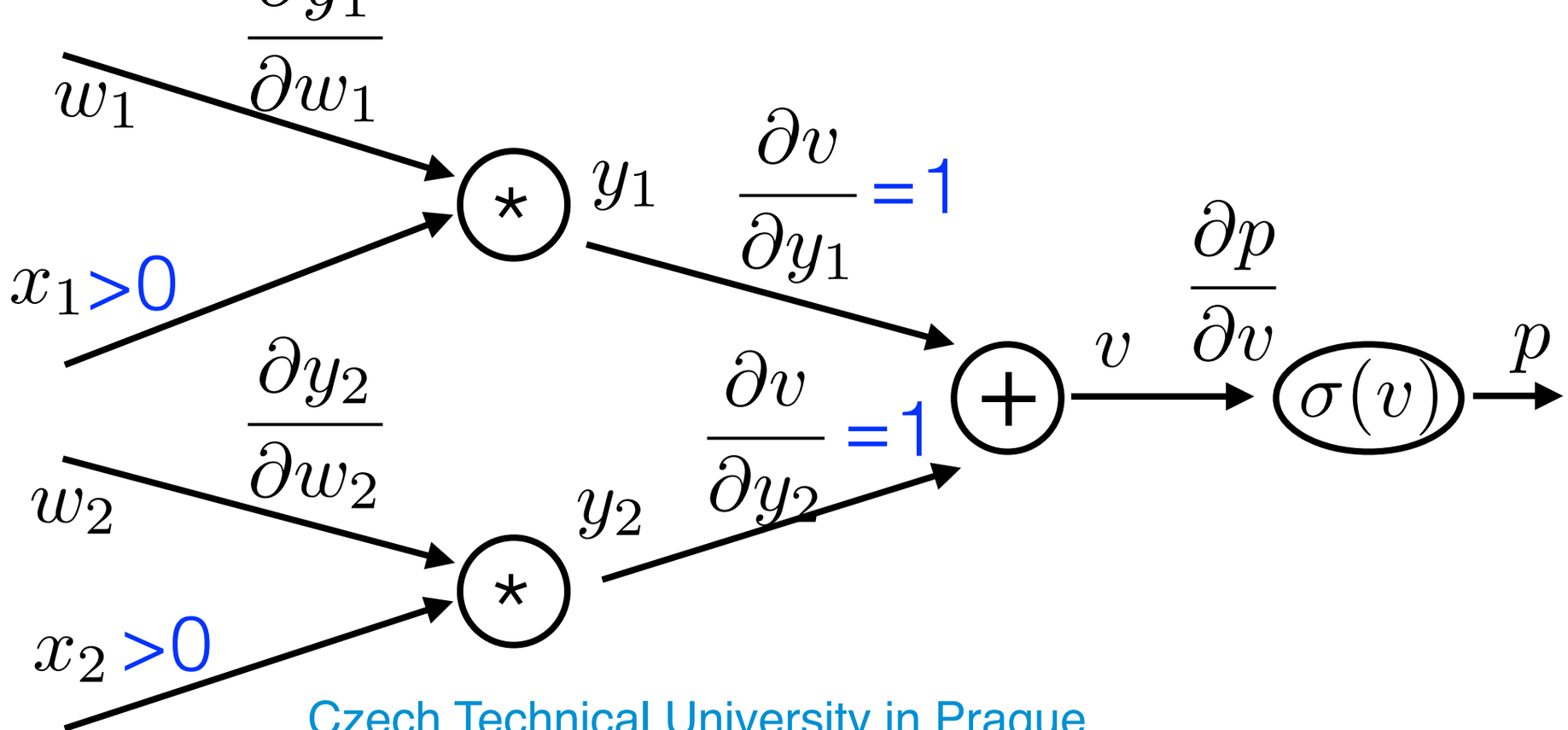
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad >0$$

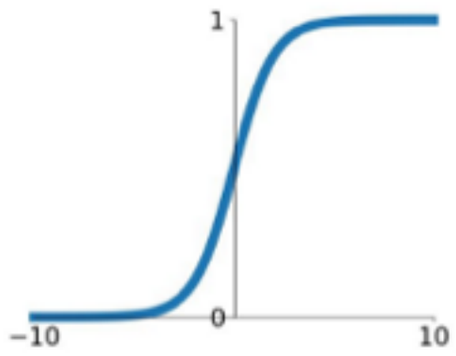
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad >0$$



Learning

Sigmoid

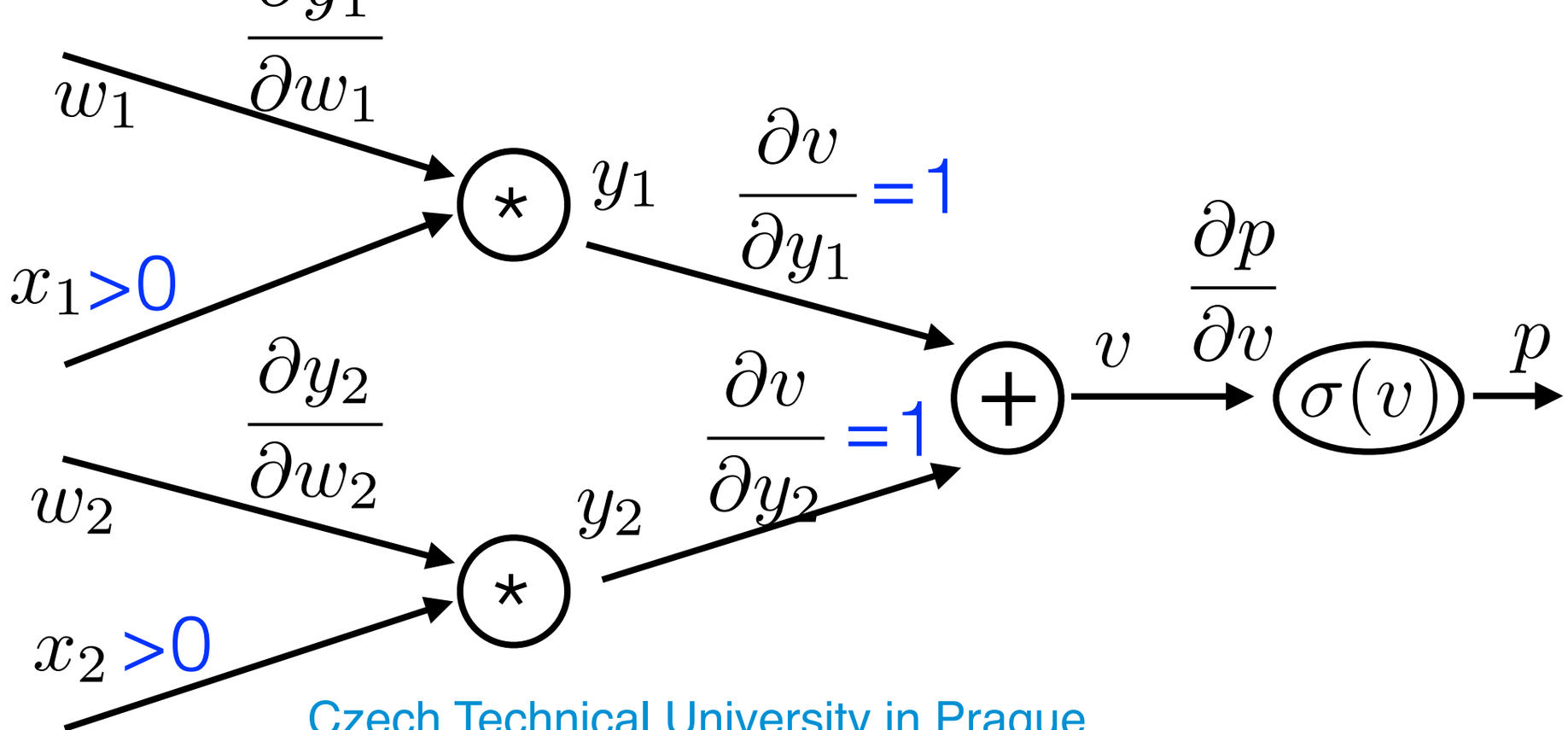
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

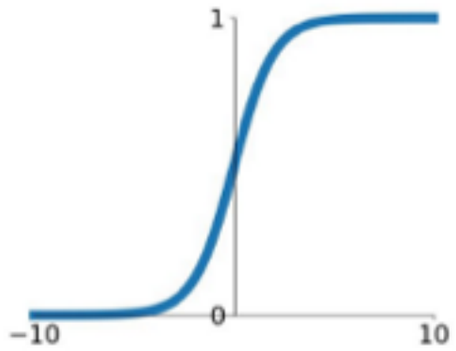
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

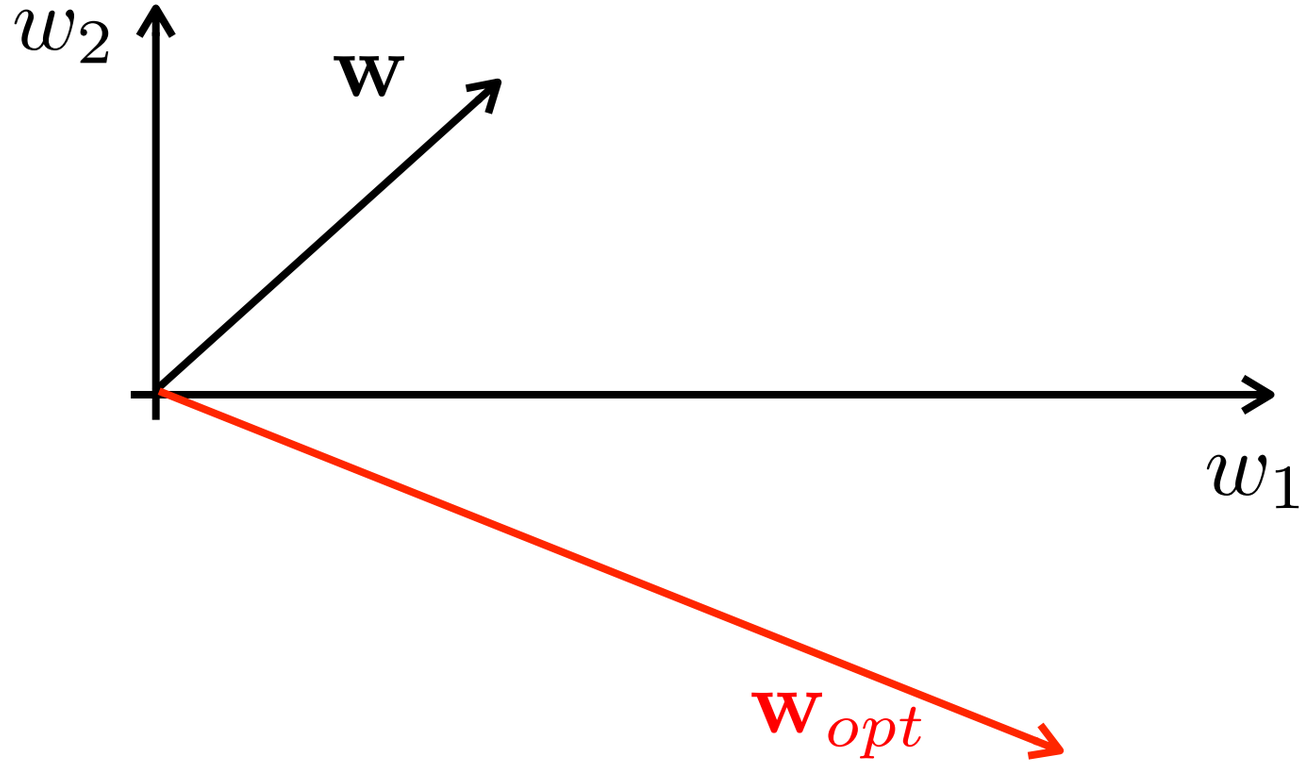
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

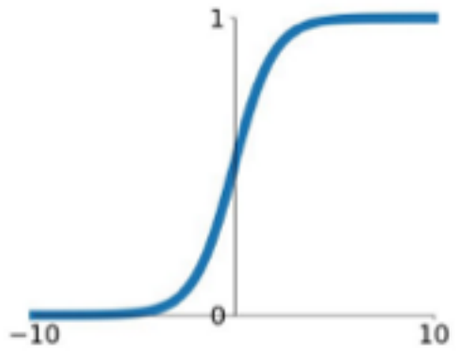
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

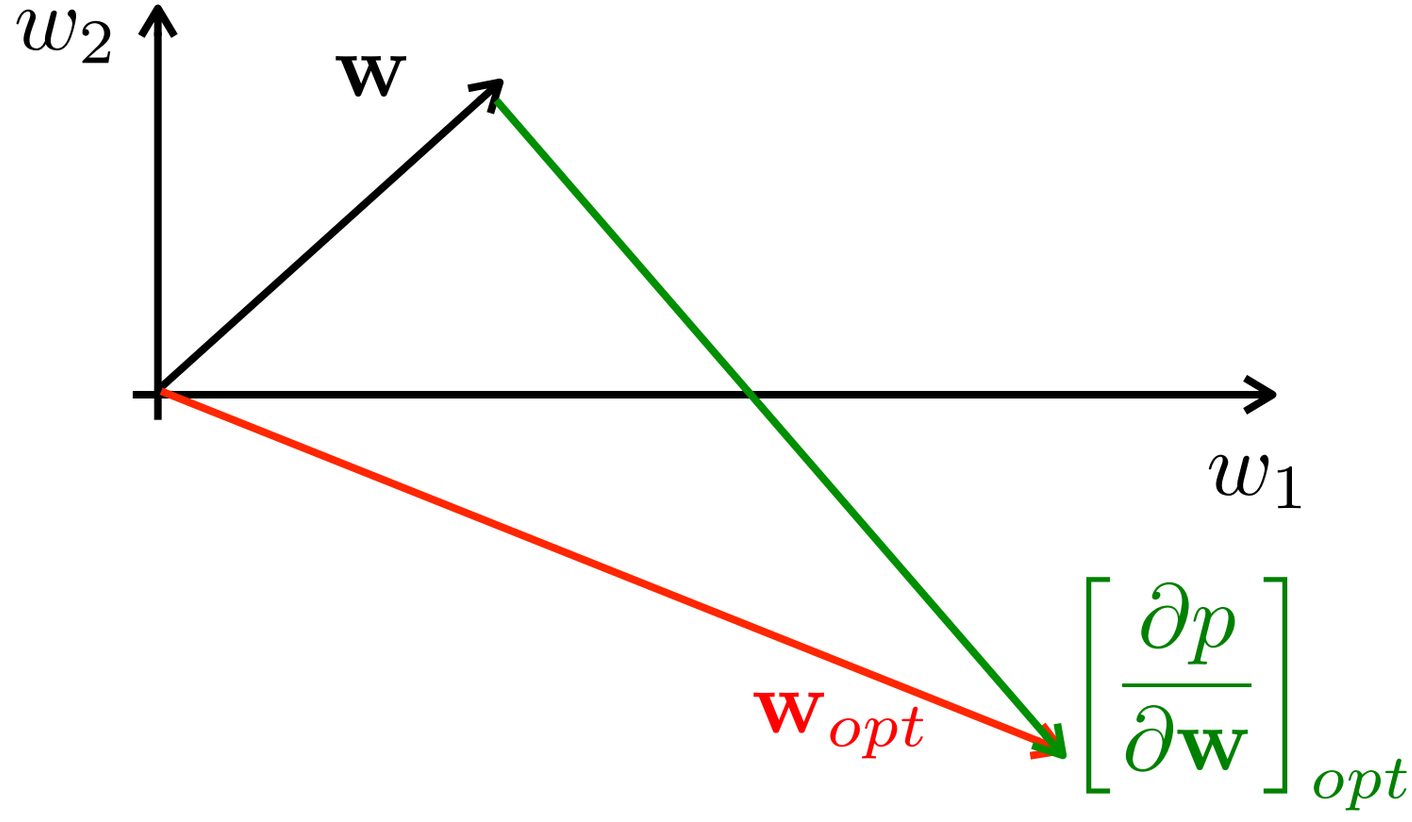
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

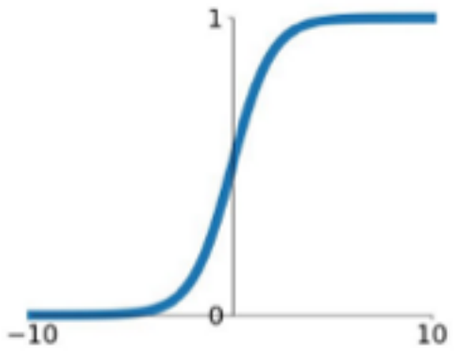
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

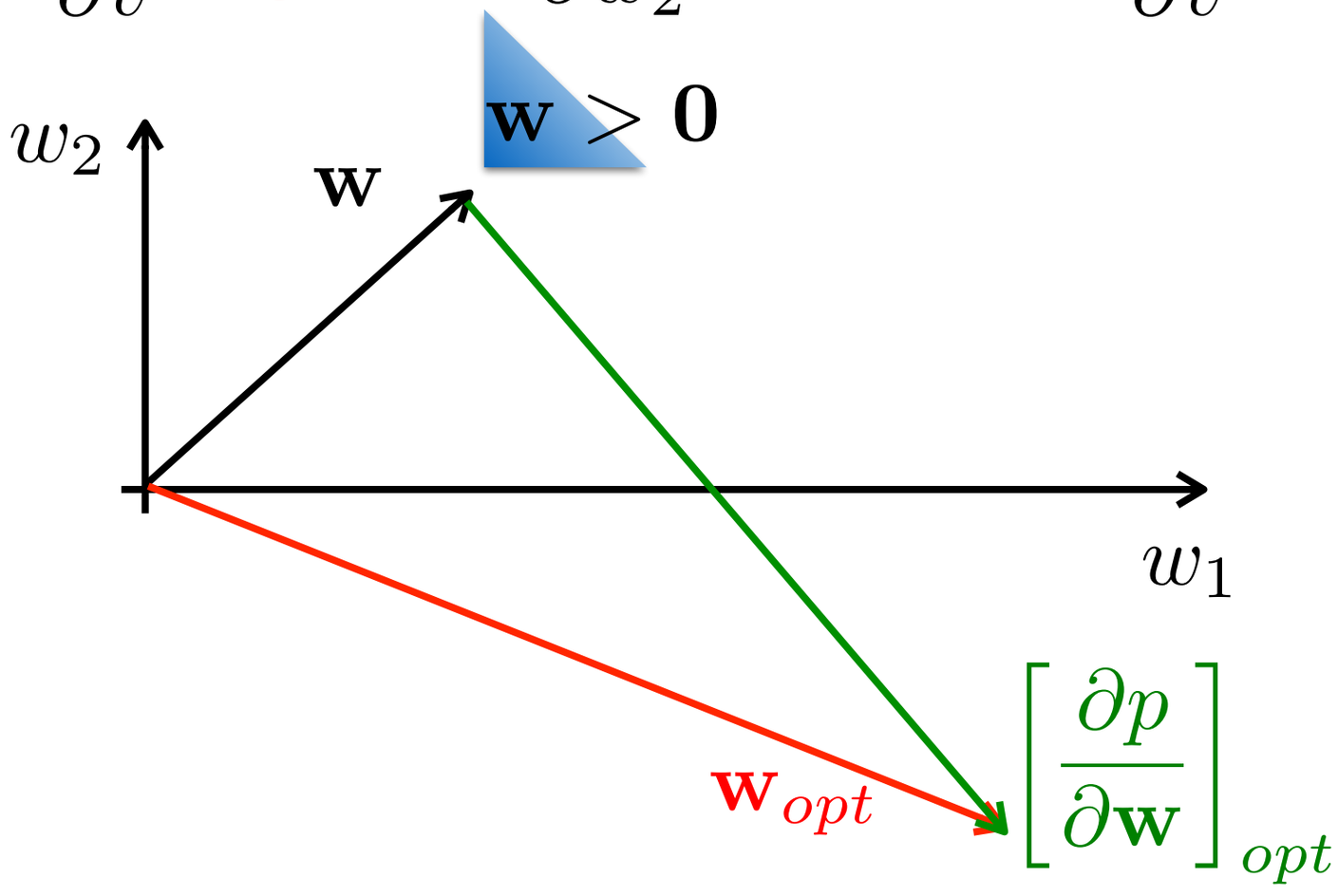
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

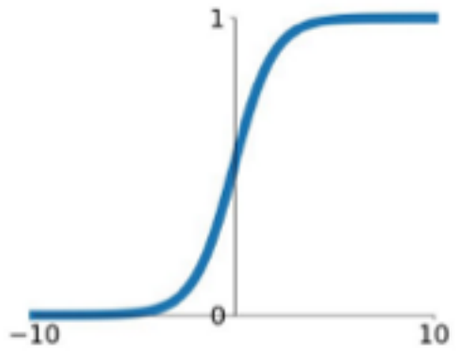
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

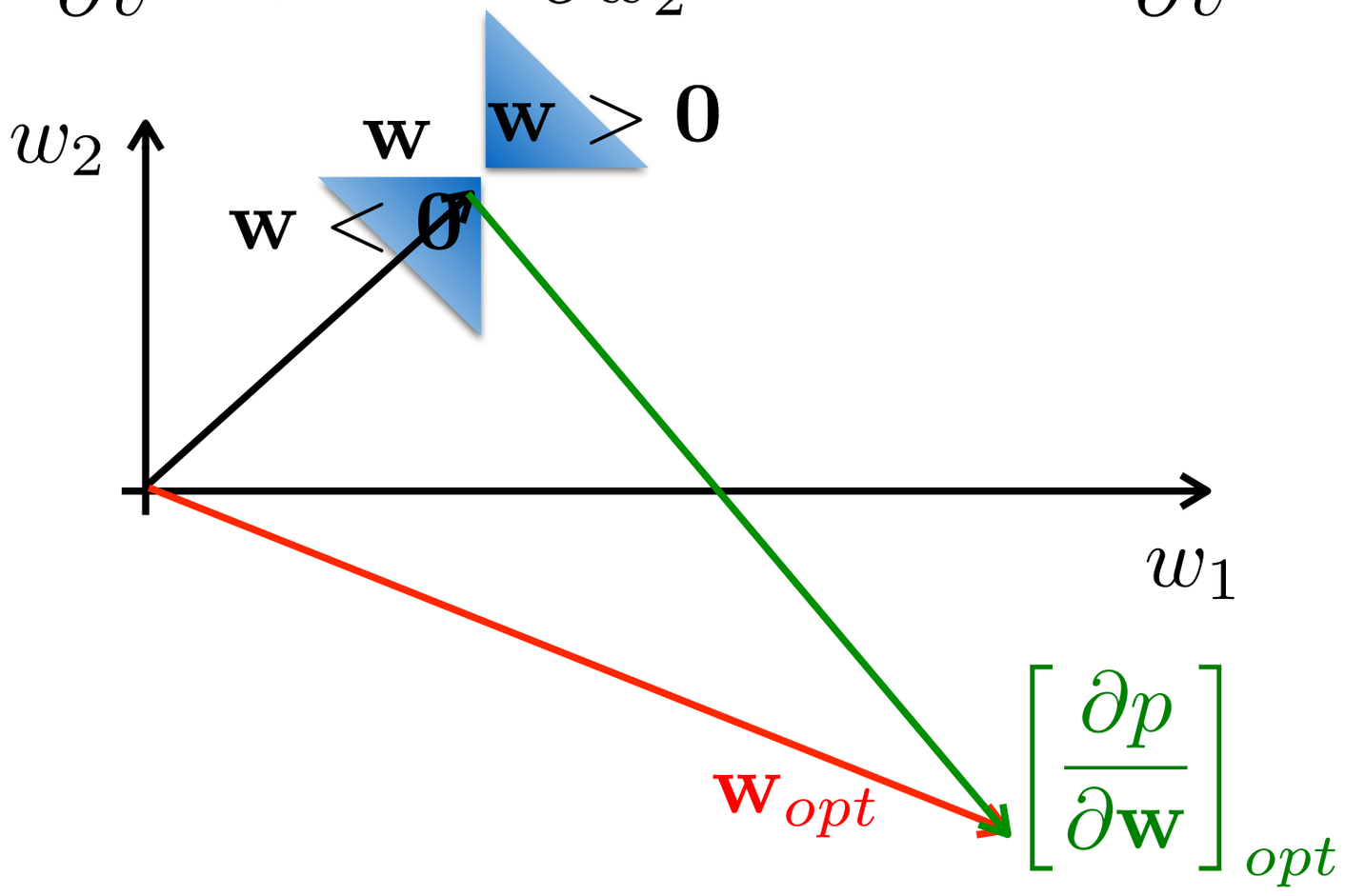
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

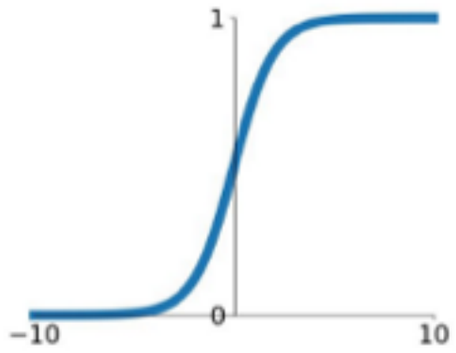
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

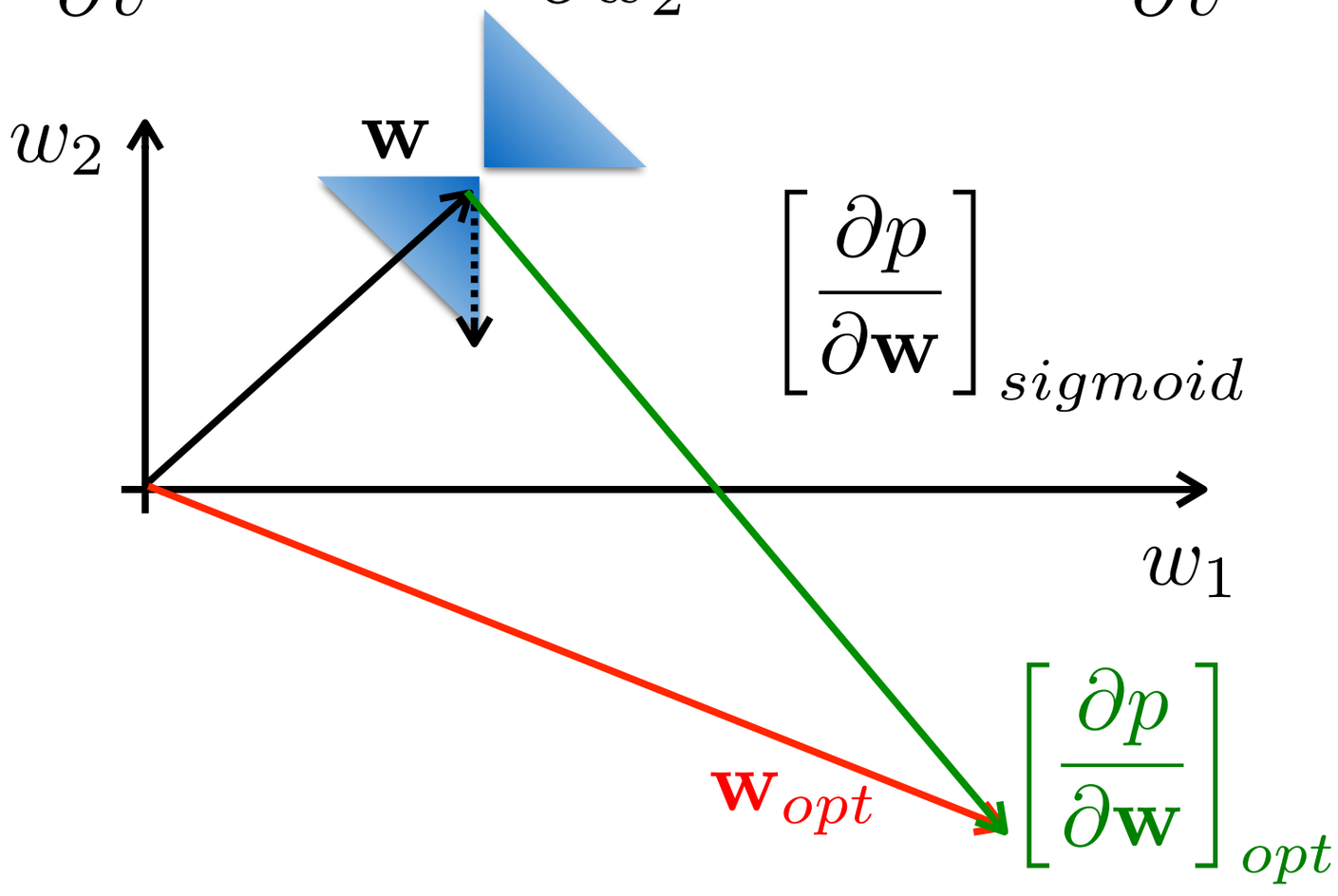
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

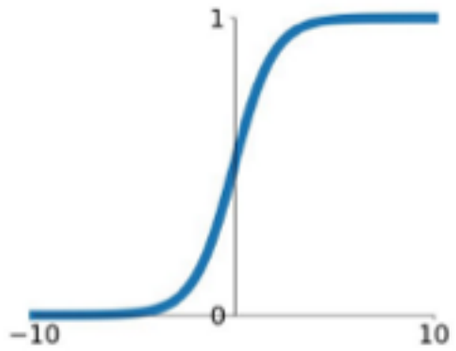
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

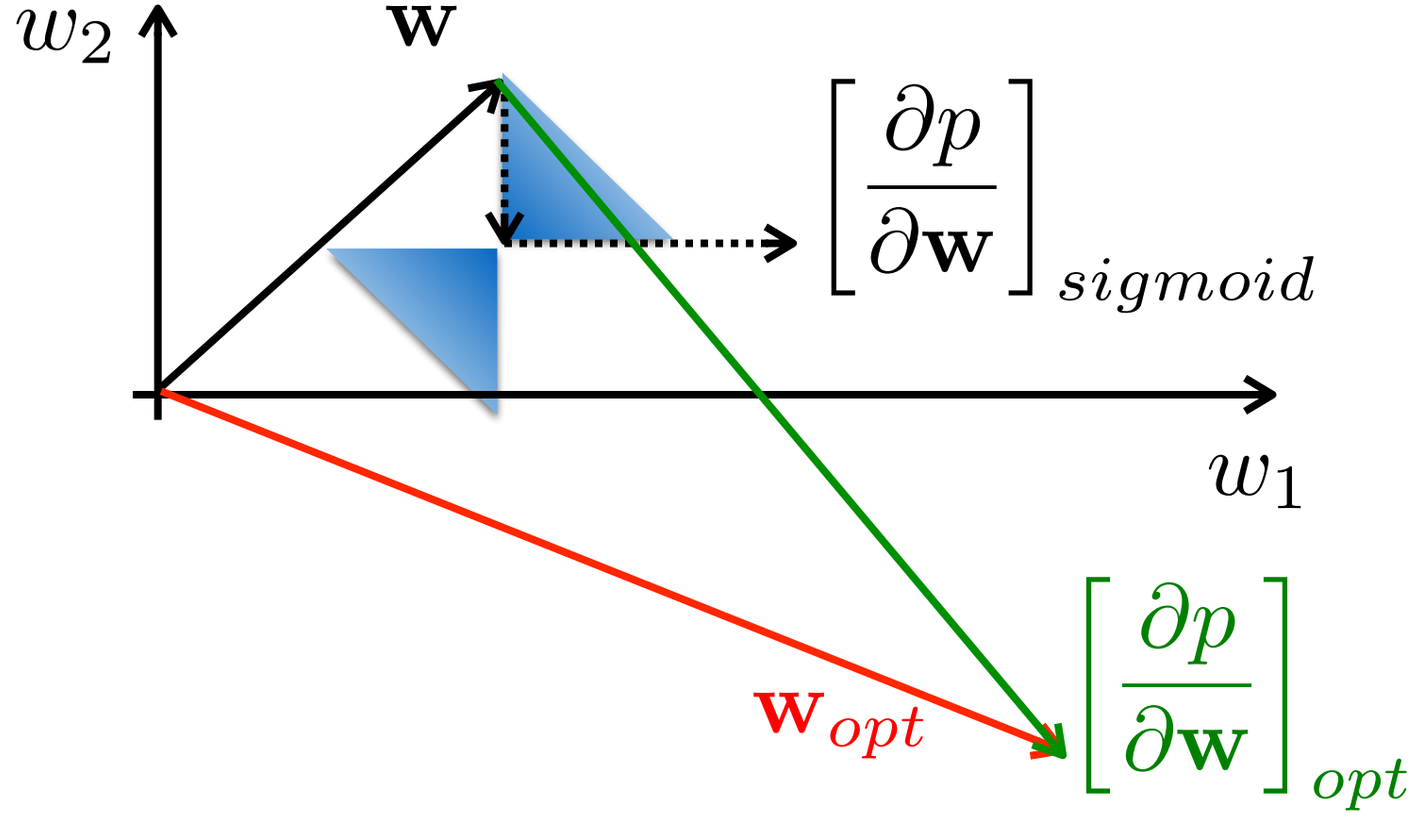
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

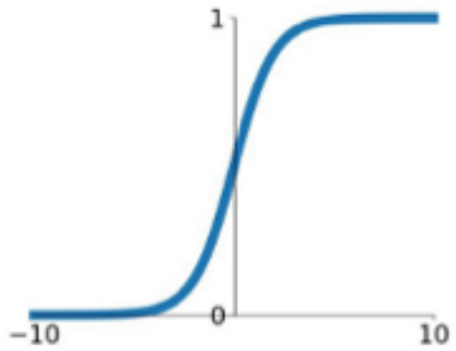
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

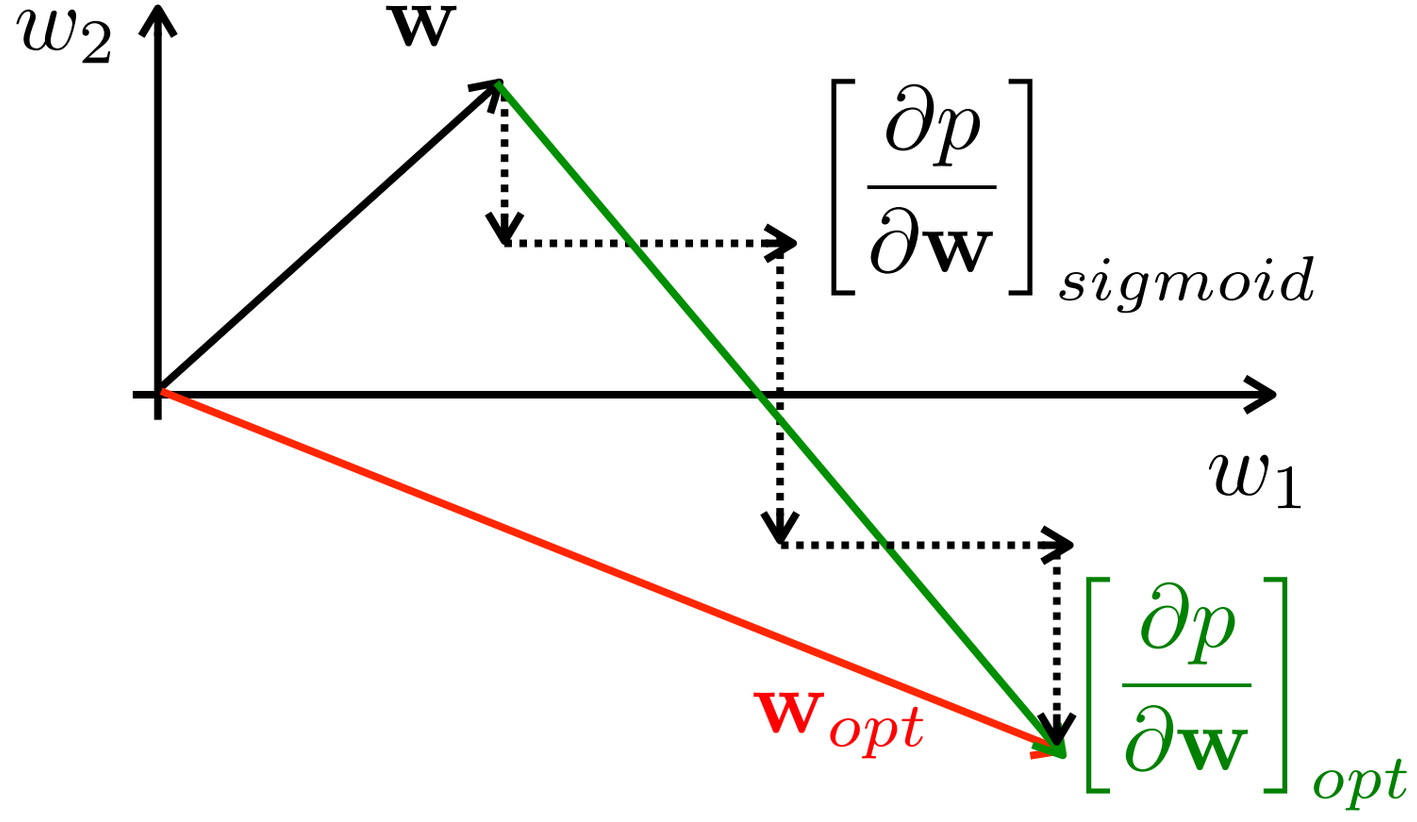
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- what happen to backprop gradient?

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

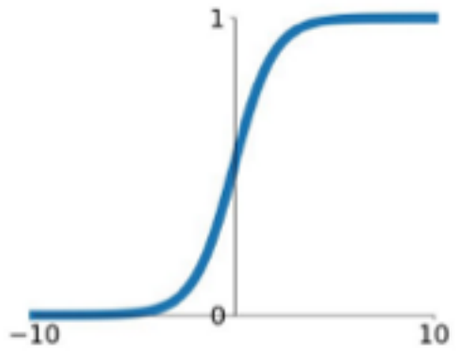
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

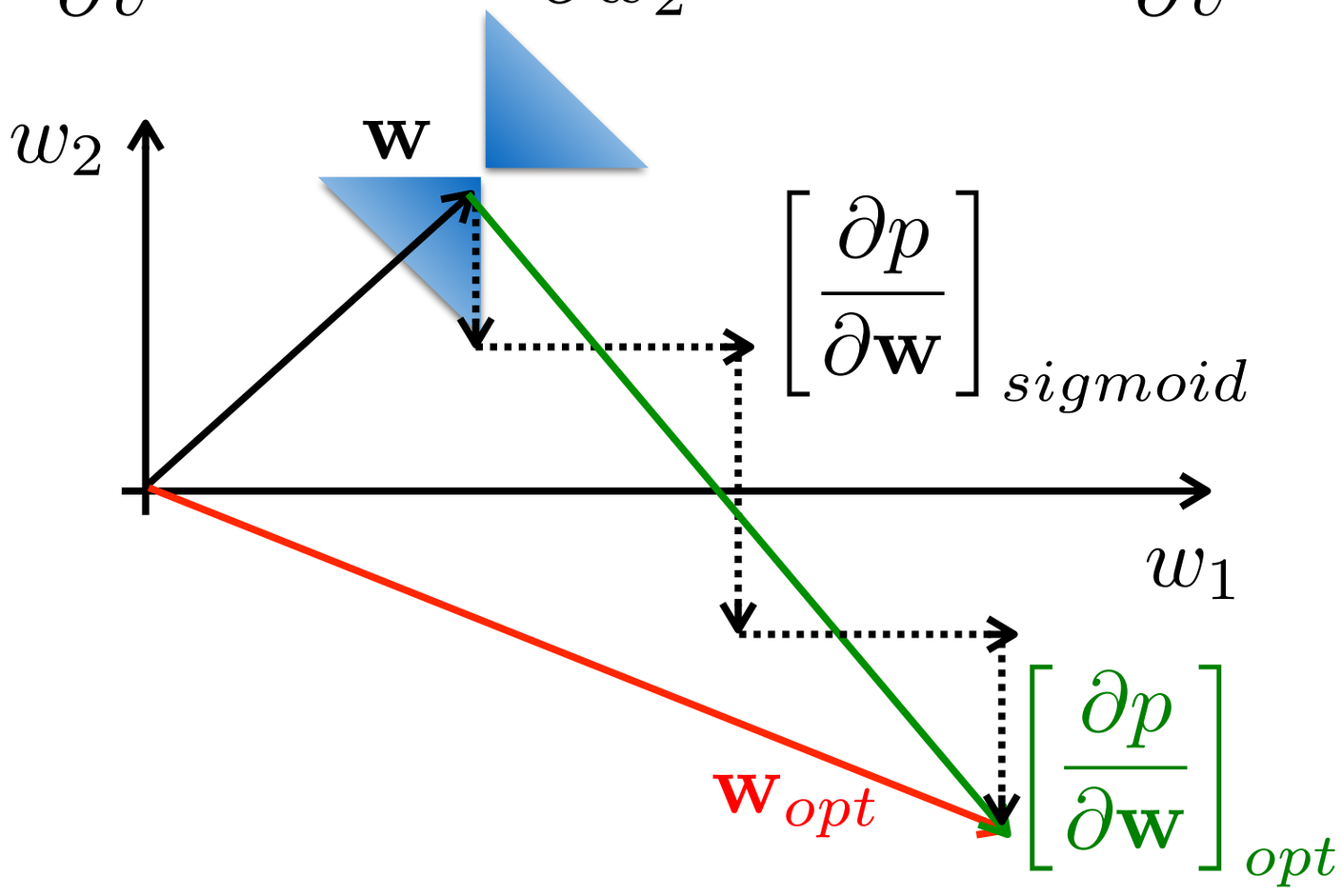
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- undesired zig-zag behaviour

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

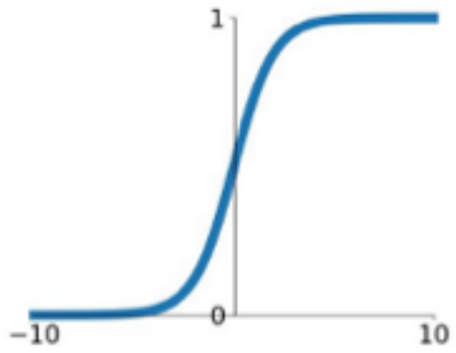
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



Learning

Sigmoid

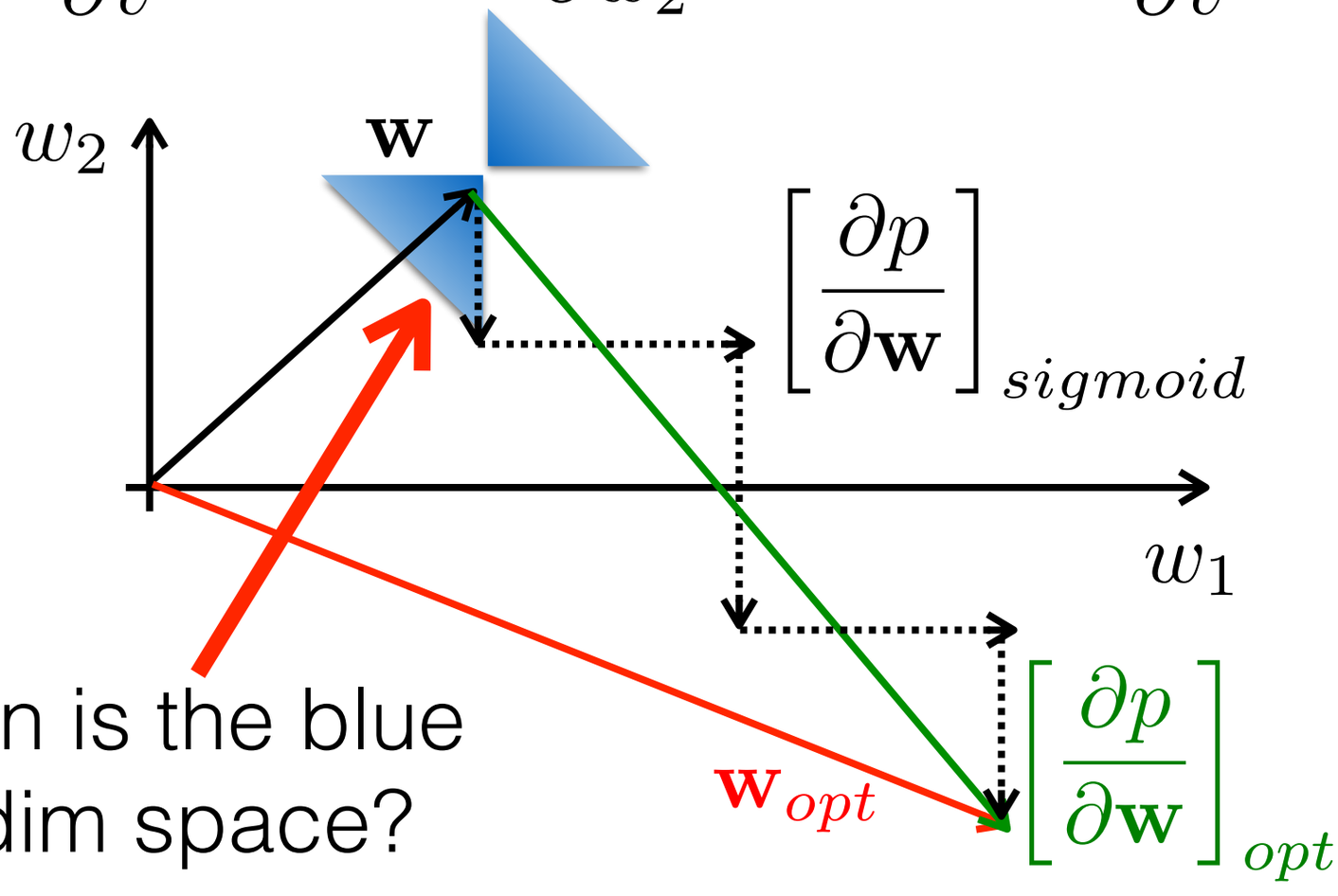
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- undesired zig-zag behaviour

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

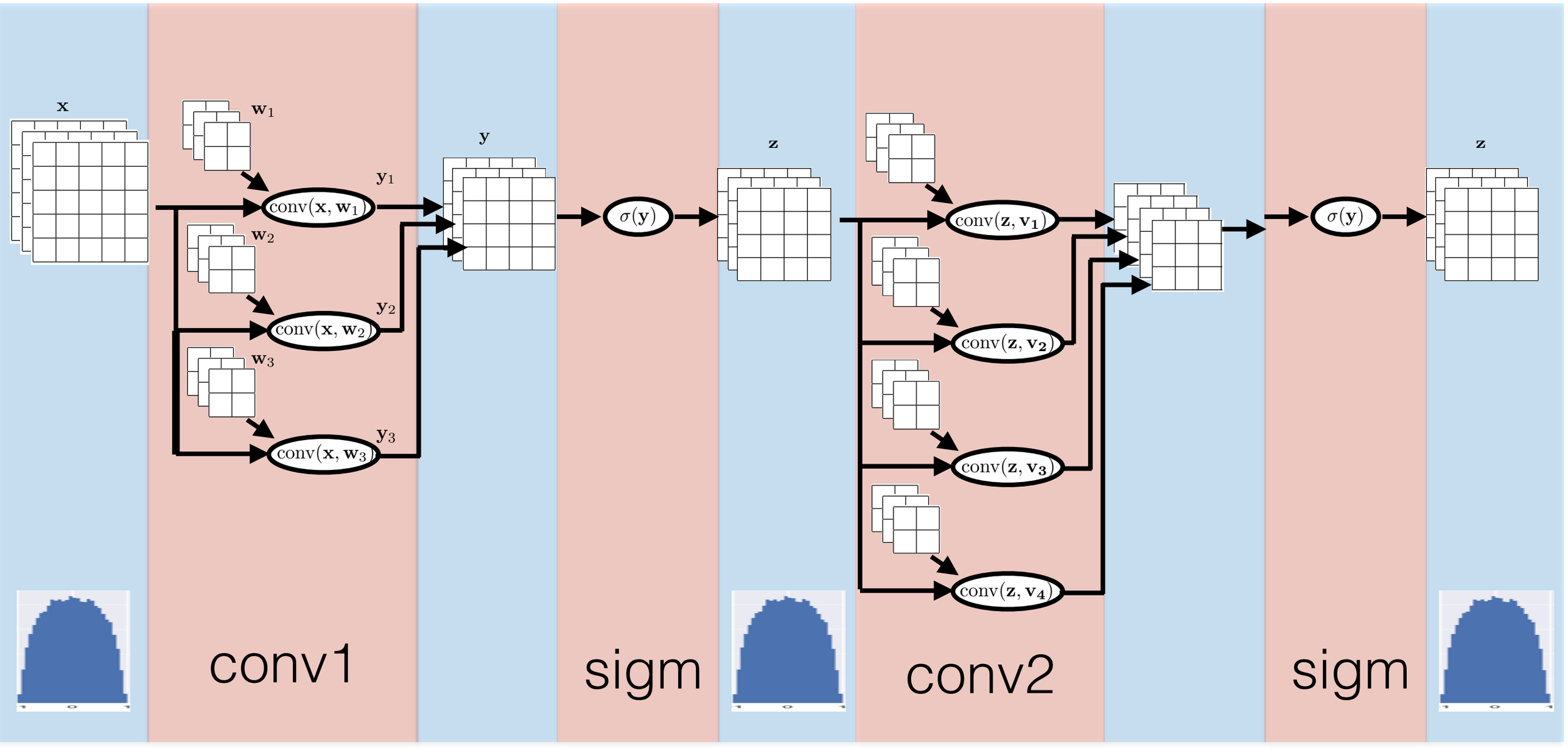
$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$



how big fraction is the blue region in 10-dim space?

Learning - summary

- you want to keep reasonable values during training !!!!



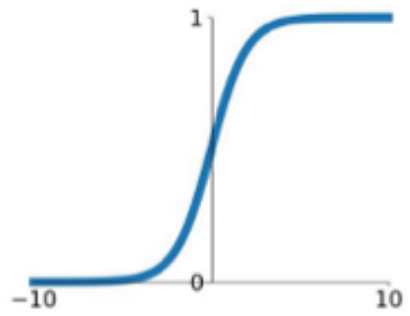
Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,
  - regularizations

Activation functions

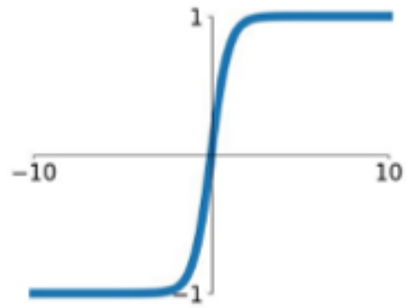
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



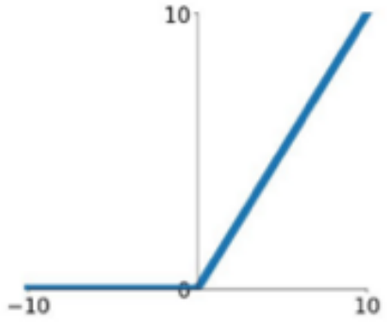
tanh

$$\tanh(x)$$



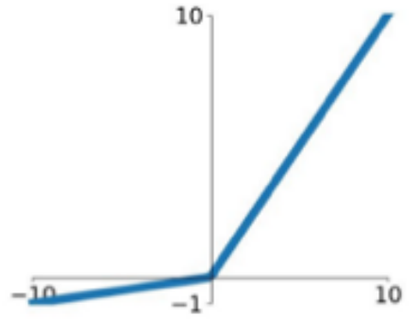
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

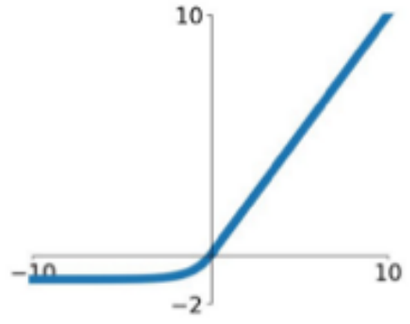


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

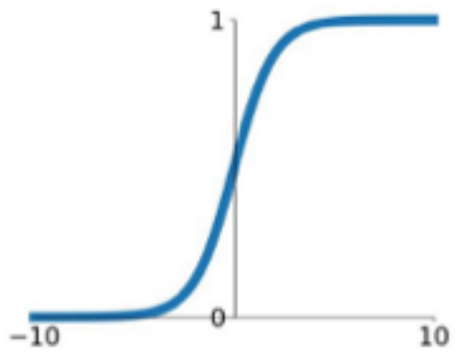
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- what happen to backprop gradient when weights are **huge**?

Sigmoid

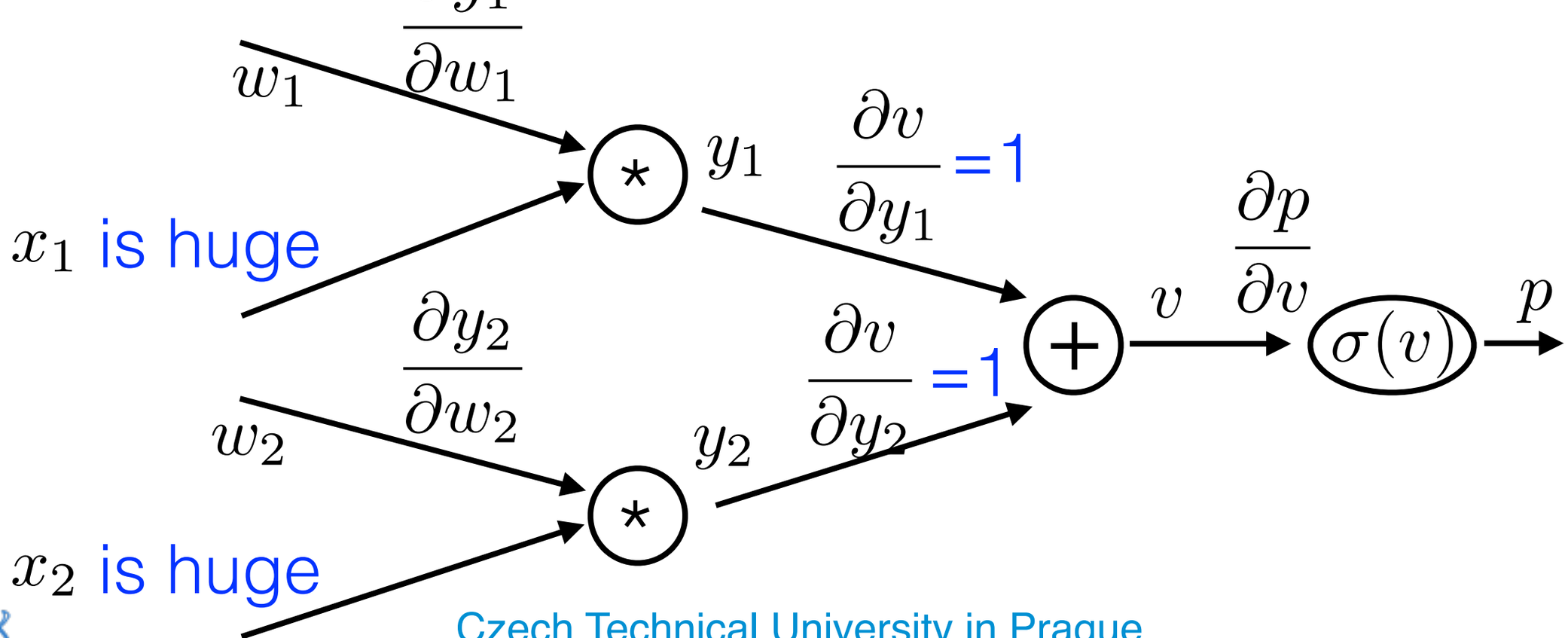
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- not zero-centered (pos. output)
- computationally expensive

$$\frac{\partial p}{\partial w_1} = \frac{\partial y_1}{\partial w_1}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} = 0$$

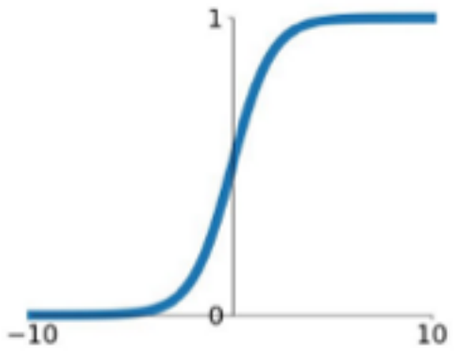
$$\frac{\partial p}{\partial w_2} = \frac{\partial y_2}{\partial w_2}\frac{\partial v}{\partial y_1}\frac{\partial p}{\partial v} = 0$$



Activation functions

Sigmoid

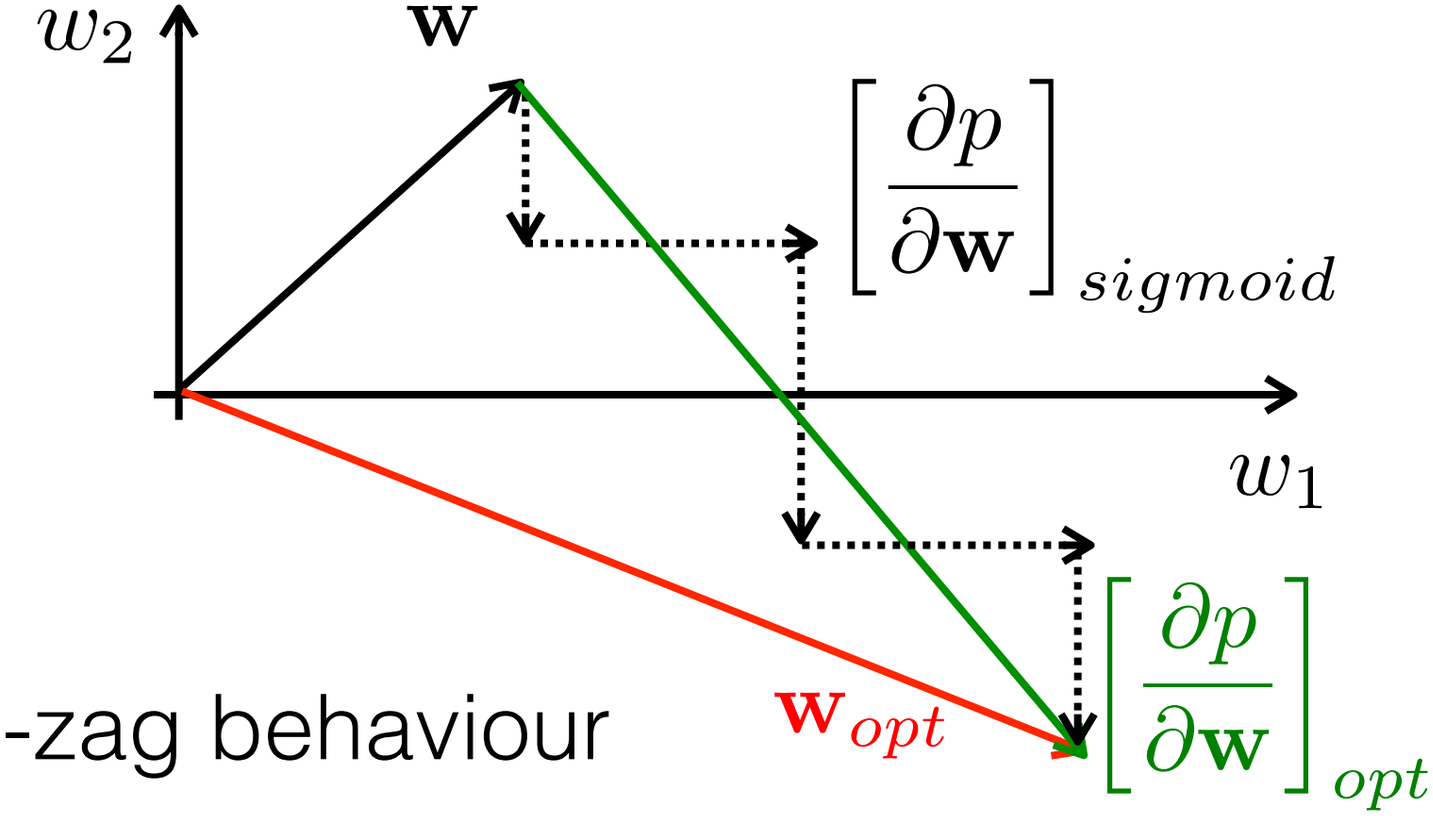
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- zero gradient when saturated
- not zero-centered (pos. output)
- computationally expensive

$$\frac{\partial p}{\partial w_1} = x_1 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

$$\frac{\partial p}{\partial w_2} = x_2 \cdot 1 \cdot \frac{\partial p}{\partial v} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

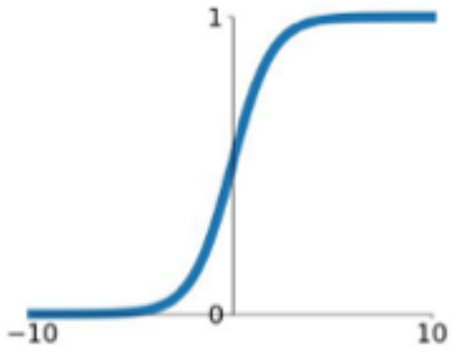


Undesired zig-zag behaviour

Activation functions

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

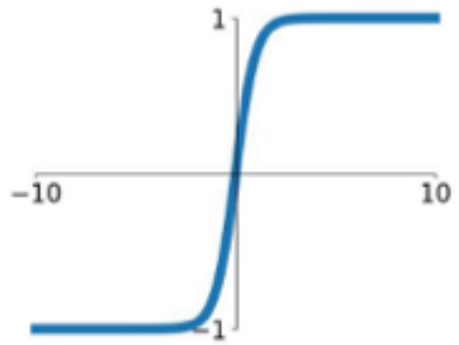


- zero gradient when saturated
- not zero-centered (pos. output)
- computationally expensive

Activation functions

tanh

$\tanh(x)$

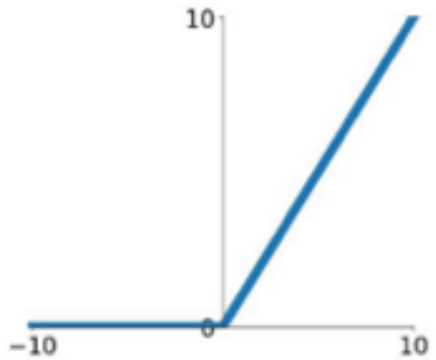


- zero gradient when saturated
- ~~not zero-centered (only positive ouputs)~~
- computationally expensive

Activation functions

ReLU

$$\max(0, x)$$

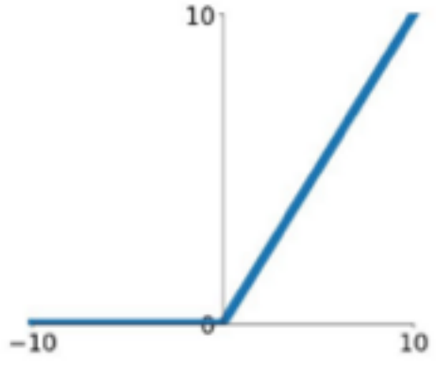


- ~~zero gradient when saturated~~ *(partially => dead ReLU!)*
- not zero-centered (only positive ouputs)
- ~~computationally expensive~~

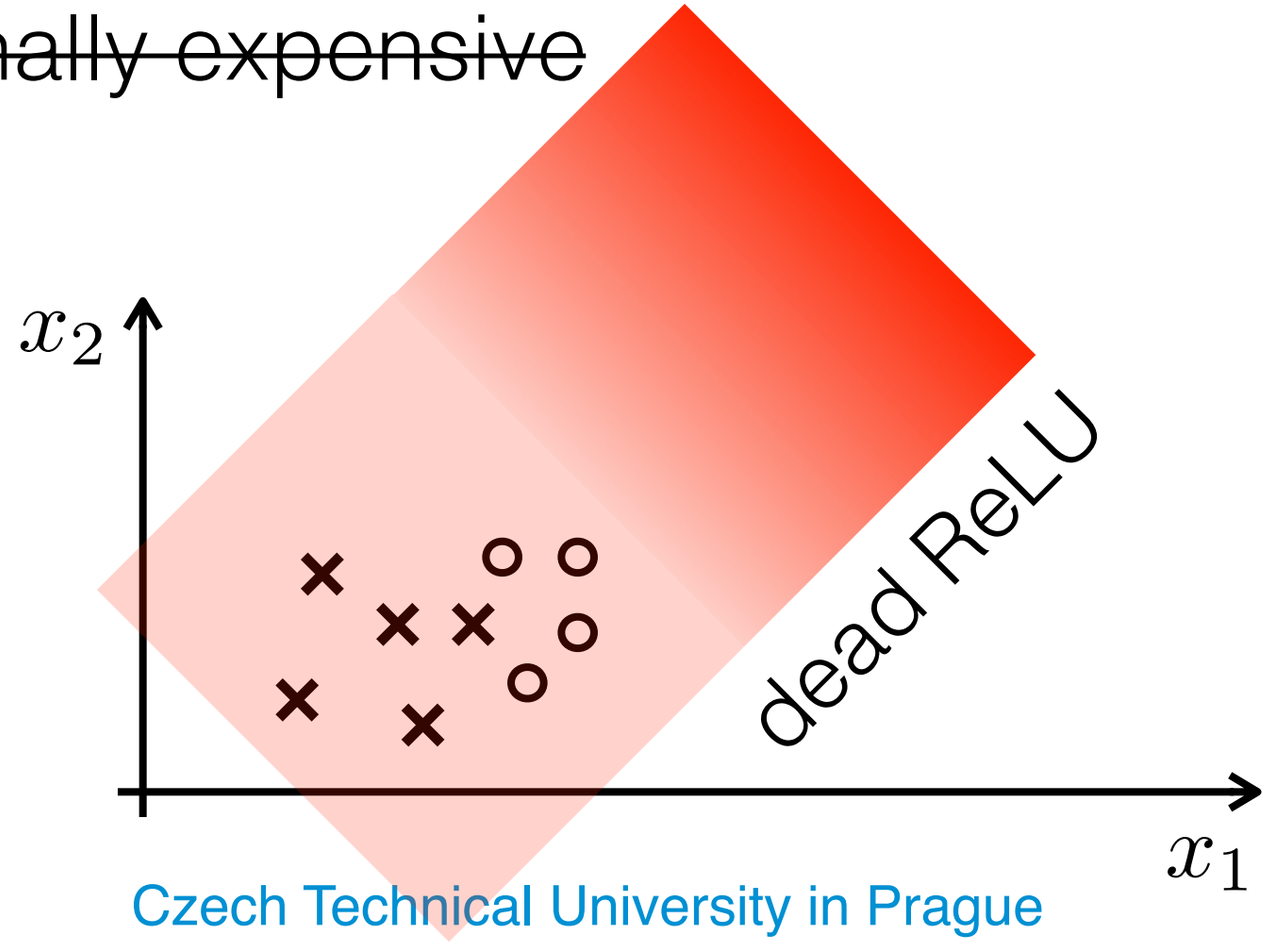
Activation functions

ReLU

$\max(0, x)$



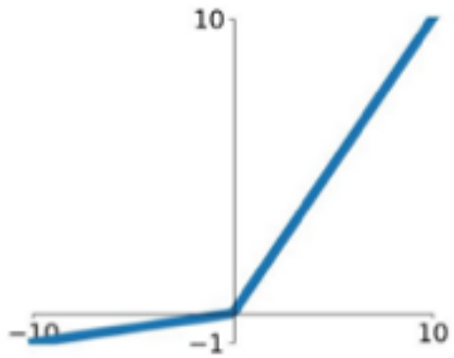
- ~~zero gradient when saturated~~ *(partially => dead ReLU!)*
- not zero-centered (only positive ouputs)
- ~~computationally expensive~~



Activation functions

Leaky ReLU

$$\max(0.1x, x)$$



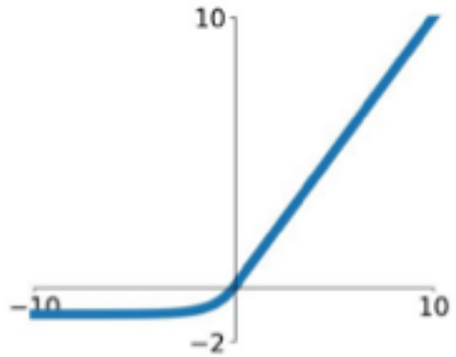
- ~~zero gradient when saturated~~
- ~~not zero-centered (only positive ouputs)~~
- ~~computationally expensive~~

Small gradient for negative values give tiny chance to recover

Activation functions

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



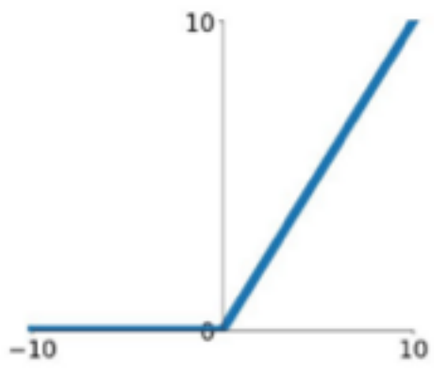
- ~~zero gradient when saturated~~ *(partially)*
- ~~not zero-centered (only positive ouputs)~~
- computationally expensive

Summary

- Use ReLU and avoid undesired properties by
  - good weight initialization
  - data preprocessing
  - batch normalization
- Still you want to keep “reasonable values”
  (i.e. small but not too much and distributed around zero)

ReLU

$$\max(0, x)$$



Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - initialization
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,
  - regularizations

Data preprocessing & initializations

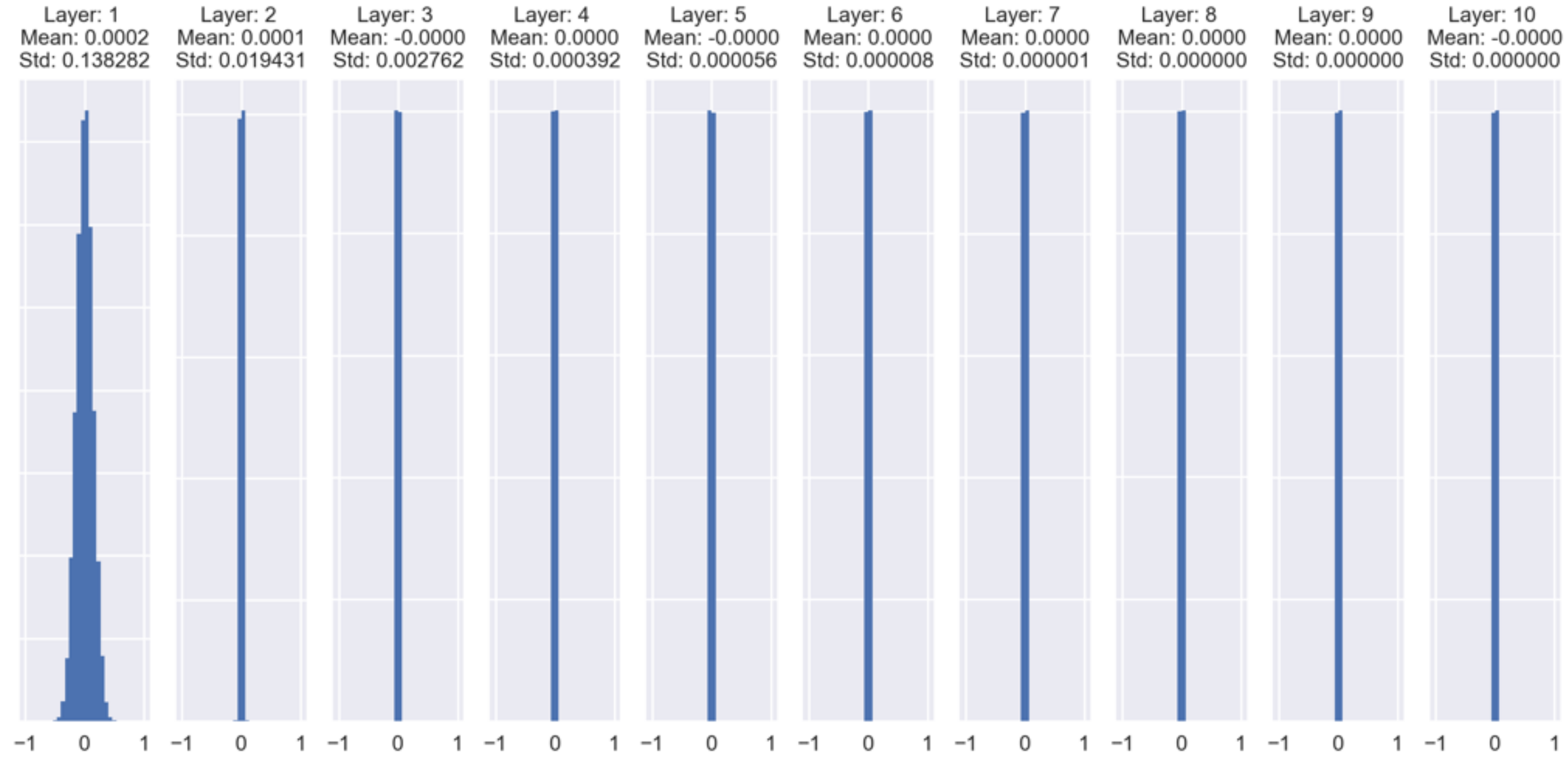
- Pixels values shifted zero mean to avoid only positive inputs and the unwanted “zig-zag” behaviour

Data preprocessing & initializations

- Pixels values shifted zero mean to avoid only positive inputs and the unwanted “zig-zag” behaviour
- Weight initialization:
  - $\mathbf{w} = 0$ all gradients the same
  - $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma)$ diminishing gradients in backprop
  - $\mathbf{w}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma * 1/N^{(i)})$ preserves variance of signal among layers (Xavier init [Glorot 2010])

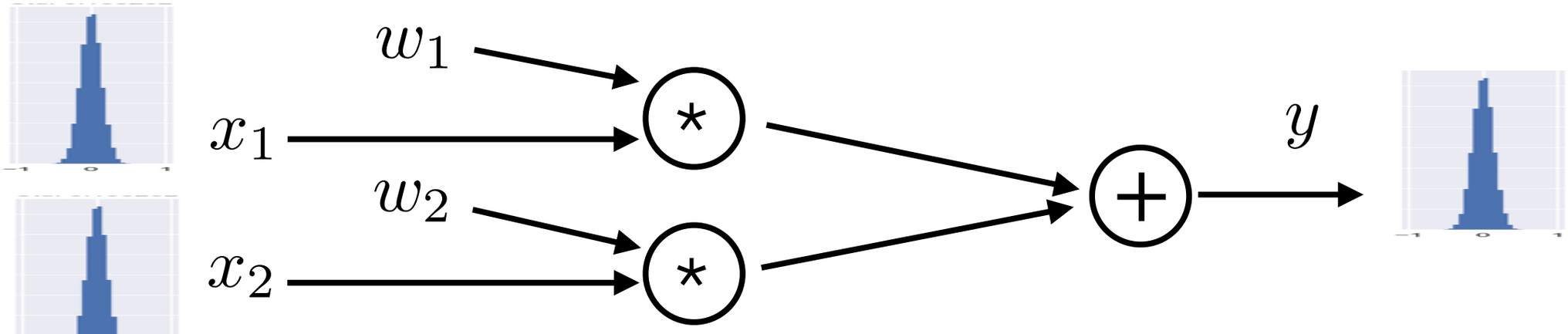
Xavier initialization [Glorot 2010]

Signal in randomly initialized weights $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma)$ forward (and backward) pass



Xavier initialization [Glorot 2010]

- We want to preserve variance of signal among layers (i.e. $\mathrm{var}(y) = \mathrm{var}(x_i)$)



$$\mathrm{var}(y) = \mathrm{var}(w_1 x_1 + w_2 x_2 + \cdots + w_N x_N) =$$

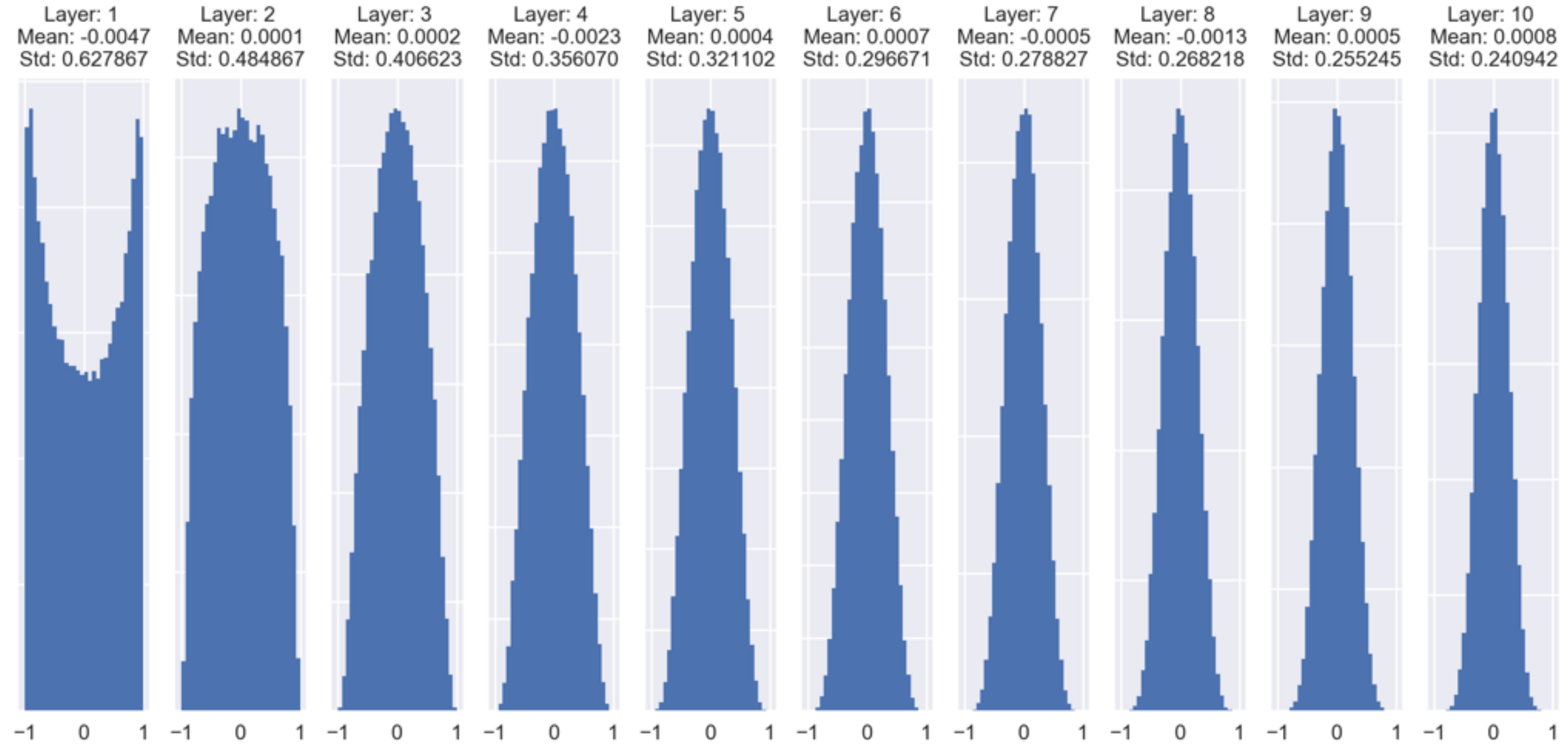
$$= \sum_{i=1}^{N} \mathbb{E}(x_i)^2 \mathrm{var}(w_i) + \mathbb{E}(w_i)^2 \mathrm{var}(x_i) + \mathrm{var}(w_i x_i) =$$

$$= \sum_{i=1}^{N} \mathrm{var}(w_i)\mathrm{var}(x_i) = N * \mathrm{var}(w_i)\mathrm{var}(x_i)$$

$$\Rightarrow \boxed{N * \mathrm{var}(w_i) = 1}$$

Xavier initialization [Glorot 2010]

Signal in Xavier initialized weights $\mathbf{w}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma * 1/N^{(i)})$ forward (and backward) pass (better but not ideal)



Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - initialization
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,
  - regularizations

Batch normalization layer [Ioffe and Szegedy 2015]
https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

- **Learning:**
- Normalize each dimension of input feature map in each layer
- Learn parameters $\gamma^{(k)}$, $\beta^{(k)}$ for each dimension k

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$x_1 \ldots x_m$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$y_1 \cdots y_m$

Batch normalization layer [Ioffe and Szegedy 2015]
https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

- **Learning:**
- Normalize each dimension of input feature map in each layer
- Learn parameters $\gamma^{(k)}$, $\beta^{(k)}$ for each dimension k

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$x_1 \ldots x_m$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$y_1 \ldots y_m$

Batch normalization layer [Ioffe and Szegedy 2015]
https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

- **Learning:**
- Normalize each dimension of input feature map in each layer
- Learn parameters $\gamma^{(k)}$, $\beta^{(k)}$ for each dimension k

$x_1 \dots x_m$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$y_1 \dots y_m$

Batch normalization layer [Ioffe and Szegedy 2015]
https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

- **Learning:**
- Normalize each dimension of input feature map in each layer
- Learn parameters $\gamma^{(k)}$, $\beta^{(k)}$ for each dimension k

$x_1 \dots x_m$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

$y_1 \dots y_m$

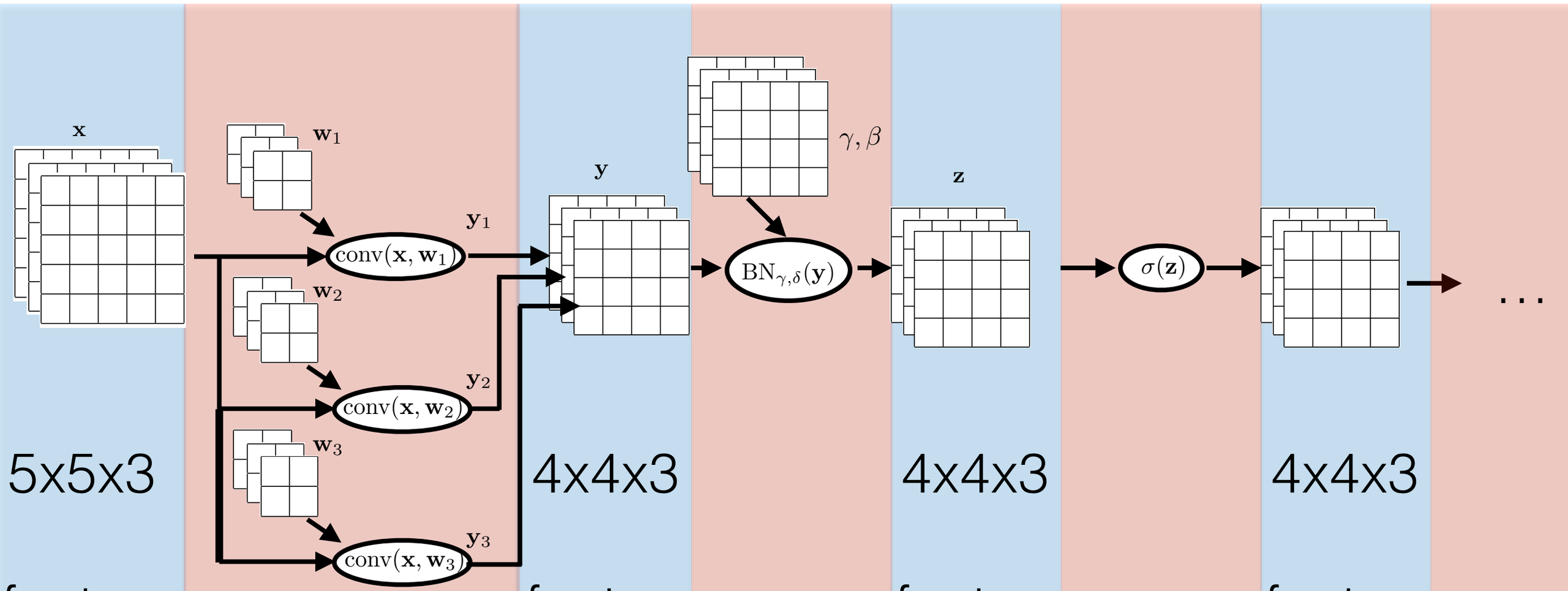
Batch normalization layer [Ioffe and Szegedy 2015] https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

- **Inference:** estimate $\mathbb{E}(x^{(k)})$ and $\mathrm{var}(x^{(k)})$
- Use learned parameters $\gamma^{(k)}$, $\beta^{(k)}$ and $\mathbb{E}(x^{(k)})$, $\mathrm{var}(x^{(k)})$

$\mathbf{x} = [x^{(1)} \ldots x^{(n)}]^\top$ $\mathbf{y} = [y^{(1)} \ldots y^{(n)}]^\top$

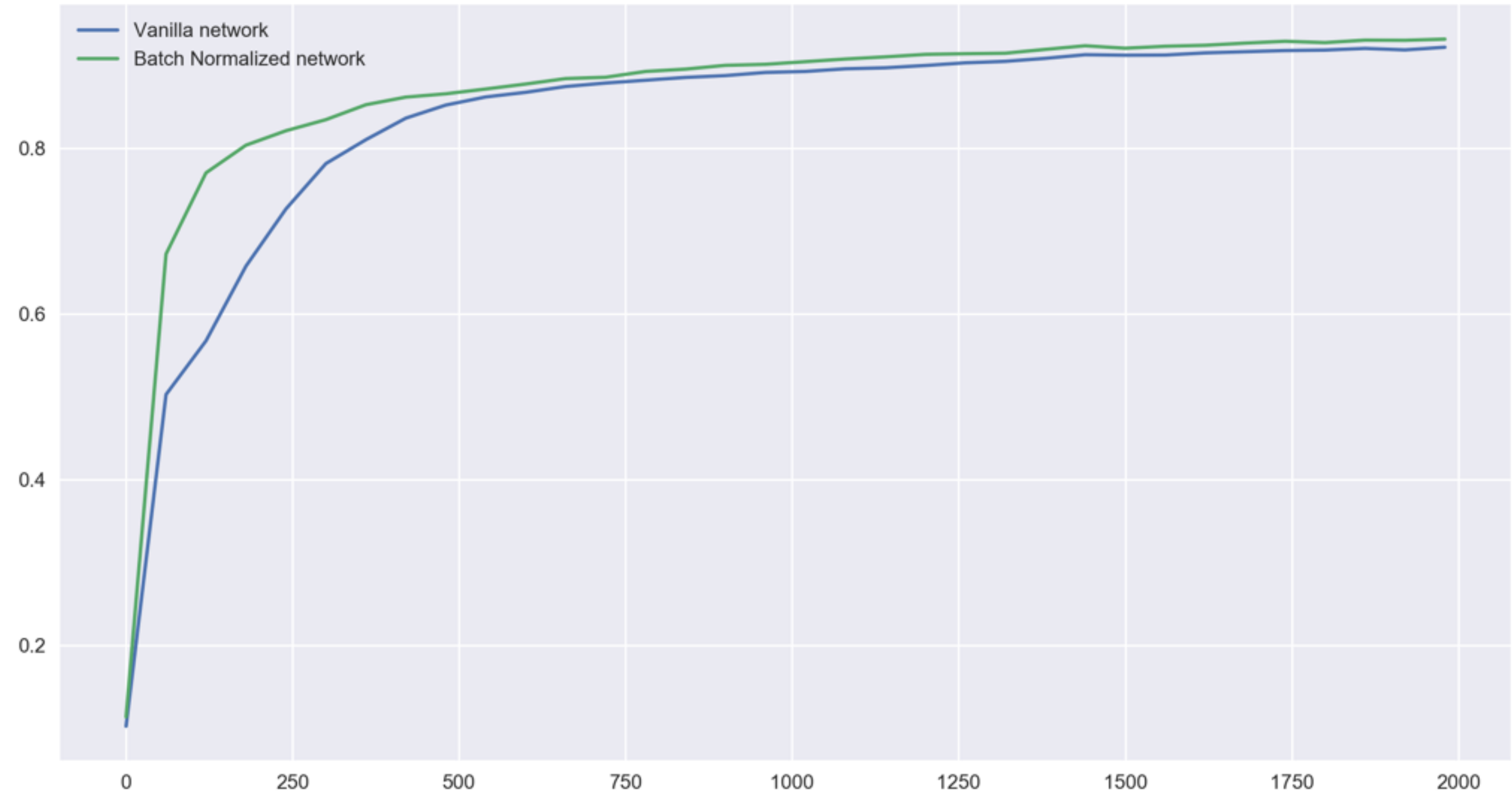
$$y^{(k)} = \gamma^{(k)} \left(\frac{x^{(k)} - \mathbb{E}(x^{(k)})}{\mathrm{var}(x^{(k)})} \right) + \beta^{(k)}$$

Batch normalization layer [Ioffe and Szegedy 2015] https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)



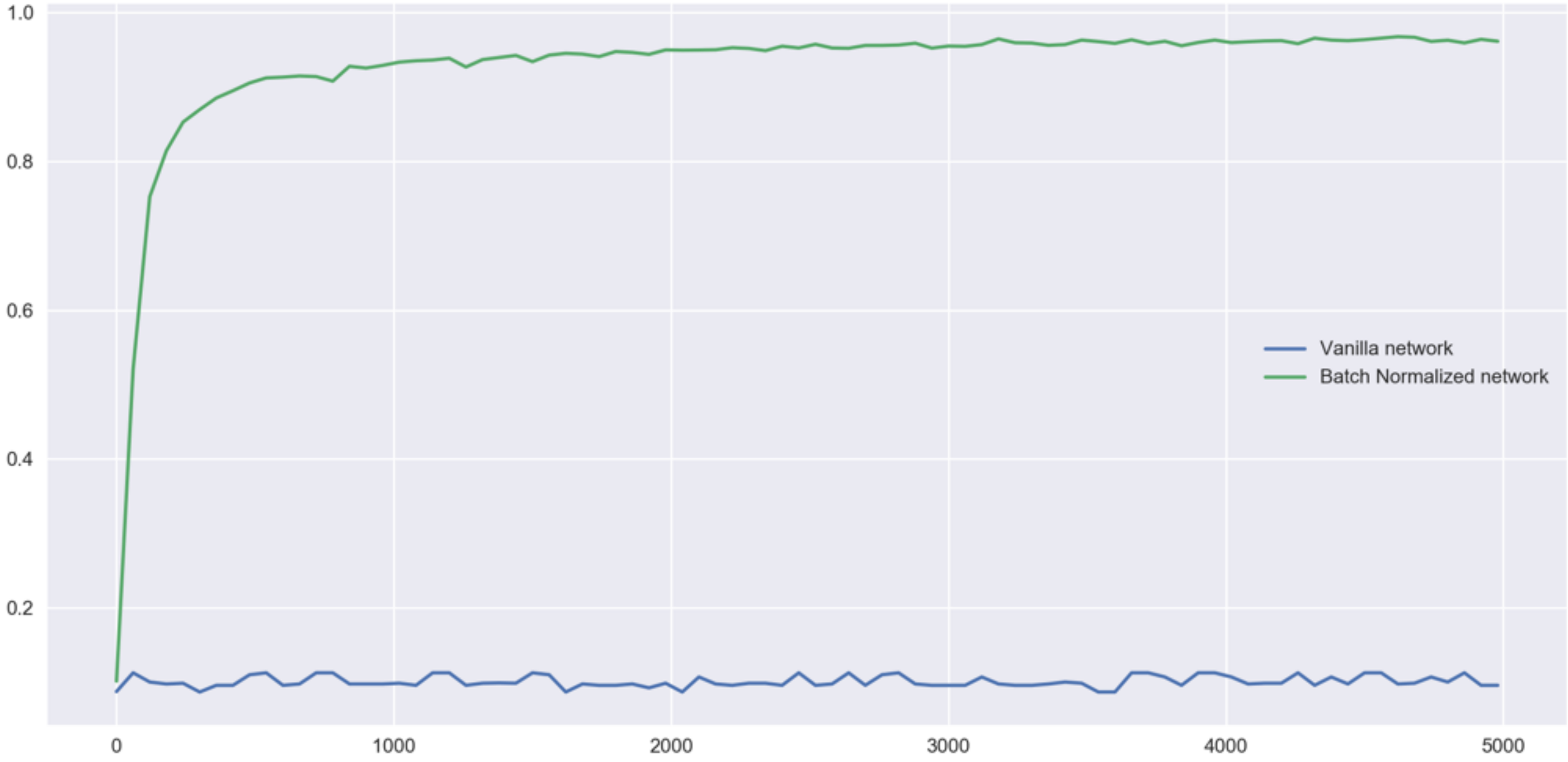
Batch normalization layer [Ioffe and Szegedy 2015] https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

Good weight initialization



Batch normalization layer [Ioffe and Szegedy 2015] https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

Bad weight initialization



Batch normalization layer [Ioffe and Szegedy 2015] https://arxiv.org/pdf/1502.03167.pdf (over 6k citation)

Summary

- Normalize each dimension of input feature map in each layer independently.
- Different behaviour for learning and inference
- BN yields
  - Reduced learning time
  - Model regularizer (one training example always normalized differently => small jittering of each sample)
  - Reduce dependency on good weight initialization

Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,
  - regularizations

Max-pooling

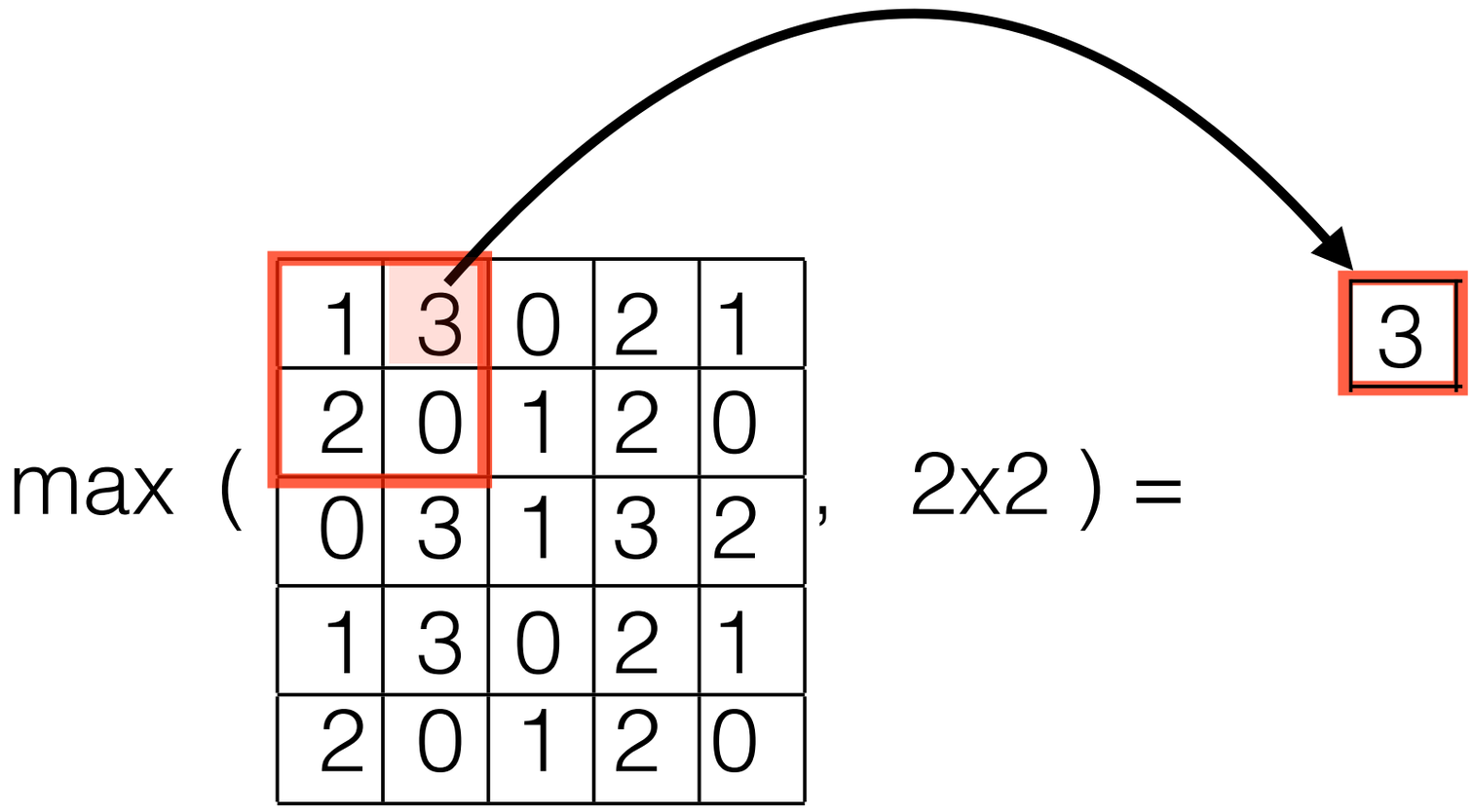


image
(5x5)

output
(**? x ?**)

Max-pooling

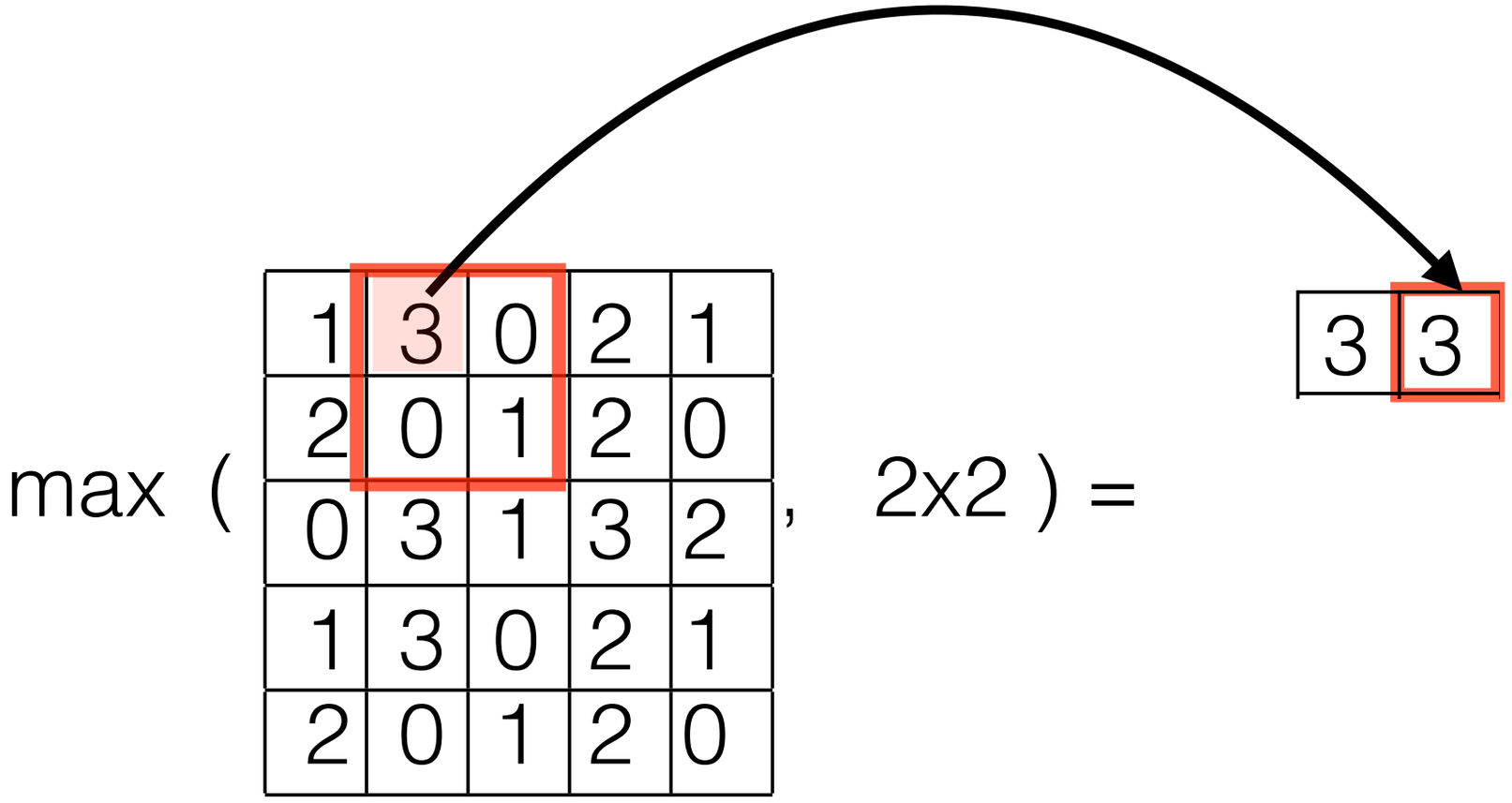


image
(5x5)

output
(**? x ?**)

Max-pooling

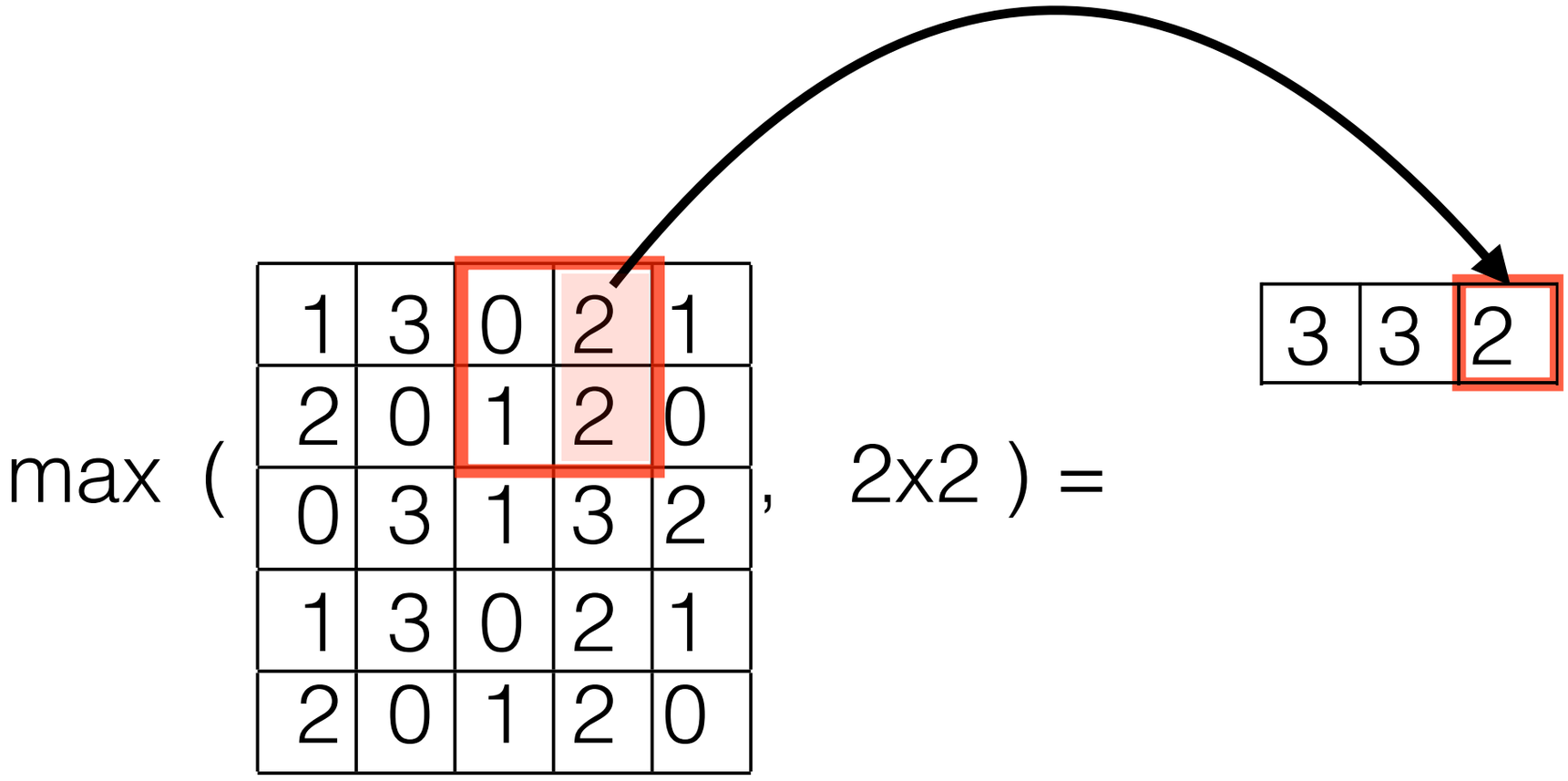


image
(5x5)

output
(**? x ?**)

Max-pooling

max ($\begin{bmatrix} 1 & 3 & 0 & 2 & 1 \\ 2 & 0 & 1 & 2 & 0 \\ 0 & 3 & 1 & 3 & 2 \\ 1 & 3 & 0 & 2 & 1 \\ 2 & 0 & 1 & 2 & 0 \end{bmatrix}$, 2x2) = $\begin{bmatrix} 3 & 3 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 2 & 2 \end{bmatrix}$

image
(5x5)

output
(**4 x 4**)

Max-pooling

$$M = (N+2*pad-K) / stride + 1$$

The same as for convolution

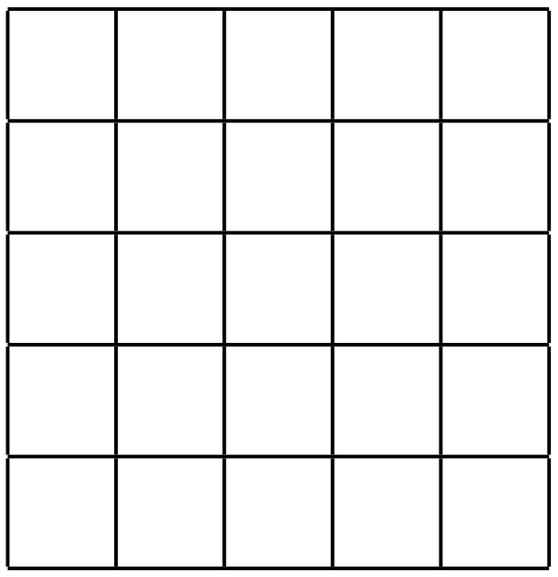
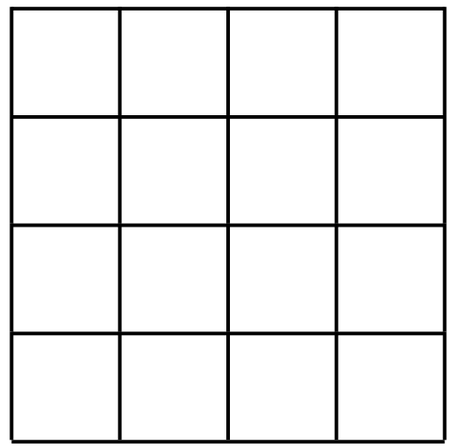
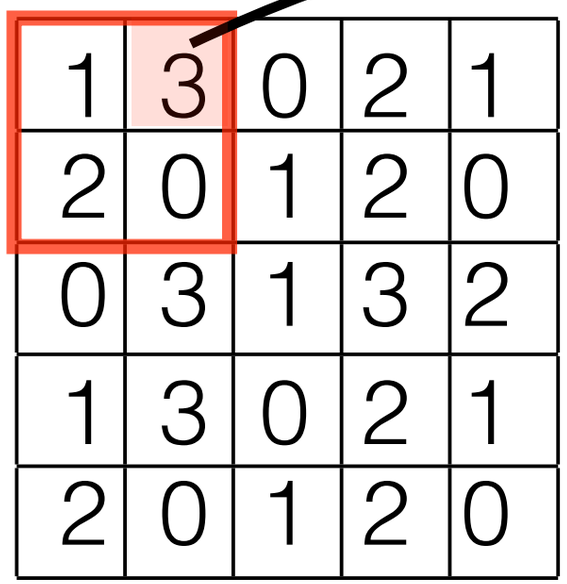
max ( , KxK) = 

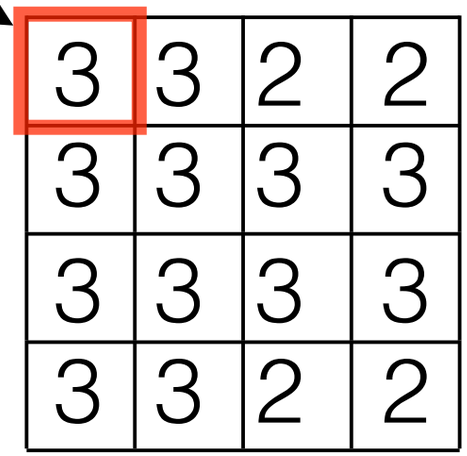
image
(NxN)

output
(M x M)

Max-pooling feed-forward

max ( , 2x2) =

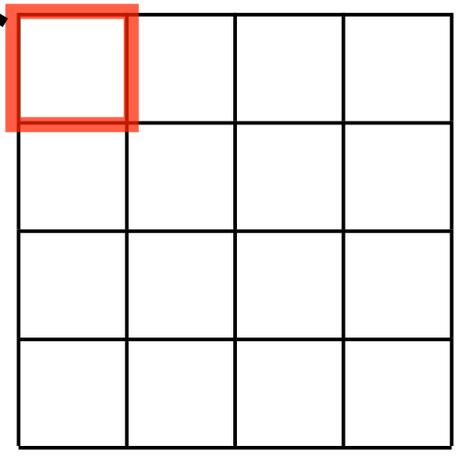
1	3	0	2	1
2	0	1	2	0
0	3	1	3	2
1	3	0	2	1
2	0	1	2	0

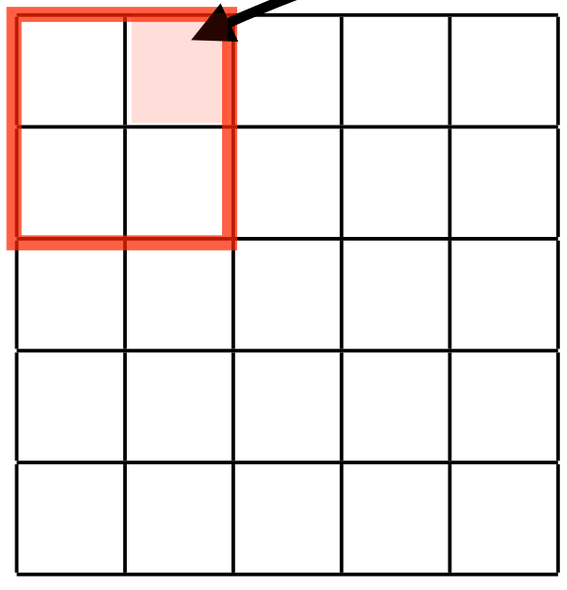


3	3	2	2
3	3	3	3
3	3	3	3
3	3	2	2

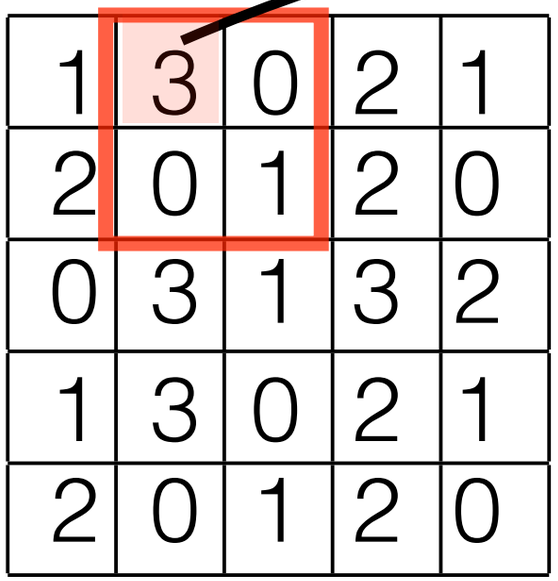
Max-pooling Backprop

upstream gradient

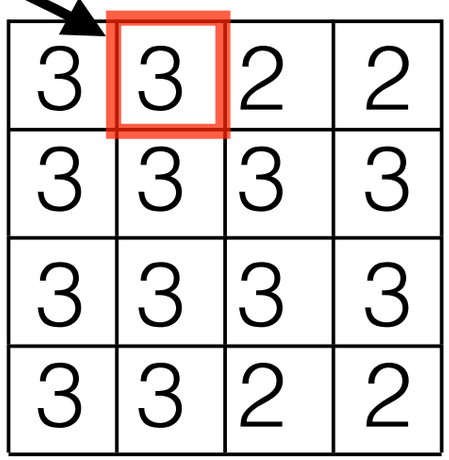


max ( , 2x2) =

Max-pooling feed-forward

max ( , 2x2) =

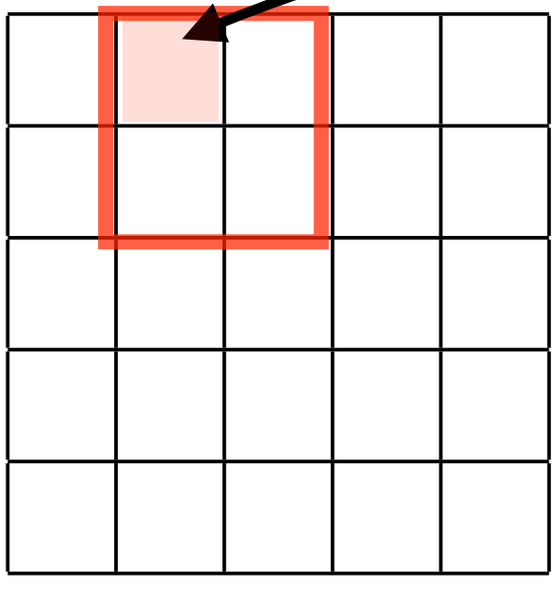
1	3	0	2	1
2	0	1	2	0
0	3	1	3	2
1	3	0	2	1
2	0	1	2	0

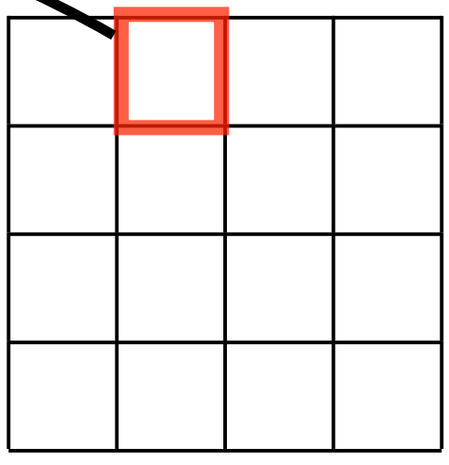


3	3	2	2
3	3	3	3
3	3	3	3
3	3	2	2

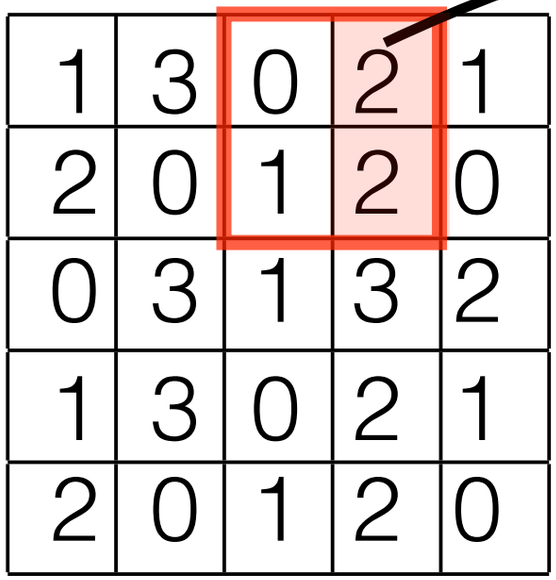
Max-pooling Backprop

upstream gradient

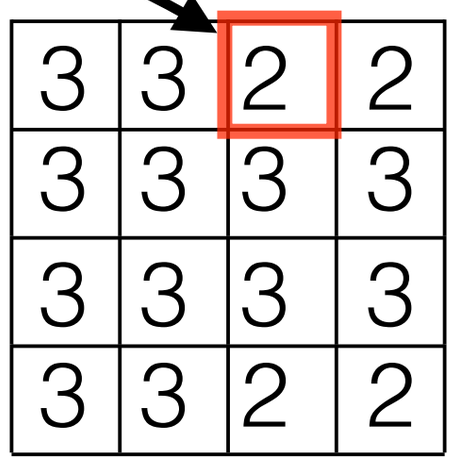
max ( , 2x2) =



Max-pooling feed-forward

max ( , 2x2) =

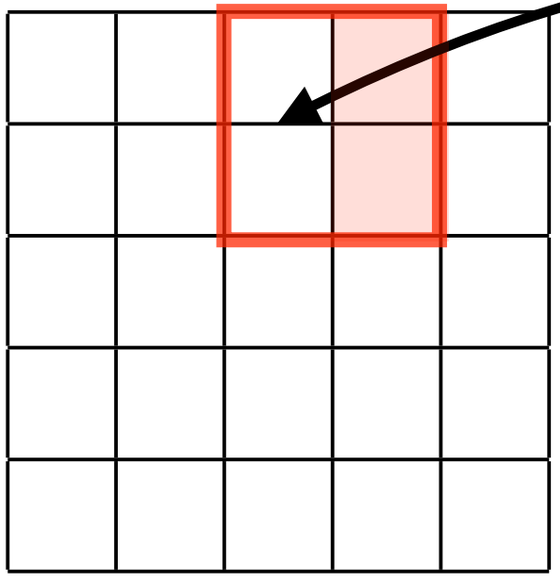
1	3	0	2	1
2	0	1	2	0
0	3	1	3	2
1	3	0	2	1
2	0	1	2	0

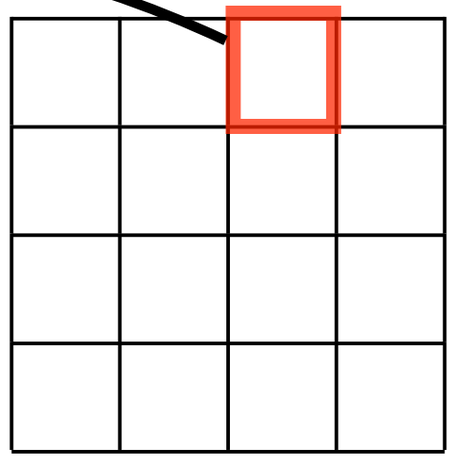


3	3	2	2
3	3	3	3
3	3	3	3
3	3	2	2

Max-pooling Backprop

upstream gradient

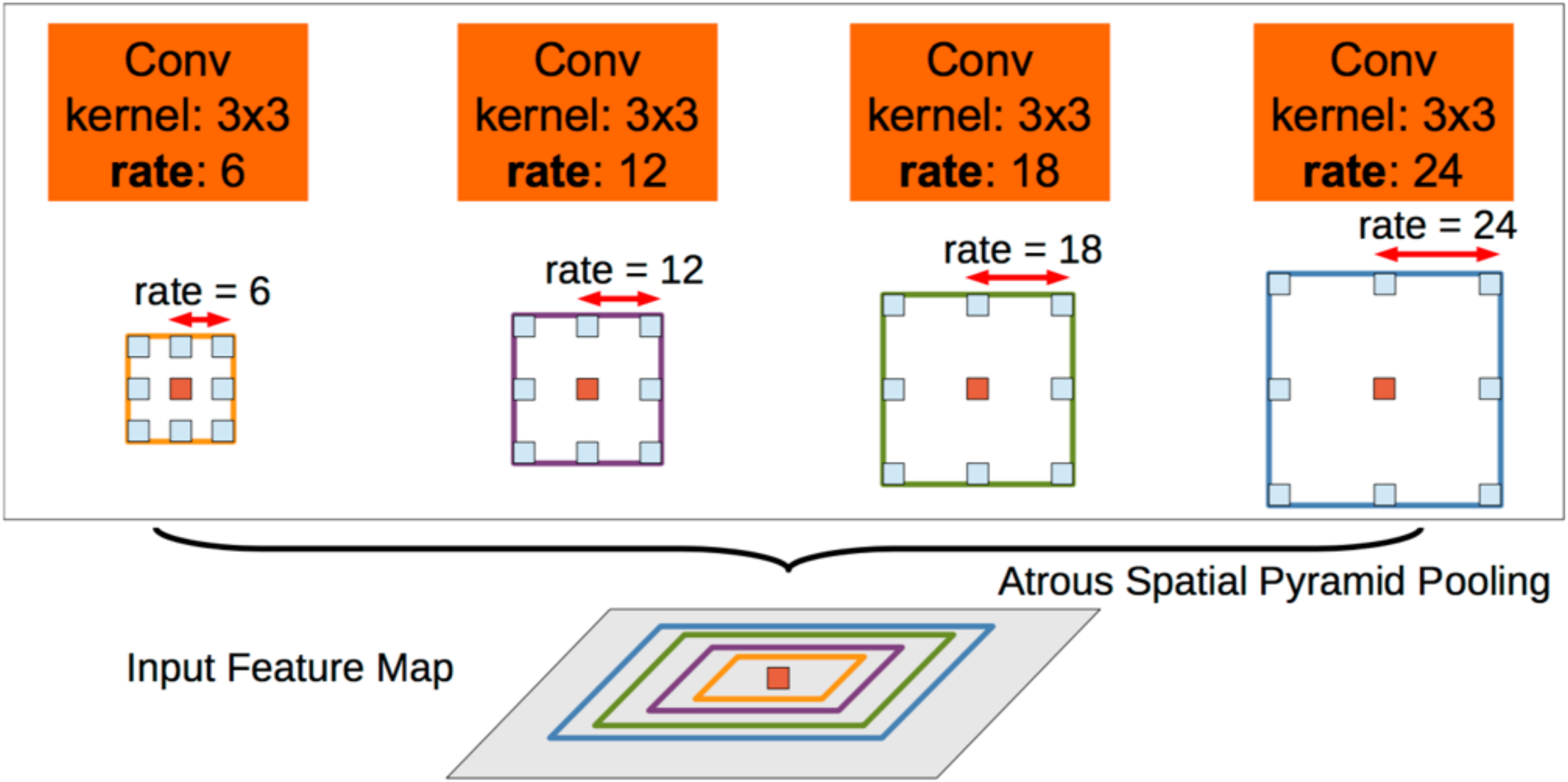
max ( , 2x2) =



Max-pooling

- Forward pass
  - similar to convolution but takes maximum over kernel
  - it has no parameters to be learnt!
- Backprop
  - propagate gradient only to active connections
- Main purpose is to reduce dimensionality and overfitting
- It seems that max pooling layers will disappear in future
  - should be avoided in generative models (GAN, VAE)
  - they can be replaced by conv-layers with larger stride in discriminative models https://arxiv.org/abs/1412.6806
  - Geoffrey Hinton: "*The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster*." (Reddit AMA)

Atrous Spatial Pyramid Pooling (ASPP)



[Chen et al. TPAMI 2018] https://arxiv.org/pdf/1606.00915.pdf

Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- regularizations
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,

Loss functions

- Regression:
  - L2 loss
  - L1 loss
- Classification:
  - cross entropy loss (N-output classifier $\mathbf{f}(\mathbf{x}, \mathbf{w})$)
  - logistic loss (single output dichotomy classifier $f(\mathbf{x}, \mathbf{w})$)

$$L_2(\mathbf{w}) = \sum_i \|\mathbf{f}(\mathbf{x}_i, \mathbf{w}) - \mathbf{y}_i\|_2^2$$

$$L_1(\mathbf{w}) = \sum_i |\mathbf{f}(\mathbf{x}_i, \mathbf{w}) - \mathbf{y}_i|$$

Loss functions

- Regression:
  - L2 loss
  - L1 loss
- Classification:
  - cross entropy loss (N-output classifier $\mathbf{f}(\mathbf{x}, \mathbf{w})$)
  - logistic loss (single output dichotomy classifier $f(\mathbf{x}, \mathbf{w})$)

(1) convert output to probability (softmax function)

$$\mathbf{s}(\mathbf{f}(\mathbf{x}, \mathbf{w})) = \begin{bmatrix} \exp(f_1(\mathbf{x}, \mathbf{w})) \\ \exp(f_2(\mathbf{x}, \mathbf{w})) \\ \vdots \\ \exp(f_N(\mathbf{x}, \mathbf{w})) \end{bmatrix} / \sum_{k=1}^{N} \exp(f_k(\mathbf{x}, \mathbf{w}))$$

(2) compute cross entropy

$$H(\mathbf{w}) = \sum_i -\log \mathbf{s}_{y_i}(\mathbf{f}(\mathbf{x}_i, \mathbf{w}))$$

Loss functions

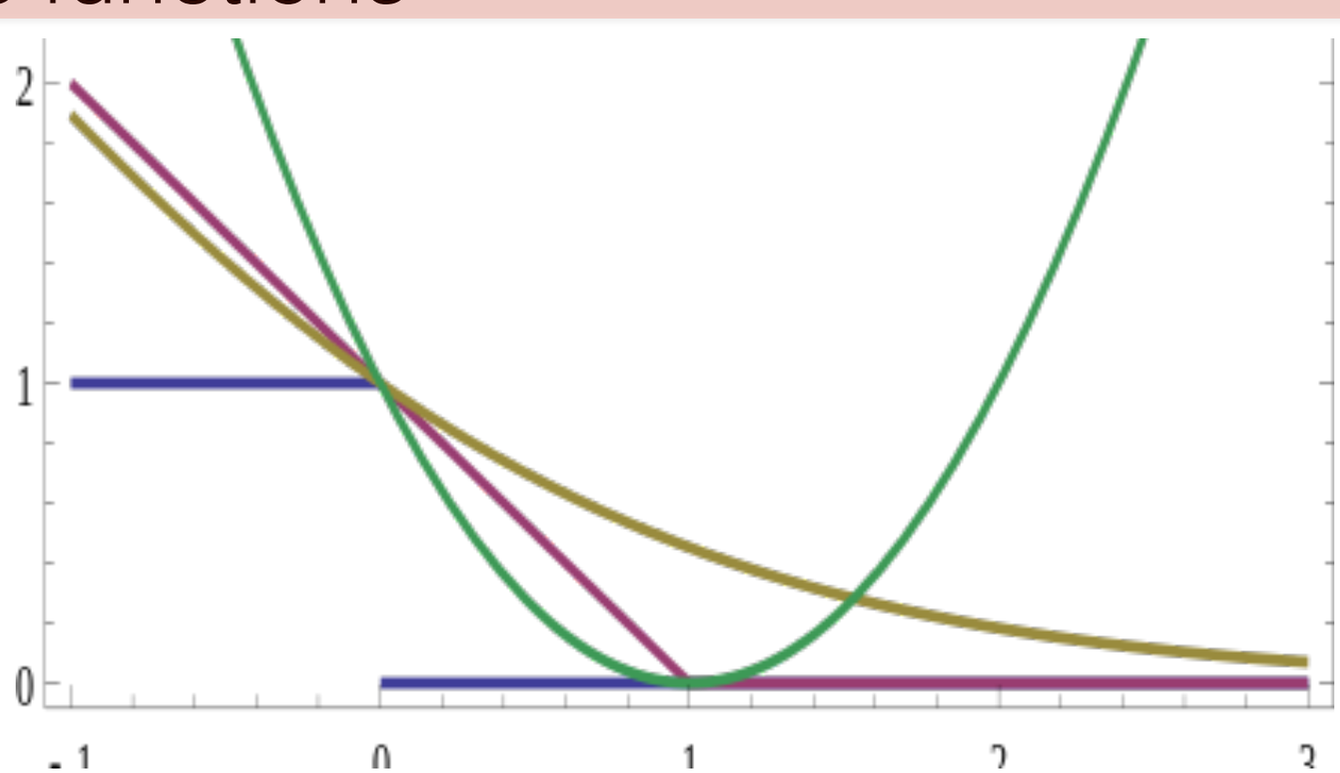
- Regression:
  - L2 loss
  - L1 loss
- Classification:
  - cross entropy loss (N-output classifier $\mathbf{f}(\mathbf{x}, \mathbf{w})$)
  - logistic loss (single output dichotomy classifier $f(\mathbf{x}, \mathbf{w})$)

$$L(\mathbf{w}) = \sum_i \log\left[1 + \exp(-y_i \, f(\mathbf{x}_i, \mathbf{w}))\right]$$

Derivative can be found here:
https://deepnotes.io/softmax-crossentropy

Loss functions

- Regression:
  - L2 loss
  - L1 loss
- Classification:
  - cross entropy loss (N-output classifier $\mathbf{f}(\mathbf{x}, \mathbf{w})$)
  - logistic loss (single output dichotomy classifier $f(\mathbf{x}, \mathbf{w})$)
  - other loss functions



https://en.wikipedia.org/wiki/Loss_functions_for_classification

Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
- regularizations
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,

Regularization

- L2, L1 norms on weights are simple regularizations
- Batch norm is regularization
- Drop out is regularization (it trains committee of experts)
- Jittering of training data is regularization

Outline

- SGD vs deterministic gradient
- what makes learning to fail
- layers:
  - activation function (i.e. non-linearities)
  - batch normalization layer
  - max-pooling layer
  - loss-layers
  - regularizations
- summary of the learning procedure
  - train, test, val data,
  - hyper-parameters,

Training procedure

- Choose:
  - Weight initialization
  - Network architecture
  - Learning rate
  - Loss + regularization
- Divide data on three represenatative 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 only observed)

Hyper parameters tuning

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