

# **Analyzing Software using Deep Learning**

## **Introduction (Part 3)**

**Prof. Dr. Michael Pradel**

**Software Lab, University of Stuttgart**

**Summer 2020**

# 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

# 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
- Identifier
- Operators
- Literals

# Abstract Syntax Tree

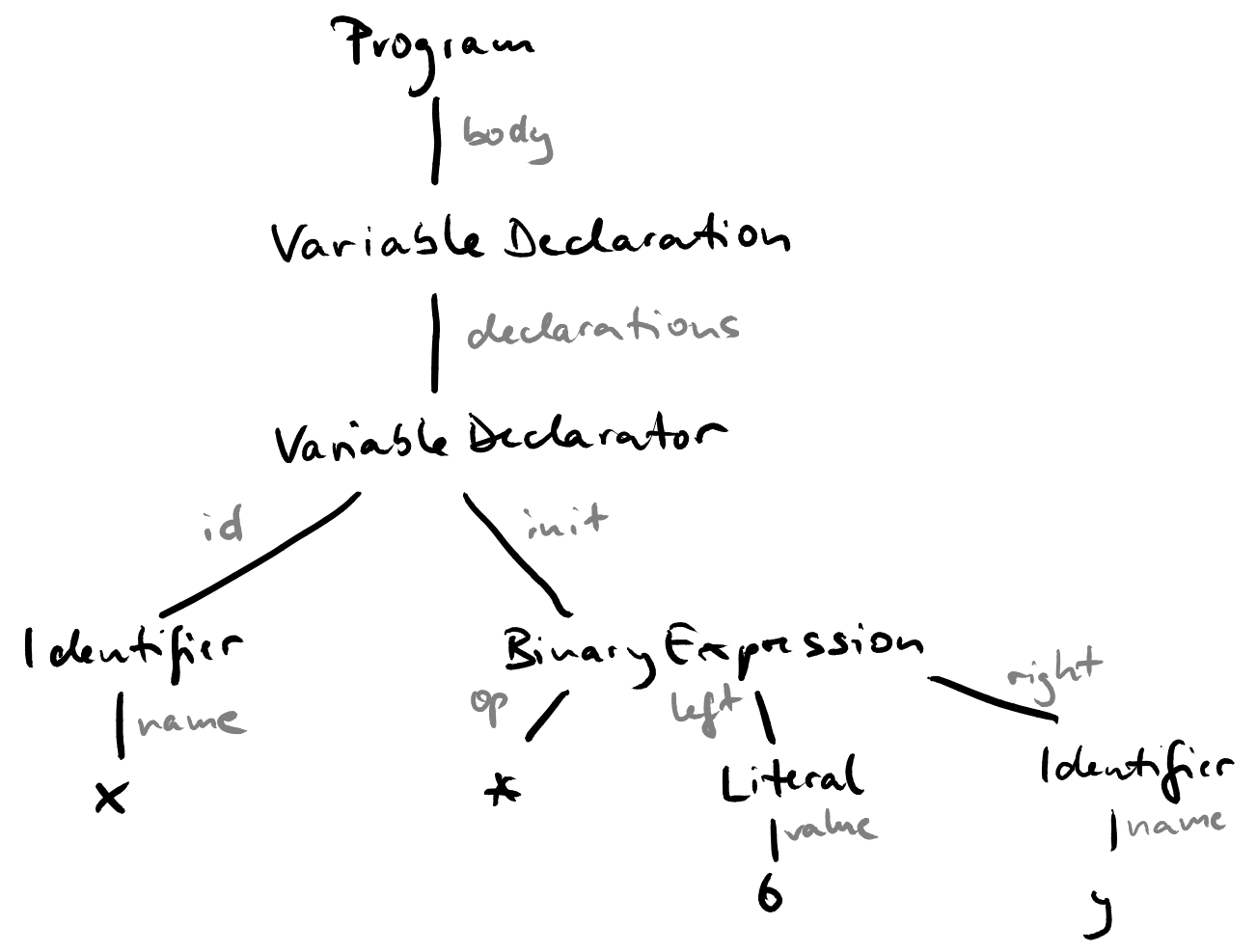
---

- **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 **Esprima** for obtaining ASTs of Javascript:  
<http://esprima.org/demo>

# Abstract Syntax Tree: Example

Example: JavaScript

var x = 6 \* y;



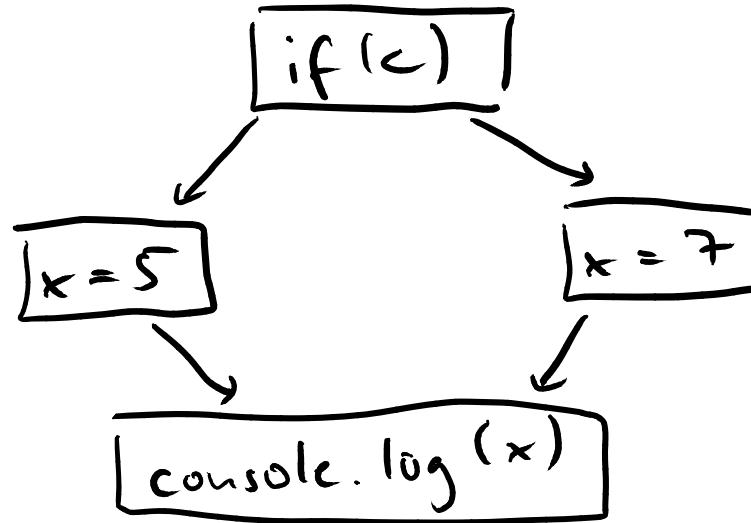
# Control Flow Graph

---

- 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)
```



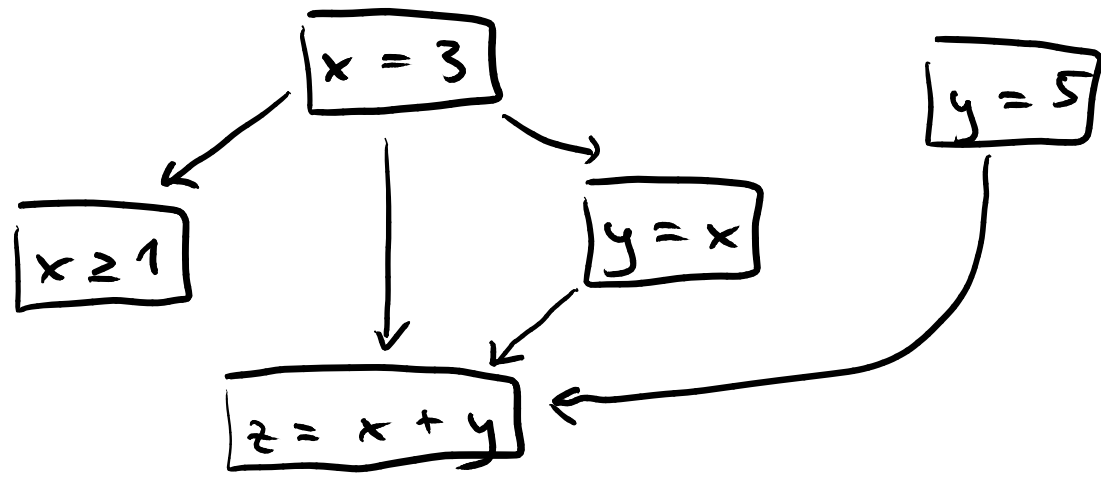
# Data Dependence Graph

---

- 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

---

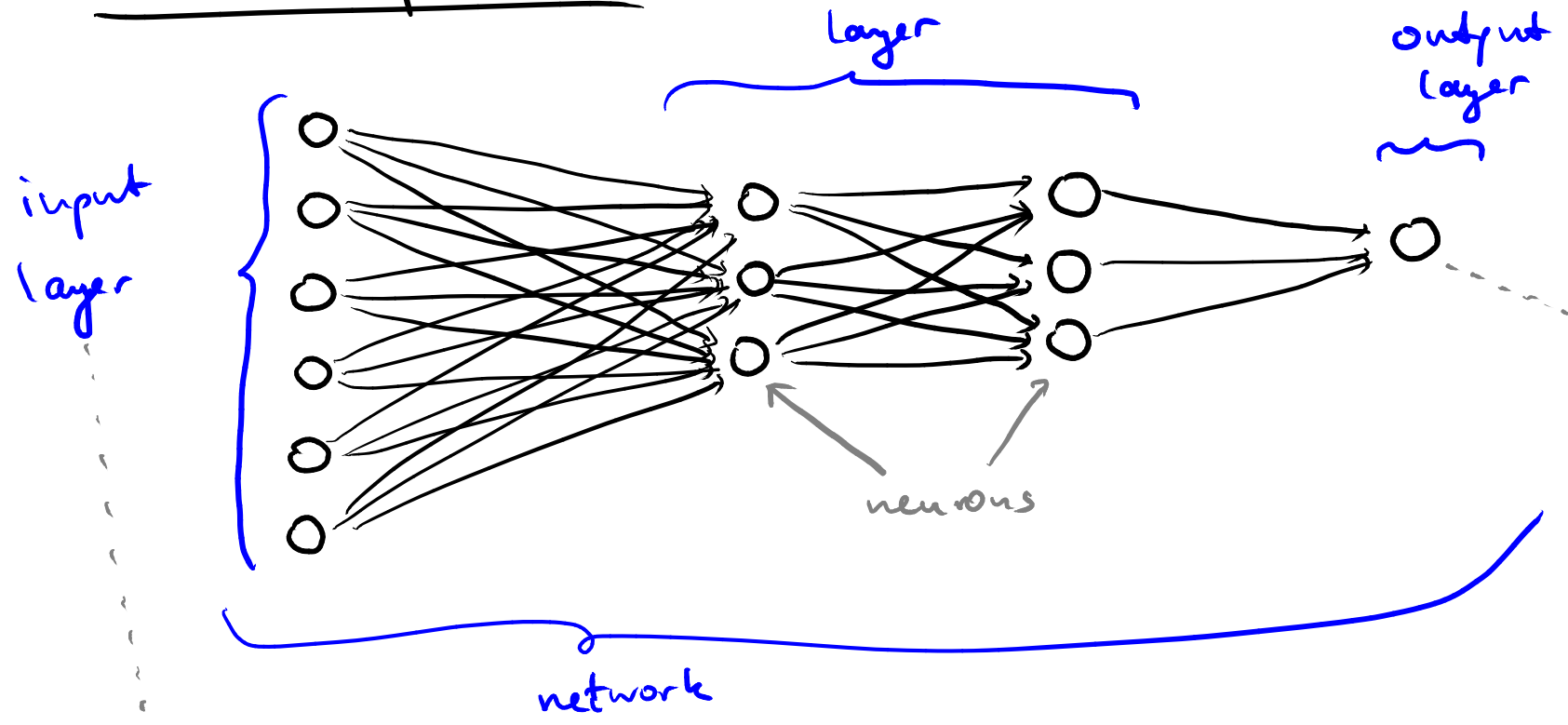
## Example: Handwriting recognition

- 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

A sample of handwritten digits from the MNIST dataset, showing the number 504192. The digits are written in black ink on a white background, with a slightly irregular, human-like appearance.

Following slides based on Chapter 1 of  
[neuralnetworksanddeeplearning.com](http://neuralnetworksanddeeplearning.com)

# Network of neurons



E.g., pixels of an image

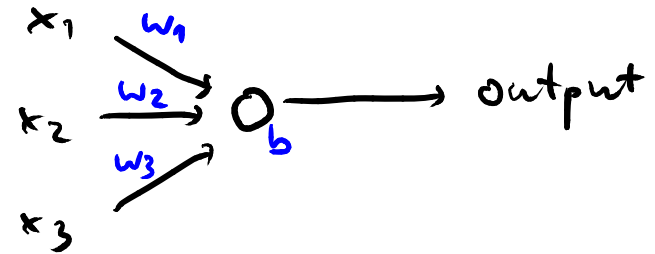
E.g., whether it's the digit 3

## Perceptrons

↳ Most basic kind of neurons

↳ Binary inputs

↳ Binary output



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

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

$w$ .. weights

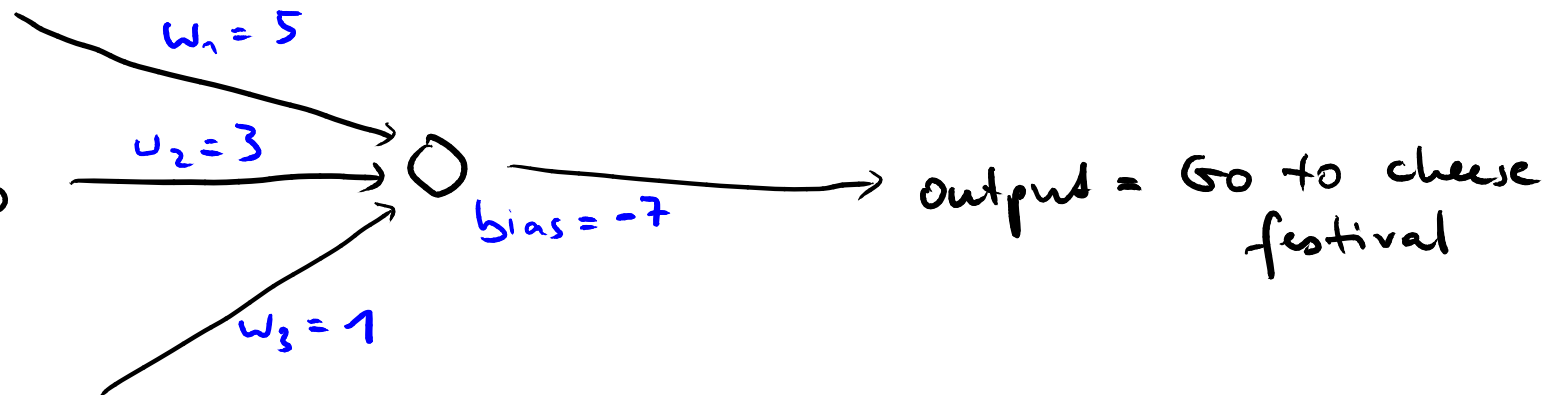
$b$ .. bias

## Example

$x_1 =$  Weather is good

$x_2 =$  Friends go

$x_3 =$  Like cheese



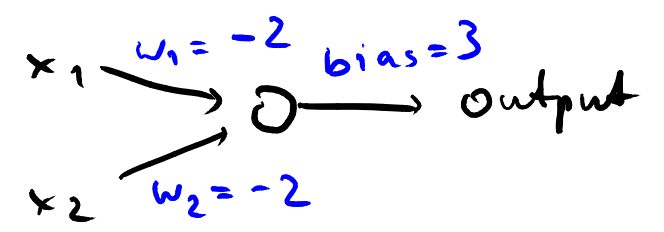
Assume:  $x_1 = 1, x_2 = 1, x_3 = 0$

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

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

# 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

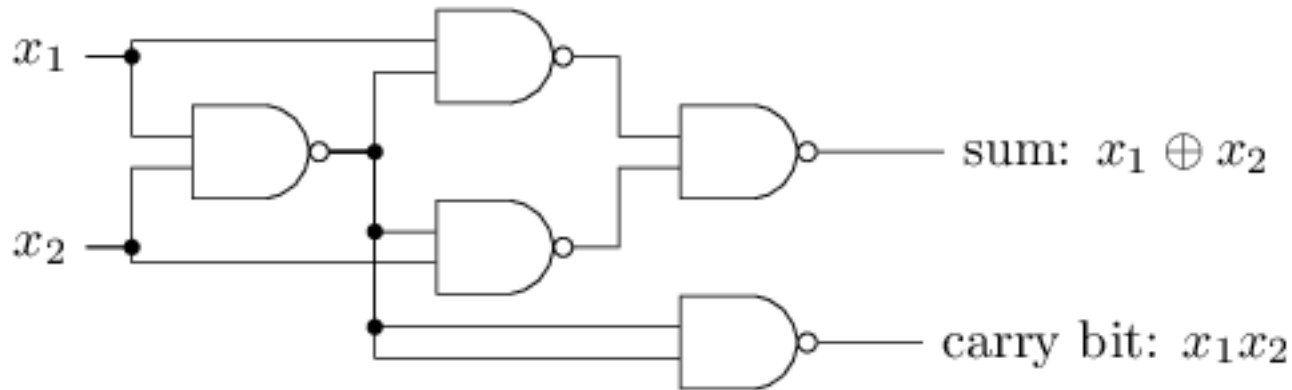
---

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

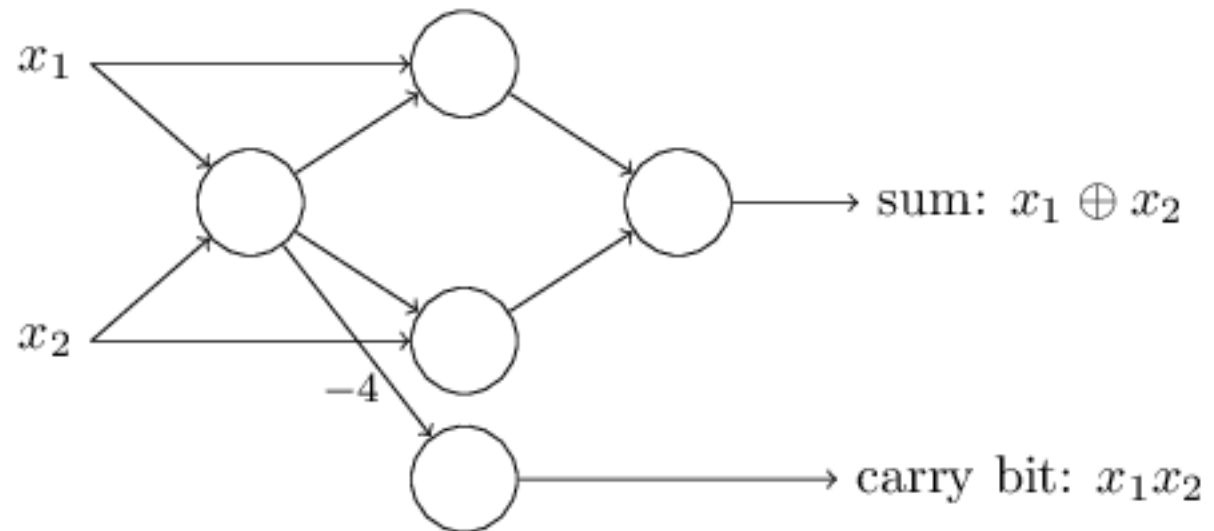
# Example: Adding Two Bits

---

**NAND gate:**



**Network of perceptrons:**

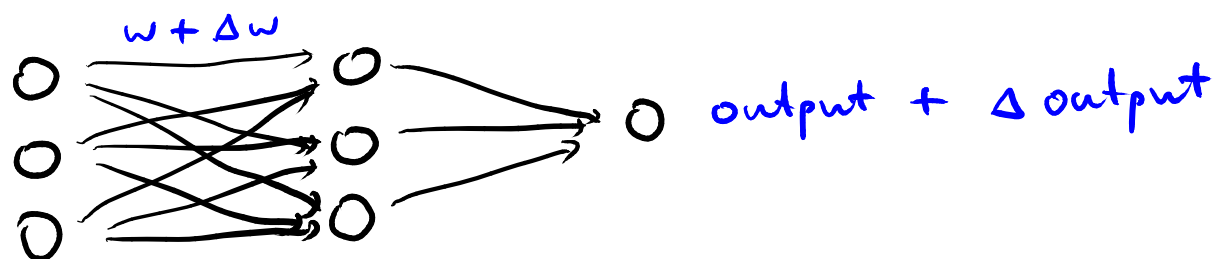


# Challenge: Set Weights and Biases

---

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

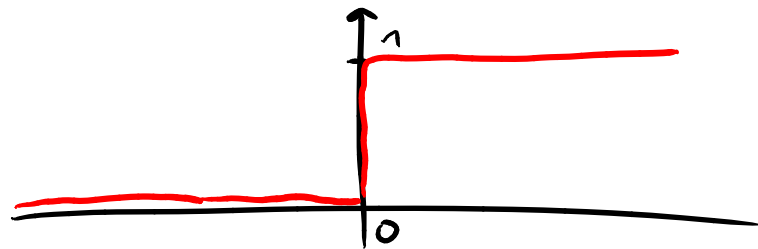
## Making Learning possible



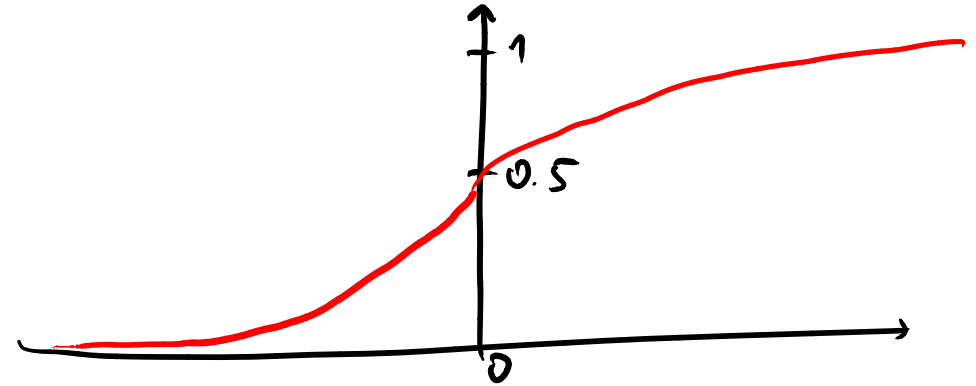
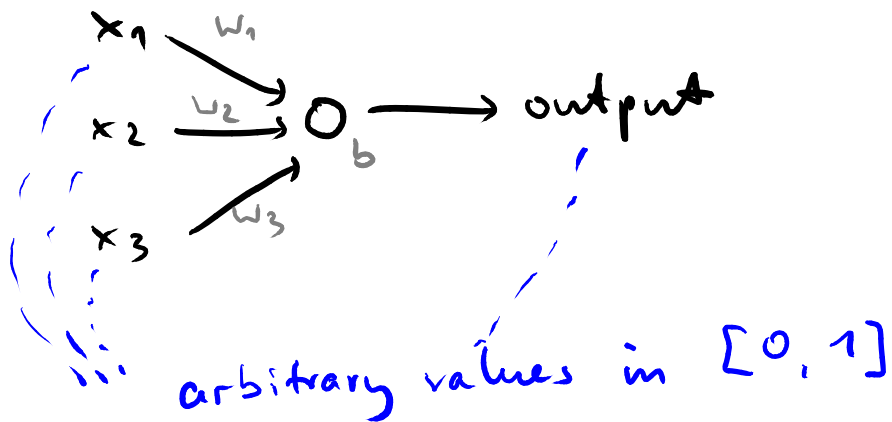
Want: Small change of weights & biases  
cause small change of output

Problem: Perceptron doesn't provide this property

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



## Sigmoid neuron



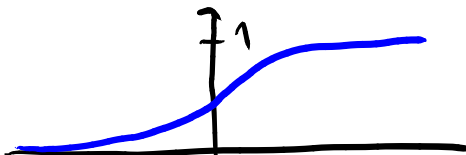
$$\text{output} = \sigma(w \cdot x + b)$$

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

→ Enables learning: Small change causes small change

## Activation Functions

step function 

sigmoid fct. /  
logistic fct. 

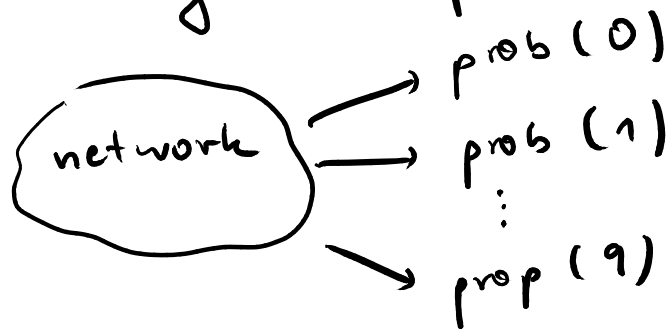
identity fct. 

rectified linear  
unit 

## Learning: Cost Function

- Cost function: feedback on how good the output is for given input

• Example:



If digit is known is 6,  
want output:

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

Actual output may be:

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

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

nb. of training inputs

.. quadratic cost fct.  
or  
mean squared error

# Quiz: Cost Function

---

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

---

Example	Actual	Desired
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?**

$$C = \frac{1}{5} \cdot \sum_x \|y(x) - a\|^2$$

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

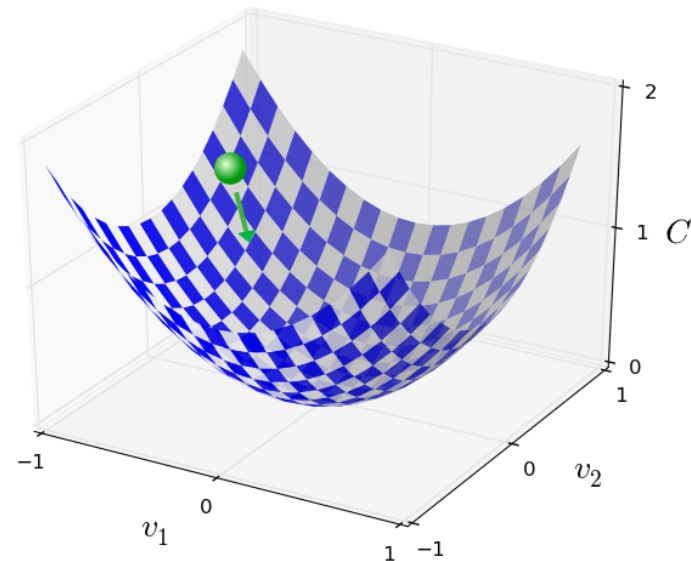
$$= \frac{1}{2} \cdot ( \|(-0.5, 0.5, 0)\|^2 + \|(0, 0, 0)\|^2 )$$

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

# 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



# 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