Zurich University
of Applied Sciences

Machine Learning
V02: Formulating learning problems

Ingredients to learning
Machine learning from scratch

With material from Andrew Y. Ng, Coursera

Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

Educational objectives

- **Name** the **parts** that make up a machine learning **solution** as well as **concrete instances** of each
- **Understand** the **linear regression** with stochastic **gradient descent** algorithm from scratch
- **Implement** a simple machine **learning algorithm from scratch** (that is, from its mathematical description)

1. INGREDIENTS TO LEARNING

Recap

What is a well-posed learning problem (according to [Mitchell, 1997])?

T:
P:
E:

There are literally thousands of learning algorithms; how can we characterize them?

Designing a learning solution

High-level view:

Task

E.g.,
- learn playing Tic-Tac-Toe

Performance measure

E.g.,
- #games won against project team

training Experience

E.g.,
- games played against itself

Non-trivial: Use prior knowledge / experience on what works under which conditions.

Non-trivial: What features to use? Then pre-processing takes 80% of the time.

Not so trivial: Often not the standard loss function of your favourite ML library.

Practitioner's view:

Representation:
$\mathcal{H}$, features

E.g.,
- Full board state as string of X/-/O chars
- Table mapping strings to winning probability; or neural network?

Evaluation:
Loss (objective) function

E.g.,
- Sparse rewards: +1 for win, 0 for draw or loss

Optimization:
Regime to search $\mathcal{H}$

E.g.,
- TD(1) reinforcement learning

Examples for practical components
From [Domingos, 2012]

(Not all possible tuples of $< representation, evaluation, optimization >$ exist / make sense)

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

In a nutshell: Use **experience** (own or read), **experiment** a lot, **constrain** solutions

How to select? (we come back to this question often…)

- Remember V01: No generally best solution available (no free lunch)
  → see V03 and V06 for best practices on model selection
- Guide: *«What **prior knowledge** is **easily** expressed in certain **features** & **models**?»*
- Relieve: **Good & compact features** are more important than model choice

Ask yourself: *«**Do I see** the sought **patterns in** these **features** alone?»*

Formulating a machine learning solution
Quizzy 1/5

Suppose we feed a learning algorithm a lot of historical weather data, and have it **learn to predict weather**. In this setting, **what is** it's training experience ***E***?

- ❑ None of these
- ❑ The probability of it correctly predicting a future date's weather
- ❑ The process of the algorithm examining a large amount of historical weather data
- ❑ The weather prediction task

Formulating a machine learning solution
Quizzy 2/5

Suppose we are working on weather prediction, and use a learning algorithm to **predict tomorrow's temperature** (in degrees Celsius). Would you treat this as a **classification or** a **regression** problem?

- ☐ Classification
- ☐ Regression

Formulating a machine learning solution
Quizzy 3/5

Suppose we are working on stock market prediction, and we would like to **predict whether** or not a particular **stock's price will be higher** tomorrow than it is today. You want to use a learning algorithm for this. Would you treat this as a **classification or a regression** problem?

- ❑ Classification
- ❑ Regression

Formulating a machine learning solution
Quizzy 4/5

Some of the problems below are best addressed using a supervised learning algorithm, and the others with an unsupervised algorithm. **Which** of the following **would you apply supervised learning** to?

- ❑ Examine a web page, and classify whether the content on the web page should be considered “child friendly” (e.g., non-pornographic etc.) or ”adult”
- ❑ Examine a large collection of emails that are known to be spam email, to discover if there are sub-types of spam mail
- ❑ In farming, given data on crop yields over the last 50 years, learn to predict next year’s crop yields
- ❑ Take a collection of 1’000 essays written on the Swiss economy, and find a way to automatically group these essay into a small number of groups that are somehow “similar” or “related”

Formulating a machine learning solution

Quizzy 5/5

Many substances that can burn…

…(such as gasoline and alcohol) have a chemical structure based on carbon atoms. For this reason they are called hydrocarbons. A chemist wants to **understand how the number of carbon atoms in a molecule affects how much energy is released** when that molecule combusts (meaning that it is burned). The chemist obtained the dataset below. In the column on the right, "kJ/mol" is the unit measuring the amount of energy released.

- ❑ Is it classification or regression?
- ❑ What is X, Y (the training data)?

- ❑ What could be the relationship? How to gain first insight?

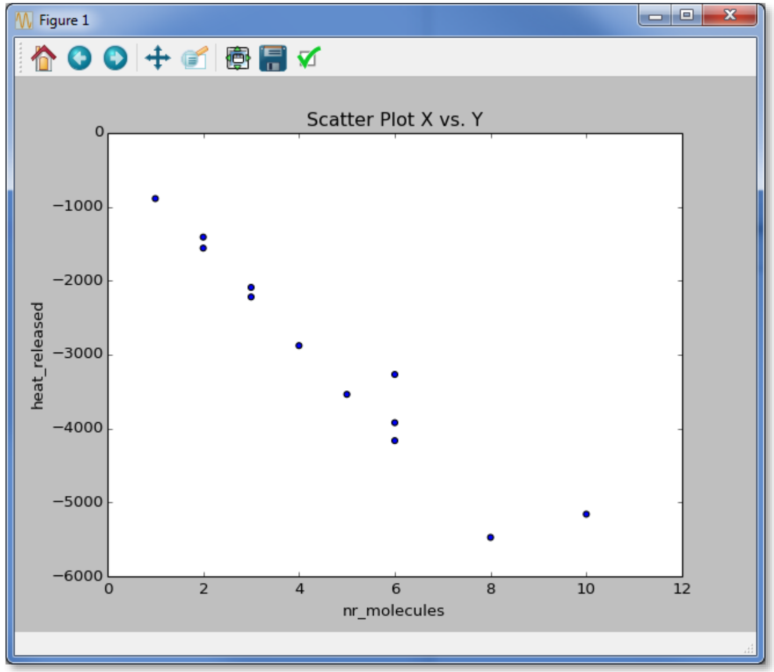
Name of molecule	Number of carbon atoms in molecule	Heat released when burned (kJ/mol)
Methane	1	-890
Ethene	2	-1411
Ethane	2	-1560
Propane	3	-2220
Cyclopropane	3	-2091
Butane	4	-2878
Pentane	5	-3537
Benzene	6	-3268
Cycloexane	6	-3920
Hexane	6	-4163
Octane	8	-5471
Napthalene	10	-5157

Getting first insights
Example: Quizzy 5/5

```
import matplotlib.pyplot as plt
import pandas as pd

data_frame = pd.read_excel("hydrocarbons.xlsx")

plt.scatter(data_frame['nr_molecules'],
data_frame['heat_release'])
plt.title("Scatter Plot X vs. Y")
plt.xlabel(data_frame.columns[1])
plt.ylabel(data_frame.columns[2])
plt.show()
```



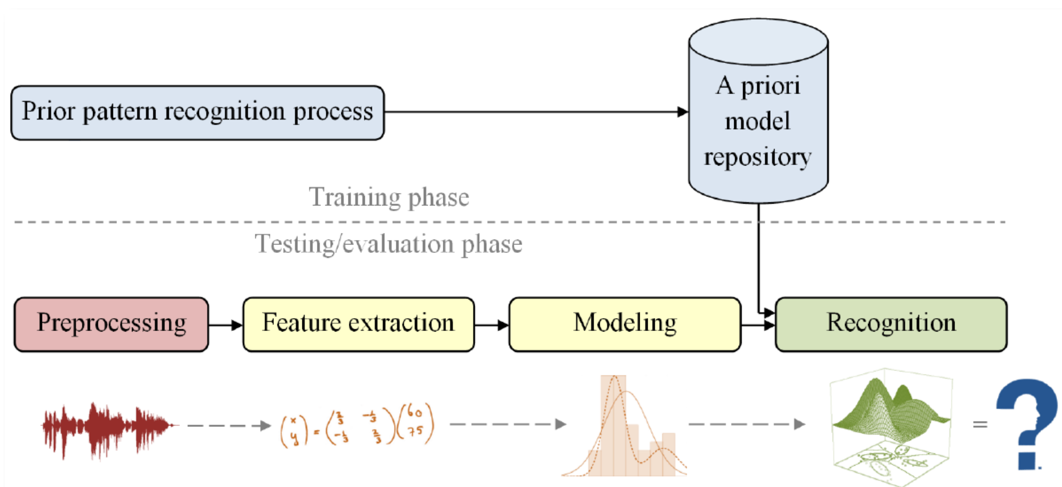
Low-hanging fruits

- Exploratory data analysis (visualization) → this and next slide
- Trying simpler models first → next section

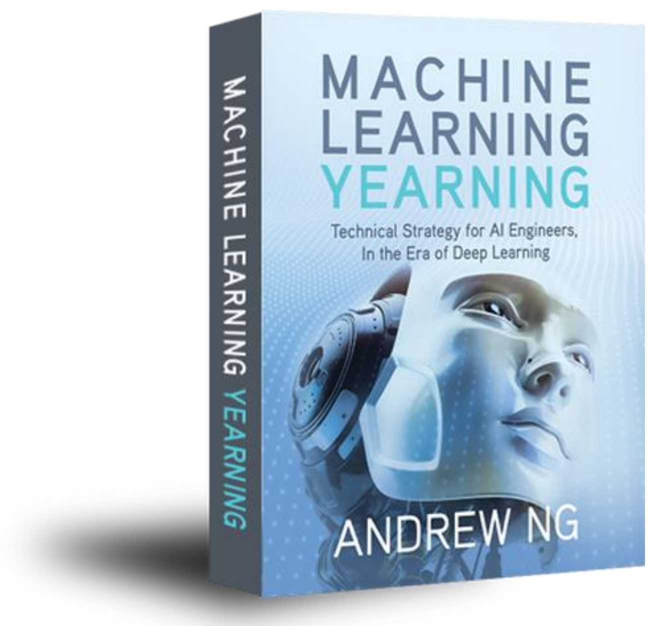
The Machine Learning development process
Exploration & experimentation

Necessity of a distinct conceptual approach
- Modeling data ≠ {software dev., business process mngmt., data base design, stat. analysis, …}

In a nutshell
- Focus on **systematic experimentation** and **rigorous evaluation** → automatized
- Best implemented by (a **pipeline of**) **scripts** → UNIX command line approach
- Data **exploration** and **rapid prototyping** is key → IPython /Jupyter (see appendix of V03)



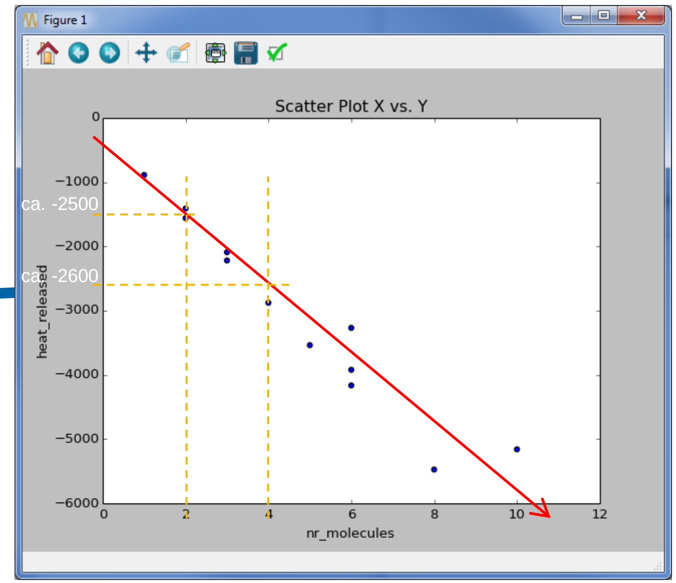
2. MACHINE LEARNING FROM SCRATCH



Carbon and combustion
Continuous example for this section

We need a solid understanding of $< Representation, Evaluation, Optimization >$

Thus, this section explores
- …a «**straight line fit**» as a simplistic model
- …all the details of **how to train** it
- …to get a feeling of **ML apart from libraries**



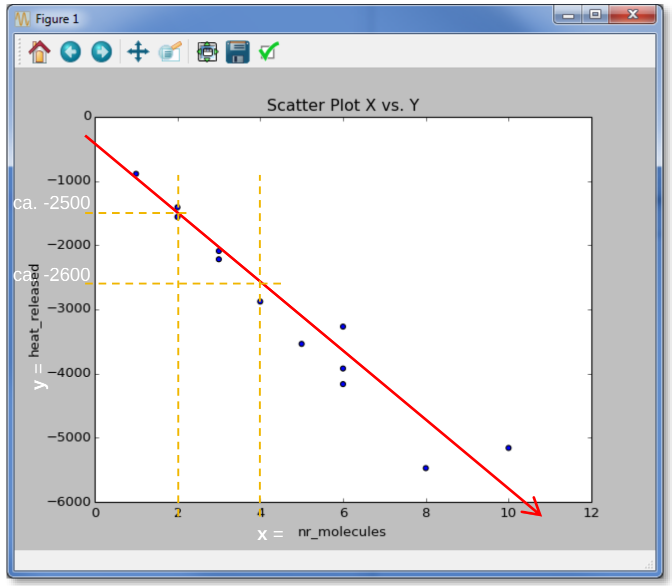
Observations
- The data shows a linear trend
- The gradient is roughly $\frac{-2500+2600}{2-4} = -50$

Conclusions
- Labeled data available → supervised learning
- Contiuous valued output → regression
- Reasonable straight line fit ➔ linear regression could be a first try

How to represent h?
..in the case of univariate linear regression

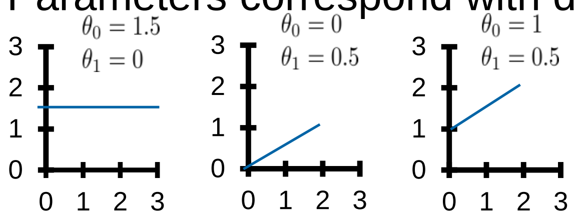
$$h\left(x,\vec{\theta}\right) = \theta_0 + \theta_1 x$$

- θ_0 = intercept, θ_1 = gradient



How to choose parameters θ_0, θ_1?

- Parameters correspond with different fits



Training Set

↓

Learning Algorithm (optimizer)

↓

x: carbon portion in molecule → ***h*** → y: estimated fuel value

Aliases: loss function, objective function $* (-1)$

Cost function J

Driver of the optimization

Zurich University of Applied Sciences

Idea

- **Choose $\boldsymbol{\theta_0}$, $\boldsymbol{\theta_1}$ so** that $\boldsymbol{h}(\boldsymbol{x})$ **is close to** $\boldsymbol{y}$ for the training examples (x, y)
- Let a function $\boldsymbol{J}(\boldsymbol{parameters})$ **number the cost of errors** made for specific parameters

We didn't mention a few parameters here → see below for complete list

$h\left(x, \vec{\theta}\right) = \theta_0 + \theta_1 x$

squared error

$$J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right)^2$$

/2 for mathematical "beauty" (i.e., ease)…

average over all N examples

Precise objective

- **Minimize $\boldsymbol{J}$** w.r.t. θ_0, θ_1
- In terms of V01:
  - $L(\hat{y}, y) = (\hat{y} - y)^2$ is the squared error loss function
  - $J(\theta_0, \theta_1, h, X, Y, L) = E_{emp}\left(h(\vec{\theta}), X, Y, L\right) = \frac{1}{2N} \sum_{i=1}^{N} L\left(h\left(x_i, \vec{\theta}\right), y_i,\right)$ is the **cost of error** with **explicit** respect to **all parameters**: Measured in terms of the empirical error over the training set (X, Y) according to the squared error loss function L for a particular hypothesis h

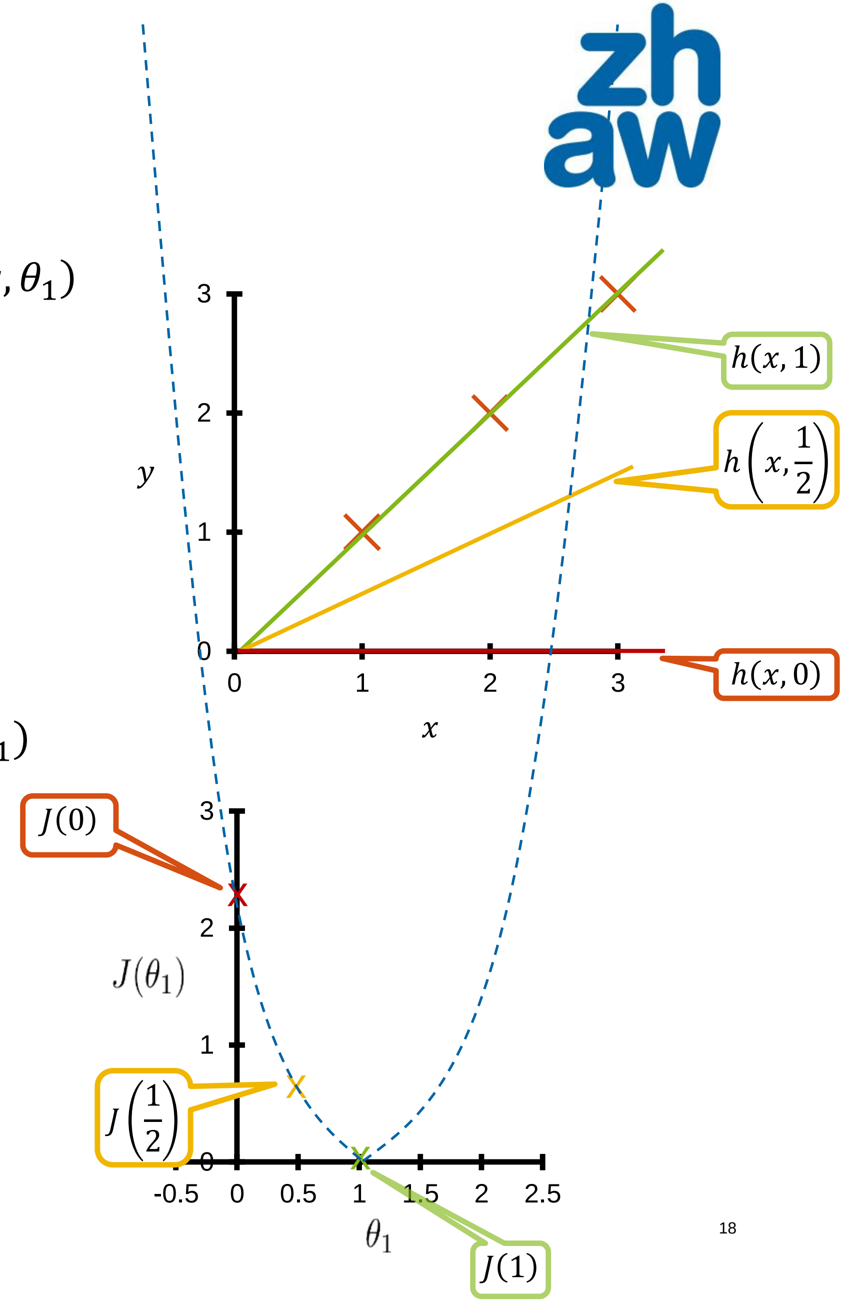
Simplified version to gain intuition
Fixed intercept at (0,0)

- Hypothesis: $h(x, \theta_1) = 0 + \theta_1 x$
- Parameter: θ_1
- Cost: $J(\theta_1) = \frac{1}{2N} \sum_{i=1}^{N} (h(x_i, \theta_1) - y_i)^2$
- Goal: $\underset{\theta_1}{\text{minimize}}\, J(\theta_1)$

Example values of J

- $J(1) = \frac{1}{6} \sum_{i=1}^{3} (h(x_i, 1) - y_i)^2$
  $= \frac{1}{6} \sum_{i=1}^{3} (x_i - y_i)^2$
  $= \frac{1}{6} \left((1-1)^2 + (2-2)^2 + (3-3)^2\right) = 0$
- $J\left(\frac{1}{2}\right) = \cdots = \frac{1}{6}\left(\left(\frac{1}{2} - 1\right)^2 + (1-2)^2 + \left(\frac{3}{2} - 3\right)^2\right)$
  $= \frac{1}{6} \cdot \frac{7}{2} \approx 0.58$
- $J(0) = \cdots = \frac{1}{6}(1^2 + 2^2 + 3^2) = \frac{1}{6} \cdot 14 \approx 2.33$
- …

- $h(x, \theta_1)$
- $J(\theta_1)$



Back to two parameters

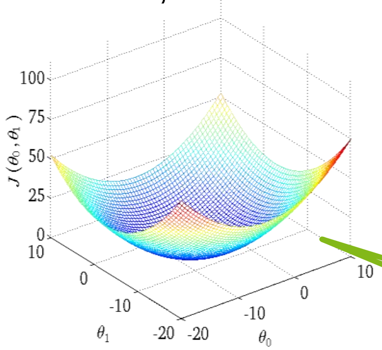
- Hypothesis: $h\left(x,\vec{\theta}\right) = \theta_0 + \theta_1 x$

  ➔ For fixed $\vec{\theta}$, this is a **function of $\boldsymbol{x}$**

- Parameters: θ_0, θ_1

- Cost: $J(\theta_0, \theta_1) = \frac{1}{2N}\sum_{i=1}^{N}\left(h\left(x_i,\vec{\theta}\right) - y_i\right)^2$

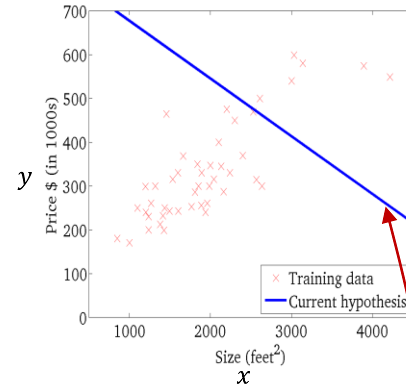
  ➔ For fixed x, this is a **function of $\vec{\boldsymbol{\theta}}$**



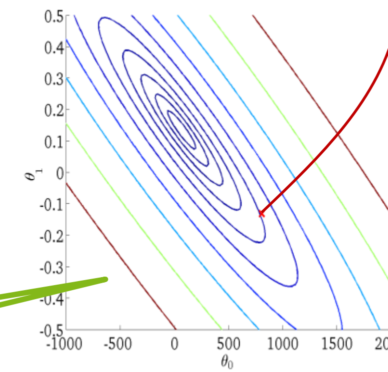
- Goal: $\underset{\theta_0,\theta_1}{\text{minimize}}\, J(\theta_0, \theta_1)$

Example: Housing prices (y) per size (x)

- $h\left(x,\vec{\theta}\right)$



- $J(\theta_0, \theta_1)$



$h(x, 800, -1.5)$

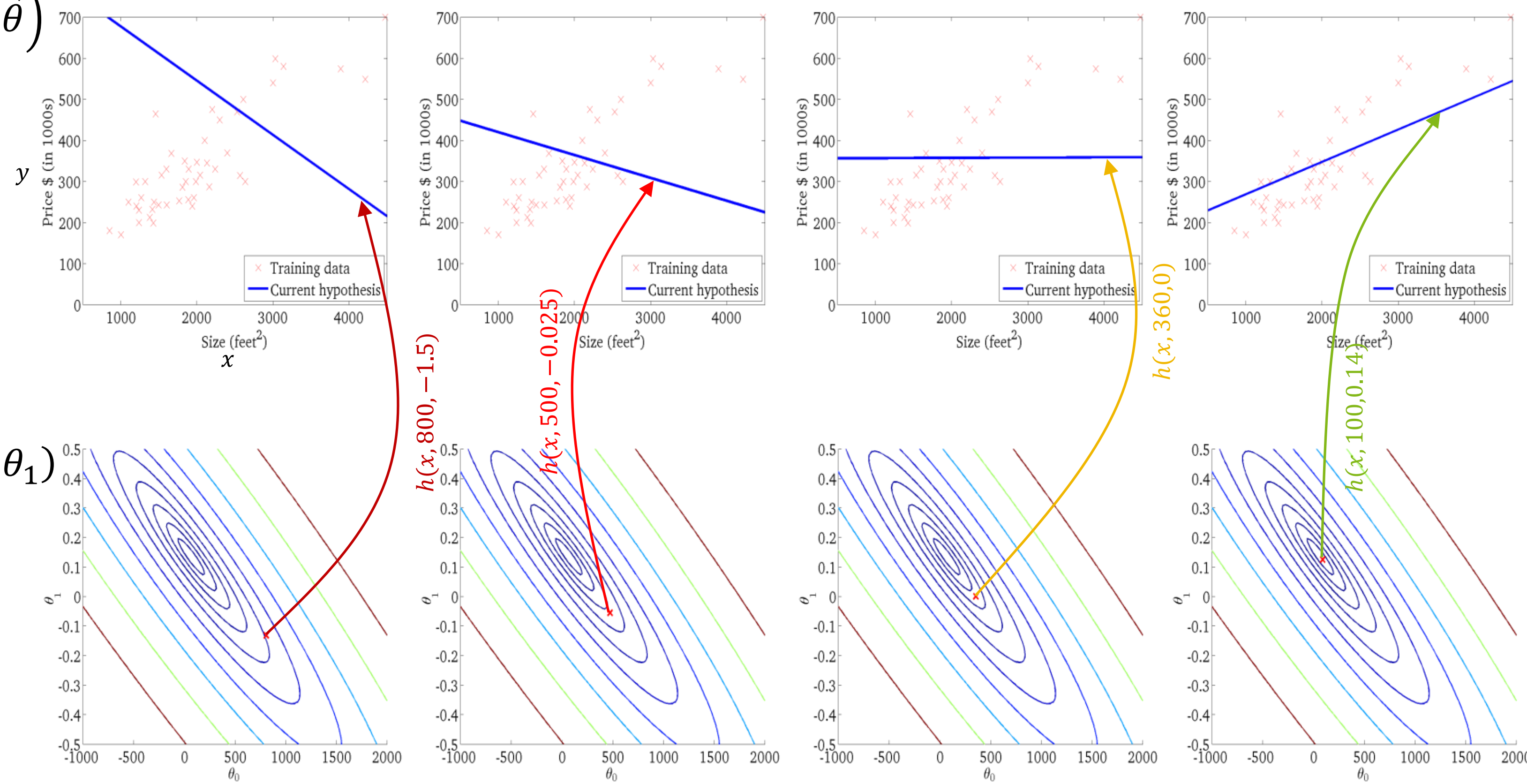
J is a convex (bowl-shaped) function of 2 variables → 3D

…and its 2D visualization as a contour plot

Back to two parameters
contd.

Example: Housing prices (y) per size (x)

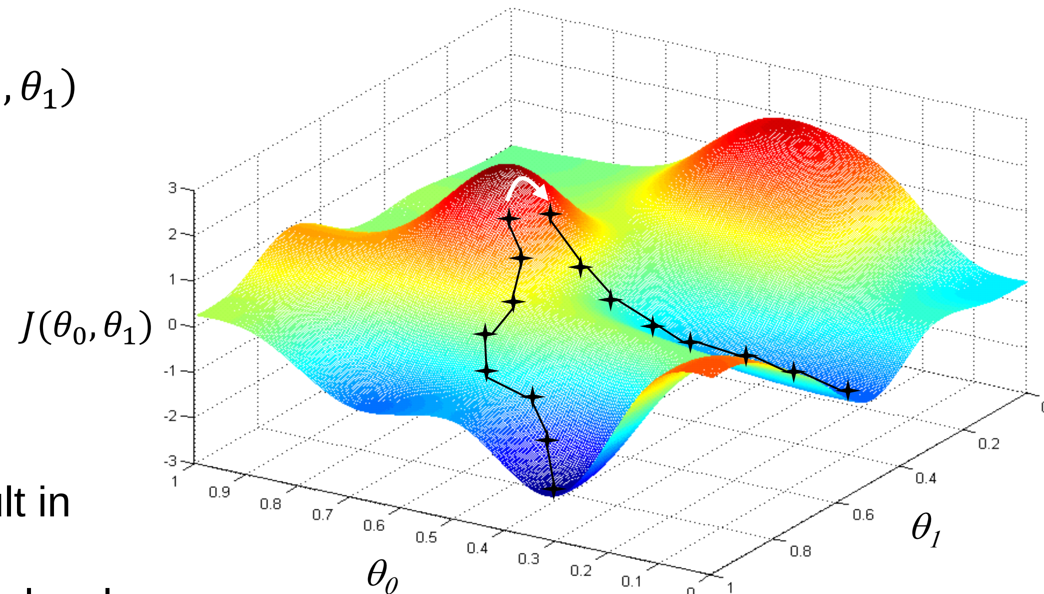
- $h\left(x,\vec{\theta}\right)$
- $J(\theta_0,\theta_1)$



Optimization by gradient descent
Numerical optimization

Have: Some function $J(\theta_0, \theta_1)$
Want: $\underset{\theta_0,\theta_1}{\text{minimize}}\, J(\theta_0, \theta_1)$

Algorithm
- Start with some θ_0, θ_1 (e.g., $(0{,}0)$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
  → Direction: steepest descent
- End: Hopefully at a minimum



Observations
- Small changes in starting point result in different **local minima**
- Assumption: cost surface is smooth, local minima are ok

Gradient descent algorithm

Zurich University of Applied Sciences

Pseudo code for gradient descent

- ```
  repeat until convergence:
      for j:=0..1:
          θ̂_j := θ_j − α · ∂/∂θ_j J(θ_o, θ_1)
      for j:=0..1:
          θ_j := θ̂_j
  ```

- $\frac{\partial}{\partial \theta_j} J(\theta_o, \theta_1)$ is the partial derivative of J w.r.t. θ_j
- $\alpha > 0$ is called the learning rate (it is a data-dependent hyperparameter of the algorithm)
- Important: **simultaneous update**!

```
tmp_0 := θ_0 − α · ∂/∂θ_0 J(θ_o, θ_1)
tmp_1 := θ_1 − α · ∂/∂θ_1 J(θ_o, θ_1)
θ_0 := tmp_0
θ_1 := tmp_1
```

```
θ_0 := θ_0 − α · ∂/∂θ_0 J(θ_o, θ_1)
θ_1 := θ_1 − α · ∂/∂θ_1 J(θ_o, θ_1)
```

Problem: changed $\boldsymbol{\theta_0}$ is already used to estimate new $\boldsymbol{\theta_1}$

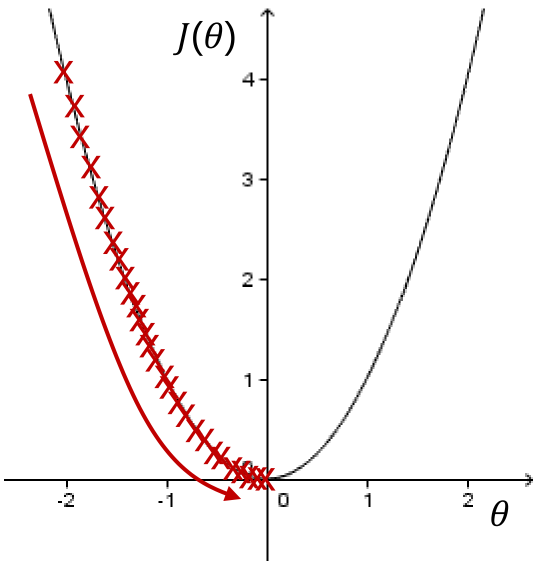
Why not solving it analytical?

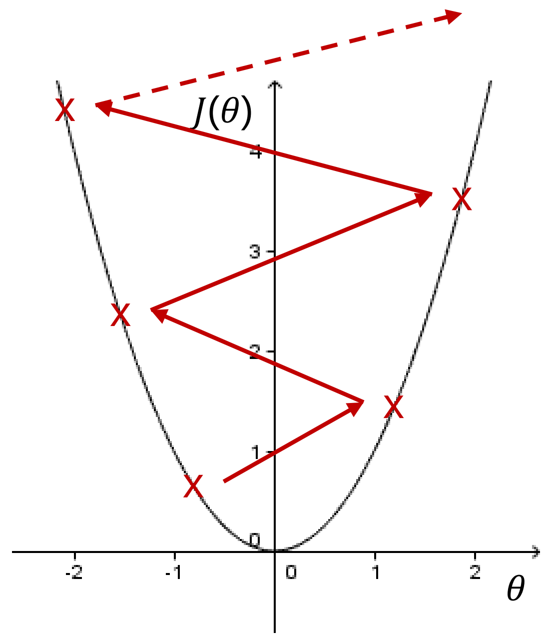
- Numerical optimization **scales better** to larger data sets
- Gradient descent also works for h's without analytical solution (e.g., neural networks)
```

Intuition behind gradient descent formulae (II)
Effect of the learning rate α

- α too **small**
  ➔ gradient descent is **slow**

- α too **large**
  ➔ gradient descent **overshoots** minimum
  ➔ no convergence or even divergence!





Gradient descent for univariate linear regression
Formal overview

Ingredients

- Representation
  - $h\left(x, \vec{\theta}\right) = \theta_0 + \theta_1 x$

How to take the derivative:
→ see appendix

- Evaluation
  - $J(\theta_0, \theta_1) = \frac{1}{2N}\sum_{i=1}^{N}\left(h\left(x_i, \vec{\theta}\right) - y_i\right)^2$
  - $\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1) = \frac{1}{N}\sum_{i=1}^{N}\left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot \overbrace{\frac{\partial}{\partial\theta_0}h\left(x_i, \vec{\theta}\right)}^{=1}$
  - $\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1) = \frac{1}{N}\sum_{i=1}^{N}\left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot \underbrace{\frac{\partial}{\partial\theta_1}h\left(x_i, \vec{\theta}\right)}_{=x_i}$
- Optimization
  - repeat until convergence:
    for j:=0..1:
    $\widehat{\theta_j} \coloneqq \theta_j - \alpha \cdot \frac{\partial}{\partial\theta_j}J(\theta_o, \theta_1)$
    for j:=0..1:
    $\theta_j \coloneqq \widehat{\theta_j}$

repeat until convergence:
$\widehat{\theta_0} \coloneqq \theta_0 - \alpha \cdot \frac{1}{N}\sum_{i=1}^{N}\left(h\left(x_i, \vec{\theta}\right) - y_i\right)$
$\widehat{\theta_1} \coloneqq \theta_1 - \alpha \cdot \frac{1}{N}\sum_{i=1}^{N}\left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot x_i$
$\theta_0 \coloneqq \widehat{\theta_0}$
$\theta_1 \coloneqq \widehat{\theta_1}$

Batch gradient descent: uses all training examples at once (as opposed to stochastic gradient descent, which uses small chunks called "mini-batches"…)

Review

- A **learning solution** needs a **representation**, an **evaluation** function and an **optimizer**

- These can be derived from the formulation of a **well-posed learning problem** as **task**, **performance** measure and training **experience**

- There is **no general solution** to deriving these concrete methods. It is problem (data-) dependent and relies on prior knowledge
- **Valid guides** are the **characteristics of methods** (inductive bias, VC theory), **experience** / **best practices** and **prior knowledge**

- **Gradient descent** is a general-purpose optimizer; implementation details (**simultaneous updates**) and **hyperparameters** are practically very relevant

P02.1: Implementing ML from scratch

Work through exercise P02.1:

- Implement the algorithms derived in this chapter just using the given formal descriptions (i.e., slide 24)
- Reflect on the methods: How transferable are experiences from one data set to the next?
- Reflect on your implementation: What took you the most time? Which part was easy for you?

APPENDIX

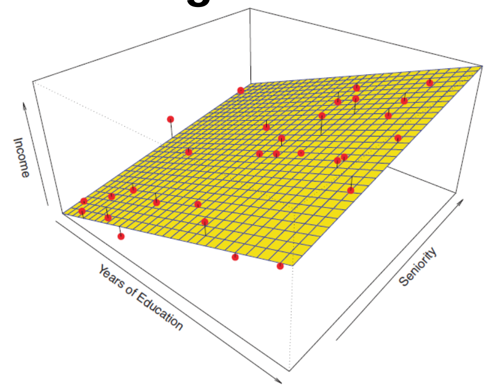
Remark: Different levels of inductive bias

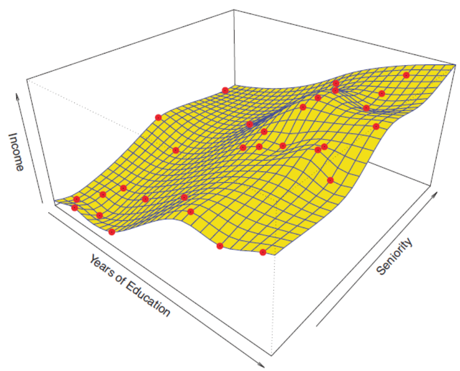
Are there more general forms of prior knowledge that universally guide learning?

application level

fundamental level

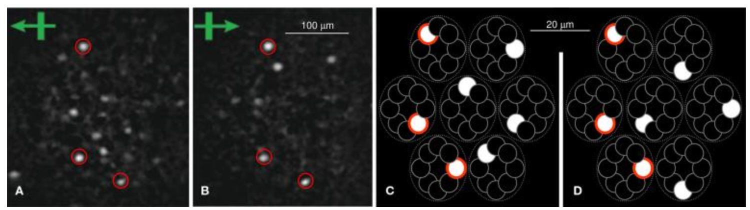
there's a linear relationship between inputs & outputs





the hypothesis space is **smooth**

learn **sparse**, **distributed** representations



Intuition behind gradient descent formulae (I)

Signs and α automatically care for proper *descent*

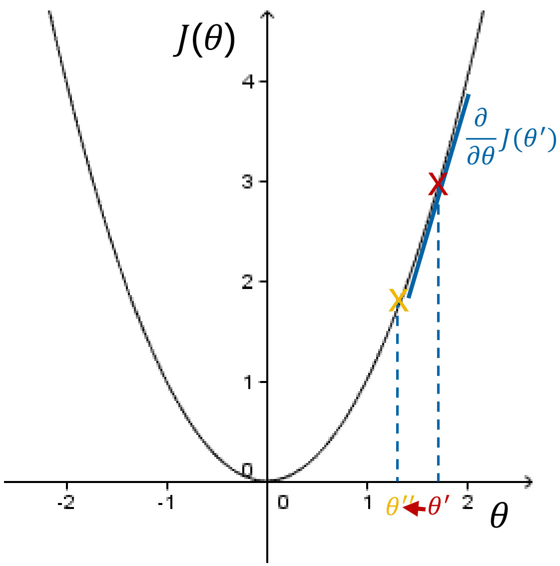
next value current value

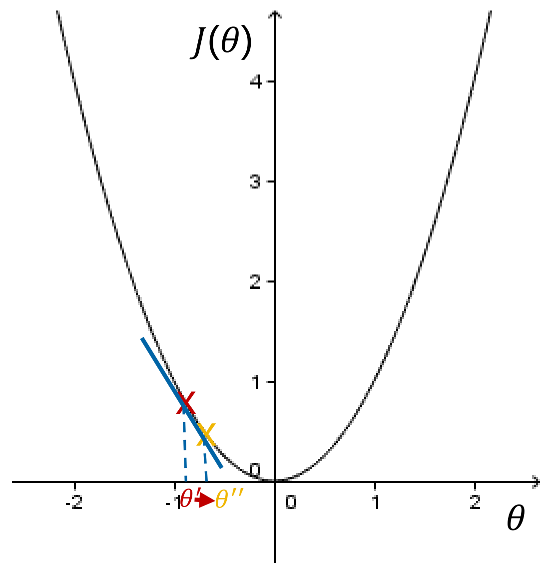
- $\theta'' := \theta' - \underbrace{\alpha}_{} \cdot \underbrace{\frac{\partial}{\partial \theta'} J(\theta')}_{}$

  positive numbers

- $\theta'' := \theta' - \alpha \cdot \underbrace{\frac{\partial}{\partial \theta'} J(\theta')}_{}$

  negative slope





- As we approach the minimum, steps automatically get smaller ➔ α may be fixed over time

Derivative of J w.r.t. θ_j

$$\frac{\partial}{\partial \theta_j} J(\theta_o, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2N} \sum_{i=1}^{N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right)^2 = \sum_{i=1}^{N} \frac{\partial}{\partial \theta_j} \frac{1}{2N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right)^2$$

Chain rule: $f\left(g(x)\right)' = f'\left(g(x)\right) \cdot g'(x)$

$$= \sum_{i=1}^{N} \frac{2}{2N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot \frac{\partial}{\partial \theta_j} \left(h\left(x_i, \vec{\theta}\right) - y_i\right)$$

$$= \sum_{i=1}^{N} \frac{1}{N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot \frac{\partial}{\partial \theta_j} h\left(x_i, \vec{\theta}\right)$$

$$= \begin{cases} j = 0 \quad \longrightarrow \quad \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot 1 \\ j = 1 \quad \longrightarrow \quad \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} \left(h\left(x_i, \vec{\theta}\right) - y_i\right) \cdot x_i \end{cases}$$

Zurich University
of Applied Sciences

Choosing cost functions

Ideal properties of a cost function

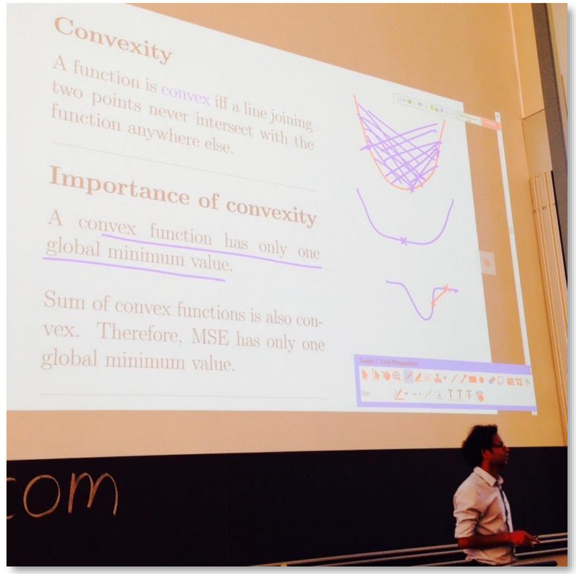
1. Being **easy to optimize** → should be a *convex* function
2. Assigning **equal cost to far and very far** off examples → makes it **robust to outliers**

Cost functions in practice

- MSE (mean-squared error) is almost always used for regression
  → it only exhibits property 1
- Making MSE level off would make the function non-convex
  → when using **MSE**, one has to care for **outliers** during **pre-processing**

➔ Cost function design is important
(because the usual one might not capture the problem well)

➔ …but care has to be taken to make it mathematically sound!



Emti Khan, EPFL, at his introductory ML course during Zurich ML Meetup #18, 25.08.2015

Further reading

- Boyd & Vandenberghe, *«Convex Optimization»*, 2004 → ch. 3
- Bertsekas, *«Convex Optimization Algorithms»*, 2015 → ch. 1
- Chu, *«Machine Learning Done Wrong»*, 2015

Examples of built-to-purpose cost functions
from [Mitchell, 1997], chapter 6.5

Certain well-known cost functions can be justified theoretically using Bayesian reasoning by showing optimality under certain assumptions:

Minimizing *squared error*

- Yields maximum likelihood (ML) hypothesis assuming Gaussian noise on the labels
  Example: Training *linear regression* to fit a straight line

Minimizing *cross entropy*

- Yields ML hypothesis assuming the labels are a probabilistic function of the training examples
- Example: Training a *neural network* to predict probabilities

CMU's Tom Mitchell, author of one of the most instructive machine learning books.

→ see appendix of V03