Learning for vision III
Convolutional networks

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

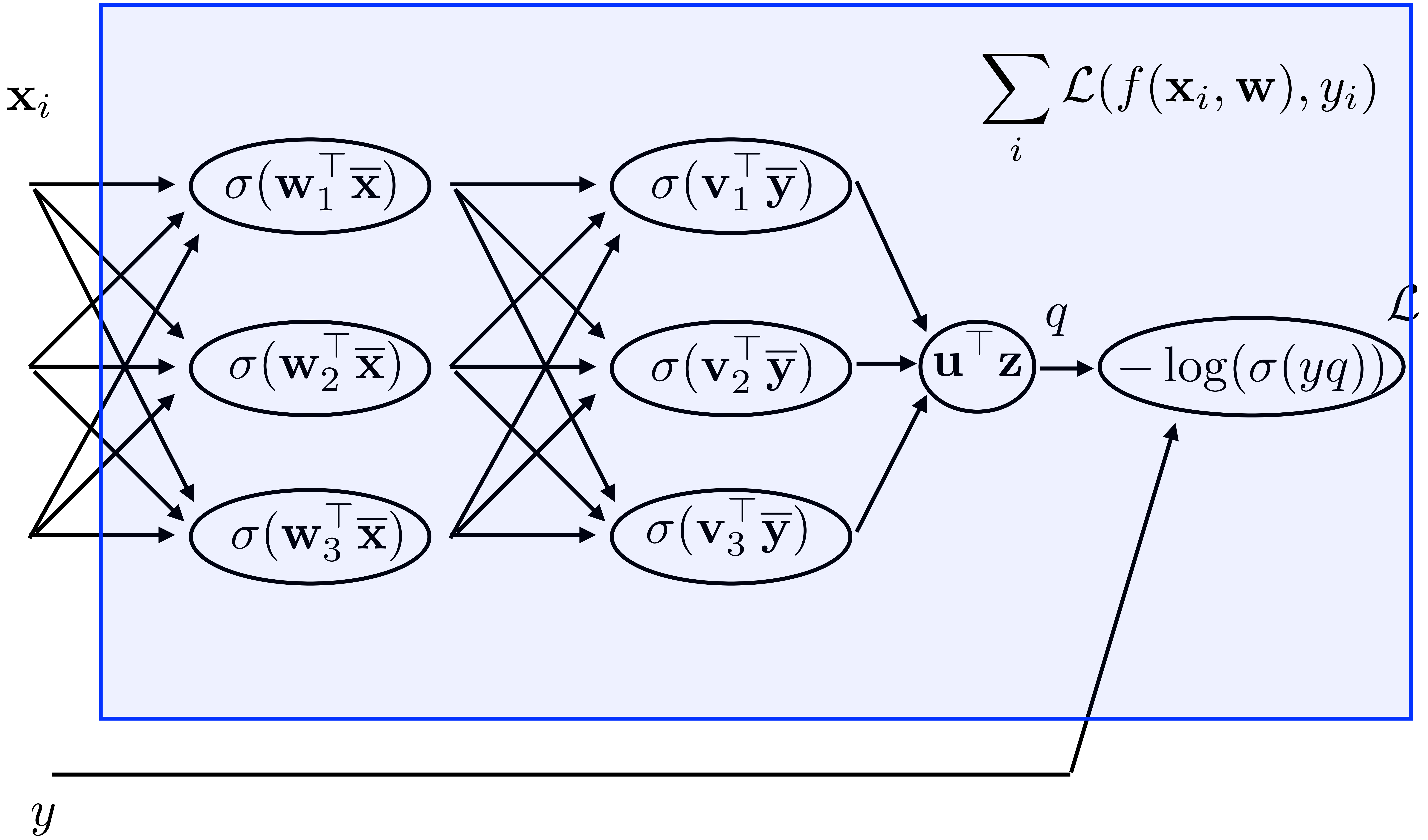
Department for Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Do clean up + if not needed, switch off remote machines (it might be switched off automatically after 24h of inactivity).

Outline

- Fully connected neural network
- Avoid overfitting by search for the NN model suitable for image processing [Hubel and Wiesel 1960].
- Feedforward and Backprop in ConvNets.
- Epmiric evaluation of classifier performance (Precision, Recall).

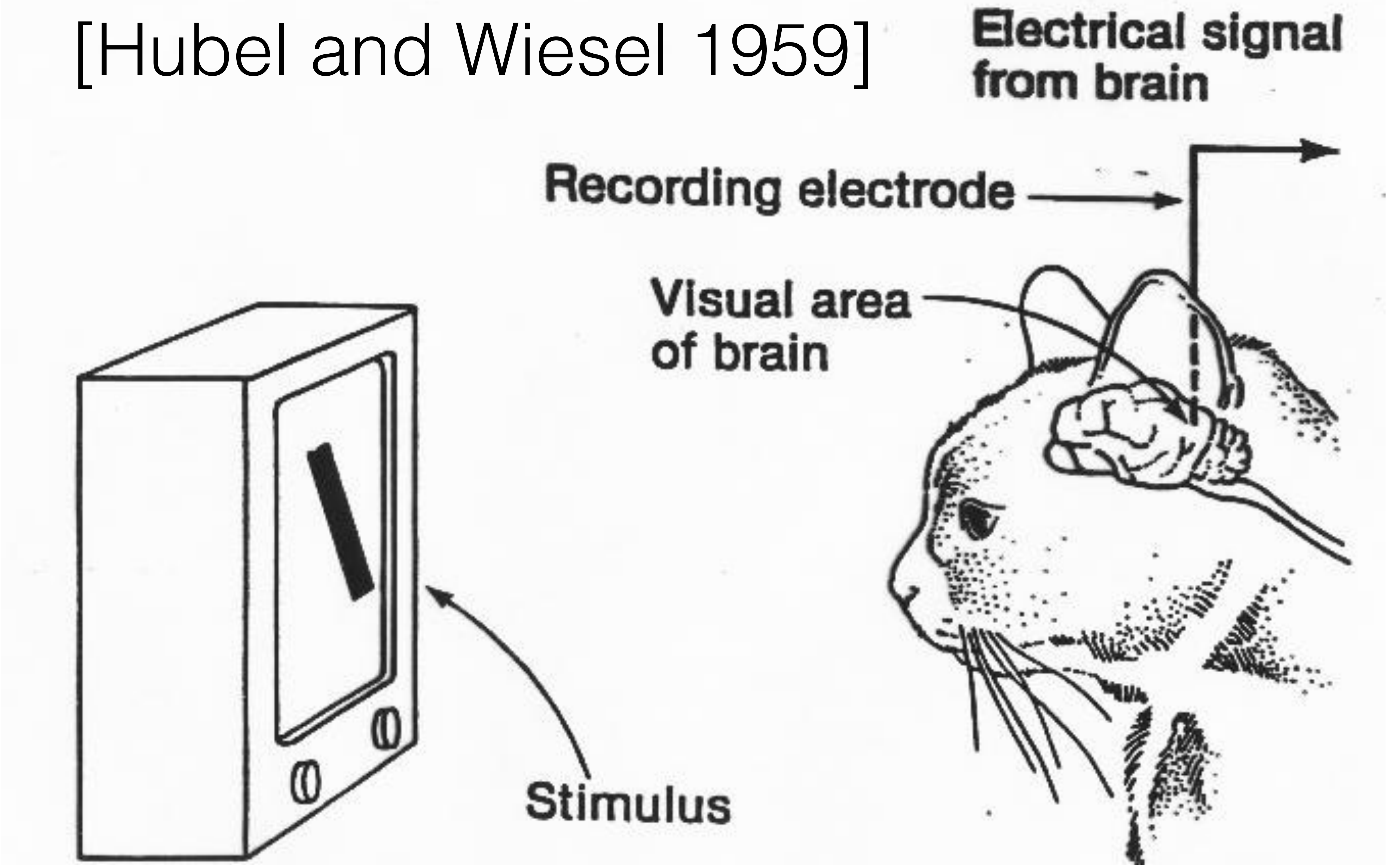
Fully connected neural network



The Tungsten Electrode [Hubel-Science-1957]

http://braintour.harvard.edu/archives/portfolio-items/hubel-and-wiesel

- Device capable to record signal from a single neuron



- Experiment with anaesthetised paralysed cat

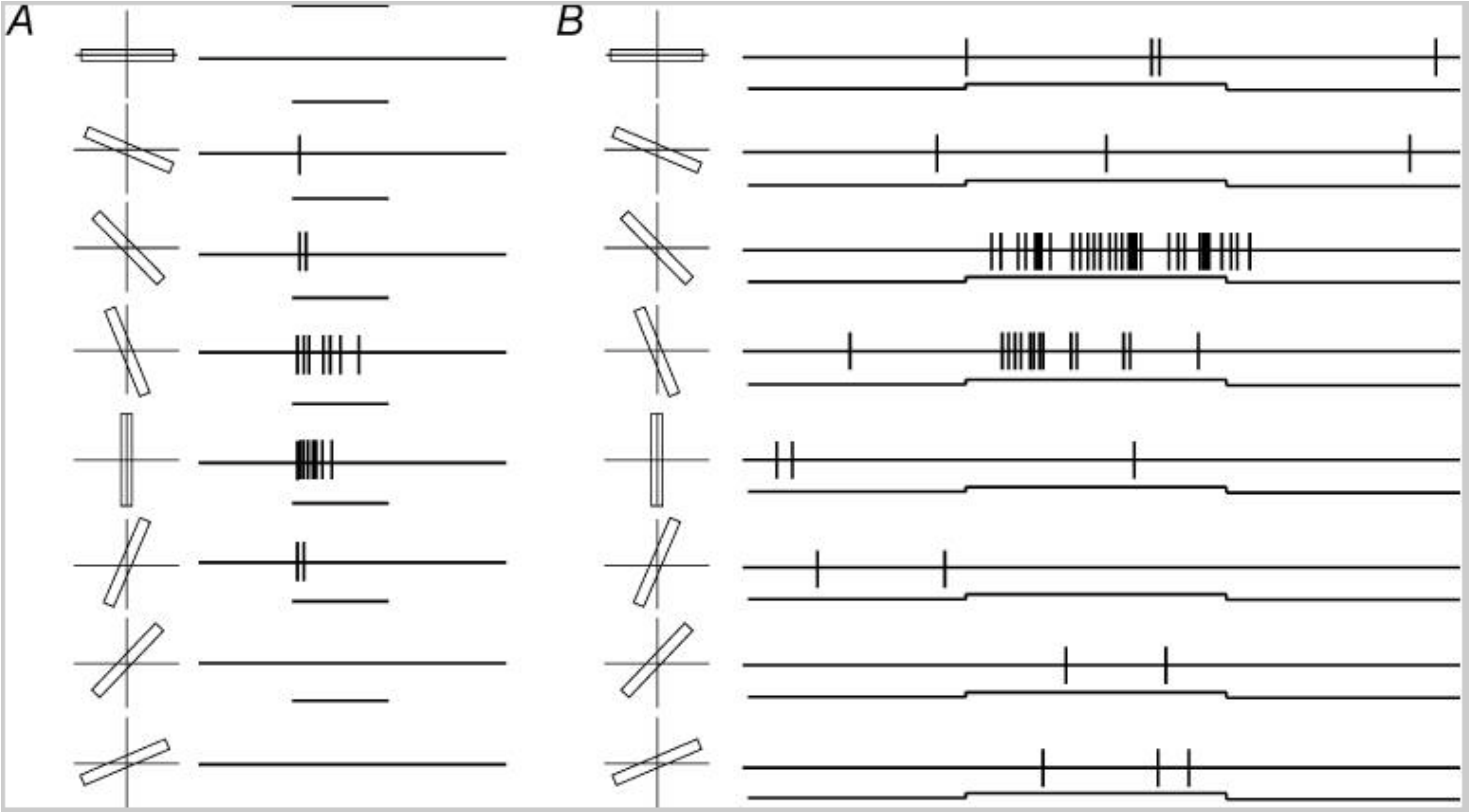
[Hubel and Wiesel 1960]

https://knowingneurons.com/2014/10/29/hubel-and-wiesel-the-neural-basis-of-visual-perception/

[Hubel and Wiesel 1960]

paralysed cat

awake monkey



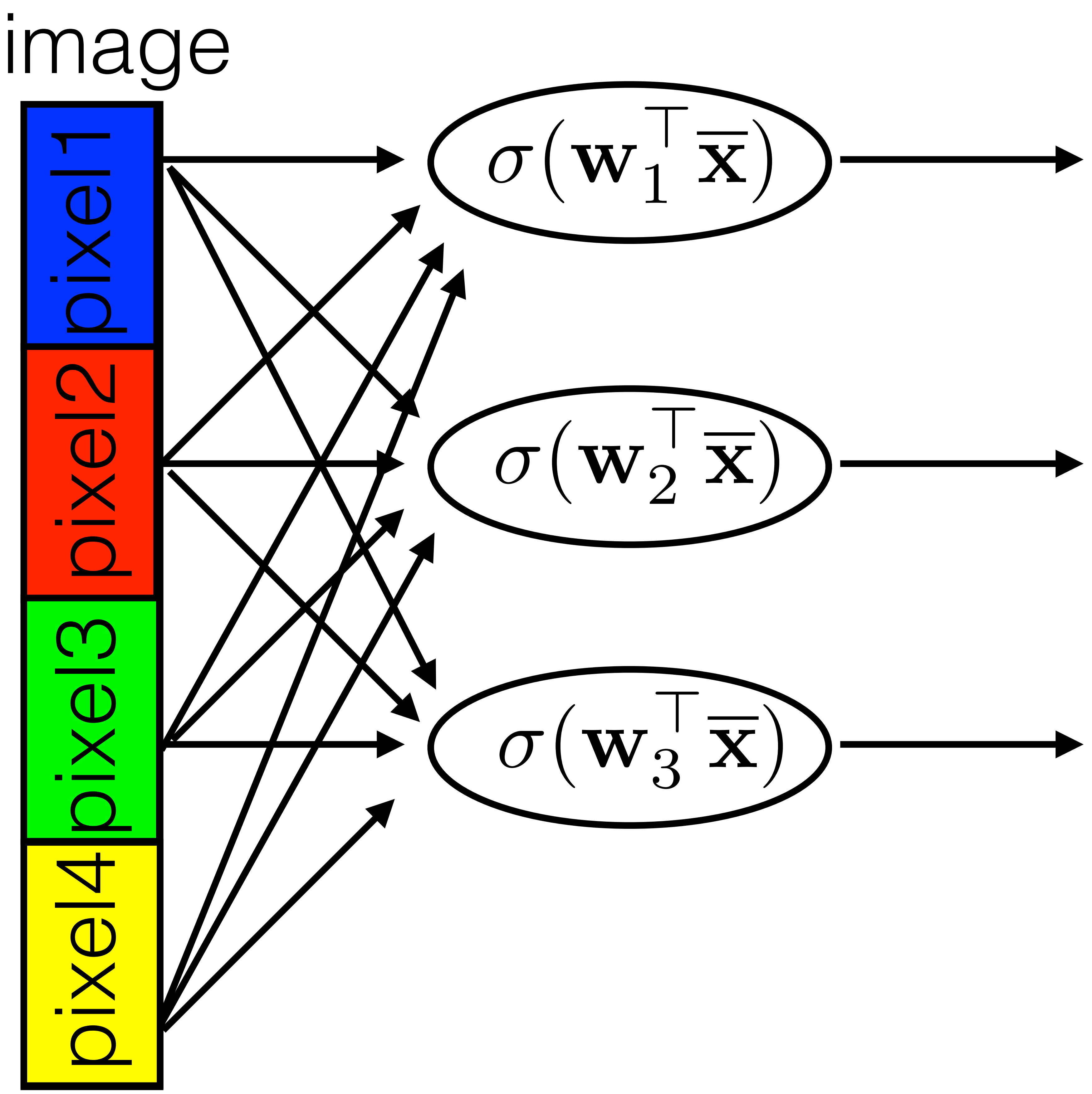
https://knowingneurons.com/2014/10/29/hubel-and-wiesel-the-neural-basis-of-visual-perception/

Hubel and Wiesel experiments in 1950s and 1960s

- Nobel Prize in Physiology and Medicine in 1981
- Dr. Hubel: “There has been a myth that the brain cannot understand itself. It is compared to a man trying to lift himself by his own bootstraps. We feel that is nonsense. The brain can be studied just as the kidney can.”

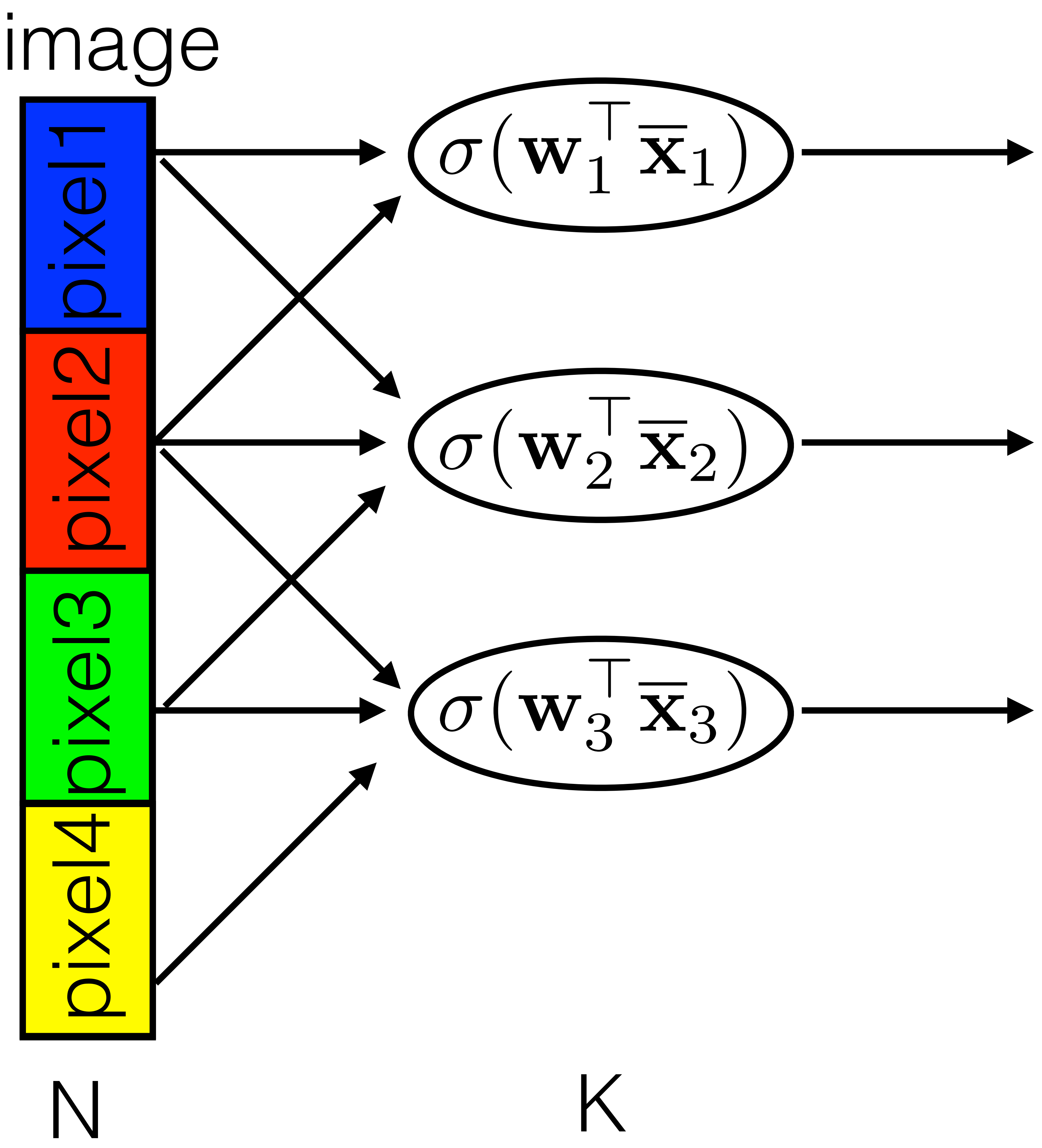
https://knowingneurons.com/2014/10/29/hubel-and-wiesel-the-neural-basis-of-visual-perception/

1. Nearby neurons process information from nearby visual fields (topographical map).



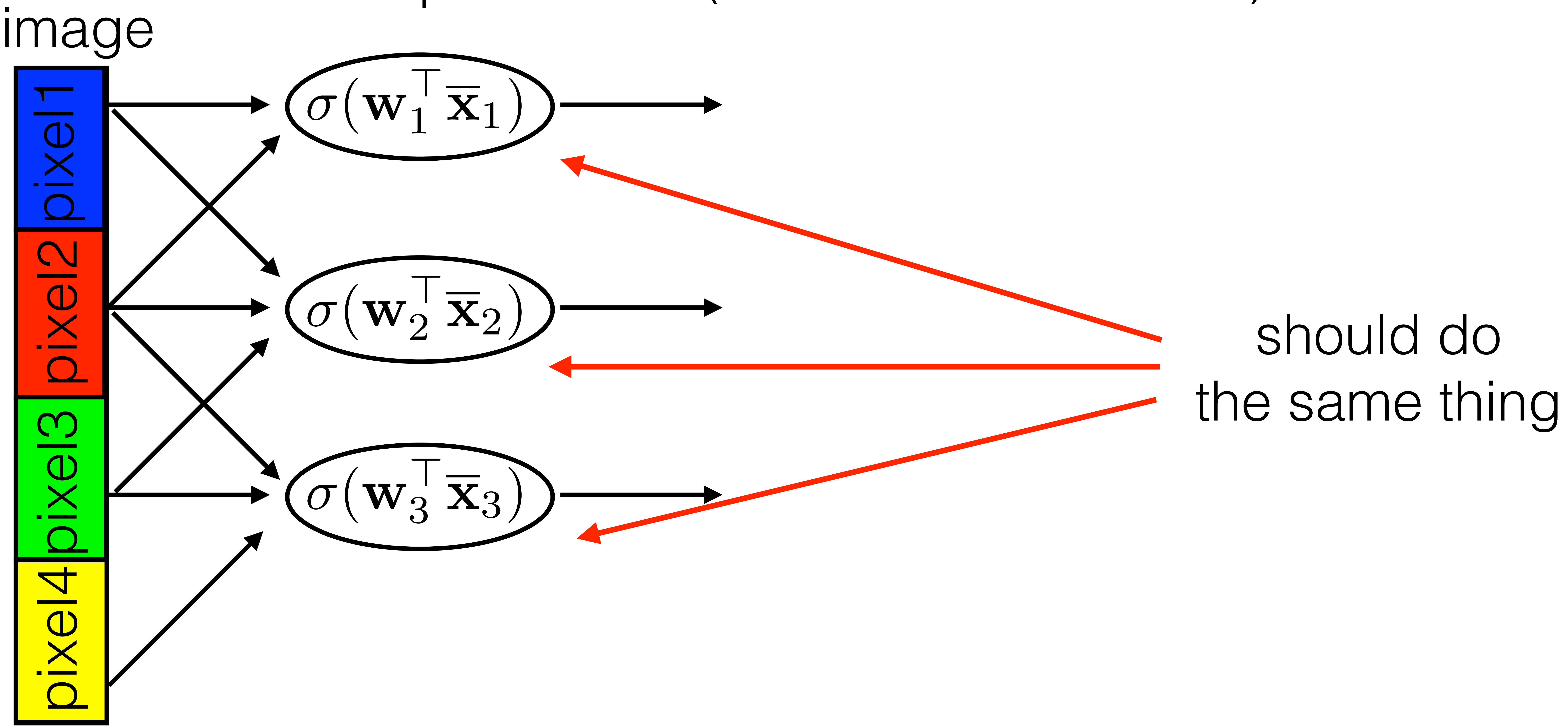
- Processing of visual information in cortex is not fully connected.

1. Nearby neurons process information from nearby visual fields (topographical map).



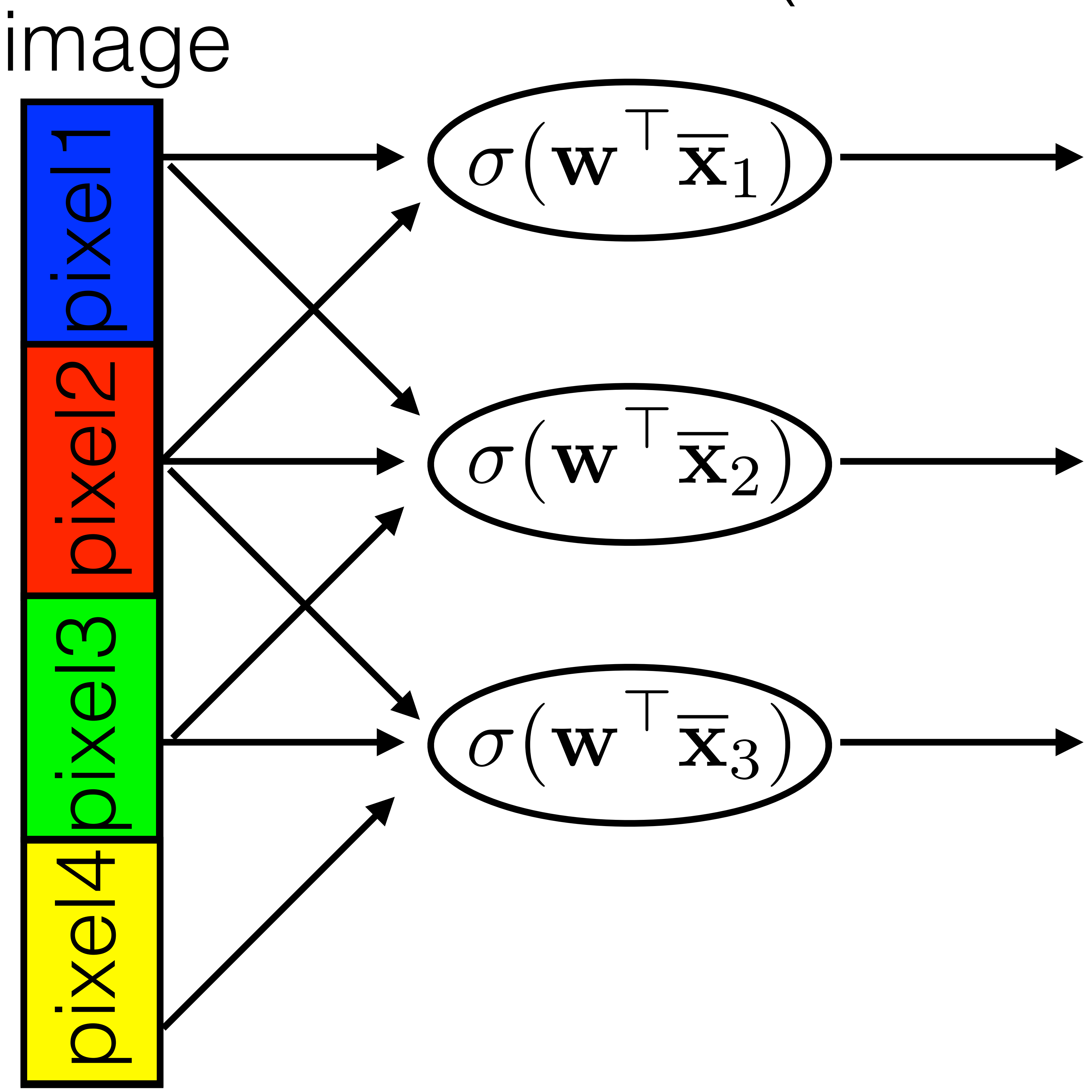
- What is dimensionality reduction for N-pixel image and n-dimensional spatial neighbourhood?

2. Different neurons detects the same edge at different positions (translation invariance)

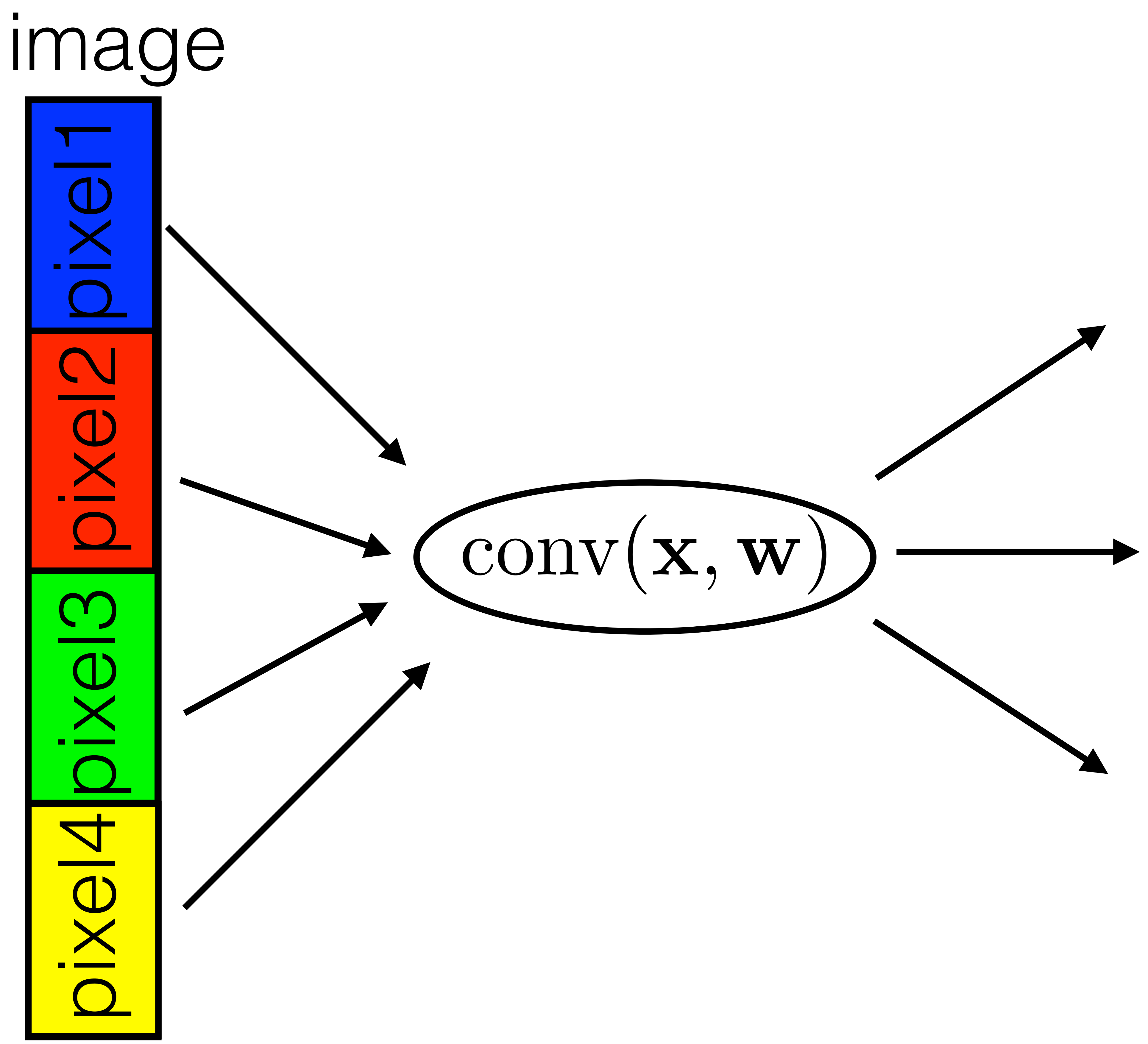


There are neurons which detect an edge on the left and there are different n which detect the same edge on the right.

2. Neurons with similar function organized into columns (translation invariance)

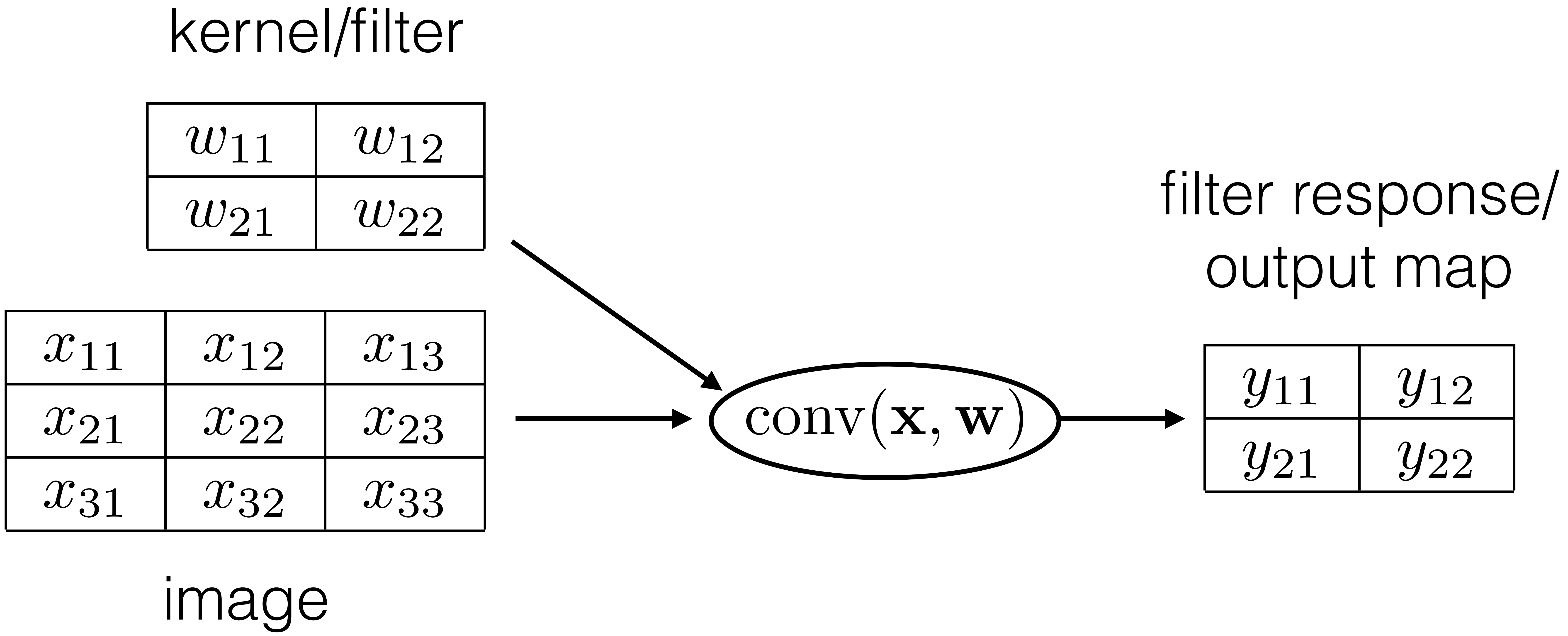


It corresponds to convolution of image $\mathbf{x}$ with kernel $\mathbf{w}$ followed by activation function



It corresponds to convolution of image $\mathbf{x}$ with kernel $\mathbf{w}$ followed by activation function

Convolution forward pass $\mathbf{y} = \mathrm{conv}(\mathbf{x}, \mathbf{w})$



Convolution forward pass $\mathbf{y} = \mathrm{conv}(\mathbf{x}, \mathbf{w})$

$$\begin{array}{|c|c|}\hline y_{11} & y_{12} \\ \hline y_{21} & y_{22} \\ \hline \end{array} = \mathrm{conv}\left(\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array}, \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array} \right)$$

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

Convolution forward pass $\mathbf{y} = \mathrm{conv}(\mathbf{x}, \mathbf{w})$

$$\begin{array}{|c|c|}\hline y_{11} & y_{12} \\ \hline y_{21} & y_{22} \\ \hline\end{array} = \mathrm{conv}\left(\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline\end{array}, \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline\end{array} \right)$$

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

Convolution forward pass $\mathbf{y} = \mathrm{conv}(\mathbf{x}, \mathbf{w})$

$$\begin{array}{|c|c|}\hline y_{11} & \boxed{y_{12}} \\ \hline y_{21} & y_{22} \\ \hline \end{array} = \mathrm{conv}\left(\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array}, \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array} \right)$$

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

Convolution forward pass $\mathbf{y} = \mathrm{conv}(\mathbf{x}, \mathbf{w})$

$$\begin{array}{|c|c|}\hline y_{11} & y_{12} \\ \hline y_{21} & y_{22} \\ \hline \end{array} = \mathrm{conv}\left(\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array}, \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array} \right)$$

$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$

$$y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}$$

Convolution forward pass $\mathbf{y} = \mathrm{conv}(\mathbf{x}, \mathbf{w})$

$$\begin{array}{|c|c|}\hline y_{11} & y_{12} \\ \hline y_{21} & \boxed{y_{22}} \\ \hline \end{array} = \mathrm{conv}\left(\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array}, \begin{array}{|c|c|}\hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array} \right)$$

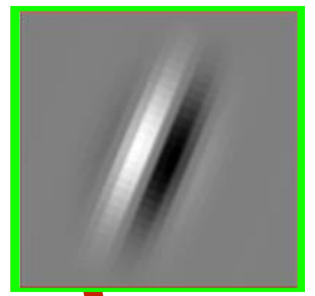
$$y_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22}$$

$$y_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23}$$

$$y_{21} = w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + w_{22}x_{32}$$

$$\boxed{y_{22} = w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + w_{22}x_{33}}$$

Feature maps

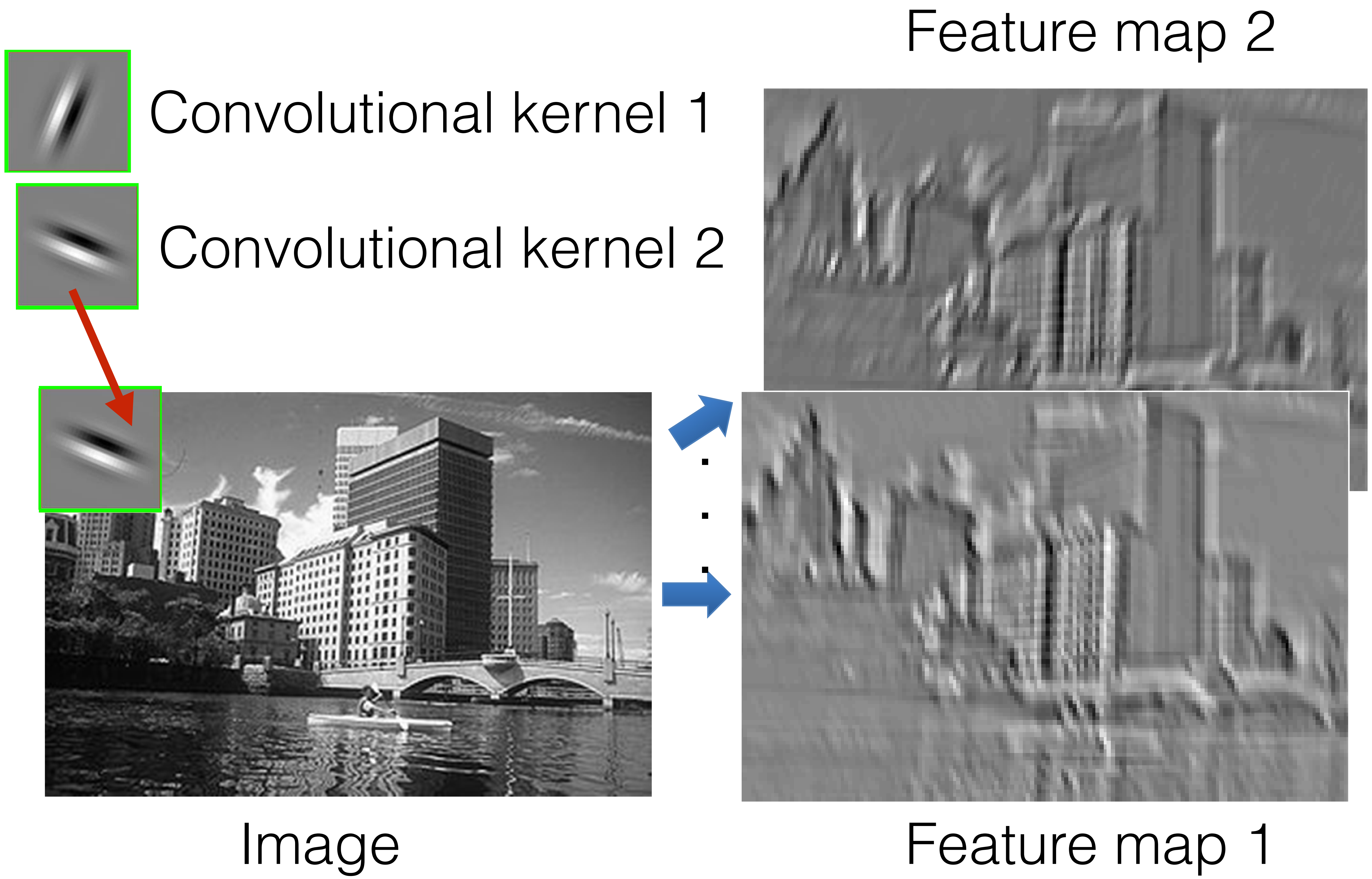


Convolutional kernel 1

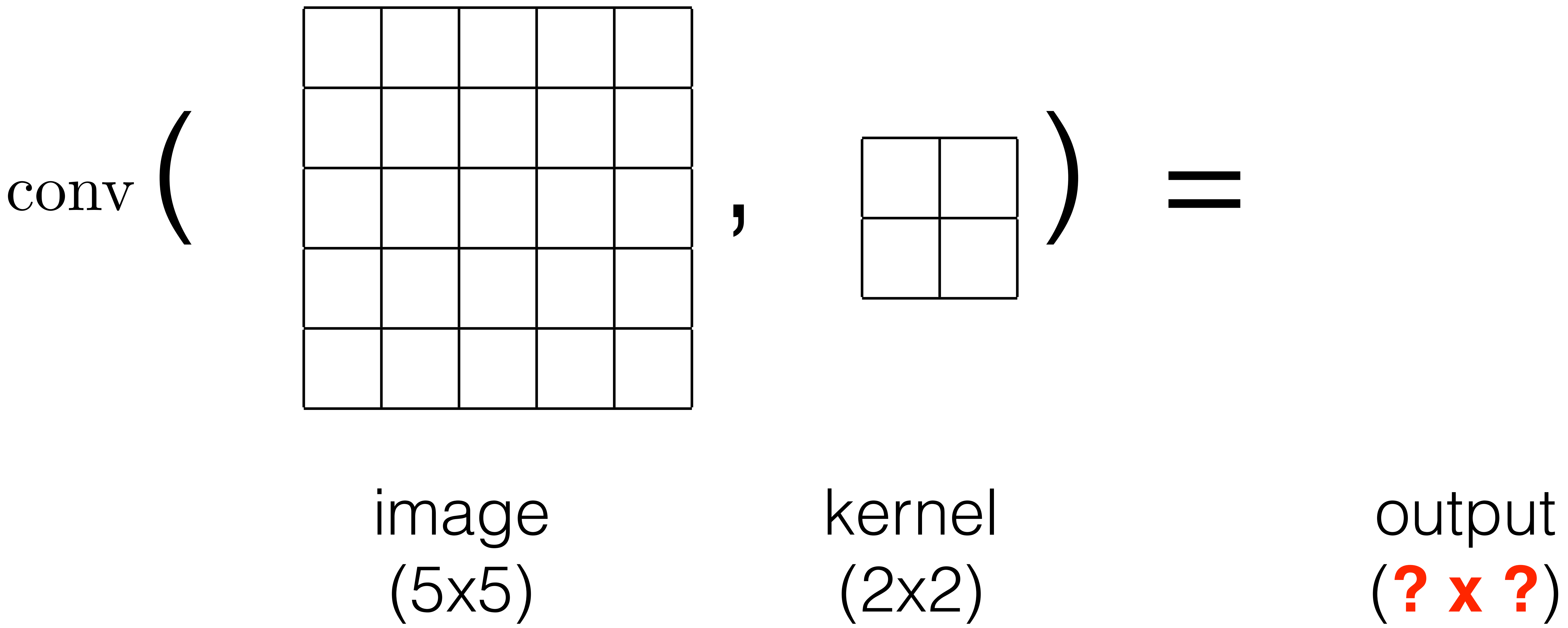
Image

Feature map 1

Feature maps



Convolution layer properties - output size



Convolution layer properties - output size

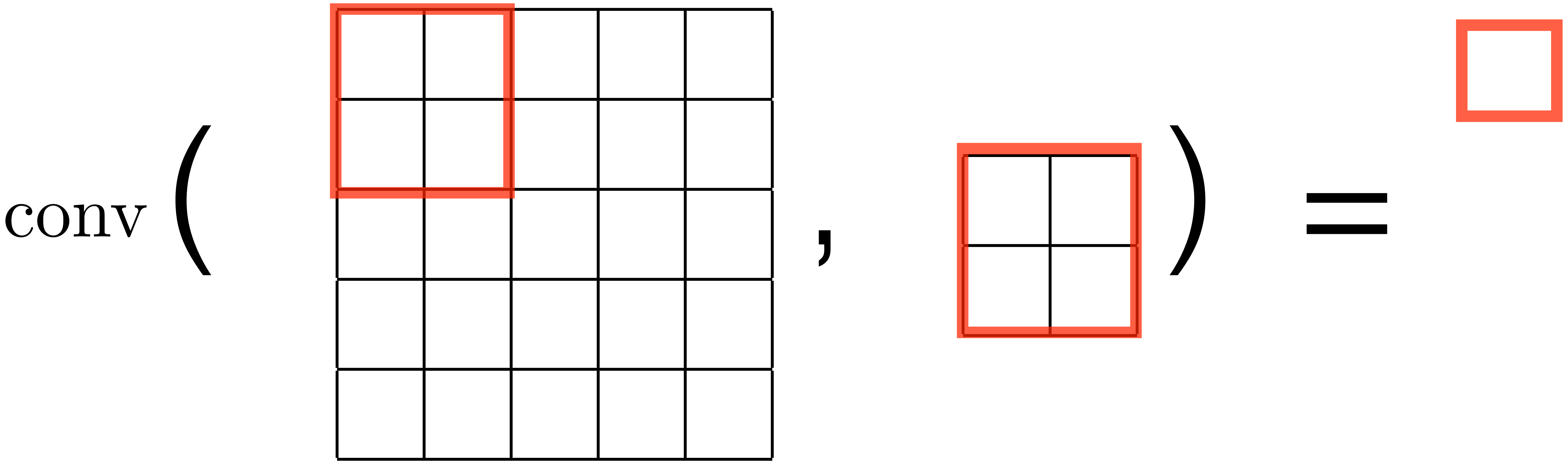


image
(5x5)

kernel
(2x2)

output
(**? x ?**)

Convolution layer properties - output size

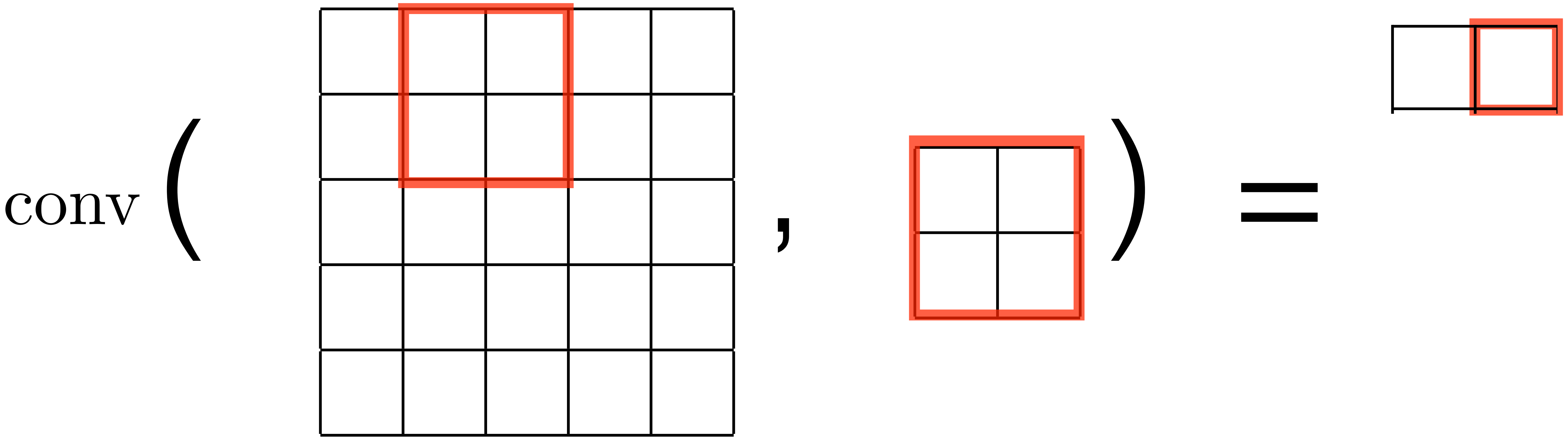


image (5x5) | kernel (2x2) | output (**? x ?**)

Convolution layer properties - output size

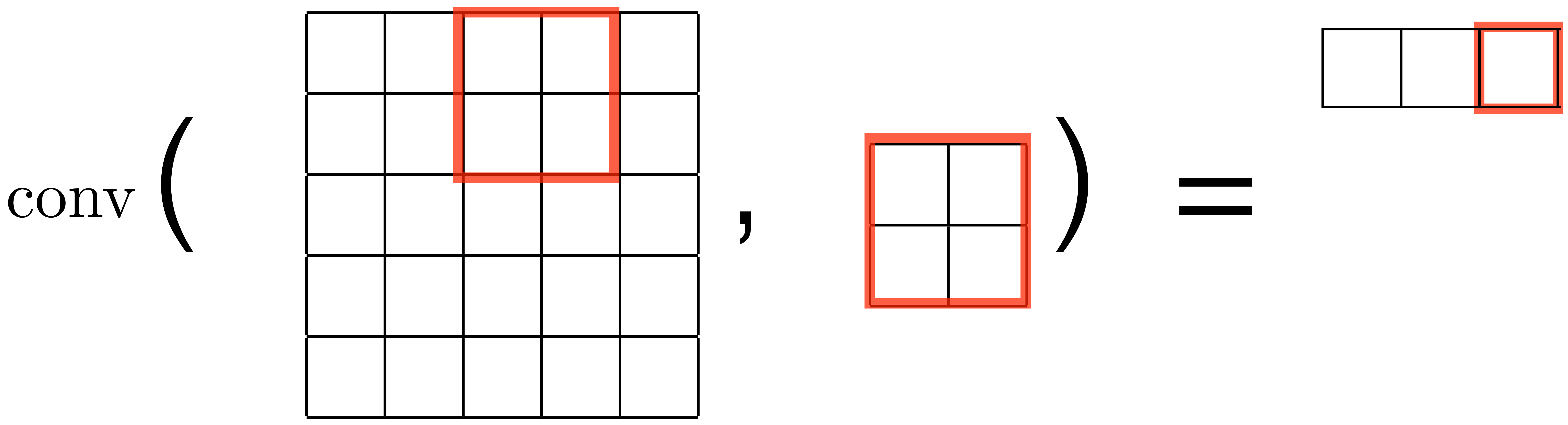


image
(5x5)

kernel
(2x2)

output
(**? x ?**)

Convolution layer properties - output size

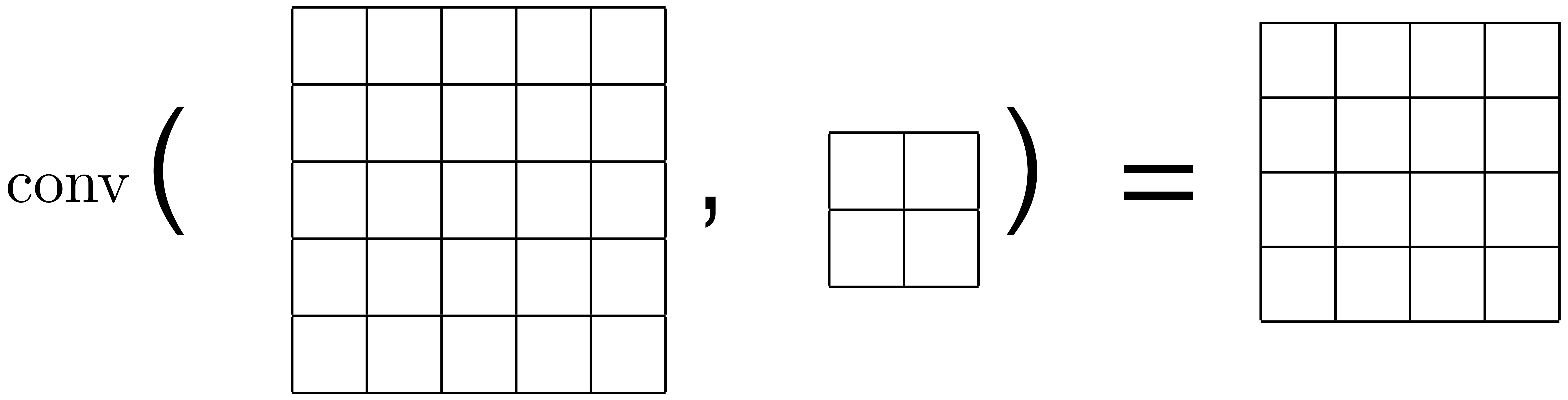


image (5x5)

kernel (2x2)

output (4x4)

Convolution layer properties - output size

$$M = N - K + 1$$

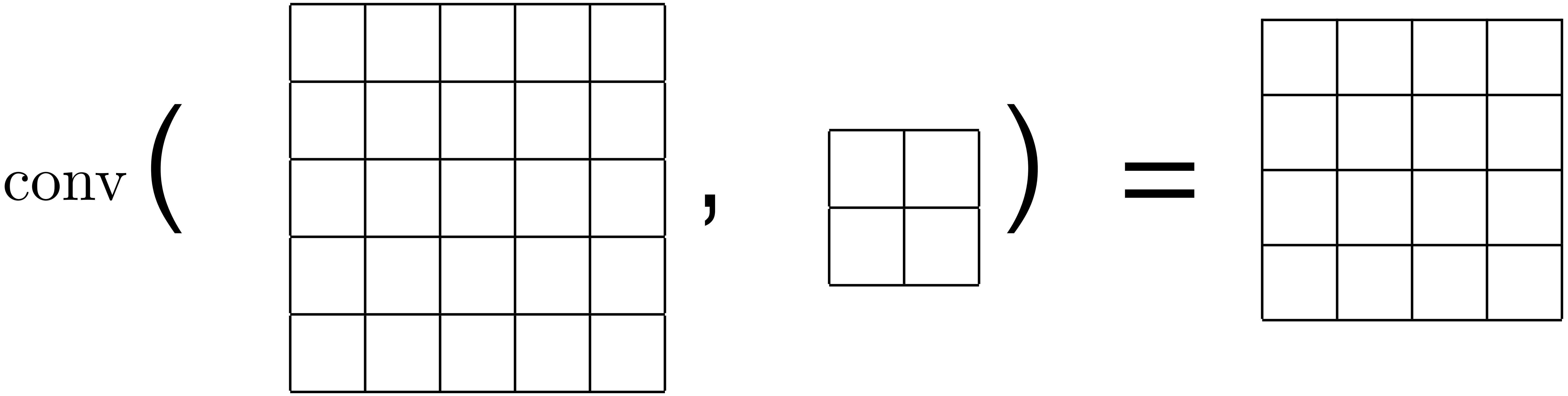


image (NxN) | kernel (KxK) | output (MxM)

Convolution layer properties - stride

stride = 1

kernel moves by 1 pixel

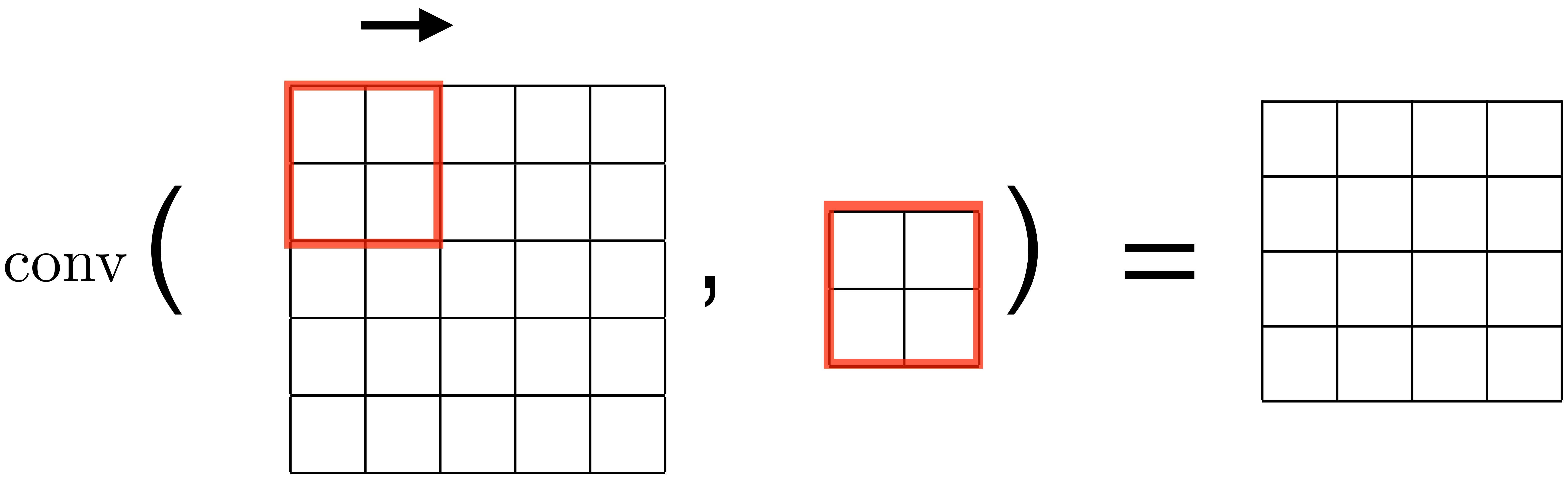


image (5x5)

kernel (2x2)

output (4x4)

Convolution layer properties - stride

stride = 3

kernel moves by 3 pixels

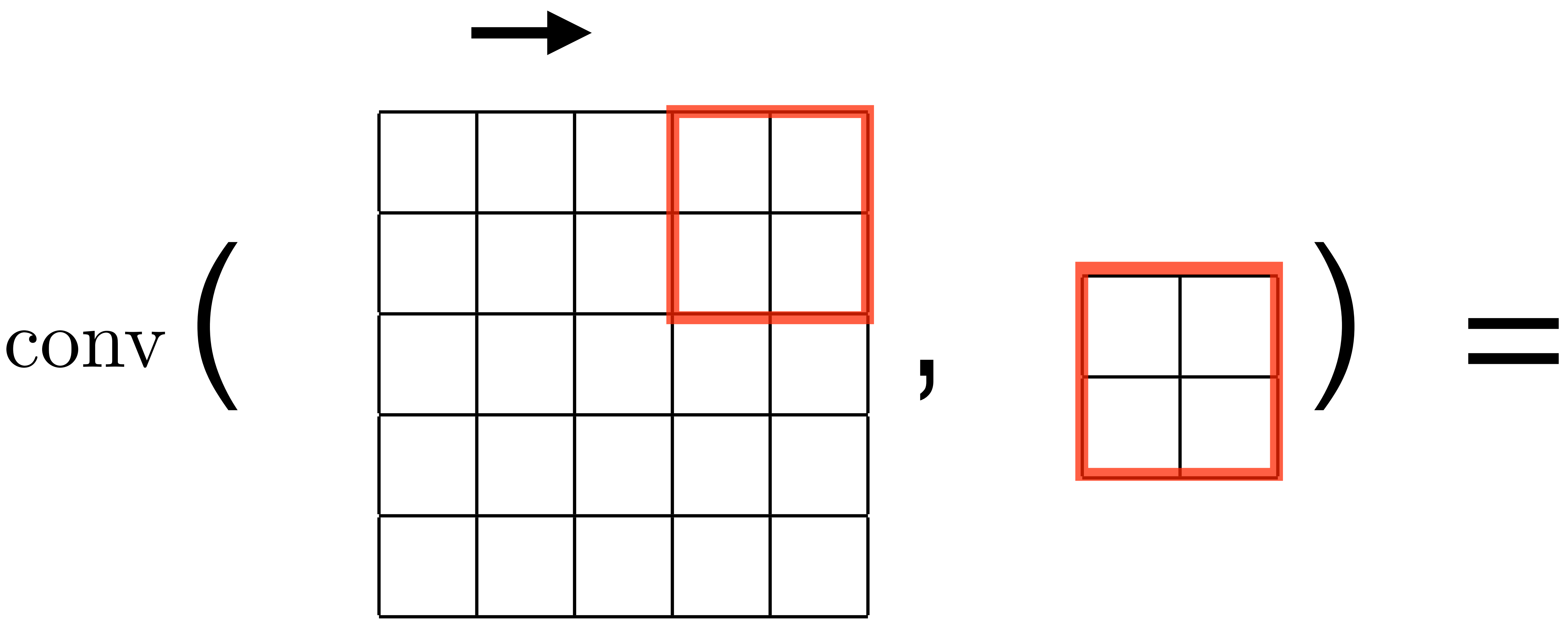


image
(5x5)

kernel
(2x2)

output
(**? x ?**)

Convolution layer properties - stride

stride = 3

kernel moves by 3 pixels

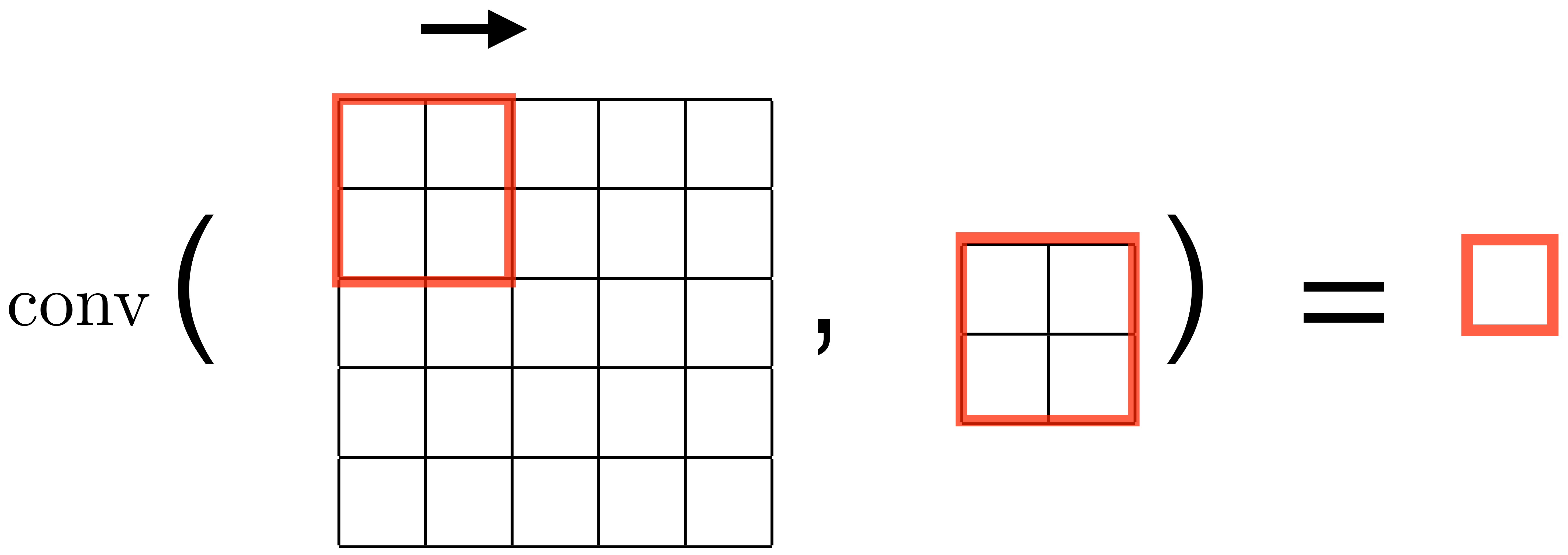


image (5x5)

kernel (2x2)

output (**? x ?**)

Convolution layer properties - stride

stride = 3

kernel moves by 3 pixels

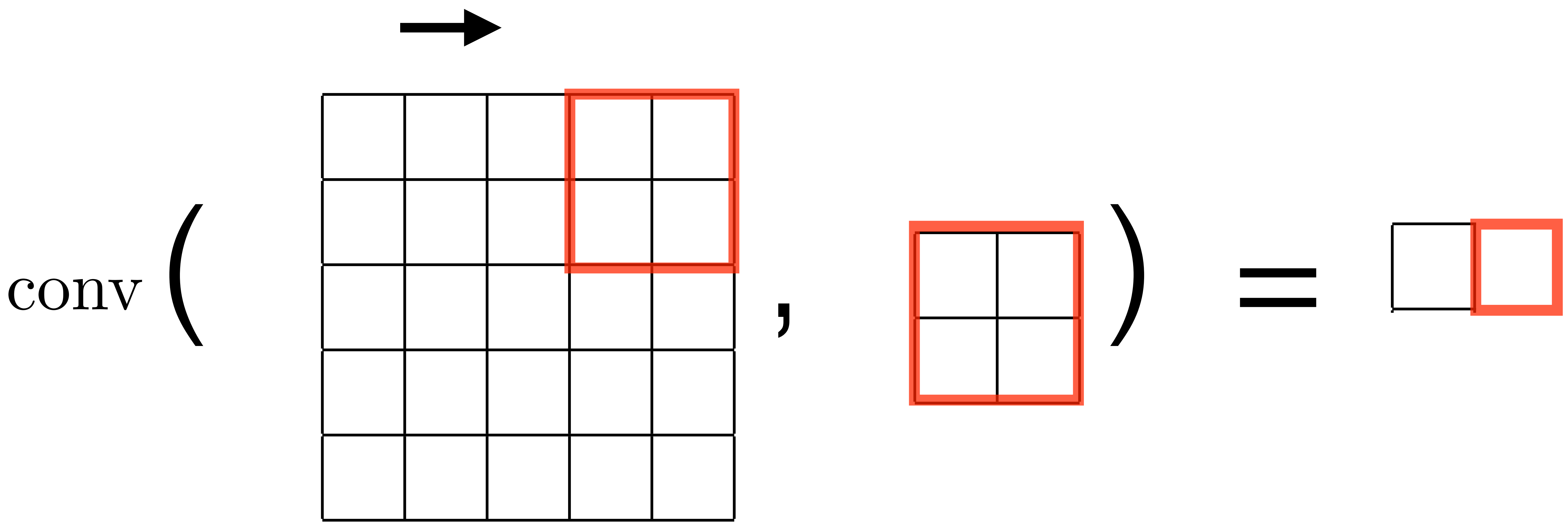


image
(5x5)

kernel
(2x2)

output
(? x ?)

Convolution layer properties - stride

stride = 3

kernel moves by 3 pixels

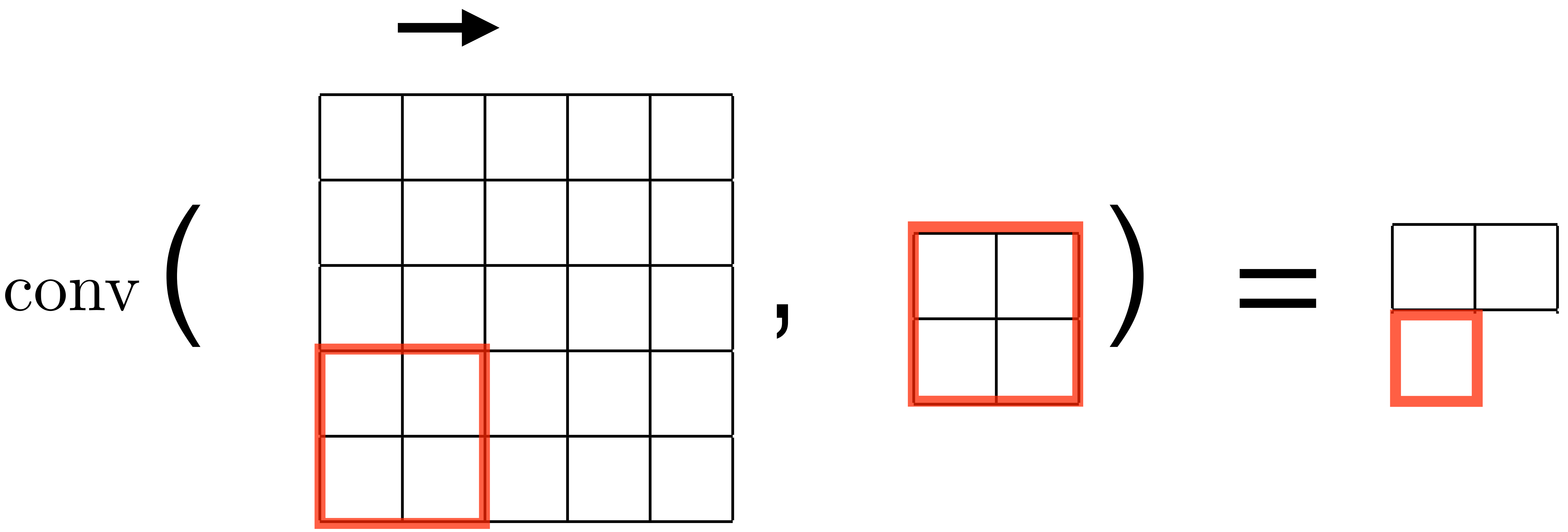


image (5x5) | kernel (2x2) | output (**? x ?**)

Convolution layer properties - stride

stride = 3

kernel moves by 3 pixels

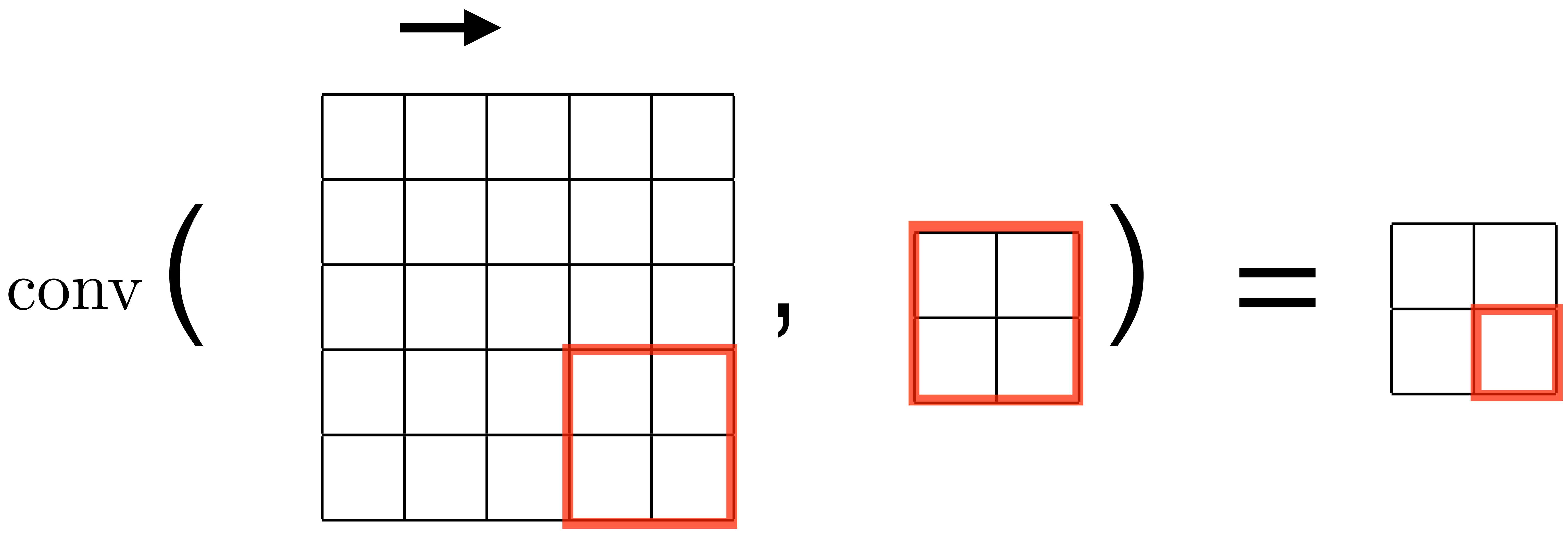


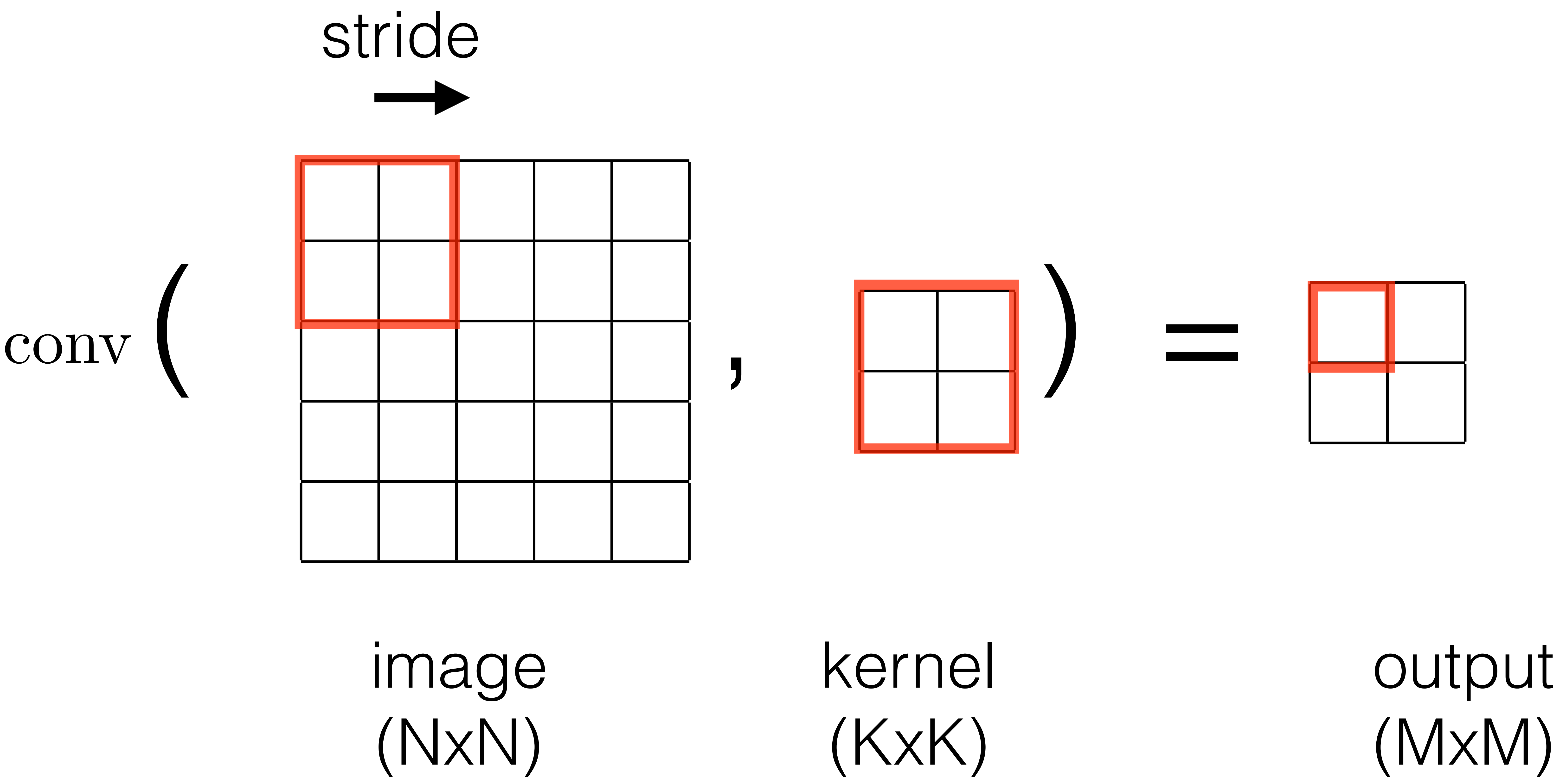
image
(5x5)

kernel
(2x2)

output
(2x2)

Convolution layer properties - stride

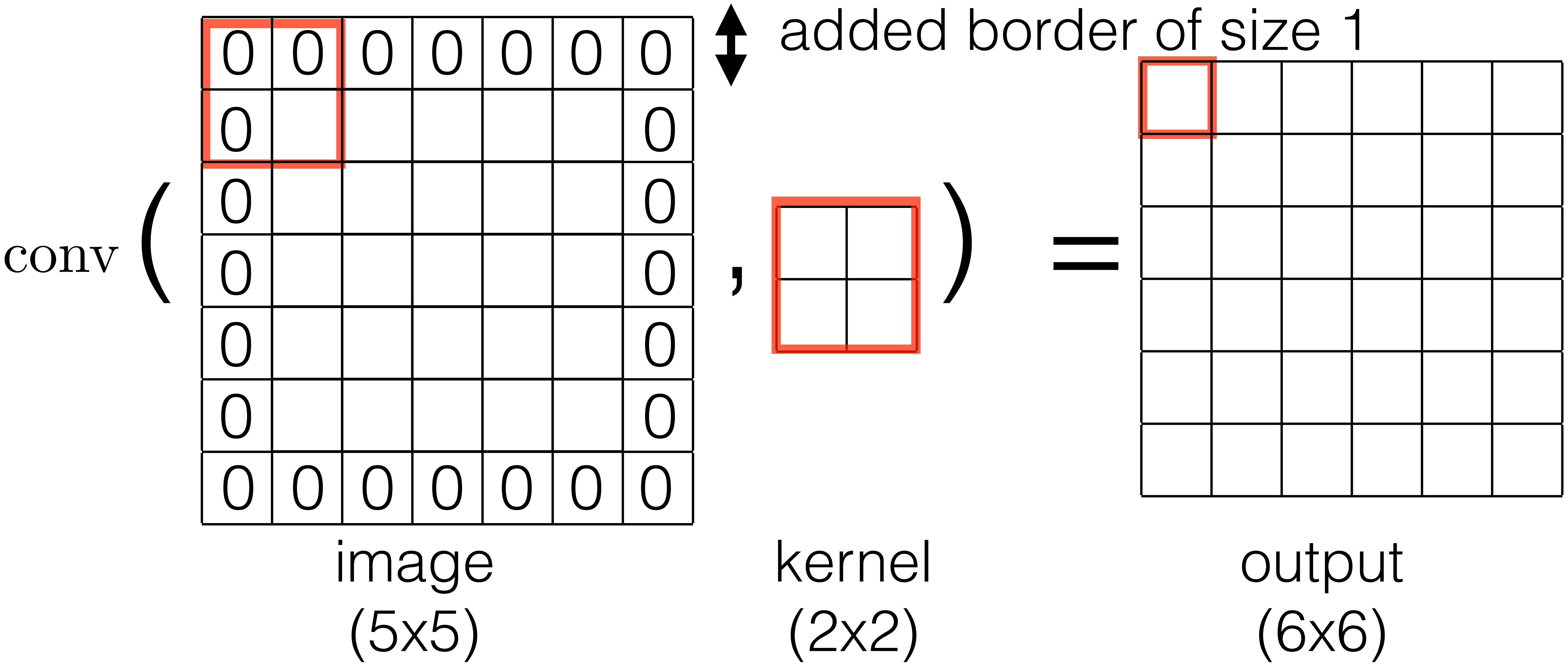
$$M = \texttt{floor}(\,(N\text{-}K) / \text{stride} + 1)$$



e.g. $M = (5\text{-}2) / 3 + 1 = 2$

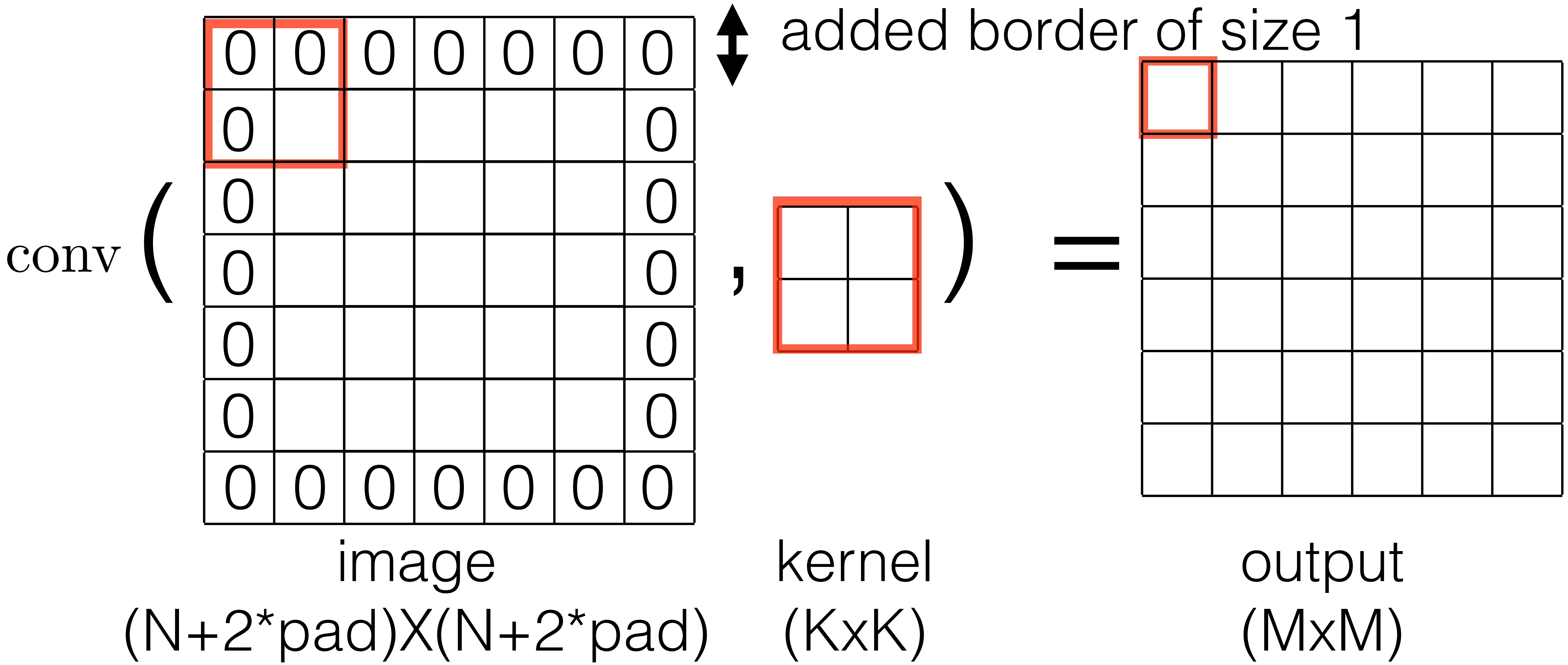
Convolution layer properties - pad

pad = 1



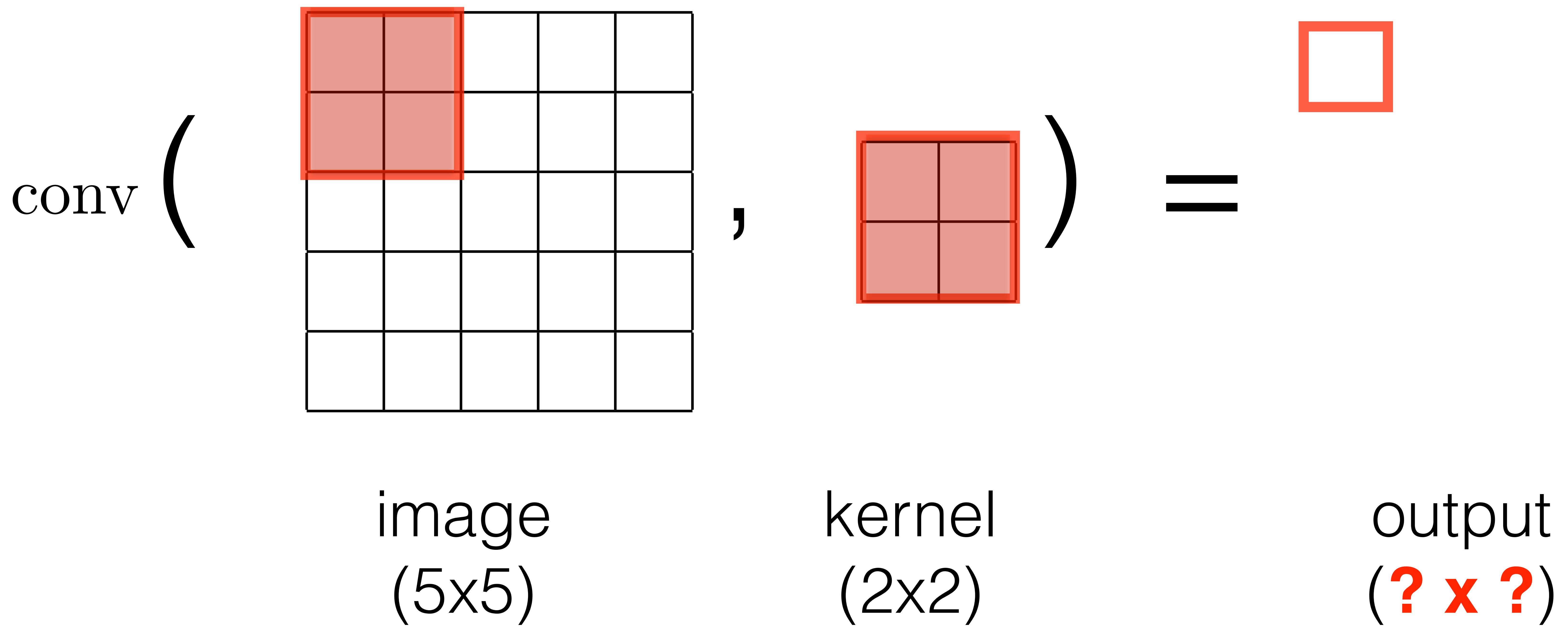
Convolution layer properties - pad

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



Convolution layer

Dilatation rate = 1



Atrous convolution layer

Dilatation rate = 2

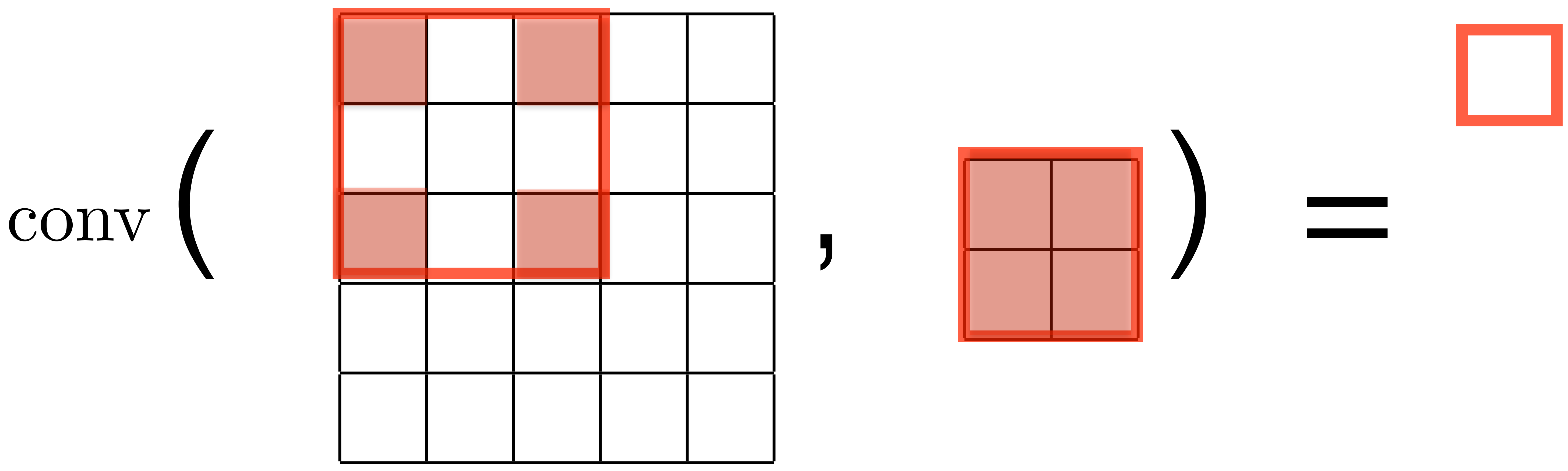


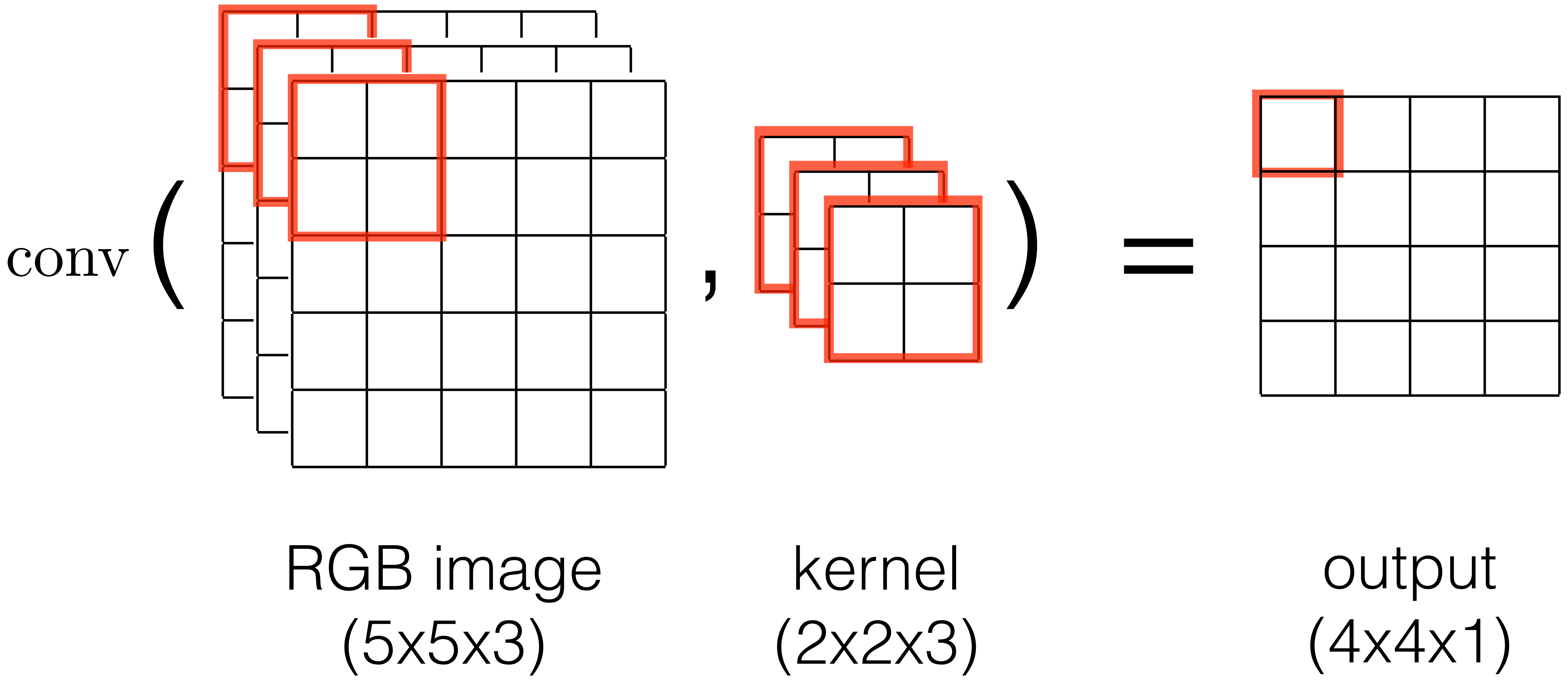
image
(5x5)

kernel
(2x2)

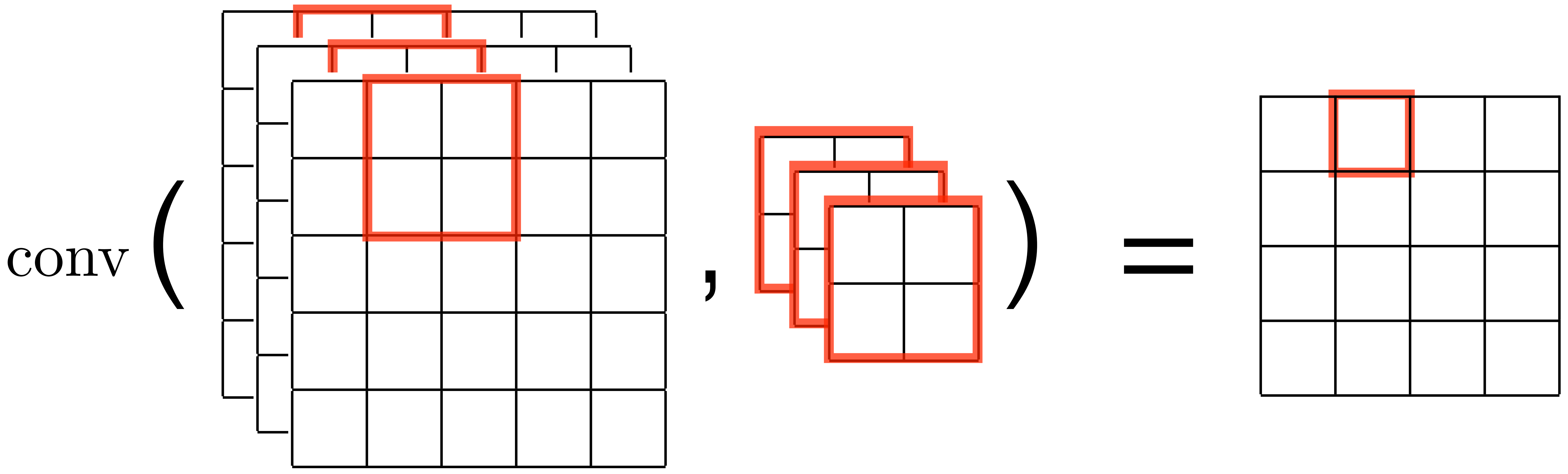
output
(**? x ?**)

Show python code

Multi-channel convolution



Multi-channel convolution

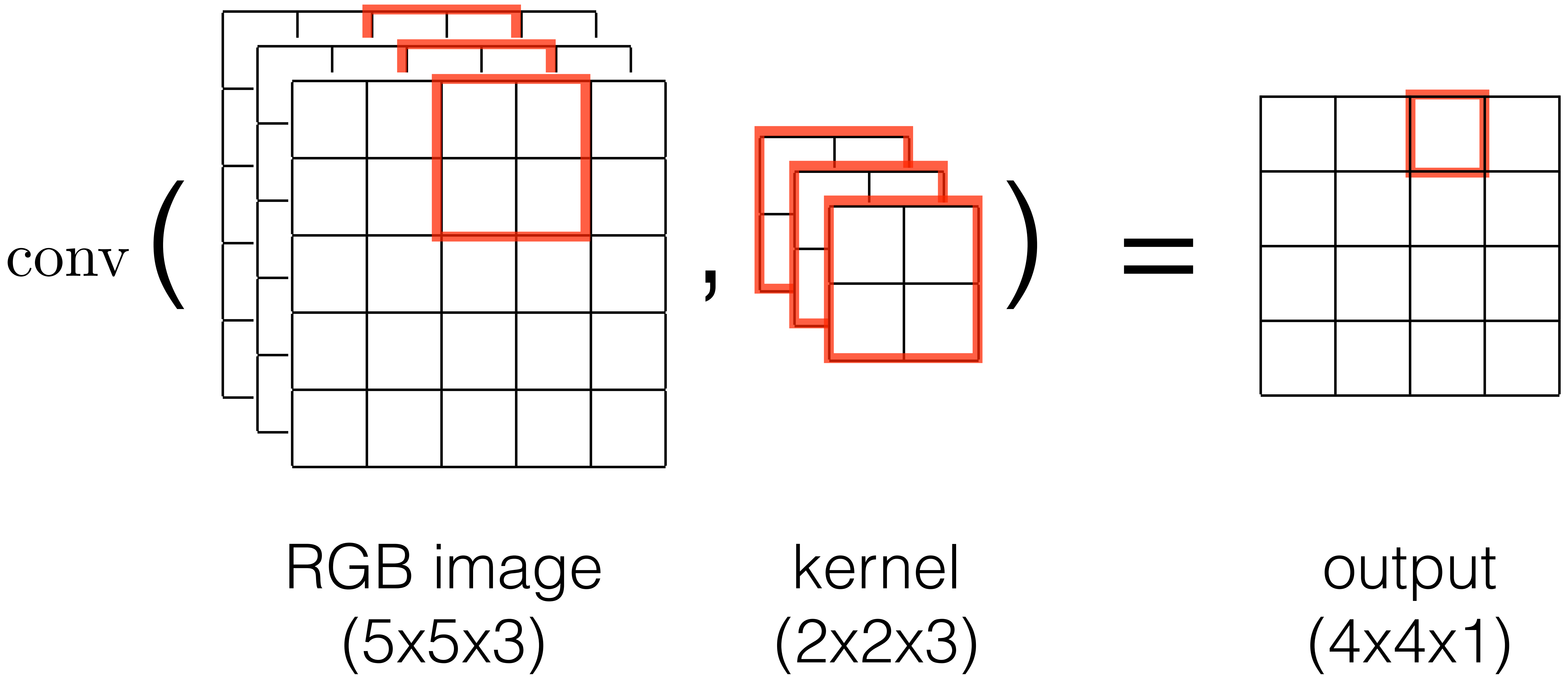


RGB image
(5x5x3)

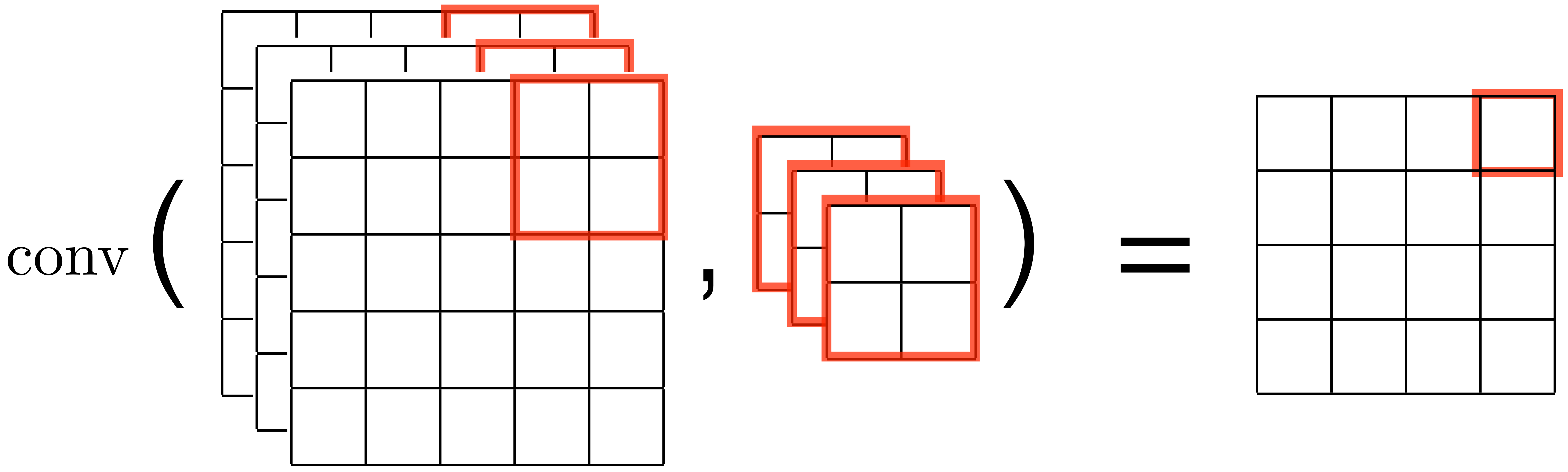
kernel
(2x2x3)

output
(4x4x1)

Multi-channel convolution



Multi-channel convolution

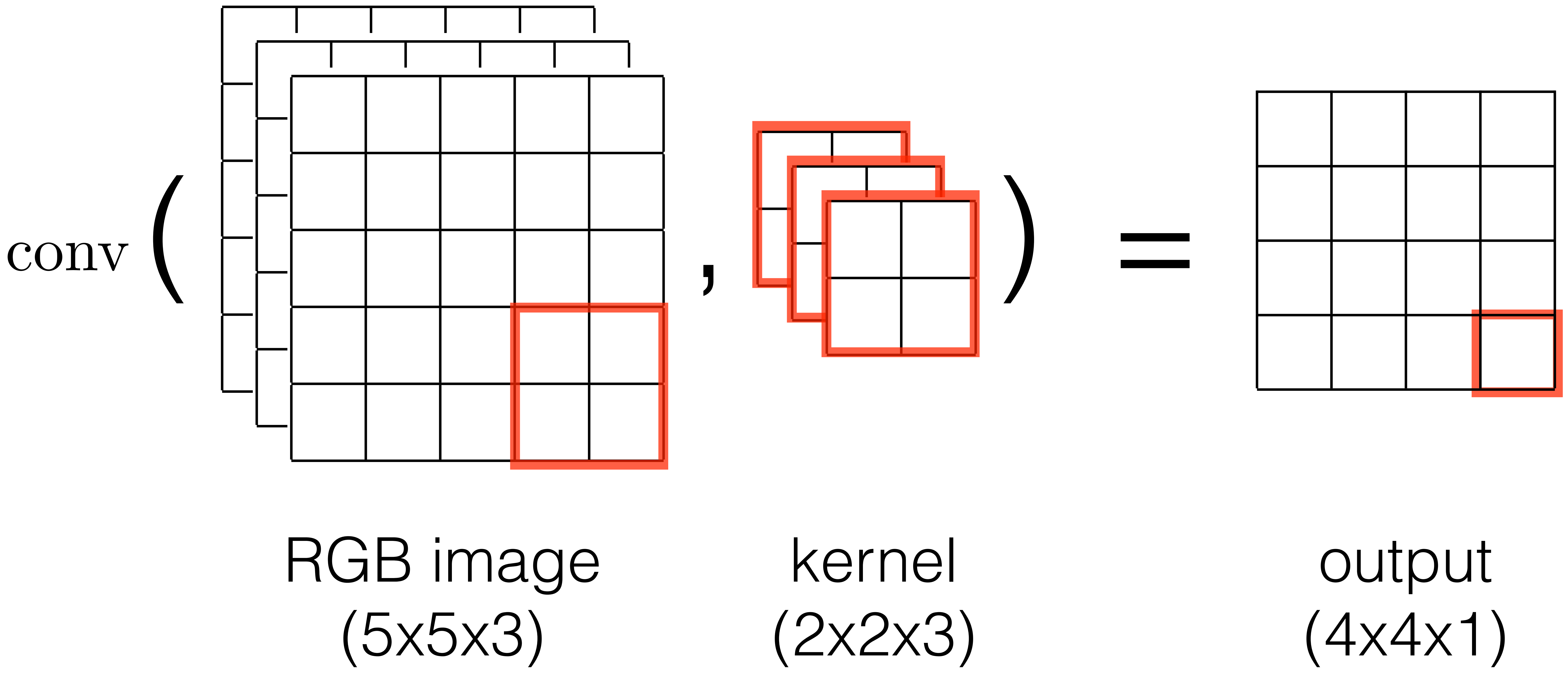


RGB image
(5x5x3)

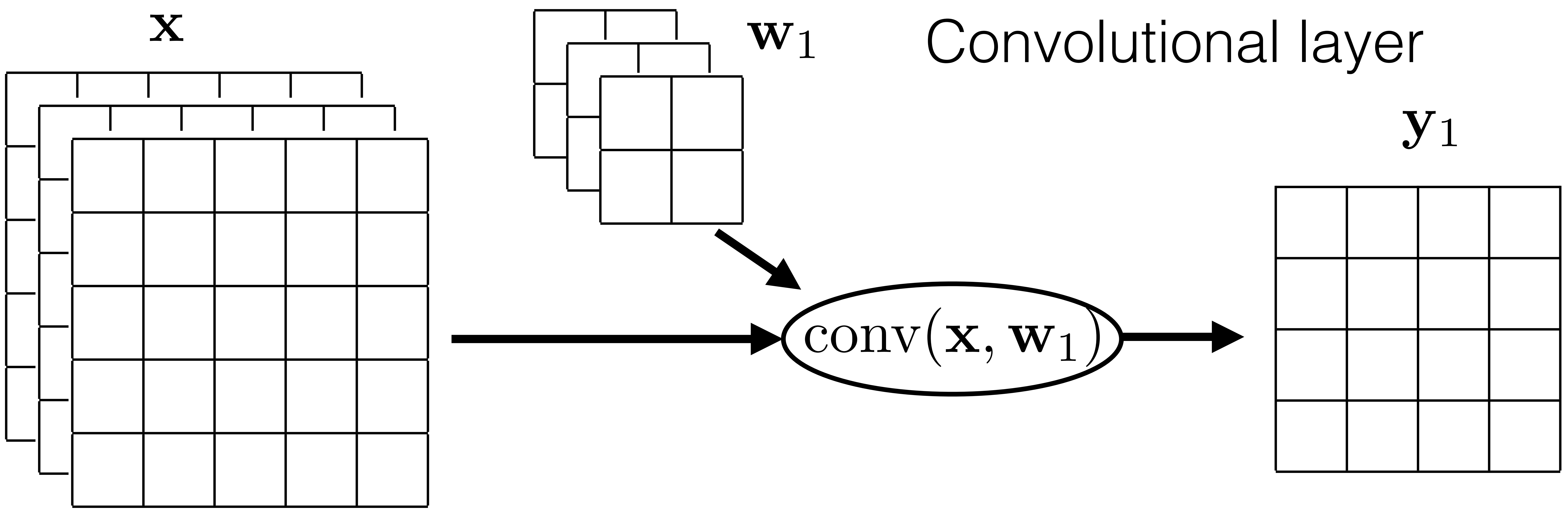
kernel
(2x2x3)

output
(4x4x1)

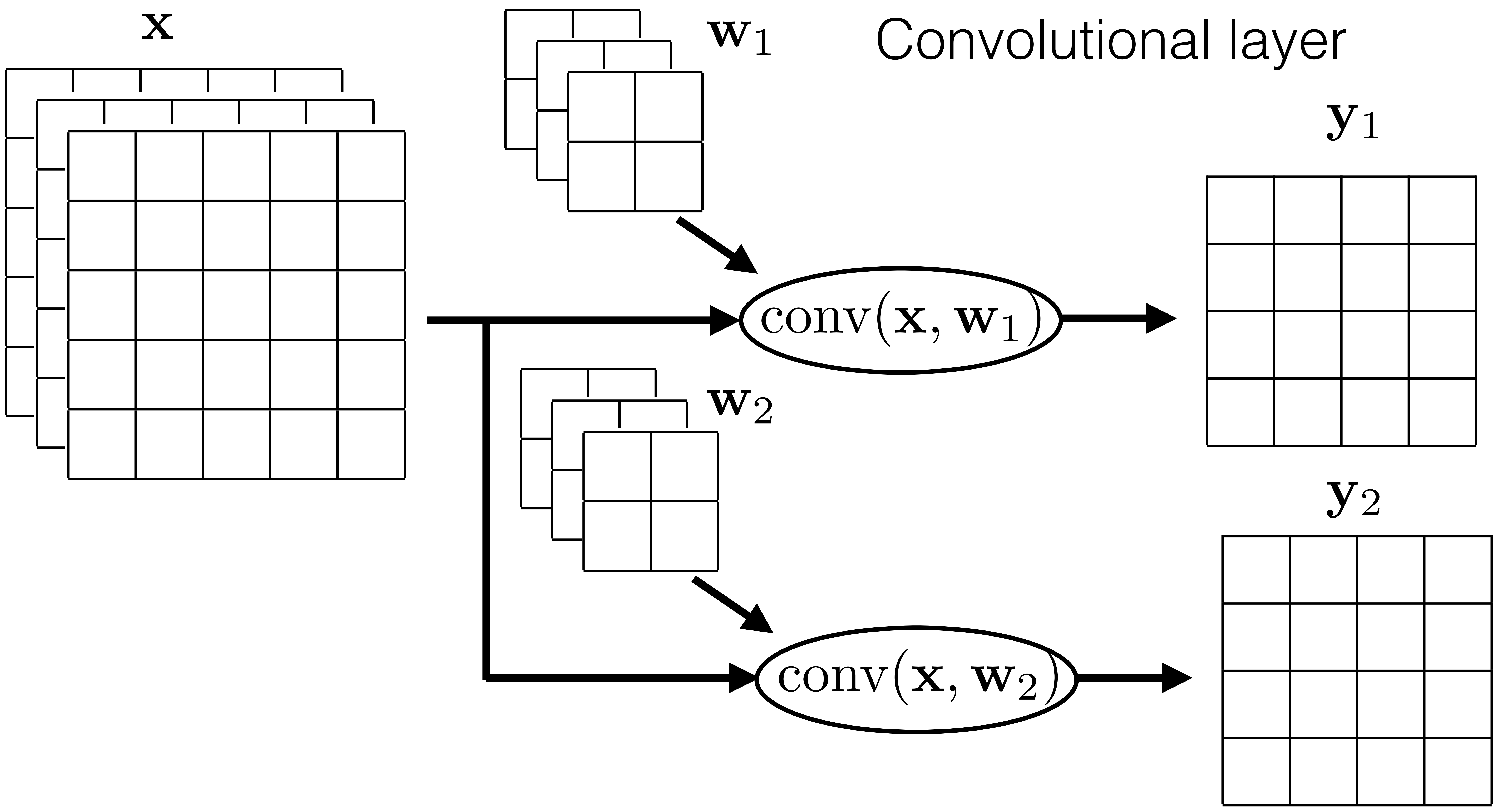
Multi-channel convolution



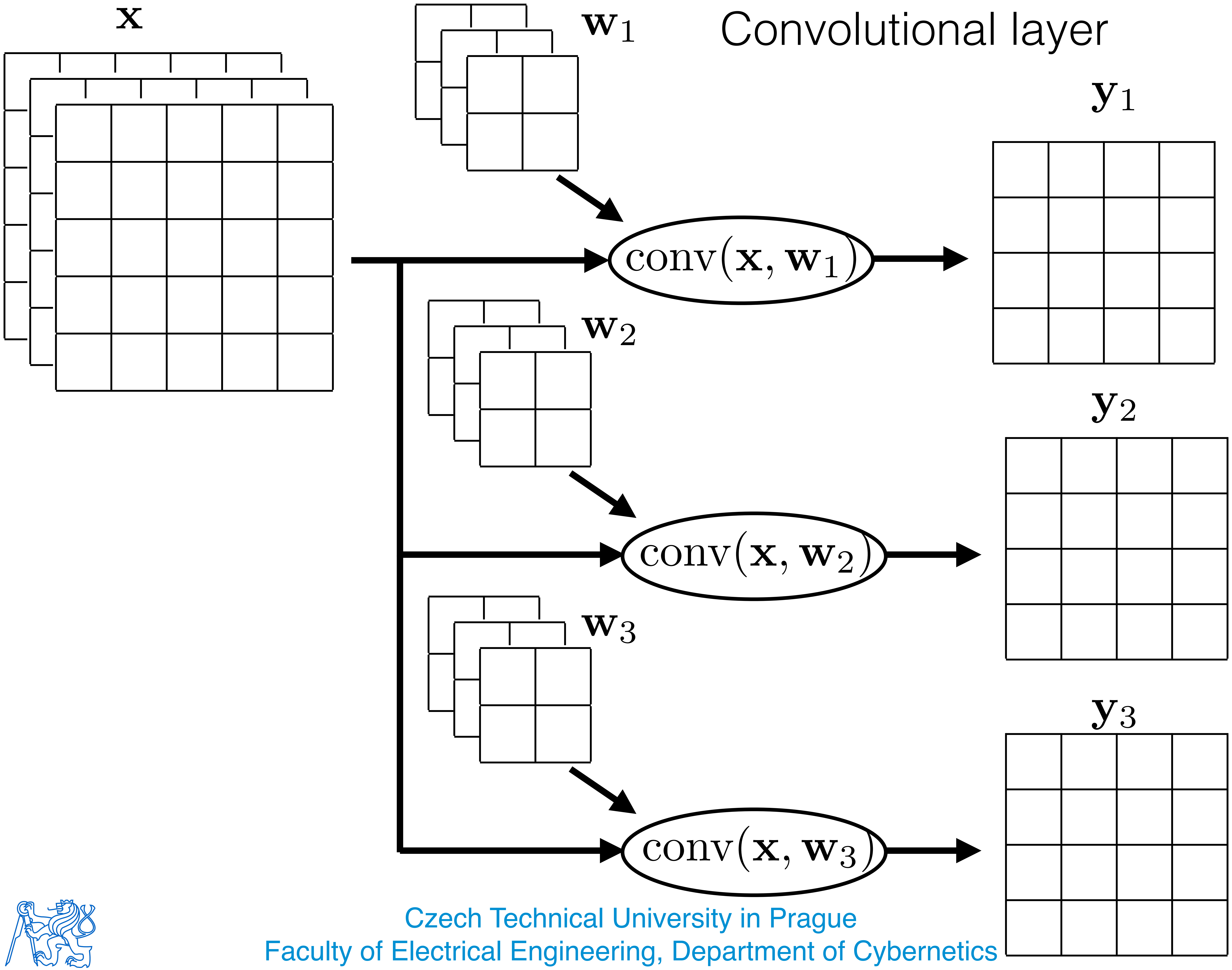
Convolutional layer



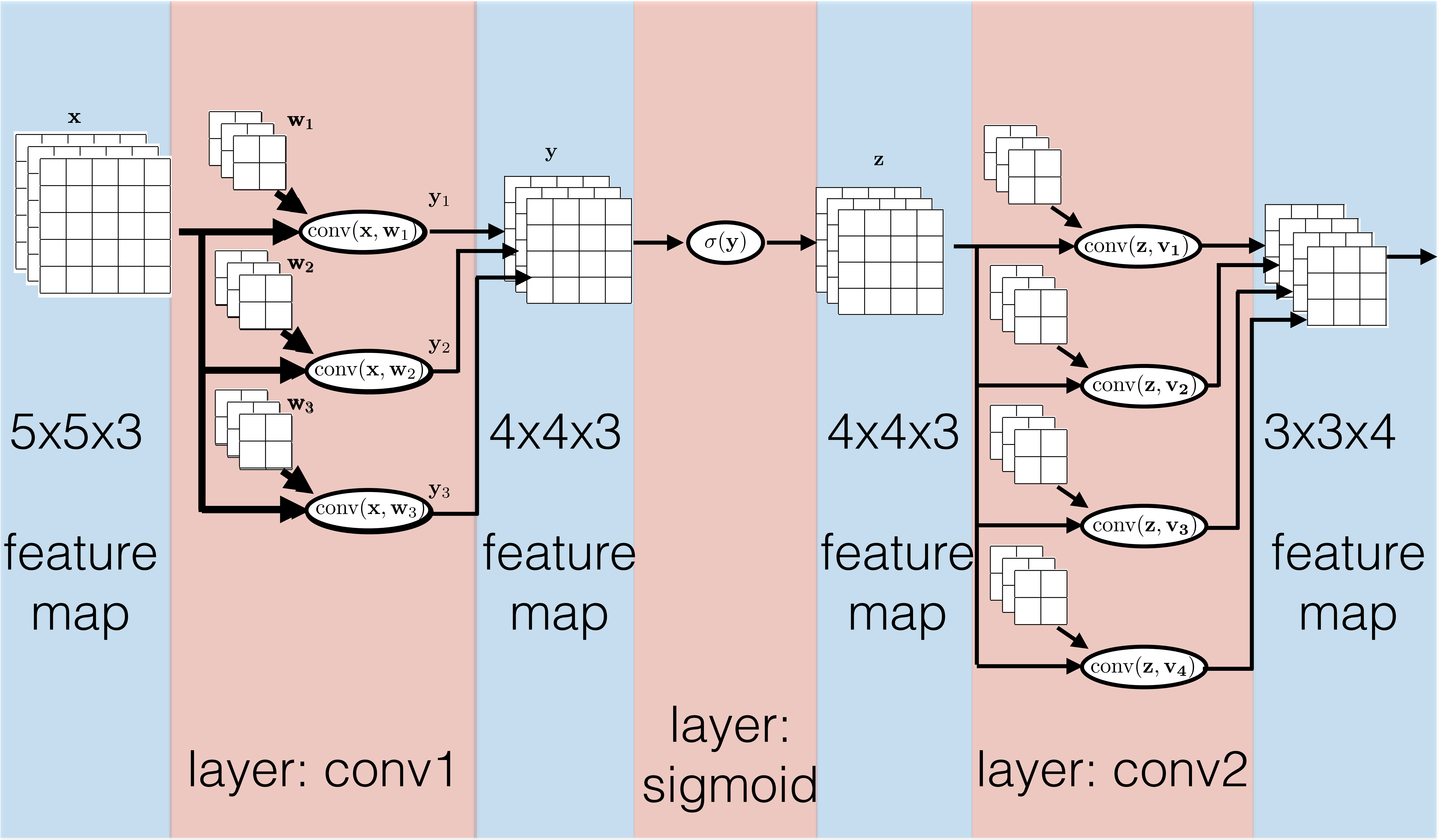
Convolutional layer



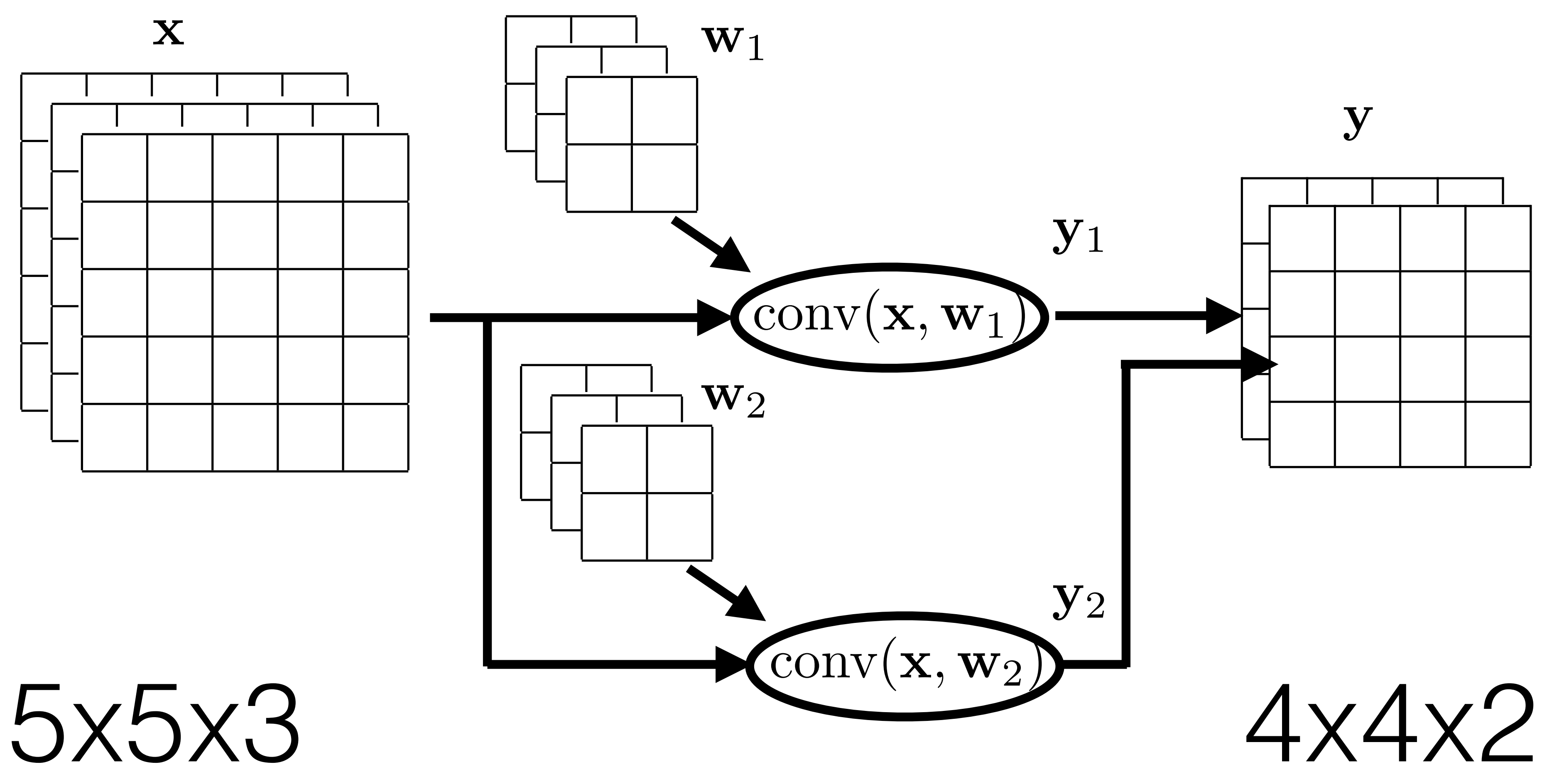
Convolutional layer



Convolutional network (ConvNet)

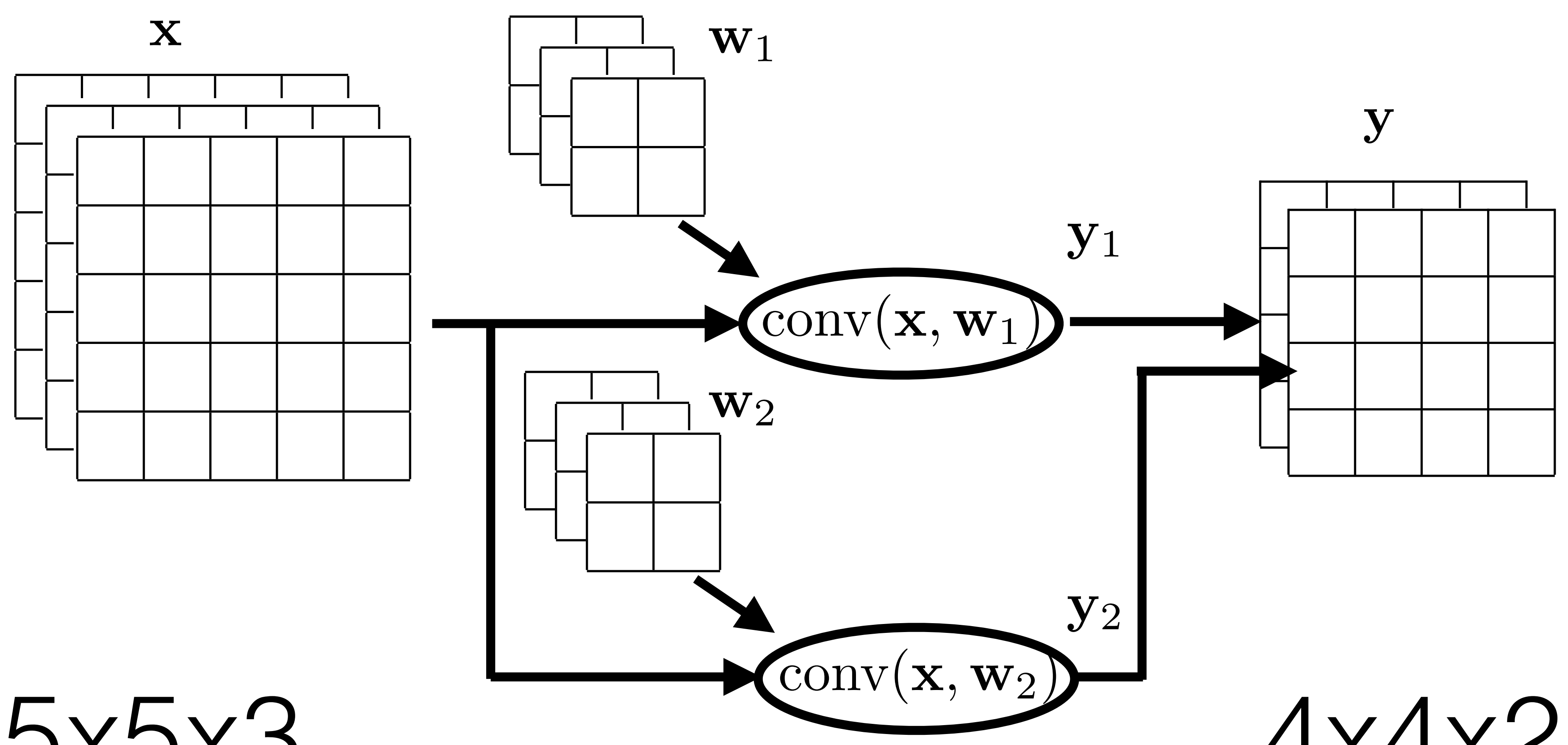


2D convolution forward pass




```python
# initialise
import torch.nn as nn
# define 2D convolutional layer
first_layer = nn.Conv2d(in_channels=3, out_channels=2,
                        kernel_size=2, stride=1,
                        padding=1)
```

2D convolution forward pass



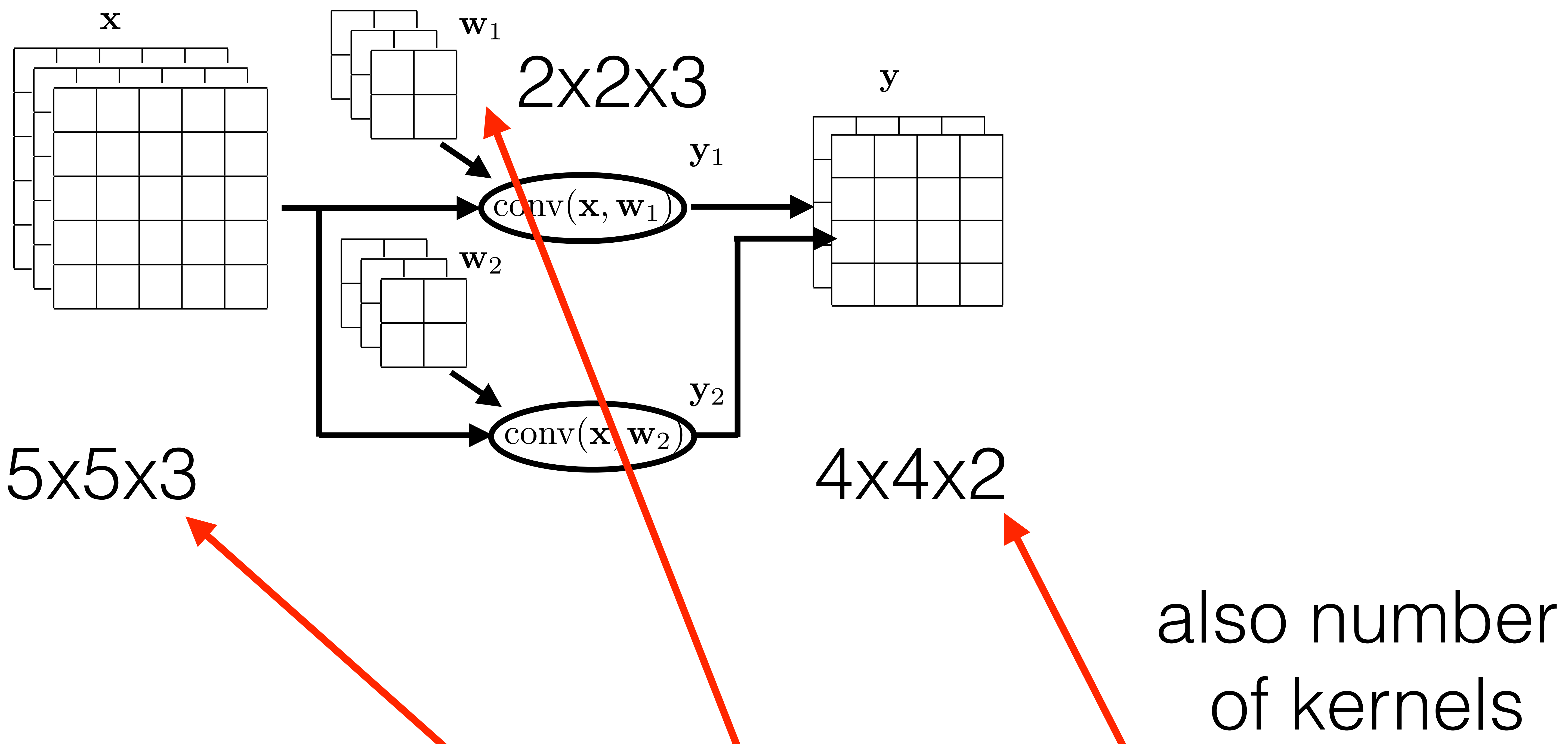
5x5x3

4x4x2

also number of kernels

```
# initialise
import torch.nn as nn
# define 2D convolutional layer
first_layer = nn.Conv2d(in_channels=3, out_channels=3,
                        kernel_size=2, stride=1,
                        padding=1)
```

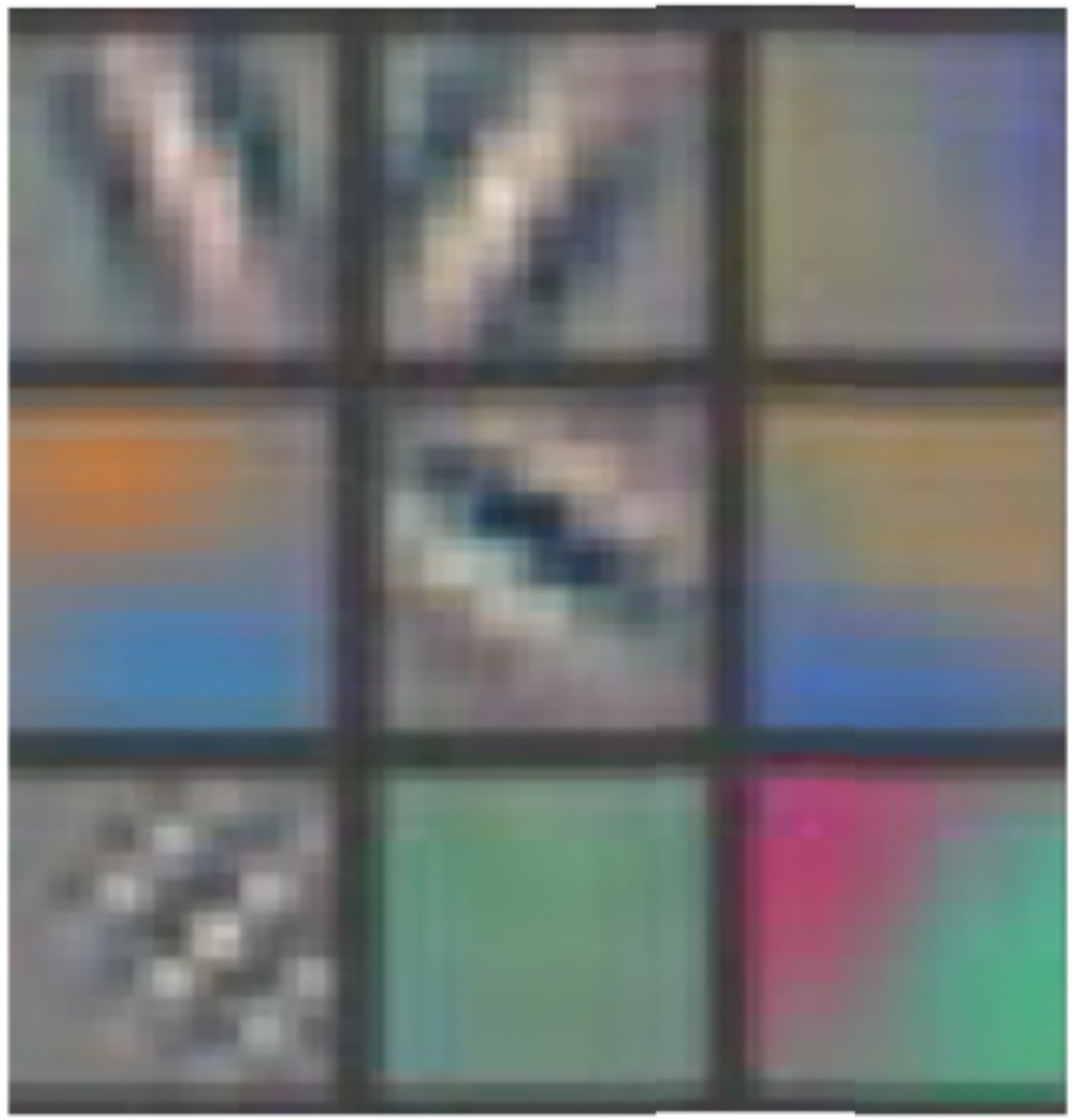
2D convolution forward pass




```
# initialise
import torch.nn as nn
# define 2D convolutional layer
first_layer = nn.Conv2d(in_channels=3, out_channels=2,
                        kernel_size=2, stride=1,
                        padding=1)
```

3. Neurons are sensitive to edges and its orientation

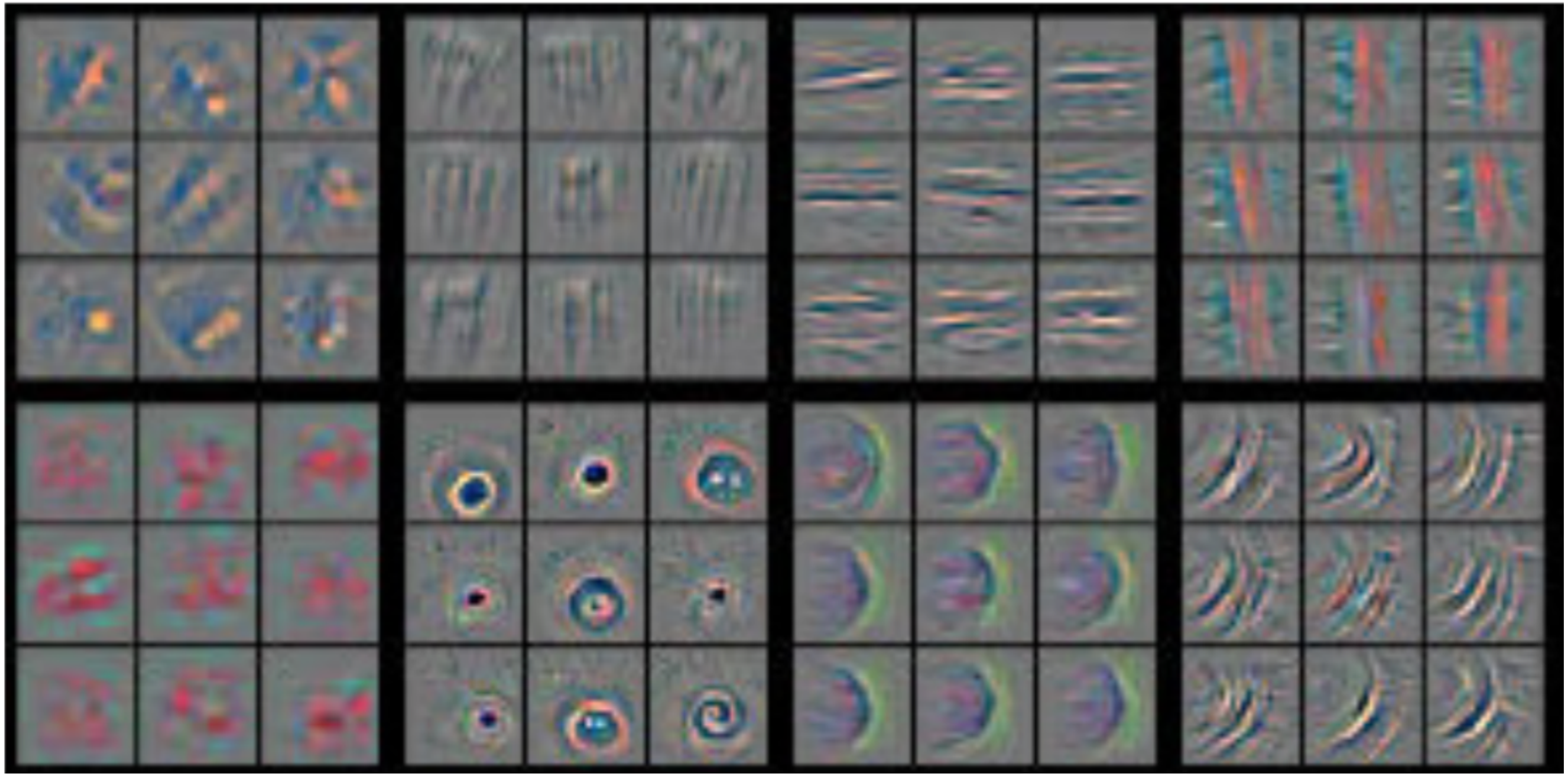
Inputs which maximized output of **layer 1**



[Zeiler and Fergus, ECCV, 2014]

3. Neurons are sensitive to edges and its orientation

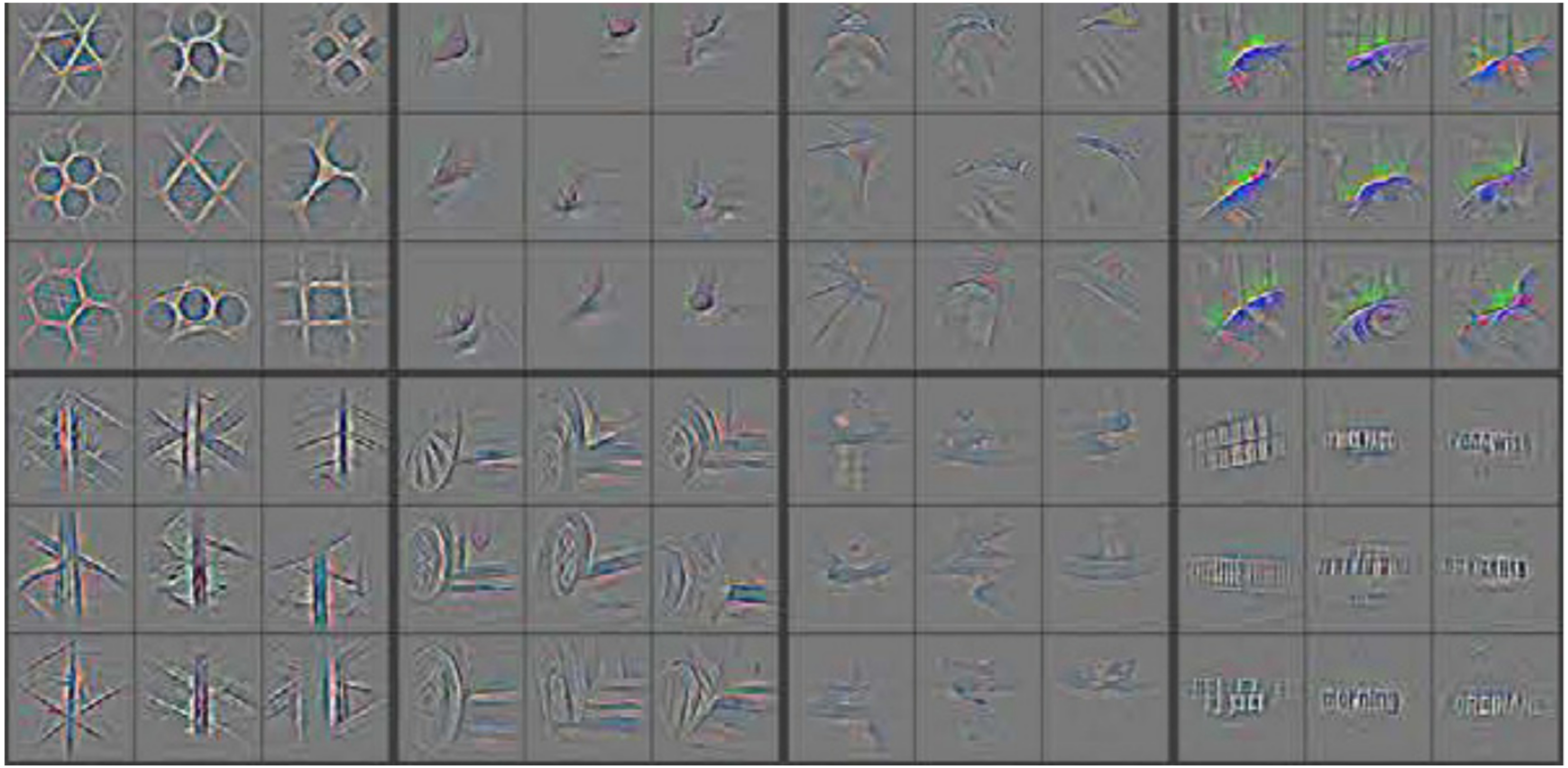
Inputs which maximized output of **layer 2**



[Zeiler and Fergus, ECCV, 2014]

3. Neurons are sensitive to edges and its orientation

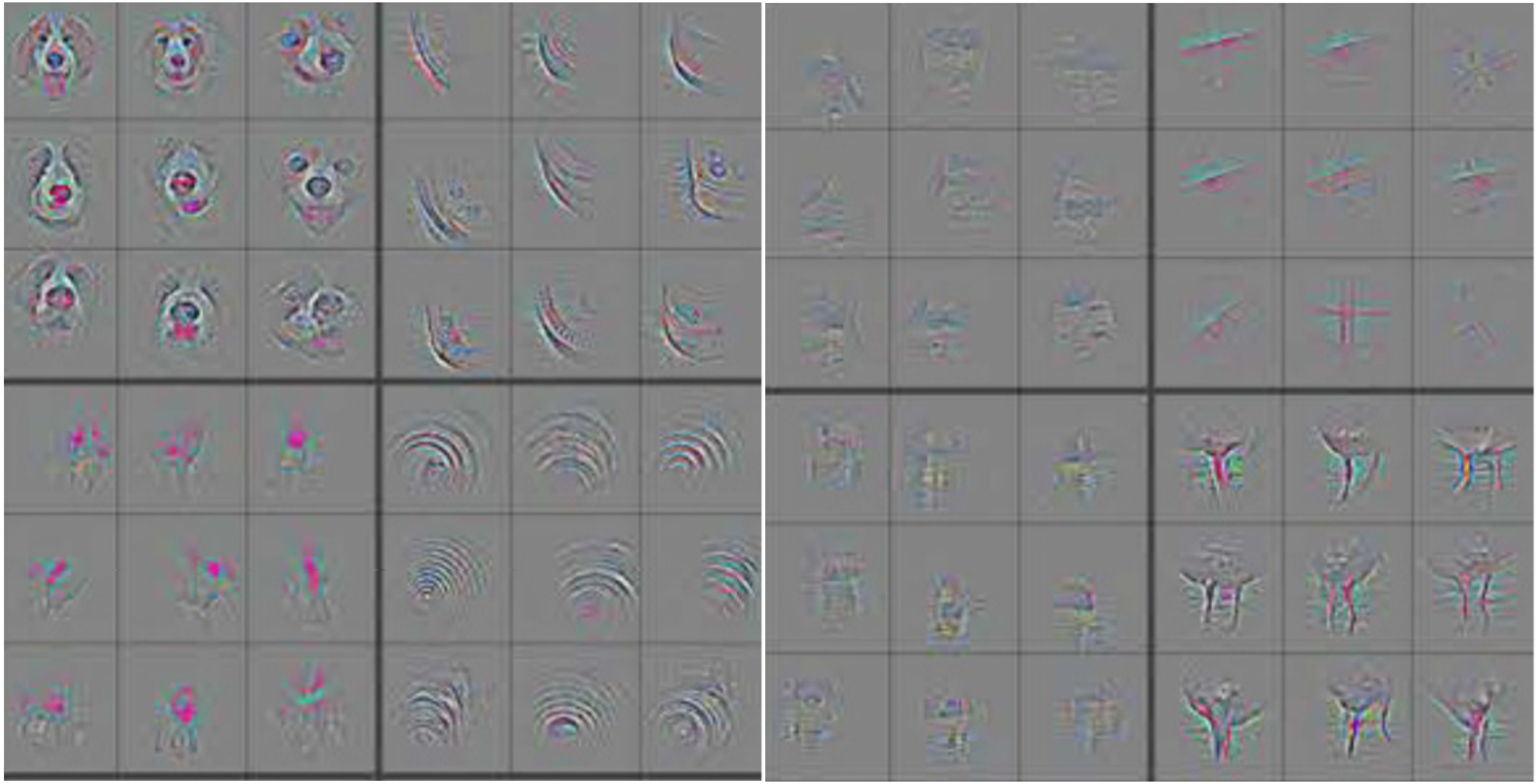
Inputs which maximized output of **layer 3**



[Zeiler and Fergus, ECCV, 2014]

3. Neurons are sensitive to edges and its orientation

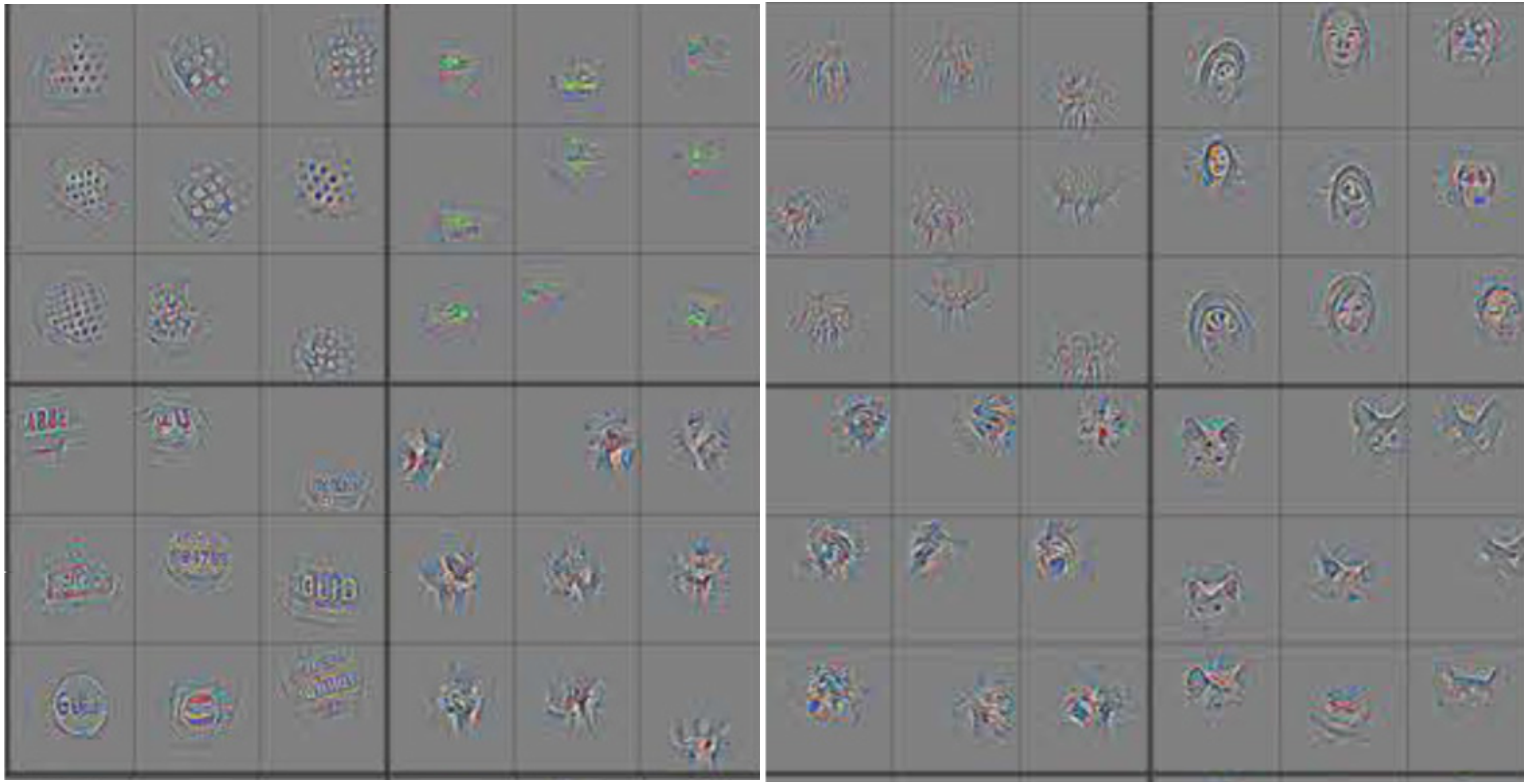
Inputs which maximized output of **layer 4**



[Zeiler and Fergus, ECCV, 2014]

3. Neurons are sensitive to edges and its orientation

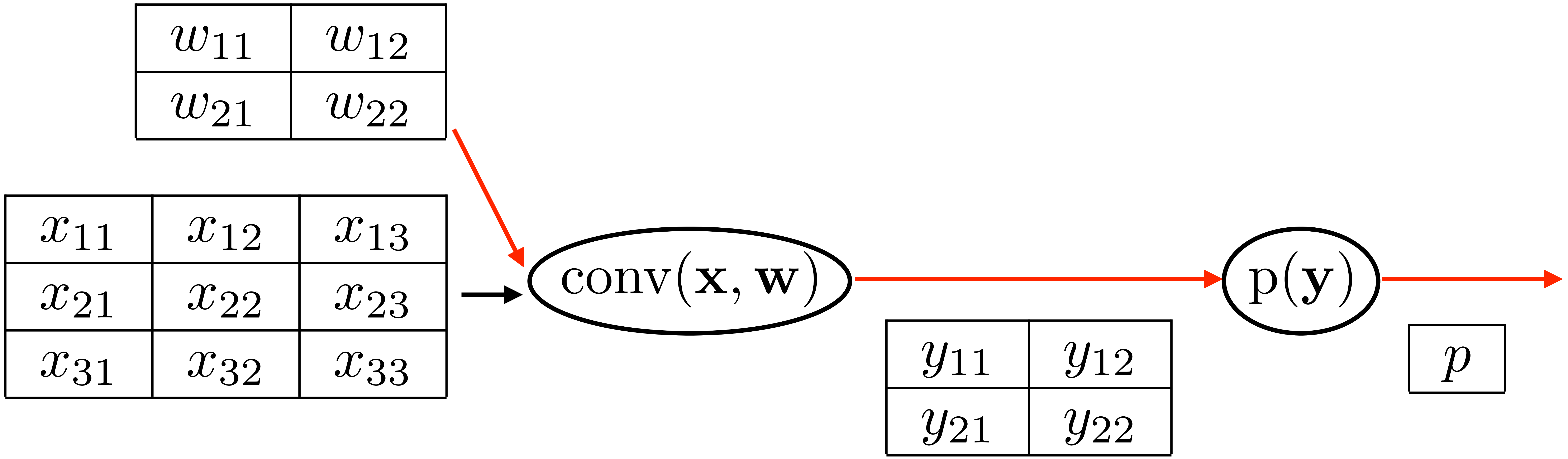
Inputs which maximized output of **layer 5**



[Zeiler and Fergus, ECCV, 2014]

Convolution backward pass

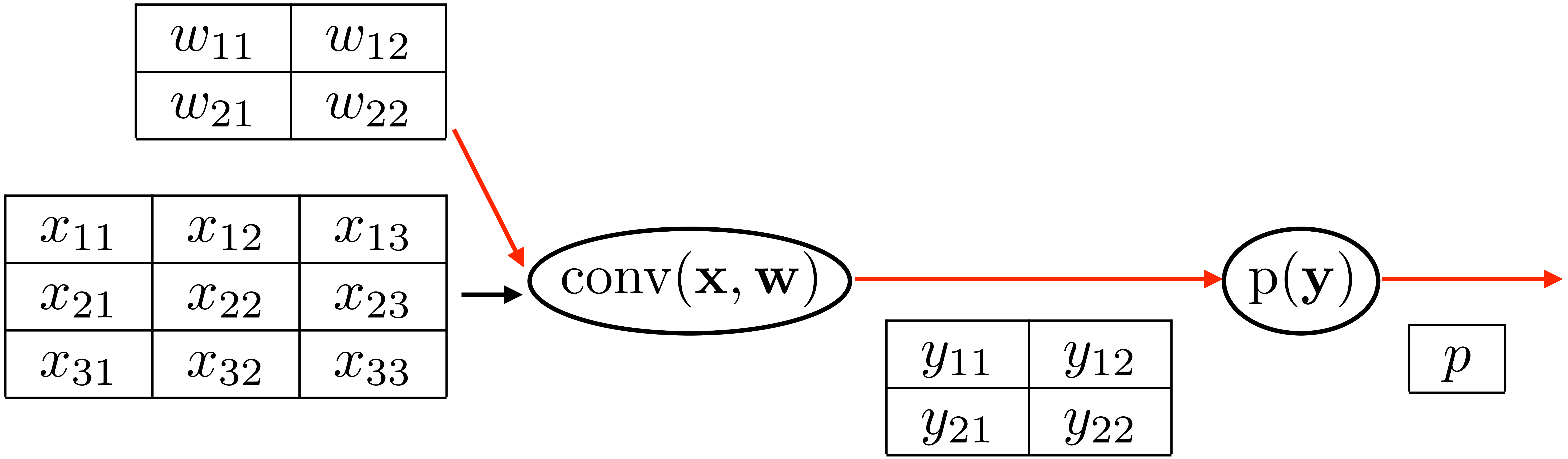
Learning of convolutional neuron => backpropagation



Convolution backward pass

$\frac{\partial p}{\partial w_{11}}$	$\frac{\partial p}{\partial w_{12}}$
$\frac{\partial p}{\partial w_{21}}$	$\frac{\partial p}{\partial w_{22}}$

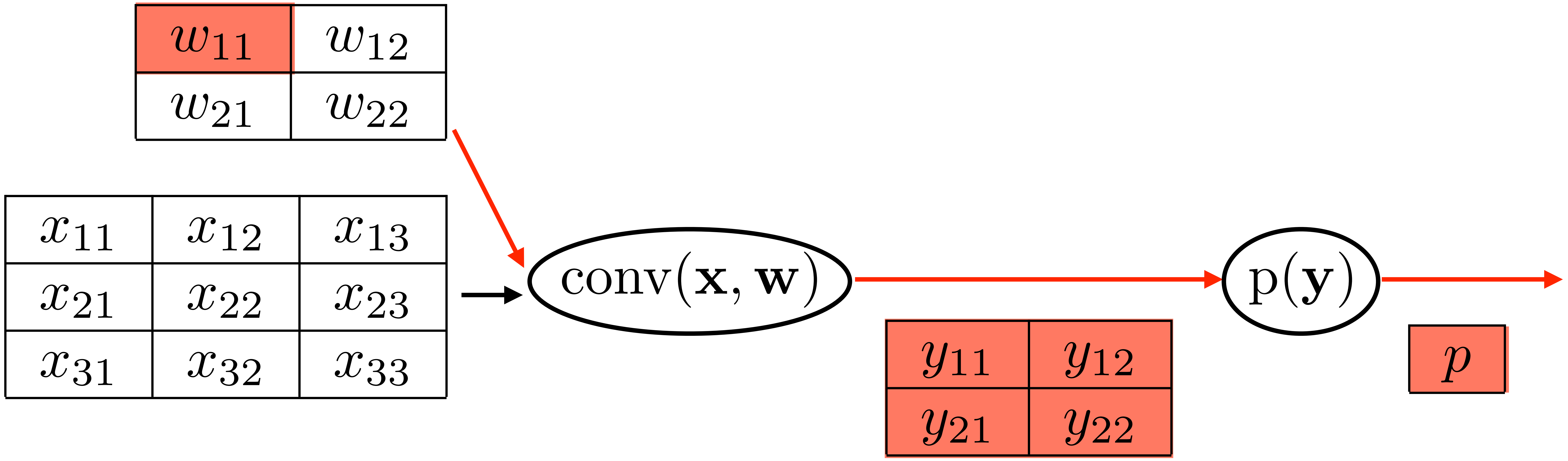
=?



Convolution backward pass

$$\begin{bmatrix} \frac{\partial p}{\partial w_{11}} & \frac{\partial p}{\partial w_{12}} \\ \frac{\partial p}{\partial w_{21}} & \frac{\partial p}{\partial w_{22}} \end{bmatrix} = ?$$

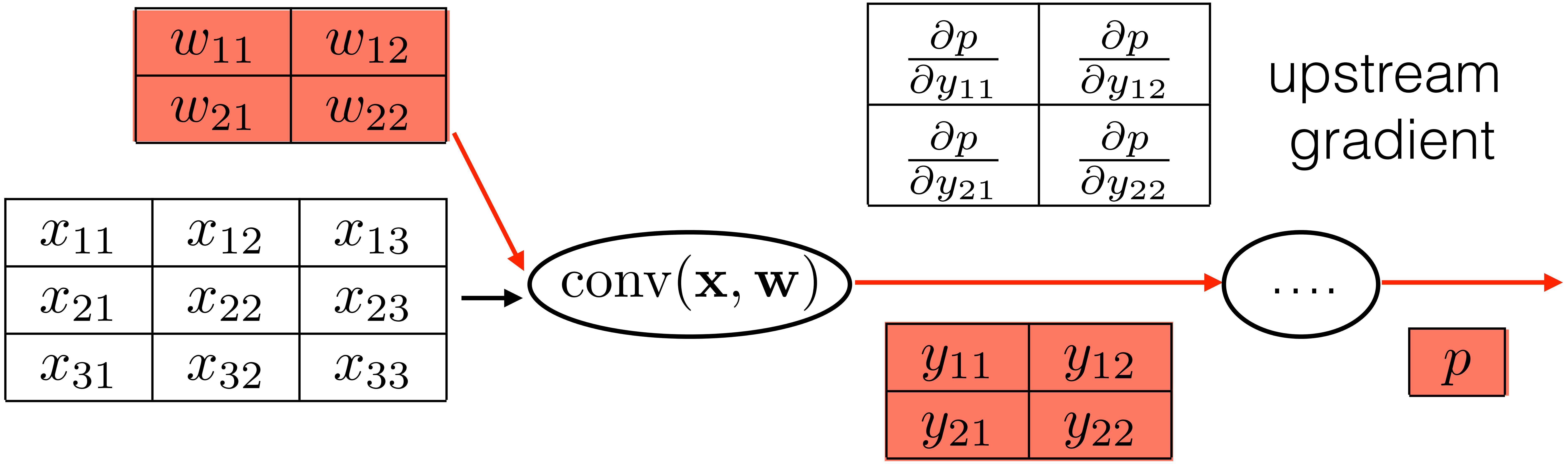
$$p(w_{11}) = \mathrm{p}(y_{11}(w_{11}),\ y_{12}(w_{11}),\ y_{21}(w_{11}),\ y_{22}(w_{11}))$$



Convolution backward pass wrt weights

- Backpropagation in convolutional layer wrt weights is:
  "**convolution of input feature map with upstream gradient**"

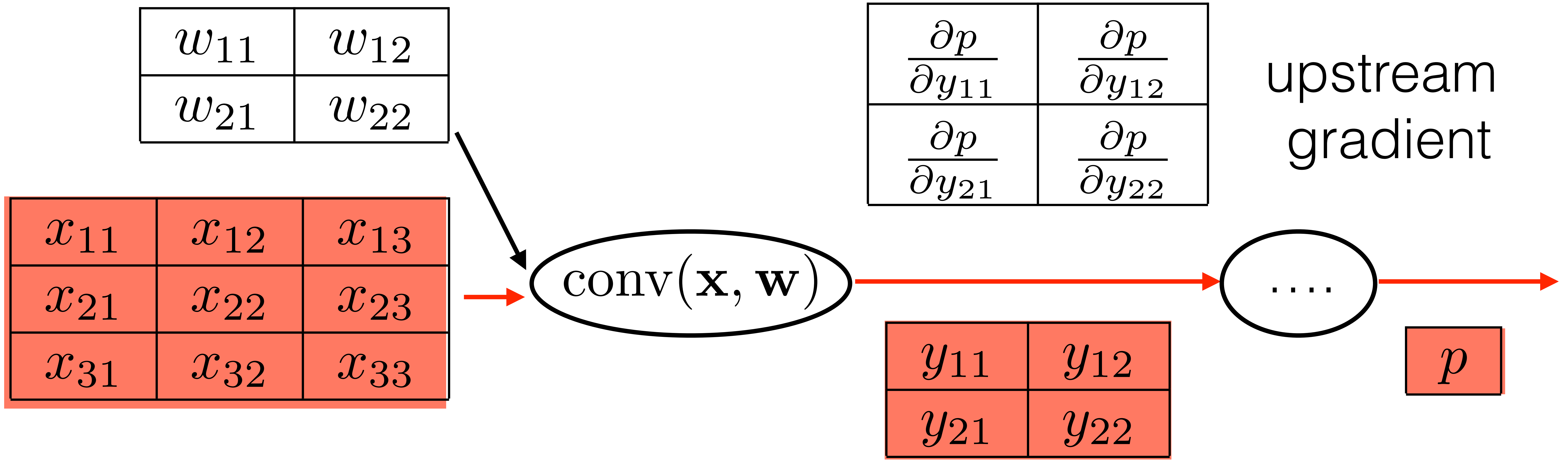
$$\begin{array}{|c|c|}\hline \frac{\partial p}{\partial w_{11}} & \frac{\partial p}{\partial w_{12}} \\ \hline \frac{\partial p}{\partial w_{21}} & \frac{\partial p}{\partial w_{22}} \\ \hline \end{array} = \mathrm{conv}\left(\begin{array}{|c|c|c|}\hline x_{11} & x_{12} & x_{13} \\ \hline x_{21} & x_{22} & x_{23} \\ \hline x_{31} & x_{32} & x_{33} \\ \hline \end{array}, \begin{array}{|c|c|}\hline \frac{\partial p}{\partial y_{11}} & \frac{\partial p}{\partial y_{12}} \\ \hline \frac{\partial p}{\partial y_{21}} & \frac{\partial p}{\partial y_{22}} \\ \hline \end{array} \right)$$



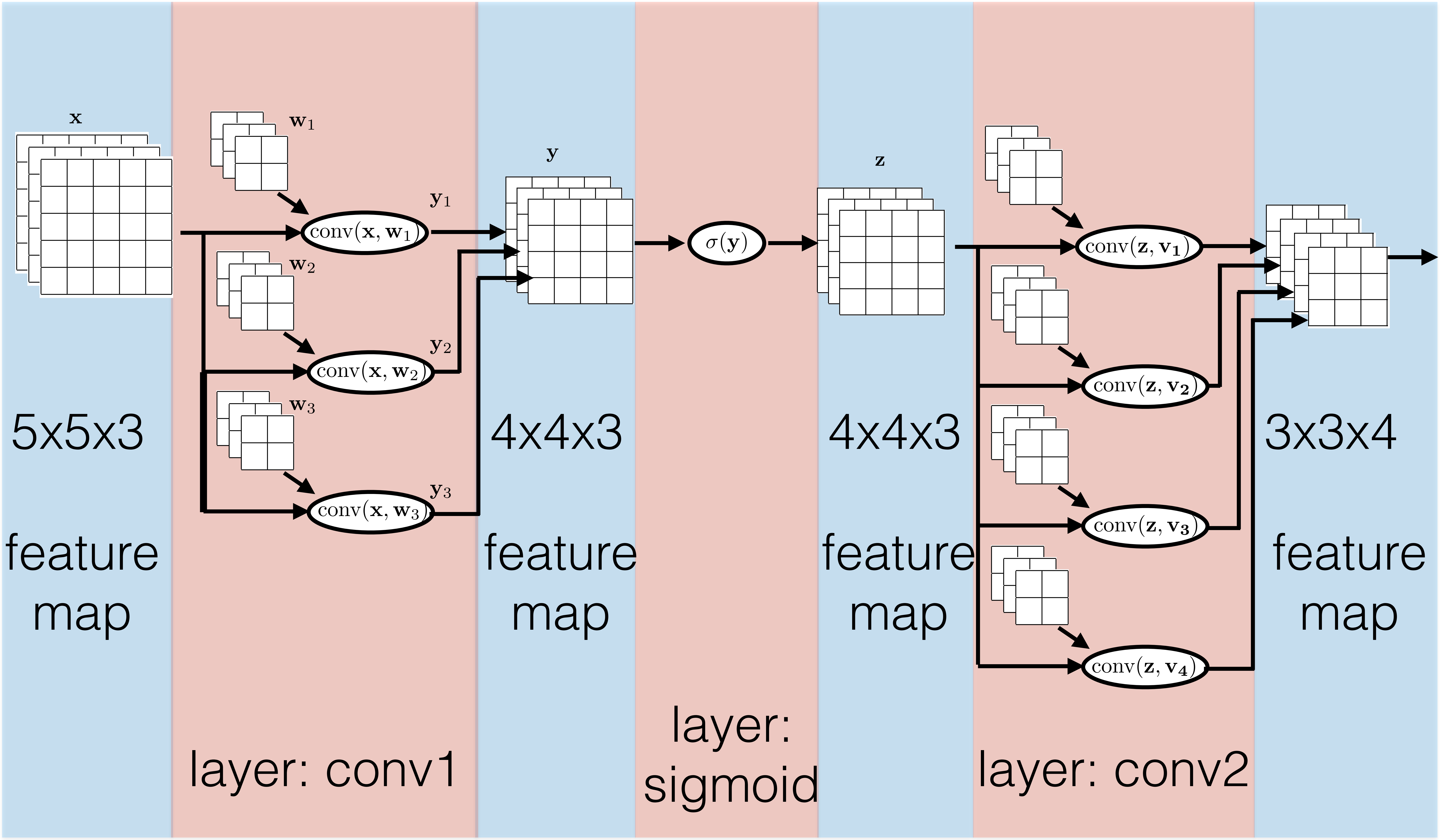
Convolution backward pass wrt input feature map

- Backpropagation in convolutional layer is:
  “**convolution of padded upstream gradient with mirrored weights**”

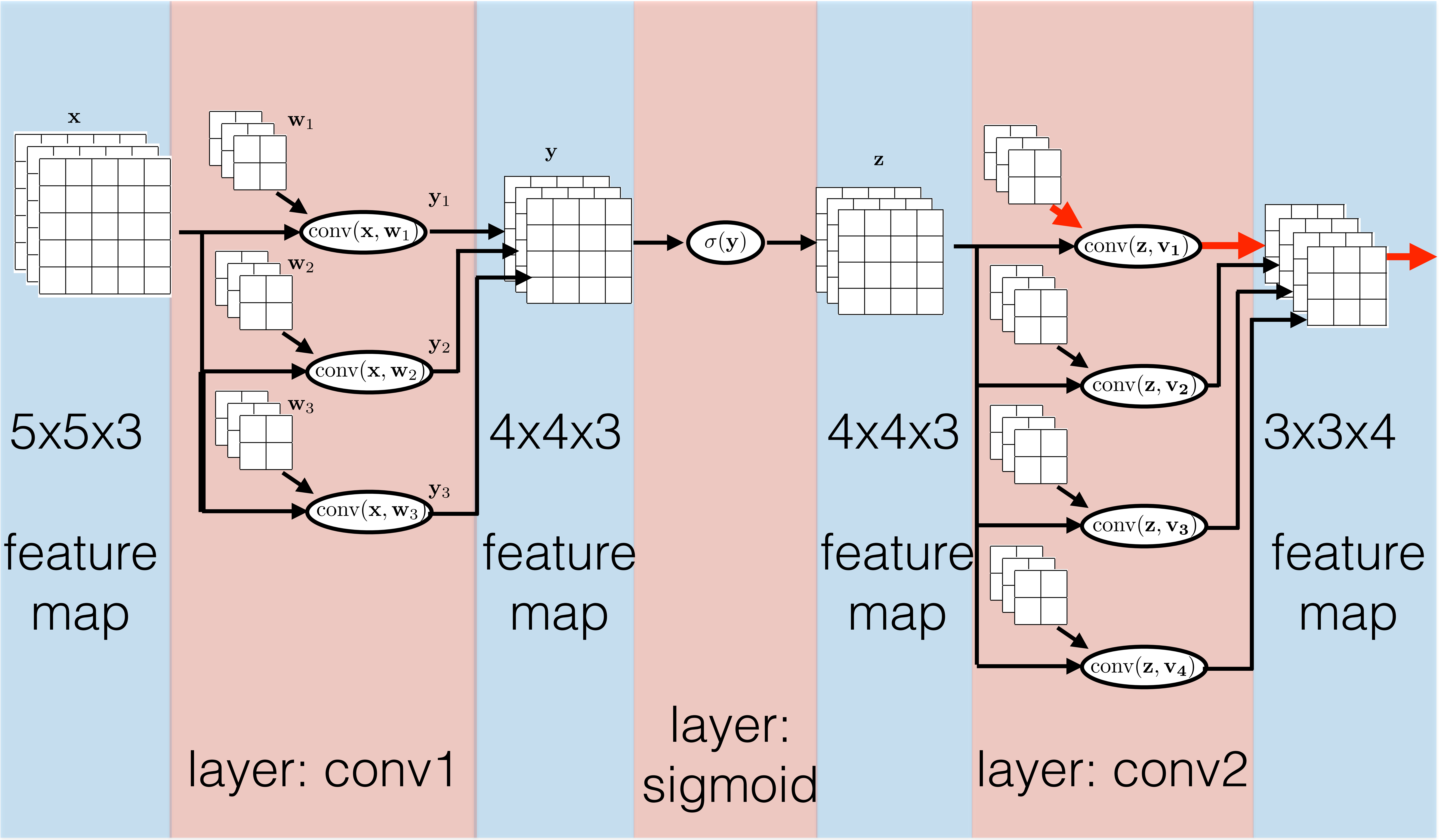
$$\begin{bmatrix} \frac{\partial p}{\partial x_{11}} & \frac{\partial p}{\partial x_{12}} & \frac{\partial p}{\partial x_{13}} \\ \frac{\partial p}{\partial x_{21}} & \frac{\partial p}{\partial x_{22}} & \frac{\partial p}{\partial x_{23}} \\ \frac{\partial p}{\partial x_{31}} & \frac{\partial p}{\partial x_{32}} & \frac{\partial p}{\partial x_{33}} \end{bmatrix} = \mathrm{conv}\left(\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{\partial p}{\partial y_{11}} & \frac{\partial p}{\partial y_{12}} & 0 \\ 0 & \frac{\partial p}{\partial y_{21}} & \frac{\partial p}{\partial y_{22}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix} \right)$$



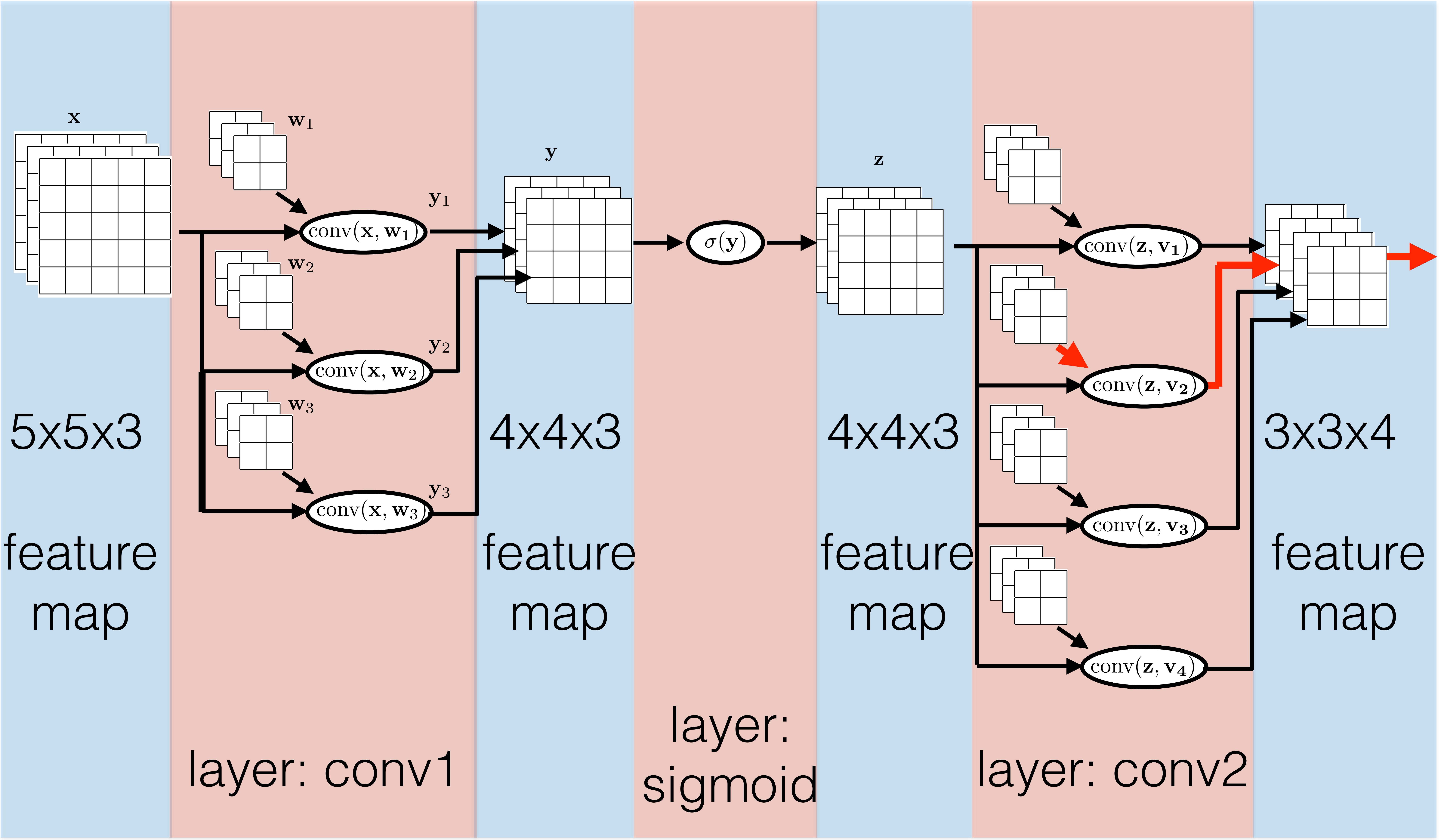
???? Convolutional network backprop ?????



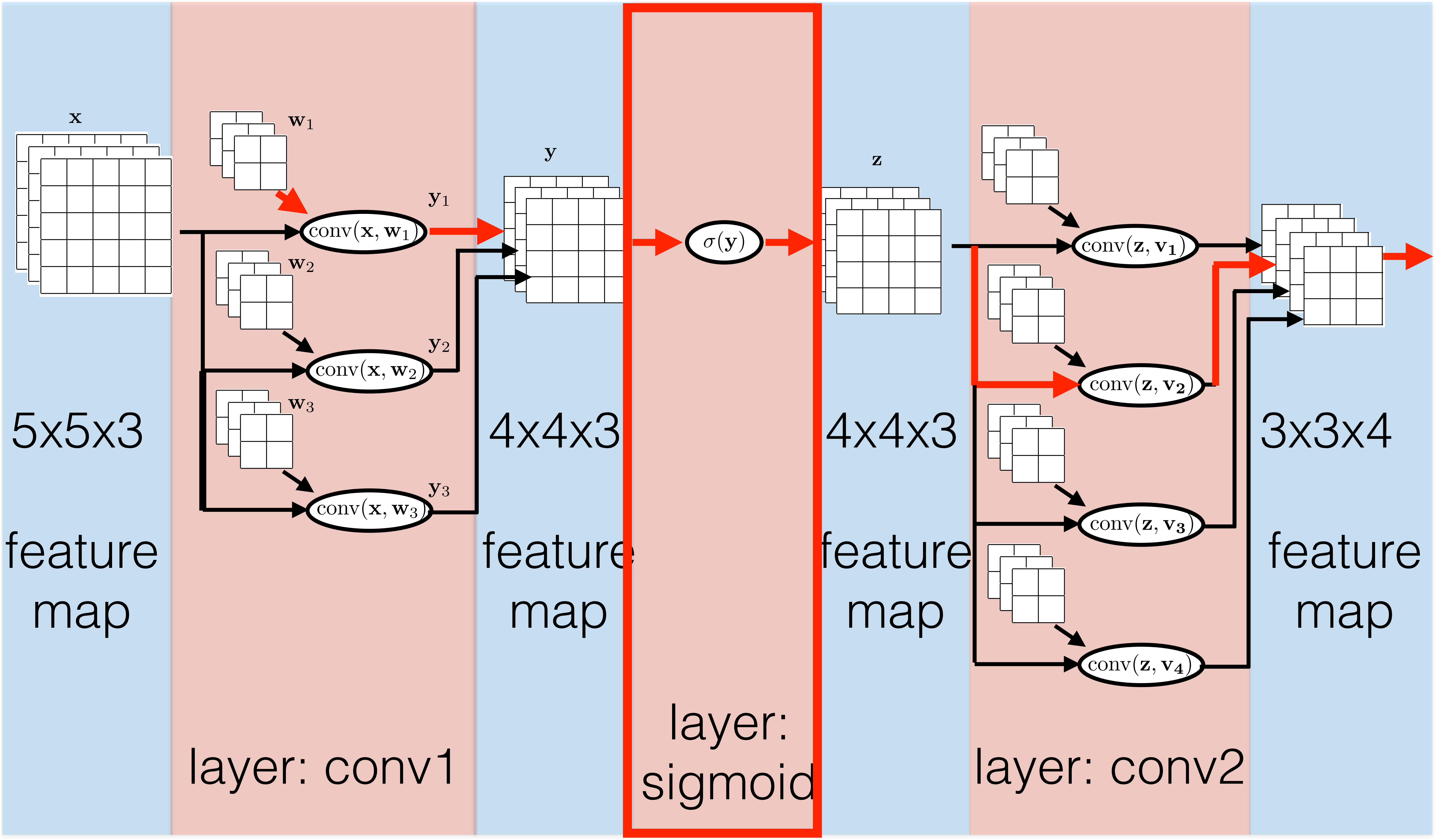
???? Convolutional network backprop ?????



???? Convolutional network backprop ?????



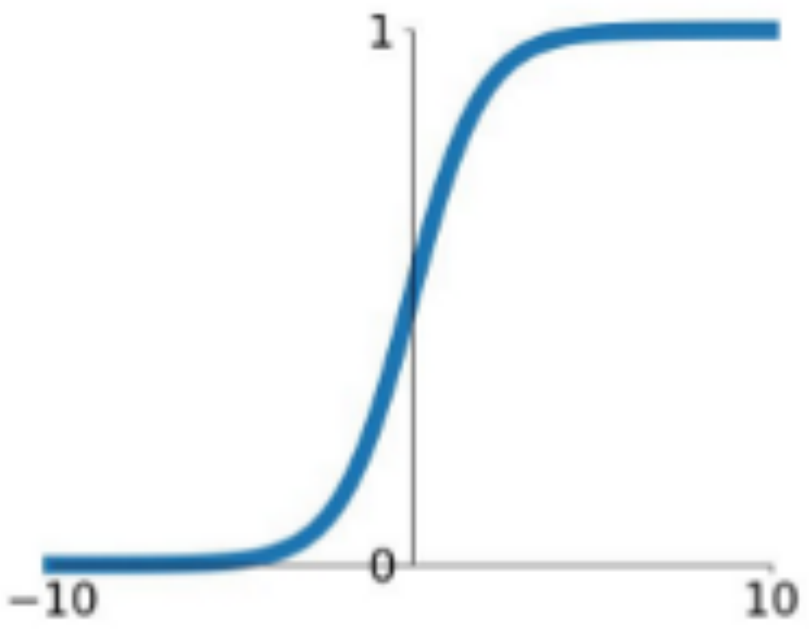
???? Convolutional network backprop ?????



Activation functions

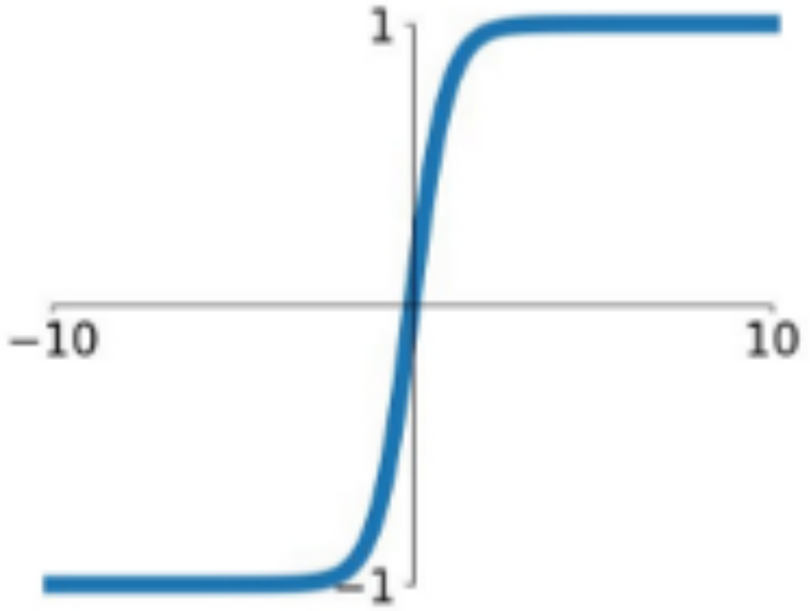
Sigmoid

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



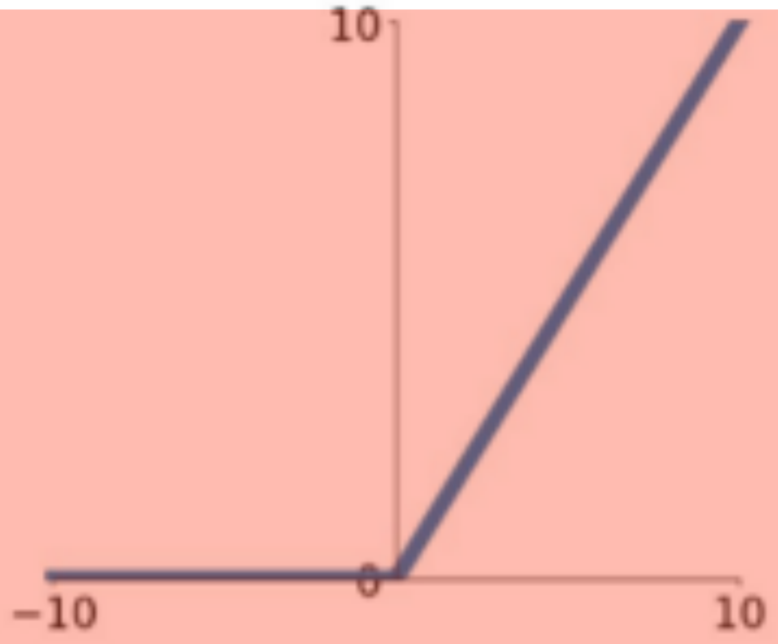
tanh

$\tanh(x)$



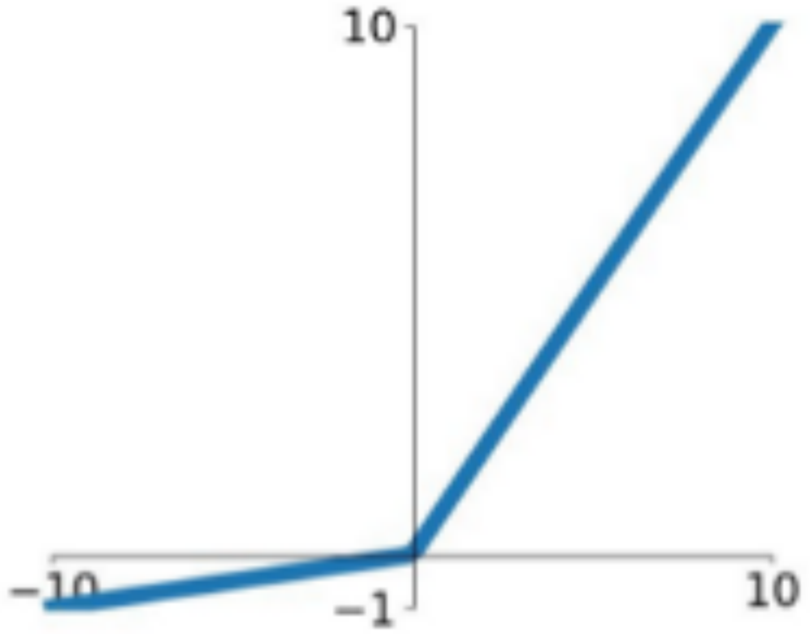
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

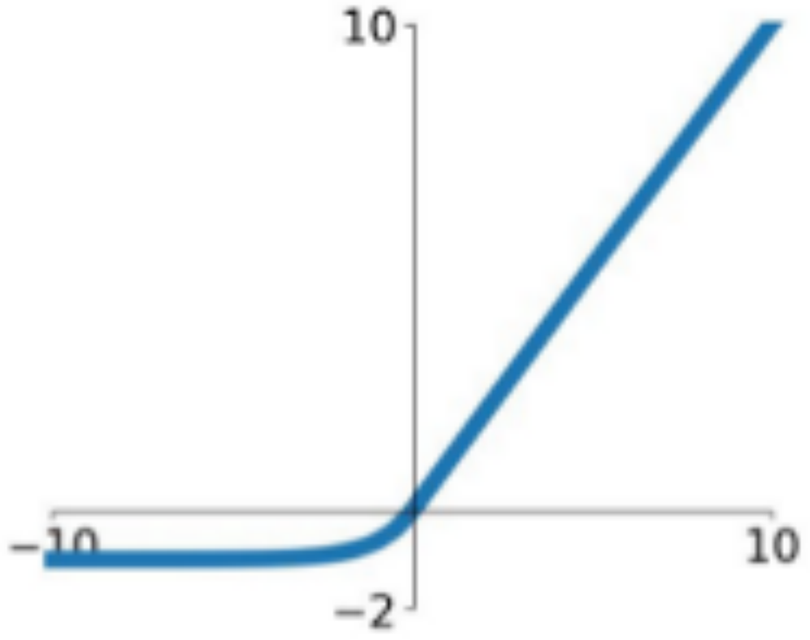


Maxout

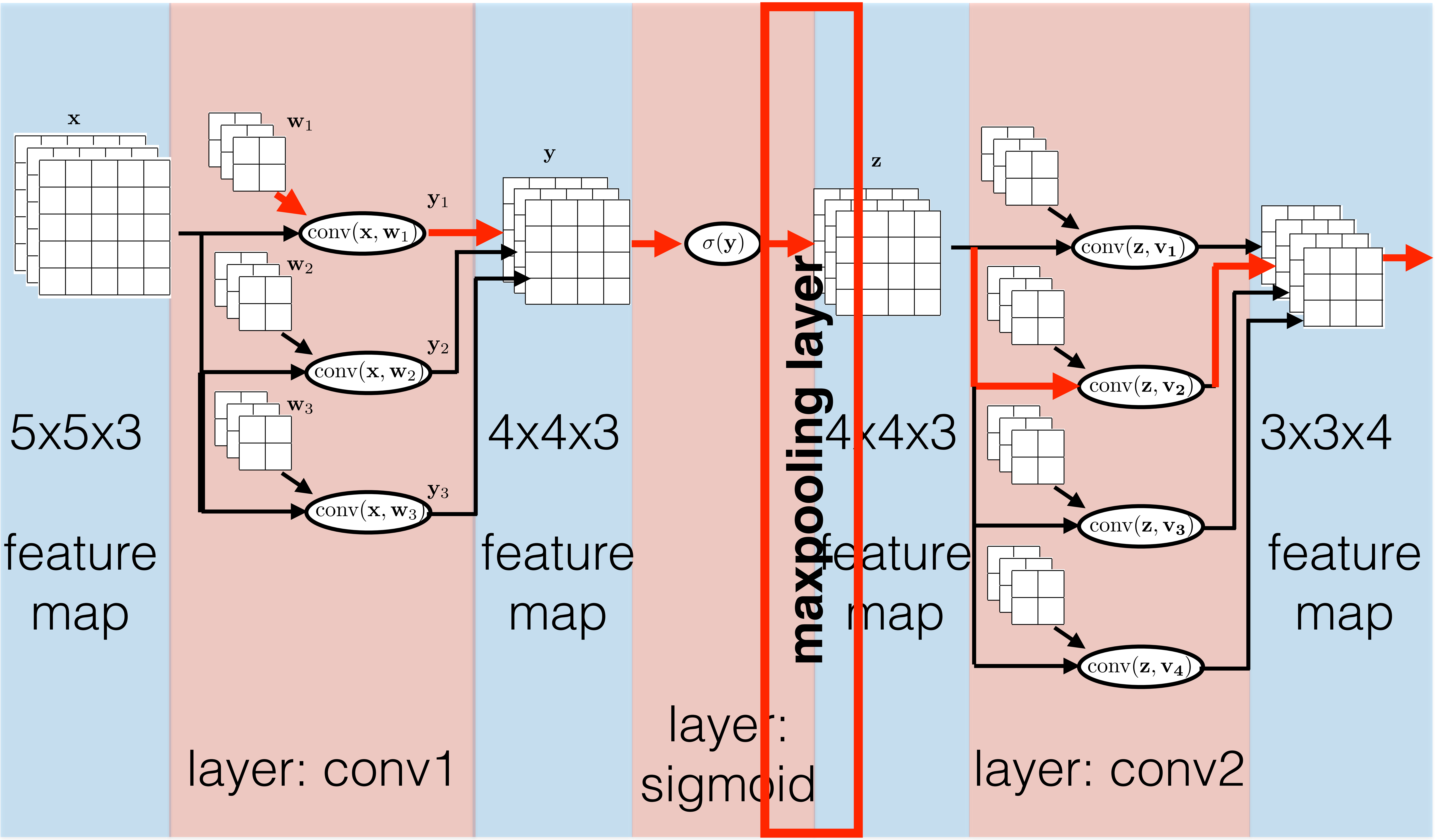
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



???? Convolutional network backprop ?????



Max-pooling

$$\max\left(\begin{array}{|c|c|c|c|c|}\hline 1 & 3 & 0 & 2 & 1 \\ \hline 2 & 0 & 1 & 4 & 0 \\ \hline 0 & 3 & 1 & 3 & 2 \\ \hline 1 & 3 & 0 & 2 & 1 \\ \hline 2 & 0 & 1 & 2 & 0 \\ \hline \end{array},\quad 2\text{x}2\right) = \boxed{3}$$

image
(5x5)

output
(**? x ?**)

Max-pooling

$$\max\left(\begin{array}{|c|c|c|c|c|}\hline 1&3&0&2&1\\\hline 2&0&1&4&0\\\hline 0&3&1&3&2\\\hline 1&3&0&2&1\\\hline 2&0&1&2&0\\\hline\end{array},\quad 2\text{x}2\right) = \begin{array}{|c|c|}\hline 3&\boxed{4}\\\hline\end{array}$$

image
(5x5)

output
(**? x ?**)

Max-pooling

max (

1	3	0	2	1
2	0	1	4	0
0	3	1	3	2
1	3	0	2	1
2	0	1	2	0

, 2x2) =

3	4
3	3

image
(5x5)

output
(**? x ?**)

Convolutional net

- Convolutional network (ConvNet) is concatenation of convolutional layers
- Backprop in ConvNet is convolution of feature maps or kernels or feature-maps with the upstream gradient.
- Feed-forward and backprop are convolutions => efficient implementation on GPU

IMAGENET

Classification results

http://image-net.org/challenges/LSVRC/2017/index

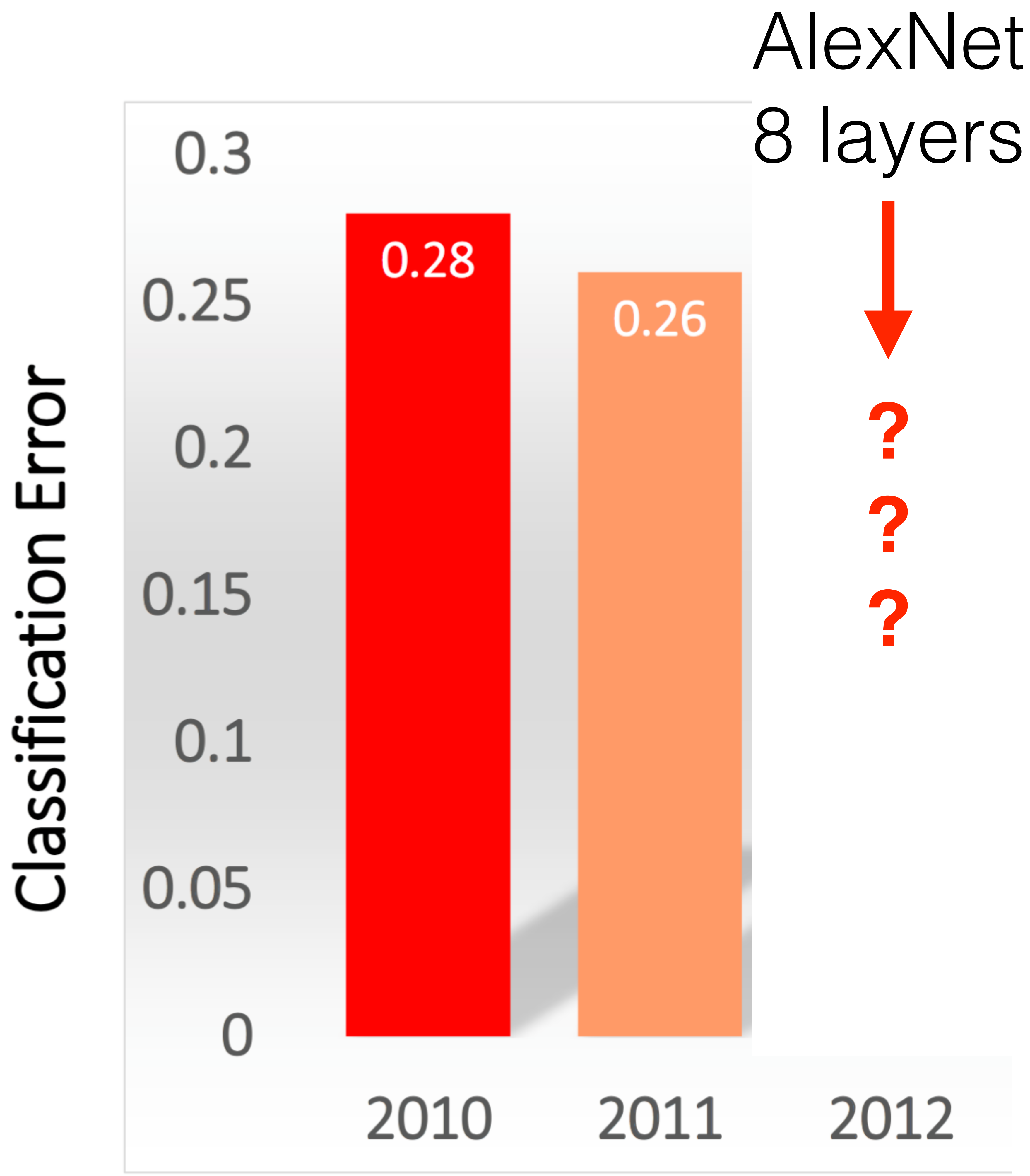
Steel drum

Output:		**Output:**	
Scale		Scale	
T-shirt	✓	T-shirt	✗
Steel drum		Giant panda	
Drumstick		Drumstick	
Mud turtle		Mud turtle	

$$\text{Error} = \frac{1}{100{,}000} \sum_{\substack{100{,}000 \\ \text{images}}} 1[\text{incorrect on image i}]$$

IMAGENET

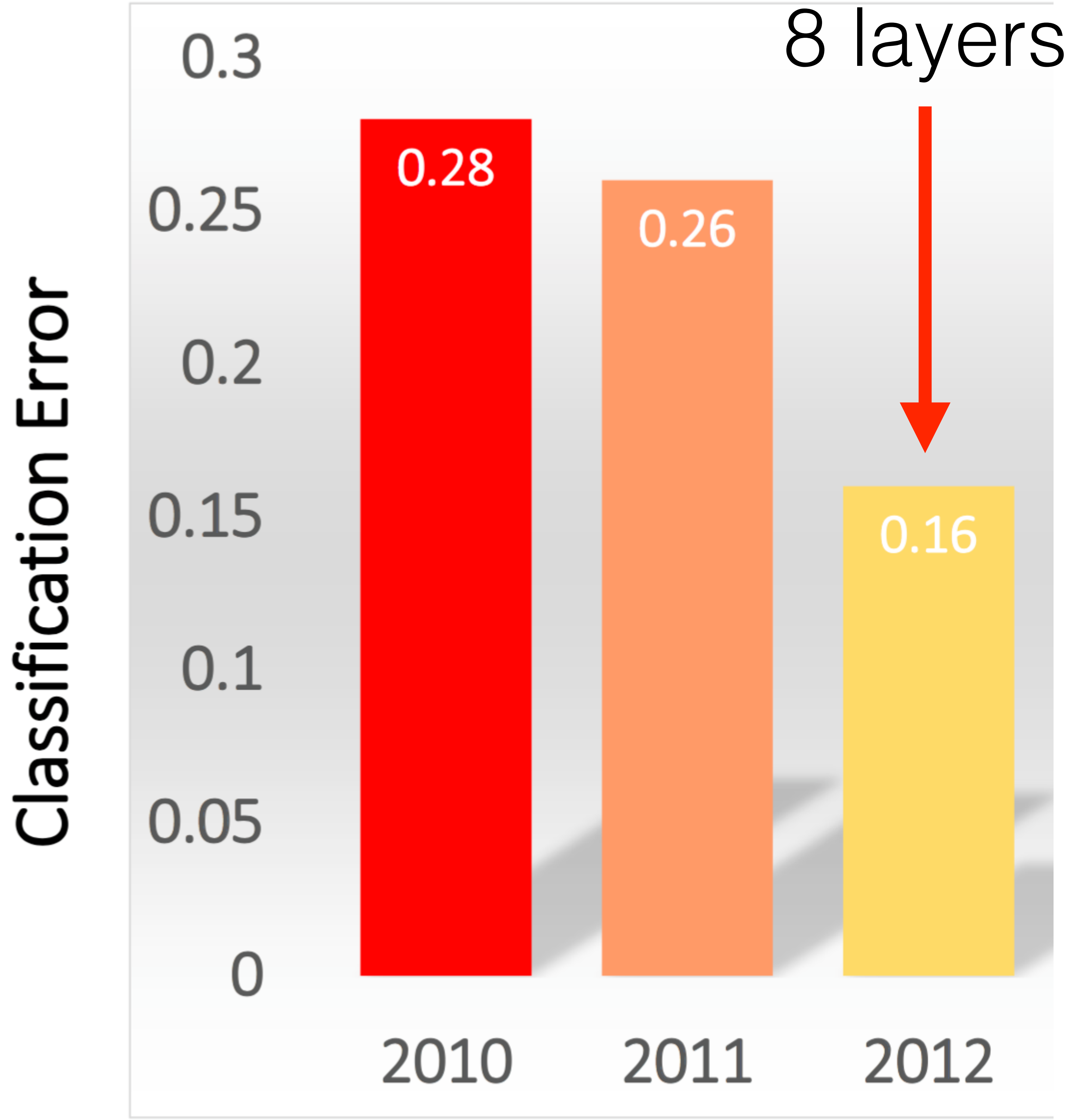
Classification results



IMAGENET

Classification results

AlexNet
8 layers

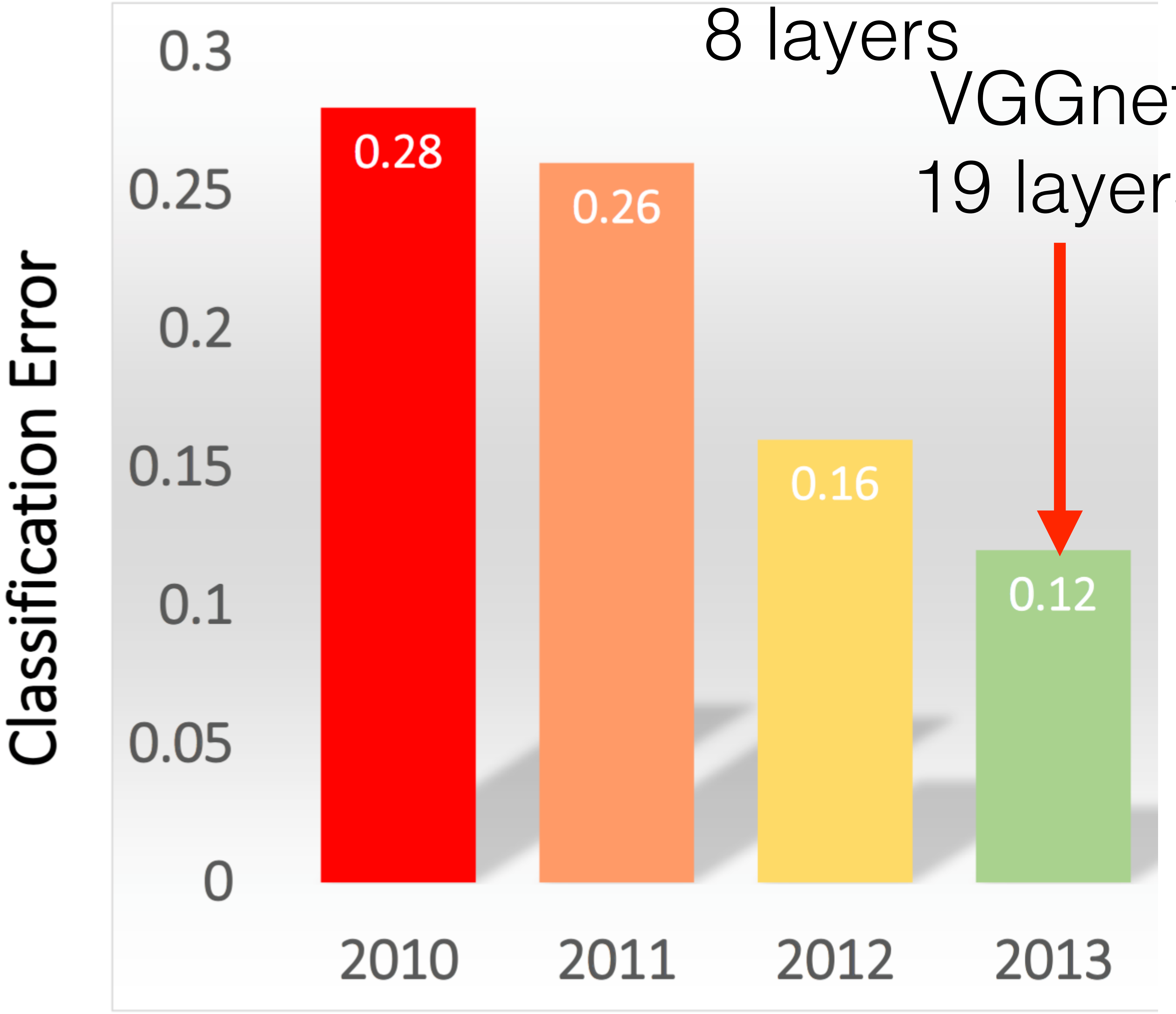


IMAGENET

Classification results

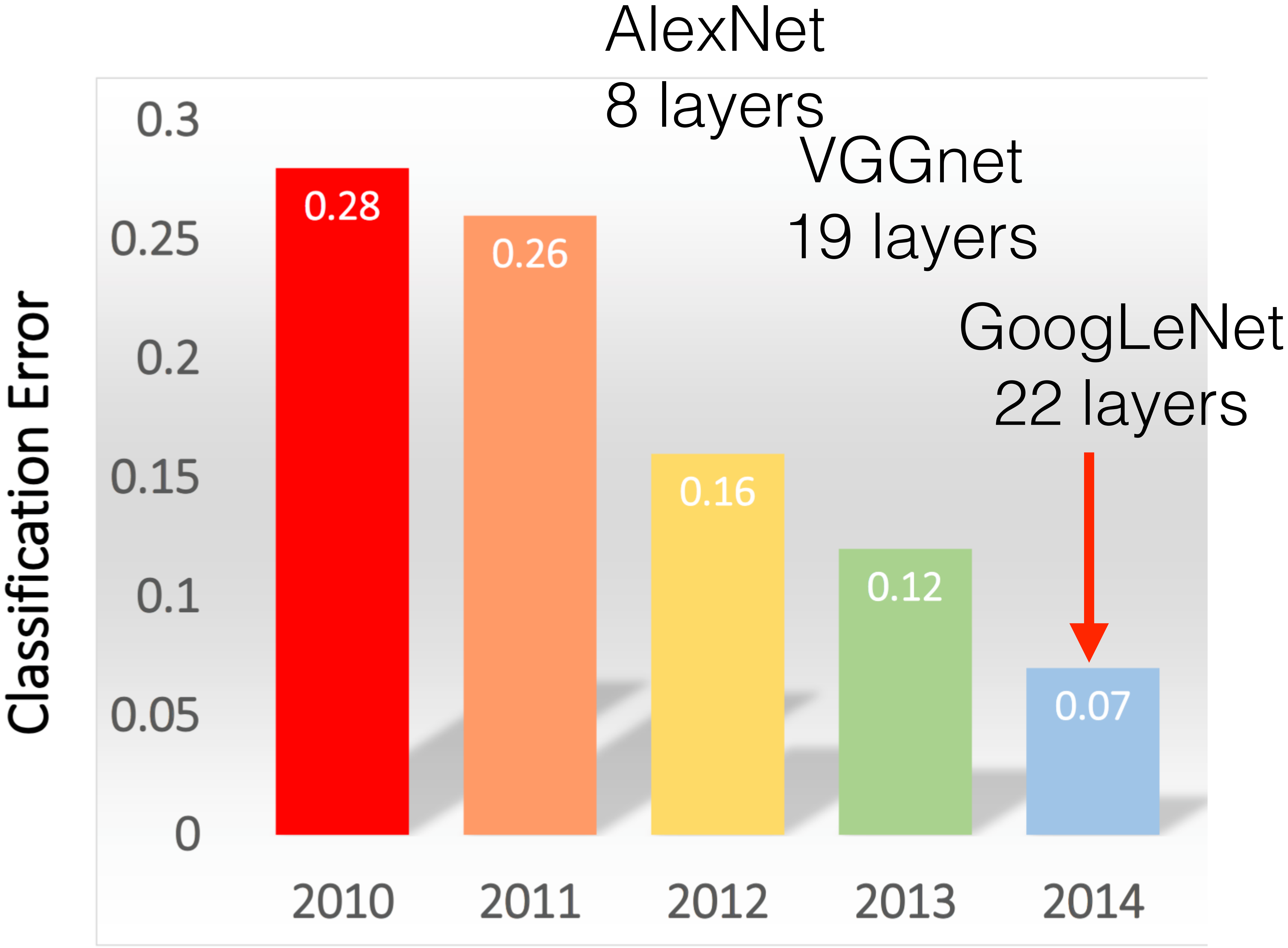
AlexNet
8 layers

VGGnet
19 layers

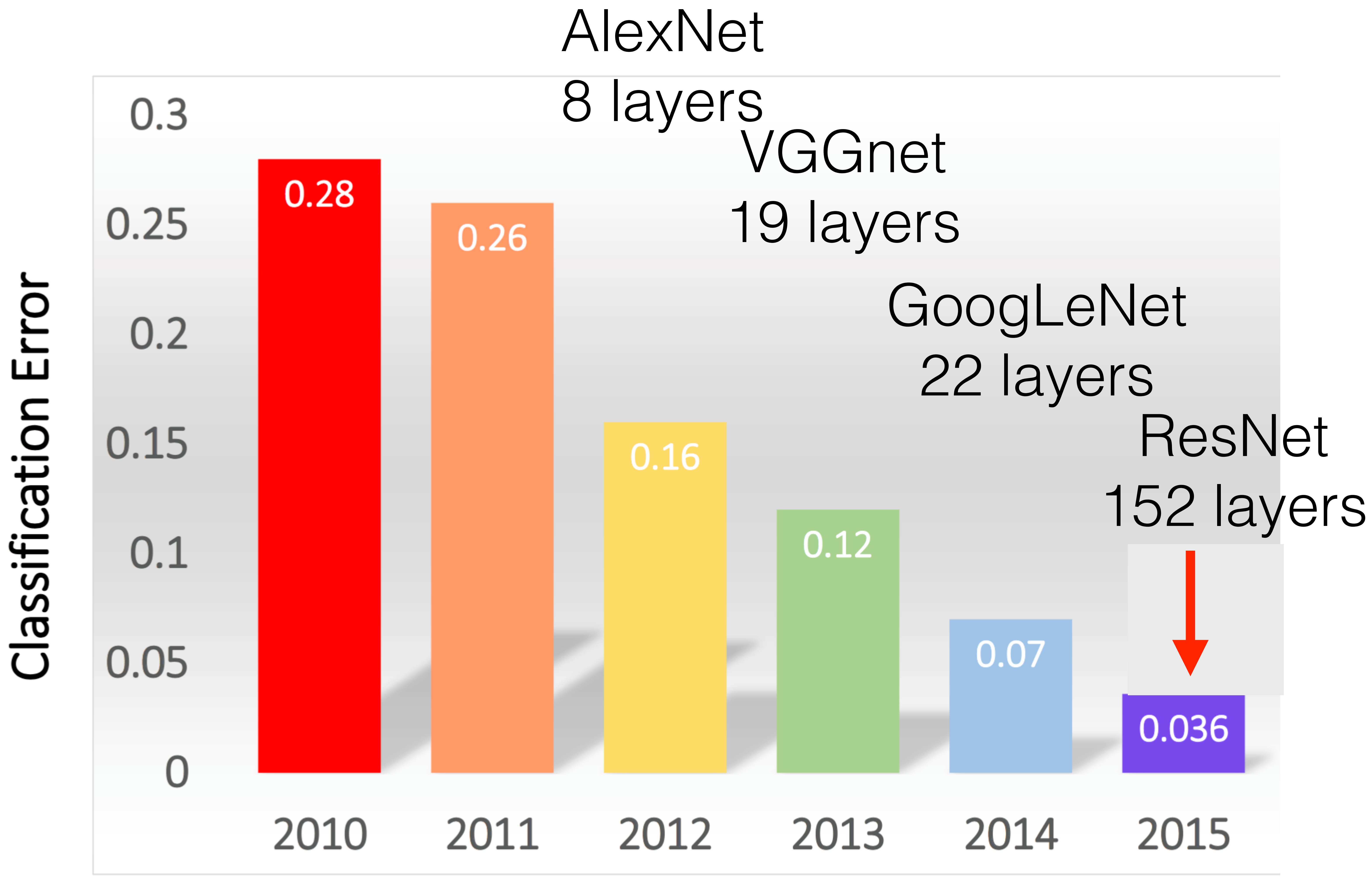


IMAGENET

Classification results

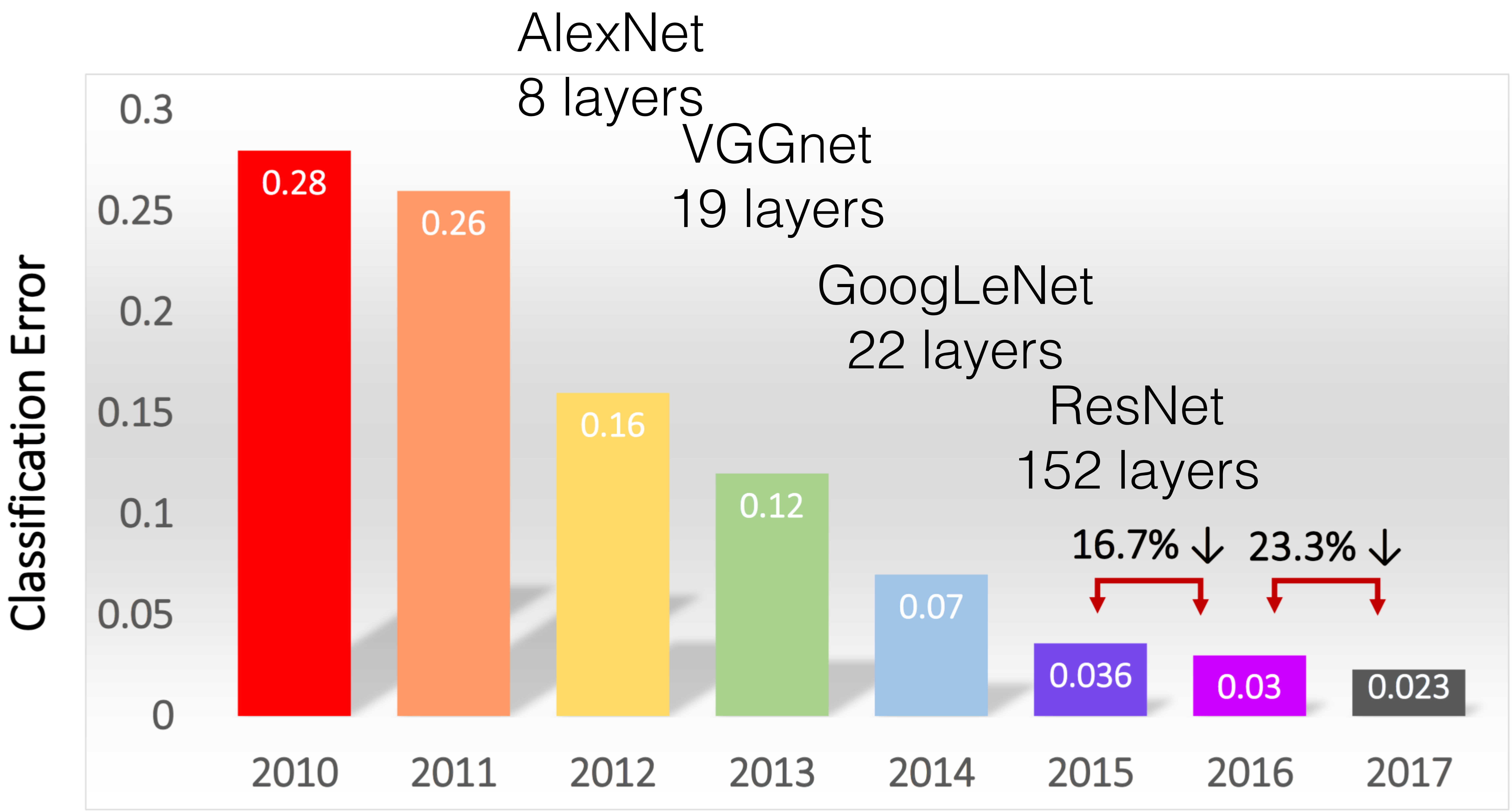


Classification results

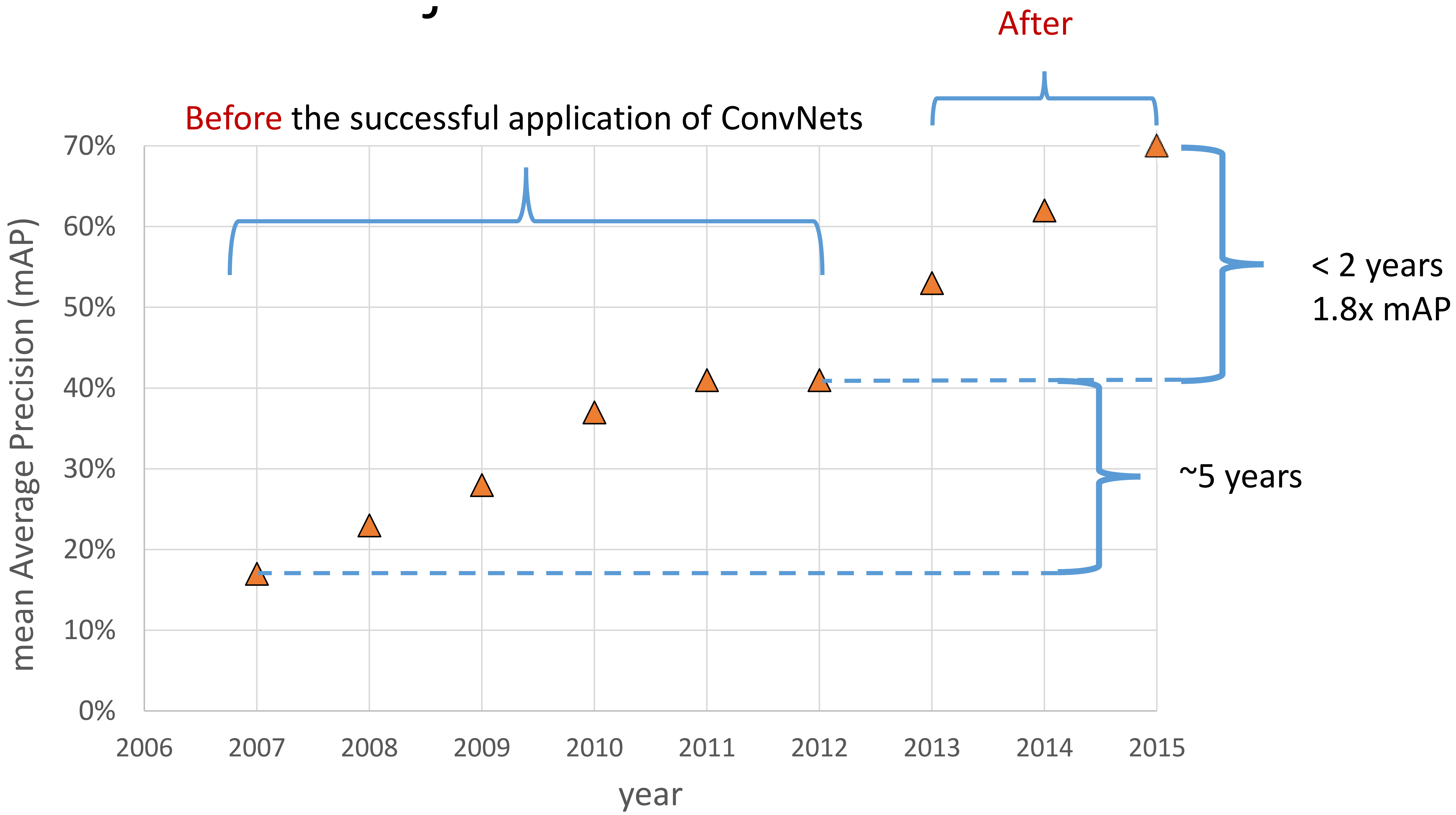


IMAGENET

Classification results



Pascal VOC object detection challenge



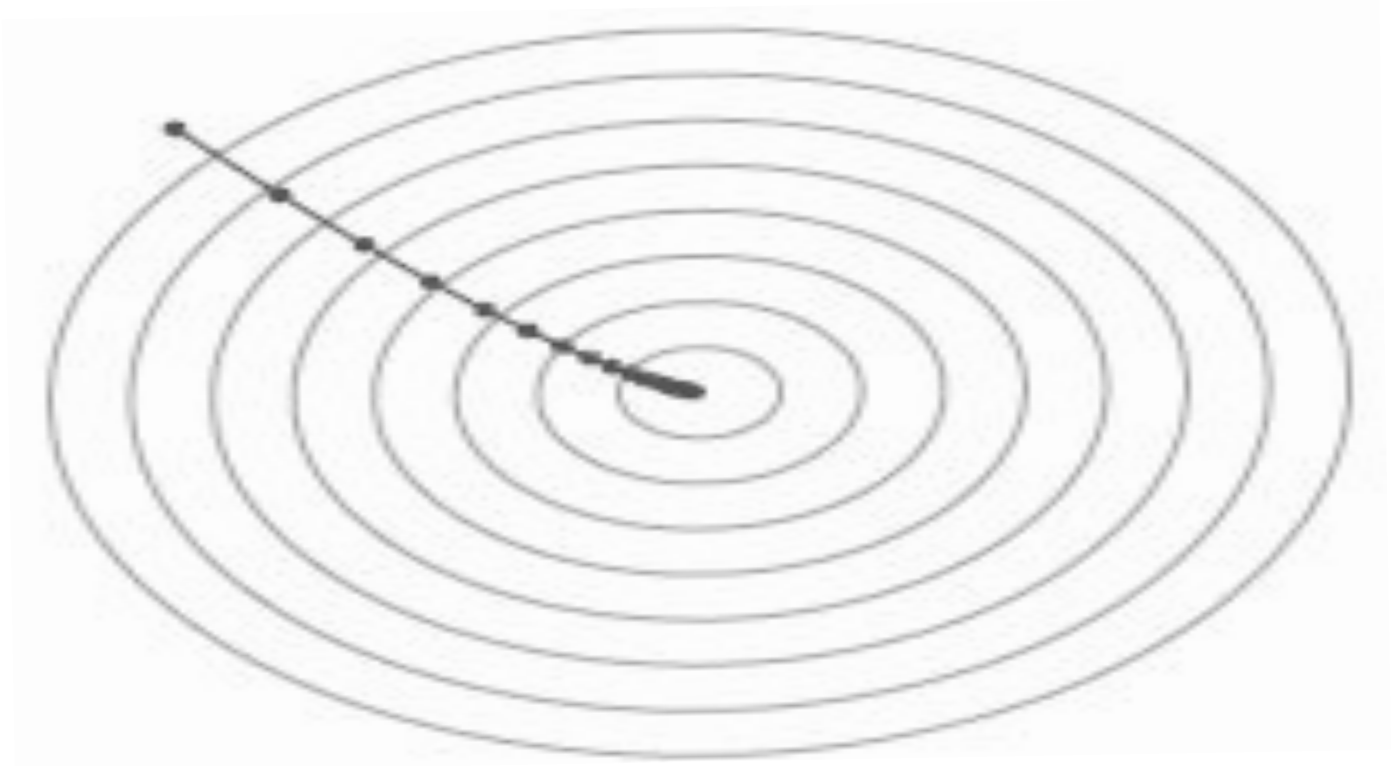
Learning as gradient minimization

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 does not fit into memory => instead estimate stochas over minibatch

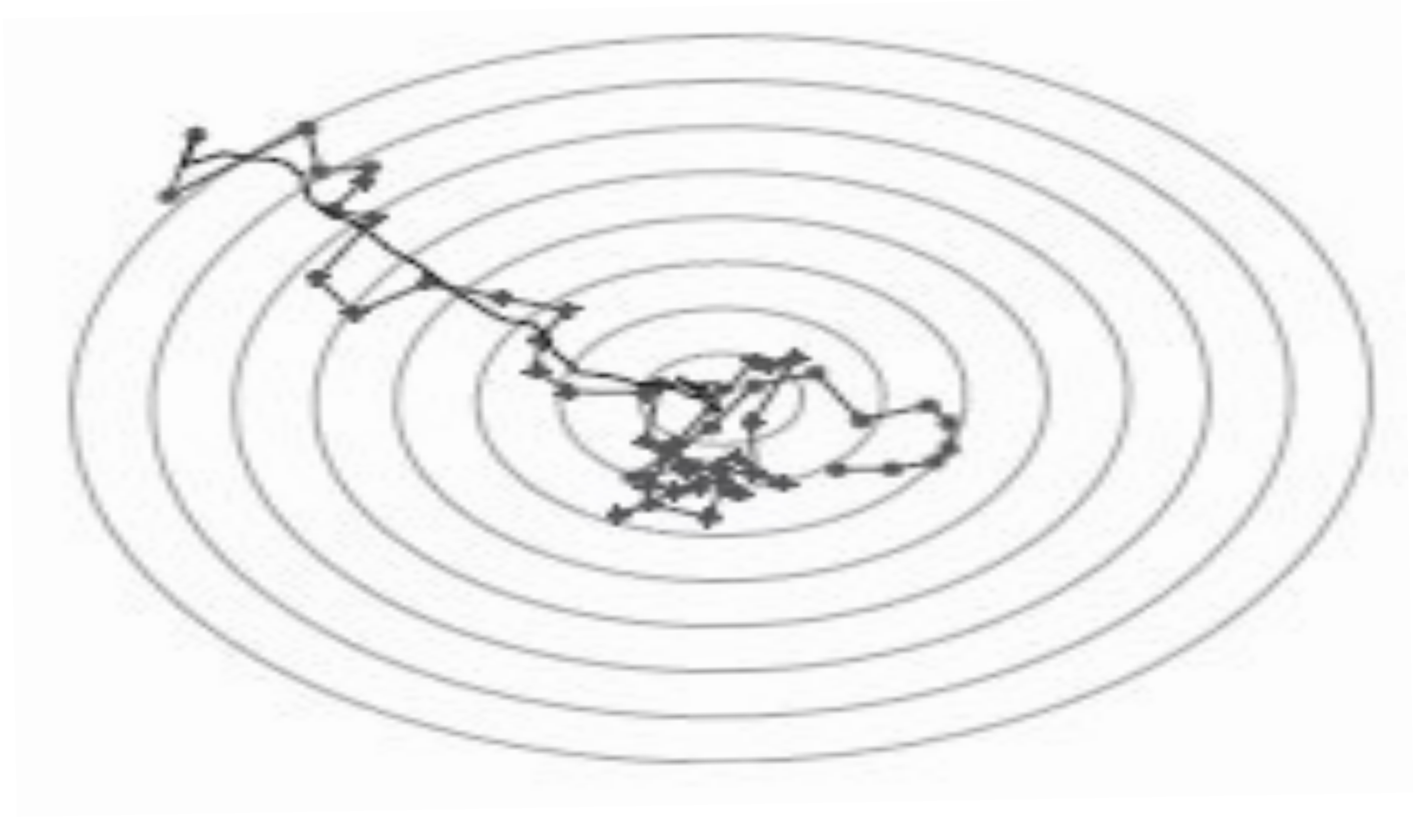
Learning as gradient minimization

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

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

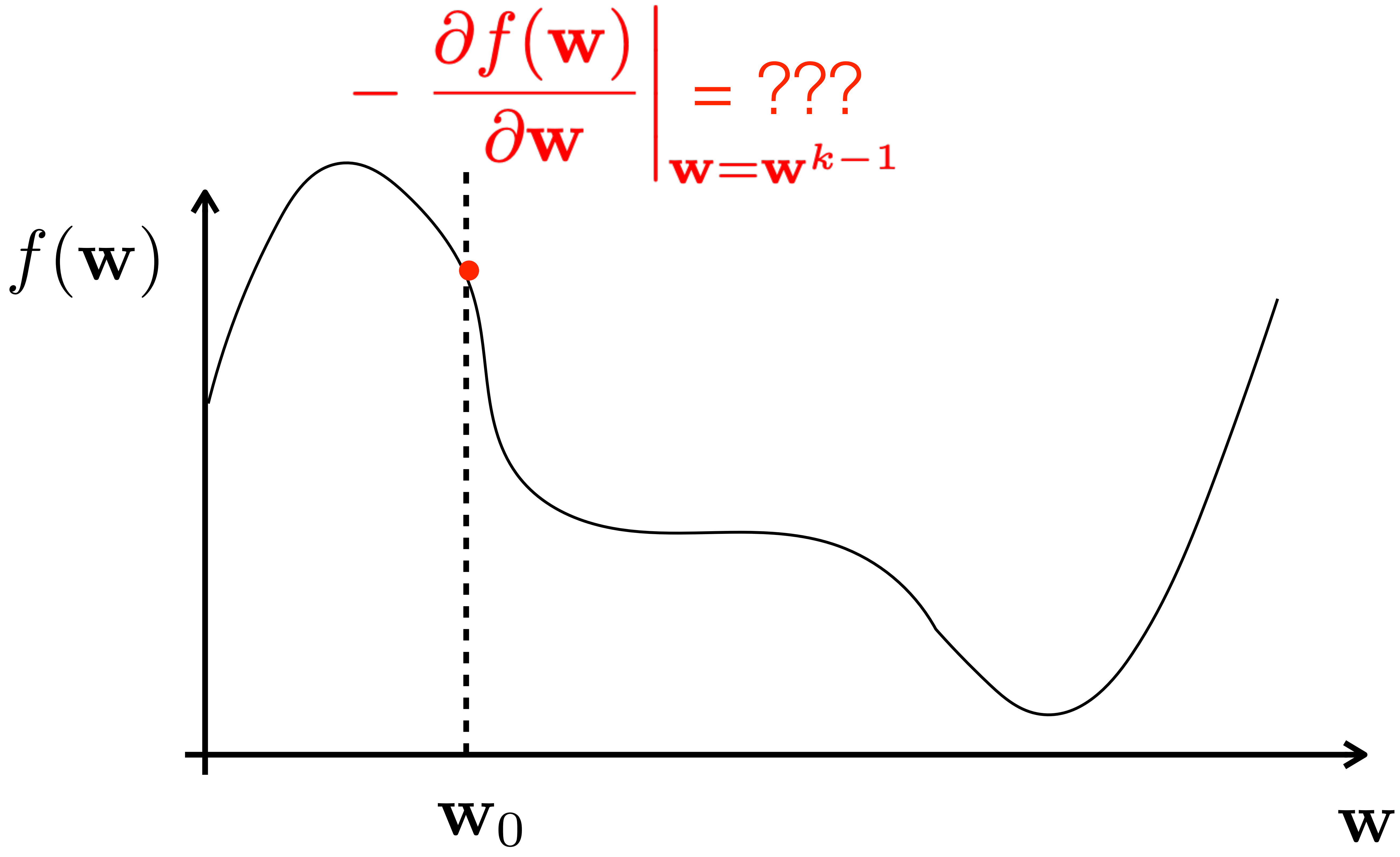
4. Update weights

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



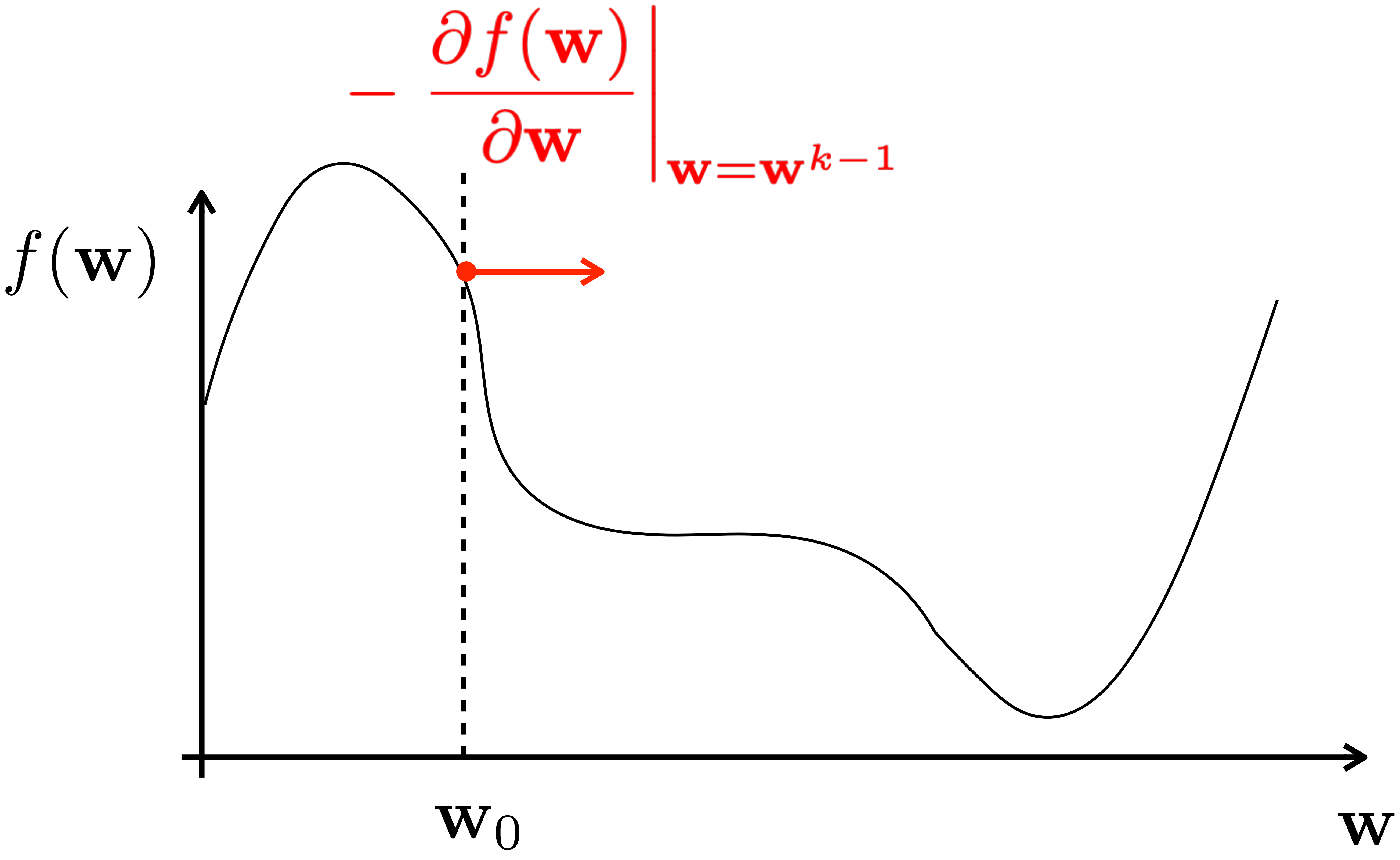
Stochastic Gradient Descent (SGD) drawbacks

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



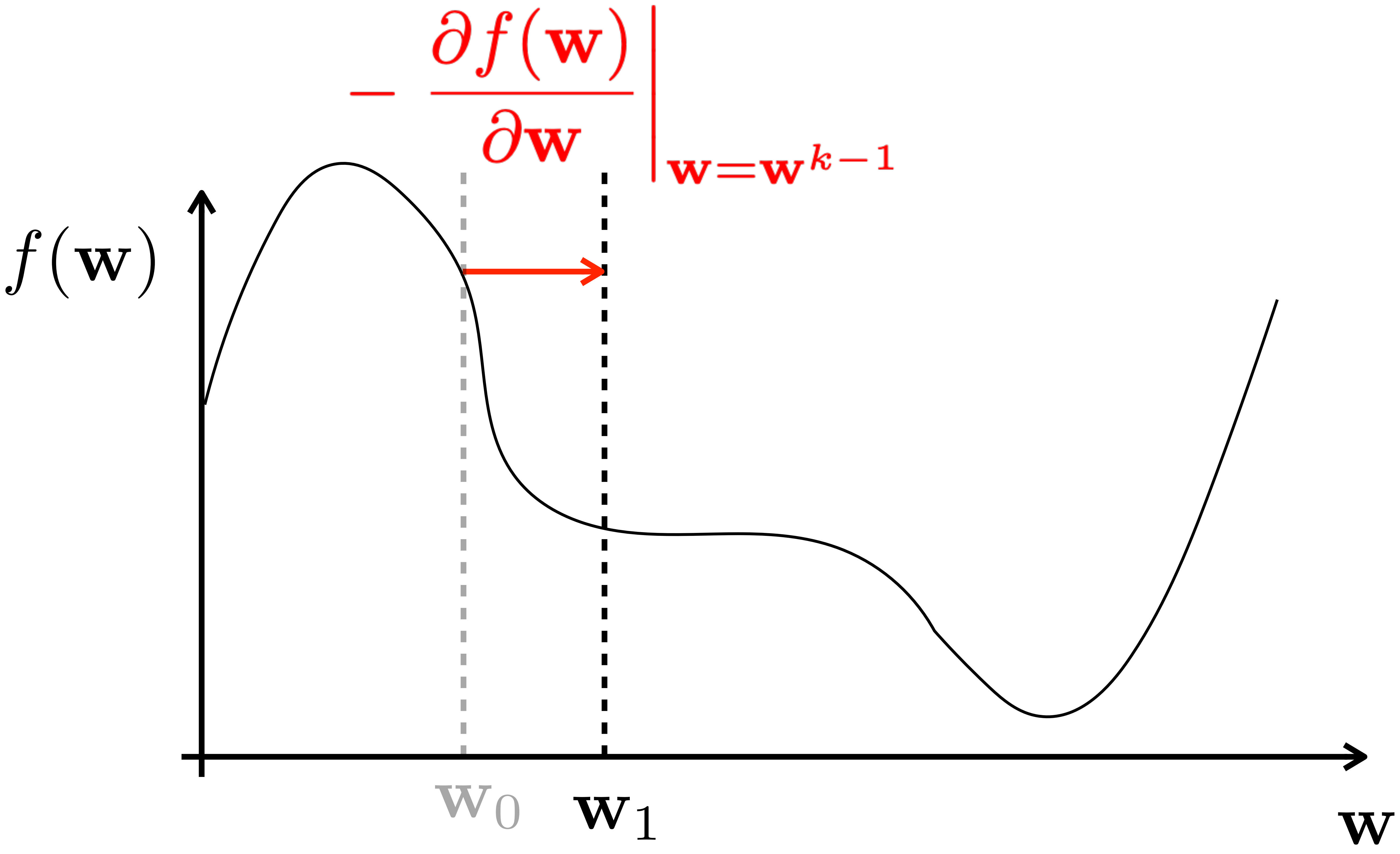
Stochastic Gradient Descent (SGD) drawbacks

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



SGD drawbacks

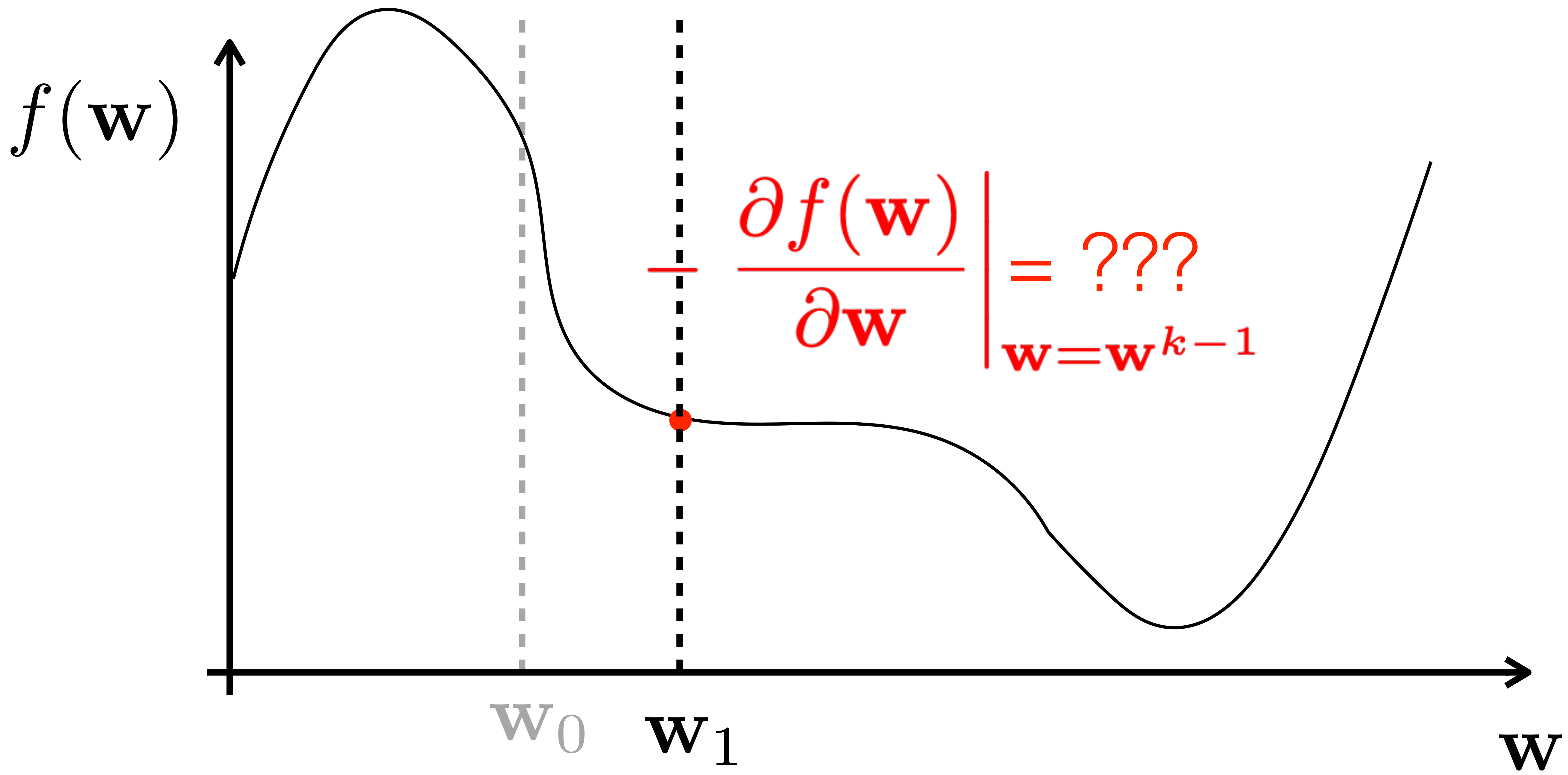
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



SGD drawbacks

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

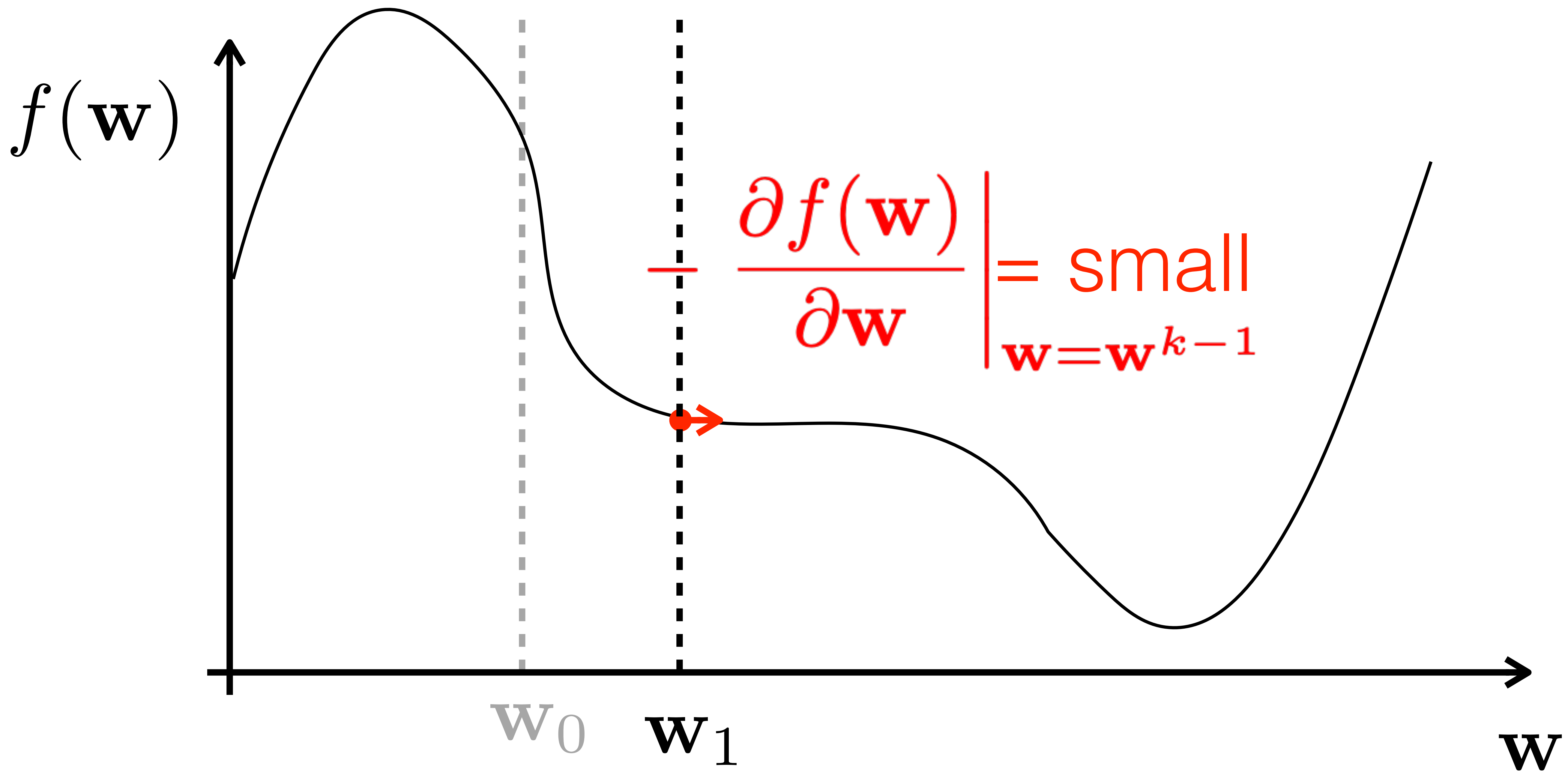
- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima



SGD drawbacks

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

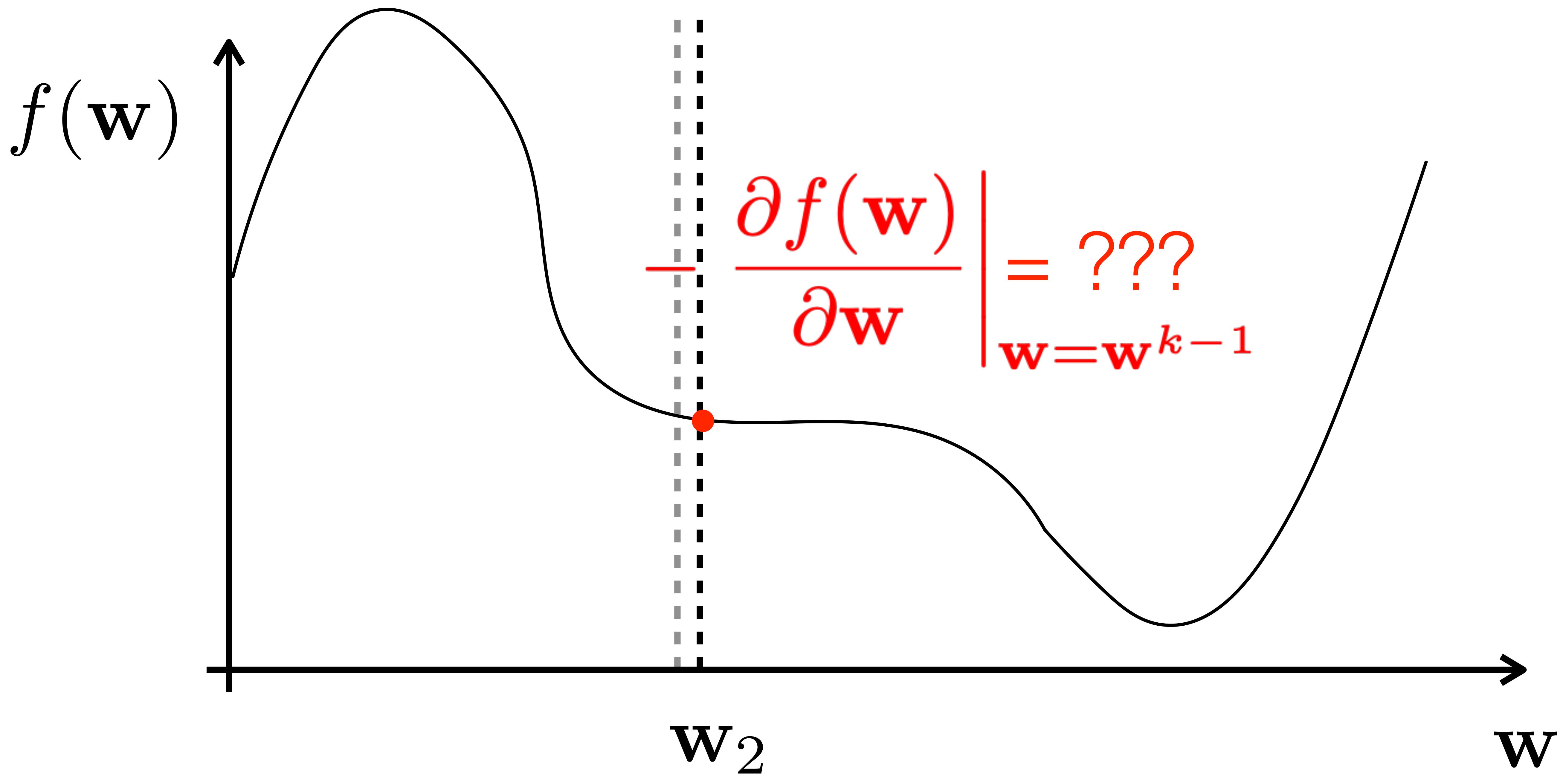
- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima



SGD drawbacks

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

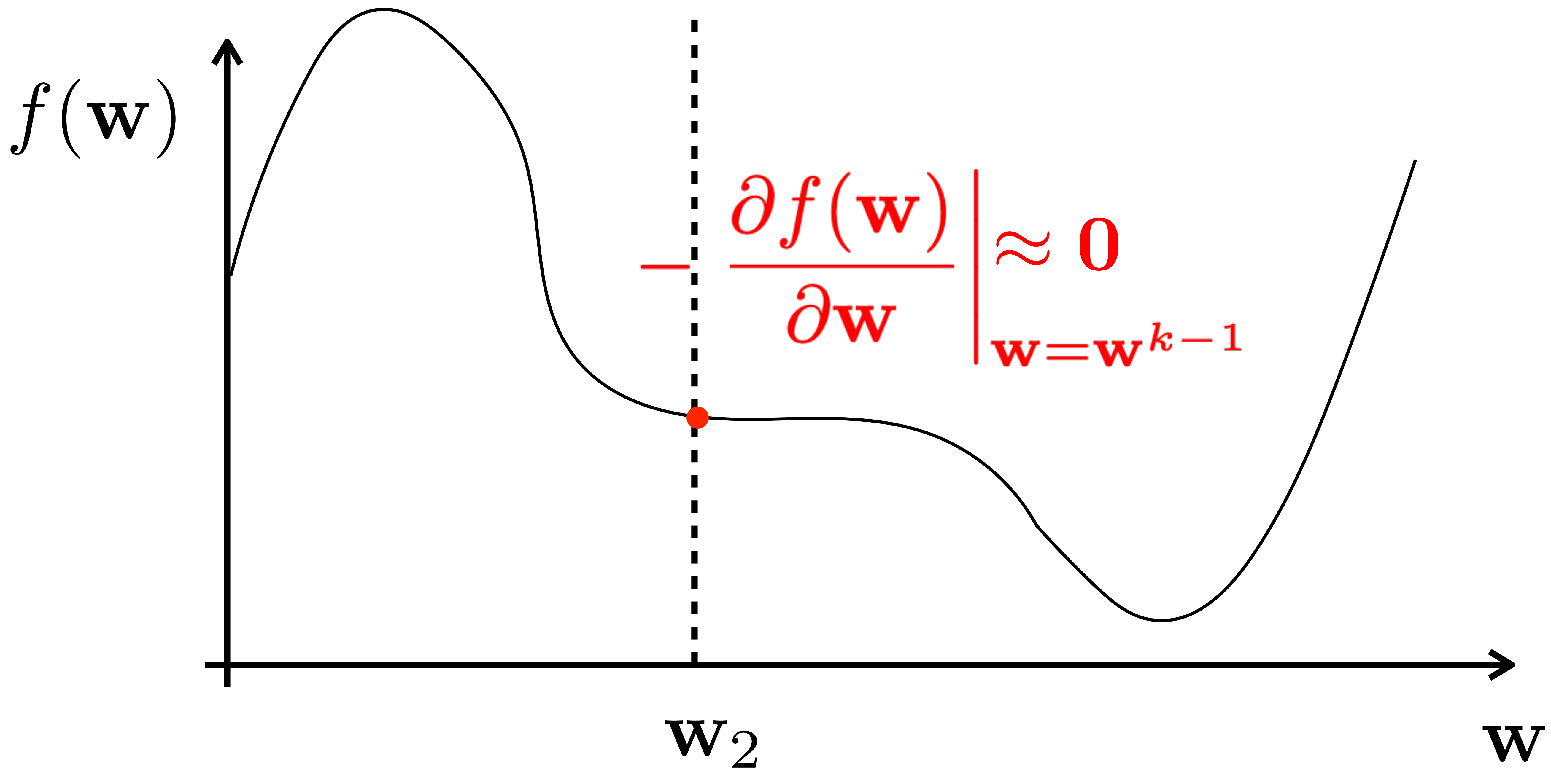
- Easily get stuck in local minima or saddle points
- There are much more saddle points than minima



SGD drawbacks

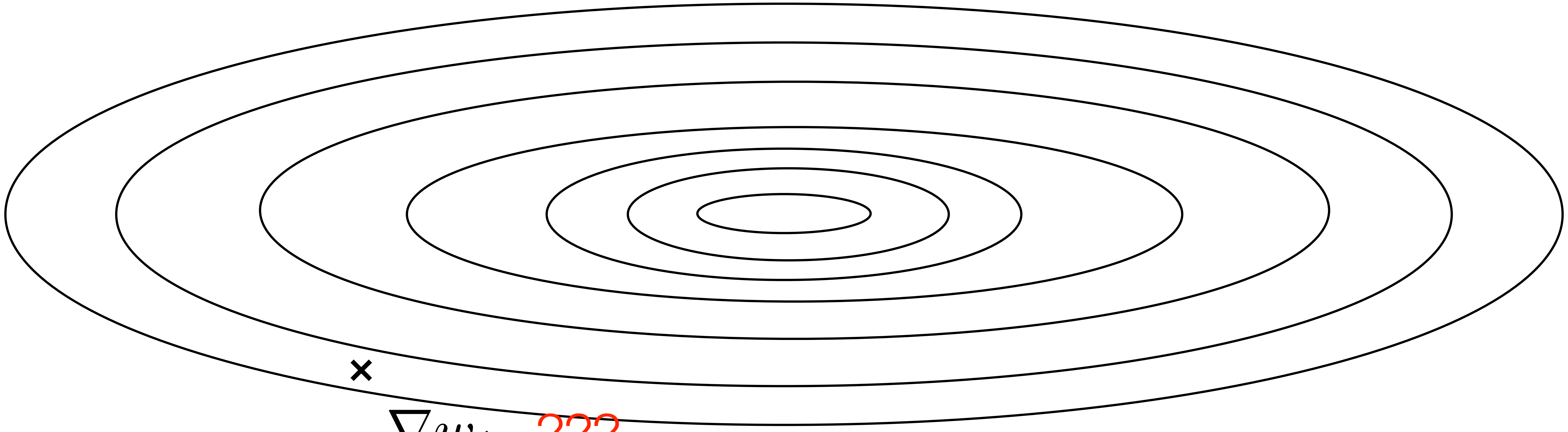
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

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



SGD in 2 dimensional weights

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



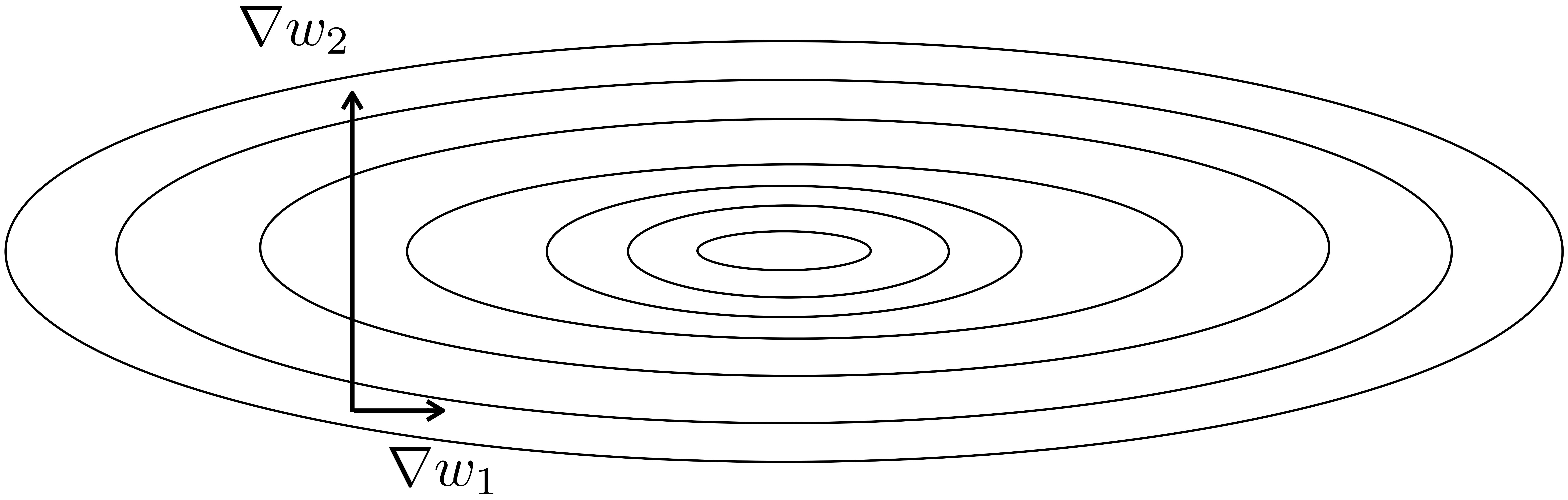
∇w_1=???

∇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

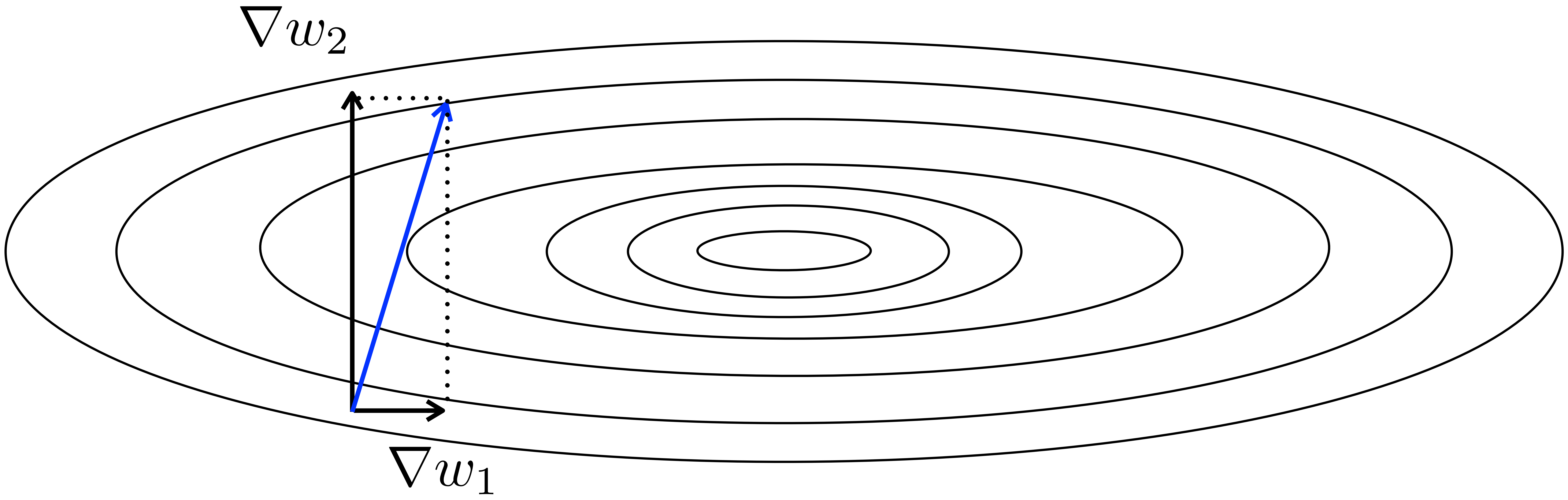
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD in 2 dimensional weights

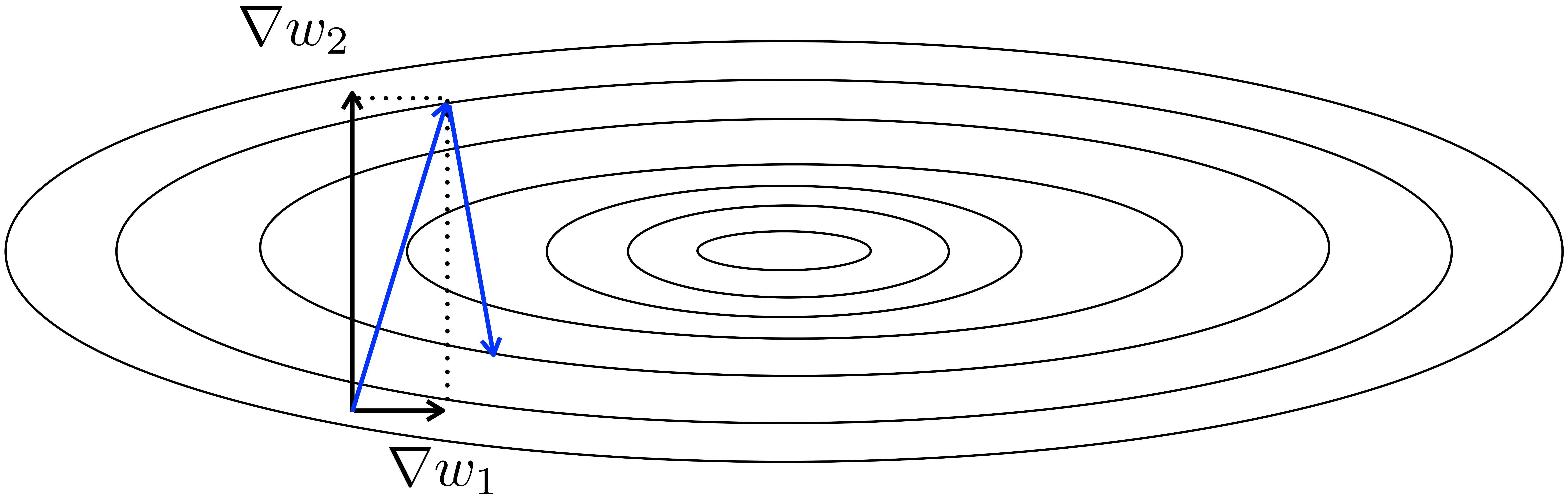
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD in 2 dimensional weights

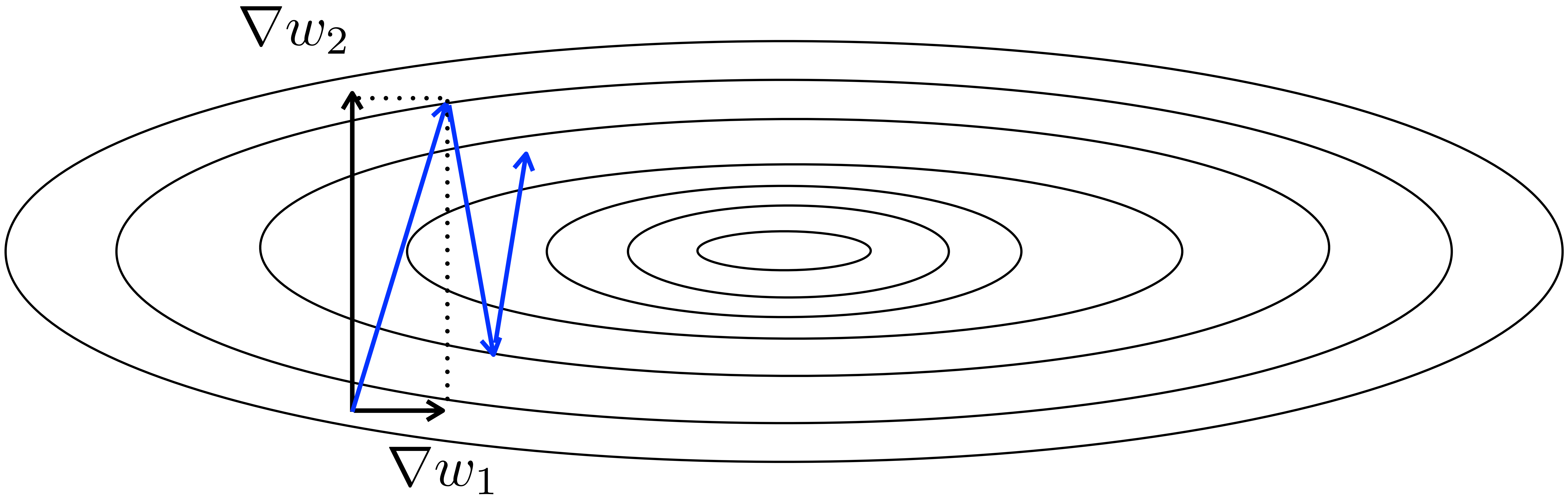
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



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

SGD in 2 dimensional weights

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

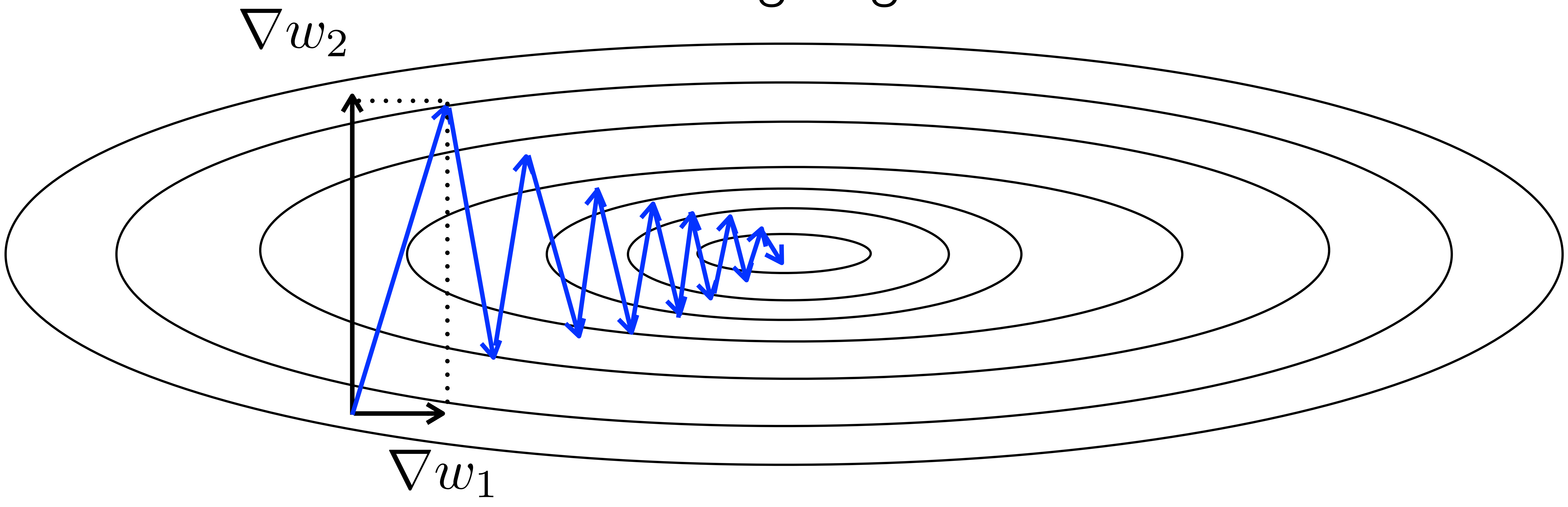


$$[\nabla w_1, \nabla w_2] = - \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

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

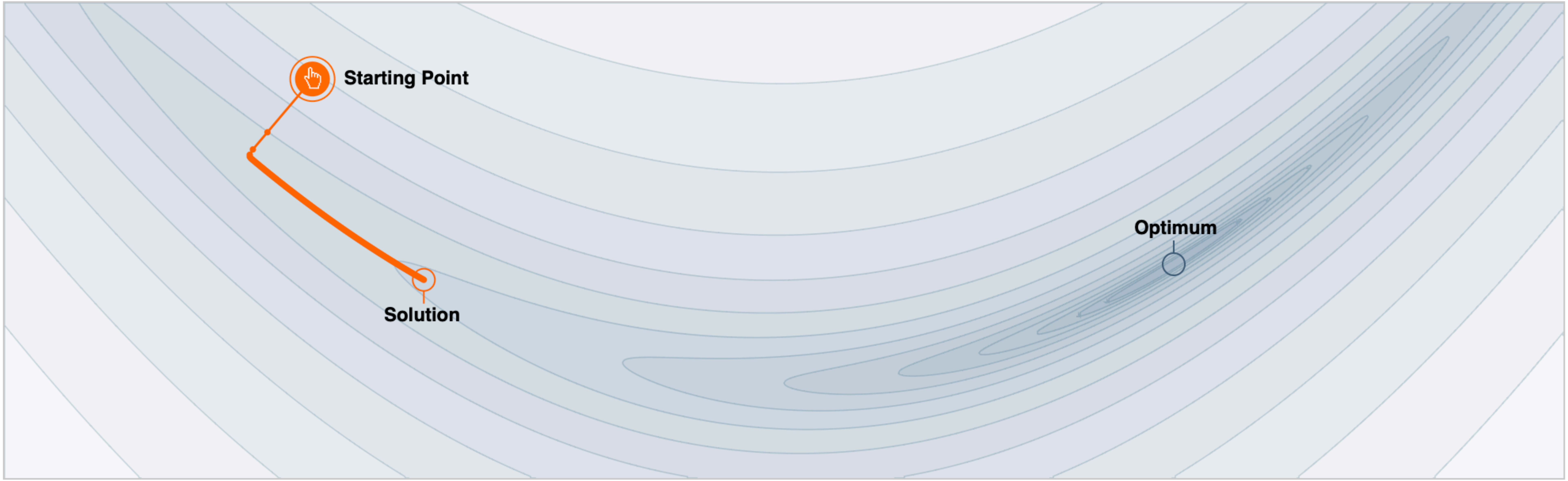


$$[\nabla w_1, \nabla w_2] = - \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

SGD drawbacks - in 2D

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

$\alpha =$ 1e-3

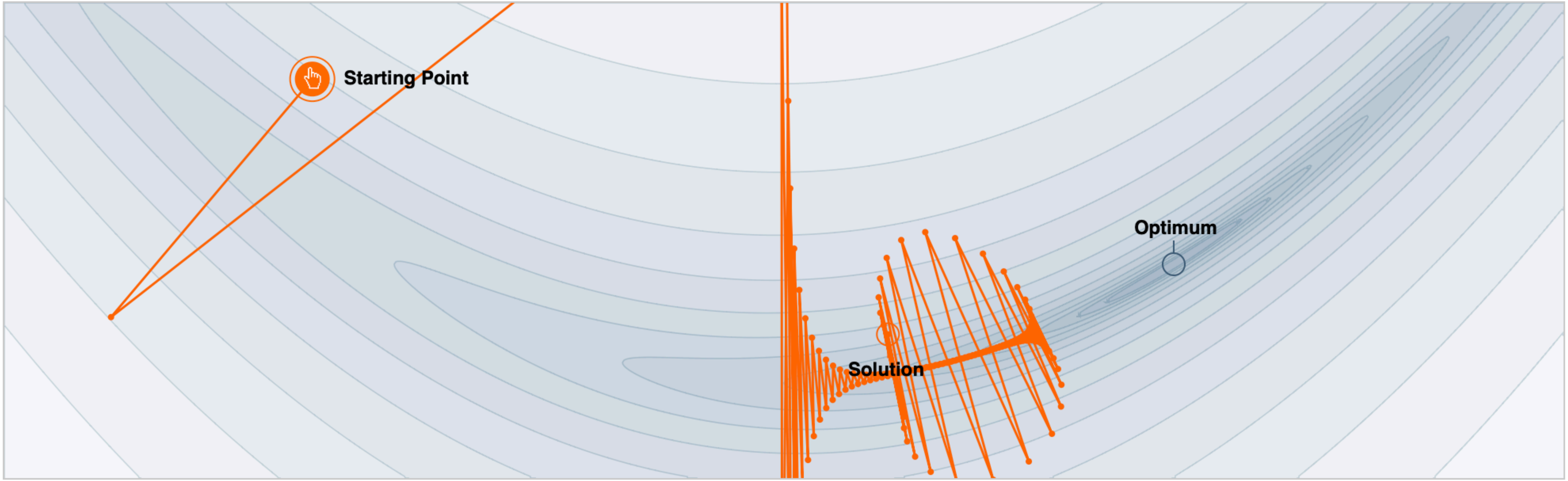


https://distill.pub/2017/momentum/

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

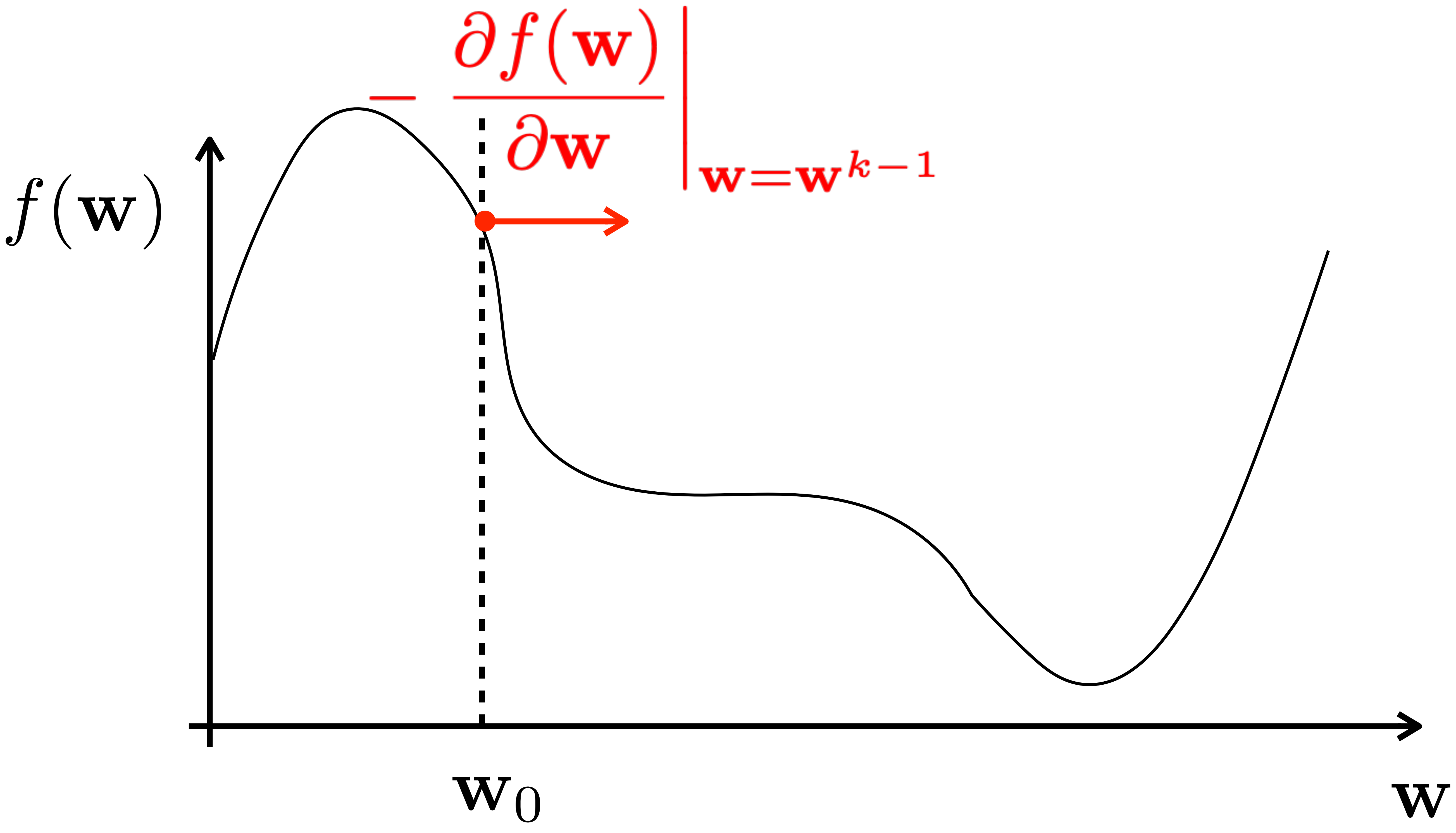


https://distill.pub/2017/momentum/

SGD + momentum

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

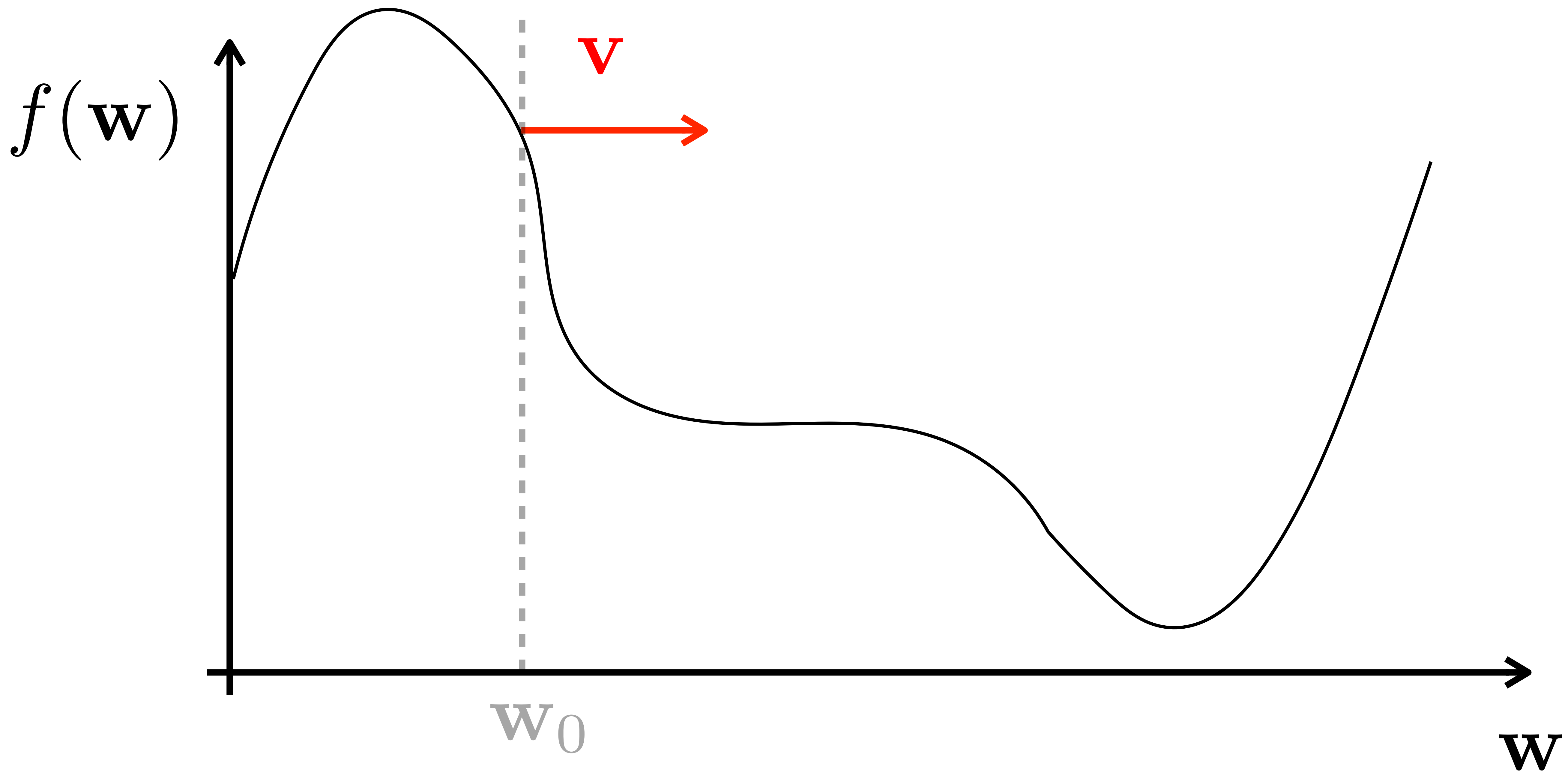
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

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

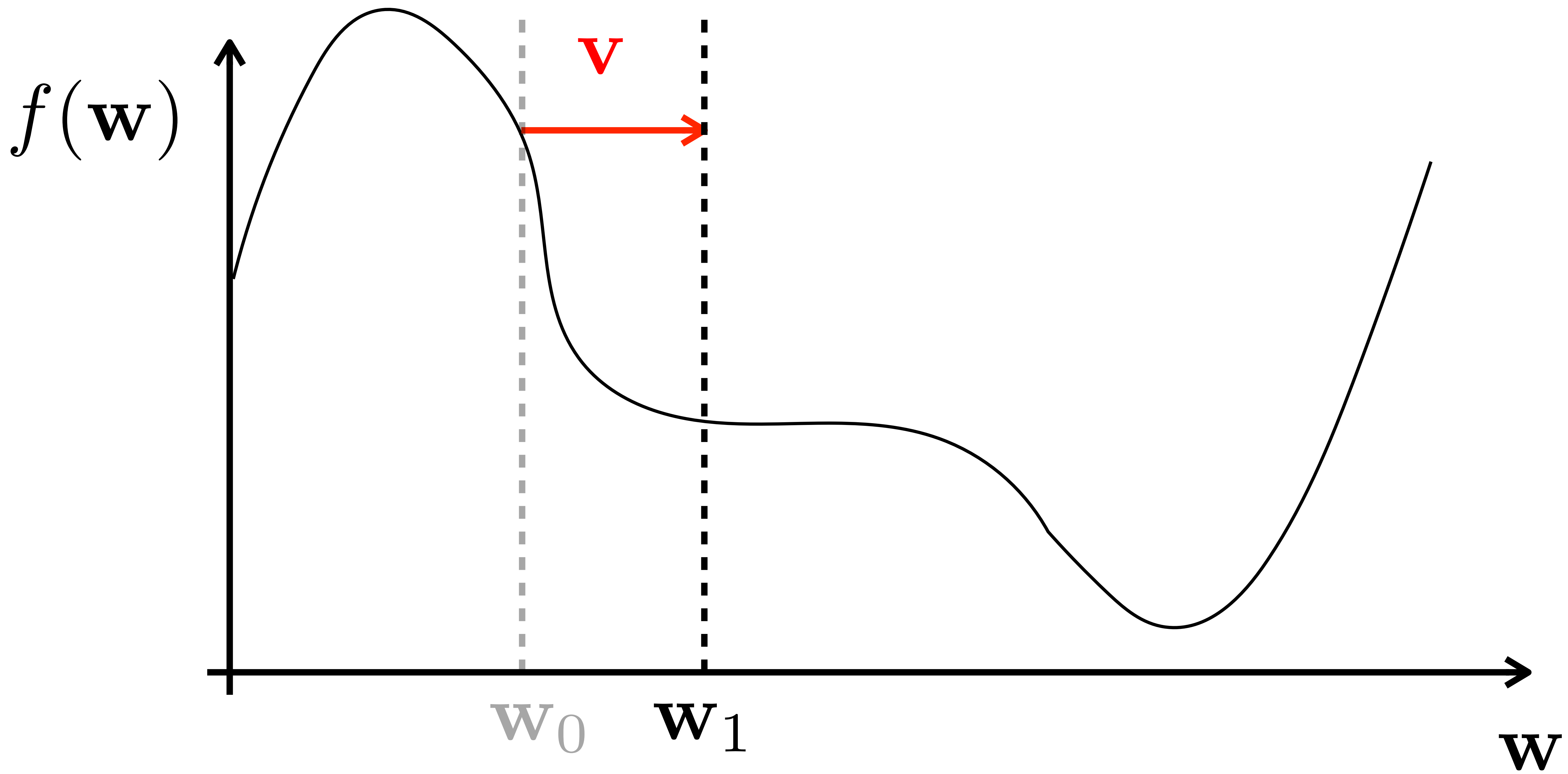
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

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

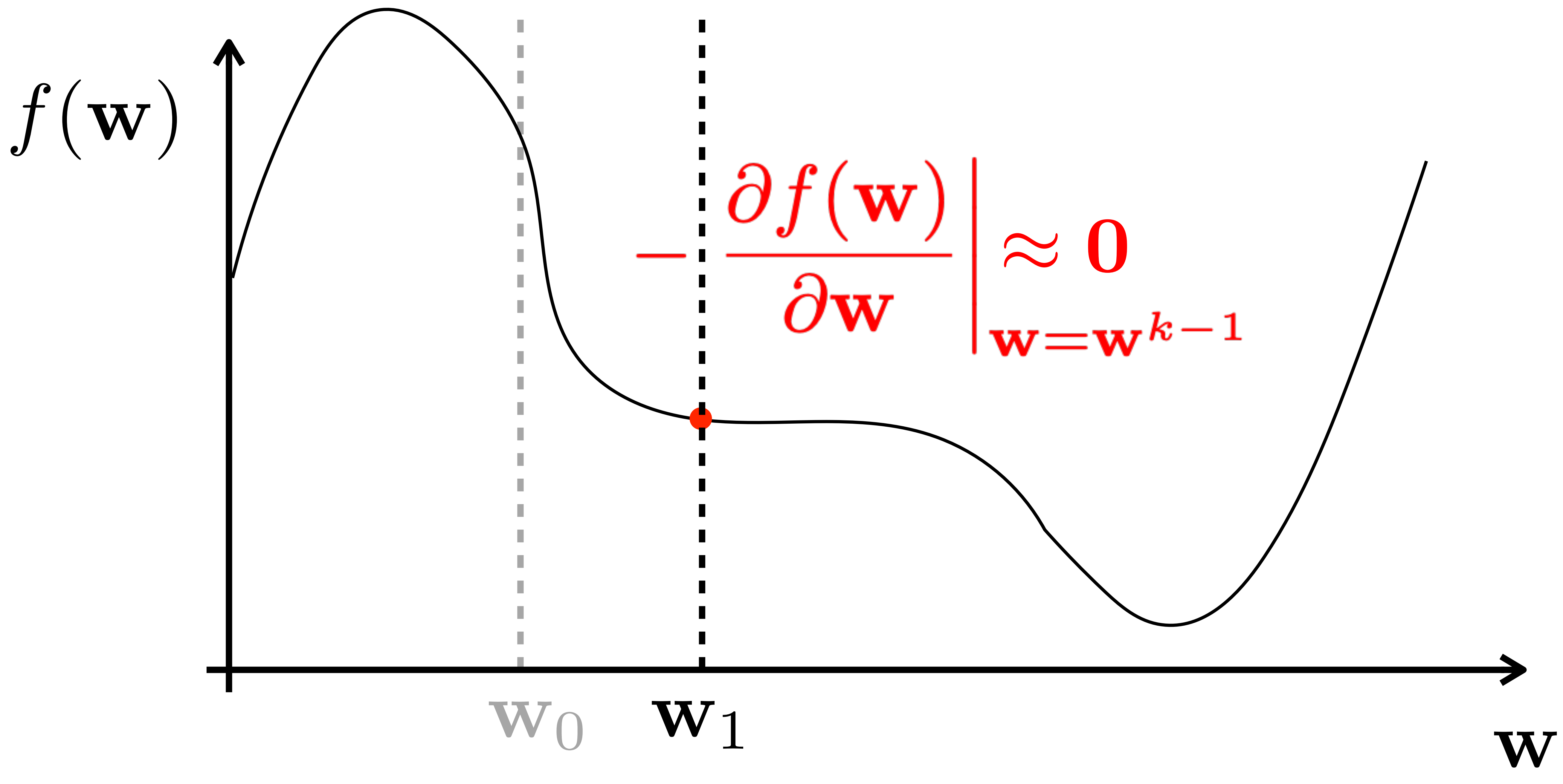
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD + momentum

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

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

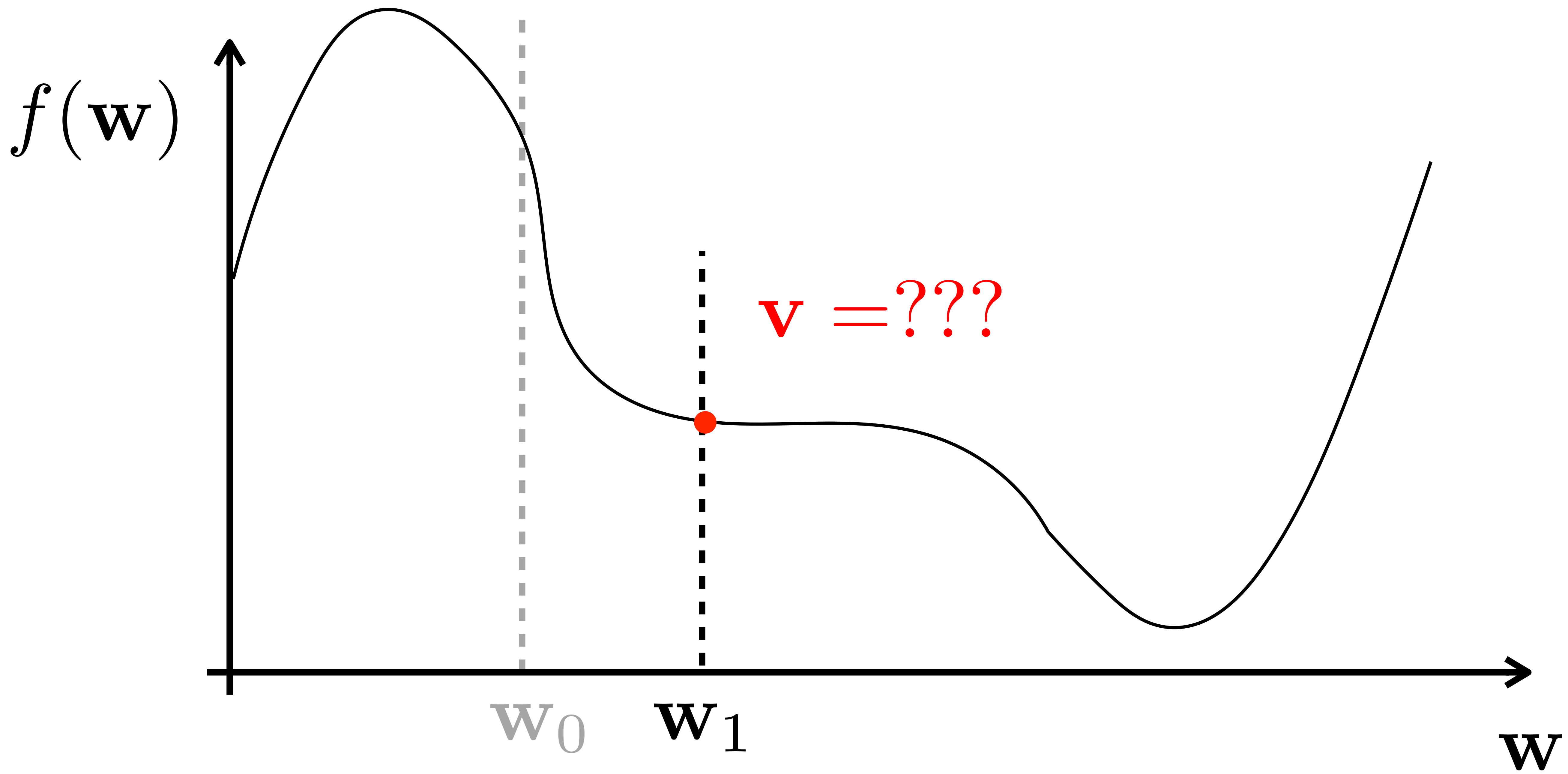


SGD + momentum

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

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

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

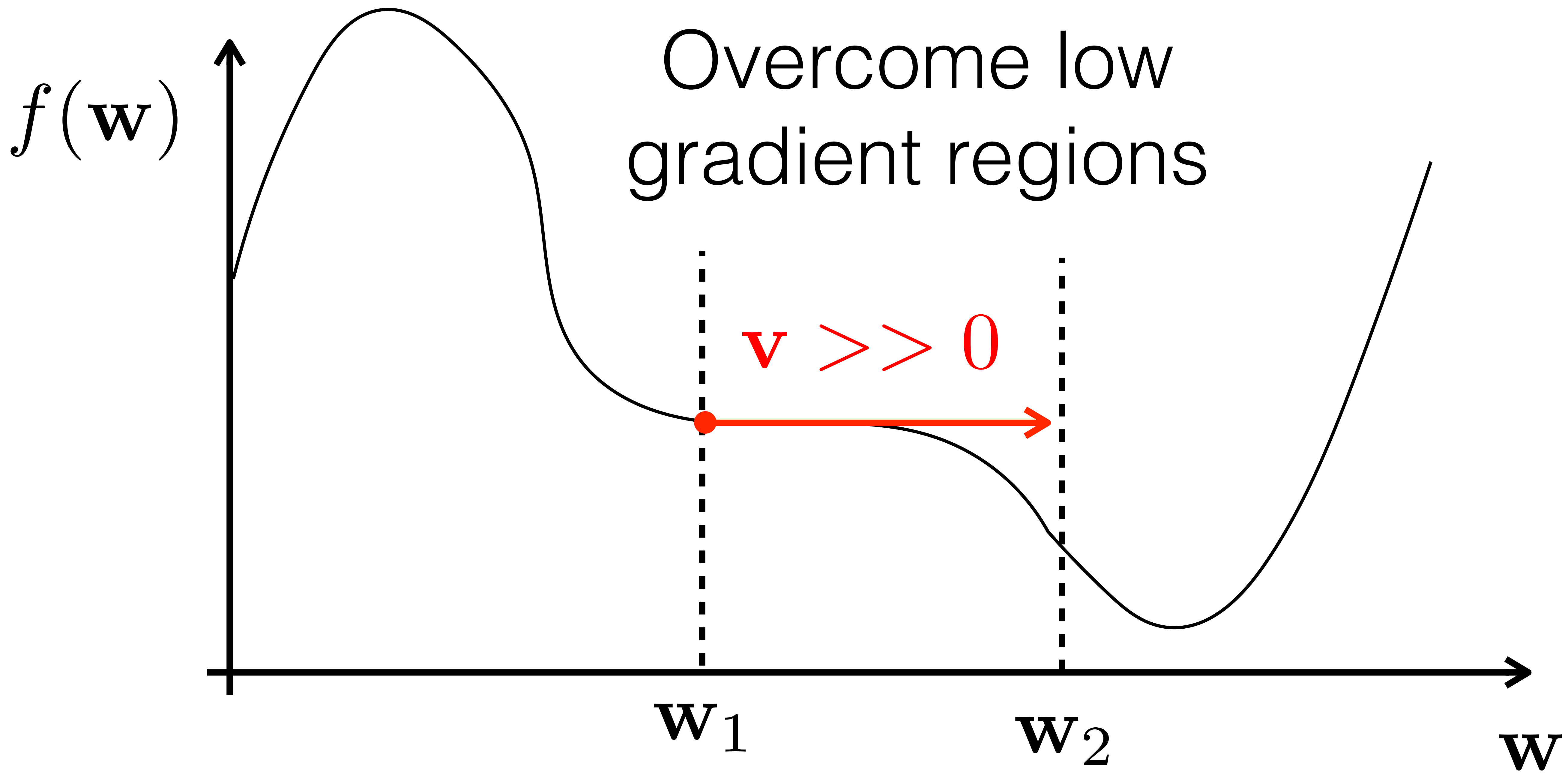


SGD + momentum

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

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

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

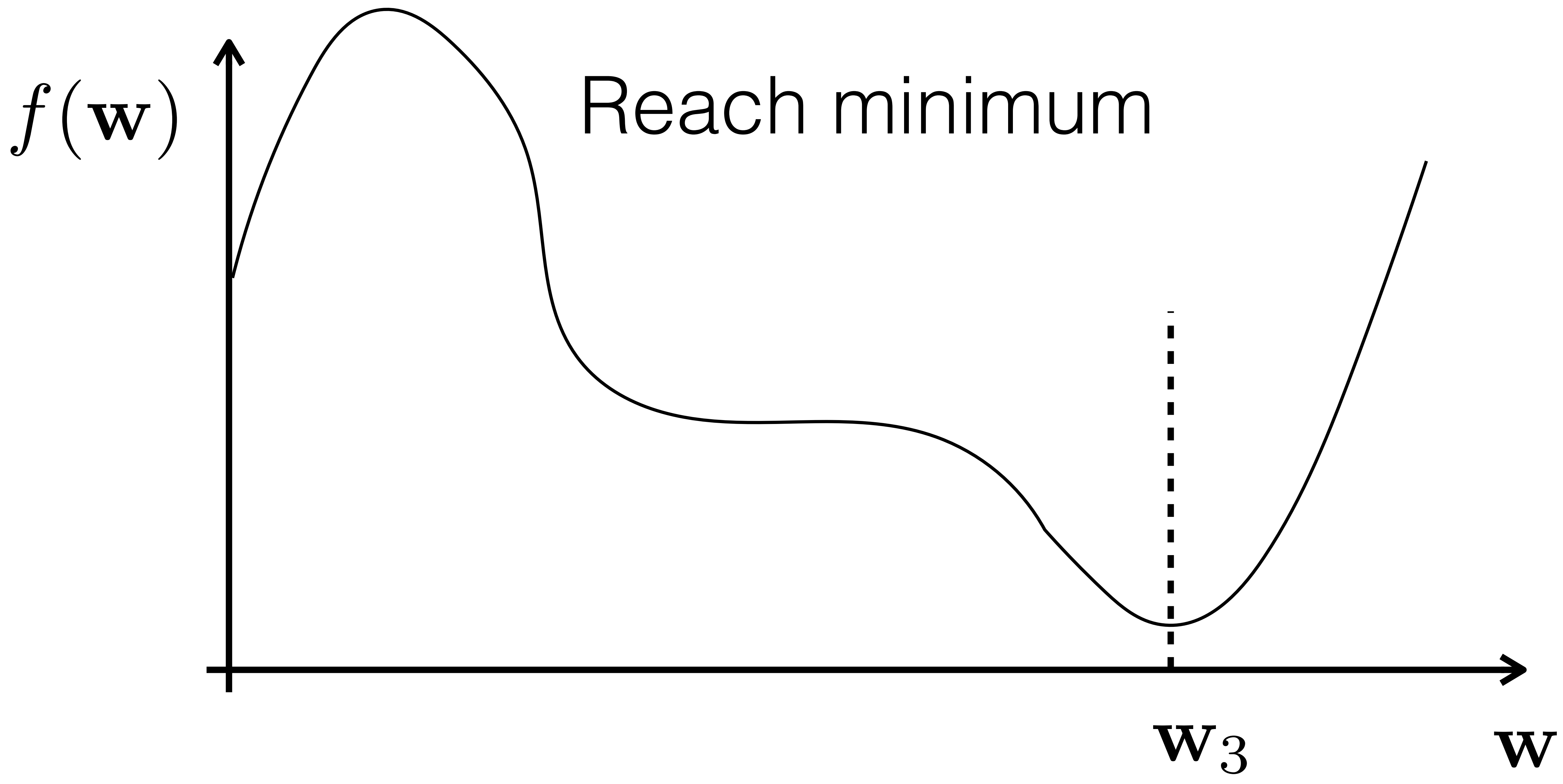


SGD + momentum

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

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

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

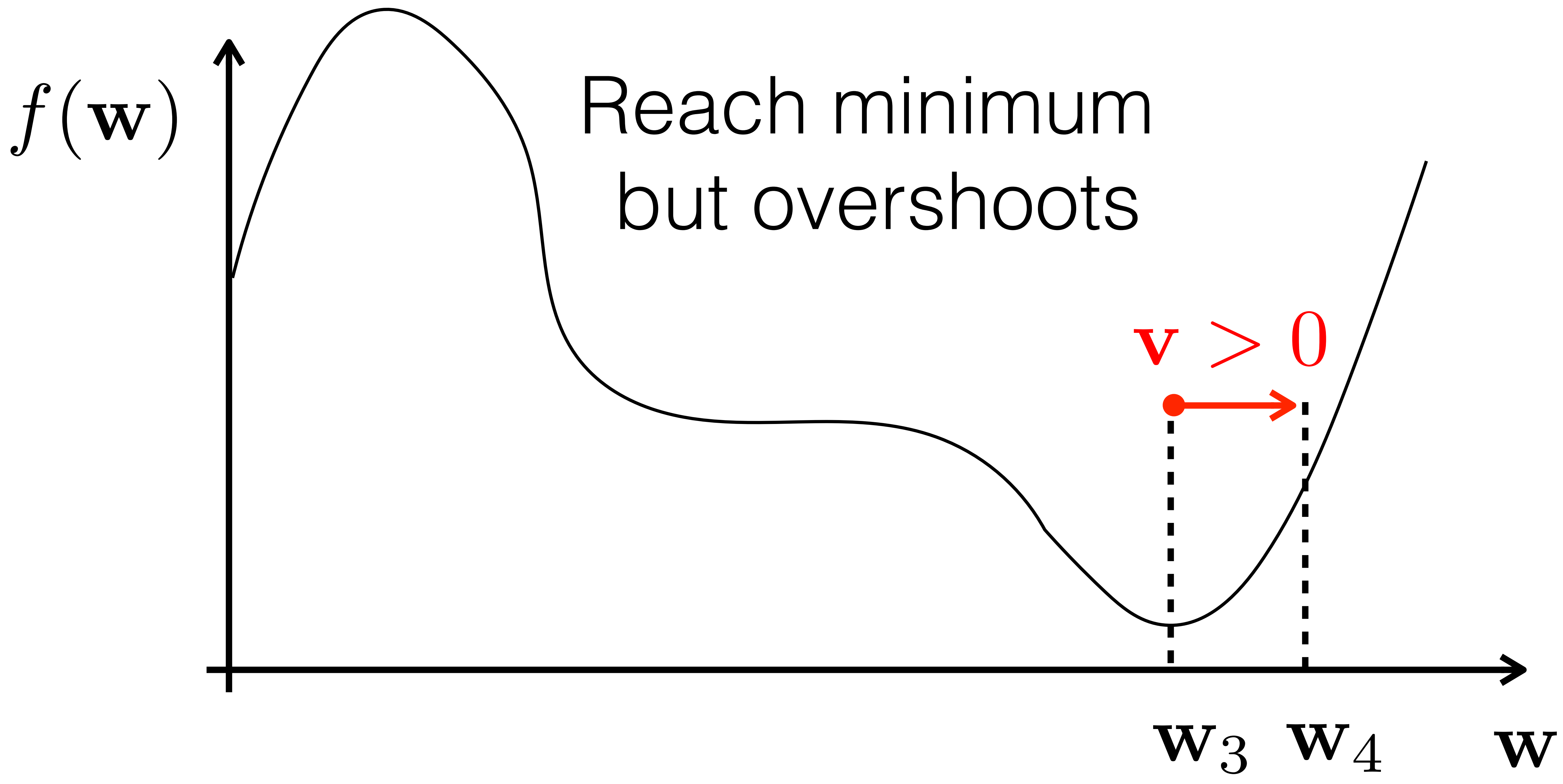


SGD + momentum

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

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

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

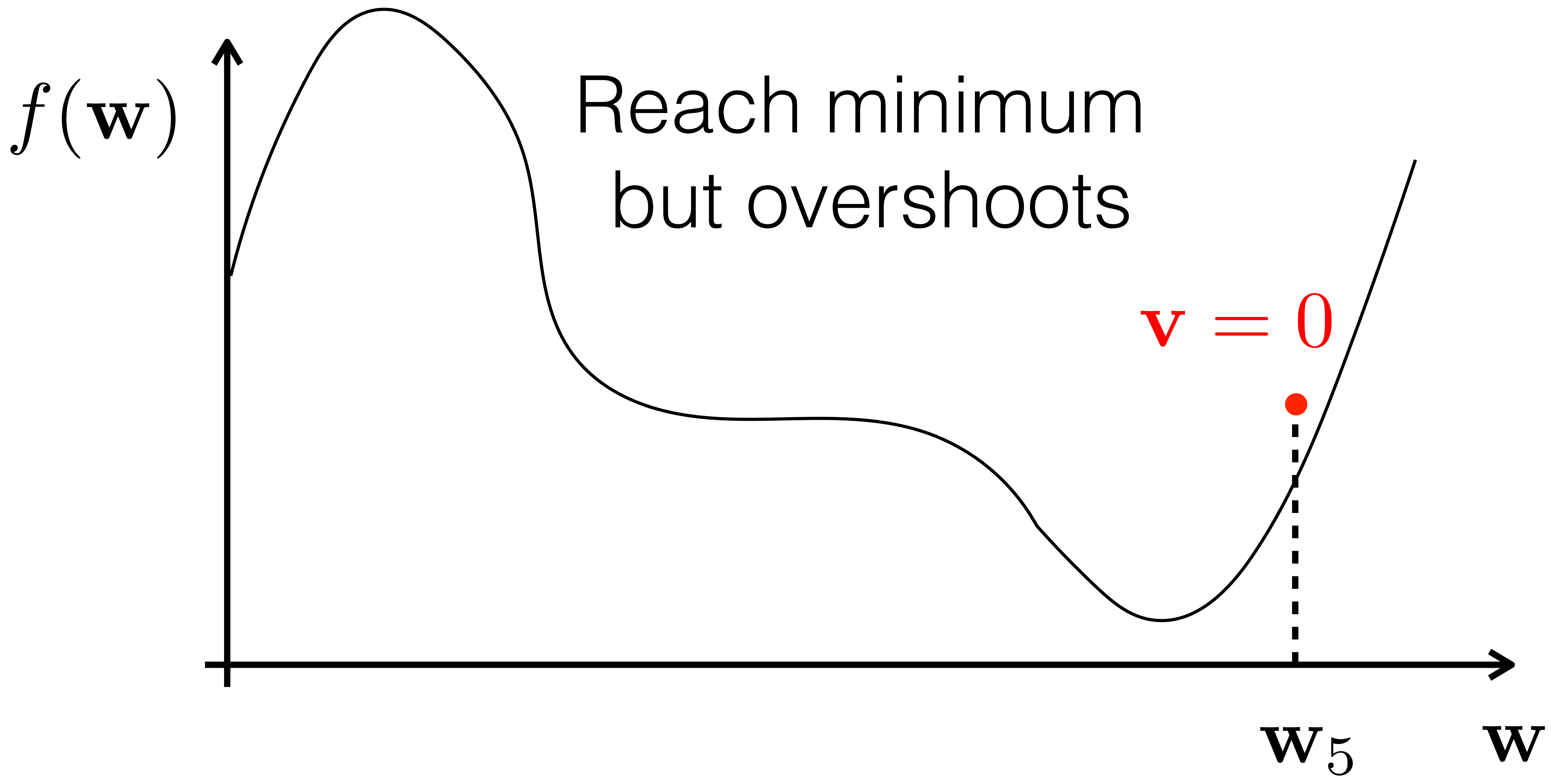


SGD + momentum

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

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

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

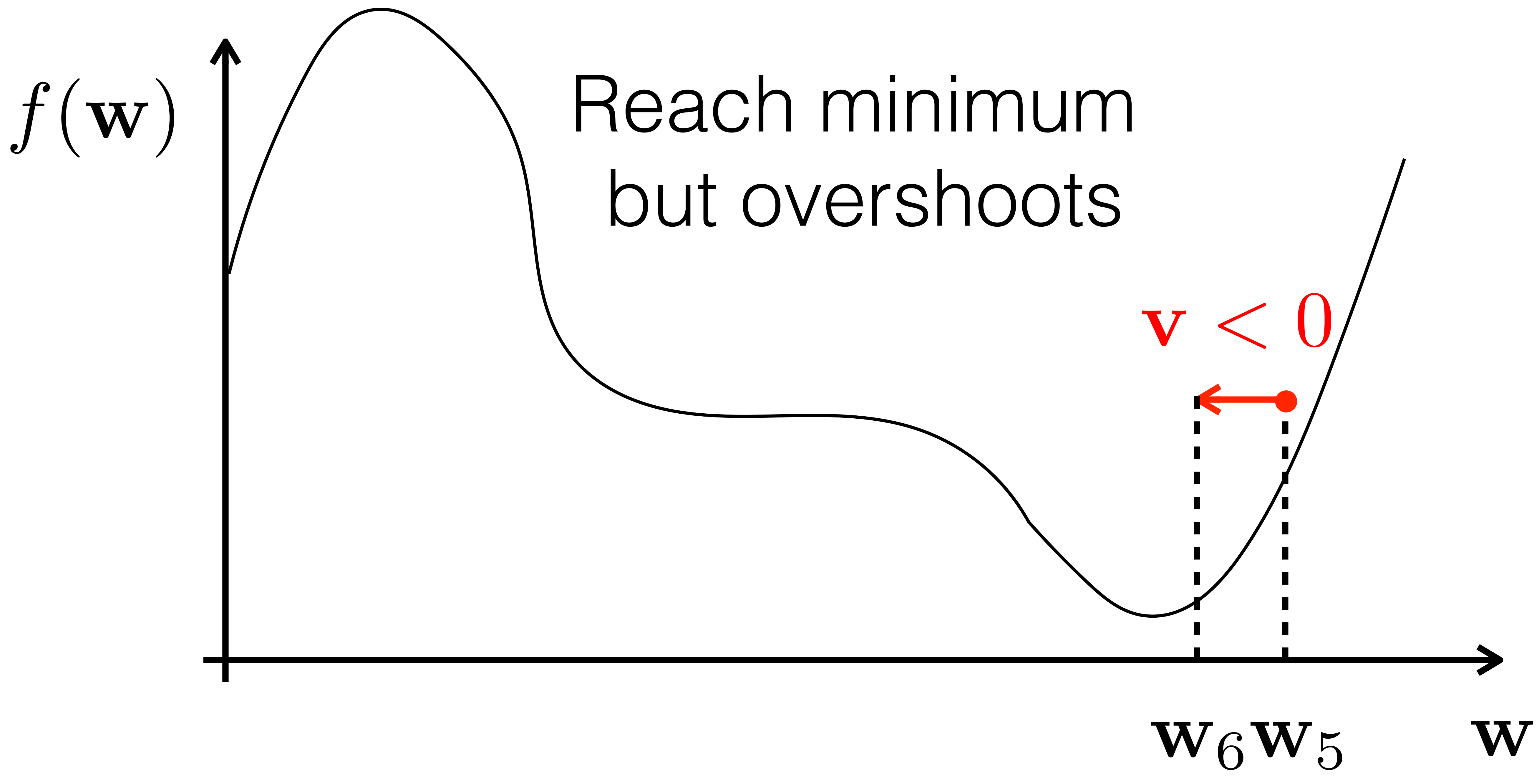


SGD + momentum

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

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

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

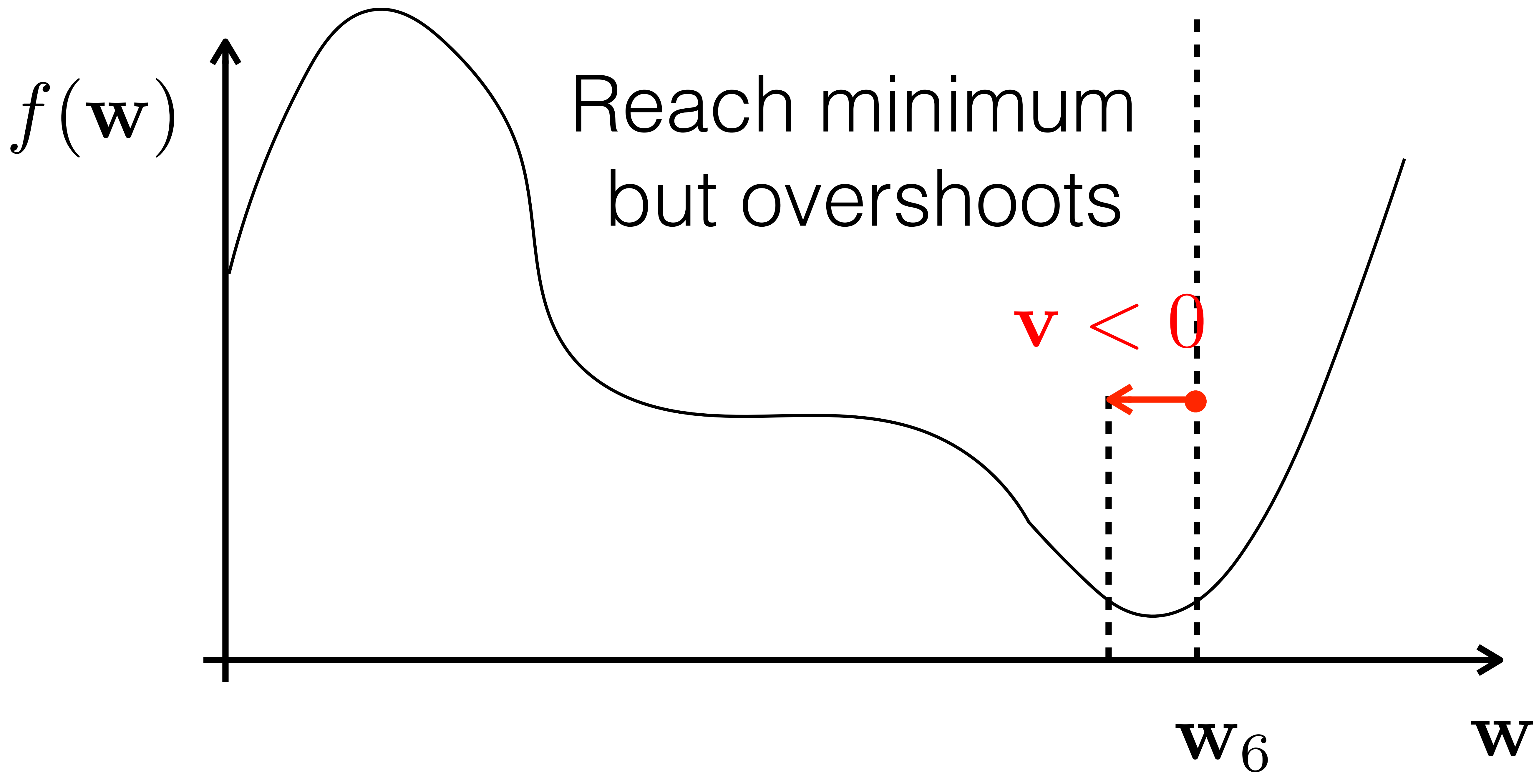


SGD + momentum

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

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

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

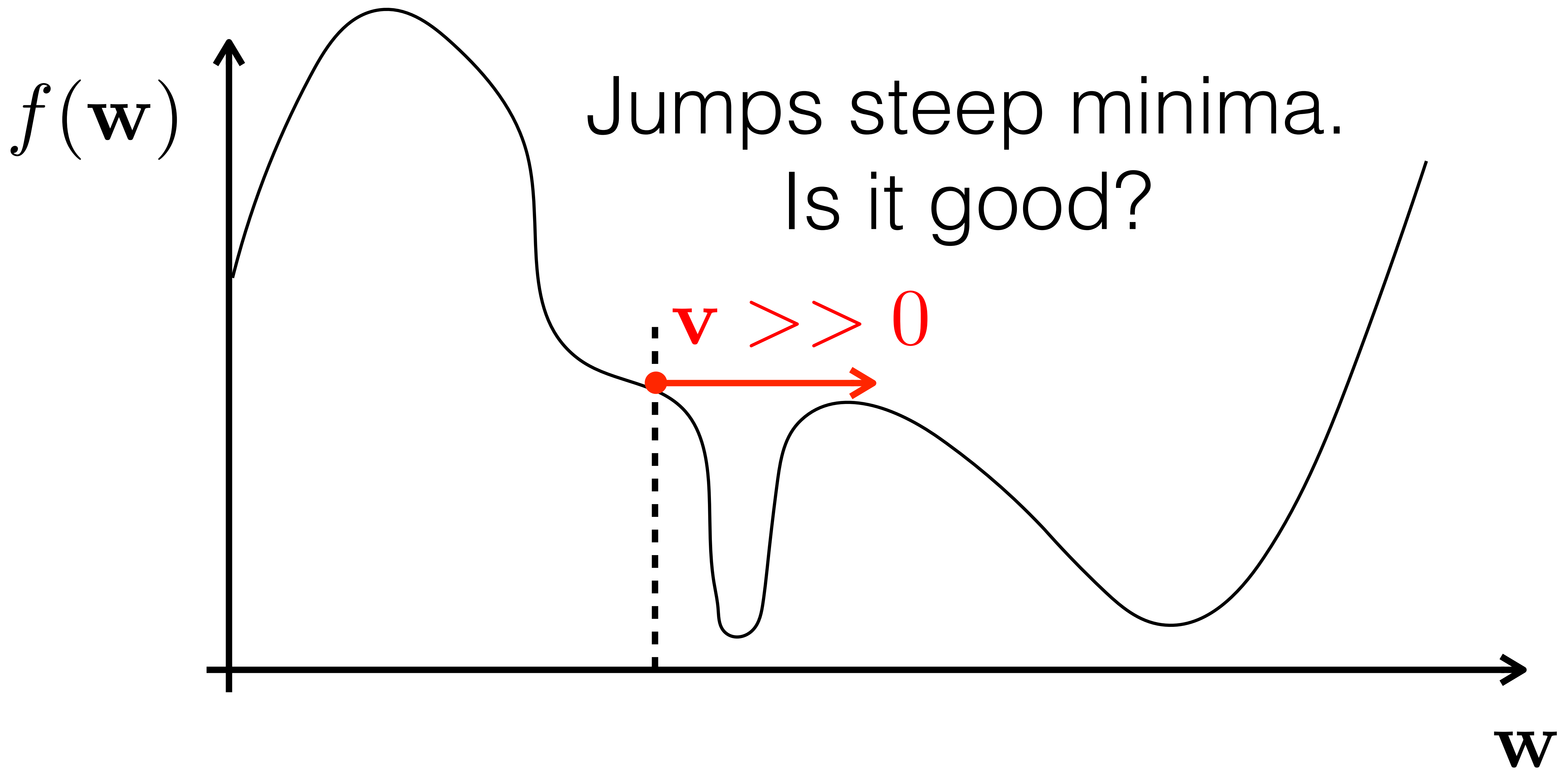


SGD + momentum

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

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

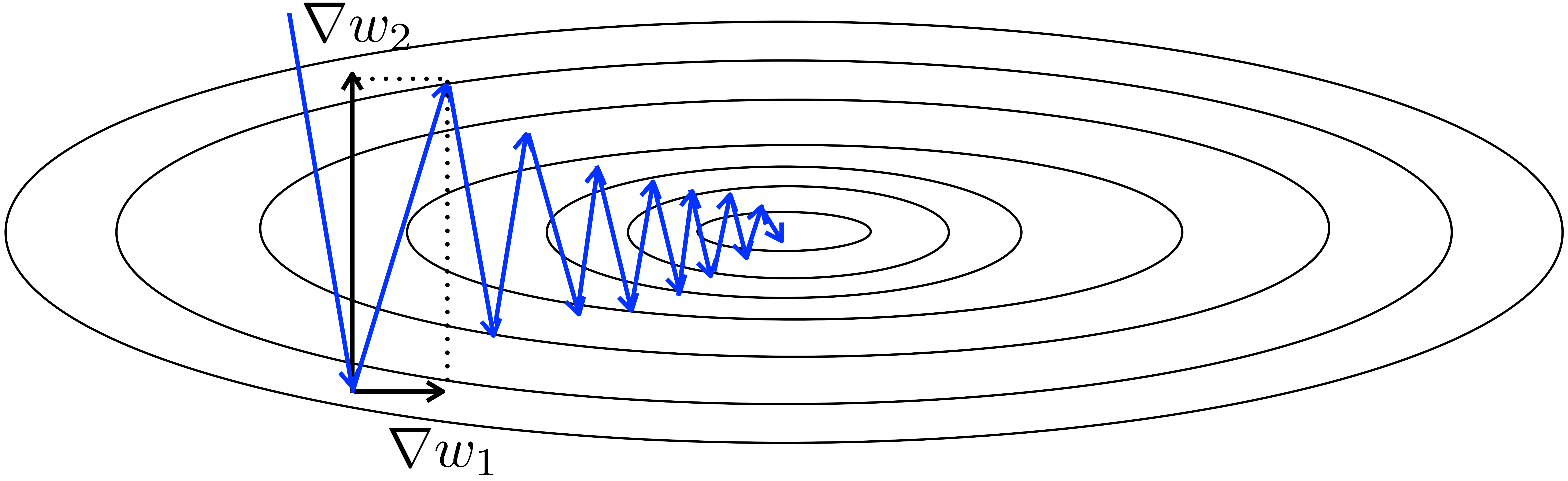
- Build velocity $\mathbf{v}$ as running average of gradients
- Rolling ball with velocity $\mathbf{v}$ and friction coeff β



“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

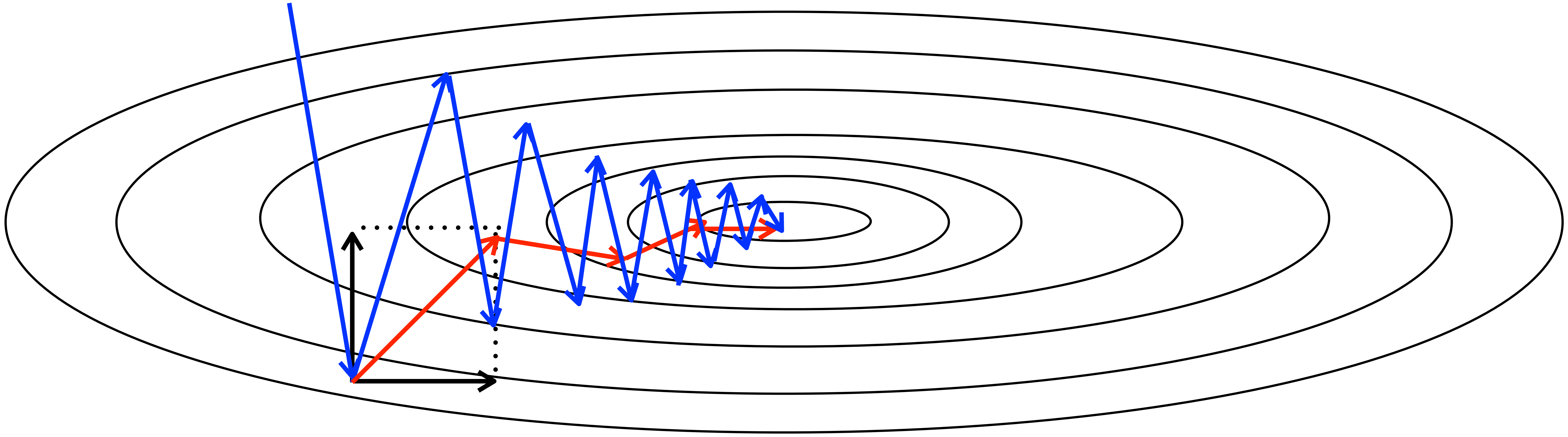


$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

“SGD” vs “SGD + momentum” in 2D

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

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



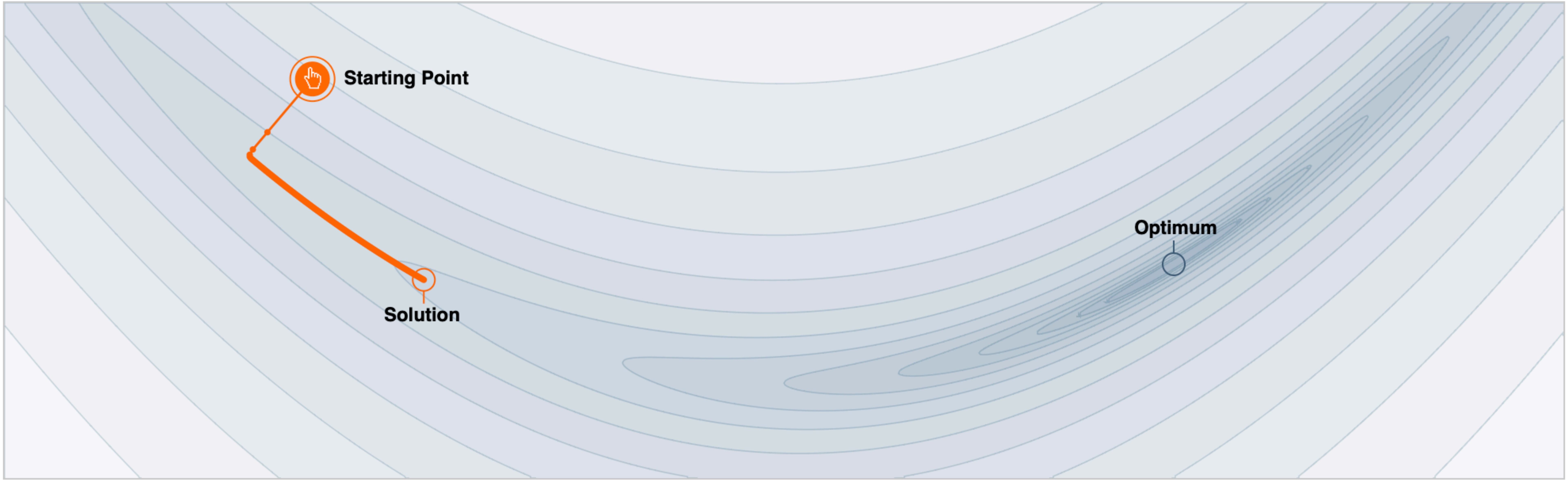
Momentum suppresses this problem partially by averaging element-wise gradients

“SGD” vs “SGD + momentum” in 2D

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

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

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



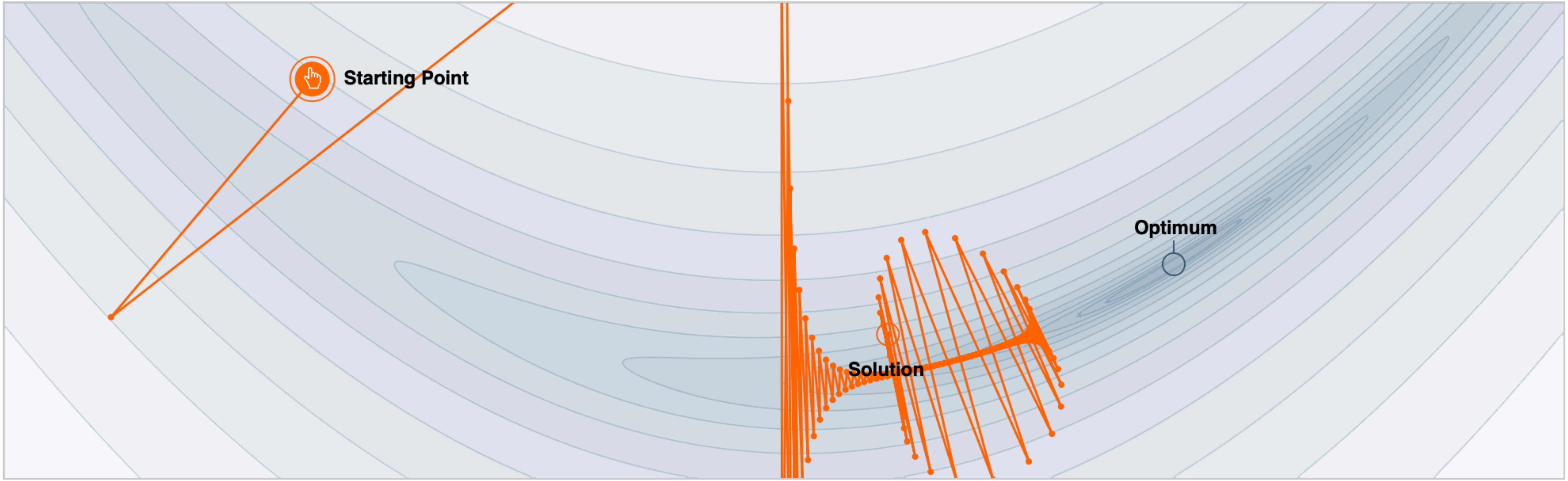
https://distill.pub/2017/momentum/

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

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

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

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



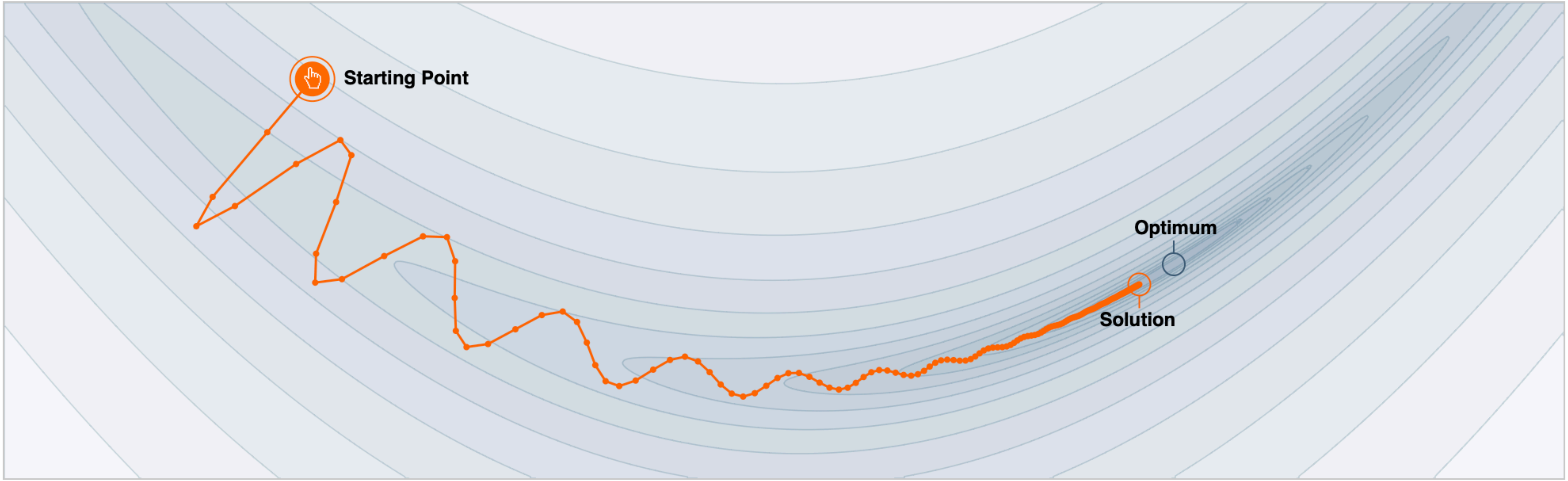
https://distill.pub/2017/momentum/

“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 $\quad\quad$ $\beta =$ 0.9



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

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()
```

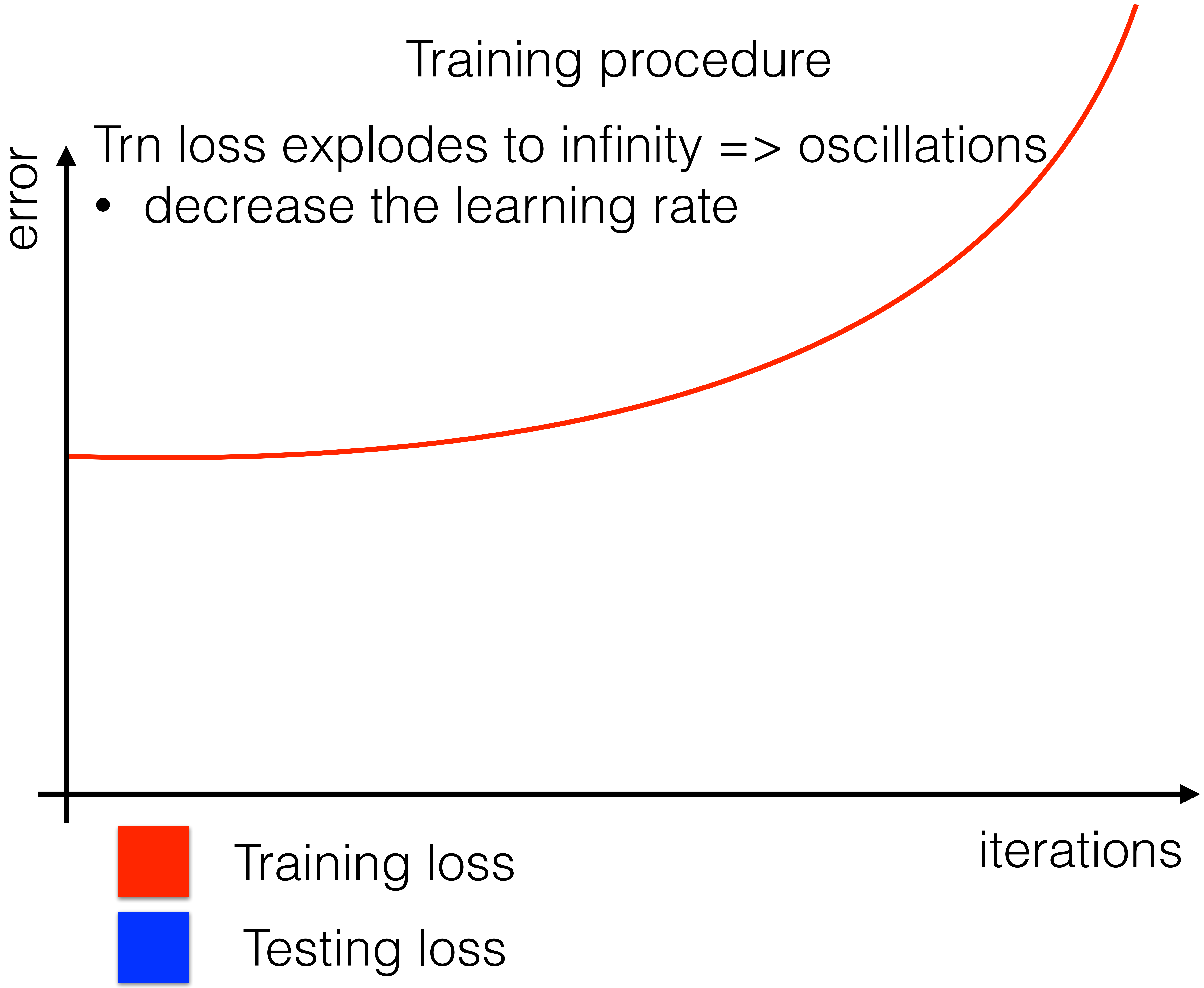
Training procedure

- Choose:
  - Weight initialization (Xavier)
  - Network architecture (ideally re-use pre-trained net)
  - 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 expected performance is measured

Training procedure

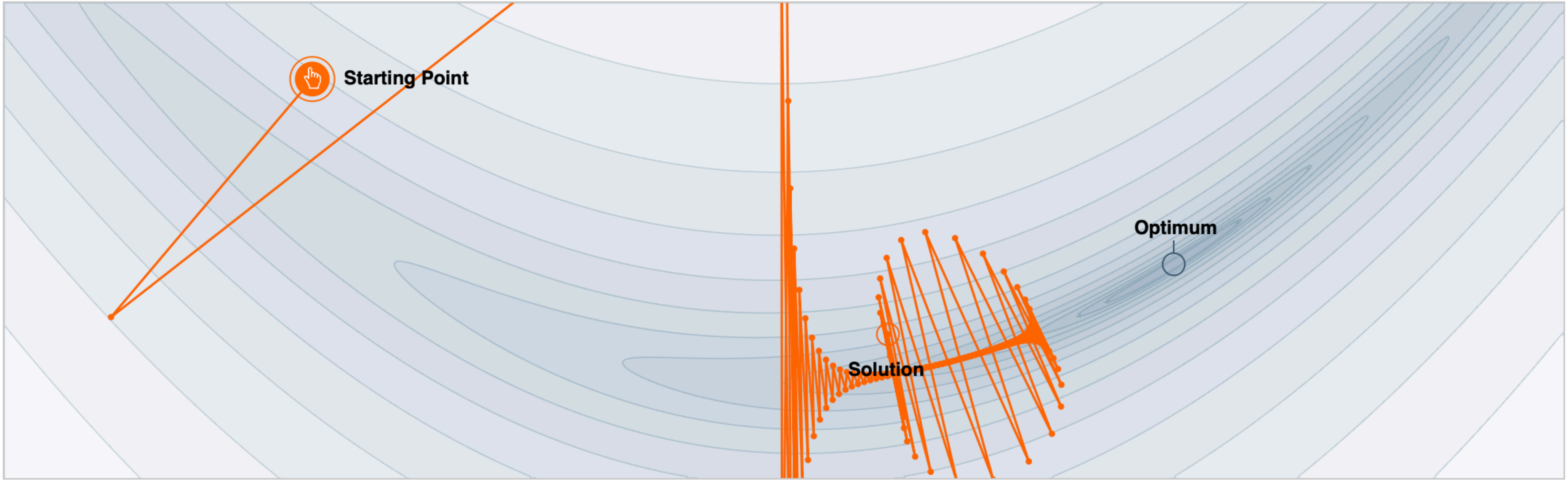
Trn loss explodes to infinity => oscillations

- decrease the learning rate



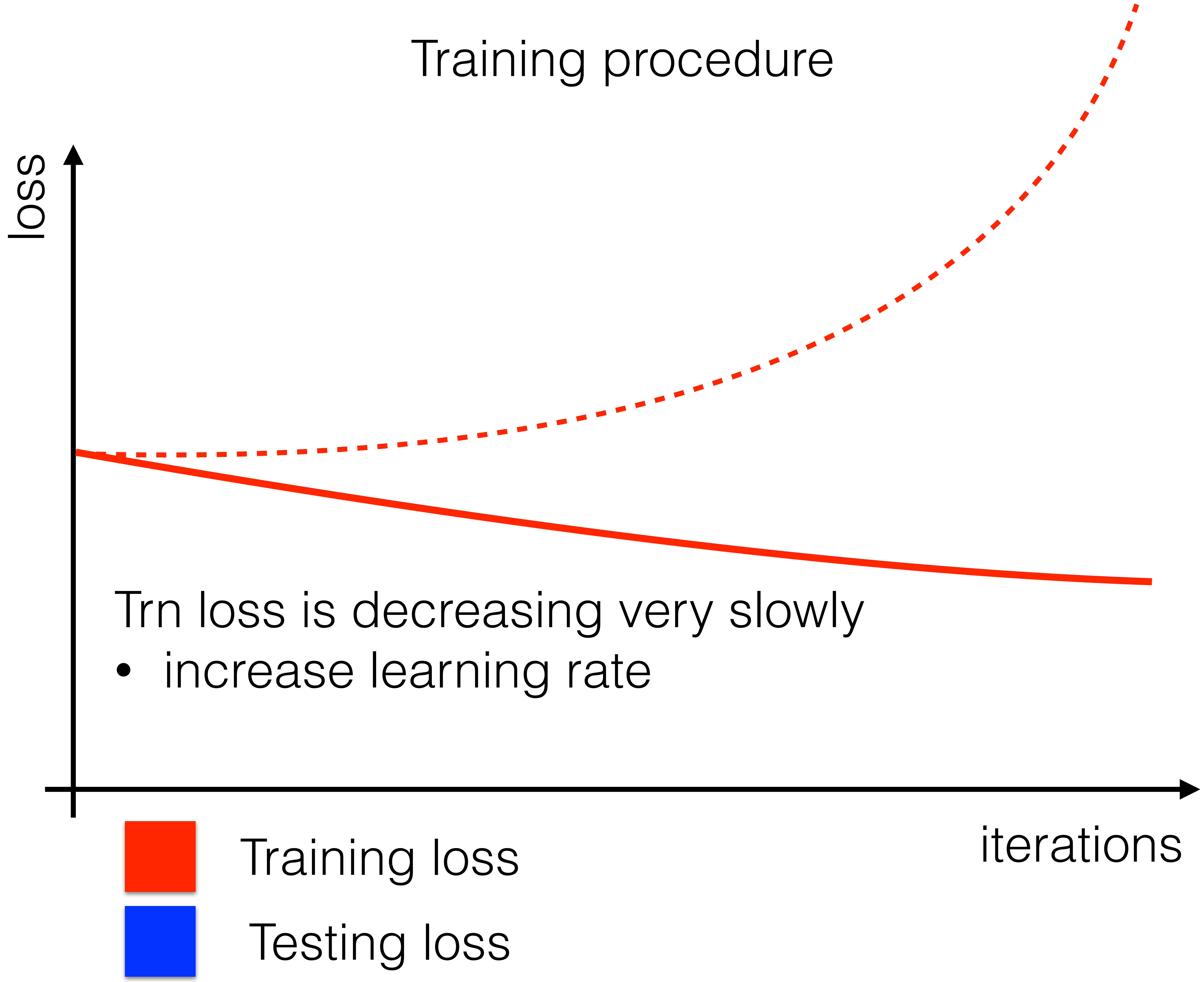
SGD drawbacks - in 2D

$\alpha = $ 5e-3



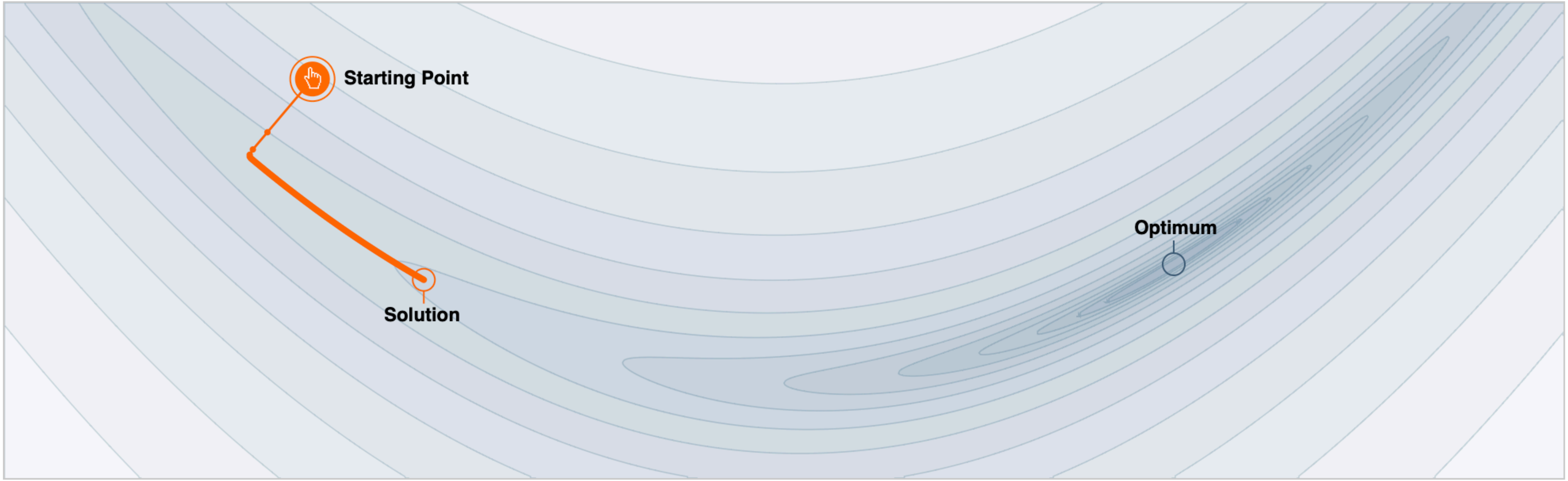
https://distill.pub/2017/momentum/

Training procedure



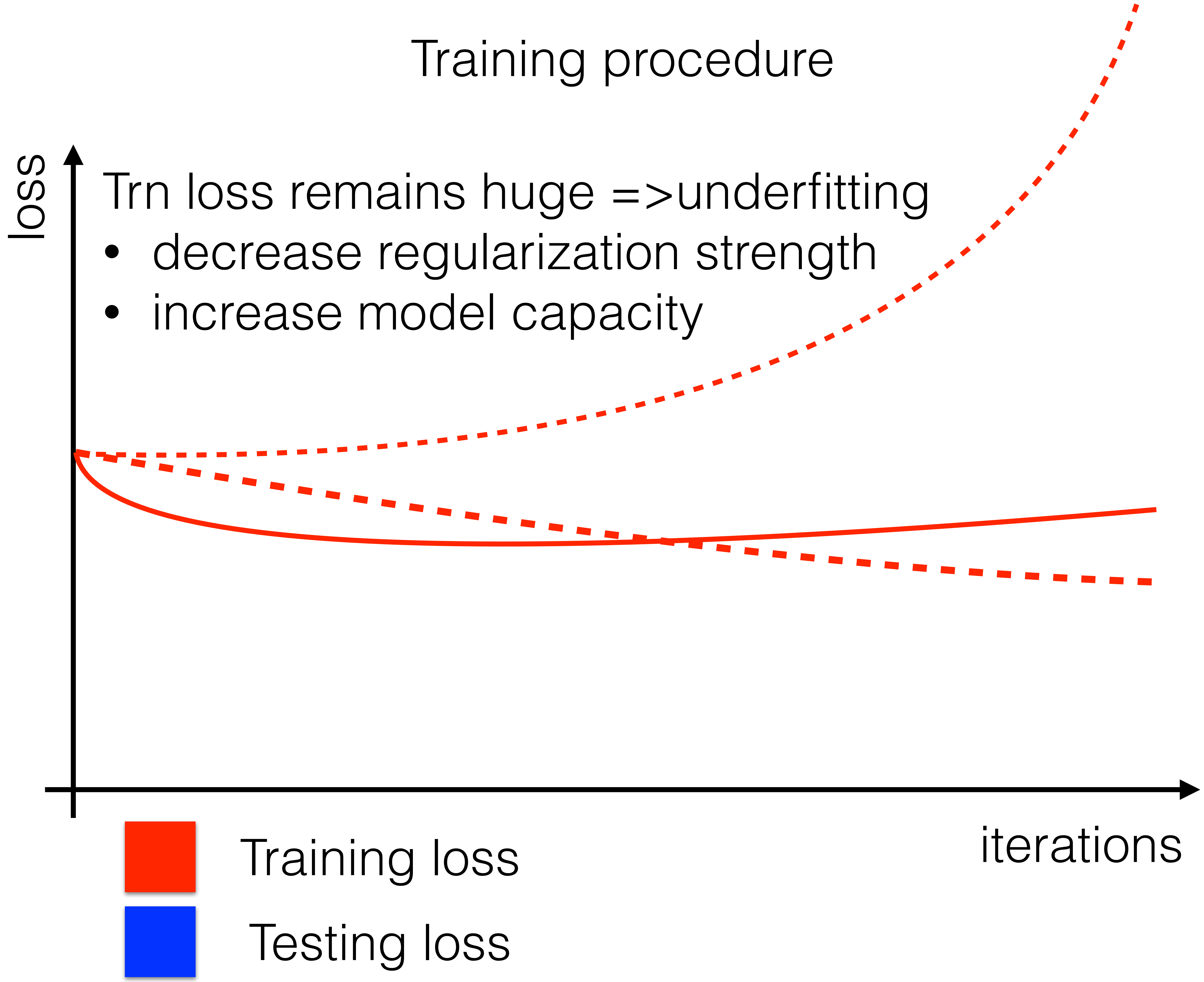
SGD drawbacks - in 2D

$\alpha =$ 1e-3



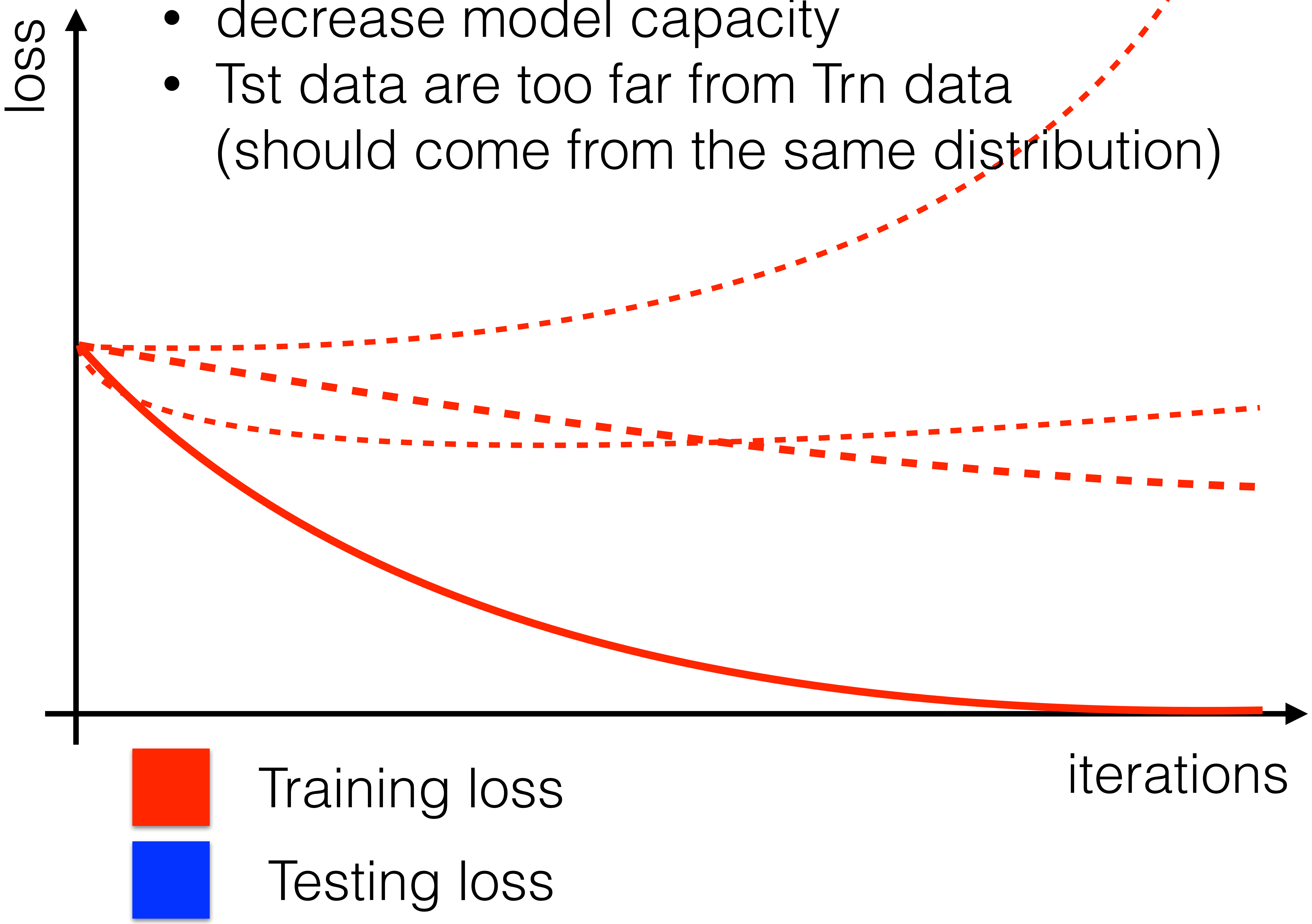
https://distill.pub/2017/momentum/

Training procedure



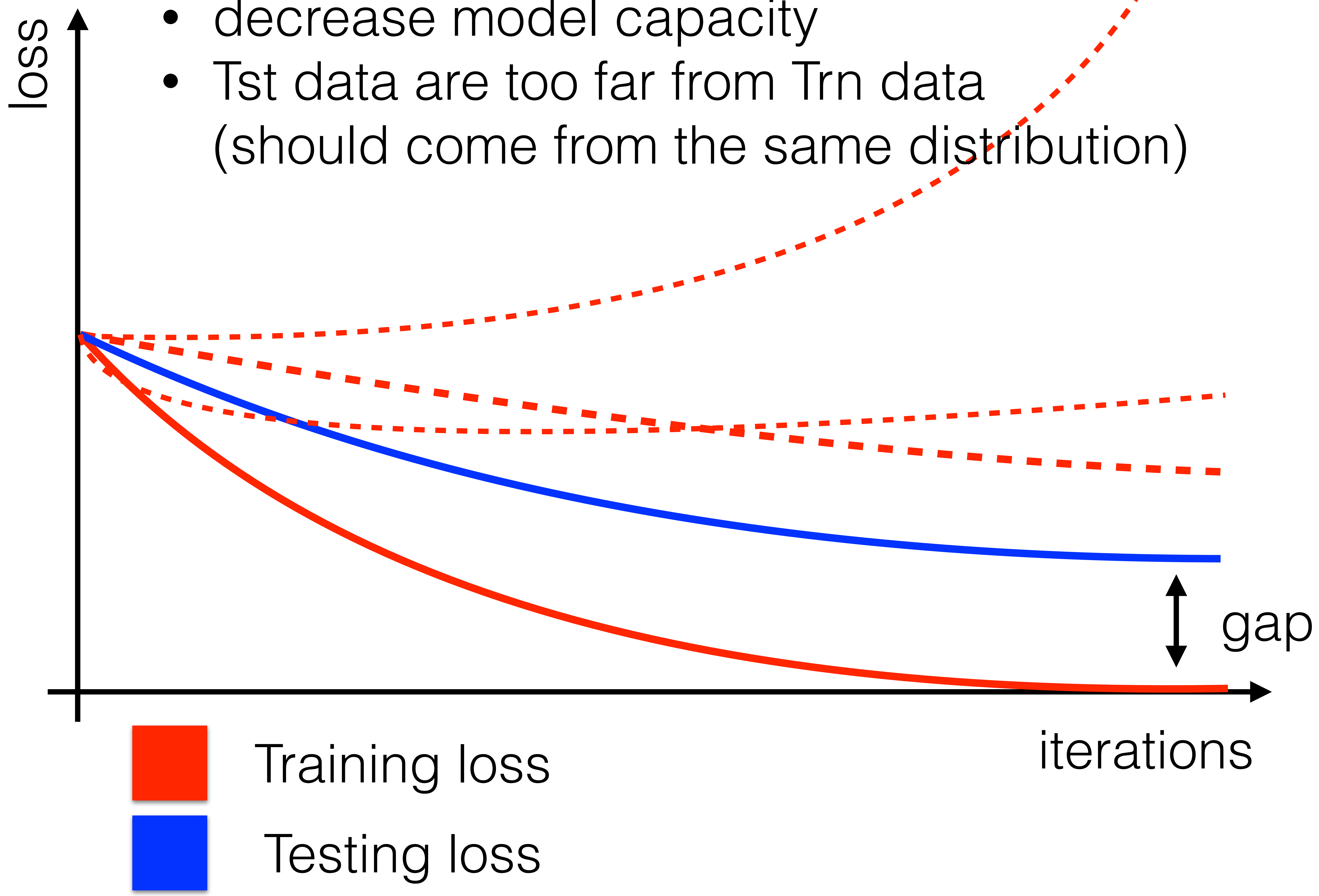
Tst loss>>Trn loss => overfitting

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



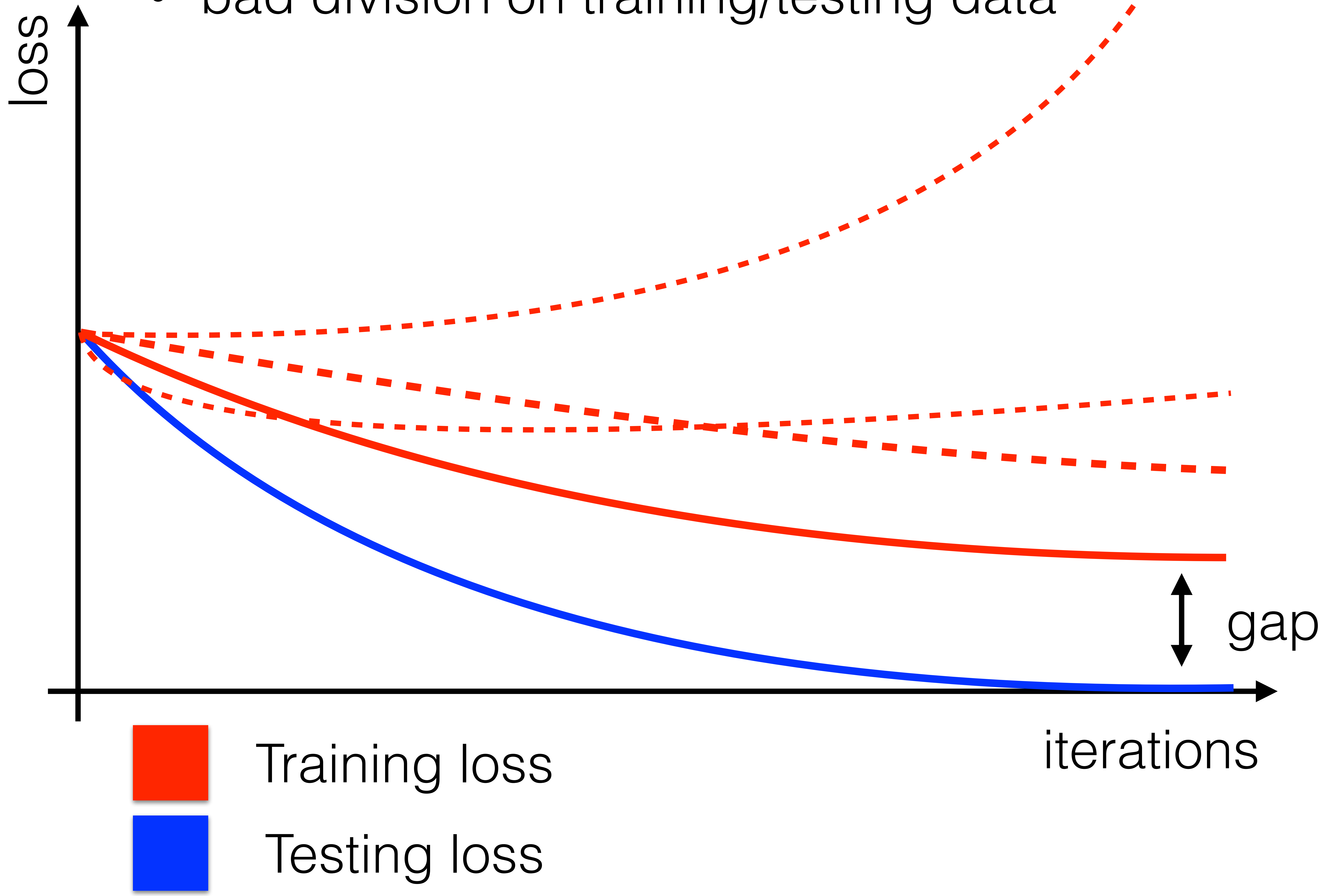
Tst loss>>Trn loss => overfitting

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

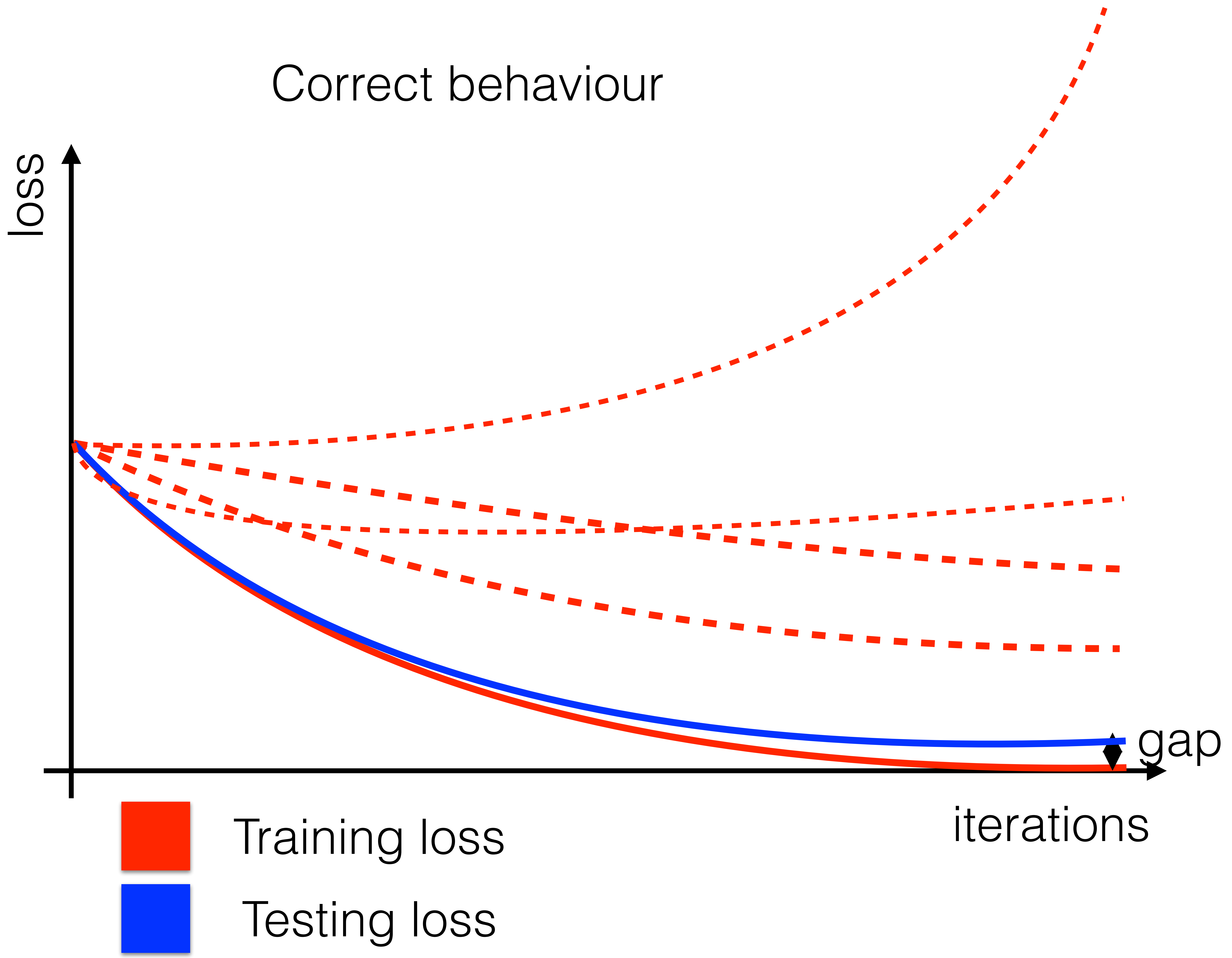


Trn loss>>Tst loss

- bad division on training/testing data



Correct behaviour



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

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:

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

CLS
CARS

GT
BKGD:

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

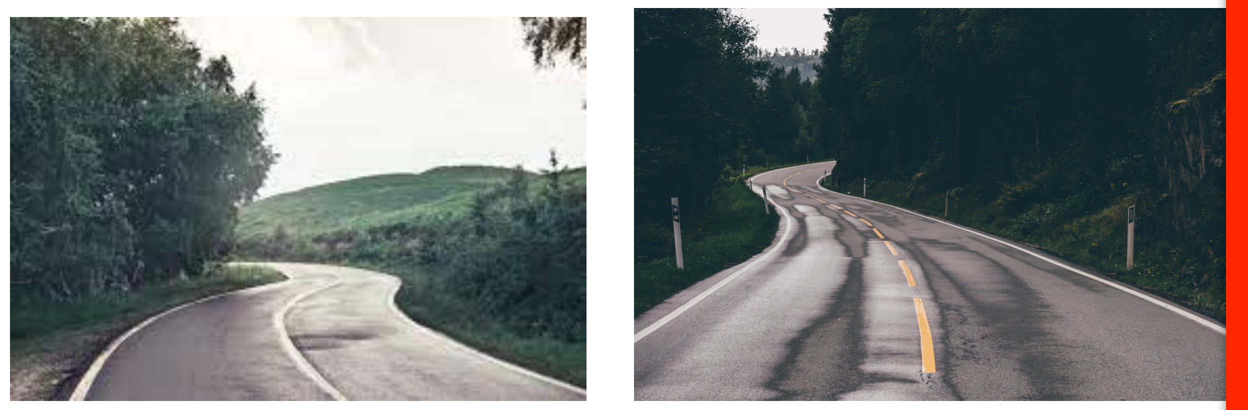
Binary classifier testing presence of potentially dangerous case:

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:

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:

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


GT BKGD:




CLS CARS



CLS BGGD:



false negative (FN) =1

false positive (FP) =2

true positive (TP) =1

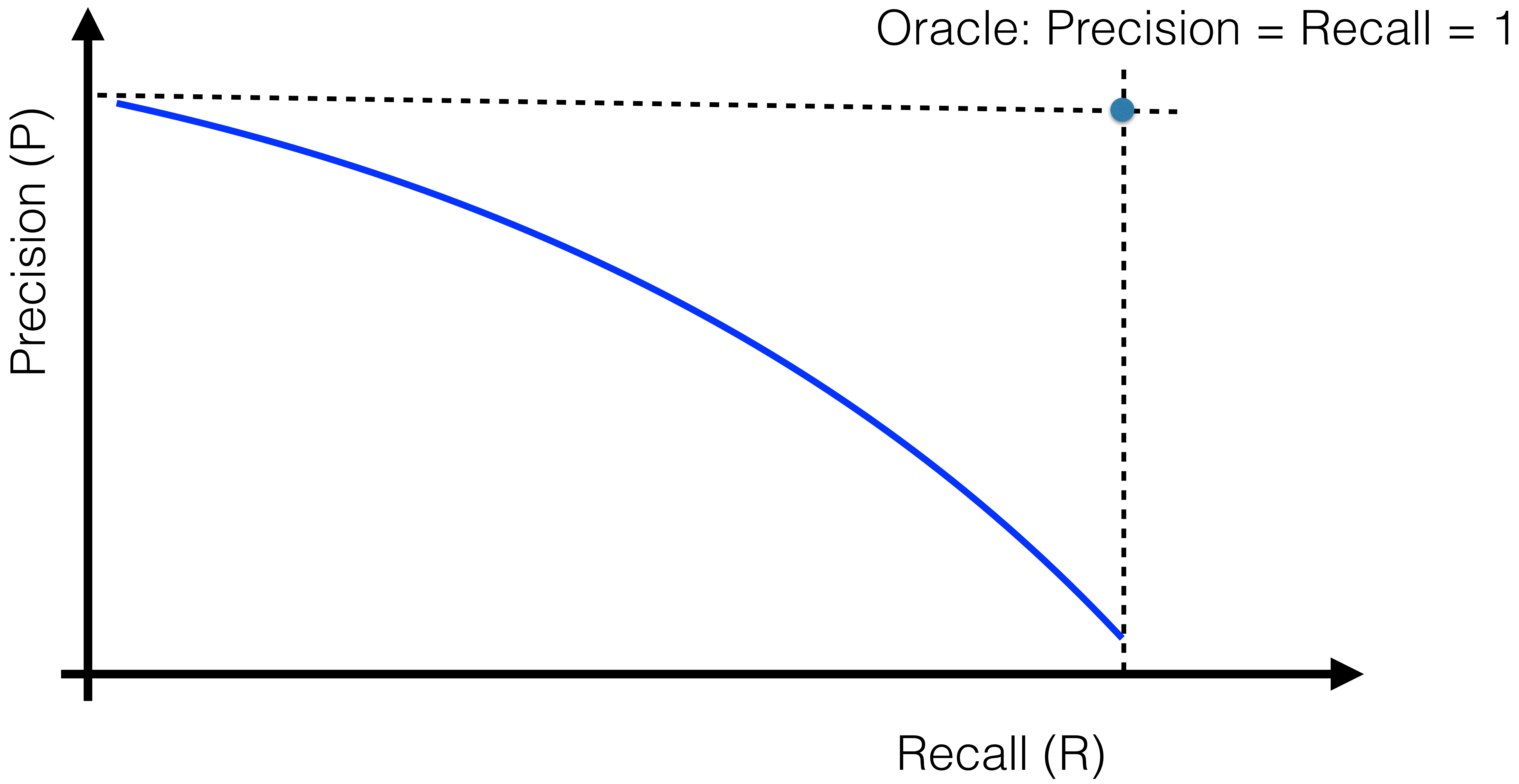
true negative (TN) =2

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

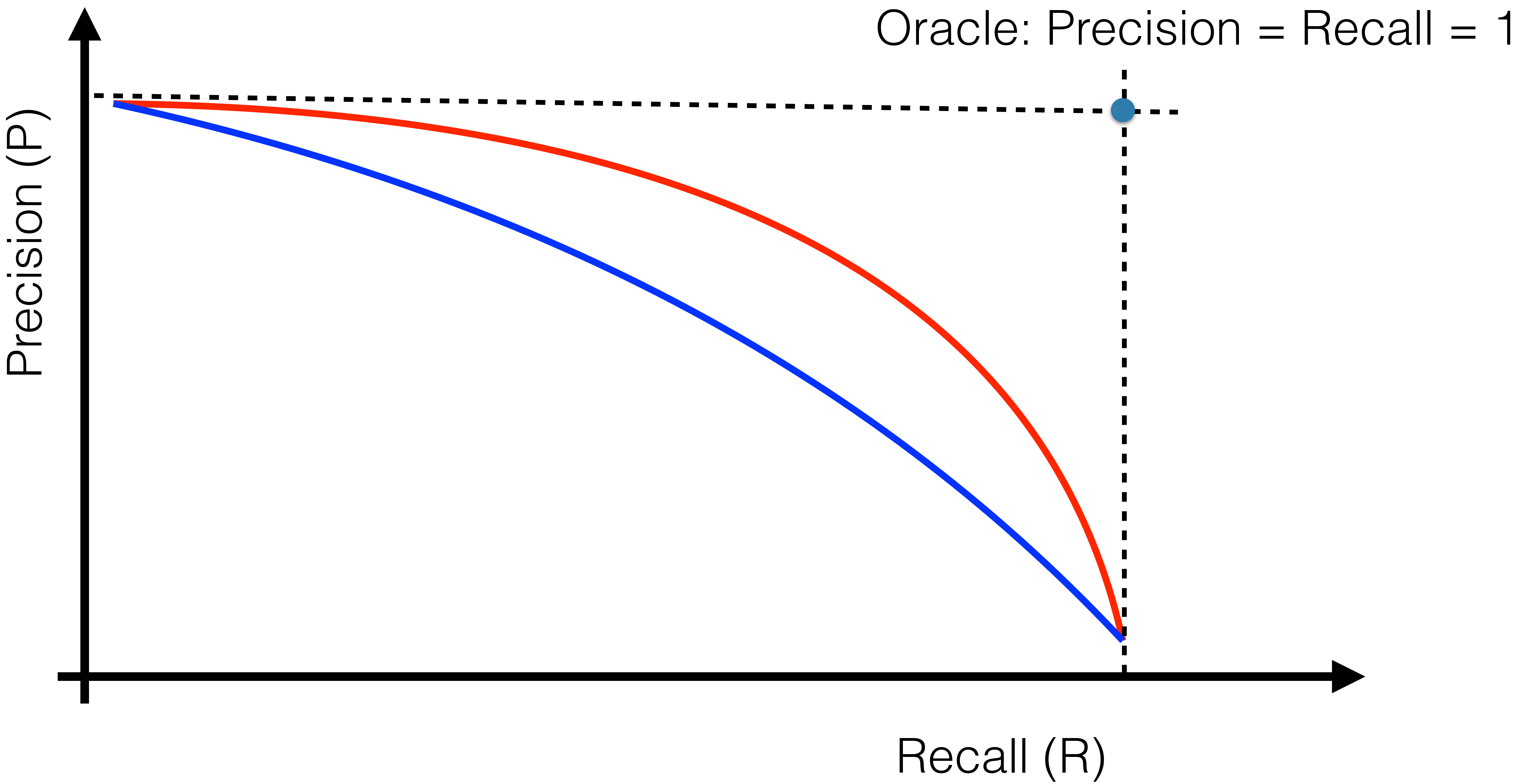
$$\text{Recall (R)} = \frac{TP}{TP + FN} = \frac{1}{1 + 1} = 1/2$$

Oracle: Precision = Recall = 1

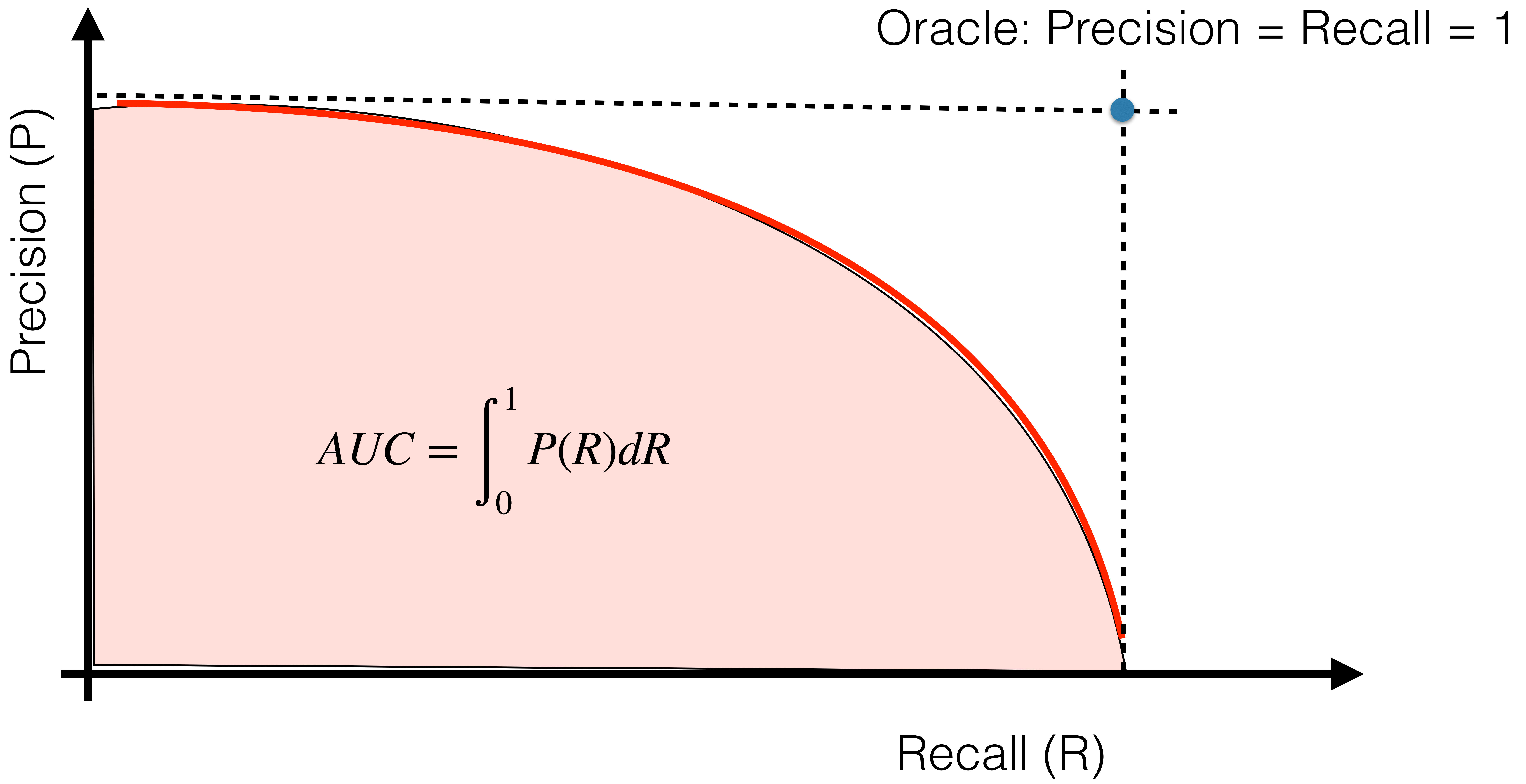
Smoothed Precision-Recall curve



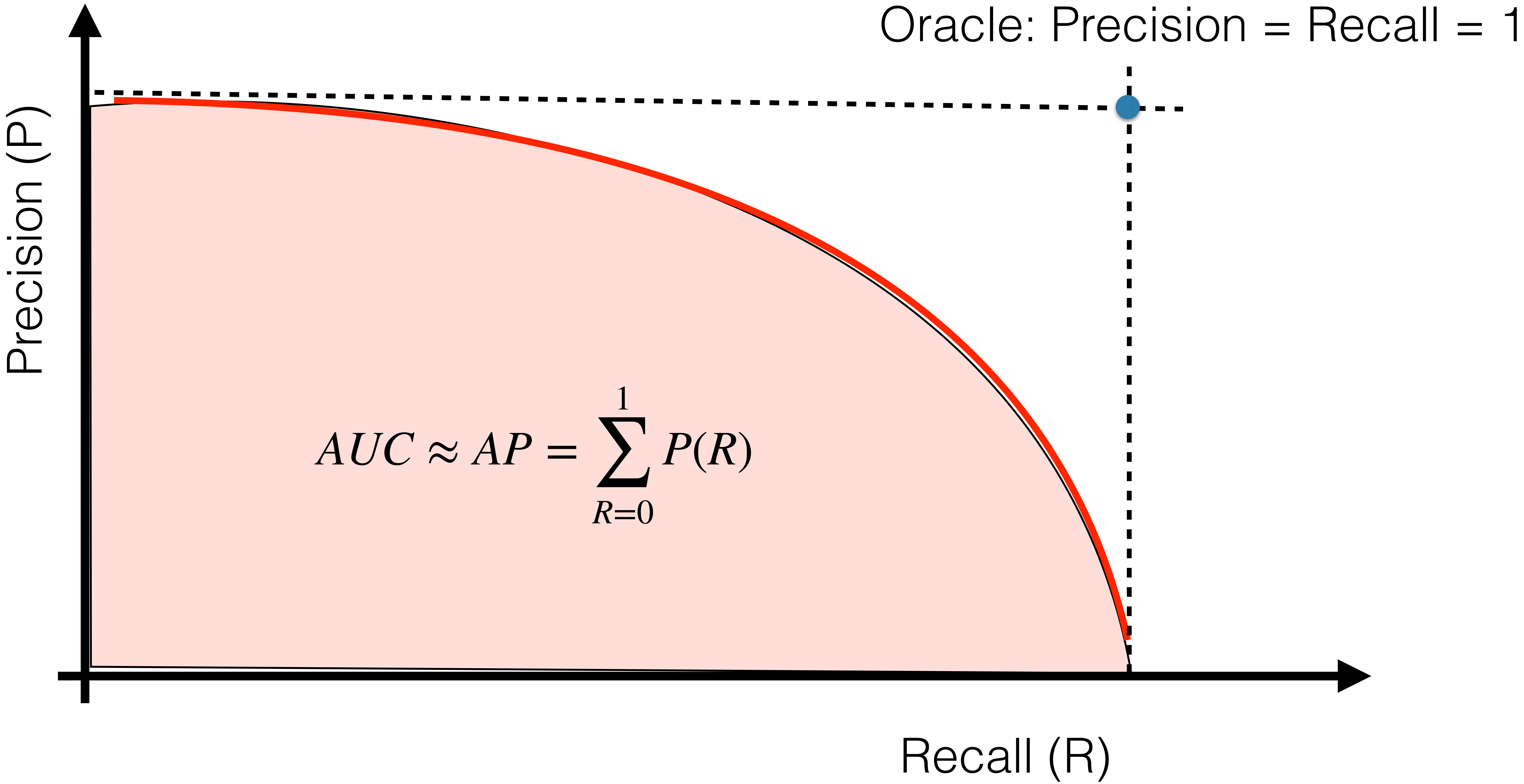
Smoothed Precision-Recall curve



Smoothed Precision-Recall curve

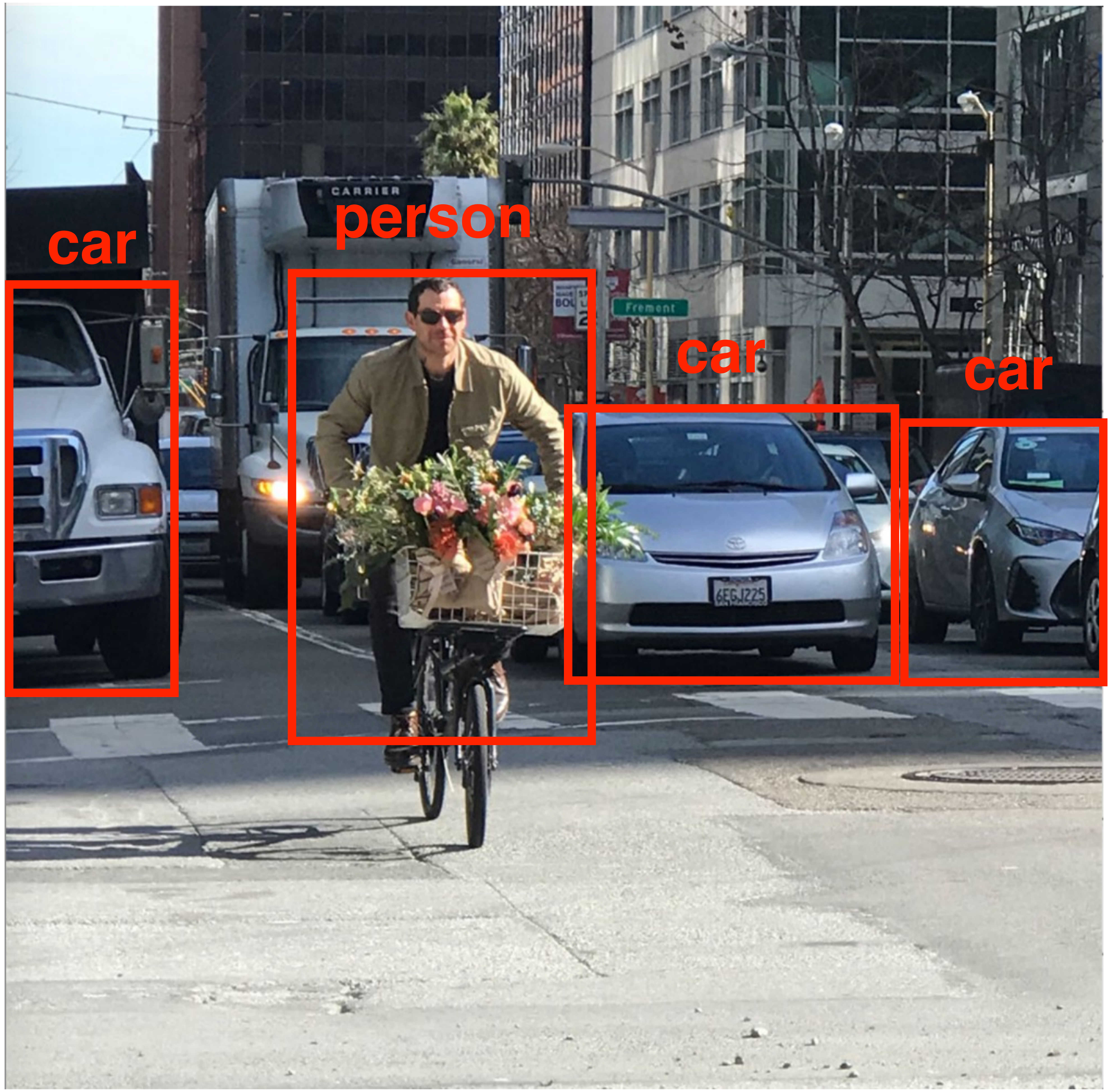


Smoothed Precision-Recall curve

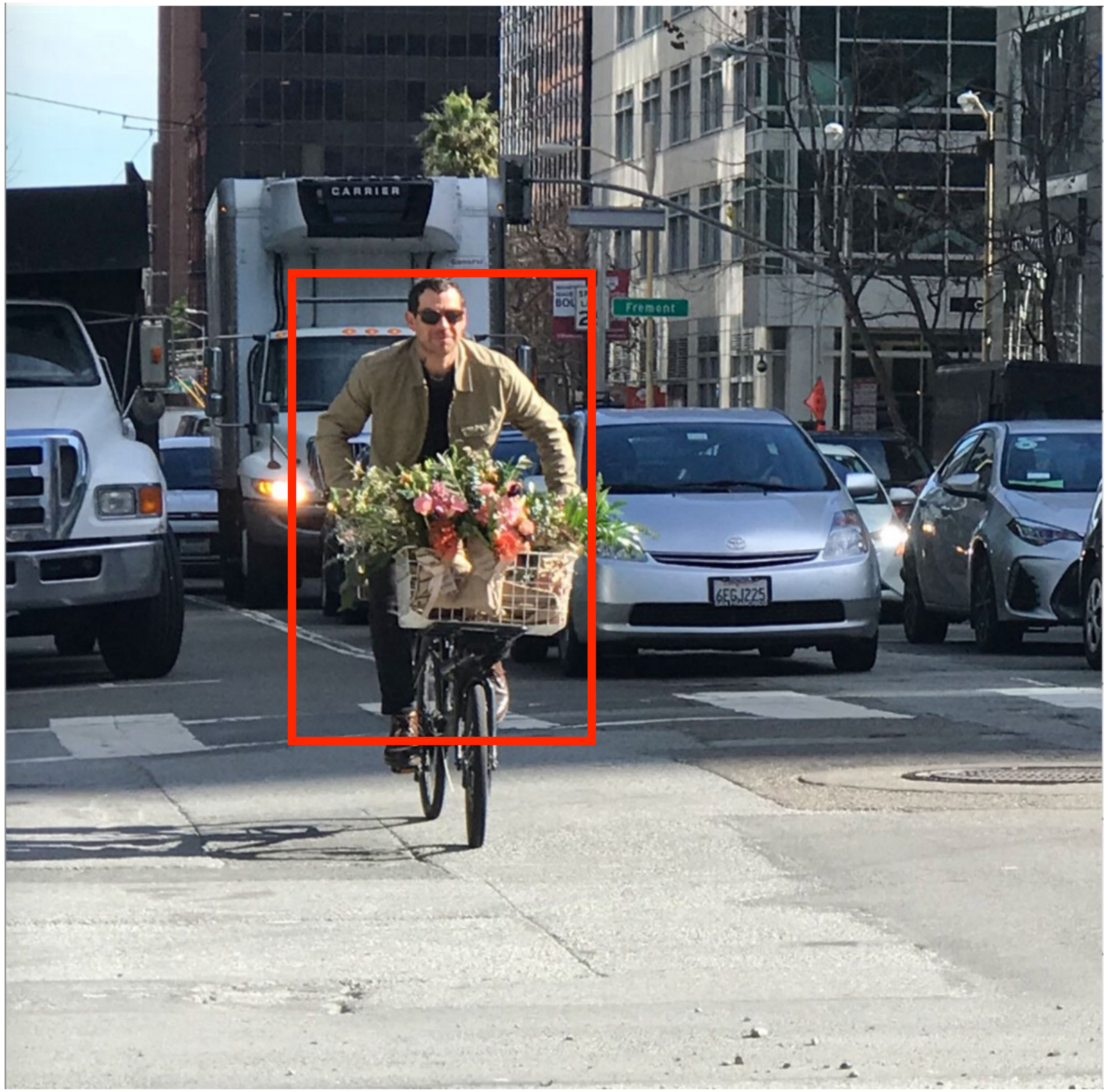


Object detection

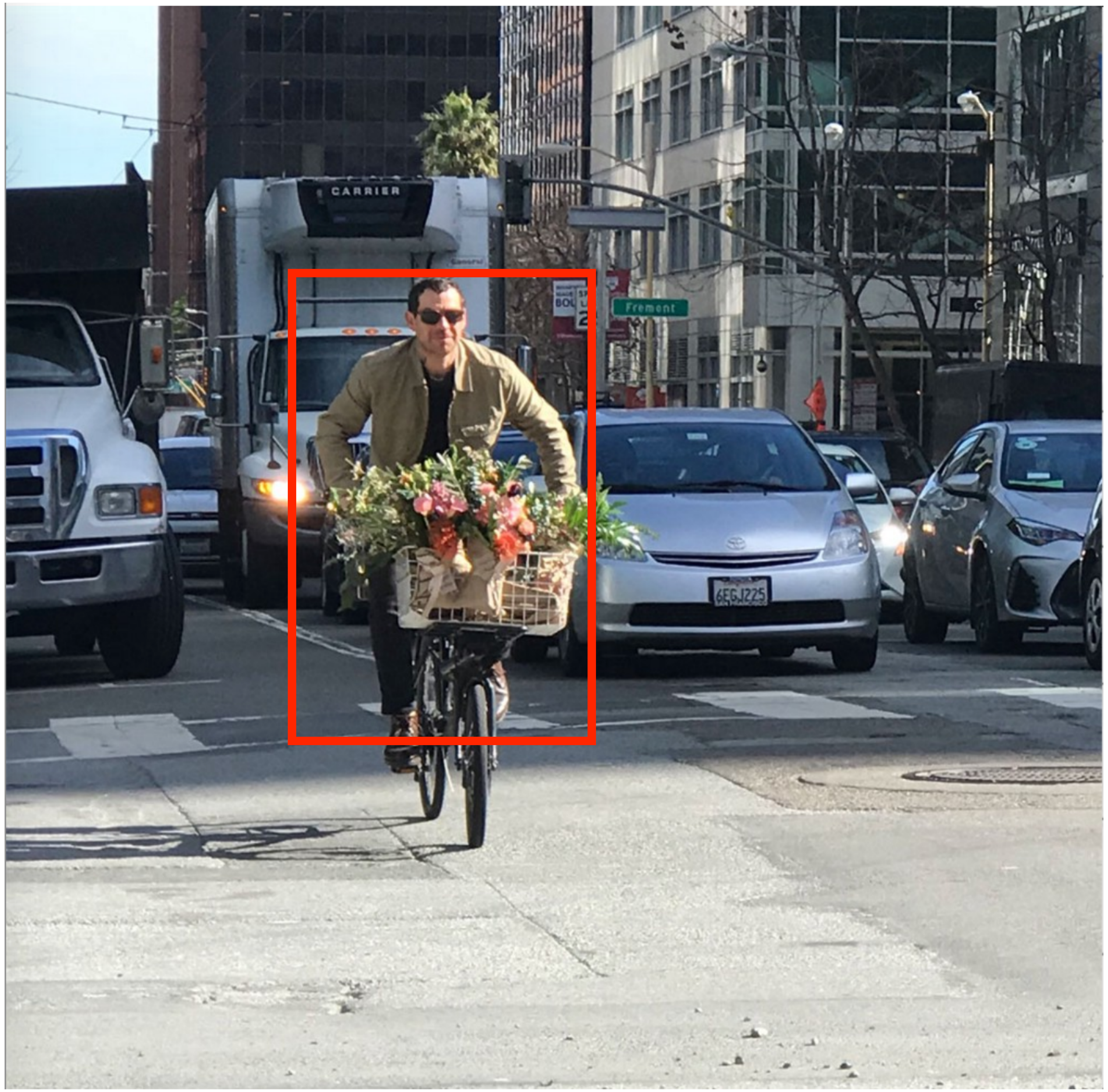
Object detection

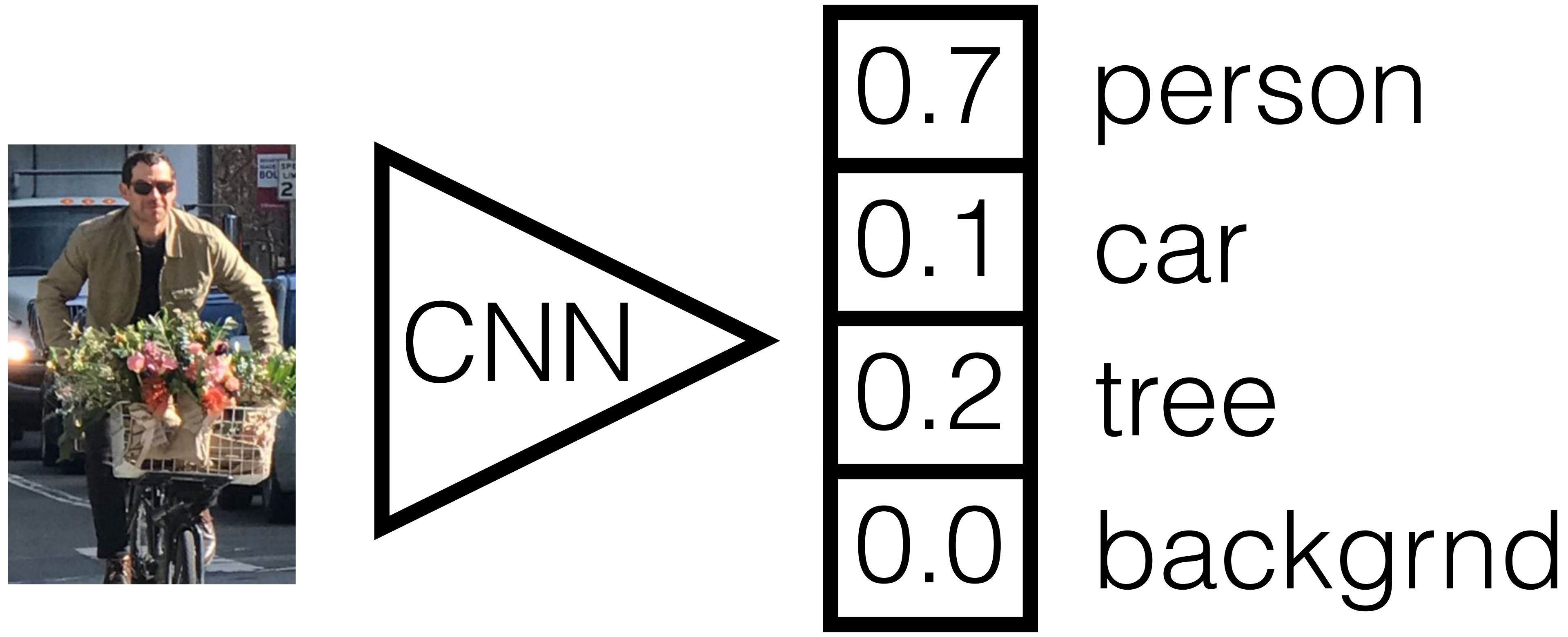


Object detection



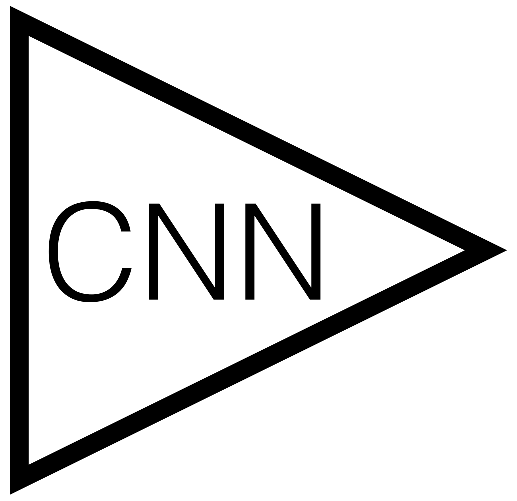
Object detection





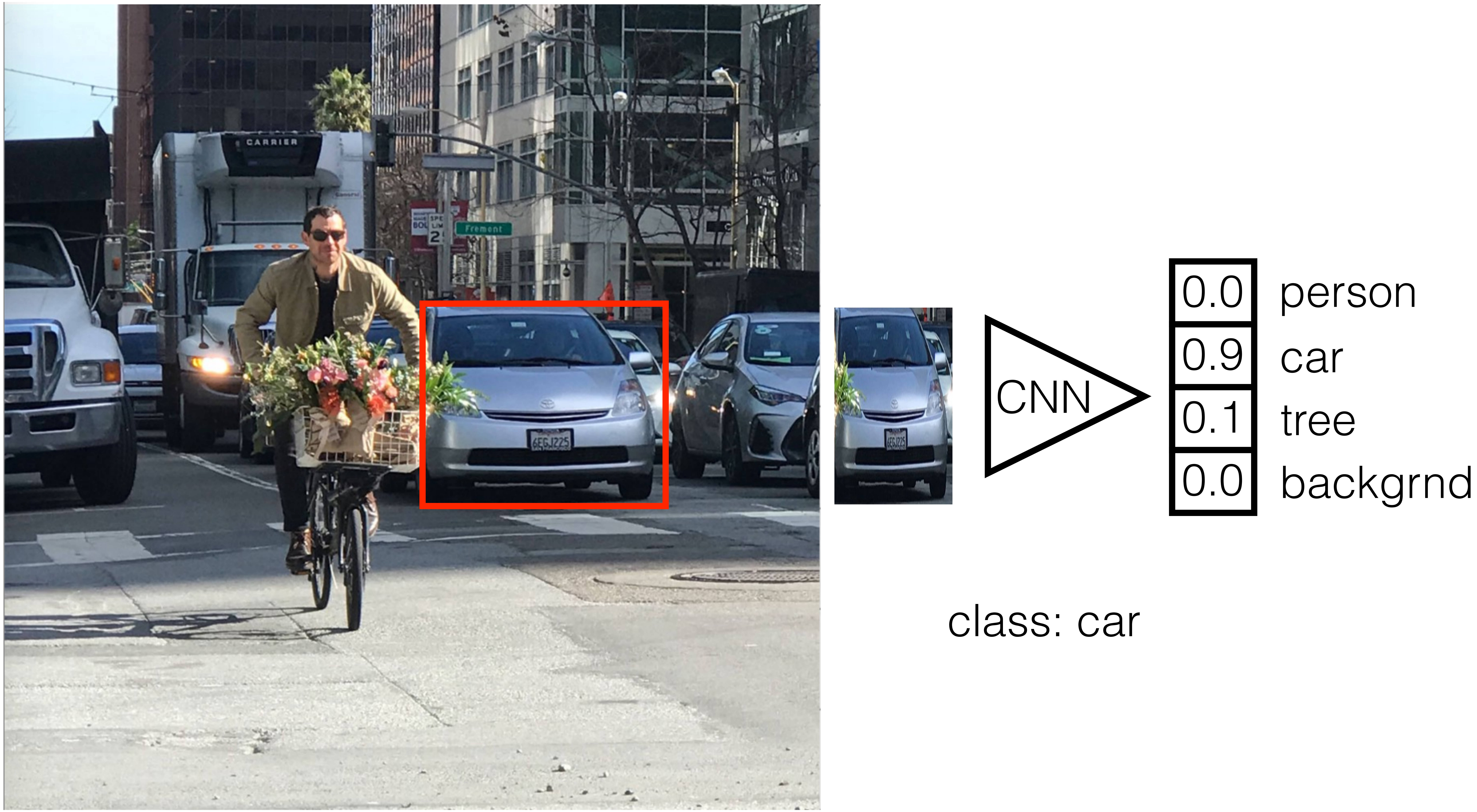
class: person

Object detection



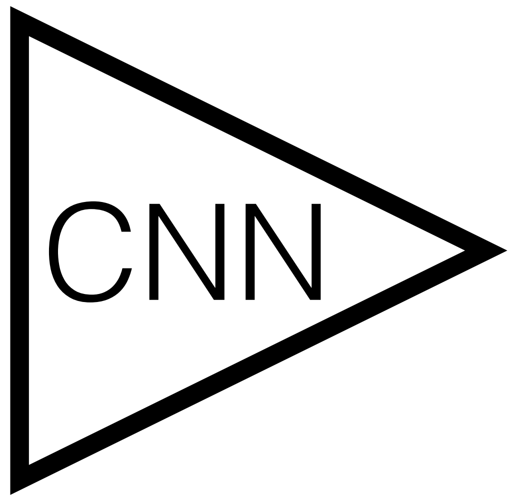
0.7	person
0.1	car
0.2	tree
0.0	backgrnd

Object detection



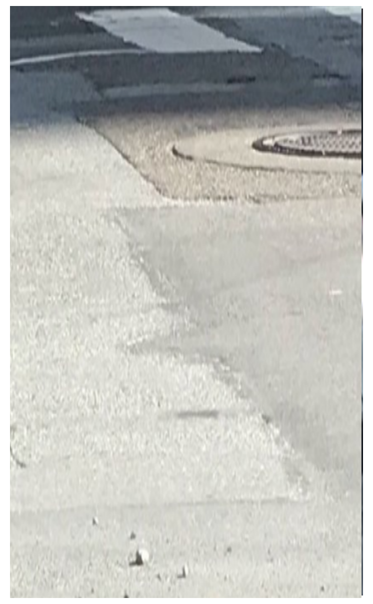
class: car

Object detection



0.0	person
0.9	car
0.1	tree
0.0	backgrnd

Object detection

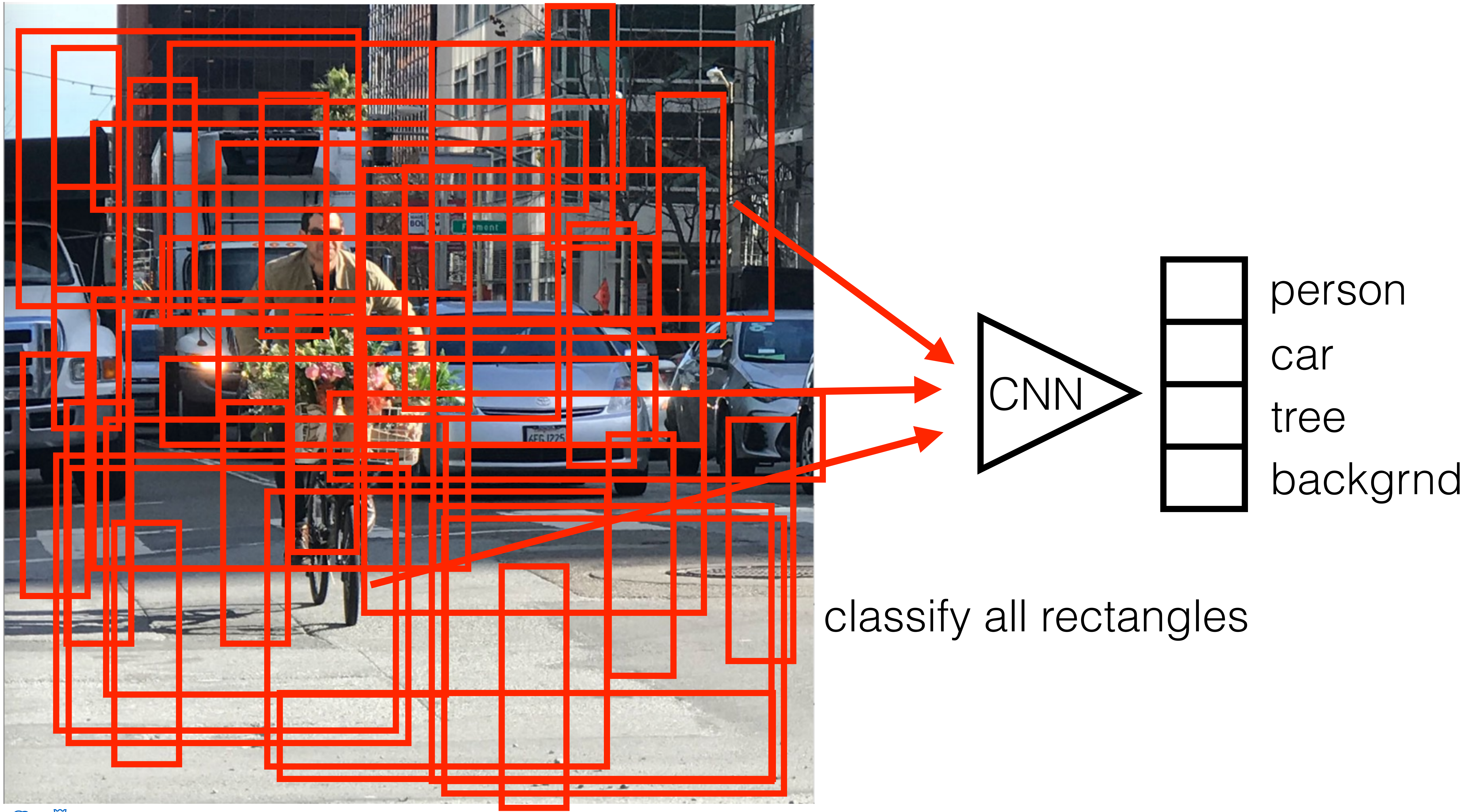


CNN

0.0	person
0.1	car
0.0	tree
0.9	backgrnd

class: background

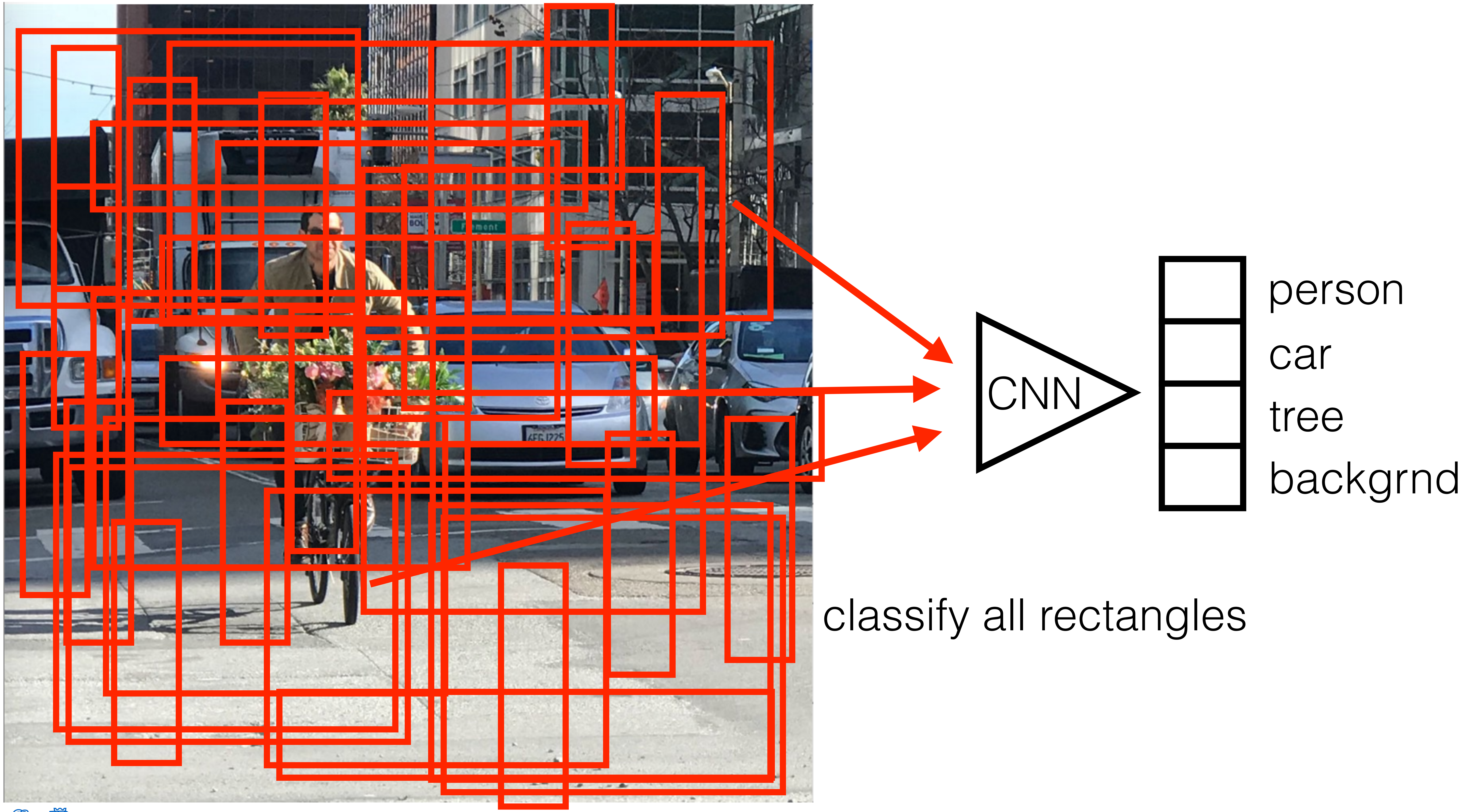
Object detection



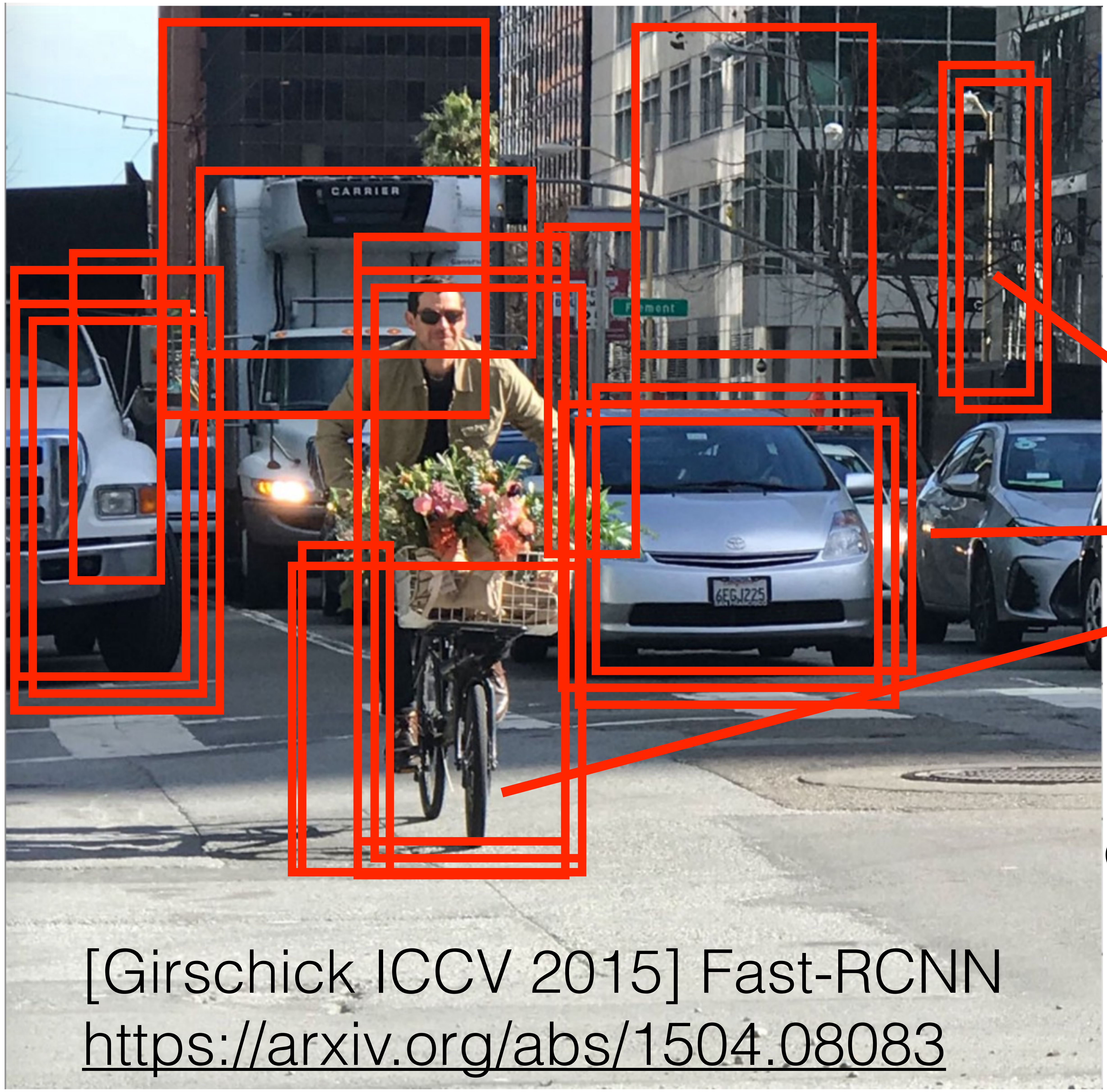
Object detection

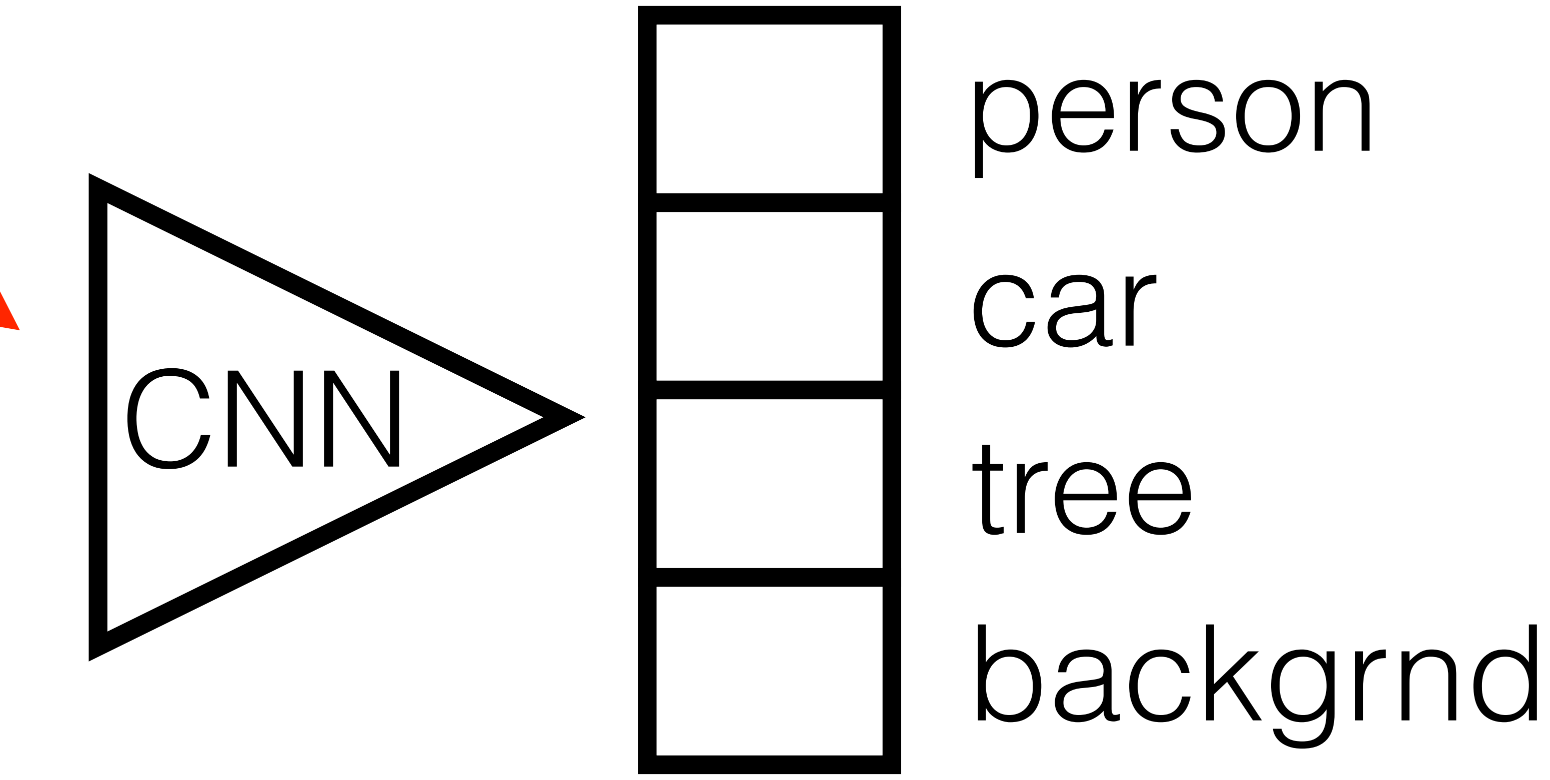
- Approach works but it takes extremely long to compute response on all rectangular sub-windows:
  H x W x Aspect_Ratio x Scales x 0.001 sec = **months**

Object detection



Object detection



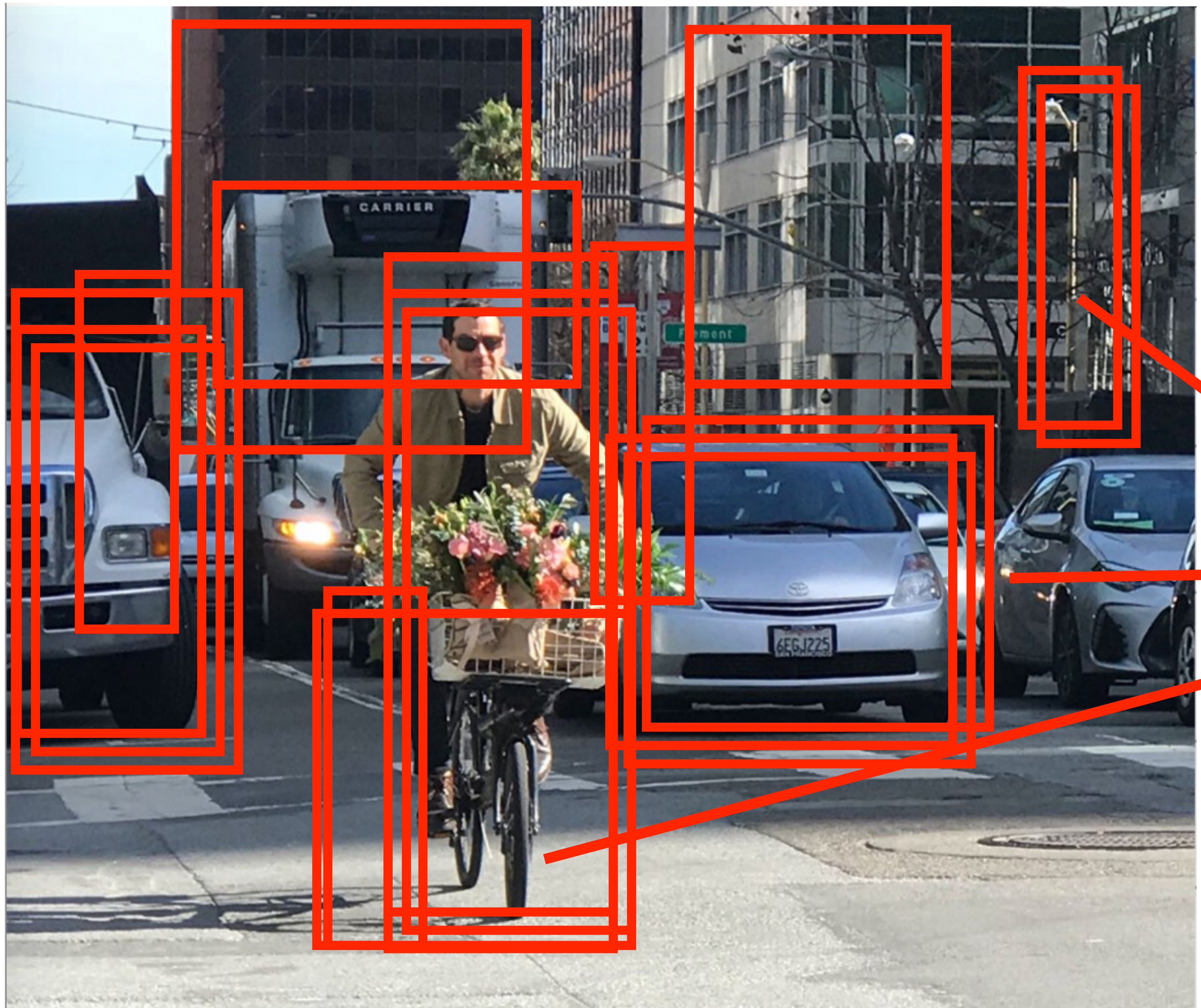


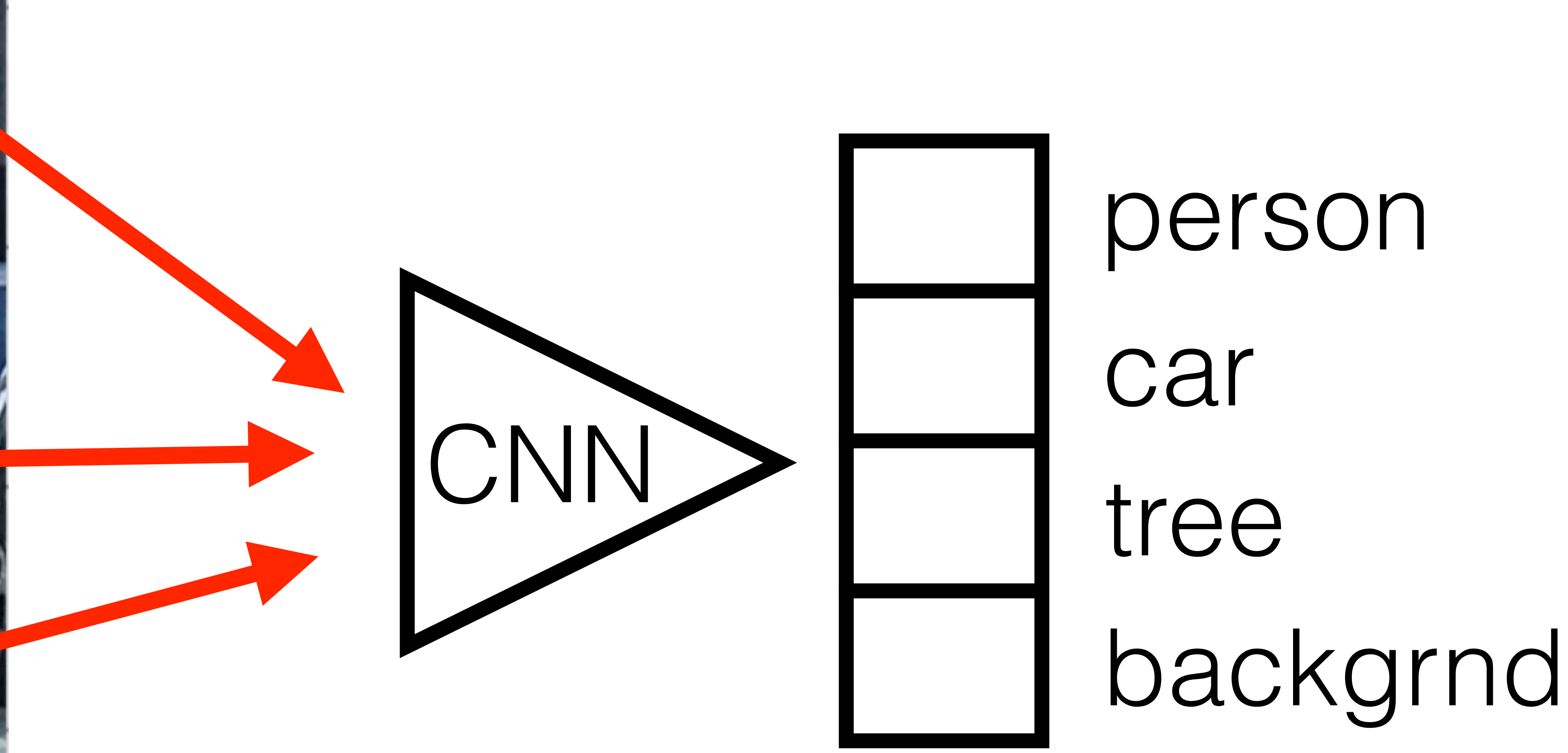
classify + align only 2k region proposals

[Girschick ICCV 2015] Fast-RCNN
https://arxiv.org/abs/1504.08083

- Approach works but it takes extremely long to compute response on all rectangular sub-windows:
  H x W x Aspect_Ratio x Scales x 0.001 sec = **months**
- Instead we can use elementary signal processing method to extract only 2k viable candidates:
  [Girschick ICCV 2015], Fast-RCNN
  https://arxiv.org/abs/1504.08083
  (find 2k cand.) + (2k cand. x 0.001 sec) = **47+2 sec = 49 sec**

Object detection

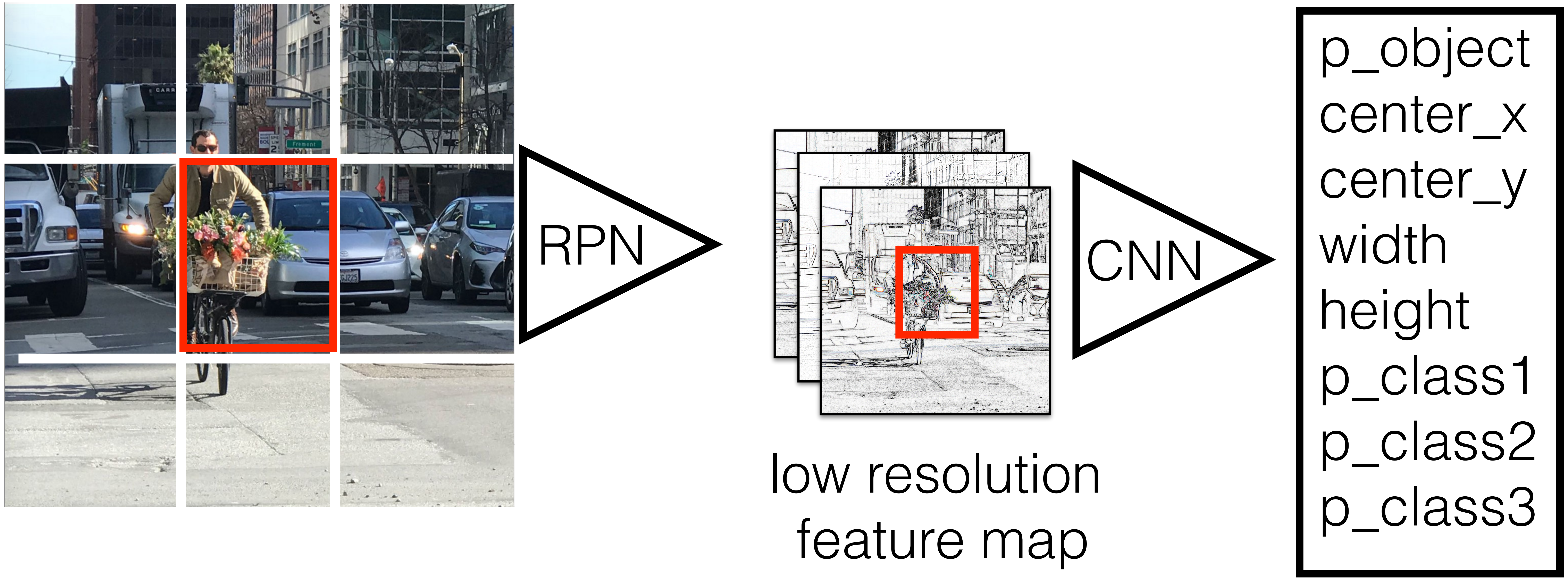




[Girschick ICCV 2015] Fast-RCNN
https://arxiv.org/abs/1504.08083

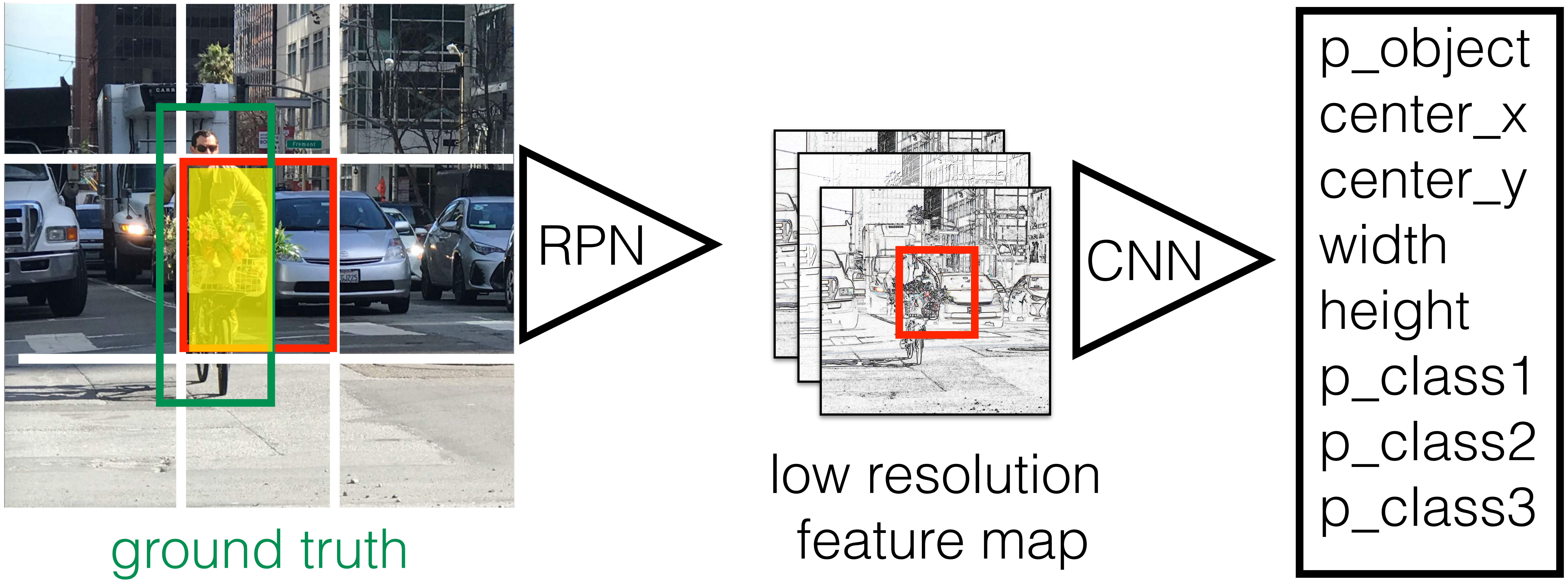
The search for region proposals is computational bottleneck !!!

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



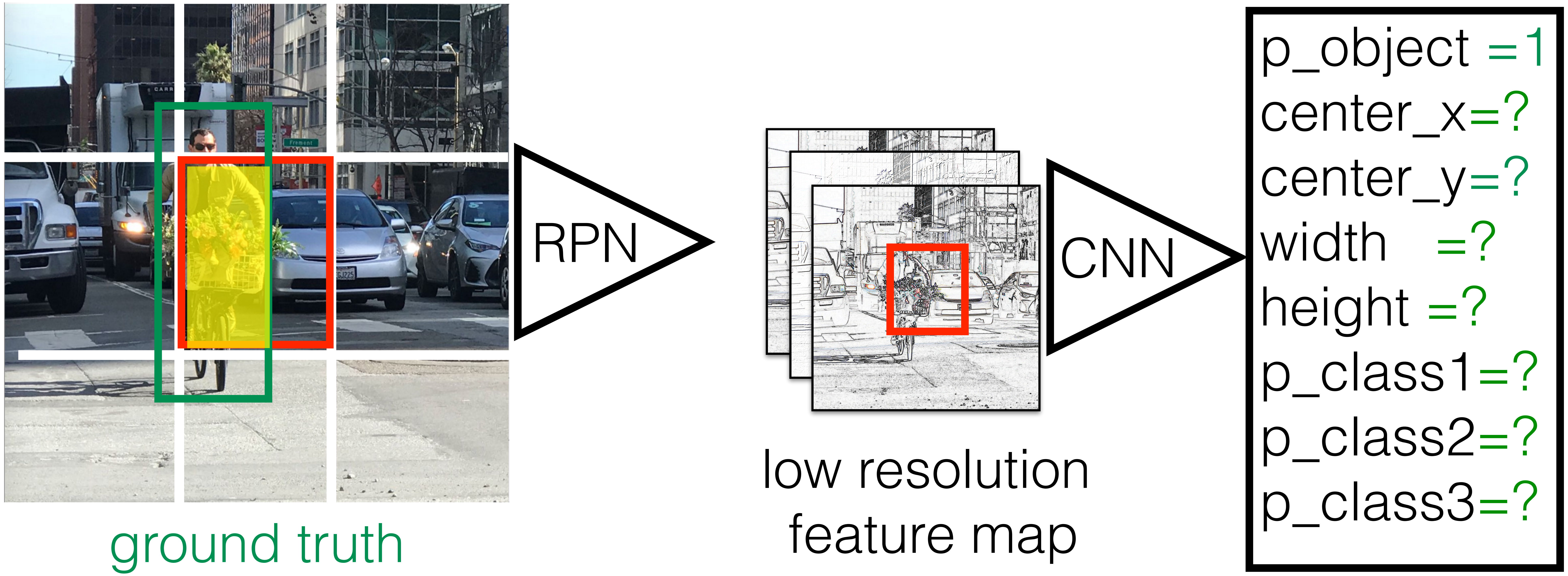
- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- learn from ground truth

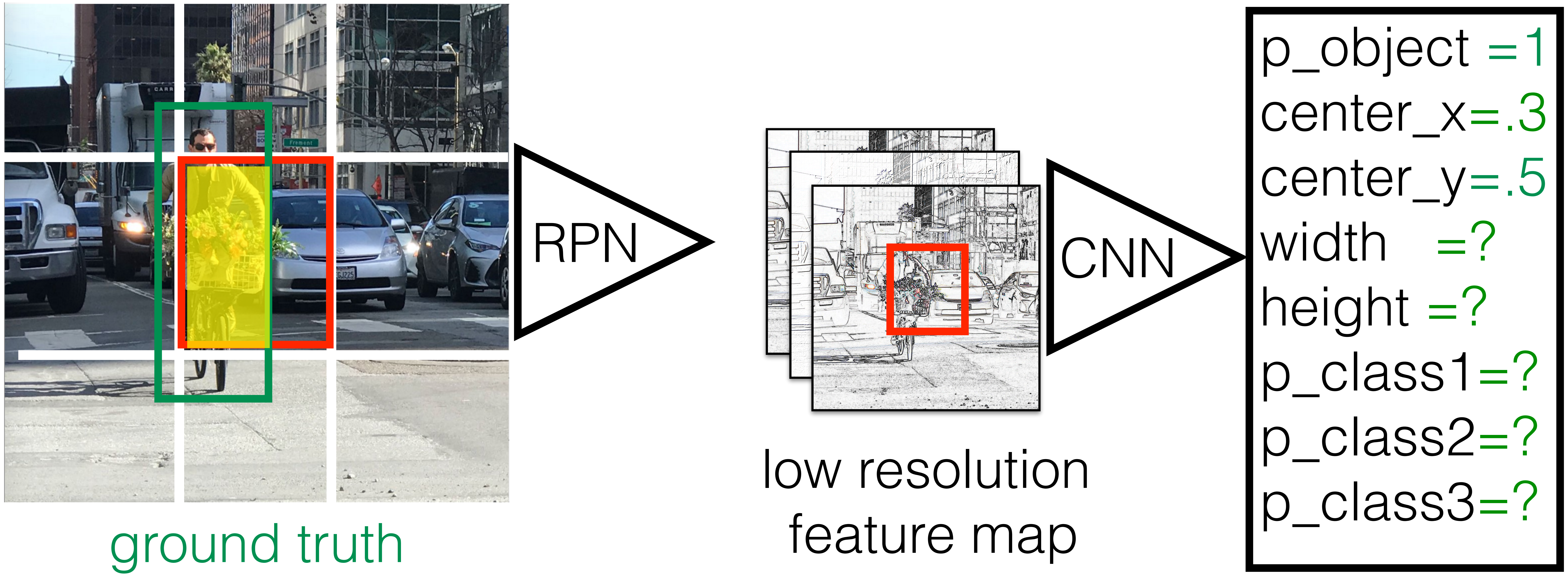
YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- ground truth: bbs with IoU>0.7 are objects,
  bbs with IoU<0.3 not objects

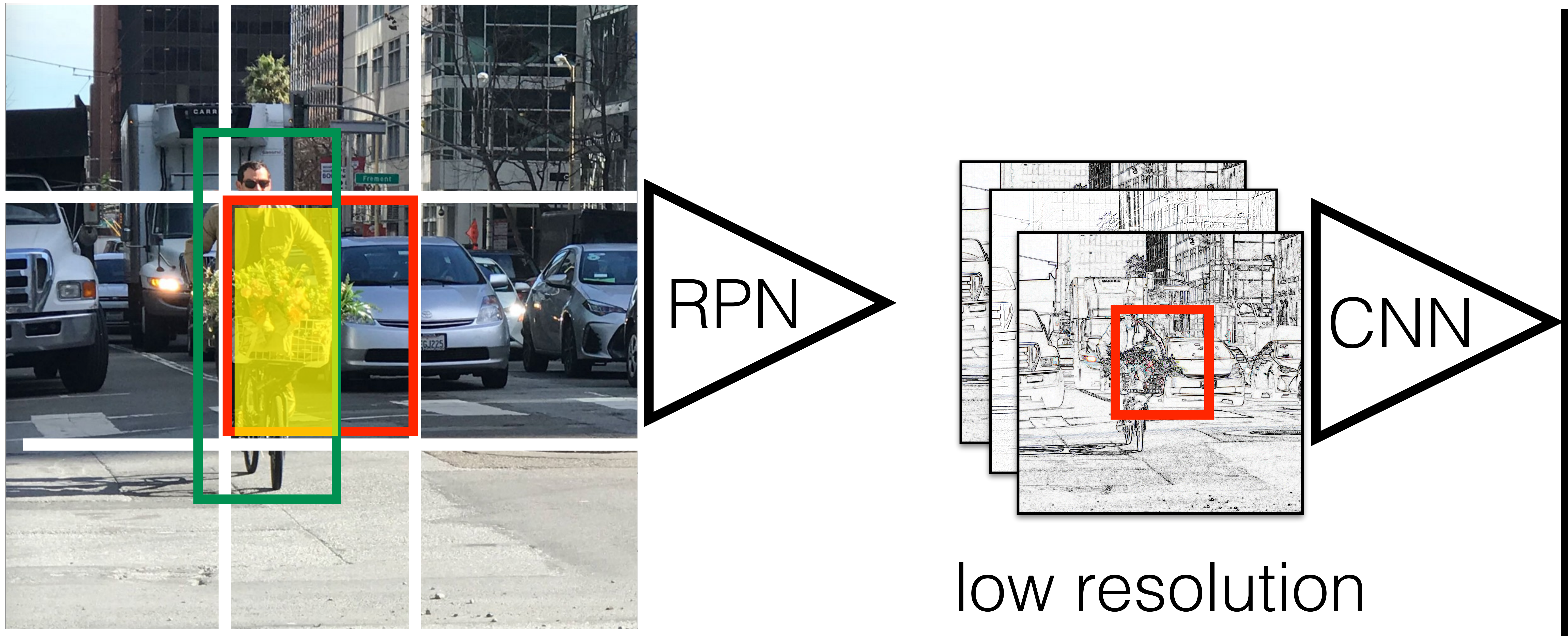
YOLO and Faster RCNN architectures

https://arxiv.org/abs/1506.01497



- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- ground truth: bbs with IoU>0.7 are objects,
  bbs with IoU<0.3 not objects

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



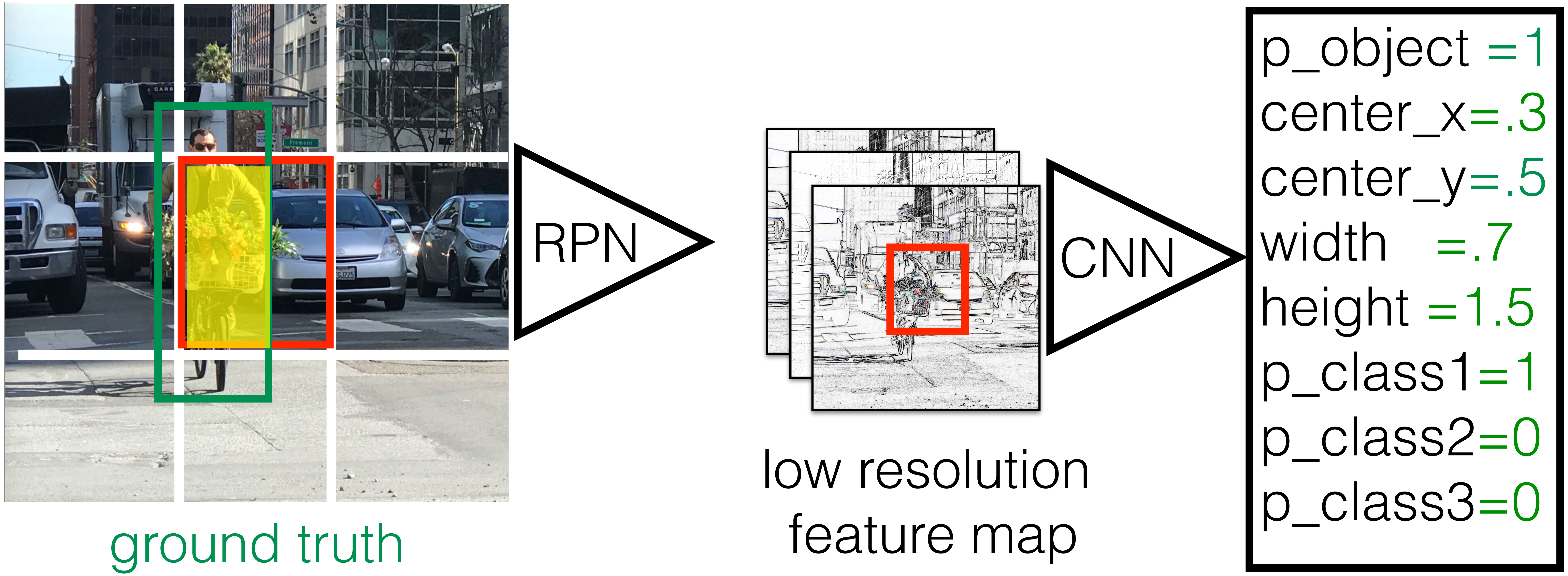
ground truth

low resolution feature map

- p_object =1
- center_x=.3
- center_y=.5
- width =.7
- height =1.5
- p_class1=?
- p_class2=?
- p_class3=?

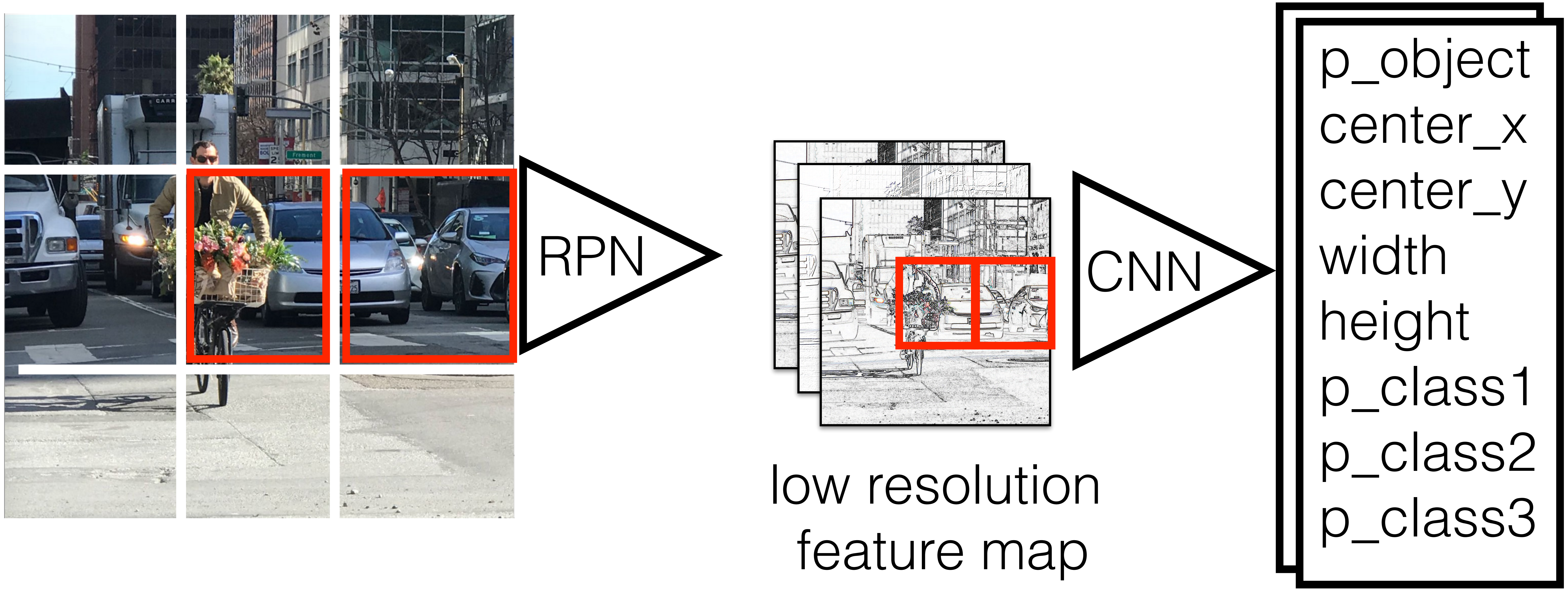
- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- ground truth: bbs with IoU>0.7 are objects,
  bbs with IoU<0.3 not objects

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



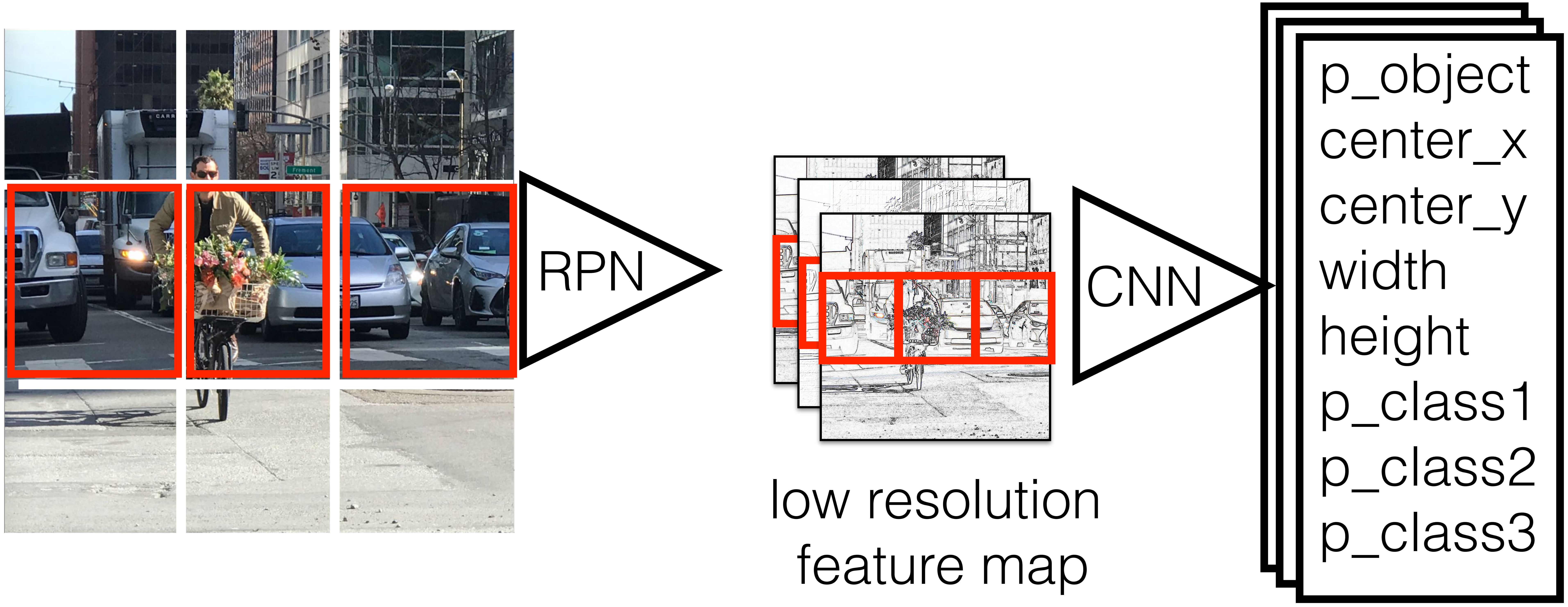
- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- ground truth: bbs with IoU>0.7 are objects,
  bbs with IoU<0.3 not objects

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- each sub-image has its own output

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497

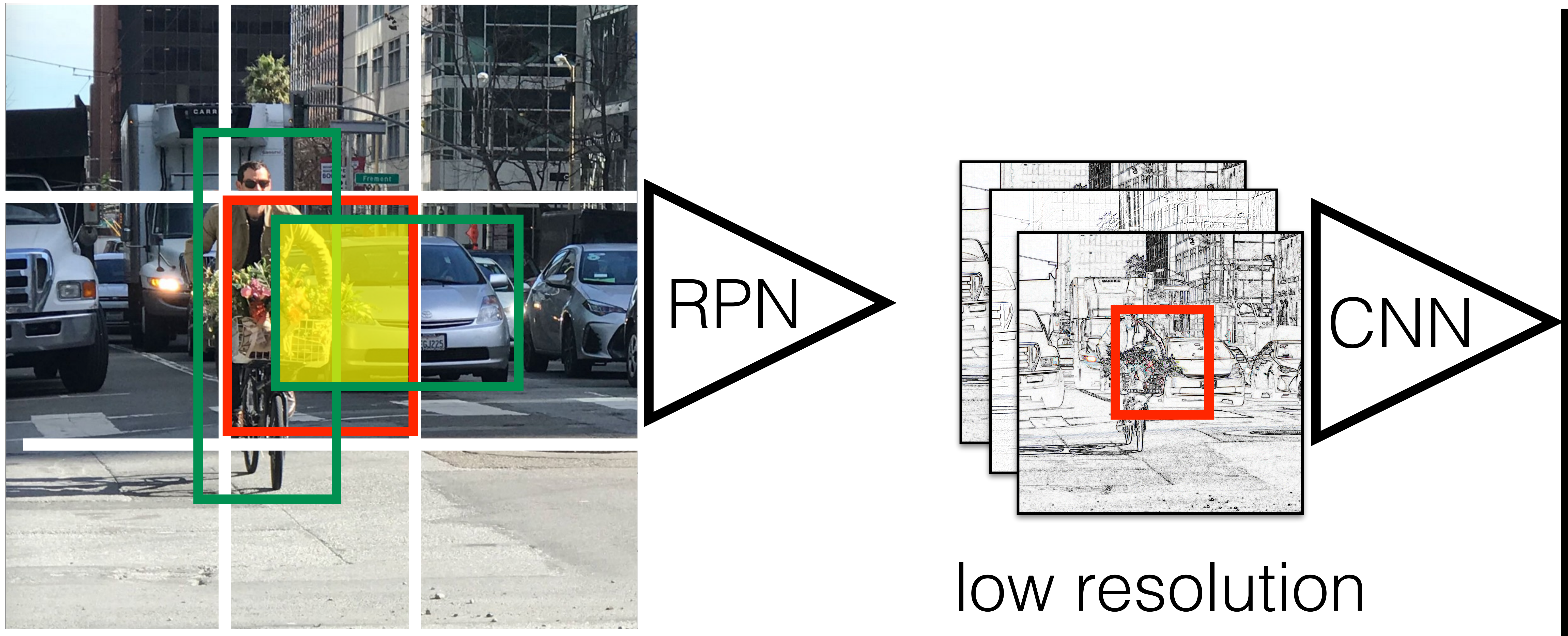


- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- each sub-image has its own output

Do you see any problem?

YOLO and Faster RCNN architectures

https://arxiv.org/abs/1506.01497



ground truth

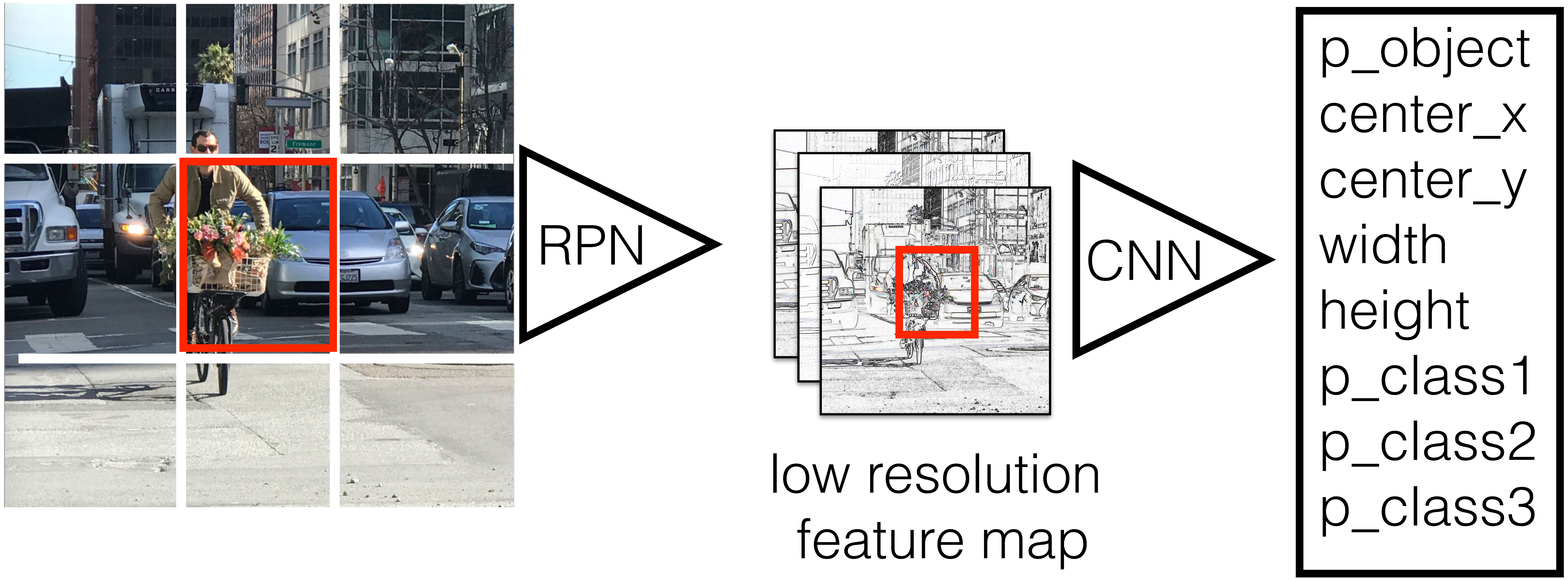
low resolution feature map

p_object =1
center_x=.3
center_y=.5
width =.7
height =1.5
p_class1=1
p_class2=1
p_class3=0

- divide image into 3x3 sub images
- predict relative position, objectness, class for each sub-im
- ground truth: bbs with IoU>0.7 are objects, bbs with IoU<0.3 not objects

=> more obj in one sub-im

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



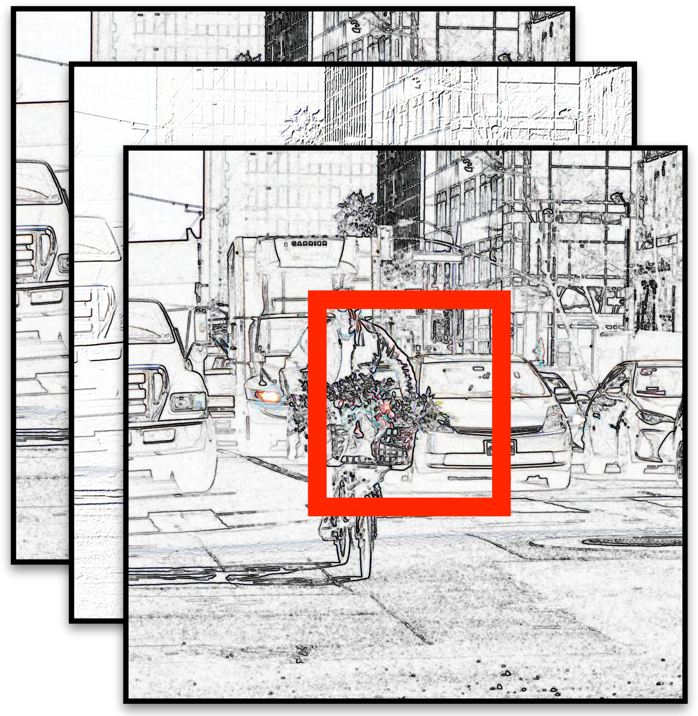
- divide image into 3x3 sub-images
- predict relative position, objectness, class for each sub-im

YOLO and Faster RCNN architectures

https://arxiv.org/abs/1506.01497

ground truth

RPN



low resolution feature map

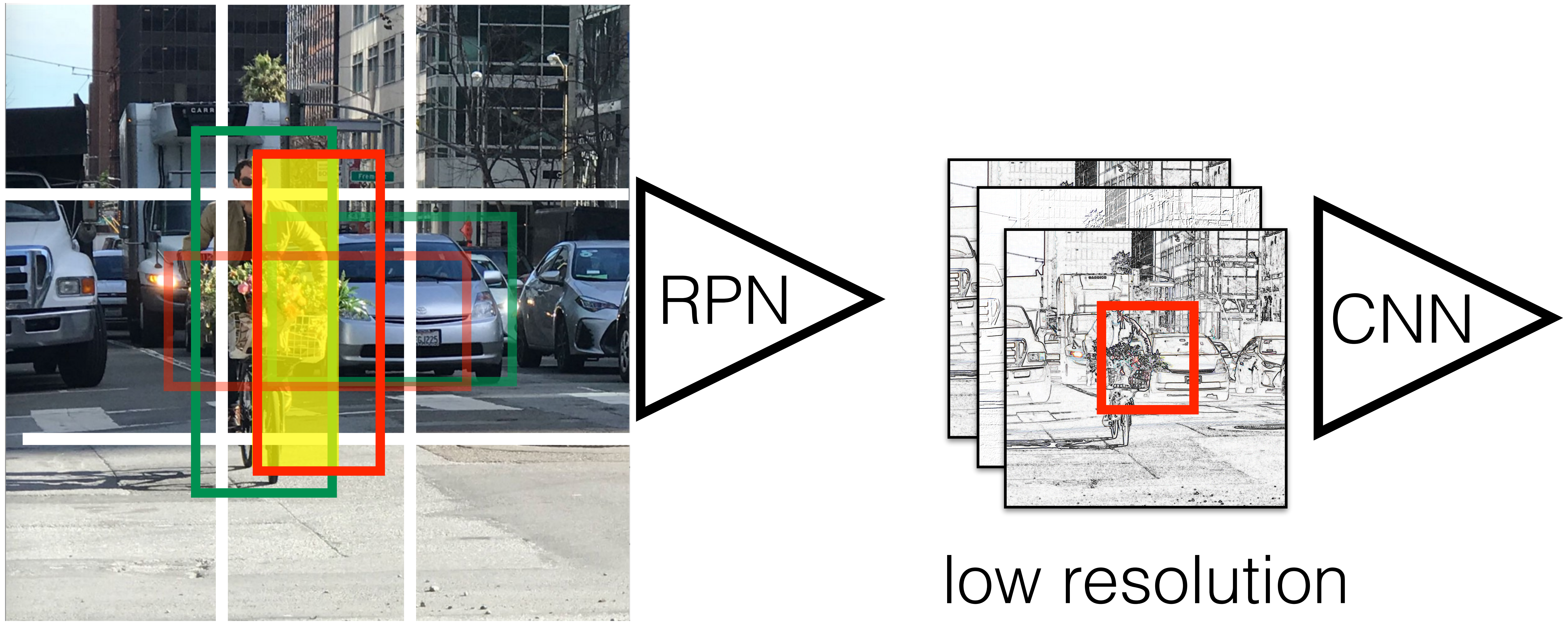
CNN

p_object
center_x
center_y
width
height
p_class1
p_class2
p_class3

p_object
center_x
center_y
width
height
p_class1
p_class2
p_class3

Introduce anchor bounding boxes

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



ground truth

low resolution feature map

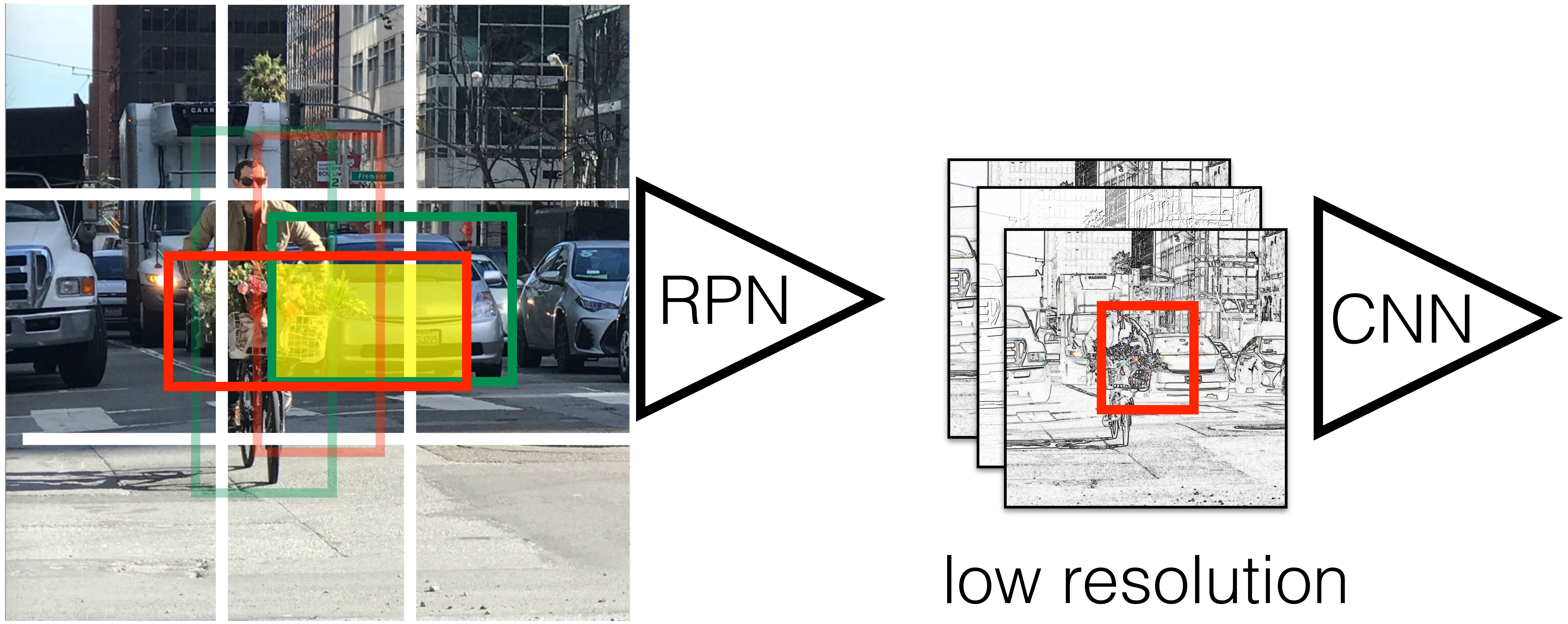
p_object
center_x
center_y
width
height
p_class1
p_class2
p_class3

p_object
center_x
center_y
width
height
p_class1
p_class2
p_class3

Introduce anchor bounding boxes

- for each anchor bb CNN predicts:
  - its “alignment with gt” (regression loss)
  - its “objectness”+”class” (classification loss)

YOLO and Faster RCNN architectures
https://arxiv.org/abs/1506.01497



ground truth

low resolution feature map

p_object
center_x
center_y
width
height
p_class1
p_class2
p_class3

p_object
center_x
center_y
width
height
p_class1
p_class2
p_class3

Introduce anchor bounding boxes

- for each anchor bb CNN predicts:
  - its “alignment with gt” (regression loss)
  - its “objectness”+”class” (classification loss)

Object detection

- Approach works but it takes extremely long to compute response on all rectangular sub-windows:
  H x W x Aspect_Ratio x Scales x 0.001 sec = **months**
- Instead we can use elementary signal processing method to extract only 2k viable candidates:
  [Girschick ICCV 2015], Fast-RCNN
  https://arxiv.org/abs/1504.08083
  (find 2k cand.) + (2k cand. x 0.001 sec) = **47+2 sec = 49 sec**
- Do region proposal by CNN => **0.1 sec**

  [Faster RCNN 2017] https://arxiv.org/abs/1506.01497
  [Redmont CVPR 2018], https://arxiv.org/abs/1804.02767
  code: https://pjreddie.com/darknet/yolo/

Deep convolutional - object detection

Summary

- Use ConvNets for images (or any other spatially structured inputs - depth images)
- Always use distinct training/testining data to avoid overfitting
- Compare results using by comparing full curves, e.g. Average Precision (AP)
- Simplified detector based on RPN will be implemented during following two labs.

Test compentecies

- Compute feedforward pass in neural nets (includinginput/output dimensionality)
- Compute backpropagation in neural nets (including convnet, sigmoid layer)
- Compute precision, recall, FP, FN, TP, TN …
- Understand object detection architecture