Ahmet

[The last Grand Vizier](https://en.wikipedia.org/wiki/Ahmet_Tevfik_Pasha)



Seal of the Great Vizier

Table of  
Contents

Introduction 1

SYSTEM STRUCTURE 3

Automated stopping service 4

Benchmarking service 5

Suggestion service 5

WEB STRUCTURE 7

Backend 7

API 7

Visualization 8

Data visualization 9

Mobile 10

Appendix 11

Papers 11

# Introduction

*Ahmet the last Vizier* would be a web framework for Black-Box Optimization (BBO). In general, a BBO problem consists in optimizing an expansive function with a limited budget of resources. The domain of is usually a high dimensional space where each dimension varies in or a limited subset of them. Furthermore, in some specific problems, could also contain some categorical dimensions, for instance in the optimization of Neural Network hyper-parameters.

The final goal of a BBO algorithm is to find the best parameter configuration within that minimizes .

In general, a BBO algorithm makes minimal assumptions about the problem (otherwise it would not have been called black-box), but an imperative assumption is:

*The system performance must be measured as a function of adjustable parameters.*

This framework follows such assumption and puts (for the time being) no further restrictions on the eligible input problems.

Thus, a valid input for *Ahmet* is a pair of elements:

1. The parameters space
2. The function shaping the real problem.

An example of parameter space in the context of Neural Network is:

# define a search space  
space = {  
 'node': {'type': TYPE.DISCRETE,  
 'values': [18, 32, 64, 48, 56, 128, 256]},  
 'dropout': {'type': TYPE.DISCRETE,  
 'values': [0.25, 0.5, 0.7, 0.9]},  
 'lr': {'type': TYPE.DISCRETE,  
 'values': [0.0001, 0.001, 0.01, 0.007, 0.08]},  
 'batch\_size': {'type': TYPE.DISCRETE,  
 'values': [10, 20, 30, 40]},  
 'epochs': {'type': TYPE.DISCRETE,  
 'values': [4, 8, 15]},  
 'optimizer': {'type': TYPE.CATEGORICAL,  
 'values': ['RMSprop', 'Adam', 'SGD']},  
 'activation': {'type': TYPE.CATEGORICAL,  
 'values': ['tanh', 'softmax', 'relu']}  
}

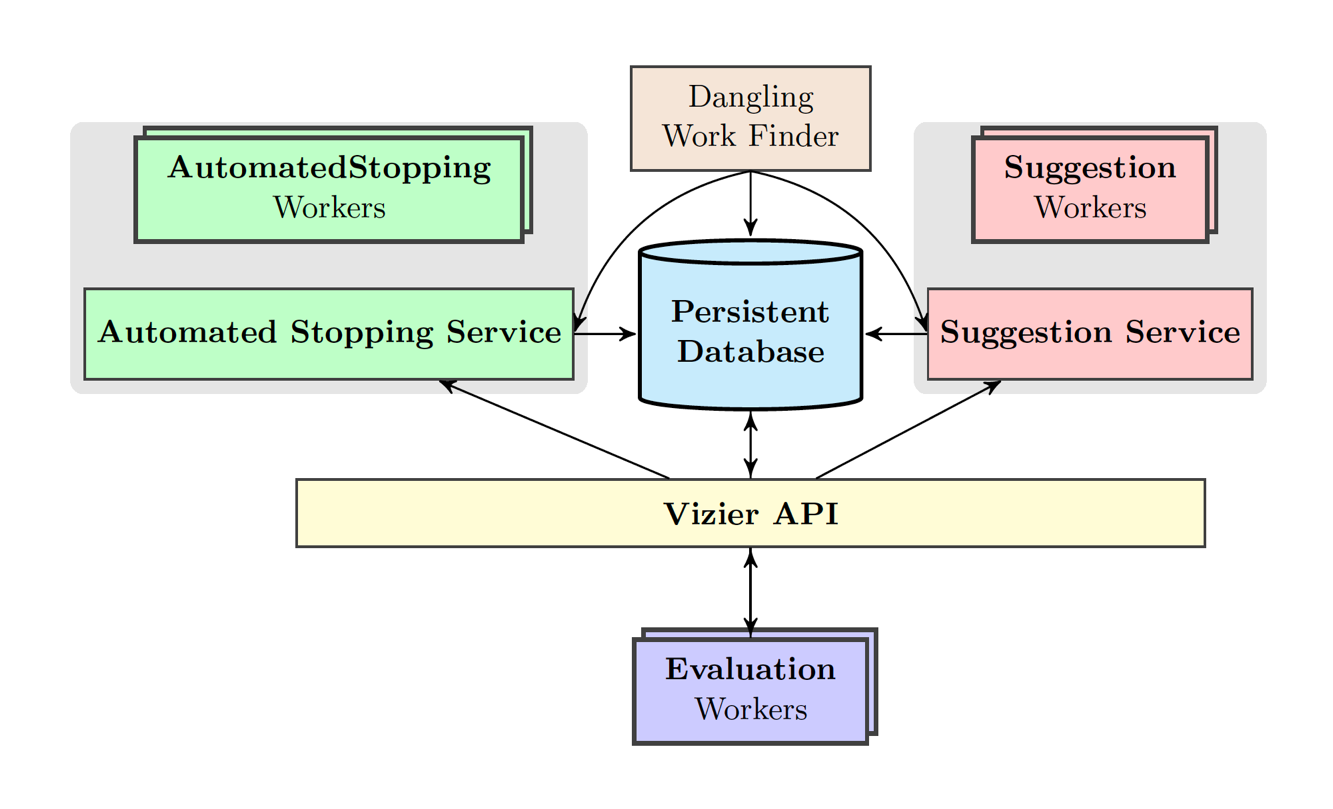
Where each dimension is described by a name and an object that defines the feasible values. With a domain space as the one above, the function should be a callable that accepts a configuration of parameters (in the form of a pair: dimension name, value) and return a value in .

The initial focus will be on machine learning problems, since they are in general costly, but this should not be seen as a sort of restriction.

Definitions:

* **Trial**: a list of parameter values that will be evaluated against .
* **Study**: it is a pair composed of a trial and its result , ().
* **Worker**: a process or a thread responsible of evaluating a trial .
* **Run**: a complete optimization execution on the input function.

# SYSTEM STRUCTURE



Benchmark Service

Figure : Ahmet structure

Ahmet API

Internally, *Ahmet* is composed by 3 different entities:

* Automated Stopping Service (ASS =D): this service monitors the evolution of a study and stops it if it detects poor performance.
* Suggestion Service (SS): core of Ahmet, the SS algorithm finds promising configurations within that will be evaluated in each study.
* Benchmark Service (BS): it is the benchmarking system for testing and evaluating the performance and characteristics of ASS algorithms and SS algorithms.

Other than these 3 entities, *the last Vizier* has a “persistent” storage system that strictly cooperates with the services in two ways:

* It helps the services in keeping all the necessary information about the current study and the overall run. These gives the baseline for the real time data visualization.
* It persistently keeps the characteristics of each ASS and SS algorithm derived from the benchmarking (and maybe also from the previous studies).

Depending on the implemented SS algorithms, the database (DB) could also be used for keeping some ad-hoc data used by the stopping service itself.

The internal structure of the system is exposed via an API that allows both a machine-machine interaction and a client-server interaction.

The last entity appearing in *Figure 1* is the evaluation worker that is the container of the function to be evaluated and its relative domain . This worker must be provided by the user in some standardized way.

## Automated stopping service

The automated stopping service is an early stopping system that periodically checks if the current study on a specific trial should be stopped early. This system could be designed by many different algorithms and some of them may also coexists.

In the first release, we will employ a simple median stopping algorithm. This type of algorithm monitors a running study of a trial and at each step *s* verifies if the value at *s* is lower than the median value of previous studies*.* If the value is lower, the study is stopped, otherwise the study goes on.

Given a trial at step *s* of a study, the study is stopped if:

Where is the set of completed studies. Thus, this type of ASS needs memory and it is effective only after some studies.

## Benchmarking service

Benchmarking service provides a benchmarking suite for different type of BBO algorithms, in particular it should benchmark continuous and discrete algorithms. A common metric must be defined.

Two possible frameworks:

* COCO (COmparing Continuous Optimisers) is a platform for systematic and sound comparisons of real-parameter global optimizers. COCO provides benchmark function testbeds, experimentation templates which are easy to parallelize, and tools for processing and visualizing data generated by one or several optimizers. <https://coco.gforge.inria.fr/COCOdoc/>
* <http://iao.hfuu.edu.cn/bbdob-gecco18> (I have to investigate it better).

## Suggestion service

The suggestion service is the brain of Ahmet. A SS algorithm should produce a trial that will be evaluated against the model and, based on the outcome, it chooses the next trail. There are many algorithms for BBO in literature, but for the first implementation of Ahmet we will use three types of algorithm: *Genetic Algorithm* (GA), *Evolutionary Algorithm* (EA) and Random Search (RS).

A really good framework for GAs is [Deap](https://deap.readthedocs.io/en/master/) and I will adopt it for providing a simple implementation of GA and for testing. While, for EA I will use an algorithm of my own called Scatter Search. Random Search is a popular strategy for optimizing NN model since it is simple and able to produce good trials for completely different problems.

# WEB STRUCTURE

## Backend

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. In this application Django will be responsible of managing the Ahmet’s backend as it has been described in previous section (System structure). The backend is then exposed through a Restful API exploiting again the Django REST framework. With this approach I can decouple the MVC model provided by Django from the UI visualization that will be realized with React and React Native (see next subsections).

Useful links:

* [https://www.django-rest-framework.org](https://www.django-rest-framework.org/)
* [https://www.djangoproject.com](https://www.djangoproject.com/)

## API

Input and Output.

## Visualization

The Dashboard implements the full functionality of the API and it is normally used for:

1. Tracking the progress of a study
2. Interactive visualization
3. Creating and deleting a study

A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generatedThe dashboard will be created with [*React*](https://reactjs.org/) and should look like this

For the time being, the necessary pages are three:

1. Creation of new optimization run. In this page the user creates, deletes and interacts with a run.
2. History of the current run; essentially a visualization of the trials explored in the hyper-parameters space.
3. Benchmarking requests and results visualization of an algorithm.

## Data visualization

Data-Driven Documents (D3) is a JavaScript library for visualizing data using a web standard. D3 combines powerful visualization and interaction techniques with a data-driven approach to DOM manipulation, giving full capabilities of modern browsers and the freedom to design the right visual interface for the data.

The dashboard structure will be realized with React as I said in the previous subsection, while D3 is just used to present data in a dynamic and “fashion” way.

The principal dynamic data presented in the UI are two:

* The current optimization run evolution.
* The explored values of the hyper-parameters space.

In addition to the dynamic data, it might be useful to add a dedicated page that contains the profile of each SS algorithms create with the benchmarking service. This page should be designed during the development of Ahmet since it is not yet clear which are the meaningful KPI (Key Performance Indices) of the algorithms.

Data-Driven Documents documentation can be found [here](https://github.com/d3/d3/wiki).

Some useful links for understanding what kind of charts can be used in Ahmet:

* <http://bl.ocks.org/charlesdguthrie/11356441> this type of plot can be useful for pointing out which are the most used values per hyper-parameter.
* <https://bl.ocks.org/boeric/6a83de20f780b42fadb9> this type of plot can be used for giving the current status of the optimization run.

## Mobile

Once the Desktop web framework has been designed it should be shameless to bring it to mobile devices exploiting [React native](https://facebook.github.io/react-native/).

# Appendix

## Papers

1. @proceedings {46180, title = {Google Vizier: A Service for Black-Box Optimization}, editor = {Daniel Golovin and Benjamin Solnik and Subhodeep Moitra and Greg Kochanski and John Elliot Karro and D. Sculley}, year = {2017}, URL = {http://www.kdd.org/kdd2017/papers/view/google-vizier-a-service-for-black-box-optimization}}