Overview of Gaussian Processes and Bayesian Optimization for Plasma-Based Accelerators

Remi Lehe

Lawrence Berkeley National Laboratory, Berkeley, California

May 18th, 2022

Artificial intelligence / machine learning is a vast, diverse field

A set of **mathematical techniques**:

- Neural networks
- Tree-based methods
- Gaussian Processes
- Support Vector Machines
- Principal Component Analysis (PCA)
- k-Nearest Neighbors
- Genetic algorithms
- …

that solve certain **tasks based on data**:

- Classification
- Regression
- Natural language processing
- Dimensionality reduction
- Recommendation
- Optimization
- …

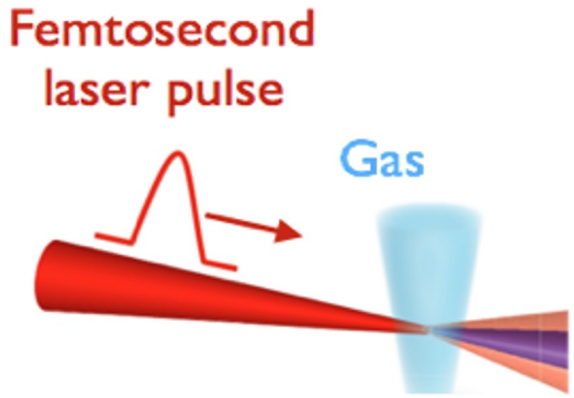
Outline

- Optimization problems in laser-plasma acceleration
- Gaussian Process models
- Bayesian optimization
- Applications in laser-plasma acceleration & current research
- Uncertainty in machine learning

Optimization task for plasma-based accelerators

Simultaneously adjust many parameters

- Gas density
- Concentration of various elements
- Laser energy
- Laser focal position
- Laser spectral properties
- Laser waist
- …



in order to **maximize one (or several) objective function**:

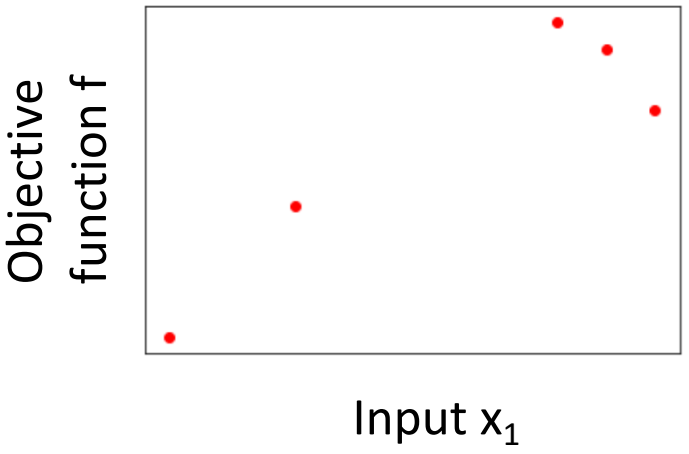
e.g.

- Electron emittance
- Electron energy
- Electron energy spread
- Electron charge
- Combinations thereof
- …

relevant for both **simulations** (design optimization) and **experiments** (real-time tuning)

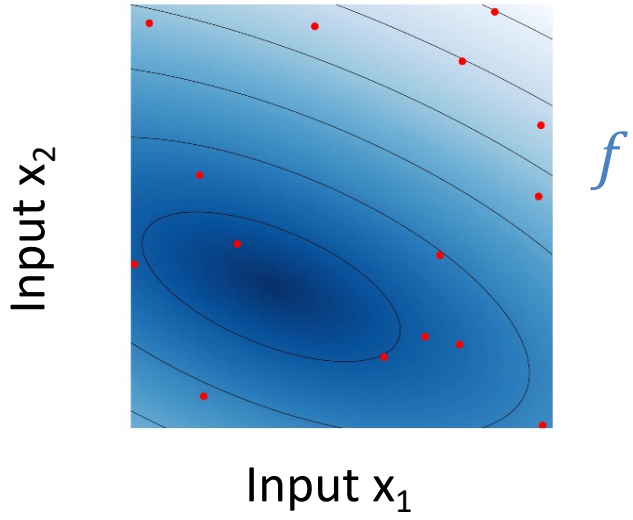
Optimization is usually done in high-dimension

1D:



Example:
x_1 = laser energy
f = beam energy spread

2D:



Example:
x_1 = laser energy
x_2 = gas density
f = beam energy spread

Multi-dimension:

x is a high-dimensional vector that contains the tunable parameters

Example:

$$\boldsymbol{x} = \begin{pmatrix} \text{Laser energy} \\ \text{Gas density} \\ \text{Laser chirp} \\ \dots \end{pmatrix}$$

Aim:

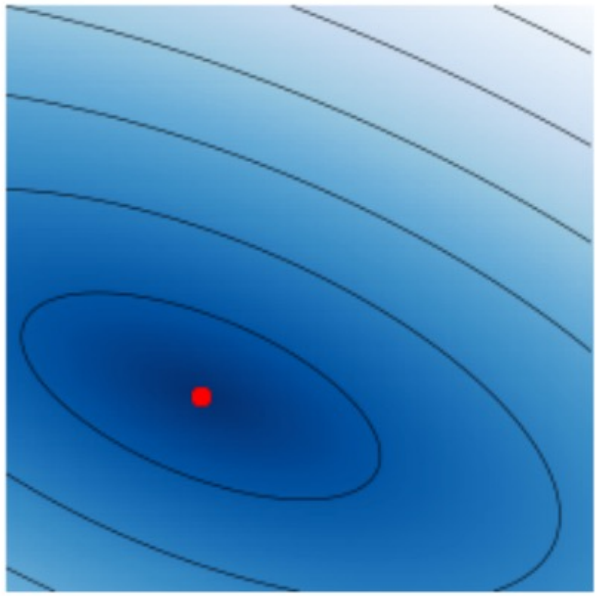
Find $\boldsymbol{x}_{max}$ such that $f(\boldsymbol{x}_{max})$ is **maximal**

High-dimensional optimization is expensive

Aim:

Find $\boldsymbol{x}_{max}$ such that $f(\boldsymbol{x}_{max})$ is **maximal**, with **few** evaluations of f

f



Motivation: evaluations of f are usually costly

- **Design studies**:
  Evaluations of f require **computationally expensive** numerical simulations

- **Online tuning:**
  Evaluations of f **take time** on the experiments
  Parameters of the machine may **drift** if it takes too long to find the minimum.

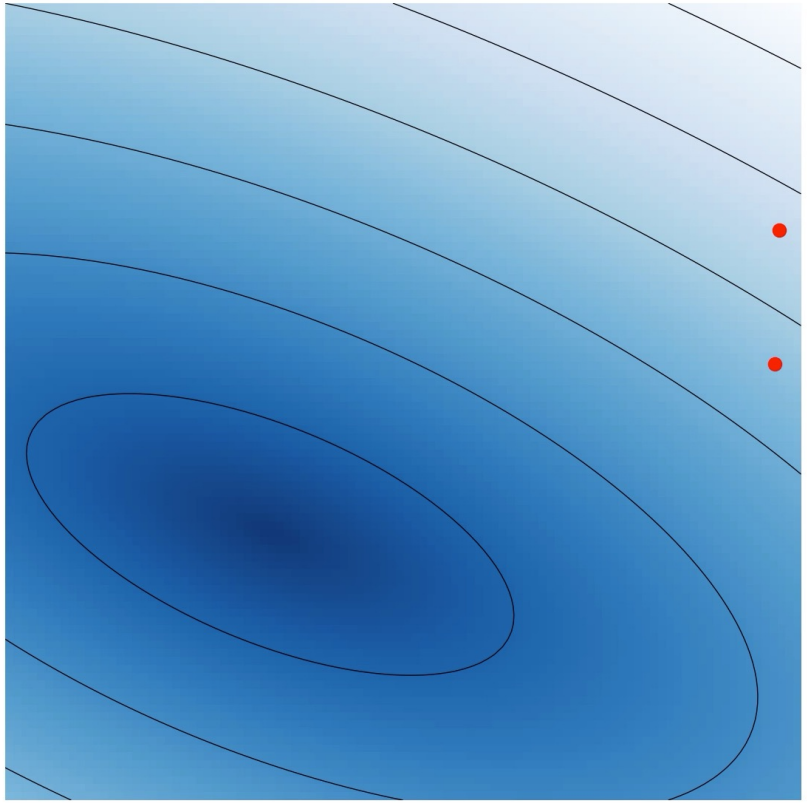
An example of naïve algorithm: random search

Algorithm:

Evaluate f at **randomly chosen points.**
At the end: find the best point among them.

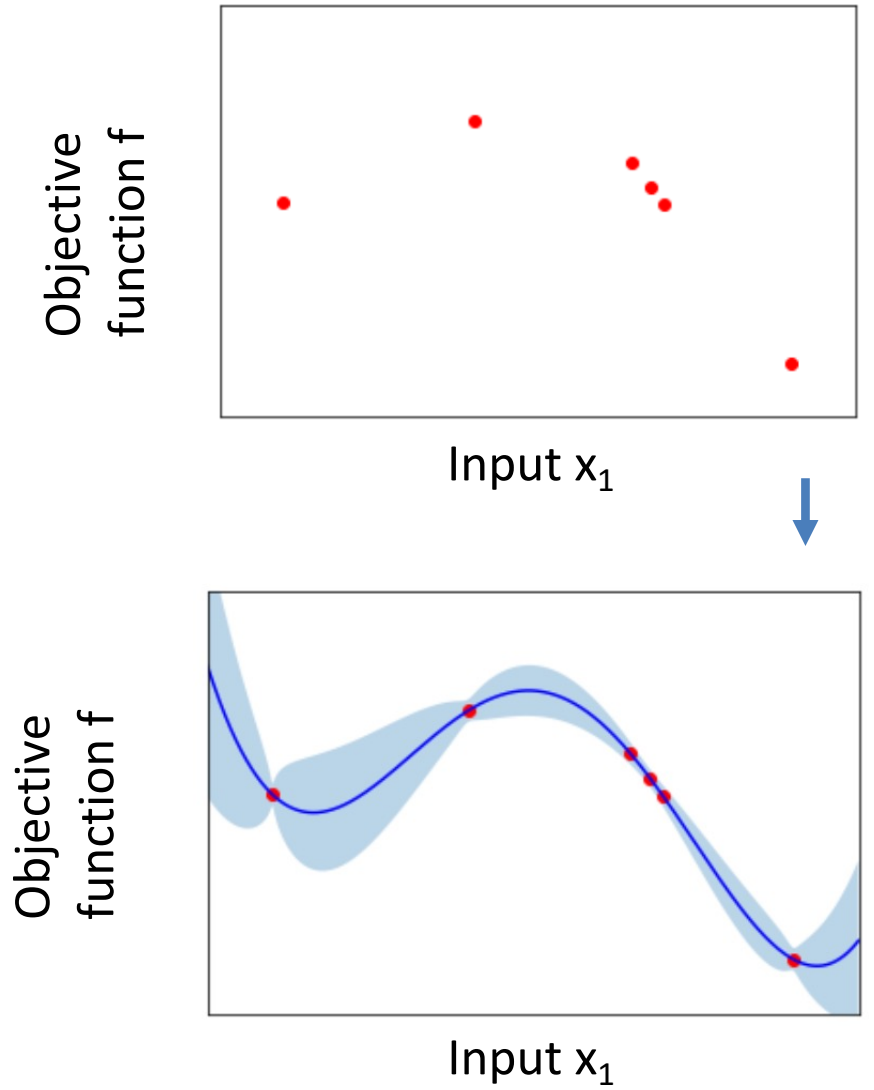
Practical consideration:

- Takes a long time to even reach interesting regions.
- May evaluate points that **are close to each other** and do not bring significantly more information
- Does not use the information from **previous evaluations of f** to decide which point to evaluate next.



Looking for a more efficient algorithm: model-based optimization

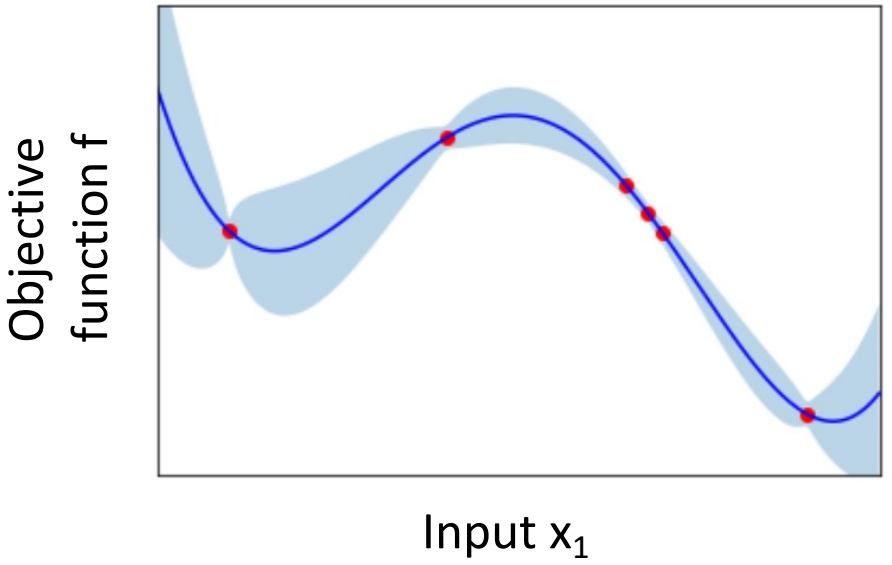
- One way to be more efficient would be to build a **guess** (or a **model**) of the function *f* at unexplored location.
- By nature, this model would need to be **probabilistic** and quantify the **uncertainty** about the values of *f.*
- This model could then be used to automatically select points that are **worth evaluating**.



Defining an appropriate model

Desirable properties:

- The model should interpolate previous data
- The uncertainty should grow away from previous data
- The scale of variation of the model should match the scale of variation of previous data.
- The model should naturally generalize to high dimension



Outline

- Optimization problems in laser-plasma acceleration
- **Gaussian Process models**
- Bayesian optimization
- Applications in laser-plasma acceleration & current research
- Uncertainty in machine learning

Gaussian process

The user defines a **kernel function** $k(\boldsymbol{x}, \boldsymbol{x}')$, which reflects assumptions on how different points are correlated.

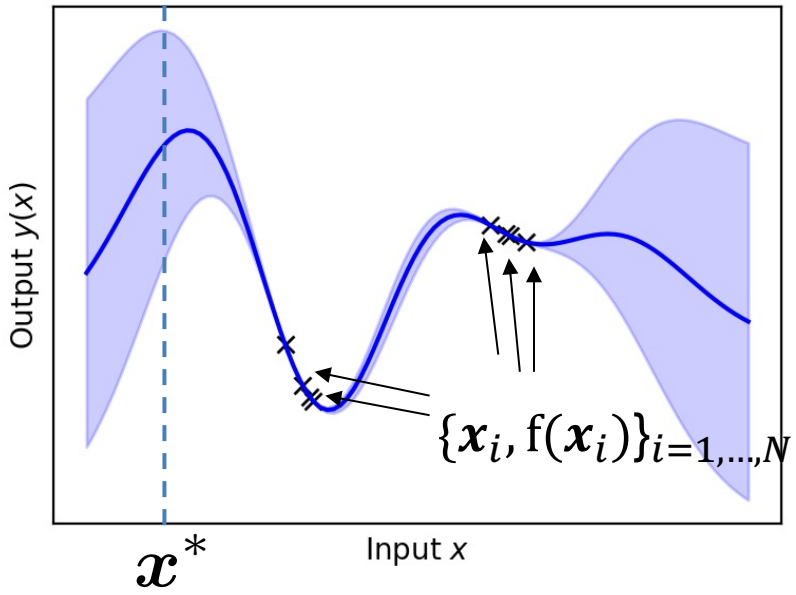
For instance:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x}')^2}{\ell^2}\right)$$

Given N **previous evaluations** $\{\boldsymbol{x}_i, \mathrm{f}(\boldsymbol{x}_i)\}_{i=1,\ldots,N}$, the **probability distribution** of $\mathrm{y}(\boldsymbol{x}^*)$ at a **new input** $\boldsymbol{x}^*$ is predicted to be Gaussian: $y(\boldsymbol{x}^*) \sim \mathcal{N}\left(m(\boldsymbol{x}^*), \sigma^2(\boldsymbol{x}^*)\right)$

$$m(\boldsymbol{x}^*) = \boldsymbol{k}^{*T} K^{-1} \boldsymbol{y}$$

$$\sigma^2(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{*T} K^{-1} \boldsymbol{k}^*$$



K: matrix of size $N \times N$, defined by $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

$\boldsymbol{y}$: vector of size N, containing previous evaluations: $\mathrm{y_i} = \mathrm{f}(\boldsymbol{x}_i)$

$\boldsymbol{k}^*$: vector of size N, defined by $k_i^* = k(\boldsymbol{x}_i, \boldsymbol{x}^*)$

(Rasmussen & Williams, "Gaussian Process for Machine Learning")

Gaussian process: hyperparameters

$$m(\boldsymbol{x}^*) = \boldsymbol{k}^{*T} K^{-1} \boldsymbol{y}$$

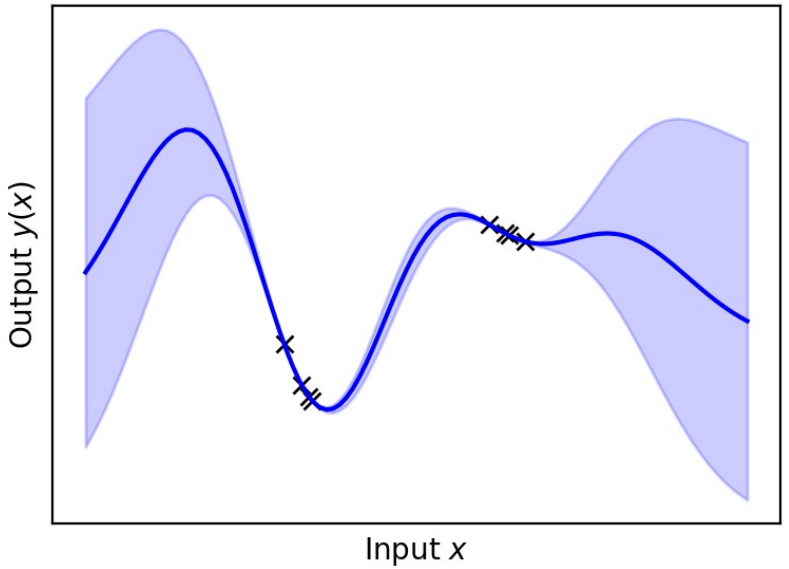
$$\sigma^2(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{*T} K^{-1} \boldsymbol{k}^*$$

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sigma_f^2 \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x'})^2}{\ell^2}\right)$$

K: matrix of size $N \times N$, defined by $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

$\boldsymbol{y}$: vector of size N, containing previous evaluations: $y_i = f(\boldsymbol{x}_i)$

$\boldsymbol{k}^*$: vector of size N, defined by $k_i^* = k(\boldsymbol{x}_i, \boldsymbol{x}^*)$



σ_f^2 and ℓ (the "hyperparameters") are **automatically tuned** in order to **match** the **amplitude** and **length scale** of the typical variations in the data.
(e.g. using maximum likelihood)

Gaussian process with estimated noise

$$m(\boldsymbol{x}^*) = \boldsymbol{k}^{*T}(K + {\sigma_\eta}^2)^{-1}\boldsymbol{y}$$

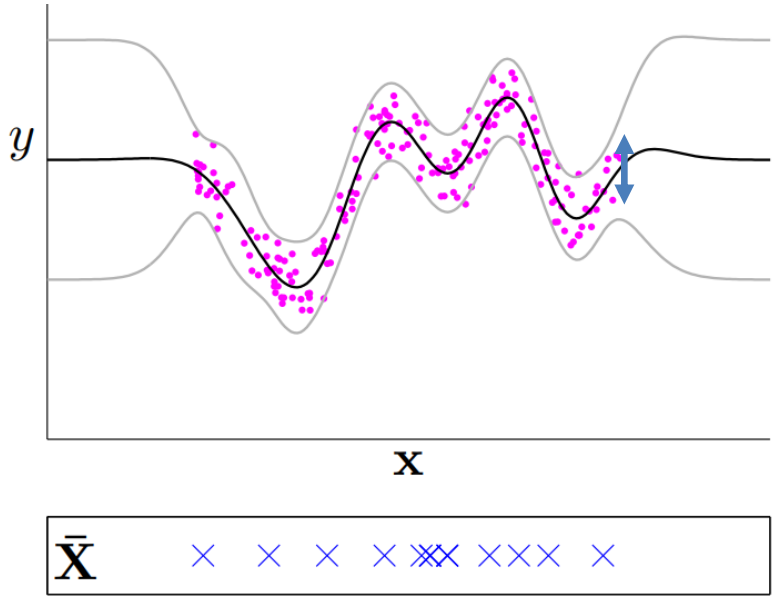
$$\sigma^2(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{*T}(K + {\sigma_\eta}^2)^{-1}\boldsymbol{k}^* + {\sigma_\eta}^2$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x}')^2}{\ell^2}\right)$$

K: matrix of size $N \times N$, defined by $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

$\boldsymbol{y}$: vector of size N, containing previous evaluations: $\mathrm{y_i} = \mathrm{f}(\boldsymbol{x}_i)$

$\boldsymbol{k}^*$: vector of size N, defined by $k_i^* = k(\boldsymbol{x}_i, \boldsymbol{x}^*)$



σ_η is also a hyperparameter that is **automatically tuned** in order to **match** the data.
(e.g. using maximum likelihood)

(Rasmussen & Williams, "Gaussian Process for Machine Learning")

Gaussian process: practical considerations

Computational cost:
Training: scales as N^3,
Predicting: scales as N^2
(where N is number of data points)
Only worth it when the function *f* to optimize is costly.

$$m(\boldsymbol{x}^*) = \boldsymbol{k}^{*T} K^{-1} \boldsymbol{y}$$

$$\sigma^2(\boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^{*T} K^{-1} \boldsymbol{k}^*$$

Choice of kernel:
Encodes preexisting knowledge (or lack thereof)
about the way in which the function varies
e.g. anisotropy, correlation between dimensions,
smoothness, etc.

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sigma_f^2 \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x'})^2}{\ell^2}\right)$$

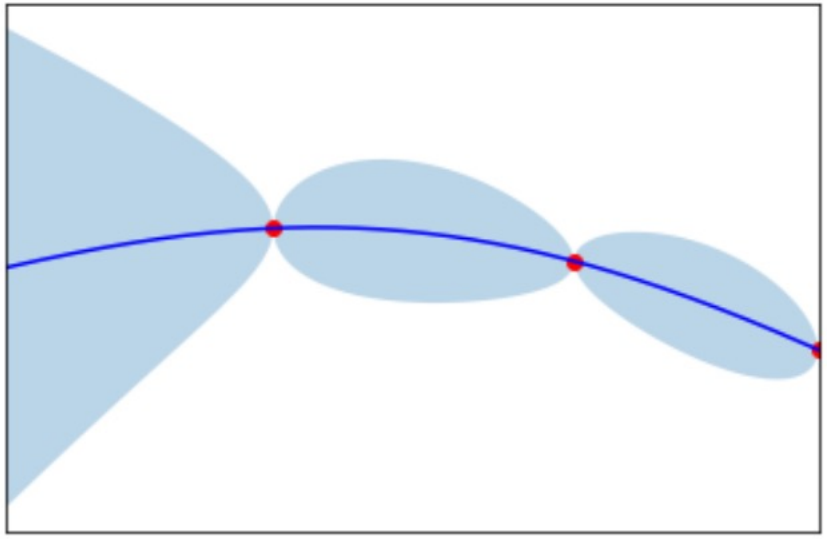
Open-source packages:
Many open-source implementation, e.g. scikit-learn, gpytorch

Outline

- Optimization problems in laser-plasma acceleration
- Gaussian Process models
- **Bayesian optimization**
- Applications in laser-plasma acceleration & current research
- Uncertainty in machine learning

Bayesian optimization

Now that we have an appropriate model:
we need to define a **rule** to automatically decide
which point to evaluate next.



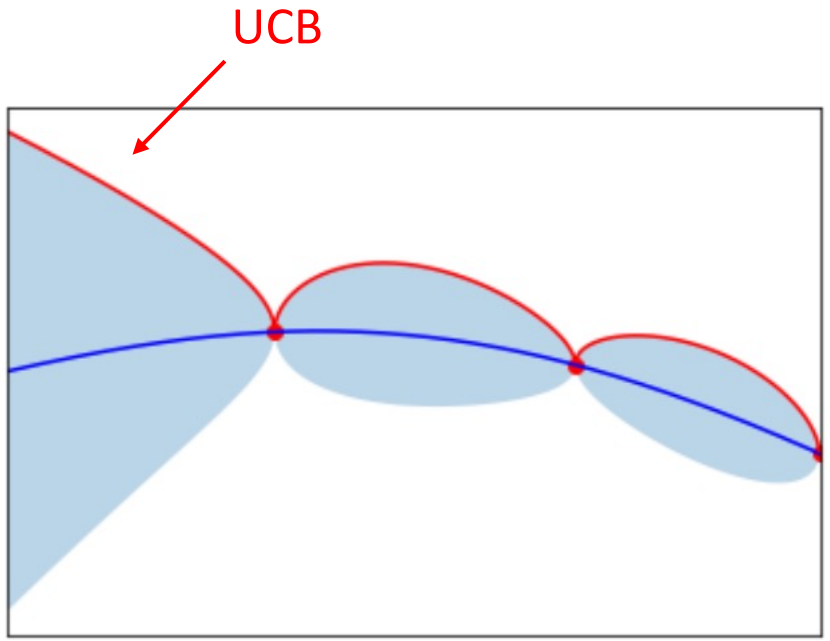
The rule should result in a healthy mix of:

- **Exploration:**
  Evaluating points where the function has high uncertainty

- **Exploitation:**
  Evaluation points close to the known best point,
  in the hope of finding an even better value.

Bayesian optimization

Rule:
Evaluate f at a point that maximizes
a well-chosen **acquisition function**.



Example of a standard acquisition function:
Upper Confidence Bound (UCB)

$$a(\boldsymbol{x}) = m(\boldsymbol{x}) + \beta\sigma(\boldsymbol{x})$$

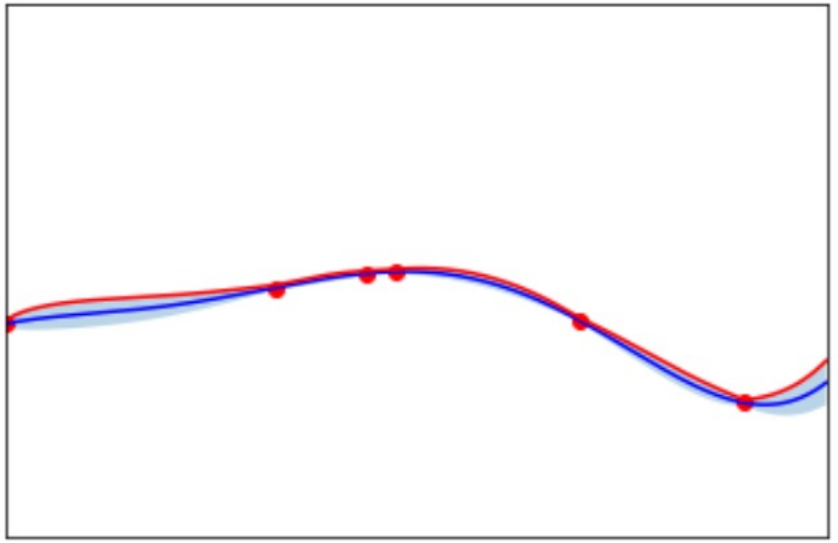
Example of another acquisition function: Probability of Improvement (PI), Expected Improvement (EI)

Bayesian optimization: full algorithm

Compute the acquisition function
Find the input $\mathbf{x_0}$ that maximizes it

Evaluate $f(\mathbf{x_0})$
(i.e. run a simulation, or take a measurement on the experiment)

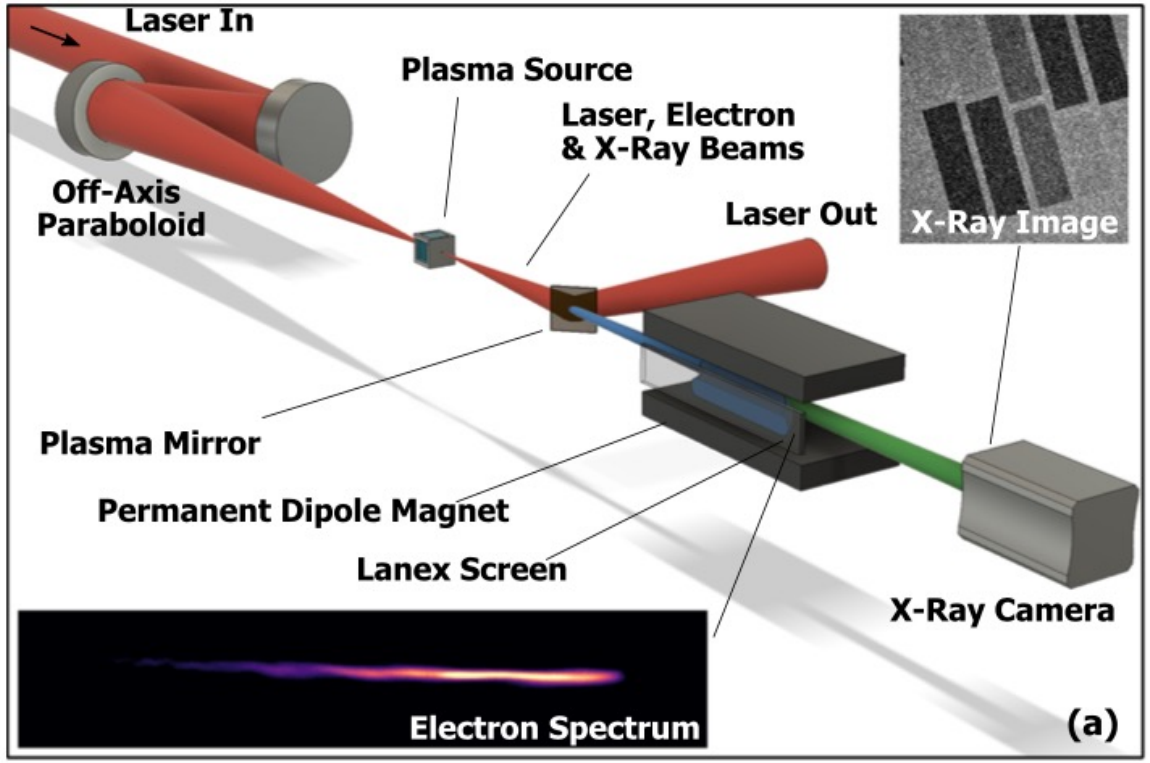
Add the point $\mathbf{x_0}$, $f(\mathbf{x_0})$
to the dataset and update the Gaussian Process model

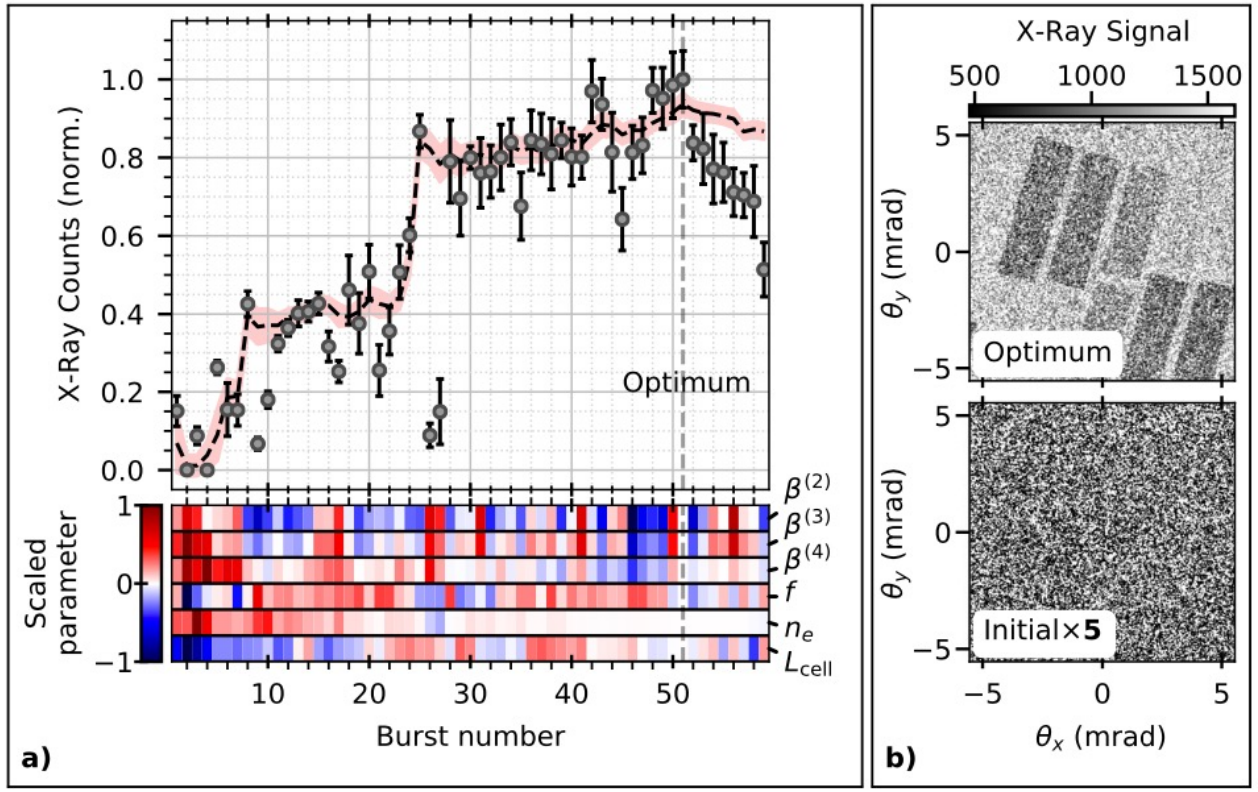


Outline

- Optimization problems in laser-plasma acceleration
- Gaussian Process models
- Bayesian optimization
- **Applications in laser-plasma acceleration & current research**
- Uncertainty in machine learning

Applications of Bayesian optimization in laser-plasma acceleration

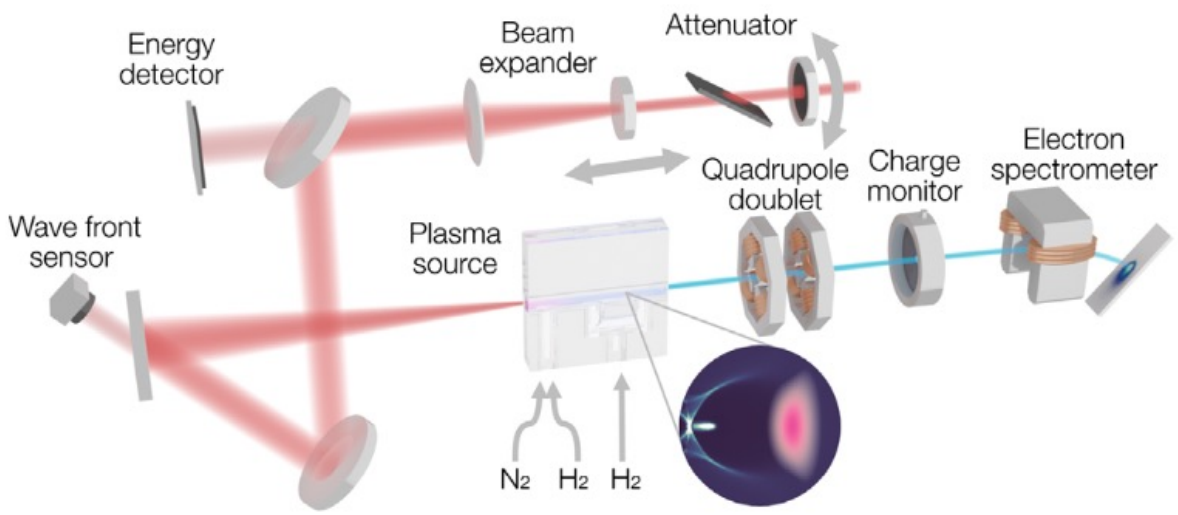


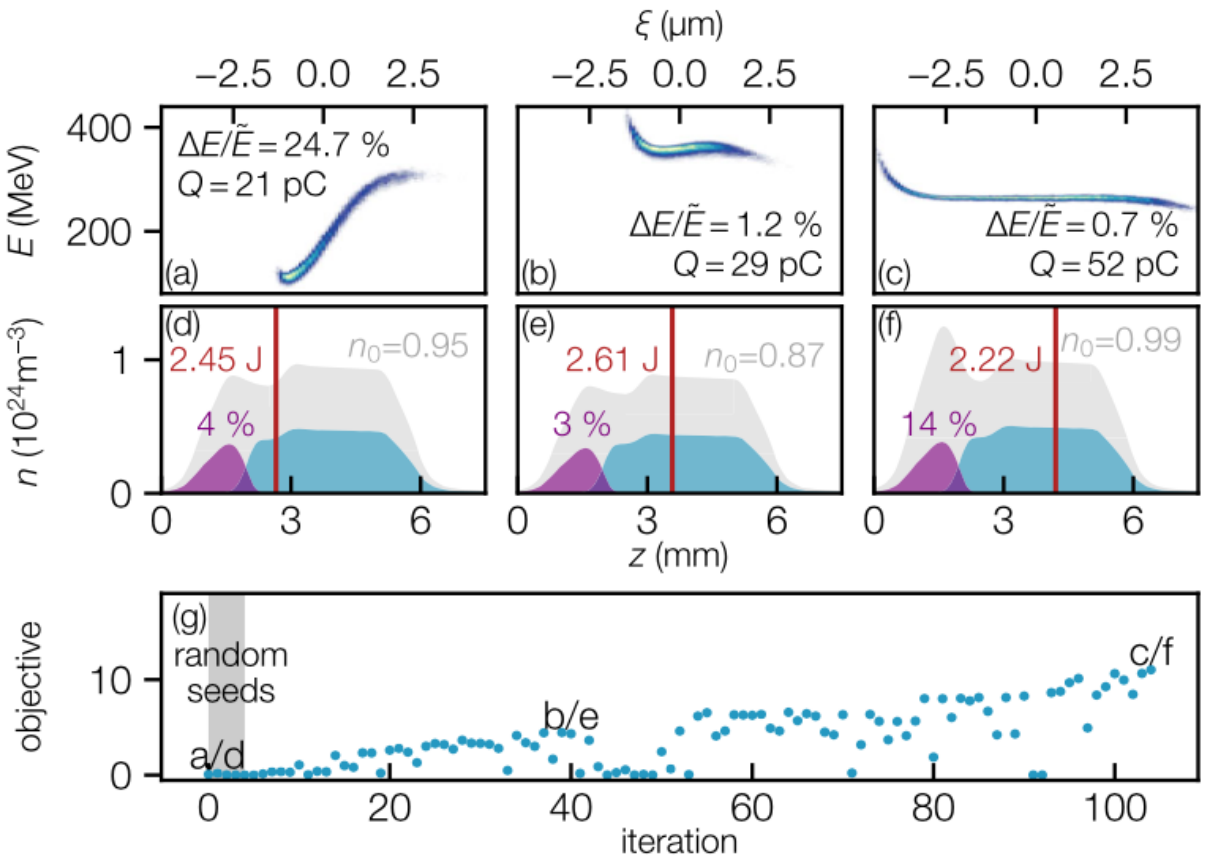


6 input parameters tuned simultaneously, to maximize the betatron X-ray yield.

R. Shalloo et al., Nature Communications (2020)

Applications of Bayesian optimization in laser-plasma acceleration





Tuning:

- background density
- amount of N2 injected
- laser energy
- laser focal position

in order to maximize **beam quality** in ionization injection $f = \sqrt{Q}\frac{\bar{E}}{\Delta E}$

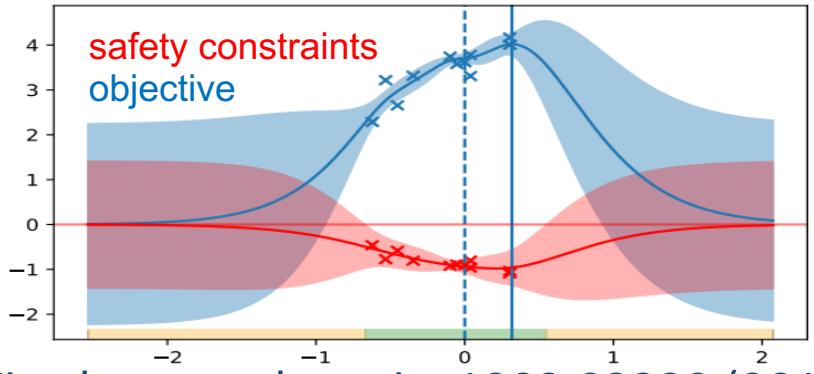
S. Jalas et al., PRL (2021)

Some areas of current research

- **Multi-fidelity Bayesian optimization**
  Using low-fidelity simulations to rapidly scan the parameter space and high-fidelity simulations when focusing on the optimal point

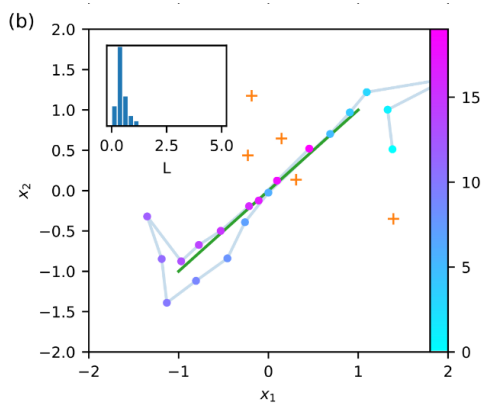
F. Irshad et al., arXiv:2112.13901 (2021)
F. Irshad, Heraeus seminar poster (2022)
A. Ferran-Pousa et al., IPAC 2022

- **How to satisfy safety constraints**
  esp. for quantities that are difficult to predict and require simulation / experiments (e.g. beam loss)



Kirschner et al., arxiv: 1902.03229 (2019)

- **Proximal optimization**
  For experiments: how to avoid repeated, large jumps in input parameters



R. Roussel et al., arxiv:2010.09824 (2021)

Some limitations of Bayesian optimization

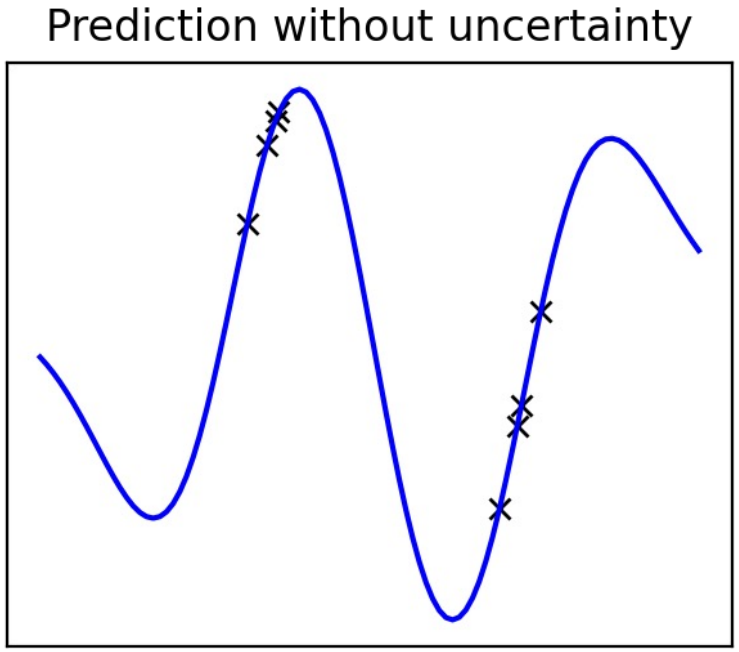
- Scaling with number of data points N (scales as N^3)
- Scaling with **input dimensions** (limited to ~10 input parameters in practice)
- Inefficient for **high-dimensional output**
  (essentially need to build a separate GP for each output)

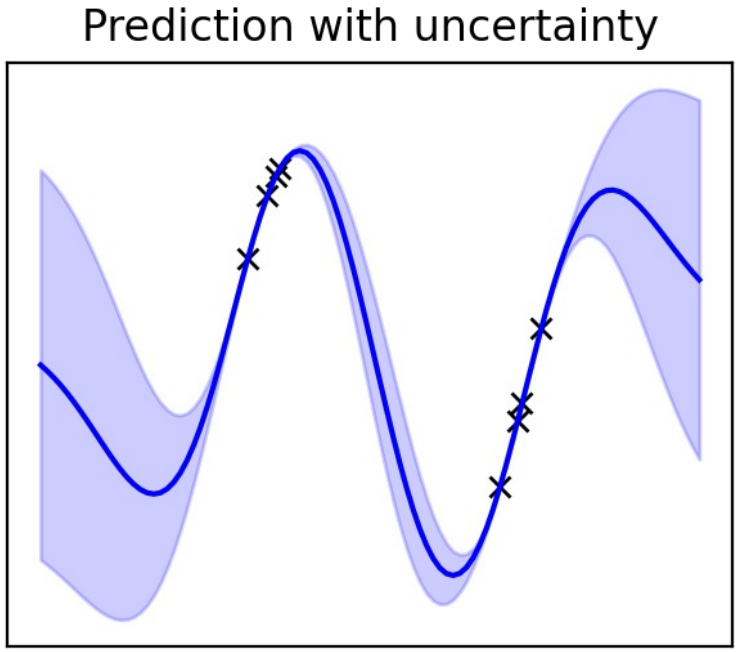
Outline

- Optimization problems in laser-plasma acceleration
- Gaussian Process models
- Bayesian optimization
- Applications in laser-plasma acceleration & current research
- **Uncertainty in machine learning**

Uncertainty in machine learning

Idea: The ML model should output a **prediction** and the corresponding **uncertainty.**

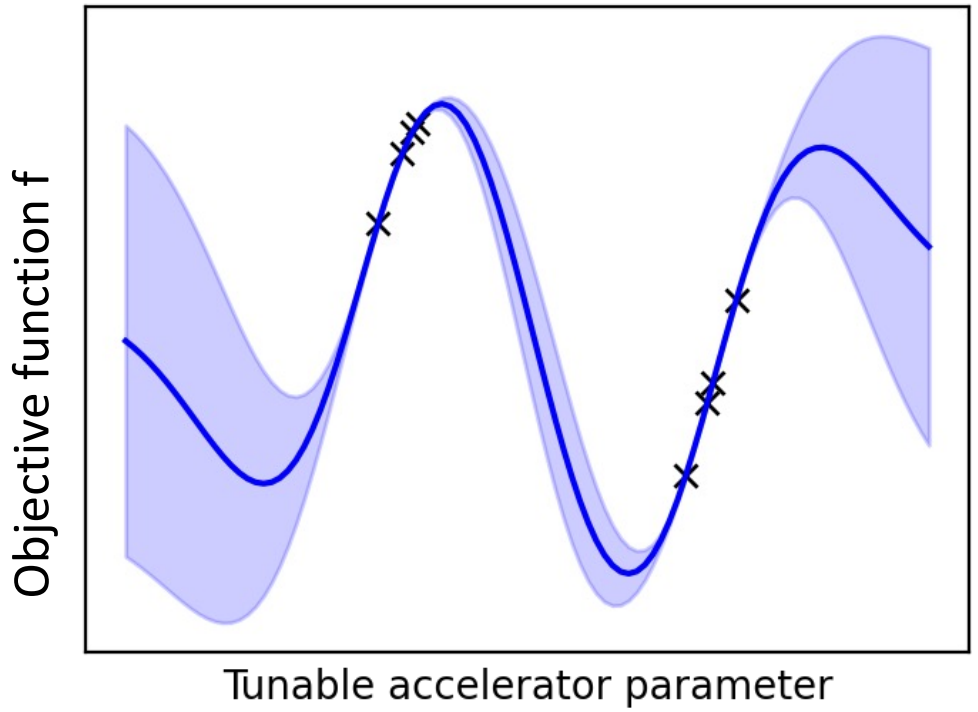




The uncertainty indicates the **probable interval** within which an actual evaluation may be. (e.g. actual measurement or simulation)

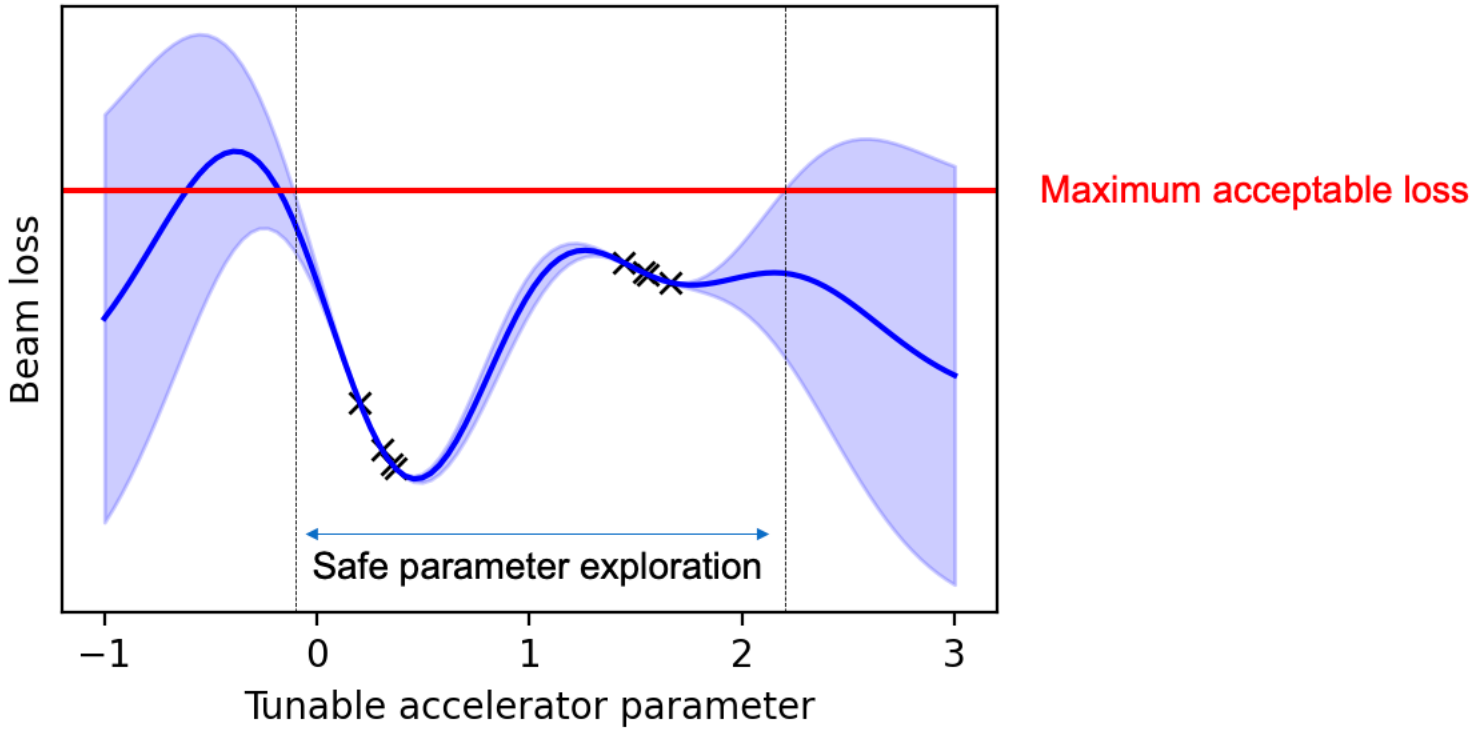
Motivation for accelerators: optimization

In the context of **model-based optimization** of accelerators:
uncertainty allows to balance **exploration and exploitation**.
(e.g. by calculating upper confidence bound, expected improvement)



Motivation for accelerators: safety constraints

For **safe operation** of accelerators:
uncertainty helps ensure that **important constraints** are not **violated**.



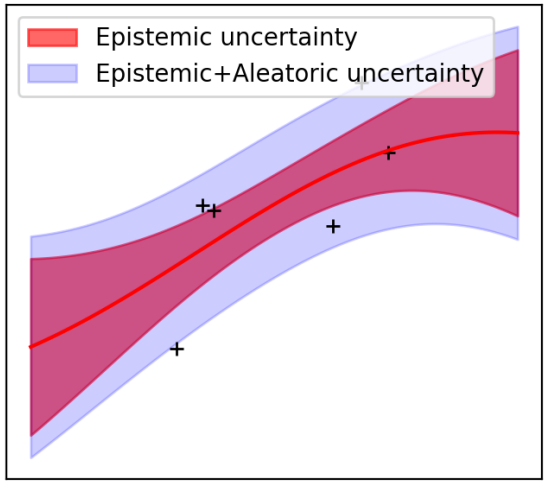
Epistemic and aleatoric uncertainty

Evaluations can often be modeled as:

$$f(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x}) + \eta$$

Underlying function
always gives the same result, for a given ***x***

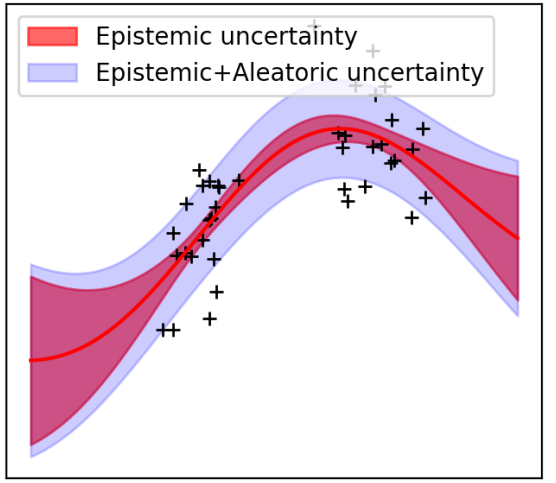
Intrinsic noise
value changes for each evaluation



Epistemic uncertainty:
uncertainty on underlying function

- increases when making predictions far from known data
- decreases when acquiring more data

Aleatoric uncertainty:
estimates the amplitude of the noise



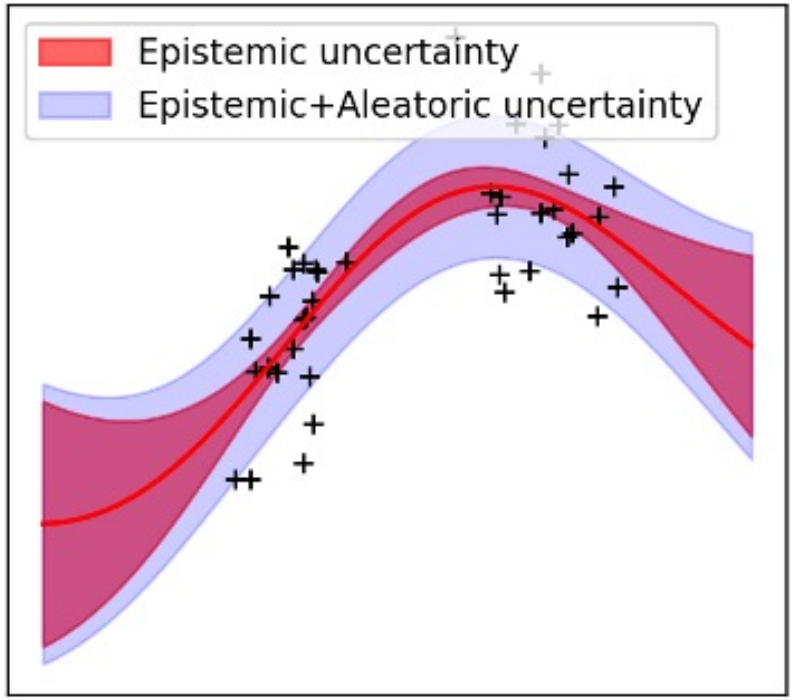
Epistemic and aleatoric uncertainty

Depending on the application, one may or may not want to include the **aleatoric part**:

Examples:

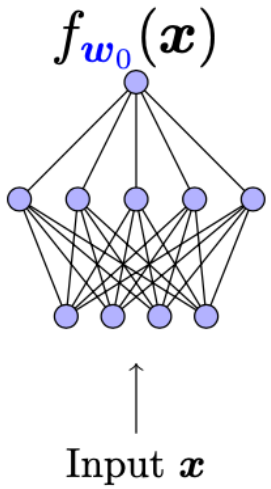
Optimizing beam emittance,
with noisy beam size measurements:
the aim is to optimize the underling function $\tilde{f}$;
the aleatoric part should not be included.

Keeping fluctuating beam loss under a threshold:
take into account aleatoric part, in order to evaluate the “worst-case scenario”.

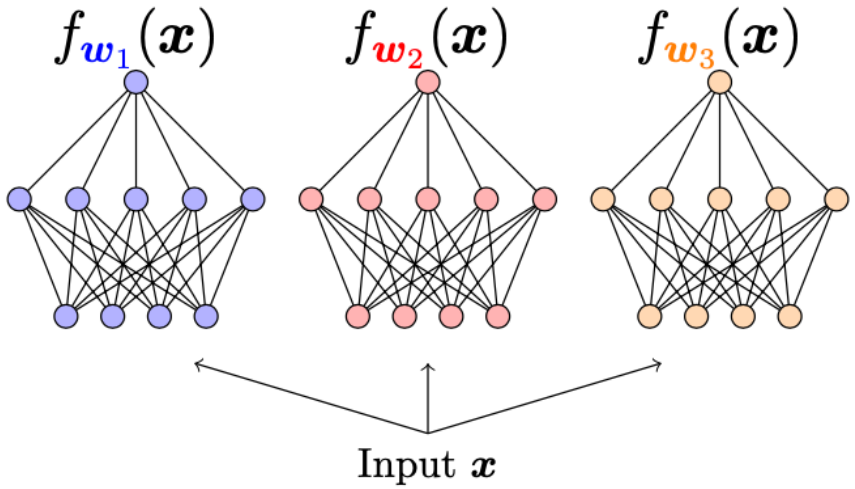


Obtaining uncertainty: ensemble of neural networks

Regular neural network



Ensemble of neural network (N=3)



Due to **randomness** in initialization and training, each neural network has **different weights**, and gives a **different answer**.

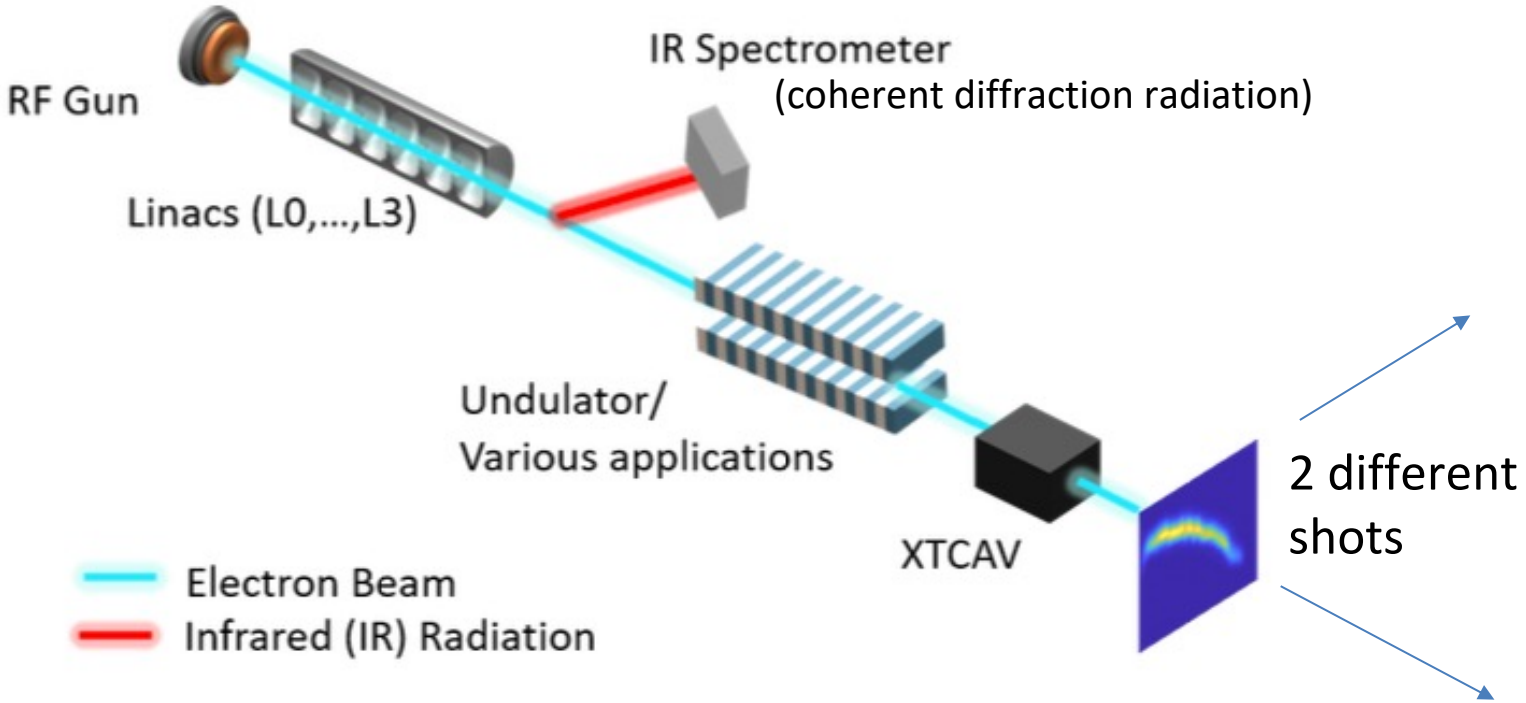
Use the **mean** as the **prediction**
Use the **standard deviation** as the **uncertainty**

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} f_{w_i}(x)$$

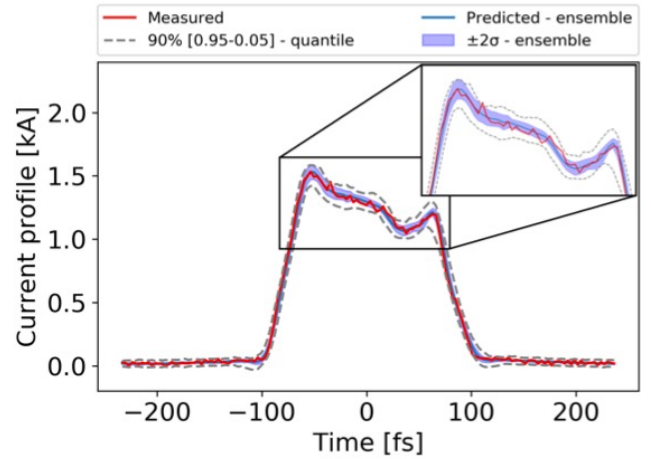
$$\sigma_f(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f_{w_i}(x) - f(x)\right)^2}$$

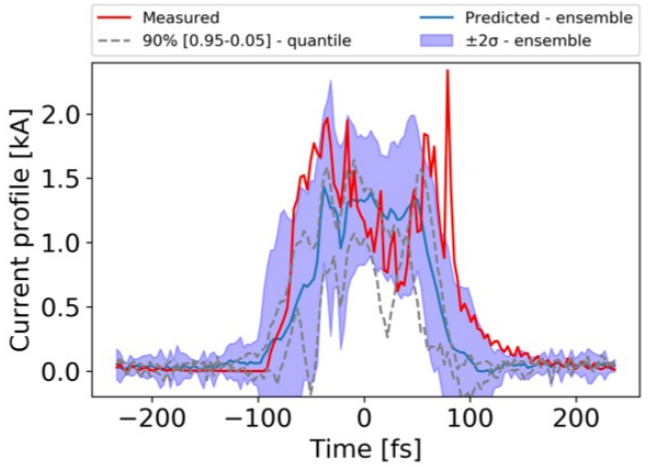
Example: uncertainty on virtual diagnostics

O. Convery et al., arXiv:2105.04654v1 (2021)



2 different shots





Ensemble of **16 independent neural networks**, trained with **bagging:**

- input: full IR spectrum
- output: 1d beam current profile

Conclusion

- **Gaussian process** is a machine learning model
  that can predict data and the corresponding uncertainty

- This model is used within **Bayesian optimization** in order to optimize
  a function while minimize the number of expensive evaluations (simulations or measurements)

- Bayesian has recently been used in several accelerators, for **autonomous tuning**

- More generally, estimating uncertainty is key for scientific applications,
  but the combination with the latest ML techniques (e.g. neural networks) is less mature.

Additional resources: U.S. Particle Accelerator School course on Optimization and Machine Learning for Accelerators

Course material + most videos freely available online at
https://slaclab.github.io/USPAS_ML/past_sessions/summer_2021/

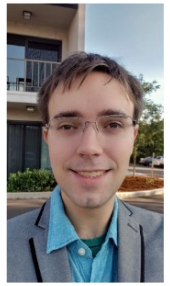
Instructors:

Auralee Edelen
(SLAC)

Adi Hanuka
(Eikon Therapeutics, prev. SLAC)

Remi Lehe
(LBNL)

Christopher Mayes
(SLAC)

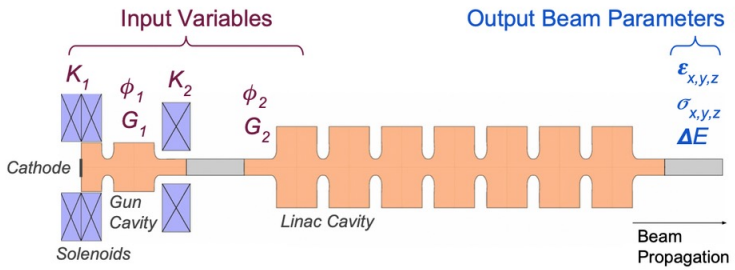


Ryan Roussel
(SLAC)

jupyter

Multi-objective optimization

Here we will find the pareto front of the AWA photoinjector problem (see below). Input variables are shown in red and output variables are shown in blue. Both the inputs and outputs are normalized to [-1,1]. Our goal is to minimize all of the output beam parameters.




```
In [9]:
# get AWA model
model = awa_model.AWAModel()
print(model.features)
print(model.targets)

x = torch.rand(5,6)
model.predict(x)
```

```
Out[9]:
['P0', 'P1', 'G0', 'G1', 'K1', 'K2']
['rms_x', 'rms_y', 'rms_s', 'emit_x', 'emit_y', 'emit_s', 'dE']
tensor([[-0.8086, -0.7883, -0.5743, -0.7200, -0.7163, -0.7291,  0.0773],
        [-0.5153, -0.5178, -0.7404, -0.6802, -0.6856, -0.8372, -0.2236],
        [-0.3100, -0.3126, -0.8209, -0.6308, -0.6387, -0.8739, -0.3733],
        [-0.7431, -0.7296, -0.6955, -0.6972, -0.7058, -0.7604,  0.0684],
        [-0.7036, -0.6949, -0.7944, -0.7247, -0.7290, -0.8013,  0.0774]],
       grad_fn=<TanhBackward>)
```

First try multi-objective optimization on a test problem

To start with we try a test problem - see description here https://datacrayon.com/posts/search-and-optimisation/practical-evolutionary-algorithms/synthetic-objective-functions-and-zdt1/ and here (page 488) https://ro.ecu.edu.au/cgi/viewcontent.cgi?article=3021&context=ecuworks

```
In [10]:
zdt = pg.problem(pg.zdt())
```

```
In [37]:
#do example NSGA-II optimization
```