Machine learning basics

A speedrun

Stefan Chmiela

ML is linearization

ML is (the art of) linearization

ML is (the art of) linearization*

ML is (the art of) linearization*

* most of the time

Supervised learning

Interpolation and generalization

N observations $\mathscr{D} = \left\{ \left(x_i, y_i \right) \right\}_{i=1}^{N}$ drawn *i.i.d.* over P

Labels $y_i = f\left(x_i \right)$ generated by unknown function

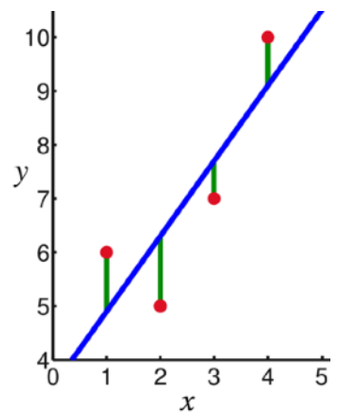
Estimation using parametrized function class $\mathscr{F} = \left\{ f_{\theta \in \Theta} \right\}$

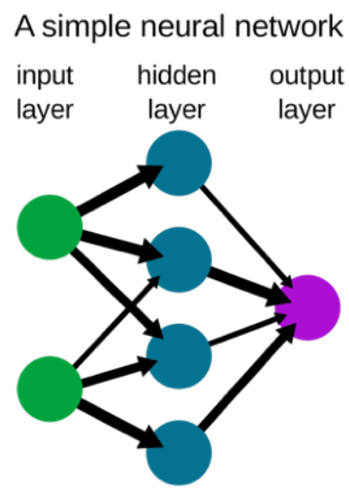
Idealized setup: no noise in the labels

Operation in *interpolation regime:*
estimated $\tilde{f} \in \mathscr{F}$ satisfies $\tilde{f}\left(x_i \right) = f\left(x_i \right)$

Real world: various noise sources, e.g.

- noisy labels
- model bias
- too few data (under-sampling)





e.g. $\theta \in \Theta$: linear regression coefficient or NN parameters

Key ingredients to ML

Key ingredients to ML

- Measure for model performance
- Strong inductive bias + regularization

Measuring performance of ML models

Interpolation and generalization

Assessing model quality: Measure *expected performance* on known and *new* samples drawn from P using some *loss* $L(\cdot, \cdot)$

$$\mathscr{R}(f) := \mathbb{E}_P L(f(x), f(x))$$

Most common: squared loss $L\left(y, y'\right) = \frac{1}{2}\left|y - y'\right|^2$ (Estimates mean label)

Learning problem: *minimize some loss function*

Also useful (Regression)

- **Absolute loss**
  $\left|f\left(\mathbf{x}_i\right) - y_i\right|$ (Estimates median label)

(Classification)

- **Hinge-loss**
  $\max\left[1 - f_{\mathbf{w}}\left(\mathbf{x}_i\right) y_i, 0\right]^p$ (SVMs)
- **Log-loss**
  $\log\left(1 + e^{-f_{\mathbf{w}}(\mathbf{x}_i) y_i}\right)$ (Logistic regression)

A successful learning scheme

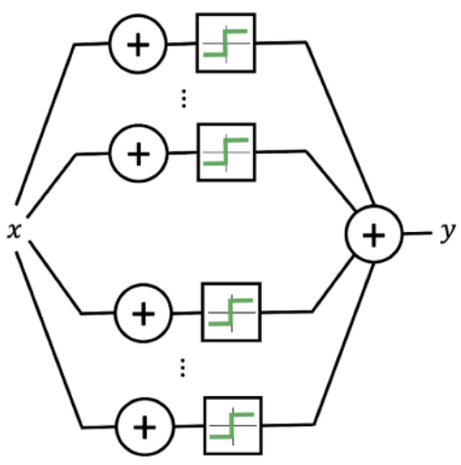
Data is not enough

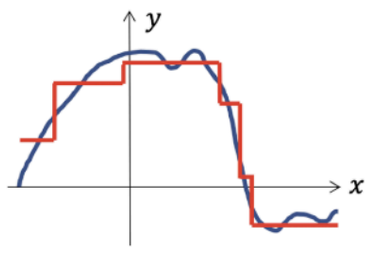
- Encodes appropriate notion of regularity in *inductive bias* for f
- Imposed through the construction of function class $\mathscr{F}$ and the use of *regularization*

preferably rich & dense

- *Universal approximation* theorems: almost arbitrary functions can be learned

  Cybenko (1989); Hornik (1991); Barron (1993); Leshno et al. (1993); Maiorov (1999); Pinkus (1999))





Multilayer perceptrons *(Rosenblatt, 1958)* are universal approximators

Absence of inductive bias?

Bronstein, Michael M., et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges." arXiv preprint arXiv:2104.13478 (2021).

A successful learning scheme

The case for regularity in candidate functions

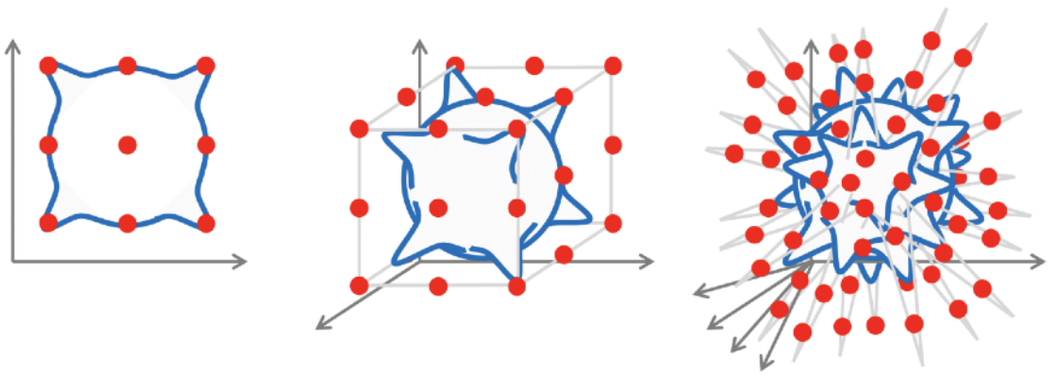
Function regularity is hard to define in high dimensions

e.g. 1-Lipschitz smoothness: $f : \mathcal{X} \to \mathbb{R}$ satisfies

$$\left|f(x) - f(x')\right| \leq \left\| x - x' \right\| \text{ for all } x, x' \in \mathcal{X}$$

"locally smooth" functions

How many observations needed for a good f?



Consider $f(x) = \sum_{j=1}^{2^d} z_j \phi\left(x - x_j\right)$

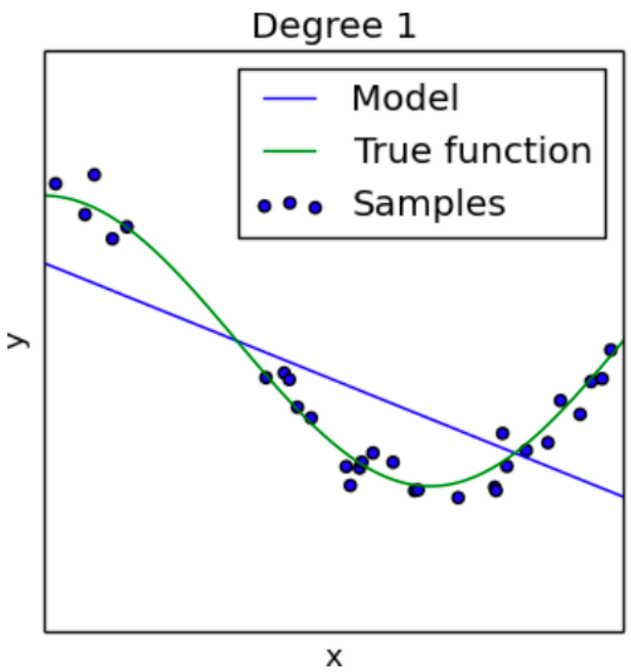
with $z_j = \pm 1, x_j \in \mathbb{R}^d$ and ϕ a locally supported 'bump'.

Exponential growth of Lipschitz class with input dimension!

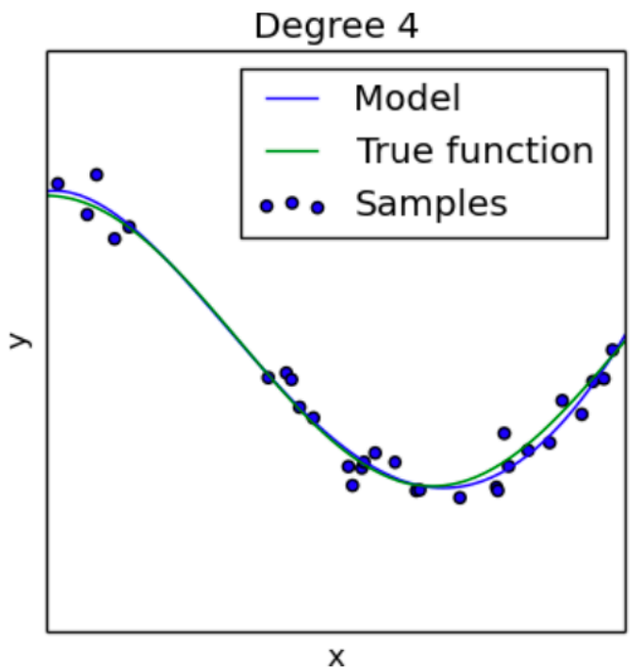
Bronstein, Michael M., et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges." arXiv preprint arXiv:2104.13478 (2021).

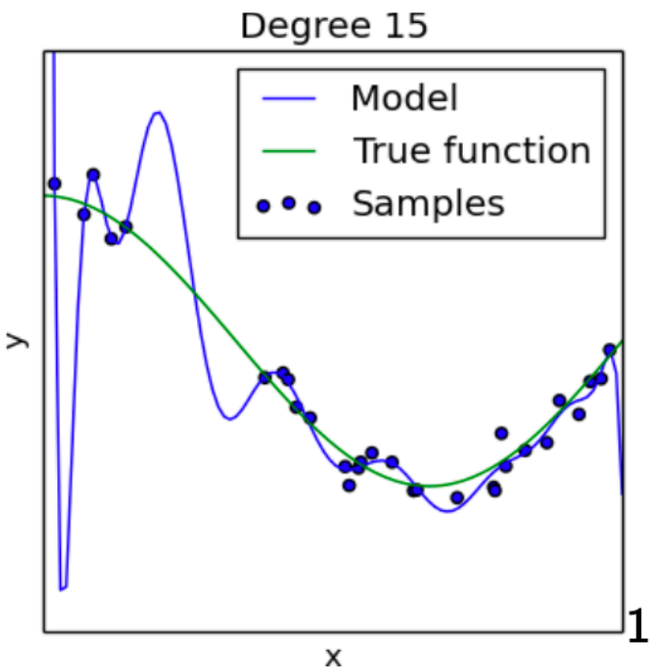
Inductive bias via function regularity

Under-fitting and over-fitting



Under-fitting





Over-fitting

We need a way to control model complexity!

Example taken from http://scikit-learn.org

Inductive bias via function regularity

Encouraging generalization

Given complexity measure on $\mathcal{F}$, $c : \mathcal{F} \rightarrow \mathbb{R}_+$,
e.g. a *norm* → e.g. weight decay on model weights $c\left(f_{\boldsymbol{\theta}}\right) = c(\boldsymbol{\theta})$

Redefine interpolation problem as

$$\tilde{f} \in \arg\min_{g \in \mathcal{F}} c(g) \quad \text{s.t.} \quad g\left(x_i\right) = f\left(x_i\right) \quad \text{for } i = 1, \ldots, N$$

Pick most regular / smooth functions
within our function class.

Regularization

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \underbrace{L\left(f_{\mathbf{w}}\left(\mathbf{x}_i\right), y_i\right)}_{\text{Data loss}} + \underbrace{\lambda r(w)}_{\text{Regularizer}}$$

Regularizer controls complexity of the solution.

Inductive bias via function regularity

Bias-variance tradeoff

- Some asymptotic error is maintained even in the limit $N \to \infty$ (model bias)

  Chosen function class $\mathcal{F}$ does not exactly match our learning problem (e.g. bad features)

- Model can be too powerful/complex for a given dataset (model variance)

  Overly complex models may follow noise in the data.

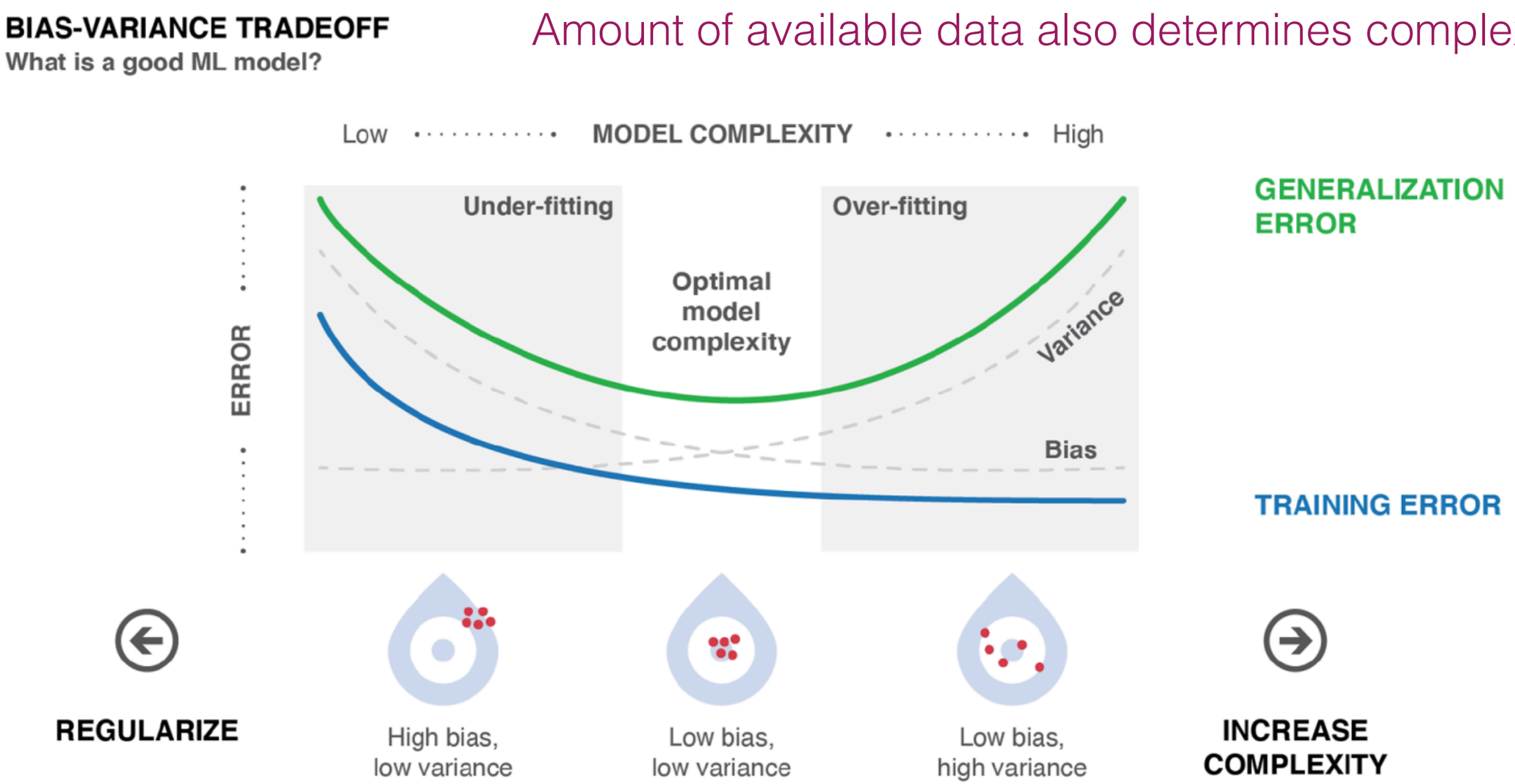
Expected loss can be decomposed:

$$\mathrm{E}\left[(\mathbf{y} - \hat{f}(\mathbf{x}))^2\right] = \mathrm{Bias}^2 + \mathrm{Var} + \mathrm{Noise}$$

Optimal model balances both terms.

Inductive bias via function regularity

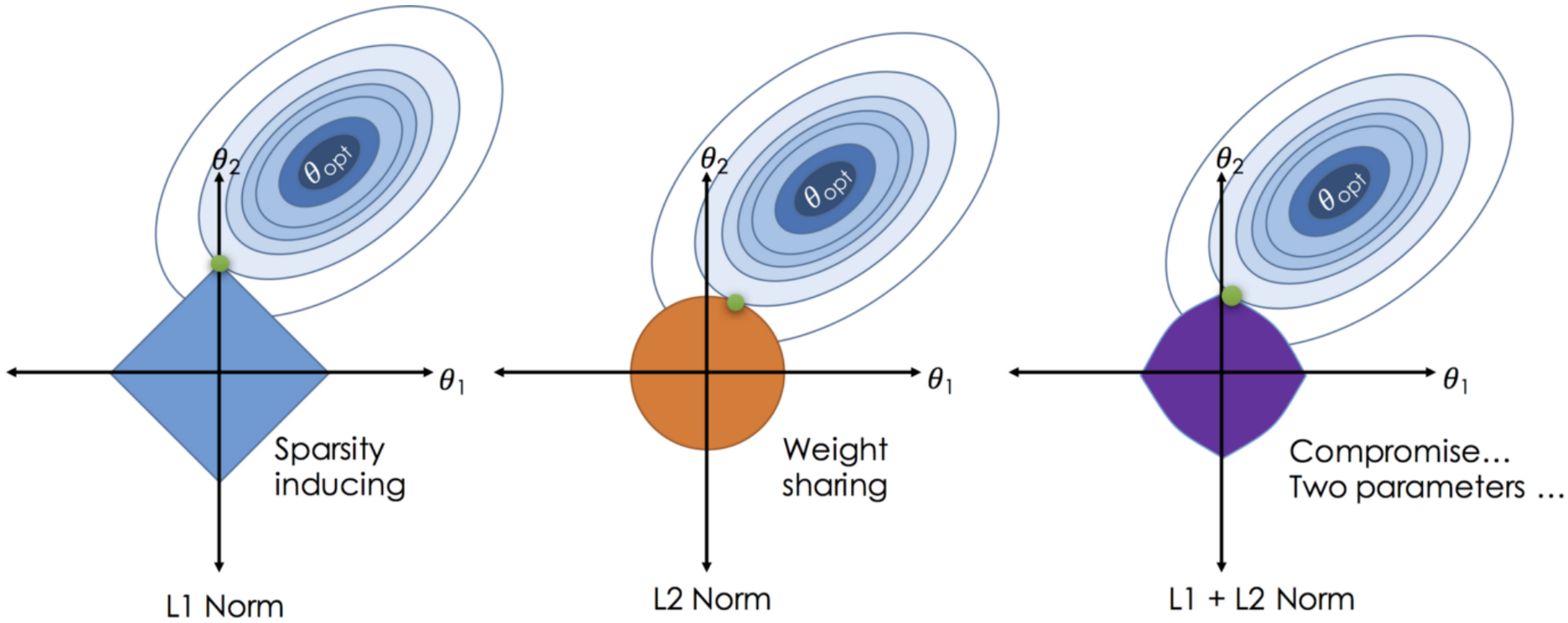
Bias-variance tradeoff



Keith, John A., et al. "Combining machine learning and computational chemistry for predictive insights into chemical systems." Chemical reviews 121.16 (2021): 9816-9872.

A successful learning scheme

Types of regularizers



Common regularization terms

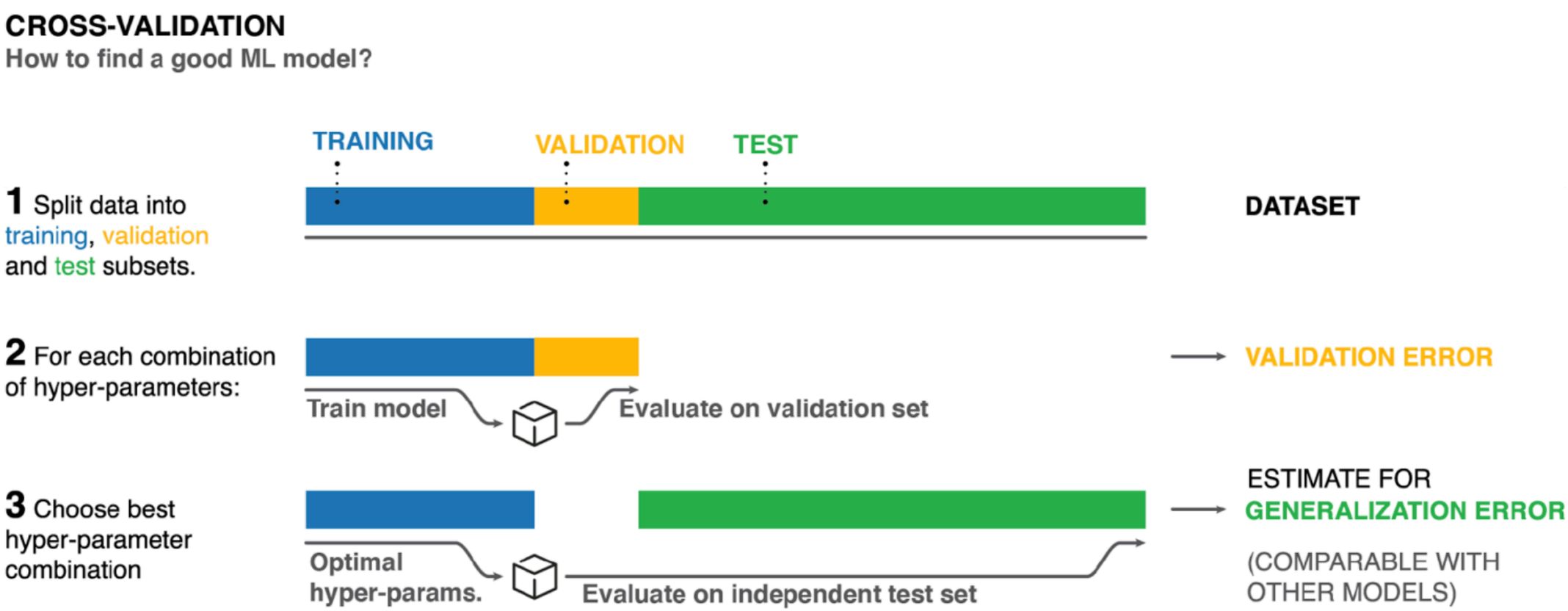
- L_2 **regularization**
  $r(\mathbf{w}) = \mathbf{w}^\top \mathbf{w} = \|\mathbf{w}\|_2^2$
  (strictly convex, differentiable)

- L_1 **regularization**
  $r(\mathbf{w}) = \|\mathbf{w}\|_1$ (non-convex, sparse solution)

- **Elastic net**
  $\alpha\|\mathbf{w}\|_1 + (1-\alpha)\|\mathbf{w}\|_2^2$ (strictly convex, non-differentiable)
  $\alpha \in [0,1)$

Linear least-squares regression

Determining hyper-parameters



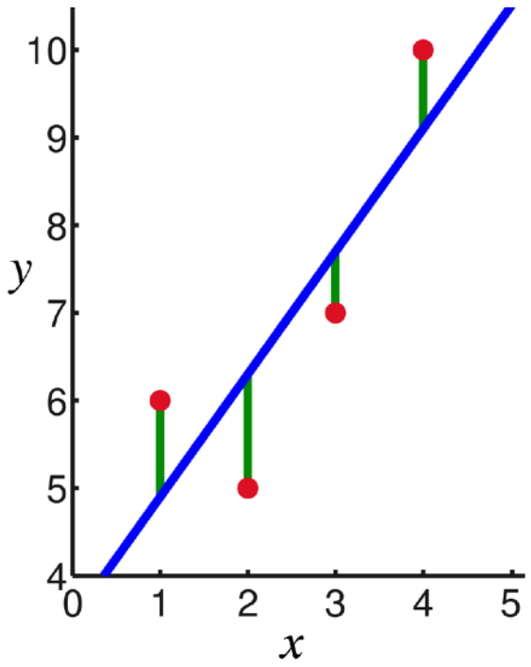
Keith, John A., et al. "Combining machine learning and computational chemistry for predictive insights into chemical systems." Chemical reviews 121.16 (2021): 9816-9872.

Example

Linear least-squares regression

Example

Model of form: $\hat{f}\left(\mathbf{x}_i; \mathbf{w}\right) = \mathbf{x}_i^\mathsf{T}\mathbf{w} = \sum_{j=1}^{N} x_{ij} w_j = \mathbf{X}\mathbf{w}$



Residual: $\mathbf{y} - \mathbf{X}\mathbf{w}$

Linear least squares regression problem: $\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$

Take gradient w.r.t. $\mathbf{w}$ and set it to zero:

Normal equations: $\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} = \mathbf{X}^\mathsf{T}\mathbf{y}$

Unique solution for full-rank $\mathbf{X}$, i.e. rank($\mathbf{X}$) $= N$

$\mathbf{w} = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} = \mathbf{X}^{+}\mathbf{y}$ $\quad$ $\mathbf{X}^{+}$: Pseudo-inverse

Linear least-squares regression

Ridge regularization (L_2)

To avoid overfitting (e.g. due to a high-dimensional feature space), penalize norm of the solution:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

Vary λ to modify model complexity (smoothness).

Setting loss-function derivative to zero leads to:

$$\mathbf{w} = \left(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

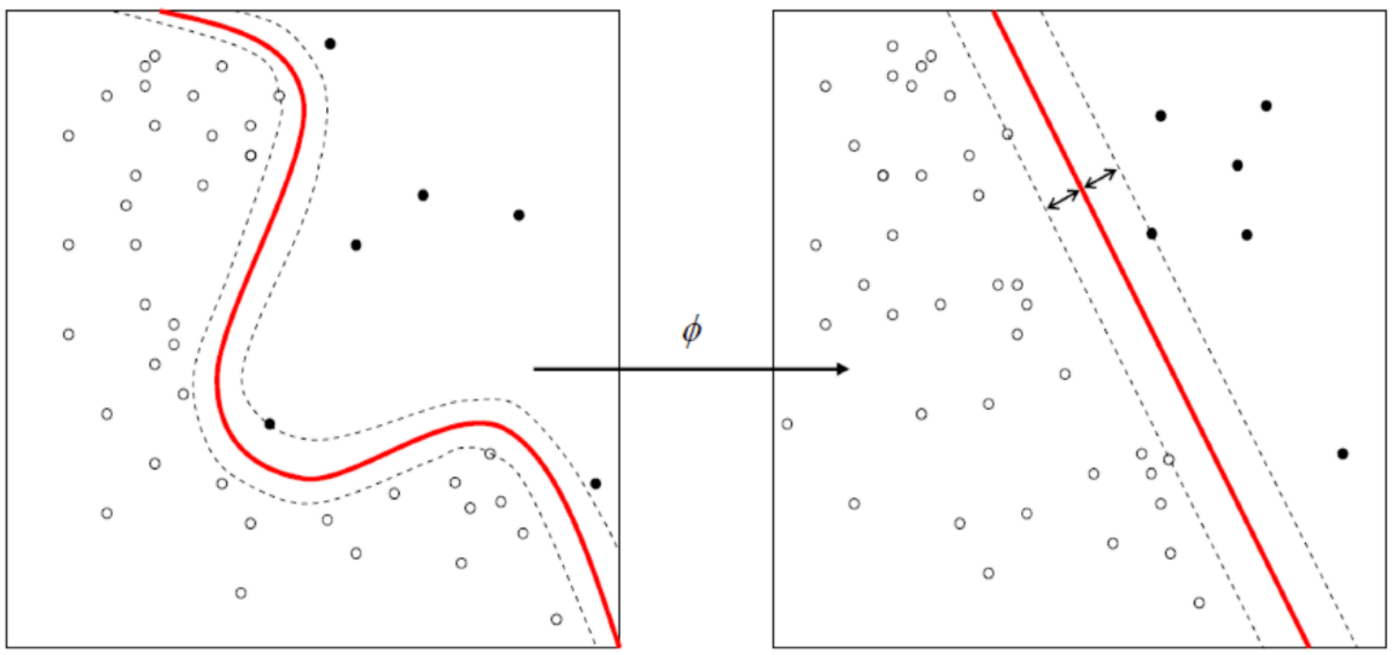
- Generalization performance
- Numerical stability

Linear least-squares regression

Example

Input *features / basis functions*:

$\mathbf{r}_i \rightarrow \mathbf{x}_i = \left(\phi_1\left(\mathbf{r}_i\right), \ldots, \phi_n\left(\mathbf{r}_i\right)\right)^{\mathsf{T}}$ for some $\mathbf{r}_i \in \mathbb{R}^d$



Using non-linear features, our model can be non-linear in $\mathbf{r}$!

Linear least-squares regression

Kernel trick

Key idea:

(1) Replace each data point with feature vector

$$\mathbf{x}_i \rightarrow \phi\left(\mathbf{x}_i\right)$$ Can be high-dimensional or even a function!

(2) Never do that transformation explicitly. Instead define scalar product between features:

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \mathbf{X}_i^\top \mathbf{X}_j = \left\langle \mathbf{X}_i, \mathbf{X}_j \right\rangle$$

(3) Only use the kernel matrix in the algorithm

$$\mathbf{K} = \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{N \times N}, K_{ij} = k\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

Linear least-squares regression

Kernel trick

To avoid overfitting (e.g. due to a high-dimensional feature space), penalize norm of the solution:

$$\min \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

Kernel ridge regression:

$$\mathbf{w} = \left(\lambda \mathbf{I} + \mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top \left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{y}$$

Use either $\mathbf{X}^\top \mathbf{X}$ or $\mathbf{X}\mathbf{X}^\top$ (whichever is cheaper)

Training

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i \qquad \alpha = \left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{y}$$

Prediction

$$\tilde{y} = \mathbf{y} \left(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{X}\tilde{\mathbf{x}}$$

Kernel trick

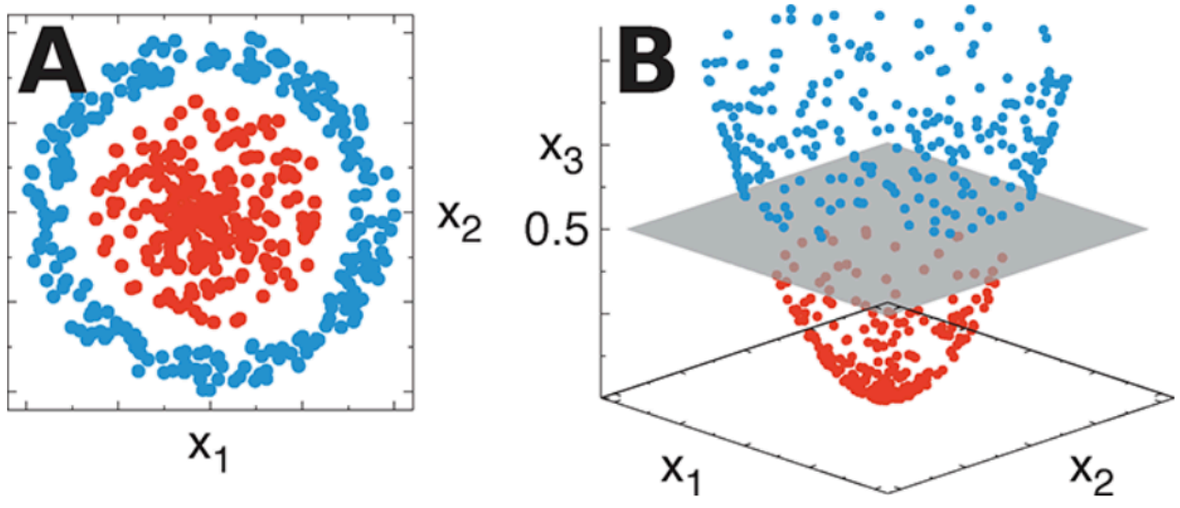
Example: Polynomial kernel

Kernel describing degree-d polynomial

$$k\left(\mathbf{x}_1, \mathbf{x}_2\right) = \left(\mathbf{x}_1^\top \mathbf{x}_2 + c\right)^d$$

Feature mapping for $d = 2$:

$$\phi_2(x) = \left[x_1^2, \ldots, x_n^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{n-1}x_n, \sqrt{2c}x_1, \ldots, \sqrt{2c}x_n, c\right]^\top$$

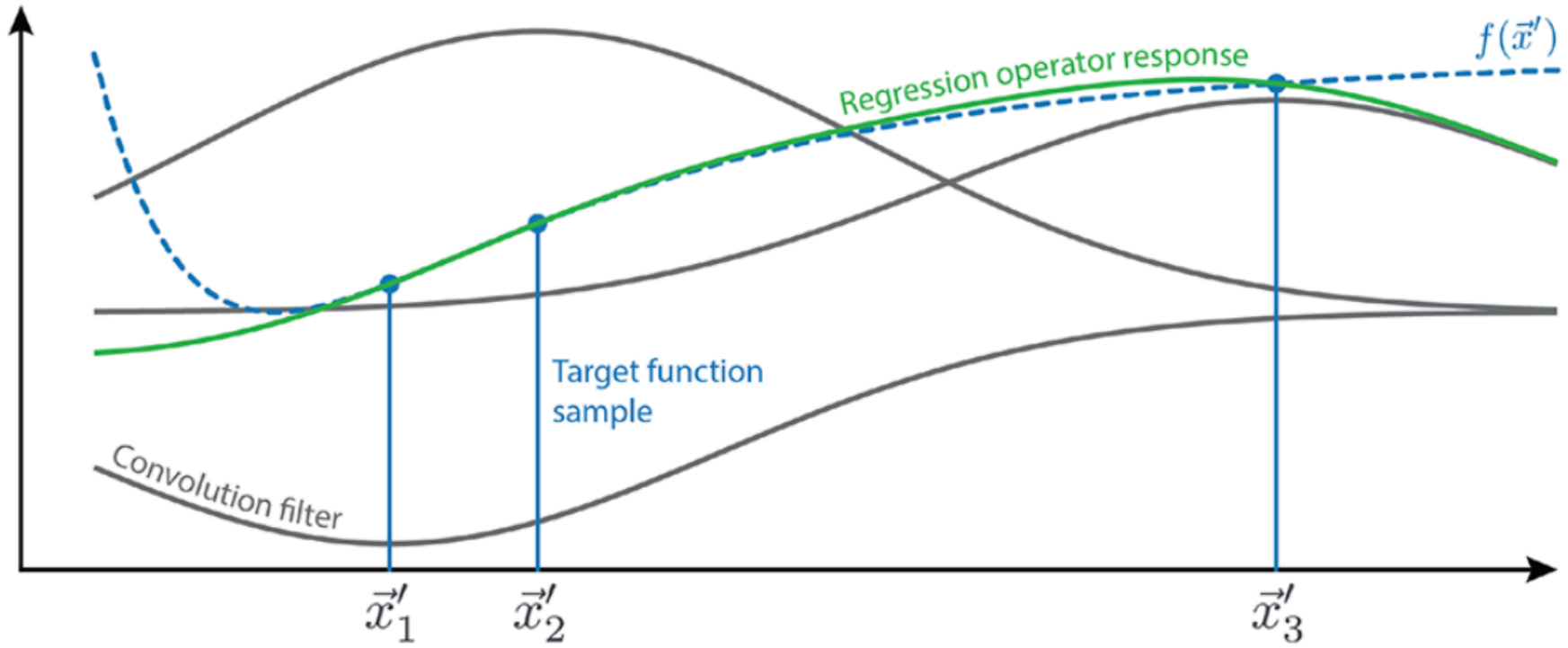


Non-linear classification problem

Unke, O. T., Chmiela, S., Sauceda, H. E., Gastegger, M., Poltavsky, I., Schütt, K. T., ... & Müller, K. R. (2021). Machine learning force fields. Chemical Reviews, 121(16), 10142-10186.

Kernel regression as linear operator

Kernels: Green's functions of operators



$$\hat{f}(\vec{x}) = T_k f(\vec{x}) = \int_{\mathcal{X}} k(\vec{x}, \vec{x}') f(\vec{x}') \, \mathrm{d}\vec{x}'$$

Discrete analogon:
(target function is only sampled partially)

$$\approx \sum^{M} k(\vec{x}, \vec{x}_i) \tilde{f}(\vec{x}'_i)$$

$$= \sum_{i}^{M} k(\vec{x}, \vec{x}_i) \alpha_i = K\vec{\alpha}$$

Regression as linear integral operator using convolution kernel $k(\vec{x}, \vec{x}')$.

Discretization of $f(\vec{x}')$:

$$\tilde{f}(\vec{x}) = \sum_{i}^{M} f(\vec{x}) \delta(\vec{x} - \vec{x}'_i)$$

Regularization; non-stationarity; lack of sampling grid, normalization:

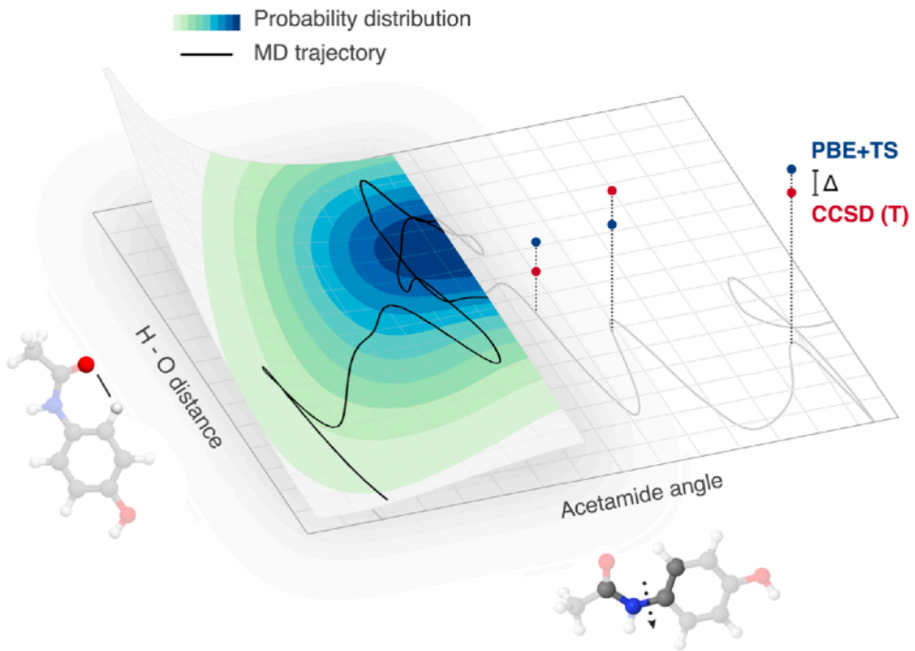
$\tilde{f}(\vec{x}_i) \to \alpha_i$

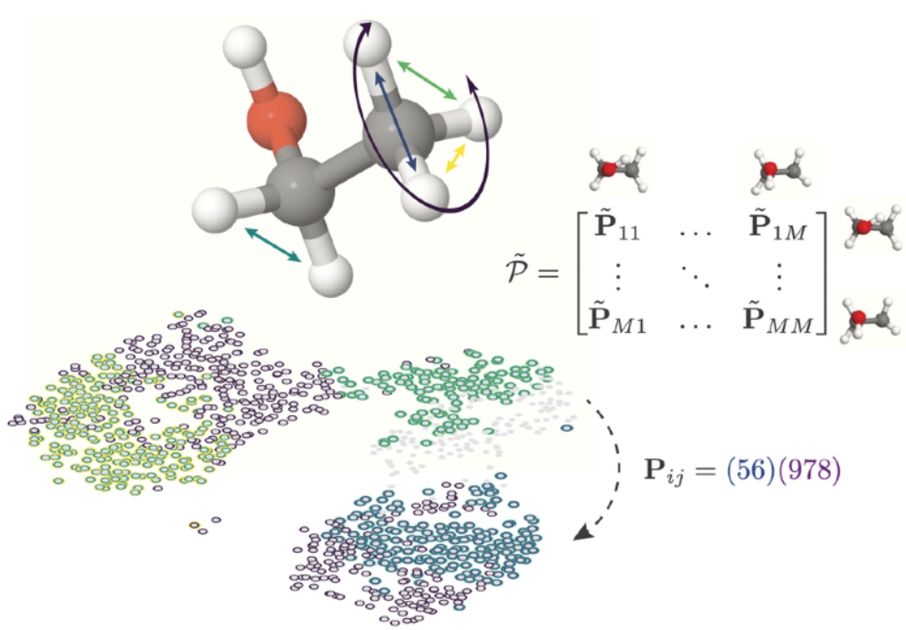
Kernel trick

Properties

- Large (even infinite) features spaces are possible.
- Solve highly non-linear learning problems without carrying out the (expensive) feature transformation.
- Models are linear in their parameters!

Better inductive biases via linear side-constraints $\hat{f}_{\mathcal{G}} = \mathcal{G}\left[T_k\right] f$!

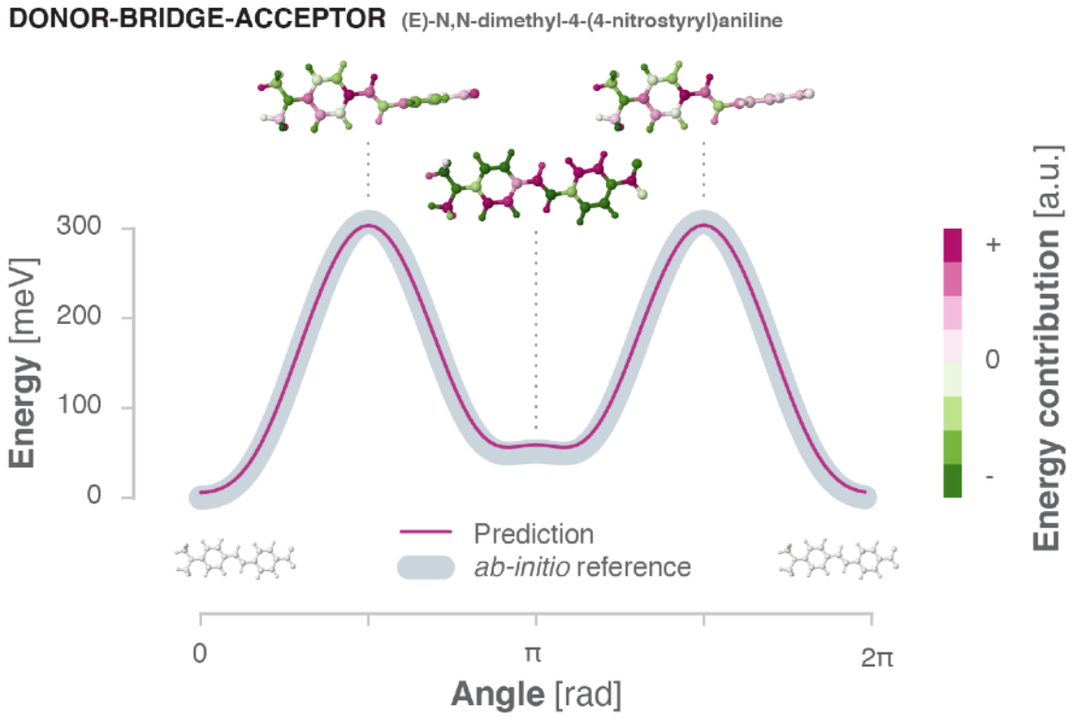




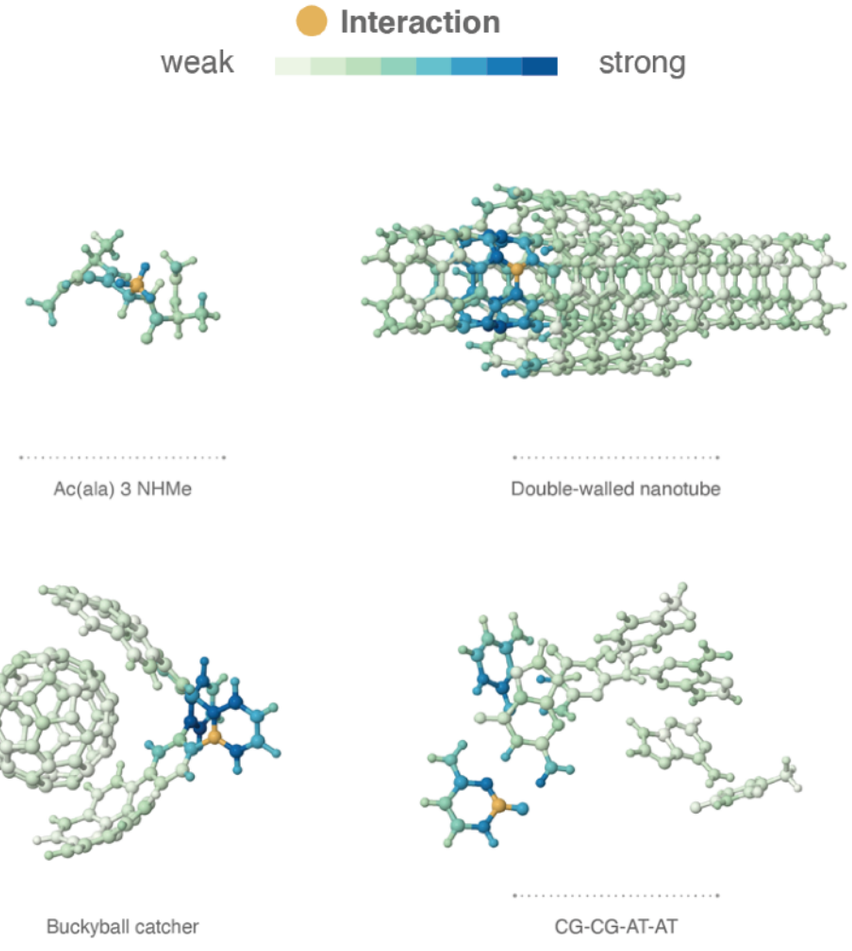
sGDML FF: energy conservation + permutational symmetry constraints

Linear models in general

”Interpretability” of results



Energy prediction



Force prediction

Contributions of individual atoms to prediction in the sGDML FF.

Training

Parameter optimization

Memory: $\mathcal{O}(N^2)$

Time: $\mathcal{O}(N^3)$

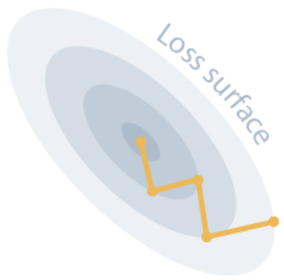
Solve for

$$\alpha = (\mathbf{X}\mathbf{X}^\top + \lambda\mathbb{I})^{-1}\mathbf{y} = \left(\mathbf{L}\mathbf{L}^\top\right)^{-1}\mathbf{y}$$

$\mathbf{X}\mathbf{X}^\top$ is symmetric, positive-definite:
Cholesky factorization with $\mathbf{L} \in \mathbb{R}^{N\times N}$

OR follow the gradient (iterative optimization, use MVPs):

Derivative of loss function: $\nabla_{\mathbf{w}} L(\hat{f}(\mathbf{x}), \mathbf{y}) = (\mathbf{X}\mathbf{X}^\top + \lambda\mathbb{I})\alpha^{t-1} - \mathbf{y}$

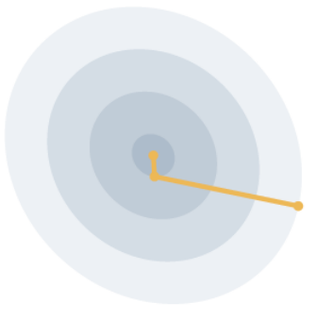


$$\alpha^t = \alpha^{t-1} - \gamma\left[(\mathbf{X}\mathbf{X}^\top + \lambda\mathbb{I})\alpha^{t-1} - \mathbf{y}\right]$$

γ - learning rate

e.g. (stochastic) gradient descent

OR Krylov subspace solver (conjugate gradients):



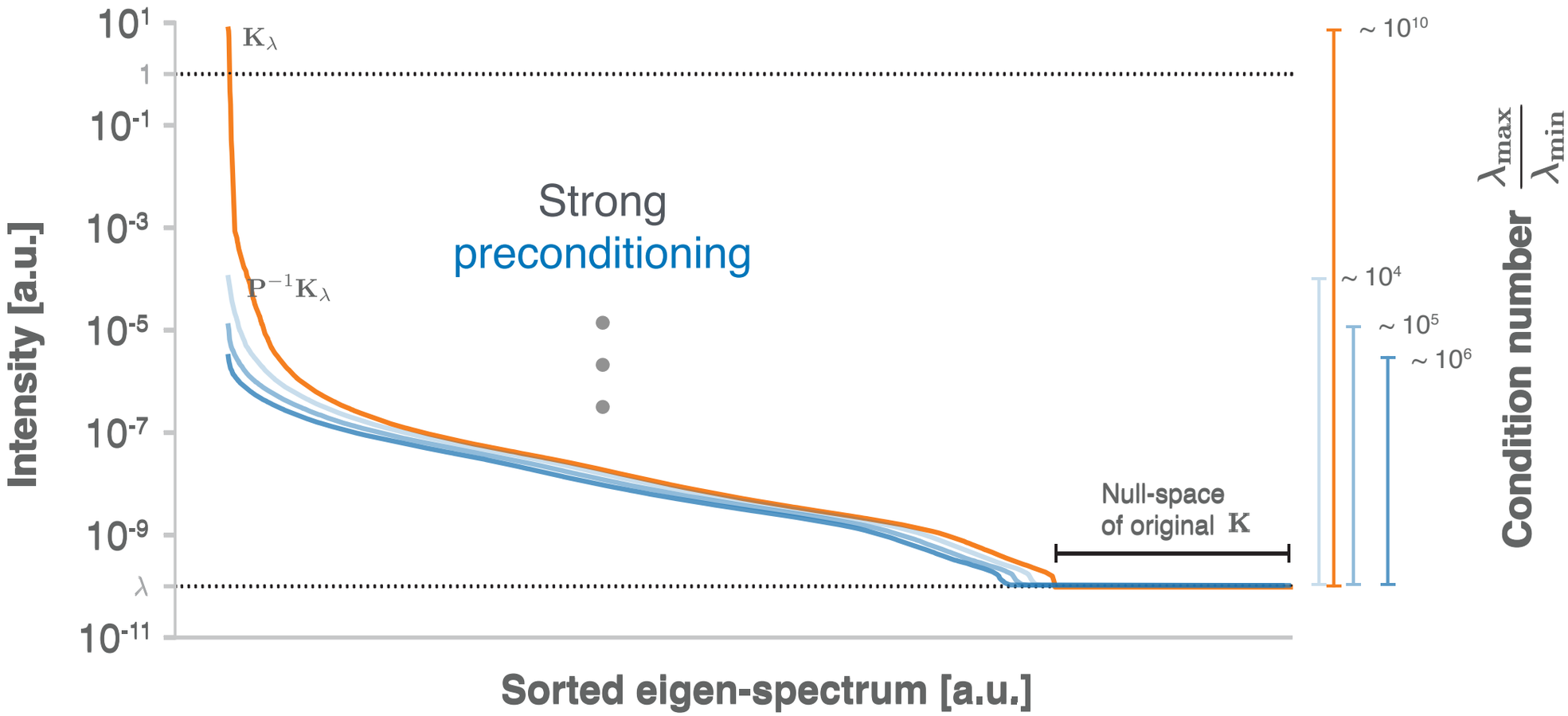
$$\gamma_t = \frac{\left\langle \vec{p}_t, \vec{y} \right\rangle}{\left\langle \vec{p}_t, \vec{p}_t \right\rangle_{\mathbf{K}_\lambda}}$$

- conjugate optimization steps
- optimal, dynamic “learning rate”

Training

Numerical stability

Large linear systems are badly conditioned!



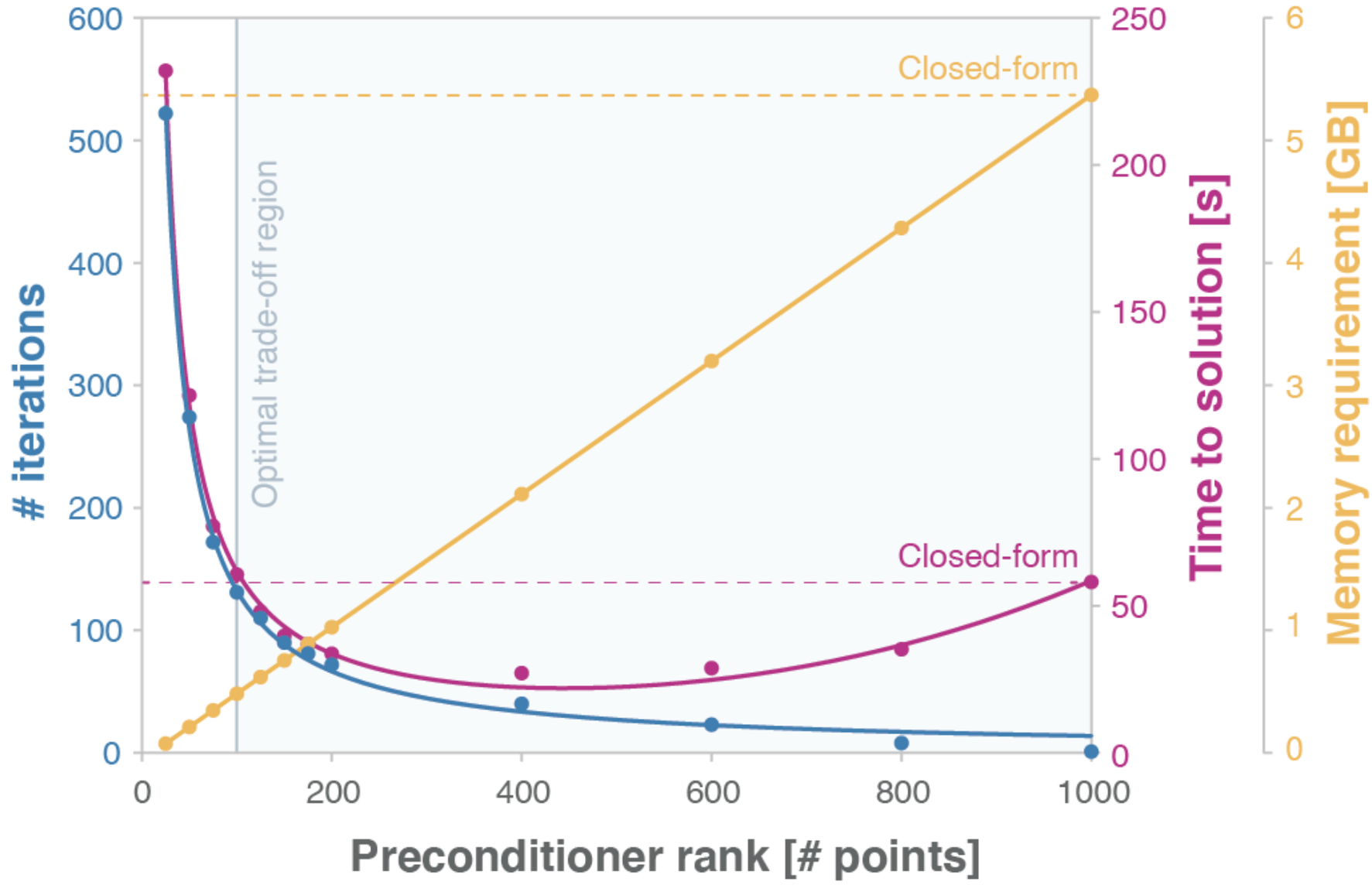
Define a better conditioned equivalent linear system (e.g. Nyström preconditioner):

$$(\mathbf{K} + \lambda I)\mathbf{c} = \mathbf{y} \quad \mapsto \quad B^\top(\mathbf{K} + \lambda I)B\beta = B^\top\mathbf{y}, \quad \mathbf{c} = B\beta$$

Training

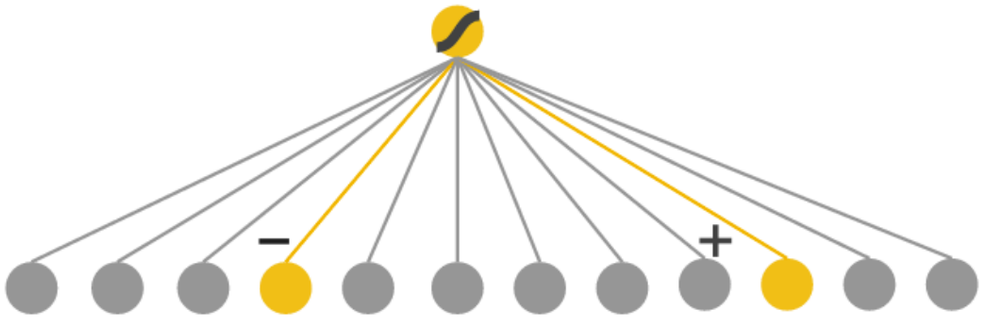
Numerical solvers

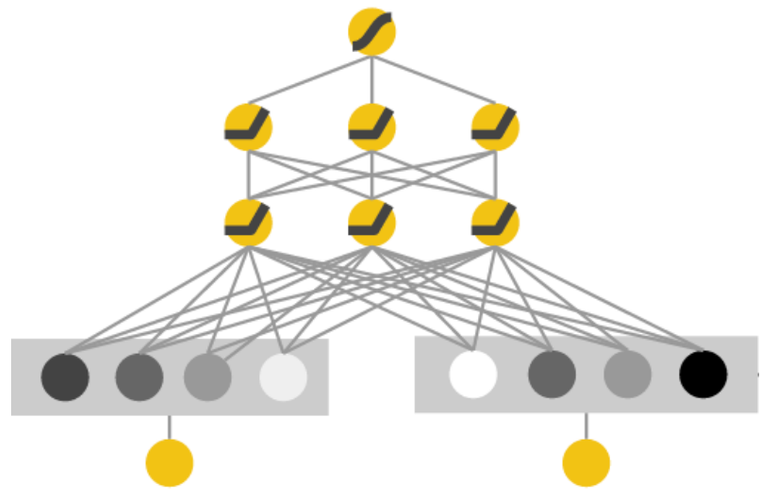
Iterative solvers can trade-off memory and runtime requirements



Deep learning

Non-linear parameter dependencies





Linear systems: shallow architecture

Non-linear parameter dependency: deep learning

(1) Introduce non-linear parameters within *features / basis functions* ϕ_i:

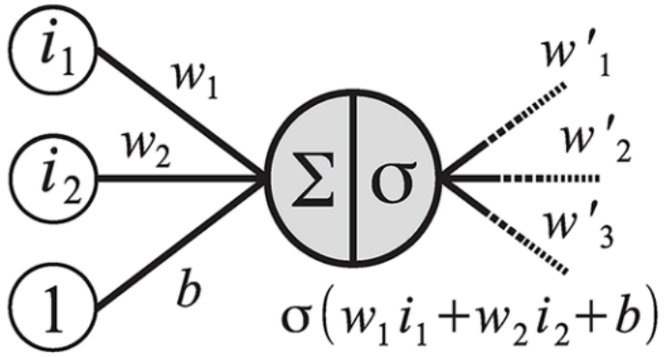
$$\mathbf{r}_i \to \mathbf{x}_i = \left(\phi_1\left(\mathbf{r}_i\right), \ldots, \phi_n\left(\mathbf{r}_i\right)\right)^{\mathsf{T}} \text{ for some } \mathbf{r}_i \in \mathbb{R}^d$$

(2) Training with some flavor of gradient descent

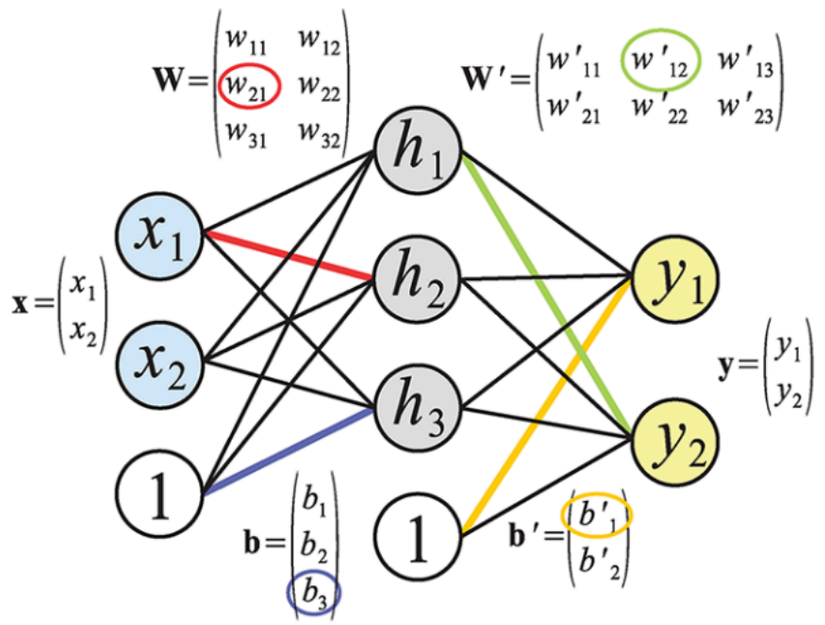
Optimization problem loses most nice properties (e.g. convexity) :(

Deep learning

Non-linear parameter dependencies



Artificial neuron



Feed-forward NN

Training via back-propagation: gradient-descent

Unke, O. T., Chmiela, S., Sauceda, H. E., Gastegger, M., Poltavsky, I., Schütt, K. T., ... & Müller, K. R. (2021). Machine learning force fields. Chemical Reviews, 121(16), 10142-10186.

The “art”-part of ML

Strong inductive biases are crucial

descriptors	comp. efficiency[b]	periodic[c]	unique	invariances[d] T	R	P	global	smooth[e]
atom-centered symmetry functions (ASCF)[411]	Ⓑ 1,2,3-body terms, cutoff	√	X	√	√	√	X	√
smooth overlap of atomic positions (SOAP)[412]	Ⓑ density based, SO(3) rotational group integration	√	X	√	√	√	X	√
Coulomb matrix (CM)[413]	Ⓐ 1,2-body terms	X	√	√	√	X	√	√
sine matrix[414]	Ⓐ 1,2-body terms	√	√	√	√	X	√	√
Ewald sum matrix[414]	Ⓐ 1,2-body terms	√	√	√	√	X	√	√
bag of bonds (BoB)[415]	Ⓐ 1,2-body terms	X	X	√	√	○	√	X
Faber–Christensen–Huang–Lilienfeld (FCHL)[416]	Ⓒ 1,2,3-body terms	√	X	√	√	√	X	√
spectrum of London and Axilrod–Teller–Muto potential (SLATM)[417]	Ⓓ 1,2,3,4-body terms	√	X	√	√	√	X	√
many-body tensor representation (MBTR)[418]	Ⓒ 1,2,3-body terms	X	X	√	√	√	√	√
atomic cluster expansion[420]	Ⓐ 1,2-body terms	√	X	√	√	√	√	√
invariant many-body interaction descriptor (MBI)[460]	Ⓑ 1,2,3-body terms	X	X	√	√	√	X	√
	neural network architectures							
deep potential—smooth edition (DeepPot-SE)[461,462]	Ⓑ 1,2,3-body terms, cutoff	√	X	√	√	√	X	√
MPNN, SchNet[352,434]	Ⓐ/Ⓑ 1,2-body terms, hierarchical	√	X	√	√	√	X	√
Cormorant[463]	Ⓑ 1,2-body terms, hierarchical	X	X	√	√	√	X	√
tensor field networks[464]	Ⓑ 1,2-body terms	√	X	√	√	√	X	√
	similarity metrics							
root mean square deviation of atomic positions (RMSD)[454]	Ⓐ 1,2-body terms, input matching	X	X	○	○	X	√	X
overlap matrix[454]	Ⓐ 1,2-body terms, input matching	X	X	√	√	√	√	X
REMatch[459]	Ⓒ 1,2-body terms, input matching	X	X	√	√	√	√	X
sGDML[207]	Ⓐ 1,2-body terms	√	√	√	√	○[f]	√	√

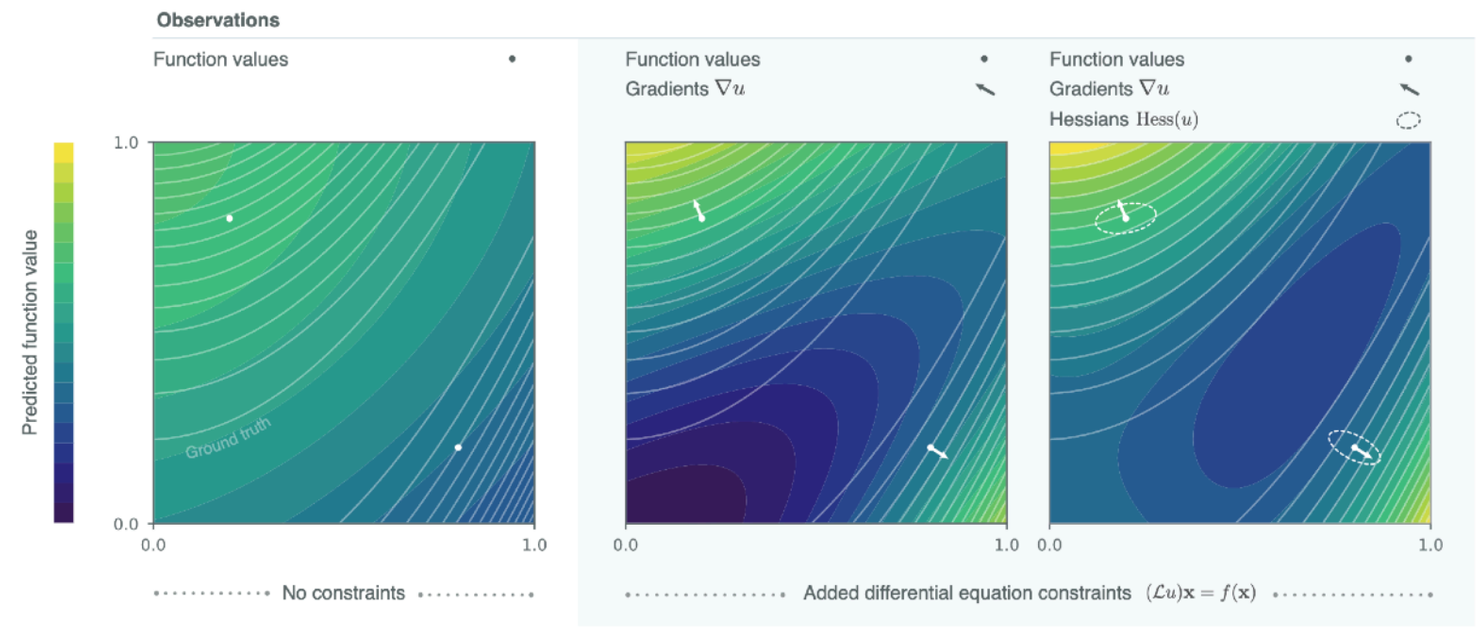
[a]“√” = satisfies condition; “○” = partially satisfies condition; “X” = does not satisfy condition. [b]Computational efficiency ranks with grades Ⓐ–Ⓓ in descending order. The efficiency class reflects the extent that the descriptor requires expensive operations (e.g., a hierarchical processing or matching of inputs). [c]Descriptor has been used within periodic boundary conditions. [d]“**T**” = translational; “**R**” = rotational; “**P**” = permutational. [e]In this context, a descriptor is referred to as smooth if its first derivative with respect to nuclear positions is continuous. [f]Only invariant to permutations represented in the training data.

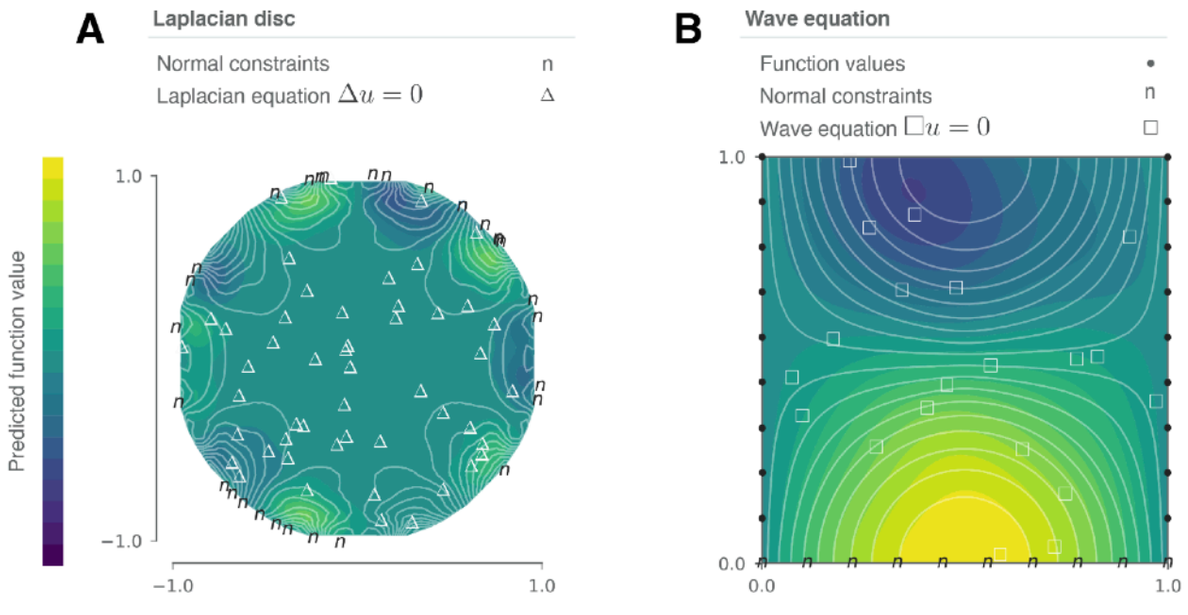
Different inductive biases in ML-FF models

Keith, John A., et al. "Combining machine learning and computational chemistry for predictive insights into chemical systems." Chemical reviews 121.16 (2021): 9816-9872.

The “art”-part of ML

Strong inductive biases are crucial



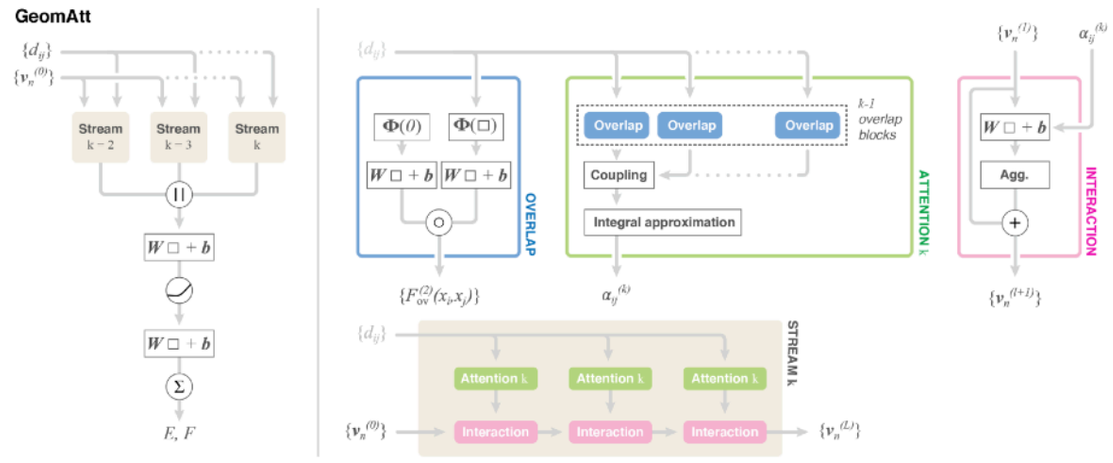


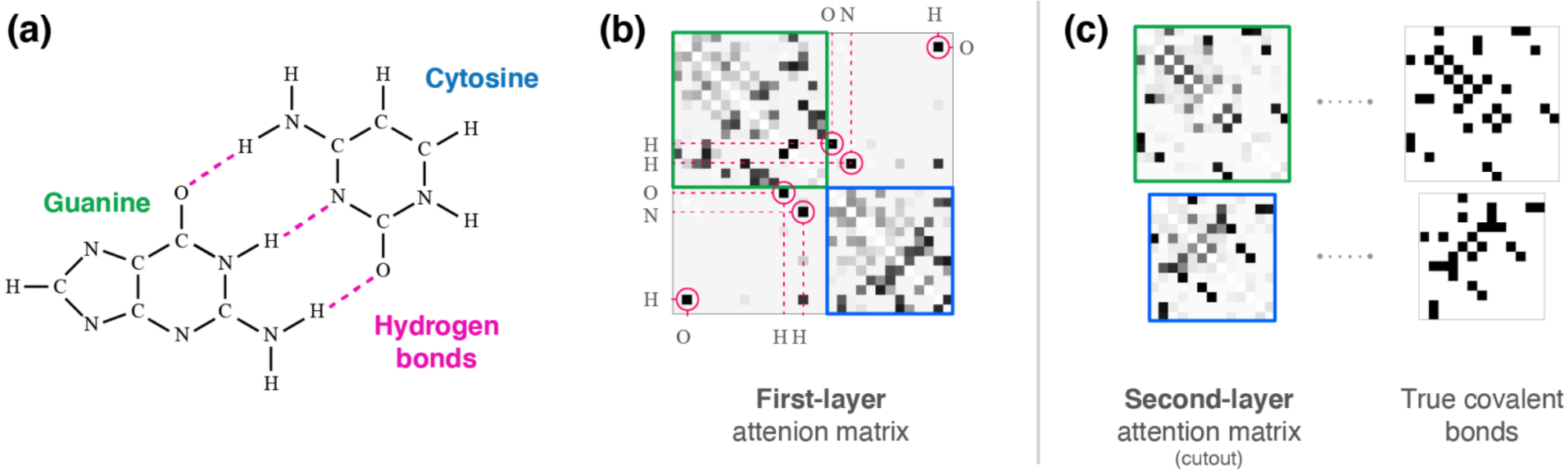
Example: Differential equations provide strong constraints.

The “art”-part of ML

Strong inductive biases are crucial

GeomAtt: Attention mechanism learns relevant interactions





Frank, Thorben, and Stefan Chmiela. "Detect the Interactions that Matter in Matter: Geometric Attention for Many-Body Systems." arXiv preprint arXiv:2106.02549 (2021).