CS698U: Topics in Computer Vision

Jan—May 2017

Lecture 2

Gaurav Sharma

Indian Institute of Technology Kanpur

www.grvsharma.com

Image Classification

Dogs — retriever, pug …
People — men, women, children …

Park, Trees, Fence …

Humans can recognize O(10k) object categories very quickly …

Object detection

"*Diego Delso*, ***Wikimedia Commons, License*** *CC-BY-SA 3.0*

Person

Elephant

Building

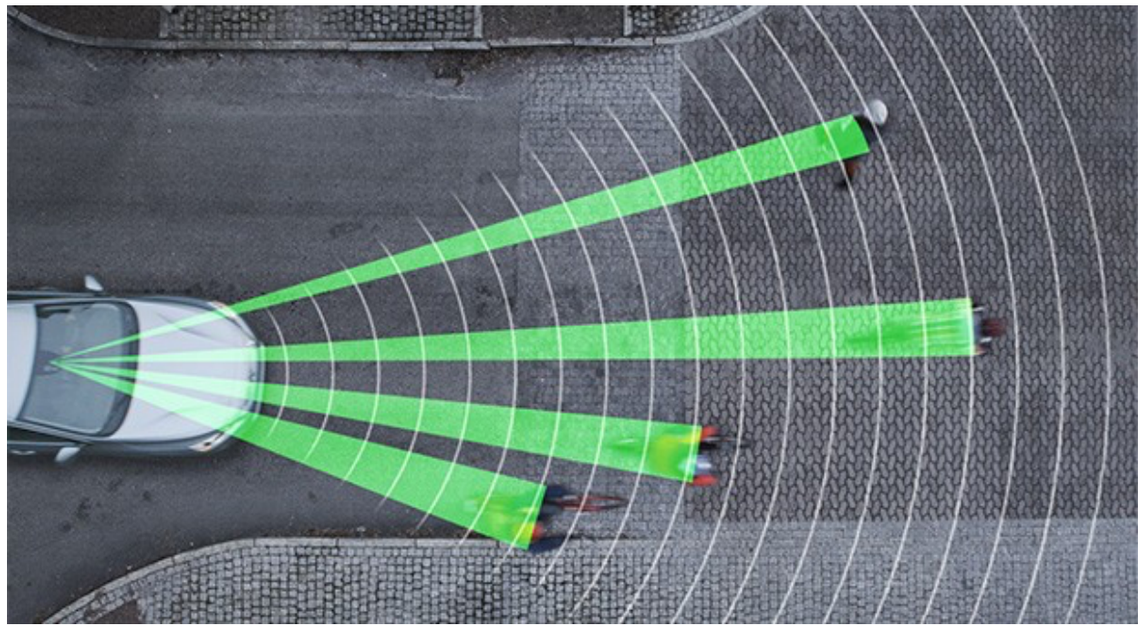
Bicycle

...

Automotive Industry …

http://images.hgmsites.net/med/toyota-night-view-features-pedestrian-detection_100211506_m.jpg

Toyota night view



https://www.engadget.com/2013/03/06/volvo-cyclsist-detection-automatic-breaking-system/

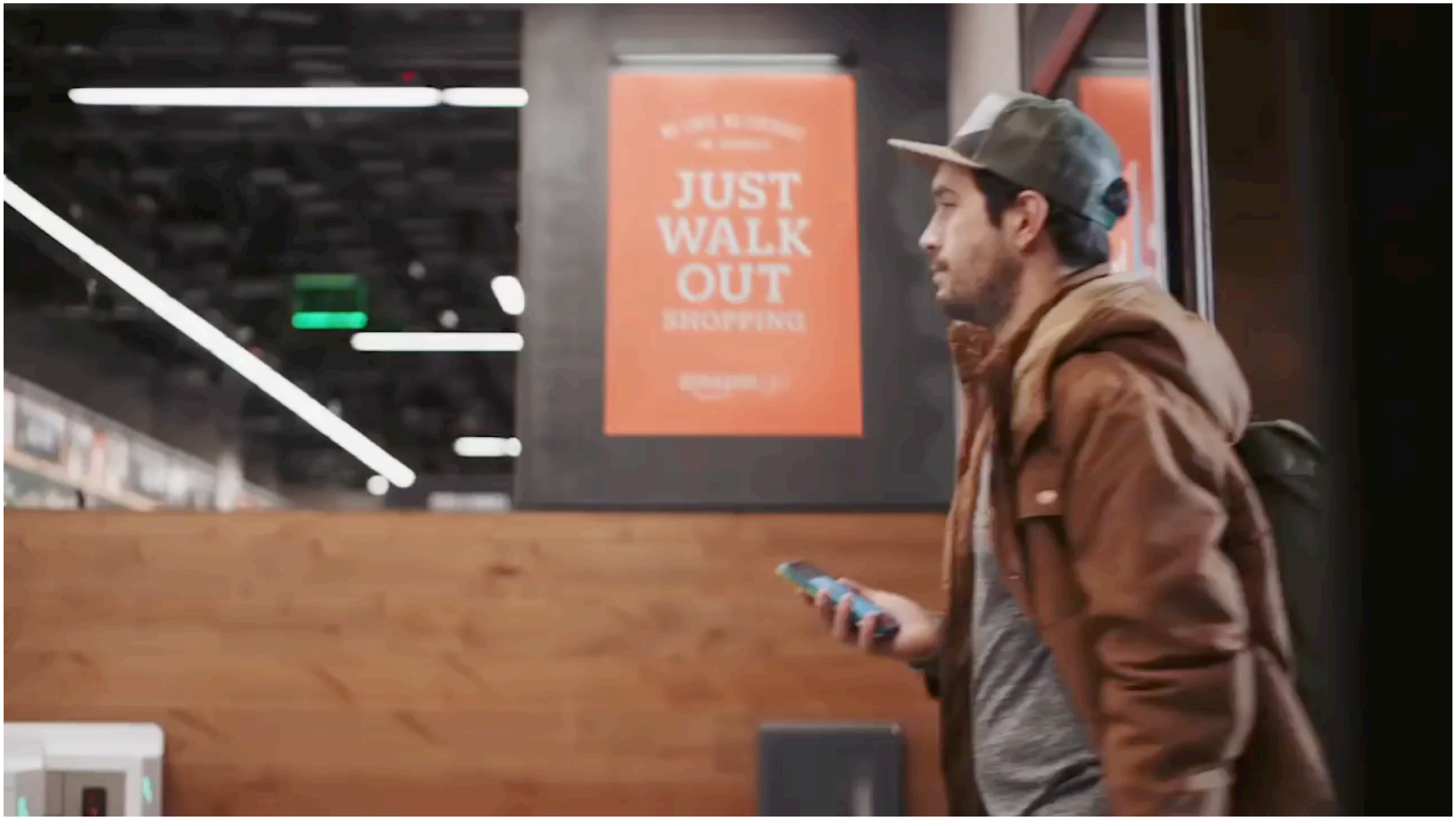
Volvo braking assist

Felzenszwalb et al., Object detection with discriminatively trained part-based models, TPAMI 2010

Smart cars …

Autonomous cars …

Closer to home …



Outline

- Basics, perceptron, multi-layer perceptron …
- Convolutional Neural Networks
  - Basic components, backpropagation
  - State-of-the-art CNNs in Computer Vision
    - AlexNet, VGG-16, GoogLeNet, ResNets
    - Object detection architectures
- Question … **interrupt at any time**

Preliminaries (assumed)

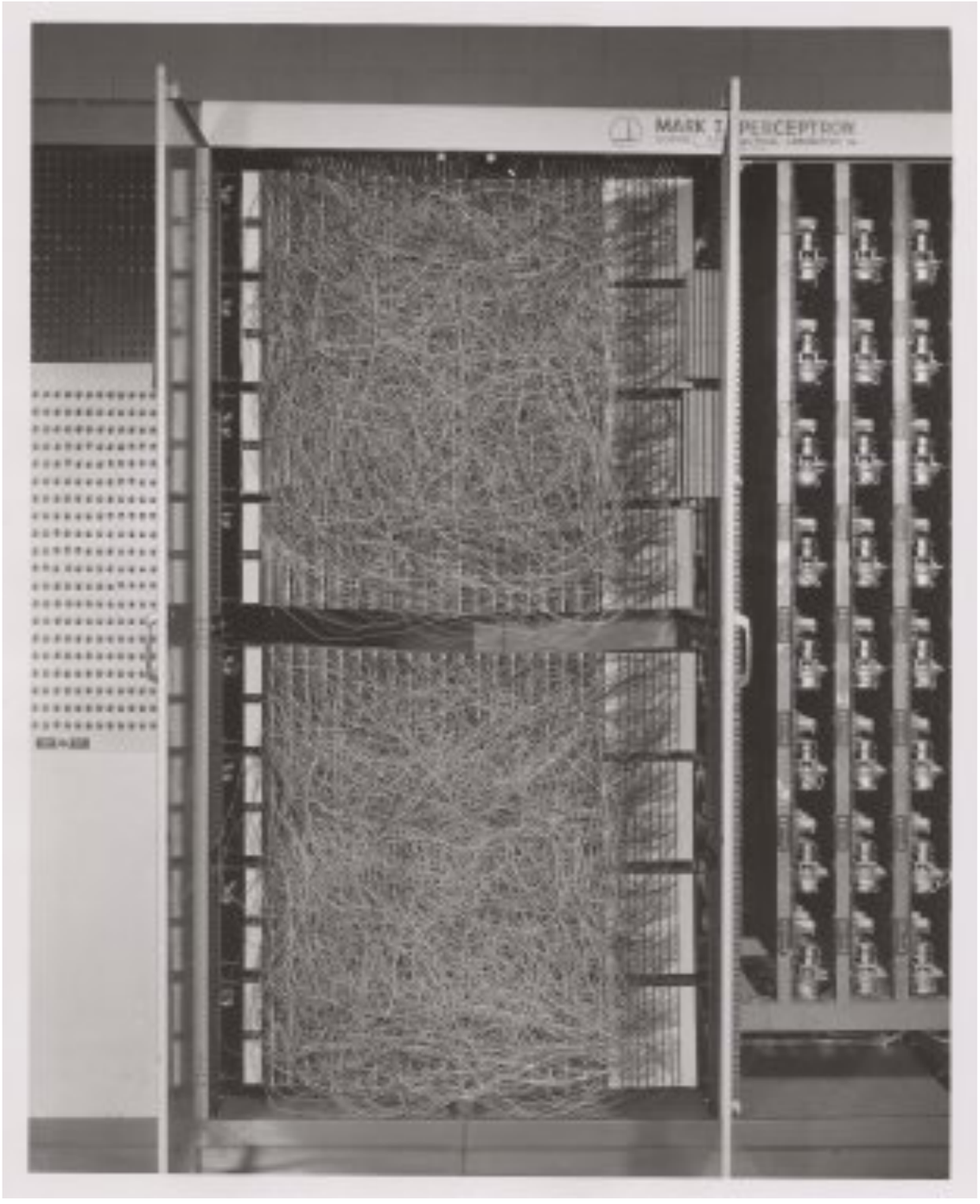
- Classification
  - Binary — is it a dog or not ?
  - Multi class — which one is it, a dog or a cat, or something else ?
  - Multi label — more than one possible; dog, animal, wood, mahogany, tree?
- Learning by empirical loss minimization
- Other tasks exist as well, e.g., regression, ranking

Perceptron Algorithm 1957

by Frank Rosenblatt
at Cornell Aeronautical Lab

The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays of potentiometers that implemented the adaptive weights

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



Perceptron Algorithm 1957

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Score $y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j]$

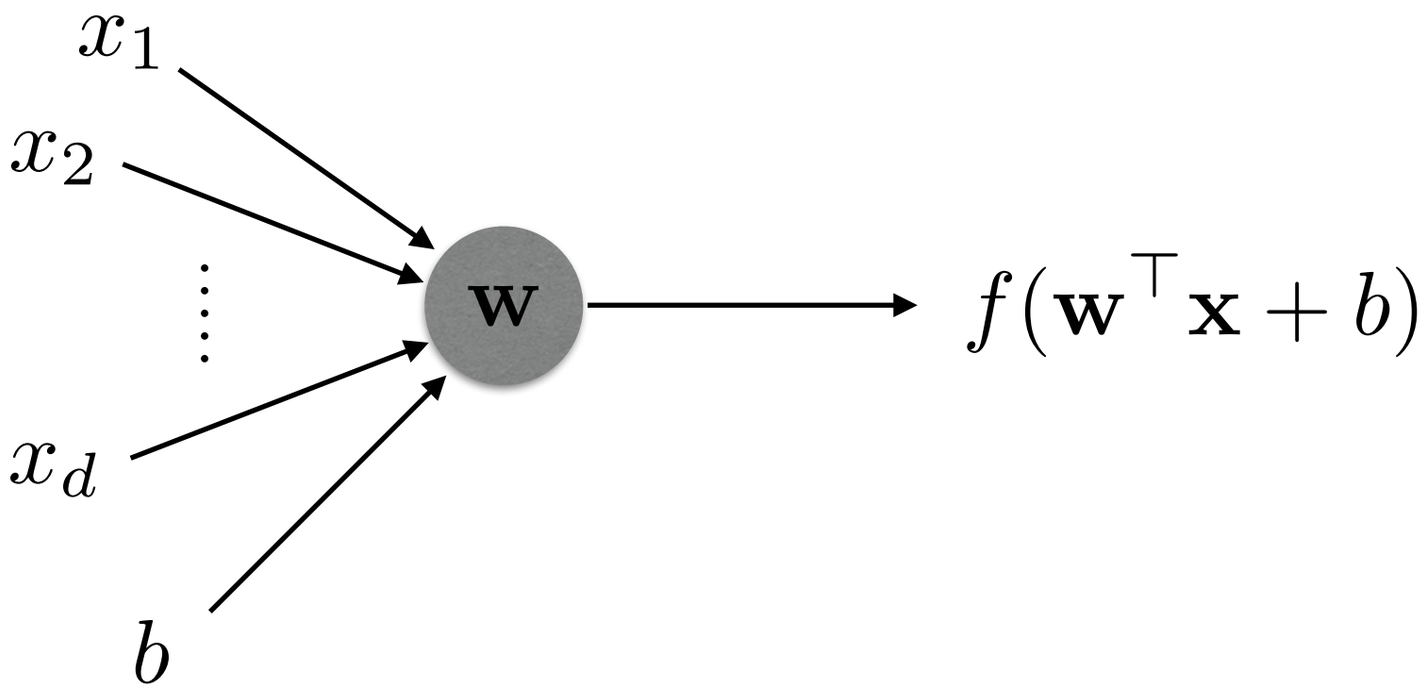
Update $w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{j,i}$

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

Multilayer Perceptron (MLP)

- Feedforward Artificial Neural Network
- Directed Graph — each layer fully connected to next
- Node = neuron with nonlinear activation function
- Learnt using *Backpropagation*
- Feedforward vs. Recurrent (another lecture)

Neuron or processing unit



$\mathbf{x} = [x_1, x_2, \ldots x_d]$ is the input vector

$\mathbf{w} = [w_1, w_2, \ldots w_d]$ is the weight vector

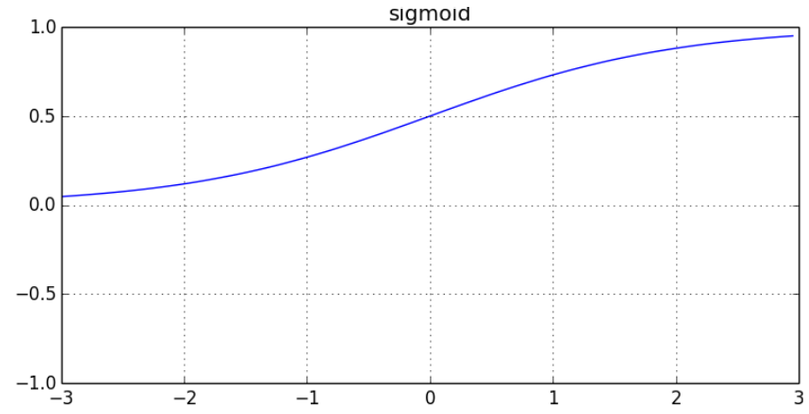
b is the bias

$f(\cdot)$ is the activation function

Different activation functions

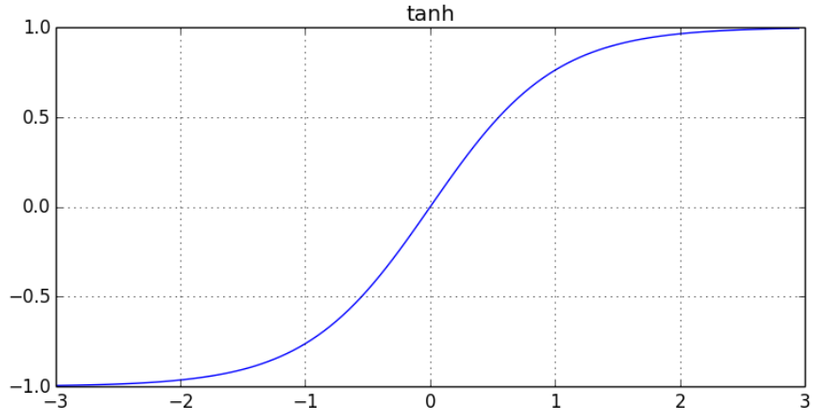
- Logistic

$$f(z) = \frac{1}{1 + e^{-z}}$$



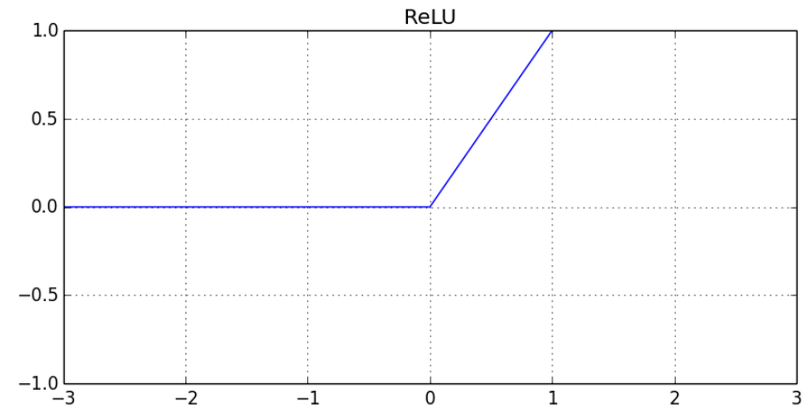
- Hyperbolic tan

$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



- Rectified Linear Unit (ReLU)

$$f(z) = \max(0, z)$$

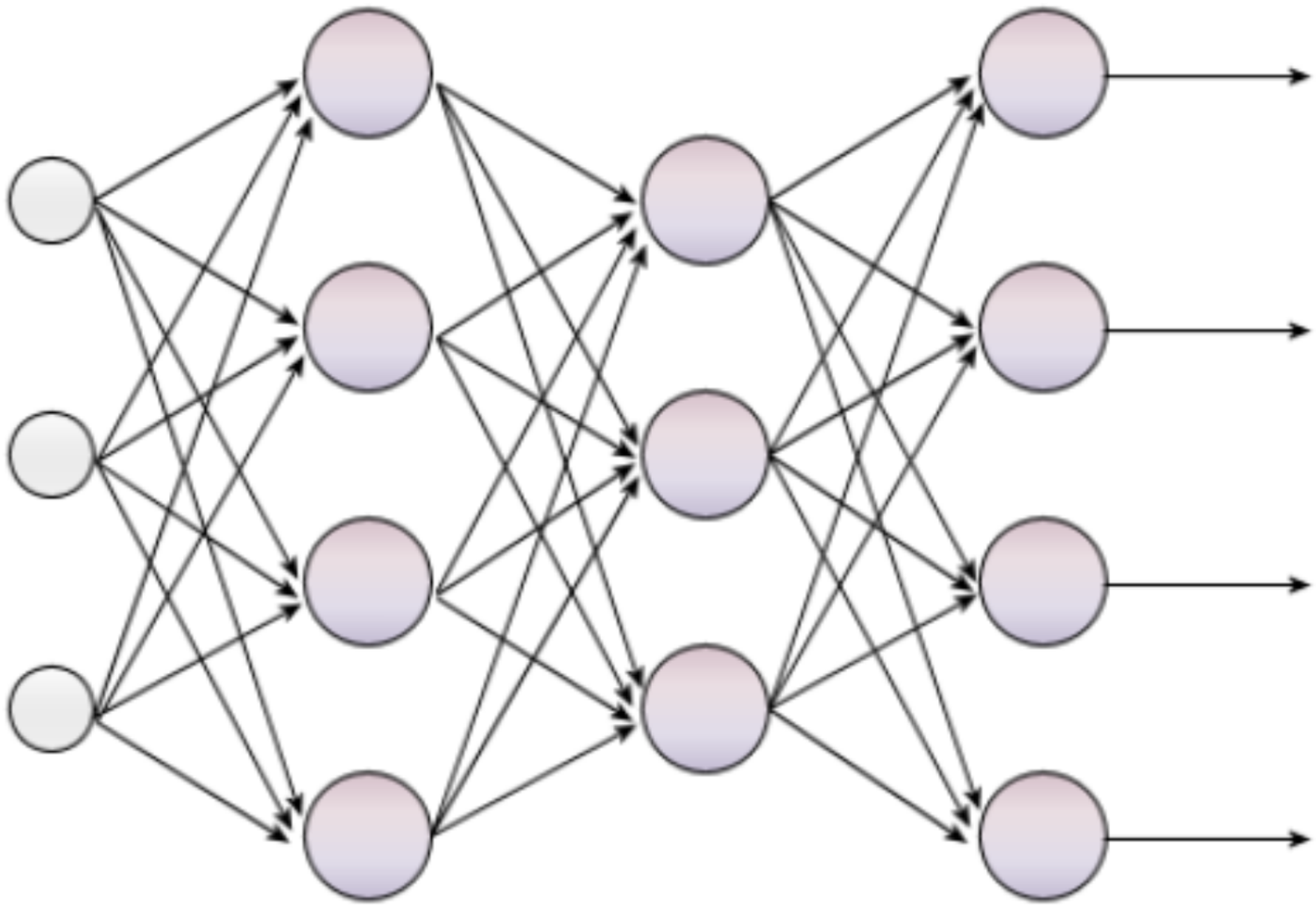


Multilayer Perceptron

Input layer

Hidden layers

Output layer



https://commons.wikimedia.org/wiki/File:Perceptron_4layers.png

ELSEVIER

Neural Networks

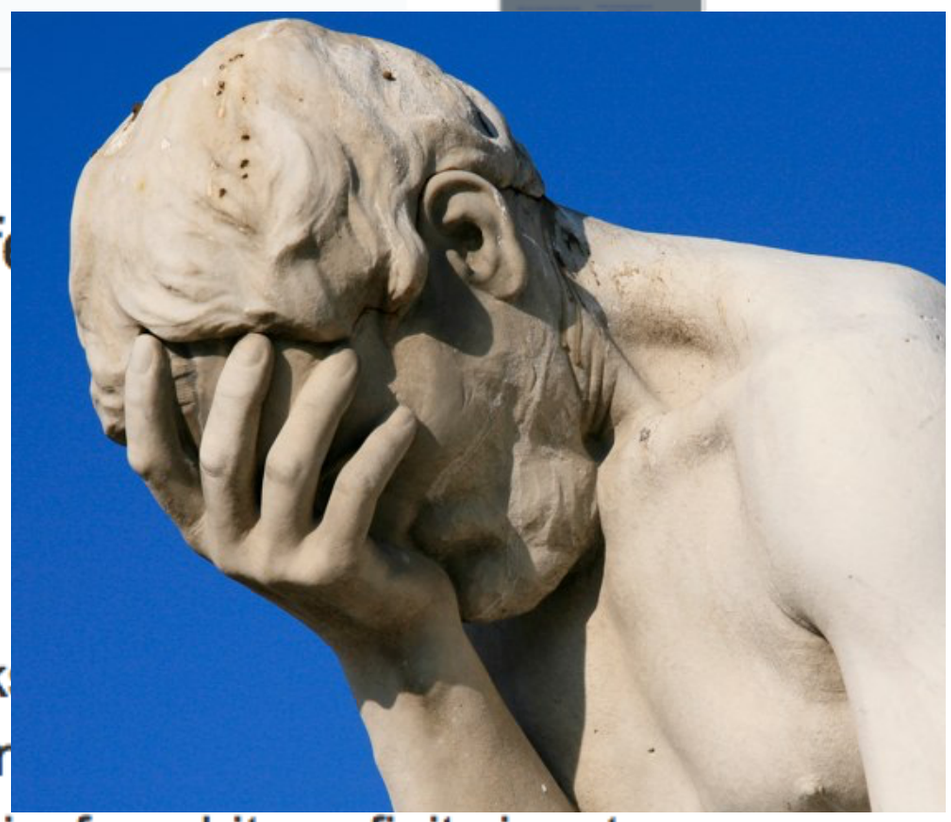
Volume 4, Issue 2, 1991, Pages 251–257

Original contribution

Approximation capabilities of multilayer f

Kurt Hornik

Technische Universität Wien, Vienna, Austria



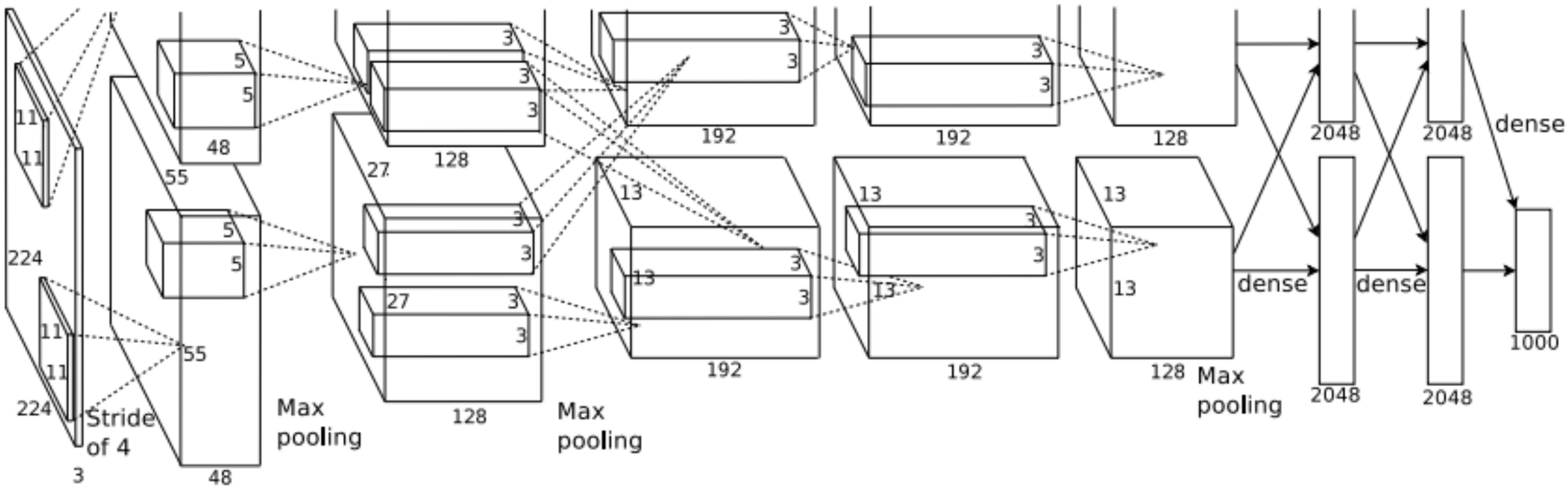
Abstract

We show that standard multilayer feedforward network
layer and arbitrary bounded and nonconstant activation
approximators with respect to $L^p(\mu)$ performance criteria, for arbitrary finite input
environment measures μ, provided only that sufficiently many hidden units are available.
If the activation function
mappings can be learned uniformly over compact input sets. We also give very general
conditions ensuring that networks with sufficiently smooth activation functions are
capable of arbitrarily accurate approximation to a function and its derivatives.

PC: Alex E. Proimos at http://flickr.com/photos/34120957@N04/4199675334

Convolutional Neural Networks

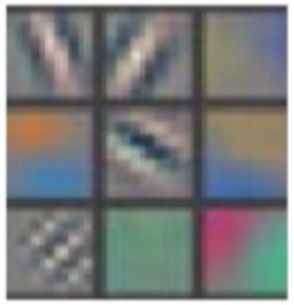
Convolutional Neural Networks



60 million parameters; 650,000 neurons

Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

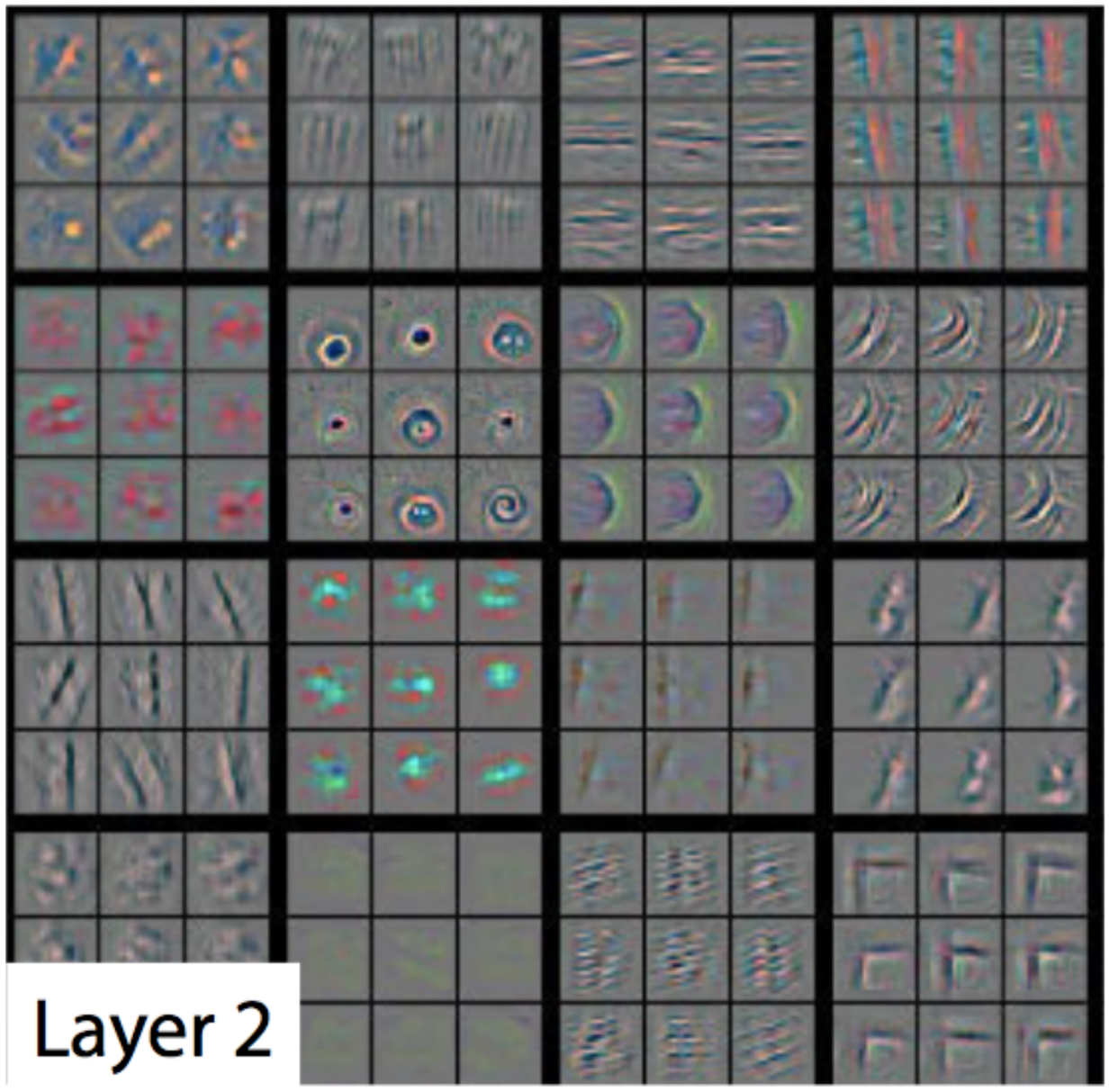
Hierarchy of Features

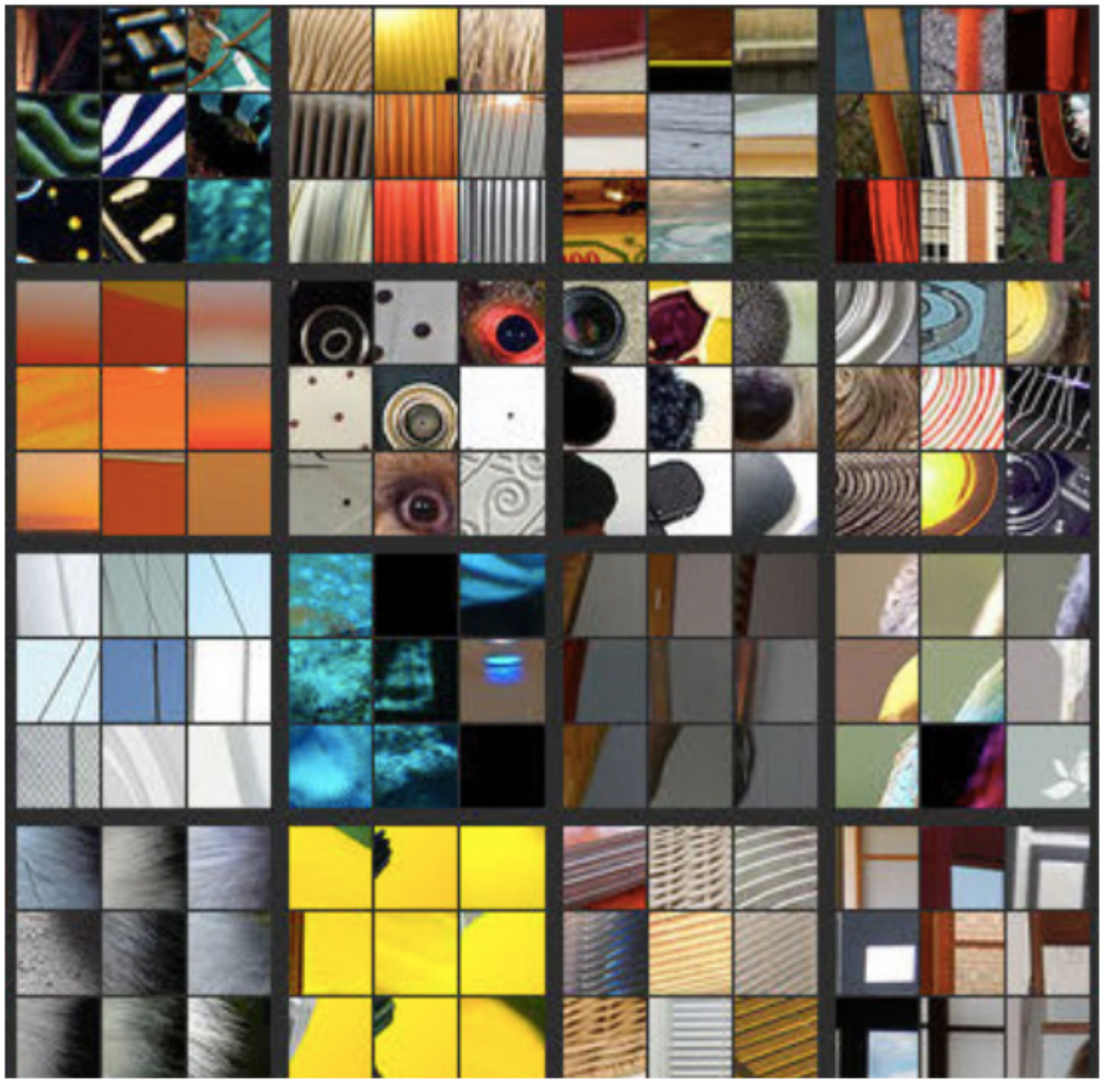


Layer 1

Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014

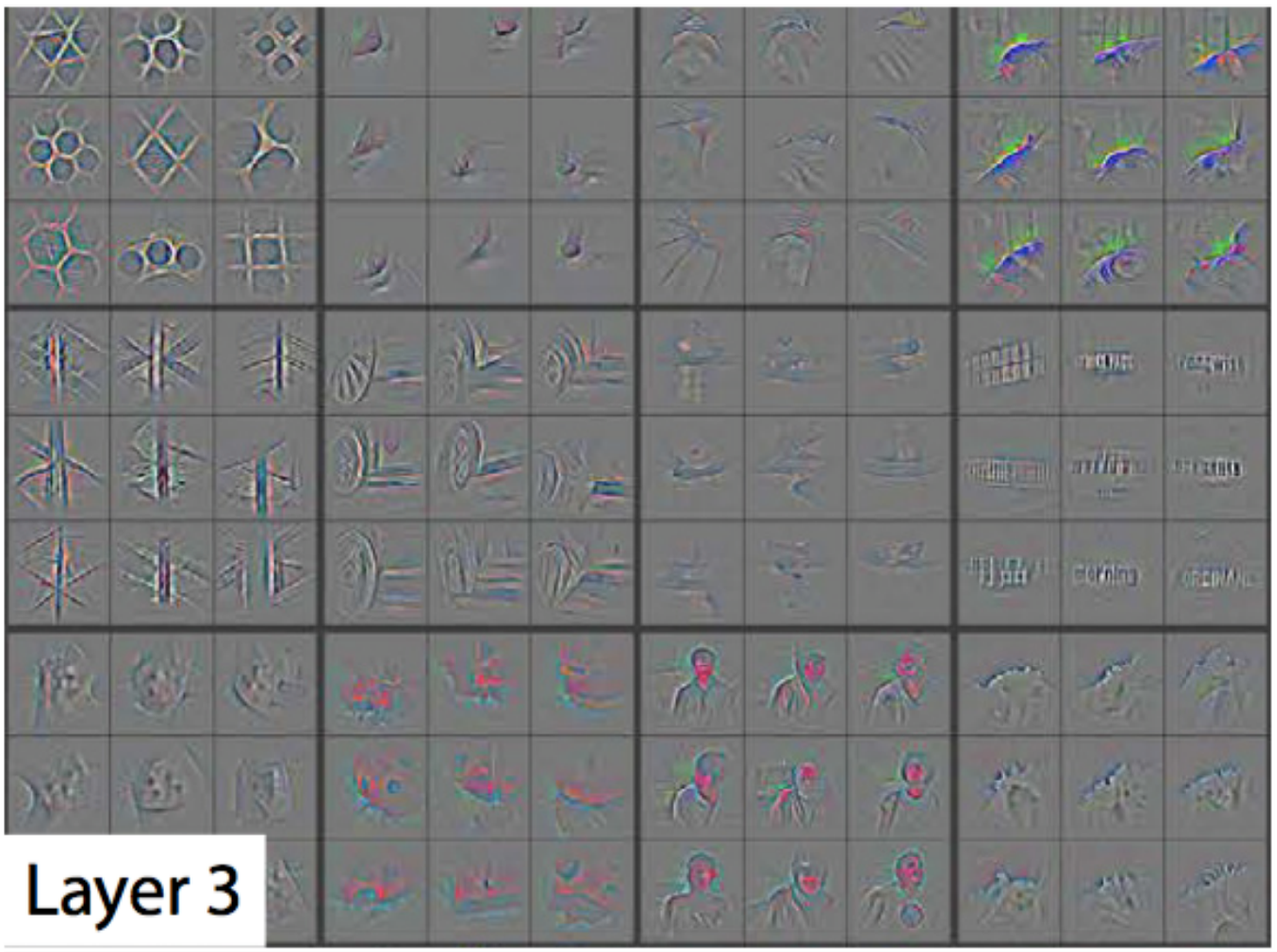
Hierarchy of Features





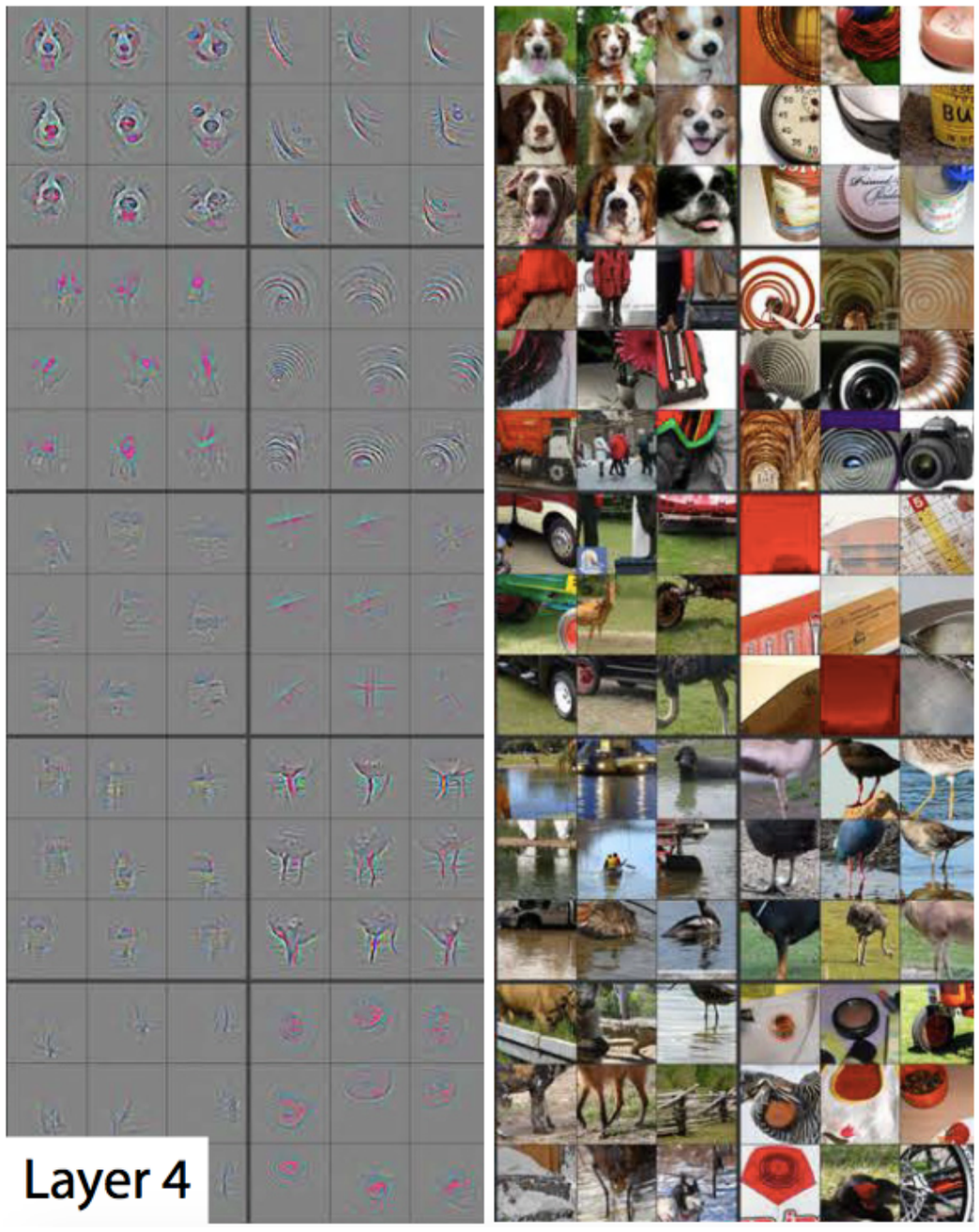
Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014

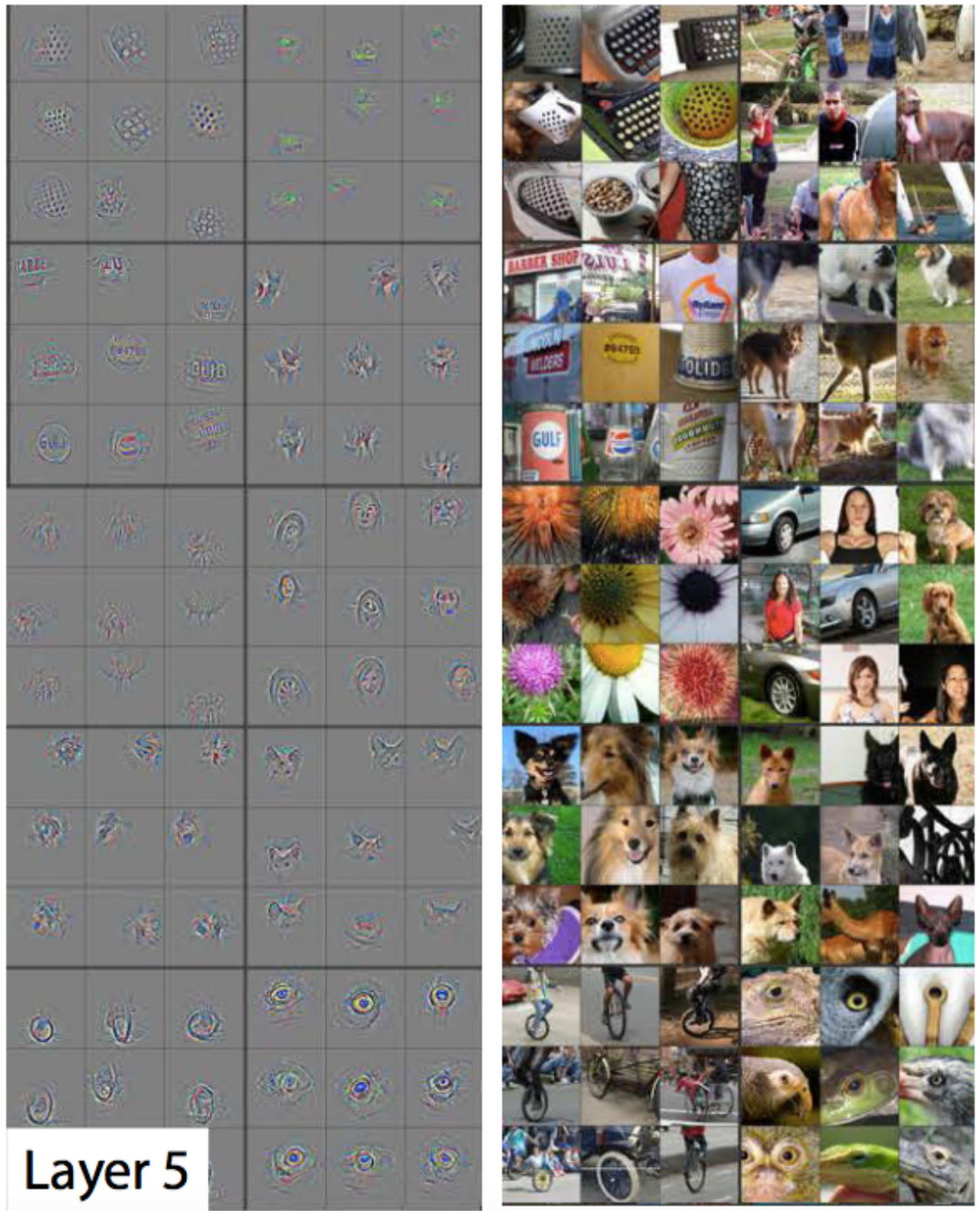
Hierarchy of Features



Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014

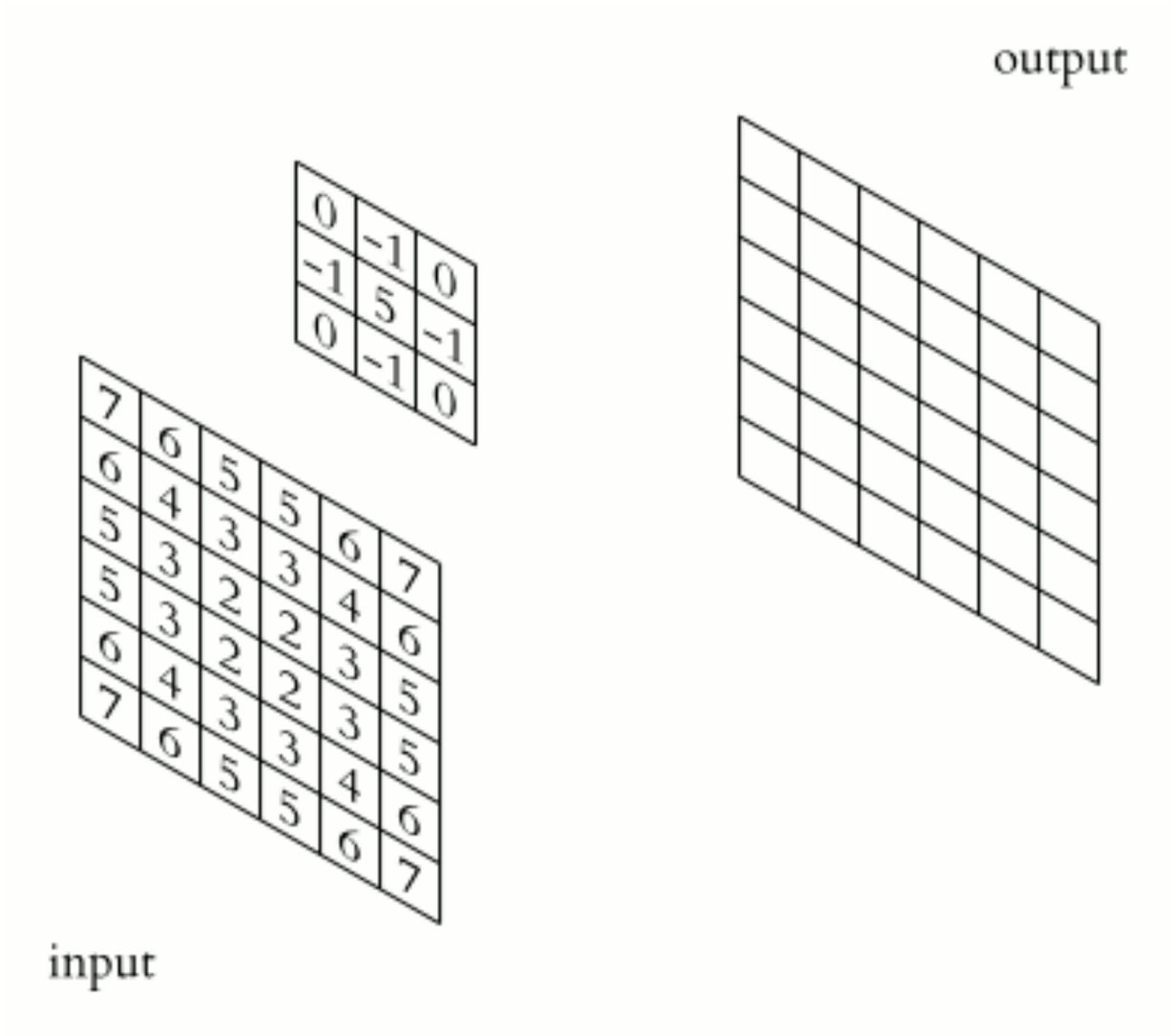
Hierarchy of Features





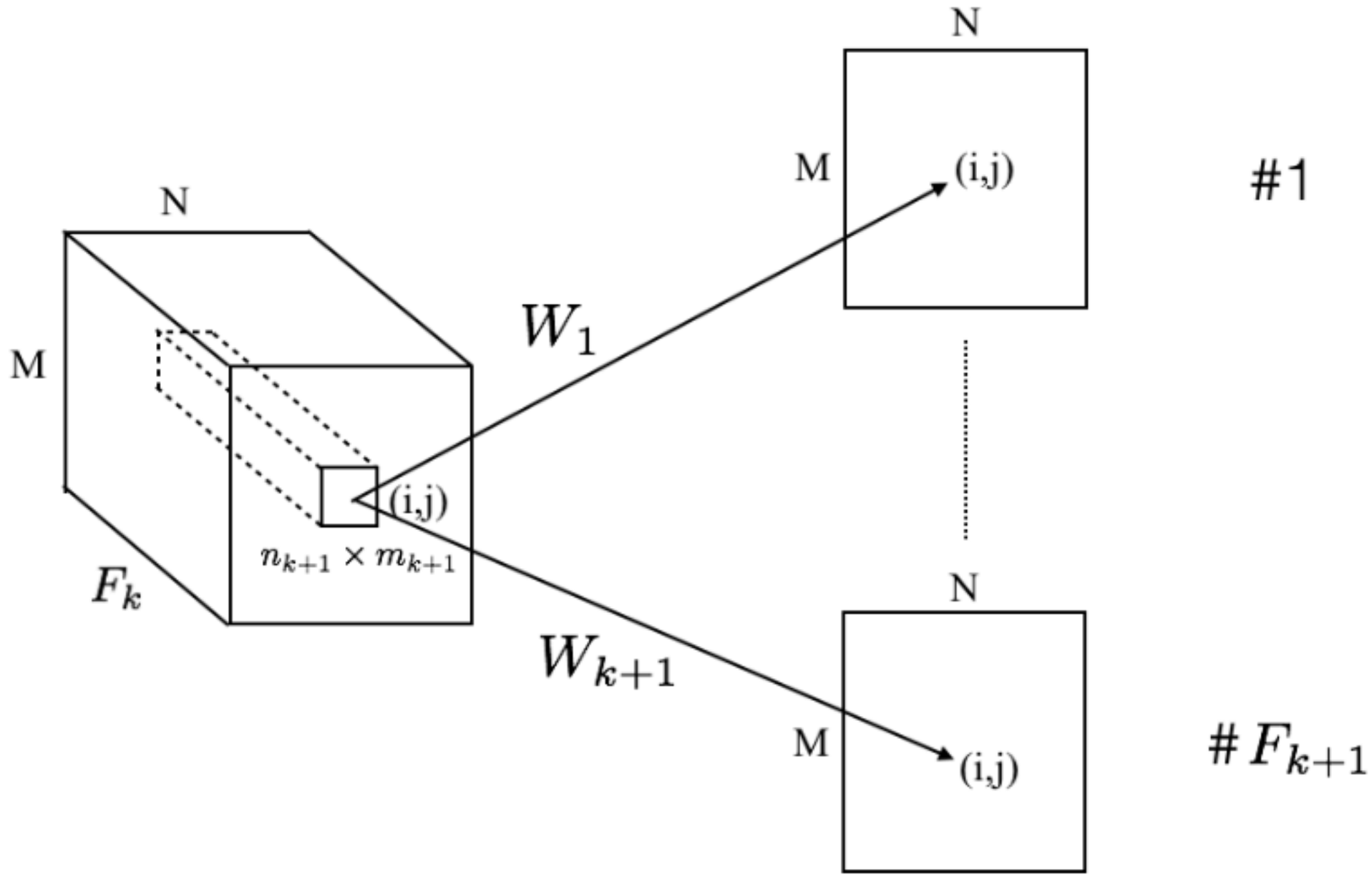
Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014

(2D) Convolution Layer



Animation by: Michael Plotke, CC-BY-SA-3.0
https://commons.wikimedia.org/wiki/File:3D_Convolution_Animation.gif

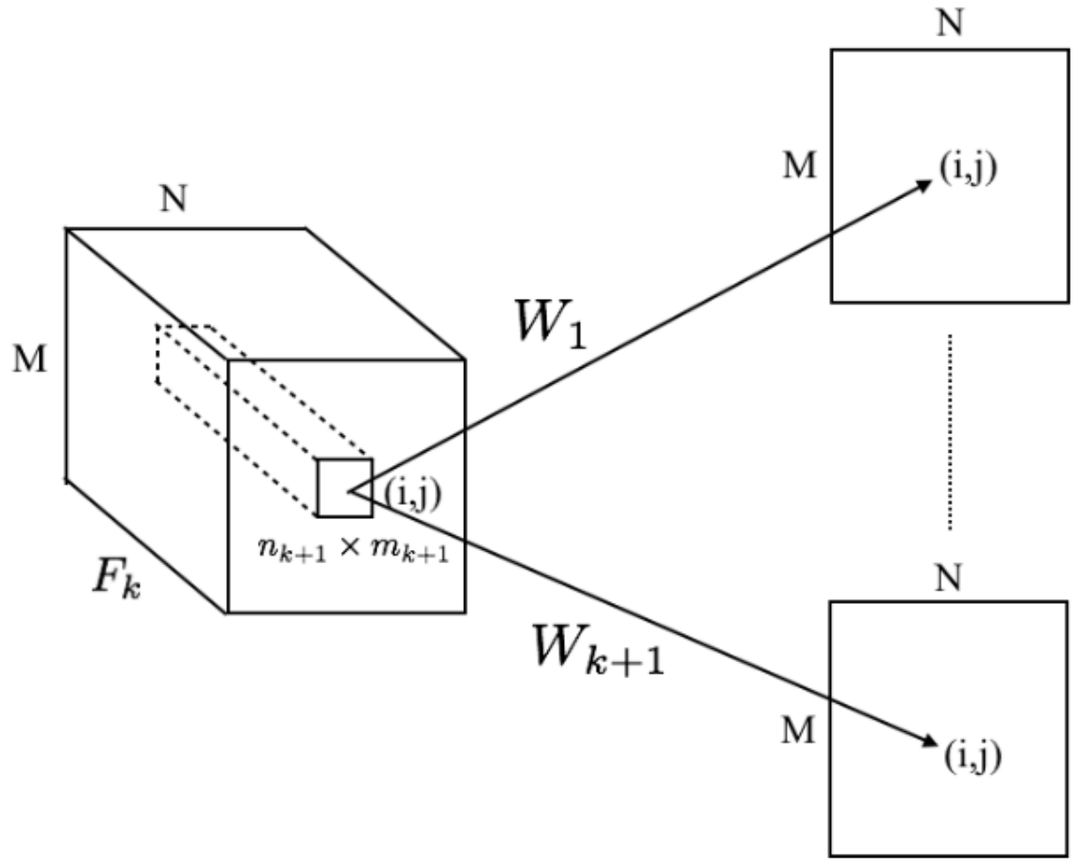
(3D) Convolution



(3D) Convolution

- Kernels are tensors

- Receptive field = spatial size of the kernel
  - here $n_{k+1} \times m_{k+1}$



- Stride = the jump in successive operation
  - here 1

Max Pooling

- Take the max in a subregion of the feature map
  - Non-linear subsampling

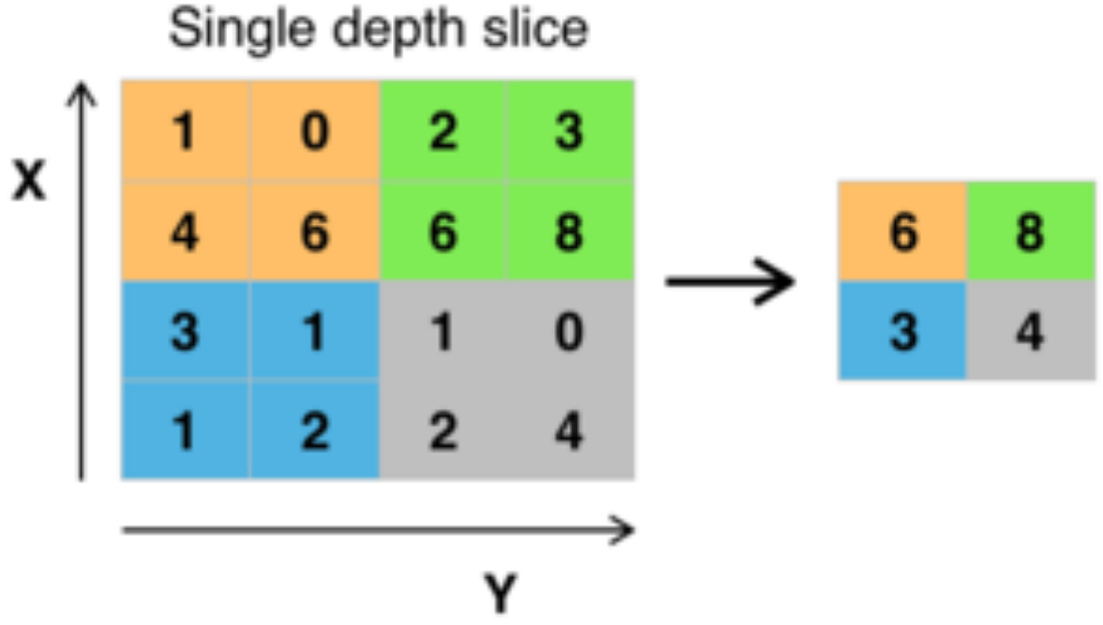
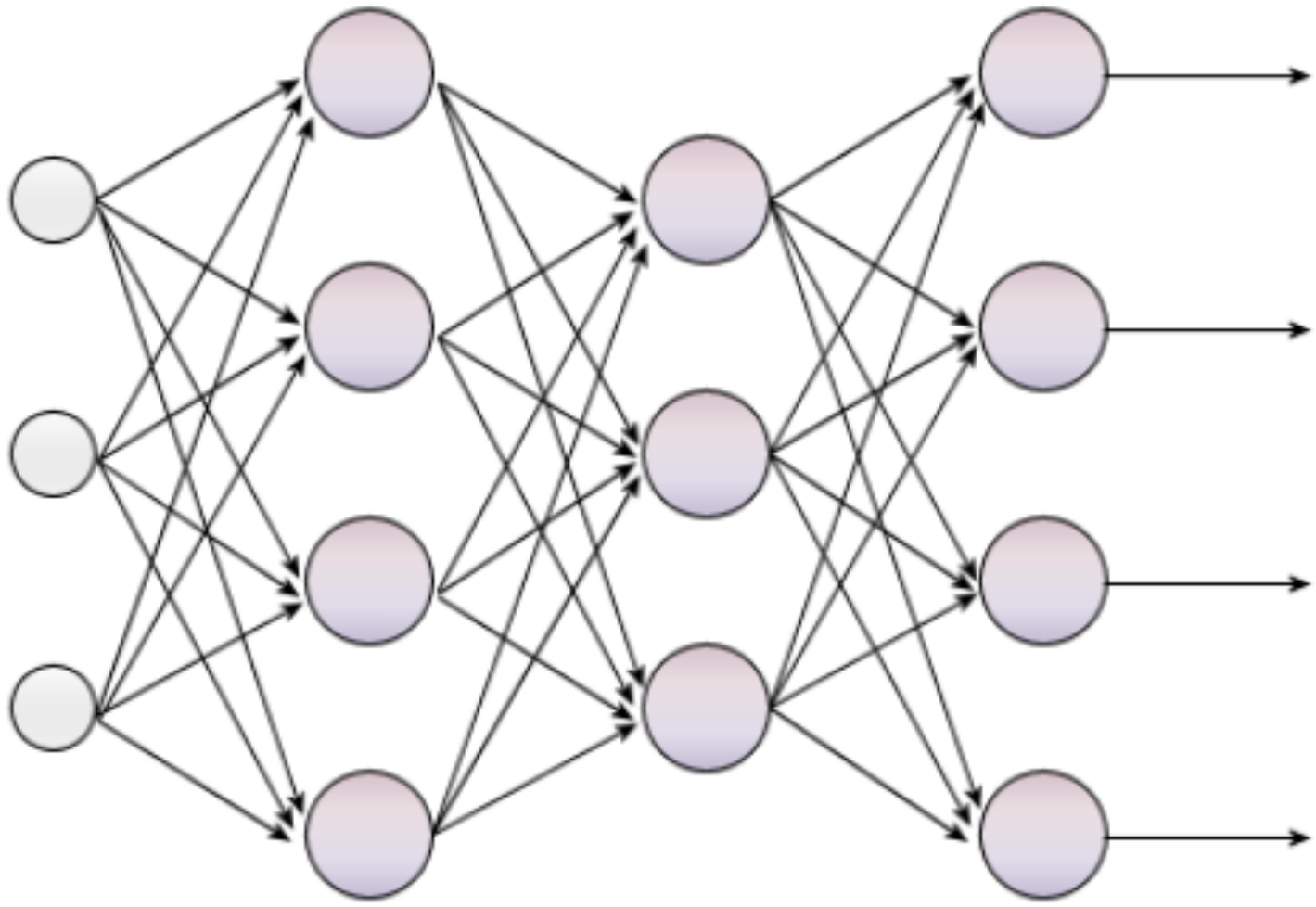


Photo by Aphex34 - Eigenes Werk, CC-BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45673581

Fully Connected Layer

- Basically matrix multiplication

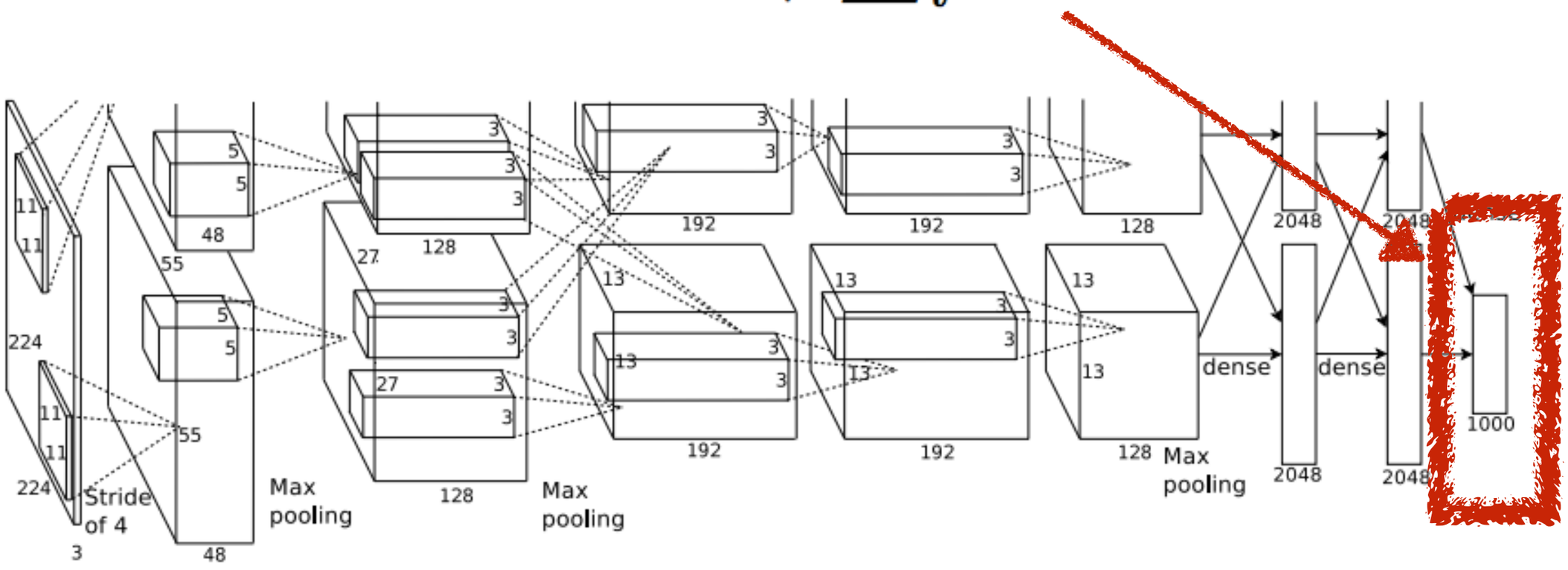


https://commons.wikimedia.org/wiki/File:Perceptron_4layers.png

Softmax Layer

- Squash the scores to be between [0,1]
  - With sum equal to 1

$$\tilde{z}_l = e^{z_l} / \sum_l e^{z_l}$$



Architecture ?

- How to design the right CNN architecture ?
  - Art more than science
  - We will see some of the best (for computer vision) CNNs in a while …
  - Usually some conv layers (w/ max-pooling) followed by some FC layers — with some engineering in between

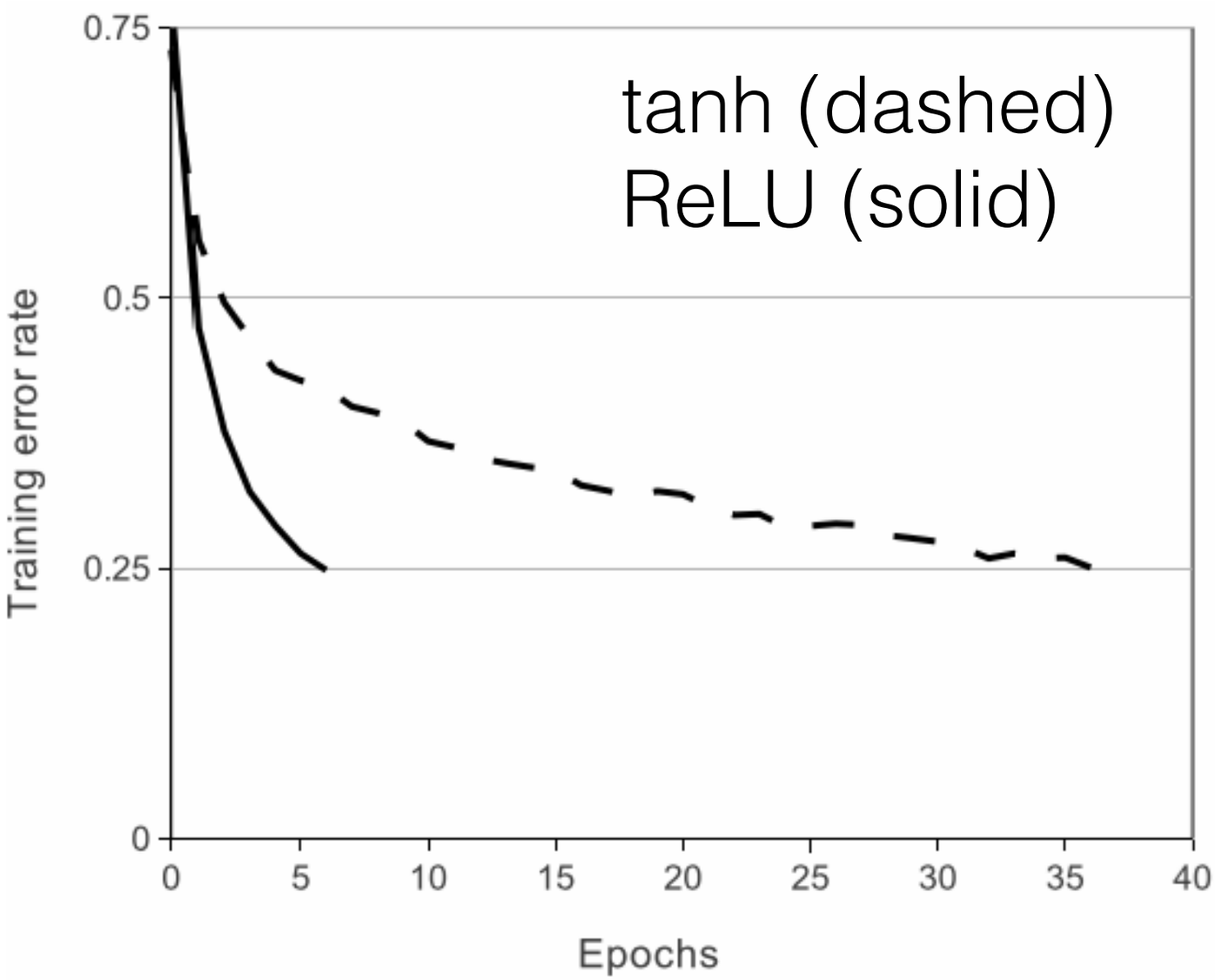
Parameters to be Learnt

- Weights (and biases) for all layers

Design choices

- Number/type of layers
- Conv kernel sizes
- Strides
- Activation function

Which activation function ?



Decide empirically

Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

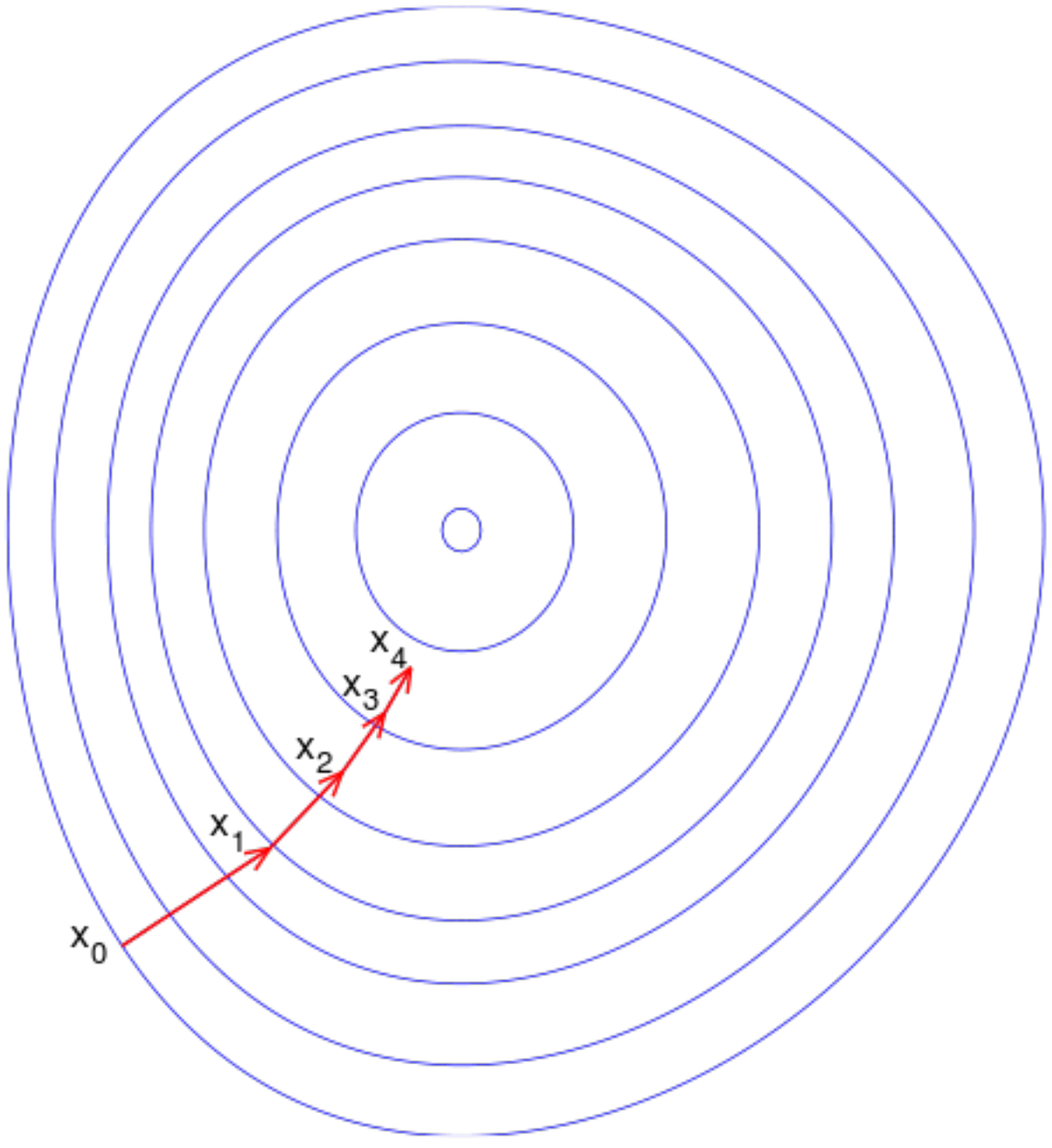
Backpropagation

- How to learn the parameters of CNN ?
- Given a training set $\{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_p, \mathbf{t}_p)\}$
- Minimize error (o is the output)

$$E = \frac{1}{2} \sum_{i=1}^{p} \|\mathbf{o}_i - \mathbf{t}_i\|^2$$

Chapter 7: R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996
https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

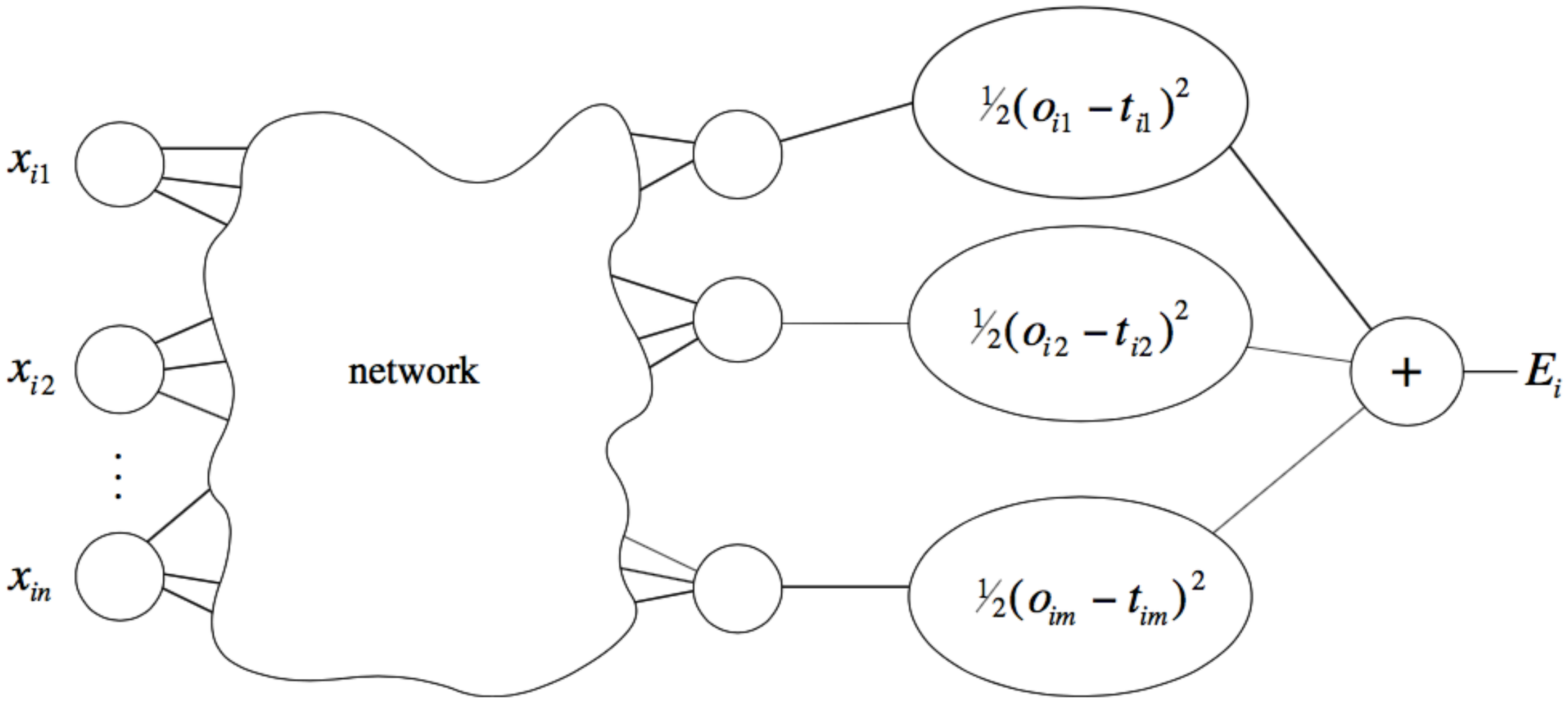
Gradient Descent (short recap)



- Iterative numerical method for minimizing a differentiable function

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0$$

Extended Network



Chapter 7: R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996
https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

Gradient Descent for CNN

- We need gradient wrt all weights

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_\ell}\right)$$

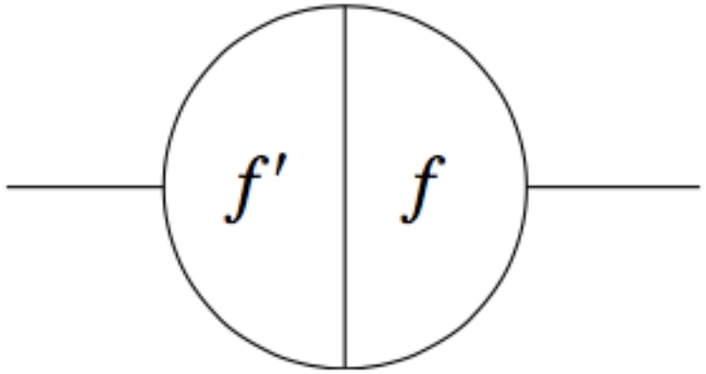
- For updating weights with

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \quad \text{for } i = 1, \ldots, \ell$$

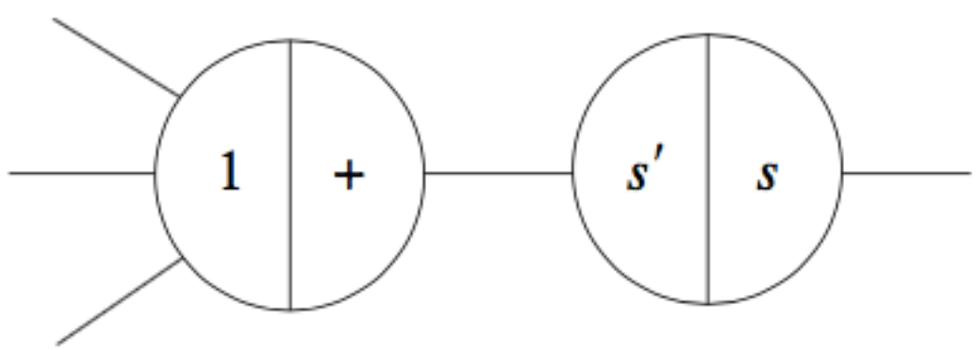
Chapter 7: R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996
https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

B-diagram

- Composite structure of nodes
  - B-diagram = Backpropagation diagram

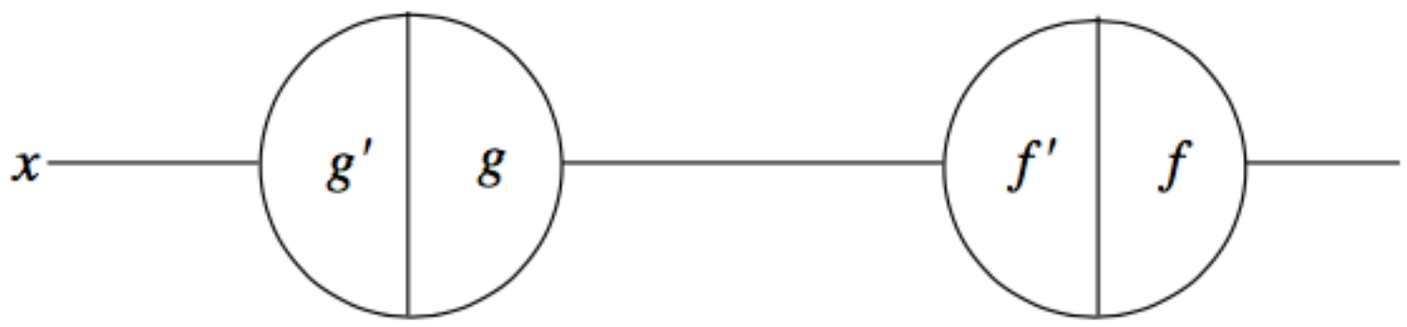


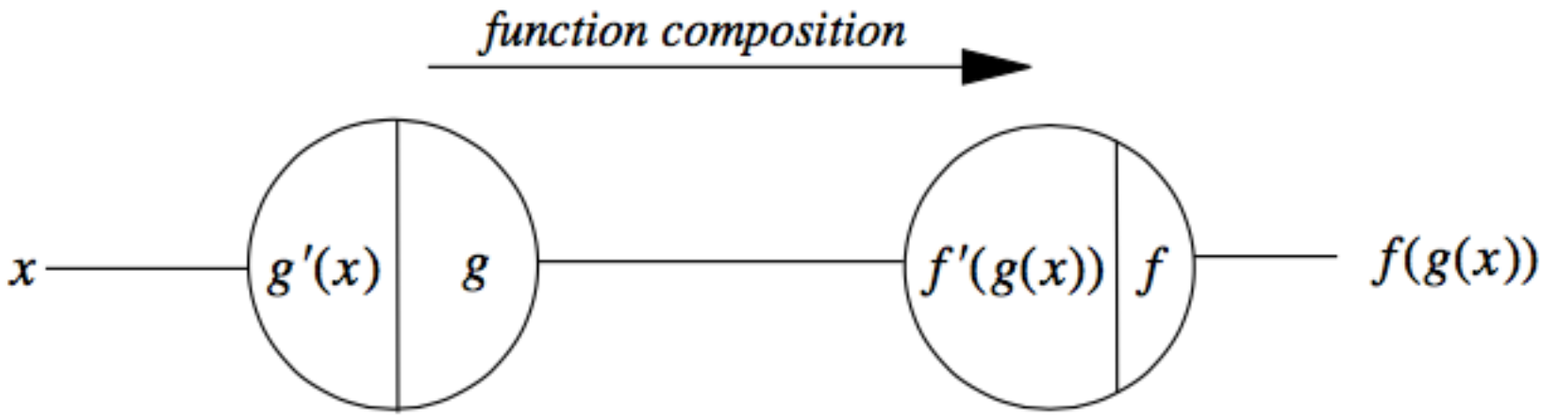
Node applying function f()



Separating integration and activation function

B-diagram

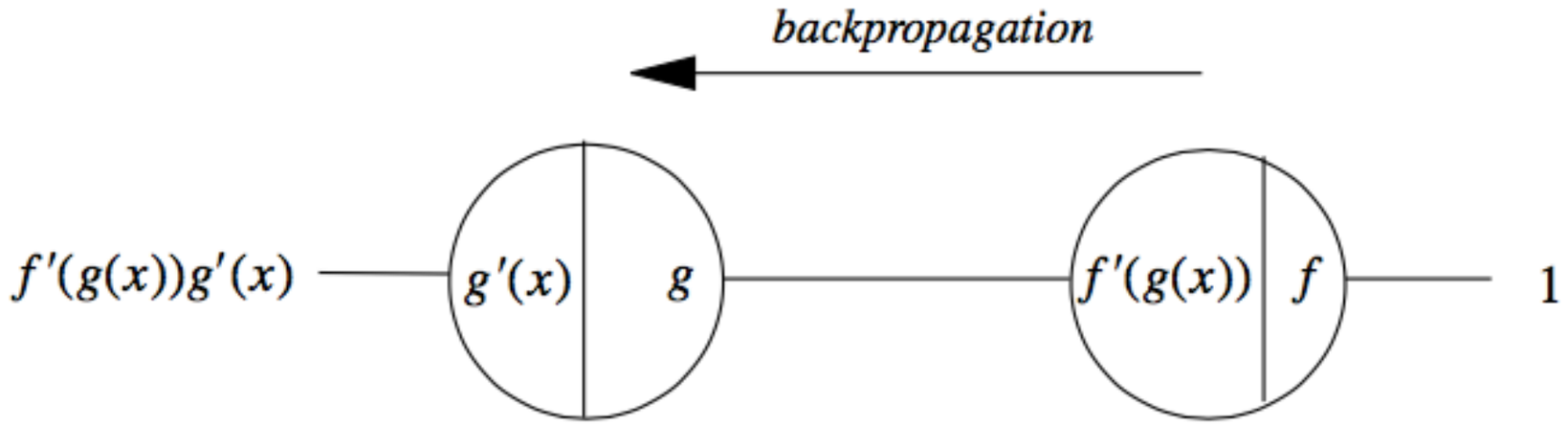




Forward pass

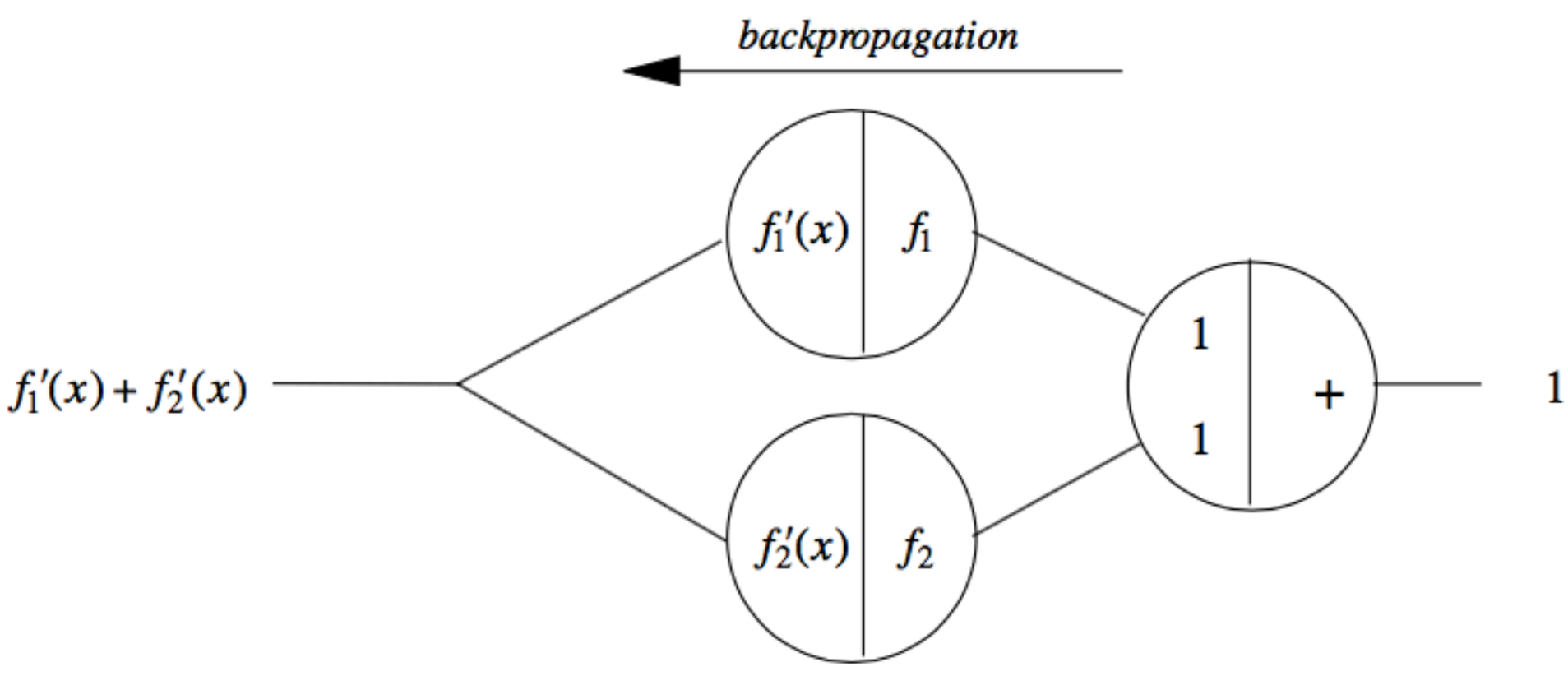
Backpropagation

- Feed 1 backwards into the network
- Multiply the input to the local gradient stored



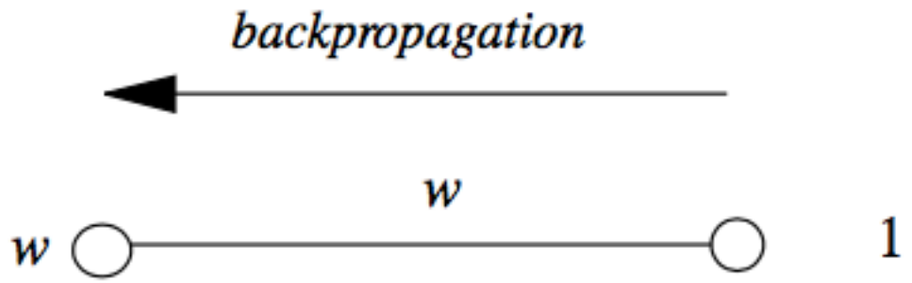
Backpropagation

- For converging edges, add the inputs



Backpropagation

- For weighted edges, multiply weight with input



Backpropagation

- Algorithm

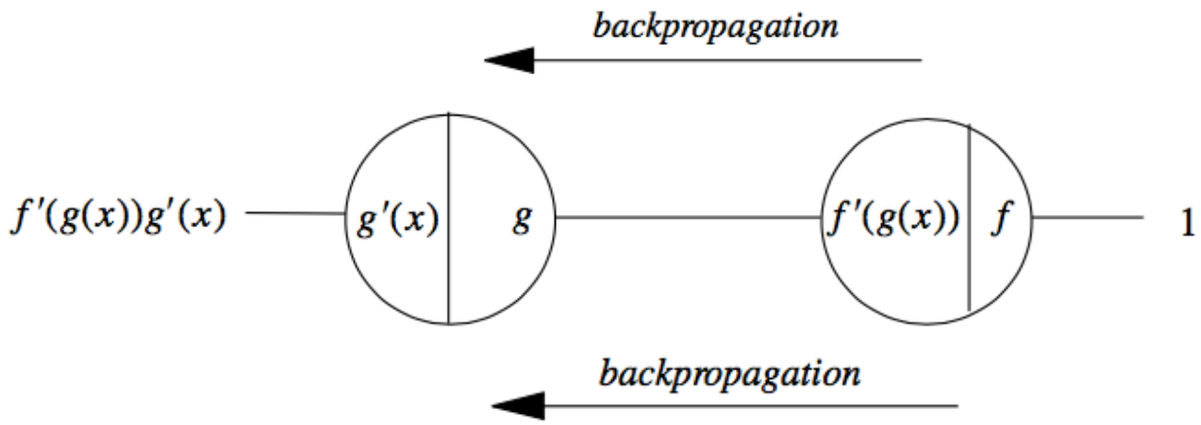
    1. Do forward pass with input x and save local gradient at each node
    2. Do backpropagation by feeding 1 back into the network till the input node

- Proposition: *The final value obtained is the gradient of network function wrt the input x*

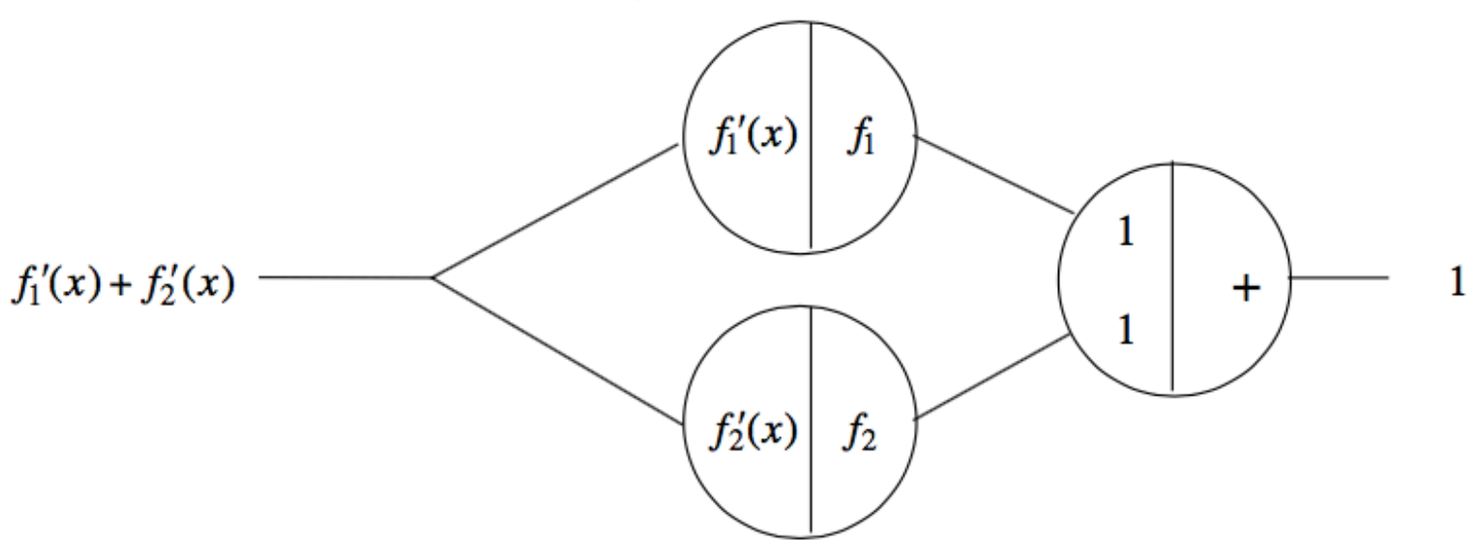
Backpropagation

- Base cases of proposition easily verified

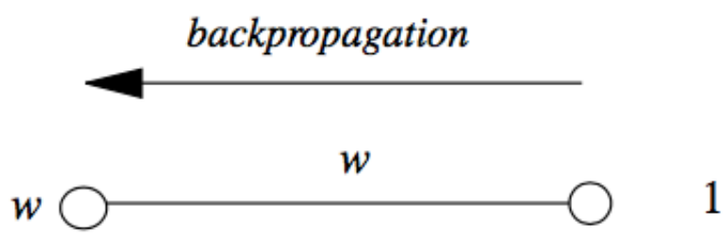
Derivative of f(g(x)) obtained



Derivative of f1(x)+f2(x) obtained

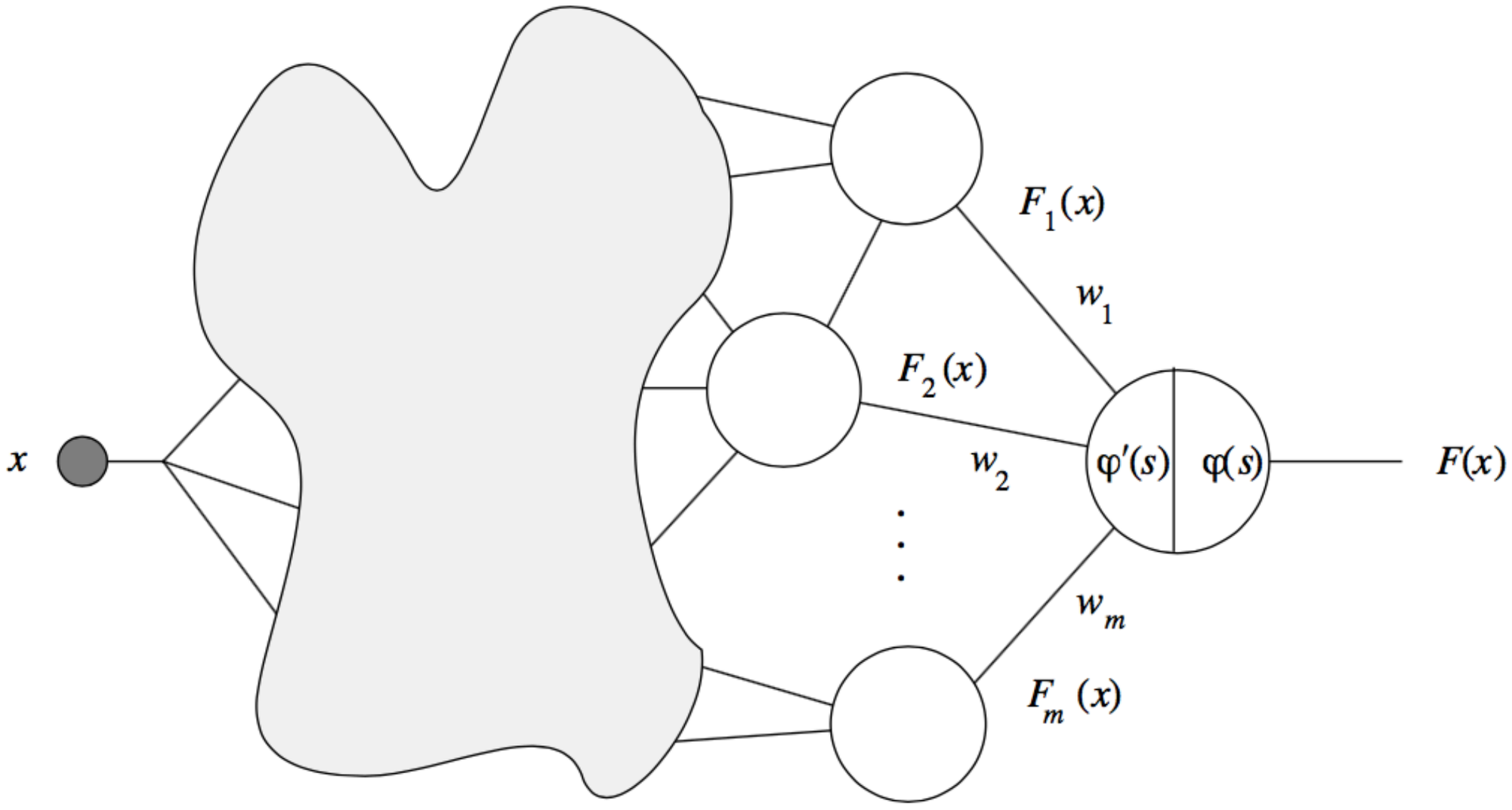


Derivative of wx obtained



Backpropagation

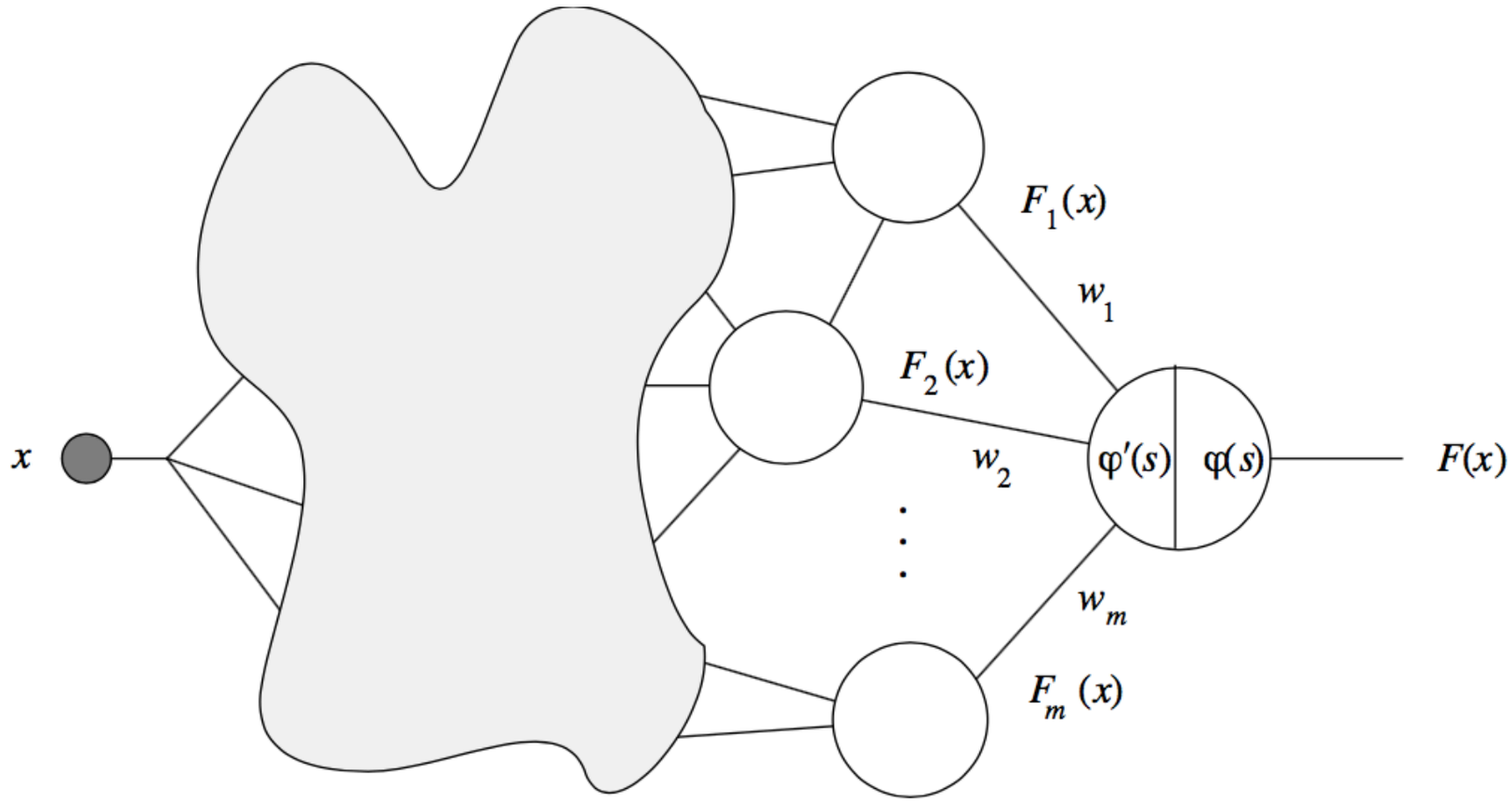
- Say true for all nets with n or less nodes



$$F(x) = \varphi(w_1 F_1(x) + w_2 F_2(x) + \cdots + w_m F_m(x))$$

Backpropagation

- Say true for all nets with n or less nodes



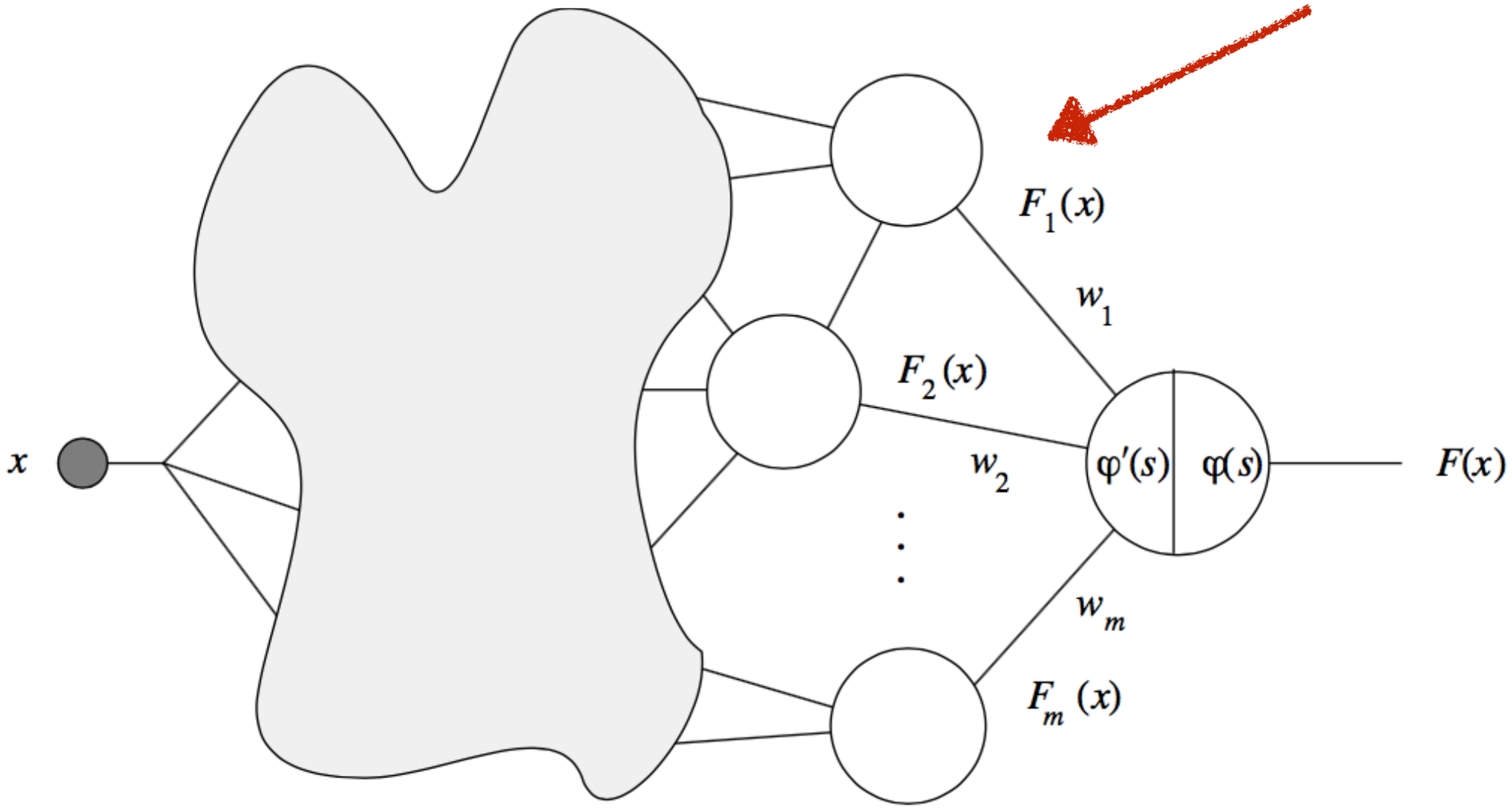
True derivative

$$F'(x) = \varphi'(s)(w_1 F_1'(x) + w_2 F_2'(x) + \cdots + w_m F_m'(x))$$

$$s = \varphi(w_1 F_1(x) + w_2 F_2(x) + \cdots + w_m F_m(x))$$

Backpropagation

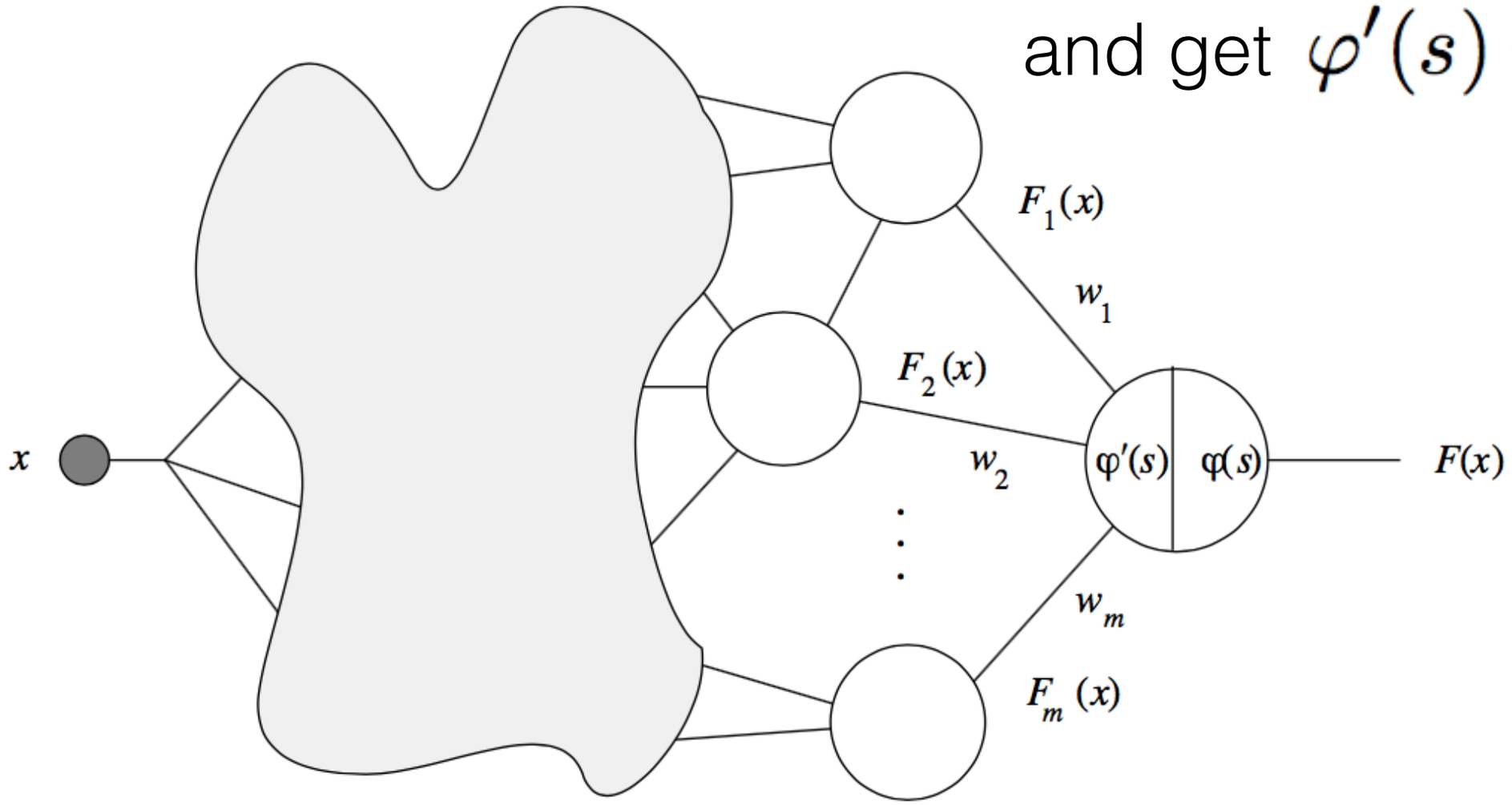
If we push 1 here, we get $F_1'(x)$



Backpropagation

Push $\varphi'(s)\, w_1$

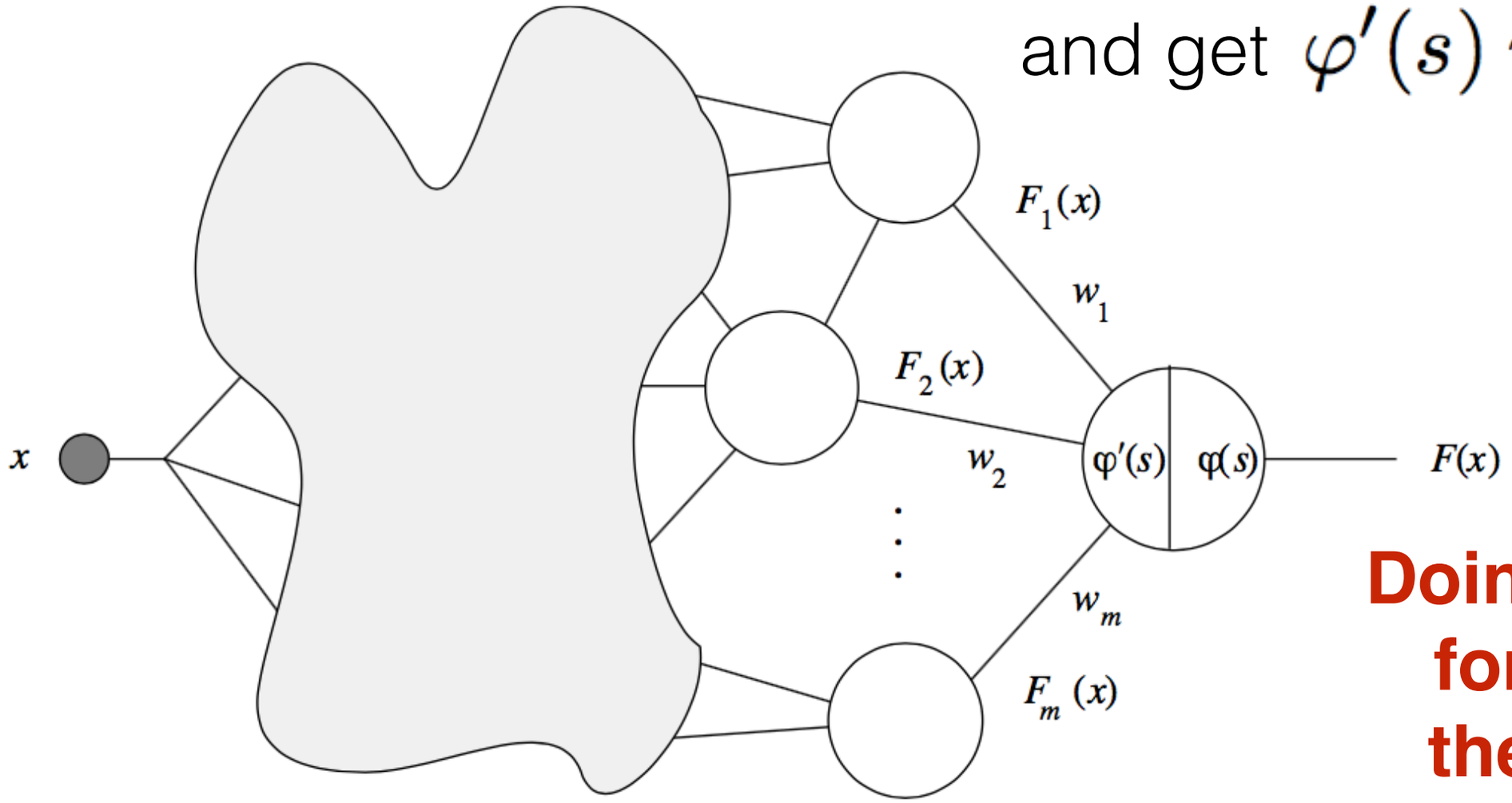
and get $\varphi'(s)\, w_1\, F_1'(x)$



Backpropagation

Push $\varphi'(s)\, w_1$

and get $\varphi'(s)\, w_1\, F_1'(x)$



Doing the same for all gives the gradient

$$F'(x) = \varphi'(s)(w_1 F_1'(x) + w_2 F_2'(x) + \cdots + w_m F_m'(x))$$

Backpropagation

- Since input (at a node) is constant on backpropagation, we have

$$\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j$$

$$\Delta w_{ij} = -\gamma o_i \delta_j$$

backpropagated error — available from backpropagation step

Training CNNs

- At any point the gradient is basically the product of the input and the back propagated error at the end point nodes resp.

$$\Delta w_{ij} = -\gamma o_i \delta_j$$

- This was for a scalar input, similar strategy for vector or matrix input