

# Statistical Learning

## *Common Core*

Understanding Machine Learning

# General Recipe

statistical learning algorithm by combining:

- **model** (e.g., linear function, Gaussian distribution)
- **objective function** (e.g., squared residuals)
- **optimization algorithm** (e.g., gradient descent)

# Consider Supervised Learning Scenario

map input to output:  $y = f(\mathbf{x})$  (estimated:  $\hat{f}(\mathbf{x})$ )

random variables  $Y$  and  $\mathbf{X} = (X_1, X_2, \dots, X_p) \leftarrow$  usually high-dimensional

curve fitting / parameter estimation:

fit train data set of  $(y_i, \mathbf{x}_i)$  pairs  $\rightarrow$  minimization of cost function

consider discriminative models:

- predict conditional density function  $p(y|\mathbf{x}_i)$   
(as opposed to generative models predicting  $p(y, \mathbf{x}) \rightarrow \mathbf{x}$  not given, more difficult)
- often just conditional mean (depending on used loss)  $E[Y|\mathbf{X} = \mathbf{x}_i]$  of  $p(y|\mathbf{x}_i)$

# Model

# Model in Linear Regression

fit:

$$y_i = \hat{\alpha} + \overbrace{\sum_{j=1}^p \hat{\beta}_j x_{ij}}^{\hat{f}(\mathbf{x}_i)} + \varepsilon_i$$

to be estimated:

- $\hat{\alpha}, \hat{\beta}$

$$\rightarrow \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n \left( y_i - \hat{f}(\mathbf{x}_i) \right)^2$$

(approximating assumed true  $\alpha, \beta, \sigma$ )

predict:

$$\hat{y}_i = E[Y|\mathbf{X} = \mathbf{x}_i] = \hat{f}(\mathbf{x}_i)$$

$$p(y|\mathbf{x}_i) = \mathcal{N}(y; \hat{y}_i, \hat{\sigma}^2)$$

Gaussian

mean

variance  
(reflected  
by  $\varepsilon_i$  in fit)

# Model Ingredients

- underlying probability distribution of  $Y$
- approximated functional form  $\hat{f}(\mathbf{x}) \rightarrow$  depends on used ML algorithm

$$p(y|\mathbf{x}_i) = \text{PDF}(y; \hat{f}(\mathbf{x}_i), \dots)$$

assumed Probability  
Density Function of target

$\hat{f}(\mathbf{x})$  often estimation  
of location parameter

other parameters  
(e.g., scale parameter)  
often not predicted  
per sample (assume  
homoscedasticity)

# Principle of Maximum Entropy

How to choose the right model, i.e., the assumed underlying probability distribution of  $Y$  generating the observed data?

Occam's razor: pick distribution with fewest assumptions

→ distribution with maximum entropy under given range and moment constraints

discrete:  $H(X) = -\sum p_k \log p_k$ , continuous:  $H(X) = -\int p(x) \log p(x) dx$

(→ bulk of entropy from tails of distributions)

examples for maximum entropy distributions:

- normal distribution: continuous on  $(-\infty, \infty)$  with particular mean and variance
- Poisson distribution: discrete on  $[0, \infty)$  with particular mean

# Objective Function



# Loss Function

loss function  $L$ : expressing deviation between prediction and target

$$L(y_i, \hat{f}(\mathbf{x}_i); \hat{\boldsymbol{\theta}})$$

with  $\hat{\boldsymbol{\theta}}$  corresponding to parameters of model  $\hat{f}(\mathbf{x})$

e.g.,  $\hat{\alpha}, \hat{\boldsymbol{\beta}}$  in linear regression

e.g., squared residuals (for regression problems):

$$L(y_i, \hat{f}(\mathbf{x}_i); \hat{\boldsymbol{\theta}}) = \left( y_i - \hat{f}(\mathbf{x}_i; \hat{\boldsymbol{\theta}}) \right)^2$$

# Cost Function

averaging losses over (empirical) training data set:

$$J(\hat{\theta}) = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{f}(\mathbf{x}_i); \hat{\theta})$$

cost function to be minimized according to model parameters  $\hat{\theta}$

→ objective function

# Optimization

# Cost Minimization

minimize training costs  $J(\hat{\boldsymbol{\theta}})$  according to model parameters  $\hat{\boldsymbol{\theta}}$ :

$$\nabla_{\hat{\boldsymbol{\theta}}} J(\hat{\boldsymbol{\theta}}) = 0$$

for mean squared error (aka least squares method):

$$\nabla_{\hat{\boldsymbol{\theta}}} \frac{1}{n} \sum_{i=1}^n \left( y_i - \hat{f}(\mathbf{x}_i; \hat{\boldsymbol{\theta}}) \right)^2 = 0$$

# Ordinary Least Squares

linear regression:  $\hat{f}(\mathbf{x}_i) = \hat{\alpha} + \sum_{j=1}^p \hat{\beta}_j x_{ij}$

matrix notation (drop intercept  $\alpha$  for simplicity here):  $\hat{\mathbf{y}} = \mathbf{X} \hat{\boldsymbol{\beta}}$

solve normal equations for training data set:

$$\nabla_{\hat{\boldsymbol{\beta}}} (\mathbf{y} - \mathbf{X} \hat{\boldsymbol{\beta}})^T (\mathbf{y} - \mathbf{X} \hat{\boldsymbol{\beta}}) = 0$$

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

but usually no closed-form solution  $\rightarrow$  need for numerical optimization

# Maximum Likelihood Estimation

# Likelihood

special objective function: likelihood function  $\mathcal{L}$

- value of predicted probability density function (model) at observed target
- as function of model parameters  $\hat{\theta}$

$$\mathcal{L}(\hat{\theta}; \mathbf{y}|\mathbf{x}) = p(\mathbf{y}|\mathbf{x}; \hat{\theta})$$

a little bit confusing:

$p(\mathbf{y}|\mathbf{x}; \hat{\theta})$  is conditional density function if function of data with  $\hat{\theta}$  fixed

$p(\mathbf{y}|\mathbf{x}; \hat{\theta})$  is conditional likelihood function if function of  $\hat{\theta}$  with data fixed

# Independence Assumption

consider training data set (of size  $n$ ) as one sample from unknown joint probability distribution of  $n$  independent (i.i.d.) sets of random variables  $(Y_1, \mathbf{X}_1), (Y_2, \mathbf{X}_2), \dots, (Y_n, \mathbf{X}_n)$   
(same thing as random sampling  $n$  times from  $(Y, \mathbf{X})$ )

→ likelihood as product of univariate probability density functions:

$$\mathcal{L}(\hat{\boldsymbol{\theta}}; \mathbf{y}|\mathbf{x}) = \prod_{i=1}^n p(y_i|\mathbf{x}_i; \hat{\boldsymbol{\theta}})$$

... to be maximized according to estimated model parameters  $\hat{\boldsymbol{\theta}}$   
(known as maximum likelihood estimation)



# Negative Log-Likelihood

model, e.g., for linear regression:

$$p(y_i|x_i) = \mathcal{N}(y; \hat{\alpha} + x_i \hat{\beta}, \sigma^2)$$

- with  $\mathbf{y}$ ,  $\mathbf{X}$  given in training
- $\hat{\boldsymbol{\theta}} = (\hat{\alpha}, \hat{\beta}, \hat{\sigma}^2)$  to be estimated

logarithmic transformation of  $\mathcal{L}$ : log-likelihood function

$$\ell(\hat{\boldsymbol{\theta}}; \mathbf{y}|\mathbf{x}) = \ln(\mathcal{L}(\hat{\boldsymbol{\theta}}; \mathbf{y}|\mathbf{x})) = \sum_{i=1}^n \ln(p(y_i|x_i; \hat{\boldsymbol{\theta}}))$$

- logarithm monotonic function  $\rightarrow$  maximum of  $\ell$  and  $\mathcal{L}$  at same  $\hat{\boldsymbol{\theta}}$  values
- most probability distributions (e.g., exponential family) only logarithmically concave (important for optimization)
- sum computationally more convenient than product

negative log-likelihood: minimization instead of maximization

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\hat{\boldsymbol{\theta}}} \ell(\hat{\boldsymbol{\theta}}; \mathbf{y}|\mathbf{x}) = \operatorname{argmin}_{\hat{\boldsymbol{\theta}}} \underbrace{(-\ell(\hat{\boldsymbol{\theta}}; \mathbf{y}|\mathbf{x}))}_{\text{objective function}}$$

# Maximum Likelihood

estimate parameters  $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_k)$  solving

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\hat{\boldsymbol{\theta}}} \ell(\hat{\boldsymbol{\theta}}; \mathbf{y} | \mathbf{x})$$

mode of likelihood  
as point estimate

→ likelihood equations (if  $\ell$  differentiable in  $\hat{\boldsymbol{\theta}}$ ):

$$\frac{\partial \ell}{\partial \hat{\theta}_1} = 0, \frac{\partial \ell}{\partial \hat{\theta}_2} = 0, \dots, \frac{\partial \ell}{\partial \hat{\theta}_k} = 0 \quad (\text{in short: } \nabla_{\hat{\boldsymbol{\theta}}} \ell = 0)$$

can be solved explicitly for some cases (e.g., ordinary least squares for linear regression)

but usually no closed-form solution

→ need for numerical optimization (e.g., gradient descent)

**least squares method**

corresponds to maximum likelihood estimation with

**Gaussian model:**  $\sim e^{-\frac{1}{2}\left(\frac{y-\hat{\mu}}{\hat{\sigma}}\right)^2}$

# Interpretation of Maximum Likelihood

quantification of  
difference between two  
probability distributions

maximum likelihood estimation corresponds to minimization of Kullback-Leibler divergence (as well as cross entropy:  $D_{KL}(p||q) = H(p, q) - H(p)$ ) between true data-generating probability distribution (manifested by empirical distribution of training data) and probability distribution of model:

$$\begin{aligned} \operatorname{argmin}_{\hat{\theta}} D_{KL} \left( p_{\text{model}}(y|\mathbf{x}; \hat{\theta}) || p_{\text{data}}(y|\mathbf{x}) \right) &= \operatorname{argmin}_{\hat{\theta}} \int p_{\text{data}}(y|\mathbf{x}) \log \frac{p_{\text{data}}(y|\mathbf{x})}{p_{\text{model}}(y|\mathbf{x}; \hat{\theta})} dy \\ &= \operatorname{argmin}_{\hat{\theta}} \frac{1}{n} \sum_{i=1}^n \log \frac{p_{\text{data}}(y_i|\mathbf{x}_i)}{p_{\text{model}}(y_i|\mathbf{x}_i; \hat{\theta})} = \operatorname{argmax}_{\hat{\theta}} \sum_{i=1}^n \ln \left( p_{\text{model}}(y_i|\mathbf{x}_i; \hat{\theta}) \right) = \operatorname{argmax}_{\hat{\theta}} \ell(\hat{\theta}; \mathbf{y} | \mathbf{x}) \end{aligned}$$

$E[\dots]$                       no contribution from  $p_{\text{data}}$

*make the model distribution match the empirical distribution*

(mean squared error corresponds to cross-entropy between empirical distribution and Gaussian model)

# Iterative Optimization

# Gradient Descent

usually (except for special cases like ordinary least squares) no closed-form solution to ML optimization problems like minimization of a cost function or maximization of a likelihood function:

$$\nabla_{\hat{\theta}} J(\hat{\theta}) = 0$$

→ need for numerical methods

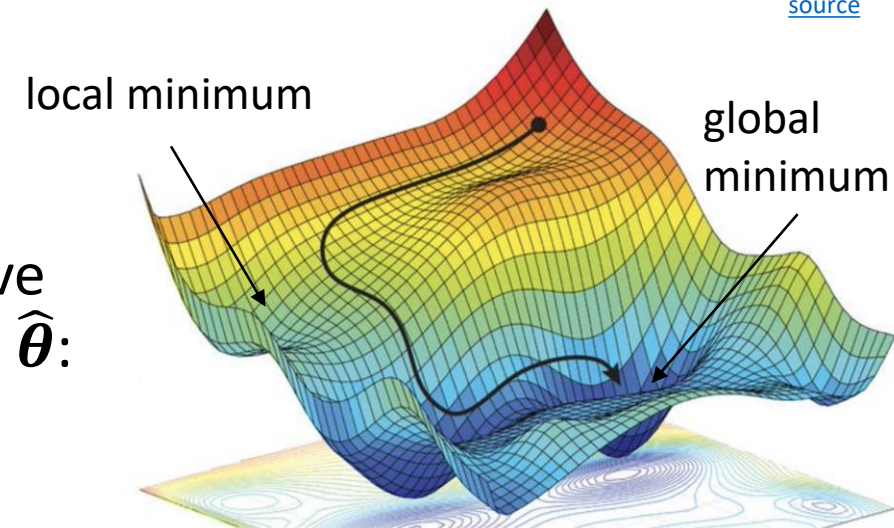
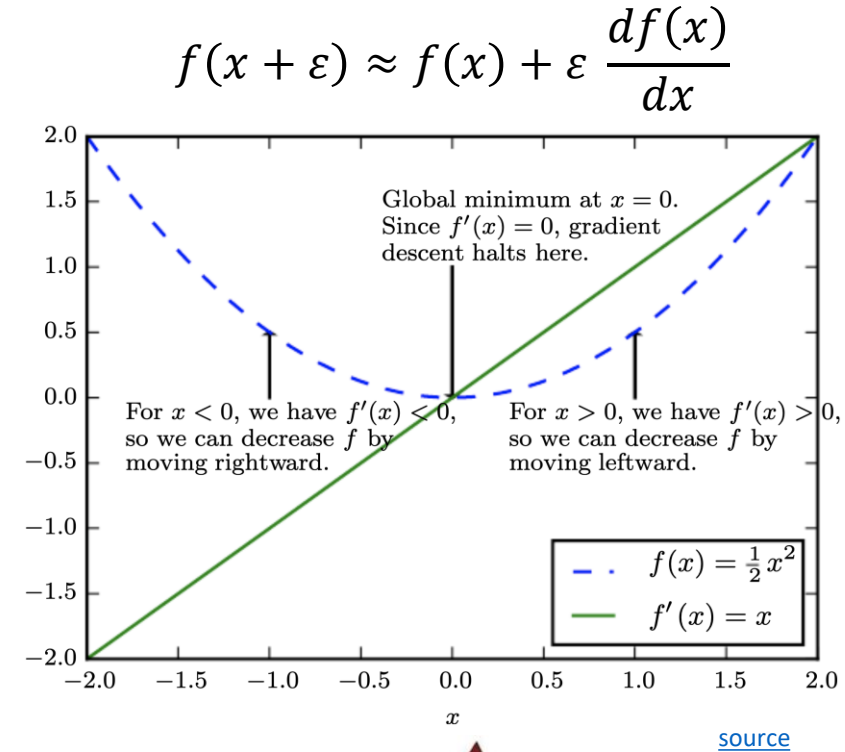
most popular choice: gradient descent

decreasing  $J$  by iteratively moving in direction of negative gradient (steepest descent) with respect to input vector  $\hat{\theta}$ :

$$\hat{\theta} \leftarrow \hat{\theta} - \eta \nabla_{\hat{\theta}} J(\hat{\theta})$$

step size  
(learning rate)

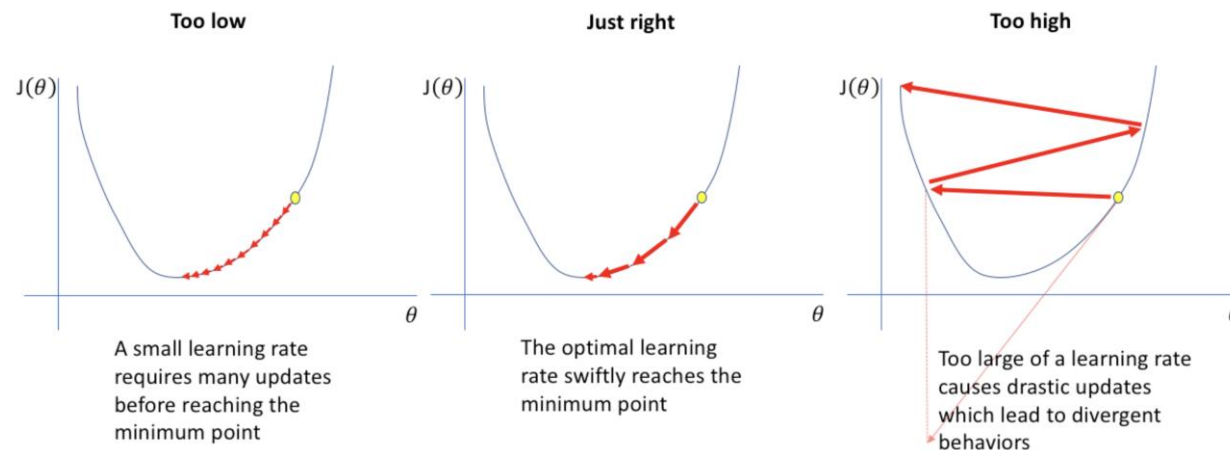
vector containing all partial derivatives



# Learning Rate

learning rate  $\eta$  corresponds to positive scalar value (hyperparameter)

- often set to small constant
- can be optimized via line search: evaluate  $J\left(\hat{\theta} - \eta \nabla_{\hat{\theta}} J(\hat{\theta})\right)$  for several values of  $\eta$  and choose  $\eta$  resulting in smallest value of objective function
- can be varied from iteration to iteration (e.g., via some heuristic): smaller steps closer to the minimum (learning rate schedule/decay)



# Batch and Stochastic Gradient Descent

in usual gradient descent: parameters  $\hat{\theta}$  (and in turn objective function  $J$ ) updated after full training epoch (one sweep through entire training data set)  $\rightarrow$  batch learning

consider:  $J(\hat{\theta}) = \frac{1}{n} \sum_{i=1}^n J_i(\hat{\theta})$  ( $J_i$  corresponds to loss function)

stochastic gradient descent (SGD) updates after each training example (gradient of  $J(\hat{\theta})$  approximated with gradient of single loss):  $\hat{\theta} \leftarrow \hat{\theta} - \eta \nabla_{\hat{\theta}} J_i(\hat{\theta})$

$\rightarrow$  online learning

- shuffling of samples after training epoch to prevent cycles
- adaptive learning rate to improve convergence

# Mini Batches

Compared to its batch mode, stochastic gradient descent helps to avoid local minima (more robust) but is computationally expensive (less efficient).

→ mini-batch SGD as compromise:

splitting training data set into small batches used to calculate model error and update parameters

at the cost of one additional hyperparameter specifying mini-batch size




# Gradient Descent with Momentum

goal: improve optimization process by avoiding gradients bouncing around the search space (dampening oscillations), accelerate in direction of minima

algorithm:

- estimate gradient  $\mathbf{g}$
- compute velocity update:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \mathbf{g}$
- apply parameter update:  $\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} + \mathbf{v}$

hyperparameter ( $0 < \alpha < 1$ ) specifying exponential decay (like learning rate  $\eta$ ,  $\alpha$  may be adapted over course of optimization)



velocity: direction and speed at which the parameters move through parameter space, set to an exponentially decaying average of the past negative gradients (continues to move in its direction)

# Bayesian Methods

# Bayesian Statistics

frequentist perspective:

- true model parameters  $\theta$  fixed (but unknown  $\rightarrow$  uncertain point estimates  $\hat{\theta}$ )
- data set from random sampling

Bayesian perspective:

- true model parameters  $\theta$  random (estimation of full distributions over parameters)
- data not random (observed)

# Bayes Theorem

$H$  : hypothesis (model)

$E$  : evidence (data)

likelihood function (derived  
from statistical model)

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

posterior probability  
(belief updated with  
observed evidence)

new data (same  
for all hypotheses  
→ uninteresting)

prior probability  
(belief in  $H$   
before evidence)

# Bayes Theorem for Parameter Estimation

data likelihood:  
model (function of  $\theta$  with data fixed), e.g.,  
for linear regression:  $\mathcal{N}(y; \alpha + \mathbf{x} \boldsymbol{\beta}, \sigma^2)$

$$p(\theta|y, \mathbf{x}) \propto p(y|\mathbf{x}, \theta) \cdot p(\theta)$$

posterior:  
probability distribution reflecting  
the effect of data on prior belief  
about parameters (increasing  
certainty with new evidence)

to be compared  
to frequentist  
point estimates  $\hat{\theta}$

prior (e.g., broad Gaussian):  
probability distribution reflecting  
initial belief about uncertain  
parameters (helps to reduce  
variance with few data)

# Bayesian Prediction

$$p(y_{n+1} | \mathbf{x}_{n+1}, (y_1, \mathbf{x}_1), \dots, (y_n, \mathbf{x}_n)) = \int \underbrace{p(y_{n+1} | \mathbf{x}_{n+1}, \boldsymbol{\theta})}_{\text{likelihood}} \underbrace{p(\boldsymbol{\theta} | (y_1, \mathbf{x}_1), \dots, (y_n, \mathbf{x}_n))}_{\text{latest posterior}} d\boldsymbol{\theta}$$

prediction of full probability distribution instead of frequentist point estimate(s) plugged into model distribution ( $p(y|\mathbf{x}) = \text{PDF}(y; \hat{f}(\mathbf{x}), \dots)$ )

drawback: high computational costs due to integral over parameter distribution (especially for large training data sets)

# Alternatives for PDF Predictions

**quantile regression:** estimate quantile  $\tau$  of distribution instead of conditional mean by minimizing pinball loss

$$(1 - \tau) \sum_{y_i < \hat{q}_i} (\hat{q}_i - y_i) + \tau \sum_{y_i \geq \hat{q}_i} (y_i - \hat{q}_i)$$

instead of squared error loss (choice of loss function defines point estimate)

- possible with various ML methods, including neural networks and tree-based methods (like random forests or gradient boosting)
- subsequently approximate full probability distribution from a few predicted quantiles by fitting (assumed distribution or spline) or [quantile-parameterized distributions](#)

**PDF assumption for model and estimation of its parameters:** assume, e.g., Gaussian or negative binomial and estimate mean and variance (by means of different ML models or corresponding likelihood)

# Common Core of Statistical Learning

Most ML algorithms can be described by the general recipe of combining models, costs, and optimization methods.

including non-linear models:

- neural networks: backpropagation
- support-vector machines: minimizing hinge loss (soft-margin SVM)
- decision trees: minimizing impurity functions (mean squared error for regression, (kind of) Kullback-Leibler divergence for classification)

and even

- unsupervised learning: self-supervised, PCA by maximum variance directions
- reinforcement learning: express rewards in loss functions



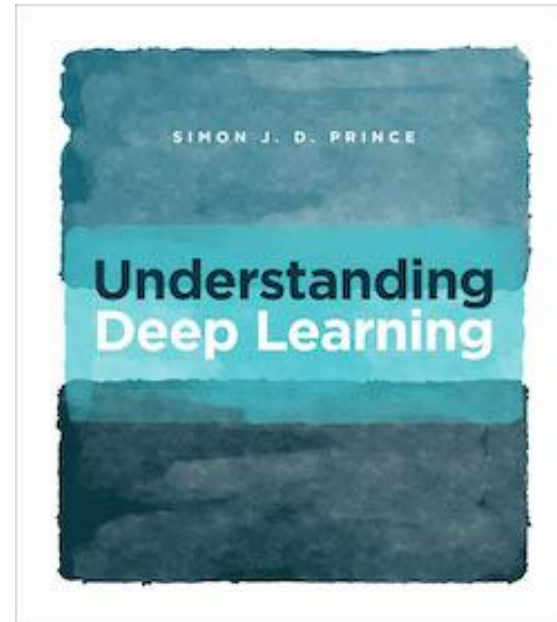
# DEEP LEARNING

Ian Goodfellow, Yoshua Bengio,  
and Aaron Courville

## Literature

---

not only deep learning, but also a nice ML  
introduction: <https://www.deeplearningbook.org/>



following on from the above,  
also covering newer topics:  
[udlbook](#)

# Overcome our Mathematical Limitations

evolution provided us with moderate math skills

AI/ML to the rescue:

- [recognition of mathematical structures and patterns](#)
- algorithm discovery: [AlphaTensor](#)
- symbolic regression ([e.g., for physics](#))