Analyzing Software using Deep Learning

Lecture 4: Classifying Programs with Convolutional Networks

Prof. Dr. Michael Pradel

Software Lab, TU Darmstadt

Plan for Today

- **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

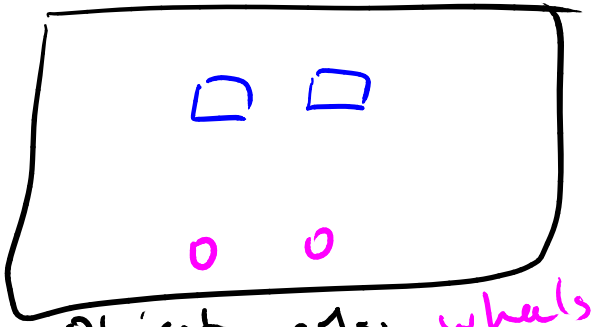
- Some neurons in visual cortex individually respond to small regions of the visual field
- Two visual cell types
  - Simple cells: Respond to straight edges with particular orientations
  - Complex cells: Sensitive to larger receptive field but insensitive to exact location of edges

Inspired work on neural network-based image recognition

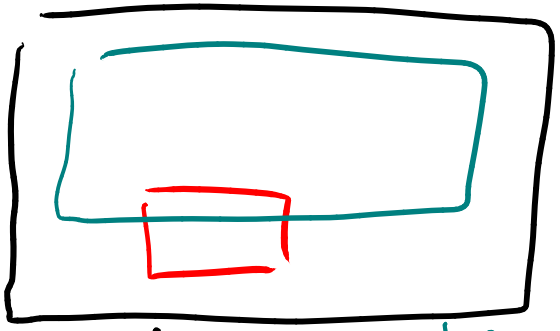
Intuition

Input data is hierarchically organized
↳ E.g. images or source code

Primitive features: Oriented edges



Object parts: wheels, windows



Objects: car house

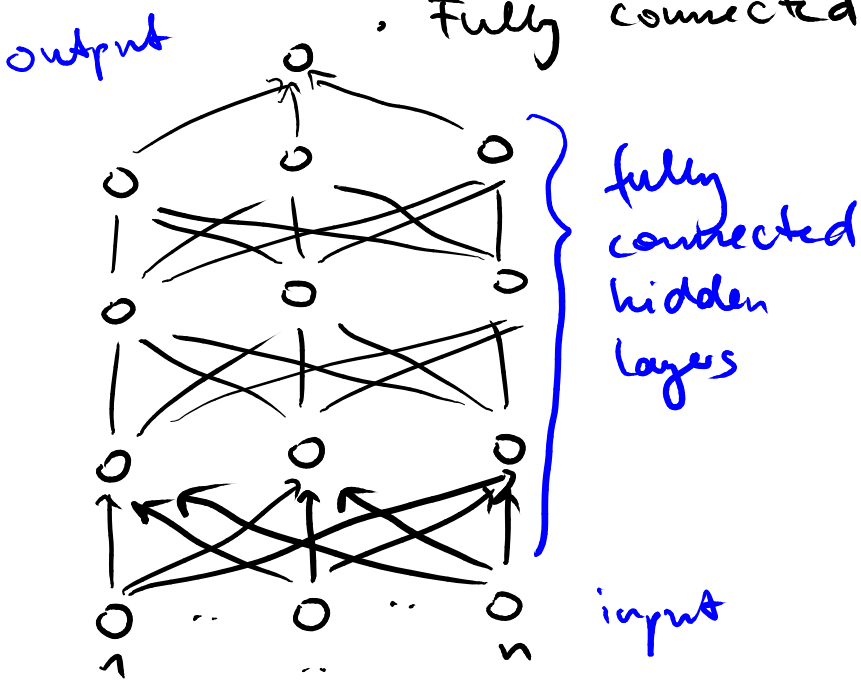
} Image recognition

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 Networks

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



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

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

$$= 3\,146\,752 \text{ weights}$$

$$= 1024^2 + 1024^2 + 1024^2 + 1024$$

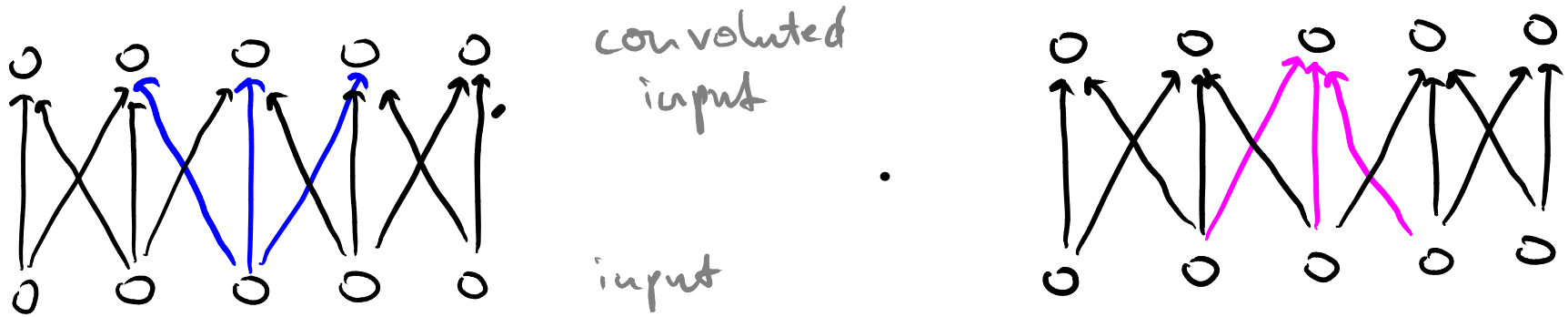
↳ Each stored in memory

↳ Each optimized individually

Reducing Complexity via Convolution

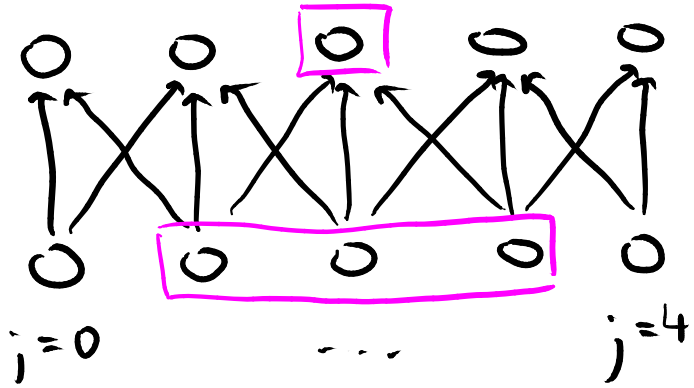
Instead of fully connected computation graph:

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



$k = 3$

Kernel: Parameters for Convolution



Convolution S

Input I

Kernel K

(vector of length 6)

S is product of part of I and K
(We ignore the 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

$S(0,0)$	$S(0,1)$	
		$S(1,2)$

Input I

a	b	c	d
e	f	g	h
i	j	k	l

Kernel K

w	x
y	z

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

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

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

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

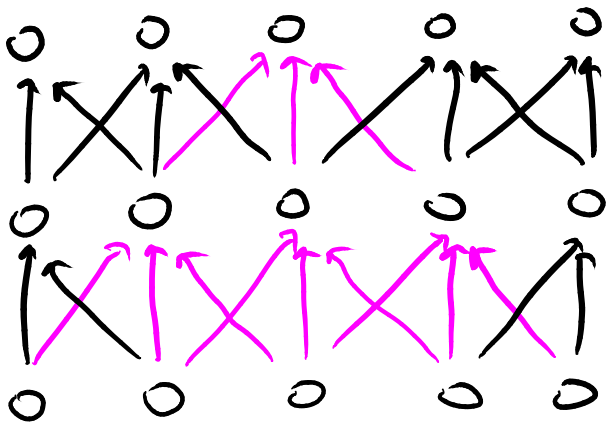
Properties

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
  - → Fewer weights to optimize
- But: In deep networks (= many layers), neurons in deeper layers may indirectly interact with many inputs



Parameter Sharing

Fully connected

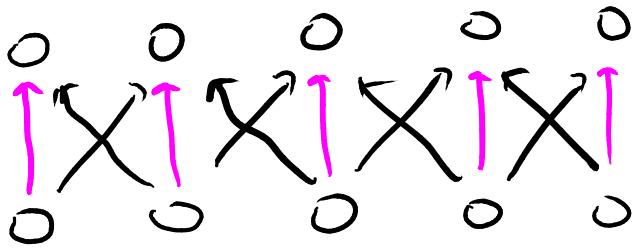
- Parameters to optimize: Weights in weight matrix
- Each weight used for exactly one connection

Convolutional

- Parameters to optimize: Weights in kernel matrix
- Same kernel used for many parts of input

→ Does not reduce time of forward propagation

→ But: reduces storage requirements



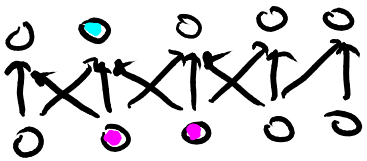
→ connections all share the same kernel parameter

Equivariant Representations

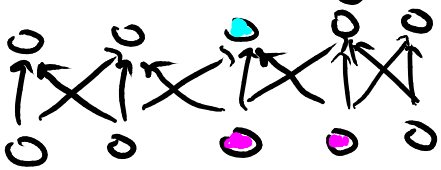
Function $f(x)$ is equivariant to function g if

$$f(g(x)) = g(f(x))$$

→ If input changes, then output changes in same way



vs.



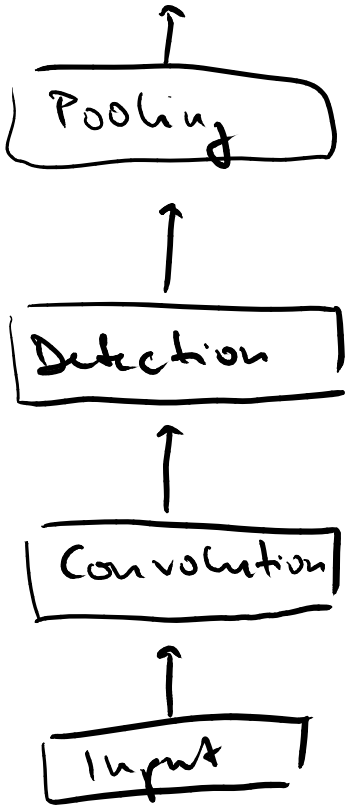
Applications:

- Images: If object is moved, its representation will move by same amount
- Source code: If a statement with a particular property occurs anywhere, its representation remains the same

Big Picture

Convolution: Typically used in combination with other steps

(next layer)



Non-linear activation function (e.g., rectified linear unit)

See before

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 the maximum output within this region
- Example :

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

$\xrightarrow{\text{Max pooling}}$

6	8
3	4

Plan for Today

- **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

Convolution of Programs

- Convolution exploits hierarchical structure of input data
- Programs are hierarchical data
  - Coarse-grained level:
    Projects – Packages – Classes – Methods
  - Code level:
    Abstract syntax trees
- Idea: Use convolution to identify important features in code

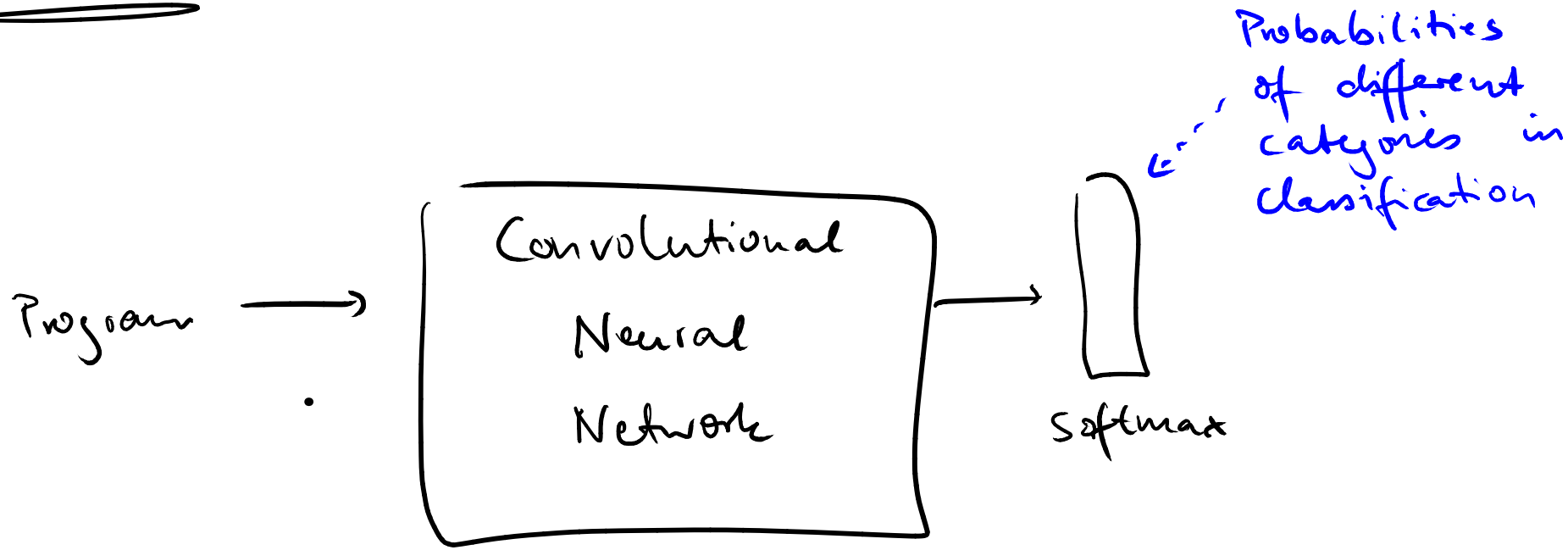
Applications

Various possible applications

Today: Classification

- Various possible ways to classify programs
  - Project where code comes from
  - Author who wrote the code
  - Instances of bug patterns
  - Malware vs. benign code
- Here: Identify functionality of code

Overview

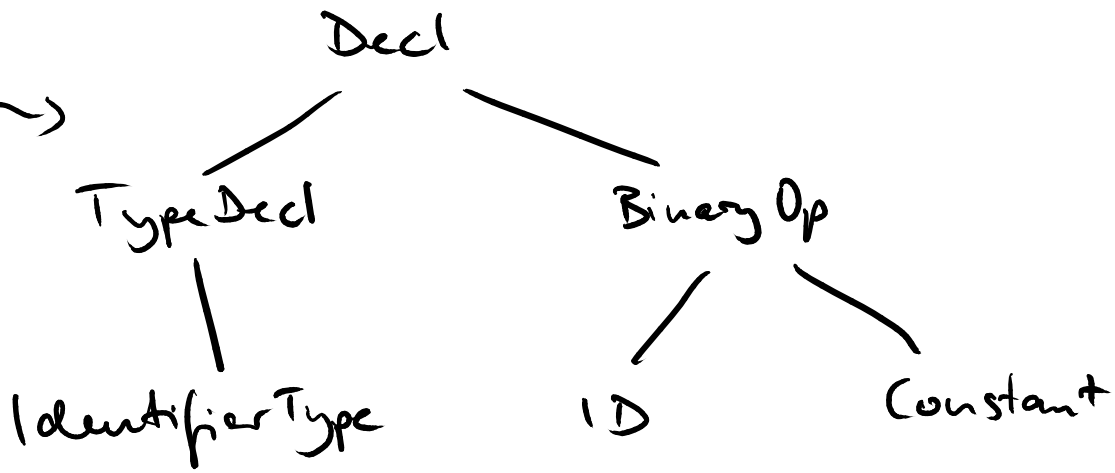

Program
Convolutional Neural Network
Softmax
Probabilities of different categories in classification

Tree Representation

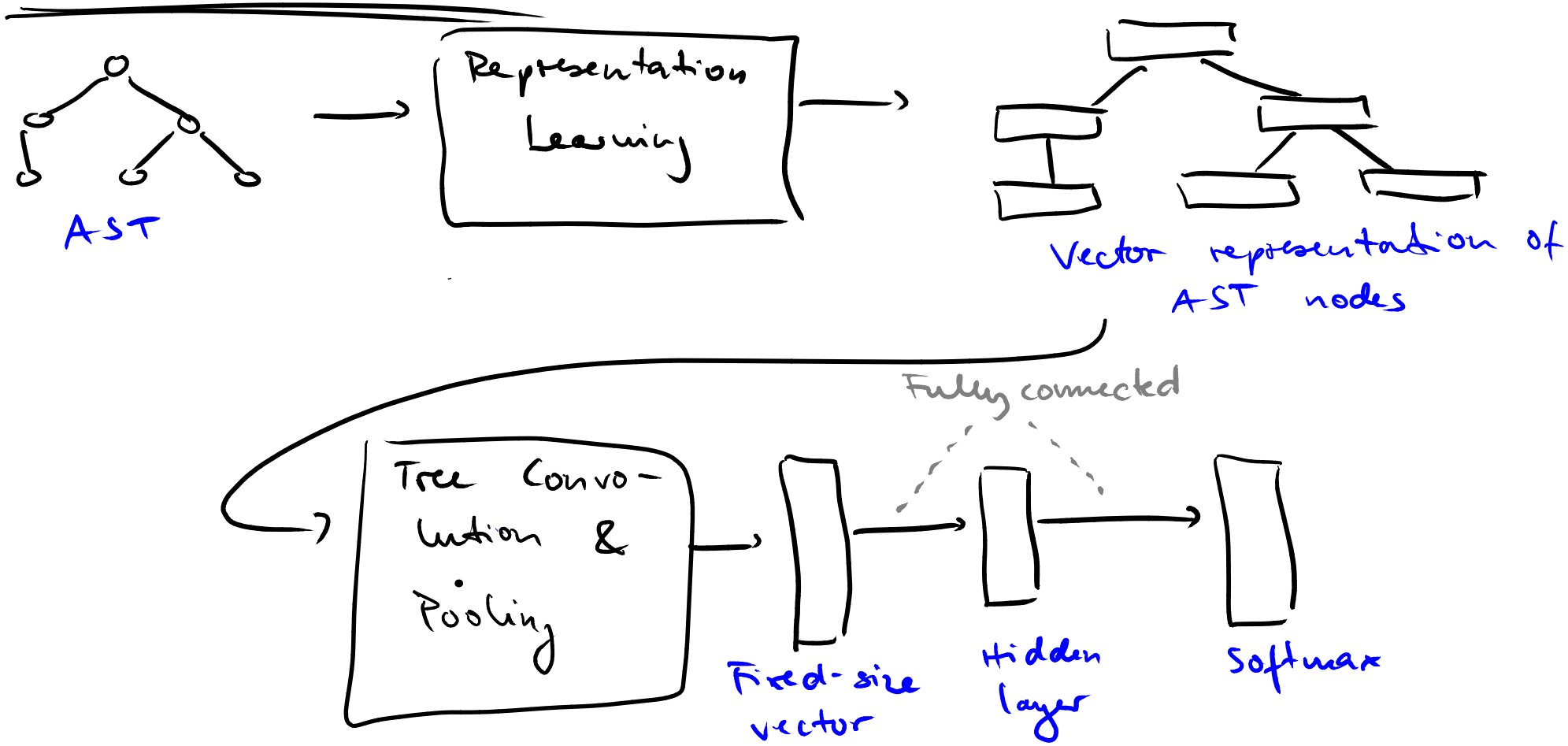
- AST where each node has at most two children ("continuous binary tree")

Example:

int a = b + 3; ⇝

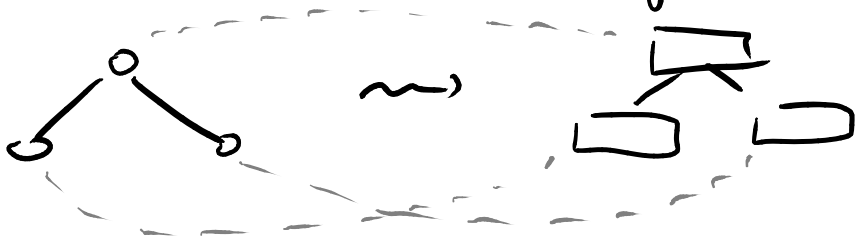


Detailed Overview



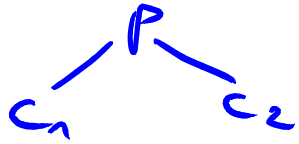
Representation Learning

- Represent AST node as fixed-size vector ("embedding")



- Similar nodes should have similar vectors
  - ↳ E.g., While ≈ For but While ≉ Constant
- Learned in separate pre-training step

Representation of Nodes



- Representation of p:

$$p_{rep} = W_{comb1} \cdot vec(p) + W_{comb2} \cdot childrenVec(p)$$

→ Goal: Parents encode information about children

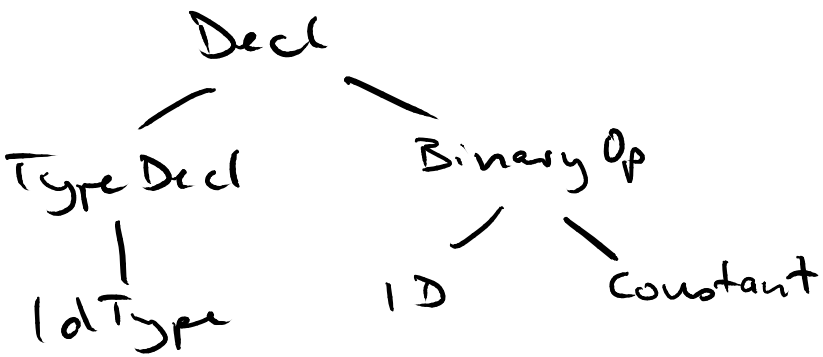
$$vec(p) \simeq childrenVec(p)$$

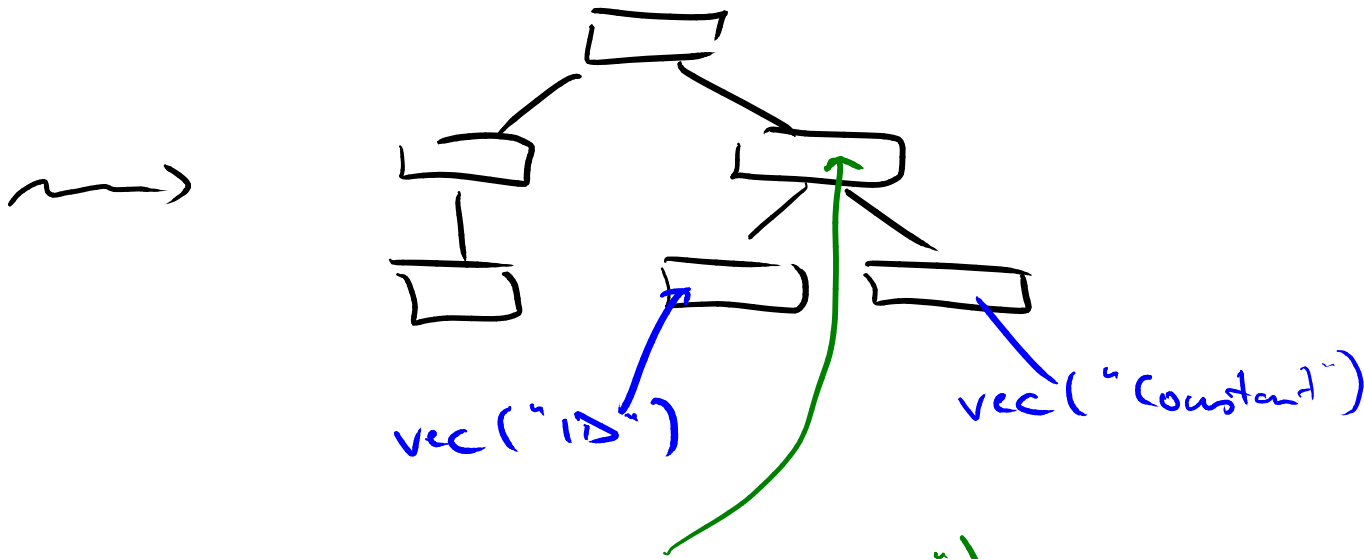
$$childrenVec(p) = \tanh\left(l_{left} \cdot W_{left}^{rep} \cdot vec(c_1) + l_{right} \cdot W_{right}^{rep} \cdot vec(c_2) + b_{rep} \right)$$

where $l_i = \dfrac{\#\text{leaves nodes under } c_i}{\#\text{leaves nodes under } p}$

↳ Minimize distance between $vec(p)$ and $childrenVec(p)$

Example





$$\cdot \quad W_{comb1} \cdot \mathrm{vec}(\text{"BinaryOp"})$$

$$+ \quad W_{comb2} \cdot \tanh\left(\tfrac{1}{2} \cdot W_{left}^{rep} \cdot \mathrm{vec}(\text{"ID"}) + \tfrac{1}{2} \cdot W_{right}^{rep} \cdot \mathrm{vec}(\text{"Constant"}) + b_{rep}\right)$$

Pre-learning

"Main" learning