Application for Remote Physics Simulation and Machine Learning

Jerome Onwunalu, <jerome.onwunalu@thedibia.com>

1. Introduction

This brief document provides some background and motivation for developing the integrated application for performing remote physics simulations, optimizations, and machine learning. The application contains other modules but this document focuses on the modules shown in Figure 1.

Physics Simulation
Optimization
Machine Learning

Figure 1: Three modules of the integrated application: Physics Simulation, Optimization and Machine Learning

In general, each of the modules can be used separately but this document focuses on options that integrate one or more modules. The tool is deployed on the cloud, and interactions with the tool are via HTTP REST API calls.

2. Physics Simulation Module

The physics simulation module contains several tools for performing 1D, 2D, and 3D fluid flow simulations on a grid. For more advanced flow models that require third-party commercial vendors, the module can generate all required input files required by the commercial applications.

In general, to perform a physics simulations, we have a set of base (x_{base}) and design parameters (x_{design}). The base parameters are parameters for the model that are fixed while the design parameters are parameters that can be altered and optimized to improve some objective function. The union of both sets of parameters constitute the necessary parameters to perform the physics simulation. Specifically, $x = [x_{base}, x_{design}]$, where x is the vector of parameters required for a simulation run.

Let $F(x)$ represent some objective function value after performing simulation with input vector x. As described above, x can be decomposed into x_{base} and x_{design}. The design parameters can be optimized by an optimization algorithm so we can improve the value of $F(x)$. Computing the value of $F(x)$ requires a physics simulation run, usually carried out remotely on the cloud servers. The optimizer is used to determine 'optimal' values of x_{design}. The solution space of parameters can be large and because of the nature of the optimization problem,special optimization algorithms are required which are described in the next section.

2.1 Challenges

The physics simulations are expensive and the computation resources increase with the size of the grid and also with the duration of the simulation.

3. Optimization Module

Optimization algorithms are employed to find the best design parameters for x_{design}. In general, the optimization problem is non-smooth, discontinuous and multi-modal. Figure X, shows the actual objective function surface for an problem taken from literature (Onwunalu and Durlofsky, 2009):

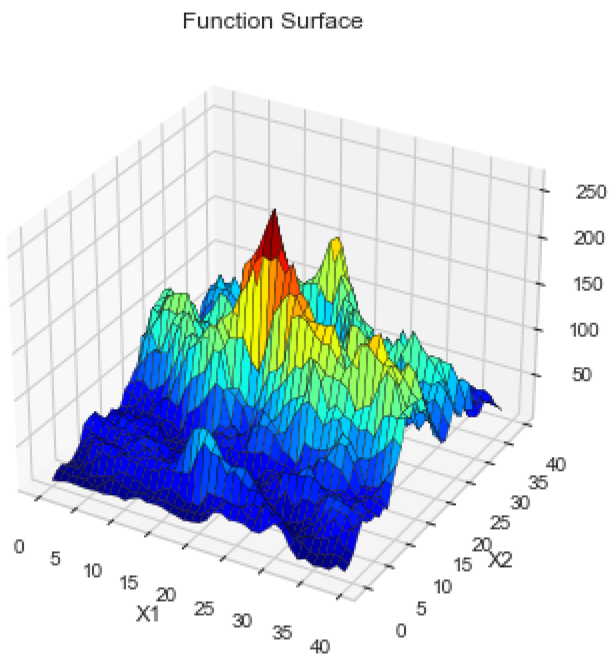

Figure 2: True objective function surface for a two-dimensional problem.

We use global stochastic optimization algorithms, e.g., genetic algorithms (GA) and particle swarm optimization algorithms (PSO), due to the nature of the optimization problem. These algorithms are less susceptible to get stuck in local minima compared to gradient-based algorithms.

In each iteration, the optimizer processes multiple solutions $x_{design}(i)$ where *i* represents the index of solution in the current iteration . Each of these values are combined with x_{base} before computing the objective function.

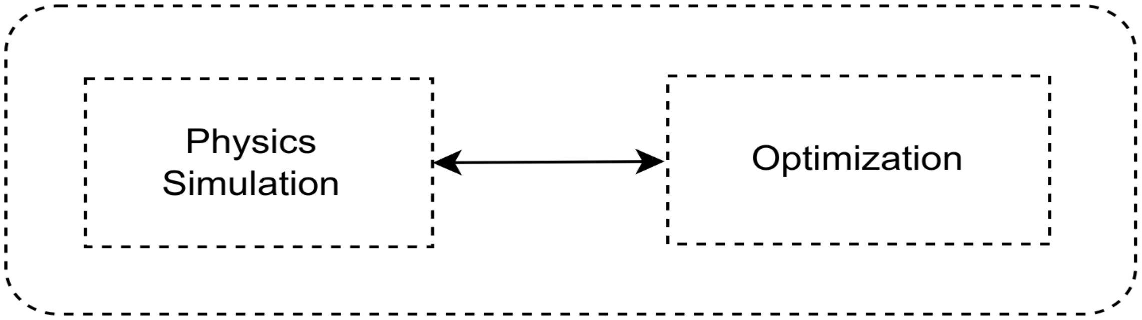

Figure 3: Interaction of the physics simulation and optimization modules. The optimizer optimizes the x_{design} parameters that are passed to the physics simulation module.

3.1 Challenges

We use global stochastic optimization algorithms, e.g., genetic algorithms and particle swarm optimization algorithms, due to the nature of the optimization problem. These algorithms tend to require many function evaluations leading to increased computational requirements. Evaluating the objective function for each sample requires performing physics flow simulations defined on a grid.

In general, the computational costs increase with the size of the grid. Models with larger grids require more resources and generally take longer to run. Additionally, the stochastic optimizers employed usually evaluate many samples and most of these samples will be poor solutions. If we can build a proxy or surrogate model (in place of physics simulation) with sufficient accuracy while running the optimization, we can use the proxy to compute prior estimates of the true objective function and decide if we should perform the simulation for each new sample of x_{design} (described later). We use machine learning to build a model that learns during the optimization run and use the model to estimate objective function values. Note that the machine model is used in an 'online fashion' and often created from scratch for each new problem instance. The next section describes the machine learning module.

4. Machine Learning Module

As described in the previous section, the machine learning model is built and run during the optimization process. The optimization algorithms are run iteratively and each iteration generates samples of x_{design}. After the simulation step, we obtain the corresponding objective function value $F(x)$ for all samples. The samples $(x, F(x))$ form the attributes and targets for our machine learning model. Thus, we have a supervised machine learning problem. Additionally, it is a regression problem because the the values of $F(x)$ are real continuous values.

From the attributes, the application constructs a final feature matrix (after removing any correlated features) that is used in the model development. Note that we use both the base and design parameters in the attribute set due to interaction of the both parameters via the physics model.

After creating the feature matrix, we perform a train-validation-test split of the data. We train the models using the train data, evaluate with the validation data, and compute generalization error using the test data and select the best model for estimating the objective function values.

We repeat the machine learning pipeline because the size attribute-target data increases after each optimization iteration. Specially, we repeat the ML pipeline after every k iterations where $k \geq 1$. The accuracy of the ML model will, in general, increase with iteration as we add more samples to the database of attribute-target pairs.

The application can use simple models like k-nearest neighbors (kNN) and k-means and fuzzy clustering. The latter models are implemented (from scratch) in Lisp in the application. In some cases, we use a mixture of models, for example, we use a kNN model in the early stages of the run (when we have insufficient data) and then switch to more robust ensemble tree-based methods, e.g. Random Forests. For the latter models, we utilize the scikit-learn machine learning library.

It is possible that the accuracy of the model is not high. In that case, we continue to compute the objective function using the more expensive physics simulation. If the model accuracy is high enough, e.g., > 75%, we can use it to get a prior estimate, $F^{prior}(x)$ for every x_{design} generated by the optimizer. Subsequently, we can then use a threshold (cut-off) objective function value to decide if we should perform the more expensive physics simulation to get the true values. If the prediction from the ML model is greater than some threshold, we perform a simulation. If it is less than the threshold, we don't perform the physics simulation step because the solution is likely to be poor.

Using an objective function value filter helps us reduce the computational requirements by ensuring that we perform simulations only for the most 'promising solutions'. Additionally, the threshold is adjusted dynamically because in general, the average of objective function values of samples tend to increase with iteration. Figure X shows the interaction of the physics simulation, optimization and machine learning modules.

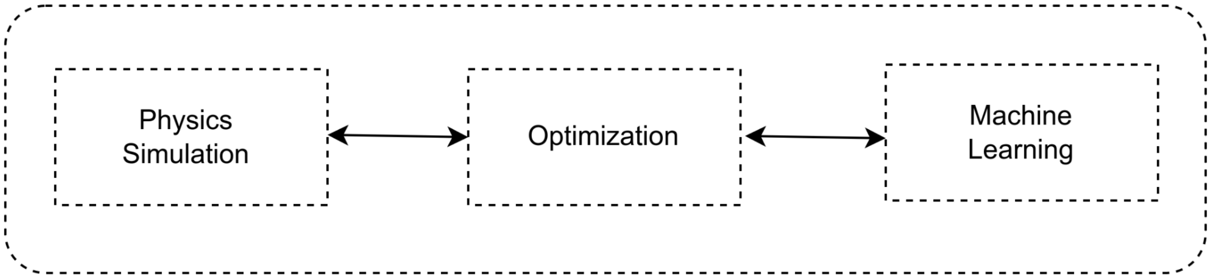

Figure 4: Interaction of the physics simulation, optimization, and machine learning modules. Samples of x and $(x,\ F(x))$ are sent from the optimization module to the machine learning module to facilitate building a regression model using supervised learning techniques.

4.1 Challenges

The generation of sample data for the machine learning model comes from the optimization algorithm. As previously described, we use iterative, global stochastic optimization algorithms (.e.g., PSO algorithm). If care is not taken, e.g. when the initial samples are generated randomly, these samples may not be well distributed in the solution space. If we build the machine learning model using these poor spatially distributed samples, the accuracy of the model will suffer when solutions in other parts of the search space are generated. So it's important to ensure samples have good coverage of the search space.

We mitigate the problem of poor initial sample distribution by ensuring that we generate uniformly distributed samples in the first iteration of the algorithms and also by balancing exploration and exploitation components of the optimizer. A better distribution of the initial samples can be obtained using experimental design techniques. Balancing the exploration and exploitation components of the optimizer is obtained by ensuring we have more exploration in early iterations, and more exploitation in the latter iterations.

5. Conclusions

This brief document provides some details and background of an integrated application for remote physics simulation, optimization and machine learning. We developed an integrated tool that addresses the challenges described earlier. The tool was designed and implemented by the author over several years. The project was recently completed and it is now currently deployed and operational.