Probabilistic Numerics in 2023

Philipp Hennig

PN School – 27 March 2023

Welcome!

The program

Monday		Tuesday		Wednesday	
9:00	**Philipp Hennig**	9:00	**Tim Sullivan**	9:00	**Carl Henrik Ek**
				10:00	**Fatemeh Yaghoobi**
10:30	coffee	10:30	coffee	10:45	coffee
11:00	**John Cunningham**	11:00	**Simo Särkkä**	11:00	**John Cockayne**
12:00	Lunch	12:00	Lunch	12:00	Lunch
13:00		13:00		13:00	**Masha Naslidnyk**
14:00	**Jonathan Wenger (P)**	14:00	**Nicholas Krämer**	13:45	coffee
16:00	coffee	16:00	coffee	14:00	**Toni Karvonen**
16:30	**Roman Garnett**	16:30	**Nathanael Bosch (P)**	15:00	**Emilia Magnani**
		19:00	Dinner @ Freistil		

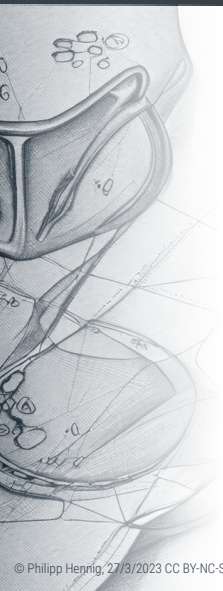
- dinner on Tuesday: for external participants only
- lunch/coffee: external participants take preference

Probabilistic Numerics in 2023

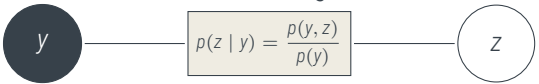
Methods research just got even hotter

Machine Learning *is* Numerical Computation

Data from the disk: – contemporary AI methods require the solution of intractable mathematical tasks



Machine Learning needs

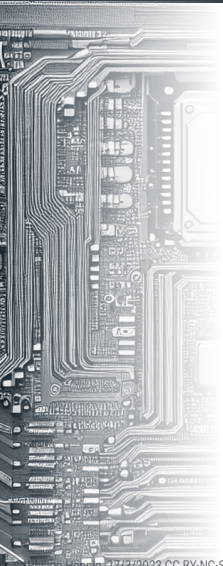


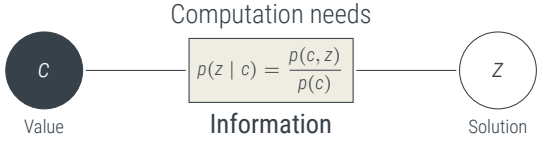
optimization	like	SGD, Adam, …	for	(stochastic) fitting
integration		MCMC, …		probabilistic inference
differential eqs.		Runge-Kutta, …		forecasting & control
linear algebra		Cholesky, CG, …		all of the above

- classic (rule-based) AI required *analytic* computations (etc. nearest neighbors, shortest paths, etc.), modern AI/ML requires *numerical* methods
- **generic methods** save design time, but ML problems have idiosyncratic challenges …

Numerical Computation *is* Machine Learning

Data from the chip: – a numerical method is an autonomous inference agent





optimization	estimate	$z = x$ s.t. $\nabla f(x) = 0$	given $\{c_i = \nabla f(x_i)\}$
integration	estimate	$z = \int_a^b f(x)\,dx$	given $\{c_i = f(x_i)\}$
simulation	estimate	$z = x(t)$ s.t. $x(t)' = f(x(t), t)$	given $\{c_i = f(x_i, t_i)\}$
linear algebra	estimate	$z = \boldsymbol{x}$ s.t. $A\boldsymbol{x} = \boldsymbol{b}$	given $\{As_i = c_i\}$

- A **probabilistic numerical method** produces a **probability measure** over the solution z to a non-analytic computational task, using tractable computations c as an **information sources**
- More details tomorrow in **Tim Sullivan**'s talk

The native framework for AI/ML computation

A probabilistic numerical algorithm is a method that casts computation of intractable quantities in terms of (potentially approximate) probabilistic / Bayesian inference.

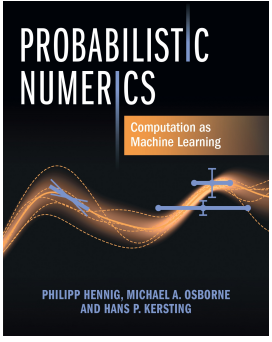
$$p(z \mid c) = \frac{p(c \mid z) \cdot p(z)}{p(c)}$$

Classics methods as special cases with $p(z) = \mathcal{GP}, p(c \mid z) = \delta(c - Az)$:

- Conjugate Gradients, Cholesky
- Gaussian quadrature
- Runge-Kutta
- Quasi-Newton (e.g. BFGS)

Advantages of probabilistic (re-) formulation include

- quantified, structured discretization uncertainty
- correct combination, and active control, of various computational and empirical information sources
- native description of imprecise computation through $p(c \mid z)$



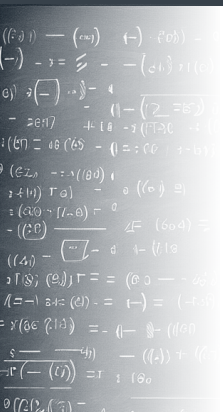
Methods never mattered more

A research agenda for the generative AI age, and an outlook for the coming days

- mechanistic (simulation) and empirical (ML) information for scientific inference
- linear algebra for array-centric computation
- software engineering for deep learning

Assimilating information about dynamical systems

In science, good algorithms matter even more than anywhere else



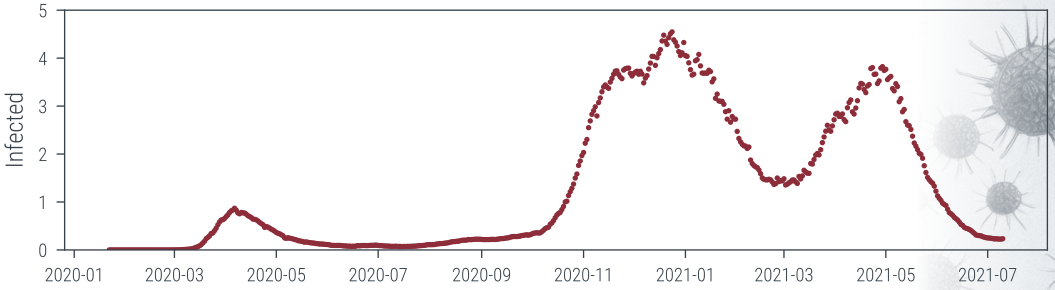
- scientific inference might still be the most impactful domain for AI
- scientific questions often combine **sparse and expensive data** with **generic but powerful mechanisms–laws of nature**
- **statistical inference** is built to extract information from the data, **simulation methods** are built to extract information from the mechanism

Probabilistic simulation methods unite simulation and statistics, for information-agnostic inference on dynamical systems.

An inference problem, or a simulation problem?

Example: Pandemic modelling

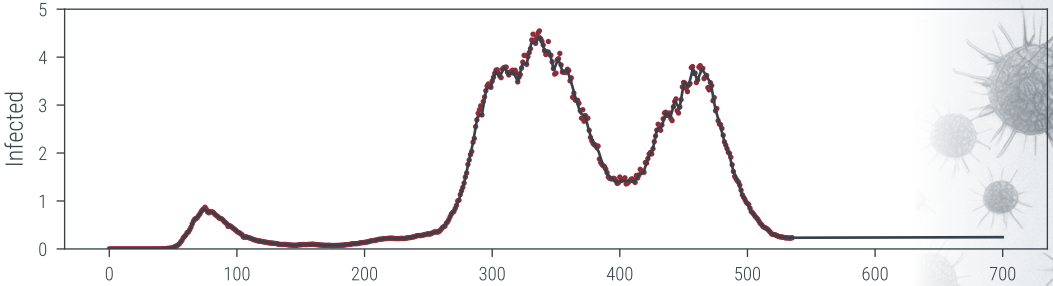
Schmidt, Krämer, Hennig, 2021, NeurIPS 2021



An inference problem, or a simulation problem?

Example: Pandemic modelling

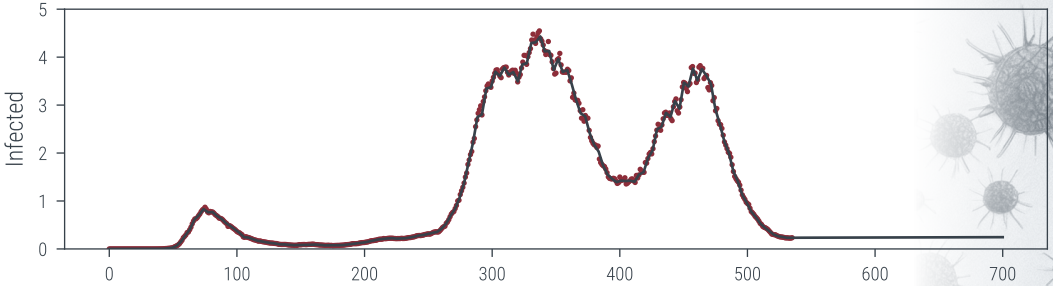
Schmidt, Krämer, Hennig, 2021, NeurIPS 2021



An inference problem, or a simulation problem?

Example: Pandemic modelling

Schmidt, Krämer, Hennig, 2021, NeurIPS 2021

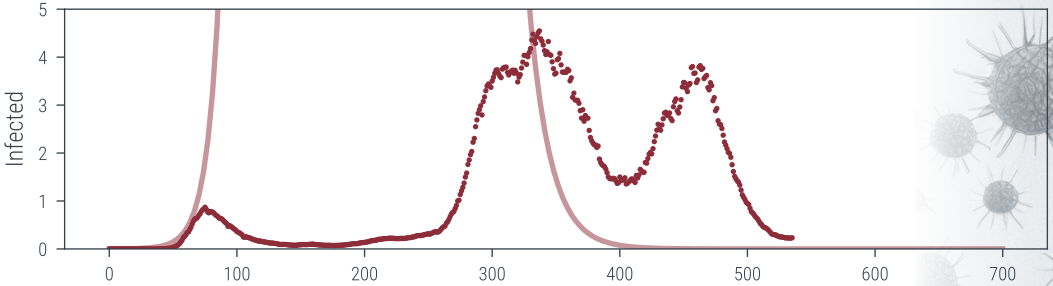


Mechanistic knowledge encodes crucial scientific insight

$$\frac{d}{dt}\mathbf{x}(t) = f(\mathbf{x}(t), \beta(t))$$

$$\frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta(t) \cdot S(t)I(t)/P \\ \beta(t) \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

An inference problem, or a simulation problem?



Mechanistic knowledge encodes crucial scientific insight

$$\frac{d}{dt}\boldsymbol{x}(t) = f(\boldsymbol{x}(t), \beta(t)) \qquad \frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta(t) \cdot S(t)I(t)/P \\ \beta(t) \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

But it requires *latent forces* that may be unknown

https://github.com/scipy/scipy/blob/v1.9.1/scipy/integrate/dop/dopri5.f

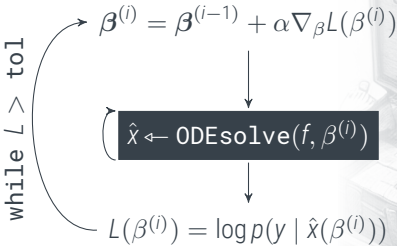
```
      SUBROUTINE DOPRI5(N,FCN,X,Y,XEND,
     &                  RTOL,ATOL,ITOL,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
C ----------------------------------------------------------
C     NUMERICAL SOLUTION OF A SYSTEM OF FIRST 0RDER
C     ORDINARY DIFFERENTIAL EQUATIONS  Y'=F(X,Y).
C     THIS IS AN EXPLICIT RUNGE-KUTTA METHOD OF ORDER (4)5
C     DUE TO DORMAND & PRINCE (WITH STEPSIZE CONTROL AND
C     DENSE OUTPUT).
C
C     AUTHORS: E. HAIRER AND G. WANNER
      A51=19372.D0/6561.D0
      A52=-25360.D0/2187.D0
      A53=64448.D0/6561.D0
      A54=-212.D0/729.D0
      A61=9017.D0/3168.D0
      A62=-355.D0/33.D0
```

Inverse Problems can be solved with with Backprop

automatic differentiation is amazing, but it does not answer scheduling of information

$$\frac{d}{dt}\begin{bmatrix} S(t) \\ I(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} -\beta(t) \cdot S(t)I(t)/P \\ \beta(t) \cdot S(t)I(t)/P - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

$$\frac{d}{dt}x = f(x(t), \beta(t))$$



- Define some loss $L(u)$, e.g.

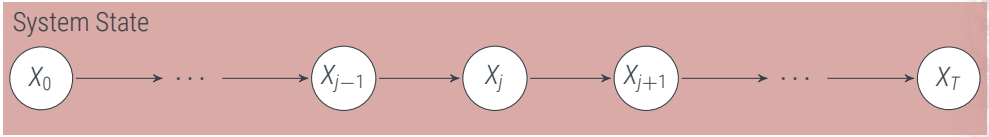
$$L(u) := \sum_i -\log p(y_i \mid \hat{x}(u)) = \sum_i (y_i - H(\hat{x}(u)))^2 + \text{const.}$$

- Compute the gradient $\nabla_u L(u^{(i)})$ with automatic differentiation
- Example: numppyro tutorial, using jax's dopri5
- Cf. *simulation-based inference (SBI), approximate Bayesian computation (ABC), Neural ODEs, etc.*

Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation

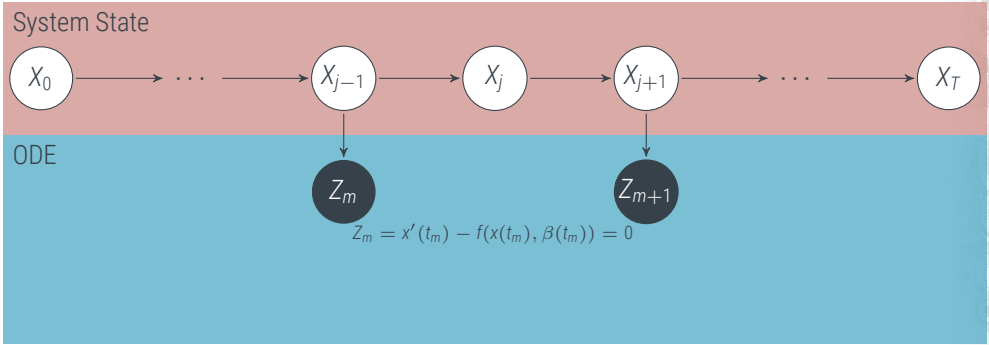
Tronarp, Kersting, Särkkä, Hennig, 2019



Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation

Tronarp, Kersting, Särkkä, Hennig, 2019

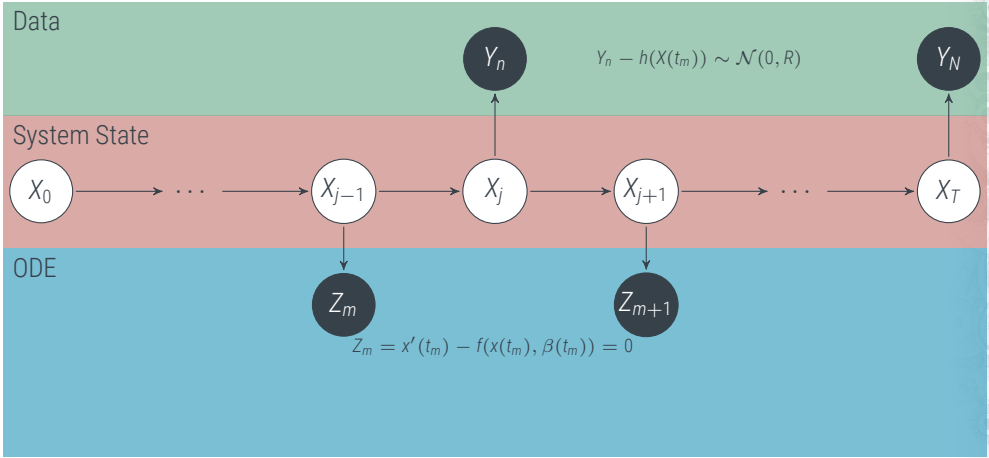


Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation

Tronarp, Kersting, Särkkä, Hennig, 2019

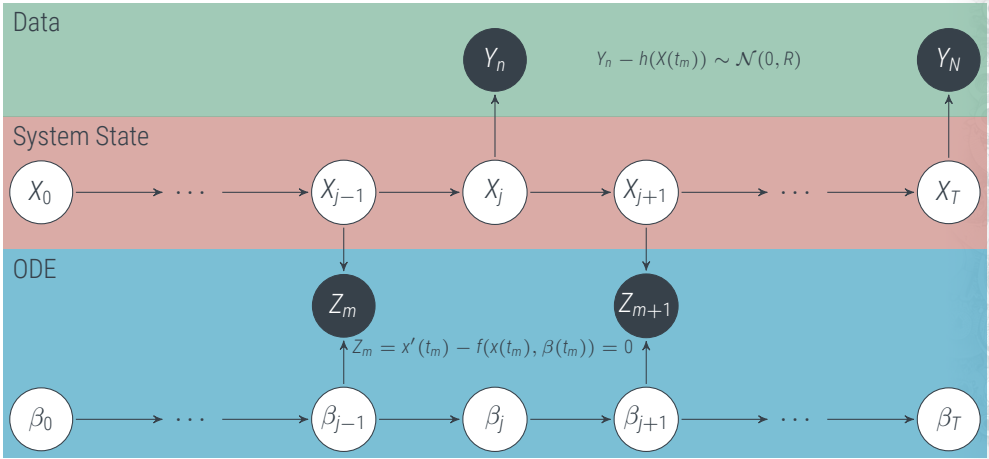
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



Simulation is inference from *information operators*

Bayesian filtering as a primitive for information assimilation

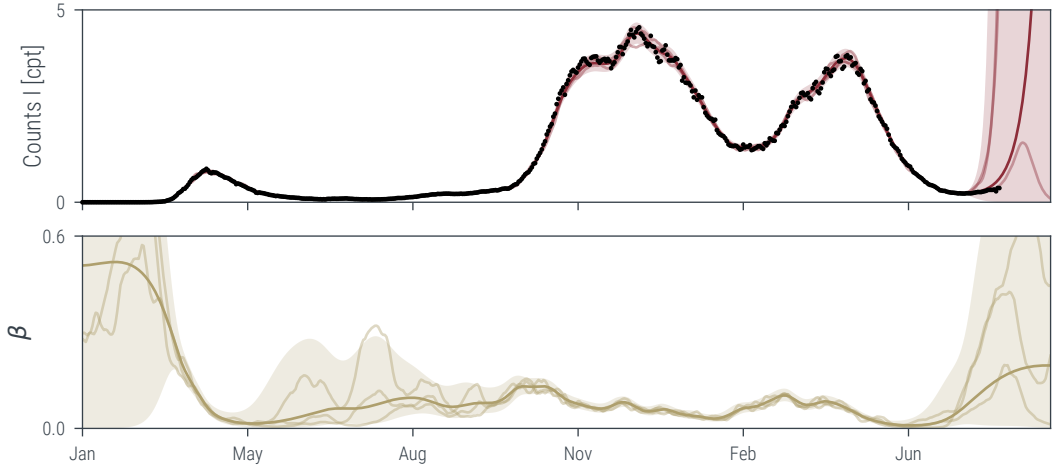
Tronarp, Kersting, Särkkä, Hennig, 2019



latent force and simulation in *one* pass

Example: Covid modelling

Schmidt, Krämer, Hennig, 2021, NeurIPS 2021



More about this *Tomorrow*

Keynotes and Tutorials on Time series and Simulation

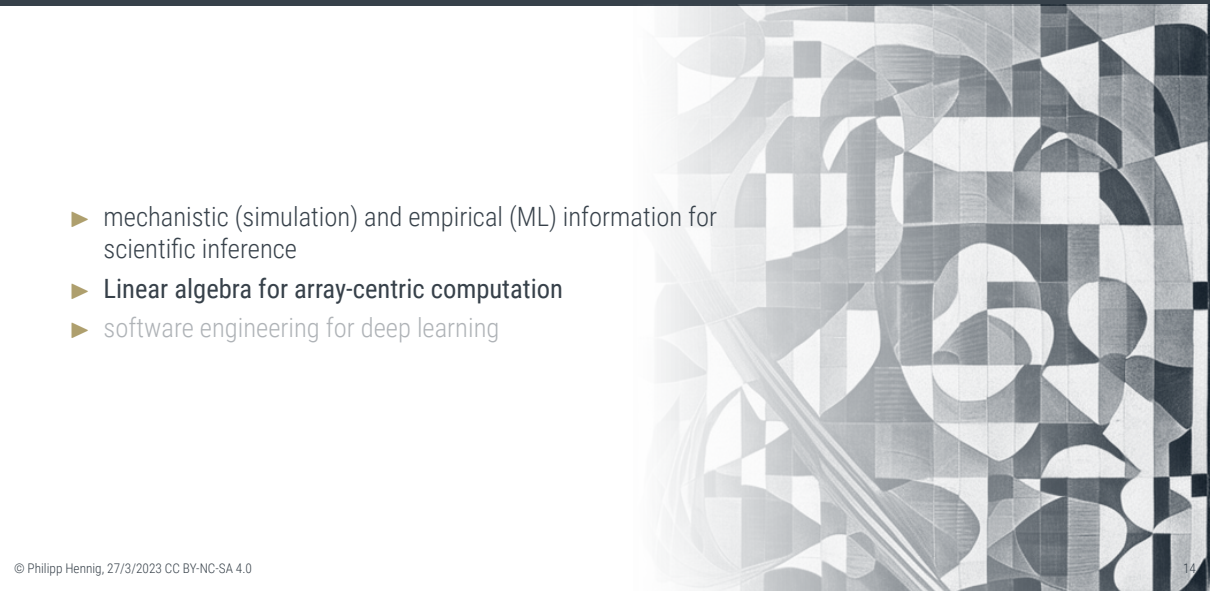
Simo Särkkä
Tuesday, 11:00–12:30

Nicholas Krämer
Tuesday, 14:00–15:30

Nathanael Bosch
Tuesday, 16:30–18:00

Methods never mattered more

A research agenda for the generative AI age, and an outlook for the coming days



- mechanistic (simulation) and empirical (ML) information for scientific inference
- **Linear algebra for array-centric computation**
- software engineering for deep learning

How much does it cost to do GP inference?

more in today's keynotes / tutorials

$$G = k_{XX} + \sigma^2 I$$
$$\mu_\bullet = k_{\bullet X} \cdot \texttt{solve}(G, (\boldsymbol{y} - \mu_X))$$
$$V_{\bullet\circ} = k_{\bullet x} - k_{\bullet X} \cdot \texttt{solve}(G, k_{X\circ})$$

Iterative linear solvers (even Cholesky) collect *projections* of G and y. **They are processing data from the disk iteratively.**
In linear algebra, **processing** and **loading** data are nearly the same thing. Thus, data is only really "there" once it has been computed on.

- read and process all data: $O(N^3)$
- construct and process k informative projections of G: $O(N^2 k)$
- read and process k rows of G: $O(Nk^2)$
- read and process k random points, ignore rest: $O(k^3)$

Why do GPs still matter?

even in the age of LLMs, linear algebra won't die

Khan et al. 2019, Kristiadi et al. 2020, 21, 22

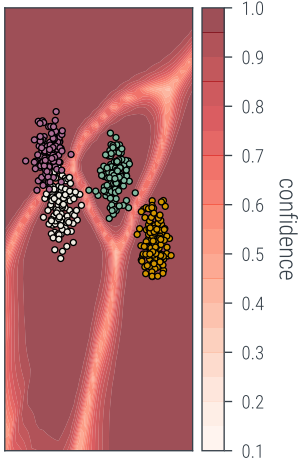
Consider a deep network trained to find $\boldsymbol{\theta}_* \in \mathbb{R}^D$ as

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right)$$

$$= \arg\max_{\boldsymbol{\theta}} \left(\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \right) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{y})$$

Theorem: [Hein et al. 2019] If $f(x, \boldsymbol{\theta})$ is a ReLU network then, for almost any $x \in \mathbb{R}^d$ and $\varepsilon > 0$, there exists an $\alpha > 0$ and a class $k \in \{1, \ldots, K\}$ such that for $z = \alpha x$

$$\frac{e^{f_k(x)}}{\sum_{r=1}^{K} e^{f_r(z)}} \geq 1 - \varepsilon \quad \text{and} \quad \lim_{\alpha \to \infty} \frac{e^{f_k(\alpha x)}}{\sum_{r=1}^{K} e^{f_r(\alpha x)}} = 1$$



Why do GPs still matter?

even in the age of LLMs, linear algebra won't die

Khan et al. 2019, Kristiadi et al. 2020, 21, 22

Consider a deep network trained to find $\boldsymbol{\theta}_* \in \mathbb{R}^D$ as

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right)$$

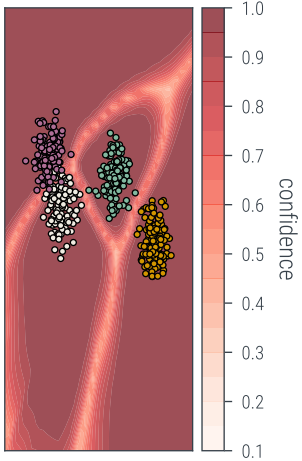
$$= \arg\max_{\boldsymbol{\theta}} \left(\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \right) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{y})$$

find the **Laplace approximation**, with $[\Psi(\boldsymbol{\theta})]_{ij} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}$

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_*) + 1/2 (\boldsymbol{\theta} - \boldsymbol{\theta}_*)^\intercal \Psi(\boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*)$$

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y}) \approx \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}_*, -\Psi^{-1})$$

This requires numerical linear algebra!



Why do GPs still matter?

even in the age of LLMs, linear algebra won't die

Khan et al. 2019, Kristiadi et al. 2020, 21, 22

Consider a deep network trained to find $\boldsymbol{\theta}_* \in \mathbb{R}^D$ as

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right)$$

$$= \arg\max_{\boldsymbol{\theta}} \left(\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) \right) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{y})$$

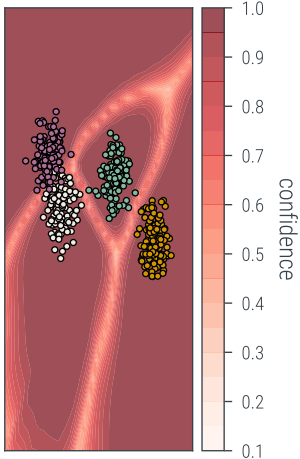
find the **Laplace approximation**, with $[\Psi(\boldsymbol{\theta})]_{ij} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}$

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_*) + 1/2 (\boldsymbol{\theta} - \boldsymbol{\theta}_*)^\intercal \Psi(\boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*)$$

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y}) \approx \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}_*, -\Psi^{-1})$$

linearization $f(\boldsymbol{x}, \boldsymbol{\theta}) = f(\boldsymbol{x}, \boldsymbol{\theta}_*) + J(\boldsymbol{x}, \boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*)$, using $[J]_{ij} = \frac{\partial f_i(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \theta_j}$

$$p(f(\boldsymbol{x}) \mid \boldsymbol{x}, \boldsymbol{y}) = \int p(f(\boldsymbol{x}, \boldsymbol{\theta}) \mid \boldsymbol{x}, \boldsymbol{y}) p(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \approx \mathcal{GP}\left(f; f(\boldsymbol{x}, \boldsymbol{\theta}_*), -J(\boldsymbol{x}) \Psi^{-1} J(\boldsymbol{x})^\intercal\right)$$



Why do GPs still matter?

even in the age of LLMs, linear algebra won't die

Khan et al. 2019, Kristiadi et al. 2020, 21, 22

Consider a deep network trained to find $\boldsymbol{\theta}_* \in \mathbb{R}^D$ as

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right)$$

$$= \arg\max_{\boldsymbol{\theta}} (\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})) = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{y})$$

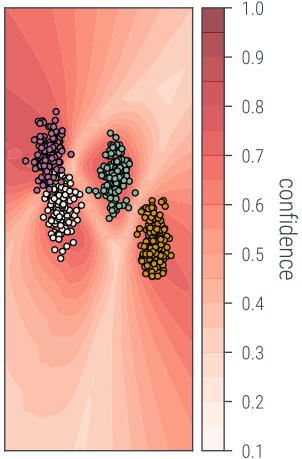
find the **Laplace approximation**, with $[\Psi(\boldsymbol{\theta})]_{ij} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}$

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_*) + 1/2 (\boldsymbol{\theta} - \boldsymbol{\theta}_*)^\intercal \Psi(\boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*)$$

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y}) \approx \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}_*, -\Psi^{-1})$$

linearization $f(\boldsymbol{x}, \boldsymbol{\theta}) = f(\boldsymbol{x}, \boldsymbol{\theta}_*) + J(\boldsymbol{x}, \boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*)$, using $[J]_{ij} = \frac{\partial f_i(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \theta_j}$

$$p(f(\boldsymbol{x}) \mid \boldsymbol{x}, \boldsymbol{y}) = \int p(f(\boldsymbol{x}, \boldsymbol{\theta}) \mid \boldsymbol{x}, \boldsymbol{y}) p(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \approx \mathcal{GP}\left(f; f(\boldsymbol{x}, \boldsymbol{\theta}_*), -J(\boldsymbol{x}) \Psi^{-1} J(\boldsymbol{x})^\intercal\right)$$



More about GPs *Today*

Keynotes and Tutorials on linear Gaussian inference and Bayesian optimization

John Cunningham
Monday, 11:00–12:30

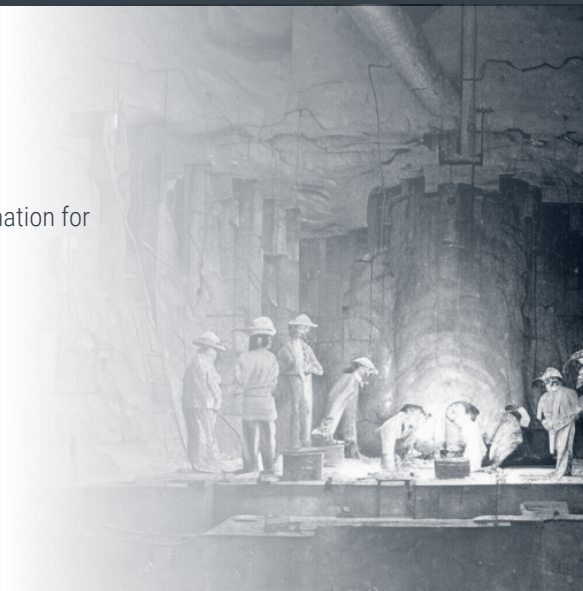
Johnathan Wenger
Monday, 14:00–15:30

Roman Garnett
Monday, 16:00–17:30

Methods never mattered more

A research agenda for the generative AI age, and an outlook for the coming days

- mechanistic (simulation) and empirical (ML) information for scientific inference
- Linear algebra for array-centric computation
- **software engineering for deep learning**

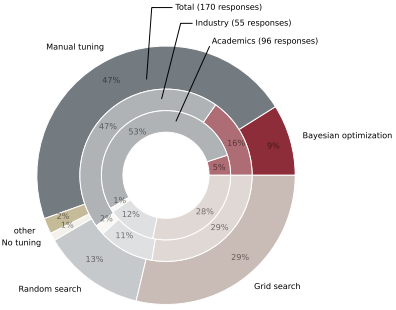


Deep Training is not Mature

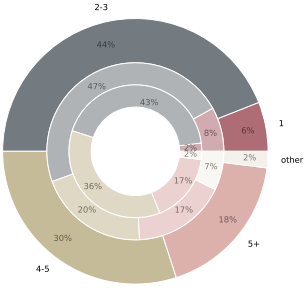
Results of a NeurIPS workshop 2023

https://hity-workshop.github.io/NeurIPS2022/

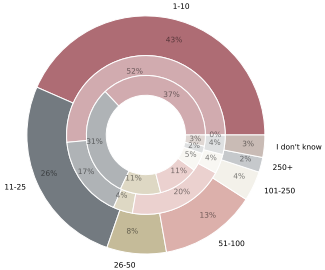
How do you tune your training?



How many parameters do you tune?



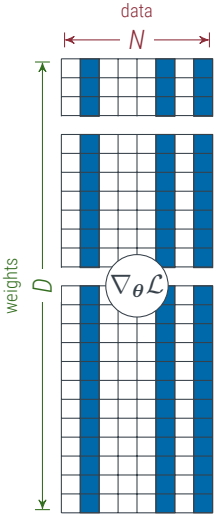
How many tuning trials do you need?



- In deep learning, computation is often an experiment to collect data
- more on automating this process in **Roman Garnett**'s talk

Array-centric Programming

The deep learning software stack is not yet mature



$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^{N} \ell(y_i; f(x_i, \boldsymbol{\theta})) + r(\boldsymbol{\theta}) \right)$$

auto-diff and **batching** ($N \ll M$)

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) &= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \ell(y_i; f(x_i, \boldsymbol{\theta})) + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}) \\ &\approx \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \ell(y_j; f(x_j, \boldsymbol{\theta})) + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}) =: g(\boldsymbol{\theta}) \end{aligned}$$

yield a **random variable**

$$p(g(\boldsymbol{\theta}) \mid \nabla_\theta \mathcal{L}) \approx \mathcal{N}\left(g(\boldsymbol{\theta}); \nabla_\theta \mathcal{L}, \Sigma \in \mathcal{O}\left(\frac{N-M}{NM} \right) \right)$$

Computing statistics is not expensive

contemporary differentiable programming frameworks constrain users

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(y_i; \boldsymbol{\theta}) \quad \approx \frac{1}{M} \sum_{j=1}^{M} \nabla \ell(y_j; \boldsymbol{\theta}) \quad =: g(\boldsymbol{\theta})$$

already computed!

```python
from torch.nn import CrossEntropyLoss, Linear

X, y = load_data()
model = Linear(784, 10)
lossfunc = CrossEntropyLoss()
loss = lossfunc(model(X), y)
loss.backward()

for param in model.parameters():
    print(param.grad)  # this is g(theta). Where is Sigma?
```

Computing statistics is not expensive

contemporary differentiable programming frameworks constrain users

Dangel, Künstner, PH, ICLR 2020, `https://backpack.pt`

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\nabla\ell(y_i;\boldsymbol{\theta}) \quad \approx \frac{1}{M}\sum_{j=1}^{M}\nabla\ell(y_j;\boldsymbol{\theta}) \quad =: g(\boldsymbol{\theta})$$

already computed!

$$\boldsymbol{\sigma}^2(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[\nabla\ell(y_i;\boldsymbol{\theta})\right]^{.2} \approx \frac{1}{M}\sum_{j=1}^{M}\left[\nabla\ell(y_j;\boldsymbol{\theta})\right]^{.2} =: \hat{\boldsymbol{\sigma}}^2(\boldsymbol{\theta})$$

variance adds minimal overhead

```python
# pip install backpack-for-pytorch
from backpack import extend, backpack, Variance

model = extend(Linear(784, 10))
lossfunc = extend(CrossEntropyLoss())
loss = lossfunc(model(X), y)

with backpack(Variance()):
    loss.backward()

for param in model.parameters():
    print(param.grad)  # this is g(theta). Where is sigma2?
    print(param.variance)  # here it is!
```

Probabilistic quantities are accessible

`pip install backpack-for-pytorch` and `https://backpack.pt`

Dangel, Künstner, Hennig, ICLR 2020

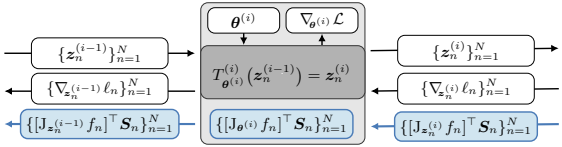
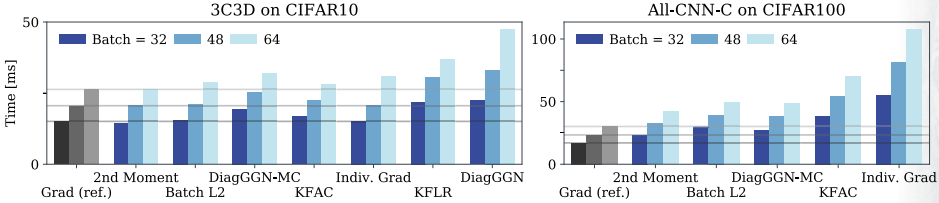


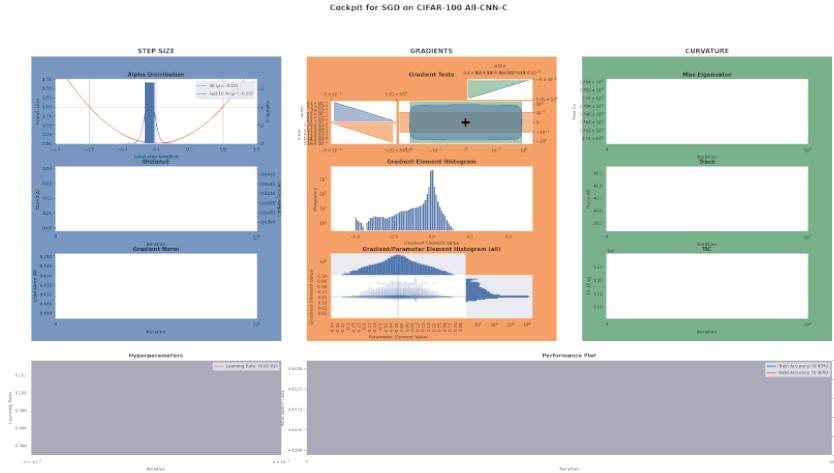
Figure 5: Schematic of the additional backward pass to compute a symmetric factorization of the GGN, $G(\boldsymbol{\theta}) = \sum_n [\mathrm{J}_{\boldsymbol{\theta}} f_n]^\top \boldsymbol{S}_n \boldsymbol{S}_n^\top [\mathrm{J}_{\boldsymbol{\theta}} f_n]$ alongside the gradient at the ith module, for N samples.

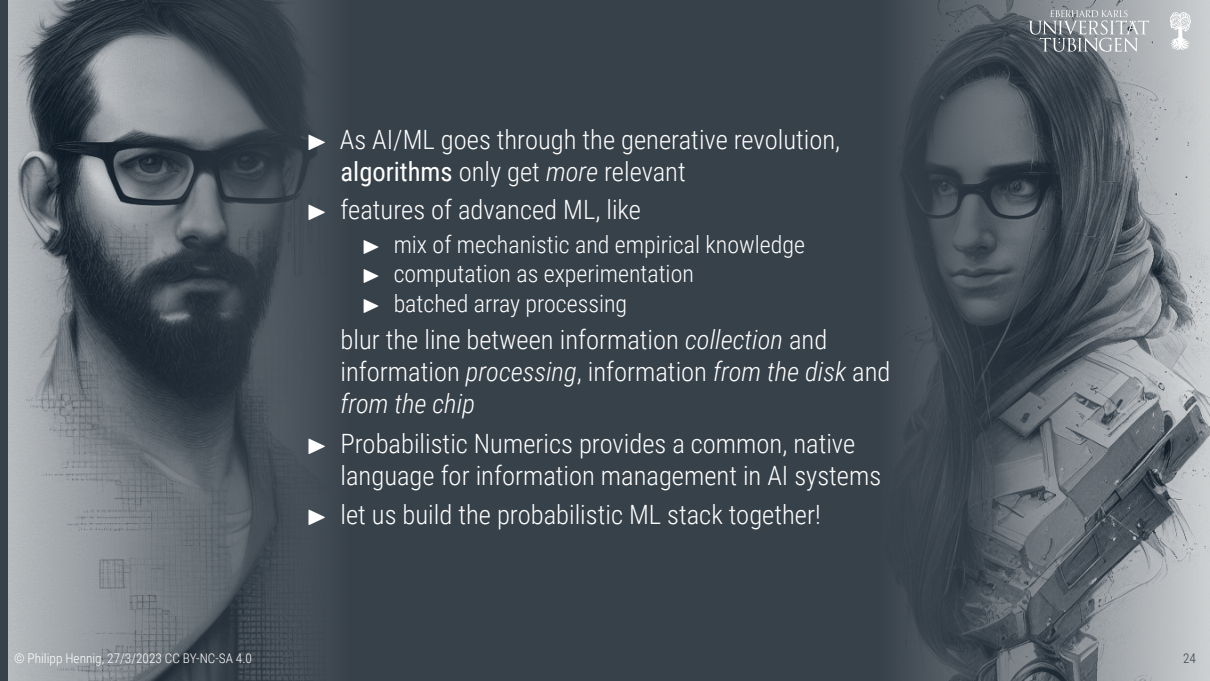


Probabilistic Software Engineering

`pip install cockpit-for-pytorch`

Dangel, Schneider, Hennig, ICML 2022



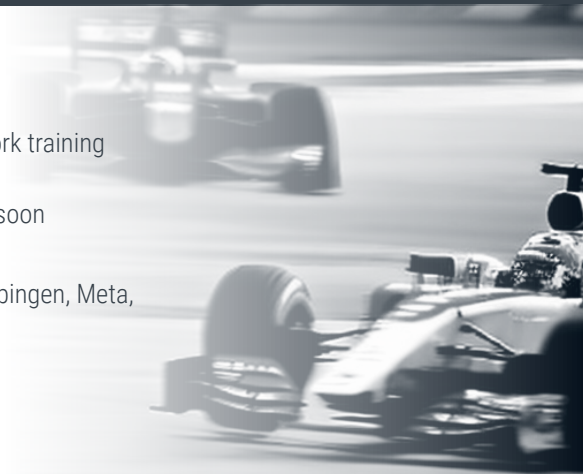


- As AI/ML goes through the generative revolution, **algorithms** only get *more* relevant
- features of advanced ML, like
  - mix of mechanistic and empirical knowledge
  - computation as experimentation
  - batched array processing

  blur the line between information *collection* and information *processing*, information *from the disk* and *from the chip*
- Probabilistic Numerics provides a common, native language for information management in AI systems
- let us build the probabilistic ML stack together!

Join the Competition

The MLCommons AlgoPerf challenge



- A competition between algorithms for neural network training speed
- Algorithmic Performance challenge to be released soon
- seperate rule-set for self-tuning algorithms
- a two year effort by 25+ scientists from Google, Tübingen, Meta, Toronto and others
- find out more from Frank Schneider

ML Commons