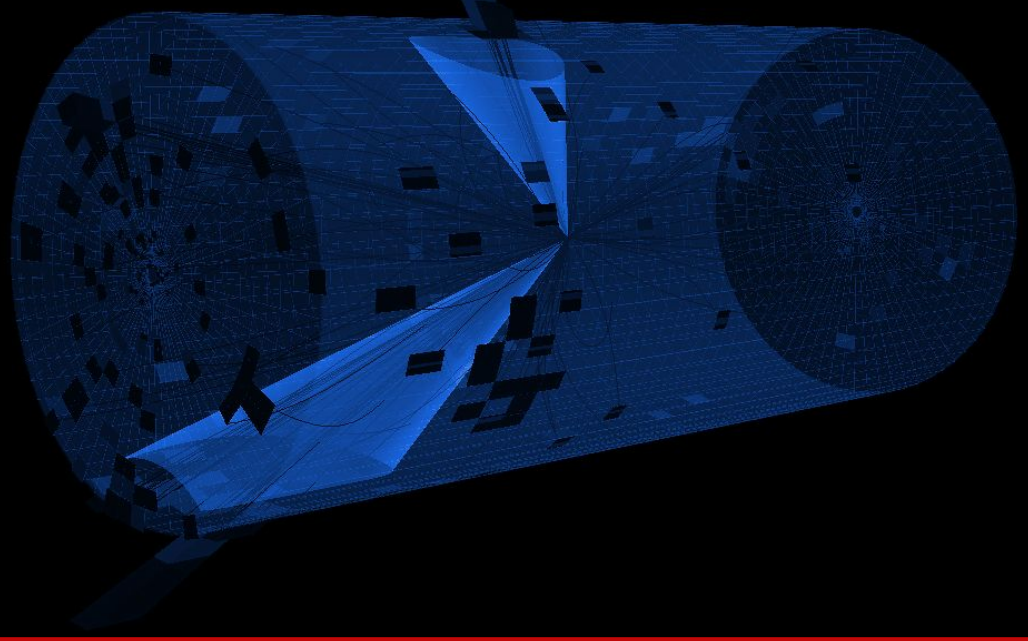
Machine Learning in High Energy Physics



https://agenda.infn.it/event/28573/timetable/#20211220

Cristiano Fanelli

cfanelli@mit.edu

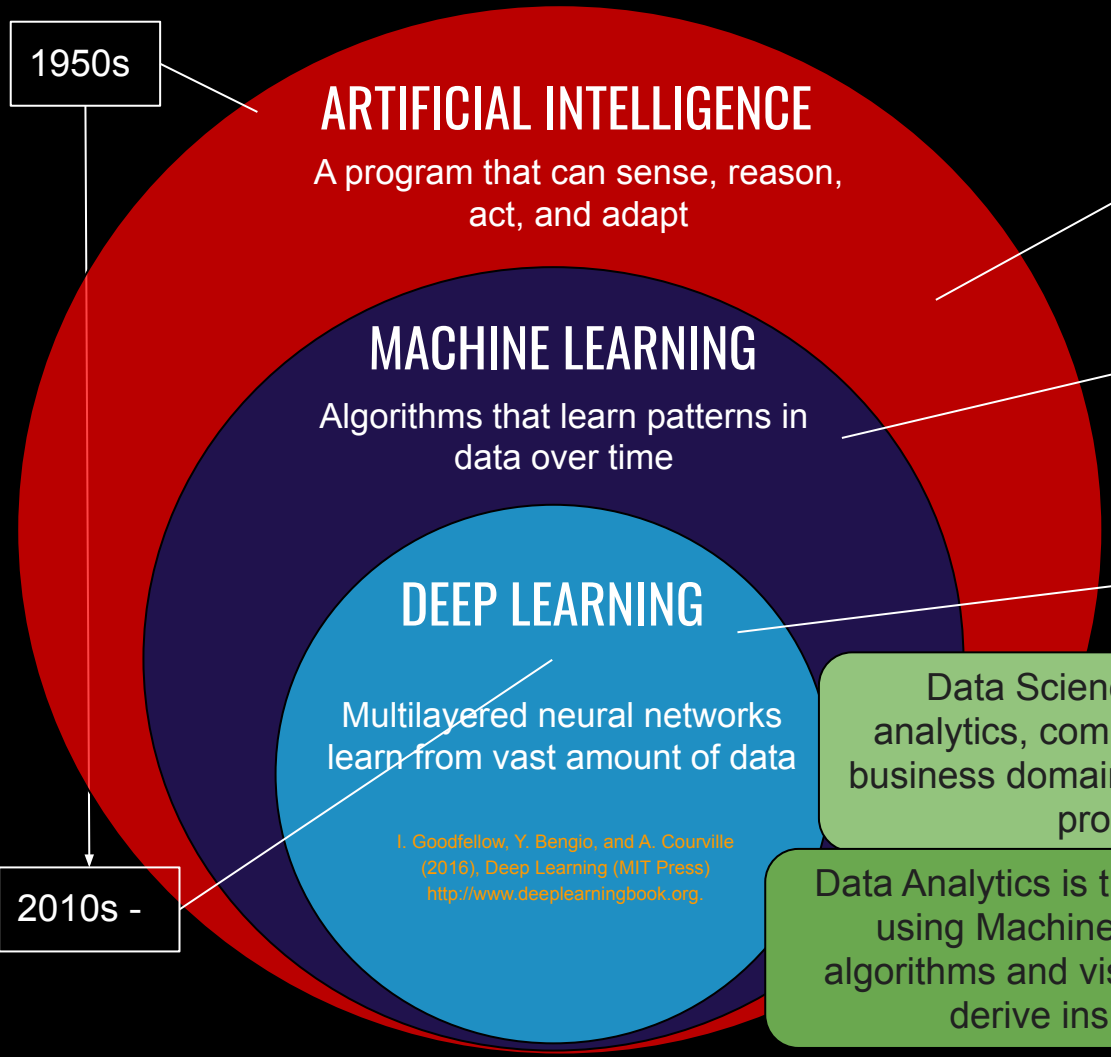
Introduction and Scope

- This is a 2 days course for beginners dipping toes into ML for the first time!
  - Goal: introduce basic concepts (a small subset of topics)
    - Starting point for experimenting and playing with ideas
    - Build upon physicist language/background
    - Pre-requisites: linear algebra, multivariate calculus, probability theory, MC-methods, (some) python…
  - Lectures are accompanied by simple hands-on sessions and real-world examples
  - References

- Course indico: https://agenda.infn.it/event/28573/
- Lectures, exercises: https://github.com/cfteach/ml4hep
- A high bias, low-variance introduction to Machine Learning [hblvi2ML]: https://arxiv.org/abs/1803.08823

Which these slides largely draw from.

- "A Living Review of Machine Learning for Particle Physics": https://iml-wg.github.io/HEPML-LivingReview/ arXiv:2102.02770 (2021)
- "Artificial Intelligence and Machine Learning in Nuclear Physics": arXiv:2112.02309 (2021)



AI

```
Bayesian methods
Genetic Algorithms,
Rules-based system,
...
```

ML

```
Random Forest,
Support Vector Machines,
XGBoost,
...
```

DL

```
CNN, RNN, GAN, ...
```

Data Science blends data analytics, computer science and business domain expertise to solve problems.

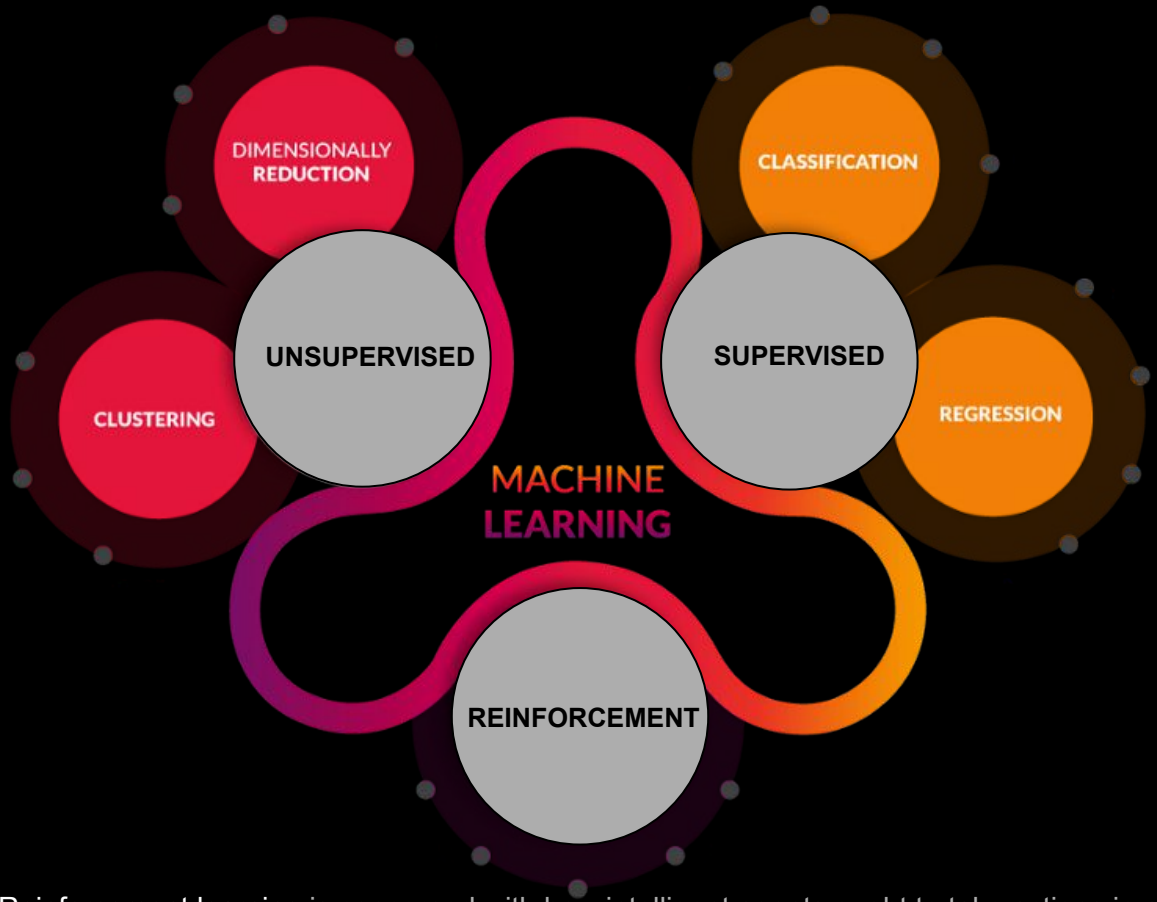
DS

Data Analytics is the practice of using Machine Learning algorithms and visualization to derive insights.

Use of AI/ML/DL in HEP becoming ubiquitous

ML

Unsupervised learning is a type of machine learning in which the algorithm is not provided with any pre-assigned labels or scores for the training data. Unsupervised learning algorithms must first self-discover any naturally occurring patterns in that training data set.

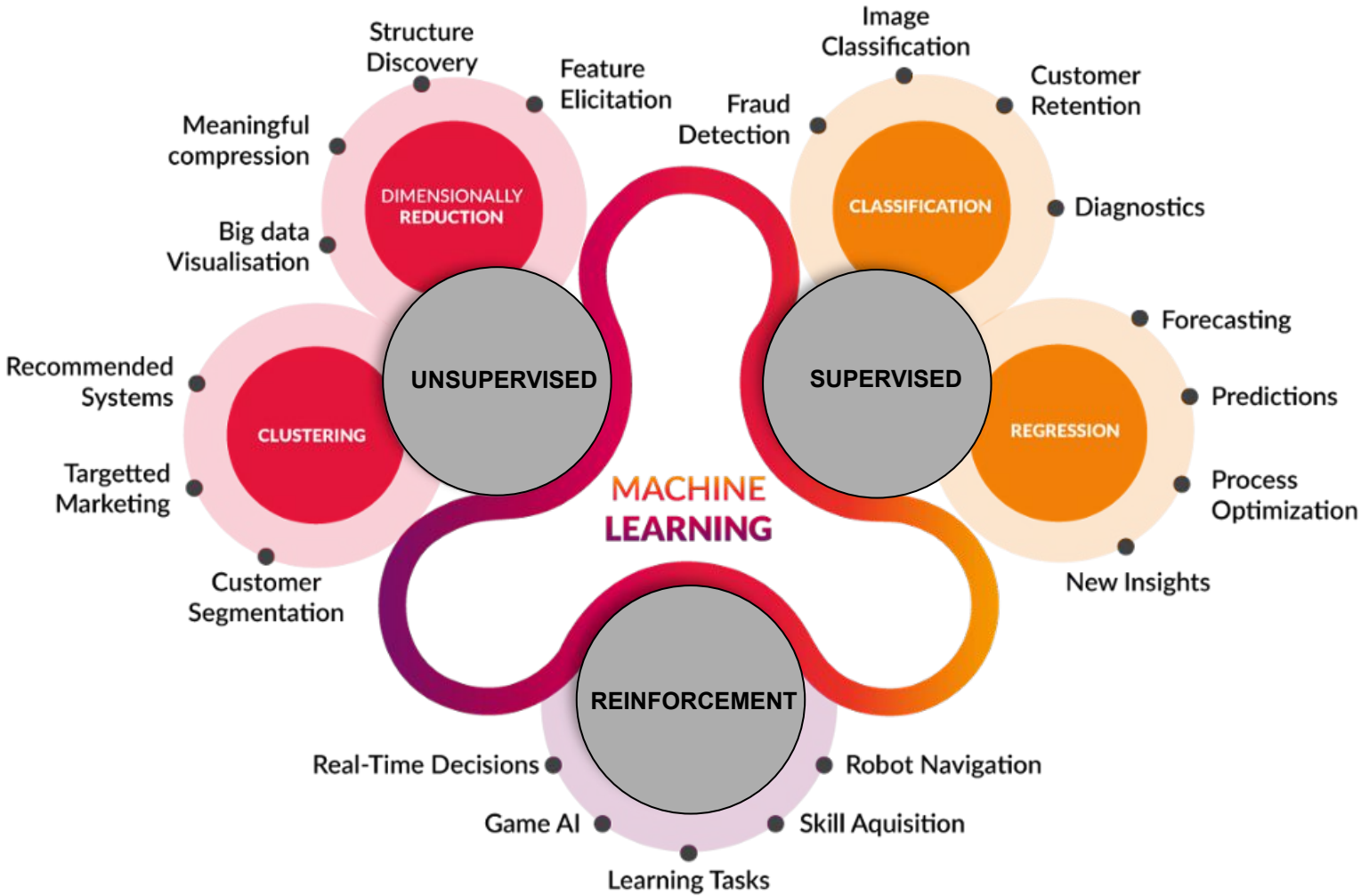


Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples

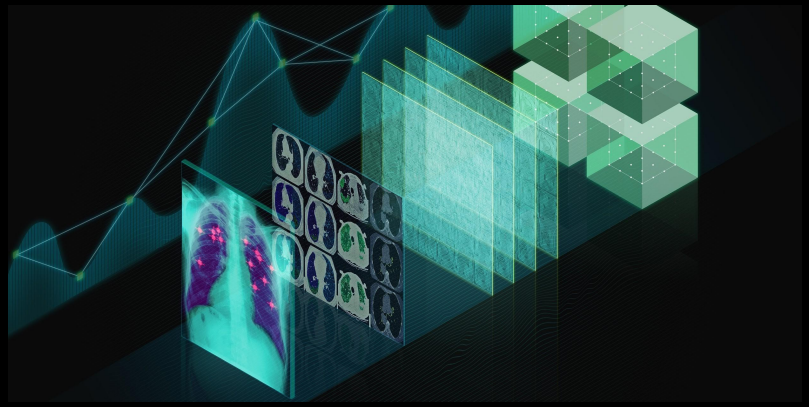
Reinforcement learning is concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward and make informed choices.

R. S. Sutton, and A. G Barto (1998), Reinforcement learning: An introduction, Vol. 1 (MIT press Cambridge)

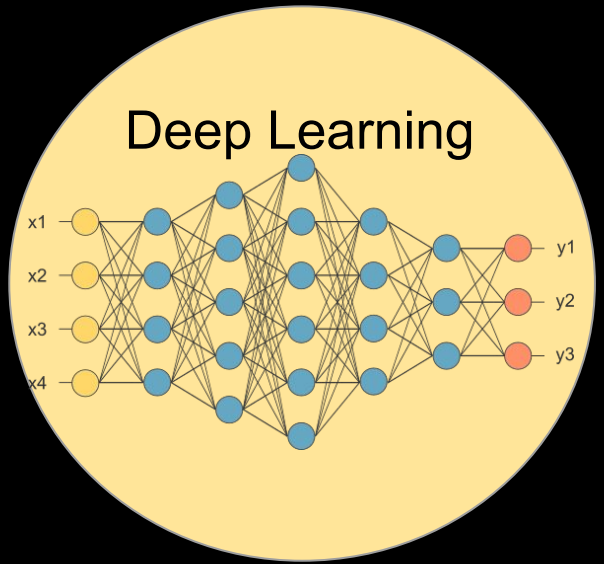
ML

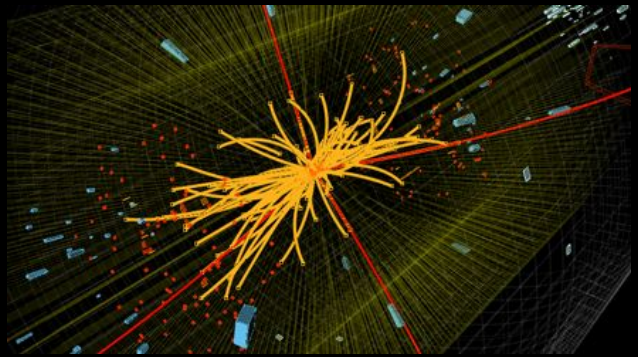

MACHINE LEARNING
UNSUPERVISED
SUPERVISED
REINFORCEMENT
DIMENSIONALLY REDUCTION
CLUSTERING
CLASSIFICATION
REGRESSION
Structure Discovery
Feature Elicitation
Meaningful compression
Big data Visualisation
Image Classification
Customer Retention
Fraud Detection
Diagnostics
Recommended Systems
Targetted Marketing
Customer Segmentation
Forecasting
Predictions
Process Optimization
New Insights
Real-Time Decisions
Robot Navigation
Game AI
Skill Aquisition
Learning Tasks

DL

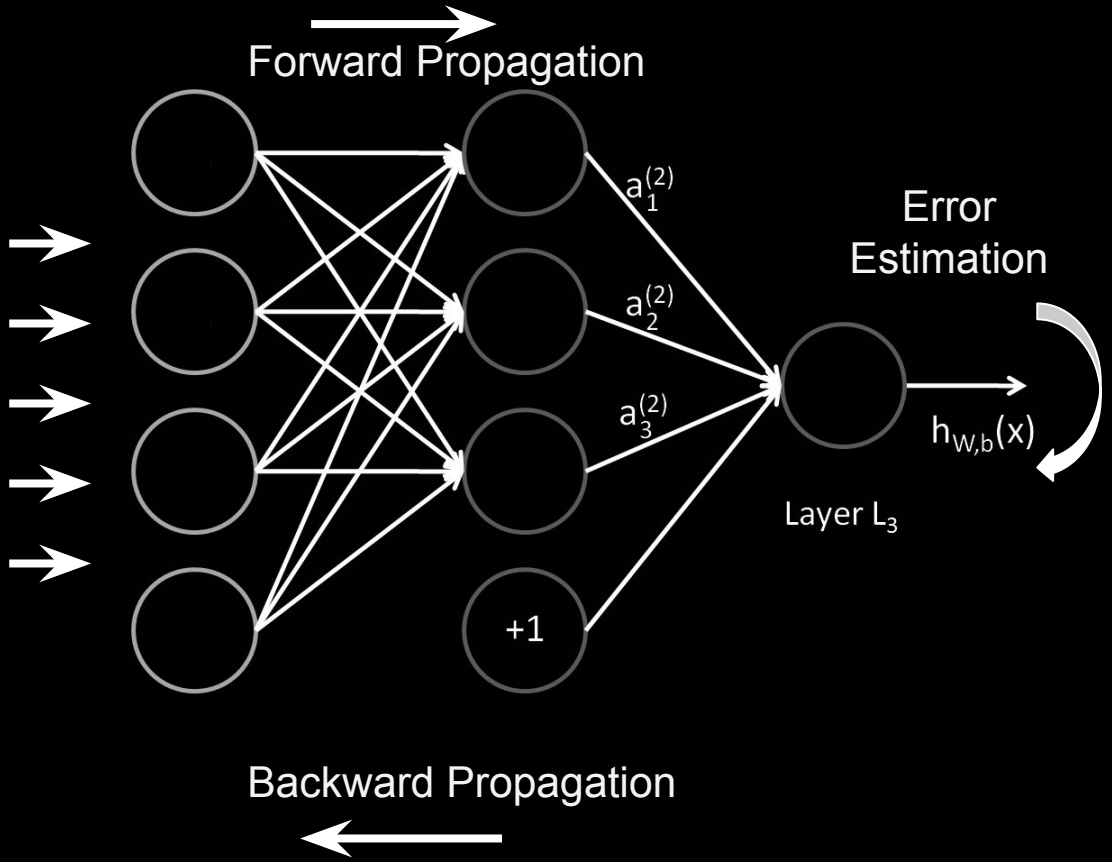



LEFT REARWARD VEHICLE CAMERA
MEDIUM RANGE VEHICLE CAMERA


Deep Learning
x1
x2
x3
x4
y1
y2
y3



DL



- The real magic about NN is the result of an optimization technique: back-propagation (how a NN works to improve its output over time)
- DL (more hidden layers) nets are good in learning non-linear functions (heavy processing tasks)
- Based on old school NN revitalized by augmented capabilities (e.g. GPU) and a plethora of new architectures (RNN, CNN, autoencoders, GAN, etc.)

Unsupervised/Supervised/Reinforcement

NIPS 2016: “If intelligence is a **cake**,
the bulk of the **cake** is **unsupervised** learning,
the icing on the **cake** is supervised learning,
and the cherry on the **cake** is reinforcement learning“

LeCun, Turing award 2018
VP and Chief AI Scientist, Facebook

Mnih et al, 1312.5602
Nature, 518.7540 (2015)

Google DeepMind

Deep Q-learning
playing Atari Breakout

Timetable/Overview

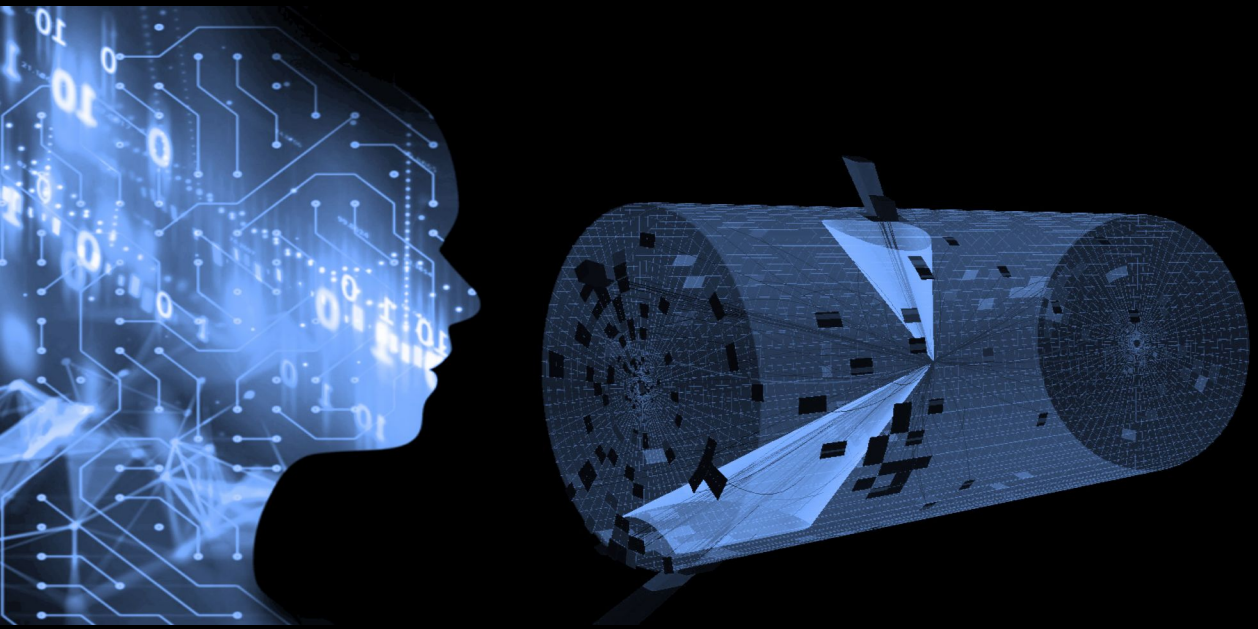
timetable can change a bit as needed

Structure and Disclaimer

- ML is broad and interdisciplinary and draw on ideas from many fields
- In these 2 days we can only cover very few ideas
- In some cases I will just give the gist of the theoretical foundation behind these concepts but I won't have time to go into the details.
- We will try to provide/point to tools to start using ML in practical problems
  - Jupyter notebooks
  - Numerous of great software packages
- Datasets:
  - SUSY dataset (5M MC samples)
  - Higgs dataset (11M)
  - Artificially created datasets

Following [hblvi2ML] we will cover:

- Regression
- "Decision trees"
- Clustering
- (Intro to) Deep Learning
- Optimization



ML4HEP

Notebooks are available at [hblvi2ml].
In these 2 days we will cover explicitly only few exercises, spanning supervised classification tasks, unsupervised approaches for clustering and optimization.

Jupyter Notebooks

- Work with python3
- Install jupyter in a virtual environment
  - `python3 -m venv env_ml4hep`
  - `source ./env_ml4hep/bin/activate`
  - `pip install jupyter` (you can also install with anaconda if you are more familiar)
- How to change/add the kernel of a jupyter notebook in a virtual environment?
  - `ipython kernel install --name "env_ml4hep" --user`
- To start notebook (from terminal, virtual environment):
  - `jupyter notebook`

We will also use colab later

What is ML?

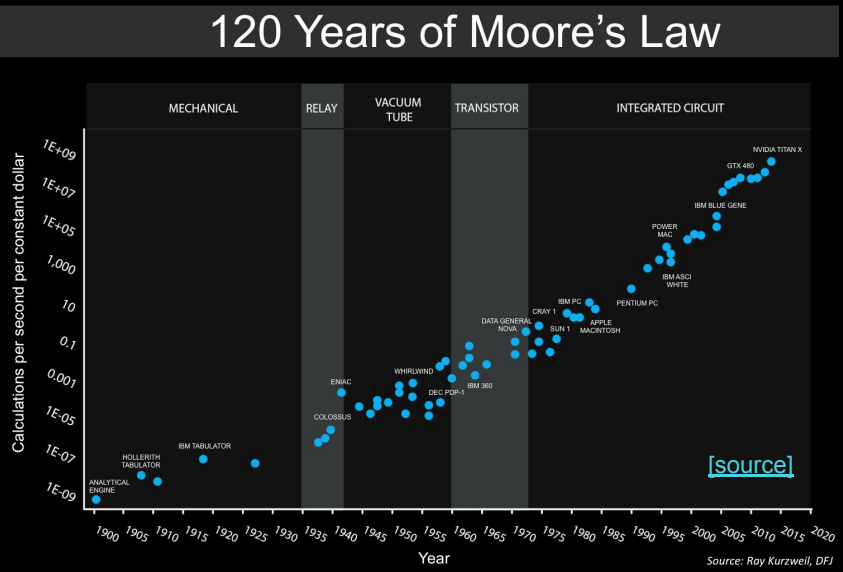
- ML, data science and statistics are fields describing how to learn from and make predictions about data.
- Techniques in ML tend to be more focused on predictions than estimation. Methods from ML tend to be applied to more complex high-dimensional problems.
- Estimation and prediction problems can be cast into a common conceptual framework related to some parameters $\boldsymbol{\theta}$ of a model $p(\mathbf{x}|\boldsymbol{\theta})$ that describes the probability of observing **x given θ**
- Fitting the model involves finding $\boldsymbol{\theta}^*$ providing best explanation for data. If fitting refers to the method of least squares, the estimated parameters maximize $\boldsymbol{\theta}^* = \mathbf{argmax}_{\boldsymbol{\theta}}\{p(\mathbf{x}|\boldsymbol{\theta})\}$.
- Although the goals of estimation and prediction are related they often lead to different approaches:
  - Estimation problems are concerned with the accuracy of θ^*.
  - Prediction problems are concerned with the ability of the model to predict new observations. We will focus on prediction.

Problems in ML typically involve inference about complex systems where we do not know the exact form of the mathematical model that describes the system.

MacKay, David JC (2003), Information theory, inference and learning algorithms (Cambridge university press)

Why is ML ubiquitous

- Last three decades unprecedented ability to generate and analyze large data sets: big data revolution spurred by exponential increase of computing power and memory
- Computations that were unthinkable can now be routinely performed on laptops.
- Specialized computing machines (e.g., GPU-based) are continuing this trend towards cheap, large scale computation.

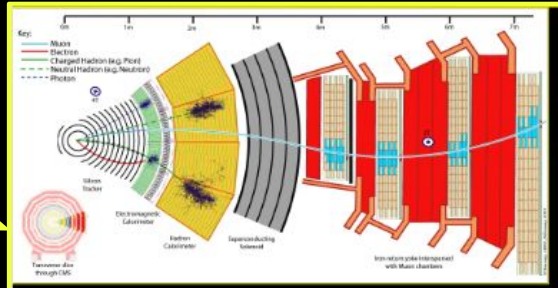


- Physicists are uniquely situated to benefit from and contribute to ML. Many core concepts in ML have their origin in physics: MC methods, variational methods, simulated annealing, energy based models etc
- HEP has been at the forefront of using big data.
  - LHC experiments produce data at a rate of 1PB/sec; after data reduction (zero suppression, custom compression algorithms) 50TB/s resulting in as much data every hour as Facebook collects globally in a year
  - At LHCb, 70% of all data retained are classified by ML and all charged particle tracks are vetted by NNs.

A. Radovic, et al. "Machine learning at the energy and intensity frontiers of particle physics." *Nature* 560.7716 (2018): 41-48.

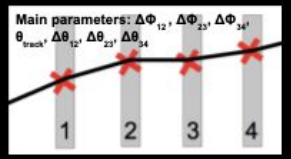
Why is ML ubiquitous

HW Trigger Muon ID @CMS



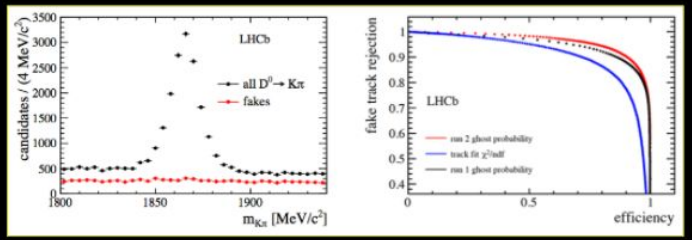
- L1 trigger responsible for selecting 100k/s interesting events out of the 40M/s
- Endcap Muon Track Finder (EMTF)
  - Needs to operate fast (~ 500 ns)
  - No tracker info available, only muon chambers
- Want Machine Learning to do the p_T assignment (implemented on FPGA)

- transverse momentum (p_T) is assigned based on curvature
- The Endcap Muon Track Finder (EMTF) needs to process hits and assign a momentum
- Interesting muons have large p_T

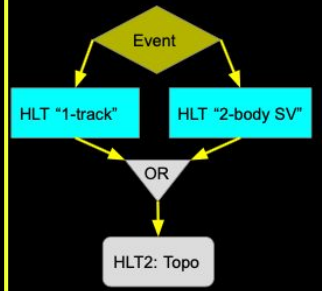


Ghost tracks killer, HLT2 Topological Trigger @LHCb

Fake-track (ghost) killing DNN based on 22 features, most important are hit multiplicities and track-segment chi2 values from tracking subsystems. Significantly reduces the rate of events selected in the HLT1.



Run in the trigger on all tracks (it must be very fast). Use of custom activation function and highly-optimized C++ implementation.



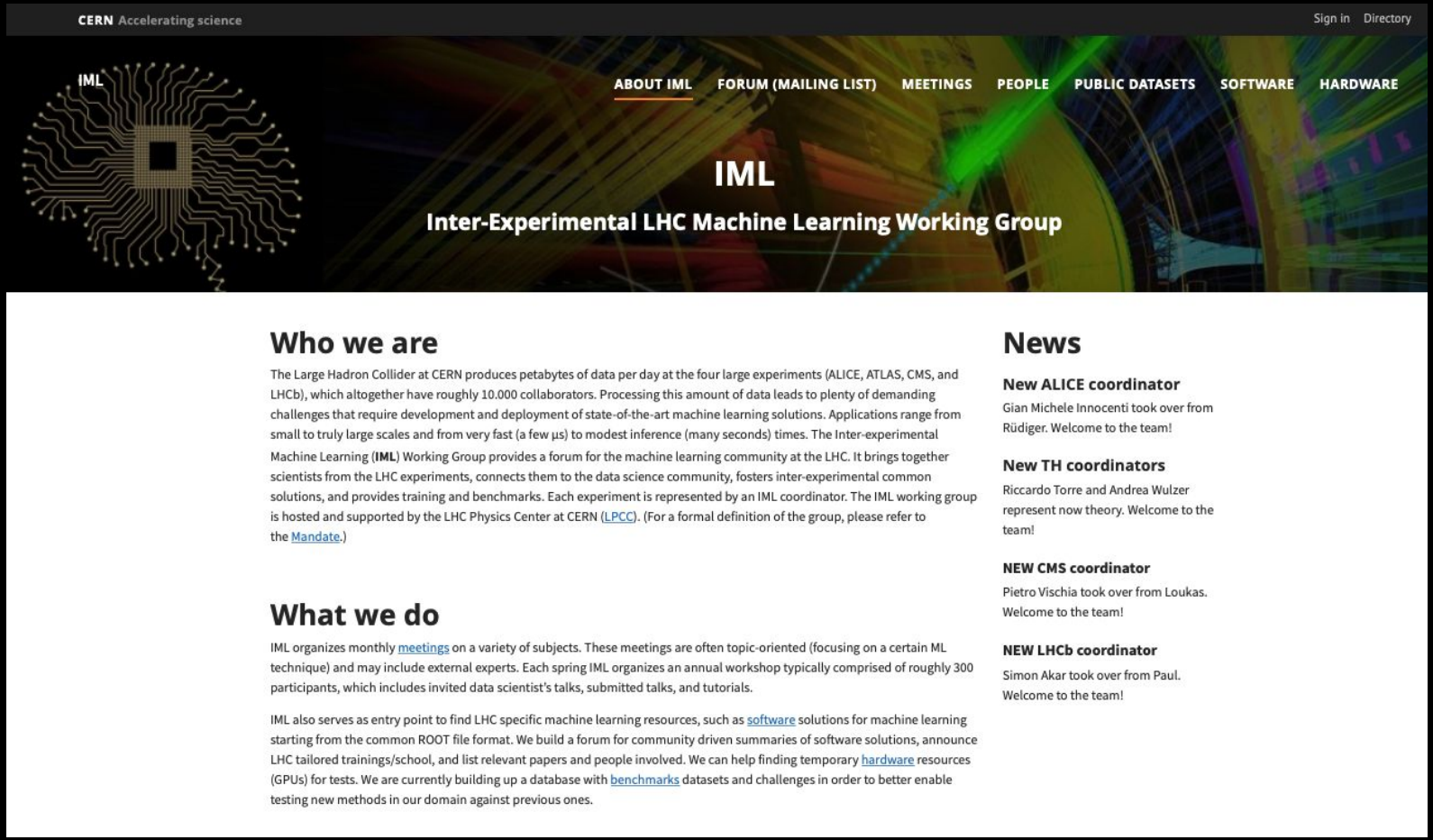
- The main b-physics trigger used by LHCb. Selects vertices which are:
  - ➢ Detached from the primary pp
  - ➢ Compatible with coming from a b-hadron decay

Consists of:

- An SV algorithm that considers 2, 3, and 4-track vertices (seeded by HLT1 ML selections).
- The ML uses a list of features: n(tracks), corrected mass, vertex χ^2, scalar track p_T sum, flight distance χ^2, pseudorapidity (PV-SV), min(track p_T), n(small IP tracks), IP χ^2, n(very b-like tracks).
- All features are discretized in the ML for stability, robustness, etc. This allows to control growth of DT.

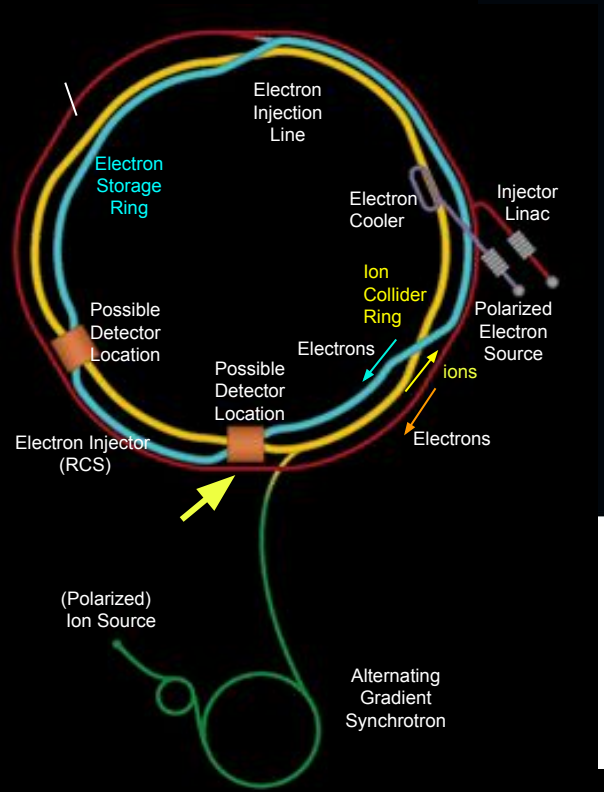
IML

https://iml.web.cern.ch



AI4EIC

https://eic.ai



SCOPE

Artificial Intelligence for the Electron Ion Collider

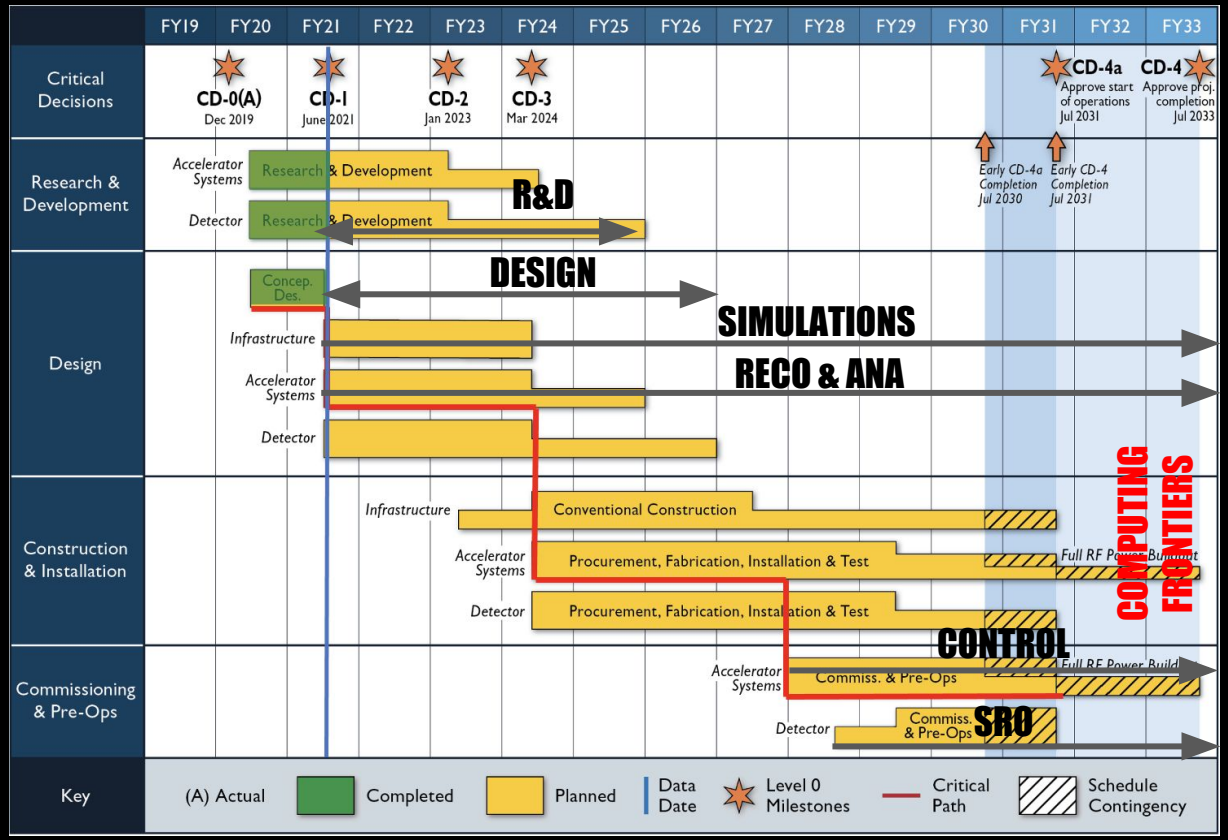
AI will be an essential part of future experiments like the Electron Ion Collider, a new $2B high-luminosity facility capable to collide high-energy electron beams with high-energy proton and ion beams that will be built at BNL in approximately 10 years from now to unlock the secrets of the "glue" that binds the building blocks of visible matter in the universe.

AI can provide new insights and discoveries from both experimental and computational data produced at user facilities.

This website includes AI activities related to EIC that will characterize the different phases of its realization.

EIC Schedule and Role of AI

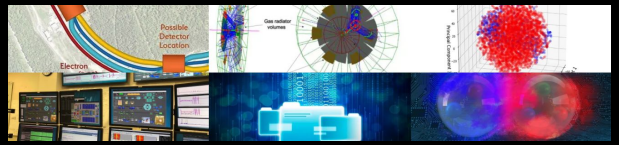
https://eic.ai

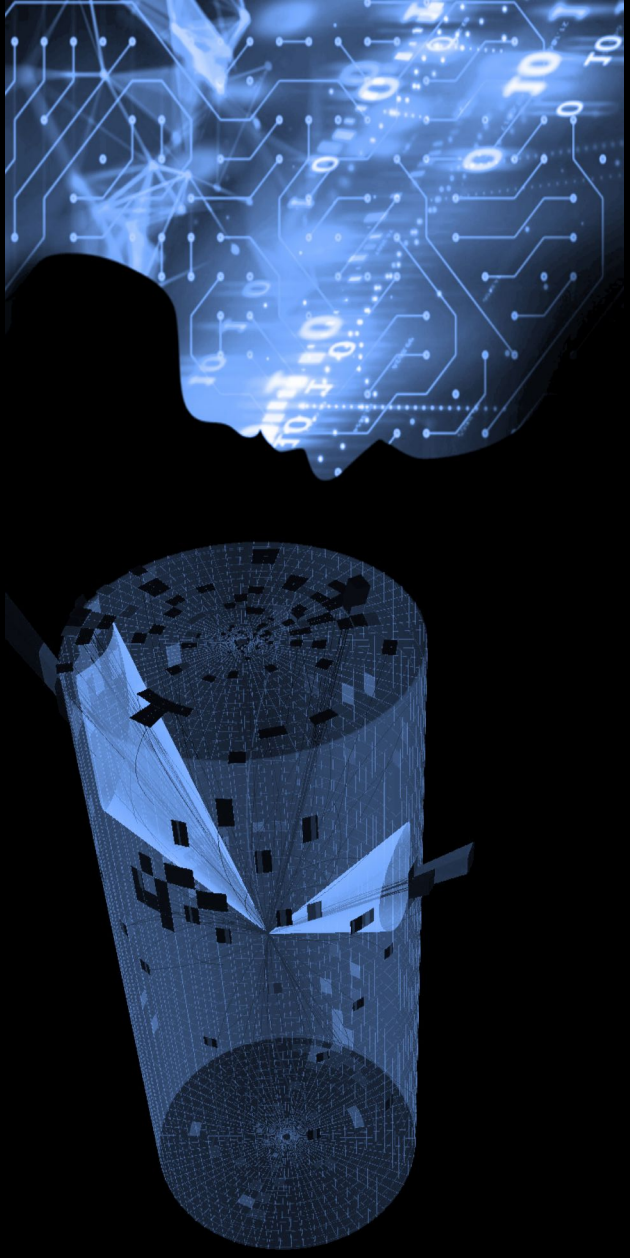


AI4EIC

AI4EIC Working Groups

- Design
- Simulations
- Data Reco & Analysis
- Controls
- Detector SRO
- Computing frontiers
- Theory and phenomenology





Why is ML difficult

Typical Problem

- Ingredients:
  - Dataset D(**X**,**y**): **X** matrix of independent variables, **y** dependent variables
  - Model f(**X**;**θ**) where f: **X**→**y** is a function of the parameters **θ**
  - Cost function C(**y**,f(**X**;**θ**)) to judge how well the model performs on the observations **y**
- The model is fit to find **θ that** minimize the cost function; commonly used cost is squared error (method of least squares)
- Recipe for prediction problems:

1. Randomly divide the dataset D into mutually exclusive D_{train} (typically 90%) and D_{test} (10%)
2. Model is fit on training data $\boldsymbol{\theta^* = \text{argmin}_\theta\{C(y_{train}, f(X_{train};\theta))\}}$
3. The performance of the model is evaluated on $C(y_{test}, f(X_{test};\theta^*))$

- Splitting data provides an unbiased estimate for the predictive performance (known as cross-validation)
- In-sample error: $E_{in} = C(y_{train}, f(X_{train};\theta))$; out-of-sample error: $E_{out} = C(y_{test}, f(X_{test};\theta))$
- E_{out} is always larger than E_{in} , $E_{out} \geq E_{in}$

Bias/Variance

- The model that provides the best explanation for the current dataset will probably not provide the best explanation for future datasets
- The discrepancy between E_{in} , E_{out} grows with the complexity of our data and of our model (increased model parameters, high dimensional space, curse of dimensionality)
- For these reasons (and for complicated models), predicting and fitting can be different things. Need to pay attention to out-of-sample performance. Fitting existing data well is fundamentally different from making predictions about new data.
- Let's see this starting from simple one-dimensional problem: we want to fit data with polynomials of different orders.
- Our ability to predict depends on the number of data points, the noise in the data, and our prior knowledge about the system

Bellman, Richard. *Dynamic programming*. Vol. 295. RAND CORP SANTA MONICA CA, 1956.

Fitting vs Predicting

- Consider probabilistic process that assigns a label y_i to an observation x_i. Data are generated from
  - $y_i = f(x_i) + \eta_i$ where η_i is a gaussian uncorrelated noise variable such that $\langle \eta_i \rangle = 0$ and $\langle \eta_i \eta_j \rangle = \delta_{ij} \sigma^2$
- To make predictions we consider a family of functions $f_\alpha(x;\boldsymbol{\theta}_\alpha)$ (different model complexity):
  - Polynomial of order 1: $f_1(x;\boldsymbol{\theta}_1) \rightarrow$ 2 pars
  - Polynomial of order 3: $f_3(x;\boldsymbol{\theta}_3) \rightarrow$ 4 pars
  - Polynomial of order 10: $f_{10}(x;\boldsymbol{\theta}_{10}) \rightarrow$ 11 pars
- Using a more complex model class may give us better prediction power but only if we have a large enough sample size to accurately learn the model parameters
- To learn $\boldsymbol{\theta}_\alpha$ we use a training dataset and test the effectiveness of the model on the test dataset
- Obviously the more data and less noise we have the better the predictions are

Fitting vs Predicting

- We utilize a test interval [0,1.2] which is larger than the training interval [0,1.0]
- Data sampled from
  - f(x) = 2x
  - $f(x) = 2x - 10x^5 + 15x^{10}$
- In absence of noise, even with a small training set (N_{train}=10<N_{test}=20) the model class that generated the data provides the best fit and also the best out-of-the sample prediction.

Training Data

Test Data

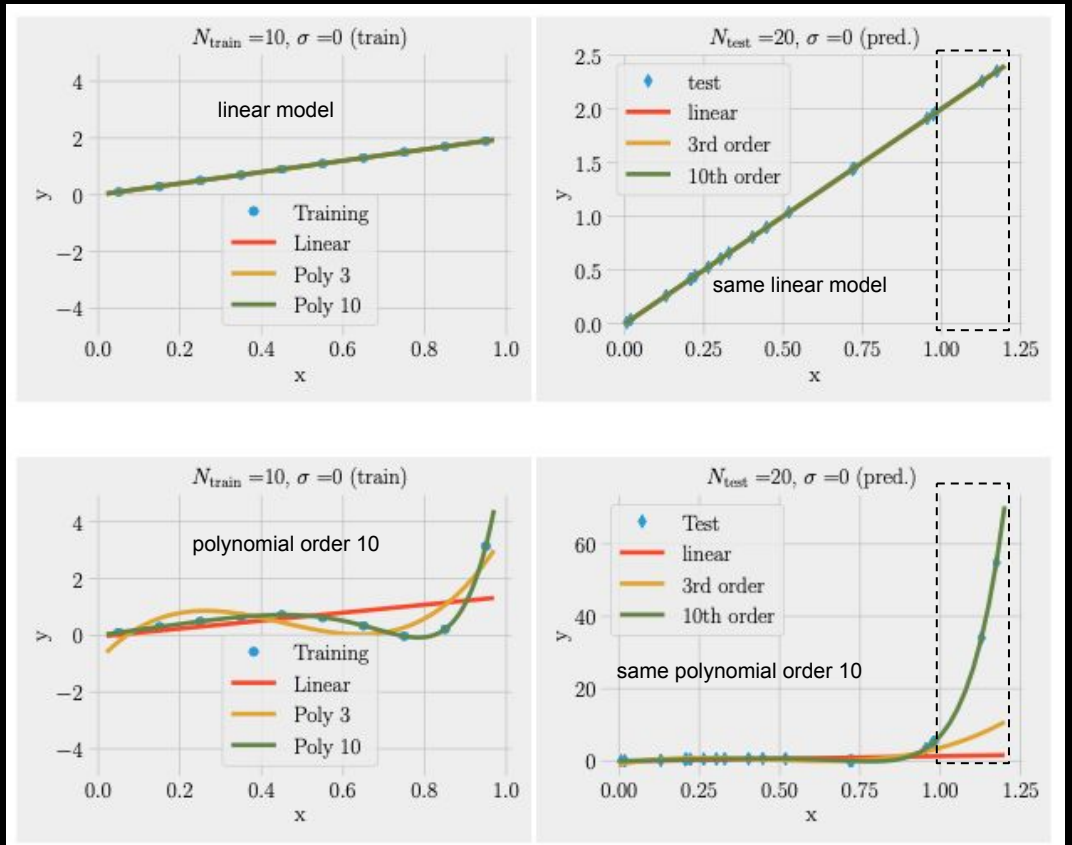


FIG. 1 **Fitting versus predicting for noiseless data.** $N_{train} = 10$ points in the range $x \in [0, 1]$ were generated from a linear model (top) or tenth-order polynomial (bottom). This data was fit using three model classes: linear models (red), all polynomials of order 3 (yellow), all polynomials of order 10 (green) and used to make prediction on $N_{test} = 20$ new data points with $x_{test} \in [0, 1.2]$ (shown on right). Notice that in the absence of noise ($\sigma = 0$), given enough data points that fitting and predicting are identical.

Fitting vs Predicting

- We utilize a test interval [0,1.2] which is larger than the training interval [0,1.0]
- Data sampled from
  - f(x) = 2x
  - f(x) = 2x -10x^5 + 15x^{10}
- Noise =1; training set (N_{train}=100>N_{test}=20); even when the model class that generated the data is a 10 order polynomial, the linear and 3rd order polynomials give better out-of-sample predictions.
- At small sample sizes, noise can create fluctuations in the data that look like genuine patterns.

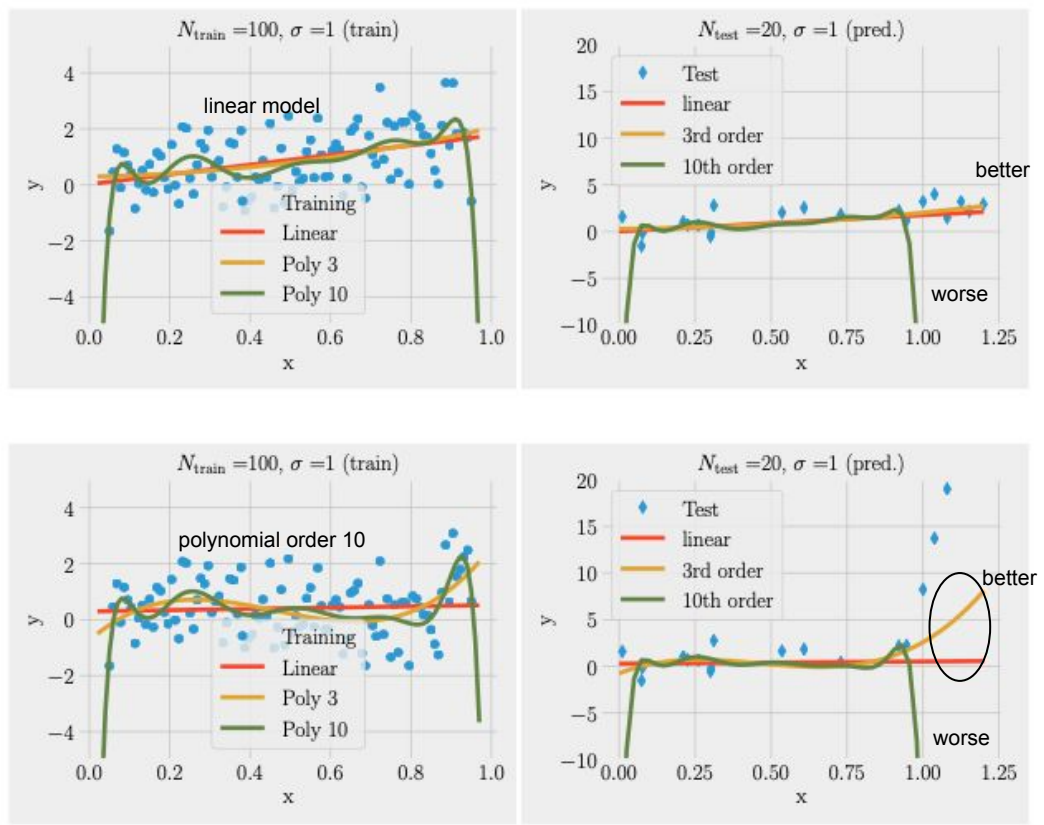


FIG. 2 **Fitting versus predicting for noisy data.** $N_{train} = 100$ noisy data points ($\sigma = 1$) in the range $x \in [0, 1]$ were generated from a linear model (top) or tenth-order polynomial (bottom). This data was fit using three model classes: linear models (red), all polynomials of order 3 (yellow), all polynomials of order 10 (green) and used to make prediction on $N_{test} = 20$ new data points with $x_{test} \in [0, 1.2]$(shown on right). Notice that even when the data was generated using a tenth order polynomial, the linear and third order polynomials give better out-of-sample predictions, especially beyond the x range over which the model was trained.

Bias/Variance Tradeoff

- We utilize a test interval [0,1.2] which is larger than the training interval [0,1.0]
- Data sampled from
  - f(x) = 2x
  - f(x) = 2x -10x^5 + 15x^{10}
- Noise =1; let's increase the training set to N_{train}=10^4

Training Data

Test Data

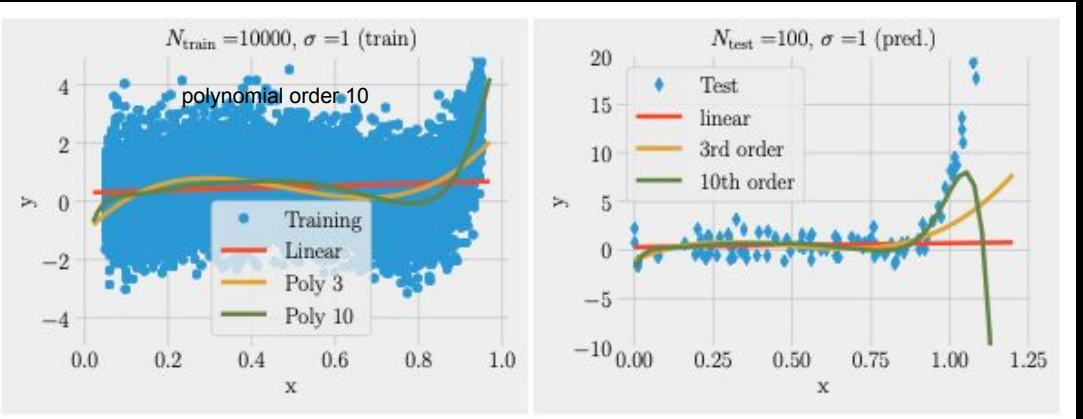


FIG. 3 **Fitting versus predicting for noisy data.** $N_{train} = 10^4$ noisy data points ($\sigma = 1$) in the range $x \in [0, 1]$ were generated from a tenth-order polynomial. This data was fit using three model classes: linear models (red), all polynomials of order 3 (yellow), all polynomials of order 10 (green) and used to make prediction on $N_{test} = 100$ new data points with $x_{test} \in [0, 1.2]$(shown on right). The tenth order polynomial gives good predictions but the model's predictive power quickly degrades beyond the training data range.

- The 10 polynomial model gives both the best fit ad the most predictive power over the entire range [0,1] and actually slightly beyond up to ~1.05, but then the predictive power quickly degrades
- This is our first experience with the bias-variance tradeoff: where the amount of data is limited, we often get better predictive performance by using a less expressive model (lower order polynomial)
  - The simpler model has more bias but is less dependent on the particular realization of the training set, i.e. has less variance.

ML is difficult

- Last one was a good example where we are good at interpolating but not at extrapolating.

- Fitting is not predicting.
  - Fitting existing data well is fundamentally different from making predictions about new data
- Using a complex model can result in overfitting
  - Better result on training data; when data size is small and the data are noise, this results in overfitting and degrade predictive performance
- For complex datasets and small training sets, simple models can be better at predicting than complex ones due to the bias-variance tradeoff
  - Even though the correct model (less bias) has better predictive performance for an infinite amount of training data, the training errors stemming from finite-size sampling (variance) can cause simpler models to outperform the more complex model
- It is difficult to generalize beyond what seen in the training dataset.

Statistical Learning Theory

- We summarize here the sense in which learning is possible with focus on supervised learning
- We begin from an unknown function y=f(x) and fix a *hypothesis set H* of all functions we want to consider
- f(x) produces a set of pairs (x_i, y_i), i=1,...,N, which serve as he observable data. Our goal is to select a function from the hypothesis set $h \in H$ that approximates f(x) as best as possible, such that $h \approx f$
- If that is possible we say we learnt f(x)
- But if the function f(x) can, in principle, take any value on unobserved inputs, how is it possible to learn in any meaningful sense?

The relationship between the in-sample error E_{in} and the out-of-sample (or generalization) error is the domain of statistical learning theory

Bias/Variance

Assumptions:
- E_{out} and E_{in} as a function of the size of training data
- We assume data come from a complicated distribution (so we won't exactly learn f(x))

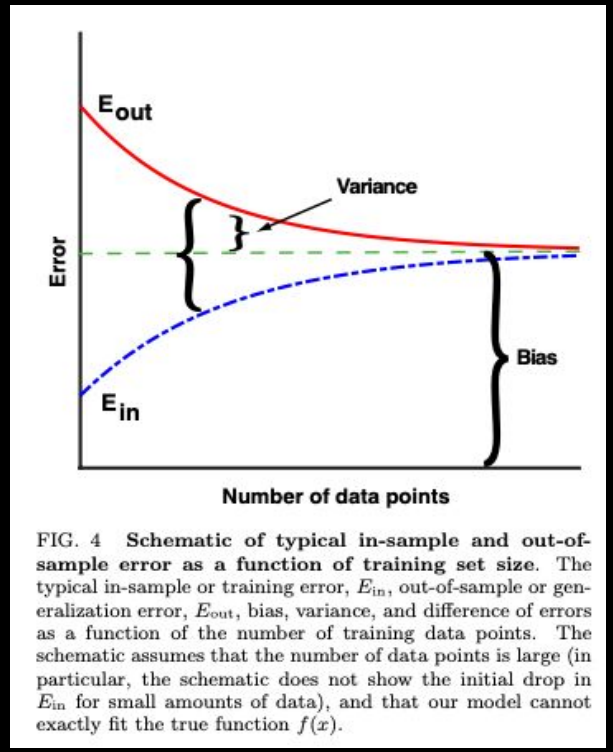


FIG. 4 **Schematic of typical in-sample and out-of-sample error as a function of training set size.** The typical in-sample or training error, E_{in}, out-of-sample or generalization error, E_{out}, bias, variance, and difference of errors as a function of the number of training data points. The schematic assumes that the number of data points is large (in particular, the schematic does not show the initial drop in E_{in} for small amounts of data), and that our model cannot exactly fit the true function $f(x)$.

- (1) The more data we train on, the more the sampling noise decreases and the training data becomes representative of the true distribution. For this reason E_{out} and E_{in} approach the same bias.
- The more data we train on, the more the sampling noise decreases and the training data becomes representative of the true distribution. For this reason E_{out} and E_{in} approach the same value called bias. The bias represents the best our model can do if we had an infinite amount of data.
- The more complex the model we use, the smaller the bias. However we do not have an infinite amount of data.
- For this reason the best predictive power is get by minimizing the E_{out}. E_{out} is decomposed in a bias and a variance, which measures the errors in training our model due to sampling noise
- The difference between E_{out} and E_{in} measures the difference between fitting and predicting.

Bias/Variance

Assumptions:
- E_{out} and E_{in} as a function of the size of training data
- We assume data come from a complicated distribution (so we won't exactly learn f(x))

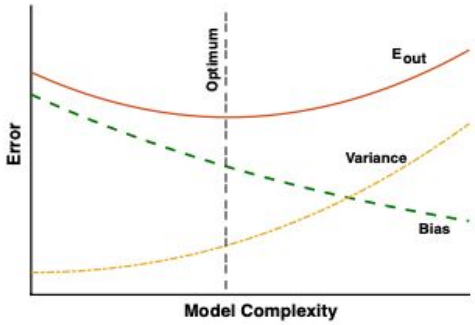


FIG. 5 **Bias-Variance tradeoff and model complexity.** This schematic shows the typical out-of-sample error E_{out} as function of the model complexity for a training dataset of fixed size. Notice how the bias always decreases with model complexity, but the variance, i.e. fluctuation in performance due to finite size sampling effects, increases with model complexity. Thus, optimal performance is achieved at intermediate levels of model complexity.

- Model complexity is a subtle concept which can in many cases be related to the number of parameters needed to approximate the true function f(x)
- E_{out} will be in general a non-monotonic function of the model complexity.
- It is generally minimized for intermediate complexity.
- Even though using a complicated model always reduces bias, at some point the model becomes too complex for the amount of training data that the generalization error becomes large due to high variance
- Thus it may be more suitable to use a more biased model with small variance.

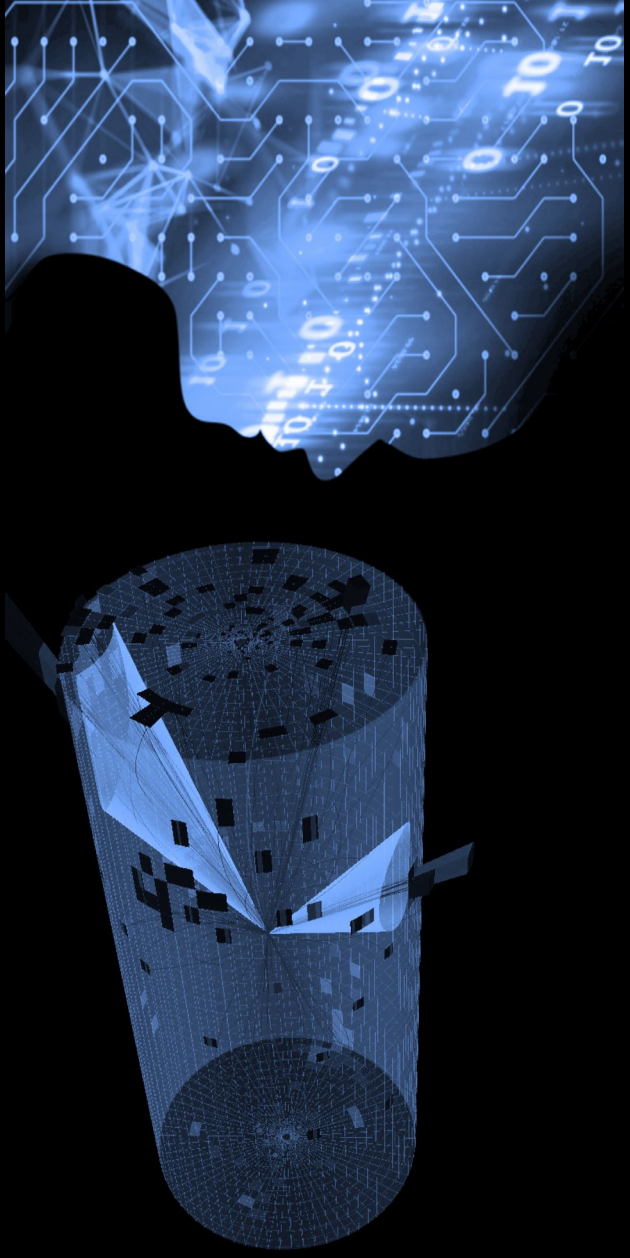
It can be shown that:

$$E_{\text{out}} = Bias^2 + Var + Noise,$$

$$Var = \sum_i \mathbb{E}_{\mathcal{D}}[(f(\boldsymbol{x}_i; \hat{\boldsymbol{\theta}}_{\mathcal{D}}) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x}_i; \hat{\boldsymbol{\theta}}_{\mathcal{D}})])^2],$$

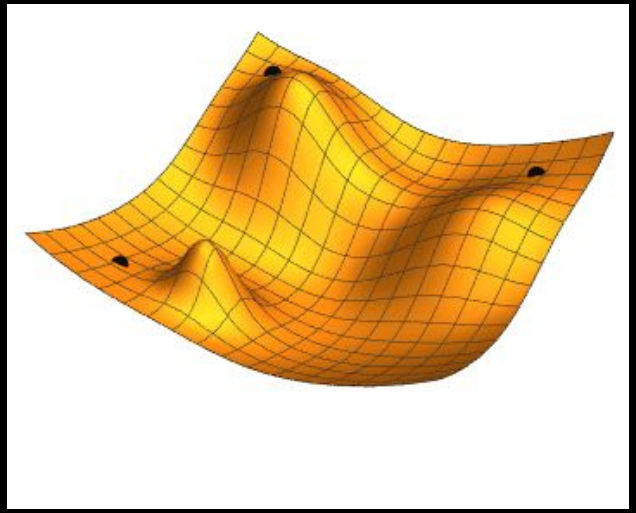
$$Bias^2 = \sum_i (f(\boldsymbol{x}_i) - \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x}_i; \hat{\boldsymbol{\theta}}_{\mathcal{D}})])^2$$

- Bias measures the deviation of the expectation value of our estimator (asymptotic value in the infinite data limit) from the true value
- Variance measures how much our estimator fluctuates due to finite-sample effects



Gradient Descent

Gradient Descent



- Almost every problem in ML starts with the same ingredients: a dataset X, a model g(θ) which is function of parameters θ, and a cost function C(X,g(θ)) describing how well the model explains the observations.
- The model is fit by minimizing the cost function.
- Gradient descent is a powerful approach to do so.

- Following this approach the training procedure ensures that the parameters flow towards a minimum of the cost function.
- GD in practice is full of surprises: the cost functions in ML are usually complicated, non-convex functions in a high-dimensional space with many local minima.
- Furthermore we almost never access the true function that we wish to minimize (we do not know it at the ground truth).
- In modern applications the number of parameters to fit is often enormous (millions of parameters and examples).

Gradient Descent

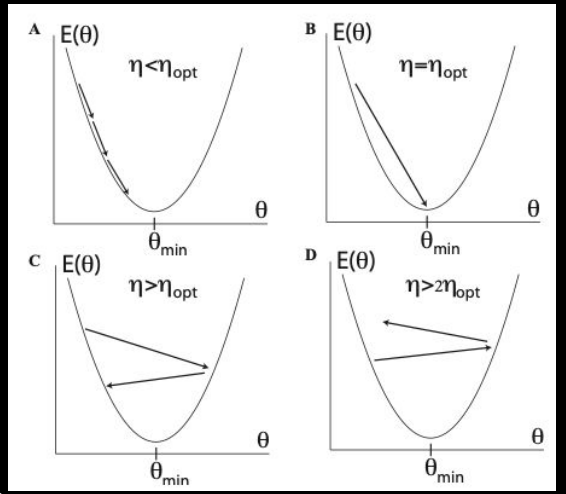
- Let's call the function we want to minimize. $E(\boldsymbol{\theta}) = \mathcal{C}(\boldsymbol{X}, g(\boldsymbol{\theta}))$
- This energy function can be written in terms of n data points $E(\boldsymbol{\theta}) = \sum_{i=1}^{n} e_i(\mathbf{x}_i, \boldsymbol{\theta})$
- In the simplest GD algorithm we iteratively update the parameters as:

$$\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t),$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

(η_t) learning rate: controls how big a step we should take in the direction of the gradient

  - For sufficiently small η_t we would converge to a local minimum, but this comes at a huge computational cost
  - If too large we can overshoot the minimum.
  - In practice we need to specify a "schedule" that decreases η_t at long times.

- The learning rate could be adapted using the inverse of the Hessian matrix (see Newton's method), so that larger steps are taken in flat directions and smaller steps in steep directions.
- In the case of a single parameter quadratic energy function we can easily identify four regimes depending on an η_{opt} as in the figure. In a multidimensional case, one could determine the largest eigenvalue λ_{max} of the Hessian and use a single learning rate for all parameters. Convergence requires $\eta < 2/\lambda_{max}$



Limitations and Alternatives: Stochastic Gradient Descent

- **Limitations of the simple GD approach**:
  - GD finds local minima
  - Gradients are computationally expensive to calculate for large datasets (sum over all data points)
  - Unlike in Newton's method, GD treats all directions in the parameter uniformly. We need to keep track of Hessian but it is computationally expensive)
  - GD is sensitive to initial conditions and can take exponential time to escape saddle points

- Stochastic Gradient Descent (SGD) is one of the most applied variants of the GD. The algorithm is stochastic.
- Where do we introduce stochasticity? It is incorporated by randomly selecting data points at each step to calculate the gradients. In other words, by approximating the gradient on a subset of the data called minibatch of size M (traditionally SGD was reserved for each data point, that is minibatch of size 1), B_k=1,..,n/M size M:(**32, 64, 128, 256, …**)
- A full iteration over all n data points, i.e. n/M minibatches, is called an epoch.

SGD algorithm

$$\mathbf{v}_t = \eta_t \nabla_\theta E^{MB}(\boldsymbol{\theta}),$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t.$$

- Benefits:
  - Stochasticity reduces chance to get stuck in local minima
  - It speeds up calculation

Gradient Descent with Momentum

(*) this is demonstrated by the physical analogy with the equation of motion for a mass m moving in a viscous medium with damping coefficient and potential

- SGD is almost always used with a “momentum” or inertia term (*)

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t,$$

(γ) momentum parameter

- v_t is a running average of recently encountered gradients. It is possible to demonstrate that the characteristic timescale for the memory used in the averaging procedure is: $(1-\gamma)^{-1}$

- Why is momentum useful? SGD momentum helps the GD algorithm gain speed in directions with persistent but small gradients even in the presence of stochasticity, while suppressing oscillations in high-curvature directions. Empirical studies show benefits in the transient phase of training, rather than during fine-tuning

- These benefits are sometimes even more pronounced in a slight modification called Nesterov Accelerated Gradient (NAG), which calculates the gradient at the expected value of the parameters

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta E(\boldsymbol{\theta}_t + \gamma \mathbf{v}_{t-1})$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t.$$

Gradient Descent with Second Moment

- RMSprop

$$\mathbf{g}_t = \nabla_\theta E(\boldsymbol{\theta})$$
$$\mathbf{s}_t = \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t + \epsilon}},$$

$\beta = 0.9$
learning rate $\eta_t \sim 10^{-3}$ (can be larger than previous methods due to the adaptive step size)
$\varepsilon \sim 10^{-8}$

The learning rate is reduced in directions where the gradient is consistently large

- ADAM

$$\mathbf{g}_t = \nabla_\theta E(\boldsymbol{\theta})$$
$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$
$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2$$
$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-(\beta_1)^t}$$
$$\hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1-(\beta_2)^t}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t + \epsilon}},$$

$\beta_1 = 0.9$
$\beta_2 = 0.99$

ADAM performs an additional bias correction to account for the fact that we are estimating the first two moments of the gradient using a running average (denoted with a hat)

It's possible to rewrite formulas as:

$$\boldsymbol{\sigma}_t^2 = \hat{\mathbf{s}}_t - (\hat{\mathbf{m}}_t)^2$$
$$\Delta\theta_{t+1} = -\eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2 + \epsilon}}$$

We adapt learning rate proportional to signal-to-noise ratio, for example if widely fluctuating, $\sigma >> m_t$

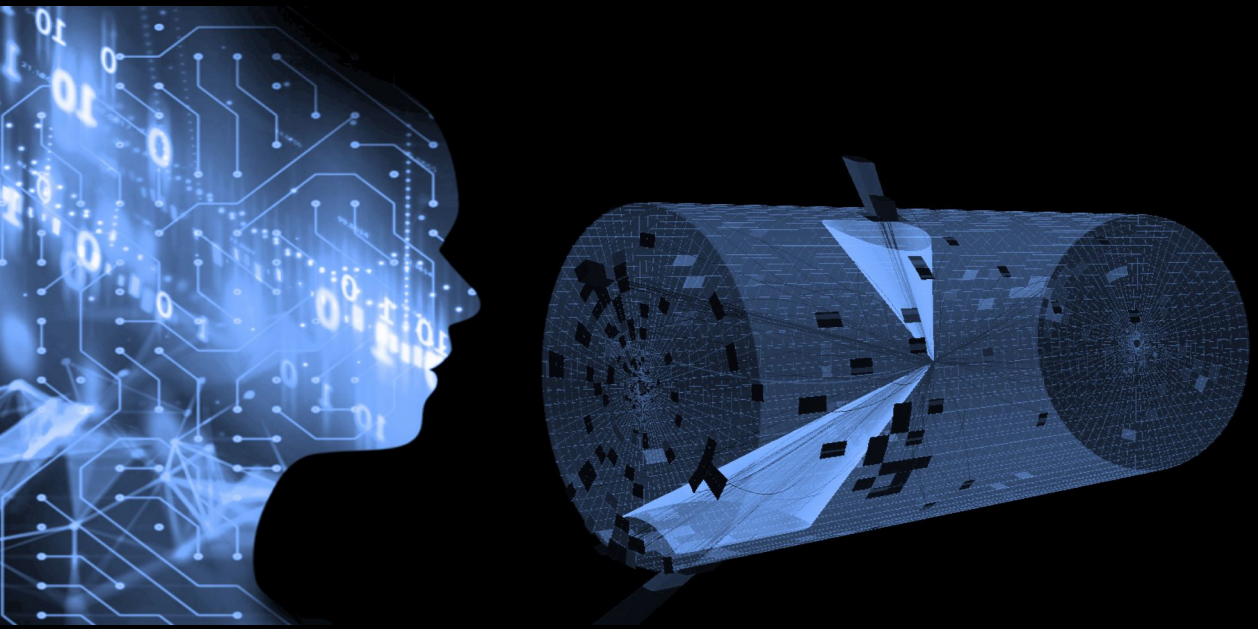
D.P. Kingma, J.L. Ba, ADAM, A method for Stochastic optimization, https://arxiv.org/pdf/1412.6980.pdf 2014 (>90k citations)

Practical tips and Further Reading

- Randomize data when making mini-batches
- Standardize inputs (learning becomes difficult when it has a mixture of steep and flat directions)
- Monitor the out-of-sample performance — validation set
- Adaptive optimization methods (ADAM, RMSprop) do not always have good generalization
  - Suffer when number of parameters exceeds number of data points [1]
  - Outperform with deep networks such as generative adversarial networks [2]

hblvi2ML Sec IV

[1] A. Wilson, et al .(2017), “The marginal value of adaptive gradient methods in machine learning,” arXiv preprint arXiv:1705.08292.
[2] I. Goodfellow, et al.(2014), “Generative adversarial nets,” in Advances in neural information processing systems, pp. 2672– 2680



Regression (in a nutshell)

Due to limited time no exercises on this part. Notebooks are available at [nblvi2ml].

Regression: the problem

scalar response

- Given a dataset with n samples $\mathcal{D} = \{(y_i, \boldsymbol{x}^{(i)})\}_{i=1}^{n}$
- Assume every sample has *p* features $\boldsymbol{x}^{(i)} \in \mathbb{R}^p$
- Let f be the true function that generated these samples via

  where $\boldsymbol{w}_{\text{true}} \in \mathbb{R}^p$ and ϵ_i is some i.i.d. white noise with zero mean and finite variance.

- One can cast the samples into an *n x p* matrix, called the design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$

  with rows $\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(i)} \in \mathbb{R}^p, \; i = 1, \cdots, n$ being observations

  and the columns $\boldsymbol{X}_{:,j} \in \mathbb{R}^n, j = 1, \cdots p$ being the measured features

- The function f is unknown explicitly and we presume its functional form.

- For example, in linear regression we assume $y_i = f(\boldsymbol{x}^{(i)}; \boldsymbol{w}_{\text{true}}) + \epsilon_i = \boldsymbol{w}_{\text{true}}^{\mathrm{T}} \boldsymbol{x}^{(i)} + \epsilon_i$

  for some unknown but fixed $\boldsymbol{w}_{\text{true}} \in \mathbb{R}^p$

Linear Regression

- We want to find a function *g* with parameters ***w*** fitting to the data *D* that can best approximate *f*

$$g(\boldsymbol{x}; \hat{\boldsymbol{w}}) \sim f$$

- When this is done, we can use this g to make predictions about the response y_0 for a new data point $\mathbf{x}_0$.
- Let's introduce the L_p norm which is helpful in regression. For any real number $p \geq 1$, we define the L^p norm of a vector $\boldsymbol{x} = (x_1, \cdots, x_d) \in \mathbb{R}^d$ to be

$$||\boldsymbol{x}||_p = (|x_1|^p + \cdots + |x_d|^p)^{\frac{1}{p}}$$

Least-square Regression

- Ordinary least squares linear regression (OLS) is defined as the minimization of the L_2 norm of the difference between the response y_i and the predictor $g(x^{(i)};w) = w^T x^{(i)}$:

$$\min_{\boldsymbol{w}\in\mathbb{R}^p} ||\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}||_2^2 = \min_{\boldsymbol{w}\in\mathbb{R}^p} \sum_{i=1}^{n} (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)} - y_i)^2$$

- We need to determine ***w*** that minimizes the L^2 error

- Geometrically speaking, the predictor $g(x^{(i)};w) = w^T x^{(i)}$ defines an hyperplane in R^p.

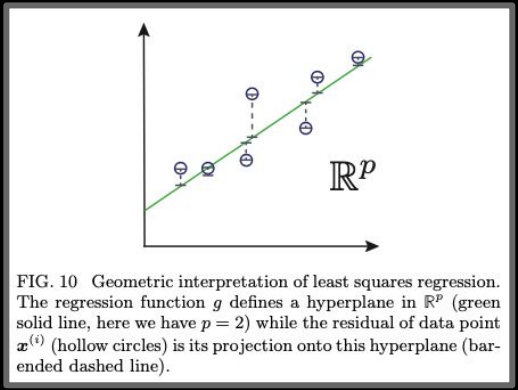


FIG. 10 Geometric interpretation of least squares regression. The regression function g defines a hyperplane in $\mathbb{R}^p$ (green solid line, here we have $p = 2$) while the residual of data point $\boldsymbol{x}^{(i)}$ (hollow circles) is its projection onto this hyperplane (bar-ended dashed line).

- This leads to the solution $\hat{w}_{\mathrm{LS}} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$

  where we assume $\mathbf{X^TX}$ is invertible, which is often the case when n ≥ p. Formally if rank(**X**) = p, the columns of X are linearly independent, then w_{LS} is unique.

- When rank(**X**)<p $\mathbf{X^TX}$ is singular, implying there are infinitely many solutions to the least-squares problem (if w0 is a solution, w_0+η is also a solution for any η such that **X**η=0.

Least-square Regression

- Having determined the least squares solution, we can calculate the best fit to our data as:

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{w}}_{\mathrm{LS}} = P_{\boldsymbol{X}}\boldsymbol{y}, \text{ where } P_{\boldsymbol{X}} = \boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}$$

- Notice that we found the solution w_{LS} in one shot, without any iterative optimization as seen e.g. for gradient descent.

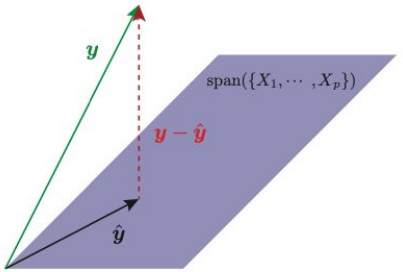


FIG. 11 The projection matrix $P_{\boldsymbol{X}}$ projects the response vector $\boldsymbol{y}$ onto the column space spanned by the columns of $\boldsymbol{X}$, $\mathrm{span}(\{\boldsymbol{X}_{:,1}, \cdots, \boldsymbol{X}_{:,p}\})$ (purple area), thus forming a fitted vector $\hat{\boldsymbol{y}}$. The residuals in Eq. (37) are illustrated by the red vector $\boldsymbol{y} - \hat{\boldsymbol{y}}$.

- As already discussed the difference between learning and fitting lies in the prediction on unseen data. It is therefore necessary to examine the out-of-sample error. The reader can find Jupyter notebooks in [hblvi2ml]

- It can be shown that the average errors are:

$$\begin{cases} \bar{E}_{\mathrm{in}} = \sigma^2\left(1 - \dfrac{p}{n}\right) \\ \bar{E}_{\mathrm{out}} = \sigma^2\left(1 + \dfrac{p}{n}\right) \end{cases} \Rightarrow \boxed{|\bar{E}_{\mathrm{in}} - \bar{E}_{\mathrm{out}}| = 2\sigma^2\frac{p}{n}} \Rightarrow$$

(provided we obtain the least squares solution w_{LS} from i.i.d. samples X and y generated through $y = \mathbf{X}\,\mathbf{w}_{true} + \varepsilon$)

If we have $p \gg n$ (i.e. high-dimensional data) the generalization error is extremely large and the model is not learning. Even when $p \approx n$ we might still not learn well due to the intrinsic noise σ^2.

To ameliorate this we use regularization and we will briefly mention two forms:
Ridge (L_2 penalty) and LASSO (L_1 penalty).

Ridge-Regression

- We add to the least squares loss function a regularizer defined as L_2 norm of the parameter vector we wish to optimize over. The penalized Ridge regression problem is:

$$\hat{\boldsymbol{w}}_{\text{Ridge}}(\lambda) = \underset{\boldsymbol{w}\in\mathbb{R}^p}{\arg\min} \left(||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda ||\boldsymbol{w}||_2^2 \right) \qquad (1)$$

- Equivalent to the constrained optimization problem:

$$\hat{\boldsymbol{w}}_{\text{Ridge}}(t) = \underset{\boldsymbol{w}\in\mathbb{R}^p:\ ||\boldsymbol{w}||_2^2 \le t}{\arg\min} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 \qquad (2)$$

meaning for any $t \geq 0$ and solution w_{Ridge} of (2), there exists a $\lambda \geq 0$ such that w_{Ridge} solves (1) — and vice versa. With the regularization term we are constraining the magnitude of the parameter vector learnt from the data.

$$\hat{\boldsymbol{w}}_{\text{Ridge}}(\lambda) = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda I_{p\times p})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

- When X is orthogonal:

$$\hat{\boldsymbol{w}}_{\text{Ridge}}(\lambda) = \frac{\hat{\boldsymbol{w}}_{\text{LS}}}{1+\lambda}$$

It is possible to demonstrate that both Ridge and LS linear regression have to project y to the column space of X. Ridge further shrinks each basis component by a factor $d_j^2/(d_j^2+\lambda^2)$, as can be obtained from singular value decomposition.

[hblvi2ml, Sec VI] Note that the equivalence between the penalized and the constrained (regularized) form of LSO does not always hold. It holds for Ridge and LASSO,

LASSO

Least absolute shrinkage and selection operator

Rockafellar, Ralph Tyrell (2015), Convex analysis (Princeton university press).

- LASSO in the penalized form is defined by

$$\hat{\boldsymbol{w}}_{\text{LASSO}}(\lambda) = \underset{\boldsymbol{w}\in\mathbb{R}^p}{\arg\min} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{w}||_1 \qquad (1)$$

- Assuming X is orthogonal, it is possible to demonstrate that:

$$\hat{w}_j^{\text{LASSO}}(\lambda) = \text{sign}(\hat{w}_j^{\text{LS}})(|\hat{w}_j^{\text{LS}}| - \lambda)_+$$

where $()_+$ indicates the positive part and w_j^{LS} is the j-th component of the LS solution.

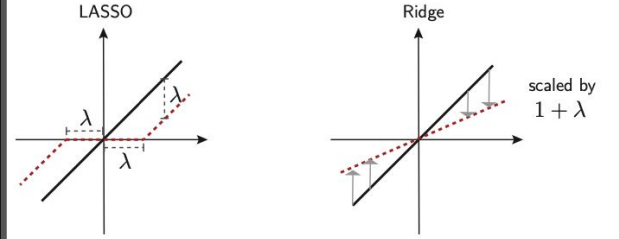


FIG. 12 [Adapted from (Friedman *et al.*, 2001)] Comparing LASSO and Ridge regression. The black 45 degree line is the unconstrained estimate for reference. The estimators are shown by red dashed lines. For LASSO, this corresponds to the soft-thresholding function Eq. (54) while for Ridge regression the solution is given by Eq. (46)

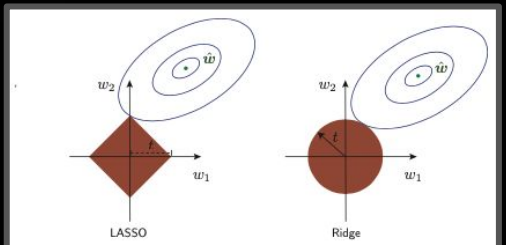


FIG. 13 [Adapted from (Friedman *et al.*, 2001)] Illustration of LASSO (left) and Ridge regression (right). The blue concentric ovals are the contours of the regression function while the red shaded regions represent the constraint functions: (left) $|w_1| + |w_2| \leq t$ and (right) $w_1^2 + w_2^2 \leq t$. Intuitively, since the constraint function of LASSO has more protrusions, the ovals tend to intersect the constraint at the vertex, as shown on the left. Since the vertices correspond to parameter vectors $\boldsymbol{w}$ with only one non-vanishing component, LASSO tends to give sparse solution.

LASSO vs Ridge

LASSO tends to give sparse solutions, i.e. many components of w_{LASSO} are zero. The L1 regularizer of LASSO has sharp protrusions, the intersection of the regressor contours tend to occur at the vertex of the feasibility region, implying the solution vector will be sparse.

[hblvi2ml, Sec VI] As in Ridge, LASSO can be written in its constrained optimization form. However to get the analytic solution of LASSO, we cannot take the gradient of (1) with respect to **w**, since the L_1-regularizer is not everywhere differentiable. Nonetheless, LASSO is a convex problem and we can invoke the subgradient optimality condition to get the solution.

In practice

[hblvi2ml] Notebook3: Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
#import seaborn

from sklearn import datasets, linear_model

# Load Training Data set with 200 examples

number_examples=200
diabetes = datasets.load_diabetes()
X = diabetes.data[:number_examples]
y = diabetes.target[:number_examples]

# Set up Lasso and Ridge Regression models
ridge=linear_model.Ridge()
lasso = linear_model.Lasso()
```

```
n_samples = 150
n_samples_train = 100
X_train, X_test = X[:n_samples_train], X[n_samples_train:]
y_train, y_test = y[:n_samples_train], y[n_samples_train:]
train_errors_ridge = list()
test_errors_ridge = list()

train_errors_lasso = list()
test_errors_lasso = list()



# Initialize coeffficients for ridge regression and Lasso

coefs_ridge = []
coefs_lasso=[]
for a in alphas:
    ridge.set_params(alpha=a)
    ridge.fit(X_train, y_train)
    coefs_ridge.append(ridge.coef_)

    # Use the coefficient of determination R^2 as the performance of predi
ction.
    train_errors_ridge.append(ridge.score(X_train, y_train))
    test_errors_ridge.append(ridge.score(X_test, y_test))

    lasso.set_params(alpha=a)
    lasso.fit(X_train, y_train)
    coefs_lasso.append(lasso.coef_)
    train_errors_lasso.append(lasso.score(X_train, y_train))
    test_errors_lasso.append(lasso.score(X_test, y_test))

################################################################################
```

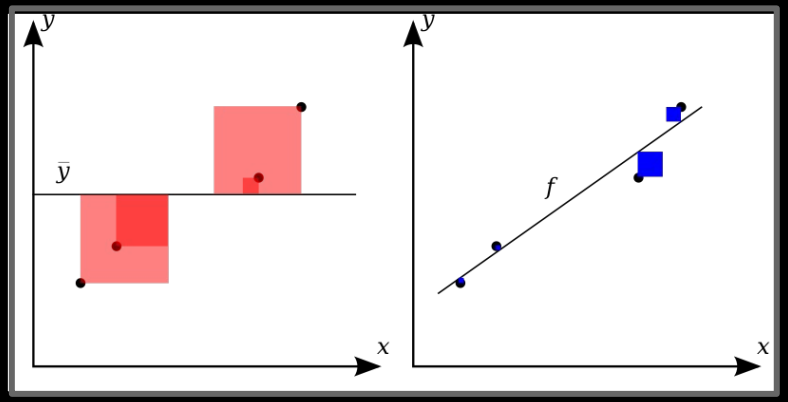
Coefficient of Determination

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left| y_i^{\text{true}} - y_i^{\text{pred}} \right|^2}{\sum_{i=1}^{n} \left| y_i^{\text{true}} - \frac{1}{n} \sum_{i=1}^{n} y_i^{\text{pred}} \right|^2}$$

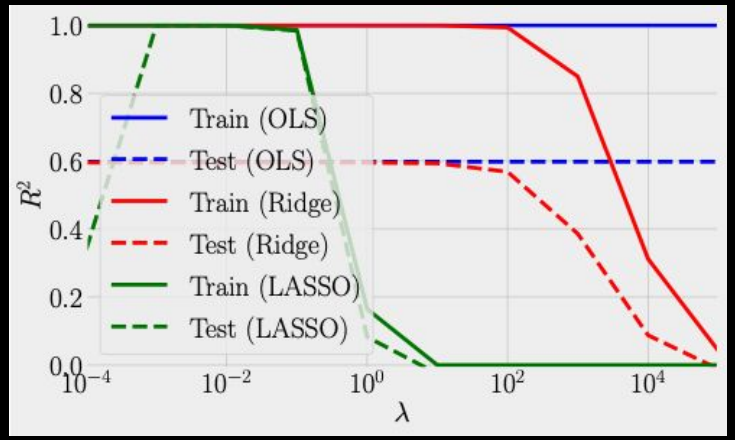
SS_{res}

SS_{tot}

- In the best case, the modeled values exactly match the observed values, which results in SS_{res} =0 and R^2=1.
- A baseline model which always predicts the average y, will have R^2=0.
- Models that have worse predictions than this baseline will have a negative R^2.
- The regularization parameter λ affects Ridge and LASSO regressions. A good practice is check the performance as a function of λ.

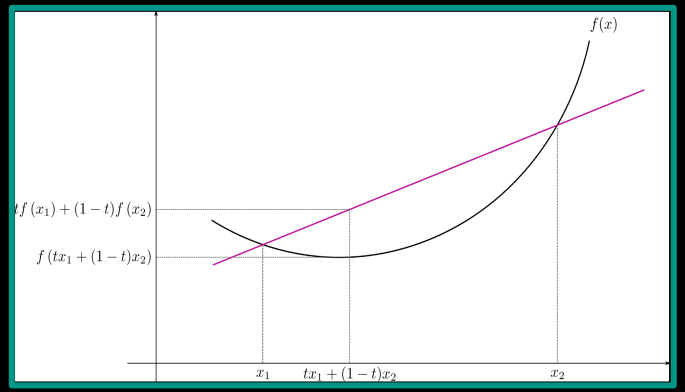


The better the linear regression (on the right) fits the data in comparison to the simple average (on the left graph), the closer the value of is to 1.
The areas of the blue squares represent the squared residuals with respect to the linear regression. The areas of the red squares represent the squared residuals with respect to the average value. Taken from here.



Example taken from [hblvi2ml] for the Isospin regression problem. Notice that LASSO test curve is not monotonic and there is a sweet spot.

Convexity and elastic net



- A simple inspection reveals that both LASSO and Ridge regression are convex in **w**.
- A function $R^n \rightarrow R$ is called convex if its domain is a convex set and for any x,y in the domain, and $t \in [0,1]$:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

- For convex functions, any local minimizer is a global minimizer: as long as we're "going down the hill" and we stop when we can't go any further, then we've hit the global minimum.
- Ridge is actually a *strictly convex* problem (assuming $\lambda>0$) due to presence of L_2 penalty. In fact, this is always true regardless of X and the solution is always well-defined.
- LASSO is not always strictly convex and hence by convexity theory, it need not have a unique solution. The LASSO solution is unique under general conditions, e.g., when X has columns in *general position* (see Tibshirani 2013). To mitigate this, one can define a modified problem called the elastic net such that the function we want to minimize is always strictly convex:

$$\min_{\mathbf{w}\in\mathbb{R}^p} ||\mathbf{Xw}-\mathbf{y}||_2^2 + \lambda||\mathbf{w}||_1 + \delta||\mathbf{w}||_2^2$$

The elastic net combines some of the desirable properties of Ridge regression (e.g., prediction) with the sparsity properties of LASSO.

Logistic regression

- Problems like classification are concerned with outcomes taking discrete variables (i.e. categories):
  - Signal vs bkgd, Phase of a system, etc
- Logistic regression deals with dichotomous outcome (True or False, 1 or 0, etc)
- The inner workings of logistic regression are valuable in the study of modern supervised deep neural networks
- In what follows:
  - Define logistic regress
  - Derive cost function (cross entropy) using a Bayesian approach
  - Discuss its minimization
  - Generalize logistic regression in the case of multiple categories (called SoftMax regression)
- We focus on the SUSY dataset of [hblvi2ml]

Categorize

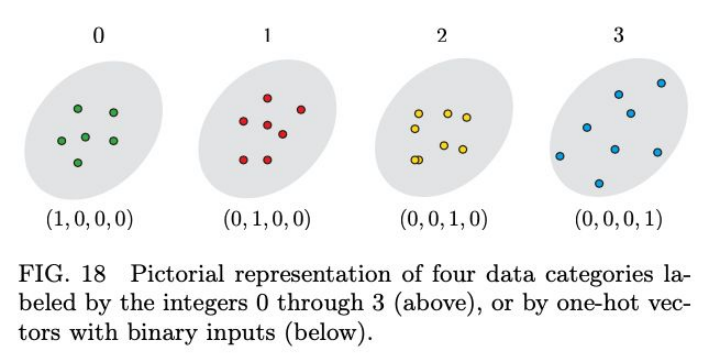
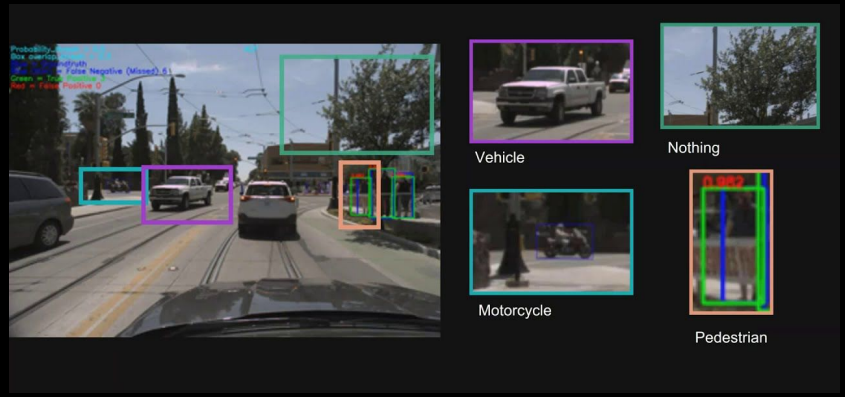


FIG. 18 Pictorial representation of four data categories labeled by the integers 0 through 3 (above), or by one-hot vectors with binary inputs (below).



- Before delving into logistic regression, it is helpful to consider a simple linear classifier that categorizes examples using a weighted linear-combination of the features and an additive offset

$$s_i = \boldsymbol{x}_i^T \boldsymbol{w} + b_0 \equiv \mathbf{x}_i^T \mathbf{w}$$

- We use the short-hand notation $\mathbf{x}_i = (1, \boldsymbol{x}_i) \quad \mathbf{w} = (b_0, \boldsymbol{w})$

  This function takes values on the entire real axis. In the case of logistic regression, however, the labels y_i are discrete variables.

- One could use a sign function for a binary classifier:

$$\{0, 1\},\ \sigma(s_i) = \mathrm{sign}(s_i) = 1 \text{ if } s_i \geq 0$$

commonly known as the Perceptron

Logistic (sigmoid) function

- Perceptron is an example of “hard classification”, i.e. each datapoint is assigned a category (0 or 1)
- It is favorable in many cases to have a “soft classifier”, that outputs the probability of a given category (given x_i, the classifier returns the probability of being in category m)
- Logistic (or sigmoid) function: $\sigma(s) = \frac{1}{1+\mathrm{e}^{-s}}$, where $1-\sigma(s) = \sigma(-s)$
- In logistic regression, the probability that a point x_i belongs to a category $y_i = \{0,1\}$ is given by:

$$P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{1+\mathrm{e}^{-\mathbf{x}_i^T\boldsymbol{\theta}}},$$
$$P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta})$$

Here θ are the weights we want to learn

Cross-entropy as cost function for logistic regression

- We now use Maximum Likelihood Estimation (MLE) to define the cost function for logistic regression. In MLE we choose parameters to maximize the probability of seeing the observed data.
- Consider a dataset D={(y_i,x_i)} with binary labels $y_i \in \{0,1\}$ from which the data points are drawn independently. The likelihood of observing the data under our model is:

$$P(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^{n} \left[\sigma(\mathbf{x}_i^T\mathbf{w})\right]^{y_i} \left[1-\sigma(\mathbf{x}_i^T\mathbf{w})\right]^{1-y_i}$$

- We can compute the log-likelihood: $$l(\mathbf{w}) = \sum_{i=1}^{n} y_i \log\sigma(\mathbf{x}_i^T\mathbf{w}) + (1-y_i)\log\left[1-\sigma(\mathbf{x}_i^T\mathbf{w})\right]$$
- The maximum likelihood estimator is defined as:
- $$\hat{\mathbf{w}} = \arg\max_{\theta} \sum_{i=1}^{n} y_i \log\sigma(\mathbf{x}_i^T\mathbf{w}) + (1-y_i)\log\left[1-\sigma(\mathbf{x}_i^T\mathbf{w})\right]$$
- The cost (error) function for logistic regres is defined as the negative log-likelihood:

$$\mathcal{C}(\mathbf{w}) = -l(\mathbf{w})$$
$$= \sum_{i=1}^{n} -y_i \log\sigma(\mathbf{x}_i^T\mathbf{w}) - (1-y_i)\log\left[1-\sigma(\mathbf{x}_i^T\mathbf{w})\right]$$

commonly known as the Cross-entropy

We note that, just as in linear regression, in practice we usually supplement the cross-entropy with additional regularization terms, usually L_1 and L_2 regularization

Minimization of the cross-entropy

- The cross-entropy is a convex function of the weights w and therefore any local minimizer is a global minimizer.
- Minimizing the cost function leads to:

$$\mathbf{0} = \nabla C(\mathbf{w}) = \sum_{i=1}^{n} \left[\sigma(\mathbf{x}_i^T \mathbf{w}) - y_i\right] \mathbf{x}_i$$

where we made use of the logistic function identity $\partial\ \sigma(s) = \sigma(s)[1-\sigma(s)]$

- The cross-entropy is a convex function of the weights w and therefore any local minimizer is a global minimizer.

- The above is a transcendental equation for **w**, the solution of which unlike linear regression, cannot be written in closed form. This is the case where we need to utilize the numerical methods previously introduced, such as gradient descent.
- Notice that Scikit's logistic regression solvers have in-built regularizers. Their role is fundamental in general to prevent overfitting.

SoftMax Regression

- So far we focused only on binary classification. We can generalize logistic regression to multi-class classification (multinomial logistic regression). The softmax in machine learning is related to the Boltzmann distribution in physics:

$$p_i = \frac{1}{Q} e^{-\varepsilon_i/(kT)} = \frac{e^{-\varepsilon_i/(kT)}}{\sum_{j=1}^{M} e^{-\varepsilon_j/(kT)}}$$

p_i is the probability of a state;
M is the number of accessible states

- In ML, can be formulated as:

$$P(y_{im'} = 1 | \boldsymbol{x}_i, \{\mathbf{w}_k\}_{k=0}^{M-1}) = \frac{\mathrm{e}^{-\mathbf{x}_i^T \mathbf{w}_{m'}}}{\sum_{m=0}^{M-1} \mathrm{e}^{-\mathbf{x}_i^T \mathbf{w}_m}}$$

where $\boldsymbol{y}_i = (1, 0, \cdots, 0)$ means the sample x_i belongs to class 1, and $y_{im'}$ is the m'-th component of the vector $\mathbf{y}_i$. From the above, it's possible to build a likelihood and define the cost:

$$\mathcal{C}(\mathbf{w}) = -\sum_{i=1}^{n} \sum_{m=0}^{M-1} y_{im} \log P(y_{im} = 1 | \boldsymbol{x}_i, \mathbf{w}_m) + (1 - y_{im}) \log \left(1 - P(y_{im} = 1 | \boldsymbol{x}_i, \mathbf{w}_m)\right)$$

Notice that for M=1 we recover the cross-entropy of the logistic regression.

Practical tips and Further Reading

- OLS can be optimized with gradient descent, Newton's method, or in closed form.

Ordinary Least Squares:

- $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2$.
- Squared loss.
- No regularization.
- Closed form: $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$.

Ridge Regression:

- $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 + \lambda ||\mathbf{w}||_2^2$.
- Squared loss.
- $l2$-regularization.
- Closed form: $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1}\mathbf{X}\mathbf{y}^\top$.

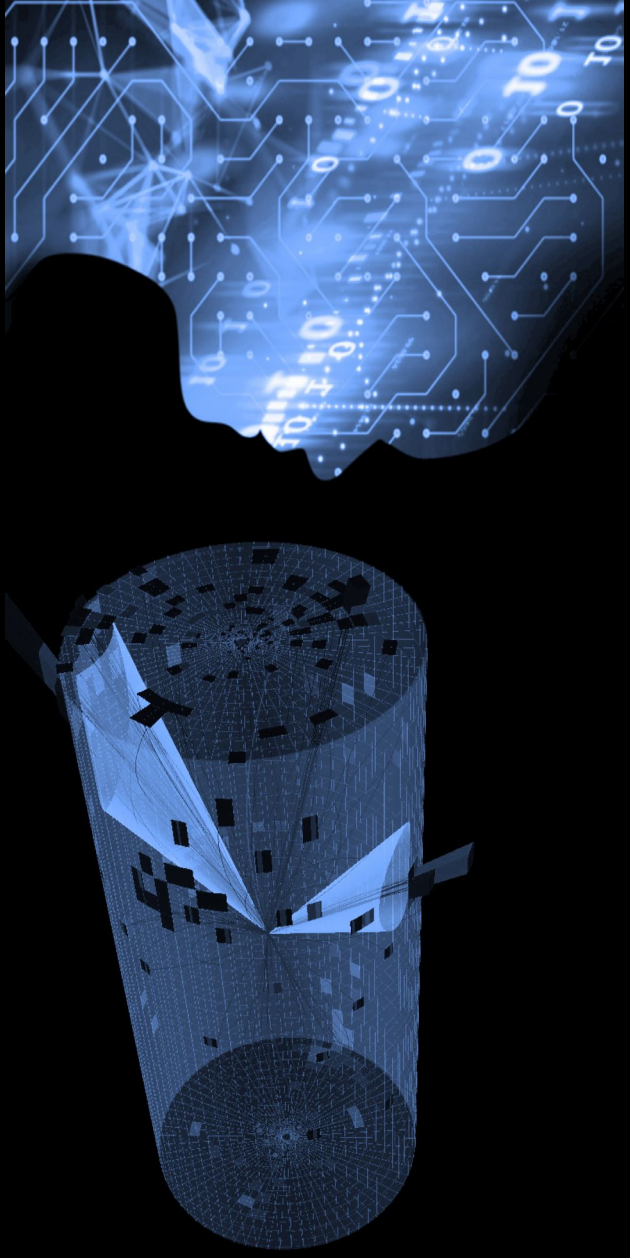
- Ridge has a closed form too.
- Read scikit-learn documentation to see details on implementation
- “Polynomial” regression
- The most commonly used loss function for Linear Regression is **Least Squared Error**, and its cost function is also known as Mean Squared Error(MSE) [The terms cost and loss functions almost refer to the same meaning. But, loss function mainly applies for a single training set as compared to the cost function which deals with a penalty for a number of training sets or the complete batch]

[1] https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote08.html
[2] https://towardsdatascience.com/optimization-of-supervised-learning-loss-function-under-the-hood-df1791391c82

Practical tips and Further Reading

- Categories: one-hot encoder One-hot encoding **ensures that machine learning does not assume that higher numbers are more important**. For example, the value '8' is bigger than the value '1', but that does not make '8' more important than '1'. The same is true for words: the value 'laughter' is not more important than 'laugh'.
- https://pytorch.org/docs/master/generated/torch.nn.CrossEntropyLoss.html Therefore, there's no need to one-hot encode your data if you have the labels already provided
- Logistic regression is easier to implement, interpret, and very efficient to train. **If the number of observations is lesser than the number of features**, Logistic Regression should not be used, otherwise, it may lead to overfitting. Logistic regression can be thought as a special case of NN with no hidden layer, that uses the sigmoid activation function and softmax with cross-entropy loss.
- A full fledged NN with hidden layers (deep network) and non-linear activation functions allows to capture highly complex functions of the features that could be characteristic of several problems.
- There are other techniques (e.g., SVMs not covered here) which can capture nonlinear functions. But NNs are popular because there are highly evolved and scalable platforms to capture more and more complex relationships by easily constructing a deep network and feeding in a lot of data

[1] https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote08.html
[2] https://towardsdatascience.com/optimization-of-supervised-learning-loss-function-under-the-hood-df1791391c82



Combining Models

Ensemble Methods

"Wisdom of the crowds", Aristotle

- One of the most powerful and widely-applied ideas in modern ML. Ensemble methods correspond to combining predictions from multiple — often weak — classifiers to improve the predictive performance.
- Also in the context of NN is common to combine the predictions from multiple NNs, e.g., in complicated image classification problems.
- The key to determining when ensemble methods work is the degree of correlation between the models [1].
- Largely used ensemble methods are:
  - Boosting
  - Random forest
  - Gradient boosted trees (e.g., XGBoost)

Importance/role of correlation* between models

1. Holding the ensemble size fixed, averaging the predictions of correlated models reduces the variance less than averaging uncorrelated models.
2. In some cases, correlations between models within an ensemble can result in an increase in bias, offsetting any potential reduction in variance gained from ensemble averaging.

One of the most dramatic examples of increased bias from correlations is the catastrophic predictive failure of all derivative models used by Wall Street during the 2008 financial crisis.

[1] Louppe, Gilles (2014), "Understanding random forests: From theory to practice," arXiv preprint arXiv:1407.7502

*E.g., the correlation coefficient between the predictions made by two randomized models based on the same training set but with different random seeds [hblvi2ml]

Bias-Variance Decomposition for Ensambles

- The bias-variance tradeoff is discussed in the context of continuous predictions like regression, but many intuitions apply also for classification tasks.
- Let's recall the bias-variance tradeoff for a single model first.
- Consider the dataset $\mathbf{X}_{\mathcal{L}} = \{(y_j, \mathbf{x}_j), j = 1 \dots N\}$ and let's assume is generated from a noisy model $y = f(\mathbf{x})$ where noise is normally distributed with mean zero and st. deviation σ_ε.
- Assume we have a predictor $\hat{g}_{\mathcal{L}}(x)$ that gives a prediction of our model for a data point x. The estimator can be chosen by minimizing the cost function squared error:

$$\mathcal{C}(\boldsymbol{X}, g(\boldsymbol{x})) = \sum_i (\mathbf{y}_i - \hat{g}_{\mathcal{L}}(\mathbf{x}_i))^2$$

- We already showed that the expected generalization error can be decomposed as:

$$\mathbb{E}_{\mathcal{L},\epsilon}[\mathcal{C}(\boldsymbol{X}, g(\boldsymbol{x}))] = Bias^2 + Var + Noise.$$

- What happens now if we have an aggregate ensemble predictor?

$$\hat{g}^A_{\mathcal{L}}(\mathbf{x}_i, \{\theta\}) = \frac{1}{M} \sum_{m=1}^{M} \hat{g}_{\mathcal{L}}(\mathbf{x}_i, \theta_m)$$

M is the dimension of the ensemble

We assume that θ parametrize members of the ensemble

θ can be seen as the hyper-parameters of models, they introduce stochasticity in the ensemble

Bias-Variance Decomposition for Ensambles

- What happens now if we have an aggregate ensemble predictor?

$$\hat{g}^A_{\mathcal{L}}(\mathbf{x}_i, \{\theta\}) = \frac{1}{M}\sum_{m=1}^{M} \hat{g}_{\mathcal{L}}(\mathbf{x}_i, \theta_m)$$

M is the dimension of the ensemble

- For a thorough derivation see [hblvi2ml, Sec. VIII].

$$Var(\mathbf{x}) = \mathbb{E}_{\mathcal{L},\theta}[(\hat{g}^A_{\mathcal{L}}(\mathbf{x}, \{\theta\}) - \mathbb{E}_{\mathcal{L},\theta}[\hat{g}^A_{\mathcal{L}}(\mathbf{x}, \{\theta\})])^2]$$

$$= \frac{1}{M^2}\left[\sum_{m,m'} \mathbb{E}_{\mathcal{L},\theta}[\hat{g}_{\mathcal{L}}(\mathbf{x}, \theta_m)\hat{g}_{\mathcal{L}}(\mathbf{x}, \theta_{m'})] - M^2 \sum_i [\mu_{\mathcal{L},\theta}(\mathbf{x})]^2\right]$$

$$= \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta} + \frac{1-\rho(\mathbf{x})}{M}\sigma^2_{\mathcal{L},\theta}.$$

Intuitively this derives from the variance of n correlated variables:

$Var(X) = \sigma^2/n + (n-1)/n\ \rho\sigma^2$

$= \rho\sigma^2 + (1-\rho)/n\ \sigma^2$

Expectations are computed over the joint distribution of datasets $\mathcal{L}$ and hyperparameters θ

$$Bias^2(\mathbf{x}) = (f(\mathbf{x}) - \mathbb{E}_{\mathcal{L},\theta}[\hat{g}^A_{\mathcal{L}}(\mathbf{x}, \{\theta\})])^2$$

$$= (f(\mathbf{x}) - \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathcal{L},\theta}[\hat{g}_{\mathcal{L}}(\mathbf{x}, \theta_m)])^2$$

$$= (f(\mathbf{x}) - \mu_{\mathcal{L},\theta})^2.$$

- Variance of the aggregate estimator depends on their correlation
- For large ensemble ($M \to \infty$) variance is significantly reduced, and for completely random ensembles ($\rho(x)=0$) it is maximally suppressed!
- Bias of the aggregate predictor is just the expected bias of a single model

Intuitions behind ensembles

Ensembles are successful for the following reasons:

1. Statistical: provided their predictions are uncorrelated, averaging several models reduces the risk of choosing the wrong hypothesis.
2. Computational: many learning algorithms rely on some greedy assumption or local search that may get stuck in local optima. An ensemble made of individual models built from many different starting points may provide a better approximation of the true unknown function
3. Representational power (expressivity): for a learning set of finite size, the true function cannot be represented by any of the candidate models in the hypothesis H. By combining several models in an ensemble, it may be possible to expand the space of representable functions and to better model the true function.

Using an ensemble allows one to reduce the variance by averaging the result of many independent classifiers. This procedure works best for unstable predictors for which errors are dominated by variance due to finite sampling.

[1] Louppe, Gilles (2014), “Understanding random forests: From theory to practice,” arXiv preprint arXiv:1407.7502

Bagging

Bootstrap AGGregation

- Imagine to have a large dataset $\mathcal{L}$ that we could partition into M smaller data sets $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_M\}$. If each is sufficiently large to create a predictor, we can create an ensemble aggregate predictor:

$$\hat{g}_{\mathcal{L}}^{A}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} g_{\mathcal{L}_i}(\mathbf{x})$$

We know this can significantly reduce variance without increasing bias.
But we need to have enough data in each partition.

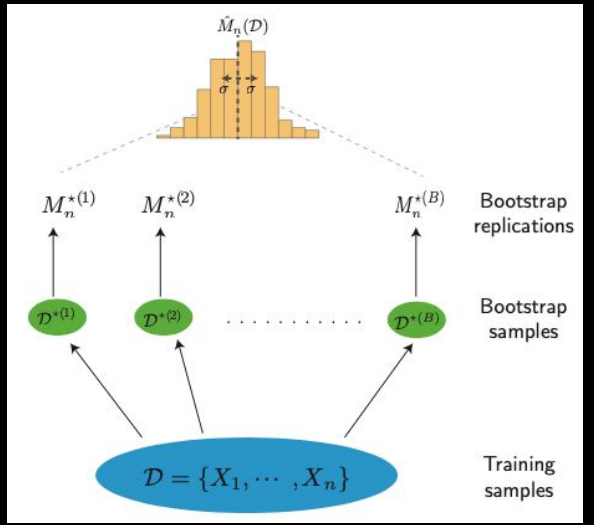
- For classification tasks where each predictor predicts a class label $j \in \{1, \ldots, J\}$, this corresponds to a majority vote:

$$\hat{g}_{\mathcal{L}}^{A}(\mathbf{x}) = \arg\max_{j} \sum_{i=1}^{M} I[g_{\mathcal{L}_i}(\mathbf{x}) = j]$$

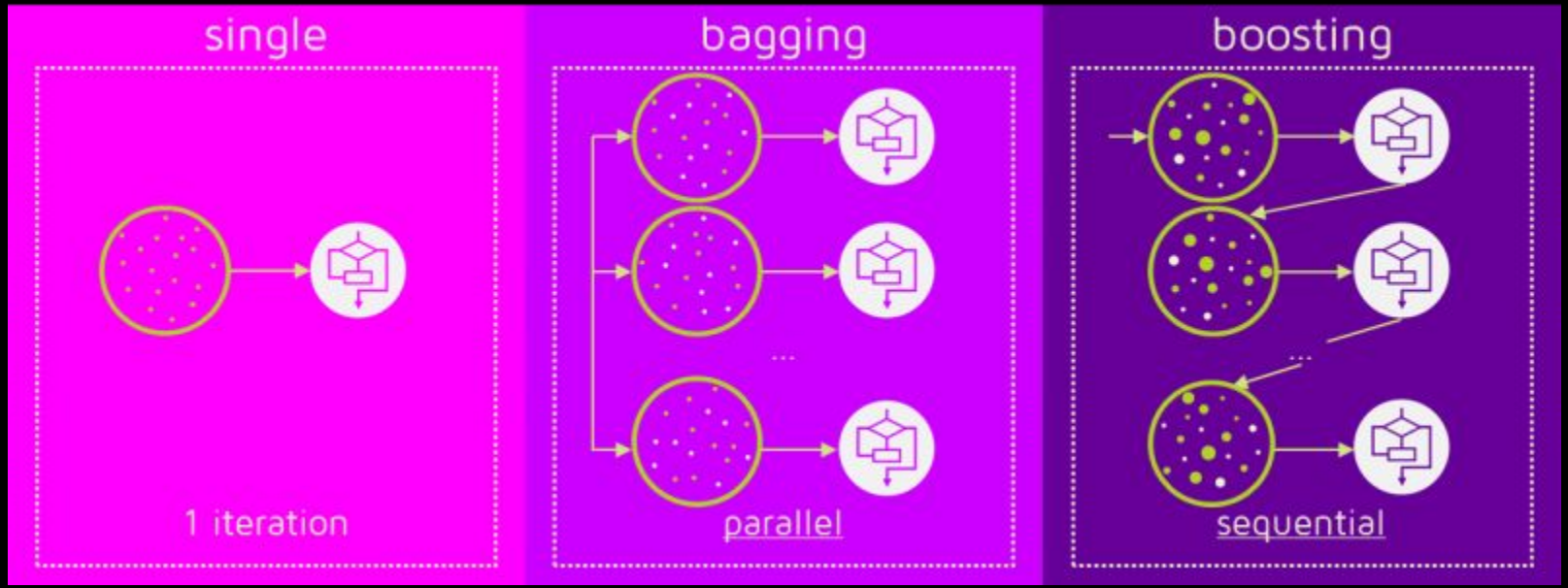
- This can be circumvented with empirical bootstrapping, that is with new bootstrapped datasets $\{\mathcal{L}^{BS}_1, \mathcal{L}^{BS}_2, \ldots, \mathcal{L}^{BS}_M\}$.

- Resampling with replacement from the original dataset

- The price we pay for using bootstrapped training datasets as opposed to really partitioned dataset is an increase in the bias of our bagged estimators.



Boosting



Boosting is another powerful ensemble method.
In bagging, the contribution of all predictors is weighted equally in the bagged (aggregate) predictor. In some problems instead one might prefer to use an “autocratic approach” that emphasizes the best predictors.

Boosting

- In boosting an ensemble of weak classifiers is combined into a boosted classifier. Each classifier is associated with a weight α_k (such that $\Sigma_k \alpha_k = 1$) that indicates how much it contributes.

$$g_A(\mathbf{x}) = \sum_{K=1}^{M} \alpha_k g_k(\mathbf{x})$$

- AdaBoost (Adaptive Boosting) is a popular technique. The aggregate classifier is formed in an iterative process.

We construct the boosted classifier as follows:

- **Initialize** $w_{t=1}(\boldsymbol{x}_n) = 1/N, n = 1, \cdots, N$.
- **For** $t = 1 \cdots, T$(desired termination step), **do**:
  1. Select a hypothesis $g_t \in \mathcal{H}$ that minimizes the weighted error

$$\epsilon_t = \sum_{i=1}^{N} w_t(\boldsymbol{x}_i) \mathbb{1}(g_t(\boldsymbol{x}_i) \neq y_i) \qquad (105)$$

  2. Let $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$, update the weight for each data $\boldsymbol{x}_n$ by

$$w_{t+1}(\boldsymbol{x}_n) \leftarrow w_t(\boldsymbol{x}_n) \frac{\exp[-\alpha_t y_n g_t(\boldsymbol{x}_n)]}{Z_t},$$

  where $Z_t = \sum_{n=1}^{N} w_t(\boldsymbol{x}_n) e^{-\alpha_t y_n g_t(\boldsymbol{x}_n)}$ ensures all weights add up to unity.

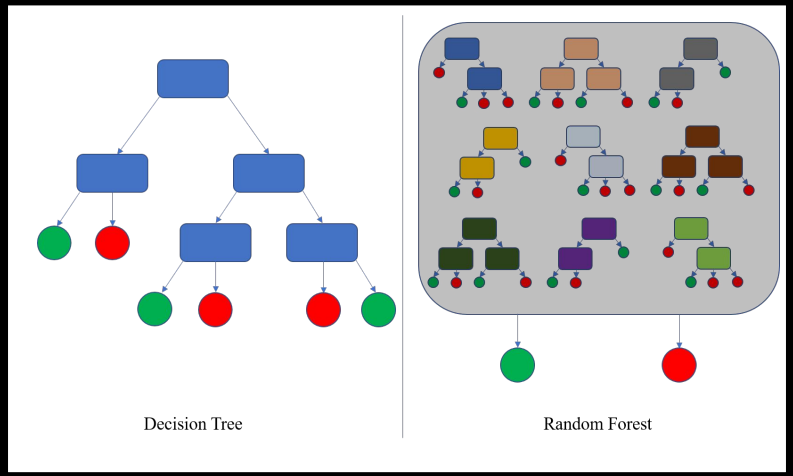
- **Output** $g_A(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\boldsymbol{x})\right)$

pseudo-code

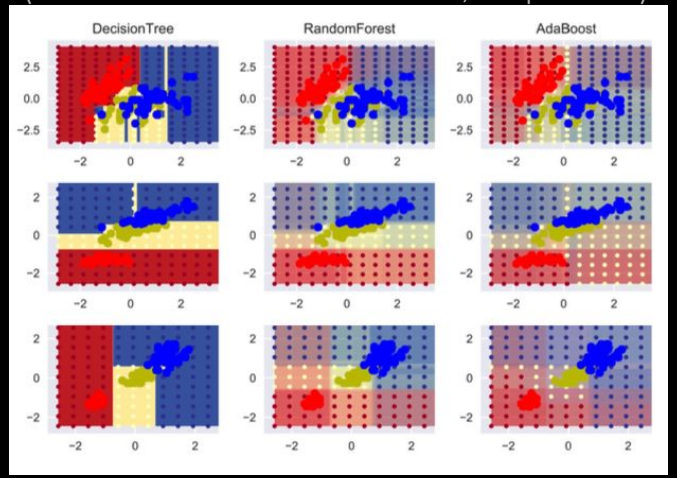
Freund, Yoav, Robert Schapire, and Naoki Abe (1999), "A short introduction to boosting," Journal-Japanese Society For Artificial Intelligence 14 (771-780), 1612.

Random Forest

- A decision tree uses a series of questions to hierarchically partition the data.
- A random forest is composed of a family of (randomized) tree-based classifier decision trees
- It is clear that more complex decision trees lead to finer partitions that give improved performance on the training set. However, this generally leads to overfitting, limiting the out-of-sample performance.
- In order to create an ensemble of decision trees, we must introduce a randomization procedure (the power of ensembles to reduce variance only manifests when randomness reduces correlations). Three ways for randomness
    - (i) Bag, (ii) feature bagging, (iii) extremized random forests (combination)



Comparison of decision surfaces
(each row a subset of 2 features, Iris problem)



Breiman, Leo (2001), “Random forests,” Machine learning 45 (1), 5–32.

Gradient Boosted Trees and XGBoost

- Gradient boosted trees combine boosting and gradient descent (in particular Newton's method) to construct ensemble of decision trees
- Ensemble are created iteratively. A cost function measures the performance of the ensemble. At each step we compute the gradient of the ensemble and add trees that move in the direction of the gradient.
- Extreme Gradient Boosting (XGBoost) is a particular technique

aggregate prediction $\hat{y}_i = g_A(\mathbf{x}_i) = \sum_{j=1}^{M} g_j(\mathbf{x}_i), \quad g_j \in \mathcal{F}$ space of tree

cost function $\mathcal{C}(\boldsymbol{X}, g_A) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{j=1}^{M} \Omega(g_j)$

Goodness of prediction (convex and differentiable)

Regularization term
Penalizes large weights on the leaves and large partitions with many leaves

$$\Omega(g) = \gamma T + \frac{\lambda}{2}||\boldsymbol{w}||_2^2$$

Ensemble is formed

$$\hat{y}_i^{(t)} = \sum_{j=1}^{t} g_j(\mathbf{x}_i) = \hat{y}_i^{(t-1)} + g_t(\mathbf{x}_i)$$

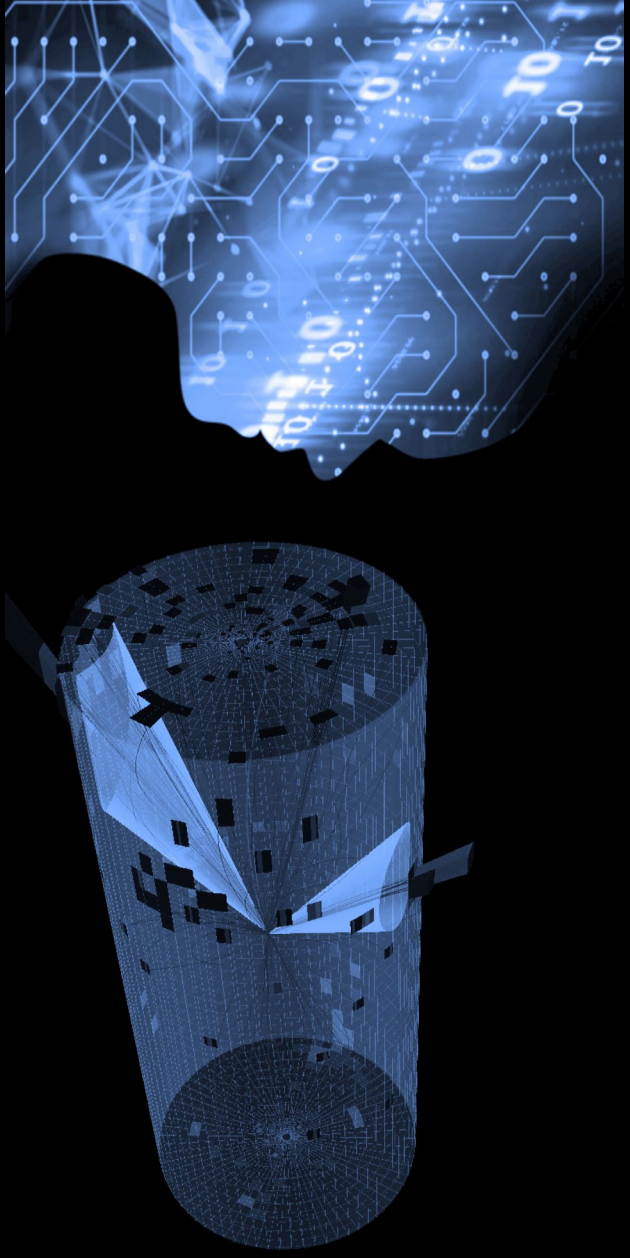
Ensemble Idea: for large t, each decision tree is a small perturbation to the predictor

An approximate greedy algorithm is run to optimize one level of the tree at a time to find optimal splits of the data. Additional regularization such as shrinkage and feature subsampling are also used.

Further Reading

- **Structured vs unstructured data:** Unstructured data can be information that is not arranged according to a pre-set data model or schema. Examples of structured data include names, dates, addresses, credit card numbers, stock information, geolocation, and more. Structured data is highly organized. Ensemble methods perform well especially on structured datasets. Neural networks generally perform better than ensemble methods on unstructured data, images, and audio.
- Feature importance

https://machinelearningmastery.com/calculate-feature-importance-with-python/



Clustering in a nutshell

K-Means

hard-clustering

STEP 1: Choose the number K of clusters

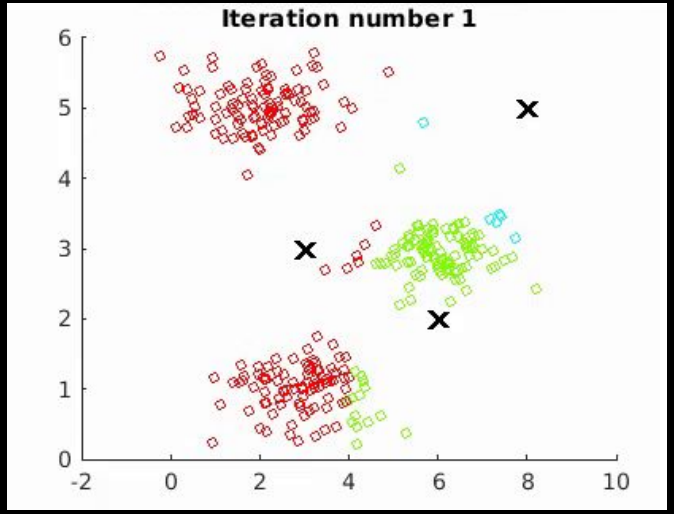
STEP 2: Select at random K points, the centroids
(not necessarily from your dataset)

STEP 3: Assign each data point to the closest centroid
(That forms K clusters)

STEP 4: Compute and place the new centroid of each cluster

STEP 5: Reassign each data point to the new closest centroid
If any reassignment took place, go to STEP 4,
otherwise go to FIN.

Your Model is Ready



K-means clustering can be formulated as: given a fixed integer K, find the cluster means {μ} and the data point assignments in order to minimize the following objective function:

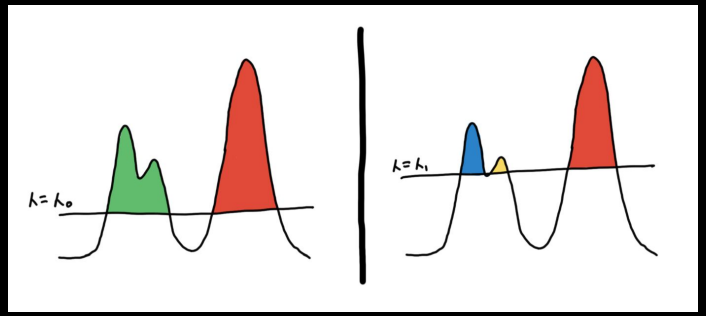
$$\mathcal{C}(\{x, \boldsymbol{\mu}\}) = \sum_{k=1}^{K} \sum_{n=1}^{N} r_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^2$$

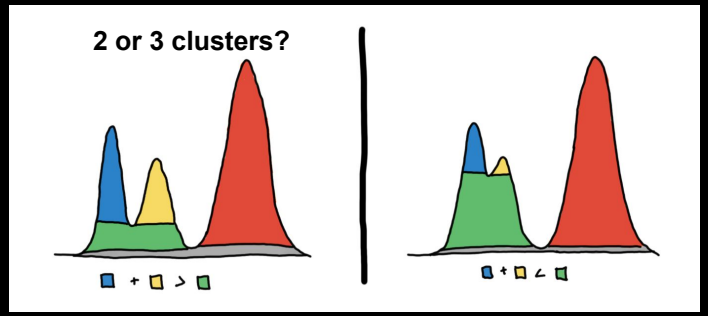
r_{nk}={0,1} binary assignment

Density Based Clustering

soft-clustering

Two different clusterings based on two different level-sets



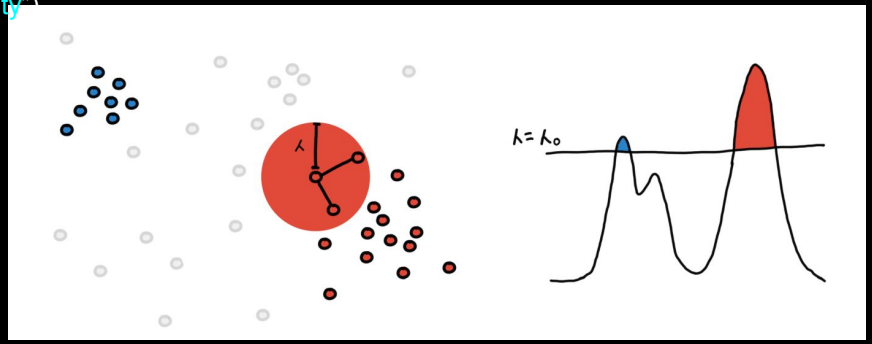


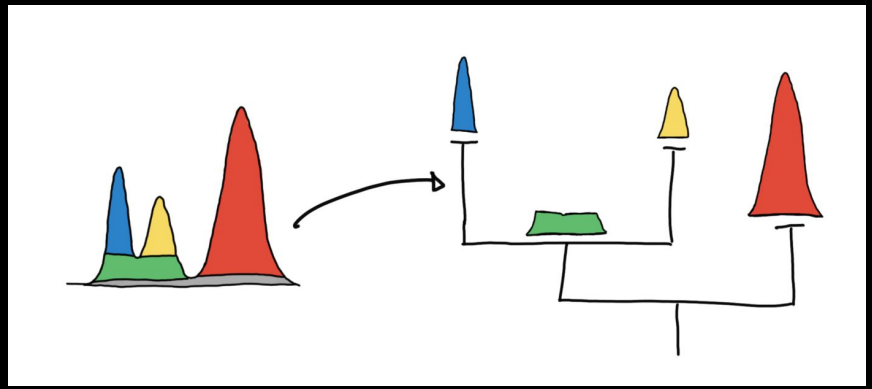
The area of the regions is the measure of “persistence”.

Maximize the persistence of the clusters under the constraint that they do not overlap.

Core distance (defined by a required # of neighbors) as estimate of density

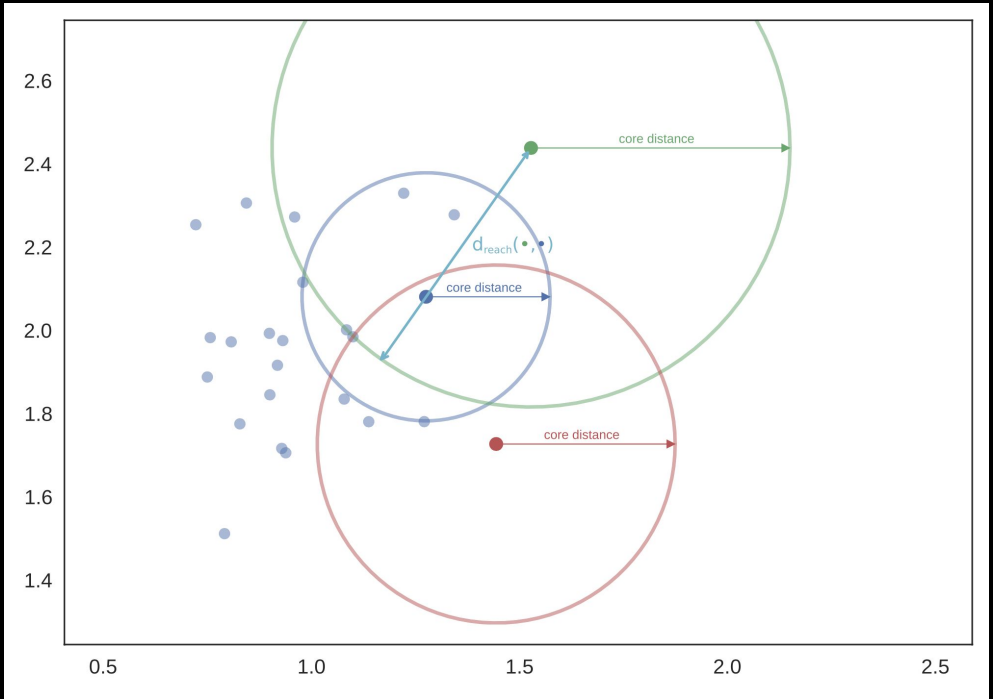
Points have to be in a high density region and close to each other (“mutual reachability”)





clusters as more likely “regions” separated by less likely regions -> densities

Density Based Clustering



Mutual reachability

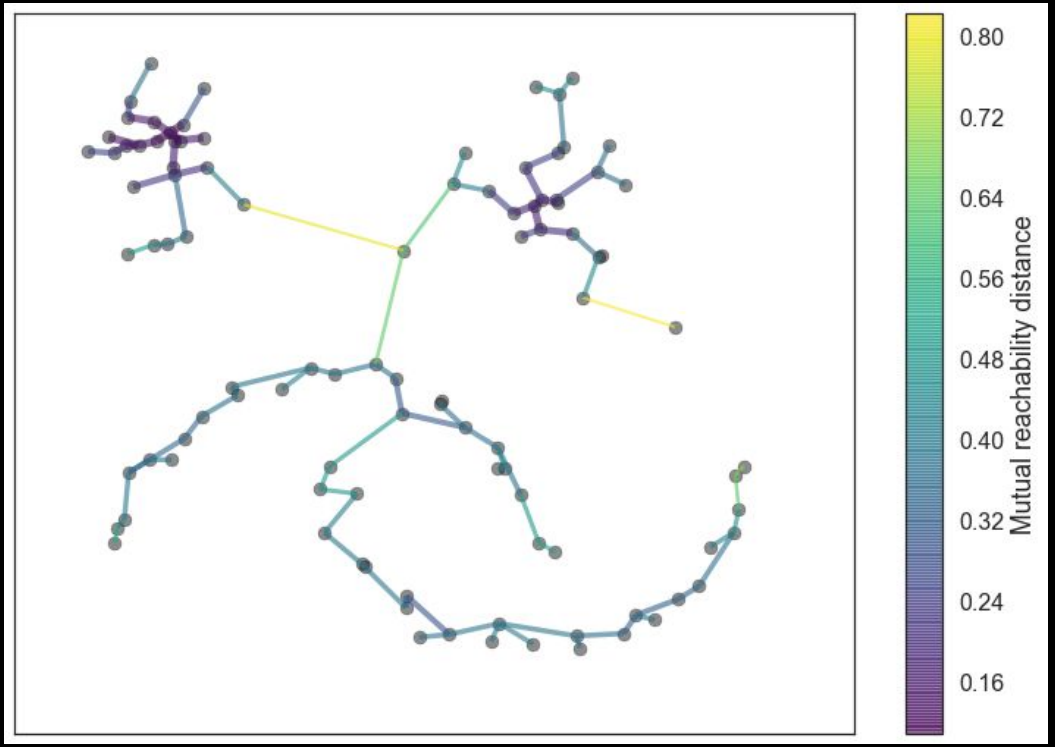
$$d_{mreach-k}(a,b) = max\{core_k(a), core_k(b), d(a,b)\}$$

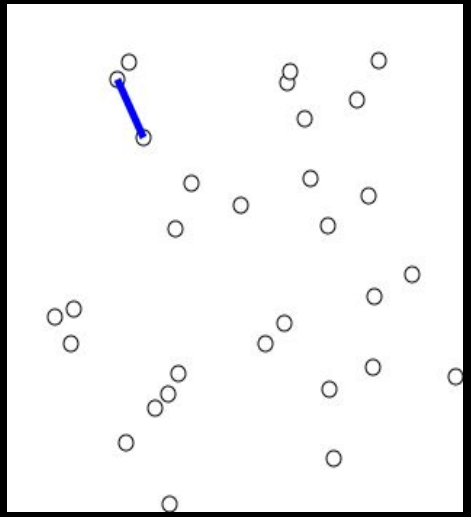
The **mutual reachability** distance is a summary at what level of "λ" two points together will connect. This is what we use as a new metric.

[1] DBSCAN, or density-based spatial clustering of applications with noise [Khan, 2014]
[2] HDBSCAN hierarchical DBSCAN [McInnes, 2017]

Hierarchical Clustering

Minimum spanning tree
https://en.wikipedia.org/wiki/Prim%27s_algorithm



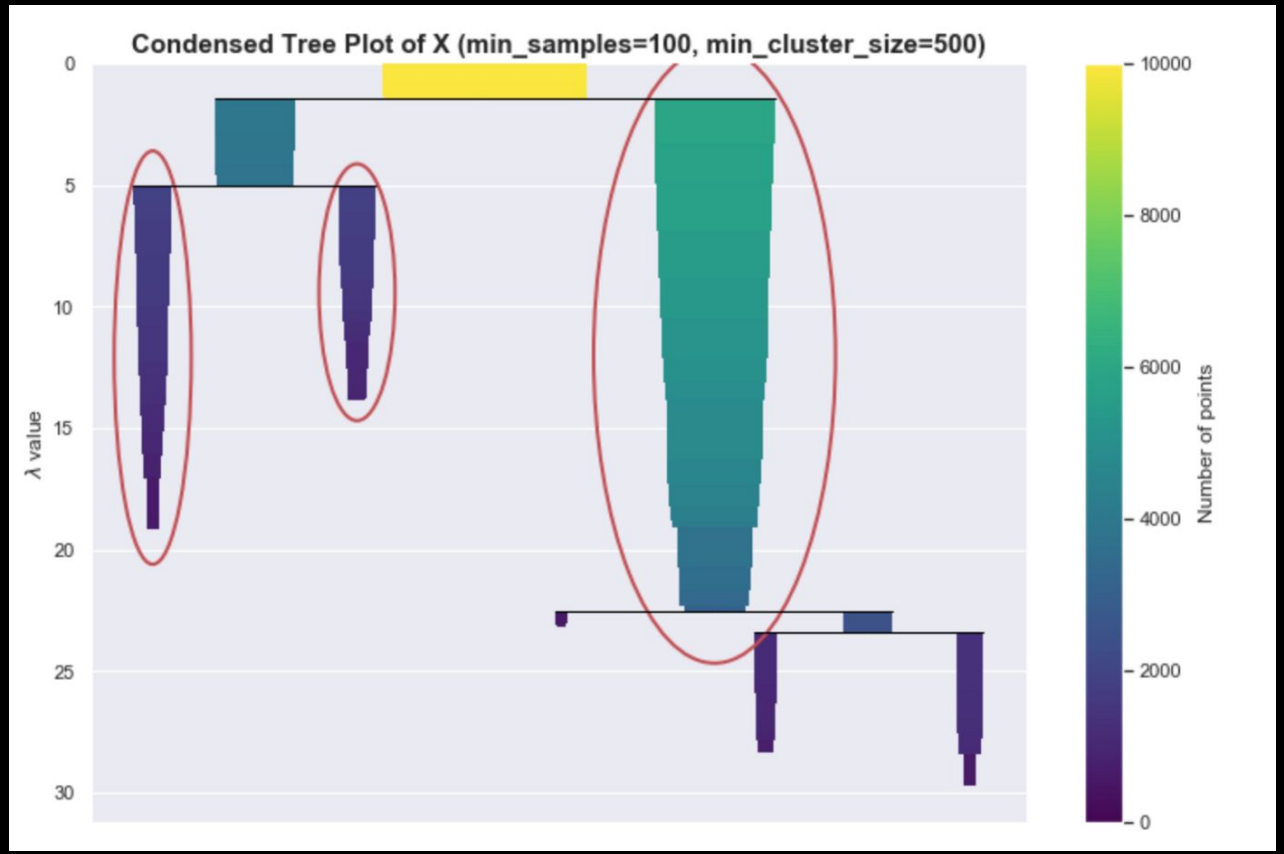


After mutual reachability… consider a threshold value, starting high, and steadily being lowered. Drop any edges with weight above that threshold. As we drop edges we will start to disconnect the graph into connected components. Eventually we will have a hierarchy of components (from completely connected to completely disconnected) at varying threshold levels). In practice this is very expensive: many edges… Fortunately graph theory furnishes us with just such a thing: the minimum spanning tree of the graph.

https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html#build-the-minimum-spanning-tree

Hierarchical Clustering

Minimum spanning tree
https://en.wikipedia.org/wiki/Prim%27s_algorithm



$O(n^2)$

A hierarchy of multiple level-sets is obtained by varying the density threshold

Visualization of the tree top-down as in the literature

hdbscan vs k-means

K-means is a semi-supervised parametric algorithm parameterized by the *K cluster centroids* (if you want K seeds). Can perform if the underlying assumptions on the shape of the clusters are not met. Clusters have to be:

- “round” or spherical
- equally sized
- equally dense
- most dense in the center of the sphere
- not contaminated by noise/outliers

Hdbscan on the other hand is an unsupervised hierarchical clustering which excels when data has:

- Arbitrarily shaped clusters
- Clusters with different sizes and densities
- Noise

PCA

- Ubiquitous method for dimensional reduction
- Inspired by the observation that in many cases, the relevant information in a signal is contained in the directions with largest variance
- Use to reduce dimensionality
- Principal components are orthogonal to each other
- Calculate the Eigenvectors and Eigenvalues of the Covariance Matrix
- We will utilize useful and easy to use python packages for **Principal Component Analysis** to create insightful plots.

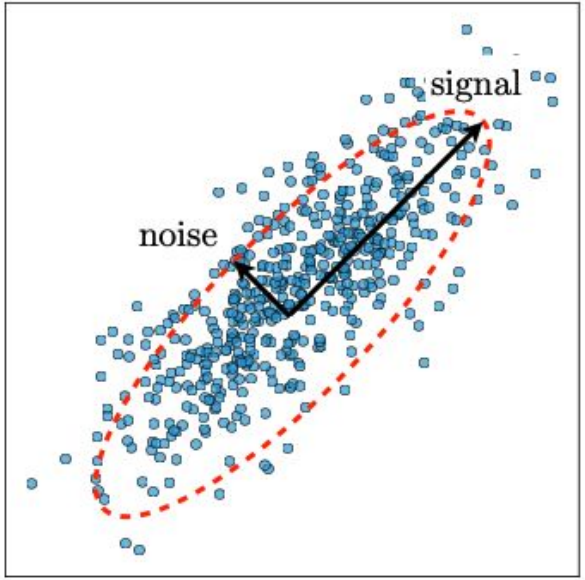


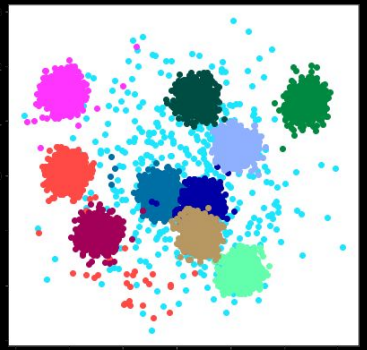
FIG. 50 PCA seeks to find the set of orthogonal directions with largest variance. This can be seen as "fitting" an ellipse to the data with the major axis corresponding to the first principal component (direction of largest variance). PCA assumes that directions with large variance correspond to the true signal in the data while directions with low variance correspond to noise.

[1] Levina, Elizaveta, and Peter J. Bickel. "Maximum likelihood estimation of intrinsic dimension." *Advances in neural information processing systems*. 2005.
[2] PCA for dimensionality reduction: Abdi, Hervé, and Lynne J. Williams. "Principal component analysis." *Wiley interdisciplinary reviews: computational statistics* 2.4 (2010): 433-459.

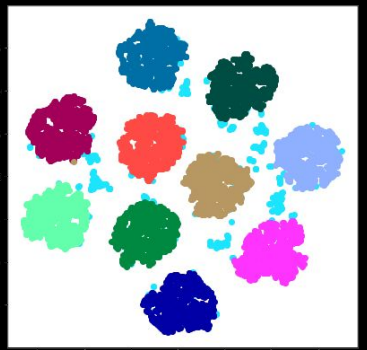
t-SNE

- t-SNE is also a method to reduce the dimension. One of the most major differences between PCA and t-SNE is it preserves only local similarities whereas PA preserves large pairwise distance maximize variance.
- Recently, t-stochastic neighbor embedding (t-SNE) has emerged as one of the go-to methods for visualizing high-dimensional data.
- Each high-dimensional training point is mapped to low-dimensional embedding coordinates, which are optimized in a way to preserve the local structure in the data.
- The idea of stochastic neighbor embedding is to associate a probability distribution to the neighborhood of each data

- t-SNE can rotate dat
- t-SNE results are stochastic (depends on initial seed)
- t-SNE preserves short distance information
- Scales are deformed
- Computationally intensive



PCA



t-SNE

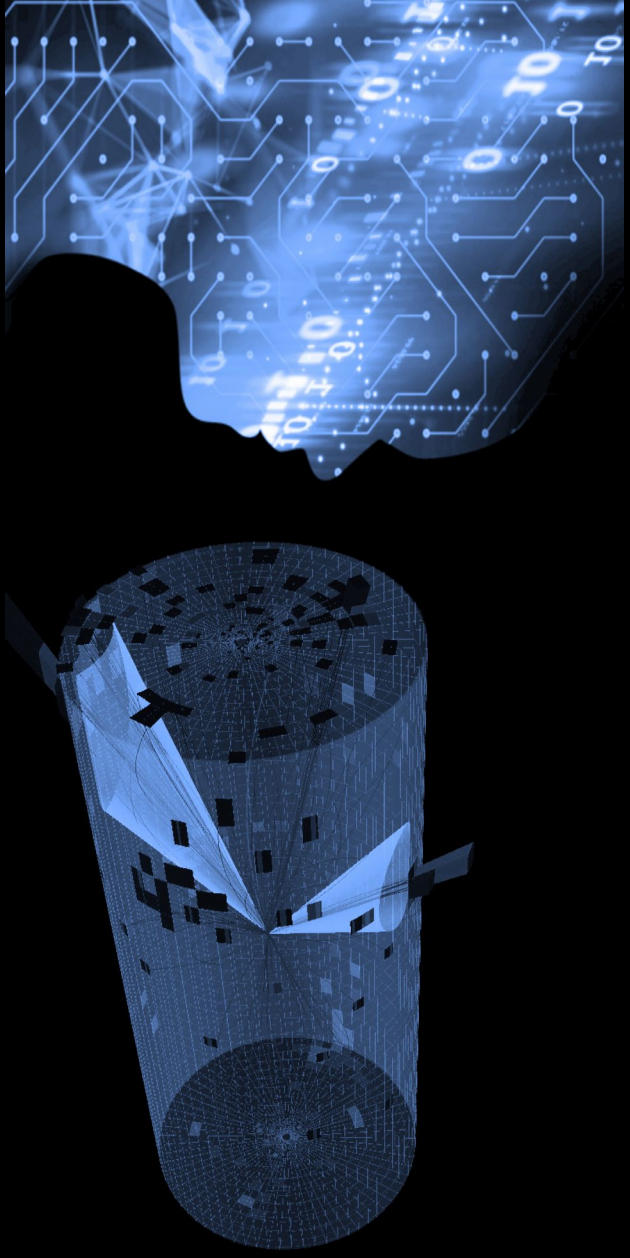
[1] Maaten, Laurens van der, and Geoffrey Hinton (2008), “Visualizing data using t-sne,” Journal of machine learning research 9 (Nov), 2579–2605.

Tips, further reading

- **Hard Clustering**: In hard clustering, **each data point either belongs to a cluster completely or not**.
- **Soft Clustering**: In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned
- **Since clustering algorithms including kmeans use distance-based measurements to determine the similarity between data points, it's recommended to standardize the data to have a mean of zero and a standard deviation of one since almost always the features in any dataset would have different units of measurements [ref]**

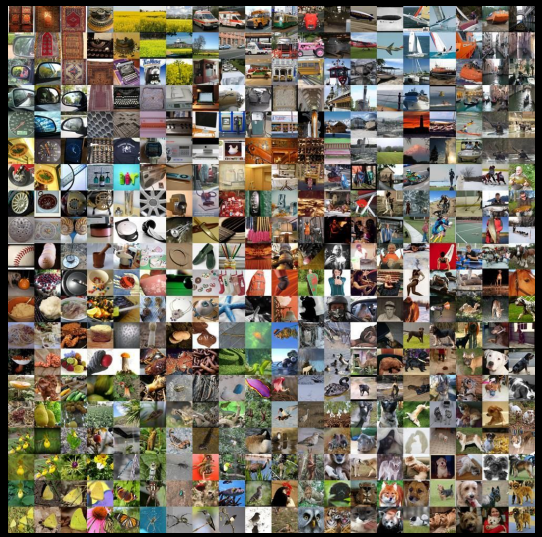
- PCA is at a disadvantage if the data has not been standardized before applying the algorithm to it.
- PCA vs t-SNE: which one to use? [ref]

Day 2



Deep Learning

Intro



- Last decade NN emerged as one of the most powerful and used supervised learning techniques
- NNs have a long history re-emerged to prominence after a rebranding as “Deep Learning” in mid 2000 by Hinton et al.
  - 2012 GPU-based DNN model (AlexNet) lower error rate on ImageNet visual recognition challenge by 12% from 28% to 16%
  - In 2016 a ML group from Microsoft achieved an error of 3.57% using an ultra deep residual neural network (ResNet) with 152 layers
- Large scale industrial deployment of DNNs has given rise to a number of high-level libraries and packages (Caffe, Keras, Pytorch, TensorFlow, etc) that made easy to code and deploy DNNs
- Conceptually can be divided into 4 categories:
  - General purpose NN for supervised learning
  - NN for image processing such as CNN
  - NN for sequential data such as RNN
  - NN for unsupervised learning

Rapidly evolving field, lots of papers. Implementation urged for deep appreciation of the power of these approaches but also of their limitations.
Their usage ubiquitous, and covering a breadth of areas, spanning from LQCD to experimental neutrino physics.

G. Hinton et al. (2006), “A fast learning algorithm for deep belief nets,” Neural computation 18 (7), 1527–1554

Building Blocks

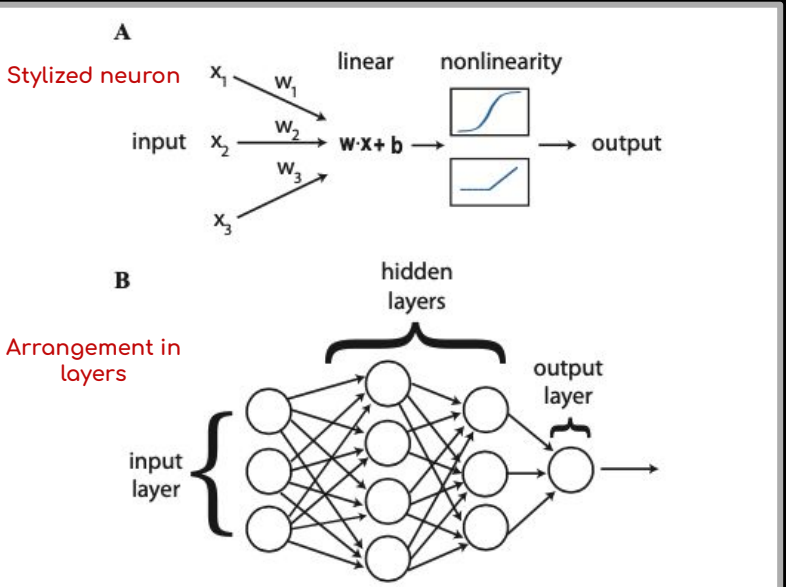


FIG. 35 **Basic architecture of neural networks.** (A) The basic components of a neural network are stylized neurons consisting of a linear transformation that weights the importance of various inputs, followed by a non-linear activation function. (b) Neurons are arranged into layers with the output of one layer serving as the input to the next layer.

- The basic unit of a neural network is a stylized neuron *i* that takes *d* input features $= (x_1, x_2, \ldots, x_d)$ and produce a scalar output $a_i(x)$
- First layer is called input layer, middle layers are called hidden layers, final layer is the output layer
- The exact function a_i depends on the type of non-linearity used in the NN
  - It can be decomposed into a linear operation that weights the relative importance of the various inputs and a non-linear transformation $\sigma_i(z)$

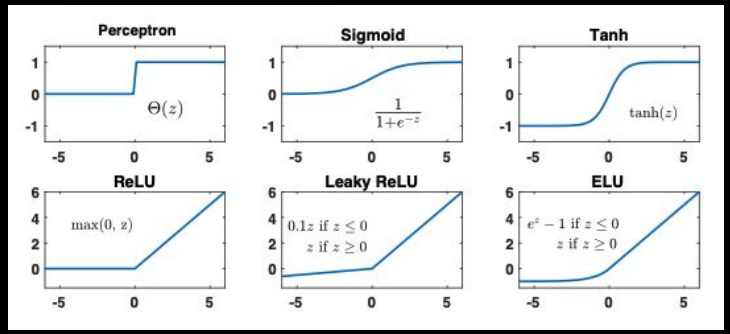
weights bias

$$z^{(i)} = \boldsymbol{w}^{(i)} \cdot \boldsymbol{x} + b^{(i)} = \mathbf{x}^T \cdot \mathbf{w}^{(i)}$$

inputs

activation

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)})$$ (non-linear functions)



Building Blocks

- Different choices of non-linear functions lead to different computational and training properties for neurons
- Remember we train NN using gradient descent based methods, that require us to take derivatives of neural input-output function with respect to weights $w^{(i)}$ and the bias $b^{(i)}$
- The derivatives of the non-linearities σ(z) have very different properties. (e.g. perceptrons have a discontinuous behavior and we cannot use gradient descent; for this reason, until recently, **tanh** and **sigmoid** was one of the most popular choices)
- However when the weights become large these activations saturate and the derivatives tends to zero (vanishing gradients). In contrast, for non-saturating activation function such as **ReLU** or **ELU**, the gradients stay finite even for large inputs.

https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

- The basic unit of a neural network is a stylized neuron *i* that takes *d* input features = $(x_1, x_2, \ldots, x_d)$ and produce a scalar output $a_i(x)$

x

- First layer is called input layer, middle layers are called hidden layers, final layer is the output layer
- The exact function a_i depends on the type of non-linearity used in the NN
  - It can be decomposed into a linear operation that weights the relative importance of the various inputs and a non-linear transformation $\sigma_i(z)$

weights bias

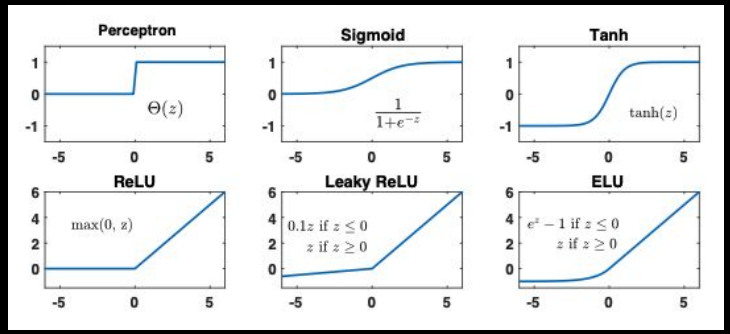
$$z^{(i)} = w^{(i)} \cdot x + b^{(i)} = \mathbf{x}^T \cdot \mathbf{w}^{(i)}$$

inputs

activation

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)})$$

(non-linear functions)



Layering Neurons: Network Architectures

- The use of hidden layers greatly expands the representational power of a neural net when compared with a simple soft-max or linear regression network.
- The most formal expression of the increased representational power of neural networks (also called the expressivity) is the universal approximation theorem which states that a neural network with a single hidden layer can approximate any continuous, multi-input/multi-output function with arbitrary accuracy.
- The more complicated a function, the more hidden units (and free parameters) are needed to approximate it.
- Adding hidden layers is also thought to allow neural nets to learn more complex features from the data.
- Choosing the exact network architecture for a neural network remains an art

Nielsen, Michael A (2015), Neural networks and deep learning

Training DNN

- The basic procedure to train neural nets is the same we used for training simpler supervised algorithm such as linear and logistic regression (construct cost/loss function and then use the gradient descent to minimize the cost and find optimal weights and biases)
- NN differs from simpler procedure in that contain multiple hidden layers that make taking the gradient computationally more difficult. We'll discuss the "backpropagation" algorithm in the next slides
- Given a data point $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in R^d$, the neural network makes a prediction $\hat{y}_i(\mathbf{w})$ where $\mathbf{w}$ are the parameters of the NN.
- Remember that depending on whether one wants to make continuous or categorical predictions, one must utilize a different kind of loss functions:
  - For continuous data we have see in linear regression the mean squared error

$$E(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(\mathbf{w}))^2$$

  - For categorical data, the most commonly used loss function is cross-entropy

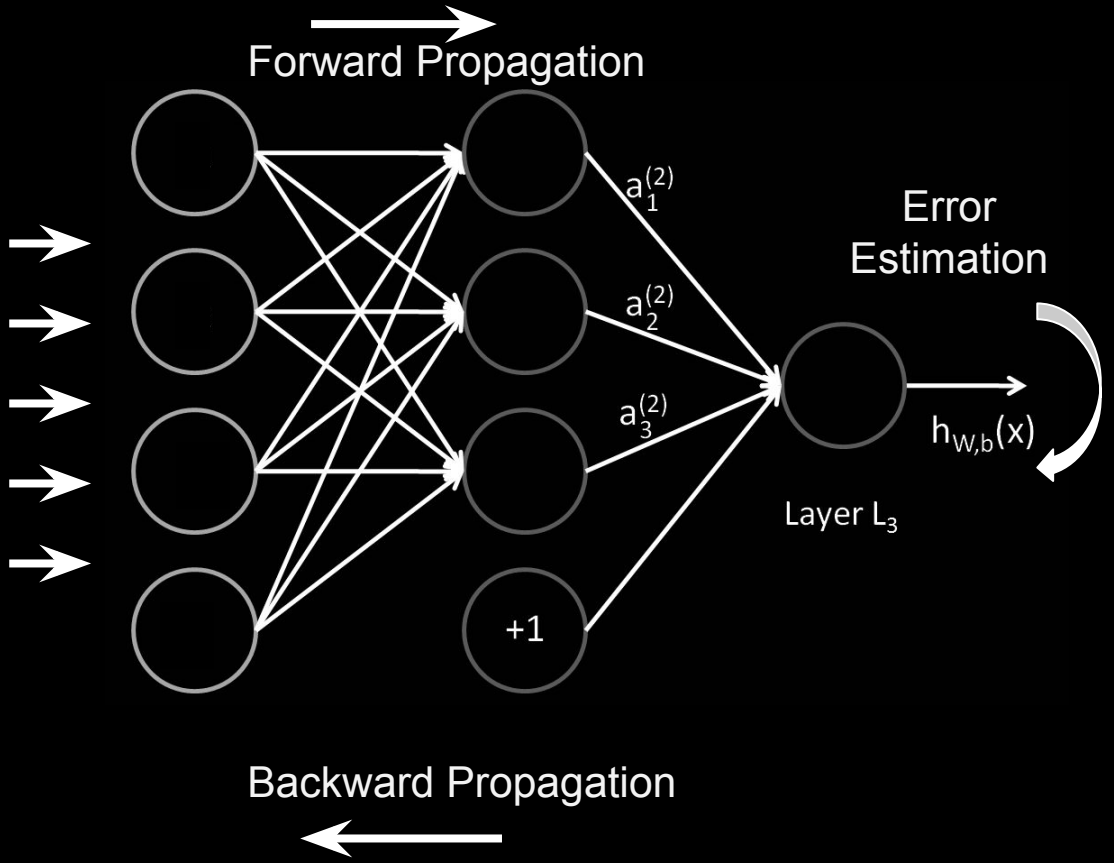
$$E(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i(\mathbf{w}) + (1 - y_i) \log\left[1 - \hat{y}_i(\mathbf{w})\right]$$

$$y \in \{0, 1, \ldots, M-1\}$$

$$E(\mathbf{w}) = -\sum_{i=1}^{n}\sum_{m=0}^{M-1} y_{im} \log \hat{y}_{im}(\mathbf{w}) + (1 - y_{im}) \log\left[1 - \hat{y}_{im}(\mathbf{w})\right]$$

Categorical cross-entropy

(from previous slides)



- The real magic about NN is the result of an optimization technique: back-propagation (how a NN works to improve its output over time)
- DL (**adding more hidden layers**) nets are good in learning non-linear functions (heavy processing tasks)
- Based on old school NN revitalized by augmented capabilities (e.g. GPU) and a plethora of new architectures (RNN, CNN, autoencoders, GAN, etc.)

Backpropagation

procedure that exploits the layered structure of neural networks to more efficiently compute gradients

Backpropagation Algorithm

1. **Activation at input layer:** calculate the activations a_j^1 of all the neurons in the input layer.
2. **Feedforward:** starting with the first layer, exploit the feed-forward architecture through Eq. (123) to compute z^l and a^l for each subsequent layer.
3. **Error at top layer:** calculate the error of the top layer using Eq. (I). This requires to know the expression for the derivative of both the cost function $E(\mathbf{w}) = E(\boldsymbol{a}^L)$ and the activation function $\sigma(z)$.
4. **"Backpropagate" the error:** use Eq. (III) to propagate the error backwards and calculate Δ_j^l for all layers.
5. **Calculate gradient:** use Eqs. (II) and (IV) to calculate $\frac{\partial E}{\partial b_j^l}$ and $\frac{\partial E}{\partial w_{jk}^l}$.

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma(z_j^l), \qquad (123) \qquad z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial a_j^l}\sigma'(z_j^l), \qquad (\mathrm{I})$$

derivative of σ

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_k \frac{\partial E}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$= \sum_k \Delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$= \left(\sum_k \Delta_k^{l+1} w_{kj}^{l+1}\right)\sigma'(z_j^l). \qquad (\mathrm{III})$$

Apply chain rule
(The error depends on neurons in layer l only through the activation of neurons in subsequent layer l+1)

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial b_j^l}\frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial E}{\partial b_j^l}, \qquad (\mathrm{II})$$

$$\frac{\partial E}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1} \qquad (\mathrm{IV})$$

See [hblvi2ml, Sec IX.C] for more details

Rumelhart, David E, and David Zipser (1985), "Feature discovery by competitive learning," Cognitive science 9 (1), 75–112.

Regularization

See [hblvi2ml, Sec IX.D] for more details

- DNN as all supervised learning algorithm must navigate the bias-variance tradeoff. Regularization techniques play an important role in ensuring that DNNs generalize well to new data.
- There is a wealth of new regularization techniques for DNNs beyond the simple L_1 and L_2 penalties
  - Implicit regularization using Stochastic Gradient Descent: initialization, hyperparmeter tuning, early stopping
  - Dropout (a cheaper alternative to ensembling)
  - Batch normalization

Introduce new batch-norm layers that standardize inputs by mean and variance of mini-batch

Prevents neurons from saturating

$$z_k^l \longrightarrow \hat{z}_k^l = \frac{z_k^l - \mathbb{E}[z_k^l]}{\sqrt{\mathrm{Var}[z_k^l]}}$$

$$\hat{z}_k^l \longrightarrow \hat{\mathrm{z}}_k^l = \gamma_k^l \hat{z}_k^l + \beta_k^l$$

New parameters introduced for each neuron

New improved learning speed

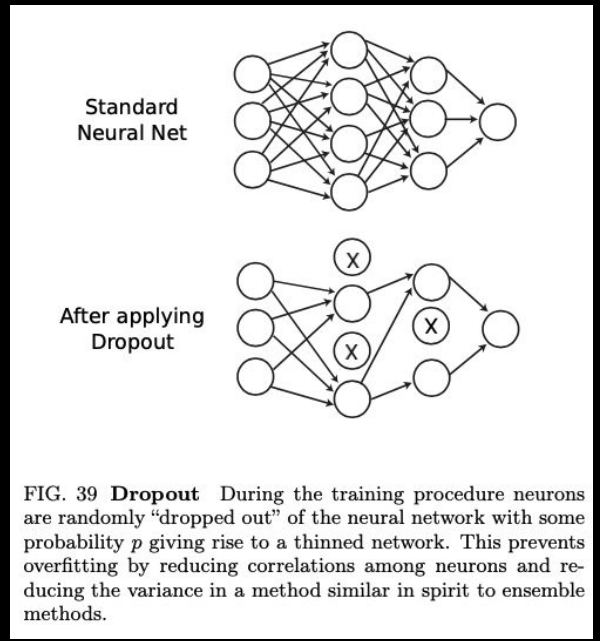


FIG. 39 **Dropout** During the training procedure neurons are randomly "dropped out" of the neural network with some probability p giving rise to a thinned network. This prevents overfitting by reducing correlations among neurons and reducing the variance in a method similar in spirit to ensemble methods.

Ioffe, Sergey, and Christian Szegedy (2015), "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in International Conference on Machine Learning, pp. 448–456.

Approaching the learning problem

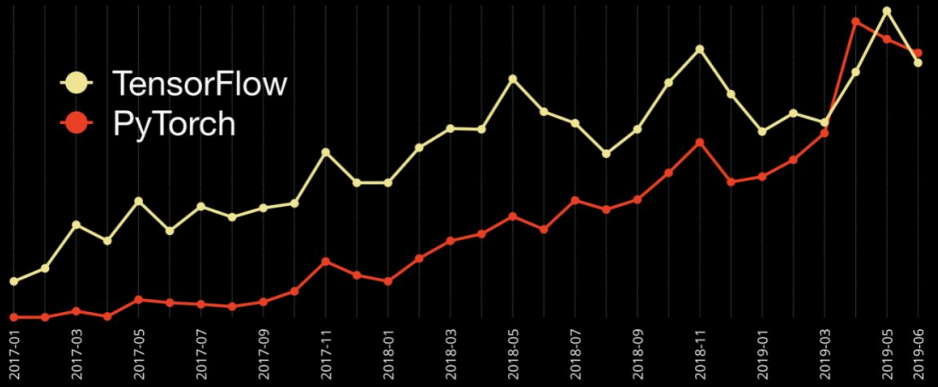
- Collect and pre-process the data.
- Define the model and its architecture.
- Choose the cost function and the optimizer.
- Train the model.
- Evaluate and study the model performance on the test data.
- Use the validation data to adjust the hyper-parameters (and, if necessary, network architecture) to optimize performance for the specific dataset.

Tips, further reading

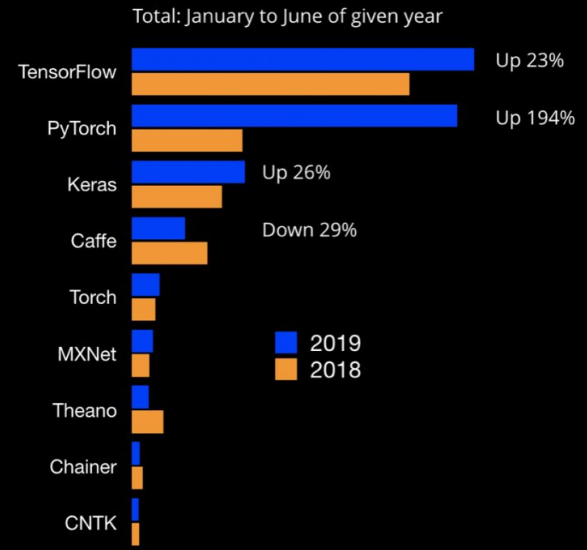
- https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/
- the most formal expression of the increased representational power of neural networks (also called the expressivity) is the universal approximation theorem which states that a neural network with a single hidden layer can approximate any continuous, multi-input/multi-output function with arbitrary accuracy. The reader is strongly urged to read the beautiful graphical proof of the theorem in Chapter 4 of Nielsen's free online book (Nielsen, Michael A (2015), Neural networks and deep learning).
- While the universal approximation property holds both for hierarchical and shallow networks, deep (multi-hidden layers) networks can **approximate the class of compositional functions as well as shallow networks (one hidden)** but with lower number of training parameters and sample complexity.
- Depending on the architecture, data, and computational resources, different optimizers may work better on the problem, though vanilla SGD is a good first choice
- Autograd: is PyTorch's automatic differentiation engine that powers neural network training. https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

PyTorch vs Tensorflow

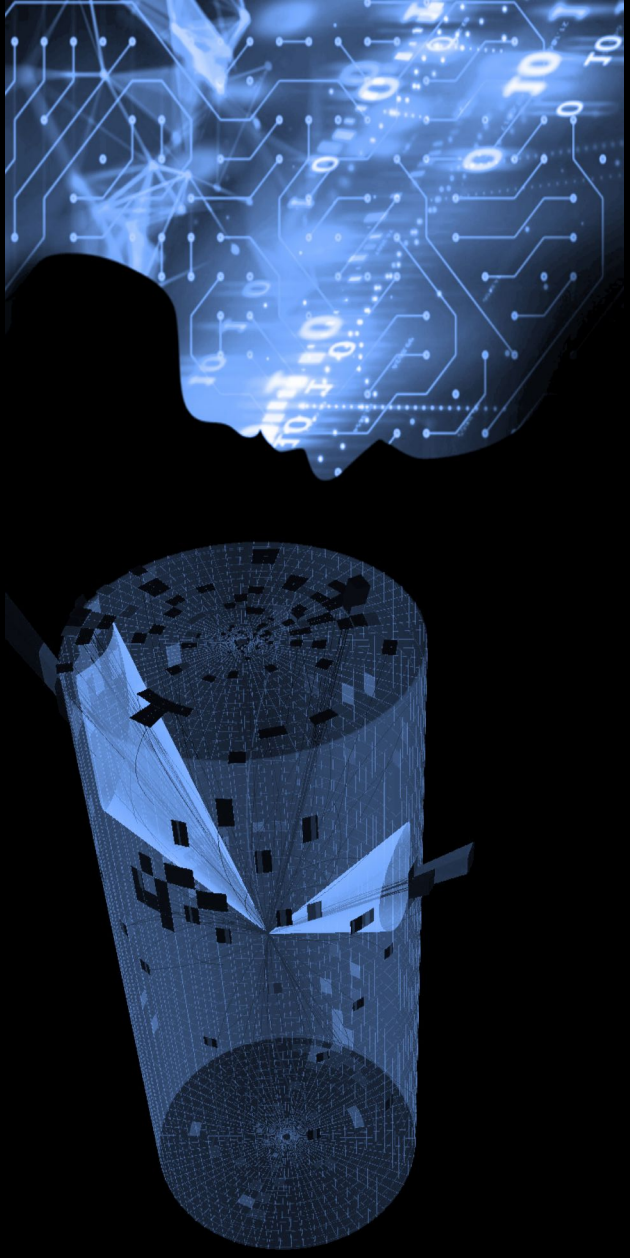
Number of papers on arxiv.org that mention a given framework



Source: Data from RISELab ; graphic from gradientflow.com



https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b



Design Optimization

AI for Design

It is a relatively new but active area of research.
Many applications in, e.g., industrial material, molecular and drug design.

Z. Zhou et al., *Scientific Reports*, vol. 9, no. 1, pp. 1–10, 2019

Guo, Kai, et al. *Materials Horizons* 8.4 (2021): 1153-1172.

Table 1 Popular ML methods in design of mechanical materials

ML method	Characteristics	Example applications in mechanical materials design
Linear regression; polynomial regression	Model the linear or polynomial relationship between input and output variables	Modulus[112] or strength[123] prediction
Support vector machine; SVR	Separate high-dimensional data space with one or a set of hyperplanes	Strength[123] or hardness[125] prediction; structural topology optimization[159]
Random forest	Construct multiple decision trees for classification or prediction	Modulus[112] or toughness[130] prediction
Feedforward neural network (FFNN); MLP	Connect nodes (neurons) with information flowing in one direction	Prediction of modulus,[97,112] strength,[93] toughness[130] or hardness;[97] prediction of hyperelastic or plastic behaviors;[143,145] identification of collision load conditions;[147] design of spinodoid metamaterials[163]
CNNs	Capture features at different hierarchical levels by calculating convolutions; operate on pixel-based or voxel-based data	Prediction of strain fields[104,105] or elastic properties[102,103] of high-contrast composites, modulus of unidirectional composites,[136] stress fields in cantilevered structures,[137] or yield strength of additive-manufactured metals;[121] prediction of fatigue crack propagation in polycrystalline alloys;[140] prediction of crystal plasticity;[120] design of tessellate composites;[107–109] design of stretchable graphene kirigami;[155] structural topology optimization[156–158]
Recurrent neural network (RNN); LSTM; GRU	Connect nodes (neurons) forming a directed graph with history information stored in hidden states; operate on sequential data	Prediction of fracture patterns in crystalline solids;[114] prediction of plastic behaviors in heterogeneous materials;[142,144] multi-scale modeling of porous media[173]
Generative adversarial networks (GANs)	Train two opponent neural networks to generate and discriminate separately until the two networks reach equilibrium; generate new data according to the distribution of training set	Prediction of modulus distribution by solving inverse elasticity problems;[138] prediction of strain or stress fields in composites;[139] composite design;[164] structural topology optimization;[165–167] architected materials design[115]
Gaussian process regression (GPR); Bayesian learning	Treat parameters as random variables and calculate the probability distribution of these variables; quantify the uncertainty of model predictions	Modulus[122] or strength[123,124] prediction; design of supercompressible and recoverable metamaterials[110]
Active learning	Interacts with a user on the fly for labeling new data; augment training data with post-hoc experiments or simulations	Strength prediction[124]
Genetic or evolutionary algorithms	Mimic evolutionary rules for optimizing objective function	Hardness prediction;[126] designs of active materials;[160,161] design of modular metamaterials[162]
Reinforcement learning	Maximize cumulative awards with agents reacting to the environments.	Deriving microstructure-based traction-separation laws[174]
Graph neural networks (GNNs)	Operate on non-Euclidean data structures; applicable tasks include link prediction, node classification and graph classification	Hardness prediction;[127] architected materials design[168]

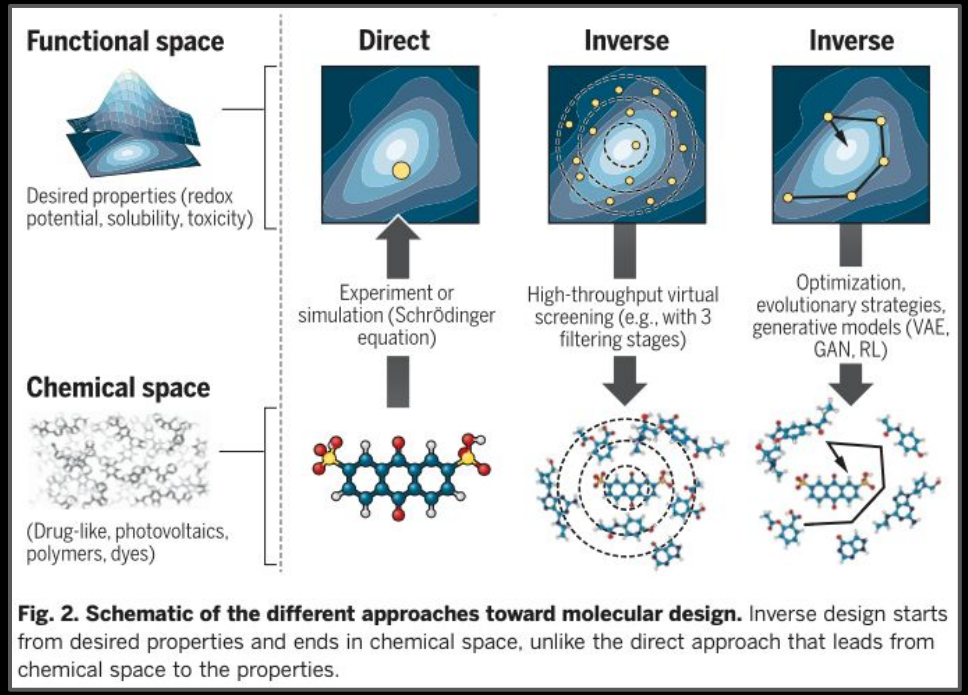


Fig. 2. Schematic of the different approaches toward molecular design. Inverse design starts from desired properties and ends in chemical space, unlike the direct approach that leads from chemical space to the properties.

B. Sanchez-Lengeling, A. Aspuru-Guzik. *Science* 361.6400 (2018): 360-365.

AI for Experimental Design in NP and HEP

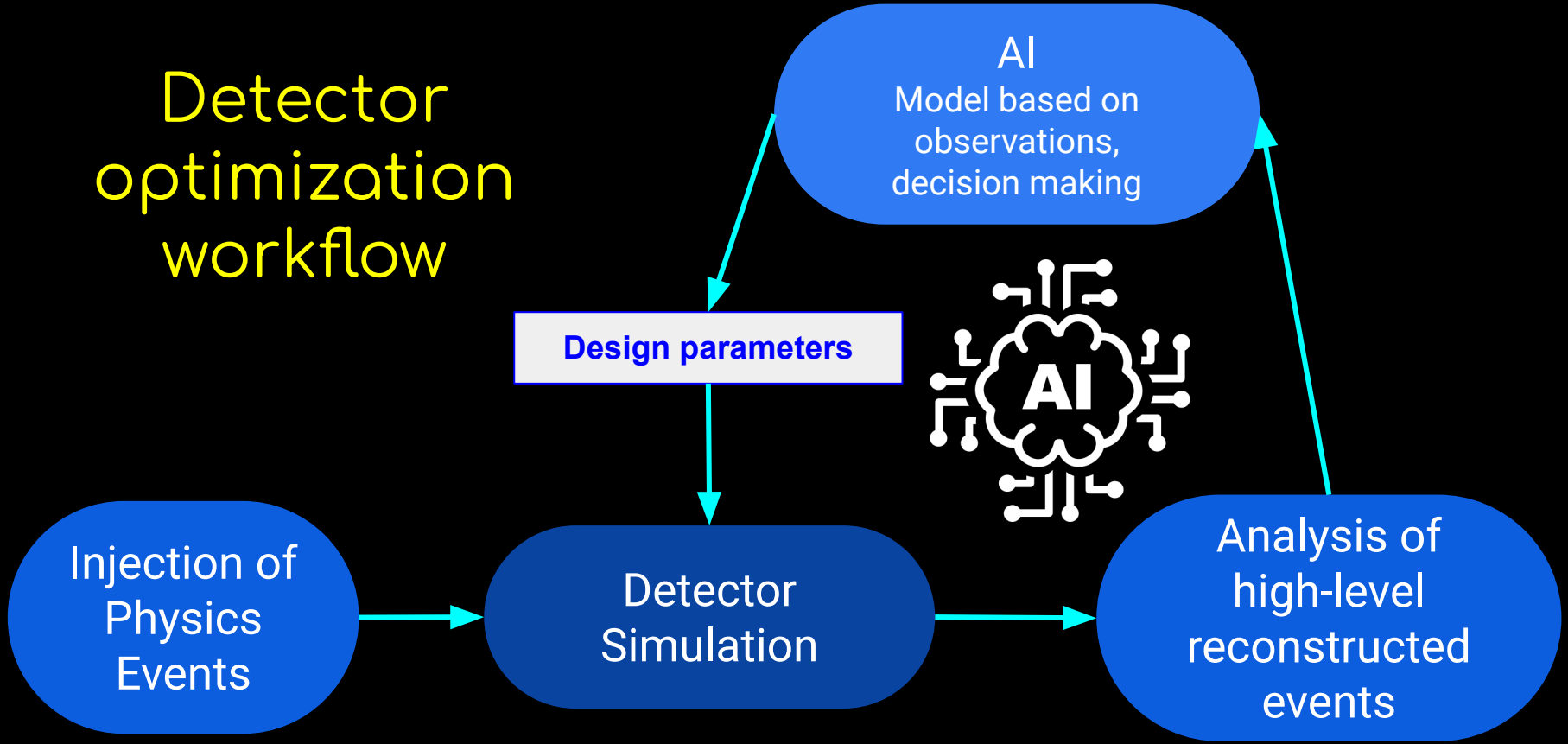
- When it comes to designing detectors and accelerators with AI this is an area at its “infancy”. What follows uses “detector” as example but applies to both detector and accelerator.
- Typically full detector design is studied once the subsystem prototypes are ready (phase constraints from the full detector or outer layers are taken into consideration).
- Need to use advanced simulations which are computationally expensive (Geant).
- Many parameters (and multiple objective functions): curse of dimensionality [1].
- Entails establishing a procedural body of instructions [2].
- The choice of a suitable algorithm is a challenge itself (no free lunch theorem [3]) and always requires some degree of customization.
- Non-differentiable terms.

AI offers SOTA solutions to solve complex optimization problems in an efficient way

What follows largely based on a series of lectures on Detector Design with AI at the AI4NP Winter School

[1] Bellman, Richard. *Dynamic programming*. Vol. 295. RAND CORP SANTA MONICA CA, 1956.
[2] CF et al. *JINST* 15.05 (2020): P05009.
[3] Wolpert, D.H., Macready, W.G., 1997. Trans. Evol. Comp 1, 67–82

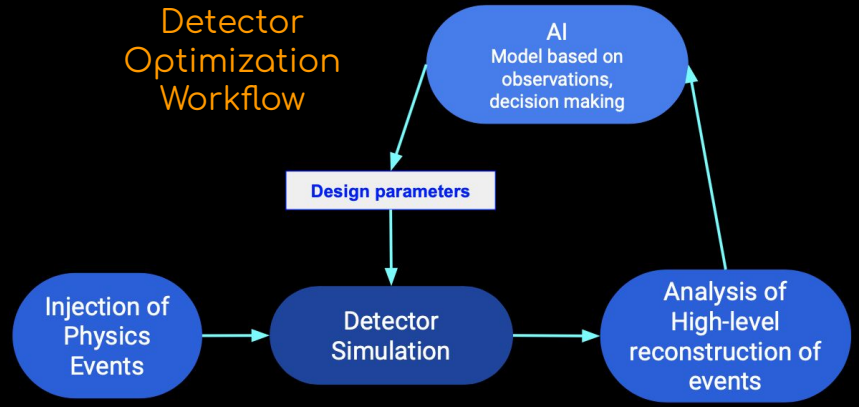
Detector optimization workflow



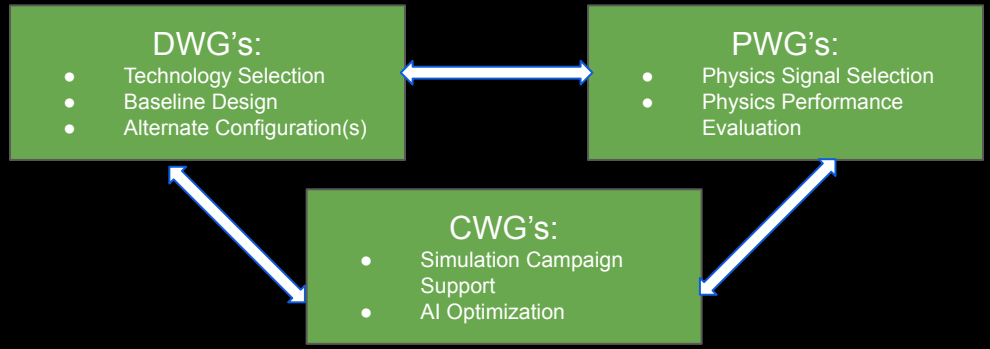
Why AI-assisted?

Optimization does not mean necessarily “fine-tuning”

- We want to use these algorithms to: (1) steer the design and suggest parameters that a “manual”/brute-force optimization will likely miss to identify; (2) further optimize some particular detector technology (see d-RICH paper, e.g., optics properties)
- AI allows to capture hidden correlations among the design parameters.
- All “steps” (physics, detector) involved in the AI optimization, strong interplay between working groups

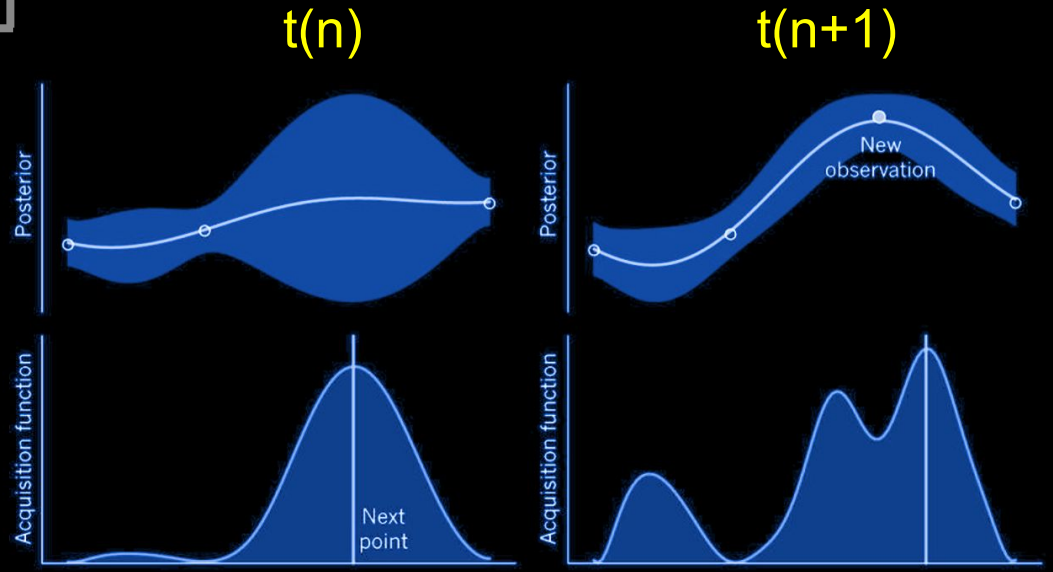


AI promotes Interaction among Working Groups



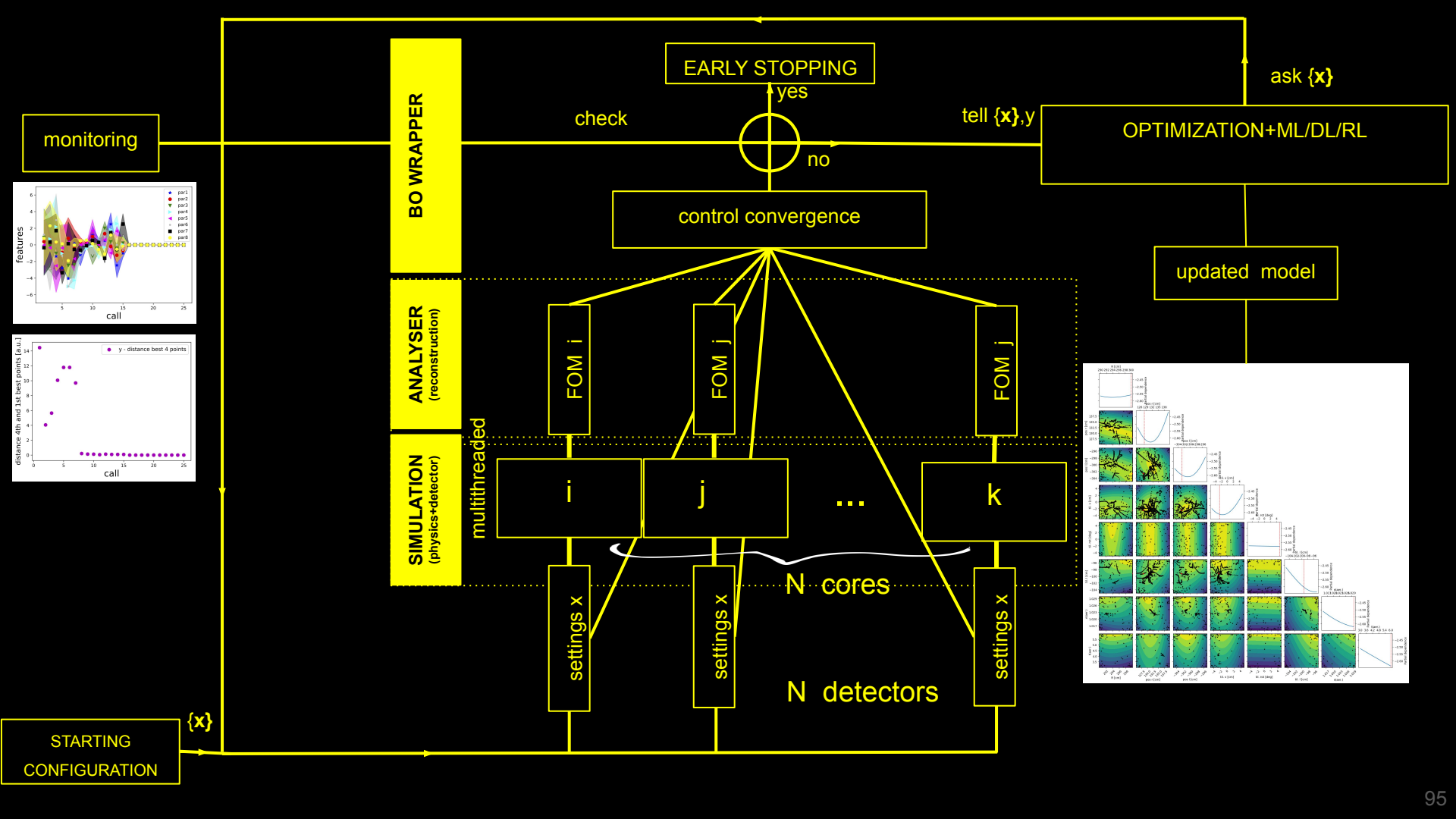
Bayesian Optimization

- BO is a sequential strategy developed for global optimization.
- After gathering evaluations we builds a posterior distribution used to construct an **acquisition function**.
- This cheap function determines what is **next query point**.



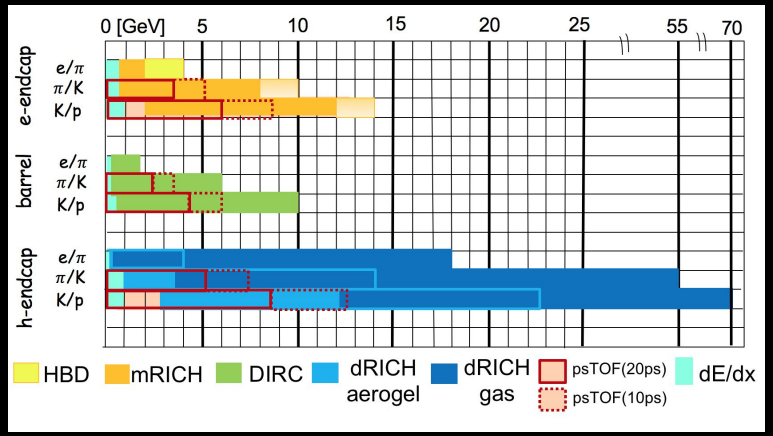
1. Select a Sample by Optimizing the Acquisition Function.
2. Evaluate the Sample With the Objective Function.
3. Update the Data and, in turn, the Surrogate Function.
4. Go To 1.

http://krasserm.github.io/2018/03/21/bayesian-optimization/
http://krasserm.github.io/2018/03/19/gaussian-processes/


monitoring
BO WRAPPER
EARLY STOPPING
yes
check
no
tell {x},y
ask {x}
OPTIMIZATION+ML/DL/RL
control convergence
updated model
ANALYSER
(reconstruction)
SIMULATION
(physics+detector)
multithreaded
FOM i
FOM j
FOM j
i
j
...
k
N cores
settings x
settings x
settings x
N detectors
STARTING
CONFIGURATION
{x}
features
call
distance 4th and 1st best points [a.u.]
y - distance best 4 points
call

dual RICH case

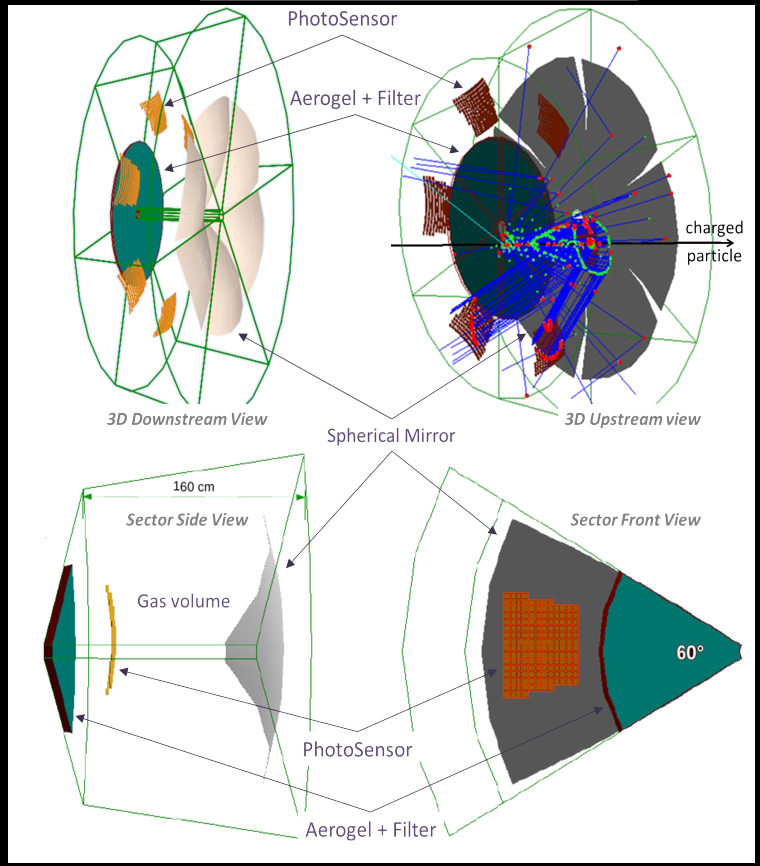
E. Cisbani, A. Del Dotto, CF*, M. Williams et al.
"AI-optimized detector design for the future Electron-Ion Collider: the dual-radiator RICH case."
Journal of Instrumentation 15.05 (2020): P05009.



- Continuous momentum coverage.
- Simple geometry and optics, cost effective.
- Legacy design from INFN, see EICUG2017

- 6 Identical open sectors (petals)
- Optical sensor elements:
  8500 cm^2/sector, 3 mm pixel
- Large focusing mirror

aerogel (4 cm, n(400 nm): 1.02)
+ 3 mm acrylic filter
+ gas (1.6 m, n(C_2F_6): 1.0008)

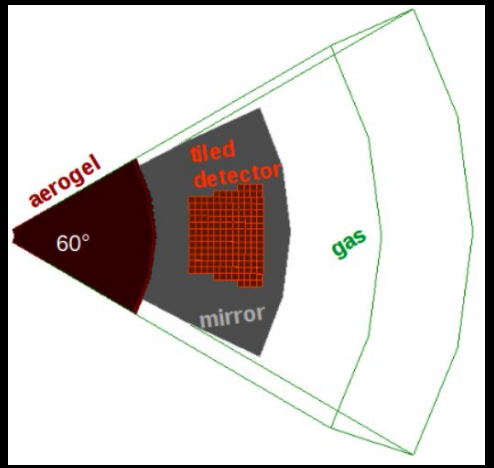


Construction Constraints

The idea is that we have a bunch of parameters to optimize that characterize the detector design.
We know from previous studies their ranges and the construction tolerances.

Variations below these values are irrelevant

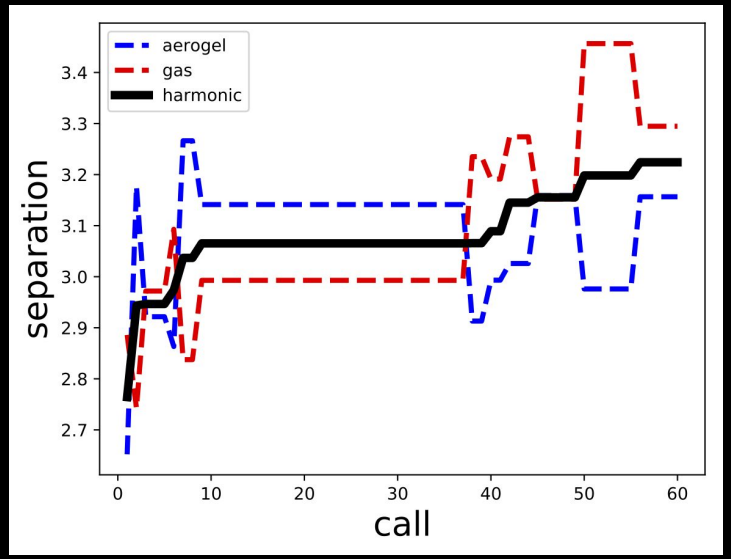
parameter	description	range [units]	tolerance [units]
R	mirror radius	[290,300] [cm]	100 [μm]
pos r	radial position of mirror center	[125,140] [cm]	100 [μm]
pos l	longitudinal position of mirror center	[-305,-295] [cm]	100 [μm]
tiles x	shift along x of tiles center	[-5,5] [cm]	100 [μm]
tiles y	shift along y of tiles center	[-5,5] [cm]	100 [μm]
tiles z	shift along z of tiles center	[-105,-95] [cm]	100 [μm]
$n_{aerogel}$	aerogel refractive index	[1.015,1.030]	0.2%
$t_{aerogel}$	aerogel thickness	[3.0,6.0] [cm]	1 [mm]

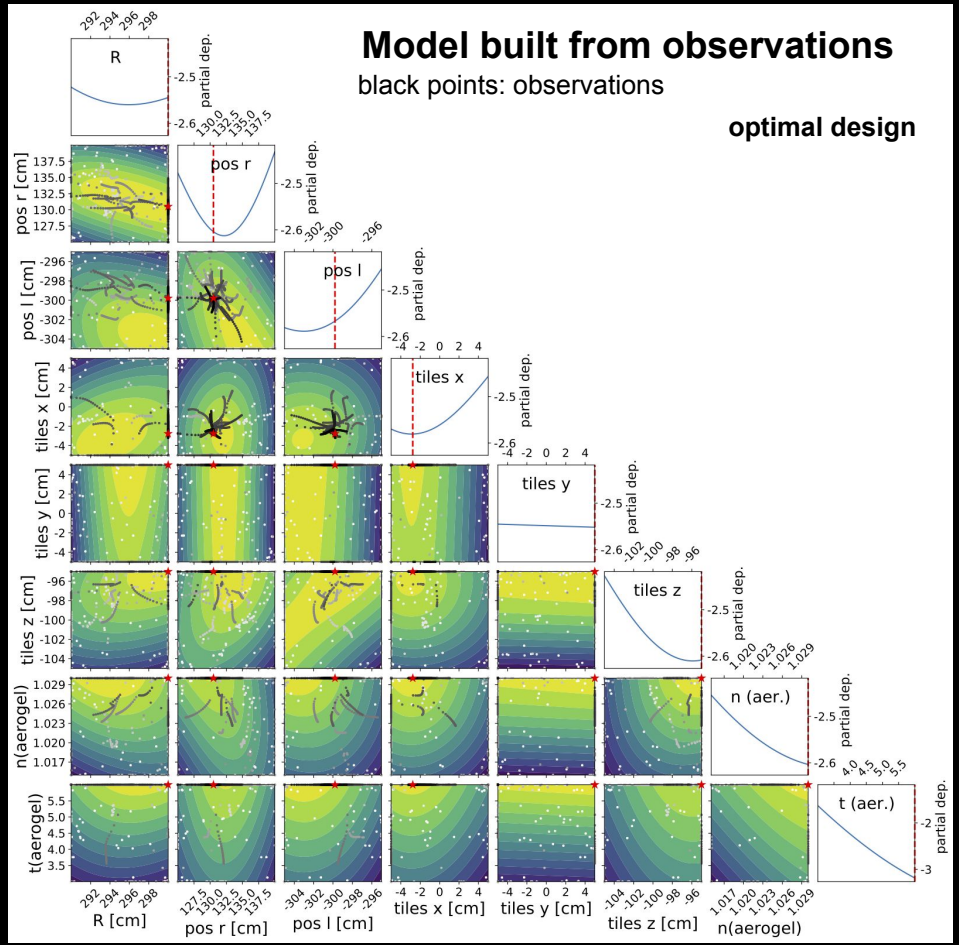


Ranges depend mainly on mechanical constraints and optics requirements.
These requirements can change in the next future based on inputs from prototyping.

Optimized FoM

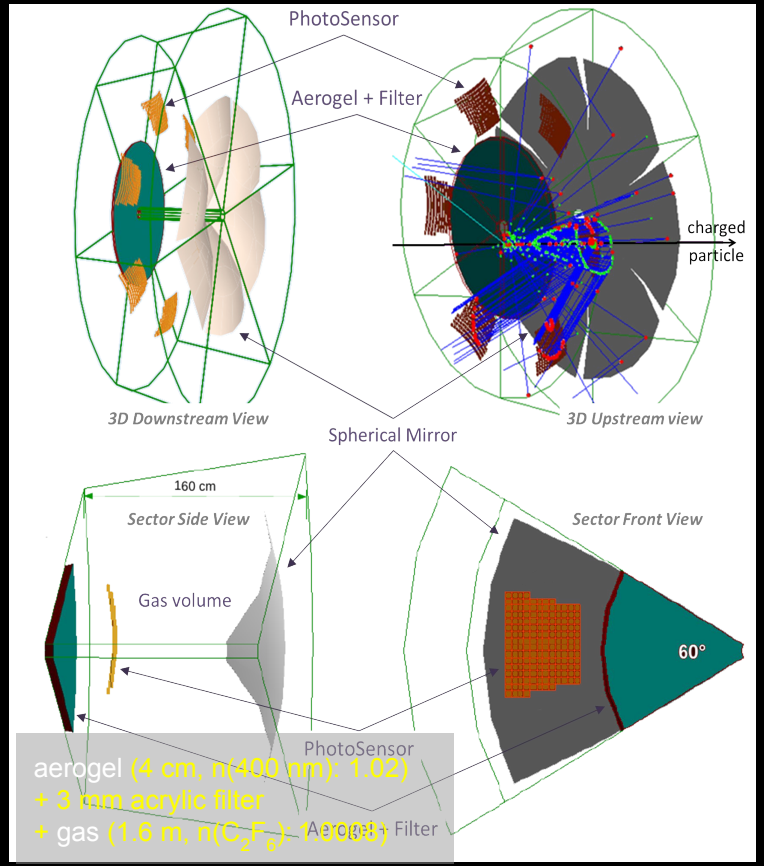
$$N\sigma = \frac{||\langle\theta_K\rangle - \langle\theta_\pi\rangle||\sqrt{N_\gamma}}{\sigma_\theta^{1p.e.}}$$

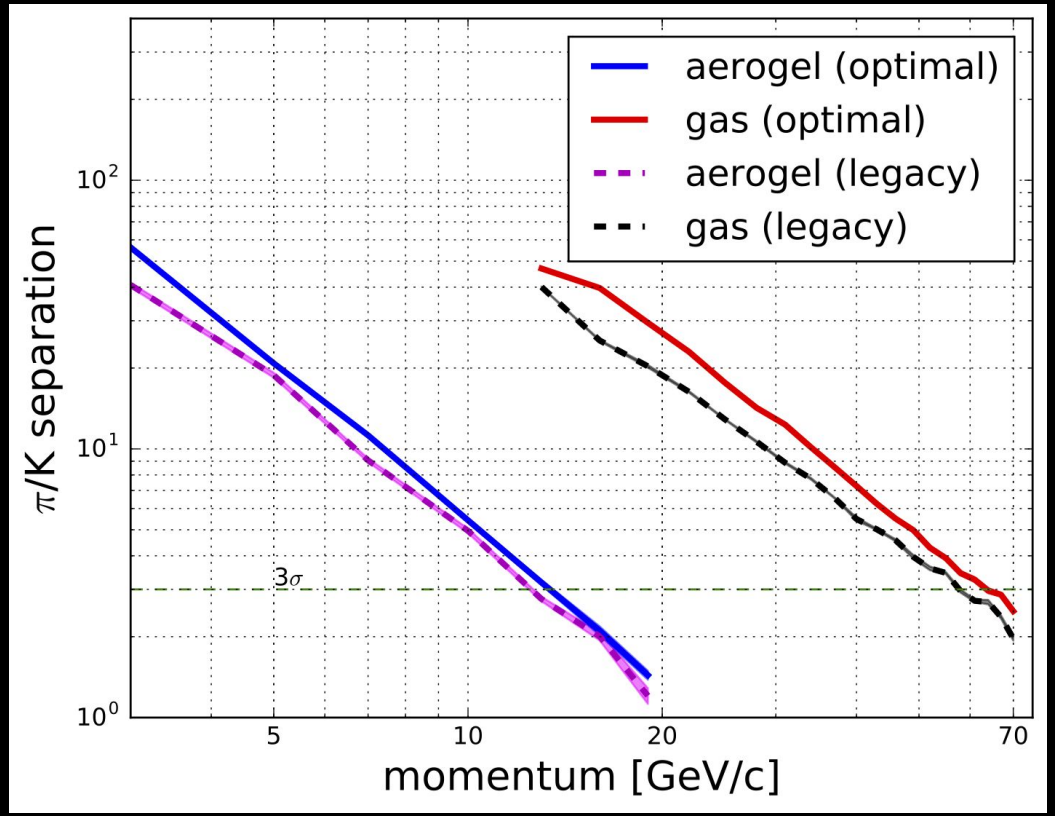




AI-optimized dRICH

E. Cisbani, A. Del Dotto, CF*, M. Williams et al.
JINST 15.05 (2020): P05009.

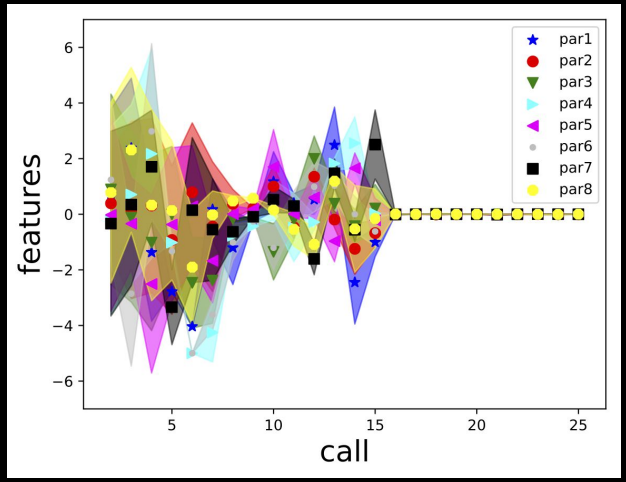


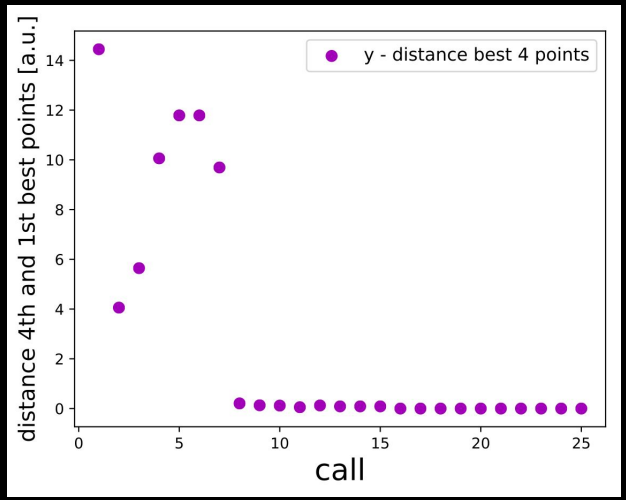


- Statistically significant Improvement in both parts.
- In particular in the gas region where the 5σ threshold shifted from 43 to 50 GeV/c and the 3σ one extended up to
- Notice that before this study we did not know “how well” the legacy design was performing.

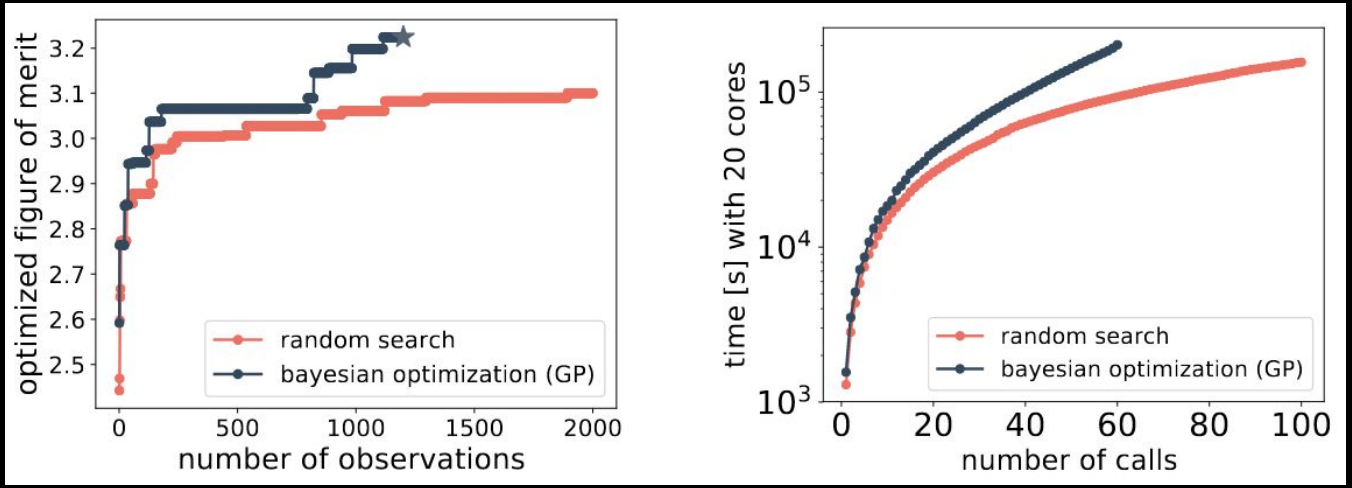
Convergence Criteria

- Can in general be applied in the design space, in the objective space, or looking at the behavior of the acquisition function.
- We defined a set of conditions to ensure convergence:
  - These correspond to the logic AND of booleans on each feature and on the variation of the figure of merit.
  - They are built on standardized Z and Fisher statistics.
- Pre-processing of data required to remove outliers.





Convergence Criteria



Each call:
400 tracks generated/core
20 cores

1 design point ~ 10 mins/CPU

Budget: 100 calls

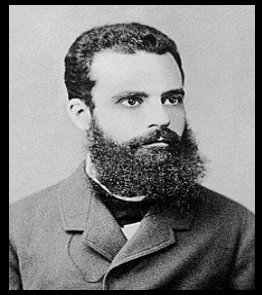
- BO with GP scales cubically with number of observations.
- Bayesian optimization methods are more promising because they offer principled approaches to weighting the importance of each dimension.
- For this 8D problem - even with 50 cores, RS looks unfeasible due to the curse of dimensionality.
  - Recall that the probability of finding the target with RS is $1-(1-v/V)^T$, where T is trials, v/V is the volume of target relative to the unit hypercube

Bergstra, Bengio, “Random search for hyper-parameter optimization”, J. Mach. Learn. Res.13 (Feb) (2012) 281–305.

Multiple Objectives!

[1] Deb, Kalyanmoy. "Multi-objective optimisation using evolutionary algorithms: an introduction." *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, London, 2011. 3-34.

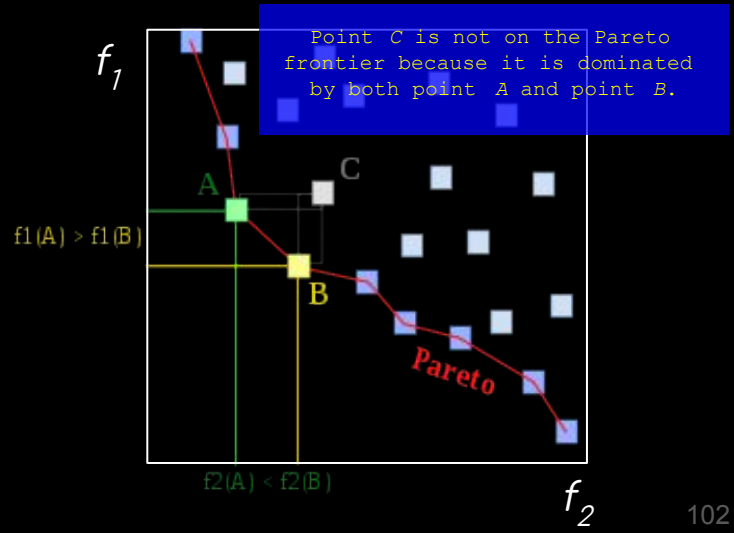
- The problem becomes challenging when the objectives are of conflict to each other, that is, the optimal solution of an objective function is different from that of the other.
- In solving such problems, with or without constraints, they give rise to a trade-off optimal solutions, popularly known as Pareto-optimal solutions.



V. Pareto, 1848-1923

- Due to the multiplicity in solutions, these problems were proposed to be solved suitably using evolutionary algorithms which use a population approach in its search procedure.

MO-based solutions are helping to reveal important hidden knowledge about a problem – a matter which is difficult to achieve otherwise [1].



Frameworks

- Notice that MOO with dynamic/evolutionary algorithms (see, e.g., [1-3]) are probably the most utilized approaches, followed by more recent developments on multi-objective bayesian optimization (see, e.g., [4-7]). Using them has the advantage of having an entire community developing those tools.

  https://github.com/topics/multi-objective-optimization

- Agent-based approaches to MOO are also possible (see, e.g., [8]), but won't be discussed here.
- Remarkably these approaches can accommodate mechanical and geometrical constraints during the optimization process.

[1] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," Advances in Engineering Software, vol. 42, no. 10, pp. 760–771, 2011.

[2] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," The Journal of Machine Learning Research, vol. 13, no. 1, pp. 2171–2175, 2012.

[3] J. Blank and K. Deb, "pymoo: Multi-objective Optimization in Python," IEEE Access, vol. 8, pp. 89497–89509, 2020

[4] M. Laumanns and J. Ocenasek, "Bayesian optimization algorithms for multi-objective optimization," in International Conference on Parallel Problem Solving from Nature, pp. 298–307, Springer, 2002.

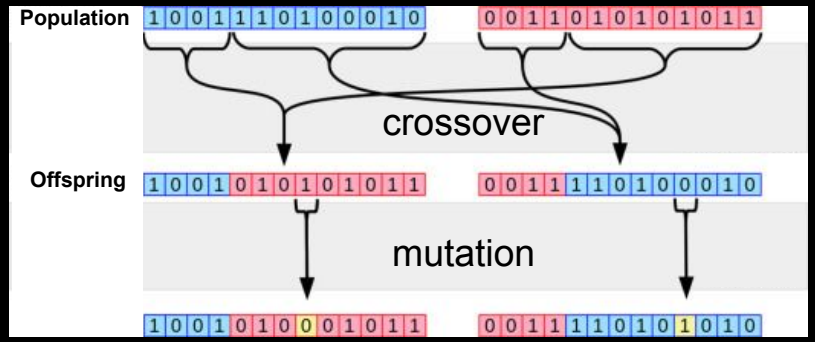
[5] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "Botorch: Programmable bayesian optimization in pytorch," arXiv preprint arXiv:1910.06403, 2019.

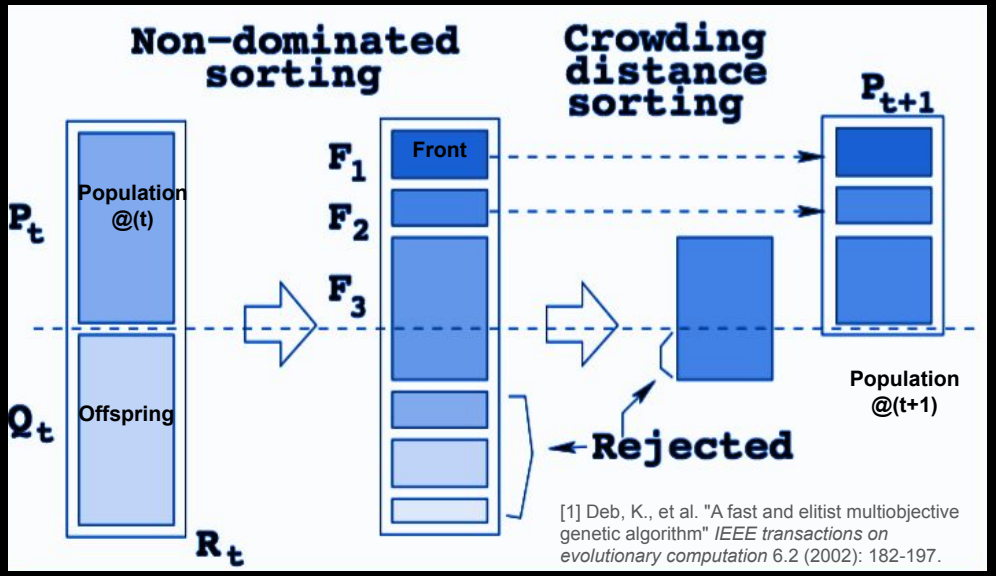
[6] P. P. Galuzio, E. H. de Vasconcelos Segundo, L. dos Santos Coelho, and V. C. Mariani, "MOBOpt—multi-objective Bayesian optimization," SoftwareX, vol. 12, p. 100520, 2020.

[7] A. Mathern, O. S. Steinholtz, A. Sjöberg, M. Önnheim, K. Ek, R. Rempling, E. Gustavsson, and M. Jirstrand, "Multi-objective constrained Bayesian optimization for structural design," Structural and Multidisciplinary Optimization, pp. 1–13, 2020.

[8] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in Advances in Neural Information Processing Systems, pp. 14636–14647, 2019

Elitist non-dominated sorting - NSGA-II



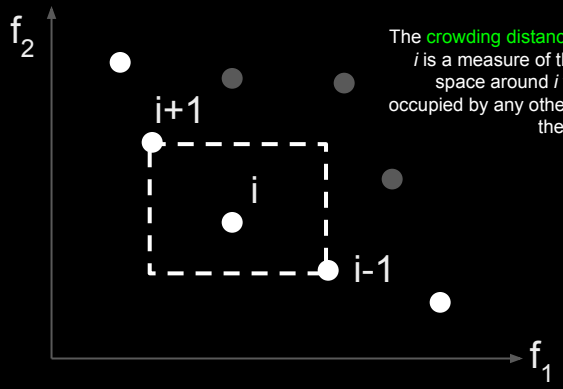


[1] Deb, K., et al. "A fast and elitist multiobjective genetic algorithm" *IEEE transactions on evolutionary computation* 6.2 (2002): 182-197.

This is one of the most popular approach (>35k citations on google scholar), characterized by:

- Use of an elitist principle
- Explicit diversity preserving mechanism
- Emphasis in non-dominated solutions

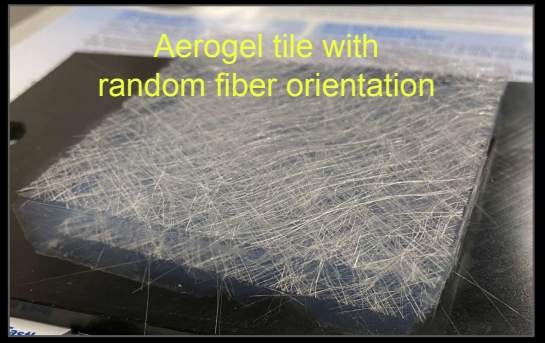
The population R_t is classified in non-dominated fronts. Not all fronts can be accommodated in the N slots of available in the new population P_{t+1}. We use crowding distance to keep those points in the last front that contribute to the highest diversity.



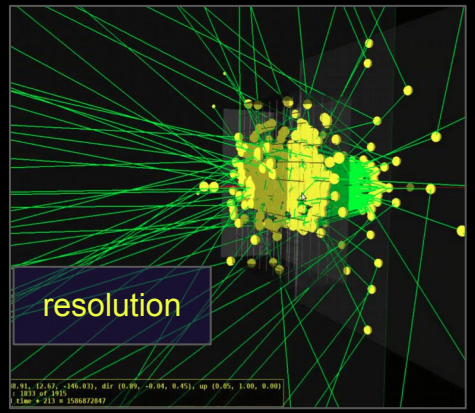
The crowding distance d_i of point i is a measure of the objective space around i which is not occupied by any other solution in the population.

Novel Aerogel Material

The team: V. Berdnikov, J. Crafts, E. Cisbani, CF, T. Horn, R. Trotta



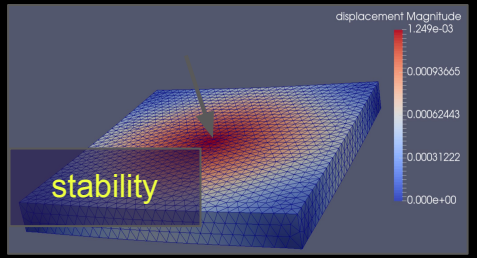
- Aerogels with low refractive indices are very fragile tiles break during production and handling, and their installation in detectors.
- To improve the mechanical strength of aerogels, Scintilex developed a reinforcement strategy. The general concept consists of introducing fibers into the aerogel that increase mechanical strength, but do not affect the optical properties of the aerogel.
- Paper in preparation.



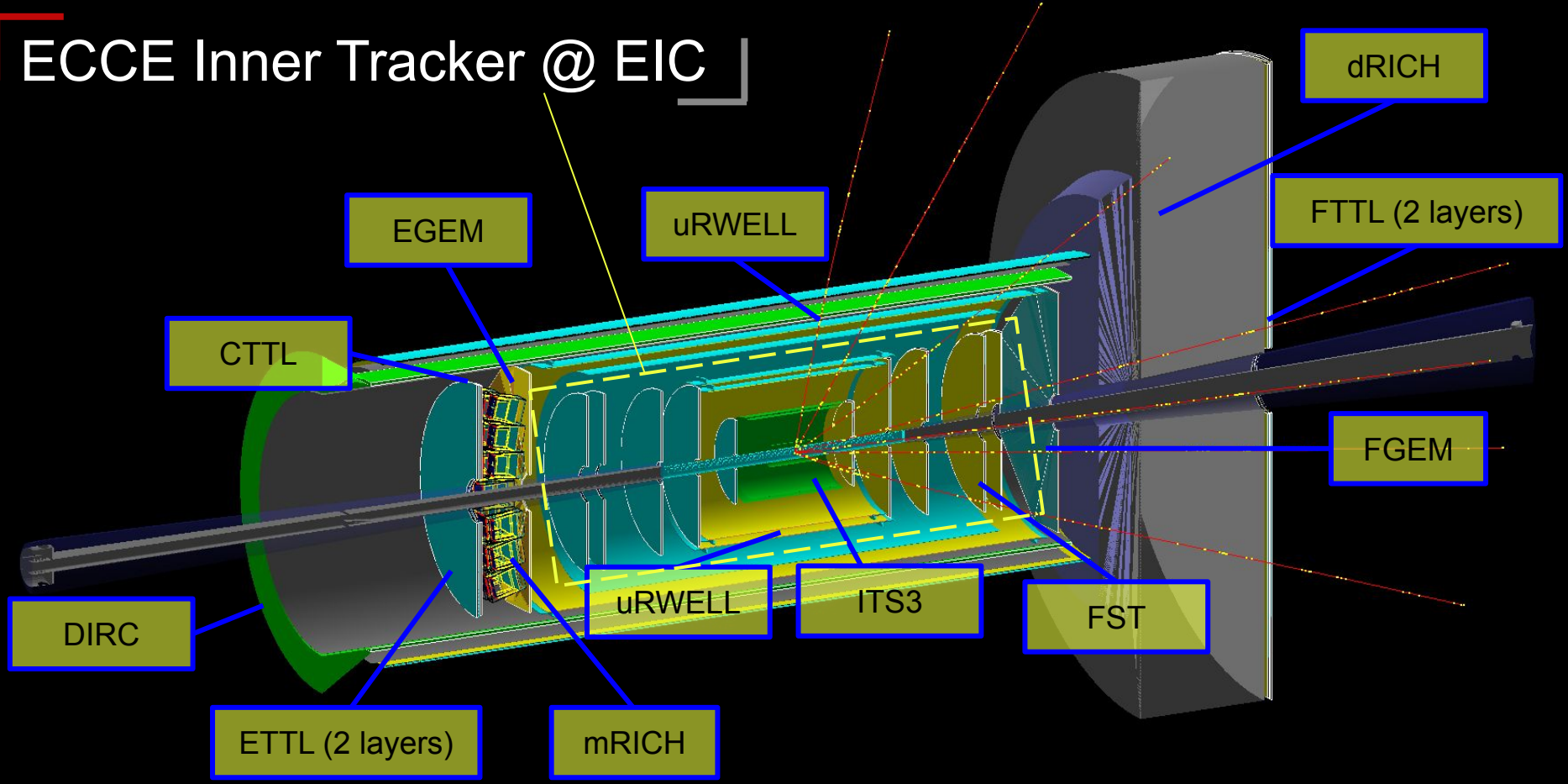
Software Stack

Simple Ring Imaging CHerenkov Geant4 based simulation
Aerogel + Optical Fibers

Gmsh - define geometry and produce mesh
ElmerGrid - convert the gmsh mesh to elmer compatible mesh
ElmerSolver - do modeling (solve linear and nonlinear equation)
Paraview - visualize Elmer Solver and provide a python interface to automate



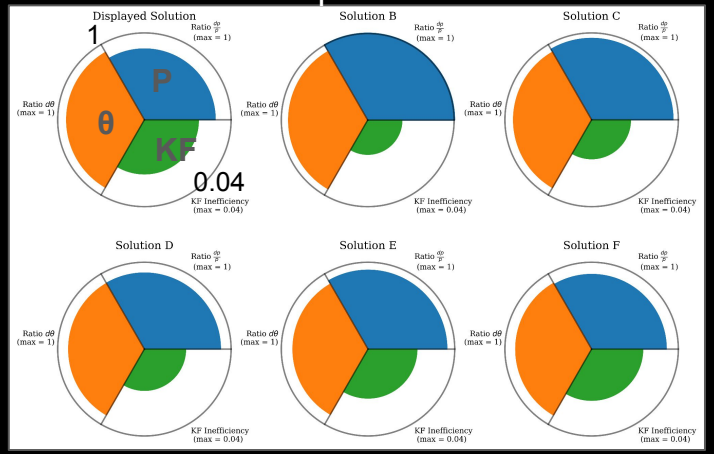
ECCE Inner Tracker @ EIC



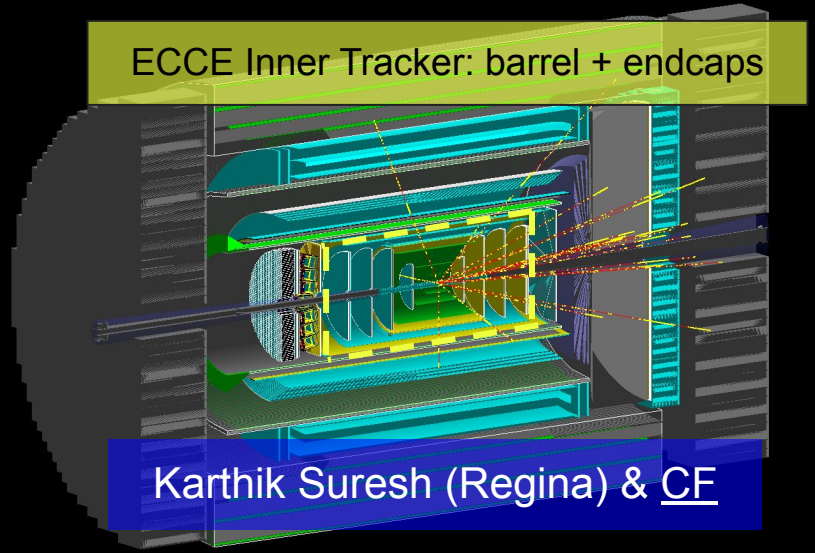
ECCE Inner Tracker @ EIC

- Design include simultaneously:
  - momentum resolution
  - angular resolution
  - Kalman filter efficiency
  - Mechanical constraints

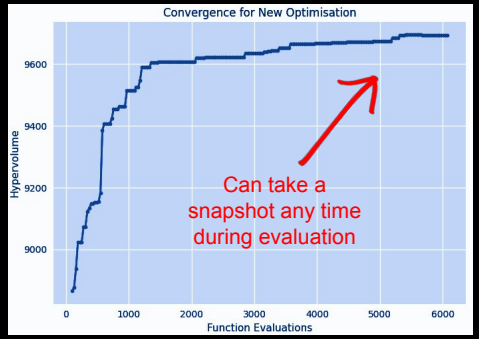
- Pareto front: multiple candidate solutions



Ratios are with respect to a reference design
Each proposed design is consistent with an Aluminum support shell from the reference design

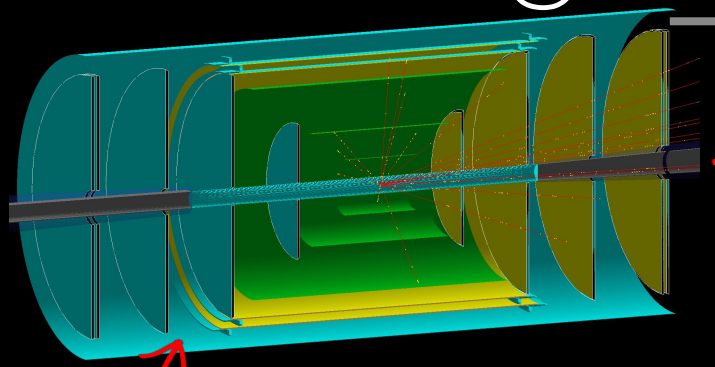


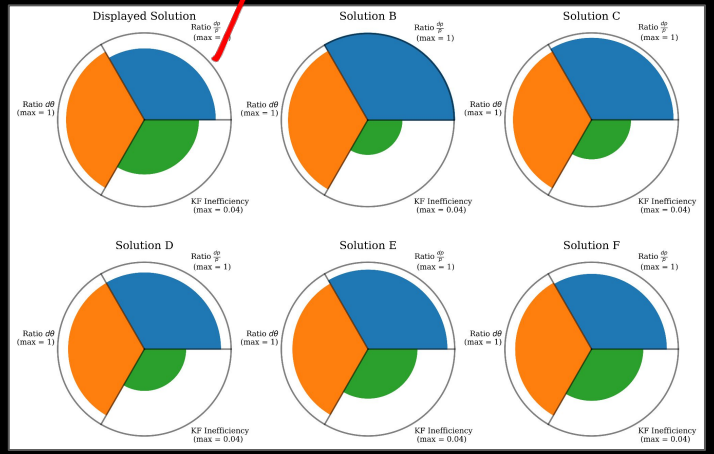
- >= 11 parameters
- 3 (4) objectives
- Population size 100
- Offspring distributed over >=30 cores
- 80000 tracks / design point
- ~1h / design point

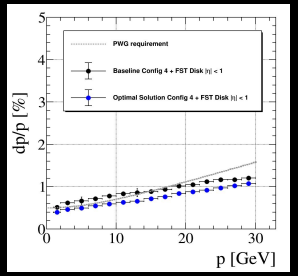


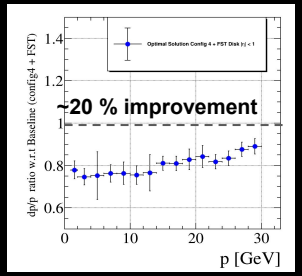
This is (already) an unprecedented attempt in detector design for complexity!

ECCE Inner Tracker @ EIC

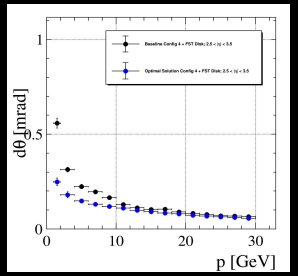


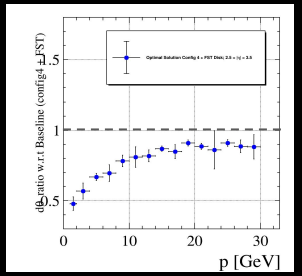






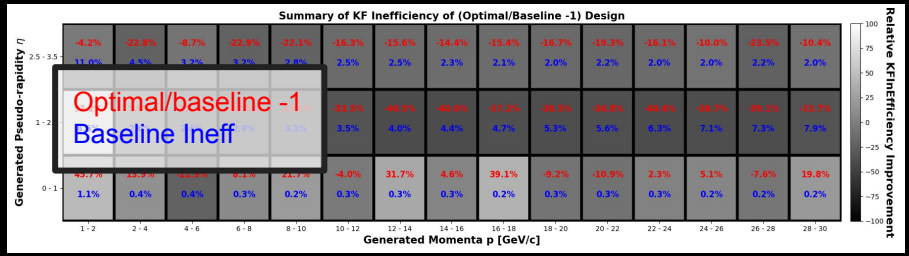
P Reso





θ Reso

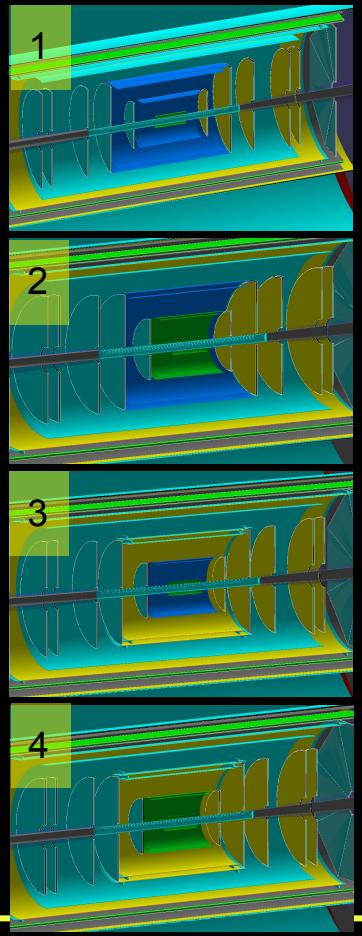
KF Ineff

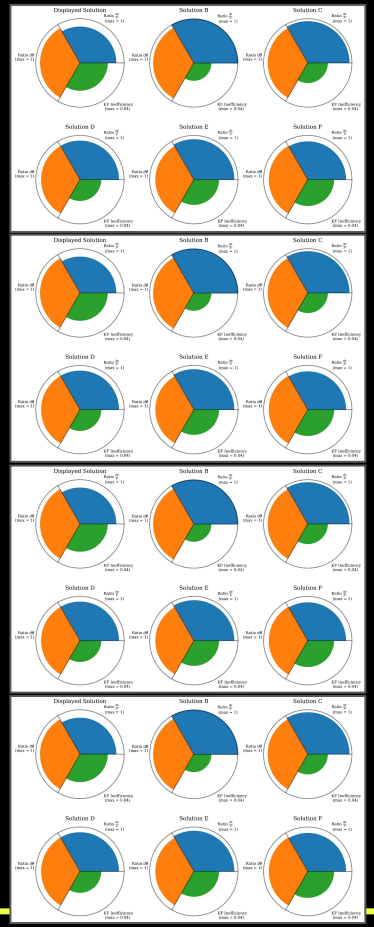


See talk by W. Phelps for more details

The decision making process done after optimization.
For each design solution in the Pareto Front one can study the corresponding detector performance.

Multiple Pipelines





Inner Tracker Barrel (+ disks in the h-endcap and e-endcap)

- Configuration 1: 2-vtx (ITS3) + 2-sagitta (ITS2) + 2-outer layer (ITS2)
- Configuration 2: 2-vtx (ITS3) + 2-sagitta (ITS3) + 2-outer layer (ITS2)
- Configuration 3: 2-vtx (ITS3) + 2-sagitta (ITS2) + 2-outer layer (uRwell)
- Configuration 4: 2-vtx (ITS3) + 2-sagitta (ITS3) + 2-outer layer (uRwell)

DWG's:
- Technology Selection
- Baseline Design
- Alternate Configuration(s)

PWG's:
- Physics Signal Selection
- Physics Performance Evaluation

CWG's:
- Simulation Campaign Support
- AI Optimization

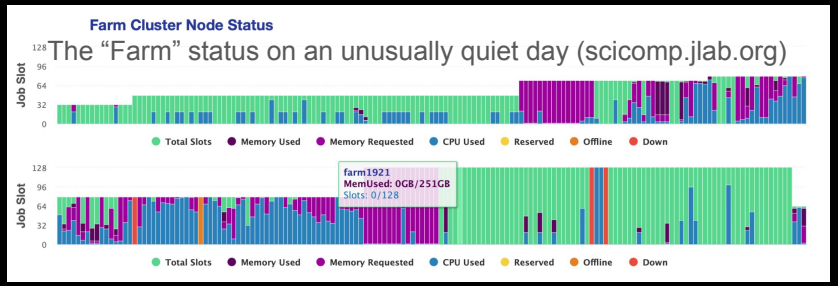
updated configurations with any additional requirements

New optimization pipelines

Resources

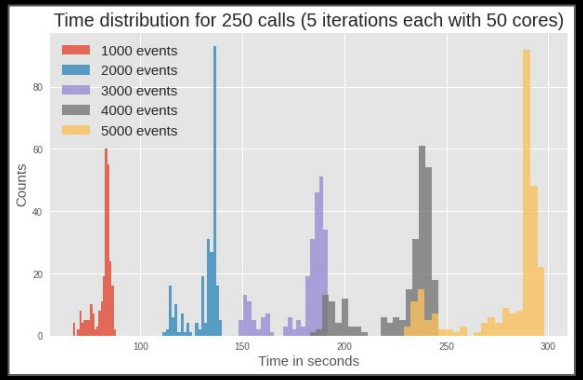
OUR PROBLEM: Inner Tracker
>= 11 parameters
3 objectives
Population size 100
Offspring distributed over >=30 cores

- running on scicomp @ JLab



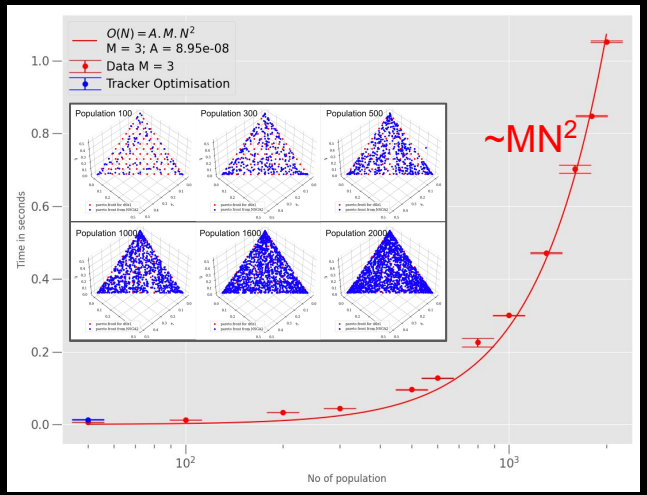
The scientific computing cluster has:
- 25k cores EIC Projects are allocated 10%
- 1PB for EIC use
- Batch use as well as interactive use supported with
  - Nodes with up to two 32 core AMD Epyc Processors (128 threads), 256GB Ram, 1TB SSD local storage
  - 3 Nodes with 4 Titan RTX Cards (24 GB Memory)
  - GPU nodes also available through jupyterhub.jlab.org

- Characterization of simulation times



Simulating 80000 in total for each evaluation, 1 evaluation is <=80 mins

- Characterization of time taken by GA + sorting



- Used a test problem DTLZ1
- Verified scaling following MN^2 and convergence to true front
- ~1s/call with 10^4 size!
- For 11 variables and 3 objectives needs ~ 10000 evaluations to converge

~10k CPUhours

MOEA Parallelization

[1] M.T. Jensen, IEEE Trans. Evol. Comput., 2003

- Well known that NSGA-II increase in computational complexity as **O(MN^2)** [1].
- A recent trend in MOEA is distributed NSGA-II and implementation on supercomputers. This is useful when large populations are needed (*e.g.*, 10^5), due to complexity and/or to approximate the Pareto front with high accuracy.
- A custom optimized parallel NSGA-II called swNSGA-II has been designed for Sunway TaihuLight [2] supercomputer.

[2] Liu, Xin, et al. IEEE Trans Parallel Distrib Syst 32.4 (2020): 975-987.

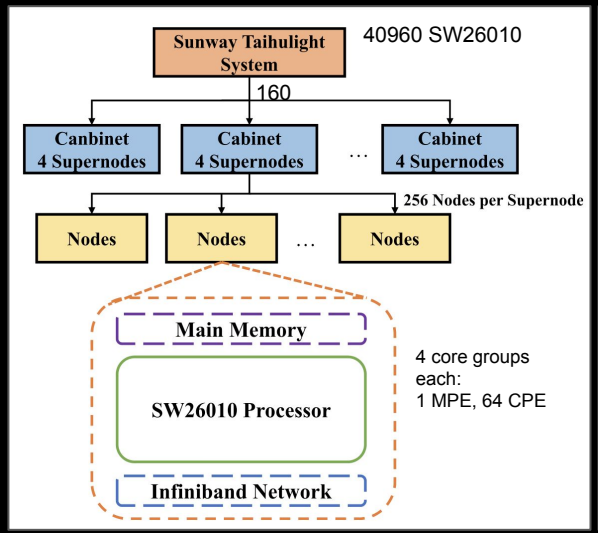


TABLE 3
The Running Time of swNSGA-II on Multiple Core Group(s)

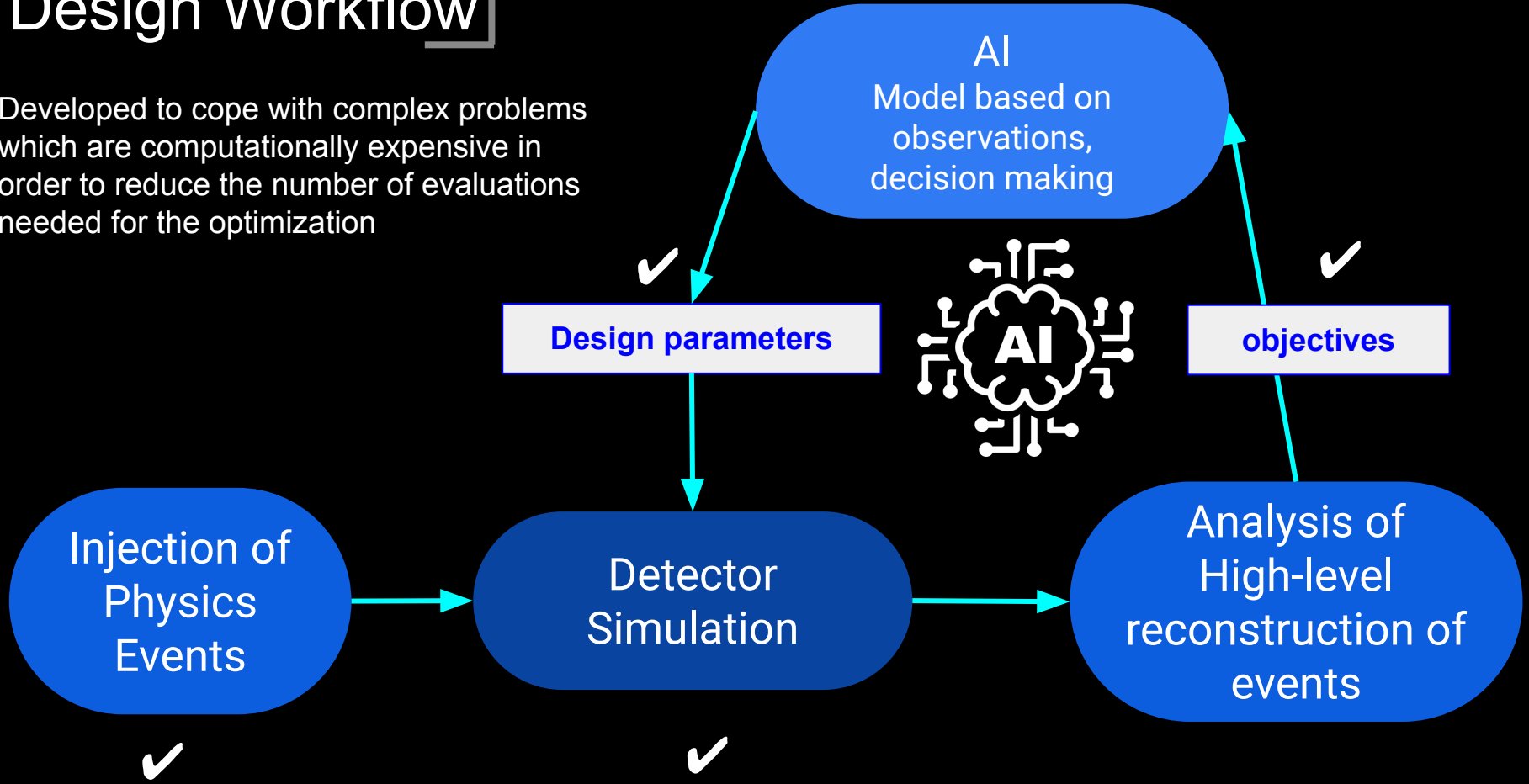
	CG(s)	Time in second(s)	Speedup
Path Planning			
NSGA-II	1*	4954.15	N/A
swNSGA-II	1	222.31	22.28
	2	51.06	24.19
	4	12.67	391.01
	8	3.53	1403.44
	16	1.15	4307.96
	32	0.34	14571.03
	64	0.19	26074.47
	100	0.12	41284.58
ZDT1			
NSGA-II	1*	3134.64	N/A
swNSGA-II	1	255.46	12.27
	2	54.77	57.23
	4	12.09	259.35
	8	2.78	1128.17
	16	0.70	4446.49
	32	0.24	13073.62
	64	0.07	45043.89
	100	0.05	62692.80

*MPE only.

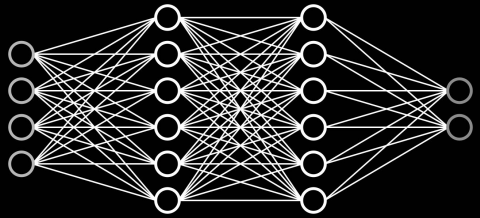
- swNSGA-II utilize process and thread level parallelism based on an improved island master-puppet model.
- Performance have been benchmarked against conventional NSGA-II with a speedup of ~$5 \cdot 10^4$ for standard optimization problems.
- Comparisons with GPU (GeForce GT 630) -based NSGA-II done using 1 core group only (64 CPE), obtaining a speedup of ~10 with large populations.

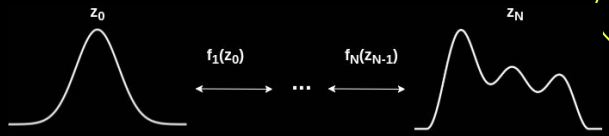
Design Workflow

Developed to cope with complex problems which are computationally expensive in order to reduce the number of evaluations needed for the optimization

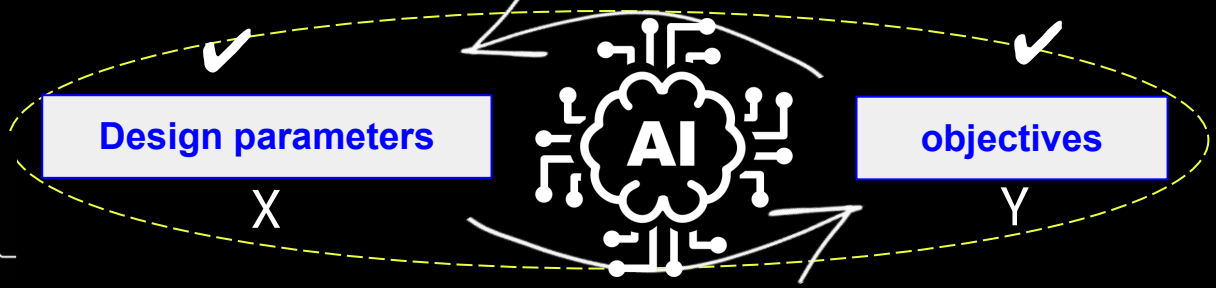


With large dataset





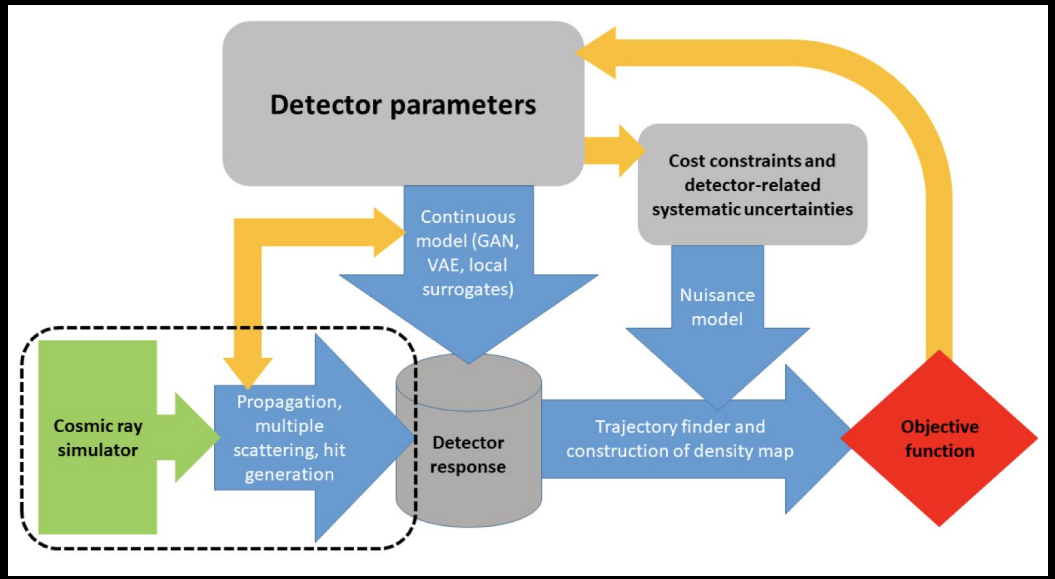
AI
Model based on observations, decision making



Injection of Physics Events → Detector Simulation → Analysis of High-level reconstruction of events

MODE

- Detectors design with AI is gaining a lot of interest.
- MODE is a recently formed collaboration of physicists and computer scientists who target the use of differentiable programming in design optimization of detectors for particle physics applications A. G. Baydin et al. Nuclear Physics News 31.1 (Mar 30, 2021): 25-28.
- Ambitious project: develop a modular, customizable, and scalable, fully differentiable pipeline for the end-to-end optimization of articulated objective functions that model in full the true goals of experimental particle physics endeavours, to ensure optimal detector performance, analysis potential, and cost-effectiveness.



Conceptual layout of an optimization pipeline for a muon radiography apparatus.

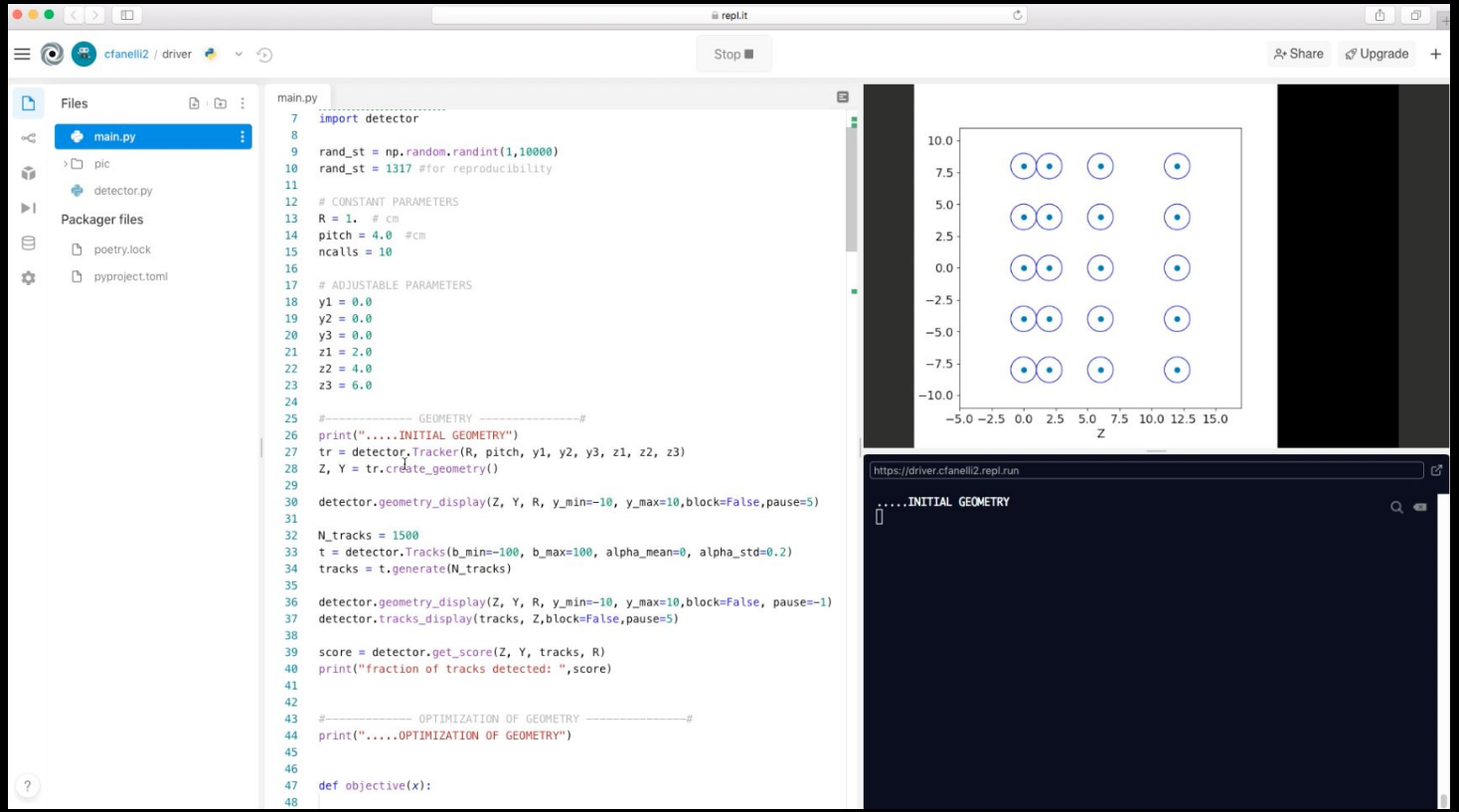
An end to end optimization requires modeling of simulations. Requires collect reference data to train the surrogate models ML implementations.

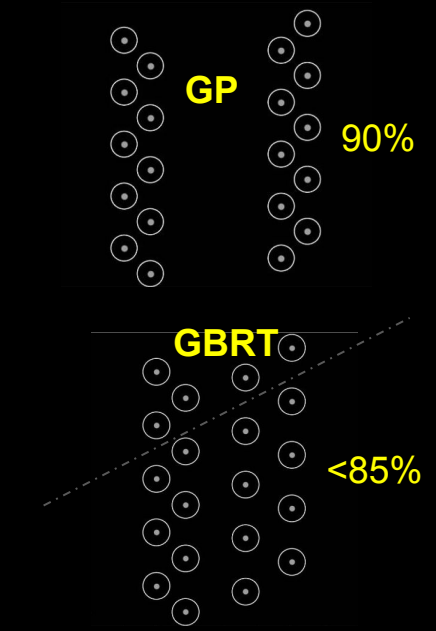
Summary on Design

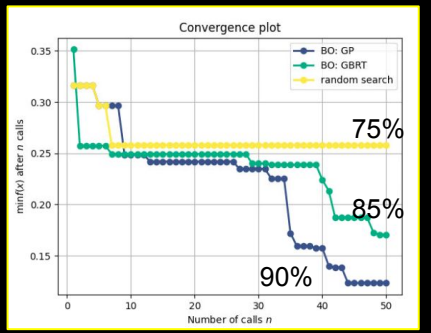
- EIC can be one of the first experiment to be designed with the support of AI.
- ECCE is leading these efforts with an unprecedented attempt in detector design (multidimensional design and objective spaces).
- None ever accomplished a multi-dimensional / multi-objective optimization of the global design, i.e., made by many sub-detectors combined together, that can be solved with AI
  - Costs can be explicitly included during the optimization provided a reliable parametrization)
  - An intrinsic overhead regards compute expensive simulations (+ reconstruction/analysis). How to speed up bottlenecks and overall these steps? See discussion in the Sessions on: Simulations, Reco & Analysis.
  - Larger populations of design points can be simulated to improve accuracy of the Pareto front in multidimensional spaces with AI-based accelerated optimizations.

Likely future detectors will be designed with the help of AI achieving optimal performance and cost reduction.
One of the conclusions from the DOE Town Halls on AI for Science on 2019 was that "*AI techniques that can optimize the design of complex, large-scale experiments have the potential to revolutionize the way experimental nuclear physics is currently done*".

Toy Model

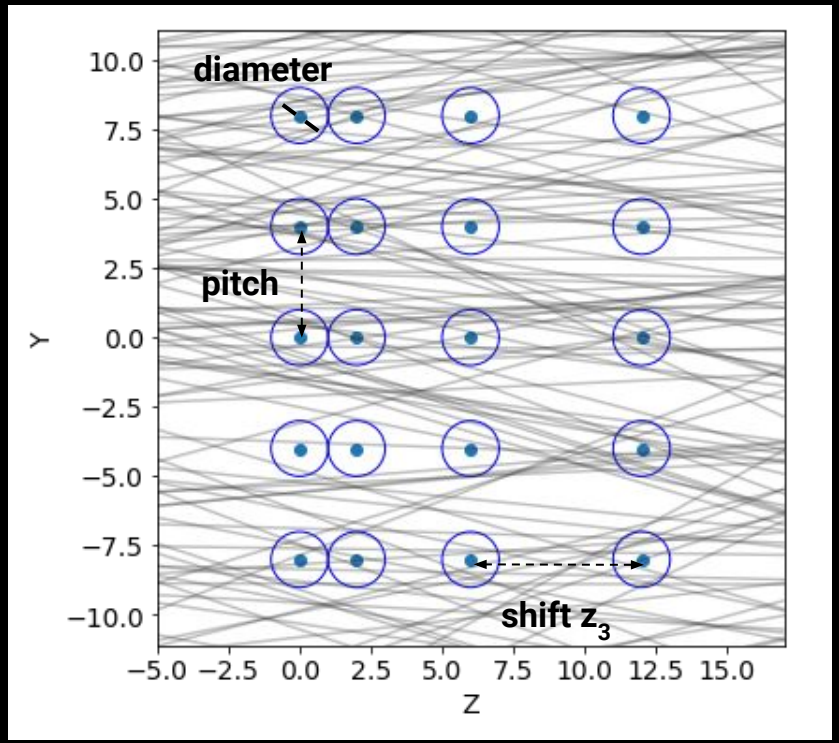

repl.it
cfanelli2 / driver
Stop
Share
Upgrade
Files
main.py
pic
detector.py
Packager files
poetry.lock
pyproject.toml
main.py
import detector
rand_st = np.random.randint(1,10000)
rand_st = 1317 #for reproducibility
# CONSTANT PARAMETERS
R = 1. # cm
pitch = 4.0 #cm
ncalls = 10
# ADJUSTABLE PARAMETERS
y1 = 0.0
y2 = 0.0
y3 = 0.0
z1 = 2.0
z2 = 4.0
z3 = 6.0
#-------------- GEOMETRY ----------------#
print(".....INITIAL GEOMETRY")
tr = detector.Tracker(R, pitch, y1, y2, y3, z1, z2, z3)
Z, Y = tr.create_geometry()
detector.geometry_display(Z, Y, R, y_min=-10, y_max=10,block=False,pause=5)
N_tracks = 1500
t = detector.Tracks(b_min=-100, b_max=100, alpha_mean=0, alpha_std=0.2)
tracks = t.generate(N_tracks)
detector.geometry_display(Z, Y, R, y_min=-10, y_max=10,block=False, pause=-1)
detector.tracks_display(tracks, Z,block=False,pause=5)
score = detector.get_score(Z, Y, tracks, R)
print("fraction of tracks detected: ",score)
#-------------- OPTIMIZATION OF GEOMETRY ----------------#
print(".....OPTIMIZATION OF GEOMETRY")
def objective(x):
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−5.0 −2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
Z
https://driver.cfanelli2.repl.run
.....INITIAL GEOMETRY


GP
90%
GBRT
<85%

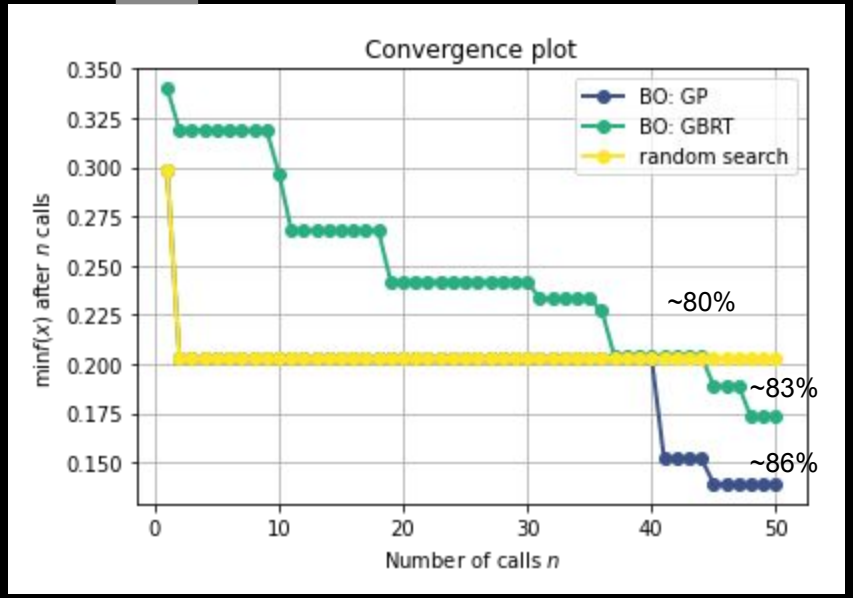

Convergence plot
BO: GP
BO: GBRT
random search
75%
85%
90%
min(x) after n calls
Number of calls n
0.35
0.30
0.25
0.20
0.15
0
10
20
30
40
50

Toy Model

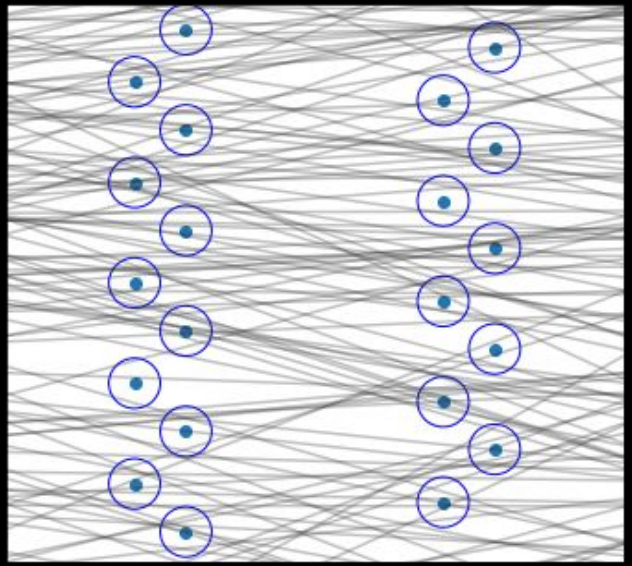
- A simple toy model is used in the hands-on session with the goal to introduce some optimization technique and useful frameworks in python.
- The toy detector consists in a 2D tracking system with 4 layers of wires.
- A total of 8 parameters can be tuned. The adjustable parameters are the radius of each wire, the pitch (along the y axis), and the shift along y and z of a plane with respect to the previous one. We will start with tuning the spatial shifts first (6 parameters).
- Straight tracks are generated at different angles and random origin. The tracker geometry and the track generation is already defined in the imported module *detector.py*.

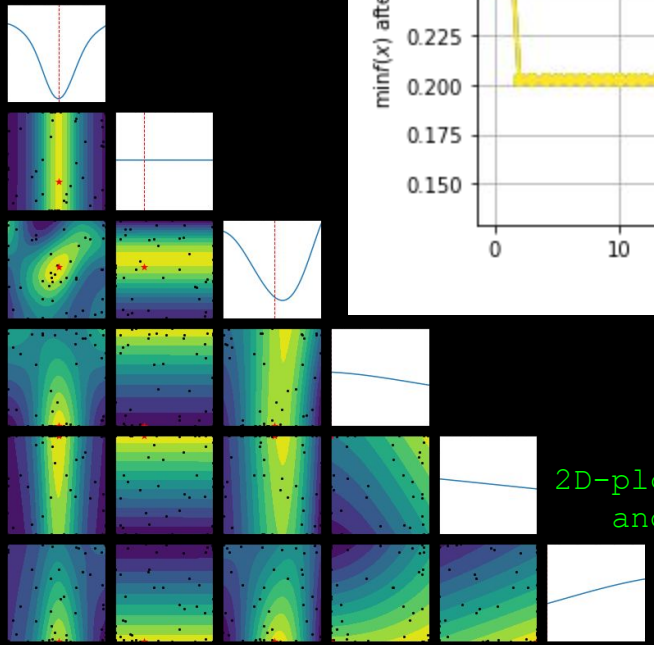


Toy Model BO



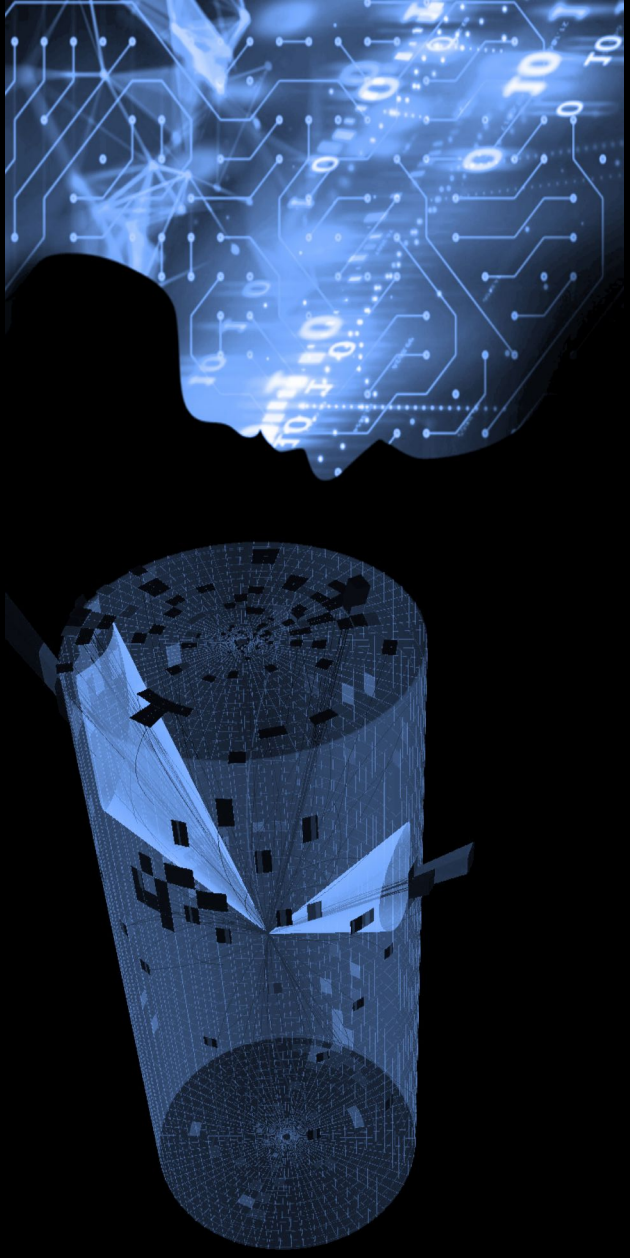
"Optimal" configuration





2D-plots of objective function and partial dependencies

Objective: Efficiency is defined as at least two wires are hit



Thank you