

# Introduction to Machine Learning

## Session 1: The Basics of Optimization, Probability, and Machine Learning

Benjamin Paaßen

The University of Sydney

IK 2020, Günne

Licensed according to CC-BY-SA 3.0



THE UNIVERSITY OF  
**SYDNEY**

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...)

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...) from **training data**

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...) from **training data** that **generalize** to **test data**.

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...) from **training data** that **generalize** to **test data**.

If you show me a data set of example inputs and outputs  $(x_1, y_1), \dots, (x_m, y_m)$ ,

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...) from **training data** that **generalize** to **test data**.

If you show me a data set of example inputs and outputs  $(x_1, y_1), \dots, (x_m, y_m)$ , machine learning means to **automatically** find a function  $f$ ,

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...) from **training data** that **generalize** to **test data**.

If you show me a data set of example inputs and outputs  $(x_1, y_1), \dots, (x_m, y_m)$ , machine learning means to **automatically** find a function  $f$ , such that  $f(x_i) \approx y_i$  for all examples

# What is Machine Learning?

## Definition (roughly)

Machine learning is concerned with **methods** to **automatically** discover **patterns** (rules, regularities, clusters, ...) from **training data** that **generalize** to **test data**.

If you show me a data set of example inputs and outputs  $(x_1, y_1), \dots, (x_m, y_m)$ , machine learning means to **automatically** find a function  $f$ , such that  $f(x_i) \approx y_i$  for all examples and for new, unseen data (**generalization**).

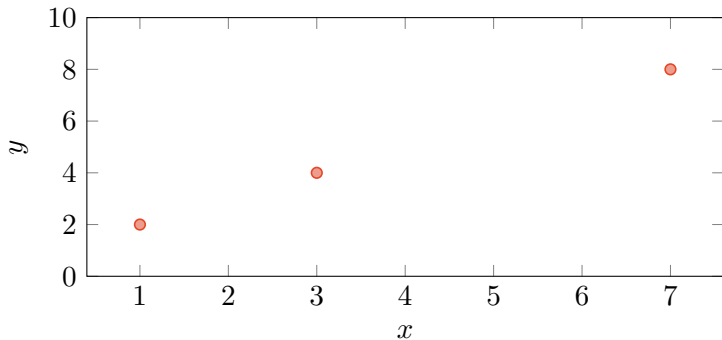


## Example: Regression

**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?

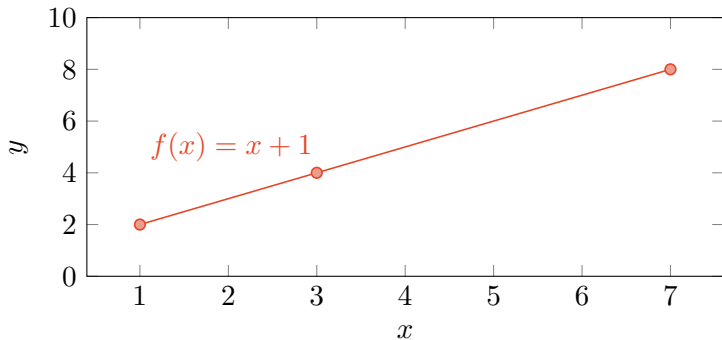
## Example: Regression

**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?



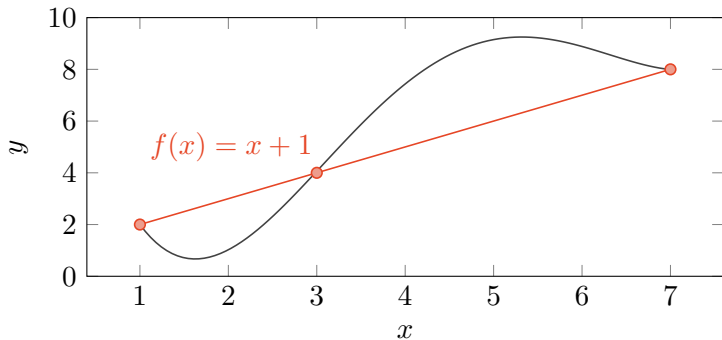
## Example: Regression

**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?



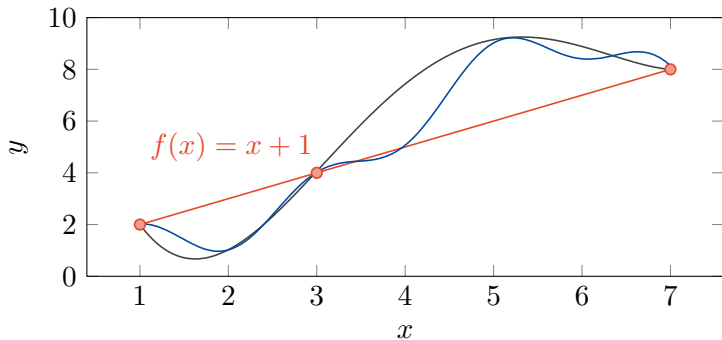
## Example: Regression

**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?



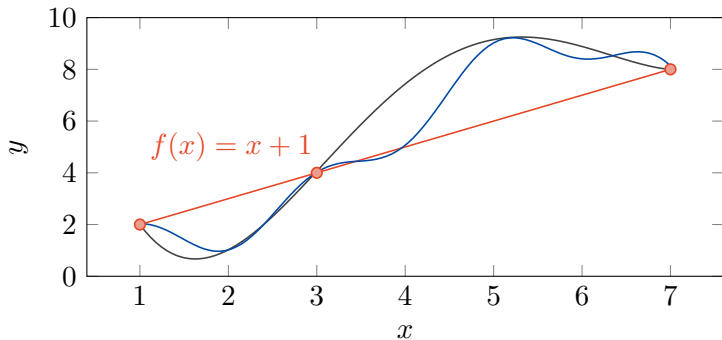
## Example: Regression

**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?



## Example: Regression

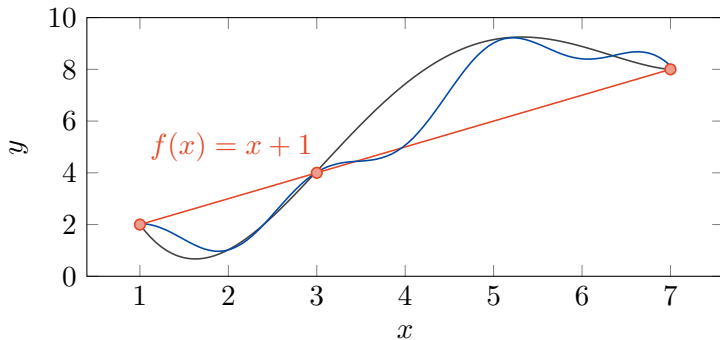
**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?



Which one is best?

## Example: Regression

**Example:**  $(1, 2), (3, 4), (7, 8)$ . What is  $f$ ?



Which one is best?  $\Rightarrow$  Depends on what **generalizes** to new data

## Why bother with learning about ML?

- ▶ ML is a useful toolbox for many tasks (**method** field, similar to statistics)



## Why bother with learning about ML?

- ▶ ML is a useful toolbox for many tasks (**method** field, similar to statistics)
- ▶ ML is a hype in research (Crew, 2019), business, and society more broadly

# Why bother with learning about ML?

- ▶ ML is a useful toolbox for many tasks (**method** field, similar to statistics)
- ▶ ML is a hype in research (Crew, 2019), business, and society more broadly - and it's good to know capabilities and **limitations**

## Learning objectives of this course

- ▶ **Conceptual knowledge:** Most important ML concepts and how they are related

## Learning objectives of this course

- ▶ **Conceptual knowledge:** Most important ML concepts and how they are related
- ▶ **Operational knowledge:** How to do ML (at least on a high level; or with voluntary programming tasks)

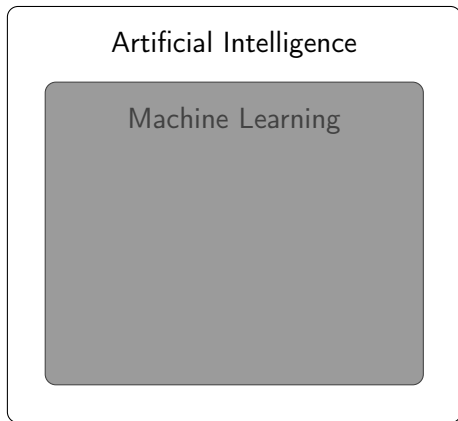
## Learning objectives of this course

- ▶ **Conceptual knowledge:** Most important ML concepts and how they are related
- ▶ **Operational knowledge:** How to do ML (at least on a high level; or with voluntary programming tasks)
- ▶ **ML literacy:** De-mystifying ML and gauging the capabilities of ML approaches

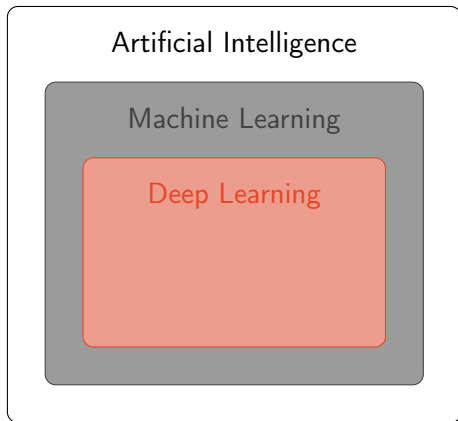
# Overview of ML

Artificial Intelligence

# Overview of ML

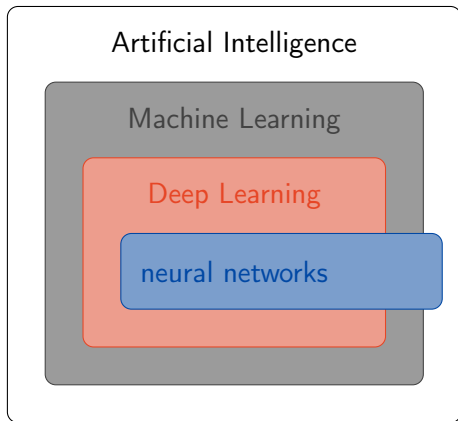


# Overview of ML

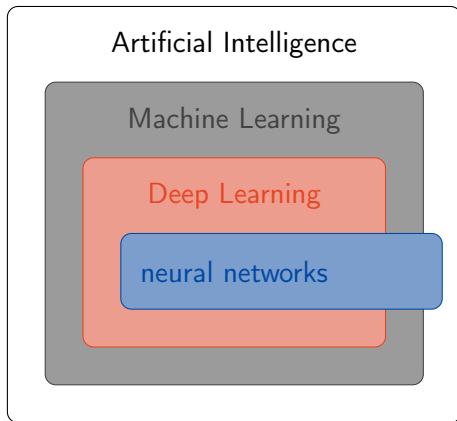




# Overview of ML



# Overview of ML



ca. 290.000 papers since 2015

ca. 630.000 papers since 2015

ca. 73.000 papers since 2015

ca. 300.000 papers since 2015

acc. to google scholar

# Structure

1. Basic concepts: Functions, Learning algorithms, Learning via optimization, linear regression (as example), regularization, probability theory, machine learning theory, how to design a ML experiment

# Structure

1. Basic concepts: Functions, Learning algorithms, Learning via optimization, linear regression (as example), regularization, probability theory, machine learning theory, how to design a ML experiment
2. “Classic” ML Tasks and Methods: The distance perspective on ML, Regression, Classification, Dimensionality Reduction, Clustering

# Structure

1. Basic concepts: Functions, Learning algorithms, Learning via optimization, linear regression (as example), regularization, probability theory, machine learning theory, how to design a ML experiment
2. “Classic” ML Tasks and Methods: The distance perspective on ML, Regression, Classification, Dimensionality Reduction, Clustering
3. “Modern” ML and neural networks: Neural network modules, recipes for neural networks, adversarial attacks

# Structure

1. Basic concepts: Functions, Learning algorithms, Learning via optimization, linear regression (as example), regularization, probability theory, machine learning theory, how to design a ML experiment
2. “Classic” ML Tasks and Methods: The distance perspective on ML, Regression, Classification, Dimensionality Reduction, Clustering
3. “Modern” ML and neural networks: Neural network modules, recipes for neural networks, adversarial attacks
4. Reinforcement learning and ethics

# Structure

1. Basic concepts: Functions, Learning algorithms, Learning via optimization, linear regression (as example), regularization, probability theory, machine learning theory, how to design a ML experiment
2. “Classic” ML Tasks and Methods: The distance perspective on ML, Regression, Classification, Dimensionality Reduction, Clustering
3. “Modern” ML and neural networks: Neural network modules, recipes for neural networks, adversarial attacks
4. Reinforcement learning and ethics

Unfortunately **not** covered: Bayesian ML, Graphical Models, Causality, random forests, ...

# Structure

1. Basic concepts: Functions, Learning algorithms, Learning via optimization, linear regression (as example), regularization, probability theory, machine learning theory, how to design a ML experiment
2. “Classic” ML Tasks and Methods: The distance perspective on ML, Regression, Classification, Dimensionality Reduction, Clustering
3. “Modern” ML and neural networks: Neural network modules, recipes for neural networks, adversarial attacks
4. Reinforcement learning and ethics

Unfortunately **not** covered: Bayesian ML, Graphical Models, Causality, random forests, ...

Note: One homework task for each session



# Basic Mathematical Concepts



THE UNIVERSITY OF  
**SYDNEY**

# Functions

## Definition: Function

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be sets (e.g. the set of all possible integers).

# Functions

## Definition: Function

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be sets (e.g. the set of all possible integers). Then, a **function**  $f$  is a set of tuples  $(x, y)$  where  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$ , such that no  $x$  occurs in two tuples.

# Functions

## Definition: Function

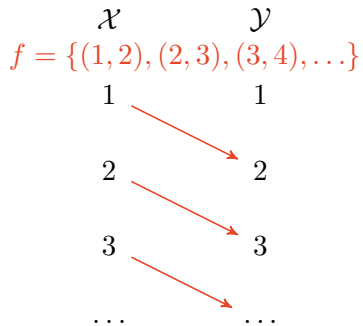
Let  $\mathcal{X}$  and  $\mathcal{Y}$  be sets (e.g. the set of all possible integers). Then, a **function**  $f$  is a set of tuples  $(x, y)$  where  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$ , such that no  $x$  occurs in two tuples.

$\mathcal{X}$	$\mathcal{Y}$
1	1
2	2
3	3
...	...

# Functions

## Definition: Function

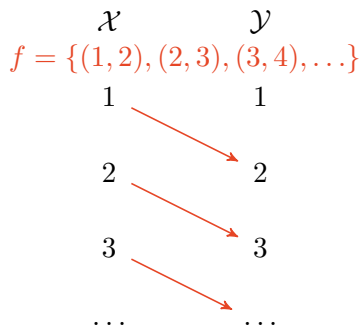
Let  $\mathcal{X}$  and  $\mathcal{Y}$  be sets (e.g. the set of all possible integers). Then, a **function**  $f$  is a set of tuples  $(x, y)$  where  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$ , such that no  $x$  occurs in two tuples.



# Functions

## Definition: Function

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be sets (e.g. the set of all possible integers). Then, a **function**  $f$  is a set of tuples  $(x, y)$  where  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$ , such that no  $x$  occurs in two tuples.

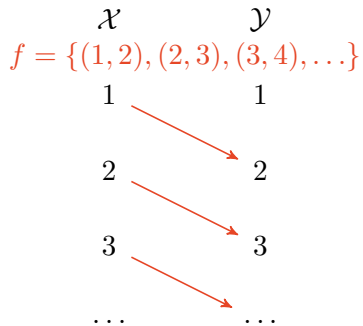


We define  $f(x)$  as  $y$  such that  $(x, y) \in f$ .

# Functions

## Definition: Function

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be sets (e.g. the set of all possible integers). Then, a **function**  $f$  is a set of tuples  $(x, y)$  where  $x \in \mathcal{X}$ ,  $y \in \mathcal{Y}$ , such that no  $x$  occurs in two tuples.



We define  $f(x)$  as  $y$  such that  $(x, y) \in f$ .

Intuitively, a function is some kind of **machine** or **program** which returns a deterministic **output** for some **input**.

# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets.



# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets. Then, we call any finite set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  a **dataset** from  $\mathcal{X} \times \mathcal{Y}$ .

# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets. Then, we call any finite set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  a **dataset** from  $\mathcal{X} \times \mathcal{Y}$ .

A **learning algorithm**  $\mathcal{A}$  is a function that maps any data set  $\mathcal{D}$  to a function  $f_{\mathcal{D}} : \mathcal{X} \rightarrow \mathcal{Y}$ .

# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets. Then, we call any finite set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  a **dataset** from  $\mathcal{X} \times \mathcal{Y}$ .

A **learning algorithm**  $\mathcal{A}$  is a function that maps any data set  $\mathcal{D}$  to a function  $f_{\mathcal{D}} : \mathcal{X} \rightarrow \mathcal{Y}$ .

We also call the output  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$  a **model** on  $\mathcal{D}$  according to  $\mathcal{A}$ .

# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets. Then, we call any finite set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  a **dataset** from  $\mathcal{X} \times \mathcal{Y}$ .

A **learning algorithm**  $\mathcal{A}$  is a function that maps any data set  $\mathcal{D}$  to a function  $f_{\mathcal{D}} : \mathcal{X} \rightarrow \mathcal{Y}$ .

We also call the output  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$  a **model** on  $\mathcal{D}$  according to  $\mathcal{A}$ .

- Intuitively, a learning algorithm estimates the relationship between the sets  $\mathcal{X}$  and  $\mathcal{Y}$  in form of a function.

# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets. Then, we call any finite set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  a **dataset** from  $\mathcal{X} \times \mathcal{Y}$ .

A **learning algorithm**  $\mathcal{A}$  is a function that maps any data set  $\mathcal{D}$  to a function  $f_{\mathcal{D}} : \mathcal{X} \rightarrow \mathcal{Y}$ .

We also call the output  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$  a **model** on  $\mathcal{D}$  according to  $\mathcal{A}$ .

- ▶ Intuitively, a learning algorithm estimates the relationship between the sets  $\mathcal{X}$  and  $\mathcal{Y}$  in form of a function.
- ▶ A good algorithm ensures that for all  $(x, y) \in \mathcal{D}$ ,  $f_{\mathcal{D}}(x) \approx y$ .

# Learning Algorithms

## Definition: Learning algorithm

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be some sets. Then, we call any finite set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  a **dataset** from  $\mathcal{X} \times \mathcal{Y}$ .

A **learning algorithm**  $\mathcal{A}$  is a function that maps any data set  $\mathcal{D}$  to a function  $f_{\mathcal{D}} : \mathcal{X} \rightarrow \mathcal{Y}$ .

We also call the output  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$  a **model** on  $\mathcal{D}$  according to  $\mathcal{A}$ .

- ▶ Intuitively, a learning algorithm estimates the relationship between the sets  $\mathcal{X}$  and  $\mathcal{Y}$  in form of a function.
- ▶ A good algorithm ensures that for all  $(x, y) \in \mathcal{D}$ ,  $f_{\mathcal{D}}(x) \approx y$ .
- ▶ ... **and** that this holds for **new data** as well

## Learning Algorithms Illustration

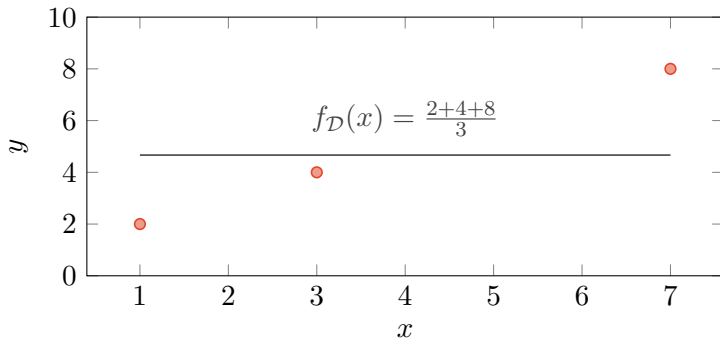
The simplest possible baseline algorithm: Always return the mean of the outputs, i.e.:

$$\mathcal{A}(\mathcal{D}) = f_{\mathcal{D}} \text{ with } f_{\mathcal{D}}(x) = \frac{1}{|\mathcal{D}|} \cdot \sum_{(x,y) \in \mathcal{D}} y$$

## Learning Algorithms Illustration

The simplest possible baseline algorithm: Always return the mean of the outputs, i.e.:

$$\mathcal{A}(\mathcal{D}) = f_{\mathcal{D}} \text{ with } f_{\mathcal{D}}(x) = \frac{1}{|\mathcal{D}|} \cdot \sum_{(x,y) \in \mathcal{D}} y$$

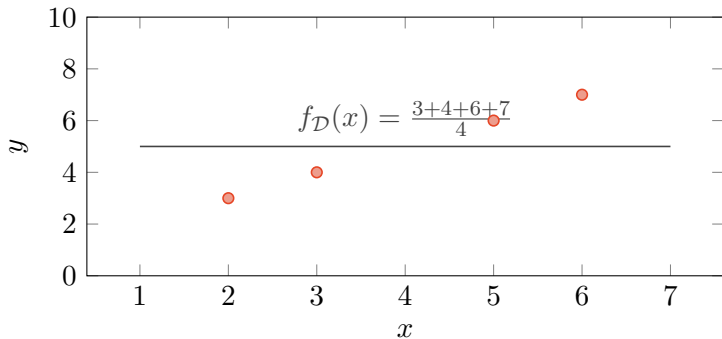




## Learning Algorithms Illustration

The simplest possible baseline algorithm: Always return the mean of the outputs, i.e.:

$$\mathcal{A}(\mathcal{D}) = f_{\mathcal{D}} \text{ with } f_{\mathcal{D}}(x) = \frac{1}{|\mathcal{D}|} \cdot \sum_{(x,y) \in \mathcal{D}} y$$



# Loss function

## Definition: Loss function

A **loss function** or **error function**  $\ell$  is a function that maps a **dataset**  $\mathcal{D}$  and a **model**  $f$  to some real number  $\ell(\mathcal{D}, f)$ .

# Loss function

## Definition: Loss function

A **loss function** or **error function**  $\ell$  is a function that maps a **dataset**  $\mathcal{D}$  and a **model**  $f$  to some real number  $\ell(\mathcal{D}, f)$ .

- Intuitively,  $\ell(\mathcal{D}, f)$  tells us how badly  $f$  reproduces the data in  $\mathcal{D}$

# Loss function

## Definition: Loss function

A **loss function** or **error function**  $\ell$  is a function that maps a **dataset**  $\mathcal{D}$  and a **model**  $f$  to some real number  $\ell(\mathcal{D}, f)$ .

- ▶ Intuitively,  $\ell(\mathcal{D}, f)$  tells us how badly  $f$  reproduces the data in  $\mathcal{D}$
- ▶ Most common example: Root mean square error (RMSE):

$$\ell_{\text{RMSE}}(\mathcal{D}, f) = \sqrt{\frac{1}{|\mathcal{D}|} \cdot \sum_{(x,y) \in \mathcal{D}} (y - f(x))^2} \quad (1)$$

# Loss function

## Definition: Loss function

A **loss function** or **error function**  $\ell$  is a function that maps a **dataset**  $\mathcal{D}$  and a **model**  $f$  to some real number  $\ell(\mathcal{D}, f)$ .

- ▶ Intuitively,  $\ell(\mathcal{D}, f)$  tells us how badly  $f$  reproduces the data in  $\mathcal{D}$
- ▶ Most common example: Root mean square error (RMSE):

$$\ell_{\text{RMSE}}(\mathcal{D}, f) = \sqrt{\frac{1}{|\mathcal{D}|} \cdot \sum_{(x,y) \in \mathcal{D}} (y - f(x))^2} \quad (1)$$

# Loss function

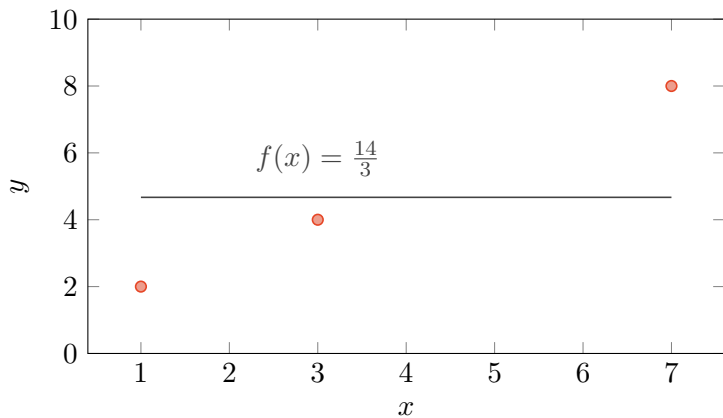
## Definition: Loss function

A **loss function** or **error function**  $\ell$  is a function that maps a **dataset**  $\mathcal{D}$  and a **model**  $f$  to some real number  $\ell(\mathcal{D}, f)$ .

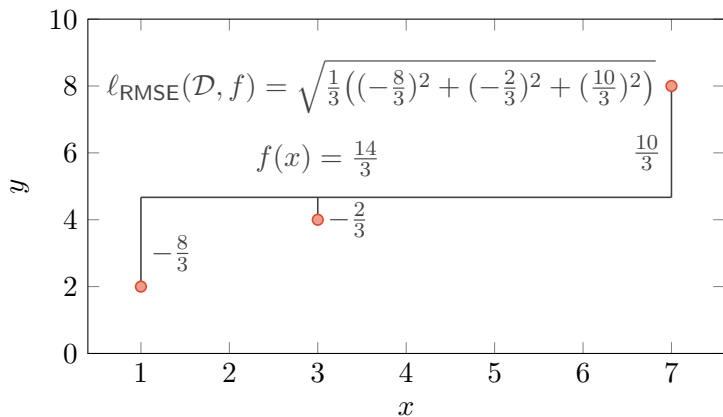
- ▶ Intuitively,  $\ell(\mathcal{D}, f)$  tells us how badly  $f$  reproduces the data in  $\mathcal{D}$
- ▶ Most common example: Root mean square error (RMSE):

$$\ell_{\text{RMSE}}(\mathcal{D}, f) = \sqrt{\frac{1}{|\mathcal{D}|} \cdot \sum_{(x,y) \in \mathcal{D}} (y - f(x))^2} \quad (1)$$

## Illustration: RMSE

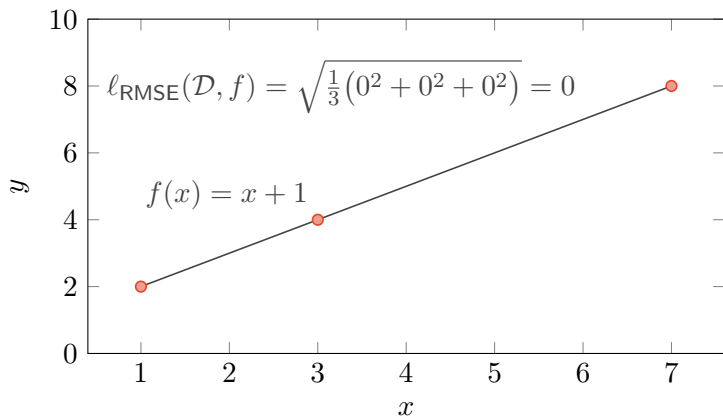


## Illustration: RMSE





## Illustration: RMSE



# Optimization



THE UNIVERSITY OF  
**SYDNEY**

## Learning as Loss Minimization

- ▶ Most learning algorithms  $\mathcal{A}$  are based on **loss minimization**, i.e.:  $\mathcal{A}(\mathcal{D})$  is defined as a model  $f$  that minimizes some loss  $\ell(\mathcal{D}, f)$

# Learning as Loss Minimization

- ▶ Most learning algorithms  $\mathcal{A}$  are based on **loss minimization**, i.e.:  $\mathcal{A}(\mathcal{D})$  is defined as a model  $f$  that minimizes some loss  $\ell(\mathcal{D}, f)$
- ▶ In other words: We want to solve the **optimization problem**

$$\min_{f \in \mathcal{F}} \ell(\mathcal{D}, f) \tag{2}$$

# Learning as Loss Minimization

- ▶ Most learning algorithms  $\mathcal{A}$  are based on **loss minimization**, i.e.:  $\mathcal{A}(\mathcal{D})$  is defined as a model  $f$  that minimizes some loss  $\ell(\mathcal{D}, f)$
- ▶ In other words: We want to solve the **optimization problem**

$$\min_{f \in \mathcal{F}} \ell(\mathcal{D}, f) \quad (2)$$

- ▶ Different algorithms differ in **loss**, **optimization strategy**, and **model class**  $\mathcal{F}$

# Learning as Loss Minimization

- ▶ Most learning algorithms  $\mathcal{A}$  are based on **loss minimization**, i.e.:  $\mathcal{A}(\mathcal{D})$  is defined as a model  $f$  that minimizes some loss  $\ell(\mathcal{D}, f)$
- ▶ In other words: We want to solve the **optimization problem**

$$\min_{f \in \mathcal{F}} \ell(\mathcal{D}, f) \quad (2)$$

- ▶ Different algorithms differ in **loss**, **optimization strategy**, and **model class**  $\mathcal{F}$
- ▶ Focus here mostly on model classes, but optimization strategies are a research field of their own (Paaßen 2019)

# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$



# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$

# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.

# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

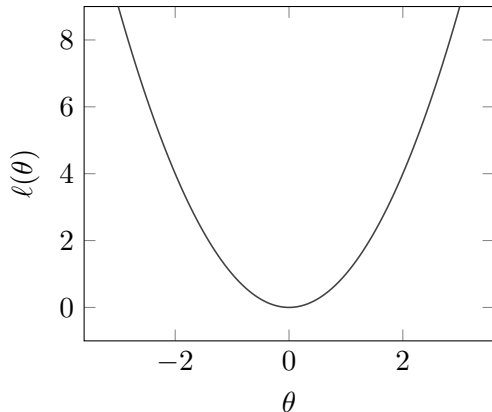
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$



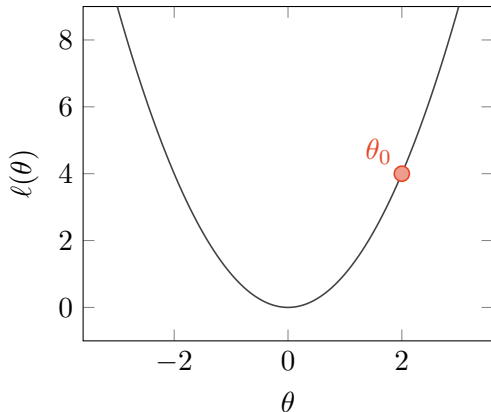
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$



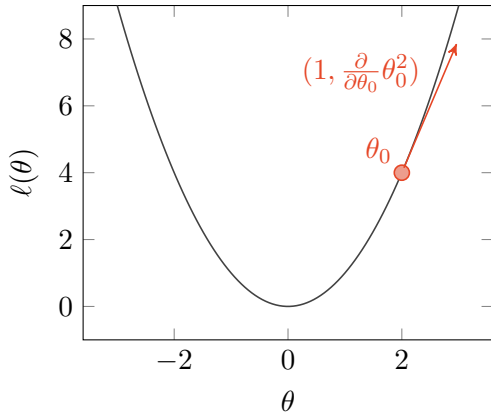
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$



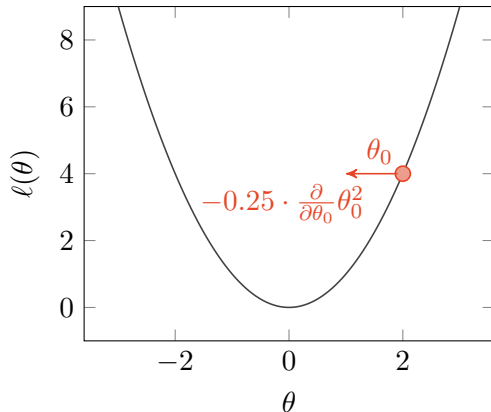
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$



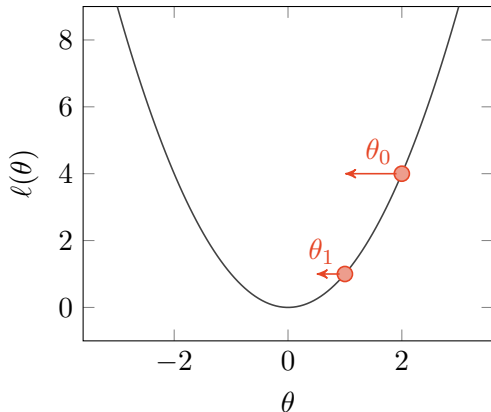
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$





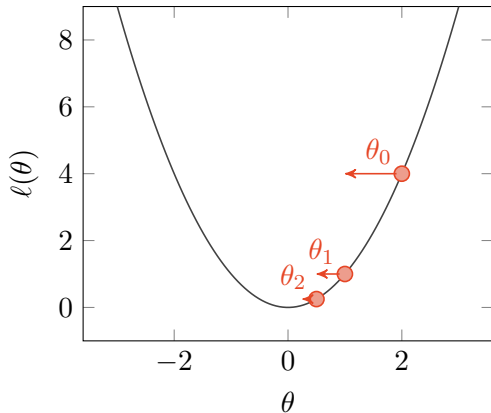
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$



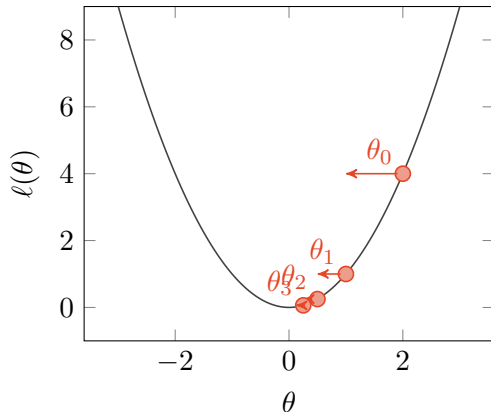
# Gradient Descent

Optimization problem:

$$\min_{\theta \in \mathbb{R}} \ell(\theta)$$

1. Start with some initial  $\theta_0$
2. Go slightly 'down' the function:  
$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{\partial}{\partial \theta_t} \ell(\theta_t)$$
3. Increase  $t$  by 1.
4. Repeat 2-3 until gradient is very small.

$$\ell(\theta) = \theta^2; \frac{\partial}{\partial \theta} \ell(\theta) = 2\theta$$



## Example: Constant Functions

1. Consider the model class of constant functions:

$$\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$$

## Example: Constant Functions

1. Consider the model class of constant functions:

$$\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$$

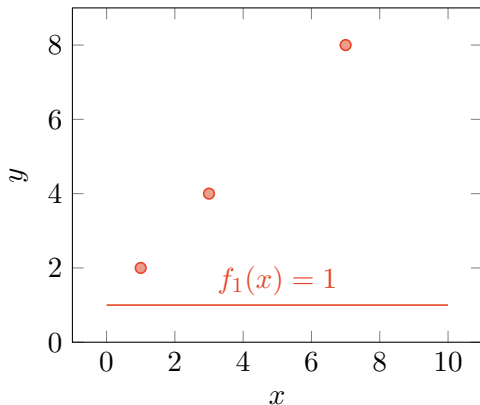
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss

## Example: Constant Functions

1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent

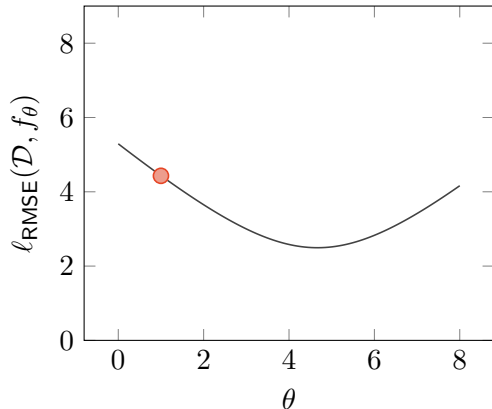
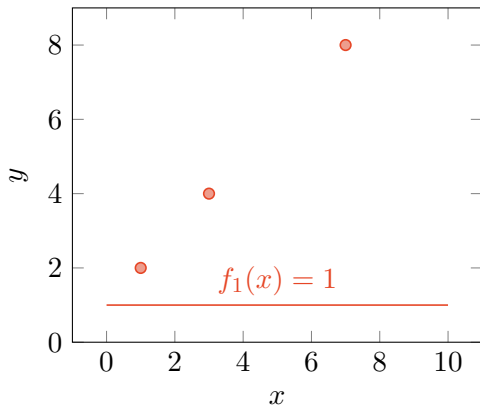
## Example: Constant Functions

1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent



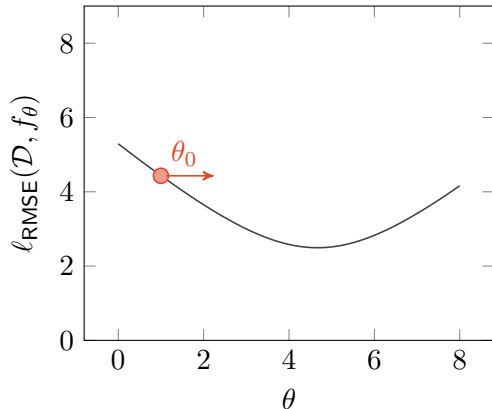
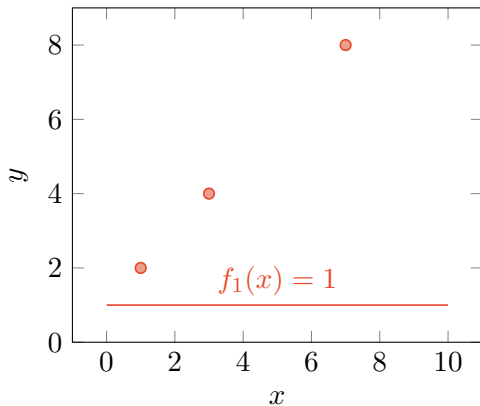
## Example: Constant Functions

1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent



## Example: Constant Functions

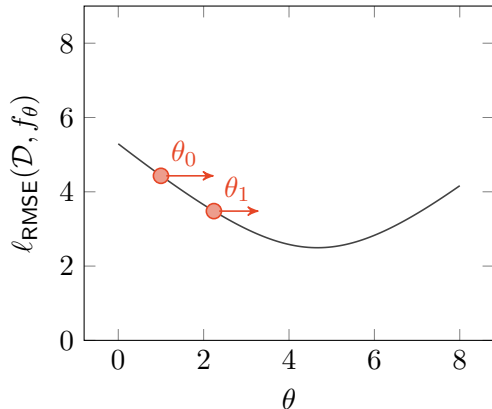
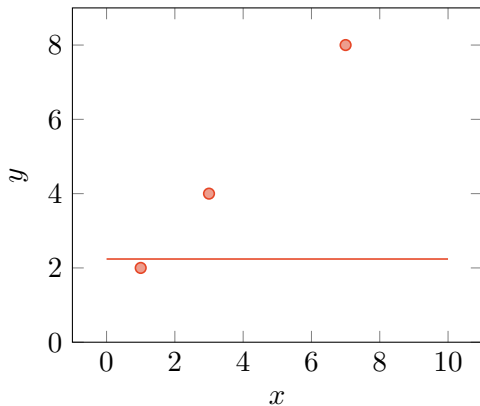
1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent





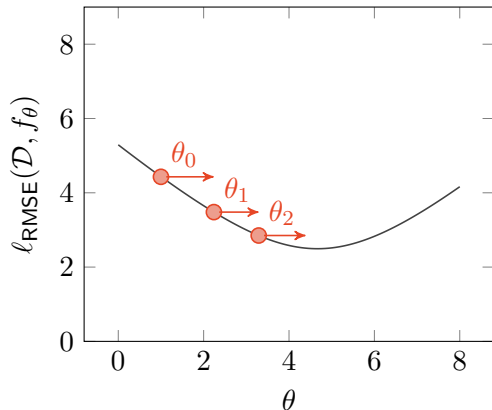
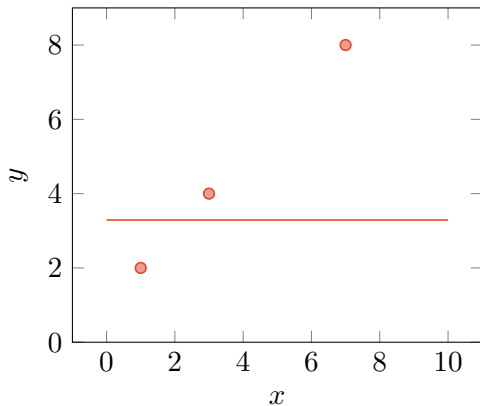
## Example: Constant Functions

1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent



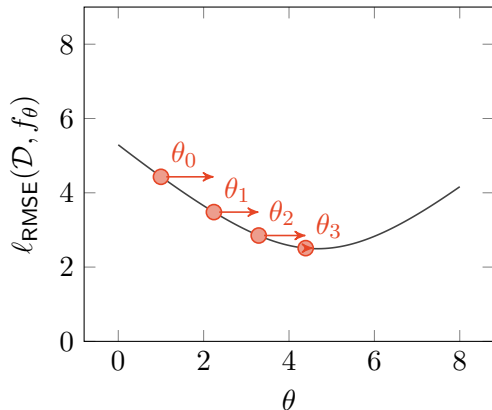
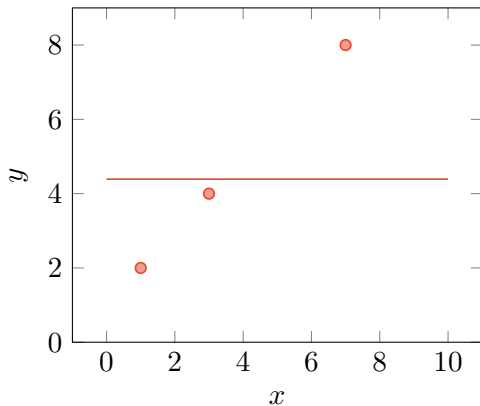
## Example: Constant Functions

1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent



## Example: Constant Functions

1. Consider the model class of constant functions:  
 $\mathcal{F} = \{f_\theta | f_\theta(x) = \theta \text{ for all } x \in \mathcal{X} \text{ and some } \theta \in \mathbb{R}\}$
2. Consider the data set  $\mathcal{D} = \{(1, 2), (3, 4), (7, 8)\}$  and RMSE loss
3. Optimize  $\theta$  via gradient descent



## Example 2: Cosine Functions

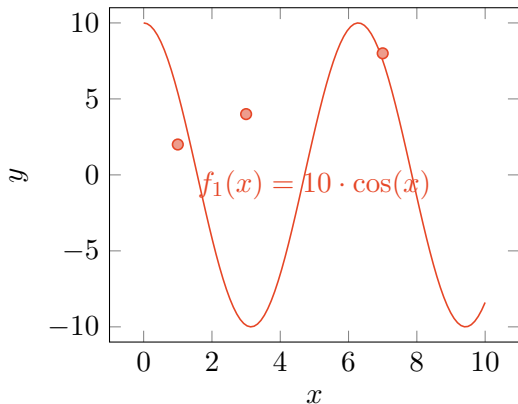
1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$

## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before

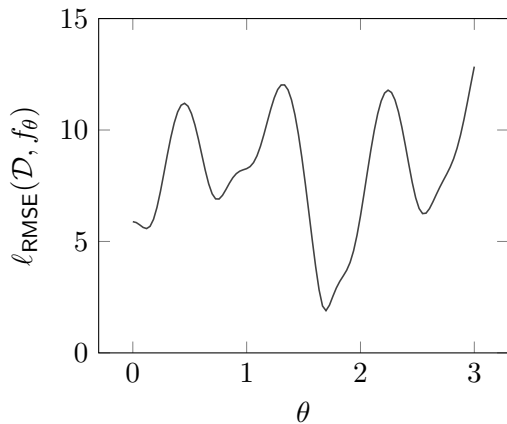
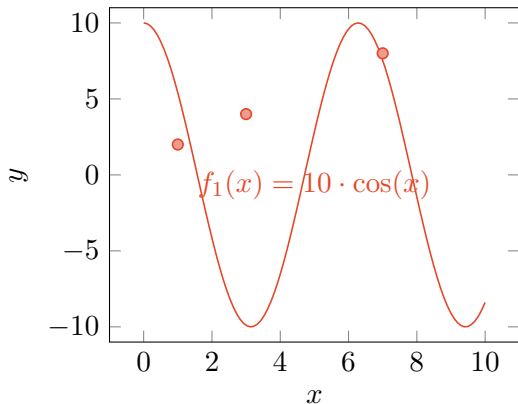
## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before



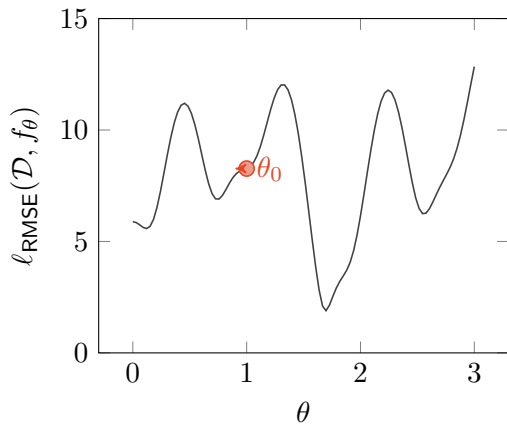
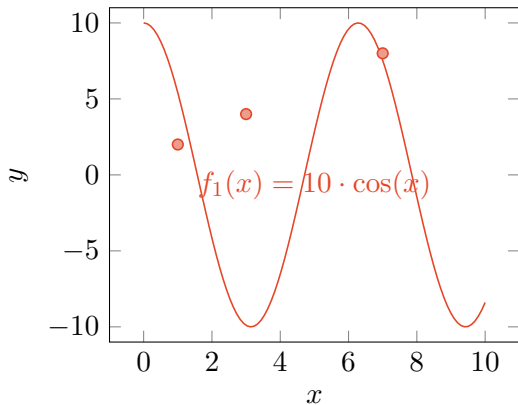
## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before



## Example 2: Cosine Functions

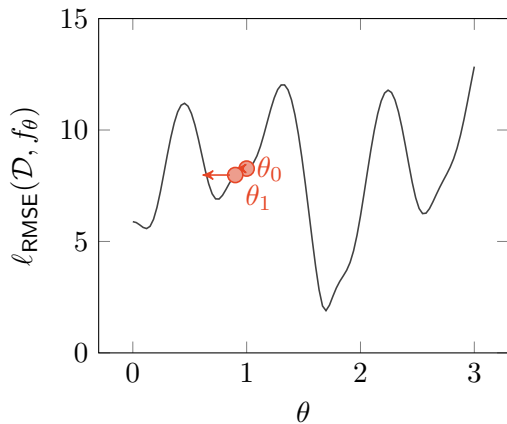
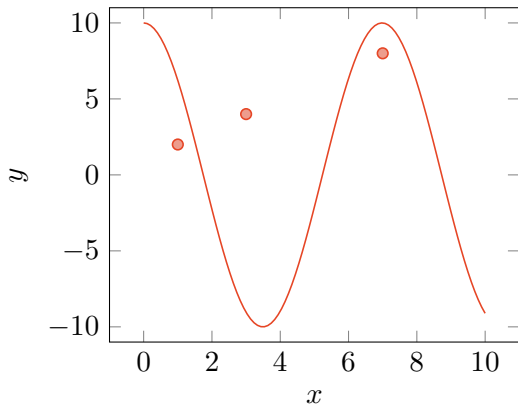
1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before





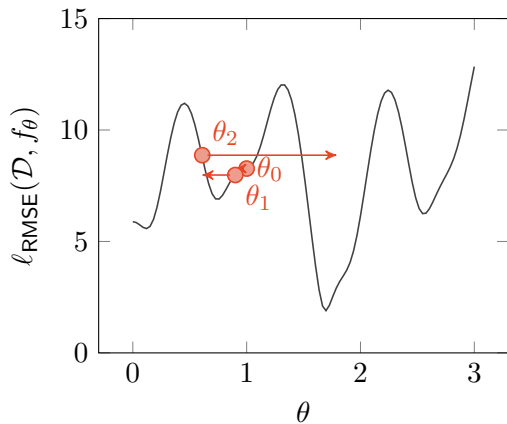
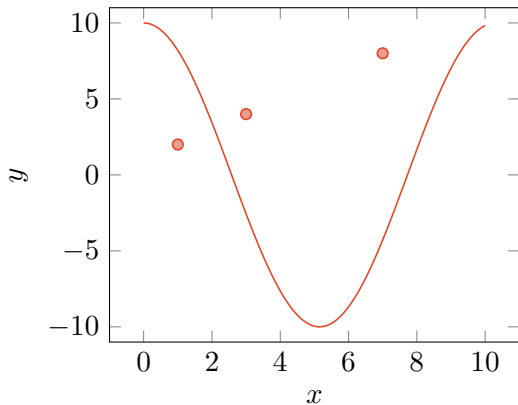
## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before



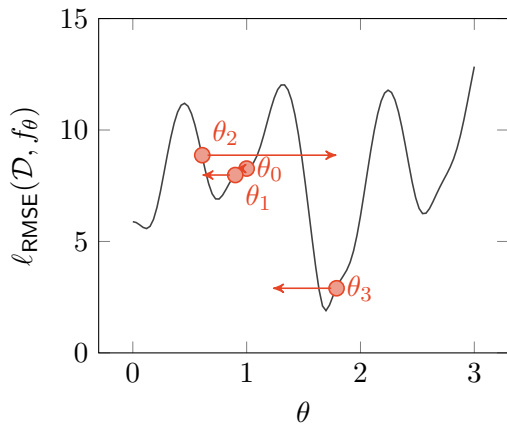
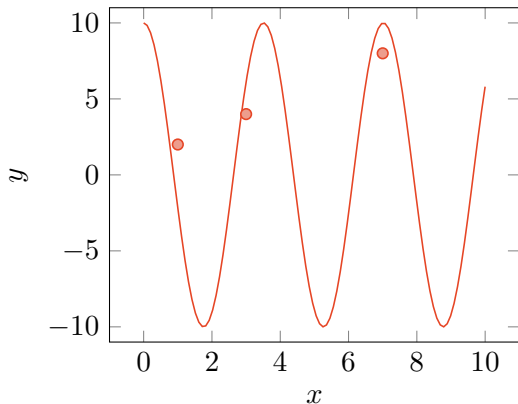
## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before



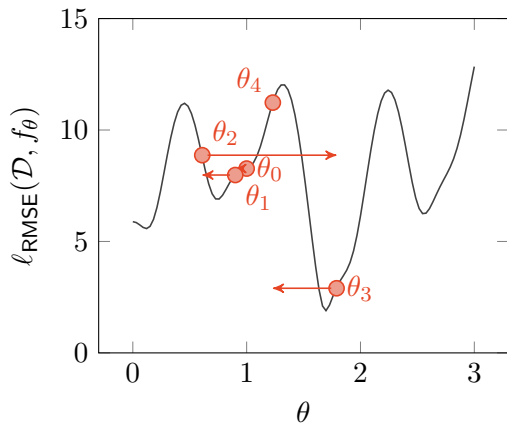
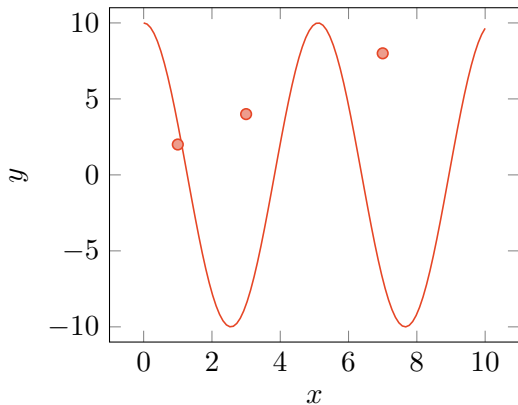
## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before



## Example 2: Cosine Functions

1. Consider the model class  $\mathcal{F} = \{f_\theta | f_\theta(x) = 10 \cdot \cos(x \cdot \theta) \text{ for some } \theta \in \mathbb{R}\}$
2. same data and loss as before

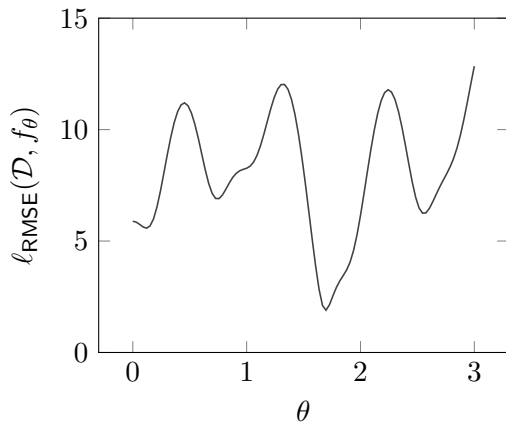


# Challenges in optimization

- ▶ saddle points, sudden gradient shifts, local optima, ...

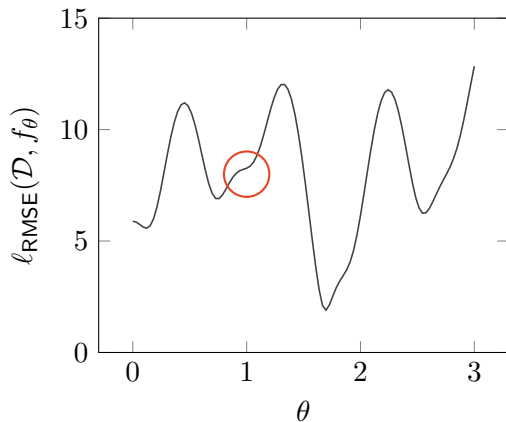
## Challenges in optimization

- ▶ saddle points, sudden gradient shifts, local optima, ...



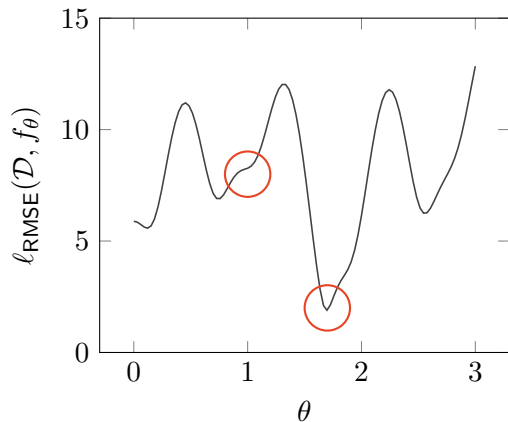
## Challenges in optimization

- ▶ saddle points, sudden gradient shifts, local optima, ...



## Challenges in optimization

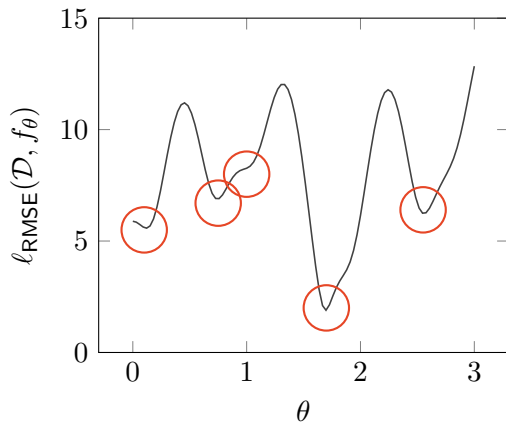
- saddle points, sudden gradient shifts, local optima, ...





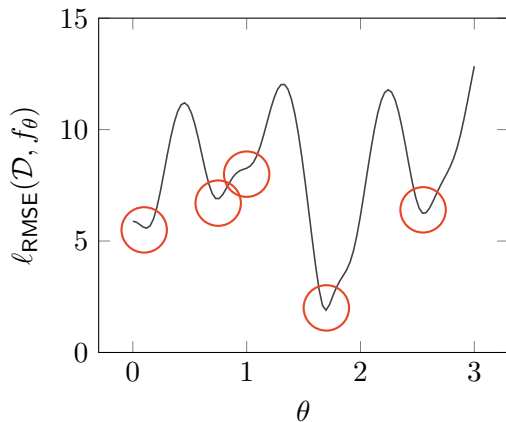
## Challenges in optimization

- saddle points, sudden gradient shifts, local optima, ...



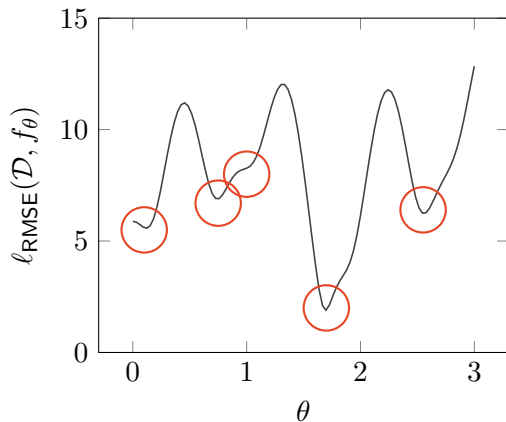
## Challenges in optimization

- ▶ saddle points, sudden gradient shifts, local optima, ...
- ▶ still problematic for better algorithms, e.g. ADAM, conjugate gradient, L-BFGS (Paaßen 2019)



## Challenges in optimization

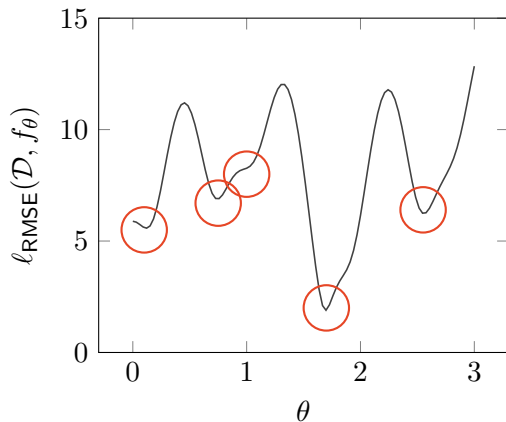
- ▶ saddle points, sudden gradient shifts, local optima, ...
- ▶ still problematic for better algorithms, e.g. ADAM, conjugate gradient, L-BFGS (Paaßen 2019)
- ▶ So, when is optimization “nice”?



## Challenges in optimization

- ▶ saddle points, sudden gradient shifts, local optima, ...
- ▶ still problematic for better algorithms, e.g. ADAM, conjugate gradient, L-BFGS (Paaßen 2019)
- ▶ So, when is optimization “nice”?

⇒ **convex** optimization



# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph

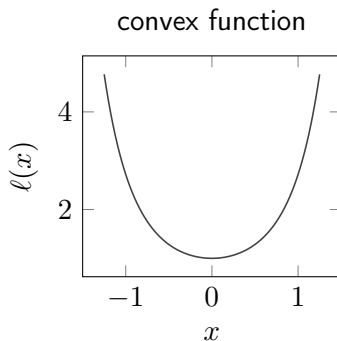
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- Intuitive: Any line between two points on the function graph is **above** the function graph





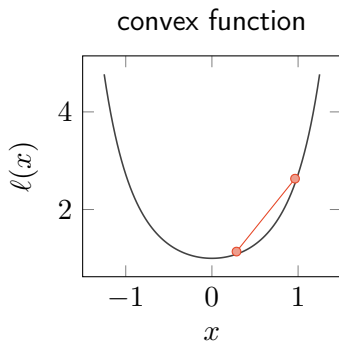
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- Intuitive: Any line between two points on the function graph is **above** the function graph



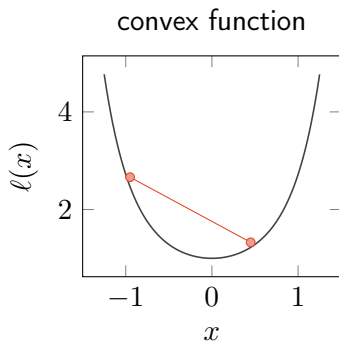
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- Intuitive: Any line between two points on the function graph is **above** the function graph



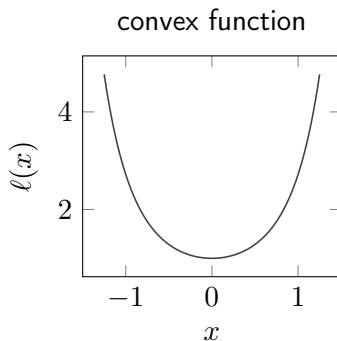
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
- ▶ **Equivalent:** The **gradient** is **below**



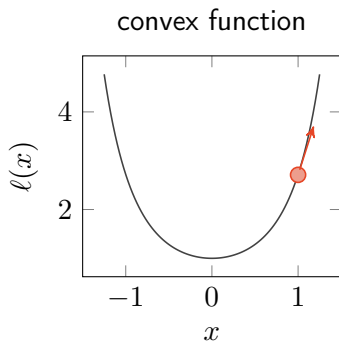
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
- ▶ **Equivalent:** The **gradient** is **below**



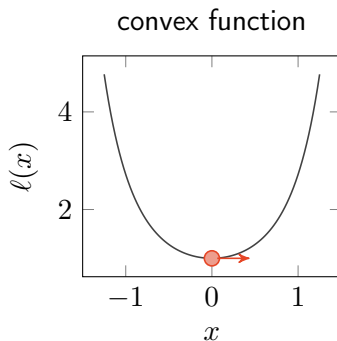
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
  - ▶ **Equivalent:** The **gradient** is **below**
- ⇒ Gradient descent finds **global** optima



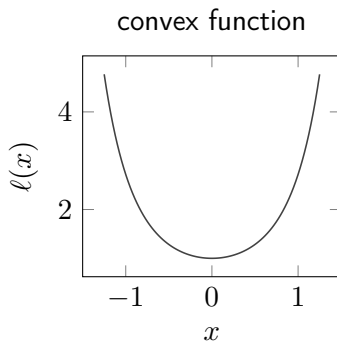
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
- ▶ **Equivalent:** The **gradient** is **below**
- ⇒ Gradient descent finds **global** optima



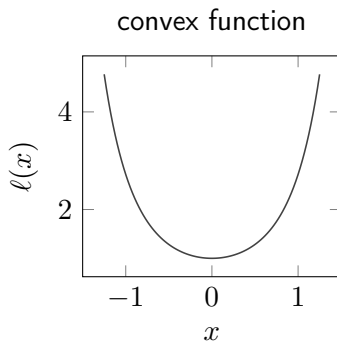
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
- ▶ **Equivalent:** The **gradient** is **below**
- ⇒ Gradient descent finds **global** optima
- ▶ **Also equivalent:** Second derivative is **always** positive



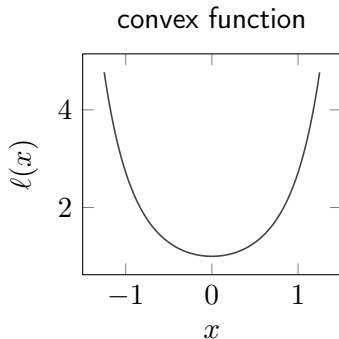
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
- ▶ **Equivalent:** The **gradient** is **below**
- ⇒ Gradient descent finds **global** optima
- ▶ **Also equivalent:** Second derivative is **always** positive
- ⇒ Gradient descent is “smooth”





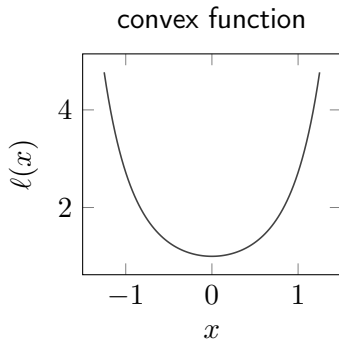
# Convex Optimization

## Convexity

A function  $\ell : \mathbb{R}^m \rightarrow \mathbb{R}$  is called **convex** if for all  $x, y \in \mathbb{R}^m$  and all  $\alpha \in [0, 1]$  it holds:

$$\ell(\alpha \cdot x + (1 - \alpha) \cdot y) \leq \alpha \cdot \ell(x) + (1 - \alpha) \cdot \ell(y)$$

- ▶ Intuitive: Any line between two points on the function graph is **above** the function graph
- ▶ **Equivalent:** The **gradient** is **below**
- ⇒ Gradient descent finds **global** optima
- ▶ **Also equivalent:** Second derivative is **always** positive
- ⇒ Gradient descent is “smooth”
- ▶ **Pro tip:** Design your loss to be **convex**



# Linear Regression



THE UNIVERSITY OF  
SYDNEY

# Introduction

One-dimensional linear regression model class:

$$\mathcal{F} = \{f_w | f_w(x) = w \cdot x, \quad w \in \mathbb{R}\}$$

# Introduction

One-dimensional linear regression model class:

$$\mathcal{F} = \{f_w | f_w(x) = w \cdot x, \quad w \in \mathbb{R}\}$$

Example: I wish to bake  $x$  vegan brownies.

My recipe is:

ingredient	amount
Zucchini [g]	150
Flour [g]	60
Cocoa powder [g]	15
Sugar [g]	60
Oil [ml]	15
...	...

For full recipe, refer to:

<https://cakeinvasion.de/>

# Introduction

One-dimensional linear regression model class:

$$\mathcal{F} = \{f_w | f_w(x) = w \cdot x, \quad w \in \mathbb{R}\}$$

Example: I wish to bake  $x$  vegan brownies.

My recipe is:

ingredient	amount
Zucchini [g]	150
Flour [g]	60
Cocoa powder [g]	15
Sugar [g]	60
Oil [ml]	15
...	...

For full recipe, refer to:

<https://cakeinvasion.de/>

- ▶ How much of each ingredient do I need to buy for  $x$  portions?

# Introduction

One-dimensional linear regression model class:

$$\mathcal{F} = \{f_w | f_w(x) = w \cdot x, \quad w \in \mathbb{R}\}$$

Example: I wish to bake  $x$  vegan brownies.

My recipe is:

ingredient	amount
Zucchini [g]	150
Flour [g]	60
Cocoa powder [g]	15
Sugar [g]	60
Oil [ml]	15
...	...

For full recipe, refer to:

<https://cakeinvasion.de/>

- ▶ How much of each ingredient do I need to buy for  $x$  portions?
- ▶ for each ingredient, one linear model with input  $x$  and coefficient  $w$  in table

# Generalized Linear Regression

- ▶ Assume a function  $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$  that maps inputs  $x$  to  $n$ -dimensional **feature vectors**  $\phi(x)$

# Generalized Linear Regression

- ▶ Assume a function  $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$  that maps inputs  $x$  to  $n$ -dimensional **feature vectors**  $\phi(x)$
- ▶ Model class of **generalized linear regression**:

$$\mathcal{F}_{\text{GLR}} = \{f_{\vec{w}} | f_{\vec{w}}(x) = \vec{w}^T \cdot \phi(x), \quad \vec{w} \in \mathbb{R}^n\}$$

where  $\vec{w}^T \cdot \phi(x) = w_1 \cdot \phi_1(x) + \dots + w_n \cdot \phi_n(x)$



# Generalized Linear Regression

- ▶ Assume a function  $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$  that maps inputs  $x$  to  $n$ -dimensional **feature vectors**  $\phi(x)$
- ▶ Model class of **generalized linear regression**:

$$\mathcal{F}_{\text{GLR}} = \{f_{\vec{w}} | f_{\vec{w}}(x) = \vec{w}^T \cdot \phi(x), \quad \vec{w} \in \mathbb{R}^n\}$$

where  $\vec{w}^T \cdot \phi(x) = w_1 \cdot \phi_1(x) + \dots + w_n \cdot \phi_n(x)$

- ▶ Example: You are the inputs and  $\phi$  maps to your respective body height, pinky finger length, and favourite color ( $n = 3$ ).

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)
- ▶ Then, we wish to solve

$$\min_{f \in \mathcal{F}_{\text{GLR}}} \ell_{\text{RMSE}}(\mathcal{D}, f)$$

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)
- ▶ Then, we wish to solve

$$\min_{f \in \mathcal{F}_{\text{GLR}}} \ell_{\text{RMSE}}(\mathcal{D}, f)$$

- ▶ First trick: Rephrase the problem

$$\min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2$$

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)

- ▶ Then, we wish to solve

$$\min_{f \in \mathcal{F}_{\text{GLR}}} \ell_{\text{RMSE}}(\mathcal{D}, f)$$

- ▶ First trick: Rephrase the problem

$$\min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2$$

- ▶ Consider the gradient (1st derivative) and Hessian (2nd derivative):

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)
- ▶ Then, we wish to solve

$$\min_{f \in \mathcal{F}_{\text{GLR}}} \ell_{\text{RMSE}}(\mathcal{D}, f)$$

- ▶ First trick: Rephrase the problem

$$\min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2$$

- ▶ Consider the gradient (1st derivative) and Hessian (2nd derivative):

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i)$$

## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)
- ▶ Then, we wish to solve

$$\min_{f \in \mathcal{F}_{\text{GLR}}} \ell_{\text{RMSE}}(\mathcal{D}, f)$$

- ▶ First trick: Rephrase the problem

$$\min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2$$

- ▶ Consider the gradient (1st derivative) and Hessian (2nd derivative):

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i)$$

$$\nabla_{\vec{w}}^2 \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = 2 \sum_{i=1}^N \phi(x_i) \cdot \phi(x_i)^T$$



## Generalized Linear Regression: Derivation

- ▶ Assume data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  and feature map  $\phi$  are given, where  $y_1, \dots, y_N$  are real numbers (e.g. your shoe size)

- ▶ Then, we wish to solve

$$\min_{f \in \mathcal{F}_{\text{GLR}}} \ell_{\text{RMSE}}(\mathcal{D}, f)$$

- ▶ First trick: Rephrase the problem

$$\min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2$$

- ▶ Consider the gradient (1st derivative) and Hessian (2nd derivative):

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i)$$

$$\nabla_{\vec{w}}^2 \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = 2 \sum_{i=1}^N \phi(x_i) \cdot \phi(x_i)^T \geq 0 \quad \Rightarrow \text{convex}$$

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i)$$

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) = 2\Phi \cdot \Phi^T \cdot \vec{w} - 2\Phi \cdot \vec{y}$$

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) = 2\Phi \cdot \Phi^T \cdot \vec{w} - 2\Phi \cdot \vec{y}$$

where  $\Phi = (\phi(x_1), \dots, \phi(x_N))$ ,  $\vec{y} = (y_1, \dots, y_N)^T$ .

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) = 2\Phi \cdot \Phi^T \cdot \vec{w} - 2\Phi \cdot \vec{y}$$

where  $\Phi = (\phi(x_1), \dots, \phi(x_N))$ ,  $\vec{y} = (y_1, \dots, y_N)^T$ .

- ▶  $2\Phi \cdot \Phi^T \cdot \vec{w}^* - 2\Phi \cdot \vec{y} \stackrel{!}{=} 0$

## Generalized Linear Regression: Derivation (2)

- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) = 2\Phi \cdot \Phi^T \cdot \vec{w} - 2\Phi \cdot \vec{y}$$

where  $\Phi = (\phi(x_1), \dots, \phi(x_N))$ ,  $\vec{y} = (y_1, \dots, y_N)^T$ .

- ▶  $2\Phi \cdot \Phi^T \cdot \vec{w}^* - 2\Phi \cdot \vec{y} \stackrel{!}{=} 0 \iff \vec{w}^* = (\Phi \cdot \Phi^T)^{-1} \cdot \Phi \cdot \vec{y}$



## Generalized Linear Regression: Derivation (2)

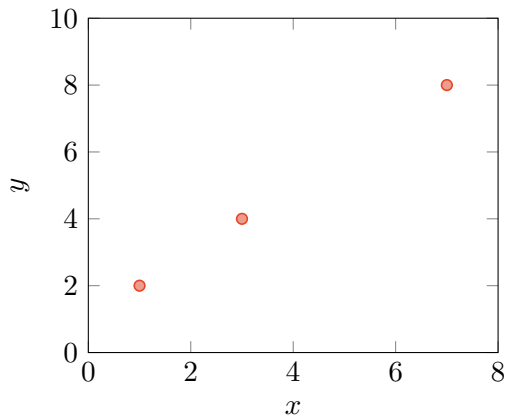
- ▶ Recall: If the loss is **convex**, any point with vanishing gradient is **global optimum**
- ▶ Let's try to find such a point analytically!

$$\nabla_{\vec{w}} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 = \sum_{i=1}^N 2 \cdot \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) = 2\Phi \cdot \Phi^T \cdot \vec{w} - 2\Phi \cdot \vec{y}$$

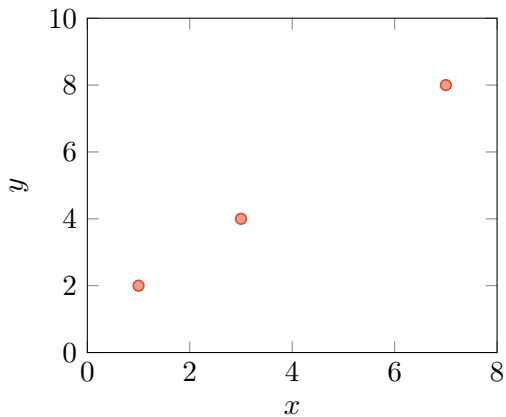
where  $\Phi = (\phi(x_1), \dots, \phi(x_N))$ ,  $\vec{y} = (y_1, \dots, y_N)^T$ .

- ▶  $2\Phi \cdot \Phi^T \cdot \vec{w}^* - 2\Phi \cdot \vec{y} \stackrel{!}{=} 0 \iff \vec{w}^* = (\Phi \cdot \Phi^T)^{-1} \cdot \Phi \cdot \vec{y}$
- ▶  $\mathcal{A}_{\text{GLR}}(\mathcal{D}) = f_{\vec{w}^*}$

## Example

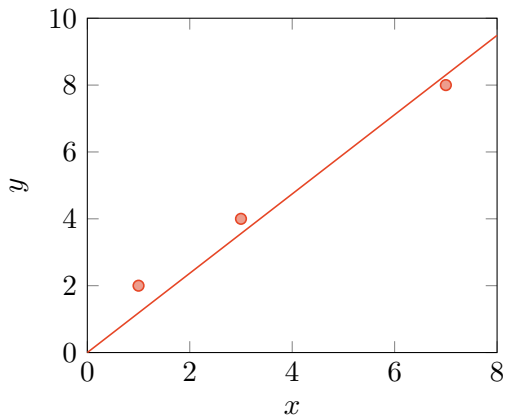


## Example



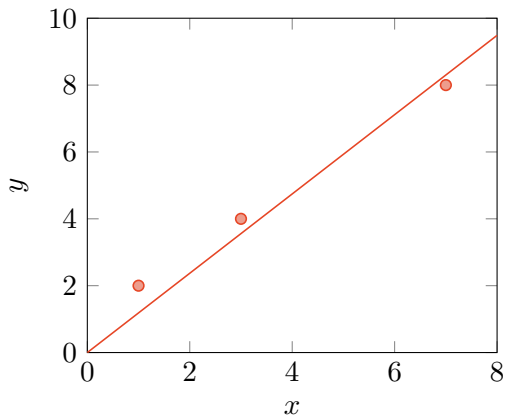
► Let's start with  $\phi(x) = x$

## Example



► Let's start with  $\phi(x) = x$

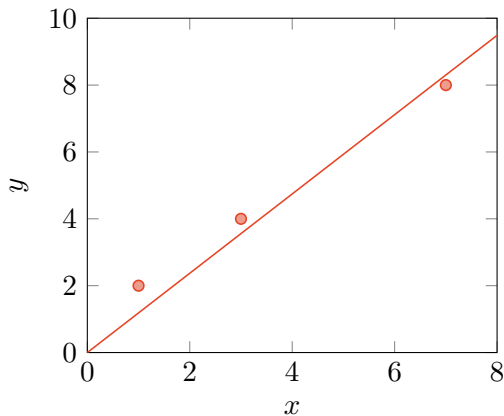
## Example



► Let's start with  $\phi(x) = x$

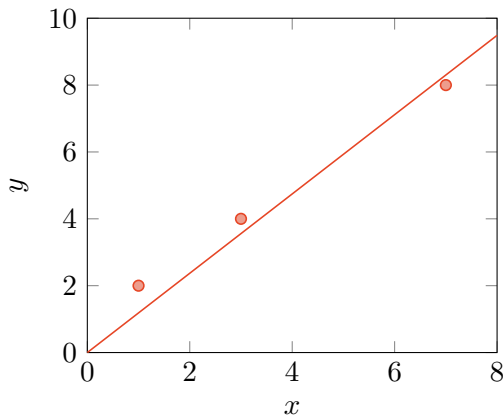
⇒ Does not work ☹

## Example



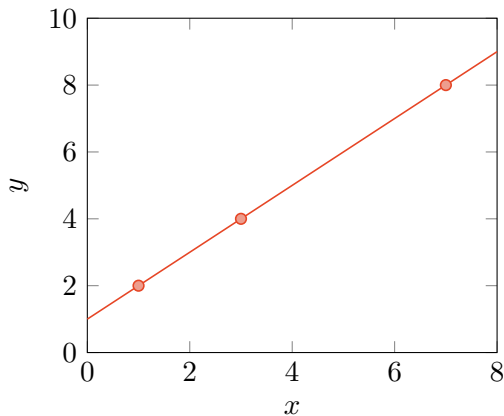
- ▶ Let's start with  $\phi(x) = x$
- ⇒ Does not work ☹
- ▶ We need a constant term; so let's try  $\phi(x) = (1, x)^T$

## Example



- ▶ Let's start with  $\phi(x) = x$
- ⇒ Does not work ☹
- ▶ We need a constant term; so let's try  
 $\phi(x) = (1, x)^T \Rightarrow$   
 $f(x) = \vec{w}^T \cdot \phi(x) = w_1 \cdot 1 + w_2 \cdot x$

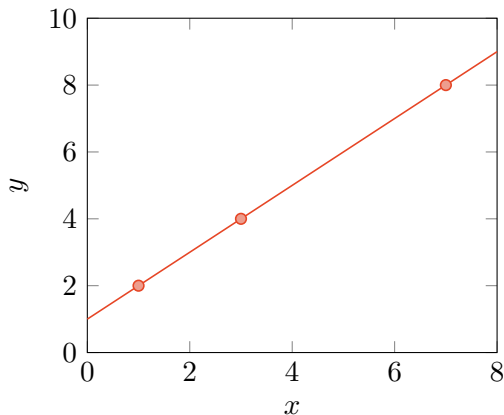
## Example



- ▶ Let's start with  $\phi(x) = x$
- ⇒ Does not work ☹
- ▶ We need a constant term; so let's try  
 $\phi(x) = (1, x)^T \Rightarrow$   
 $f(x) = \vec{w}^T \cdot \phi(x) = w_1 \cdot 1 + w_2 \cdot x$



## Example



▶ Let's start with  $\phi(x) = x$

⇒ Does not work ☹

▶ We need a constant term; so let's try

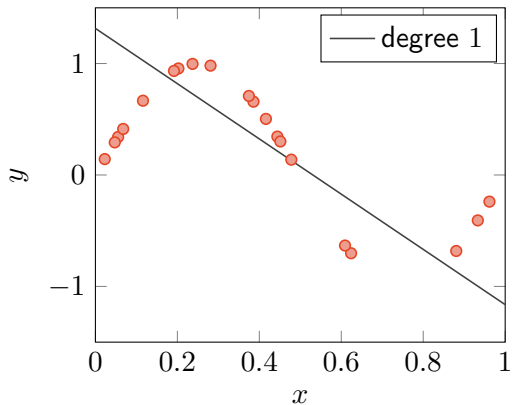
$$\phi(x) = (1, x)^T \Rightarrow$$

$$f(x) = \vec{w}^T \cdot \phi(x) = w_1 \cdot 1 + w_2 \cdot x$$

⇒ ☺

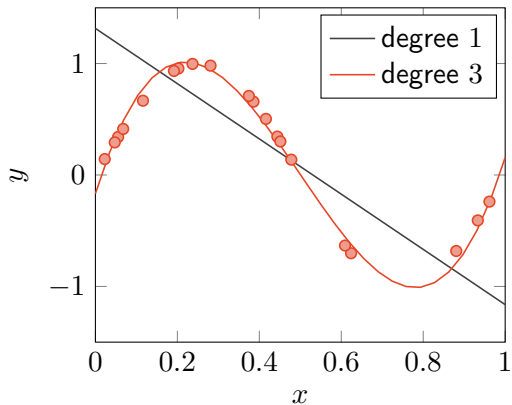
# Polynomial Regression

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



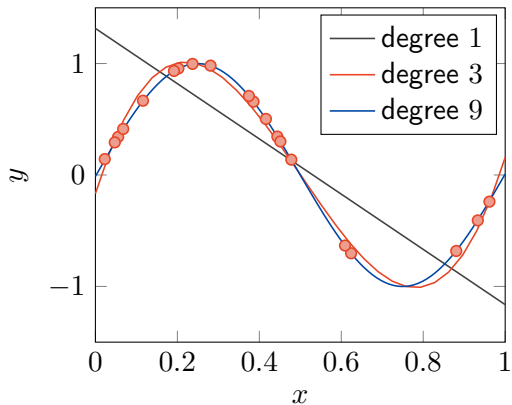
# Polynomial Regression

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



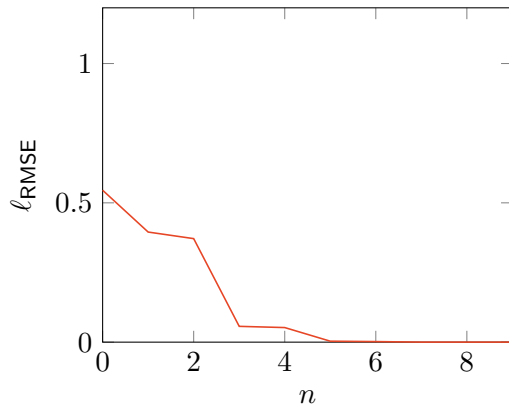
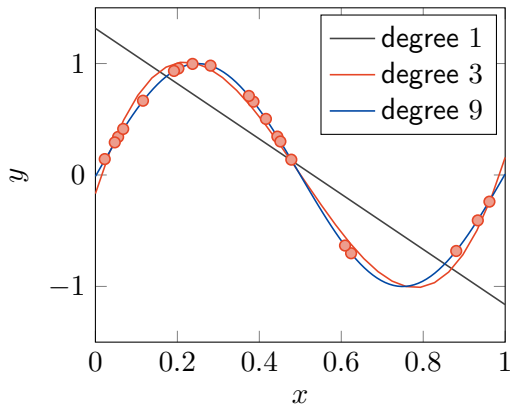
# Polynomial Regression

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



# Polynomial Regression

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



# Universal Approximation

Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator**

# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$

# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ ,



# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ , there exists an  $n_\epsilon$ , such that

# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ , there exists an  $n_\epsilon$ , such that  $\min_{f \in \mathcal{F}} \ell_{\text{RMSE}}(\mathcal{D}, \mathcal{F}_{n_\epsilon}) < \epsilon$ .

# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ , there exists an  $n_\epsilon$ , such that  $\min_{f \in \mathcal{F}} \ell_{\text{RMSE}}(\mathcal{D}, \mathcal{F}_{n_\epsilon}) < \epsilon$ .

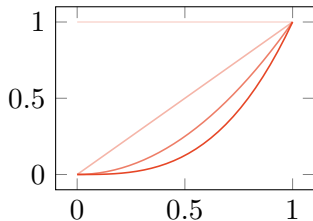
Generalized Linear Regression is a universal approximator for surprisingly many  $\phi$ , e.g.:

# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ , there exists an  $n_\epsilon$ , such that  $\min_{f \in \mathcal{F}} \ell_{\text{RMSE}}(\mathcal{D}, \mathcal{F}_{n_\epsilon}) < \epsilon$ .

Generalized Linear Regression is a universal approximator for surprisingly many  $\phi$ , e.g.:  
polynoms



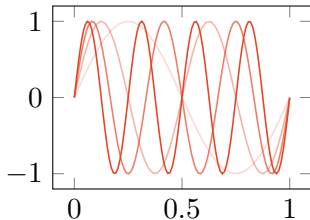
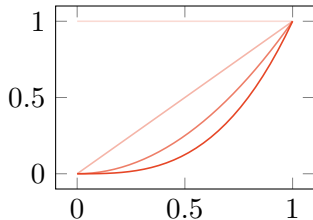
# Universal Approximation

## Definition: Universal Approximator (roughly)

We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ , there exists an  $n_\epsilon$ , such that  $\min_{f \in \mathcal{F}} \ell_{\text{RMSE}}(\mathcal{D}, \mathcal{F}_{n_\epsilon}) < \epsilon$ .

Generalized Linear Regression is a universal approximator for surprisingly many  $\phi$ , e.g.:

polynomials                      sine/cosine waves



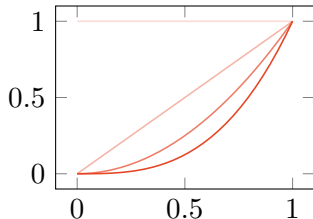
# Universal Approximation

## Definition: Universal Approximator (roughly)

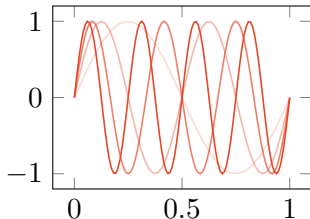
We call a (parametrized) model class  $\mathcal{F}_n$  a **universal approximator** if for any smooth data set of real numbers  $\mathcal{D}$  and any  $\epsilon > 0$ , there exists an  $n_\epsilon$ , such that  $\min_{f \in \mathcal{F}} \ell_{\text{RMSE}}(\mathcal{D}, \mathcal{F}_{n_\epsilon}) < \epsilon$ .

Generalized Linear Regression is a universal approximator for surprisingly many  $\phi$ , e.g.:

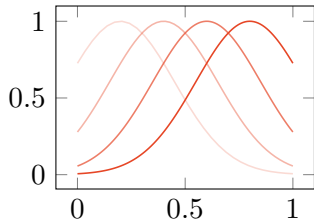
polynoms



sine/cosine waves



Radial Basis Functions



## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions.

## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions. A simple example:  
 $\phi(\vec{x}) = (\tanh(\vec{u}_1^T \cdot \vec{x}), \dots, \tanh(\vec{u}_n^T \cdot \vec{x}))$  for random (!)  $\vec{u}_1, \dots, \vec{u}_n$



## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions. A simple example:  
 $\phi(\vec{x}) = (\tanh(\vec{u}_1^T \cdot \vec{x}), \dots, \tanh(\vec{u}_n^T \cdot \vec{x}))$  for random (!)  $\vec{u}_1, \dots, \vec{u}_n \Rightarrow$  Basic trick behind neural engineering framework, extreme learning machines, echo state networks ...

## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions. A simple example:  
 $\phi(\vec{x}) = (\tanh(\vec{u}_1^T \cdot \vec{x}), \dots, \tanh(\vec{u}_n^T \cdot \vec{x}))$  for random (!)  $\vec{u}_1, \dots, \vec{u}_n \Rightarrow$  Basic trick behind neural engineering framework, extreme learning machines, echo state networks ...
- ▶ For all  $\phi$ : Higher  $n \rightarrow$  better approximation on the training data

## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions. A simple example:  
 $\phi(\vec{x}) = (\tanh(\vec{u}_1^T \cdot \vec{x}), \dots, \tanh(\vec{u}_n^T \cdot \vec{x}))$  for random (!)  $\vec{u}_1, \dots, \vec{u}_n \Rightarrow$  Basic trick behind neural engineering framework, extreme learning machines, echo state networks ...
- ▶ For all  $\phi$ : Higher  $n \rightarrow$  better approximation on the training data
- ▶ **But** lower  $n$  is more efficient and more interpretable

## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions. A simple example:  
 $\phi(\vec{x}) = (\tanh(\vec{u}_1^T \cdot \vec{x}), \dots, \tanh(\vec{u}_n^T \cdot \vec{x}))$  for random (!)  $\vec{u}_1, \dots, \vec{u}_n \Rightarrow$  Basic trick behind neural engineering framework, extreme learning machines, echo state networks ...
- ▶ For all  $\phi$ : Higher  $n \rightarrow$  better approximation on the training data
- ▶ **But** lower  $n$  is more efficient and more interpretable  $\Rightarrow$  homework task

## Universal Approximation (II)

- ▶ Universal approximation also works in multiple dimensions. A simple example:  
 $\phi(\vec{x}) = (\tanh(\vec{u}_1^T \cdot \vec{x}), \dots, \tanh(\vec{u}_n^T \cdot \vec{x}))$  for random (!)  $\vec{u}_1, \dots, \vec{u}_n \Rightarrow$  Basic trick behind neural engineering framework, extreme learning machines, echo state networks ...
- ▶ For all  $\phi$ : Higher  $n \rightarrow$  better approximation on the training data
- ▶ **But** lower  $n$  is more efficient and more interpretable  $\Rightarrow$  homework task
- ▶ Also, lower  $n$  is more robust to input noise

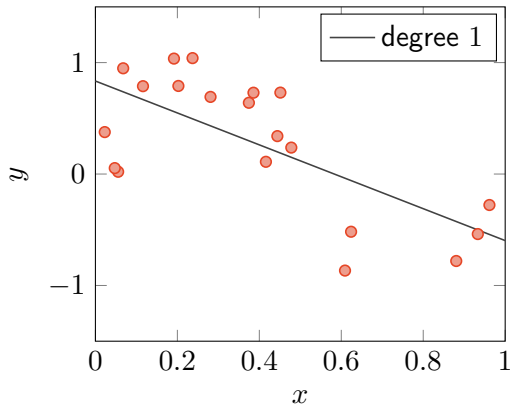
# Regularization



THE UNIVERSITY OF  
SYDNEY

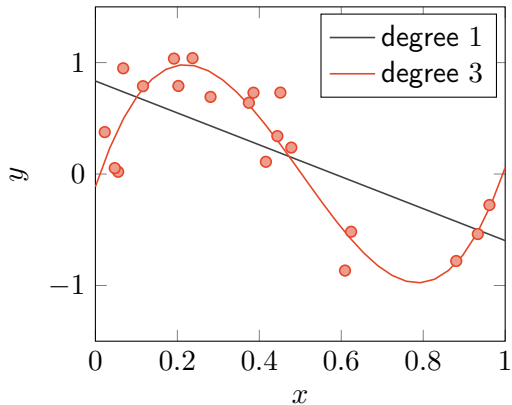
## Polynomial Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



## Polynomial Regression with Noise

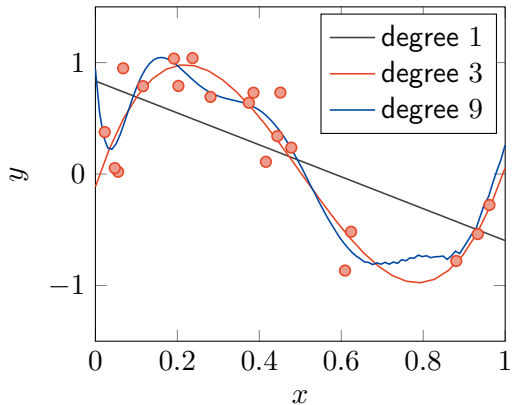
Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .





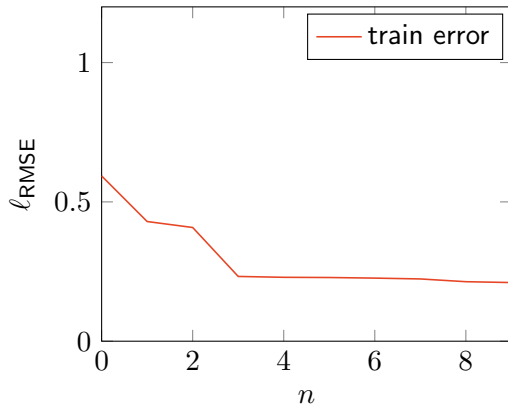
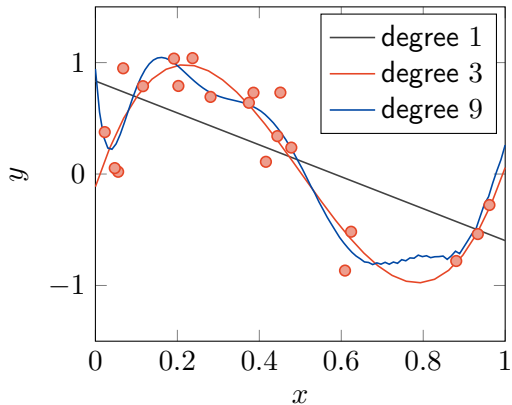
## Polynomial Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



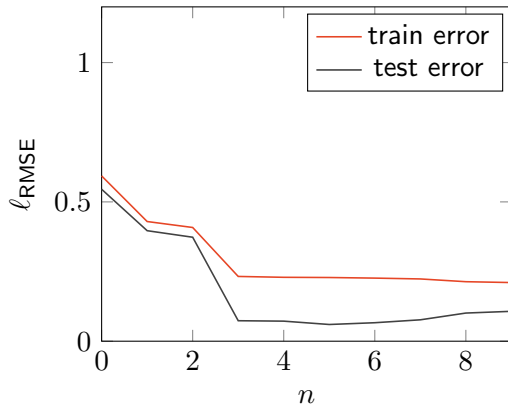
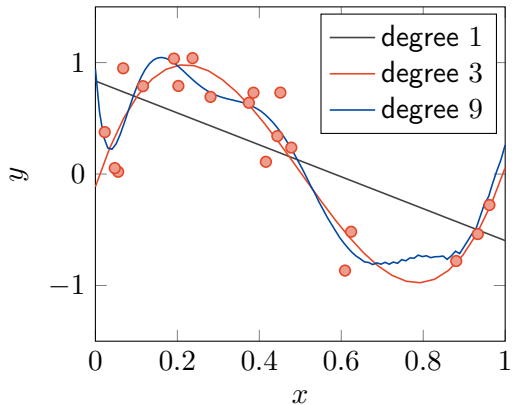
## Polynomial Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



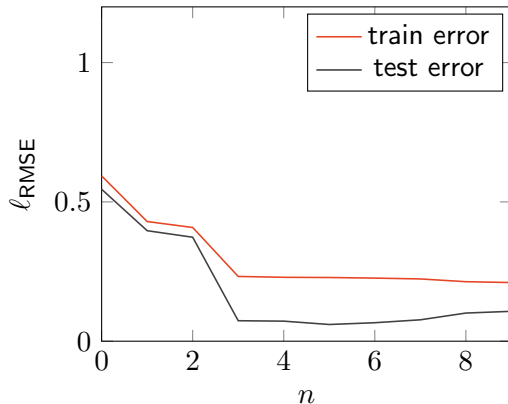
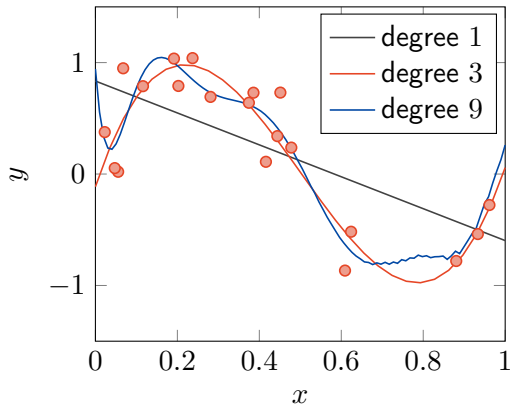
## Polynomial Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



## Polynomial Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ .



$\Rightarrow$  For high model complexity, model can fit the noise and does not generalize  
(**overfitting**)

## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )

## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )
- ▶ ... because the optimization will do anything to decrease the training error even a tiny bit

## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )
  - ▶ ... because the optimization will do anything to decrease the training error even a tiny bit
- ⇒ Punish extreme parameter values (**regularization**)

## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )
  - ▶ ... because the optimization will do anything to decrease the training error even a tiny bit
- ⇒ Punish extreme parameter values (**regularization**)

$$\min_{\vec{w} \in \mathbb{R}^n} \overbrace{\sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2}^{\ell} + \lambda \cdot \vec{w}^T \cdot \vec{w}$$



## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )
  - ▶ ... because the optimization will do anything to decrease the training error even a tiny bit
- ⇒ Punish extreme parameter values (**regularization**)

$$\min_{\vec{w} \in \mathbb{R}^n} \overbrace{\sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2}^{\ell} + \lambda \cdot \vec{w}^T \cdot \vec{w}$$
$$\nabla_{\vec{w}} \ell = 2 \cdot \sum_{i=1}^N \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) + 2 \cdot \lambda \cdot \vec{w}$$

## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )
  - ▶ ... because the optimization will do anything to decrease the training error even a tiny bit
- ⇒ Punish extreme parameter values (**regularization**)

$$\begin{aligned} \min_{\vec{w} \in \mathbb{R}^n} \quad & \overbrace{\sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2}^{\ell} + \lambda \cdot \vec{w}^T \cdot \vec{w} \\ \nabla_{\vec{w}} \ell = 2 \cdot \quad & \sum_{i=1}^N \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) + 2 \cdot \lambda \cdot \vec{w} \\ \nabla_{\vec{w}}^2 \ell = 2 \quad & \sum_{i=1}^N \phi(x_i) \cdot \phi(x_i)^T + \lambda \cdot I \end{aligned}$$

## Regularization (I)

- ▶ empirically, overfitting behavior relates to extreme parameter values ( $> 10^5$ )
  - ▶ ... because the optimization will do anything to decrease the training error even a tiny bit
- ⇒ Punish extreme parameter values (**regularization**)

$$\begin{aligned} \min_{\vec{w} \in \mathbb{R}^n} \quad & \overbrace{\sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2}^{\ell} + \lambda \cdot \vec{w}^T \cdot \vec{w} \\ \nabla_{\vec{w}} \ell = 2 \cdot \quad & \sum_{i=1}^N \phi(x_i) \cdot (\phi(x_i)^T \cdot \vec{w} - y_i) + 2 \cdot \lambda \cdot \vec{w} \\ \nabla_{\vec{w}}^2 \ell = 2 \sum_{i=1}^N \quad & \phi(x_i) \cdot \phi(x_i)^T + \lambda \cdot I \quad \Rightarrow \text{still convex} \end{aligned}$$

## Regularization (II)

$$\nabla_{\vec{w}} \ell \stackrel{!}{=} 0$$

## Regularization (II)

$$\nabla_{\vec{w}} \ell \stackrel{!}{=} 0$$

$$2 \cdot \Phi \cdot \Phi^T \cdot \vec{w} - \Phi \cdot \vec{y} + 2 \cdot \lambda \cdot \vec{w} \stackrel{!}{=} 0$$

## Regularization (II)

$$\nabla_{\vec{w}} \ell \stackrel{!}{=} 0$$

$$2 \cdot \Phi \cdot \Phi^T \cdot \vec{w} - \Phi \cdot \vec{y} + 2 \cdot \lambda \cdot \vec{w} \stackrel{!}{=} 0$$

$$(\Phi \cdot \Phi^T + \lambda \cdot I) \cdot \vec{w} \stackrel{!}{=} \Phi \cdot \vec{y}$$

## Regularization (II)

$$\nabla_{\vec{w}} \ell \stackrel{!}{=} 0$$

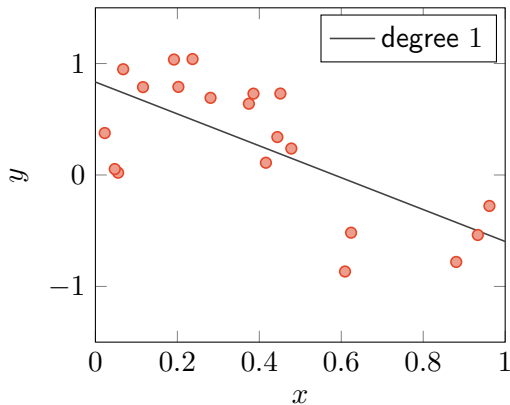
$$2 \cdot \Phi \cdot \Phi^T \cdot \vec{w} - \Phi \cdot \vec{y} + 2 \cdot \lambda \cdot \vec{w} \stackrel{!}{=} 0$$

$$(\Phi \cdot \Phi^T + \lambda \cdot I) \cdot \vec{w} \stackrel{!}{=} \Phi \cdot \vec{y}$$

$$\vec{w} \stackrel{!}{=} (\Phi \cdot \Phi^T + \lambda \cdot I)^{-1} \cdot \Phi \cdot \vec{y}$$

## Regularized Regression with Noise

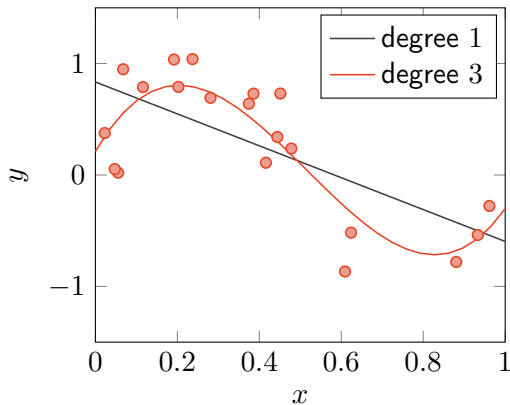
Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ ,  $\lambda = 10^{-3}$ .





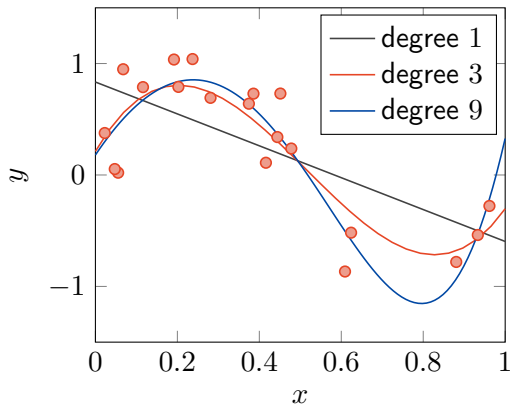
## Regularized Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ ,  $\lambda = 10^{-3}$ .



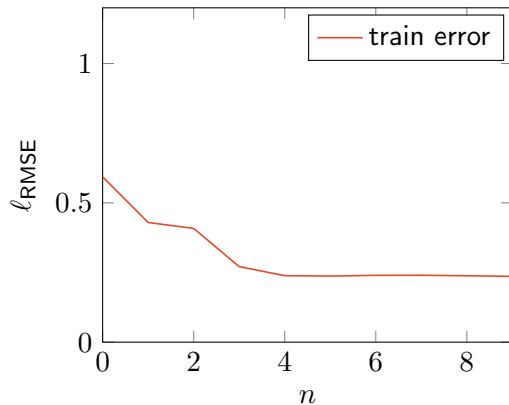
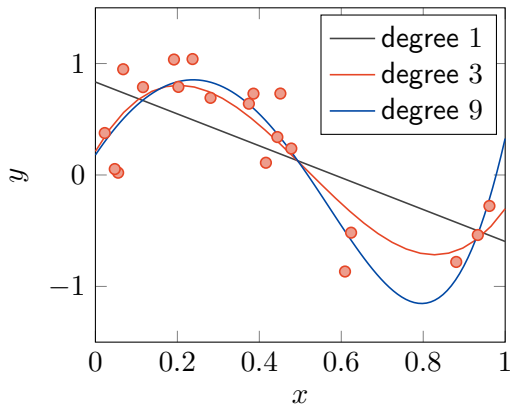
## Regularized Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ ,  $\lambda = 10^{-3}$ .



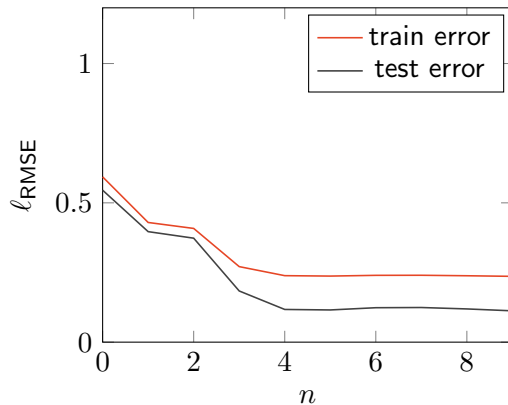
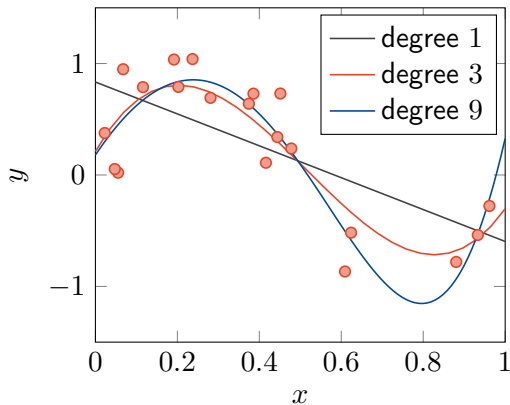
## Regularized Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ ,  $\lambda = 10^{-3}$ .



## Regularized Regression with Noise

Let's consider  $\phi_n(x) = (1, x, x^2, \dots, x^n)$ ,  $\lambda = 10^{-3}$ .



## Summary

- ▶ Generalized linear regression is a powerful tool for prediction, especially with good feature map  $\phi$

## Summary

- ▶ Generalized linear regression is a powerful tool for prediction, especially with good feature map  $\phi$
- ▶ Already illustrates many typical challenges of ML: Underfitting, overfitting, generalization, model architecture, feature processing, ...

## Summary

- ▶ Generalized linear regression is a powerful tool for prediction, especially with good feature map  $\phi$
- ▶ Already illustrates many typical challenges of ML: Underfitting, overfitting, generalization, model architecture, feature processing, ...
- ▶ Success in practice depends on understanding data, features, optimization, numerics, ...

# Probabilities

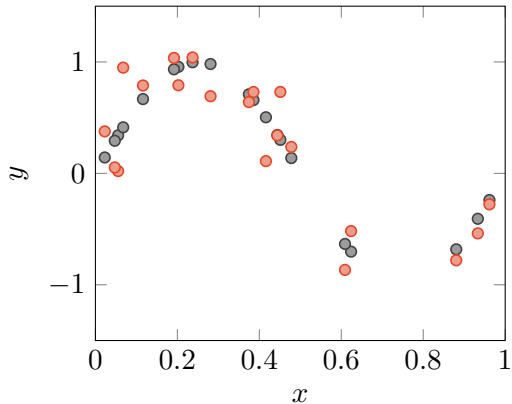


THE UNIVERSITY OF  
SYDNEY



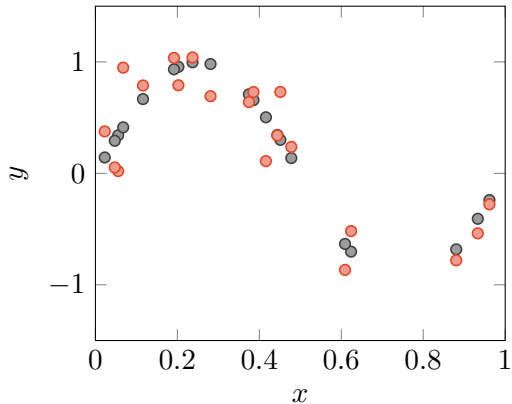
## Generalization Redux

We expect generalization between these data sets.

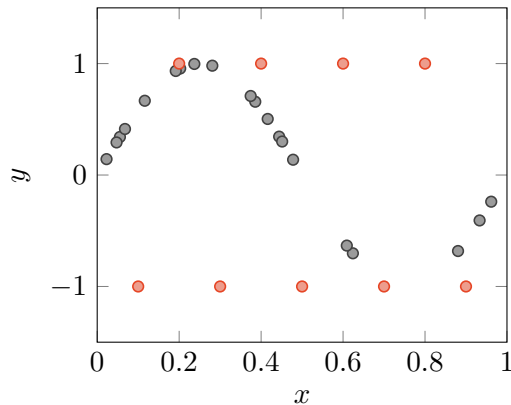


## Generalization Redux

We expect generalization between these data sets.

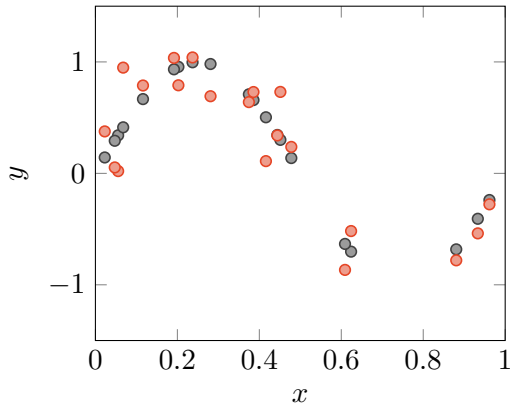


But how about these?

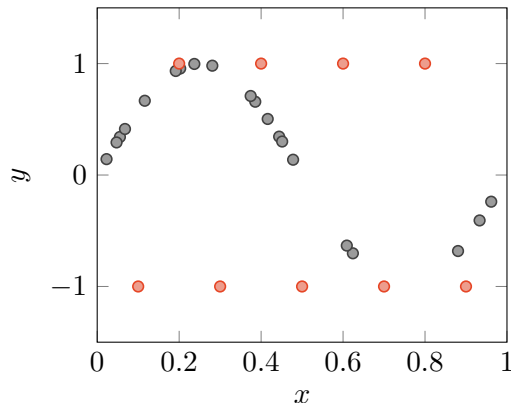


## Generalization Redux

We expect generalization between these data sets.



But how about these?



⇒ We only want to generalize only to datasets from the **same data source**

# Standard Data Generation Model

input (randomly generated)

$x$

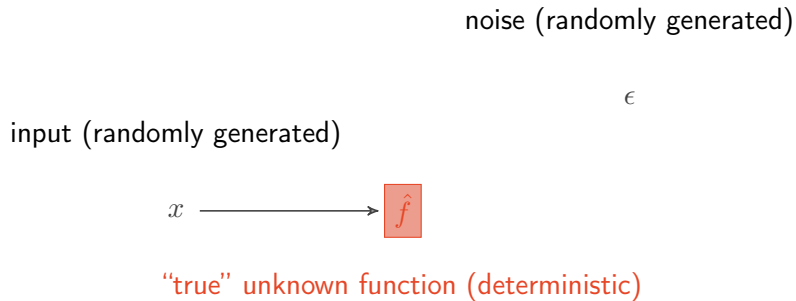
# Standard Data Generation Model

input (randomly generated)

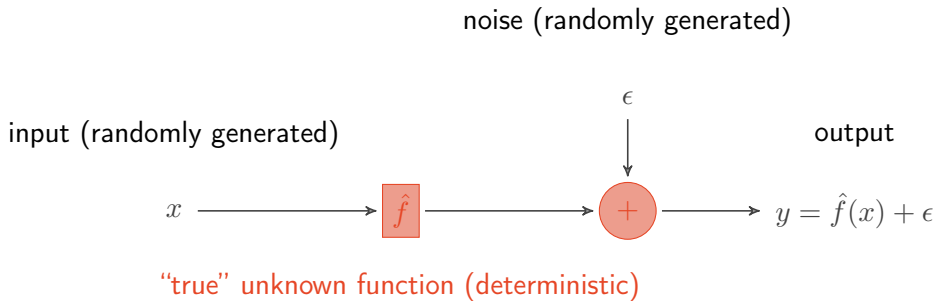


“true” unknown function (deterministic)

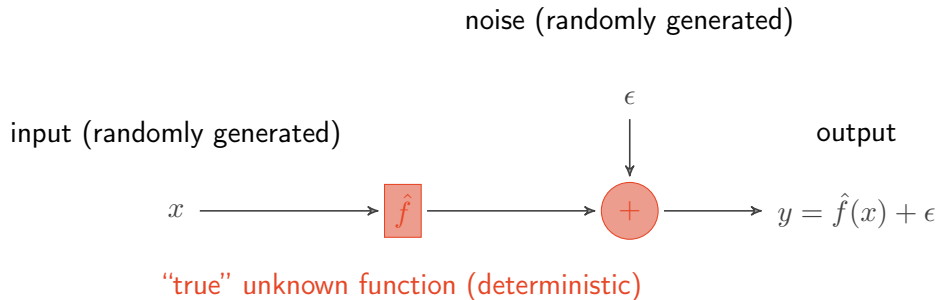
# Standard Data Generation Model



# Standard Data Generation Model



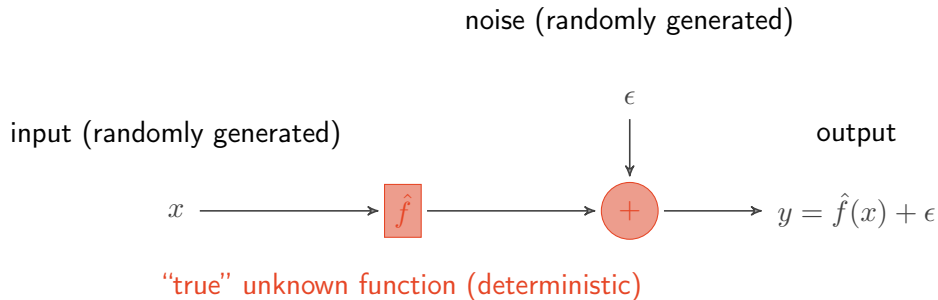
# Standard Data Generation Model



- Assumes that the data are **identically** and **independently** distributed (i.i.d.)



# Standard Data Generation Model



- ▶ Assumes that the data are **identically** and **independently** distributed (i.i.d.)
- ▶  $x$ ,  $\epsilon$ , and  $y$  are **random variables**

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable that can take a **value**  $x$  from a set  $\Omega_X$

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable that can take a **value**  $x$  from a set  $\Omega_X$  with some **probability**  $p_X(x) \in [0, 1]$ .

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable that can take a **value**  $x$  from a set  $\Omega_X$  with some **probability**  $p_X(x) \in [0, 1]$ .
- ▶ We call the function  $p_X : \Omega_X \rightarrow \mathbb{R}^+$  with  $\int_{x \in \Omega_X} p_X(x) dx = 1$  a **probability density**

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable that can take a **value**  $x$  from a set  $\Omega_X$  with some **probability**  $p_X(x) \in [0, 1]$ .
- ▶ We call the function  $p_X : \Omega_X \rightarrow \mathbb{R}^+$  with  $\int_{x \in \Omega_X} p_X(x) dx = 1$  a **probability density**
- ▶ A probability **distribution** is a function  $P_X$  that assigns probabilities to subsets  $A \subseteq \Omega_X$ , i.e.  $P(X \in A) := \int_{x \in A} p_X(x) dx$ .

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable that can take a **value**  $x$  from a set  $\Omega_X$  with some **probability**  $p_X(x) \in [0, 1]$ .
- ▶ We call the function  $p_X : \Omega_X \rightarrow \mathbb{R}^+$  with  $\int_{x \in \Omega_X} p_X(x) dx = 1$  a **probability density**
- ▶ A probability **distribution** is a function  $P_X$  that assigns probabilities to subsets  $A \subseteq \Omega_X$ , i.e.  $P(X \in A) := \int_{x \in A} p_X(x) dx$ .
- ▶ We often write  $p(x)$  or  $P(A)$  for short, if  $X$  is clear

# Random Variables

- ▶ Intuitively, a random variable  $X$  is a variable that can take a **value**  $x$  from a set  $\Omega_X$  with some **probability**  $p_X(x) \in [0, 1]$ .
- ▶ We call the function  $p_X : \Omega_X \rightarrow \mathbb{R}^+$  with  $\int_{x \in \Omega_X} p_X(x) dx = 1$  a **probability density**
- ▶ A probability **distribution** is a function  $P_X$  that assigns probabilities to subsets  $A \subseteq \Omega_X$ , i.e.  $P(X \in A) := \int_{x \in A} p_X(x) dx$ .
- ▶ We often write  $p(x)$  or  $P(A)$  for short, if  $X$  is clear
- ▶ The precise definition is more complicated! Refer e.g. to Wikipedia



## $N$ coins

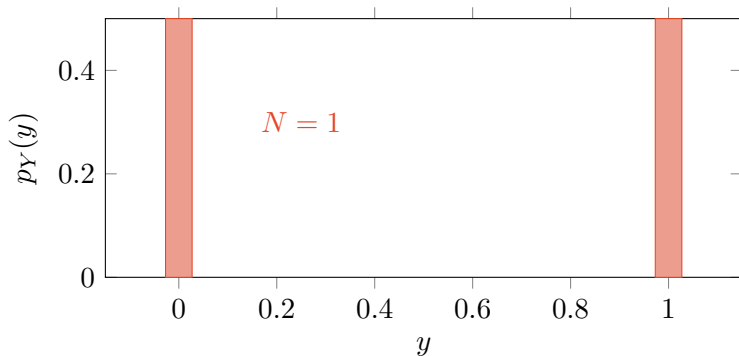
- ▶ Consider  $N$  random variables  $X_i$ , each representing a fair coin toss with  $\Omega = \{0, 1\}$  and  $p_{X_i}(0) = p_{X_i}(1) = \frac{1}{2}$ .

## $N$ coins

- ▶ Consider  $N$  random variables  $X_i$ , each representing a fair coin toss with  $\Omega = \{0, 1\}$  and  $p_{X_i}(0) = p_{X_i}(1) = \frac{1}{2}$ .
- ▶ Then, consider the **mean** as a random variable, i.e.  $Y := \frac{1}{N} \cdot (X_1 + \dots + X_N)$ .

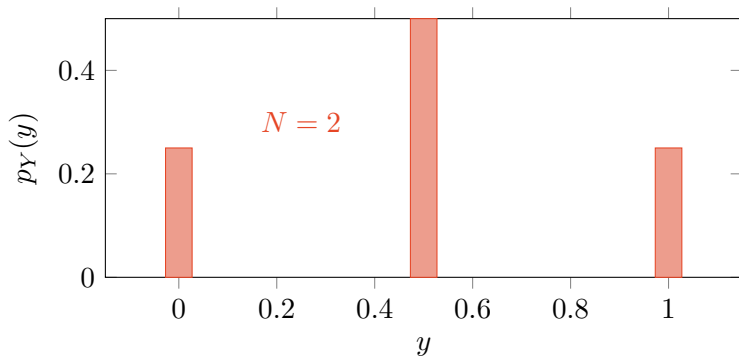
## $N$ coins

- ▶ Consider  $N$  random variables  $X_i$ , each representing a fair coin toss with  $\Omega = \{0, 1\}$  and  $p_{X_i}(0) = p_{X_i}(1) = \frac{1}{2}$ .
- ▶ Then, consider the **mean** as a random variable, i.e.  $Y := \frac{1}{N} \cdot (X_1 + \dots + X_N)$ .



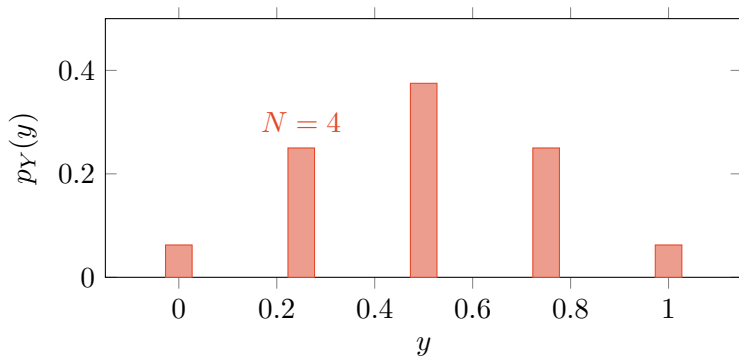
## $N$ coins

- ▶ Consider  $N$  random variables  $X_i$ , each representing a fair coin toss with  $\Omega = \{0, 1\}$  and  $p_{X_i}(0) = p_{X_i}(1) = \frac{1}{2}$ .
- ▶ Then, consider the **mean** as a random variable, i.e.  $Y := \frac{1}{N} \cdot (X_1 + \dots + X_N)$ .



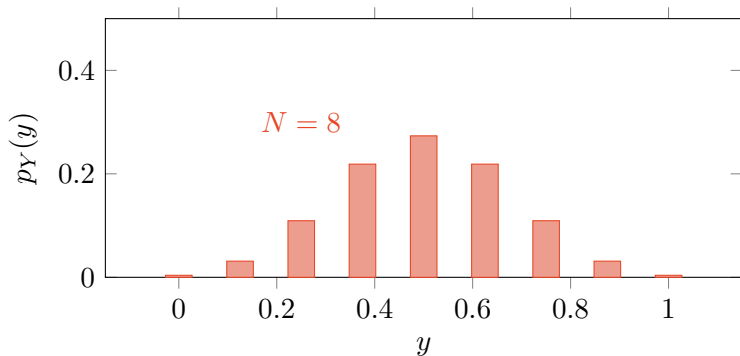
## $N$ coins

- ▶ Consider  $N$  random variables  $X_i$ , each representing a fair coin toss with  $\Omega = \{0, 1\}$  and  $p_{X_i}(0) = p_{X_i}(1) = \frac{1}{2}$ .
- ▶ Then, consider the **mean** as a random variable, i.e.  $Y := \frac{1}{N} \cdot (X_1 + \dots + X_N)$ .



## $N$ coins

- ▶ Consider  $N$  random variables  $X_i$ , each representing a fair coin toss with  $\Omega = \{0, 1\}$  and  $p_{X_i}(0) = p_{X_i}(1) = \frac{1}{2}$ .
- ▶ Then, consider the **mean** as a random variable, i.e.  $Y := \frac{1}{N} \cdot (X_1 + \dots + X_N)$ .



## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)

## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)
- ▶ Density:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$



## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)
- ▶ Density:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

- ▶  $\mu$  and  $\sigma$  are **parameters** of the density called **mean** and **standard deviation**

## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)
- ▶ Density:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

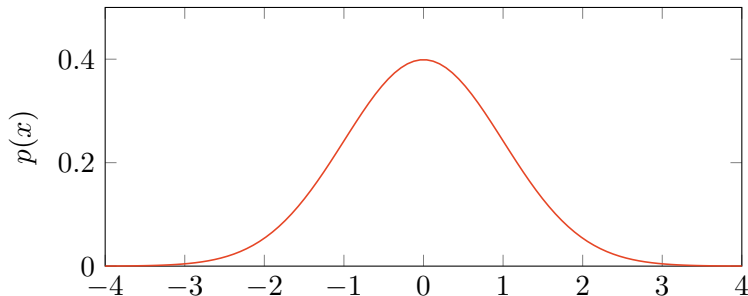
- ▶  $\mu$  and  $\sigma$  are **parameters** of the density called **mean** and **standard deviation**
- ▶ We often write  $p(x) = \mathcal{N}(x|\mu, \sigma)$  for short

## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)
- ▶ Density:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

- ▶  $\mu$  and  $\sigma$  are **parameters** of the density called **mean** and **standard deviation**
- ▶ We often write  $p(x) = \mathcal{N}(x|\mu, \sigma)$  for short

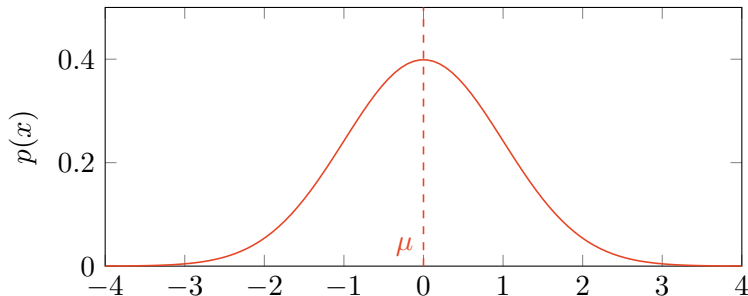


## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)
- ▶ Density:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

- ▶  $\mu$  and  $\sigma$  are **parameters** of the density called **mean** and **standard deviation**
- ▶ We often write  $p(x) = \mathcal{N}(x|\mu, \sigma)$  for short

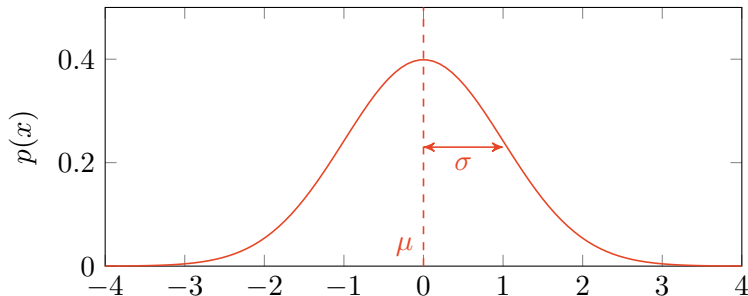


## The Gaussian density

- ▶ Any sum of many (i.i.d., non-trivial) random variables becomes **Gaussian**/normally distributed (central limit theorem)
- ▶ Density:

$$p(x) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

- ▶  $\mu$  and  $\sigma$  are **parameters** of the density called **mean** and **standard deviation**
- ▶ We often write  $p(x) = \mathcal{N}(x|\mu, \sigma)$  for short



## Dependent Densities

- ▶ Recall: Data generation model has three random variables: Input  $x$  with density  $p_x$ , noise  $\epsilon$  with density  $p_\epsilon$ , and output  $y$  with density  $p_y$

## Dependent Densities

- ▶ Recall: Data generation model has three random variables: Input  $x$  with density  $p_x$ , noise  $\epsilon$  with density  $p_\epsilon$ , and output  $p_y$  with density  $p_y$
- ▶ We can choose  $p_x$  and  $p_\epsilon$  independently (based on our belief about the data)

## Dependent Densities

- ▶ Recall: Data generation model has three random variables: Input  $x$  with density  $p_x$ , noise  $\epsilon$  with density  $p_\epsilon$ , and output  $p_y$  with density  $p_y$
- ▶ We can choose  $p_x$  and  $p_\epsilon$  independently (based on our belief about the data)
- ▶ But  $y$  **depends** on  $x$  and  $\epsilon$  :



# Dependent Densities

- ▶ Recall: Data generation model has three random variables: Input  $x$  with density  $p_x$ , noise  $\epsilon$  with density  $p_\epsilon$ , and output  $y$  with density  $p_y$
- ▶ We can choose  $p_x$  and  $p_\epsilon$  independently (based on our belief about the data)
- ▶ But  $y$  **depends** on  $x$  and  $\epsilon$  :  $y = f(x) + \epsilon$

# Dependent Densities

- ▶ Recall: Data generation model has three random variables: Input  $x$  with density  $p_x$ , noise  $\epsilon$  with density  $p_\epsilon$ , and output  $y$  with density  $p_y$
  - ▶ We can choose  $p_x$  and  $p_\epsilon$  independently (based on our belief about the data)
  - ▶ But  $y$  **depends** on  $x$  and  $\epsilon$  :  $y = f(x) + \epsilon$
- ⇒ We need a **joint** density  $p_{x,y}$

# Dependent Densities

- ▶ Recall: Data generation model has three random variables: Input  $x$  with density  $p_x$ , noise  $\epsilon$  with density  $p_\epsilon$ , and output  $y$  with density  $p_y$
  - ▶ We can choose  $p_x$  and  $p_\epsilon$  independently (based on our belief about the data)
  - ▶ But  $y$  **depends** on  $x$  and  $\epsilon$  :  $y = f(x) + \epsilon$
- ⇒ We need a **joint** density  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_\epsilon(\hat{y} - f(\hat{x}))$

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$
- ▶  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_{\epsilon}(\hat{y} - f(\hat{x}))$

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$
- ▶  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_{\epsilon}(\hat{y} - f(\hat{x}))$

<u><math>p_{x,y}</math></u>	1	2	3	4	5	6	7	8
-----------------------------	---	---	---	---	---	---	---	---

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$
- ▶  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_{\epsilon}(\hat{y} - f(\hat{x}))$

$p_{x,y}$	1	2	3	4	5	6	7	8
1	$\frac{1}{6} \cdot \frac{1}{4}$							



## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$
- ▶  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_{\epsilon}(\hat{y} - f(\hat{x}))$

$p_{x,y}$	1	2	3	4	5	6	7	8
1	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0	0	0	0

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$
- ▶  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_{\epsilon}(\hat{y} - f(\hat{x}))$

$p_{x,y}$	1	2	3	4	5	6	7	8
1	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0	0	0	0
2	0	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0	0	0

## Example: Joint Density

- ▶  $x$ : Fair six-sided die;  $f(x) = x + 1$ ;  $y = f(x) + \epsilon$
- ▶  $\epsilon$ :  $p_{\epsilon}(-1) = \frac{1}{4}, p_{\epsilon}(0) = \frac{1}{2}, p_{\epsilon}(1) = \frac{1}{4}$
- ▶  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_{\epsilon}(\hat{y} - f(\hat{x}))$

$p_{x,y}$	1	2	3	4	5	6	7	8
1	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0	0	0	0
2	0	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0	0	0
3	0	0	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0	0
4	0	0	0	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0	0
5	0	0	0	0	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$	0
6	0	0	0	0	0	$\frac{1}{24}$	$\frac{1}{12}$	$\frac{1}{24}$

## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$

## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$
- ▶ Conditional Density:  $p_{y|x=\hat{x}}(\hat{y}) = p_{x,y}(\hat{x}, \hat{y})/p_x(\hat{x})$

## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$
- ▶ Conditional Density:  $p_{y|x=\hat{x}}(\hat{y}) = p_{x,y}(\hat{x}, \hat{y})/p_x(\hat{x})$  (often  $p(\hat{y}|\hat{x})$  for short)

## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$
- ▶ Conditional Density:  $p_{y|x=\hat{x}}(\hat{y}) = p_{x,y}(\hat{x}, \hat{y})/p_x(\hat{x})$  (often  $p(\hat{y}|\hat{x})$  for short)
- ▶ we call  $x$  and  $y$  **independent** if  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_y(\hat{y})$  for all  $\hat{x}, \hat{y}$

## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$
- ▶ Conditional Density:  $p_{y|x=\hat{x}}(\hat{y}) = p_{x,y}(\hat{x}, \hat{y})/p_x(\hat{x})$  (often  $p(\hat{y}|\hat{x})$  for short)
- ▶ we call  $x$  and  $y$  **independent** if  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_y(\hat{y})$  for all  $\hat{x}, \hat{y}$

Example:

<u><math>\hat{x}</math></u>	<u><math>p_x(\hat{x})</math></u>	<u><math>p(1 \hat{x})</math></u>	<u><math>p(2 \hat{x})</math></u>	<u><math>p(3 \hat{x})</math></u>	<u><math>p(4 \hat{x})</math></u>	<u><math>p(5 \hat{x})</math></u>	<u><math>p(6 \hat{x})</math></u>	<u><math>p(7 \hat{x})</math></u>	<u><math>p(8 \hat{x})</math></u>
-----------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------



## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$
- ▶ Conditional Density:  $p_{y|x=\hat{x}}(\hat{y}) = p_{x,y}(\hat{x}, \hat{y})/p_x(\hat{x})$  (often  $p(\hat{y}|\hat{x})$  for short)
- ▶ we call  $x$  and  $y$  **independent** if  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_y(\hat{y})$  for all  $\hat{x}, \hat{y}$

Example:

$\hat{x}$	$p_x(\hat{x})$	$p(1 \hat{x})$	$p(2 \hat{x})$	$p(3 \hat{x})$	$p(4 \hat{x})$	$p(5 \hat{x})$	$p(6 \hat{x})$	$p(7 \hat{x})$	$p(8 \hat{x})$
1	$\frac{1}{6}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0	0	0

## Marginal & Conditional Density

- ▶ Marginal Density:  $p_x(\hat{x}) = \sum_{\hat{y} \in \Omega_y} p_{x,y}(\hat{x}, \hat{y})$
- ▶ Conditional Density:  $p_{y|x=\hat{x}}(\hat{y}) = p_{x,y}(\hat{x}, \hat{y})/p_x(\hat{x})$  (often  $p(\hat{y}|\hat{x})$  for short)
- ▶ we call  $x$  and  $y$  **independent** if  $p_{x,y}(\hat{x}, \hat{y}) = p_x(\hat{x}) \cdot p_y(\hat{y})$  for all  $\hat{x}, \hat{y}$

Example:

$\hat{x}$	$p_x(\hat{x})$	$p(1 \hat{x})$	$p(2 \hat{x})$	$p(3 \hat{x})$	$p(4 \hat{x})$	$p(5 \hat{x})$	$p(6 \hat{x})$	$p(7 \hat{x})$	$p(8 \hat{x})$
1	$\frac{1}{6}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0	0	0
2	$\frac{1}{6}$	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0	0
3	$\frac{1}{6}$	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0
4	$\frac{1}{6}$	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0
5	$\frac{1}{6}$	0	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0
6	$\frac{1}{6}$	0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$

## Construction

- ▶ Conceptually, we would start by specifying a full joint density and infer all properties (marginal, conditional, independence) from that

# Construction

- ▶ Conceptually, we would start by specifying a full joint density and infer all properties (marginal, conditional, independence) from that
- ▶ Practically, we start with independence assumptions, marginals, and conditionals and infer the joint density from that:  $p_{y,x}(\hat{x}, \hat{y}) = p_{y|x=\hat{x}}(\hat{y}) \cdot p_x(\hat{x})$

# Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent

## Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
- ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$

## Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
- ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$
- ▶ We assume **Gaussian** noise, i.e.  $p_\epsilon(\hat{\epsilon}) = \mathcal{N}(\hat{\epsilon}|0, \sigma)$  for some  $\sigma > 0$

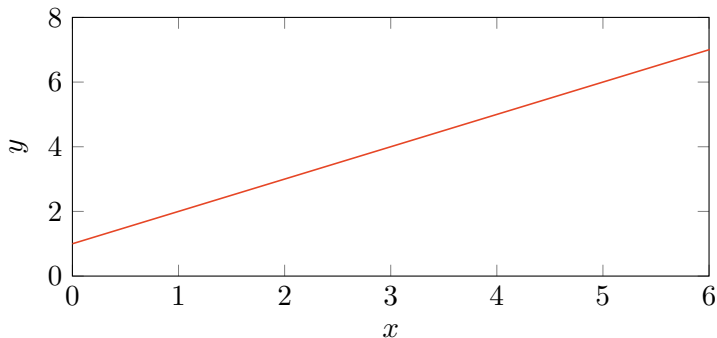
## Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
  - ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$
  - ▶ We assume **Gaussian** noise, i.e.  $p_\epsilon(\hat{\epsilon}) = \mathcal{N}(\hat{\epsilon}|0, \sigma)$  for some  $\sigma > 0$
- ⇒ conditional:  $p_{y|x=\hat{x}}(\hat{y}) = \mathcal{N}(\hat{y}|f(\hat{x}), \sigma)$  where  $f$  is the “true” underlying function



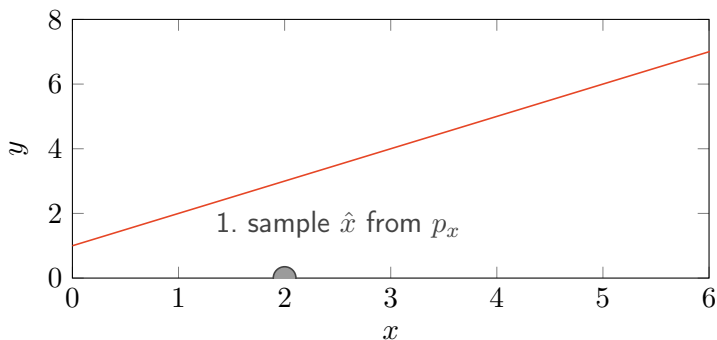
# Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
  - ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$
  - ▶ We assume **Gaussian** noise, i.e.  $p_\epsilon(\hat{\epsilon}) = \mathcal{N}(\hat{\epsilon}|0, \sigma)$  for some  $\sigma > 0$
- ⇒ conditional:  $p_{y|x=\hat{x}}(\hat{y}) = \mathcal{N}(\hat{y}|f(\hat{x}), \sigma)$  where  $f$  is the “true” underlying function



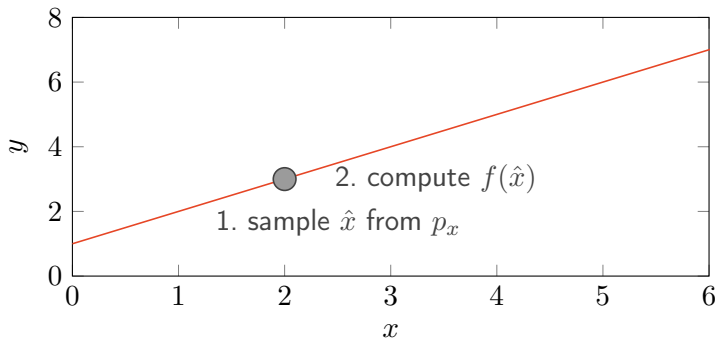
# Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
  - ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$
  - ▶ We assume **Gaussian** noise, i.e.  $p_\epsilon(\hat{\epsilon}) = \mathcal{N}(\hat{\epsilon}|0, \sigma)$  for some  $\sigma > 0$
- ⇒ conditional:  $p_{y|x=\hat{x}}(\hat{y}) = \mathcal{N}(\hat{y}|f(\hat{x}), \sigma)$  where  $f$  is the “true” underlying function



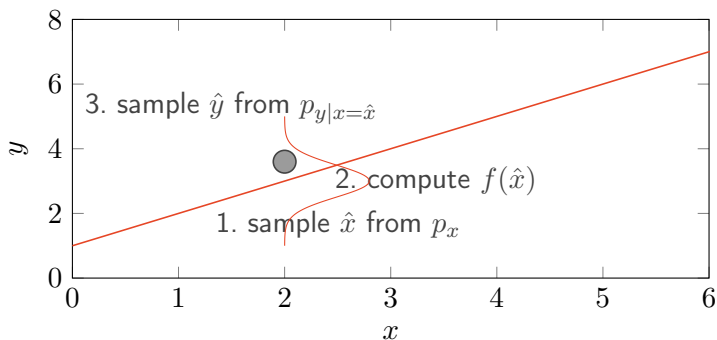
# Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
  - ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$
  - ▶ We assume **Gaussian** noise, i.e.  $p_\epsilon(\hat{\epsilon}) = \mathcal{N}(\hat{\epsilon}|0, \sigma)$  for some  $\sigma > 0$
- ⇒ conditional:  $p_{y|x=\hat{x}}(\hat{y}) = \mathcal{N}(\hat{y}|f(\hat{x}), \sigma)$  where  $f$  is the “true” underlying function



# Probabilistic Regression

- ▶ Recall: We assume that input and noise are independent
  - ▶ We assume **uniform** data generation, i.e.  $p_x(\hat{x}) = \frac{1}{C}$  for some constant  $C$
  - ▶ We assume **Gaussian** noise, i.e.  $p_\epsilon(\hat{\epsilon}) = \mathcal{N}(\hat{\epsilon}|0, \sigma)$  for some  $\sigma > 0$
- ⇒ conditional:  $p_{y|x=\hat{x}}(\hat{y}) = \mathcal{N}(\hat{y}|f(\hat{x}), \sigma)$  where  $f$  is the “true” underlying function



## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i)$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} -\log \left( \prod_{i=1}^N p_{x,y}(x_i, y_i) \right) \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N -\log(p_{x,y}(x_i, y_i)) \end{aligned}$$



## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N -\log(p_{y|x=x_i}(x_i, y_i) \cdot p_x(x_i)) \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N -\log(p_{y|x=x_i}(x_i, y_i)) - \log(p_x(x_i)) \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N -\log(p_{y|x=x_i}(x_i, y_i)) - \log\left(\frac{1}{C}\right) \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N -\log(\mathcal{N}(y_i | f(x_i), \sigma)) \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N -\log \left( \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp \left[ -\frac{1}{2} \frac{(f(x_i) - y_i)^2}{\sigma^2} \right] \right) \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N \frac{1}{2} \log(2\pi \cdot \sigma^2) + \frac{1}{2} \frac{(f(x_i) - y_i)^2}{\sigma^2} \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N (f(x_i) - y_i)^2 \end{aligned}$$

## Probabilistic Regression (II)

- ▶ Assume that example data set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$  is generated as above
- ▶ Find the function  $f$  that maximizes the **likelihood** of the dataset

$$\begin{aligned} & \max_{f \in \mathcal{F}} \prod_{i=1}^N p_{x,y}(x_i, y_i) \\ \iff & \min_{f \in \mathcal{F}} \sum_{i=1}^N (f(x_i) - y_i)^2 \end{aligned}$$

$\Rightarrow$  Equivalent to RMSE minimization



## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself

## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f})$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \max_{\hat{f} \in \mathcal{F}} p_{f,\mathcal{D}}(\hat{f}, \hat{\mathcal{D}}) / p_{\mathcal{D}}(\hat{\mathcal{D}}) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \max_{\hat{f} \in \mathcal{F}} p_{f,\mathcal{D}}(\hat{f}, \hat{\mathcal{D}}) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \max_{\hat{f} \in \mathcal{F}} p_{\mathcal{D}|f=\hat{f}}(\hat{\mathcal{D}}) \cdot p_f(\hat{f}) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \max_{\hat{f} \in \mathcal{F}} \left( \prod_{i=1}^N p_{y,x|f=\hat{f}}(y_i, x_i) \right) \cdot p_f(\hat{f}) \end{aligned}$$

## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\hat{f} \in \mathcal{F}} -\log \left( \prod_{i=1}^N p_{y,x|f=\hat{f}}(y_i, x_i) \right) - \log \left( p_f(\hat{f}) \right) \end{aligned}$$



## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\hat{f} \in \mathcal{F}} \sum_{i=1}^N -\log \left( p_{y,x|f=\hat{f}}(y_i, x_i) \right) - \log \left( p_f(\hat{f}) \right) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\hat{f} \in \mathcal{F}} \sum_{i=1}^N -\log \left( p_{y|x=x_i, f=\hat{f}}(y_i) \cdot p_x(x_i) \right) - \log \left( p_f(\hat{f}) \right) \end{aligned}$$

## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\hat{f} \in \mathcal{F}} \sum_{i=1}^N -\log \left( p_{y|x=x_i, f=\hat{f}}(y_i) \right) - \log \left( p_f(\hat{f}) \right) \end{aligned}$$

## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\hat{f} \in \mathcal{F}} \sum_{i=1}^N -\log \left( \mathcal{N}(y|\hat{f}(x_i), \sigma^2) \right) - \log \left( p_f(\hat{f}) \right) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\hat{f} \in \mathcal{F}} \sum_{i=1}^N \frac{1}{2} \frac{(\hat{f}(x_i) - y_i)^2}{\sigma^2} - \log(p_f(\hat{f})) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N \frac{1}{2} \frac{(\vec{w}^T \cdot \phi(x_i) - y_i)^2}{\sigma^2} - \log \left( \prod_{j=1}^n \mathcal{N}(w_n|0, \lambda) \right) \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N \frac{1}{2} \frac{(\vec{w}^T \cdot \phi(x_i) - y_i)^2}{\sigma^2} + \frac{1}{2} \sum_{j=1}^n \frac{w_j^2}{\lambda^2} \end{aligned}$$

# Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 + \frac{\sigma^2}{\lambda^2} \vec{w}^T \cdot \vec{w} \end{aligned}$$



## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 + \frac{\sigma^2}{\lambda^2} \vec{w}^T \cdot \vec{w} \end{aligned}$$

⇒ Equivalent to Regularized RMSE minimization

## Probabilistic Regression with Bayesian Flavour

- ▶ Consider  $f$  (or the parameters of  $f$ ) as a random variable itself
- ▶ Define a **prior**  $p_f$ , e.g.  $p_f(\hat{f}) = \mathcal{N}(w_1|0, \lambda) \cdot \dots \cdot \mathcal{N}(w_n|0, \lambda)$
- ▶ Maximize the **posterior** probability for the model

$$\begin{aligned} & \max_{\hat{f} \in \mathcal{F}} p_{f|\mathcal{D}=\hat{\mathcal{D}}}(\hat{f}) \\ \iff & \min_{\vec{w} \in \mathbb{R}^n} \sum_{i=1}^N (\vec{w}^T \cdot \phi(x_i) - y_i)^2 + \frac{\sigma^2}{\lambda^2} \vec{w}^T \cdot \vec{w} \end{aligned}$$

⇒ Equivalent to Regularized RMSE minimization

- ▶ Note: “Truly” Bayesian modelling would do an average across all models (Bishop 2006)

# Lessons from Machine Learning Theory



THE UNIVERSITY OF  
SYDNEY

## Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?

## Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

# Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

Definition: PAC (adapted from Shalev-Shwartz and Ben-David 2014)

Let  $p_{x,y}$  be a density over an input set  $\mathcal{X}$  and an output set  $\mathcal{Y}$ .

# Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

Definition: PAC (adapted from Shalev-Shwartz and Ben-David 2014)

Let  $p_{x,y}$  be a density over an input set  $\mathcal{X}$  and an output set  $\mathcal{Y}$ . We call a learning algorithm  $\mathcal{A}$  **probably approximately correct** with bounds  $\delta, \epsilon \in (0, 1)$  on  $p_{x,y}$

# Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

Definition: PAC (adapted from Shalev-Shwartz and Ben-David 2014)

Let  $p_{x,y}$  be a density over an input set  $\mathcal{X}$  and an output set  $\mathcal{Y}$ . We call a learning algorithm  $\mathcal{A}$  **probably approximately correct** with bounds  $\delta, \epsilon \in (0, 1)$  on  $p_{x,y}$  if a  $1 - \delta$  fraction of sufficiently large datasets  $\mathcal{D}$  sampled from  $p_{x,y}$



# Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

Definition: PAC (adapted from Shalev-Shwartz and Ben-David 2014)

Let  $p_{x,y}$  be a density over an input set  $\mathcal{X}$  and an output set  $\mathcal{Y}$ . We call a learning algorithm  $\mathcal{A}$  **probably approximately correct** with bounds  $\delta, \epsilon \in (0, 1)$  on  $p_{x,y}$  if a  $1 - \delta$  fraction of sufficiently large datasets  $\mathcal{D}$  sampled from  $p_{x,y}$  yield a model  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$

# Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

Definition: PAC (adapted from Shalev-Shwartz and Ben-David 2014)

Let  $p_{x,y}$  be a density over an input set  $\mathcal{X}$  and an output set  $\mathcal{Y}$ . We call a learning algorithm  $\mathcal{A}$  **probably approximately correct** with bounds  $\delta, \epsilon \in (0, 1)$  on  $p_{x,y}$  if a  $1 - \delta$  fraction of sufficiently large datasets  $\mathcal{D}$  sampled from  $p_{x,y}$  yield a model  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$  such that

$$\int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\mathcal{D}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \leq \epsilon$$

# Probably Approximately Correct (PAC)

- ▶ Question: Under a probabilistic view, what does generalization mean?
- ▶ Intuitively: An algorithm generalizes if, given sufficient data, it **probably** finds a model that has **low error** on the **'true' data distribution**.

Definition: PAC (adapted from Shalev-Shwartz and Ben-David 2014)

Let  $p_{x,y}$  be a density over an input set  $\mathcal{X}$  and an output set  $\mathcal{Y}$ . We call a learning algorithm  $\mathcal{A}$  **probably approximately correct** with bounds  $\delta, \epsilon \in (0, 1)$  on  $p_{x,y}$  if a  $1 - \delta$  fraction of sufficiently large datasets  $\mathcal{D}$  sampled from  $p_{x,y}$  yield a model  $f_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$  such that

$$\int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\mathcal{D}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \leq \epsilon$$

Proving PAC properties is one of the key objectives in **Statistical Machine Learning Theory** (Shalev-Shwartz and Ben-David 2014).

## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?

## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?
- ▶ Let  $f_{\mathcal{D}} := \mathcal{A}(\mathcal{D})$  and let  $\bar{f}(x) := \int f_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(\mathcal{D}) d\mathcal{D}$

## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?
- ▶ Let  $f_{\mathcal{D}} := \mathcal{A}(\mathcal{D})$  and let  $\bar{f}(x) := \int f_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(\mathcal{D}) d\mathcal{D}$
- ▶ Consider for some  $(x, y)$  the average error a learned model will make.

## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?
- ▶ Let  $f_{\mathcal{D}} := \mathcal{A}(\mathcal{D})$  and let  $\bar{f}(x) := \int f_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(\mathcal{D}) d\mathcal{D}$
- ▶ Consider for some  $(x, y)$  the average error a learned model will make.

$$\int (f_{\hat{\mathcal{D}}}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$

## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?
- ▶ Let  $f_{\mathcal{D}} := \mathcal{A}(\mathcal{D})$  and let  $\bar{f}(x) := \int f_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(\mathcal{D}) d\mathcal{D}$
- ▶ Consider for some  $(x, y)$  the average error a learned model will make.

$$\int (f_{\hat{\mathcal{D}}}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} = \int (f_{\hat{\mathcal{D}}}(x) - \bar{f}(x) + \bar{f}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$



## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?
- ▶ Let  $f_{\mathcal{D}} := \mathcal{A}(\mathcal{D})$  and let  $\bar{f}(x) := \int f_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(\mathcal{D}) d\mathcal{D}$
- ▶ Consider for some  $(x, y)$  the average error a learned model will make.

$$\begin{aligned} \int (f_{\hat{\mathcal{D}}}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} &= \int (f_{\hat{\mathcal{D}}}(x) - \bar{f}(x) + \bar{f}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} \\ &= \underbrace{\int (f_{\hat{\mathcal{D}}}(x) - \bar{f}(x))^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}}_{\text{Variance / "Overcuriosity"}} + \underbrace{(\bar{f}(x) - y)^2}_{\text{Bias / "Undercuriosity"}} \end{aligned}$$

## Bias-Variance decomposition

- ▶ Question: Why does regularization help for generalization?
- ▶ Let  $f_{\mathcal{D}} := \mathcal{A}(\mathcal{D})$  and let  $\bar{f}(x) := \int f_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(\mathcal{D}) d\mathcal{D}$
- ▶ Consider for some  $(x, y)$  the average error a learned model will make.

$$\begin{aligned} \int (f_{\hat{\mathcal{D}}}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} &= \int (f_{\hat{\mathcal{D}}}(x) - \bar{f}(x) + \bar{f}(x) - y)^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} \\ &= \underbrace{\int (f_{\hat{\mathcal{D}}}(x) - \bar{f}(x))^2 \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}}_{\text{Variance / "Overcuriosity"}} + \underbrace{(\bar{f}(x) - y)^2}_{\text{Bias / "Undercuriosity"}} \end{aligned}$$

Regularization can strongly reduce variance while slightly increasing bias  $\Rightarrow$  better generalization

# Crossvalidation and Experimental Design



THE UNIVERSITY OF  
SYDNEY

## Crossvalidation

- ▶ PAC is based on the 'true' data distribution

## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know

## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$

## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} \\ \approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^C}(\hat{x}) - \hat{y})^2$$



## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

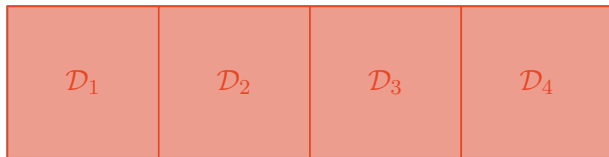
$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$
$$\approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^C}(\hat{x}) - \hat{y})^2$$



## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

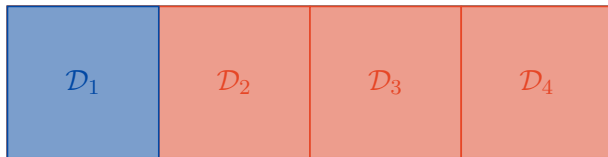
$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$
$$\approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^c}(\hat{x}) - \hat{y})^2$$



## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$
$$\approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^C}(\hat{x}) - \hat{y})^2$$



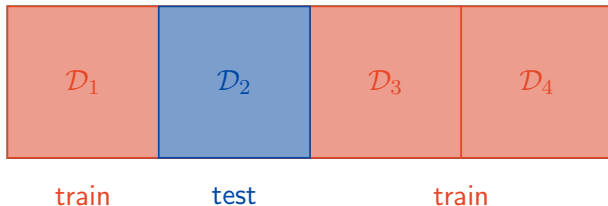
test

train

## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

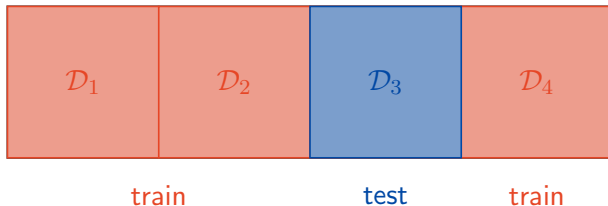
$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$
$$\approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^C}(\hat{x}) - \hat{y})^2$$



## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

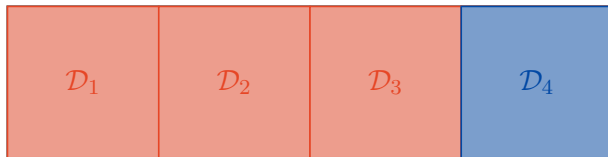
$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}}$$
$$\approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^c}(\hat{x}) - \hat{y})^2$$



## Crossvalidation

- ▶ PAC is based on the 'true' data distribution ... which we don't know
- ▶ But we can approximate the distribution over all possible data sets by constructing **disjoint subsets**  $\mathcal{D}_1, \dots, \mathcal{D}_M$  of our data set  $\mathcal{D}$

$$\int \left( \int_{\mathcal{X}} \int_{\mathcal{Y}} (f_{\hat{\mathcal{D}}}(\hat{x}) - \hat{y})^2 \cdot p_{x,y}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \right) \cdot p_{\mathcal{D}}(\hat{\mathcal{D}}) d\hat{\mathcal{D}} \\ \approx \frac{1}{M} \sum_{j=1}^M \frac{1}{|\mathcal{D}_j|} \sum_{(\hat{x}, \hat{y}) \in \mathcal{D}_j} (f_{\mathcal{D}_j^c}(\hat{x}) - \hat{y})^2$$



train

test

## How to design a ML experiment

Setting: We have a cool new learning algorithm  $\mathcal{A}$  and want to compare it against baseline learning algorithms  $\mathcal{A}_1, \dots, \mathcal{A}_K$

## How to design a ML experiment

Setting: We have a cool new learning algorithm  $\mathcal{A}$  and want to compare it against baseline learning algorithms  $\mathcal{A}_1, \dots, \mathcal{A}_K$

- ▶ Get the best possible reference implementation for each baseline  $\mathcal{A}_1, \dots, \mathcal{A}_K$



## How to design a ML experiment

Setting: We have a cool new learning algorithm  $\mathcal{A}$  and want to compare it against baseline learning algorithms  $\mathcal{A}_1, \dots, \mathcal{A}_K$

- ▶ Get the best possible reference implementation for each baseline  $\mathcal{A}_1, \dots, \mathcal{A}_K$
- ▶ Collect multiple (!) data sets that represent your target domain well

## How to design a ML experiment

Setting: We have a cool new learning algorithm  $\mathcal{A}$  and want to compare it against baseline learning algorithms  $\mathcal{A}_1, \dots, \mathcal{A}_K$

- ▶ Get the best possible reference implementation for each baseline  $\mathcal{A}_1, \dots, \mathcal{A}_K$
- ▶ Collect multiple (!) data sets that represent your target domain well
- ▶ Approximate the generalization error for each algorithm on each data set using crossvalidation

## How to design a ML experiment

Setting: We have a cool new learning algorithm  $\mathcal{A}$  and want to compare it against baseline learning algorithms  $\mathcal{A}_1, \dots, \mathcal{A}_K$

- ▶ Get the best possible reference implementation for each baseline  $\mathcal{A}_1, \dots, \mathcal{A}_K$
- ▶ Collect multiple (!) data sets that represent your target domain well
- ▶ Approximate the generalization error for each algorithm on each data set using crossvalidation
- ▶ Use  $K$  paired statistical tests to compare the errors of  $\mathcal{A}$  with each  $\mathcal{A}_1, \dots, \mathcal{A}_K$  (e.g. Wilcoxon sign-rank test); use Bonferroni correction

## How to design a ML experiment

Setting: We have a cool new learning algorithm  $\mathcal{A}$  and want to compare it against baseline learning algorithms  $\mathcal{A}_1, \dots, \mathcal{A}_K$

- ▶ Get the best possible reference implementation for each baseline  $\mathcal{A}_1, \dots, \mathcal{A}_K$
- ▶ Collect multiple (!) data sets that represent your target domain well
- ▶ Approximate the generalization error for each algorithm on each data set using crossvalidation
- ▶ Use  $K$  paired statistical tests to compare the errors of  $\mathcal{A}$  with each  $\mathcal{A}_1, \dots, \mathcal{A}_K$  (e.g. Wilcoxon sign-rank test); use Bonferroni correction
- ▶ If too few datasets: Use the errors in each crossvalidation fold

# Example: Reading a typical machine learning paper

## Tree Edit Distance Learning via Adaptive Symbol Embeddings

Benjamin Paalen<sup>1</sup> Claudio Gallicchio<sup>2</sup> Alessio Micheli<sup>2</sup> Barbara Hammer<sup>1</sup>

### Abstract

Metric learning has the aim to improve classification accuracy by learning a distance measure which brings data points from the same class closer together and pushes data points from different classes further apart. Recent research has demonstrated that metric learning approaches can also be applied to trees, such as molecular structures, abstract syntax trees of computer programs, or syntax trees of natural language, by learning the cost function of an edit distance, i.e. the costs of replacing, deleting, or inserting nodes in a tree. However, learning such costs directly may yield an edit distance which violates metric axioms, is challenging to interpret, and may not generalize well. In this contribution, we propose a novel metric learning approach for trees which we call embedding edit distance learning (BEDL) and which learns an edit distance indirectly by embedding the tree nodes as vectors, such that the Euclidean distance between those vectors supports class discrimination. We learn such embeddings by reducing the distance to prototypical trees from the same class and increasing the distance to prototypical trees from different classes. In our experiments, we show that BEDL improves upon the state-of-the-art in metric learning for trees on six benchmark data sets, ranging from computer science over biomedical data to a natural language processing data set containing over 300,000 nodes.

### 1. Introduction

Many classification approaches in machine learning explicitly or implicitly rely on some measure of distance (Kulis, 2013; Bellet et al., 2014; Mokbel et al., 2015). This is par-

ticularly apparent in case of the  $k$ -nearest neighbor classifier which classifies data points by assigning the label of the majority of the  $k$  nearest neighbors according to a given distance (Cover & Hart, 1967); or in case of learning vector quantization approaches which classify data points by assigning the label of the closest prototype according to a given distance (Kohonen, 1995). The success of such machine learning approaches hinges on the distance being *discriminative*, that is, data points from the same class being generally closer compared to data points from different classes. If the distance does not fulfill this criterion, one has to *adapt or learn* the distance measure with respect to the data, which is the topic of *metric learning* (Kulis, 2013; Bellet et al., 2014).

Most prior research in metric learning has focused on learning a generalization of the Euclidean distance according to some cost function (Kulis, 2013; Bellet et al., 2014). However, the Euclidean distance is not applicable to non-vectorial data, such as protein sequences, abstract syntax trees of computer programs, or syntax trees of natural language. To process those kinds of data, *edit distances* are a popular option, in particular the tree edit distance (Zhang & Shasha, 1989). In this contribution, we develop a novel metric learning scheme for the tree edit distance which we call embedding edit distance learning (BEDL).

While past research on metric learning for trees does exist (Bellet et al., 2014), BEDL goes beyond the state-of-the-art in multiple aspects:

- Based on the work of Bellet et al. (2012), we provide a generalized re-formulation of the edit distance which lends itself to metric learning, and can be applied to any kind of edit distance which uses replacement, deletion, and insertion operations. Furthermore, we consider not only one optimal edit script for metric learning, but all co-optimal edit scripts via a novel forward-backward algorithm.
- Our approach requires only a linear number of data tuples for metric learning, as we represent classes by few *prototypes*, which are selected via median learning vector quantization (Nebel et al., 2015).
- Most importantly, we do not directly learn the operation costs for the string edit distance, but instead

arXiv:1806.05009v3 [cs.LG] 16 Jul 2018

<sup>1</sup>Cognitive Interaction Technology, Bielefeld University, Germany  
<sup>2</sup>Department of Computer Science, University of Pisa, Italy  
Correspondence to: Benjamin Paalen <bpaalen@techfak.uni-bielefeld.de>

Proceedings of the 35<sup>th</sup> International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

# Example: Reading a typical machine learning paper

We did a cool new algorithm;  
it can do x

arXiv:1806.05009v3 [cs.LG] 16 Jul 2018

## Tree Edit Distance Learning via Adaptive Symbol Embeddings

Benjamin Paullen<sup>1</sup> Claudio Gallicchio<sup>2</sup> Alessio Micheli<sup>2</sup> Barbara Hammer<sup>1</sup>

### Abstract

Metric learning has the aim to improve classification accuracy by learning a distance measure which brings data points from the same class closer together and pushes data points from different classes further apart. Recent research has demonstrated that metric learning approaches can also be applied to trees, such as molecular structures, abstract syntax trees of computer programs, or syntax trees of natural language, by learning the cost function of an edit distance, i.e. the costs of replacing, deleting, or inserting nodes in a tree. However, learning such costs directly may yield an edit distance which violates metric axioms, is challenging to interpret, and may not generalize well. In this contribution, we propose a novel metric learning approach for trees which we call embedding edit distance learning (BEDL) and which learns an edit distance indirectly by embedding the tree nodes as vectors, such that the Euclidean distance between those vectors supports class discrimination. We learn such embeddings by reducing the distance to prototypical trees from the same class and increasing the distance to prototypical trees from different classes. In our experiments, we show that BEDL improves upon the state-of-the-art in metric learning for trees on six benchmark data sets, ranging from computer science over biomedical data to a natural language processing data set containing over 300,000 nodes.

### 1. Introduction

Many classification approaches in machine learning explicitly or implicitly rely on some measure of distance (Kulis, 2013; Bellet et al., 2014; Mokbel et al., 2015). This is par-

ticularly apparent in case of the  $k$ -nearest neighbor classifier which classifies data points by assigning the label of the majority of the  $k$  nearest neighbors according to a given distance (Cover & Hart, 1967); or in case of learning vector quantization approaches which classify data points by assigning the label of the closest prototype according to a given distance (Kohonen, 1995). The success of such machine learning approaches hinges on the distance being *discriminative*, that is, data points from the same class being generally closer compared to data points from different classes. If the distance does not fulfill this criterion, one has to *adapt or learn* the distance measure with respect to the data, which is the topic of *metric learning* (Kulis, 2013; Bellet et al., 2014).

Most prior research in metric learning has focused on learning a generalization of the Euclidean distance according to some cost function (Kulis, 2013; Bellet et al., 2014). However, the Euclidean distance is not applicable to non-vectorial data, such as protein sequences, abstract syntax trees of computer programs, or syntax trees of natural language. To process those kinds of data, *edit distances* are a popular option, in particular the tree edit distance (Zhang & Shasha, 1989). In this contribution, we develop a novel metric learning scheme for the tree edit distance which we call embedding edit distance learning (BEDL).

While past research on metric learning for trees does exist (Bellet et al., 2014), BEDL goes beyond the state-of-the-art in multiple aspects:

- Based on the work of Bellet et al. (2012), we provide a generalized re-formulation of the edit distance which lends itself to metric learning, and can be applied to any kind of edit distance which uses replacement, deletion, and insertion operations. Furthermore, we consider not only one optimal edit script for metric learning, but all co-optimal edit scripts via a novel forward-backward algorithm.
- Our approach requires only a linear number of data tuples for metric learning, as we represent classes by few *prototypes*, which are selected via median learning vector quantization (Nebel et al., 2015).
- Most importantly, we do not directly learn the operation costs for the string edit distance, but instead

<sup>1</sup>Cognitive Interaction Technology, Bielefeld University, Germany  
<sup>2</sup>Department of Computer Science, University of Pisa, Italy  
Correspondence to: Benjamin Paullen <bpaullen@techfak.uni-bielefeld.de>

Proceedings of the 35<sup>th</sup> International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

# Example: Reading a typical machine learning paper

arXiv:1806.05009v3 [cs.LG] 16 Jul 2018

## Tree Edit Distance Learning via Adaptive Symbol Embeddings

Benjamin Paullen<sup>1</sup> Claudio Gallicchio<sup>2</sup> Alessio Michel<sup>2</sup> Barbara Hammer<sup>1</sup>

### Abstract

Metric learning has the aim to improve classification accuracy by learning a distance measure which brings data points from the same class closer together and pushes data points from different classes further apart. Recent research has demonstrated that metric learning approaches can also be applied to trees, such as molecular structures, abstract syntax trees of computer programs, or syntax trees of natural language, by learning the cost function of an edit distance, i.e. the costs of replacing, deleting, or inserting nodes in a tree. However, learning such costs directly may yield an edit distance which violates metric axioms, is challenging to interpret, and may not generalize well. In this contribution, we propose a novel metric learning approach for trees which we call embedding edit distance learning (BEDL) and which learns an edit distance indirectly by embedding the tree nodes as vectors, such that the Euclidean distance between those vectors supports class discrimination. We learn such embeddings by reducing the distance to prototypical trees from the same class and increasing the distance to prototypical trees from different classes. In our experiments, we show that BEDL improves upon the state-of-the-art in metric learning for trees on six benchmark data sets, ranging from computer science over biomedical data to a natural language processing data set containing over 300,000 nodes.

### 1. Introduction

Many classification approaches in machine learning explicitly or implicitly rely on some measure of distance (Kulis, 2013; Bellet et al., 2014; Mokbel et al., 2015). This is par-

ticularly apparent in case of the  $k$ -nearest neighbor classifier which classifies data points by assigning the label of the majority of the  $k$  nearest neighbors according to a given distance (Cover & Hart, 1967); or in case of learning vector quantization approaches which classify data points by assigning the label of the closest prototype according to a given distance (Kohonen, 1995). The success of such machine learning approaches hinges on the distance being *discriminative*, that is, data points from the same class being generally closer compared to data points from different classes. If the distance does not fulfill this criterion, one has to *adapt or learn* the distance measure with respect to the data, which is the topic of *metric learning* (Kulis, 2013; Bellet et al., 2014).

Most prior research in metric learning has focused on learning a generalization of the Euclidean distance according to some cost function (Kulis, 2013; Bellet et al., 2014). However, the Euclidean distance is not applicable to non-vectorial data, such as protein sequences, abstract syntax trees of computer programs, or syntax trees of natural language. To process these kinds of data, *edit distances* are a popular option, in particular the tree edit distance (Zhang & Shasha, 1989). In this contribution, we develop a novel metric learning scheme for the tree edit distance which we call embedding edit distance learning (BEDL).

While past research on metric learning for trees does exist (Bellet et al., 2014), BEDL goes beyond the state-of-the-art in multiple aspects:

- Based on the work of Bellet et al. (2012), we provide a generalized re-formulation of the edit distance which lends itself to metric learning, and can be applied to any kind of edit distance which uses replacement, deletion, and insertion operations. Furthermore, we consider not only one optimal edit script for metric learning, but all co-optimal edit scripts via a novel forward-backward algorithm.
- Our approach requires only a linear number of data tuples for metric learning, as we represent classes by few *prototypes*, which are selected via median learning vector quantization (Nebel et al., 2015).
- Most importantly, we do not directly learn the operation costs for the string edit distance, but instead

This is what we can do better than before

<sup>1</sup>Cognitive Interaction Technology, Bielefeld University, Germany  
<sup>2</sup>Department of Computer Science, University of Pisa, Italy  
Correspondence to: Benjamin Paullen <bpaulen@techfak.uni-bielefeld.de>

Proceedings of the 35<sup>th</sup> International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

# Example: Reading a typical machine learning paper

We checked that this hasn't been done before, really!

16 14 13 12 11 10 9 8 7 6 5 4 3 2 1

## Tree Edit Distance Learning via Adaptive Symbol Embeddings

learn a vectorial embedding of the label alphabet for our data structures, which yields Euclidean operation costs. This re-formulation ensures that the resulting edit distance conforms to all metric axioms. Further, we can interpret the resulting embedding vectors via visualization, their pairwise distances and norms.

We begin by discussing related work, then we describe BEDL in more detail, and finally we evaluate BEDL experimentally and discuss the results.

### 2. Related Work

Our work is related to multiple areas of machine learning, most notably distances on structured data, metric learning, and vector embeddings.

In the past decades, multiple distance measures for structured data - i.e. sequences, trees, and graphs - have been suggested. In particular, one could define a distance based on existing string, tree, and graph kernels (Du San Martino & Sperduti, 2010), such as Weisfeiler-Lehman Graph Kernels (Shervashidze et al., 2011), topological distance-based tree kernels (Aiolli et al., 2015), or deep graph kernels (Yang et al. & Vishwanathan, 2015). Such kernels achieve state-of-the-art results on structured data and can be adapted to training data via multiple-kernel learning (Aiolli & Donini, 2015), or kernels based on Hidden-Markov-Model states (Bacciu et al., 2018). Kernels, however, have drawbacks in terms of interpretability, as a higher distance value does not necessarily relate to any kind of intuitive difference between the input trees. Further, kernel matrices are by definition limited to be positive semi-definite, which may be an undue restriction for certain data sets (Schleif & Tino, 2015).

If one strives for an interpretable measure of distance, *edit distances* are a popular choice, for example for the comparison of protein sequences in bioinformatics (Smith & Waterman, 1981), or abstract syntax trees for intelligent tutoring systems (Pudenz et al., 2018). Here, we focus on the tree edit distance, which permits deletions, insertions, and replacements of single nodes to transform an ordered tree  $\hat{x}$  into another ordered tree  $\hat{y}$  (Zhang & Shasha, 1989). Such ordered trees are the most general data structures which can still be treated efficiently via edit distances, as edit distances on unordered trees and general graphs are provably NP-hard (Zhang et al., 1992; Zeng et al., 2009). Furthermore, the tree edit distance includes the edit distance on sequences as a special case, such that it can be seen as a representative for edit distances as such.

Metric learning for the tree edit distance corresponds to adapting the costs of edit operations in order to bring trees from the same class closer and push trees from different classes further apart. Almost all of the existing approaches,

however, only bring trees from the same class closer together (Bellet et al., 2014). For example, Boyer et al. (2007) have proposed to replace the tree edit distance by the negative log probability of all tree edit scripts which transform the left input tree  $\hat{x}$  into the right input tree  $\hat{y}$ . Accordingly, the costs of edit operations change to probabilities of replacing, deleting, or inserting a certain label. These edit probabilities are adapted to maximize the probability that trees from the same class are edited into each other (Boyer et al., 2007). To replace generative models by discriminative ones, Bellet et al. (2012, 2016), have proposed to learn an edit distance  $d$ , such that the corresponding similarity  $2 \cdot \exp[-d(\hat{x}, \hat{y})] - 1$  is "good" as defined by the goodness-framework of (Balcan et al., 2008). Goodness according to this framework means that a linear separator with low error between the classes exists in the space of similarities (Balcan et al., 2008; Bellet et al., 2012). Bellet et al. (2012) have experimentally shown that this approach outperforms generative edit distance metric learning and have also established generalization guarantees based on the goodness framework. Therefore, this *good edit similarity learning* (GESL) approach of Bellet et al. (2012) is our main reference method.

Our novel approach is strongly inspired by GESL. However, our approach goes beyond GESL in key aspects. First, we utilize a different cost function, namely the generalized learning vector quantization (GLVQ) cost function, which quantifies how much closer every data point is to the closest prototypical example from the same class compared to the closest prototypical example from another class (Sato & Yamada, 1995). Just as GESL, LVQ methods are theoretically well-justified because it yields a maximum-margin classifier (Schneider et al., 2009), and have been successfully applied for metric learning on the string edit distance in the past (Mokbel et al., 2015; Pudenz et al., 2016). However, in contrast to GESL, GLVQ also provides a principled way to select prototypical examples for metric learning, and is flexible enough to not only learn a cost matrix, but also a vectorial embedding of the tree labels, such that the Euclidean distance on these embeddings provides a discriminative cost function.

While embedding approaches are common in the literature, prior work has focused mostly on embedding trees as a whole, for example via graph kernel approaches (Aiolli et al., 2015; Bacciu et al., 2018; Du San Martino & Sperduti, 2010; Shervashidze et al., 2011; Yang et al. & Vishwanathan, 2015), recursive neural networks (Gallicchio & Micheli, 2013; Hammer et al., 2007; Socher et al., 2013), or dimensionality reduction approaches (Gisbrecht et al., 2015). In this contribution, we wish to obtain an embedding for the single elements of a tree and maintain the tree structure. As of yet, such approaches only exist for sequences, namely in the form of



# Example: Reading a typical machine learning paper

We are very good and math and our method is justified

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Tree Edit Distance Learning via Adaptive Symbol Embeddings	
recurrent neural network for natural language processing tasks (Cho et al., 2014; Sutskever et al., 2014). In addition to word embeddings for trees, our approach also provides a corresponding tree edit distance, which is optimized for classification, and offers an intuitive view on the data, supporting applications like intelligent tutoring systems (Paufler et al., 2018).	for any $x \in \mathcal{X}$ , and such that the triangular inequality is fulfilled.
<b>3. Background</b>	<b>Theorem 1.</b> Let $c$ be a pseudo-metric on $\mathcal{X} \cup \{-\}$ . Then, the corresponding tree edit distance $d_c(x, y)$ can be computed in $\mathcal{O}( x ^2 \cdot  y ^2)$ using a dynamic programming scheme.
In this section, we revisit the basic problem of tree edit distance learning by first introducing the tree edit distance of Zhang & Shasha (1989), as well as the metric learning formalization suggested by Bellet et al. (2012).	Conversely, if $c$ violates the triangular inequality, the dynamic programming scheme overestimates the tree edit distance.
<b>3.1. Tree Edit Distance</b>	<i>Proof.</i> Refer to Zhang & Shasha (1989) for a proof of the first claim, and refer to the supplementary material (Paufler, 2018b) for a proof of the second claim. $\square$
We define a tree $\bar{x}$ over some set $\mathcal{X}$ as $x(\bar{x}_1, \dots, \bar{x}_R)$ where $x \in \mathcal{X}$ and $\bar{x}_1, \dots, \bar{x}_R$ is a (potentially empty) list of trees over $\mathcal{X}$ . We denote the set of all possible trees over $\mathcal{X}$ as $\mathcal{T}(\mathcal{X})$ . Further, we call $x$ the label of the tree. We define the size of a tree $x(\bar{x}_1, \dots, \bar{x}_R)$ as $ x  := 1 + \sum_{r=1}^R  \bar{x}_r $ . Finally, we call a list of trees $\bar{x}_1, \dots, \bar{x}_R$ a forest. Note that every tree is also a forest.	Beyond enabling us to compute the tree edit distance efficiently, pseudo-metric cost functions $c$ also ensure that the resulting tree edit distance $d_c$ is a pseudo-metric itself.
Next, we introduce edits over trees. In general, a tree edit $\delta$ is a function which transforms a forest into a forest (Paufler et al., 2018). In this particular case, we are only concerned with three kinds of edits, namely deletions, which remove a certain label from a forest; insertions, which insert a certain label into a forest; and replacements, which remove a certain label from a forest and put another label in its place. For example, deleting $x$ from a tree $x(y, z)$ results in the forest $y, z$ . Inserting $x$ into this forest as parent of $y$ results in the forest $x(y), z$ . Finally, replacing $z$ with $q$ in this forest results in the forest $x(y), q$ .	<b>Theorem 2.</b> Let $c$ be a pseudo-metric on $\mathcal{X} \cup \{-\}$ . Then, the corresponding tree edit distance $d_c$ is a pseudo-metric on the set of possible trees over $\mathcal{X}$ .
We associate each edit with a cost via a function $c: (\mathcal{X} \cup \{-\})^2 \rightarrow \mathbb{R}$ . In particular, we define the cost of a deletion of label $x$ as $c(x, -)$ , the cost of an insertion of label $y$ as $c(-, y)$ , and the cost of a replacement of label $x$ with label $y$ as $c(x, y)$ . We define the cost of a sequence of edits $\delta_1, \dots, \delta_T$ as the sum over the costs of all edits.	However, if $c$ violates any of the pseudo-metric properties (except for the triangular inequality), we can construct examples such that $d_c$ violates the same pseudo-metric properties.
Finally, we define the tree edit distance $d_c(\bar{x}, \bar{y})$ between any two trees $\bar{x}$ and $\bar{y}$ according to $c$ as the cost of the cheapest sequence of edits that transforms $\bar{x}$ to $\bar{y}$ . For example, if all edits have a cost of 1, the edit distance $d_c(x(y, z), q(z(q)))$ between the trees $x(y, z)$ and $q(z(q))$ is 3 because the cheapest sequence of edits is to replace $x$ with $q$ , delete $y$ , and insert $q$ .	<i>Proof.</i> Refer to the supplementary material (Paufler, 2018b). $\square$
Zhang & Shasha (1989) showed that the tree edit distance can be computed efficiently using a dynamic programming algorithm if $c$ is a pseudo-metric, meaning that $c$ is a non-negative and symmetric function, such that $c(x, x) = 0$	Both of these theorems make a pseudo-metric cost function $c$ desirable. However, ensuring pseudo-metric properties on $c$ may be challenging in metric learning, which is one of our key motivations for vectorial embeddings.
	<b>3.2. Tree Edit Distance Learning</b>
	Tree edit distance learning essentially means to adapt the cost function $c$ , such that the resulting tree edit distance $d_c$ is better suited for the task at hand. Following Bellet et al. (2012, 2014), we frame tree edit distance learning as minimizing some loss function over a set of positive pairs of trees $P \subset \mathcal{T}(\mathcal{X})^2$ and negative pairs of trees $N \subset \mathcal{T}(\mathcal{X})^2$ , that is, trees which should be close and far away respectively. In particular, given a loss function $L$ we wish to solve the optimization problem:
	$\min E(d_c, P, N) \quad (1)$
	In our contribution, we build upon the good edit similarity learning (GESL) approach of Bellet et al. (2012), who

# Example: Reading a typical machine learning paper

If you want to repeat our experiments, you need to do all this stuff

of the **pseudo**-edit distance  $d_e(x, y)$  with respect to  $\tilde{a}(x)$ :

$$\nabla_{\tilde{a}(x)} \tilde{d}_A(x, y) = \sum_{i=1}^{|x|} \delta(x, x_i) \cdot \left[ \sum_{j=1}^{|y|+1} P_{\alpha}(\tilde{x}, \tilde{y})_{i,j} \cdot \frac{\tilde{a}(x_i) - \tilde{a}(y_j)}{|\tilde{a}(x_i) - \tilde{a}(y_j)|} \right] + \sum_{j=1}^{|y|} \delta(x, y_j) \cdot \left[ \sum_{i=1}^{|x|+1} P_{\alpha}(\tilde{x}, \tilde{y})_{i,j} \cdot \frac{\tilde{a}(x_i) - \tilde{a}(x_i)}{|\tilde{a}(x_i) - \tilde{a}(x_i)|} \right]$$

where  $\delta$  is the Kronecker-Delta, i.e.:  $\delta(x, y) = 1$  if  $x = y$  and 0 otherwise.

Finally, we can plug this result into Equation 5, which yields a gradient  $\nabla_{\tilde{a}(x)} E$ , such that we can learn the vectorial embedding of  $X$  via gradient techniques. Note that prior theory on metric learning on the GLVQ cost function suggests that the learned embedding will degenerate to a very low rank-matrix such that the model may become overly simplistic (Biehl et al., 2015). To prevent such a degeneration, we follow the regularization recommendation of Schneider et al. (2010) and add the term  $\beta \cdot \log(\det(A^T \cdot A))$  to the GLVQ loss  $\mathcal{L}$ , which adds the gradient  $\beta \cdot 2 \cdot A^{-1T}$  where  $A^1$  is the Moore-Penrose **Pseudoinverse** of  $A$ . Additionally, we follow the regularization approach of good edit similarity learning (Bellet et al., 2012) and add the Frobenius-norm  $\beta \cdot \|A\|_F^2$  to the loss, which adds the gradient  $\beta \cdot 2 \cdot A$ .

As initialization of the vectorial embedding we use a  $U$ -dimensional simplex with side length 1, which leads to  $c_d(x, y) = 0$  if  $x = y$  and 1 otherwise (refer to the supplementary material for a more detailed look into this initialization (Paalen, 2018b)).

Regarding computational complexity, we can analyze the gradient computation. To compute a gradient, we first need to select the closest correct and closest wrong prototype for every data point, which is possible in  $O(m \cdot R)$ . Then, we need to compute the gradient for each data point via Equation 6, which is possible in  $O(m \cdot n^2 \cdot V)$  where  $n$  is the largest tree size in the data set. Computing the regularization requires  $O(V^2)$  due to the **Pseudoinverse**, resulting in  $O(m \cdot R + m \cdot n^2 \cdot V + V^2)$  overall. How often the gradient needs to be computed depends on the optimizer, but can typically be regarded as a constant. In our experiments, we limit the number of gradient computations to 200.

### 5. Experiments

In our experiment, we investigate whether our proposed metric learning scheme, embedding edit distance learning (BEDL), is able to improve classification accuracy beyond the default initialization, whether BEDL improves upon the accuracy obtained by good edit similarity learning (Bellet et al., 2012), and whether the resulting embedding

Tree Edit Distance Learning via Adaptive Symbol Embeddings

yields insight regarding the classification task in question. In particular, we evaluate on the following data sets, including a variety of domains and data set sizes.

**Strings:** A two-class data set of 200 strings of length 12, adapted from Mikkel et al. (2015). Strings in class 1 consist of 6 a or b symbols, followed by a c or d, followed by another 5 a or b symbols. Which of the two respective symbols is selected is chosen uniformly at random. Strings in class 2 are constructed in much the same way, except that they consist of 5 a or b symbols, followed by a c or d, followed by another 6 a or b symbols. Note that the classes can be neither discriminated via length nor via symbol frequency features. The decisive discriminative feature is where a c or d is located in the string.

**MiniPalindrome and Sorting:** Two data sets of Java programs, where classes represent different strategies to solve a programming task. The MiniPalindrome data set contains 48 programs implementing one of eight strategies to detect whether an input string contains only palindromes (Paalen, 2016a), and the Sorting data set contains 64 programs implementing either a BubbleSort or an InsertionSort strategy (Paalen, 2016b). The programs are represented by their abstract syntax tree where the label corresponds to one of 24 programming concepts (e.g. class declaration, function declaration, method call, etc.).

**Cystic and Leukemia:** Two data sets from KEGG Glycan data base (Hashimoto et al., 2006) adapted from Gallicchio & Micheli (2013), where one class corresponds to benign molecules and the other class corresponds to molecules associated with cystic fibrosis or leukemia respectively. The molecules are represented as trees, where the label corresponds to mono-saccharide identifiers (one of 29 and one of 57 for Cystic and Leukemia, respectively), and the roots are chosen according to biological meaning (Hashimoto et al., 2006). The cystic data set contains 160, the Leukemia data set 442 molecules.

**Sentiment:** A large-scale two-class data set of 9613 sentences from movie reviews, where one class (4650 trees) corresponds to negative and the other class (4963 trees) to positive reviews. The sentences are represented by their syntax trees, where inner nodes are unlabeled and leaves are labeled with one of over 30,000 words (Socher et al., 2013). Note that GESL is not practically applicable for this data set, as the number of parameters to learn scales quadratically with the number of words, i.e.  $> 30,000^2$ . To make BEDL applicable in this case, we initialize the vectorial embedding with the 300-dimensional Common Crawl GloVe embedding (Pennington et al., 2014), which we reduce via PCA, retaining 95% of the data variance ( $V = 16.4 \pm 2.3$  dimensions on average  $\pm$  standard deviation). We adapt this initial embedding via a linear transformation  $\Omega \in \mathbb{R}^{V \times V}$  which we learn via BEDL. Fur

# Example: Reading a typical machine learning paper

Tree Edit Distance Learning via Adaptive Symbol Embeddings

ther, we replace the cost function with the cosine distance  $\cos(\vec{x}, \vec{y}) = \frac{1}{2} \cdot ((1 + \vec{x}^T \cdot \vec{y}) / (\|\vec{x}\| \cdot \|\vec{y}\|))$ , which is the recommended distance measure for the GloVe word embedding (Pennington et al., 2014) (refer to the supplementary material for the gradient; Puaßen (2018b)).

On each data set, we perform a crossvalidation<sup>5</sup> and compare the average test error across folds. In particular, we compare the error when using the initial tree edit distance with the error when using the **gesdl** edit distance learned via good edit similarity learning (GESL), and the tree edit distance learned via our proposed approach (BEDL).

In general, we would expect that a discriminative metric learned for one classifier also facilitates classification using other classifiers. Therefore, we report the classification error for four classifiers, namely the median generalized learning vector quantization classifier (MGLVQ) for which our method is optimized, the goodness classifier for which GESL is optimized (Bellet et al., 2012), the K-nearest neighbor (KNN) classifier, and the support vector machine (SVM) based on the radial basis function kernel. In order to ensure a kernel matrix for SVM, we set negative eigenvalues to zero (clip Eigenvalue correction). Note that this eigenvalue correction requires cubic runtime in terms of the number of data points and is thus prohibitively slow for large data set sizes. Therefore, for the Sentiment data set, we trained the classifiers on a randomly selected sample of 300 points from the training data.

We optimized all hyper-parameters in a nested 5-fold crossvalidation, namely the number of prototypes  $K$  for MGLVQ and LVQ metric learning in the range  $[1, 15]$ , the number of neighbors for KNN in the range  $[1, 15]$ , the kernel bandwidth for SVM in the range  $[0.1, 10]$ , the sparsity parameter  $\lambda$  for the goodness classifier in the range  $[10^{-5}, 10]$ , and the regularization strength  $\beta$  for GESL and BEDL in the range  $2 \cdot K \cdot m \cdot [10^{-6}, 10^{-2}]$ . We chose the number of prototypes for BEDL, as well as the number of neighbors for GESL, as the optimal number of prototypes  $K$  for MGLVQ.

As implementations, we used custom implementations of KNN, MGLVQ, the goodness classifier, GESL, and BEDL, which are available at <https://doi.org/10.4139/unibi/2919994>. For SVM, we utilized the LIBSVM standard implementation (Chang & Lin, 2011). All experiments were

<sup>5</sup> We used 20 folds for Strings and Sentiment, 10 for Cystic and Leukemia, 8 for Sorting and 6 for MalignIndurine. For the programming data sets, the number of folds had to be reduced to ensure that each fold still contained a meaningful number of data points. For the Cystic and Leukemia data set, our ten folds were consistent with the paper of Galkuchio & Micheli (2013). In all cases, folds were generated such that the label distribution of the overall data set was maintained.

Table 1. The mean test classification error and runtimes for metric learning, averaged over the cross validation trials, as well as the standard deviation. The x-axis shows the metric learning schemes, the y-axis the different classifiers used for evaluation. The table is sub-divided for each data set. The lowest classification error for each data set is highlighted via bold print.

classifier	initial	GESL	BEDL
Strings			
KNN	21.0 ± 10.2%	23.0 ± 10.6%	<b>6.0 ± 0.0%</b>
MGLVQ	36.0 ± 15.7%	34.0 ± 11.0%	<b>6.0 ± 0.0%</b>
SVM	9.0 ± 11.2%	10.0 ± 8.6%	<b>0.0 ± 0.0%</b>
goodness	11.5 ± 9.3%	0.5 ± 2.2%	<b>0.0 ± 0.0%</b>
runtime [s]	0 ± 0	0.030 ± 0.002	1.077 ± 0.008
MalignIndurine			
KNN	12.5 ± 11.2%	12.5 ± 7.9%	10.4 ± 9.4%
MGLVQ	2.1 ± 1.1%	4.2 ± 6.5%	<b>0.0 ± 0.0%</b>
SVM	4.2 ± 6.5%	20.8 ± 15.1%	<b>0.0 ± 0.0%</b>
goodness	6.2 ± 6.8%	14.6 ± 5.1%	8.3 ± 10.2%
runtime [s]	0 ± 0	0.105 ± 0.014	2.780 ± 0.031
Sorting			
KNN	15.6 ± 8.8%	18.8 ± 10.4%	10.9 ± 8.0%
MGLVQ	14.1 ± 10.4%	14.1 ± 8.0%	14.1 ± 8.0%
SVM	10.9 ± 8.0%	<b>0.4 ± 0.4%</b>	<b>0.4 ± 0.4%</b>
goodness	15.6 ± 11.1%	17.2 ± 14.8%	17.2 ± 9.3%
runtime [s]	0 ± 0	0.352 ± 0.102	3.358 ± 0.748
Cystic			
KNN	33.2 ± 6.6%	32.5 ± 10.1%	28.1 ± 8.5%
MGLVQ	34.4 ± 6.8%	33.1 ± 9.8%	30.0 ± 10.1%
SVM	28.1 ± 8.0%	33.1 ± 8.9%	29.4 ± 12.1%
goodness	28.1 ± 8.5%	26.2 ± 14.4%	<b>24.4 ± 13.3%</b>
runtime [s]	0 ± 0	0.353 ± 0.292	0.864 ± 0.767
Leukemia			
KNN	7.5 ± 2.6%	8.2 ± 4.0%	7.3 ± 4.3%
MGLVQ	9.5 ± 4.0%	10.9 ± 4.7%	9.5 ± 3.0%
SVM	7.0 ± 4.1%	8.8 ± 2.9%	6.8 ± 4.7%
goodness	<b>6.1 ± 4.3%</b>	10.0 ± 4.4%	6.3 ± 3.4%
runtime [s]	0 ± 0	2.208 ± 0.919	6.550 ± 2.706
Sentiment			
KNN	40.2 ± 2.8%	—	38.2 ± 3.3%
MGLVQ	44.0 ± 2.0%	—	34.2 ± 2.7%
SVM	34.3 ± 3.0%	—	<b>33.3 ± 8.6%</b>
goodness	43.7 ± 1.9%	—	42.5 ± 3.1%
runtime [s]	0 ± 0	—	69.385 ± 36.064

performed on a consumer-grade laptop with an Intel Core i7-7700HQ CPU.

The results of our experiments are displayed in Table 5. In all data sets and for all classifiers, BEDL yields lower classification error compared to GESL. For the Strings data set we can also verify this result statistically with a one-sided Wilcoxon signed rank test ( $p < 10^{-6}$ ). Furthermore, in all but the Leukemia data set, BEDL yields the overall best classification results, and is close to optimal for the Leukemia data set (0.2% difference). In all cases, BEDL could improve the accuracy for KNN, in five out of six cases for SVM (the exception being the Cystic data set), in four out of six cases for MGLVQ (in Sorting and Leukemia it stayed equal), and in three out of six cases for the goodness classifier. For the Strings and Sentiment data sets we

# Example: Reading a typical machine learning paper

We tested on lots of data

Tree Edit Distance Learning via Adaptive Symbol Embeddings

ther, we replace the cost function with the cosine distance  $\cos(\vec{x}, \vec{y}) = \frac{1}{2} - \frac{1}{2} \cdot ((1 + \vec{x}^T \cdot \vec{y}) / (||\vec{x}|| \cdot ||\vec{y}||))$ , which is the recommended distance measure for the GloVe word embedding (Pennington et al., 2014) (refer to the supplementary material for the gradient; Puaßen (2018b)).

On each data set, we perform a crossvalidation<sup>5</sup> and compare the average test error across folds. In particular, we compare the error when using the initial tree edit distance with the error when using the **gesdl** edit distance learned via good edit similarity learning (GESL), and the tree edit distance learned via our proposed approach (BEDL).

In general, we would expect that a discriminative metric learned for one classifier also facilitates classification using other classifiers. Therefore, we report the classification error for four classifiers, namely the median generalized learning vector quantization classifier (MGLVQ) for which our method is optimized, the goodness classifier for which GESL is optimized (Bellef et al., 2012), the K-nearest neighbor (KNN) classifier, and the support vector machine (SVM) based on the radial basis function kernel. In order to ensure a kernel matrix for SVM, we set negative eigenvalues to zero (clip Eigenvalue correction). Note that this eigenvalue correction requires cubic runtime in terms of the number of data points and is thus prohibitively slow for large data set sizes. Therefore, for the Sentiment data set, we trained the classifier on a randomly selected sample of 300 points from the training data.

We optimized all hyper-parameters in a nested 5-fold crossvalidation, namely the number of prototypes  $K$  for MGLVQ and LVQ metric learning in the range  $[1, 15]$ , the number of neighbors for KNN in the range  $[1, 15]$ , the kernel bandwidth for SVM in the range  $[10^{-4}, 10]$ , the sparsity parameter  $\lambda$  for the goodness classifier in the range  $[10^{-5}, 10]$ , and the regularization strength  $\beta$  for GESL and BEDL in the range  $2 \cdot K \cdot m \cdot [10^{-6}, 10^{-2}]$ . We chose the number of prototypes for BEDL, as well as the number of neighbors for GESL, as the optimal number of prototypes  $K$  for MGLVQ.

As implementations, we used custom implementations of KNN, MGLVQ, the goodness classifier, GESL, and BEDL, which are available at <https://doi.org/10.4139/unibi/2919994>. For SVM, we utilized the LIBSVM standard implementation (Chang & Lin, 2011). All experiments were

<sup>5</sup>We used 20 folds for Strings and Sentiment, 10 for Cystic and Leukemia, 8 for Sorting and 6 for Mushroom. For the programming data sets, the number of folds had to be reduced to ensure that each fold still contained a meaningful number of data points. For the Cystic and Leukemia data set, our ten folds were consistent with the paper of Galka et al. (2013). In all cases, folds were generated such that the label distribution of the overall data set was maintained.

Table 1. The mean test classification error and runtimes for metric learning, averaged over the cross validation trials, as well as the standard deviation. The x-axis shows the metric learning schemes, the y-axis the different classifiers used for evaluation. The table is sub-divided for each data set. The lowest classification error for each data set is highlighted via bold print.

classifier	initial	GESL	BEDL
Strings			
KNN	21.0 ± 10.2%	23.0 ± 10.6%	<b>6.0 ± 0.0%</b>
MGLVQ	36.0 ± 15.7%	34.0 ± 11.0%	<b>6.0 ± 0.0%</b>
SVM	9.0 ± 1.2%	10.0 ± 8.6%	<b>0.0 ± 0.0%</b>
goodness	11.5 ± 9.3%	0.5 ± 2.2%	<b>0.0 ± 0.0%</b>
runtime [s]	0 ± 0	0.030 ± 0.002	1.077 ± 0.008
Mushroom			
KNN	12.5 ± 11.2%	12.5 ± 7.9%	10.4 ± 9.4%
MGLVQ	2.1 ± 1.1%	4.2 ± 6.5%	<b>0.0 ± 0.0%</b>
SVM	4.2 ± 6.5%	20.8 ± 15.1%	<b>0.0 ± 0.0%</b>
goodness	6.2 ± 6.8%	14.6 ± 5.1%	8.3 ± 10.2%
runtime [s]	0 ± 0	0.105 ± 0.014	2.780 ± 0.031
Sorting			
KNN	15.6 ± 8.8%	18.8 ± 10.4%	10.9 ± 8.0%
MGLVQ	14.1 ± 10.4%	14.1 ± 8.0%	14.1 ± 8.0%
SVM	10.9 ± 8.0%	<b>0.4 ± 0.4%</b>	<b>0.4 ± 0.4%</b>
goodness	15.6 ± 11.1%	17.2 ± 14.8%	17.2 ± 9.3%
runtime [s]	0 ± 0	0.352 ± 0.102	3.358 ± 0.748
Cystic			
KNN	33.2 ± 6.6%	32.5 ± 10.1%	28.1 ± 8.3%
MGLVQ	34.4 ± 6.8%	33.1 ± 9.8%	30.0 ± 10.1%
SVM	28.1 ± 8.0%	33.1 ± 8.9%	29.4 ± 12.1%
goodness	28.1 ± 8.5%	26.2 ± 14.4%	<b>24.4 ± 13.3%</b>
runtime [s]	0 ± 0	0.353 ± 0.292	0.864 ± 0.767
Leukemia			
KNN	7.5 ± 2.6%	8.2 ± 4.6%	7.3 ± 4.3%
MGLVQ	9.5 ± 4.0%	10.9 ± 4.7%	9.5 ± 3.0%
SVM	7.0 ± 4.1%	8.8 ± 2.9%	6.8 ± 4.7%
goodness	<b>6.1 ± 4.3%</b>	10.0 ± 4.4%	6.3 ± 3.4%
runtime [s]	0 ± 0	2.208 ± 0.919	6.550 ± 2.706
Sentiment			
KNN	40.2 ± 2.8%	—	38.2 ± 3.3%
MGLVQ	44.0 ± 2.6%	—	39.2 ± 2.7%
SVM	34.3 ± 3.0%	—	<b>33.3 ± 8.6%</b>
goodness	43.7 ± 1.9%	—	42.5 ± 1.1%
runtime [s]	0 ± 0	—	69.385 ± 36.064

performed on a consumer-grade laptop with an Intel Core i7-7700HQ CPU.

The results of our experiments are displayed in Table 5. In all data sets and for all classifiers, BEDL yields lower classification error compared to GESL. For the Strings data set we can also verify this result statistically with a one-sided Wilcoxon signed rank test ( $p < 10^{-6}$ ). Furthermore, in all but the Leukemia data set, BEDL yields the overall best classification results, and is close to optimal for the Leukemia data set (0.2% difference). In all cases, BEDL could improve the accuracy for KNN, in five out of six cases for SVM (the exception being the Cystic data set), in four out of six cases for MGLVQ (in Sorting and Leukemia it stayed equal), and in three out of six cases for the goodness classifier. For the Strings and Sentiment data sets we

# Example: Reading a typical machine learning paper

## Tree Edit Distance Learning via Adaptive Symbol Embeddings

ther, we replace the cost function with the cosine distance  $\cos(\vec{x}, \vec{y}) = \frac{1}{2} - \frac{1}{2} \cdot ((1 + \vec{x}^T \cdot \vec{y})^\alpha - 1)^\beta$ , where  $\alpha$  is the recommended distance metric for the word embedding (Pennington et al., 2014) (refer to the supplementary material for the gradient; Puaßen (2018b)).

On each data set, we perform a crossvalidation<sup>5</sup> and compare the average test error across folds. In particular, we compare the error when using the initial tree edit distance with the error when using the **gesdl** edit distance learned via good edit similarity learning (GESL), and the tree edit distance learned via our proposed approach (BEDL).

In general, we would expect that a discriminative metric learned for one classifier also facilitates classification using other classifiers. Therefore, we report the classification error for four classifiers, namely the median generalized learning vector quantization classifier (MGLVQ) for which our method is optimized, the goodness classifier for which GESL is optimized (Bellef et al., 2012), the K-nearest neighbor (KNN) classifier, and the support vector machine (SVM) based on the radial basis function kernel. In order to ensure a kernel matrix for SVM, we set negative eigenvalues to zero (clip Eigenvalue correction). Note that this eigenvalue correction requires cubic runtime in terms of the number of data points and is thus prohibitively slow for large data set sizes. Therefore, for the Sentiment data set, we trained the classifier on a randomly selected sample of 300 points from the training data.

We optimized all hyper-parameters in a nested 5-fold crossvalidation, namely the number of prototypes  $K$  for MGLVQ and LVQ metric learning in the range  $[1, 15]$ , the number of neighbors for KNN in the range  $[1, 15]$ , the kernel bandwidth for SVM in the range  $[10^{-4}, 10]$ , the sparsity parameter  $\lambda$  for the goodness classifier in the range  $[10^{-5}, 10]$ , and the regularization strength  $\beta$  for GESL and BEDL in the range  $2 \cdot K \cdot m \cdot [10^{-6}, 10^{-2}]$ . We chose the number of prototypes for BEDL, as well as the number of neighbors for GESL, as the optimal number of prototypes  $K$  for MGLVQ.

As implementations, we used custom implementations of KNN, MGLVQ, the goodness classifier, GESL, and BEDL, which are available at <https://doi.org/10.4139/unibi/2919994>. For SVM, we utilized the LIBSVM standard implementation (Chang & Lin, 2011). All experiments were

<sup>5</sup> We used 20 folds for Strings and Sentiment, 10 for Cystic and Leukemia, 8 for Sorting and 6 for Mushroom. For the programming data sets, the number of folds had to be reduced to ensure that each fold still contained a meaningful number of data points. For the Cystic and Leukemia data set, our ten folds were consistent with the paper of Galkuchio & Micheli (2013). In all cases, folds were generated such that the label distribution of the overall data set was maintained.

Table 1. The mean test classification error and runtimes for metric learning methods and classifiers on 10 data sets. The x-axis is subdivided for each data set. The lowest classification error for each data set is highlighted via bold font.

classifier	initial	GESL	BEDL
Strings			
KNN	21.0 ± 10.2%	23.0 ± 10.6%	<b>6.0 ± 0.0%</b>
MGLVQ	36.0 ± 15.7%	34.0 ± 11.0%	<b>6.0 ± 0.0%</b>
SVM	9.0 ± 11.2%	10.0 ± 8.6%	<b>0.0 ± 0.0%</b>
goodness	11.5 ± 9.3%	0.5 ± 2.2%	<b>0.0 ± 0.0%</b>
runtime [s]	0 ± 0	0.030 ± 0.010	1.077 ± 0.008
Mushroom			
KNN	12.5 ± 11.2%	12.5 ± 7.9%	10.4 ± 9.4%
MGLVQ	2.1 ± 1.1%	4.2 ± 6.5%	<b>0.0 ± 0.0%</b>
SVM	4.2 ± 6.5%	20.8 ± 15.1%	<b>0.0 ± 0.0%</b>
goodness	6.2 ± 6.8%	14.6 ± 5.1%	8.3 ± 10.2%
runtime [s]	0 ± 0	0.105 ± 0.014	2.780 ± 0.031
Sorting			
KNN	15.6 ± 8.8%	18.8 ± 10.4%	10.9 ± 8.0%
MGLVQ	14.1 ± 10.4%	14.1 ± 8.0%	14.1 ± 8.0%
SVM	10.9 ± 8.0%	<b>0.4 ± 0.4%</b>	<b>0.4 ± 0.4%</b>
goodness	15.6 ± 11.1%	17.2 ± 14.8%	17.2 ± 9.3%
runtime [s]	0 ± 0	0.352 ± 0.102	3.358 ± 0.748
Cystic			
KNN	33.2 ± 6.6%	32.5 ± 10.1%	28.1 ± 8.3%
MGLVQ	34.4 ± 6.8%	33.1 ± 9.8%	30.0 ± 10.1%
SVM	28.1 ± 8.0%	33.1 ± 8.8%	29.4 ± 12.1%
goodness	28.1 ± 8.5%	26.2 ± 14.4%	<b>24.4 ± 13.3%</b>
runtime [s]	0 ± 0	0.353 ± 0.292	0.864 ± 0.767
Leukemia			
KNN	7.5 ± 2.6%	8.2 ± 4.6%	7.3 ± 4.3%
MGLVQ	9.5 ± 4.0%	10.9 ± 4.7%	9.5 ± 3.0%
SVM	7.0 ± 4.1%	8.8 ± 2.9%	6.8 ± 4.7%
goodness	<b>6.1 ± 4.3%</b>	10.0 ± 4.4%	6.3 ± 3.4%
runtime [s]	0 ± 0	2.208 ± 0.919	6.550 ± 2.706
Sentiment			
KNN	40.2 ± 2.8%	—	38.2 ± 3.3%
MGLVQ	44.0 ± 2.6%	—	38.2 ± 3.3%
SVM	34.3 ± 3.0%	—	<b>33.3 ± 8.6%</b>
goodness	43.7 ± 1.9%	—	42.5 ± 3.1%
runtime [s]	0 ± 0	—	69.385 ± 36.064

performed on a consumer-grade laptop with an Intel Core i7-7700HQ CPU.

The results of our experiments are displayed in Table 5. In all data sets and for all classifiers, BEDL yields lower classification error compared to GESL. For the Strings data set we can also verify this result statistically with a one-sided Wilcoxon signed rank test ( $p < 10^{-6}$ ). Furthermore, in all but the Leukemia data set, BEDL yields the overall best classification results, and is close to optimal for the Leukemia data set (0.2% difference). In all cases, BEDL could improve the accuracy for KNN, in five out of six cases for SVM (the exception being the Cystic data set), in four out of six cases for MGLVQ (in Sorting and Leukemia it stayed equal), and in three out of six cases for the goodness classifier. For the Strings and Sentiment data sets we

We tested on lots of data

Previous methods did okay ...

# Example: Reading a typical machine learning paper

We tested on lots of data

## Tree Edit Distance Learning via Adaptive Symbol Embeddings

ther, we replace the cost function with the cosine distance  $\cos(\vec{x}, \vec{y}) = \frac{1}{2} - \frac{1}{2} \cdot ((1 + \vec{x}^T \cdot \vec{y})^\alpha - \Omega)^\beta$ , where  $\vec{x}$  and  $\vec{y}$  are the recommended distance functions for the word embedding (Pennington et al., 2014) (refer to the supplementary material for the gradient; Puaßen (2018b)).

On each data set, we perform a crossvalidation<sup>5</sup> and compare the average test error across folds. In particular, we compare the error when using the initial tree edit distance with the error when using the ~~proposed~~ edit distance learned via good edit similarity learning (GESL), and the tree edit distance learned via our proposed approach (BEDL).

In general, we would expect that a discriminative metric learned for one classifier also facilitates classification using other classifiers. Therefore, we report the classification error for four classifiers, namely the median generalized learning vector quantization classifier (MGLVQ) for which our method is optimized, the goodness classifier for which GESL is optimized (Bellet et al., 2012), the K-nearest neighbor (KNN) classifier, and the support vector machine (SVM) based on the radial basis function kernel. In order to ensure a kernel matrix for SVM, we set negative eigenvalues to zero (i.e., Eigenvalue correction). Note that this eigenvalue correction requires cubic runtime in terms of the number of data points and is thus prohibitively slow for large data set sizes. Therefore, for the Sentiment data set, we trained the classifier on a randomly selected sample of 300 points from the training data.

We optimized all hyper-parameters in a nested 5-fold crossvalidation, namely the number of prototypes  $K$  for MGLVQ and LVQ metric learning in the range  $[1, 15]$ , the number of neighbors for KNN in the range  $[1, 15]$ , the kernel bandwidth for SVM in the range  $[10^{-1}, 10]$ , the sparsity parameter  $\lambda$  for the goodness classifier in the range  $[10^{-5}, 10]$ , and the regularization strength  $\beta$  for GESL and BEDL in the range  $2 \cdot K \cdot m \cdot [10^{-6}, 10^{-2}]$ . We chose the number of prototypes for GESL, as well as the number of neighbors for GESL, as the optimal number of prototypes  $K$  for MGLVQ.

As implementations, we used custom implementations of KNN, MGLVQ, the goodness classifier, GESL, and BEDL, which are available at <https://doi.org/10.4139/unibi/2919994>. For SVM, we utilized the LIBSVM standard implementation (Chang & Lin, 2011). All experiments were

<sup>5</sup>We used 20 folds for Strings and Sentiment, 10 for Cystic and Leukemia, 8 for Sorting and 6 for Mushroom. For the programming data sets, the number of folds had to be reduced to ensure that each fold still contained a meaningful number of data points. For the Cystic and Leukemia data set, our ten folds were consistent with the paper of Galkuchio & Micheli (2013). In all cases, folds were generated such that the label distribution of the overall data set was maintained.

Table 1. The mean test classification error and runtimes for metric learning methods on seven data sets. The x-axis shows the methods, the y-axis the different classifiers used for evaluation. The table is sub-divided for each data set. The lowest classification error for each data set is highlighted via bold print.

classifier	initial	GESL	BEDL
Strings			
KNN	21.0 ± 10.2%	23.0 ± 10.6%	<b>0.0 ± 0.0%</b>
MGLVQ	36.0 ± 15.7%	34.0 ± 11.0%	<b>0.0 ± 0.0%</b>
SVM	9.0 ± 11.2%	10.0 ± 8.6%	<b>0.0 ± 0.0%</b>
goodness	11.5 ± 9.3%	0.5 ± 2.2%	<b>0.0 ± 0.0%</b>
runtime [s]	0 ± 0	0.030 ± 0.002	1.077 ± 0.008
Mushroom			
KNN	12.5 ± 11.2%	12.5 ± 7.9%	10.4 ± 9.4%
MGLVQ	2.1 ± 1.1%	4.2 ± 6.5%	<b>0.0 ± 0.0%</b>
SVM	4.2 ± 6.5%	20.8 ± 15.1%	<b>0.0 ± 0.0%</b>
goodness	6.2 ± 6.8%	14.6 ± 5.1%	8.3 ± 10.2%
runtime [s]	0 ± 0	0.105 ± 0.014	2.780 ± 0.031
Sorting			
KNN	15.6 ± 8.8%	18.8 ± 10.4%	10.0 ± 6.0%
MGLVQ	14.1 ± 10.4%	14.1 ± 8.0%	14.1 ± 8.0%
SVM	10.0 ± 6.0%	<b>0.4 ± 0.4%</b>	0.4 ± 0.4%
goodness	15.6 ± 11.1%	17.2 ± 14.8%	17.2 ± 9.3%
runtime [s]	0 ± 0	0.352 ± 0.102	3.358 ± 0.716
Cystic			
KNN	33.2 ± 6.6%	32.5 ± 10.1%	28.1 ± 9.3%
MGLVQ	34.4 ± 6.8%	33.1 ± 9.8%	30.0 ± 20.1%
SVM	28.1 ± 9.0%	33.1 ± 9.8%	39.1 ± 22.3%
goodness	28.1 ± 8.5%	26.2 ± 14.4%	24.4 ± 13.3%
runtime [s]	0 ± 0	0.353 ± 0.292	0.864 ± 0.707
Leukemia			
KNN	7.5 ± 2.6%	8.2 ± 4.0%	7.3 ± 4.3%
MGLVQ	9.5 ± 4.0%	10.9 ± 4.7%	9.5 ± 3.0%
SVM	7.0 ± 4.1%	8.8 ± 2.9%	6.8 ± 4.7%
goodness	6.1 ± 4.3%	10.0 ± 4.4%	6.3 ± 3.4%
runtime [s]	0 ± 0	2.208 ± 0.919	6.550 ± 2.706
Sentiment			
KNN	40.2 ± 2.8%	—	38.2 ± 3.3%
MGLVQ	44.0 ± 2.0%	—	39.2 ± 3.7%
SVM	34.3 ± 3.0%	—	39.2 ± 3.6%
goodness	43.7 ± 1.9%	—	42.3 ± 1.1%
runtime [s]	0 ± 0	—	40.385 ± 50.064

performed on a consumer-grade laptop with an Intel Core i7-7700HQ CPU.

The results of our experiments are displayed in Table 5. In all data sets and for all classifiers, BEDL yields lower classification error compared to GESL. For the Strings data set we can also verify this result statistically with a one-sided Wilcoxon signed rank test ( $p < 10^{-6}$ ). Furthermore, in all but the Leukemia data set, BEDL yields the overall best classification results, and is close to optimal for the Leukemia data set (0.2% difference). In all cases, BEDL could improve the accuracy for KNN, in five out of six cases for SVM (the exception being the Cystic data set), in four out of six cases for MGLVQ (in Sorting and Leukemia it stayed equal), and in three out of six cases for the goodness classifier. For the Strings and Sentiment data sets we

Previous methods did okay ...

but we did better ...

# Literature



THE UNIVERSITY OF  
SYDNEY

# Literature I

Bishop, Christopher M. (2006). **Pattern Recognition and Machine Learning**. Berlin/Heidelberg, Germany: Springer. ISBN: 0387310738.

Paaßen, Benjamin (2019). **Lecture Notes on Applied Optimization**. Bielefeld University. URL: <https://pub.uni-bielefeld.de/record/2935200>.

Shalev-Shwartz, Shai and Shai Ben-David (2014). **Understanding Machine Learning - From Theory to Algorithms**. Cambridge, UK: Cambridge University Press. URL: <https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf>.