Analyzing Software using Deep Learning

Summarizing Programs with Convolutional Networks (Part 1)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- **Convolutional networks**
  - Motivation and basics
  - Properties
  - Pooling
- **Tree convolution for program classification**

  Based on ”Convolutional Neural Networks over Tree Structures for Programming Language Processing” by Mou et al., 2016

Historical Motivation: Neuroscience

Receptive fields of cats and monkeys

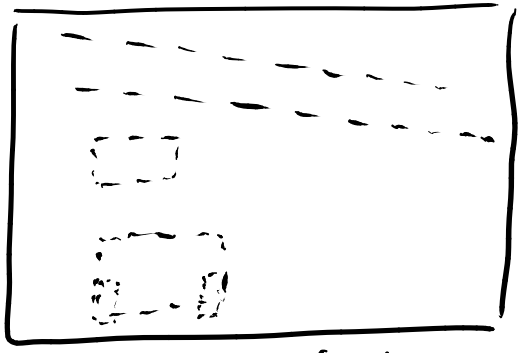
- Some neurons in visual cortex individually respond to small regions of the visual field
- Two visual cell types
  - Simple cells: Respond to straight edges with particular orientations
  - Complex cells: Sensitive to larger receptive field but insensitive to exact location of edges

Inspired work on neural network-based image recognition

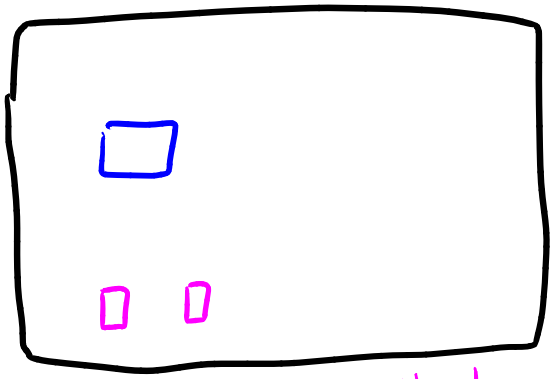
Intuition

Input data is hierarchically organized

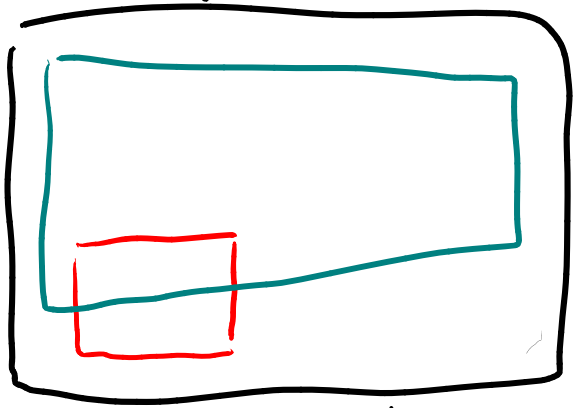
↳ E.g., image or source code



Primitive features: Oriented edges



Object parts: wheels, windows



Objects: car, house

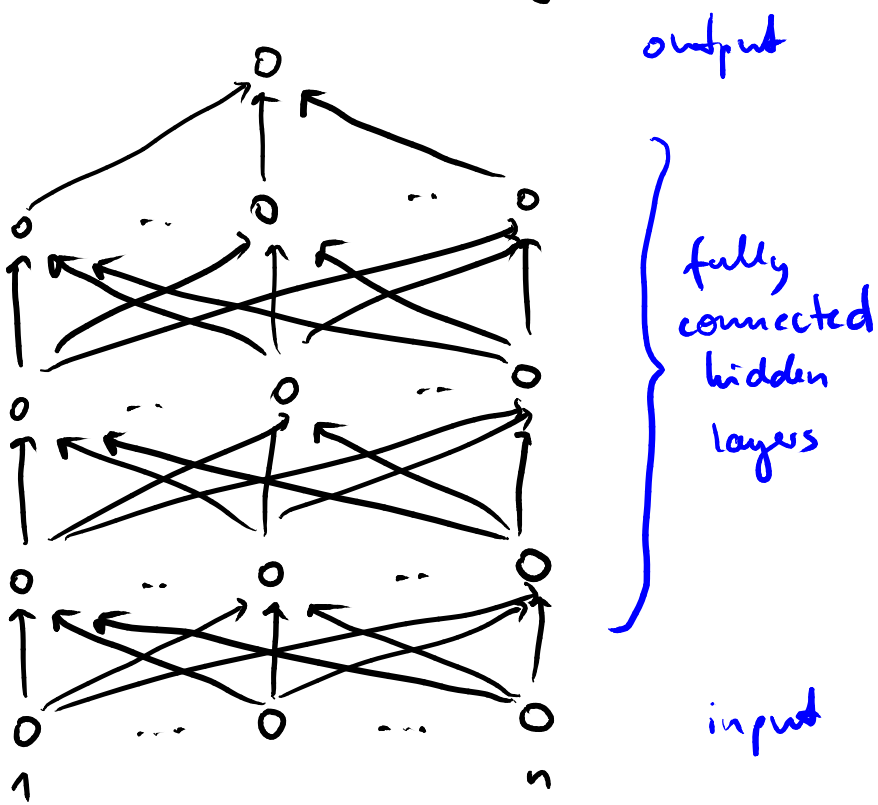
} Object recognition

Convolutional Networks

- Feedforward neural network architecture
- Connectivity pattern exploits hierarchical structure of input data
- Not a fully connected network
- Convolution function: Mathematical approximation of stimuli within receptive field
- Applications:
  - Image and video recognition
  - Natural language processing
  - Classification of programs

Complexity of Fully Connected Neural Network

Suppose:
- Input of length n
- Single output
- 3 hidden layers, fully connected



How many weight does the network have for $n = 32 \times 32 = 1024$ (e.g., small image)

$$n^2 + n^2 + n^2 + n$$

≈ 3.1 million weights

↳ Each stored in memory

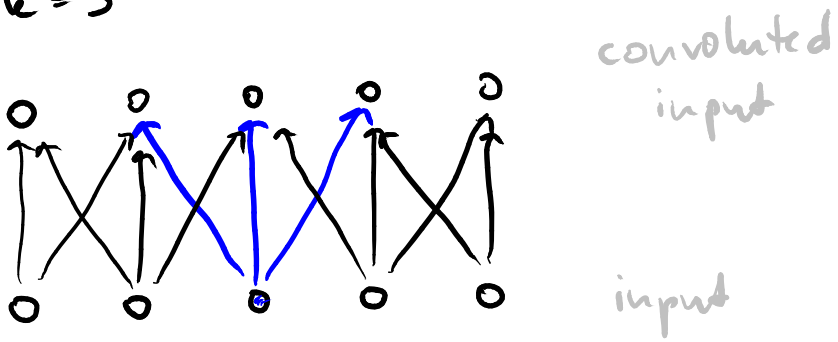
↳ Each optimized individually

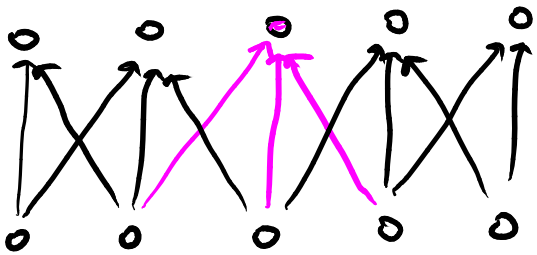
Reducing Complexity via Convolution

Instead of fully connected computation graph

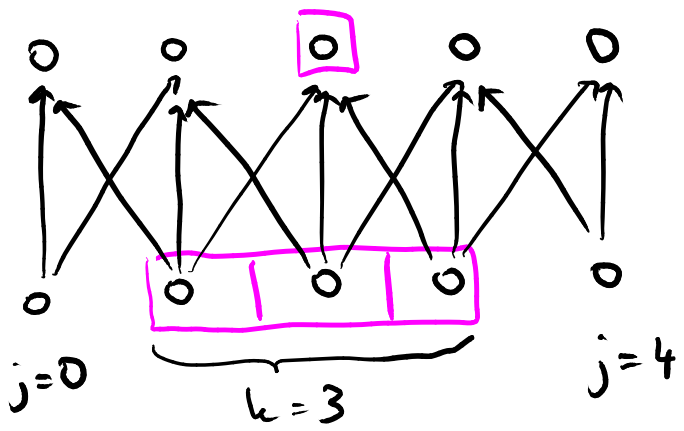
- Each input influences at most k neurons
- Each neuron in convoluted layer is activated by at most k neurons

$k = 3$





Kernel : Parameters for Convolution



Convolution S

Input I

Kernel K
(vector of length k)

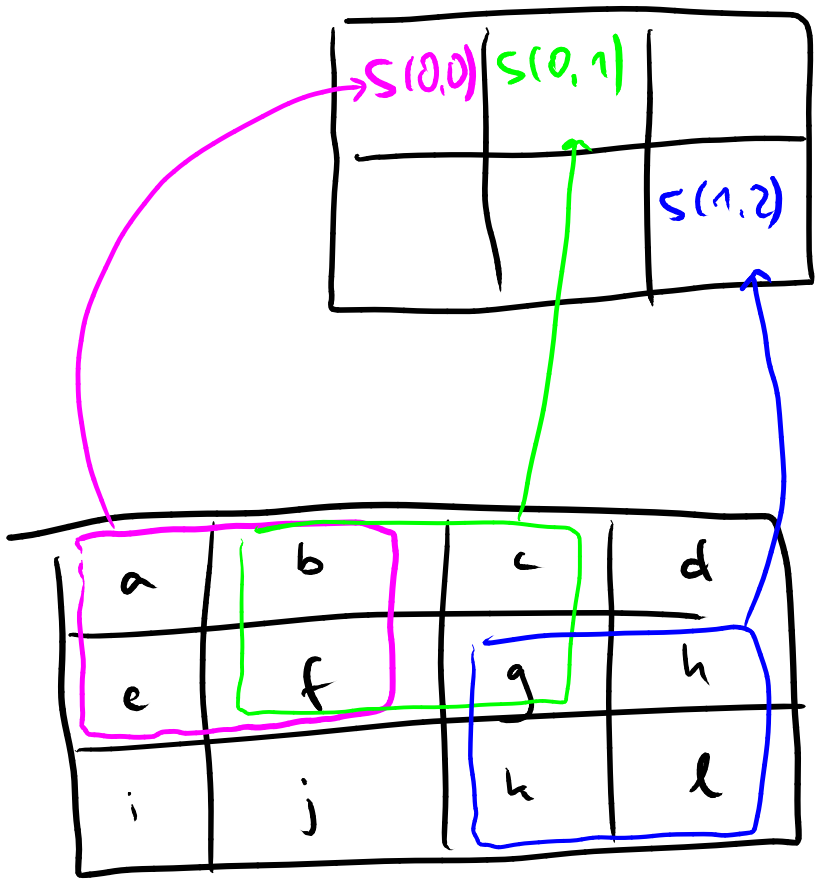
S is product of part of I and K
(We ignore boundaries here.)

Example: $K = [2, 5, 3]$ $\quad I = [3, 1, 2, 3, 4]$

$$s(2) = 1 \cdot 2 + 2 \cdot 5 + 3 \cdot 3$$

Two-Dimensional Scenario

↳ E.g., pixels of an image



Convolution S

Input I

Kernel K

w	x
y	z

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

$$S(0,0) = a \cdot w + b \cdot x + e \cdot y + f \cdot z$$

$$S(0,1) = b \cdot w + c \cdot x + f \cdot y + g \cdot z$$

$$S(1,2) = g \cdot w + h \cdot x + k \cdot y + l \cdot z$$

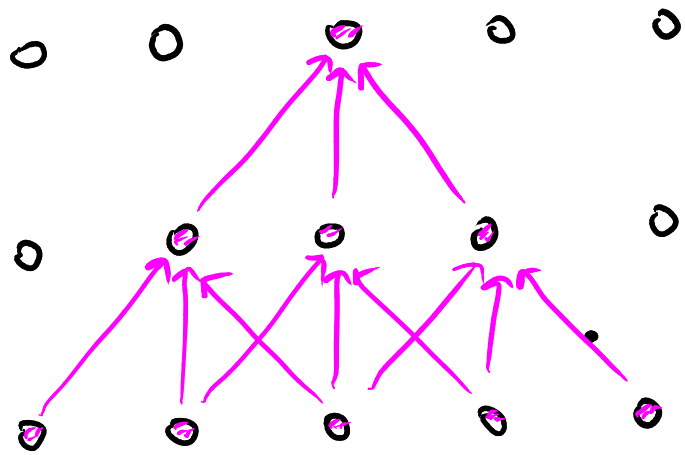
Properties

Three properties that help improve learning:

- Sparse interactions
- Parameter sharing
- Equivariant representations

Sparse Interactions

- Fewer connections than in fully connected network
  - → Fewer weights to store
  - → — " — to optimize
- But: In deep network (many layers), neurons in deeper layers may indirectly interact with many inputs



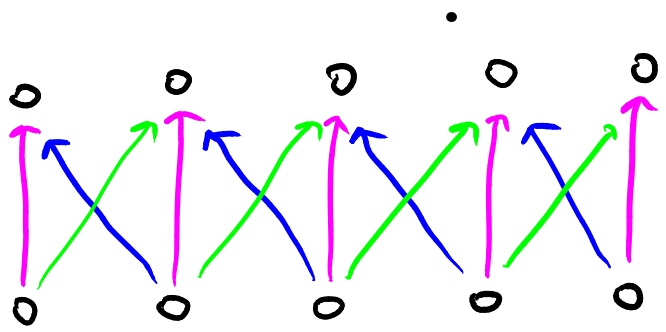
Parameter Sharing

Fully connected

- Parameters to optimize:
  Weights in weight matrix
- Each weight used for <u>one</u> connection

Convolutional

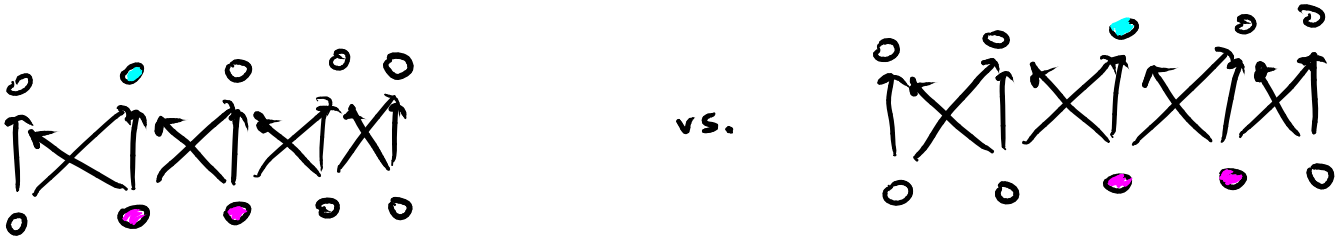
- Parameters to optimize:
  Weight in kernel matrix
- Same weights used for <u>many</u> parts of the input



connections share the <u>same</u> kernel parameters

Equivariant Representations

→ If input changes, output changes in <u>same</u> way no matter where the input is



→ Applications:

- Images: Representation stays same if objects are moved
- Source code: Statement with partic. property
  ↳ Can occur anywhere, its representation remains the same

Big Picture

Convolution : Typically used in combination with other steps

...

↑

Pooling

↑

Detection ----- Non-linear activation function

↑

Convolution

↑

Input

Pooling

- Form of downsampling
- Replaces output at certain location with summary of nearby outputs
- Intuition: Exact location of a ”feature” is less important than its presence and its location w.r.t. other features

Max Pooling

- Summarize region into maximum output within this region

- Example

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

$\longrightarrow$

6	8
3	4