Ceng 793 – Advanced Deep Learning

# Week 3 – Overview: Convolutional Neural Networks & Recurrent Neural Networks

Spring 2017

Emre Akbas & Sinan Kalkan

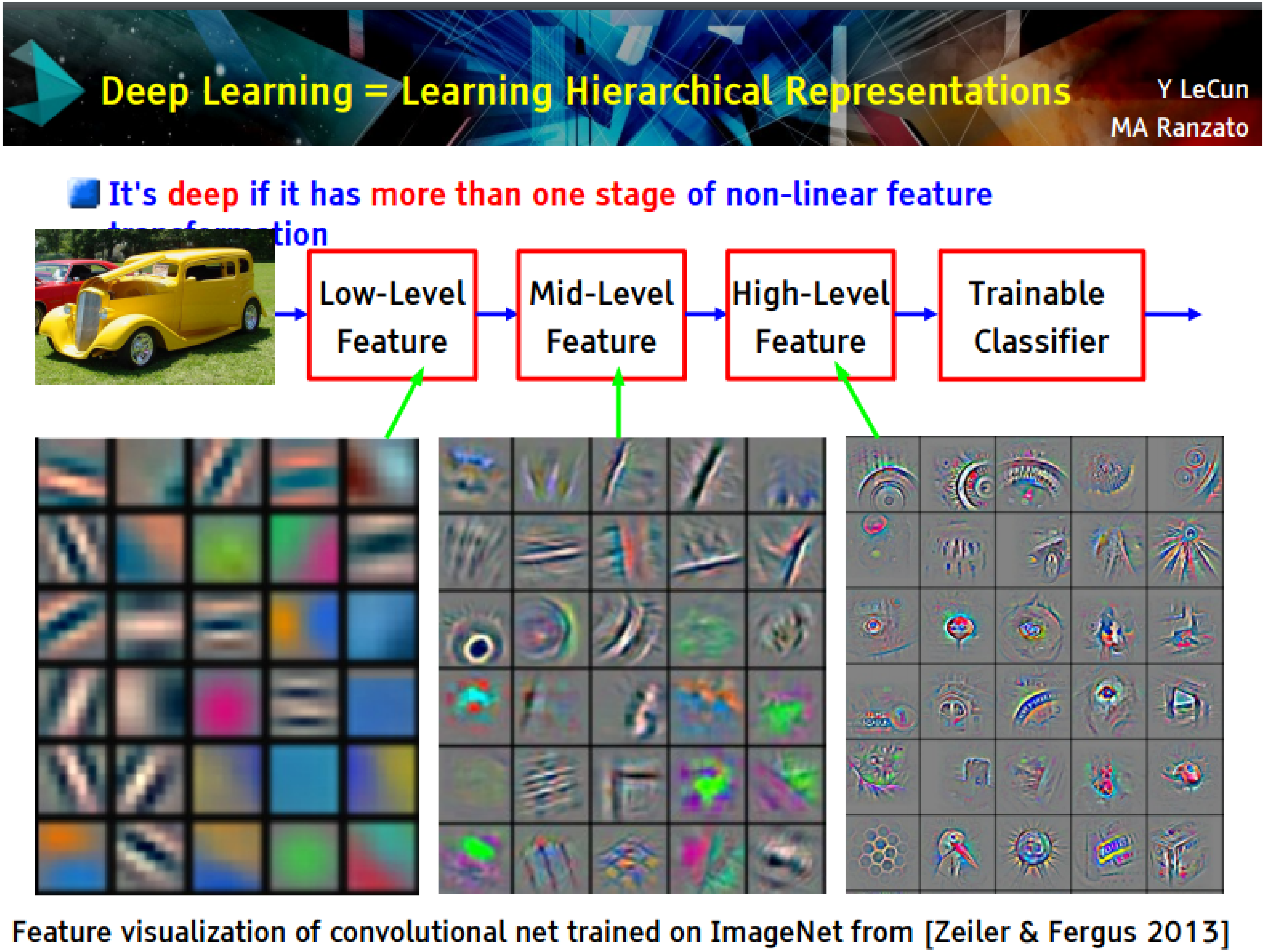
# Regular ANN vs CNN?

- ANN → fully connected.
  - Uses matrix multiplication to compute the next layer.
- CNN → sparse connections.
  - Uses convolution to compute the next layer.
- Everything else stays almost the same
  - Activation functions
  - Cost functions
  - Training (back-propagation)
  - …
- CNNs are more suitable for data with grid topology.
  - e.g. images (2-D grid), videos (3-D grid), time series data (1-D grid).

CNNs learn both:

- Hierarchical representations of the data, and
- Supervised decision boundary on these representations

at the same time.




[Slide by Yann LeCun http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf]

# Convolution

We use it **to extract information** from a signal.

$$s[t]=(x\star w)[t]=\sum_{a=-\infty}^{a=\infty} x[a]w[a+t]$$

Feature map

Input

kernel

Naming convention in computer vision and DL.

Computes **similarity** of two signals. Can be used to find patterns (template matching with normalized cross-correlation).

We use it **to extract information** from a signal.

Sliding dot-product

$$s[t]=(x\star w)[t]=\sum_{a=-\infty}^{a=\infty} x[a]\,w[a+t]$$

Feature map

Input

kernel

Naming convention in computer vision and DL.

Computes **similarity** of two signals. Can be used to find patterns (template matching with normalized cross-correlation).

# Convolution or cross-correlation ?

Both are linear, shift-invariant operations.

Cross-correlation:

$$s[t]=(x\star w)[t]=\sum_{a=-\infty}^{a=\infty} x[a]w[a+t]$$

Convolution:

$$s[t]=(x*w)[t]=\sum_{a=-\infty}^{a=\infty} x[a]w[t-a]$$

Identical operations except that the kernel is flipped in convolution.
If the kernel is symmetric, then they are identical.

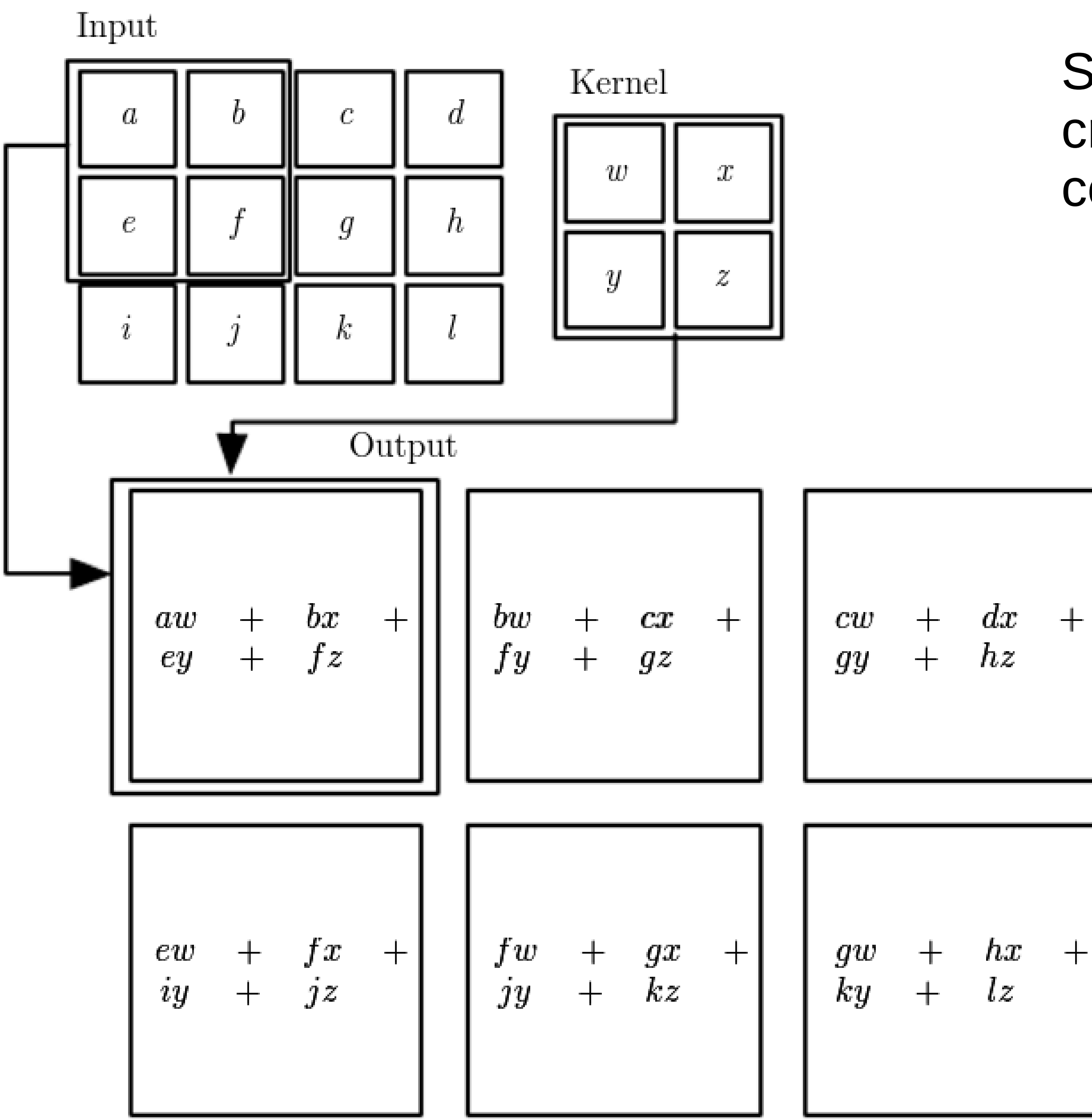
# Convolution or cross-correlation ?

Many machine learning libraries implement cross-correlation but call it convolution.

This is the formula for cross-correlation in **2D**:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

# Convolution example




Strictly speaking, this is a cross-correlation, not convolution.

Figure 9.1 from Goodfellow et al. (2016).

# Motivation behind ConvNets

1) Sparse interactions

2) Parameter sharing

3) Equivariant representations

4) Ability to process inputs of variable sizes

1) Sparse interactions

2) Parameter sharing

3) Equivariant representations

4) Ability to process inputs of variable sizes

**Localized neurons**

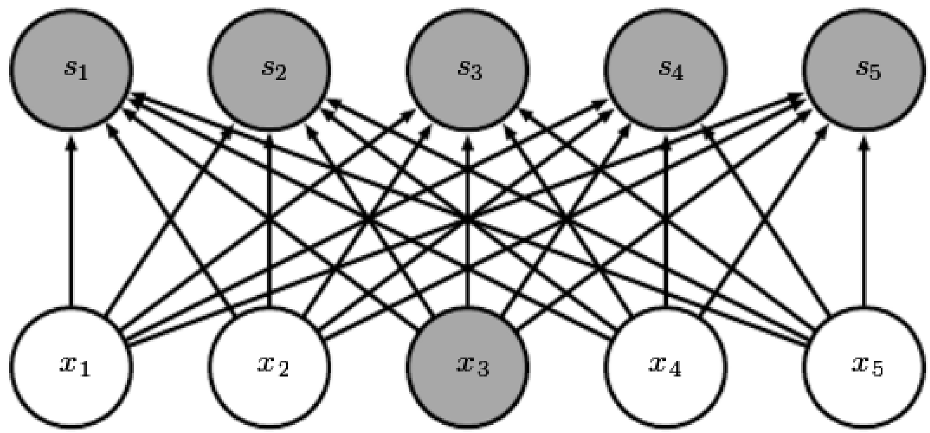
**Using the same localized neuron at different locations**

**If the input moves, the output representation moves equivalently.**

**By using scalable pooling regions**

# 1) Sparse interactions

In a regular ANN (i.e. MLP), nodes are fully-connected




In CNN, sparse connections:

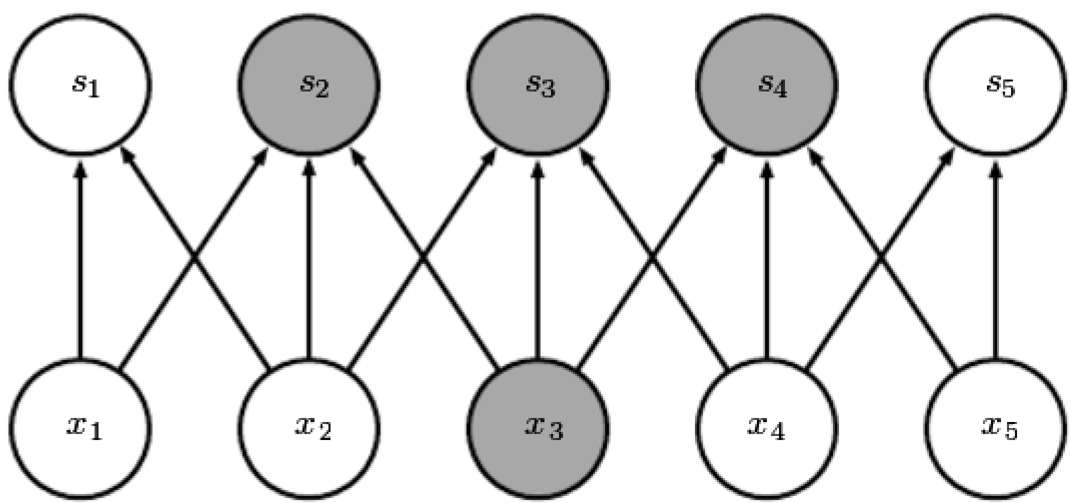
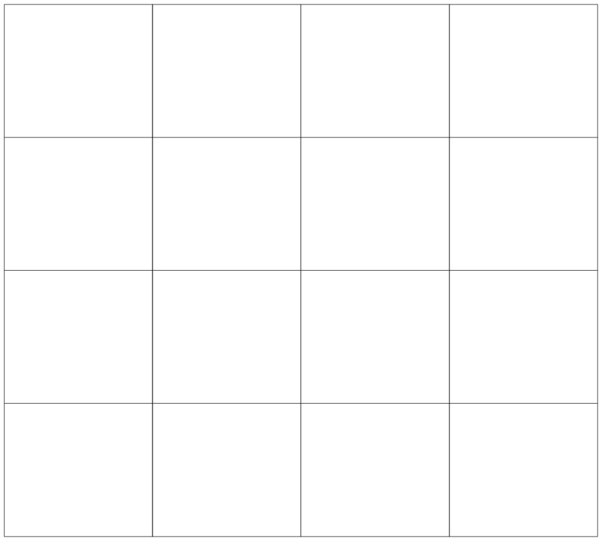
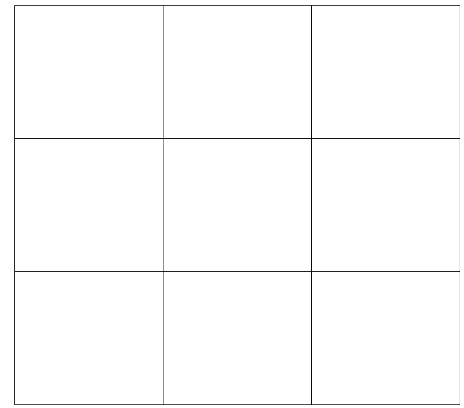



Figure 9.2 from Goodfellow et al. (2016).

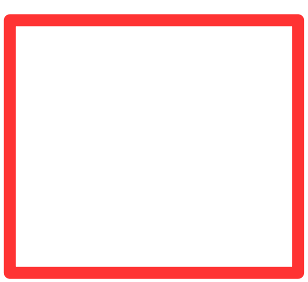
# Sparse interactions



$1^{st}$ (input) layer: 4x4 image

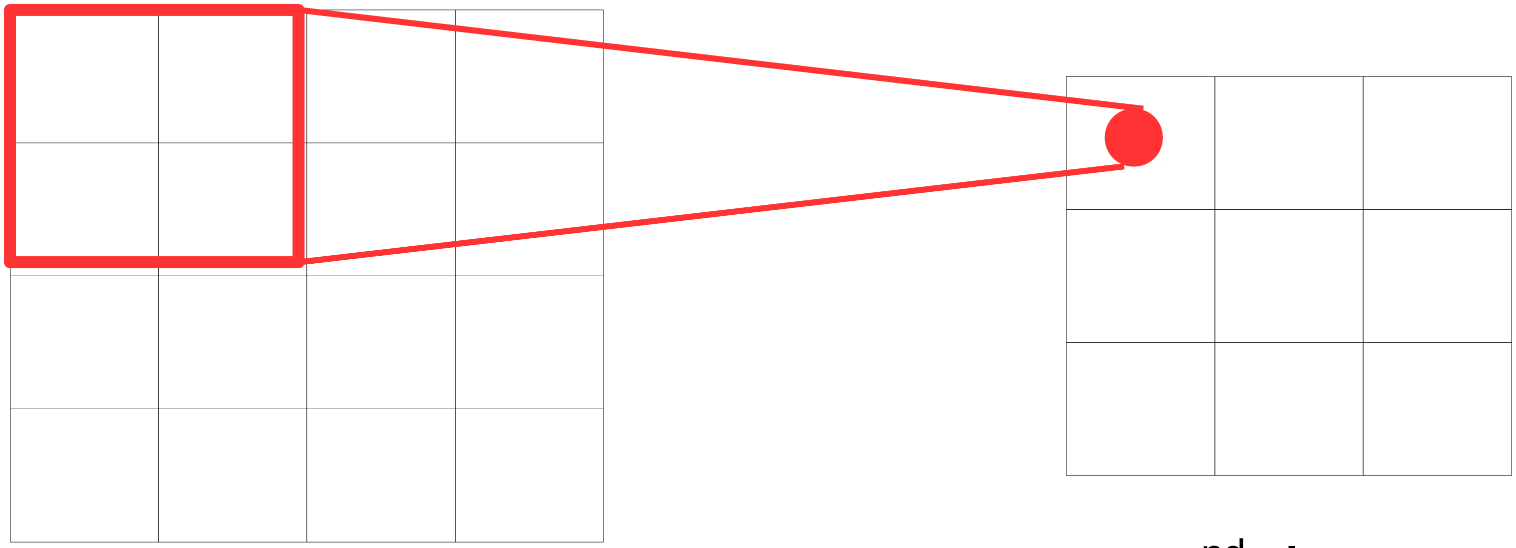


$2^{nd}$ layer



2x2 filter

# Sparse interactions

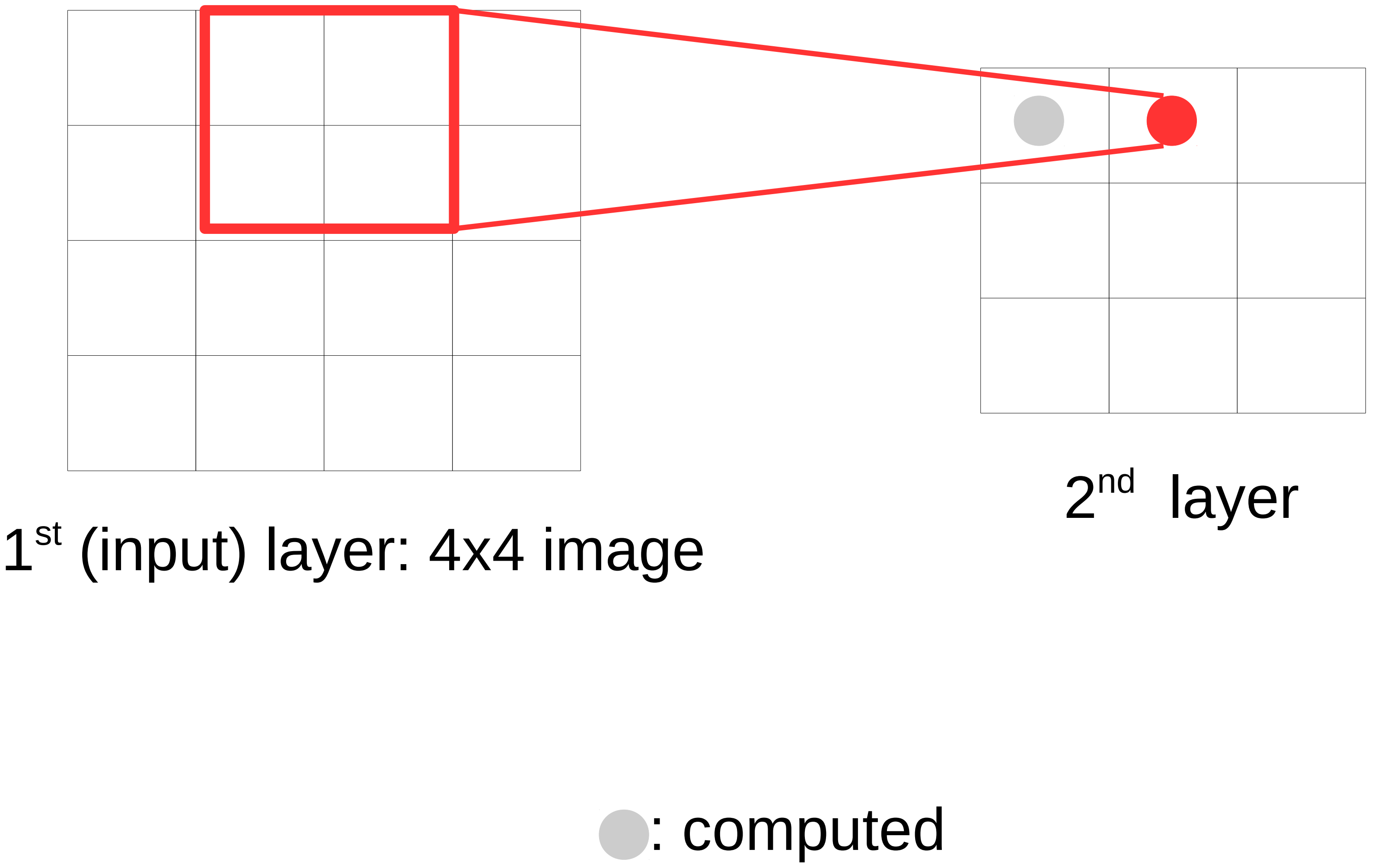


2nd layer

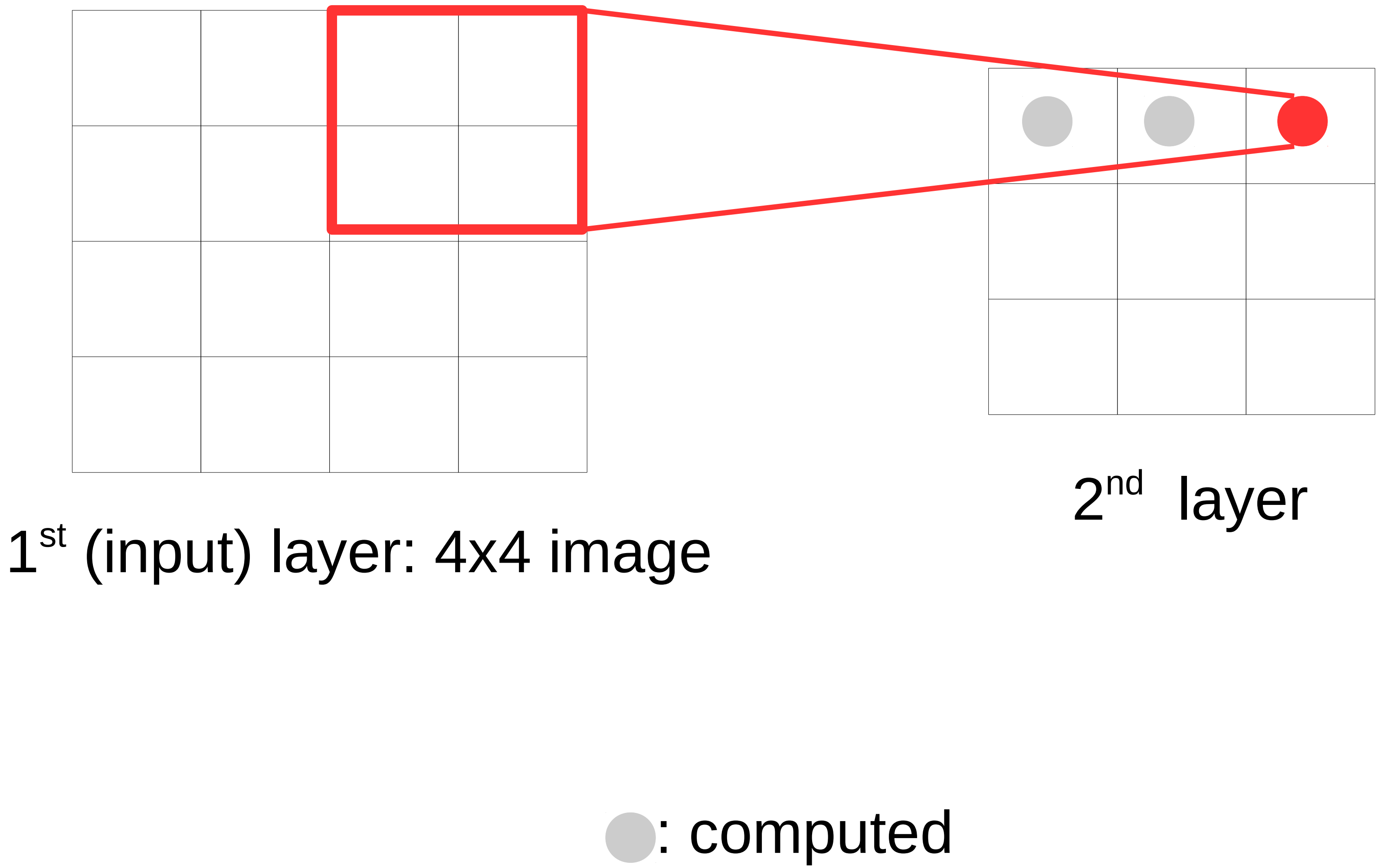
1st (input) layer: 4x4 image

Node in the 2nd layer is not fully-connected to the nodes in the 1st layer.

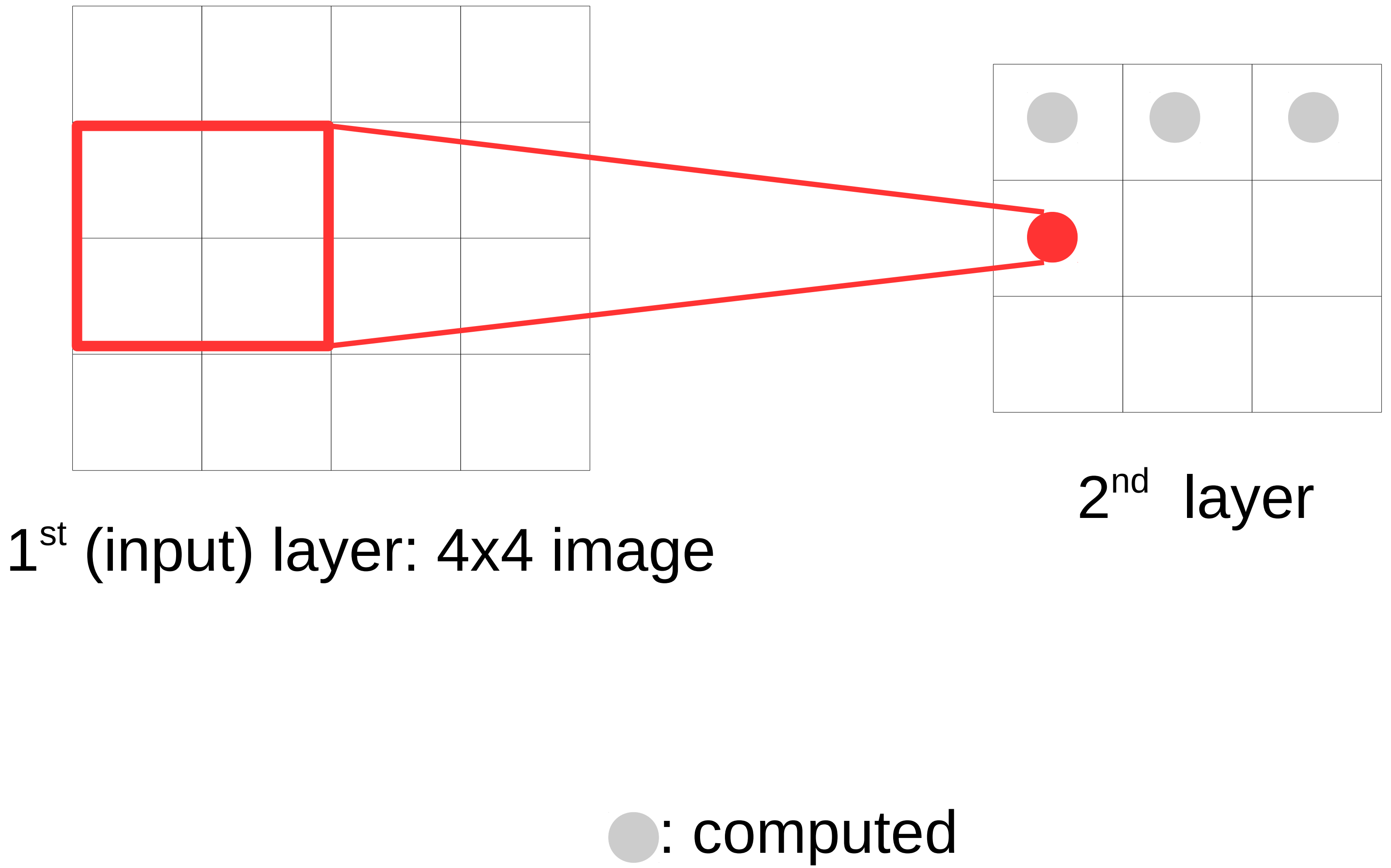
# Sparse interactions



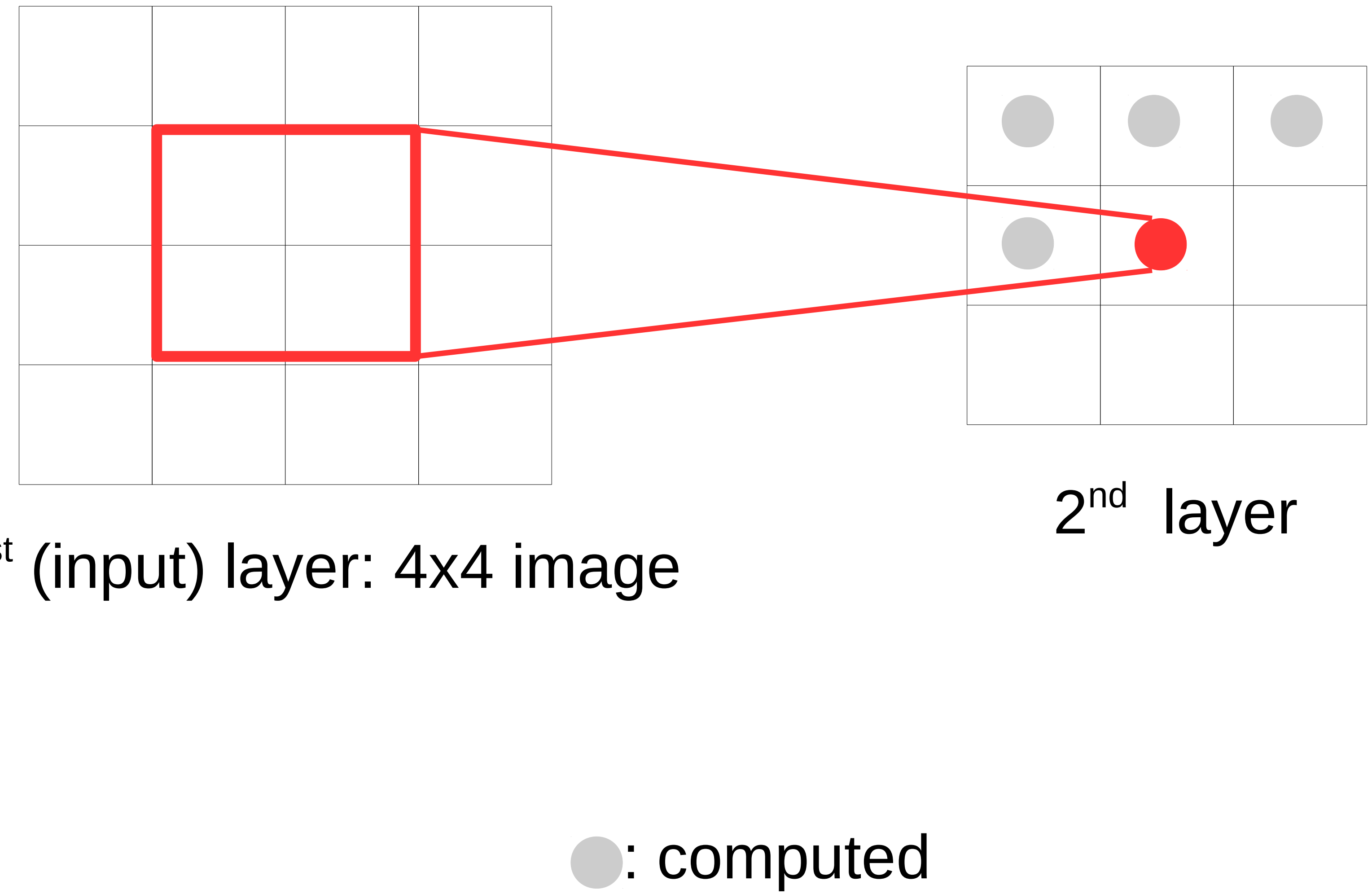
# Sparse interactions



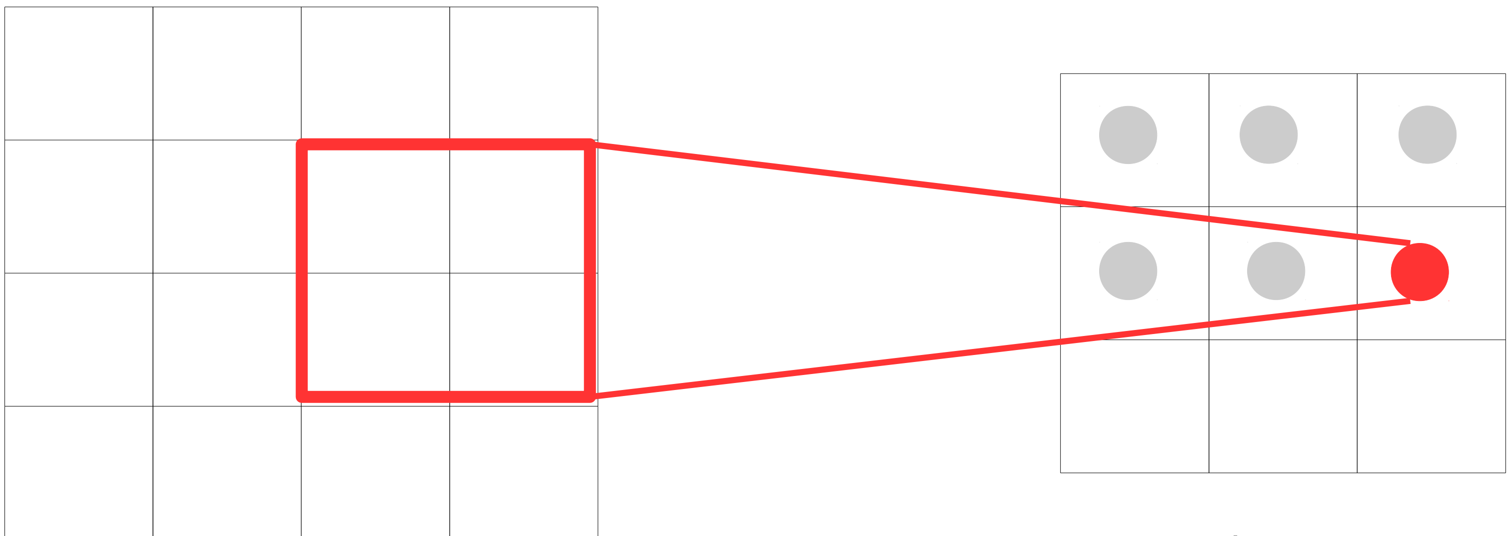
# Sparse interactions



# Sparse interactions



# Sparse interactions



$1^{st}$ (input) layer: 4x4 image

$2^{nd}$ layer

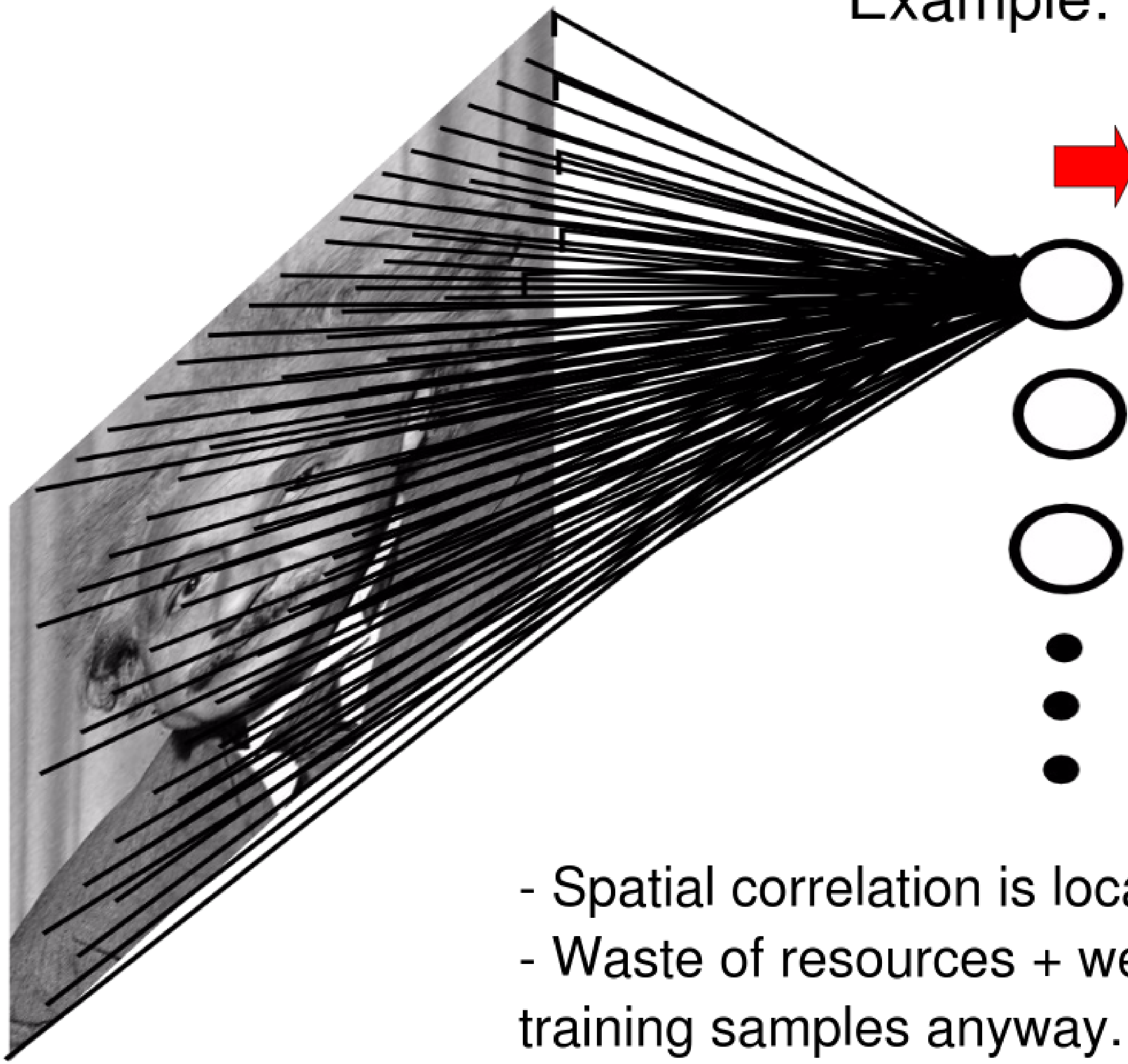
: computed

# But why do we need this sparsity?

# But why do we need this sparsity?

- Sparse connections reduce complexity.

# Fully Connected Layer



Example: 200x200 image
40K hidden units
→ **~2B parameters**!!!

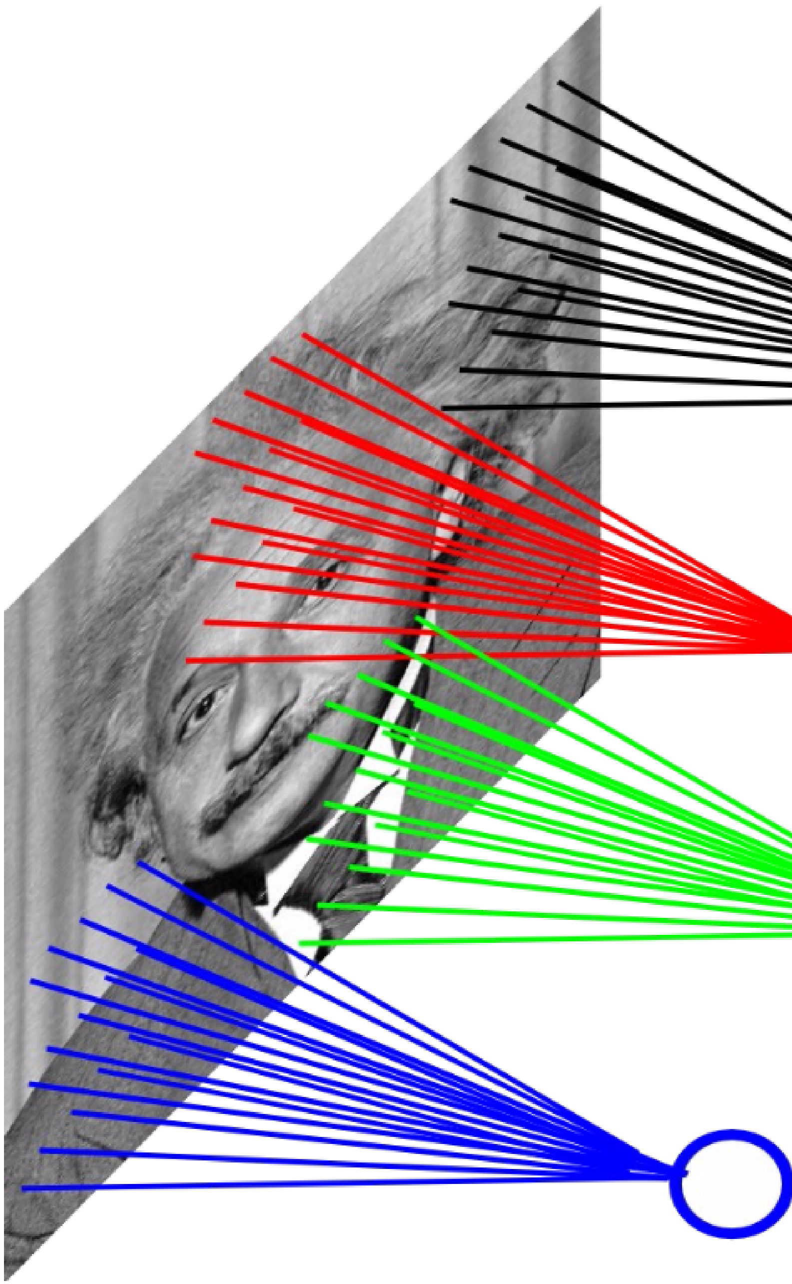
- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

40

Ranzato

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]

# Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

**Note:** This parameterization is good when input image is registered (e.g., face recognition).



Ranzato

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]

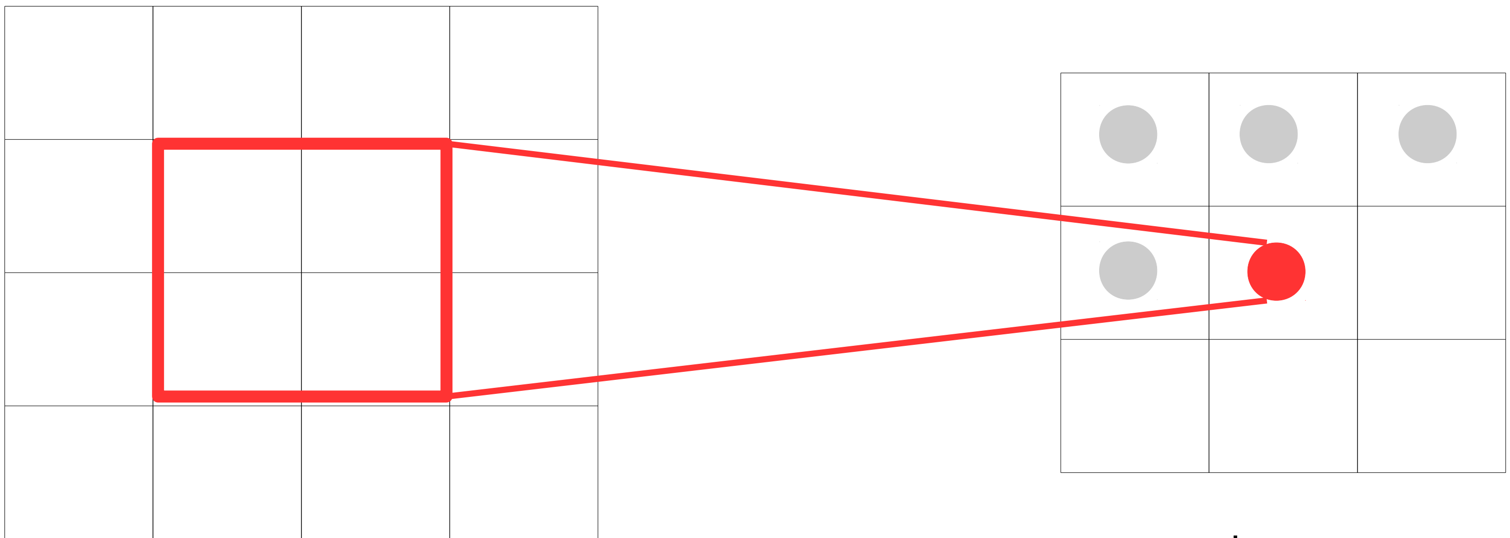
# Sparse interactions

Complexity of fully-connected vs sparse:

*m*: # of nodes in the $1^{st}$ layer
*n*: # of nodes in the $2^{nd}$ layer
*k*: # of elements in the filter

Fully-connected: $O(mn)$
Sparse: $O(nk)$ where, typically, $k \ll m$

# 2) Parameter Sharing



2nd layer

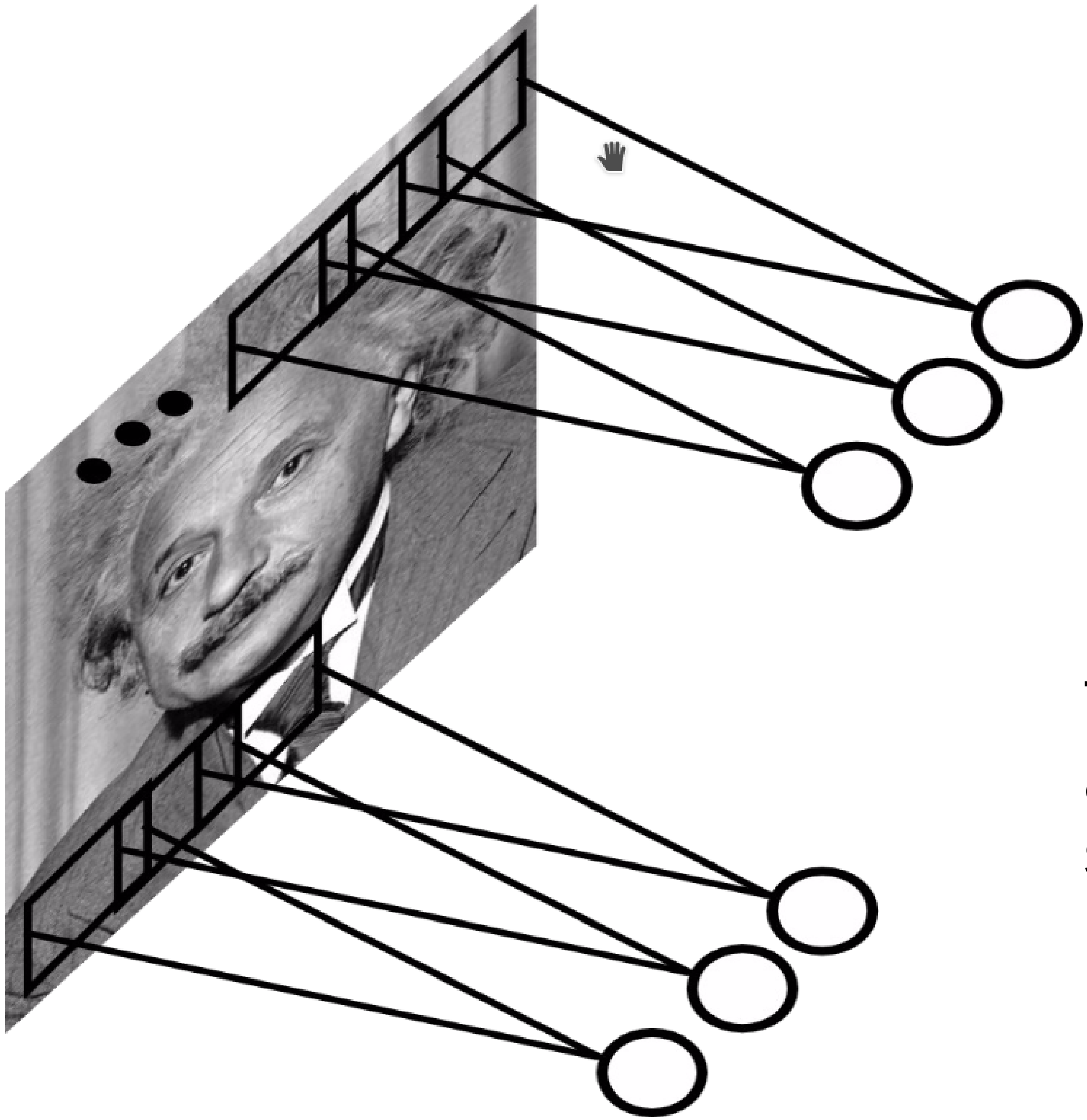
1st (input) layer: 4x4 image

Same neuron or kernel or filter (the red window) is applied at all locations of the input layer.

# of total parameters to be learned and storage requirements dramatically reduced.

Note *m* and *n* are roughly the same, but *k* is much less than *m*.

# Convolutional Layer



These six circles are actually the same neuron.

43

Ranzato

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]

# 3) Equivariance

General definition: **If**

***representation*(*transform*(x)) = *transform*(*representation*(x))**

**then *representation* is equivariant to the *transform*.**

# 3) Equivariance

General definition: **If**

***representation*(*transform*(x)) = *transform*(*representation*(x))**

**then *representation* is equivariant to the *transform*.**

Convolution is equivariant to translation. This is a direct consequence of parameter sharing.

Useful when detecting structures that are common in the input. E.g. edges in an image. Equivariance in early layers is good.

We are able to achieve translation-invariance (via max-pooling) due to this property.

# 4) Ability to process arbitrary sized inputs

Fully-connected networks accept fixed-size input vector.

In ConvNets, we can use “pooling” to summarize the input into a fixed-size vector/matrix.

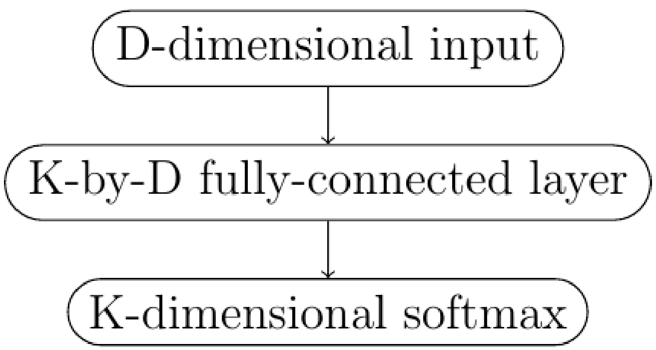
*Scale the pooling region with respect to the input size.*

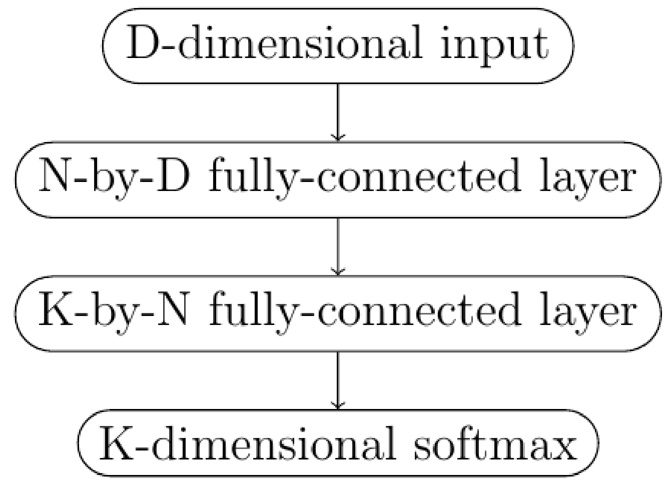
# After convolution...

# After convolution...

## Question 6 [10 points]

Consider the two MLP models given below. Comment on the capacities of these two models. Which model has more capacity? Why?

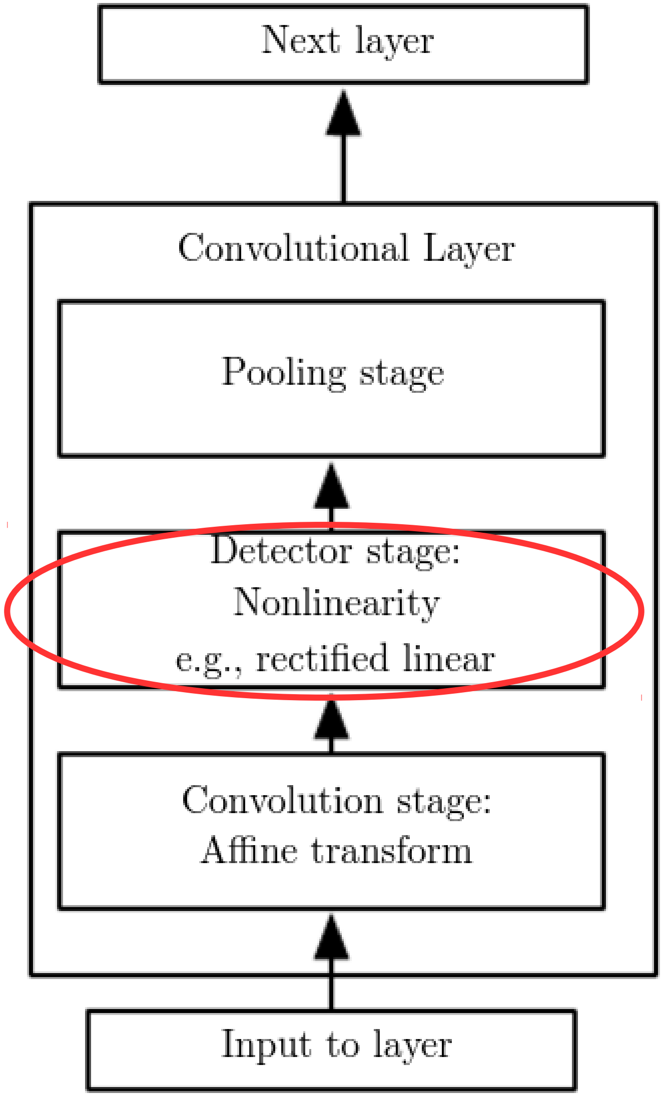



**Your answer:**

# After convolution...

the next operations: nonlinearity and pooling.




We have already seen many non-linear activation functions.

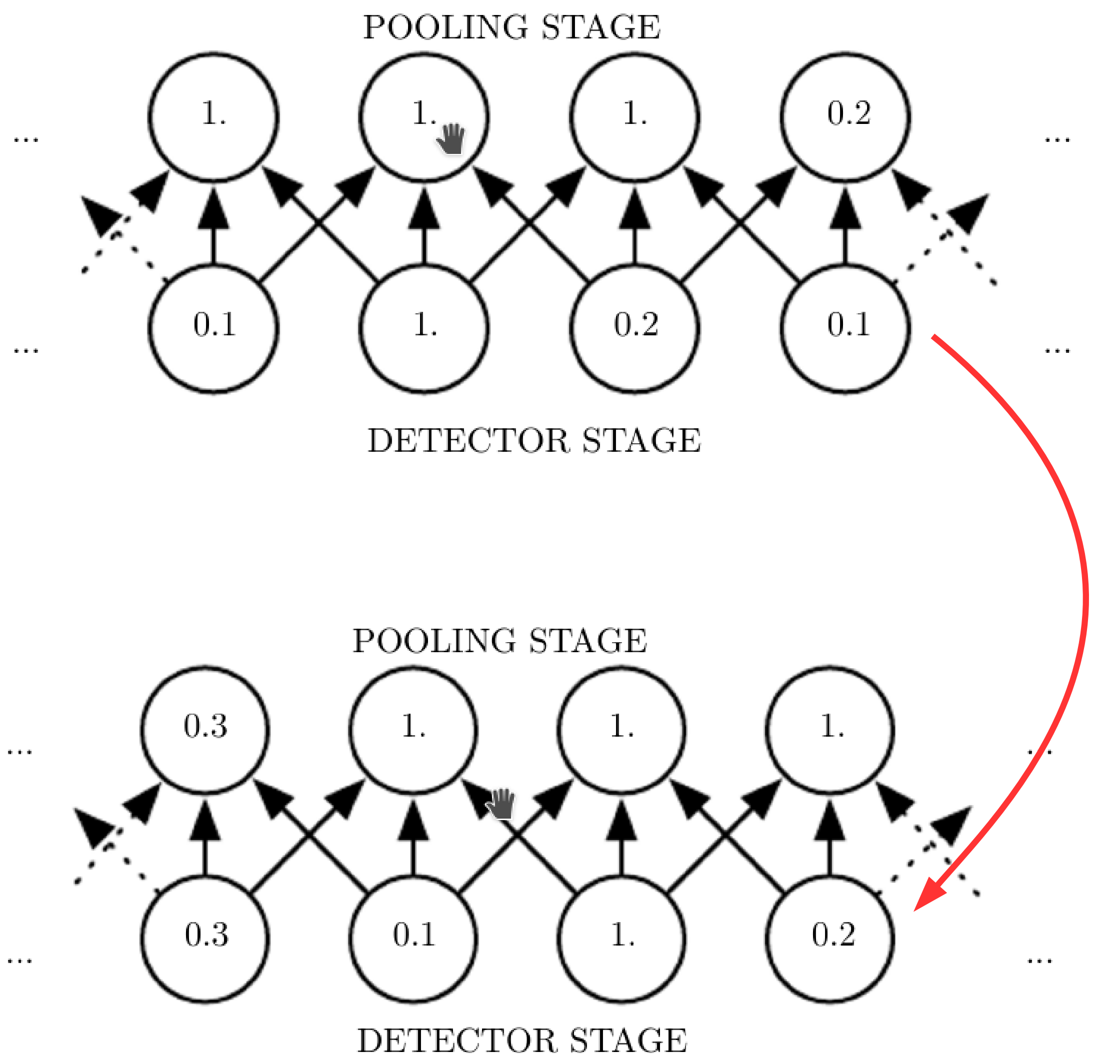
ReLU is the most widely used one.

# Pooling

A pooling function takes the output of the previous layer at a certain location *L* and computes a "summary" of the neighborhood around *L*.

E.g. max-pooling [Zhou and Chellappa (1988)]

# Max-pooling



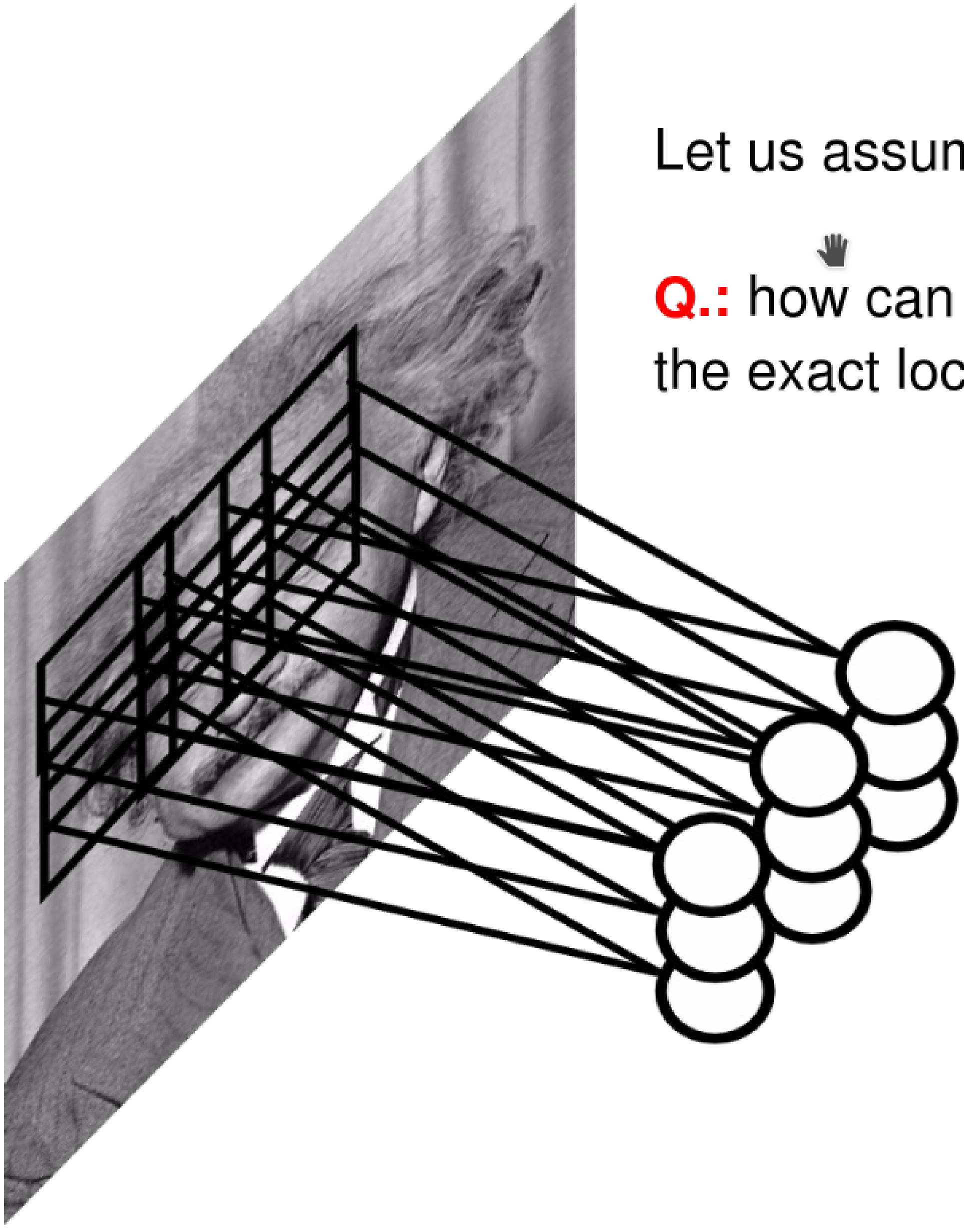

Max-pooling introduces **invariance**.

Input layer has shifted to the right 1-pixel.

But only half of the values in the output layer have changed.

Figure 9.8 from Goodfellow et al. (2016).

# Pooling Layer



Let us assume filter is an "eye" detector.

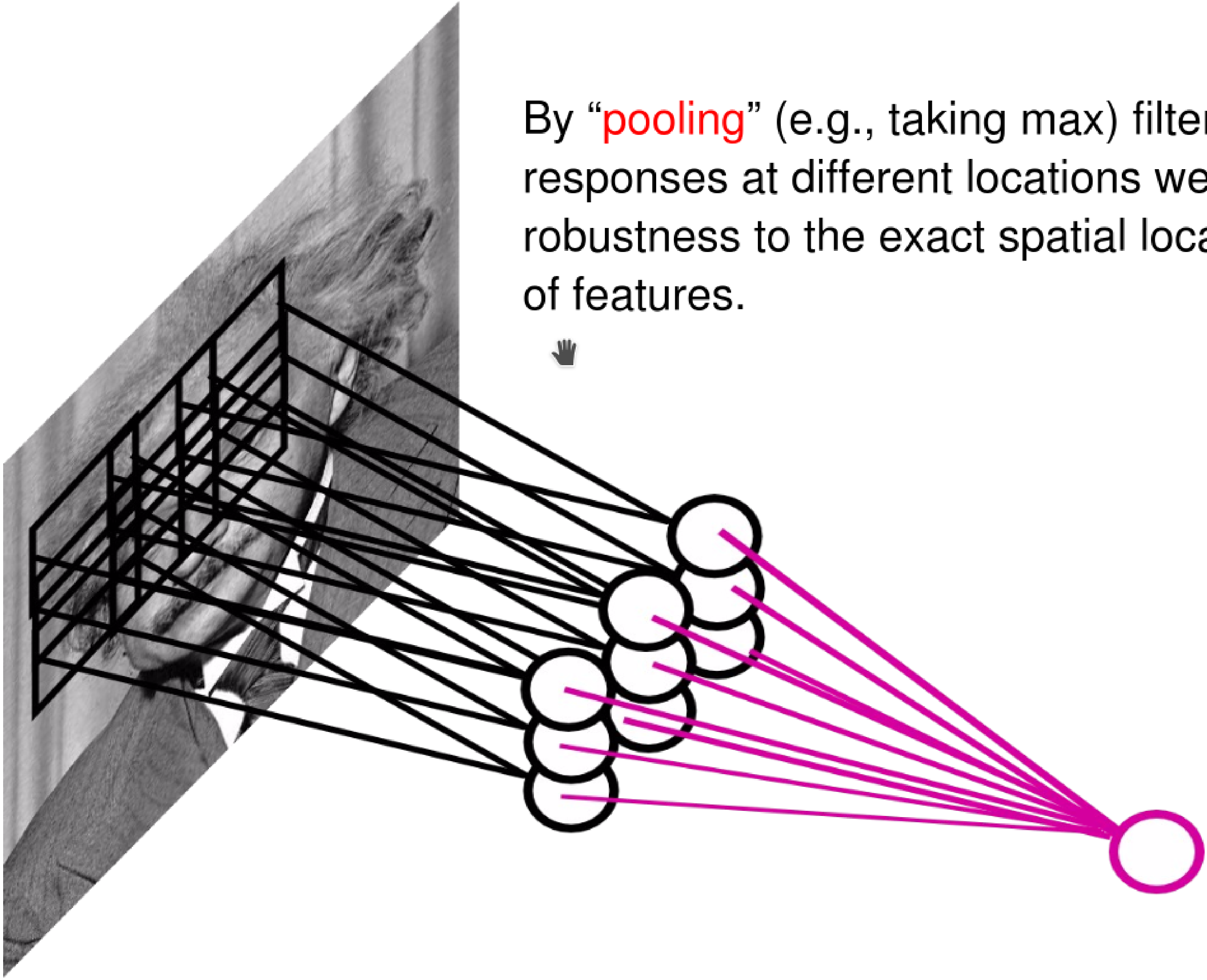
**Q.:** how can we make the detection robust to the exact location of the eye?



Ranzato

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]

# Pooling Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

68

Ranzato

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]

Spatial pooling produces invariance to translation. Pooling over channels produces other invariances. E.g. Maxout networks by Goodfellow et al. (2013).

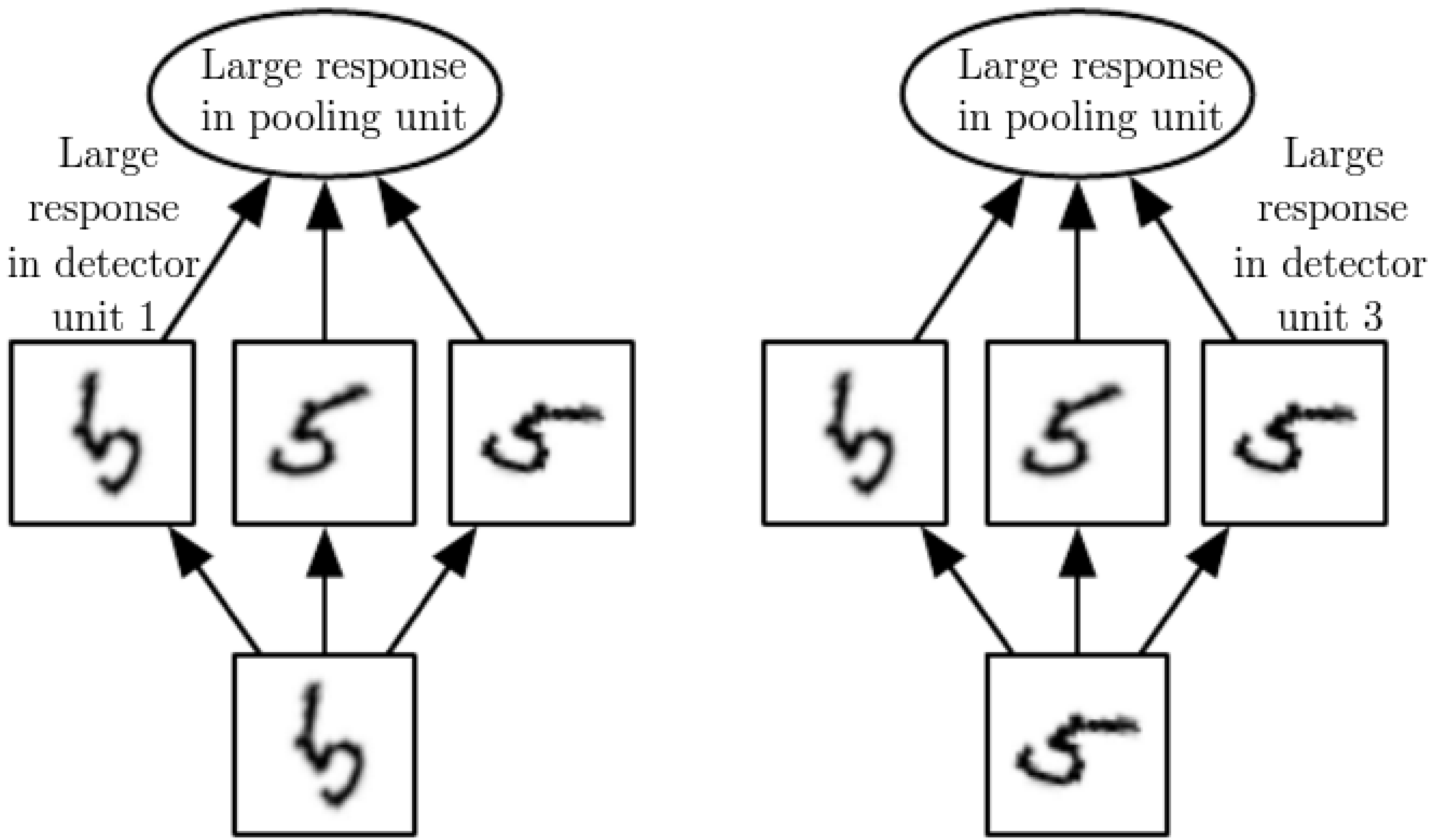



Figure 9.9 from Goodfellow et al. (2016).

Pooling summarizes.

We can make a sparse summary by using a stride larger than 1.

This reduces the computational complexity and memory requirements.

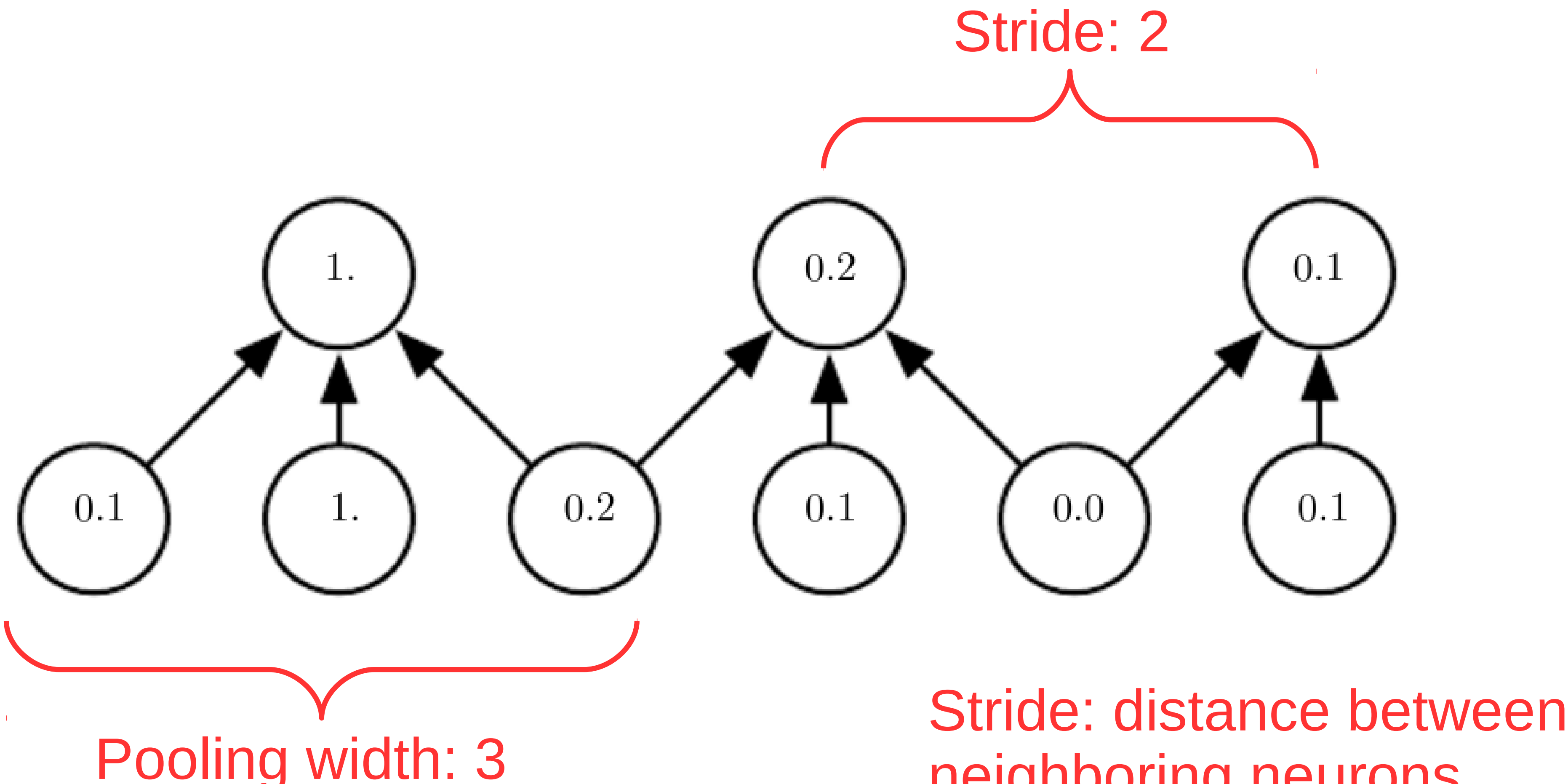
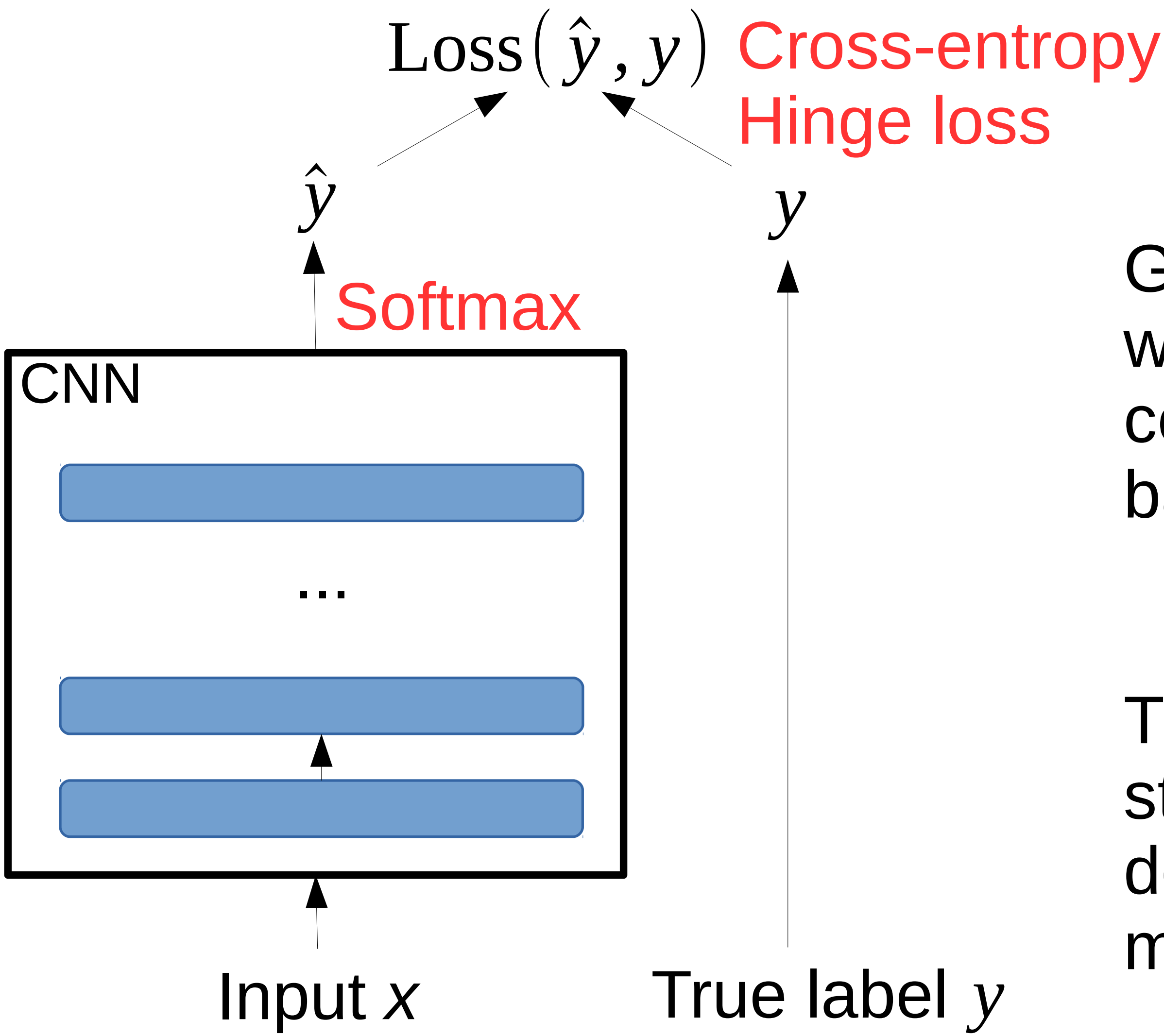



Figure 9.10 from Goodfellow et al. (2016).

# Putting everything together




Gradient of loss w.r.t. parameters are computed using backpropagation.

Then, use a stochastic gradient descent method to minimize loss.

# Cross-entropy

$$L(\theta) = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log q_{ic}$$

$y_i$ is a C-dimensional one-hot vector

$q_i$ is the softmax of f(x)

## What does softmax do?

- Normalizes the raw output scores by the neural network
- Emphasizes the max score

$$q_{ic} = \frac{e^{f_c(x_i)}}{\sum_k e^{f_k(x_i)}}$$

# Cross-entropy

$$L(\theta) = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log q_{ic}$$

Where does this expression come from?

$y_i$ is a C-dimensional one-hot vector

$q_i$ is the softmax of f(x)

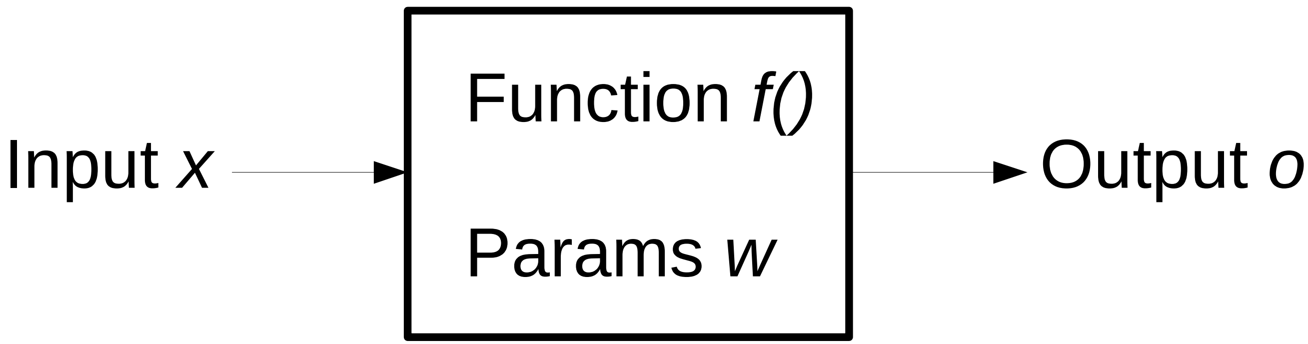
What does softmax do?

- Normalizes the raw output scores by the neural network
- Emphasizes the max score

$$q_{ic} = \frac{e^{f_c(x_i)}}{\sum_k e^{f_k(x_i)}}$$

# Modular Backpropagation

A computing block:

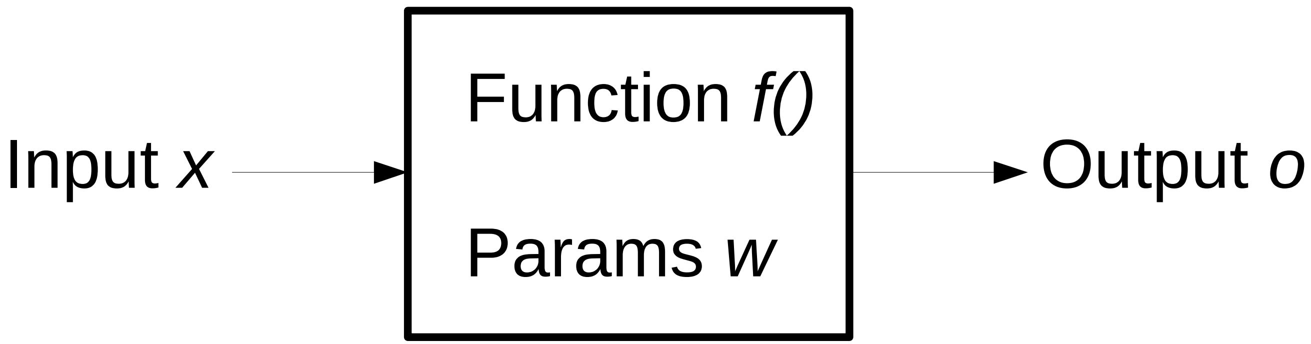



Forward pass: $o = f(x; w)$

Derivative of output w.r.t. input: $\frac{\partial o}{\partial x} = \frac{\partial f(x; w)}{\partial x}$

Derivative of output w.r.t. parameters: $\frac{\partial o}{\partial w} = \frac{\partial f(x; w)}{\partial w}$

A computing block:



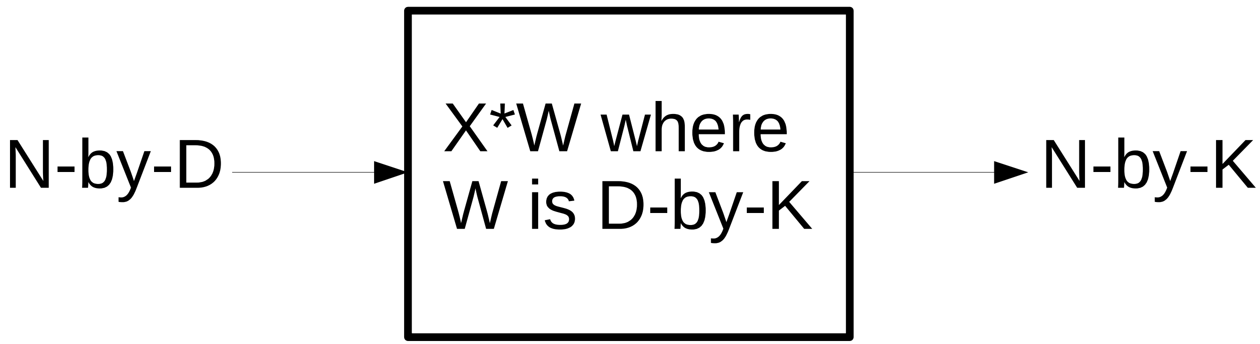

Typically, X, o and w are vectors or matrices. Care has to be taken in computing the derivatives.

Forward pass: $o = f(x; w)$

Derivative of output w.r.t. input: $\frac{\partial o}{\partial x} = \frac{\partial f(x; w)}{\partial x}$

Derivative of output w.r.t. parameters: $\frac{\partial o}{\partial w} = \frac{\partial f(x; w)}{\partial w}$

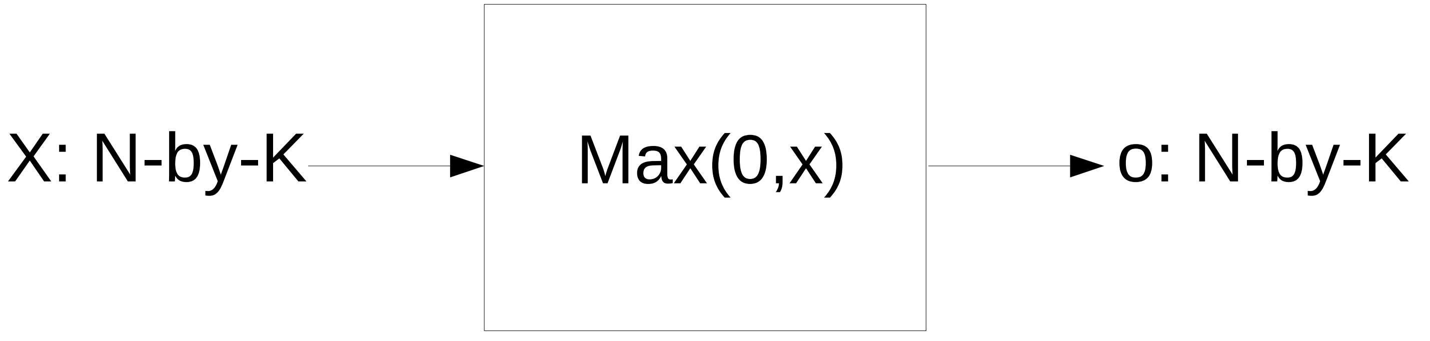
E.g. a fully connected layer with D input nodes and K output nodes, receiving N examples.




Derivative of output w.r.t. input: $\frac{\partial o}{\partial X} = W$

Derivative of output w.r.t. parameters: $\frac{\partial o}{\partial W} = X$

E.g. do the same for a ReLU layer receiving N-by-K input

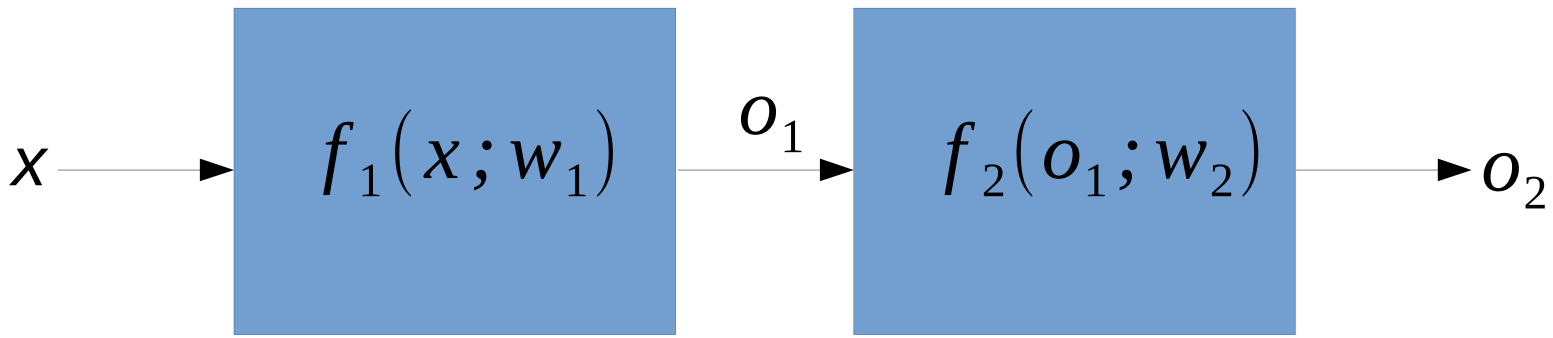



Derivative of output w.r.t. input:

$$\frac{\partial o}{\partial x_{ij}} = \begin{cases} 1 & \text{if} \quad x_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Derivative of output w.r.t. parameters: No parameters, nothing to learn

# Multiple blocks




To update $w_2$

$$\frac{\partial o_2}{\partial w_2}$$

To update $w_1$

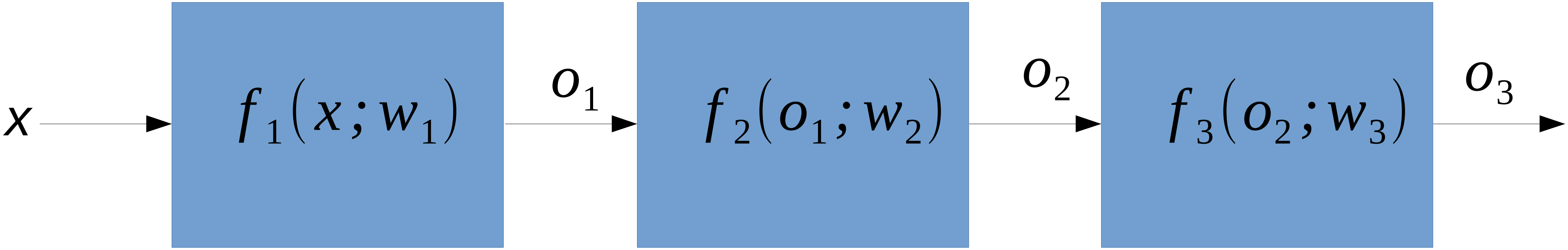
$$\frac{\partial o_2}{\partial w_1} = \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_1}$$

Each block has its own:

- Derivative w.r.t. input
- Derivative w.r.t. parameters.

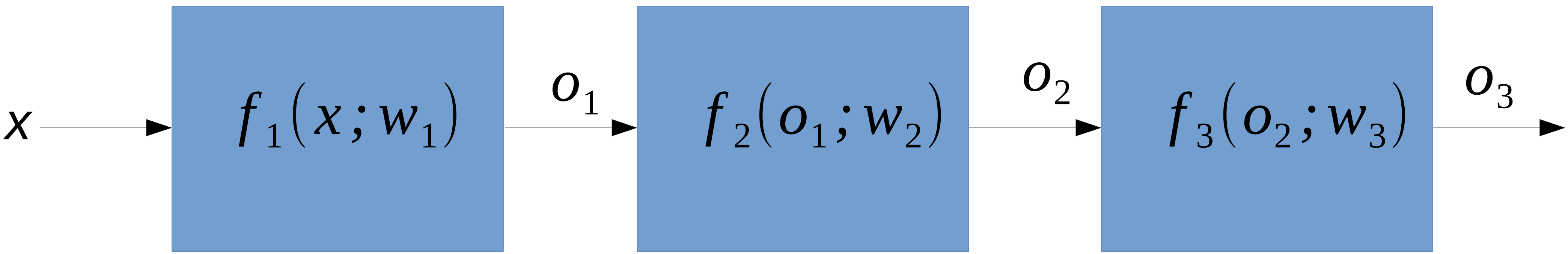
When you are back-propagating, be careful which one to use.

# Multiple blocks




$$\frac{\partial o_3}{\partial w_1} = \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_1}$$

# Multiple blocks




$$\frac{\partial o_3}{\partial w_1} = \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_1}$$

Last step: multiply with derivative w.r.t. parameters

Chain the “derivatives w.r.t. to input”

# References


- Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013a).Maxout networks. In S. Dasgupta and D. McAllester, editors, ICML’13 , pages 1319–1327
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
- LeCun, Y. (1989). Generalization and network design strategies. Technical Report. CRG-TR-89-4, University of Toronto.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.