



Machine learning basics

Pavlo O. Dral
Xiamen University, P.R. China

Visiting Professor in
Nicolaus Copernicus University, Poland

4 July 2024

dr-dral.com

Search docs

CONTENTS:

1. Get started with MAtom@XACS

1.1. Slides

1.2. Video

1.3. Geometry optimization

INTERMOLECULAR INTERACTION

1. Documentation of GAMESS-US Input

2. Practice in energy decomposition analysis with XEDA

3. Practice with BLW

4. Practice in energy decomposition analysis with SAPT

VALENCE BOND THEORY

1. The Understanding of H₂

2. Computing of F₂

1. Get started with MAtom@XACS

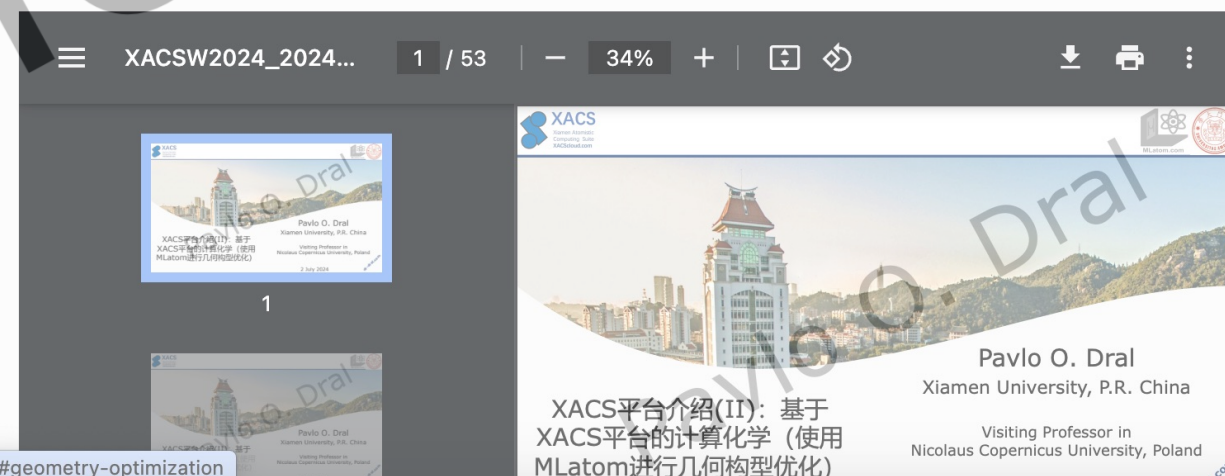
1. Get started with MAtom@XACS

This tutorial will use the MAtom@XACS program.

- Slides
- Video
- Geometry optimization

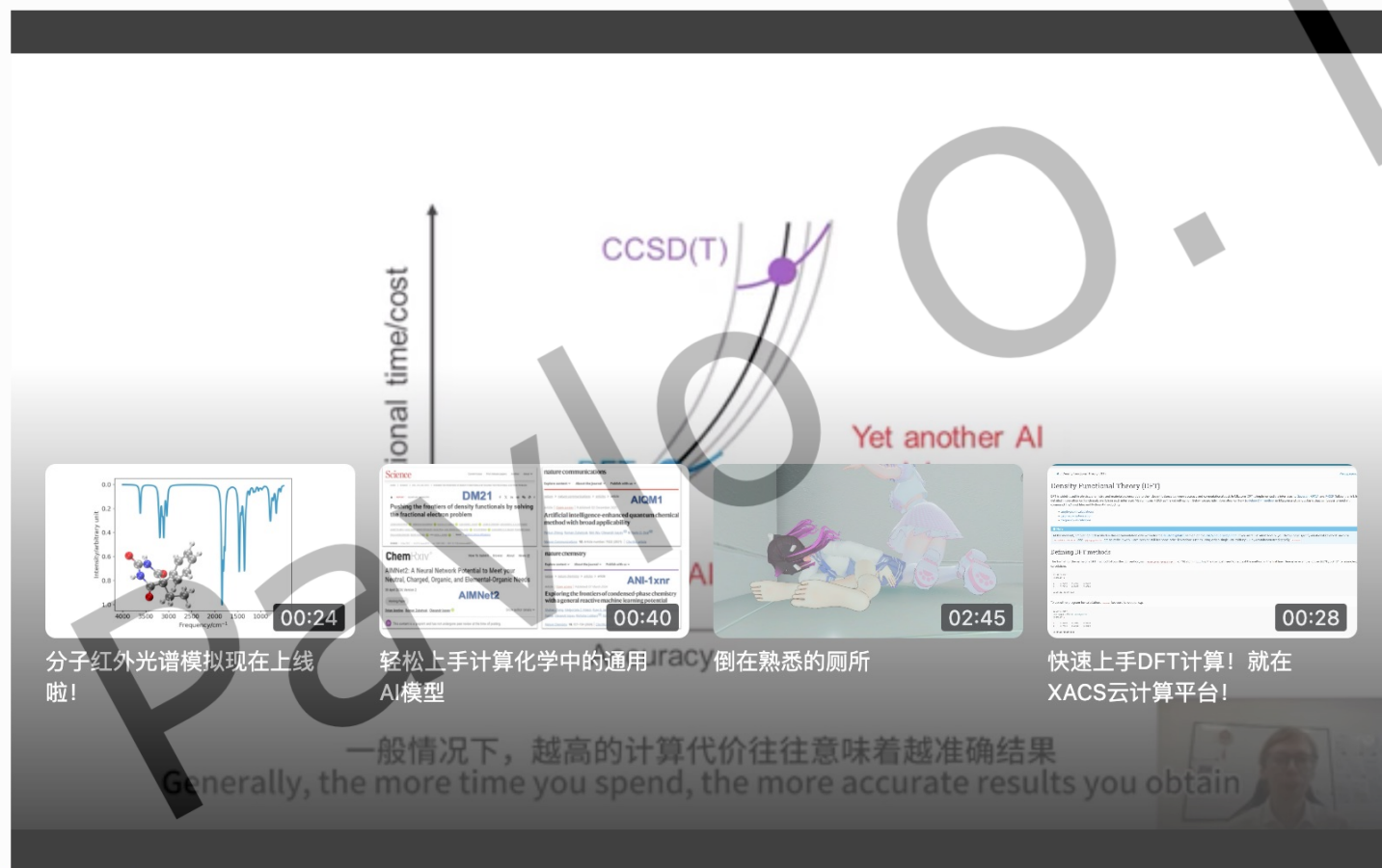
1.1. Slides

Slides:



1.3. Geometry optimization

We will get started by showcasing the power of ML to supercharge the computational chemistry simulations. We will use the most recent and advanced concept of performing simulations with the universal and updatable AI-enhanced quantum mechanical (UAIQM) models:



ional time/cost

CCSD(T)

Yet another AI

DM21
Pushing the frontiers of density functionals by solving the fractional electron problem

AIMNet2
A fully self-supervised neural network for molecular representation

ANI-1xnp
Expanding the frontiers of condensed phase chemistry with general reactive machine learning models

00:24

00:40

02:45

00:28

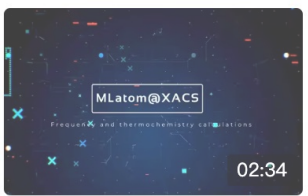
分子红外光谱模拟现在上线啦!

轻松上手计算化学中的通用AI模型

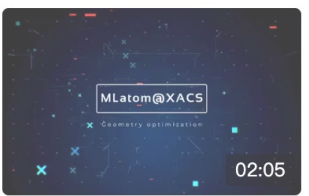
快速上手DFT计算! 就在XACS云计算平台!

一般情况下, 越高的计算代价往往意味着越准确结果
Generally, the more time you spend, the more accurate results you obtain

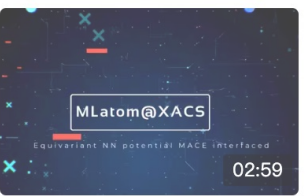
Let's say you want to optimize the molecular geometry. Now you do not need to worry which method (DFT or *ab initio*, etc.) to choose: MAtom will choose the best model automatically for you.



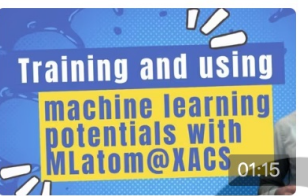
使用MLatom@XACS进行频率和热化学计算
207 1-29



使用MLatom@XACS进行几何优化
169 1-18



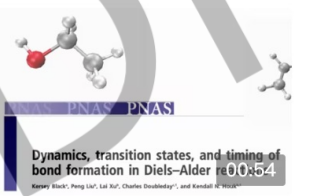
在MLatom@XACS中使用等变神经网络势能面——MACE接口
196 1-10



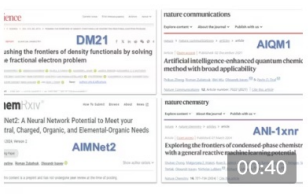
在MLatom@XACS中训练和使用机器学习势
146 4-10



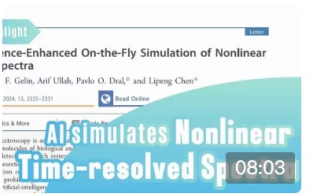
比DFT更快更准确：MLatom@XACS中的AIQM1
249 4-17



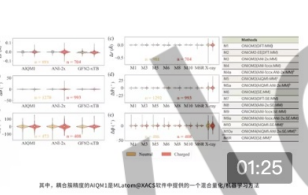
像在PNAS和JACS论文中一样研究反应机理的准经典轨迹！
1906 5-8



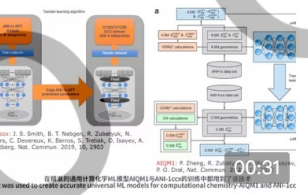
轻松上手计算化学中的通用AI模型
919 5-15



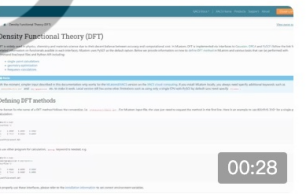
XACS应用赏析：MLatom@XACS多用于AI增强的非线性时间分辨光谱
329 2-28



Nat. Commun.: AIQM1赋能加快多尺度量化精修可靠的蛋白质-药物结合
197 5-22



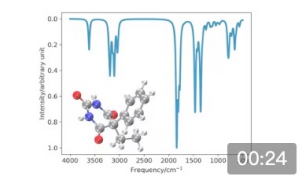
迁移学习：使用更少的数据构建更好的AI模型
208 5-29



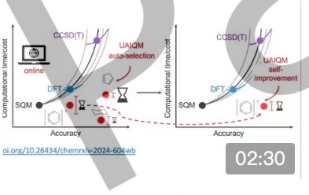
快速上手DFT计算！就在XACS云计算平台！
416 6-5



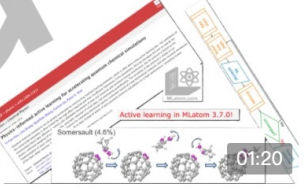
JCTC: 使用量子力学和机器学习方法模拟面跳跃动力学
582 6-12



分子红外光谱模拟现在上线啦！
1136 6-20



UAIQM1: 通用且可更新的AI模型让你火力全开！
355 6-27



主动学习：数据高效的机器学习势构建
50 1小时前

What year was this Review published?

Review

Neural networks: A new method for solving chemical problems or just a passing phase?

Authors??

Recent work on **neural networks in chemistry** is reviewed and essential background to this fast-spreading method is given. Emphasis is placed on the back-propagation algorithm, because of the extensive use of this form of learning. Hopfield networks, adaptive bidirectional associative memory, and Kohonen learning are briefly described and discussed. **Applications in spectroscopy (mass, infrared, ultraviolet, NMR), potentiometry, structure/activity relationships, protein structure, process control and chemical reactivity are summarized.**

Analytica Chimica Acta, 248 (1991) 1–30
Elsevier Science Publishers B V , Amsterdam

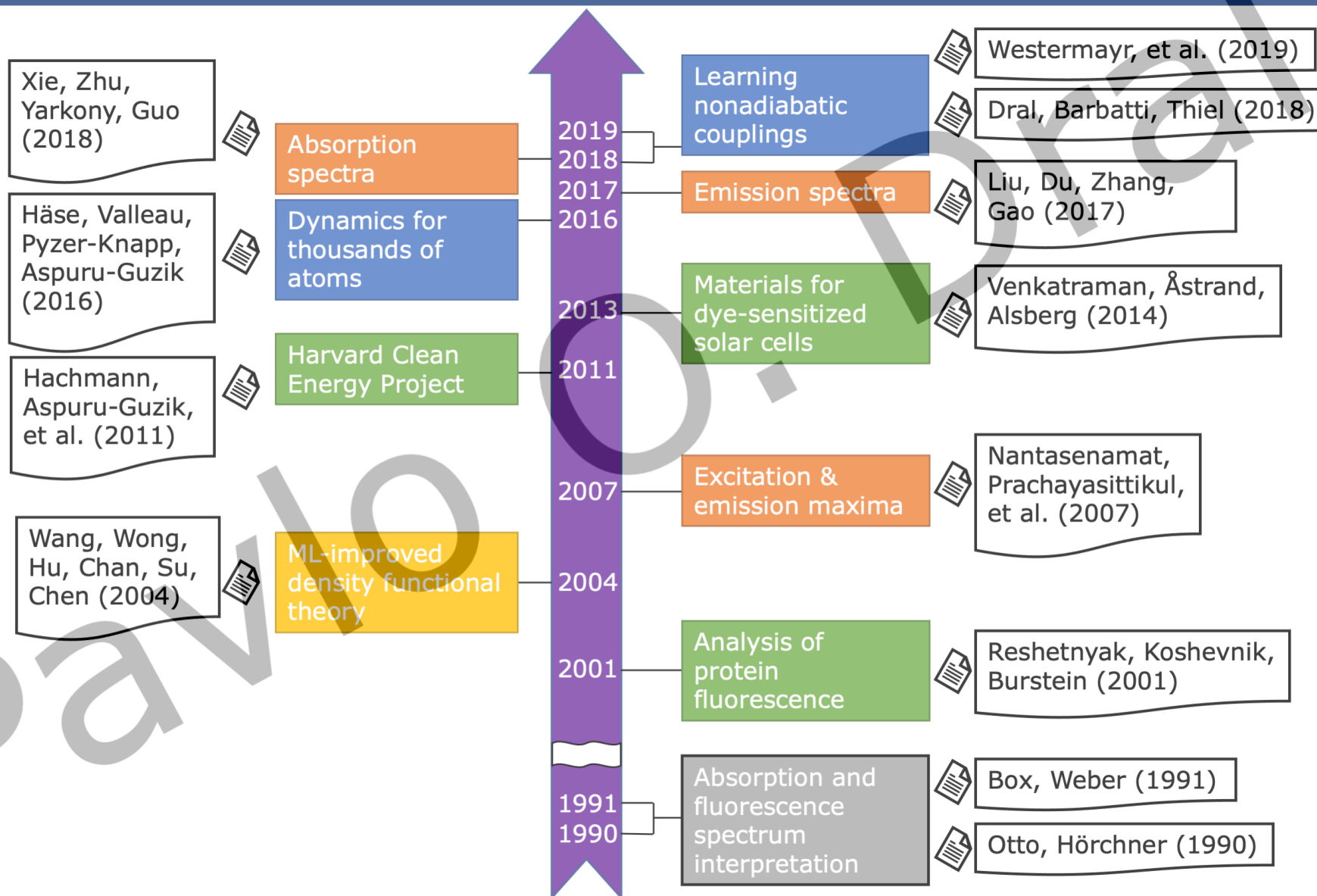
Review

Neural networks: A new method for solving chemical problems or just a passing phase?

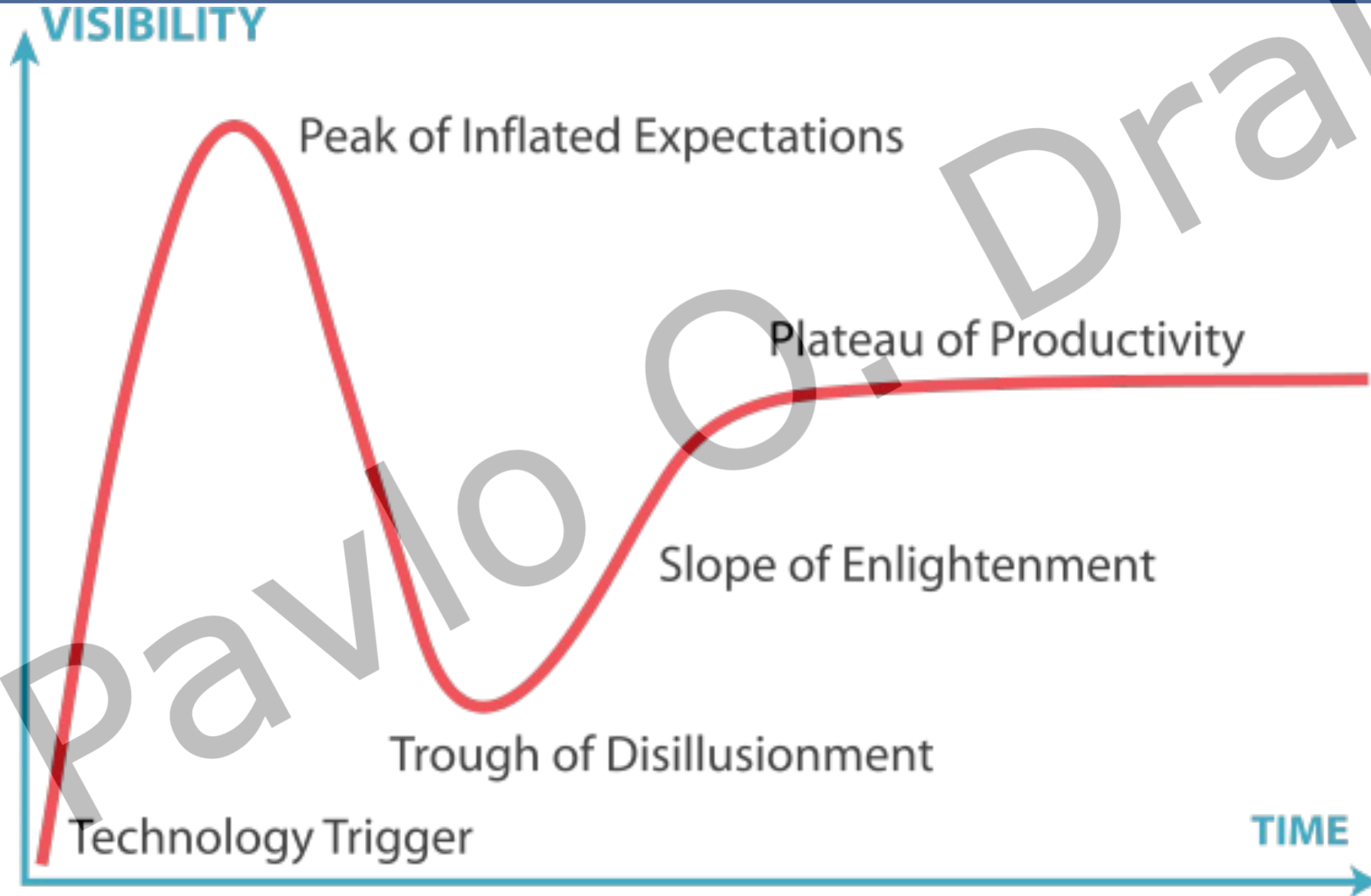
J. Zupan ^{*.1} and J. Gasteiger

Organisch-chemisches Institut, Technische Universität München, D-8046 Garching (Germany)

Recent work on **neural networks in chemistry** is reviewed and essential background to this fast-spreading method is given. Emphasis is placed on the back-propagation algorithm, because of the extensive use of this form of learning. Hopfield networks, adaptive bidirectional associative memory, and Kohonen learning are briefly described and discussed. **Applications in spectroscopy (mass, infrared, ultraviolet, NMR), potentiometry, structure/activity relationships, protein structure, process control and chemical reactivity are summarized.**



Gartner's hype cycle

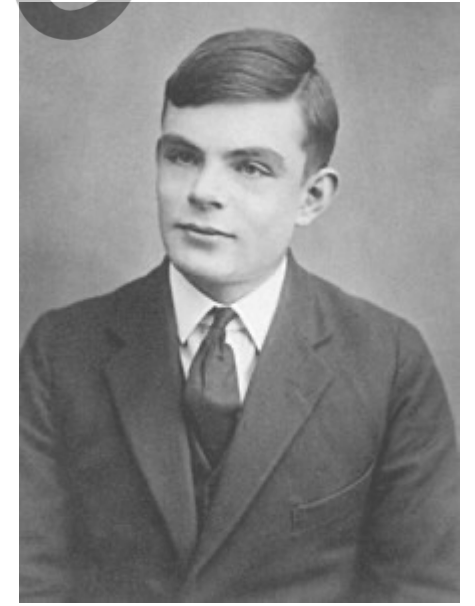




Number of possible move sequences in chess is very big:

No brute-force solution of chess is possible

- Making machines play chess was considered by the pioneers of AI as an **important milestone for AI**.
- In **1950s** the first computer chess programs were developed with contributions by Alan Turing.
- In **1997** computer has for the first time defeated the best human player in a match



But are the computer chess programs **intelligent**?

The world champion Magnus Carlsen in 2017: "The problem is that it still feels like you are playing somebody **stupid** when you are playing the computer."



Photo by Andreas Kontokanis from Piraeus, Greece (Carlsen Magnus) [CC BY-SA 2.0], via Wikimedia Commons


The chess programs Magnus Carlsen has been talking about are **coded explicitly by humans**. Algorithms and hardware have been improved ca. **70 years**.

Can ML beat 70 years of human effort?

Pavlo

The name 'Machine learning' (ML) implies that machines try to learn from [Big] data by themselves **without being programmed explicitly** by humans.

ML is an application of a broader artificial intelligence (AI) field to practical problems.

On 5 December 2017 a pre-print by Google's DeepMind team[1] has been published about AlphaZero ML-based program:[2]  DeepMind

- It was trained only by self-play for **8 hours**
- It could beat the best computer chess program in a match
- Chess grandmasters describe AlphaZero's play as **human-like** and were impressed by its positional and attacking play

[1] D. Hassabis et al., *arXiv:1712.01815*, **2017**

[2] D. Hassabis et al., *Science* **2018**, 362, 1140

This is a power of ML: it can (self-)learn so fast that it delivers better results than programs coded explicitly by humans over decades!

Pavlo O. Dral

This is a power of ML: it can (self-)learn so fast that it delivers better results than programs coded explicitly by humans over decades!

The “only” thing it requires is lots of data and good hardware: we have it all nowadays

Pavlo O. Dral

Chemistry is rich in Big Data:

- Number of possible compounds is infinite
- Number of points on a potential energy surface (PES) is infinite

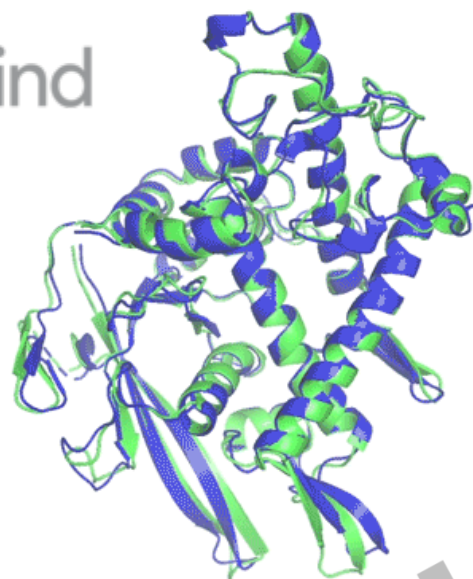
Pavlo O. Dral

Chemistry is more difficult than chess:

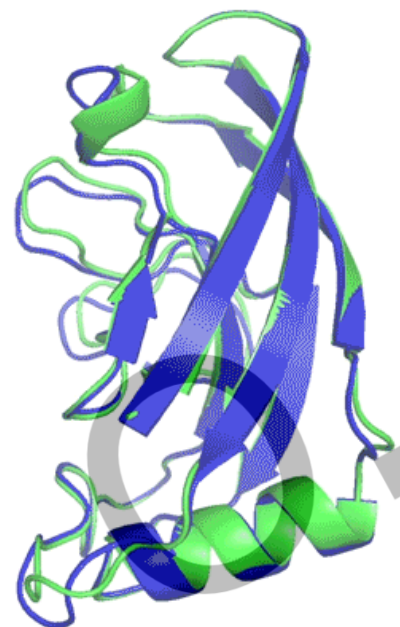
After many years of ML in chemistry we are only starting to use its potential

Pavlo O. Dral

ML in chemistry: AlphaFold (2)

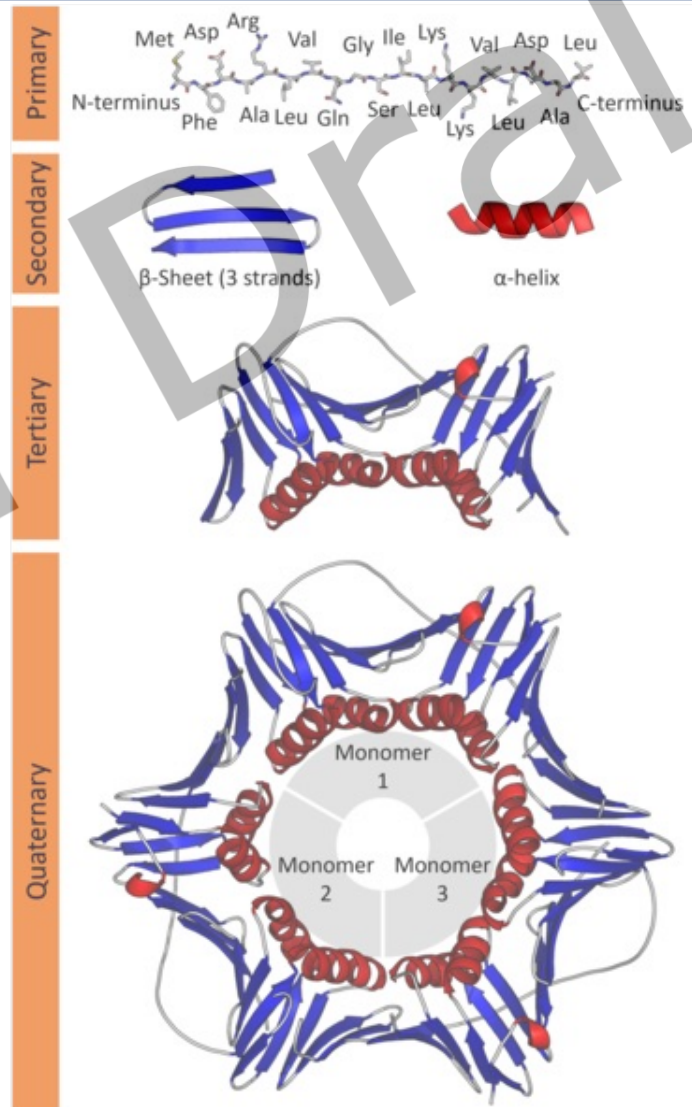


T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



T1049 / 6y4f
93.3 GDT
(adhesin tip)

● Experimental result
● Computational prediction



Thomas Shafee, CC BY 4.0, via Wikimedia Commons

Time-independent Schrödinger equation

$$\hat{H}\Psi = E\Psi$$

$$\hat{H} = -\sum_{i=1}^N \frac{1}{2} \nabla_i^2 - \sum_{A=1}^M \frac{1}{2M_A} \nabla_A^2 - \sum_{i=1}^N \sum_{A=1}^M \frac{Z_A}{r_{i,A}} + \sum_{i=1}^N \sum_{j>i}^N \frac{1}{r_{i,j}} + \sum_{A=1}^M \sum_{B>1}^M \frac{Z_A Z_B}{R_{A,B}}$$

\hat{H} - Hamiltonian

Ψ - wavefunction

E - energy

Pavlo O. Dral

Bond length in H_2

Experiment:

0.7414 Å

Quantum Chemistry:
(FCI/aug-cc-pV6Z)

0.7415 Å, ~5 CPU-days


Pavlo

O. Dral

The Schrödinger equation

$$\hat{H}\Psi = E\Psi$$

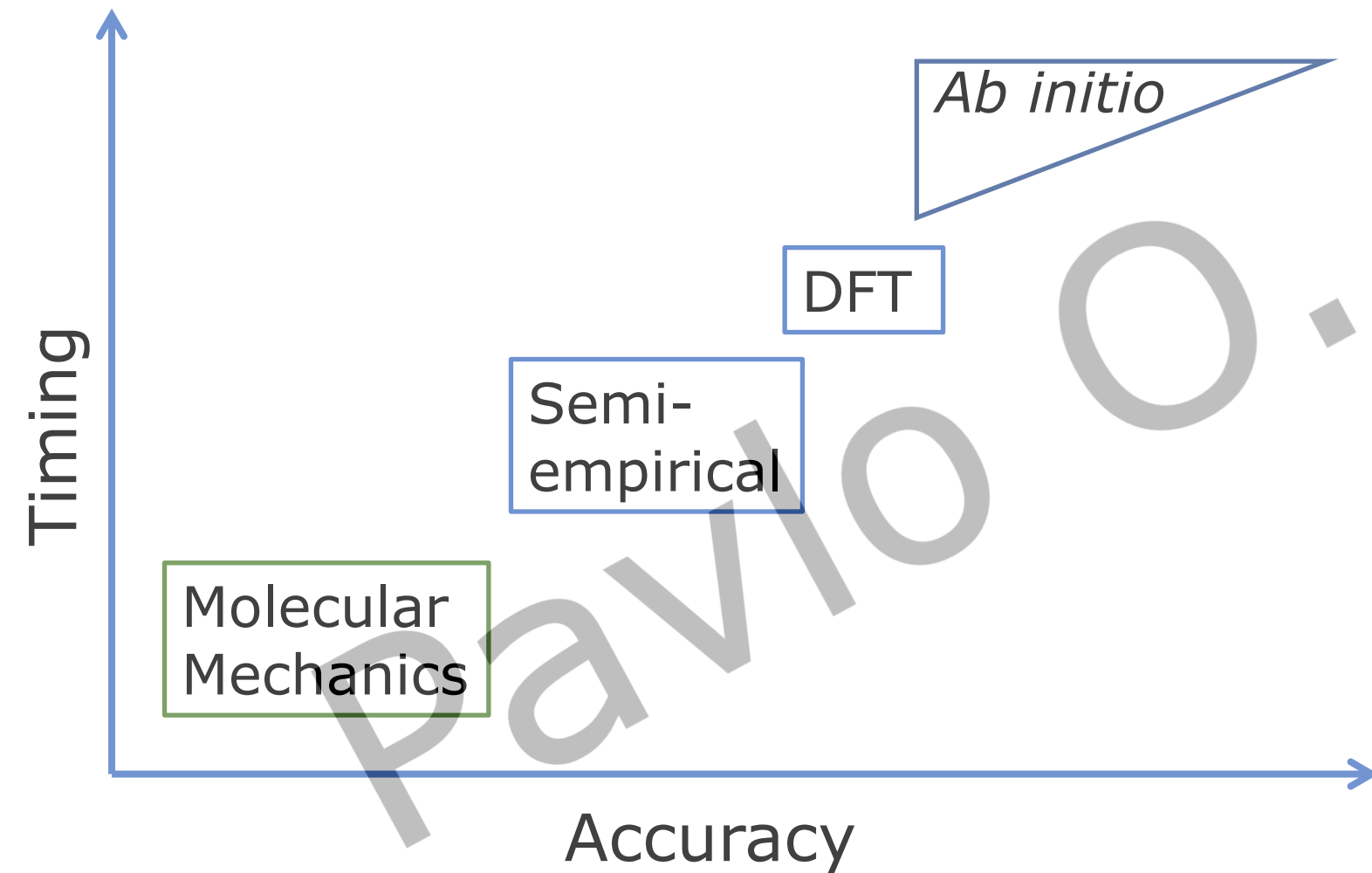
$$\hat{H} = -\sum_{i=1}^N \frac{1}{2} \nabla_i^2 - \sum_{A=1}^M \frac{1}{2M_A} \nabla_A^2 - \sum_{i=1}^N \sum_{A=1}^M \frac{Z_A}{r_{i,A}} + \sum_{i=1}^N \sum_{j>i}^N \frac{1}{r_{i,j}} + \sum_{A=1}^M \sum_{B>1}^M \frac{Z_A Z_B}{R_{A,B}}$$



 The electronic Schrödinger equation can be solved exactly **only** for two-body systems (e.g. hydrogen atom with 1 nucleus + 1 electron)



Approximations are needed!



Experiment: 0.7414 Å

Quantum Chemistry: 0.7415 Å, ~5 CPU-days
(FCI/aug-cc-pV6Z)

Machine learning (ML): ??? Å, time?

Pavlo O. Dral

B3LYP/6-31G*

geomopt

xyz **GFN2-xTB**

geomopt

xyz **MP2/cc-pVDZ**

geomopt

xyzfile **ANI-1ccx**

geomopt

xyzfile **MLmodelType=ANI MLmodelIn=ani.pt**

geomopt

xyz **UAIQM**

geomopt

xyzfile='5'

C	0.0000000000	0.0000000000	0.0000000000
H	1.0870000000	0.0000000000	0.0000000000
H	-0.3623333220	-1.0248334322	-0.0000000000
H	-0.3623333220	0.5124167161	-0.8875317869
H	-0.3623333220	0.5124167161	0.8875317869
'			

VALENCE BOND THEORY

1. The Understanding of H₂
2. Computing of F₂
3. Resonance in C₆H₆
4. Computing of O₂
5. Computing of post-VBSCF methods
6. Computing of diabatic states with VB theory
7. Menshutkin Reaction NH₃ + CH₃Cl → [NH₃CH₃]⁺ + Cl⁻
8. Charge-shift bonding in propellane

MACHINE LEARNING

1. Machine learning basics

- 1.1. Power of ML
- 1.2. Training your first ML model
- 1.3. ML algorithms
2. ML for PES
3. Universal machine learning models
4. Spectroscopy

[Home](#) / 1. Machine learning basics

1. Machine learning basics

1.1. Power of ML

Power of ML that it can make fast and accurate.

For example, let's use the pure ML model trained for the hydrogen molecule to optimize its geometry:

Example ML-basics-1.

Calculate the bond length in H₂ molecule with machine learning (ML) (see instructions below).

Write down:

1. Bond length you obtained.
2. How much time it took to complete the calculations.

The MLatom@XACS input file ([h2_opt_kreg.inp](#)) is pretty self-explanatory:

```
geomopt                # Request geometry optimization
MLmodelType=KREG       # with the ML model of the KREG type
MLmodelIn=energies.unf # in energies.unf file
XYZfile='
2

H          0.000          0.000          0.000
H          0.000          0.000          0.800
```

Example ML-basics-1.

Calculate the bond length in H₂ molecule with machine learning (ML) (see instructions below).

Write down:

1. Bond length you obtained.
2. How much time it took to complete the calculations.

The MLatom@XACS input file ([h2_opt_kreg.inp](#)) is pretty self-explanatory:

```
geomopt           # Request geometry optimization
MLmodelType=KREG  # with the ML model of the KREG type
MLmodelIn=energies.unf # in energies.unf file
XYZfile='
2
H      0.000      0.000      0.000
H      0.000      0.000      0.800
.
optXYZ=eq_KREG.xyz # optimized geometry output
```


You also need to upload the file with ML model [energies.unf](#) as *auxiliary file* on the XACS cloud.

↑ or edit XACS input file:

```
1 geomopt          # Request geometry optimization
2 MLmodelType=KREG # with the ML model of the KREG type
3 MLmodelIn=energies.unf # in energies.unf file
4 XYZfile='
5 2
6
7 H          0.000      0.000      0.000
8 H          0.000      0.000      0.800
9 '
10 optXYZ=eq_KREG.xyz # optimized geometry output
```

Additional auxiliary file (optional)

↑ auxiliary file(s) 

Name	Operation
energies.unf	

Experiment: 0.7414 Å

Quantum Chemistry: 0.7415 Å, ~5 CPU-days
(FCI/aug-cc-pV6Z)

Machine learning (ML): ??? Å, time?

Pavlo O. Dral

Experiment: 0.7414 Å

Quantum Chemistry: 0.7415 Å, ~5 CPU-days
(FCI/aug-cc-pV6Z)

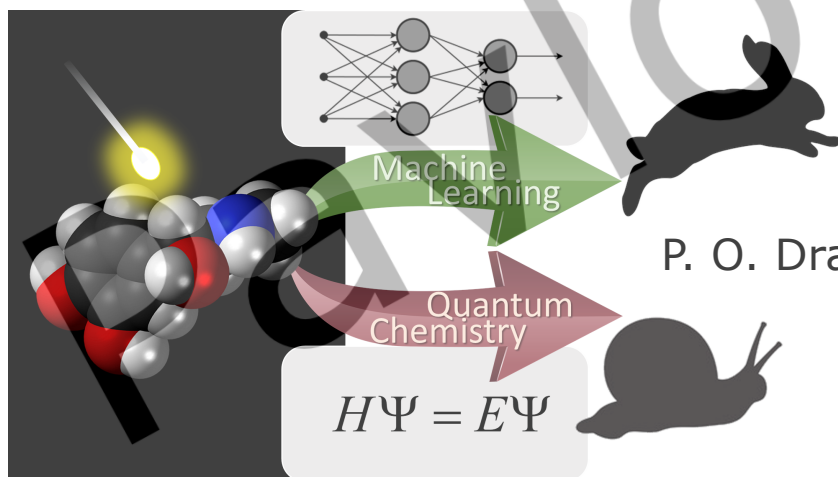
Machine learning (ML): 0.7415 Å, ~0.3 seconds

Pavlo O. Dral

Experiment: 0.7414 Å

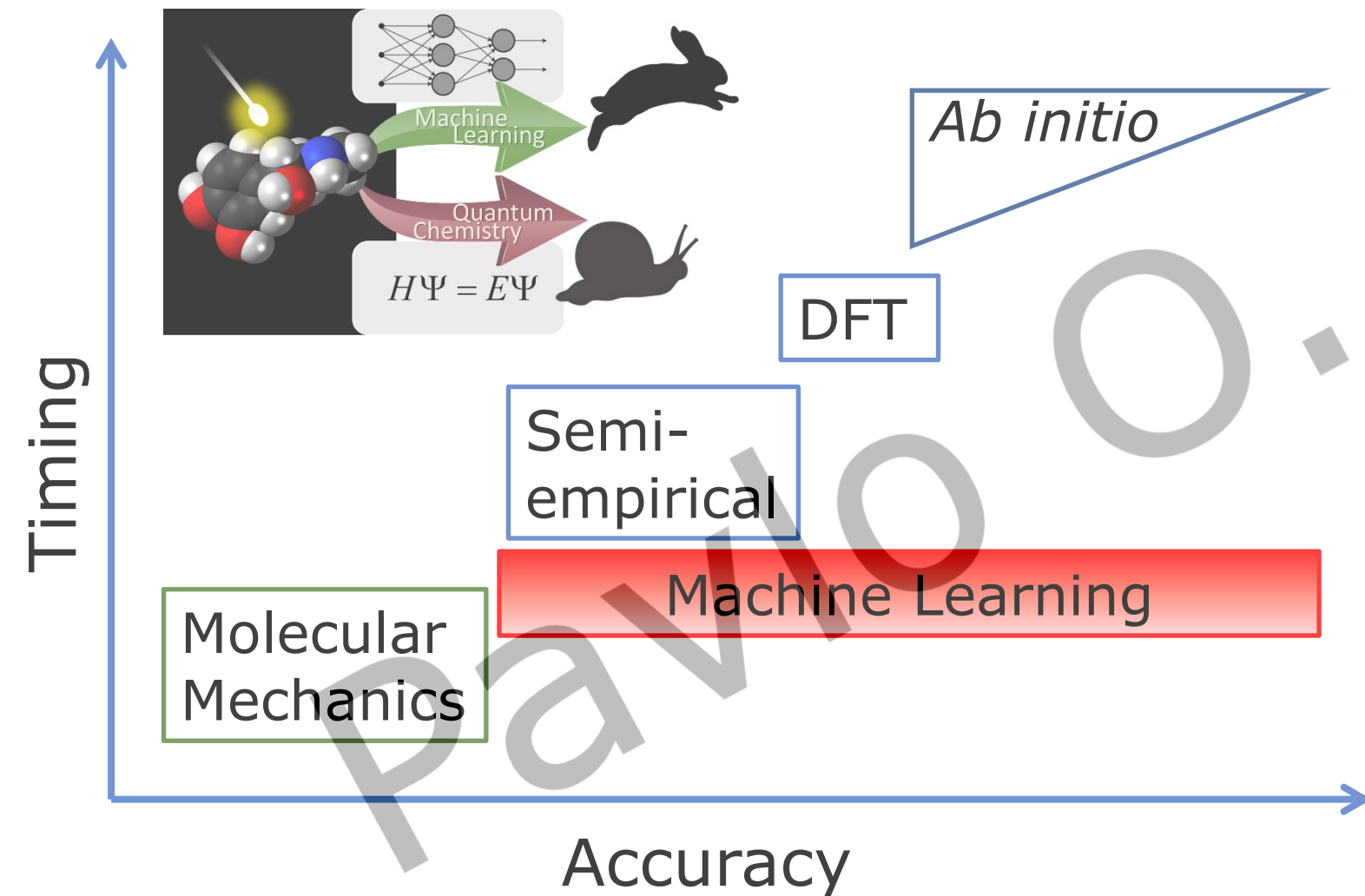
Quantum Chemistry: 0.7415 Å, ~5 CPU-days
(FCI/aug-cc-pV6Z)

Machine learning (ML): 0.7415 Å, ~0.3 seconds



P. O. Dral, M. Barbatti, *Nat. Rev. Chem.* **2021**, 5, 388

P. O. Dral, M. Barbatti, *Nat. Rev. Chem.* **2021**, 5, 388



The name 'Machine learning' (ML) implies that machines try to learn from [Big] data by themselves **without being programmed explicitly** by humans.

ML is an application of a broader artificial intelligence (AI) field to practical problems.

Pavlo

Quantum Chemistry

a) solve $\hat{H}\Psi = E\Psi$



Training Data

b) calculate E_{ML} etc.

c) solve $\hat{H}_{ML}\Psi = E\Psi$

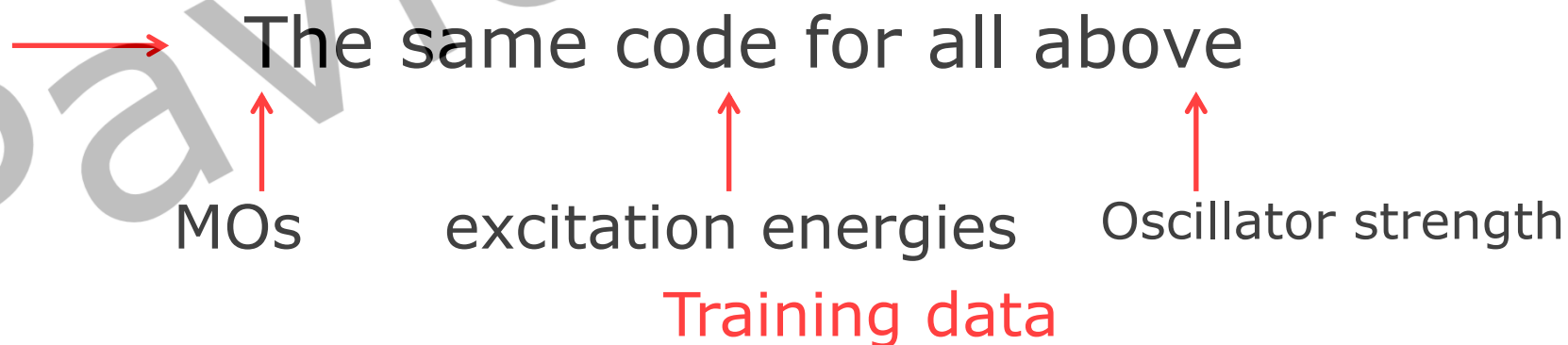
d) predict and use Ψ_{ML}

Machine
Learning

Conventional programming in quantum chemistry:

- Code for molecular orbitals
- Code for excitation energies
- Code for oscillator strengths
- ...

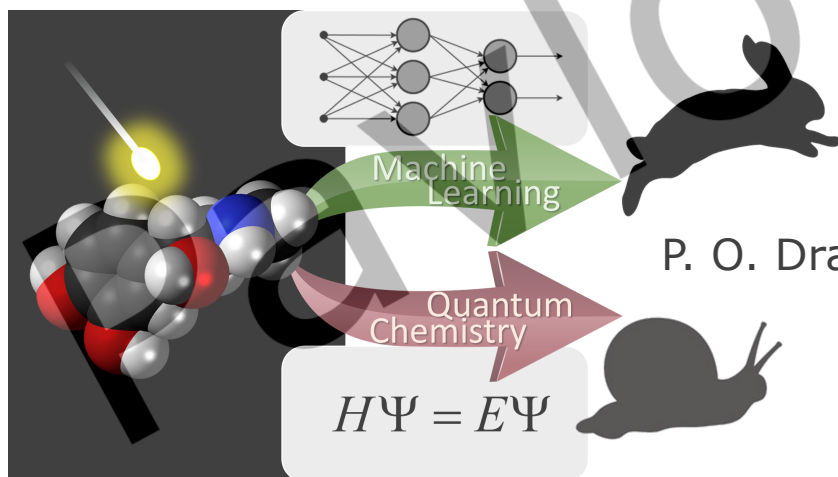
Machine learning (in principle – adaptations required!):



Experiment: 0.7414 Å

Quantum Chemistry: 0.7415 Å, ~5 CPU-days
(FCI/aug-cc-pV6Z)

Machine learning (ML): 0.7415 Å, ~0.3 seconds



P. O. Dral, M. Barbatti, *Nat. Rev. Chem.* **2021**, 5, 388

How to get such an ML model?

- data
 - energies at FCI: E_FCI_451.dat)
 - XYZ geometries (h2.xyz)
- ML model
 - KREG (train.inp)

Train the ML model for the H₂ molecule (see instructions below) and obtain the bond length with this model.

Write down:

1. How much time did it take to train the model?
2. What are the training and validation errors?
3. What is the bond length of H₂ obtained with your model?

The MLatom@XACS input file ([h2_train_KREG.inp](#)) with explanation in comments:

```
createMLmodel      # Specify the task for MLatom
MLmodelType=KREG   # Specify the model type
MLmodelOut=energies.unf # Save model in energies.unf
XYZfile=h2.xyz     # File with XYZ geometries
Yfile=E_FCI_451.dat # The file with FCI energies (in Hartree)
sigma=opt          # Optimize hyperparameter sigma
lgSigmaL=-4        # Lower bound of log2(sigma)
lambda=opt         # Optimize hyperparameter lambda
```

For now, do not worry about hyperparameters in input file, I will explain them in another lecture.

You need two auxiliary files with data:

- [E_FCI_451.dat](#) - energies in Hartree.
- [h2.xyz](#) - geometries.

You can examine the data files to get an idea what they contain.

Types of Machine Learning

Pavlo O. Dral

Supervised Machine Learning

Input (x) \rightarrow $f(x)$ \rightarrow Output (y)



Given collection of known $\{x,y\}$ find a function $f(x)$

Pavlo

Supervised Machine Learning

Input (x) \rightarrow $f(x)$ \rightarrow Output (y)

Given collection of known $\{x,y\}$ find a function $f(x)$

Use this function for making new predictions given just $\{x'\}$

Supervised Machine Learning

Input (x) \rightarrow $f(x)$ \rightarrow Output (y)

Given collection of known $\{x,y\}$ find a function $f(x)$
training set \rightarrow train \uparrow ML model

Use this function for making new predictions given just $\{x'\}$

Pavlio O. Dral

Unsupervised Machine Learning

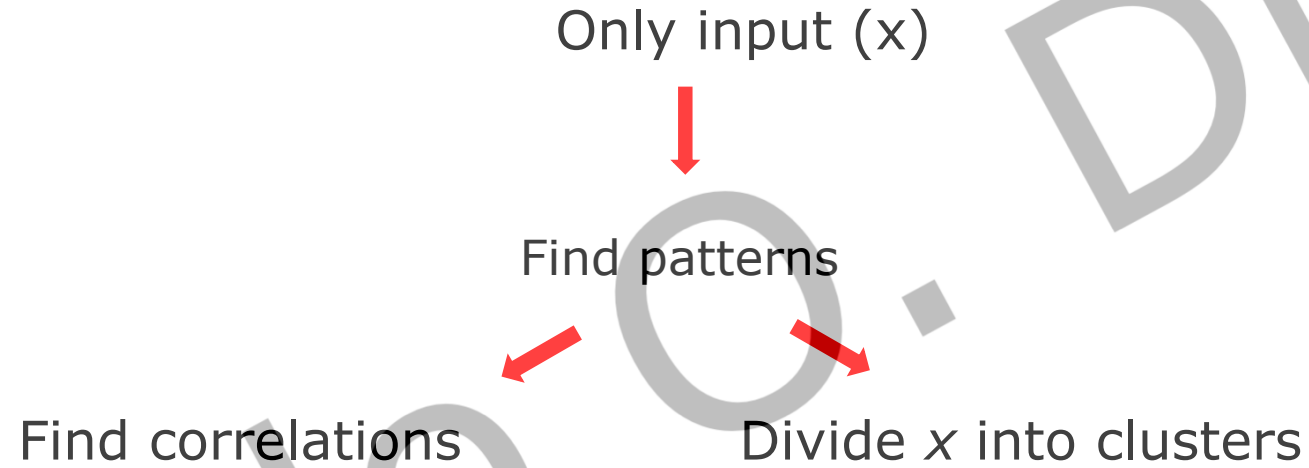
Pavlo O. Dral

Unsupervised Machine Learning

Only input (x)

Pavlo O. Dral

Unsupervised Machine Learning



Pavlio O. Dral

Supervised Machine Learning

Input (x) \rightarrow $f(x)$ \rightarrow Output (y)

Given collection of known $\{x,y\}$ find a function $f(x)$
training set \rightarrow train \uparrow ML model

Use this function for making new predictions given just $\{x'\}$

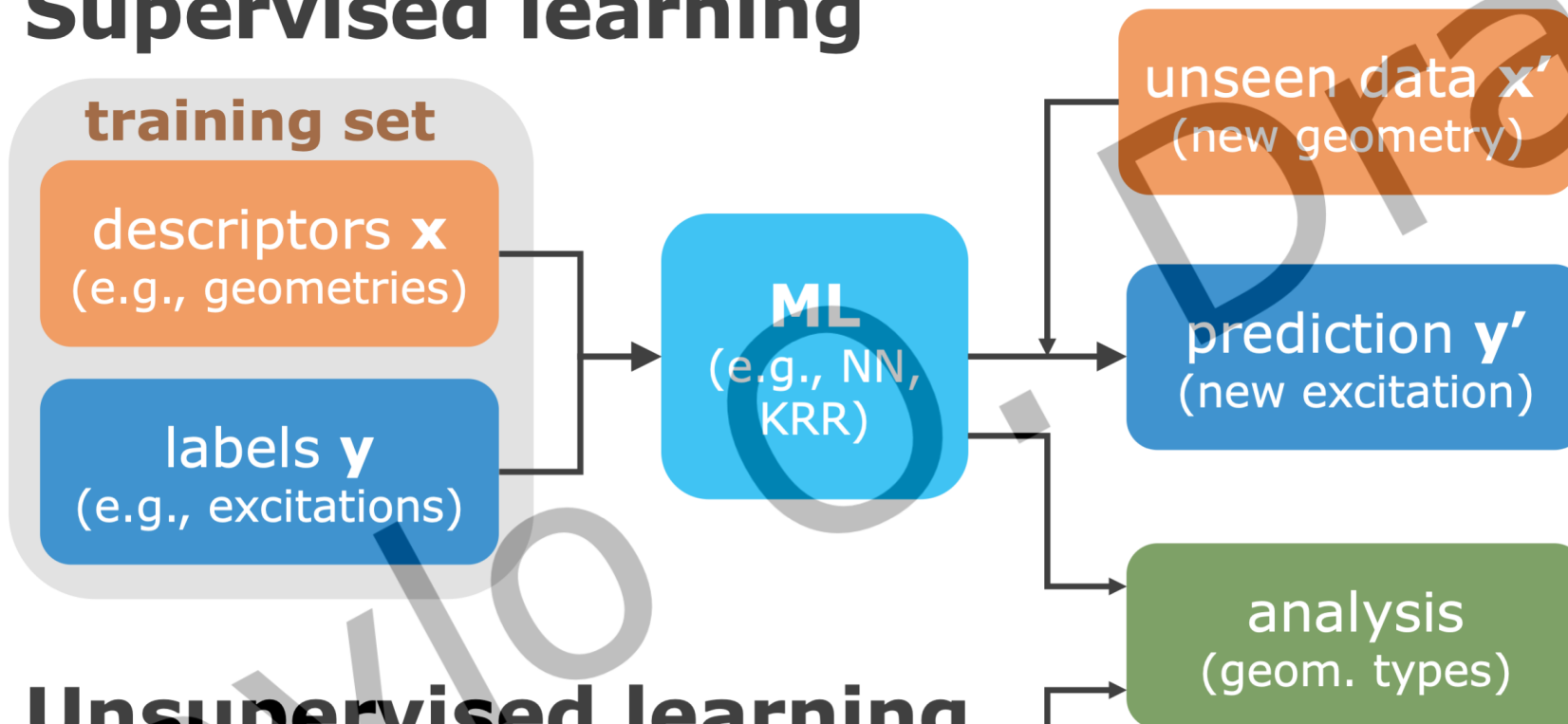
Unsupervised Machine Learning

Only input (x)

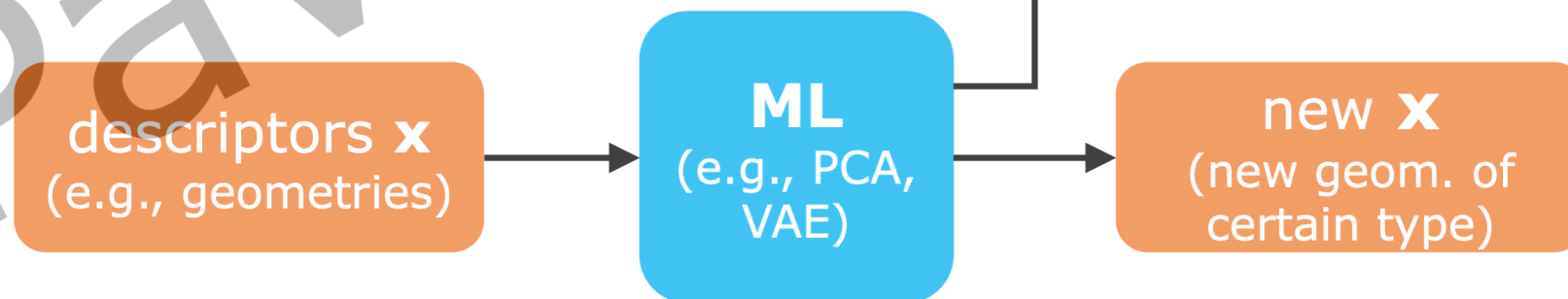
Find correlations

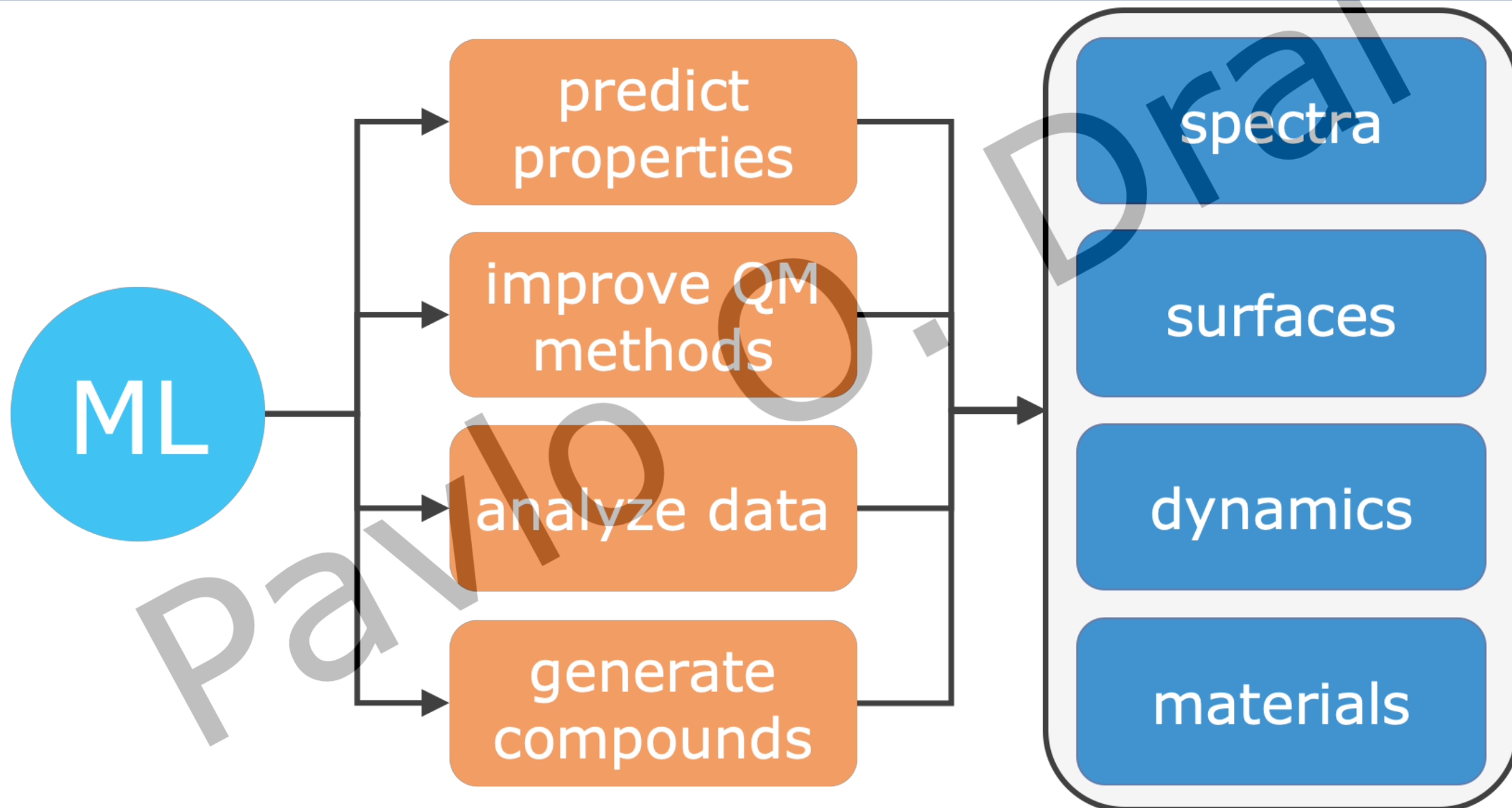
Divide x into clusters

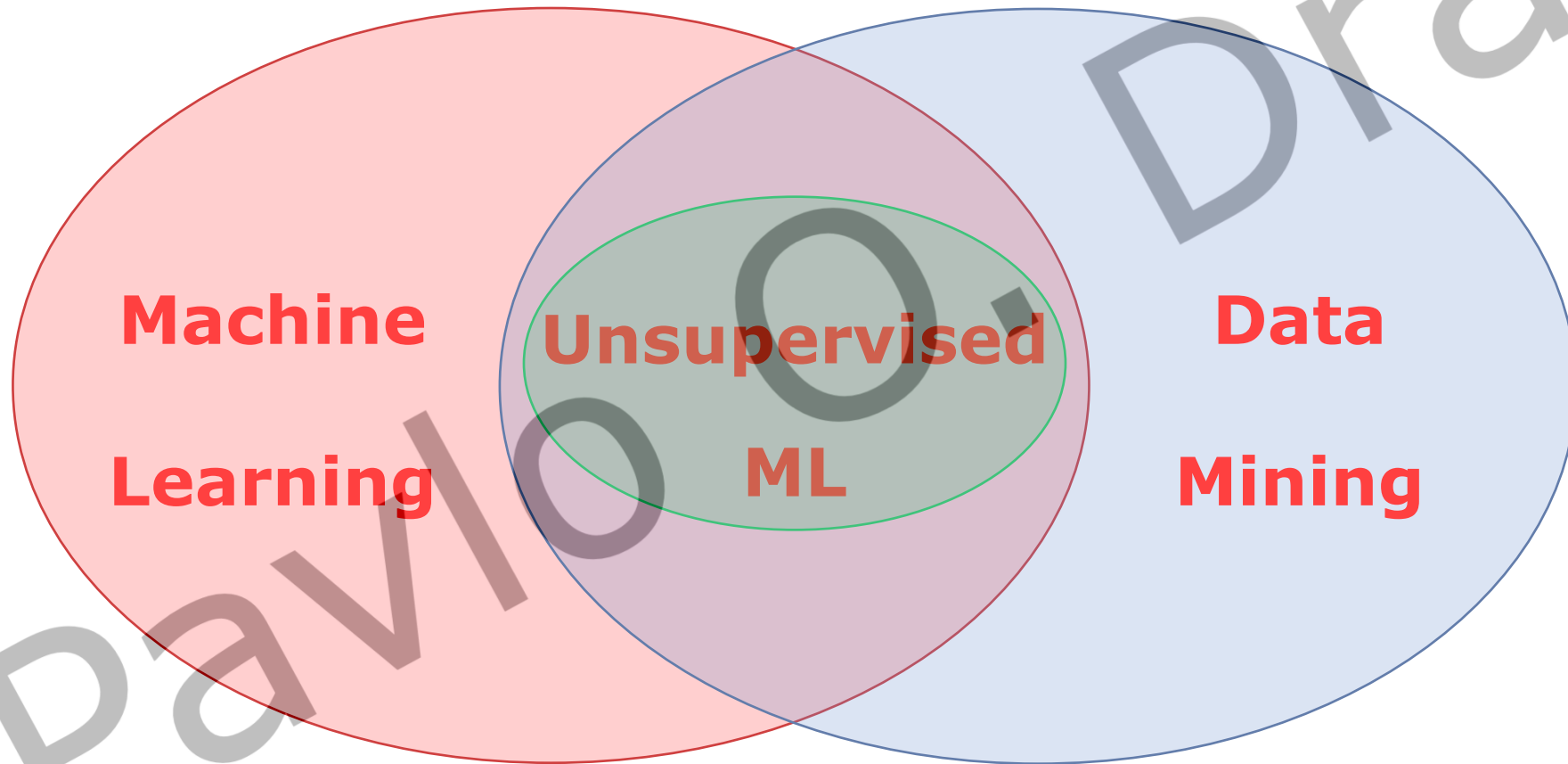
Supervised learning



Unsupervised learning



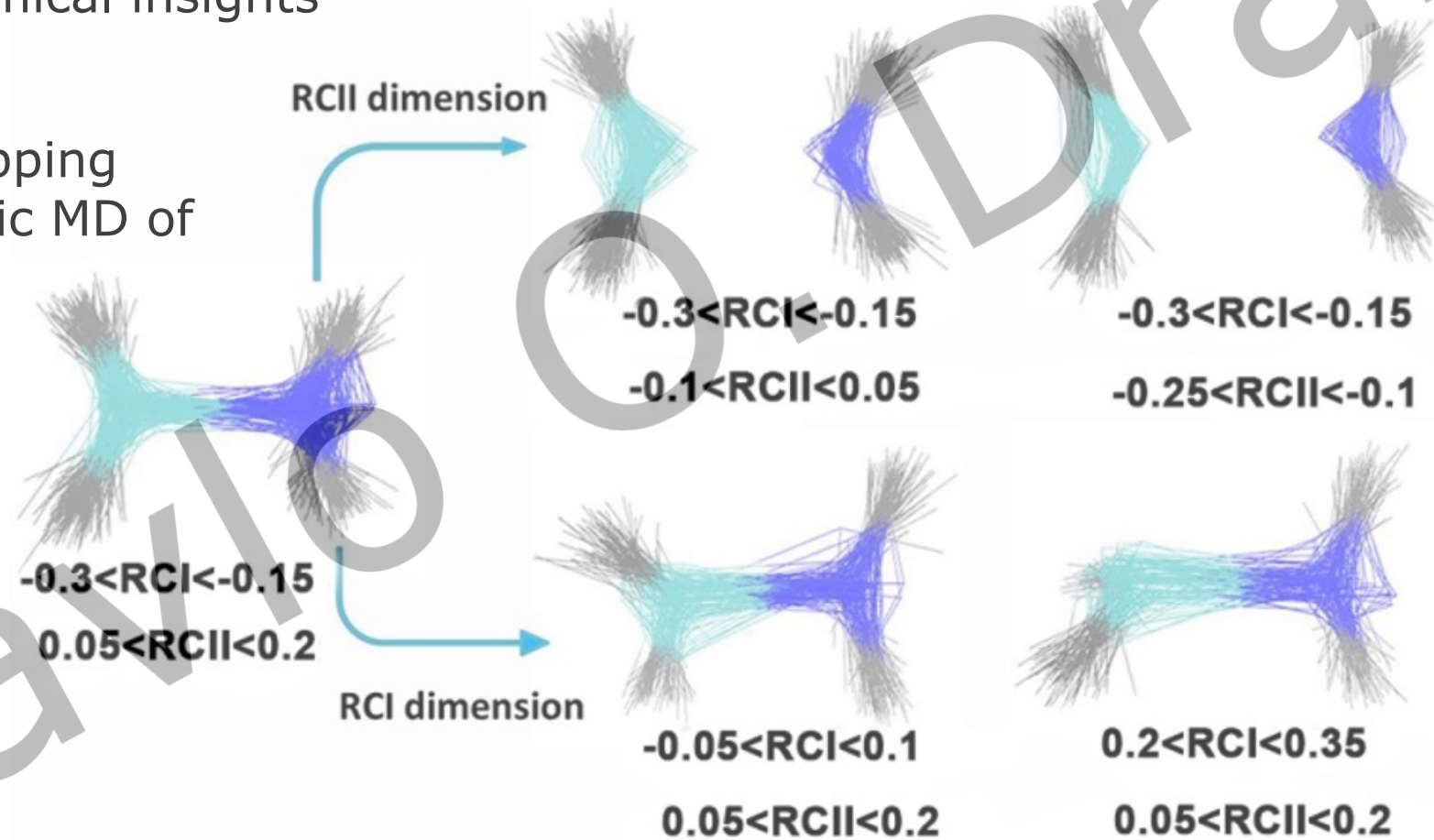




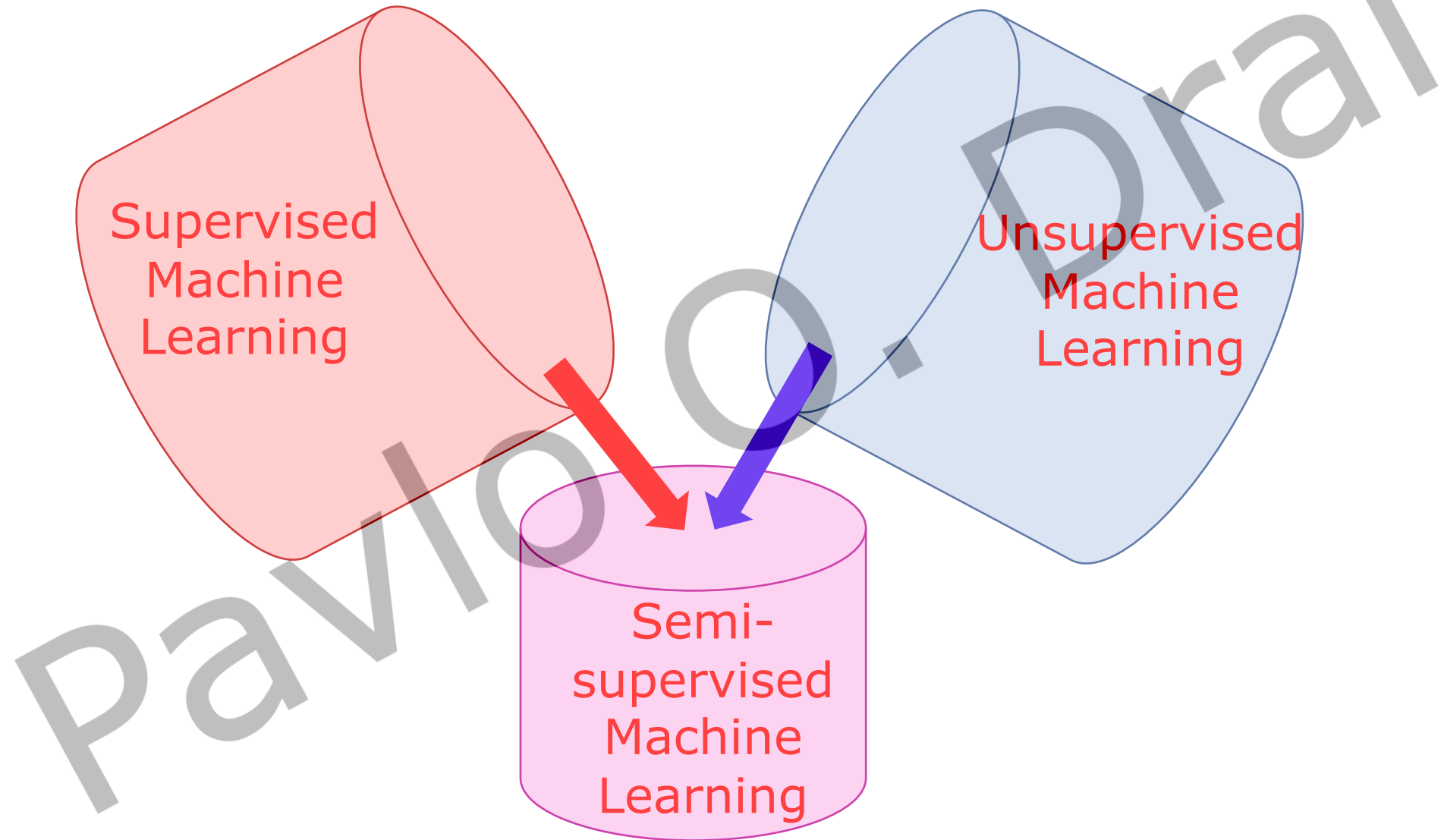
Unsupervised ML is useful for analyzing MD trajectories and getting physicochemical insights

On-the-fly
surface-hopping
nonadiabatic MD of
 CH_2NH_2^+

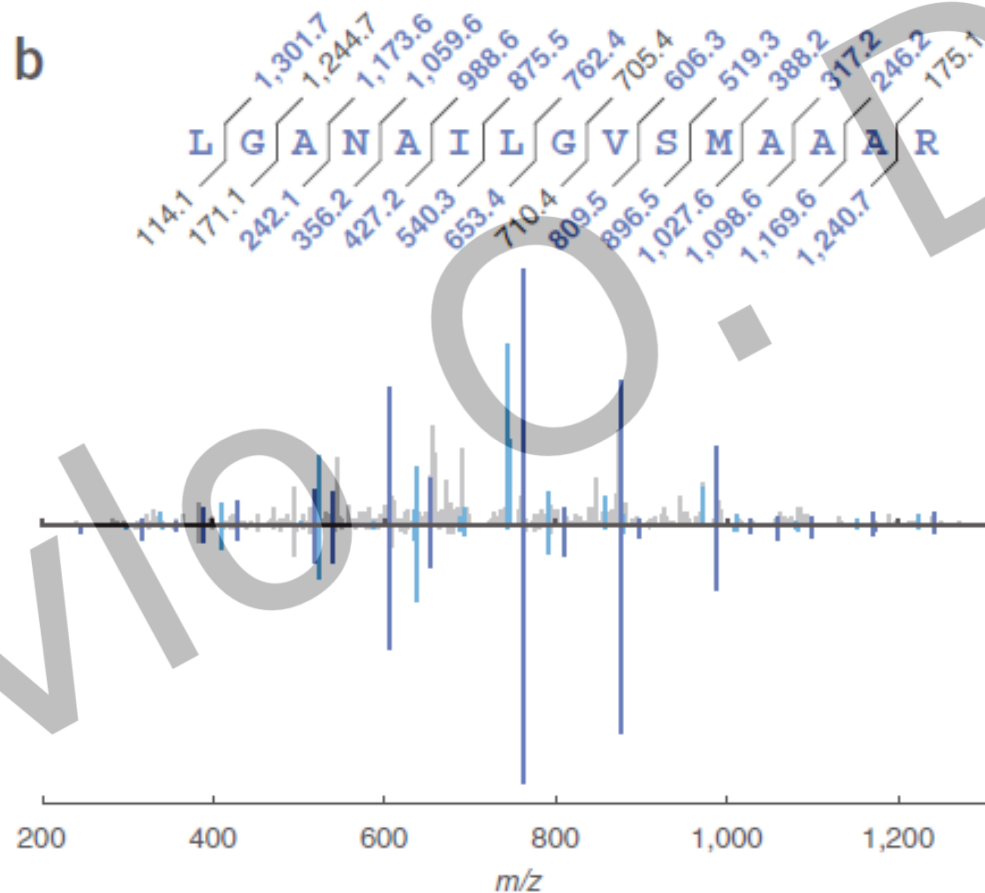
PS6



Types of Machine Learning



Semi-supervised ML is often used in protein research, e.g. it can improve identification of correct peptide from mass-spectra



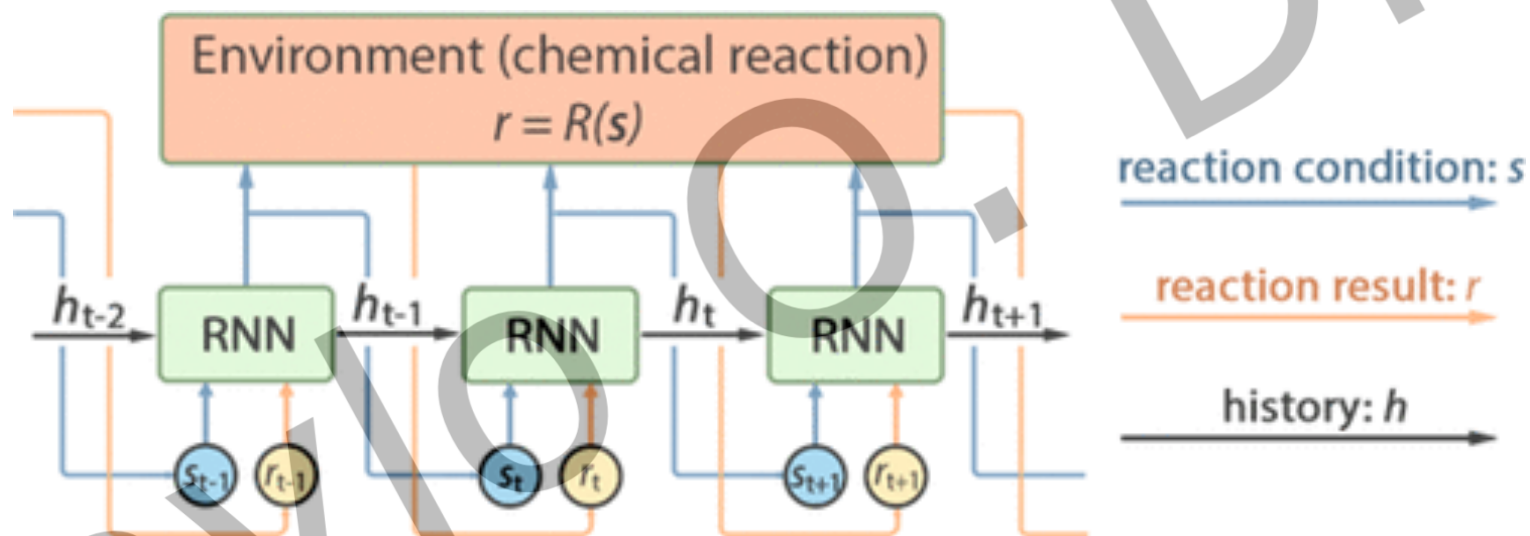
Reinforcement Learning

ML tries to maximize rewards from environment

In case of chess:
Tries to maximize the number of wins



For example, reinforcement learning can be used in chemistry for optimizing chemical reactions



Supervised Machine Learning

Input (x) \rightarrow $f(x)$ \rightarrow Output (y)

Given collection of known $\{x,y\}$ find a function $f(x)$
training set \rightarrow train \uparrow ML model

Use this function for making new predictions given just $\{x'\}$

- Data
- Choice of x (descriptor)
- Choice of y (labels)
- Fitting function (ML algorithm, ML model)
- Optimization of ML model parameters

(Supervised)

Machine learning serves for function approximation[1]

ML takes little time for making new predictions

Quantum Chemical Property(molecule) = function(nuclear coordinates)

What we need to have:

- Data
- **QC Model (Hamiltonian)**
vs
ML Model (not constrained by physical model)
- Parameters

What we need to do:

- Fit parameters to achieve an optimization goal

The difference between QC and ML models

- The goal of QC model (Hamiltonian) is to be physically correct as much as possible
- The goal of ML Model is to generalize (not just to fit!) from data as good as possible

Pavlio O. Dral

Q: What parameters are in HF/3-21G?

Pavlo O. Dral

Parameters (in red) are in the basis set, e.g., for hydrogen at 3-21G:

$$\varphi'_{1s}(\mathbf{r}) = \sum_{i=1}^2 d'_{i,1s} \left(\frac{8\alpha'^3}{\pi^3} \right)^{1/4} \exp(-\alpha' r^2)$$

$$\varphi''_{1s}(\mathbf{r}) = \sum_{i=1}^2 \left(\frac{8\alpha''^3}{\pi^3} \right)^{1/4} \exp(-\alpha'' r^2)$$

How these parameters were obtained?
By optimizing them so that SCF atomic energy reaches minimum

Q: What parameters are in B3LYP?

Pavlo O. Dral

Parameters (in red) in B3LYP:

$$E_{XC} = (1 - a)E_X^{LDA} + aE_X^{HF} + bE_X^{GGA} + cE_C^{GGA} + (1 - c)E_C^{LDA}$$

How this parameters were obtained?

They were optimized on the G2 data set with atomization energies, ionization potentials, proton affinities and total atomic energies (a = 0.20, b = 0.72, and c = 0.81).

B3LYP: A. D. Becke. *J. Chem. Phys.* **1993**, 98, 1372.

Book: W. Koch, M. C. Holthausen, *A Chemist's Guide to Density Functional Theory*. Second ed.; WILEY-VCH Verlag GmbH: Weinheim, **2001**; pp. 293

ML algorithms

Pavlo O. Dral

- Various types of neural networks (NN), **deep learning**
- Gaussian processes (GP)
- Kernel ridge regression (KRR)
- Support vector machines (SVMs) & support vector regression (SVR)
- Linear regression!
- Decision trees
- k-Nearest neighbor algorithm
- and many more...

Parametric vs nonparametric algorithms

Pavlo

Dr. Dral

$f(x; \text{parameters})$

Linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

Number of parameters is fixed: parametric model

Neural networks are also parametric models

Kernel ridge regression (KRR)

$$f(\mathbf{x}_i; \mathbf{p}) = \sum_{j=1}^{N_{\text{tr}}} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j; \mathbf{b})$$

Number of parameters depends on number of training points:
nonparametric model, e.g. KRR

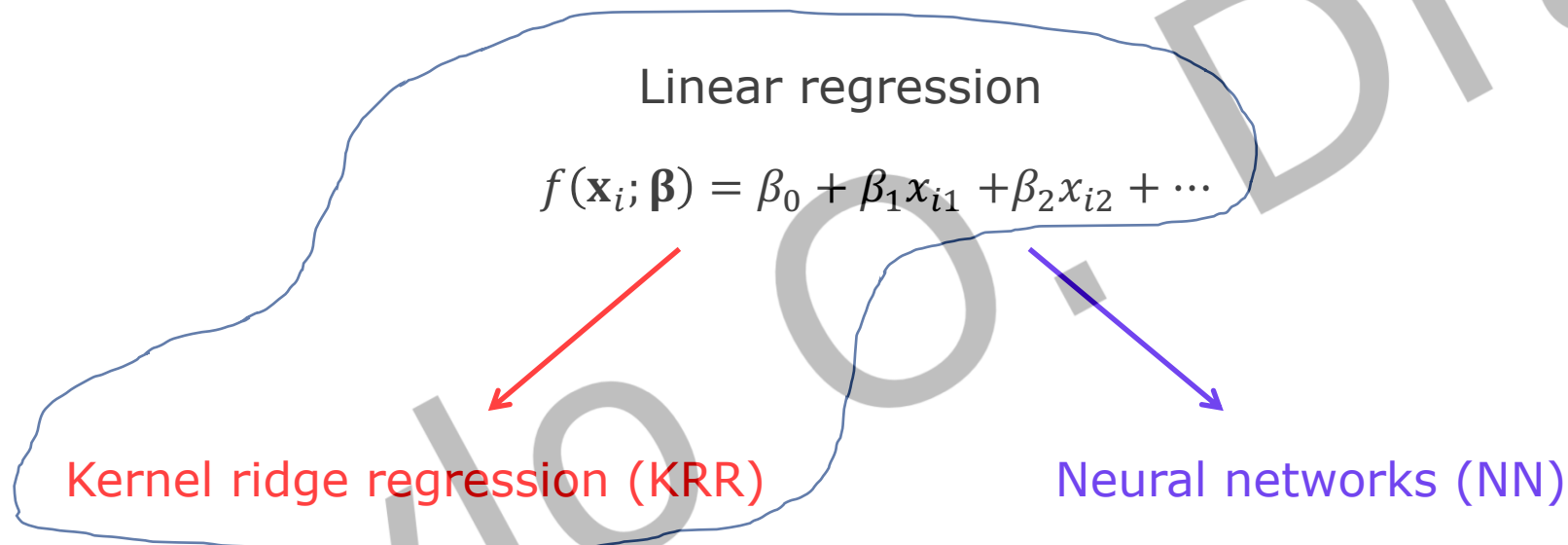
- Various types of neural networks (NN), **deep learning**
- Gaussian processes (GP)
- Kernel ridge regression (KRR)
- Support vector machines (SVMs) & support vector regression (SVR)
- Linear regression!
- Decision trees
- k-Nearest neighbor algorithm
- and many more...

Linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

Kernel ridge regression (KRR)

Neural networks (NN)



Multiple linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \sum_{j=1}^p \beta_j x_{ij}$$

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \mathbf{x}_i^T \boldsymbol{\beta}$$

How to find the coefficients β_j ?

Multiple linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \sum_{j=1}^p \beta_j x_{ij}$$

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \mathbf{x}_i^T \boldsymbol{\beta}$$

We can find the coefficients $\boldsymbol{\beta}$ using the method of least squares, where coefficients are fit to get the minimum residual sum of squares (RSS) with respect to the training set with N_{tr} reference values \mathbf{y} :

$$\arg \min_{\boldsymbol{\beta}} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \boldsymbol{\beta}) - y_i)^2$$

$$\arg \min_{\beta} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \beta) - y_i)^2$$

$$L(\beta) = \sum_{i=1}^{N_{tr}} (\mathbf{x}_i^T \beta - y_i)^2$$

$$L(\beta) = (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y})$$

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{N_{tr}1} & \cdots & x_{N_{tr}p} \end{pmatrix}$$

$$\frac{\partial L(\beta)}{\partial \beta} = 2\mathbf{X}^T (\mathbf{X}\beta - \mathbf{y}) = \mathbf{0}$$

Linear regression

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y}$$

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Linear regression has an analytical solution!

While it is very advantageous, it assumes that the data follow the linear distribution, which is often not the case

Pavlo

Dr-dral

Task 4: Fit linear model $E = aR$ on a training set with 20 points sampled along H_2 dissociation curve (energies E in Hartree at FCI/aug-cc-pV6Z; internuclear distances R in Angstrom)

Calculate R^2 and residual sum of squares (RSS)

Task 5: Fit linear regression with intercept b ?

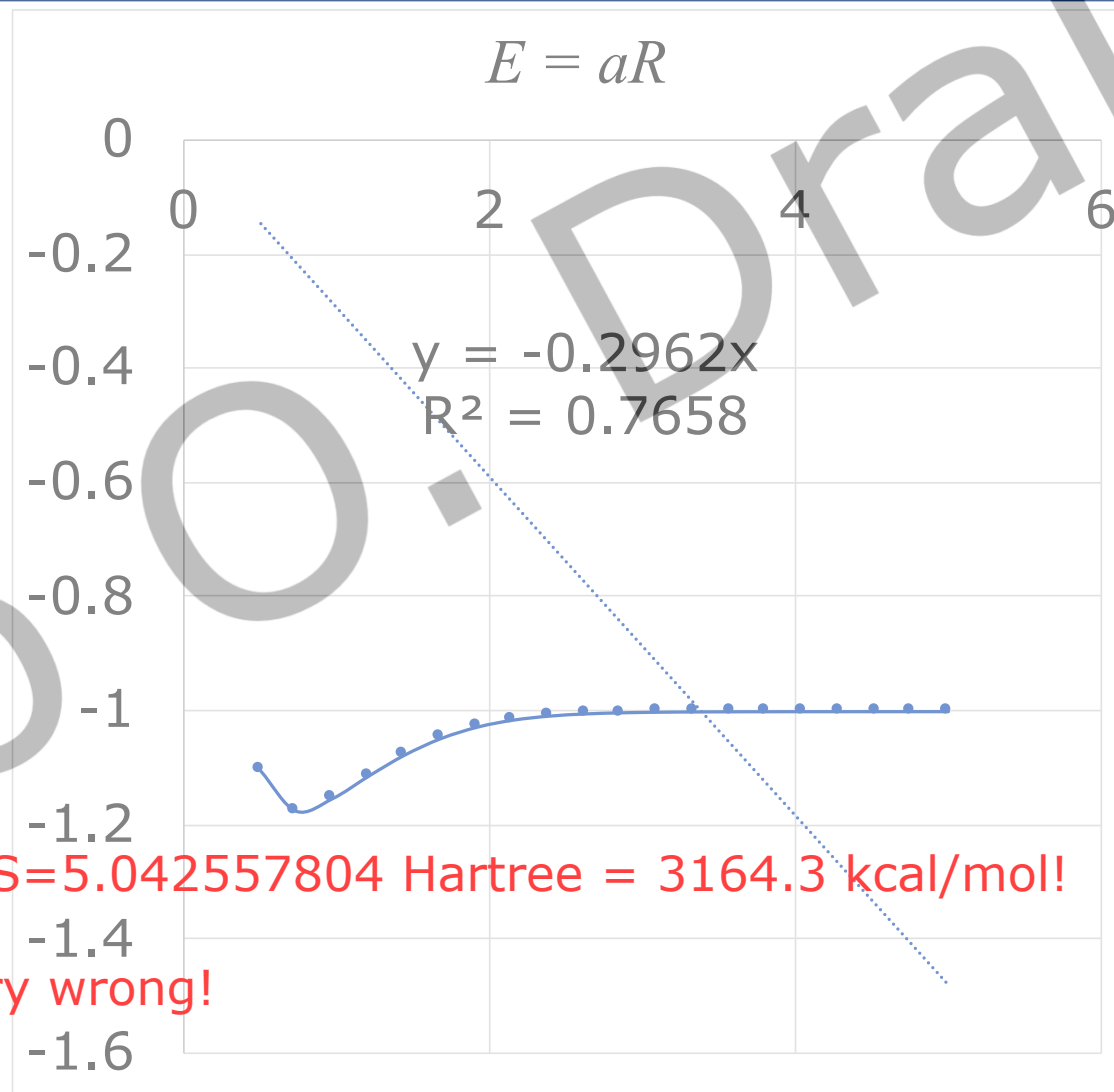
Calculate R^2 and residual sum of squares (RSS)

$$x_{i1} = R$$

$$E = aR + b$$
$$RSS = \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \boldsymbol{\beta}) - y_i)^2$$

Linear regression

H₂ dissociation curve



Task 5: Fit linear regression with intercept b ?

Calculate R^2 and residual sum of squares (RSS)

$$E = aR + b$$

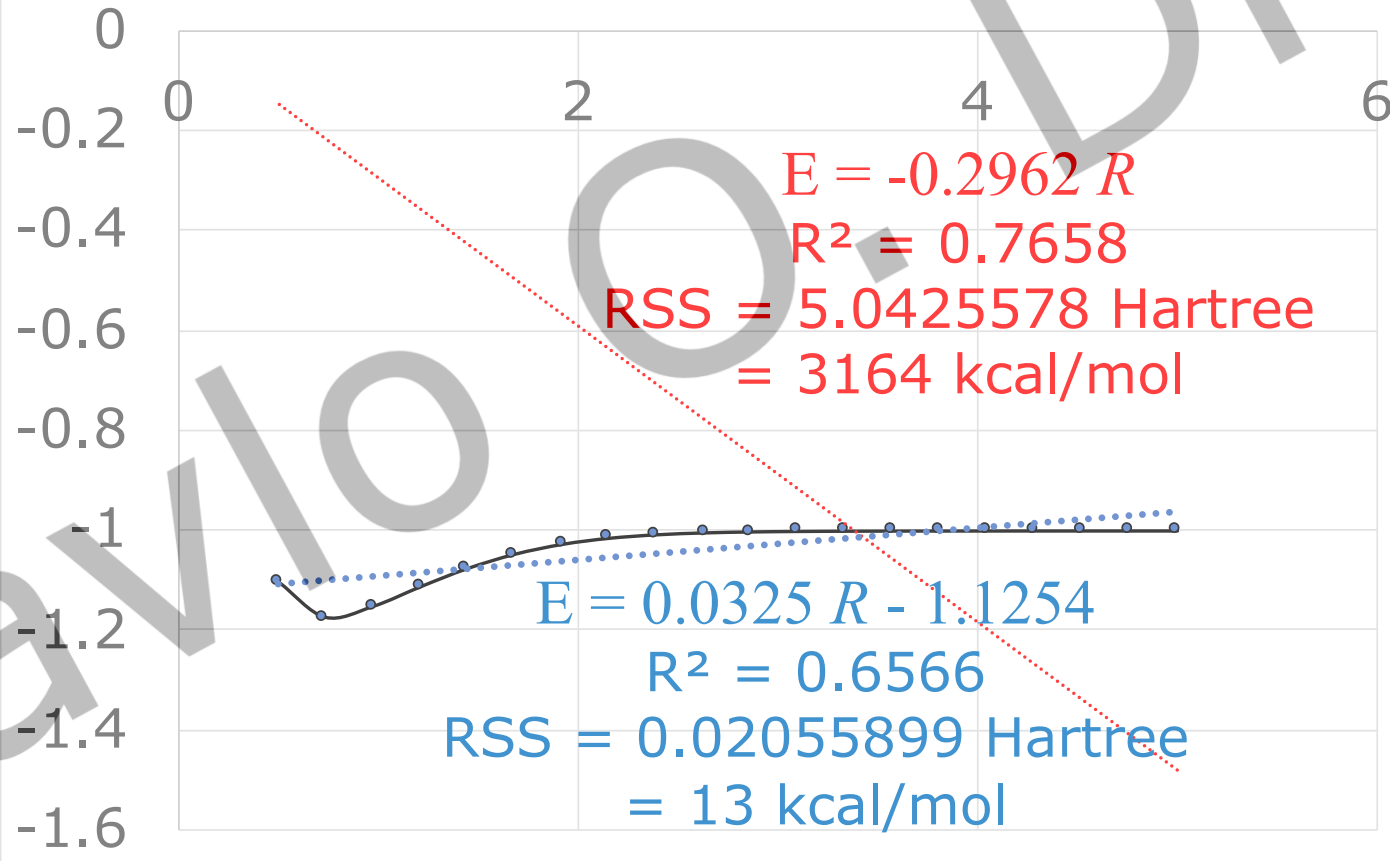
$$\text{RSS} = \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \boldsymbol{\beta}) - y_i)^2$$

Linear regression

Much better,
but still qualitatively wrong!

$$E = aR$$

$$E = aR + b$$



Q: What about linear regression with intercept b ?

$$E = aR + b$$

It is equivalent to mapping function $R \rightarrow (R, 1)$, $\Phi((R)) = (R, 1)$, where Φ maps from p -dimensional input space into p^d -dimensional **feature space**

Now we can solve multiple linear regression with two variables

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_1 x_{i1} + \beta_2 x_{i2} = \beta_1 R + \beta_2 1 = aR + b$$

$$x_{i1} = R_i$$

$$x_{i2} = 1$$

$$\beta_1 = a$$

$$\beta_2 = b$$

Any ideas how to get the dissociation curve shape right?

Pavlo O. Dral

Q: Any ideas how to get the dissociation curve shape right?

We can use mapping $R \rightarrow (R^{-6}, R^{-12}, 1)$ inspired by Lennard-Jones potential, this allows us to treat data set (E, R) nonlinear in input space (R) using vectors in feature space $(R^{-6}, R^{-12}, 1)$

Now we can solve multiple linear regression with three variables:

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} = \beta_1 R_i^{-6} + \beta_2 R_i^{-12} + \beta_3 1 = a R_i^{-6} + b R_i^{-12} + c$$

$$x_{i1} = R_i^{-6}$$

$$x_{i2} = R_i^{-12}$$

$$x_{i3} = 1$$

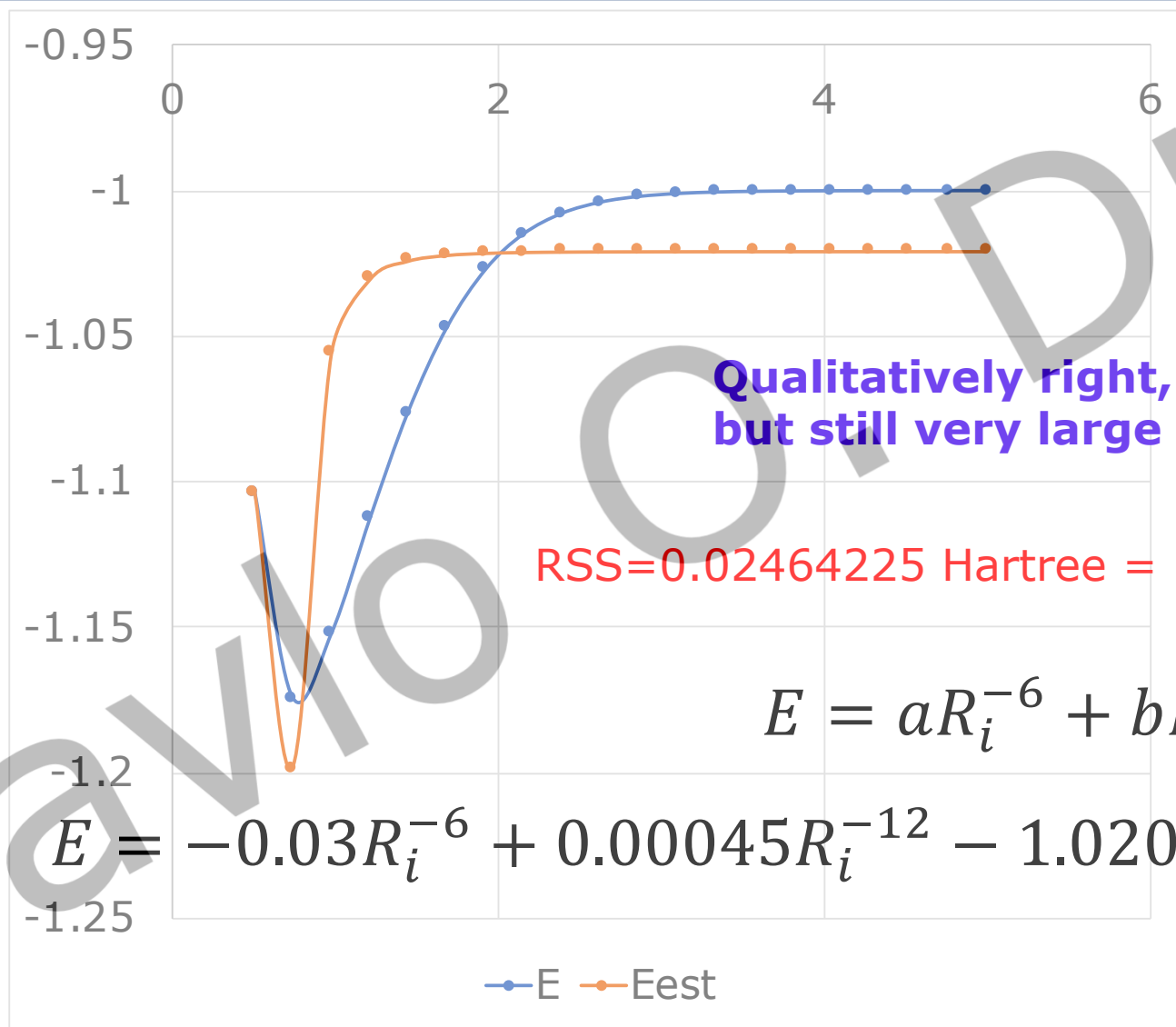
$$\beta_1 = a$$

$$\beta_2 = b$$

$$\beta_3 = c$$

Max Pinheiro Jr, P. O. Dral, Kernel methods.
In *Quantum Chemistry in the Age of Machine Learning*,
P. O. Dral, Ed. Elsevier: **2023**.
Paperback ISBN: 9780323900492

Linear regression



Can we extend it to more variables and make it more flexible?

Yes! We can go to infinite number of variables!

How?

Using a kernel trick

Pavlo O. Dral

Let's rewrite the linear regression equation by representing the regression coefficients via a sum over all training points:

$$f(\mathbf{x}'; \boldsymbol{\beta}) = \sum_{j=1}^p \beta_j x'_j$$

$$\beta_j = \sum_{i=1}^{N_{tr}} \alpha_i x_{ij}$$

$$f(\mathbf{x}') = \sum_{j=1}^p \left(\sum_{i=1}^{N_{tr}} \alpha_i x_{ij} \right) x'_j = \sum_{i=1}^{N_{tr}} \alpha_i \sum_{j=1}^p x_{ij} x'_j = \sum_{i=1}^{N_{tr}} \alpha_i \mathbf{x}_i^T \mathbf{x}'$$

$$\mathbf{x}_i^T \mathbf{x}' = \langle \mathbf{x}_i, \mathbf{x}' \rangle$$

Dot-product = inner product = scalar product
of two vectors

As we have seen before, we can map vectors \mathbf{x} and \mathbf{x}' from p -dimensional input space into p^d -dimensional feature space using mapping function Φ :

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}')$$

In previous examples we knew the mapping function and explicit forms of vectors in the feature space. But all we need is a dot-product between vectors in the feature space, not their explicit forms. Such dot-product is called **kernel** denoted $k(\mathbf{x}_i, \mathbf{x}')$ and it is calculated using vectors in the input space (not feature space!):

$$k(\mathbf{x}_i, \mathbf{x}') = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}')$$

The kernel trick is substitution of the calculation of the dot-product using explicit representations of vectors in the feature space by using a kernel function:

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

This is a kernel-based machine learning function.

Kernel trick allows us to use tools of linear regression for data nonlinear in the input space by converting variables into (higher dimensional) feature space.

Q: How to find the regression coefficients α ?

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

This is a kernel-based machine learning function.

Kernel trick allows us to use tools of linear regression for data nonlinear in the input space by converting variables into (higher dimensional) feature space.

We can find the coefficients α using the method of least squares, where coefficients are fit to get the minimum residual sum of squares (RSS) with respect to the training set with N_{tr} reference values \mathbf{y} :

$$\arg \min_{\alpha} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \alpha) - y_i)^2$$

$$\arg \min_{\alpha} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \alpha) - y_i)^2$$

$$L(\beta) = \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \alpha) - y_i)^2$$

$$f(\mathbf{x}_i) = \sum_{j=1}^{N_{tr}} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$L(\beta) = \sum_{i=1}^{N_{tr}} \left(\sum_{j=1}^{N_{tr}} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - y_i \right)^2$$

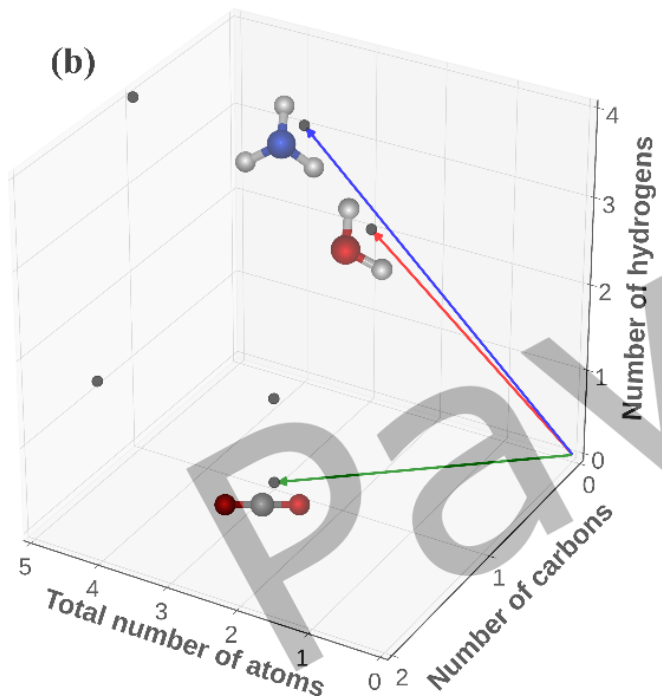
$$L(\beta) = (\mathbf{K}\alpha - \mathbf{y})^T (\mathbf{K}\alpha - \mathbf{y})$$

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_{N_{tr}}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_{N_{tr}}, \mathbf{x}_1) & \cdots & k(\mathbf{x}_{N_{tr}}, \mathbf{x}_{N_{tr}}) \end{pmatrix}$$

Kernel matrix

(a) Molecule num_atoms num_carbons num_hydrogens

Molecule	num_atoms	num_carbons	num_hydrogens
H2O	3	0	2
HCN	3	1	1
CO2	3	1	0
NH3	4	0	3
C2H2	4	2	2
CH4	5	1	4

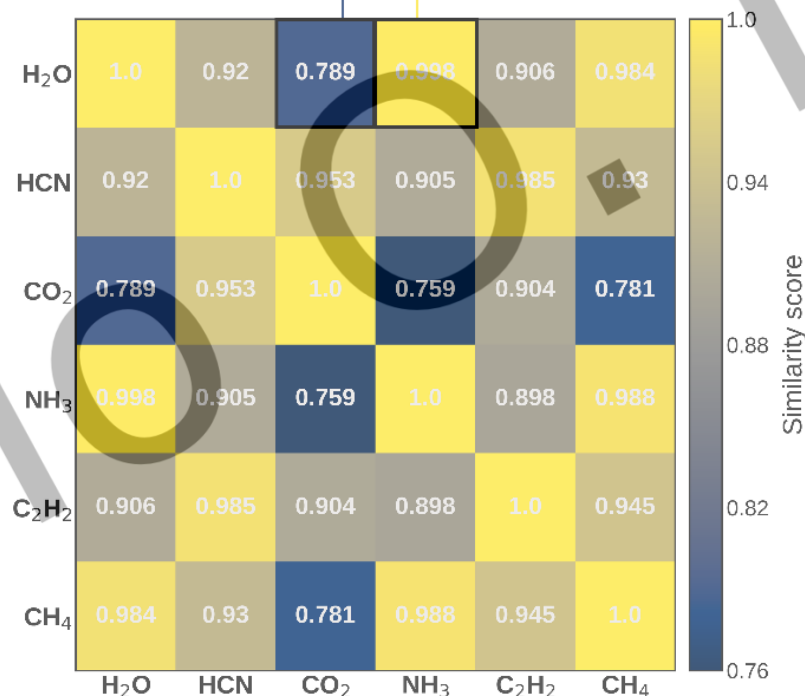


(c)

$$K_{ij} = \langle \text{vector}_i, \text{vector}_j \rangle$$

Kernel matrix
(matrix measuring similarities
between points – here cosine)

$$\cos \theta = \mathbf{a} \cdot \mathbf{b} / (\|\mathbf{a}\| \|\mathbf{b}\|)$$



$$\arg \min_{\alpha} \sum_{i=1}^{N} (f(\mathbf{x}_i; \alpha) - y_i)^2$$

$$L(\beta) = (\mathbf{K}\alpha - \mathbf{y})^T (\mathbf{K}\alpha - \mathbf{y})$$

$$\frac{\partial L(\alpha)}{\partial \alpha} = 2\mathbf{K}^T (\mathbf{K}\alpha - \mathbf{y}) = 2\mathbf{K}(\mathbf{K}\alpha - \mathbf{y}) = 2\mathbf{K}\mathbf{K}\alpha - 2\mathbf{K}\mathbf{y} = 0$$

$$\mathbf{K}\mathbf{K}\alpha = \mathbf{K}\mathbf{y}$$

$$\mathbf{K}^{-1}\mathbf{K}\mathbf{K}\alpha = \mathbf{K}^{-1}\mathbf{K}\mathbf{y}$$

$$\mathbf{K}\alpha = \mathbf{y}$$

$$\alpha = \mathbf{K}^{-1}\mathbf{y}$$

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

This is a kernel-based machine learning function.

If the kernel function is itself a dot-product:

$$k(\mathbf{x}_i, \mathbf{x}') = \mathbf{x}_i^T \mathbf{x}'$$

The expression becomes equivalent to the linear regression as we have seen above and that is why such a dot-product kernel is also called "linear kernel":

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i \mathbf{x}_i^T \mathbf{x}' = \sum_{j=1}^p \beta_j x'_j$$

$$\beta_j = \sum_{i=1}^{N_{tr}} \alpha_i x_{ij}$$

$$\beta_j = \sum_{i=1}^{N_{tr}} \alpha_i x_{ij}$$

One can get the same β_j for infinite combinations of α_i

Some solutions will have very large α_i with opposite signs trying to compensate each other

$$\beta = \sum_{i=1}^{N_{tr}=20} \alpha_i x_i = \sum_{i=1}^{N_{tr}=20} \alpha_i R_i$$

$$\beta = (-0.2962 \cdot 2) \cdot 0.5 + \sum_{i=2}^{N_{tr}=20} 0x_i$$

$$\beta = 198476910439 \cdot 0.5 - 19847691043.95924 \cdot 5 + \sum_{i=3}^{N_{tr}=20} 0x_i$$

$$f(x; \beta) = \beta x = -0.2962x$$

$$\alpha = K^{-1}y$$

**Prone to overfitting,
numerically unstable
Often, K is not invertible
matrix**

Ridge regression – belongs to shrinkage methods (useful for feature importance analysis)

$$\arg \min_{\beta} (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y}) + \lambda \beta^T \beta$$

$$\beta = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Another example of shrinkage method is the lasso

$$\arg \min_{\beta} (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y}) + \lambda \sum_{i=1}^p |\beta_i|$$

Identity matrix

$$\mathbf{I} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

Kernel ridge regression (KRR)

$$\arg \min_{\alpha} (\mathbf{K}\alpha - \mathbf{y})^T (\mathbf{K}\alpha - \mathbf{y}) + \lambda \alpha^T \mathbf{K}\alpha$$

$$\alpha = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Coefficient magnitude is forced to shrink with larger λ in these methods
 λ is nonnegative regularization hyperparameter, smoothens function and makes solution numerically more stable.

$$f(\mathbf{x}') = \sum_{i=1}^{N_{\text{tr}}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

This is a kernel-based machine learning function.

One of the popular kernel functions is the Gaussian kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma^2} \sum_s^{N_x} (x_{i,s} - x_{j,s})^2\right)$$

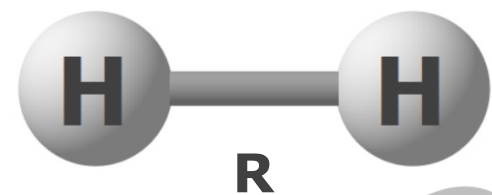
It maps vectors \mathbf{x} from N_x -dimensional input space into infinite-dimensional feature space.

σ is a positive hyperparameter defining the length scale of the Gaussian function.

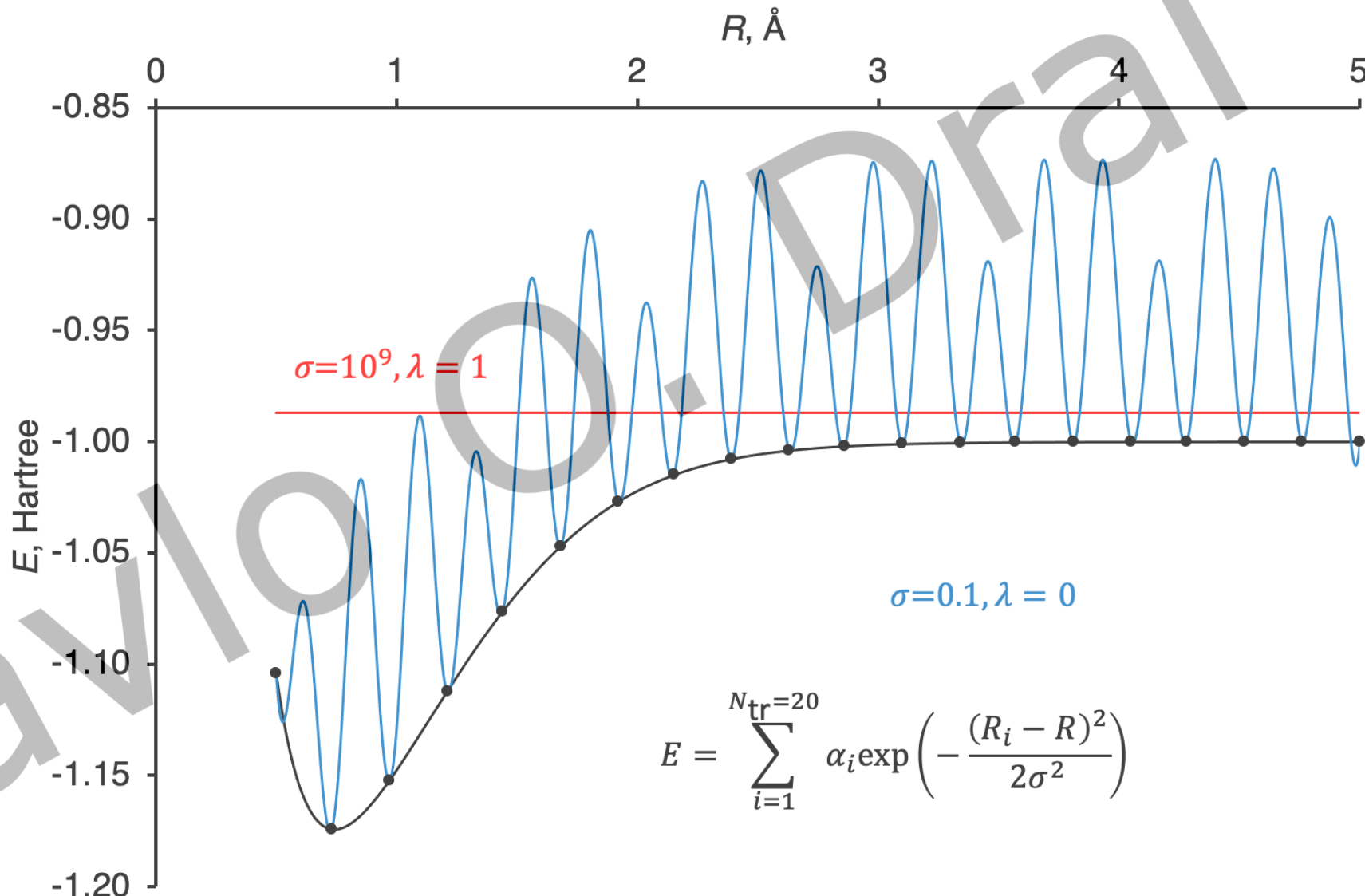
Many ML algorithms can give you practically ideal predictions for their own training set

Underfitting
Low variance
High bias

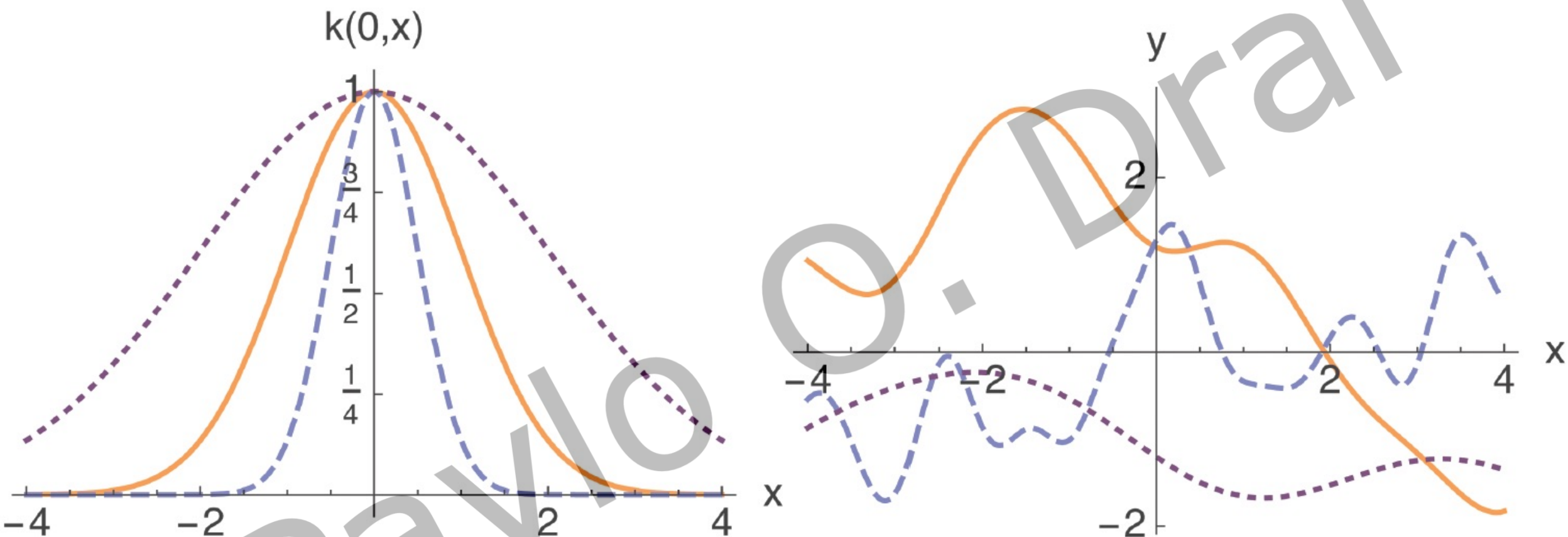
H₂ dissociation curve



Overfitting
High variance
Low bias



$$E = \sum_{i=1}^{N_{\text{tr}}=20} \alpha_i \exp\left(-\frac{(R_i - R)^2}{2\sigma^2}\right)$$



(b) Gaussian kernel with $\sigma = 0.5, 1, 2$ (dashed, solid, dotted lines).

Take KRR with Gaussian kernel

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i \exp\left(-\frac{1}{2\sigma^2} \sum_s^{N_x} (x_{i,s} - x'_s)^2\right)$$

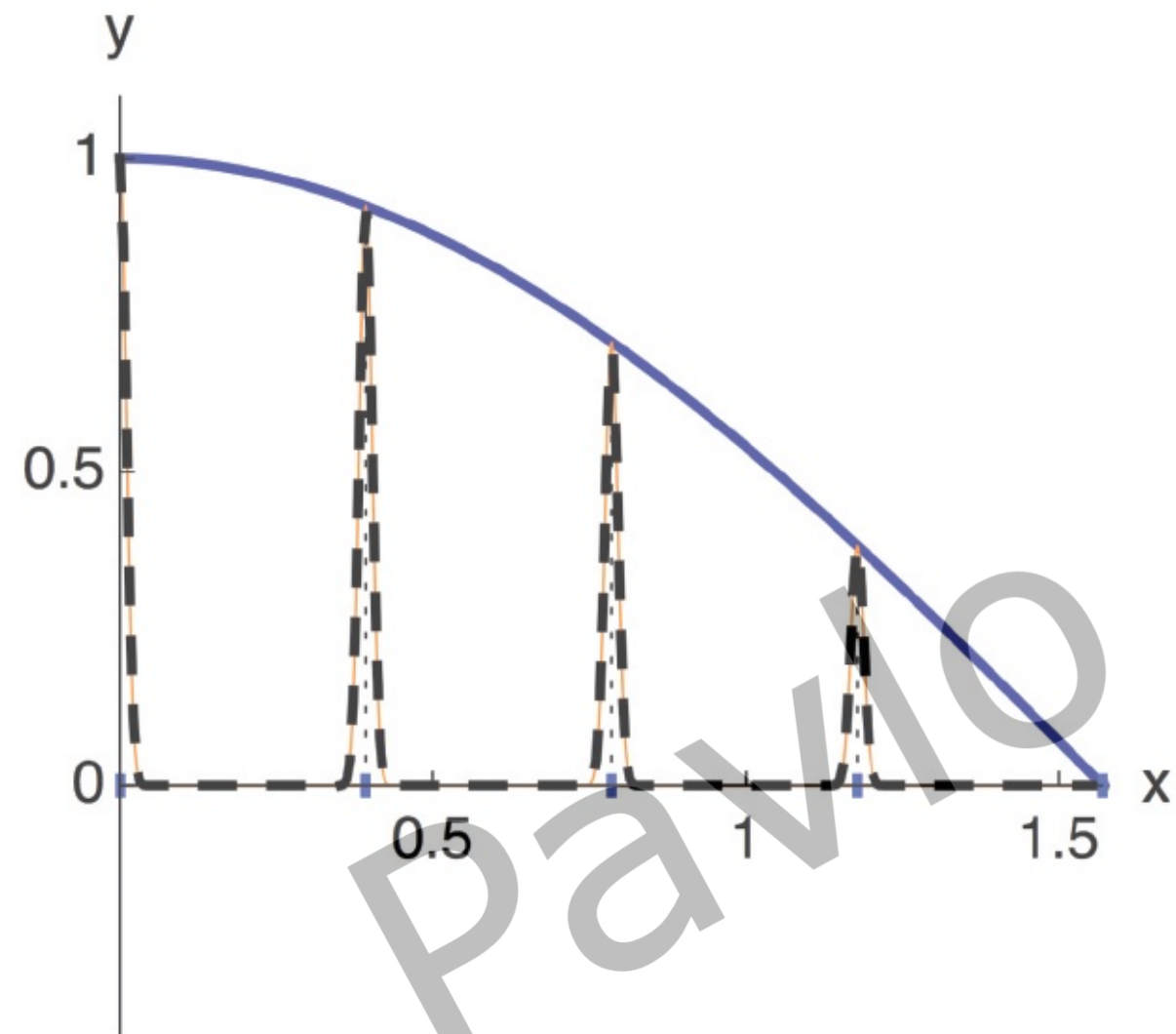
and consider what happens for very small $\sigma \rightarrow 0$:

$$f(\mathbf{x}') = \begin{cases} \alpha_i, & \text{for } \mathbf{x}' = \mathbf{x}_i \\ 0, & \text{for } \mathbf{x}' \neq \mathbf{x}_i \end{cases}$$

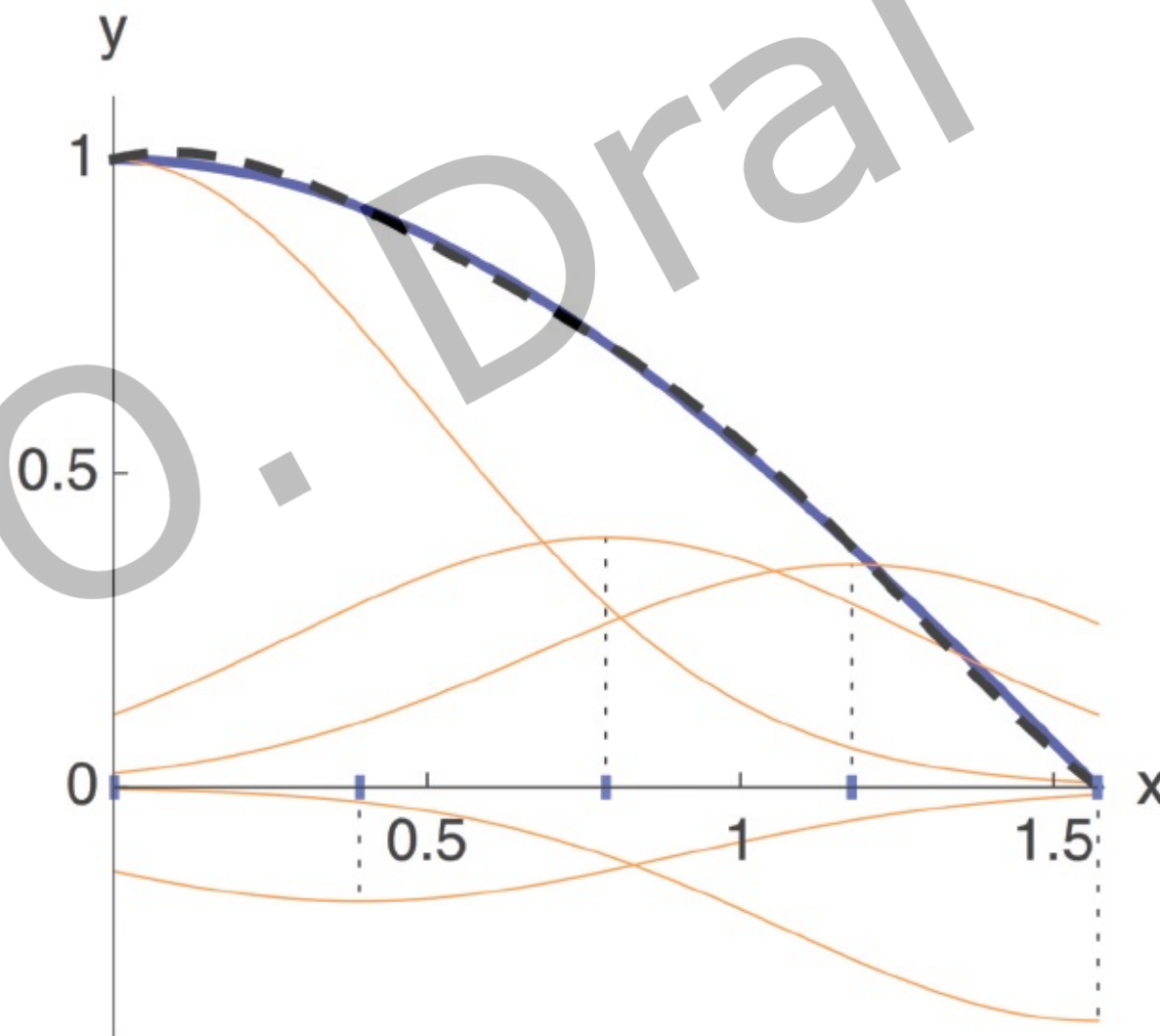
and consider what happens for very large $\sigma \rightarrow \infty$:

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i = \text{const}$$

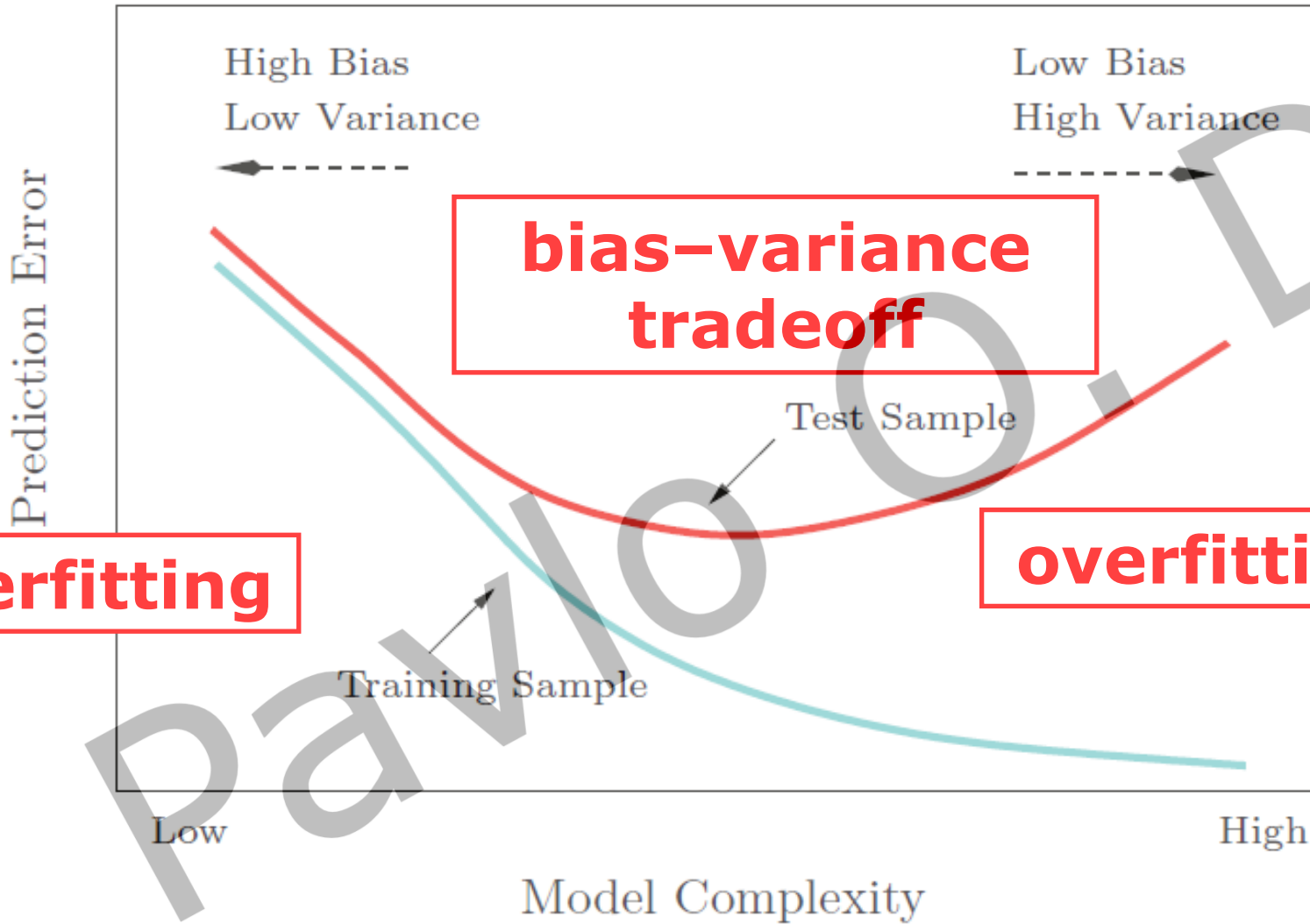
KRR with Gaussian kernel



(a) Overfitting ($\sigma = 0.01$)



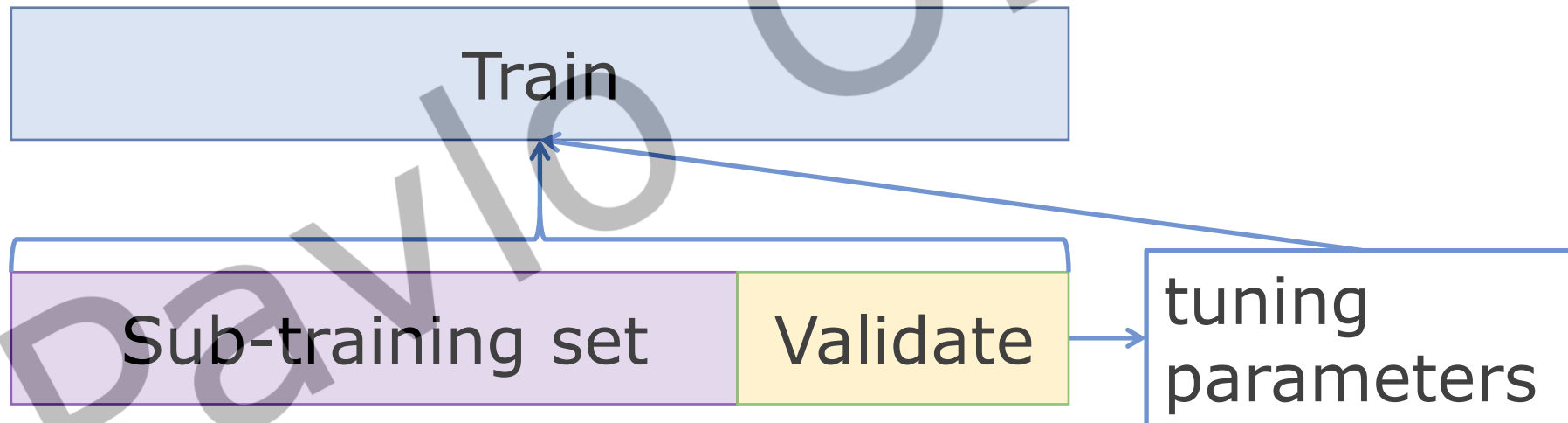
(b) Fitting ($\sigma = 0.5$)



Q: How to choose hyperparameters?

Pavlo O. Dral

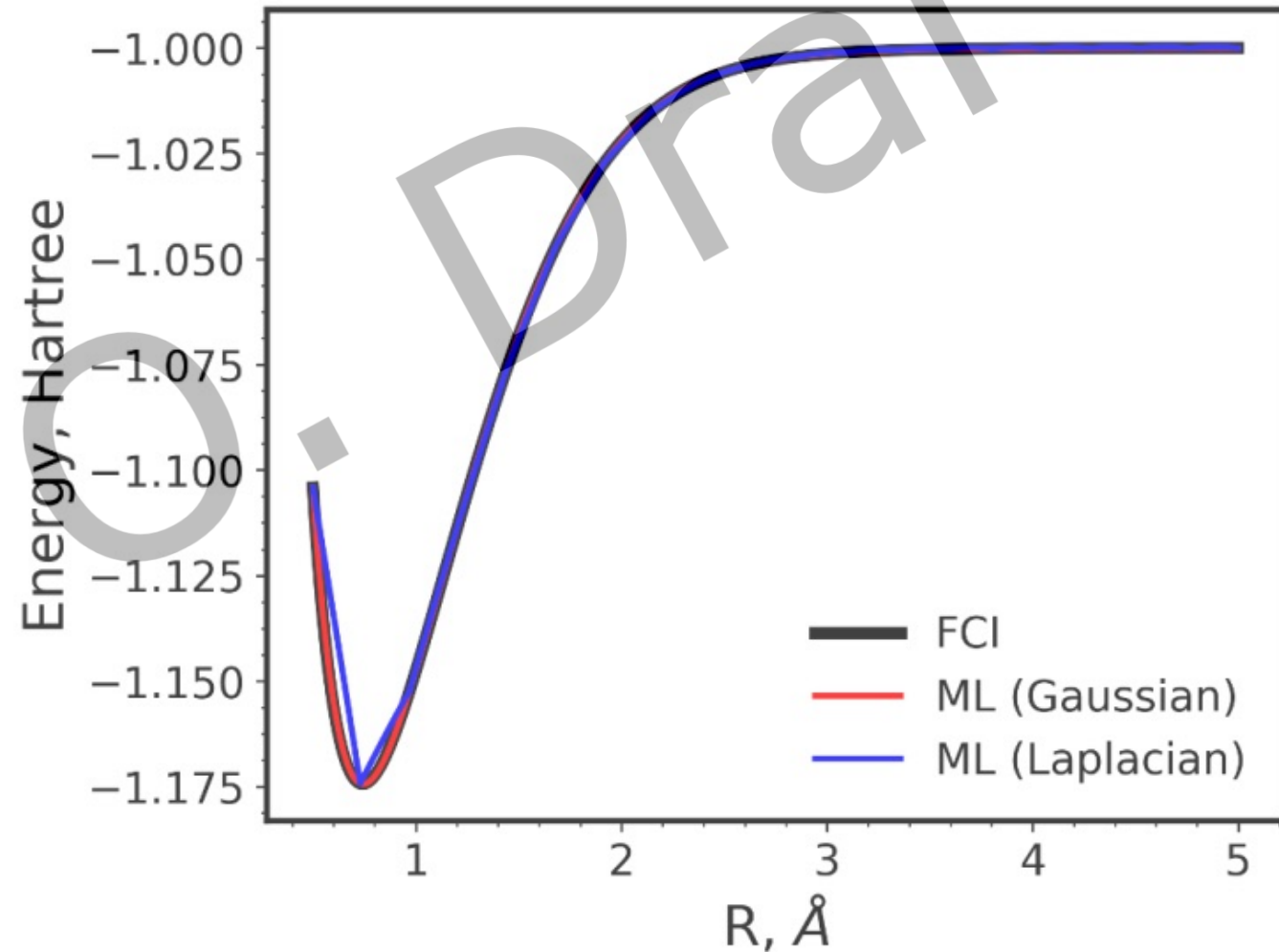
We target minimal error **not** in the training set, but in the validation set for models trained on the sub-training set.



H₂ dissociation curve

Full CI calculations:
more than 30 min
for one value of R.

ML trained on 20 points needs
less than 1 sec.
for hundreds of other points



5-fold Cross-validation

Training set with randomly shuffled items

Pavlo O. Dral

5-fold Cross-validation

Training set with randomly shuffled items

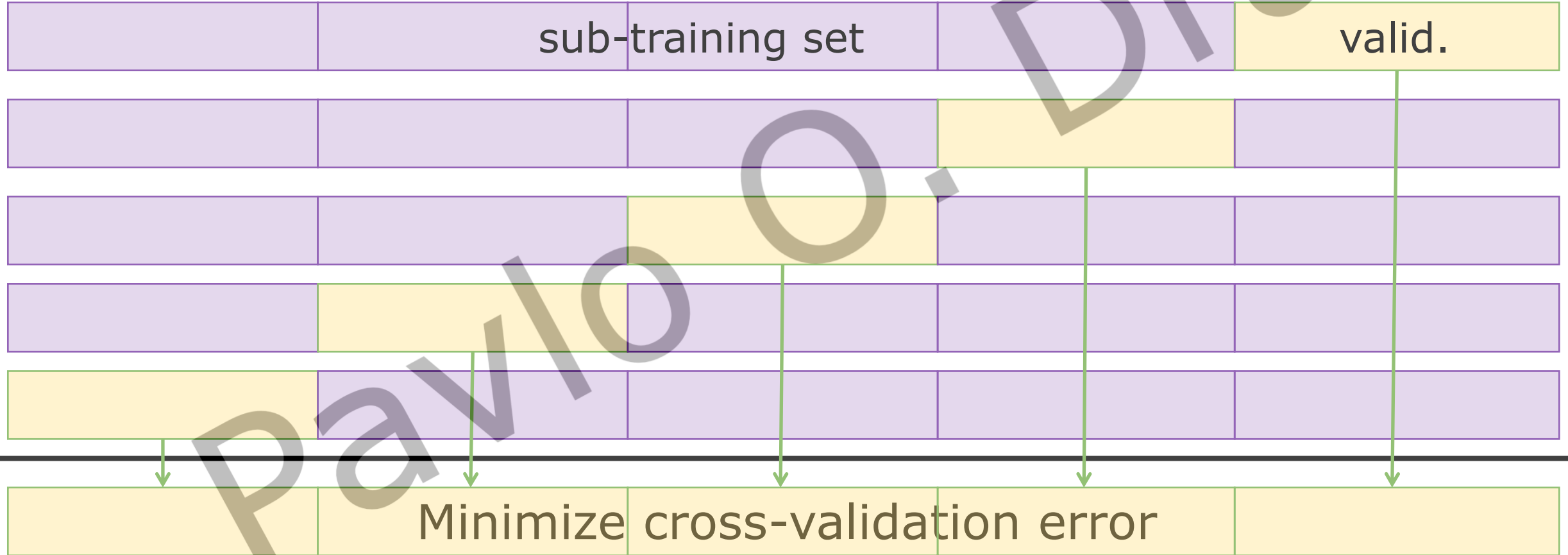
sub-training set

valid.

Pavlo O. Dral

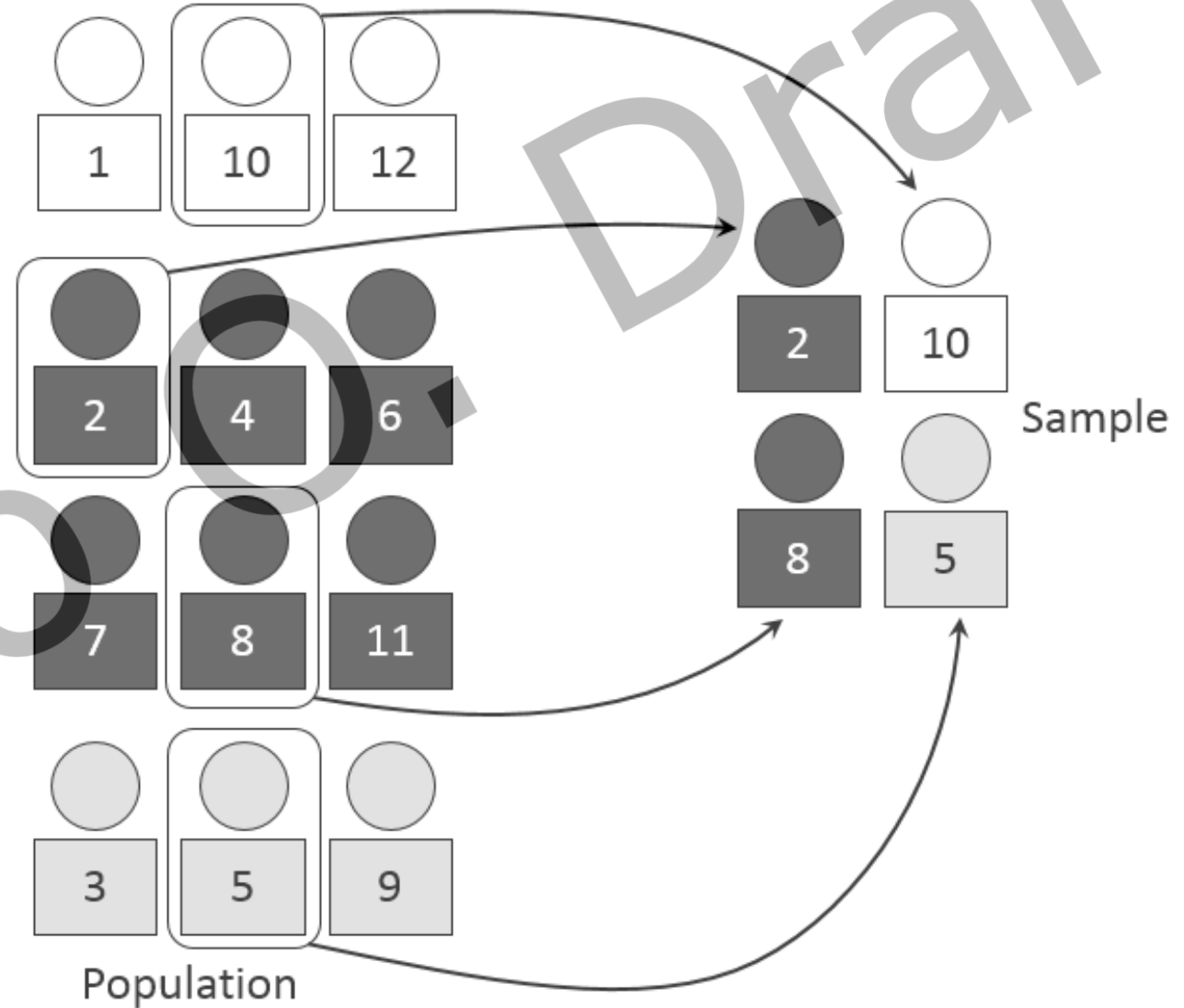
5-fold Cross-validation

Training set with randomly shuffled items



Random sampling for model selection is not always a good idea

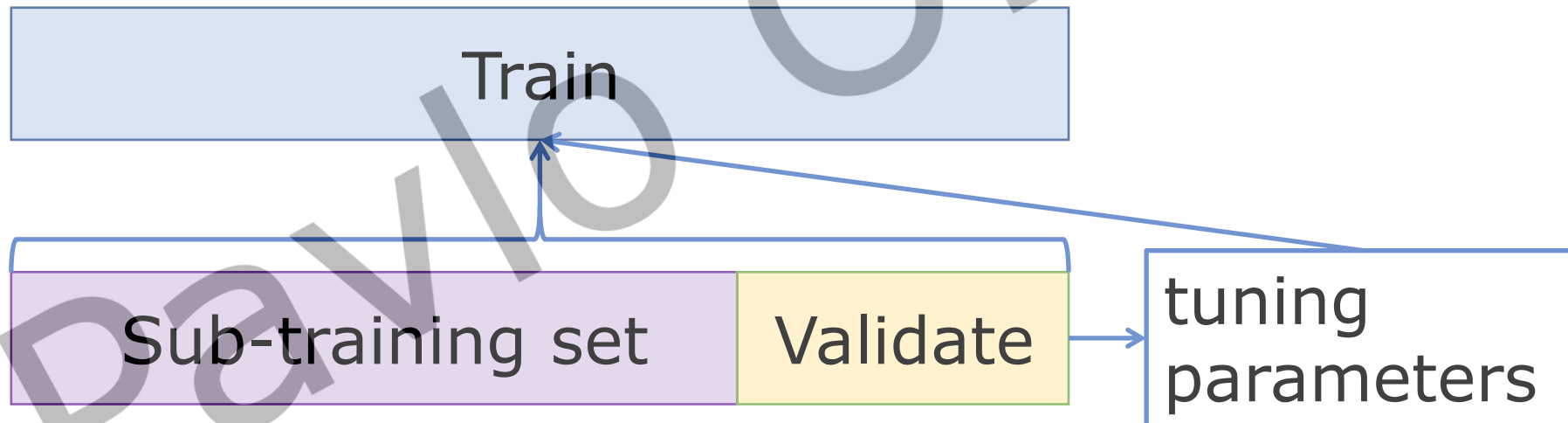
Sometimes, **stratification** is preferable



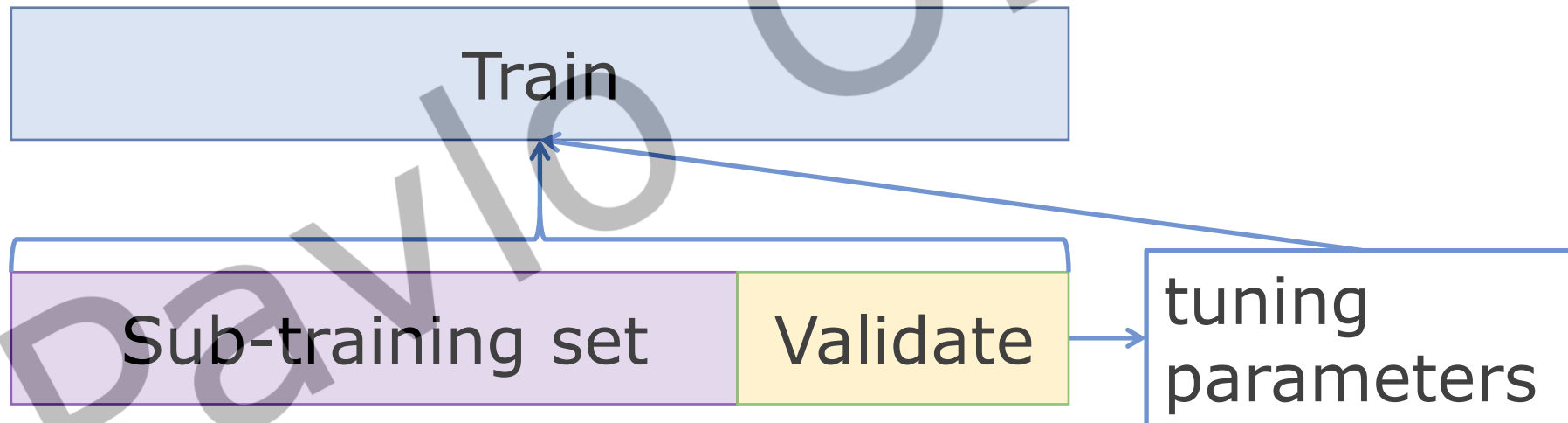
Model Evaluation
(estimation of the generalization error)

Pavlo O. Dral

- Often ML error for its own training set is close to zero

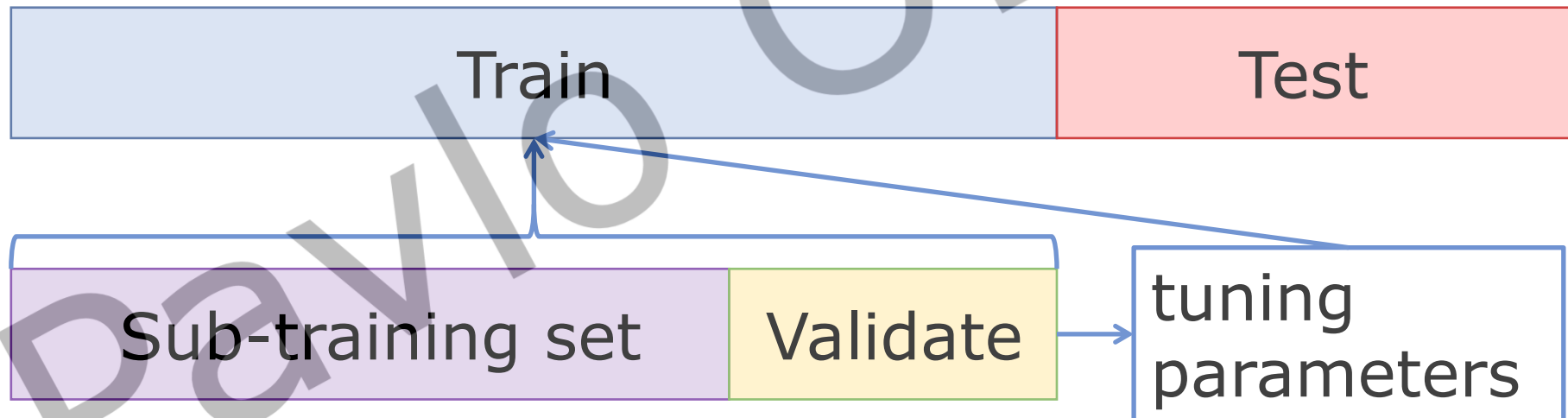


- Often ML error for its own training set is close to zero
- Using errors in the validation set would be also incorrect, because their minimization is a part of the training process



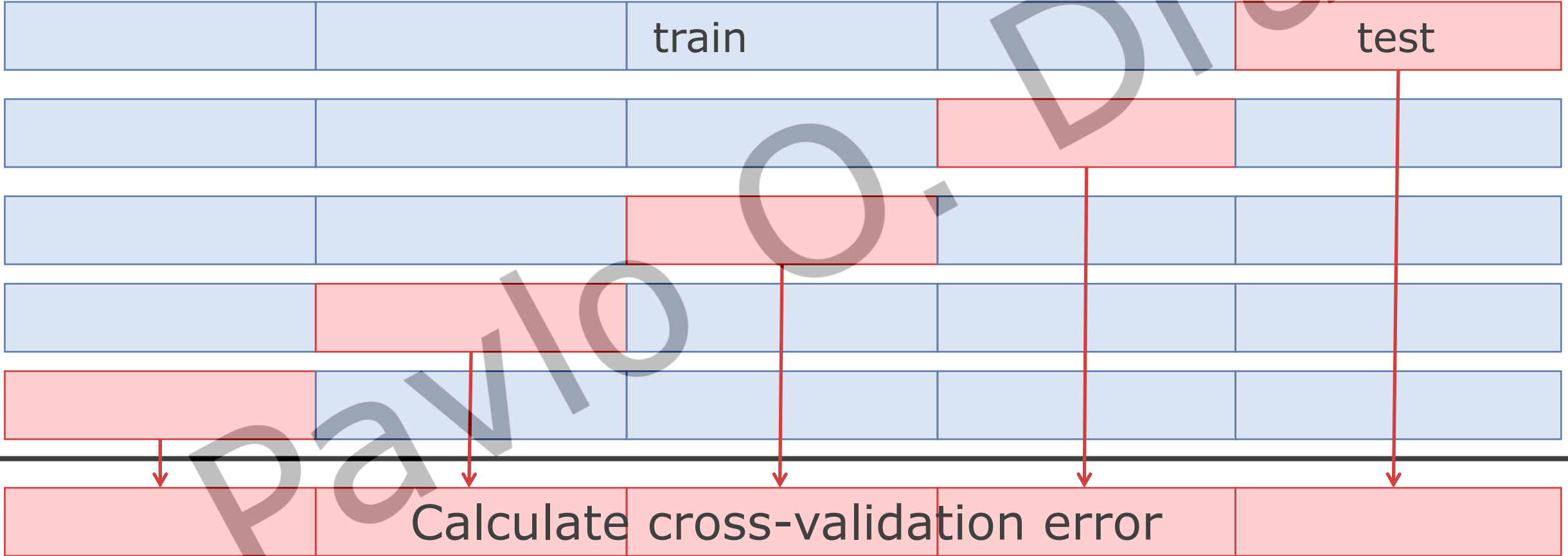
ML: Error Estimation

- Often ML error for its own training set is close to zero
- Using errors in the validation set would be also incorrect, because their minimization is a part of the training process
- We should estimate errors on a **completely independent test set**



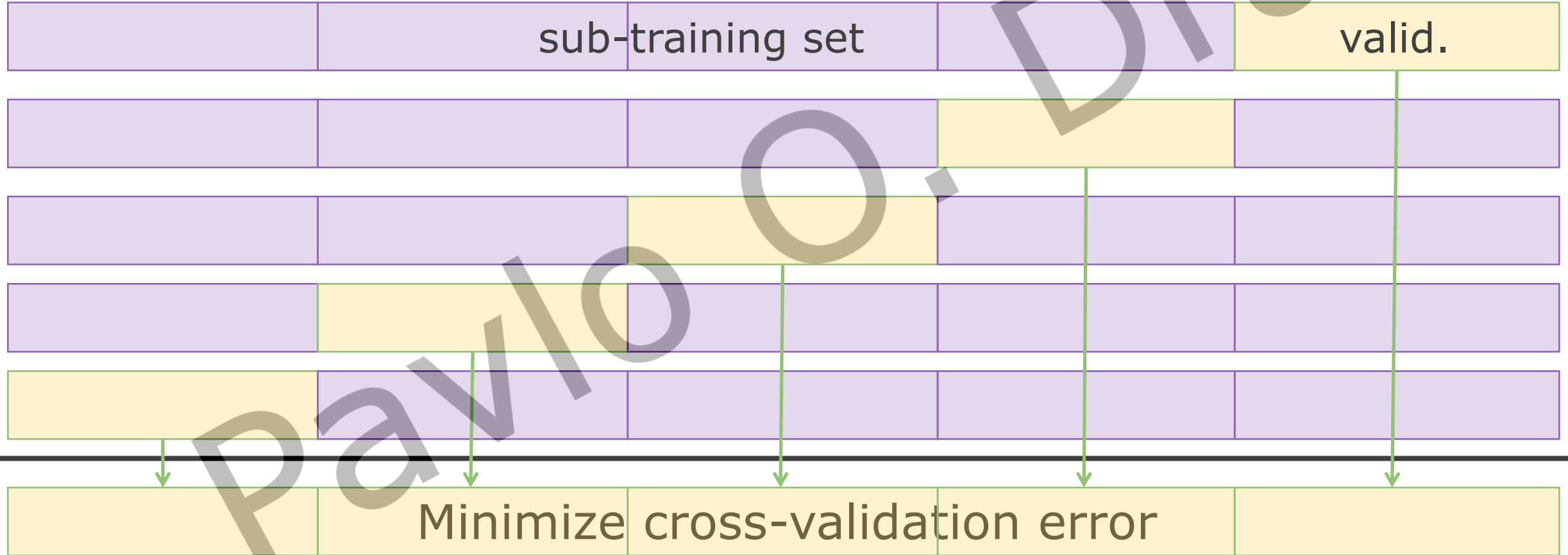
5-fold Cross-validation

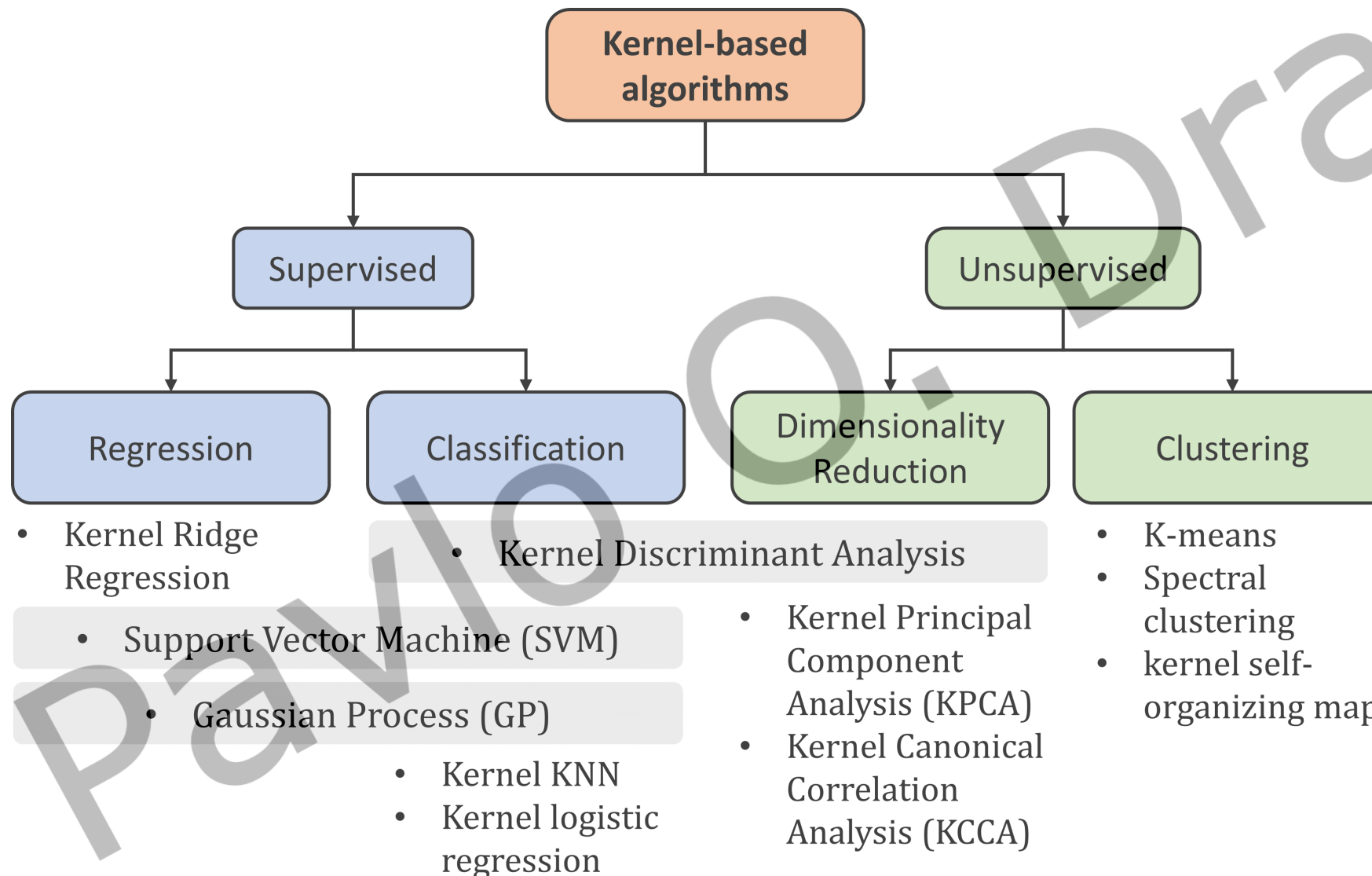
Entire set with randomly shuffled original data



5-fold Cross-validation

Training set with randomly shuffled items





- Kernel ridge regression (KRR)

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

- Gaussian processes (GP, kriging)

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

- Support vector machines (SVM)

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}'), \quad 0 < \alpha_i < C$$

Prediction
functions are
the same for
KRR and GP!

Pavlio Dral

- Kernel ridge regression gives the same prediction as Gaussian processes:

$$f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

- Gaussian processes also provide:
 - variance V

$$V(\mathbf{x}') = k(\mathbf{x}', \mathbf{x}') - \mathbf{k}'^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}'$$

$$\mathbf{k}' = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}') \\ \vdots \\ k(\mathbf{x}_{N_{tr}}, \mathbf{x}') \end{pmatrix}$$

- Marginal likelihood:

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2} \mathbf{y}^T \boldsymbol{\alpha} - \frac{1}{2} \log |\mathbf{K} + \lambda \mathbf{I}| - \frac{N_{tr}}{2} \log 2\pi$$

Hyperparameters in kernel function can be found by optimizing log marginal likelihood, for which derivatives are taken, e.g. $\frac{\partial \log p(\mathbf{y}|\mathbf{X}, \sigma)}{\partial \sigma}$

Advantages of kernel methods:

- Nonparametric models, i.e., do not assume a specific behavior of data (compare to parametric model such as linear regression)
- Explicitly incorporate training data, thus very flexible and accurate
- Closed (analytical) solution, i.e. fast training

$$\alpha = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \quad f(\mathbf{x}') = \sum_{i=1}^{N_{tr}} \alpha_i k(\mathbf{x}_i, \mathbf{x}')$$

Disadvantages:

- Slow training for lots of training data (scales as $O(N_{tr}^3)$)
- Requires lots of RAM to store the kernel matrix (scales as $O(N_{tr}^2)$)
- Prediction time slows down with more training data (scales as $O(N_{tr}^1)$)

Pavlio O. Dral

Table 1 Required memory (RAM) for storing kernel matrix built with increasing number of training points.

Training set size	RAM size
$100 = 10^2$	78 kB
$1000 = 10^3$	7.6 MB
$10,000 = 10^4$	0.75 GB
$50,000 = 5 \times 10^4$	19 GB
$100,000 = 10^5$	75 GB
$500,000 = 5 \times 10^5$	1.8 TB
$1000,000 = 10^6$	7.3 TB

Table 2 CPU time needed for calculating regression coefficients for increasing number of training points, assuming that it takes 10 s for 10,000 training points.

Training set size	Time
$100 = 10^2$	0.01 milliseconds
$1000 = 10^3$	0.01 s
$10,000 = 10^4$	10 s
$50,000 = 5 \times 10^4$	21 min
$100,000 = 10^5$	2.8 h
$500,000 = 5 \times 10^5$	15 days
$1000,000 = 10^6$	3.9 months

Solutions:

- Reduce the training set by selecting the most relevant points[1,2]
- Sparsification techniques[3]
- Construct high-dimensional kernels as products of one-dimensional kernels[4]

See, for example:

[1] Dral, Owens, Yurchenko, Thiel, *J. Chem. Phys.* **2017**, 146, 244108

[2] Hu, Xie, Li, Li, Lan, *J. Phys. Chem. Lett.* **2018**, 9, 2725

[3] Bartók, Csányi, *Int. J. Quantum Chem.* **2015**, 115, 1051

[4] Unke, Meuwly, *J. Chem. Inf. Model.* **2017**, 57, 1923

Neural networks

Pavlo O. Dral

Linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

Kernel ridge regression (KRR)

Neural networks (NN)

Linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

Kernel ridge regression (KRR)

Neural networks (NN)

Linear regression

$$\hat{y} = f(\mathbf{x}; \mathbf{w}, b) = b + w_1x_1 + w_2x_2 + \cdots + w_px_p = \mathbf{x}^T \mathbf{w} + b$$



Neural networks (NNs): the single hidden layer, feed-forward network

$$\hat{y} = f(\mathbf{x}; \boldsymbol{\alpha}, \mathbf{a}, \mathbf{w}, b) = b + w_1h_1(\mathbf{x}; \boldsymbol{\alpha}_1, a_1) + \cdots + w_Mh_M(\mathbf{x}; \boldsymbol{\alpha}_M, a_M) = \mathbf{h}^T \mathbf{w} + b$$

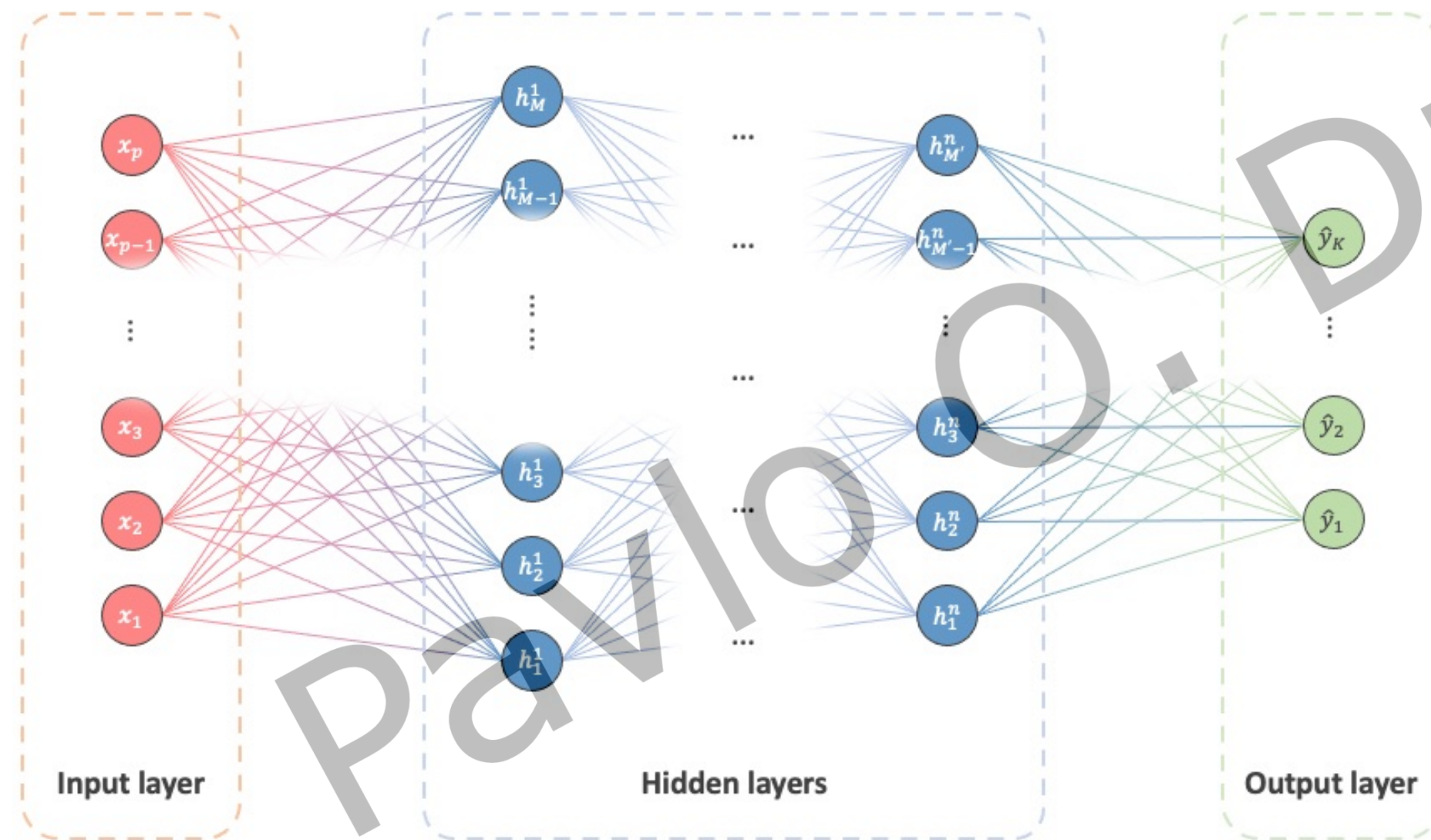
$$h_m(\mathbf{x}; \boldsymbol{\alpha}_m, a_m) = g(\mathbf{a}_m + \alpha_{m1}x_1 + \alpha_{m2}x_2 + \cdots + \alpha_{mp}x_p) = g(\mathbf{x}^T \boldsymbol{\alpha}_m + a_m)$$

$$\hat{y} = f(\mathbf{x}; \boldsymbol{\alpha}, \mathbf{a}, \mathbf{w}, b) = f^{(2)}(\mathbf{h}; \mathbf{w}, b) = f^{(2)}(f^{(1)}(\mathbf{x}))$$

Activation functions:

$$g(v) = \exp(-a(v - c)^2) \quad \text{radial basis function (RBF)}$$

Neural network (NN)



NN consists of

- layers
- nodes=units=neurons

The single hidden layer,
feed-forward NN

$w, \alpha \dots$ weights

$a, b \dots$ biases

$$f(\mathbf{x}; \boldsymbol{\alpha}, \mathbf{a}, \mathbf{w}, b) = f^{(2)}(\mathbf{h}; \mathbf{w}, b)$$

Linear regression

$$\hat{y} = f(\mathbf{x}; \mathbf{w}, b) = b + w_1x_1 + w_2x_2 + \dots + w_px_p = \mathbf{x}^T \mathbf{w} + b$$



Neural networks (NNs): the single hidden layer, feed-forward network

$$\hat{y} = f(\mathbf{x}; \boldsymbol{\alpha}, \mathbf{a}, \mathbf{w}, b) = b + w_1h_1(\mathbf{x}; \boldsymbol{\alpha}_1, a_1) + \dots + w_Mh_M(\mathbf{x}; \boldsymbol{\alpha}_M, a_M) = \mathbf{h}^T \mathbf{w} + b$$

$$h_m(\mathbf{x}; \boldsymbol{\alpha}_m, a_m) = g(a_m + \alpha_{m1}x_1 + \alpha_{m2}x_2 + \dots + \alpha_{mp}x_p) = g(\mathbf{x}^T \boldsymbol{\alpha}_m + a_m)$$

g is the activation function. If:

- g is the identity function, NN is equivalent to linear regression $g(v) = v$

$$g(a_m + \alpha_{m1}x_1 + \alpha_{m2}x_2 + \dots + \alpha_{mp}x_p) = a_m + \alpha_{m1}x_1 + \alpha_{m2}x_2 + \dots + \alpha_{mp}x_p$$

- Typically, g is used for the nonlinear transformation making the NN flexible

$$g(v) = \exp(-a(v - c)^2) \quad \text{radial basis function (RBF)}$$

Table 1. Overview of a selection of popular activation functions.

Names	Equation
linear function	$g(v) = v$
identity function[2]	
rectified linear unit (ReLU) [2]	$g(v) = \max(0, v)$
exponential linear unit (ELU)[5]	$g(v) = \begin{cases} v & \text{if } v \geq 0 \\ a(\exp(v) - 1) & \text{otherwise} \end{cases}$ <p>where a is a parameter</p>
continuously differentiable exponential linear unit (CELU)[6]	$g(v) = \begin{cases} v & \text{if } v \geq 0 \\ a \left(\exp\left(\frac{v}{a}\right) - 1 \right) & \text{otherwise} \end{cases}$ <p>where a is a parameter</p>
	$g(v) = v \cdot \frac{1}{2} \left[1 + \operatorname{erf}\left(\frac{v}{\sqrt{2}}\right) \right]$ <p>faster approximated versions:</p>
Gaussian error linear unit (GELU)[7]	$g(v) = 0.5v \left(1 + \tanh \left[\sqrt{\frac{2}{\pi}} (v + 0.044715v^3) \right] \right)$ $g(v) = v \cdot \sigma(1.702v) = v \cdot \frac{1}{1 + \exp(-1.702v)}$

P. O. Dral, A. Kananenka, F. Ge, B.-X. Xue, Neural Networks. In *Quantum Chemistry in the Age of Machine Learning*, 1st ed.; P. O. Dral, Ed. Elsevier: 2023.

radial basis function

(RBF)[1-2]

$$g(v) = \exp(-a(v - c)^2)$$

where a and c are parameters

logistic sigmoid

function[1-2]

$$g(v) = \sigma(v) = \frac{1}{1 + \exp(-v)}$$

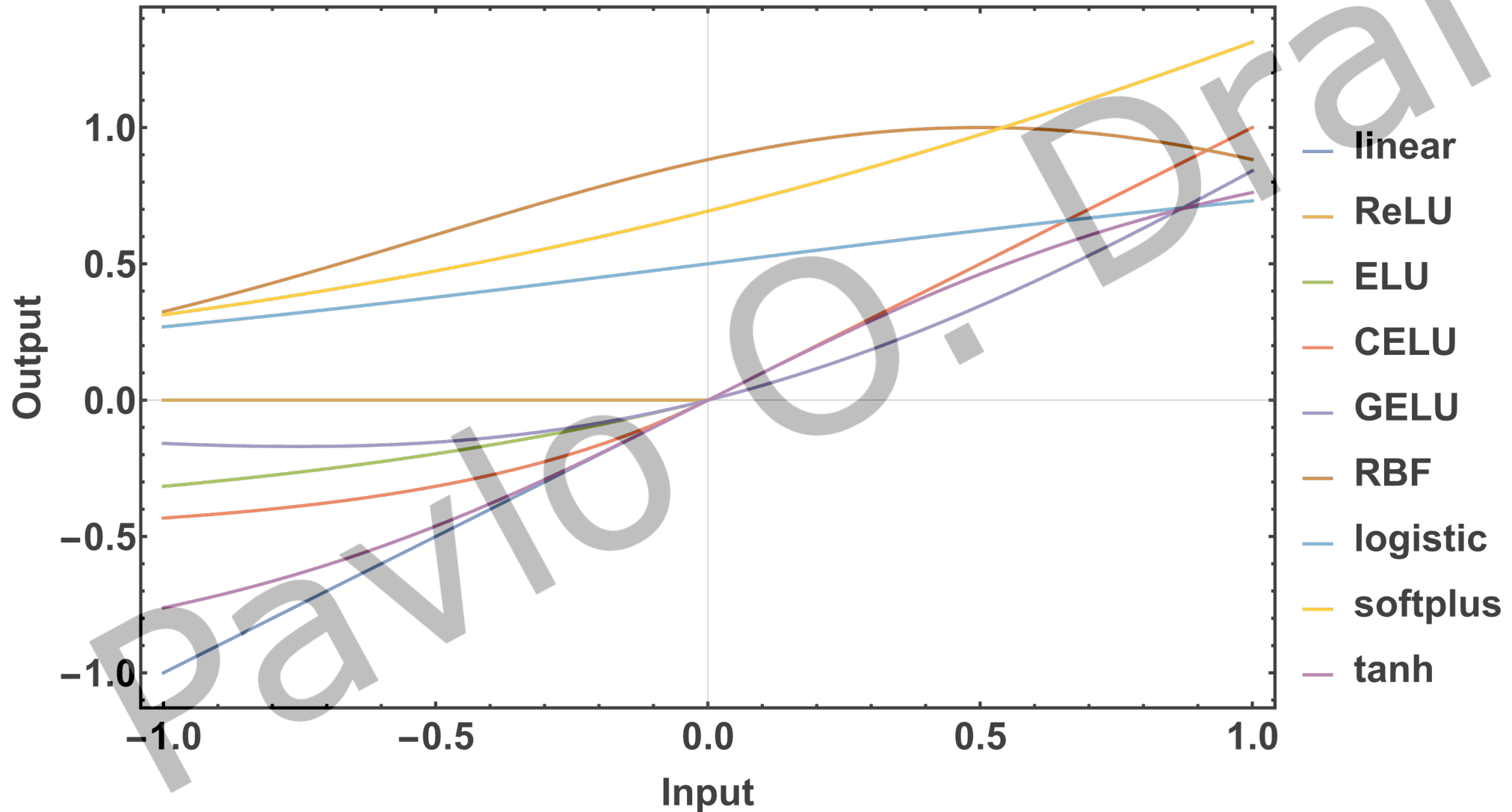
softplus function[2]

$$g(v) = \log(1 + \exp(v))$$

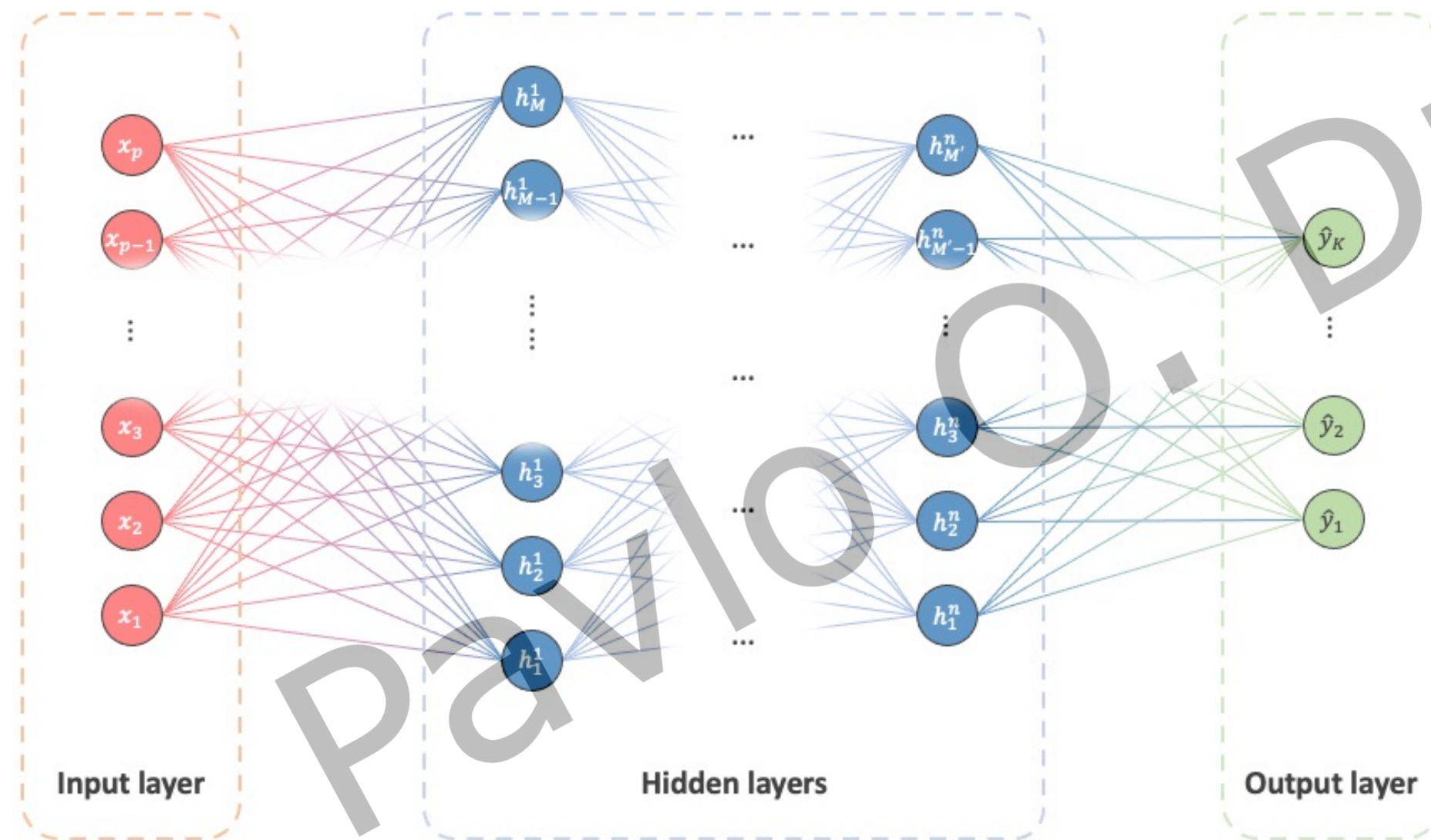
hyperbolic tangent

function[2]

$$g(v) = T(v) = \tanh(v)$$



Neural network (NN)



NN consists of

- layers
- nodes=units=neurons

The single hidden layer,
feed-forward NN

$w, \alpha \dots$ weights

$a, b \dots$ biases

$$f(\mathbf{x}; \boldsymbol{\alpha}, \mathbf{a}, \mathbf{w}, b) = f^{(2)}(\mathbf{h}; \mathbf{w}, b)$$

To train NN means to find its weights θ usually by solving this minimization task:

$$\arg \min_{\theta} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \theta) - y_i)^2$$

To avoid overfitting this solution can be regularized using weight decay approach (recall ridge regression and KRR):

$$\arg \min_{\theta} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \theta) - y_i)^2 + \lambda \sum_{j=1}^{N_p} \theta_j^2$$

$$\theta = \mathbf{w}, \alpha, \mathbf{a}, b$$

parameters

$$\hat{y} = f(\mathbf{x}; \alpha, \mathbf{a}, \mathbf{w}, b) = b + w_1 h_1(\mathbf{x}; \alpha_1, a_1) + \dots + w_M h_M(\mathbf{x}; \alpha_M, a_M) = \mathbf{h}^T \mathbf{w} + b$$

$$h_m(\mathbf{x}; \alpha_m, a_m) = g(a_m + \alpha_{m1}x_1 + \alpha_{m2}x_2 + \dots + \alpha_{mp}x_p) = g(\mathbf{x}^T \alpha_m + a_m)$$

To train NN means to find its weights θ usually by solving this minimization task:

$$\arg \min_{\theta} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \theta) - y_i)^2$$

To avoid overfitting this solution can be regularized using weight decay approach (recall ridge regression and KRR):

$$\arg \min_{\theta} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \theta) - y_i)^2 + \lambda \sum_{j=1}^{N_p} \theta_j^2$$

$$\theta = \mathbf{w}, \alpha, \mathbf{a}, b$$

parameters

$$\hat{y} = f(\mathbf{x}; \alpha, \mathbf{a}, \mathbf{w}, b) = b + w_1 h_1(\mathbf{x}; \alpha_1, a_1) + \dots + w_M h_M(\mathbf{x}; \alpha_M, a_M) = \mathbf{h}^T \mathbf{w} + b$$

$$h_m(\mathbf{x}; \alpha_m, a_m) = g(a_m + \alpha_{m1}x_1 + \alpha_{m2}x_2 + \dots + \alpha_{mp}x_p) = g(\mathbf{x}^T \alpha_m + a_m)$$

In contrast to linear regression and kernel methods, closed solution is unknown

Issues with NNs:

In contrast to linear regression and kernel methods, no closed solution exists

Solutions are unstable and difficult to find.

Computationally expensive optimization problem should be solved and it therefore often can be speed up by using GPUs instead of CPUs.

GPUs are however much more expensive and difficult to get and optimization is still quite slow.

One of the popular approaches for fitting is **back-propagation**.

Back-propagation:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^N (f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i)^2$$

gradient descent update with learning rate γ

$$\theta_k^{(r+1)} = \theta_k^{(r)} - \gamma \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_k}$$

Well parallelized:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^N L_i = \sum_{i=1}^N (f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i)^2$$

The training set is often split into the minibatches (**batches**)

Update of parameters after the sweep over the entire training set is called an *epoch*.

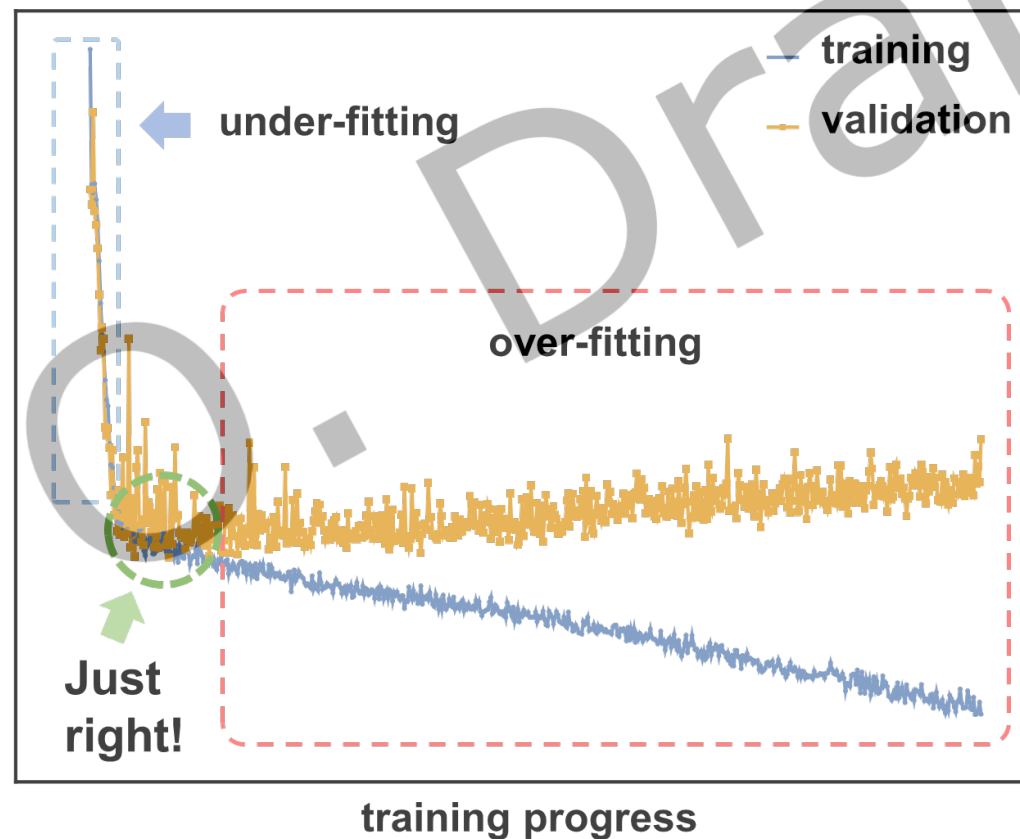
Issues with NNs:

Overfitting, regularization
methods to deal with it:

- weight decay

$$\arg \min_{\theta} \sum_{i=1}^{N_{tr}} (f(\mathbf{x}_i; \theta) - y_i)^2 + \lambda \sum_{j=1}^{N_p} \theta_j^2$$

- early stopping
- data augmentation



Issues with NNs:

- Input values should be scaled, usually standardized to center the inputs and scale them so that their standard deviation is 1 (Z-score normalization)
- It is also important to center reference data
- Number of hidden layers and units should be adjusted often by manual experimentation

Pavlo O. Dral

Issues with NNs:

- Initial guess of weights strongly influences the final parameter values
- Starting with zero values prevents back-propagation algorithm to find better solutions
- Starting with too large values often leads to large generalization errors

Pavlo O. Dral

Issues with NNs:

- Initial guess of weights strongly influences the final parameter values
- Starting with zero values prevents back-propagation algorithm to find better solutions
- Starting with too large values often leads to large generalization errors

Thus, one can get lot of different NNs fitted on the same data!

One can exploit this:

- Take average of multiple NNs to get more stable prediction
- Use deviation between NN predictions to estimate prediction uncertainty (e.g. useful in active learning)

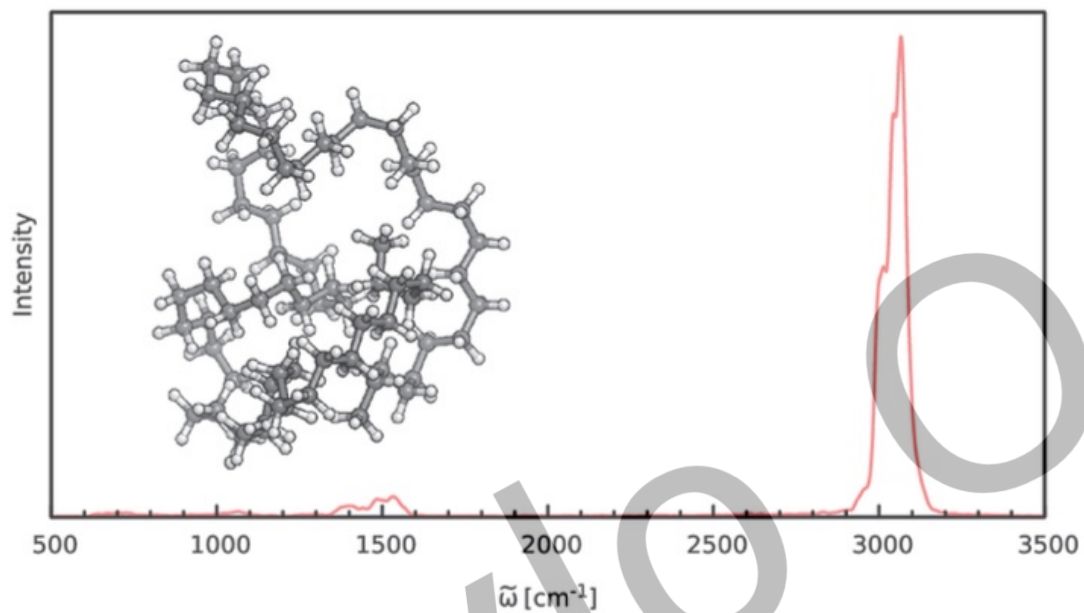
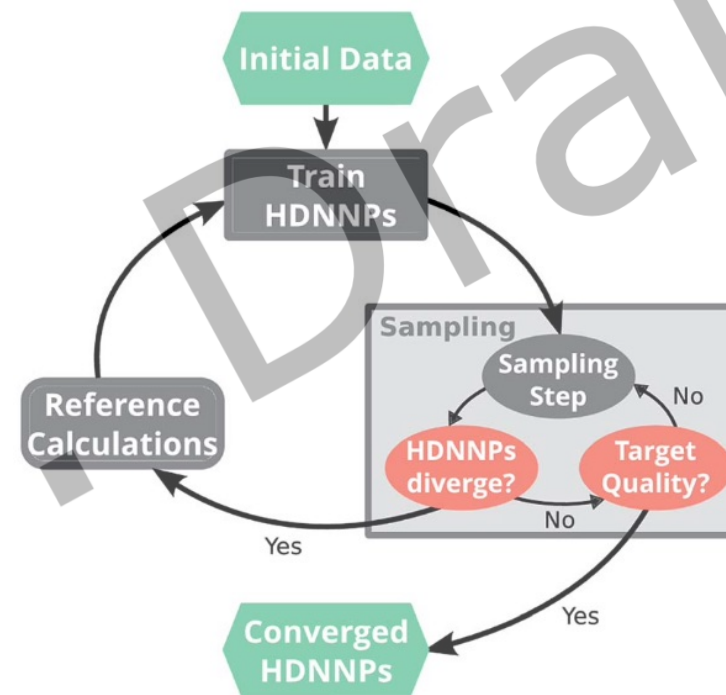


Fig. 6 IR spectrum of the $C_{69}H_{140}$ alkane as predicted by the ML model based on the B2PLYP method.

















Deep learning is based on neural networks (NN) with large depth (for feed-forward neural network – more than one hidden unit) in contrast to shallow neural network

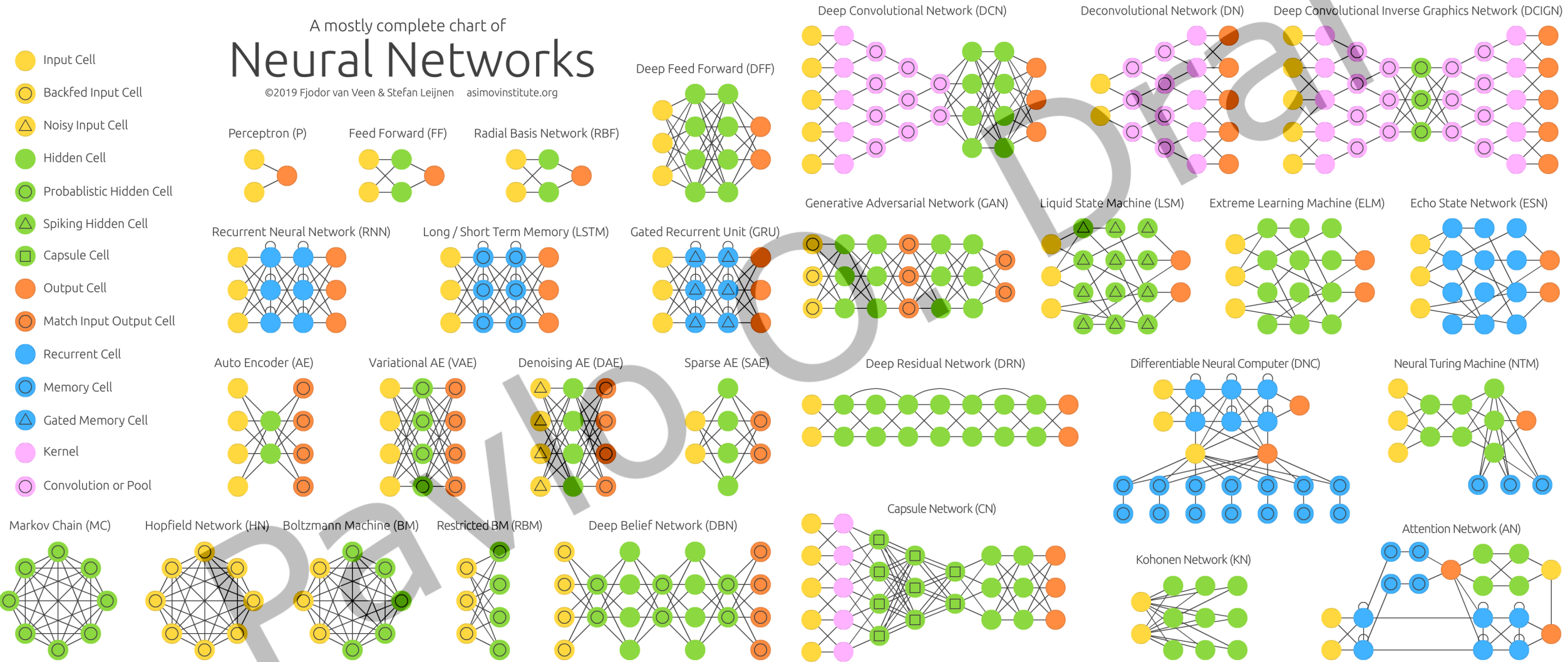
Some of other types of neural networks:

- Convolutional networks
- Recurrent neural networks
- Autoencoders

A mostly complete chart of Neural Networks

©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool



***Parametric vs nonparametric
algorithms***

Pavlo O. Dral

**What method to choose?
Kernel methods or neural networks?**

Pavlo O. Dral

$f(x; \text{parameters})$

Linear regression

$$f(\mathbf{x}_i; \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots$$

Number of parameters is fixed: parametric model

Neural networks are also parametric models

Kernel ridge regression (KRR)

$$f(\mathbf{x}_i; \mathbf{p}) = \sum_{j=1}^{N_{\text{tr}}} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j; \mathbf{b})$$

Number of parameters depends on number of training points:
nonparametric model, e.g. KRR

All have some advantages and disadvantages, but often provide results with similar accuracy.

In many cases **it is not possible to claim that one fitting method is better than another.** The **choice will depend on experience and taste.**[1]

However: You should be aware of the **law of the hammer** and do not try to use a hammer for every problem only because you already have a hammer.

