Analyzing Software using Deep Learning

Introduction

Prof. Dr. Michael Pradel

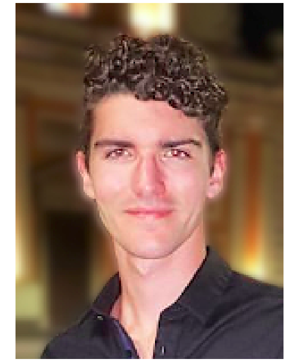
Software Lab, University of Stuttgart

Summer 2022

About Me: Michael Pradel

- **Since 9/2019: Full Professor at University of Stuttgart**
- **Before**
  - Studies at TU Dresden, ECP (Paris), and EPFL (Lausanne)
  - PhD at ETH Zurich, Switzerland
  - Postdoctoral researcher at UC Berkeley, USA
  - Assistant Professor at TU Darmstadt
  - Sabbatical at Facebook, Menlo Park, USA

About the Software Lab

- **My research group since 2014**
- **Focus: Tools and techniques for building reliable, efficient, and secure software**
  - Program testing and analysis
  - Machine learning, security
- **Thesis and job opportunities**

Overview

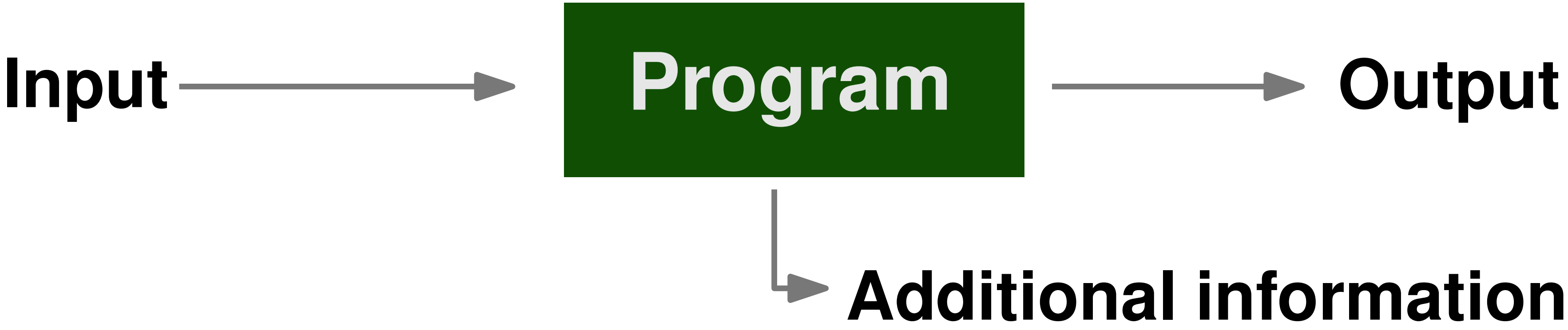
- **Motivation**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization**
  - Lectures and final exam
  - Course project
- **Basics**
  - Program analysis
  - Deep learning

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities



What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

Why Do We Need It?

Basis for various tools that make developers productive

- Compilers
- Bug finding tools
- Performance profilers
- Code completion
- Automated testing
- Code summarization/documentation

Traditional Approaches

- **Analysis has built-in knowledge about the problem to solve**
- **Significant human effort to create a program analysis**
  - Conceptual challenges
  - Implementation effort
- **Analyze a single program at a time**

Learning from Existing Data

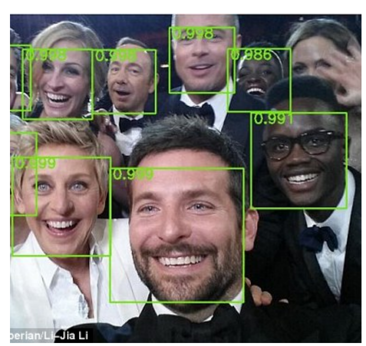
- **Huge amounts of existing code ("big code")**
- **Programs are regular and repetitive**
- **Machine learning: Extract knowledge and apply in new contexts**
- **E.g., learn how to ..**
  - .. complete partial code
  - .. use an API
  - .. find and fix programming errors
  - .. create inputs for testing

Deep Learning

Class of machine learning algorithms

- Neural network architectures
- ”Deep” = multiple layers
- Features and representation of inputs are extracted automatically

Revolutionizes entire areas



AlphaGo

This Course

Intersection of program analysis and deep learning

- Some of the basics:
  E.g., program representations, neural network architectures
- State of the art research results:
  Based on recent research papers
- Hands-on experience:
  Coding project

Not This Course

What this course is not about

- Detailed coverage of program analysis
- Detailed coverage of machine learning
- Programming tutorial for some ML library

Check out related courses

- E.g., ”Program Analysis” (winter semester)

Overview

- **Motivation**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization** ←
  - Lectures and final exam
  - Course project
- **Basics**
  - Program analysis
  - Deep learning

Organization

- **Until May 17: Lectures**
- **From May 17: Course project**
- **End of semester: Final exam**

Organization

Grading:

- **Until May 17: Lectures**
- **From May 17: Course project** → **50%**
- **End of semester: Final exam** → **50%**

Lectures

- **Nine lectures**
- **Mondays (9:45am) and Tuesdays (2:00pm)**
  - Not all slots are used: Check the schedule at https://software-lab.org/teaching/summer2022/asdl/
- **Reading material:**
  Recent research papers

Course Project

- **Individual, independent project**
- **Same task for everybody**
- **Implement and evaluate a neural software analysis that detects bugs**
- **Based on existing tools**
  - PyTorch library for machine learning
  - Python as implementation and target language
- **More details on May 17**

Final Exam

- Content of lectures and reading material
- Open book
- One hour
- Will test your understanding, not your memory
- Alternative: Combined oral exam (“Vertiefungsprüfung”)

Ilias

Platform for discussions and sharing additional material

- Please register for the course
- Use the forum for all questions related to the course
- Messages sent to all students go via Ilias
- See link on https://software-lab.org/teaching/summer2022/asdl/

Plan for Today

- **Introduction**
  - What the course is about
  - Why it is interesting
  - How it can help you
- **Organization**
  - Lectures and final exam
  - Course project
- **Basics** ←
  - Program analysis
  - Deep learning

Program Representations

Many ways to represent (parts of) a program

- Sequence of characters
- Sequence of tokens
- Abstract syntax tree
- Control flow graph
- Data dependence graph
- Call graph
- etc.

Program Representations

Many ways to represent (parts of) a program

- Sequence of characters
- Sequence of tokens
- Abstract syntax tree
- Control flow graph
- Data dependence graph
- Call graph
- etc.

Tokens

Tokenizer (or lexer)

- Part of compiler
- Splits sequence of characters into subsequences called tokens

E.g., for Java, six kinds of tokens:

- Identifiers, e.g., `MyClass`
- Keywords, e.g., `if`
- Separators, e.g., `.` or `{`
- Operators, e.g., `*` or `++`
- Literals, e.g., `23` or `"hi"`
- Comments, e.g., `/* bla */`

Token: Example

```
if (flag == true) {
    name = "Joe";
}
```

Keyword

Separators

Identifiers

Operators

Literals

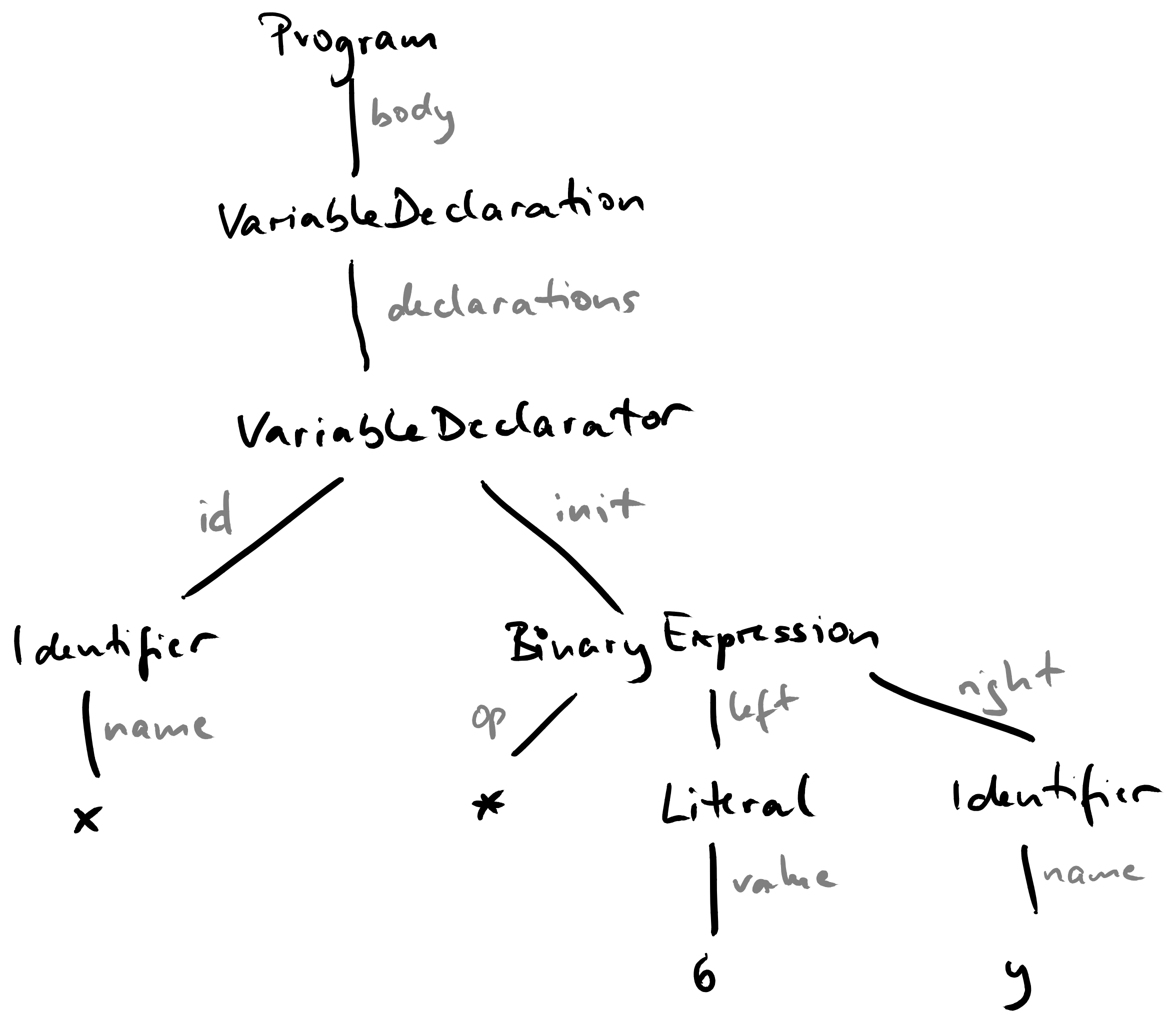
Abstract Syntax Tree

- Tree representation of source code
- ”Abstract” because some details of syntax omitted
  - E.g., { in Java
- Nodes: Construct in source code
- Edges: Parent-child relationship
- Check out this page for obtaining ASTs of various languages: https://astexplorer.net/

Abstract Syntax Tree: Example

Example: JavaScript

var x = 6*y;

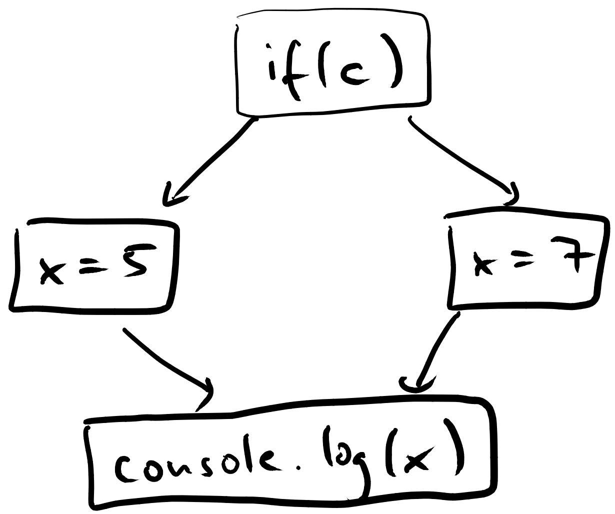


Control Flow Graph

- **Models flow of control through a program**
- **Graph (N, E) with**
  - Nodes N: Basic blocks = Sequence of operations executed together
  - Edges E: Possible transfers of control
- **Typically on the method-level**

Control Flow Graph: Example

```
if (c) {
    x = 5
} else {
    x = 7
}
console.log(x)
```

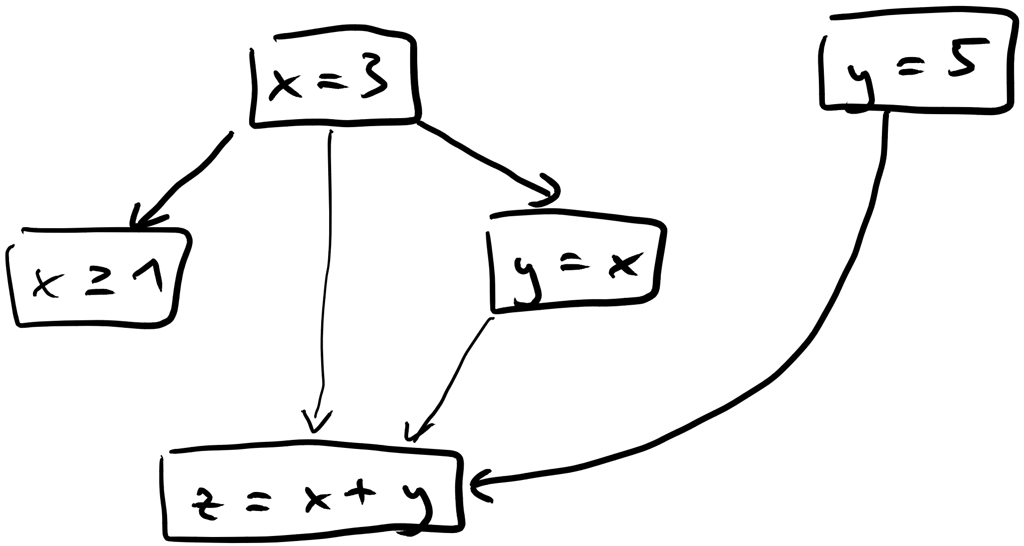


Data Dependence Graph

- **Models flow of data from “definition” to “use”**
- **Graph** (N, E) **with**
  - Nodes N: Operations that define and/or use data
  - Edges E: Possible definition-use relationships
    - Edge $e = (n_1, n_2)$ means n_2 may use data defined at n_1

Data Dependence Graph: Example

```
x = 3
y = 5
if (x ≥ 1) {
    y = x
}
z = x + y
```



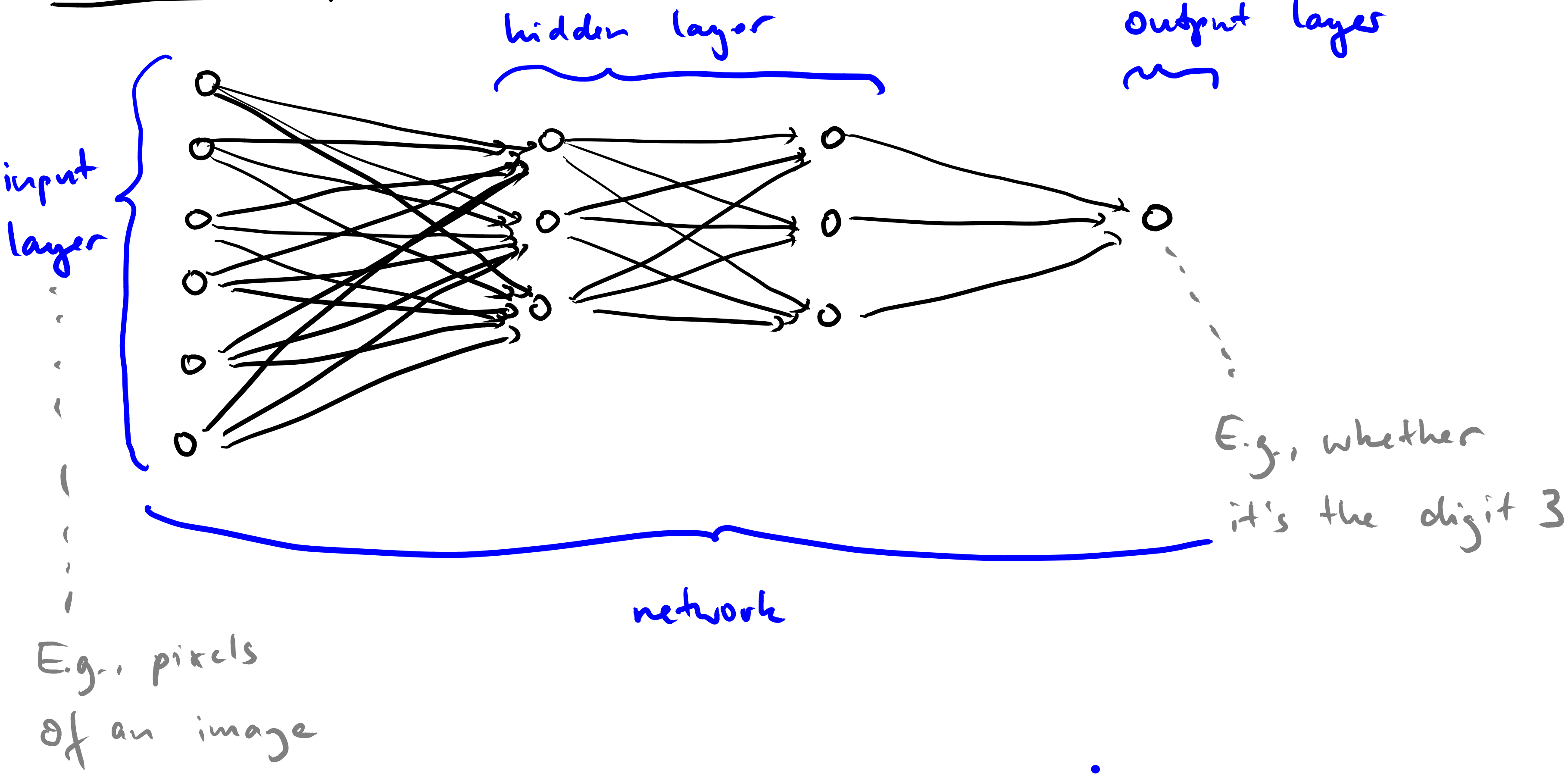
Deep Learning: Example

Example: Handwriting recognition

- Goal: Recognize digits 0..9
- Easy for a human but challenging for a computer
- Idea: Learn from a large number of training examples
- Deep learning: $> 99\%$ accuracy

Following slides based on Chapter 1 of neuralnetworksanddeeplearning.com

Network of neurons

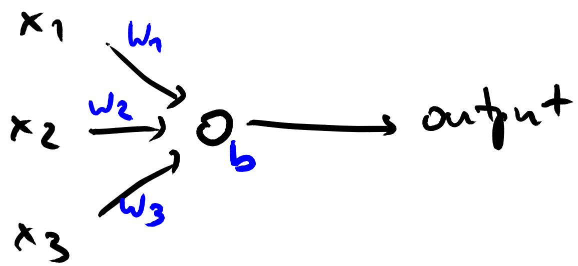


Perceptrons

↳ Most basic kind of neuron

↳ Binary inputs

↳ Binary output



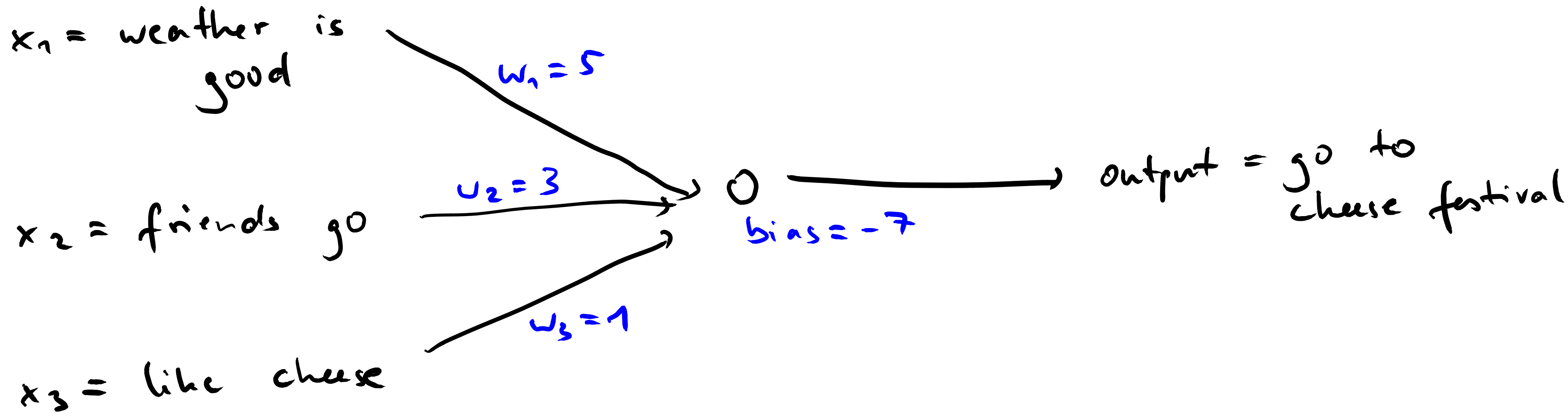
w … weights

b … bias

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j \cdot x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j \cdot x_j > \text{threshold} \end{cases}$$

$$= \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Example



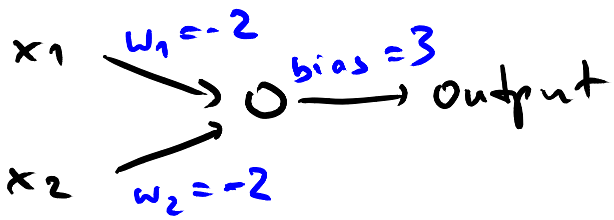
Assume: $x_1 = 1, \; x_2 = 1, \; x_3 = 0$

$$w \cdot x = 5 \cdot 1 + 3 \cdot 1 + 0 \cdot 1 = 8$$

$$\text{output} = \begin{cases} 0 & \text{if } 8 - 7 \leq 0 \\ 1 & \text{if } 8 - 7 > 0 \end{cases} \longrightarrow \text{Go to festival}$$

Computing Logical Functions

NAND gate



x_1	x_2	output
0	0	1
0	1	1
1	0	1
1	1	0

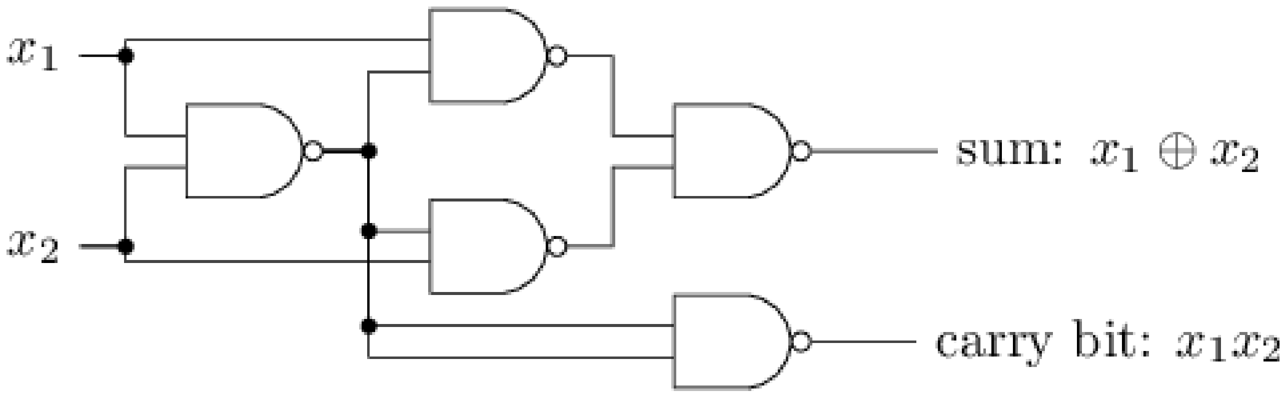
because $0+3>0$

Universal Computation

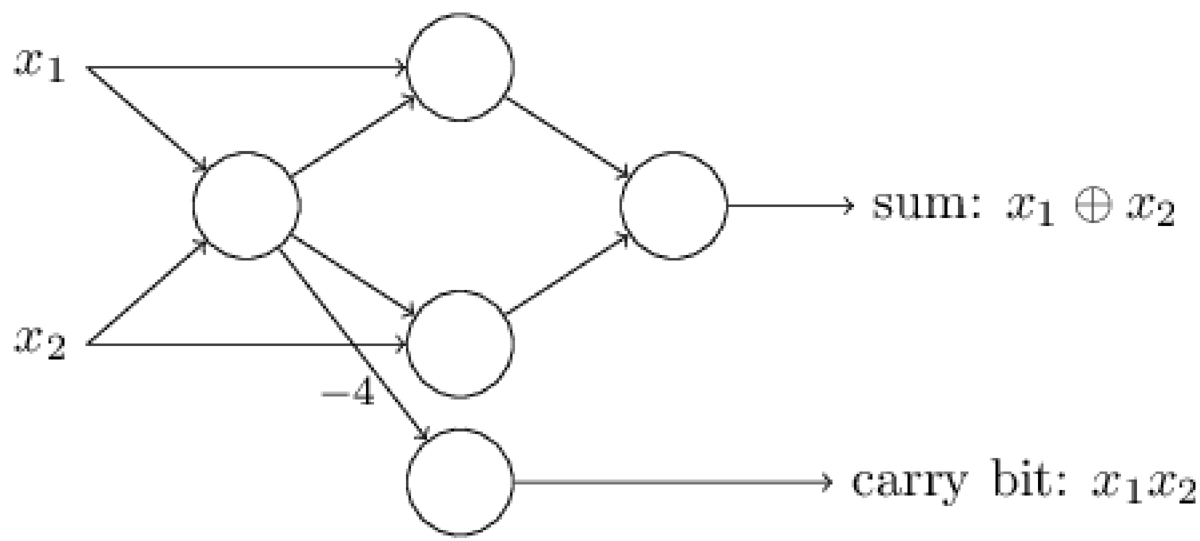
- **Networks of NAND perceptrons can simulate every circuit containing only NAND gates**
- **Can express arbitrary computations!**

Example: Adding Two Bits

NAND gate:



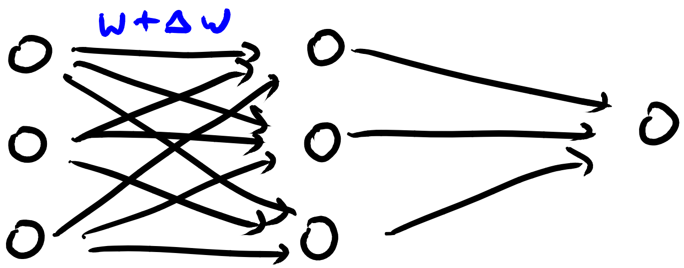
Network of perceptrons:



Challenge: Set Weights and Biases

- More complex networks can perform arbitrary computations
- How to decide on the weights and biases?
- Option 1: Hand-tune them
  $\rightarrow$ Infeasible for complex networks
- Option 2: Learn them
  $\rightarrow$ Key idea behind machine learning with neural networks

Making learning possible

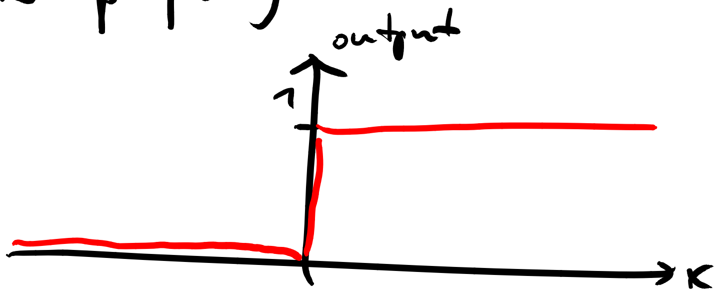


output + Δoutput

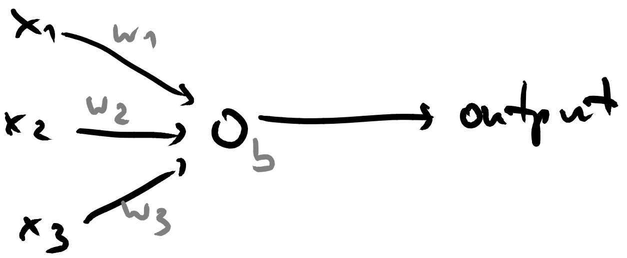
Want: Small change of weights & biases causes small change of output

Problem: Perceptron doesn't provide this property

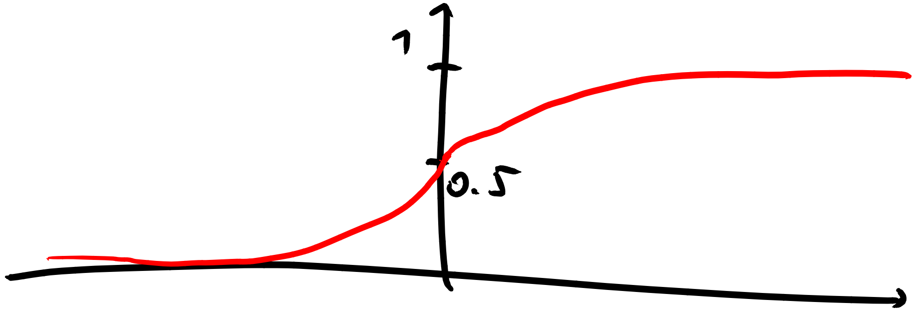
$$\text{output} = \text{step}(w \cdot x + b)$$



Sigmoid neuron



arbitrary values in [0, 1]



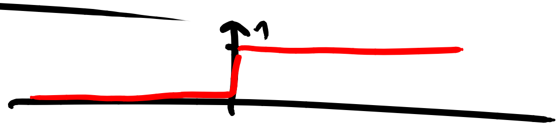
$$\text{output} = \sigma(w \cdot x + b)$$

sigmoid fct.: $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1 + \exp\left(-\left(\sum_j w_j \cdot x_j + b\right)\right)}$

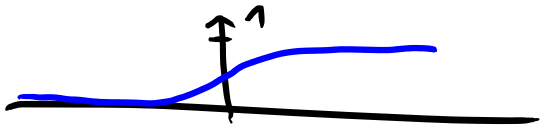
→ Enables learning: Small change causes small change

Activation functions

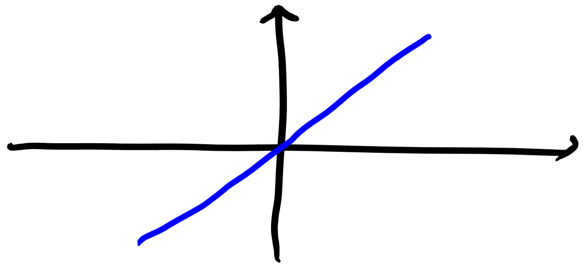
step function



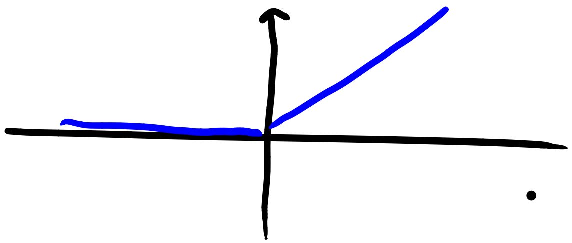
sigmoid fct./
logistic fct.



identity fct.



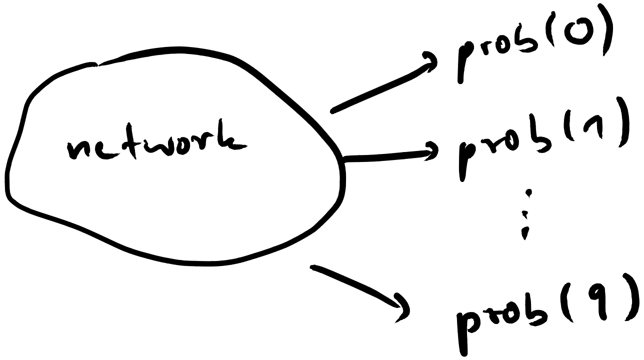
rectified linear unit ("relu")



Learning: Cost Function

↳ Feedback on how good output is for given input

Example:



If digit is known to be 6, want output:

$$y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)$$

Actual output may be:

$$a = (0, 0, 0, 0.2, 0, 0, 0.7, 0.1, 0, 0)$$

$$C = \frac{1}{n} \cdot \sum_x \| y(x) - a \|^2$$

... quadratic cost fct. or mean squared error

n … nb. of training inputs

Quiz: Cost Function

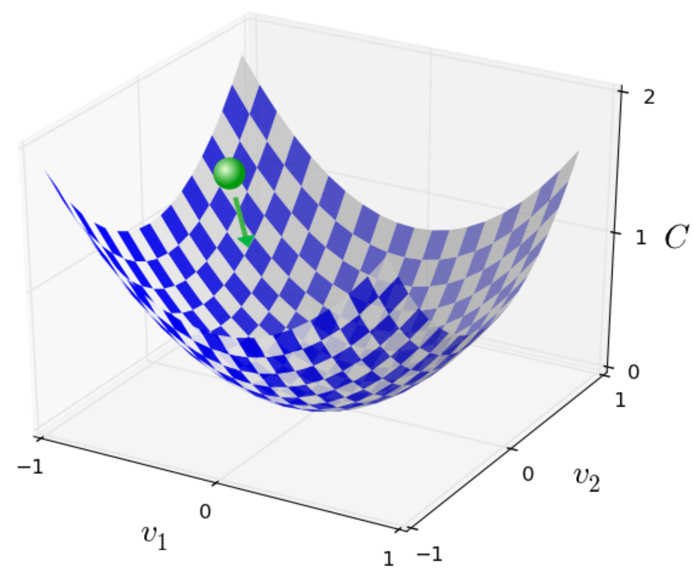
- **Recognition of hand-written digits**
- **Only digits 0, 1, and 2**
- **Training examples:**

Example	Desired	Actual
1	$(0, 1, 0)^T$	$(0.5, 0.5, 0)^T$
2	$(1, 0, 0)^T$	$(1, 0, 0)^T$

- **What is the value of the cost function?**

Goal: Minimize Cost Function

- **Goal of learning: Find weights and biases that minimize the cost function**
- **Approach: Gradient descent**
  - Compute gradient of C: Vector of partial derivatives
  - "Move" closer toward minimum step-by-step
  - Learning rate determines step size



$$C = \frac{1}{n} \cdot \sum_x \| y(x) - a \|^2$$

$$\|(x, y, z)\| = \sqrt{x^2 + y^2 + z^2}$$

$$= \frac{1}{2} \cdot \left(\|(-0.5, 0.5, 0)\|^2 + \|(0, 0, 0)\|^2 \right)$$

$$= \frac{1}{2} \cdot (0.5 + 0) = 0.25$$

Training Examples

- **Effort of computing gradient depends on number of examples**
- **Stochastic gradient descent**
  - Use small sample of all examples
  - Compute estimate of true gradient
- **Epochs and mini-batches**
  - Split training examples into k mini-batches
  - Train network with each mini-batch
  - Epoch: Each mini-batch used exactly once