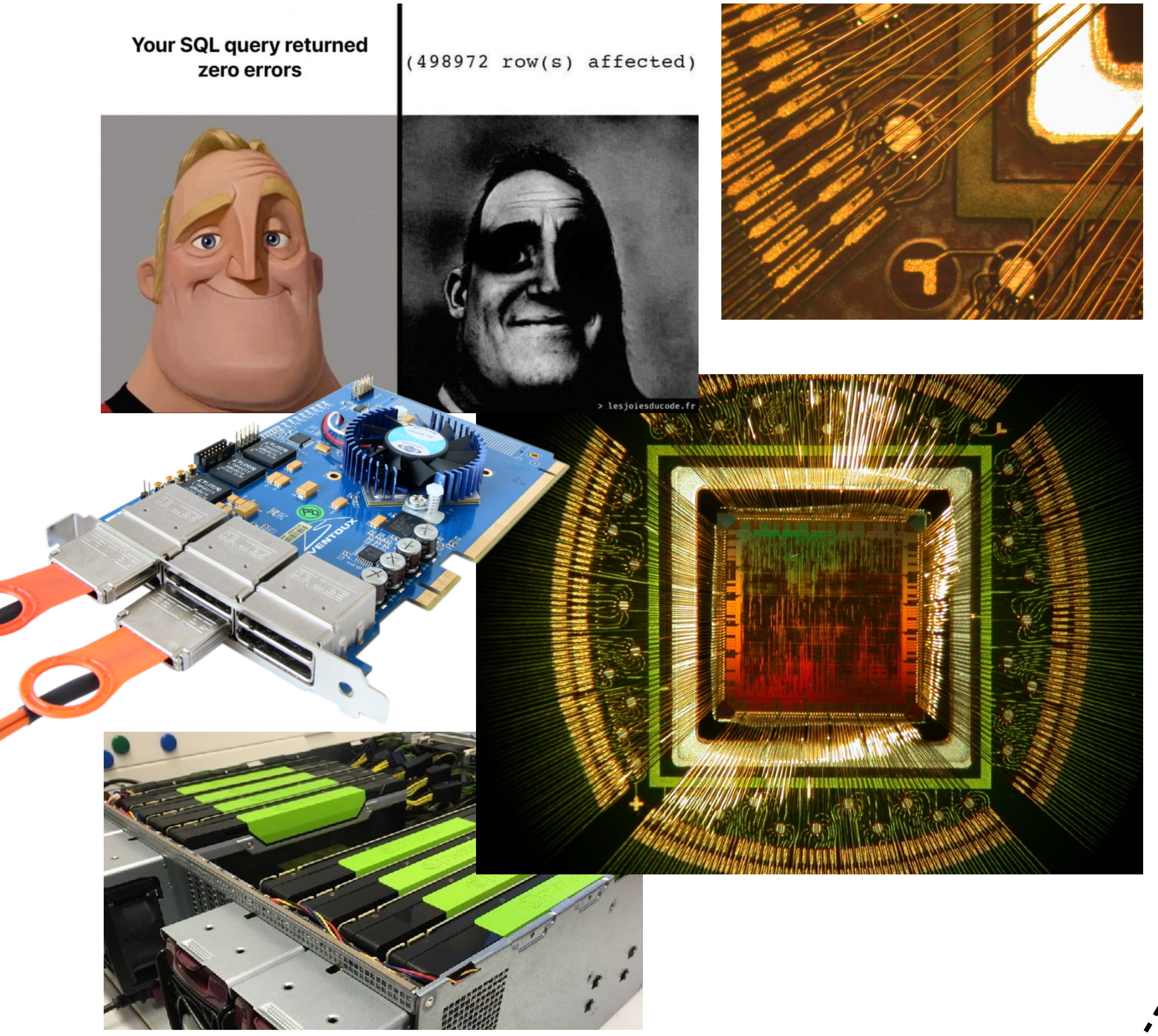
ANFÄNGERPRAKTIKUM
NEURAL NETWORKS FROM SCRATCH
INTRODUCTION

Hendrik Borras, Franz Kevin Stehle
hendrik.borras@ziti.uni-heidelberg.de, kevin.stehle@ziti.uni-heidelberg.de
HAWAII Group, Institute of Computer Engineering
Heidelberg University

ABOUT US

HARDWARE & ARTIFICIAL INTELLIGENCE
HAWAII LAB

From: database engineer, HW designer (ASICS, FPGA), HPC



$$\mathbf{x}^l = \Phi(\mathbf{W} \oplus \mathbf{x}^{l-1} + \mathbf{b}^l)$$

Neural Architectures

tvm

Compiler

Plethora of HW

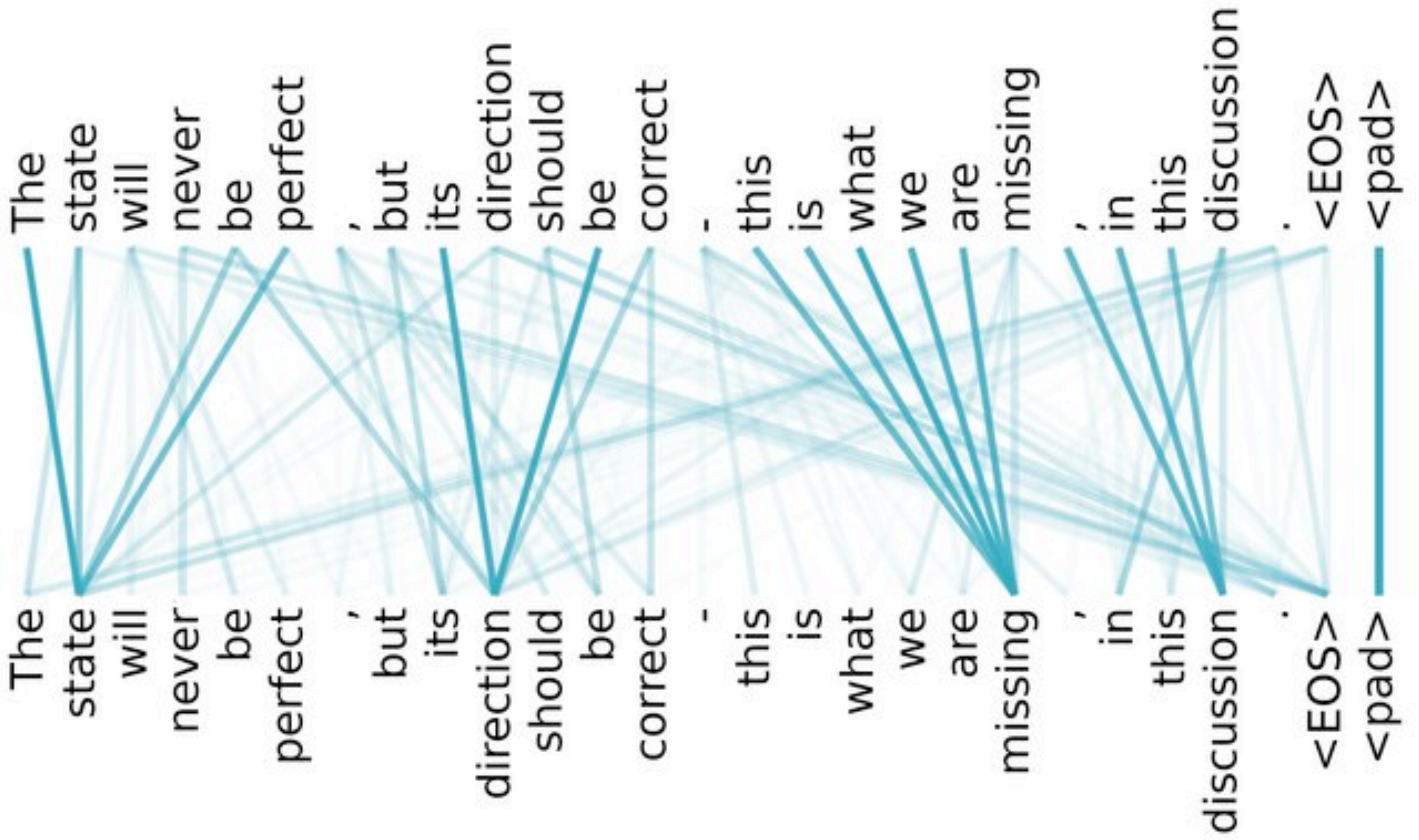
$$perf[\frac{ops}{s}] = p[Watt] \cdot e[\frac{ops}{J}]$$

$$P = afCV^2 + VI_{leakage}$$

To: vertically integrated approach to efficient ML => HW systems for AI

ML APPLICATIONS

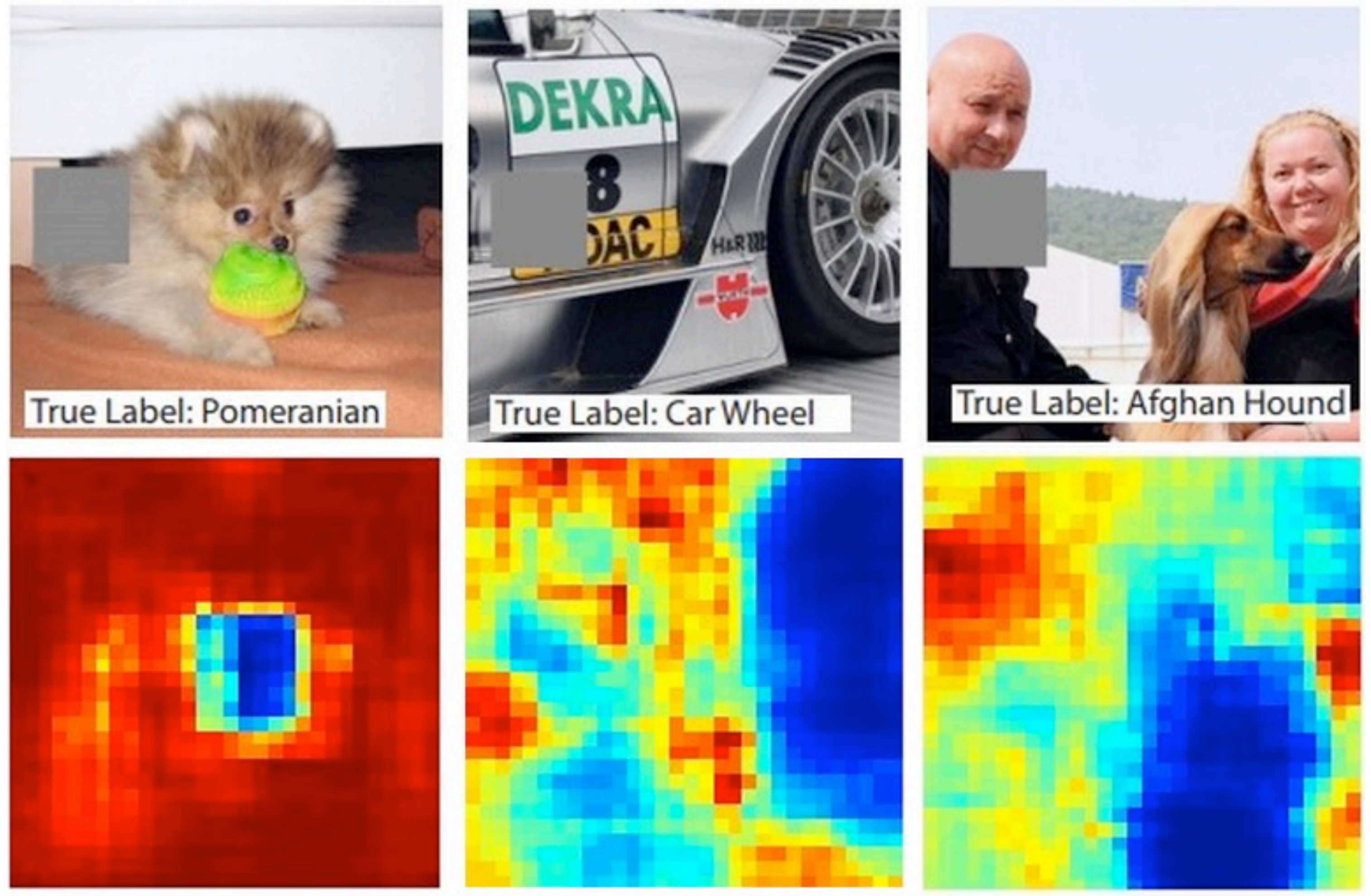
Language Processing



Robotics

Image

Processing



Speech

Recognition

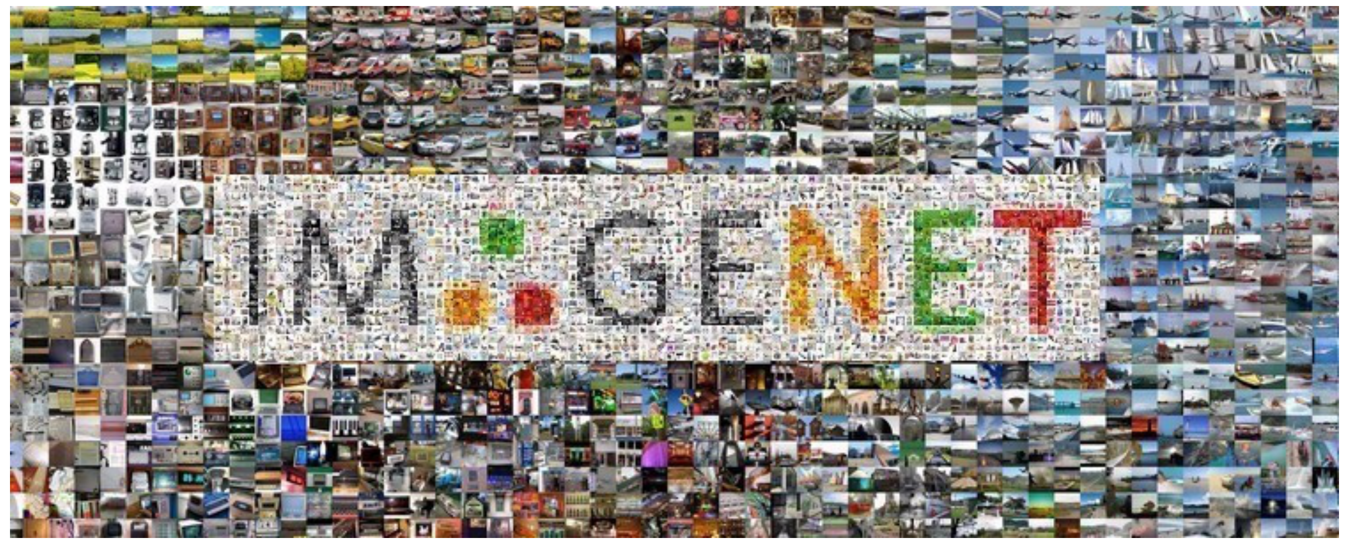
MODERN ML

Image & video: classification, object localization & detection

Speech and language: speech recognition, natural language processing

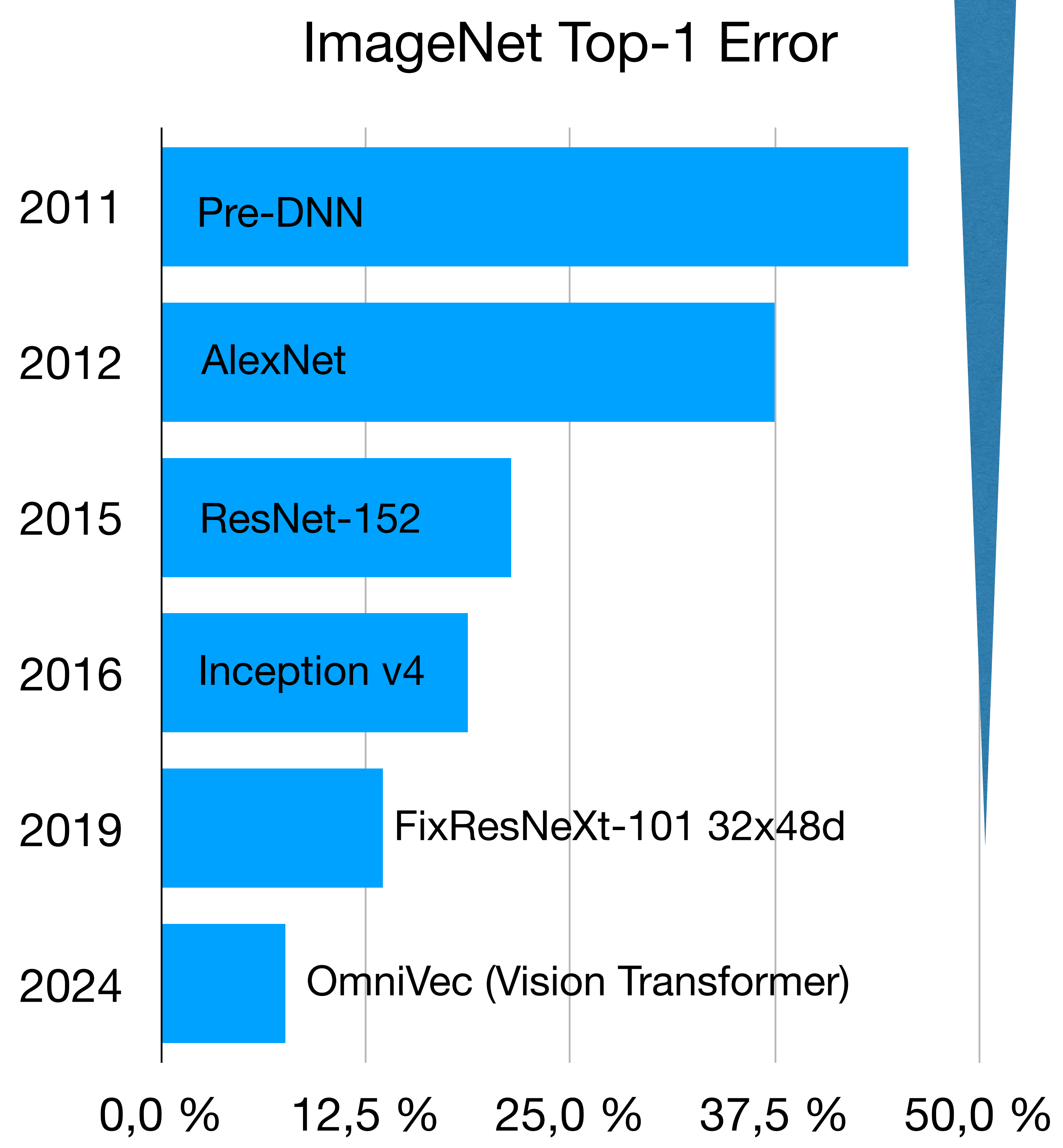
Medical: imaging, genetics, disease prediction

Other: playing of video games, robotics



IMAGENET: 1000 classes

FixResNeXt-101 32x48d:
Training: ~ $O(10^{20})$ OPs total
Inference: ~ $O(10^{12})$ OPs/sample



Artificial Neural Networks (ANNs) deliver state-of-the-art accuracy for many AI tasks … at the cost of extremely high computational complexity

DATASET COMPARISON

	Type	Dataset Samples	Dataset Size
MNIST	Image	60k train + 10k test	~ 45 MB
CIFAR-10	Image	50k train + 10k test	~ 176 MB
ILSVRC2015	Image	1.38M	~ 150 GB
FineVideo	Video	43k	~ 600 GB (3.4k hours)
The Pile	Text	211M (documents)	~ 825 GB
LLAMA Pretraining Sets	Text		~ 4.7 TB

Trains on a reasonable laptop in ~ 10min

Trains in ~ 3 weeks on 2k A100 GPUs consuming ~ 449 MWh [1]

[1] Touvron, H., "LLaMA: Open and Efficient Foundation Language Models", *arXiv e-prints*, Art. no. arXiv:2302.13971, 2023. doi:10.48550/arXiv.2302.13971.

ORGANIZATION

OBJECTIVES

Objectives: The students …

… learn about the mathematical foundations of machine learning

… start applying their skills by implementing a basic model that learns to perform the XOR operation

… continue on to multi-layered models by implementing a multi-layer perceptron (MLP) from scratch

… experience first-hand the requirement of using parallel architectures, in our case GPUs, when scaling up neural networks and learn how to bring their models to the GPU

… apply their acquired knowledge on more complex architectures by implementing a Transformer model from scratch

… implement a more complex models/techniques based on their acquired knowledge as their final project

METHODS & PREREQUISITES

Methodology

- Strong focus on learning from hands-on experience
- Learning to implement neural networks starting with pure Python without *any* additional packages, with usage of the common numerical packages (numpy, CuPy, Scikit-learn) following after -> Allows for a look under the hood not easily possible using modern ML libraries
- Students can choose from a large selection of final project topics based on their specific personal interests

Prerequisites

Passed exams in:

- Einführung in die Praktische Informatik (IPI)
- Programmierkurs (IPK)
- Lineare Algebra 1 (MA4) oder Mathematik für Informatik 1 (IMI1)

Practical experience:

- Intermediate proficiency in Python

ORGANIZATION

Lectures – 2 hours/week

- Lecturers:
  - Hendrik Borras (hendrik.borras@ziti.uni-heidelberg.de)
  - Kevin Stehle (kevin.stehle@ziti.uni-heidelberg.de)
- Time: Wednesday, 14:00 ct

Exercises

- Groups of 2 or 3 students
- Tutorial: Wednesday, after the lecture
- Mixture of reading/exercises/programming/experiments

Project-Based Grading

- Work to be done in groups – individual work must be visible
- Students implement, document, and present an ML program
- Grades are determined by the quality of the project, report, and presentation

ASSIGNMENTS

Practical exercises: Coding, experiments, and reading

Reading & feedback based on paper review

Ideal review here is 2 sentences for each of the following:

1. Primary contribution
2. Key insight of the contribution
3. Your opinion/reaction to the content

Review: rating relative to all other papers (of this venue)

Strong reject, weak reject, weak accept, strong accept

"Old" papers: Optionally give an opinion on how correct the work was in hindsight

AGENDA

Datum	Vorlesung	Übung
23.10	Einführung	Reading + Polynomial curve fitting
30.10	XOR learning & Visualization	XOR learning + Visualization with WandB
6.11	NN/MLP learning	MLP from scratch
13.11	GPUs	Cluster access and GPU acceleration (CuPy or cuda-numba)
20.11	Attention & Transformers	Transformer from scratch + Project proposals
27.11	Project proposal discussion and kick-off	
…	N-times Project updates and questions	
KW 8?	Poster session	

ADDITIONAL MATERIAL

Papers

- Badillo et al.: An Introduction to Machine Learning (https://ascpt.onlinelibrary.wiley.com/doi/pdfdirect/10.1002/cpt.1796)
- Vaswani et al.: Attention Is All You Need (https://arxiv.org/abs/1706.03762)
- Horowitz: Computing's energy problem (and what we can do about it) (https://ieeexplore.ieee.org/document/6757323)

Textbooks

- Goodfellow et al.: Deep Learning (https://www.deeplearningbook.org)
- Tunstall, Lewis: Natural Language Processing with Transformers (en: https://katalog.ub.uni-heidelberg.de/cgi-bin/titel.cgi?katkey=68944723, de: https://katalog.ub.uni-heidelberg.de/cgi-bin/titel.cgi?katkey=69054303)
- Alammar, Grootendorst: Hands-On Large Language Models (https://katalog.ub.uni-heidelberg.de/cgi-bin/titel.cgi?katkey=ext_FETCH-LOGICAL-p892-d8d60503679c7f2b78a36513f71ff1195247d65c3c983e61ae98b4d6692e36293)
- Hwu et al.: Programming Massively Parallel Processors (https://www.sciencedirect.com/book/9780323912310/programming-massively-parallel-processors)

Other

- Deep Learning Cheat Sheet (https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning)

ADDITIONAL MATERIAL

https://csg.ziti.uni-heidelberg.de/teaching/ap_nn_from_scratch_materials/

Uploaded here:

- Exercises
- Lecture slides
- Additional materials

LINEAR AND POLYNOMIAL REGRESSION

Learning, generalization, model selection, regularization, overfitting

With material from Andrew Ng (CS229 lecture notes) and Christopher Bishop (Pattern Recognition and Machine Learning)

SUPERVISED LEARNING

Based on the given housing data, is it possible to learn to predict the costs of other houses?

➡ Prediction of "Unseen data"

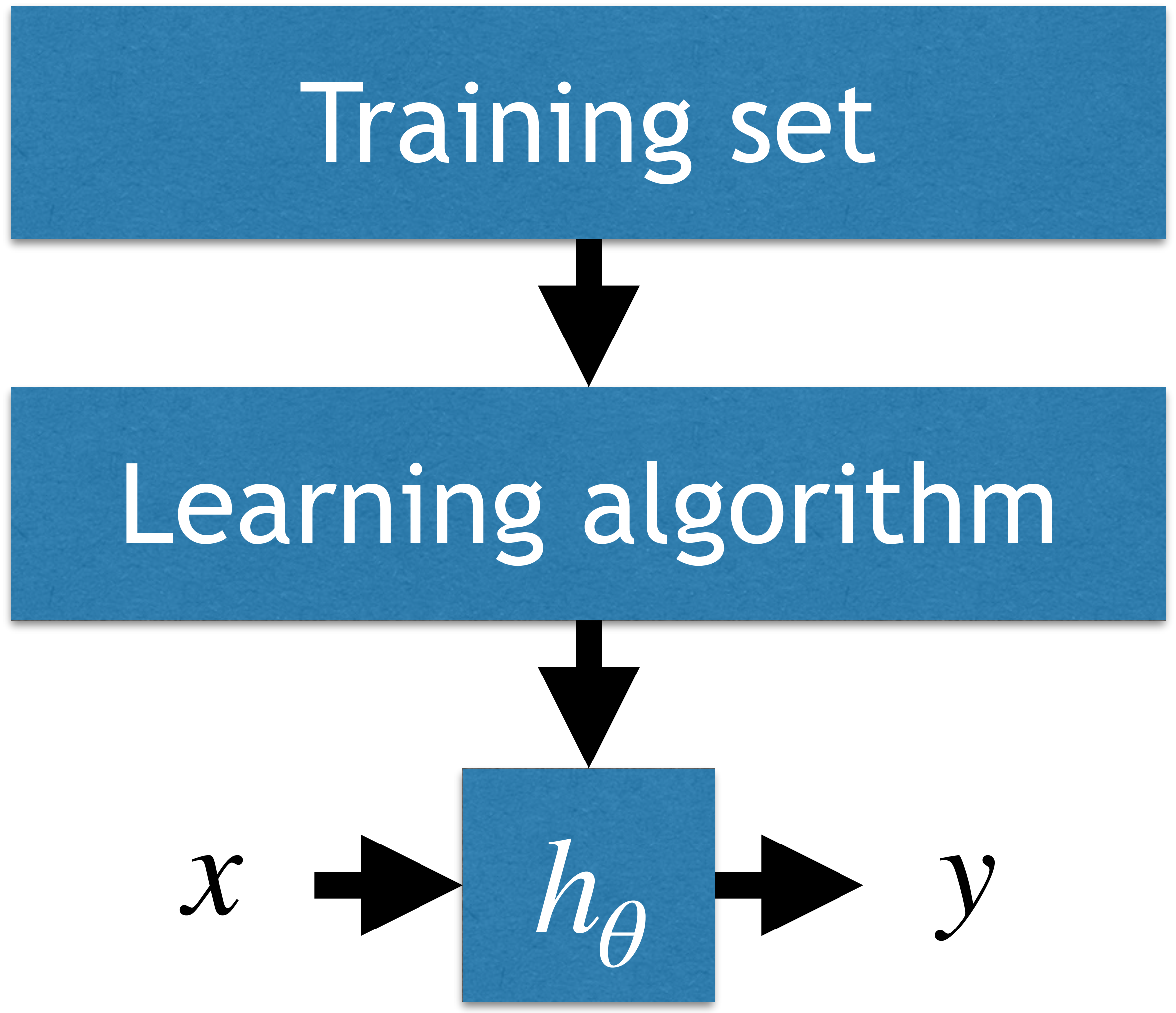
Living area (feet^2)	#bedrooms	Price (1000\$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Notation

- $x^{(i)}$: Input features of sample i
- $t^{(i)}$: Target variable (or output variable or label) of sample i
- $(x^{(i)}, t^{(i)})$: Training sample (or observation) i
- Training set: set of all training samples (size N)

Supervised learning problem: find good prediction function $y = h_\theta(x)$

- θ (theta) are the parameters (weights) of the model
- Classification (discrete) vs. regression (continuous) problem



LINEAR REGRESSION

$$\mathbf{x} = (x_1, x_2)^T = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$$

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Supervised learning: choose function h

$$y = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Simplification given D model parameters:

$$h_\theta(\mathbf{x}) = h(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^T \mathbf{x}$$ (model intercept θ_0 by $x_0 = 1$)

Learning: make $h(x)$ close to t for the N training samples we have

Cost (or error or loss) function "how close is that": $J(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left(h_\theta(x^{(n)}) - t^{(n)}\right)^2$

Least-squares method to find the optimal parameters by minimizing this sum of squared residuals

GRADIENT DESCENT

Choose θ such that $J(\theta)$ is minimal

Start with initial guess of θ, repeatedly perform gradient descent:

$\theta_d := \theta_d - \alpha \frac{\partial}{\partial \theta_d} J(\theta)$, simultaneously for all $d = 1,\ldots,D$ and learning rate α

$$\frac{\partial}{\partial \theta_d} J(\theta) = \frac{\partial}{\partial \theta_d} \frac{1}{2} \sum_{n=1}^{N} (h_\theta(x) - t)^2 = \frac{2}{2} \sum_{n=1}^{N} (h_\theta(x) - t) \cdot \frac{\partial}{\partial \theta_d} \Big((\sum_{i=1}^{D} \theta_i x_i) - t \Big) = \sum_{n=1}^{N} (h_\theta(x) - t) x_d$$

Hint: remember chain rule of calculus - for $f(x) = u\big(v(x)\big), f'(x) = u'\big(v(x)\big)v'(x)$

=> Update rule: $\theta_d := \theta_d + \alpha \sum_n \big(t^{(n)} - h_\theta(x^{(n)})\big) x_d^{(n)}$

Magnitude of update is proportional to error term

Which set of the training samples (elements n) to consider for one update?

BATCH GRADIENT DESCENT

Only one global optima as J is a convex quadratic function

Batch gradient descent: $\forall d \in D$

$$\theta_d := \theta_d + \alpha \sum_{n=1}^{N} \left(t^{(n)} - h_\theta(x^{(n)})\right) x_d^{(n)}$$

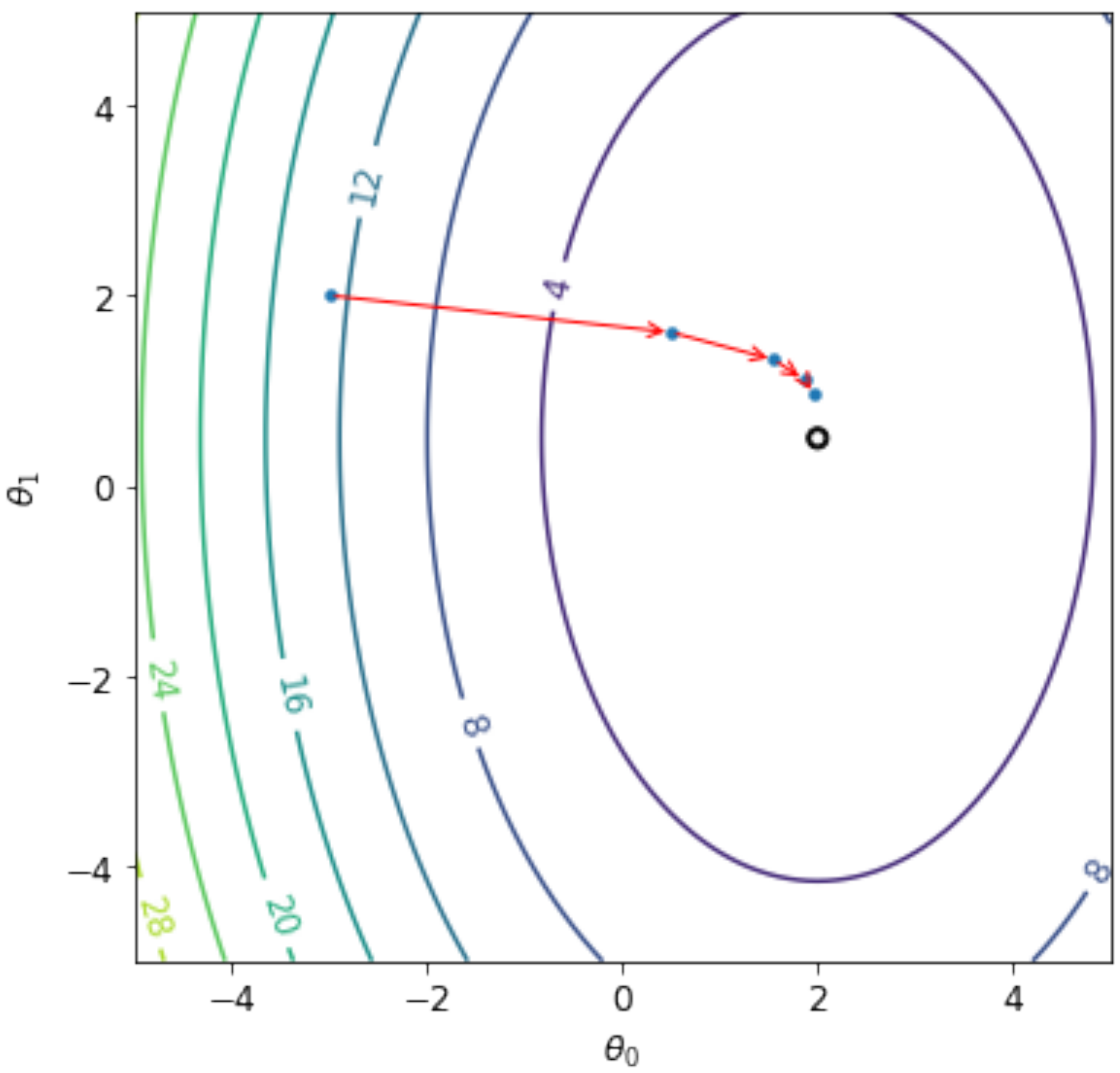
Repeat until convergence

Looks at every training sample ($\forall n \in N$) on every step

Number of steps depend on convergence

Guaranteed to be optimal, but expensive

Cost function



STOCHASTIC (INCREMENTAL) GRADIENT DESCENT

Scanning the complete data set for every step can be costly

Stochastic gradient descent is based on randomly selecting training samples to perform gradient descent

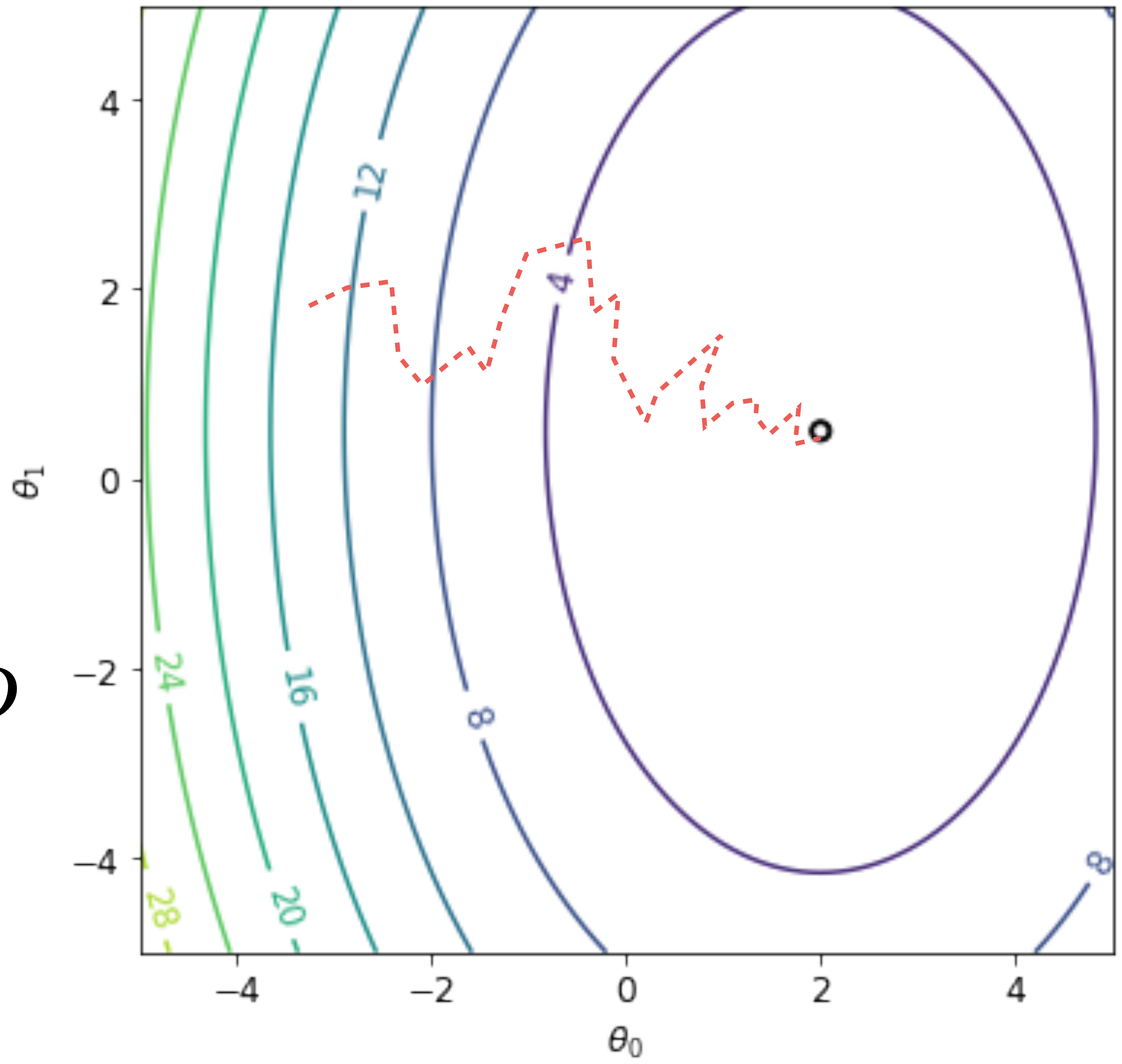
for all n in N:

$$\theta_d := \theta_d + \alpha\left(t^{(n)} - h_\theta(x^{(n)})\right)x_d^{(n)}; \forall d \in D$$

Repeat until convergence

Makes progress for each training sample

Cost function



POLYNOMIAL CURVE FITTING

Training set: N observations of $\mathbf{x} = (x_1, \ldots, x_N)^T$ and $\mathbf{t} = (t_1, \ldots, t_N)^T$

Ground truth: $t = sin(2\pi x)$, but (Gaussian) noise present

Many data sets have an underlying regularity, but observations are corrupted by random noise

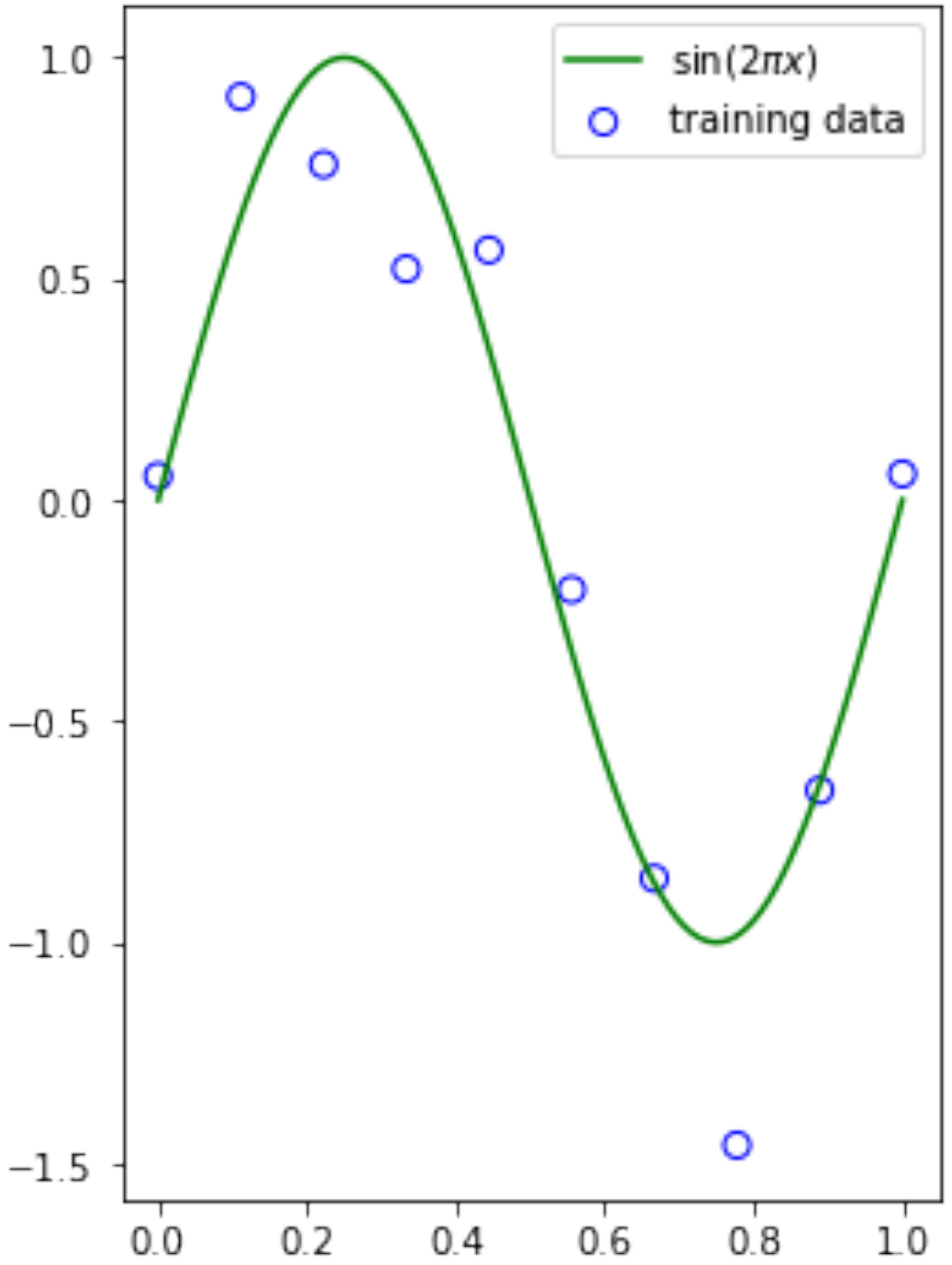
Objective: make good predictions $\hat{y}$ of new values $\hat{x}$

<u>Generalize</u> from a finite data set

Model: polynomial function of order of M

$$h(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{m=0}^{M} w_m x^m$$

Although $h(x, \mathbf{w})$ is a nonlinear function of x, it is a linear function of the coefficients $\mathbf{w}$ => linear model



FITTING

Determine the coefficients $\mathbf{w}$ by fitting to N training samples

Minimize error function $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h(x_n, \mathbf{w}) - t_n\right)^2$

Again: quadratic function of coefficients $\mathbf{w}$

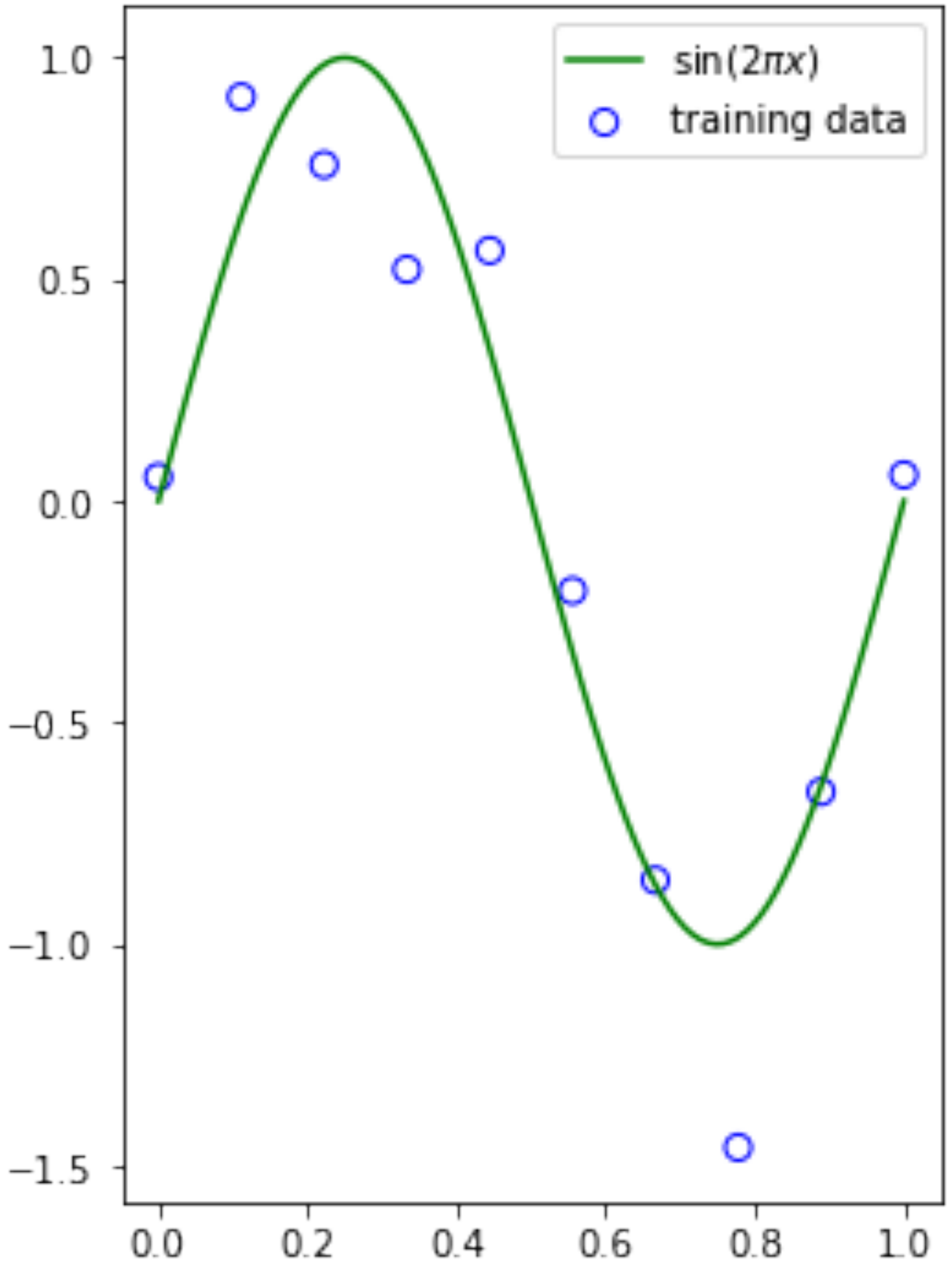
=> partial derivates (with respect to the coefficients) are linear in the elements of $\mathbf{w}$

=> unique solution $\mathbf{w}^*$

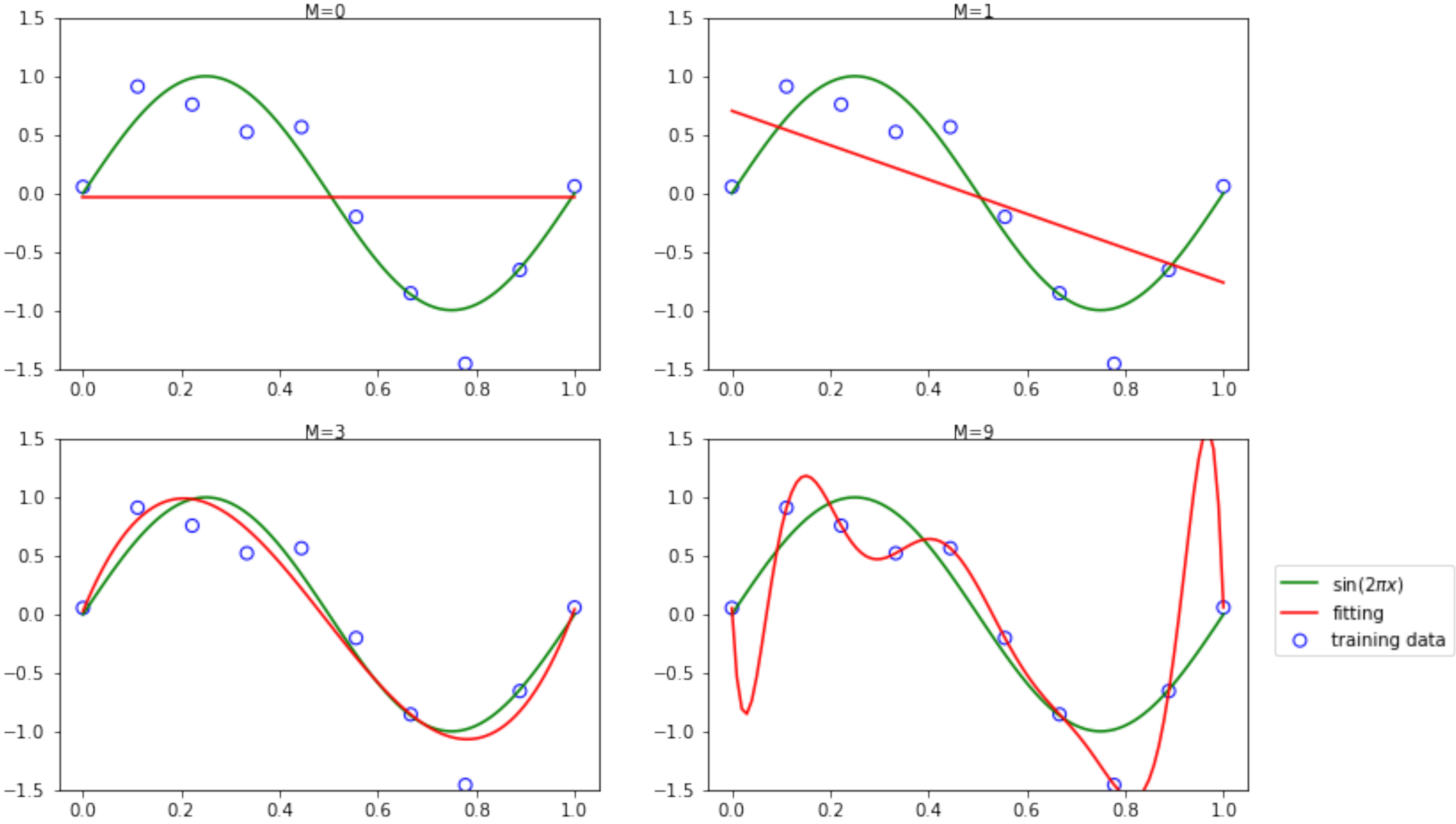
But what about order M?

$$h(x, \mathbf{w}) = \sum_{m=0}^{M} w_m x^m$$

=> model selection



MODEL SELECTION



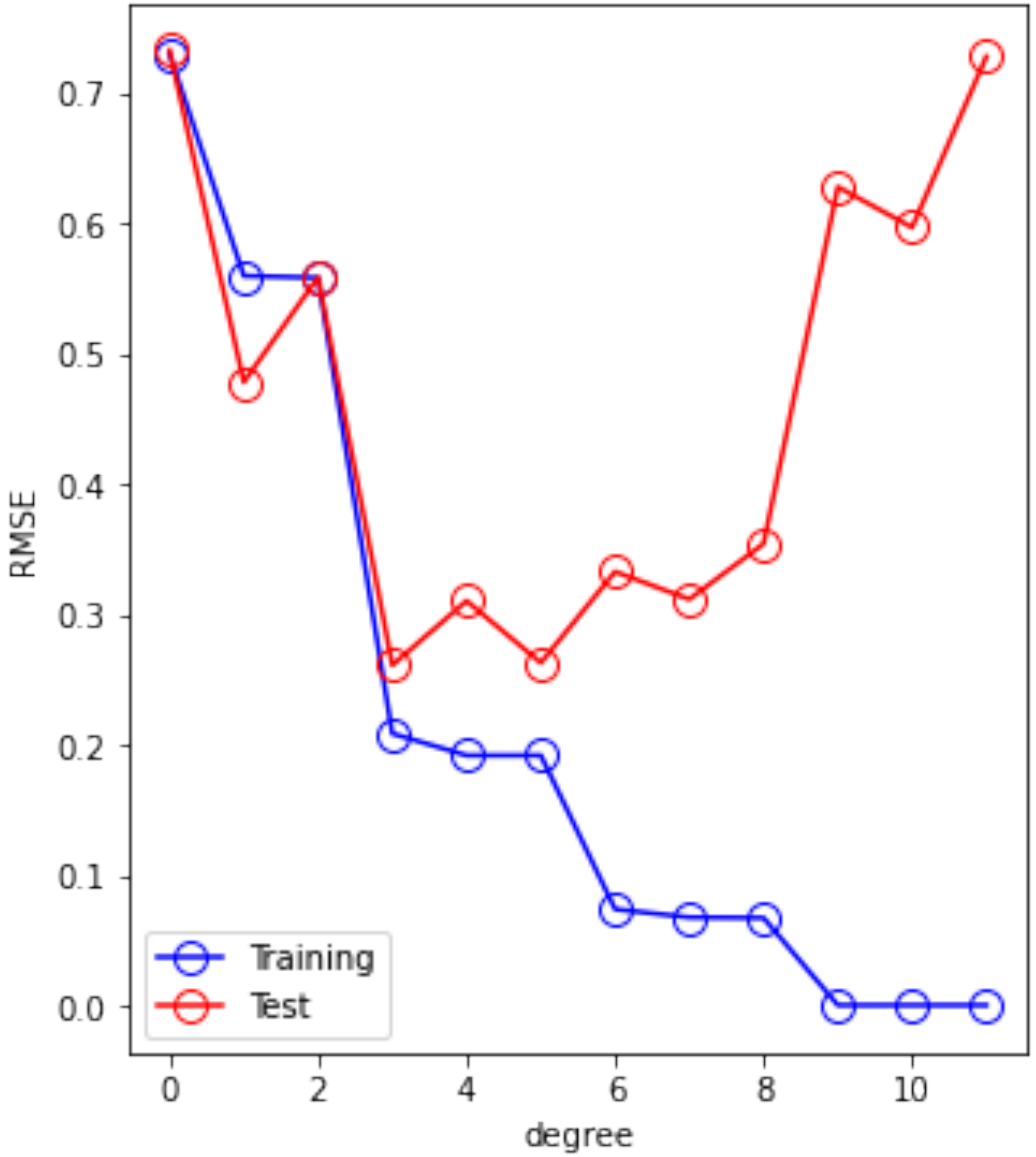
GENERALIZATION AND OVERFITTING

Good generalization: making accurate predictions for new (unseen) data

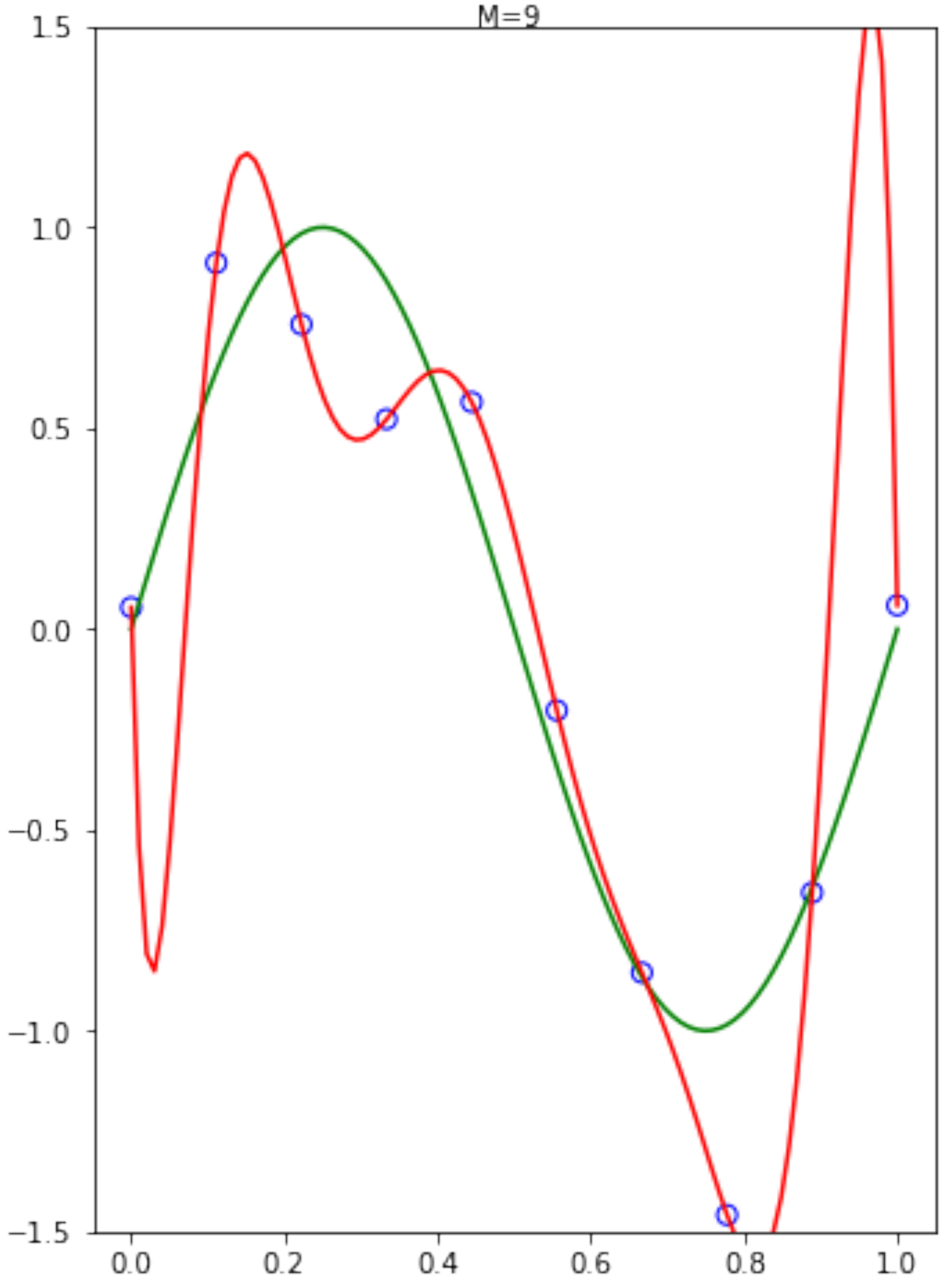
- Test set: here generated the same way as training set
- Usual procedure: Split dataset into training, test (and sometimes validation) sets, with the test set remaining unknown to the model during training (Very important!)

Identify overfitting

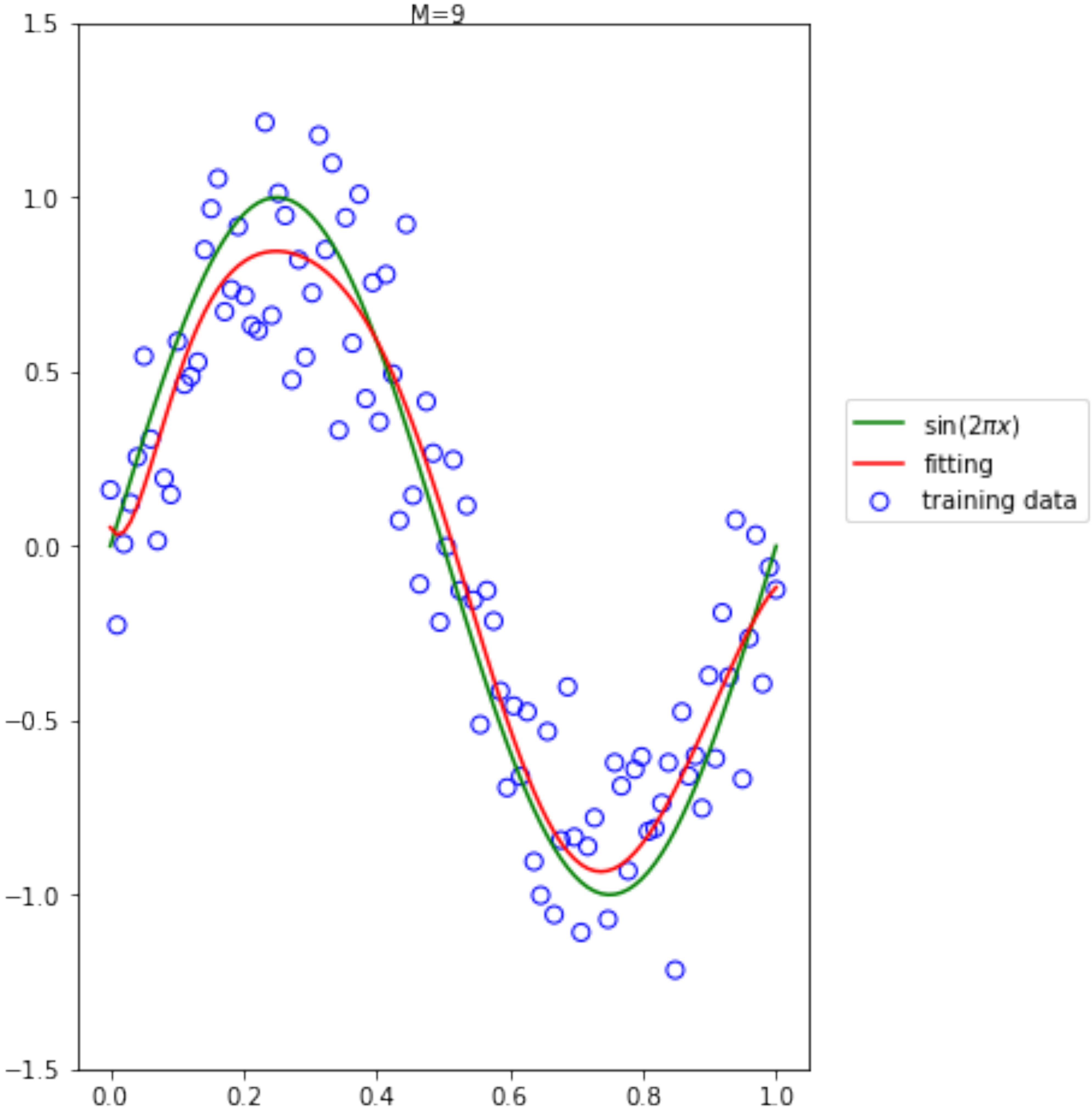
- Training error: $E(\mathbf{w}^*)$ for the training set
- Test error: $E(\mathbf{w}^*)$ for the test set
- If datasets are of different size: $E_{RMS} = \sqrt{2E(\mathbf{w})/N}$



MODEL SELECTION DEPENDS ON DATA SET SIZE



N=10



N=100

REGULARIZATION

Regularization can control overfitting by adding a penalty term to the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h(x_n, \mathbf{w}) - t_n\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|$$

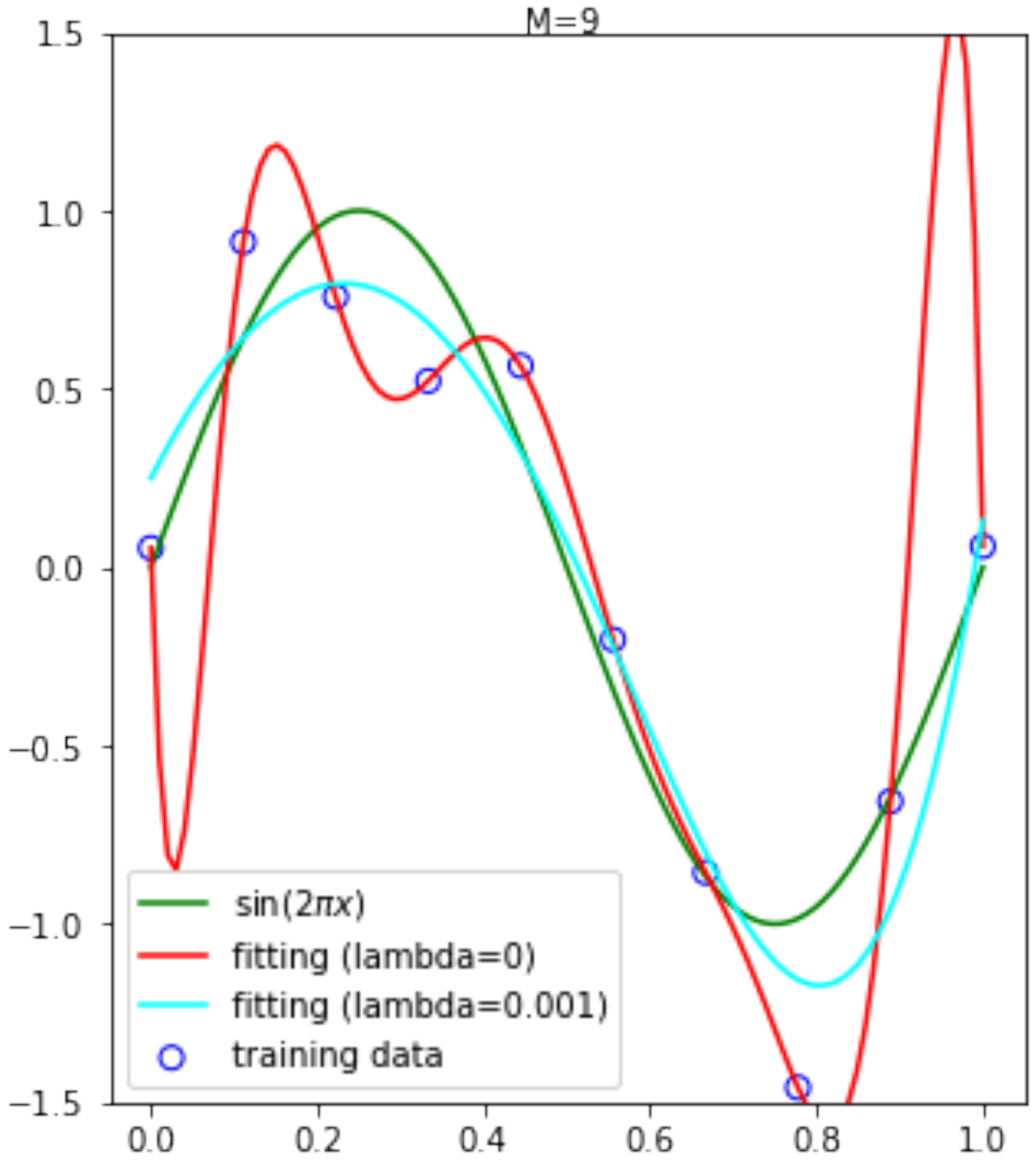
where $\|\mathbf{w}\| = \mathbf{w}^T\mathbf{w}$

λ governs the relative importance of the regularization term

Such shrinkage methods reduce the value of the coefficients

Quadratic regularizer: *ridge regression* or *weight decay* or *L2 regularization*

Validation set to optimize either M or λ



WRAPPING UP

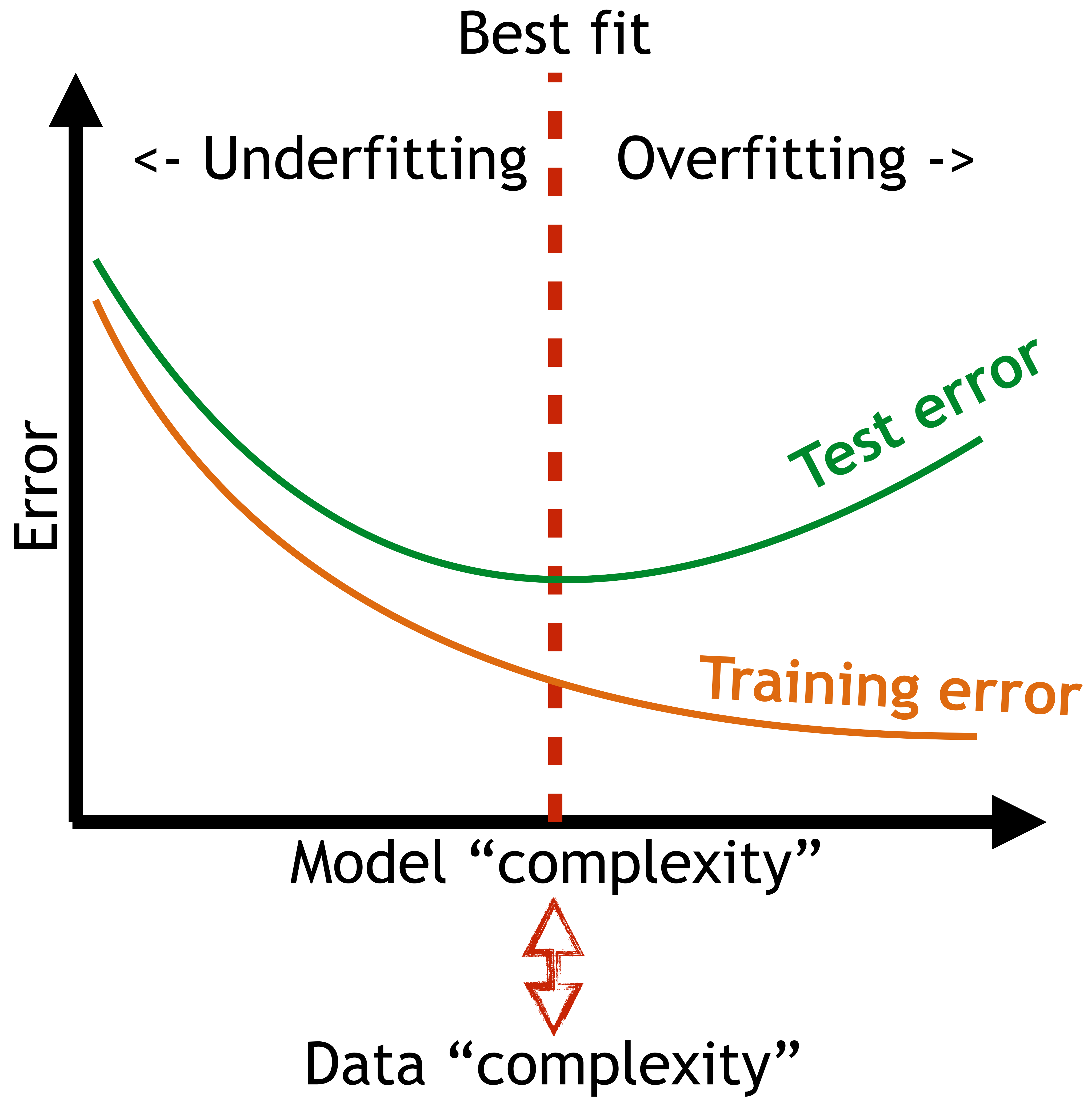
SUMMARY

This course: Teaches how Neural Networks actually work under the hood

Linear regression example

- Parameters and how they are learned
- Generalization and model selection
- Overfitting and regularization
- <u>Linear models are *not* universal approximators</u>

<u>Artificial Neural Networks (ANNs) *are* universal approximators</u>, but their gains are paid in higher computational complexity and lower interpretability



SHORT 5 MIN BREAK

THEN: EXERCISE AND PROJECT GROUPS

FORMALITIES

Exercises and Projects are to be done in groups

Exercises are not graded

- However: A group must submit solutions for at least 2/3 of the exercises
- Hands-on part of the first part of the practical
- Ensure consistent knowledge between the groups

Solutions are to be submitted digitally via E-Mail

In the tutorial (i.e. here) solutions are presented and discussed

Group assignment: Next slide

GROUP ASSIGNMENT

Please choose a group now

Fill out the form, that's being passed around

Groups of 2 to 3 members are allowed

Please inform us about changes in your group

THIS WEEKS EXERCISE

Reading one paper and writing a review

- Try to write a short and concise review! (significantly less than one page)
- See guidelines from the lecture

Polynomial curve fitting

- Become accustomed to Python and array/tensor notation with numpy
- Overfitting example from the lecture

https://csg.ziti.uni-heidelberg.de/teaching/ap_nn_from_scratch_materials/