Learning to Choose Optimizers

Martin van der Schelling (martin_van_der_schelling@brown.edu)[1] and Miguel A. Bessa (miguel_bessa@brown.edu)[2]

[1]Doctoral Researcher at Delft University of Technology, the Netherlands
[2]Associate Professor at Brown University, USA

ENGN2350 Data-Driven Design and Analysis of Structures and Materials — October 20, 2023

Objective: Use machine learning to choose optimizers for an unseen problem – "learning to optimize".
General Instructions: Each group has to deliver **one PDF report** and a **ZIP-file including their code**. The code should be easy to read, properly commented and it should be possible to replicate the results of the report.

Introduction

Conventional materials design is based on experimental trial-and-error. However, just like solving an exam by trial-and-error requires many attempts, this is an impractical strategy because an overwhelming number of combinations become possible as the design space increases. One way to address this issue is to create much more data by using predictive physics-based models[1], but simulating each design can be slow because these computer analysis need to be accurate. This remains true even considering the ever-increasing computational resources.

What if we had a more "intelligent" approach to design? In the past decade, machine learning techniques have soared to prominence in many applied fields of science and engineering. These techniques are able to discover patterns in input data, reducing the design space and informing the optimization process. The new generation of data-driven material design is here where data from predictive models and experiments can be used by machine learned models to discover new things. Once a model is trained, evaluating a material property using the model is significantly faster than measuring the property in an experiment or computing it in a simulation (1; 2).

F3DASM framework

The framework for data-driven design and analysis of structures and materials (`f3dasm`) is an attempt to develop a systematic approach of inverting the material design process (3). The framework integrates the following fields:

- Design of experiments (DoE), where input variables describing the microstructure, properties and external conditions of the system to be evaluated are sampled and where the search space is determined.
- Data generation, usually by computational analysis, where a material response database is created.
- Machine learning and optimization, where we either train a surrogate model to fit our experimental findings or iteratively improve the model to obtain a new design.

In this assignment the data generation part is very simple, as we will not be using any numerical method[2]. Instead, we will focus on the optimization part of the framework. The package is written in Python.

Optimization

Optimization is a critical part of the data-driven process. The goal is to find the best parameters for some objective defined by the objective function $f(\mathbf{x})$ that depends on an input vector $\mathbf{x} = (x_1, ..., x_d)$ with d variables and, in some cases, subjected to some equality or inequality constraints that define the feasible solution space $\mathcal{X}$. Each point in the solution space could, for instance, represent a micro-structure arrangement of a material.

Evaluating a sample of the objective function will result in an output vector $\mathbf{y}$. In analogy to the micro-structure input, this can represent some mechanical properties. For this project, we only consider single-objective optimization, i.e. where $\mathbf{y}$ is just a scalar. Note that generally we don't know the underlying objective function. We can only get information by sampling from it.

[1]Such as finite element analyses, molecular dynamics, density functional theory, etc.
[2]You will not need to setup nor run computer simulations to create new data.

Trying all possibilities to find the optimum is not really feasible, as it requires a lot of (computational) resources and time. Instead, we try to iteratively improve the current solution with an optimization algorithm.

For each iteration t, the current solution $\mathbf{x}_t$ is altered to acquire a new solution $\mathbf{x}_{t+1}$. Ultimately, we want to find an optimal set of values $\mathbf{x}^*$ that will minimize or maximize the objective function $f(\mathbf{x})$. We can express a minimization optimization problem with the following formulation:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \; \forall \; \mathbf{x} \in \mathcal{X} \tag{1}$$

A minimization in $f(\mathbf{x})$ is the same as a maximization in $-f(\mathbf{x})$. For the sake of consistency, we will view the optimization as a minimization problem for f.

Learning to optimize

When solving a new optimization problem, we do not know what is the best optimization algorithm and its parameters that would quickly get us to the optimum. Unfortunately, the "one optimizer to rule them all" does not exist. According to the famous paper of Wolpert et al. (4), algorithms tend to exploit certain problem-specific characteristics, and if this particular feature is not present in the optimization problem then the performance of the optimizer will be worse than random search, i.e. random sampling (random trial-and-error!).

However, what if we could train a machine learning model on many different optimization problems considering many different optimizers? The machine learning algorithm would start to learn when an optimizer should be used or not. If it is trained properly, then the next time we consider a new problem this algorithm could make an informed decision on which optimizer to use at a given iteration! This is a new field of research called "learning to optimize".

Adequate training so that a machine learning algorithm can "learn to optimize" is challenging because being prepared for any kind of optimization problem requires a diverse set of algorithms to mix and match and a diverse set of optimization problems to learn from.

In the masters thesis of van der Schelling (5), different possibilities of a implementing a general optimizer that can adapt to problem-specific features of the input data have been investigated. The thesis proposed a data-driven framework for general optimization problems to adapt the algorithm during optimization. With an adaptive data-driven optimization strategy, we eliminate the need to hand-design a machine learning model. By integrating this idea into the `f3dasm` framework, the proposed software package does not require any machine learning architectural choices beforehand. This will increase accessibility of the framework to pure experimentalists.

Aim of the project

The aim of the project is to design a general data-driven optimization strategy with machine learning that competes with the conventional hand-engineered optimizers. You will investigate how different optimization algorithms behave on different problem-specific characteristics.

You will use the `f3dasm` framework to extract the data to build your own machine learning model. You will **not** generate any new data, but use the datasets provided (`big_dataset`). At the end of the project, you have created a model that will recommend one of the given optimizers to a new optimization problem for the entire training. You will benchmark your 'strategy' to the individual static optimizers.

At the end of the project, all the groups have created their data-driven optimization strategy model and will compete to see who has the best model! The models will be assessed on two test sets: one of the sets will be sampled from the same domain as the training set and the other test set will have benchmark problems that are slightly out-of-distribution, e.g. the boundaries of the parameters are shifted slightly. These sets will not be shared with you during the project, you have to use the training set to train your model.

At the end of the project you will gain:

1. Knowledge on different optimizers and why they do or do not work.
2. Understand the key challenges of selecting appropriate machine learning models for applications.

Provided data and resources

The students will receive two dataset of the training dynamics of a set of benchmark functions, optimized with a variety of hand-engineered optimizers:

- a big dataset with 2943 benchmark functions (`big_dataset`)
- a subset of the big dataset ("small dataset") with 100 benchmark functions (`small_dataset`)

Additionally the students will get a Python file (`l2o.py`) which contains helper-functions and protocol classes to be used in the project.

Design-of-experiments

The design of experiments consists of the following parameters: the name of the benchmark-function[3] (`function_name`), the number of input dimensions d (`dimensionality`), the standard deviation of Gaussian noise on the objective value (`noise`), the seed of the random number generator (`seed`) and the maximum number of function evaluations (`budget`).

All the benchmark-functions have been altered in the following way:

- Each benchmark-function is off-set by a random vector. As most of the benchmark functions have their global minimum in the center of the design space, we want to avoid optimizers to "cheat" their way to the optimum by immediately jumping to the origin.
- Each benchmark-function has been scaled to the $[0, 1]^d$ domain.

Before optimization, 30 initial solutions have been sampled in the search-space $[0, 1]^d$ with Latin Hypercube sampling. Each of the benchmark-functions is optimized for the given budget with 5 optimizers. No prior hyper-parameter optimization has been done on these algorithms; we have used the default hyper-parameters. The optimization algorithm is forced to give solutions inside the box-constraint boundaries $[0, 1]^d$.

The following optimizers have been used:

- Adam (source)
- Covariance matrix adaptation evolution strategy (CMA-ES) (source)
- Limited-memory BFGS with box constraints (L-BFGS-B) (source)
- Particle Swarm Optimization (PSO) (source)
- Random Search (source)

Each optimization run is repeated for 10 realization; which means that we start with different initial samples and a different seed to the optimization algorithm. The off-set of the benchmark functions remains the same across realizations!

The design-of-experiments can be found in the `small_dataset_domain.pkl` pickle file.

Input data

The input data are randomly sampled benchmark functions with the parameters described in the previous section. It can be found in the `small_dataset_input.csv` CSV file.

Output data

The output data consist of two `xarray` DataSet objects per experiment:

1. `raw`: Complete history of the optimization trajectory for each of the optimizers on the benchmark problems
2. `post`: Combination of post-processed data and the characteristics of the benchmark problem

Since the files for the complete optimization trajectories are very large (for the big dataset 86 GB combined), **only the small dataset will come with the raw files included**. Thus, the big dataset will only have the `post` data. The small dataset will have both the `raw` and `post` data.

[3] All the implemented benchmark-functions can be found here.

Coordinates	Size	Values
iterations	`budget`	`range(0, 2030)`
optimizer	5	`['CMAES', 'PSO', 'Adam', 'LBFGSB', 'RandomSearch']`
realization	10	`range(0, 10)`
input_dim	`dimensionality`	`['x0', 'x1' ... ]`
output_dim	1	`['y']`

Data Variables		Data-type
input	`optimizer x realization x iterations x input_dim`	`numpy array (dtype=float64)`
output	`optimizer x realization x iterations x input_dim`	`numpy array (dtype=float64)`

Attributes	Data-type
- number_of_samples	`int`
- realization_seeds	`list[int]`
- function_seed	`int`
- function_name	`str`
- function_noise	`float`
- function_dimensionality	`int`
- function_global_minimum	`float`
- function_features	`list[str]`

Table 1: Summary of the `xarray` dataset '`raw`'

Raw optimization data

Full optimization history of 2030[4] iterations and 10 realization for each of the 5 optimization algorithms. The data variables are:

- `input`: The input ($\mathbf{X}_{0..t}$) for each of the 10 realizations and each of the 5 optimization algorithms for the 2030 iterations. The number of input dimensions is dependent on the chosen value of d (`dimensionality`).
- `output`: The output-parameters ($\mathbf{y}_{0..t}$) for each of the 10 realizations and each of the 5 optimization algorithms for the 2030 iterations. In this project we consider a single-objective optimization problem.

This is stored as a NetCDF file in an `xarray` Dataset format. In the `f3dasm` output data file (`small_dataset_output.csv`), you will find a relative filepath to this file for each of the benchmark functions at the column `'path_raw'` Table 1 gives a summary of the structure of this dataset.

Post-processed optimization data

Post-processed data of `raw`. For each problem, 9 additional sub-problems are created where we consider a restricted number of iterations[5]. Sub-problems are sampled within [30, 2030] with Latin Hypercube sampling. Each of the sub-problems have an unique ID which we call `itemID`. The data variables in the `xarray` are features of each of the sub-problems:

- `dim`: the number of input dimensions of the sub-problem.
- `budget`: the maximum number of function evaluations of the sub-problem.
- `noise`: if the benchmark function is noisy (1) or not (0).
- `convex`: if the benchmark function is convex (1) or not (0).
- `separable`: if the benchmark function is separable (1) or not (0).
- `multimodal`: if the benchmark function is multimodal (1) or not (0).
- `output_samples`: the objective value vector of the 30 initial solutions for each realization.
- `ranking`: Ranking[6] of the optimization algorithms on the sub-problem. Lower means better!

[4] This number is the initial samples (30) + the 2000 iterations = 2030

[5] In total there are 10 sub-problems, including the original problem

[6] This metric is calculated by taking the best found solution after the maximum number of function evaluations of the sub-problem for each optimizer and taking the median value over the realizations. Then, the values are min-max scaled between the best and worst objective value in the dataset.

Coordinates	Size	Values
itemID	10	`range(job_number, job_number+10)`
optimizer	5	`['CMAES', 'PSO', 'Adam', 'LBFGSB', 'RandomSearch']`
realization	10	`range(0, 10)`
output_dim	1	`['y']`
iterations	30	`range(0, 30)`

Data Variables		Data-type
dim	`itemID`	`numpy array (dtype=int64)`
budget	`itemID`	`numpy array (dtype=int64)`
noise	`itemID`	`numpy array (dtype=int64)`
convex	`itemID`	`numpy array (dtype=int64)`
separable	`itemID`	`numpy array (dtype=int64)`
multimodal	`itemID`	`numpy array (dtype=int64)`
samples_output	`itemID x realization x output_dim x iterations`	`numpy array (dtype=float64)`
ranking	`itemID x optimizer x output_dim`	`numpy array (dtype=float64)`
perf_profile	`itemID x optimizer x realization x output_dim`	`numpy array (dtype=float64)`

Table 2: Summary of the `xarray` dataset '`post`'

- `perf_profile`: the performance profile for each sub-problem, realization and optimizer. This is used for plotting the performance over all the benchmark functions.

This is stored as a NetCDF file in an `xarray` Dataset format. In the `f3dasm` output data file (`small_dataset_output.csv`), you will find a relative filepath to this file for each of the benchmark functions at the column `'path_post'` Table 2 gives a summary of the structure of this dataset.

Getting started

Reading

1. Read sections 2.2, 2.3 and 2.4 from the literature review part of this MSc thesis: A data-driven heuristic decision strategy for data-scarce optimization
2. Read about the metric 'performance profile' we are using in the dataset in this paper note.
3. Apart from the Python packages you have seen during the class (`numpy`, `pandas` and `matplotlib`), we will make use of the `xarray`: a package for structuring multidimensional data. You can look around at their documentation page to learn more about it.

Installation

1. Make sure you have cloned and updated the `3dasm_course` GitHub repository.
2. Download the dataset from this Google Drive link.
3. Inside the `post` and `raw` folders of both datasets, unpack the compressed `.zip` files.
4. In your `3dasm` environment[7], install `f3dasm` version 1.4.3 and `dask`[8] packages:

```
pip install f3dasm==1.4.3 dask
```

[7] If you have problems with running Python locally, you can use Google Colab for this project.
[8] The `dask` package is used for parallel lazy-loading of multiple files

Questions to be answered

For the first three questions, you only have to consider the `small_dataset`.

Investigate the optimization data

To get familiar with the dataset, try to replicate the following experiments:

1. We start off by opening one of the experiments `raw` data of the **small dataset**

   1.1. Choose the experiment with index 691 from the dataset. Note down the characteristics (e.g. dimensionality, noise level, noticeable loss-landscape characteristics). Before looking at the results: which optimizers do you expect to solve the problem with ease?

   1.2. Open the `raw` dataset in Python. You can use the `open_one_dataset_raw()` helper function.

   1.3. Plot the objective values 'y' found with the Adam optimizer for the first realization with respect to the iteration number.

   1.4. Repeat question 1.3 for the second realization. Is there a difference in the performance? If so, can you explain why?

   1.5. In the figure you created in questions 1.3, overlay the objective values 'y' found with the remaining optimizers for the first realization. Are they behaving differently?

   1.6. We want to compare the best found solution at a given iteration in a so-called convergence plot. Take the cumulative minimum of the objective values with respect to the iteration number for each of the realizations. After that, take the median value over the realizations. Repeat this for every optimizer in the dataset[9].

   1.7. Plot the results you obtained in the previous exercise. Make sure your figure has a legend and you label each optimizer. Which optimizer is the 'winner' and why?

   1.8. Load up experiment 280 and repeat questions 1.6 and 1.7 for this dataset How do these results relate to the 'No Free Lunch' theorem (4) ?

Filter on the problem characteristics

2. In order to compare the optimization performance across different problems, we have to use a performance metric. To save you the hassle of post-processing the raw data, I have already done that for you in the `post` files with the 'performance profile' metric.

   2.1. Load the *small dataset* `post` dataset. You can use the `open_all_datasets_post()` helper function.

   2.2. Plot the performance profile of the small dataset. You can use the `plot_perf_profile()` helper function.

   2.3. Now we will be looking at particular sub-sets of the benchmark problems. We will filter the dataset on particular problem characteristics[10]. Consider the following 3 filters:

      i. All problems that have a dimensionality that is 10 or lower

      ii. All problems that are both noiseless and convex

      iii. All problems that are both separable and a maximum budget of 100 iterations or lower

      Plot the performance profiles of each of the three sub-sets. Can you explain the differences between the performance profiles?

Data-driven optimization strategy

3. Now we are going to create and learn a model that automatically chooses an optimizer to use on a particular benchmark problem!

   3.1. Import the `CustomStrategy` class from the code provided and create a new class that inherits from this class.

      - Add a class attribute `name` with the name of your strategy

[9] Hint: you might want to use the cummin() function from the `pandas` library!
[10] Hint: you might want to use the xarray.Dataset.where function!

- Add a `predict()` method to the class that accepts `post` dataset. This function returns an 'iterable' (e.g. a list) with names of the optimizers that have been recommended: one of '`"Adam"`, `"CMAES"`, `"LBFGSB"`, `"PSO"` or '`"RandomSearch"`'.

  You can use the following template to create your own strategy class:

```
class YourStrategy(CustomStrategy):
    name: str = "your_strategy_name"

    def predict(self, features: xarray.Dataset) -> Iterable[str]:
        ...
        # Here your prediction model!
        return prediction
```

  You can access the individual features from the `post` dataset in your function: `features.dim`, `features.budget`, `features.noise`, `features.convex`, `features.separable`, `featuresmultimodal`, `features.samples_output`.

  Note that you can add other methods to `YourStrategy` to your liking!

3.2. Import the KNeighborsClassifier from `scikit-learn` and create a classifier with $K = 5$ neighbours.

3.3. Train the classifier on the `dim`, `budget`, `noise`, `convex`, `separable` and `multimodal` features for each sub-problem. As labels you can use the optimizer with the best (=lowest) `ranking` value for each sub-problem!

3.4. After you have trained the model, implement it in your `YourStrategy` object so that the `predict()` function will predict the labels of the test problems.

3.5. In order to evaluate your strategy, create an instance of the `StrategyManager` and pass it the `ExperimentData` object of your test set and a list of the strategies that you want to evaluate. Then, you can call the `compute_performance_profiles()` method to create the performance profile of each of the single static optimizers, the best strategy of the test set (upper bound), the worst strategy (lower bound) and the strategies you have provided:

```
my_strategy = YourStrategy(...)
strategy_manager = StrategyManager(experimentdata, [my_strategy])
performance_profiles  = strategy_manager.compute_performance_profiles
    ()
```

You can also plot the performance profile directly with the `strategy_manager.plot()` function.

Create you own optimization strategy model!

4. Now it is time to get creative! Create your own data-driven optimization strategy with any model that you think is suitable for the task and train/evaluate it on the `big_dataset`!

   - You can not use the `ranking` and `perf_profile` for predicting, since you won't normally have that information during the testing phase!
   - Benchmark your data-driven optimization strategy against the k-NN classifier you created in question 3.
   - Do hyper-parameter optimization of your model and report the performances of the different instances.

   Note: For all the models that you train, you must report the hyper-parameters and the choices you made. Don't be afraid to report the results of models that perform poorly: the knowledge gained from failed solutions is as valuable as the one gained from solutions that actually work. You should also provide a short description of the new machine learning algorithm considered in your project if it was not covered in class.

Report

Each group has to deliver **one PDF report** and a **ZIP-file including their code**.

- Make sure that your data-driven strategy created in question 4 follows the format of the `CustomStrategy` class.
- You can use Jupyter notebooks for the Python code. If you are using external libraries, mention them and the used version in the Jupyter notebook.
- The code should be easy to read, properly commented and it should be possible to replicate the results of the report.

References

[1] S. Kadulkar, Z. M. Sherman, V. Ganesan, and T. M. Truskett, "Machine Learning-Assisted Design of Material Properties," *Annual Review of Chemical and Biomolecular Engineering*, vol. 13, mar 2022.

[2] K. Guo, Z. Yang, C.-H. Yu, and M. J. Buehler, "Artificial intelligence and machine learning in design of mechanical materials," *Materials Horizons*, vol. 8, no. 4, pp. 1153–1172, 2021.

[3] M. A. Bessa, R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen, and W. K. Liu, "A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality," *Computer Methods in Applied Mechanics and Engineering*, vol. 320, pp. 633–667, jun 2017.

[4] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[5] M. van der Schelling, "A data-driven heuristic decision strategy for data-scarce optimization: with an application towards bio-based composites," mathesis, Delft University of Technology, Mar. 2021.