Hands-On Python Guide to Optuna – A New Hyperparameter Optimization Tool

analyticsindiamag.com/hands-on-python-guide-to-optuna-a-new-hyperparameter-optimization-tool

Aishwarya Verma February 1, 2021

Hyperparameter Optimization is getting deeper and deeper as the complexity in deep learning models increases. Many handy tools have been developed to tune the parameters like HyperOpt, SMAC, Spearmint, etc. However, these existing tool kits have some serious issues that can't be neglected as we progress.

- Firstly, all previous hyperparameter optimization frameworks require the user to construct the parameter-search space for each model statically, and it can be a challenging task for large-scale experiments.
- Secondly, most developed tools do not contain a pruning strategy(a process of cutting unpromising trials based on the immediate learning curve).
- Lastly, to handle a variety of models in various situations, the architecture shall be able to handle both small and large scale experiments with minimum setup requirements.

To address the above concerns, The Japanese AI company Preferred Networks, has developed an advanced self-contained hyperparameter optimization framework called Optuna. Optuna is an open-source hyperparameter optimization toolkit designed to deal with machine learning and non-machine learning(as long as we can define the objective function). It provides a very imperative interface to fully support Python language with the highest modularity level in code.

Register for our upcoming Masterclass>>

Features of Optuna

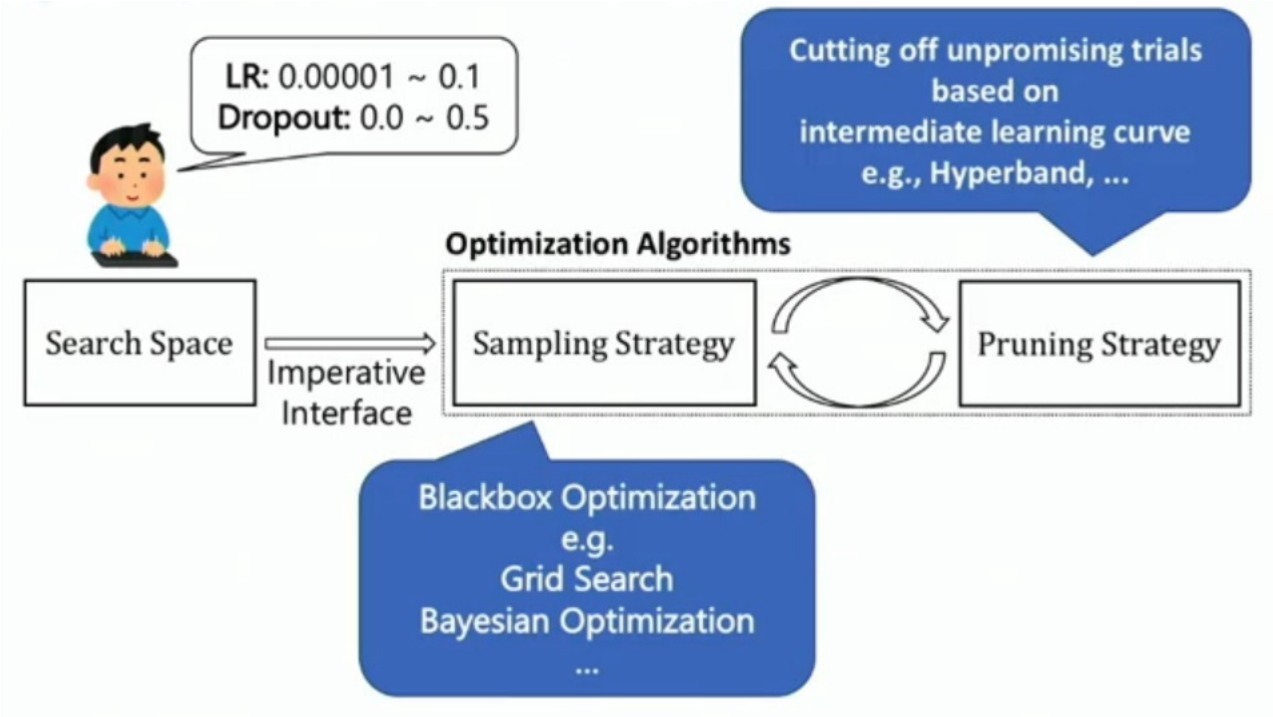
Intuitive : It provides imperative(a.k.a define by run) interface that allow user to dynamically construct the search space

Efficient : It provides many sampling and puning strategies that allow some user customization

Versatile :

- Lightweight : Optuna has minimum software dependency and hence is robust to many workflows and platforms
- Distributed : Optuna provides asynchronous parallelization of trials and almost linear scalability.
- Dashboard : Optuna provides analysis functionality with python code and dashboard also.

An overview of hyperparameter optimization process via Optuna



Source : Official Video Tutorial

Samplers Algorithms available in Optuna

Pruning Algorithms available in Optuna

Installation

Install Optuna toolkit in Python via Pip

```
!pip install optuna
```

Looking for a job change? Let us help you.

Basic Structure of Python code with Optuna

In the undermentioned image, import the *optuna*package, create an objective function with parameter *trial*(specifies the number of trials), write your machine learning model within the function and return the trained model's evaluation. Now, create an optuna study and specify that particular objective function(minimize/maximize). Finally, optimize your created *study* by passing the objective function and number of trials.

```
import optuna

def objective(trial):  # `trial` is an object passed by Optuna.
    some_machine_learning_logic(trial)  # Write your machine learning logic here.
    return evaluation_score  # Return the evaluation score of the trained model.

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=N_TRIALS)  # Specify the number of trials.
```

Optuna Demo – Optimize Machine Learning Model

1. Import all the required packages. The code snippet is available here.

2. Define an objective function with the trial parameter. The objective function should contain the machine learning logic i.e., fit the model on data(iris dataset), predict the test data and return the evaluation score as illustrated below.

```
## In optuna, A Trial represents a single call of the objective function
## Study shows an optimization session which contains a set of trials
## In this demo, "alpha" is the hyperparameter which is need to be optimized
def objective(trial):

    # hyperparameter setting, trial.suggest_uniform will suggest uniform
hyperparameter
    #alpha between the range of 0.0 to 2.0, lowest value of interval is closed and
    #when low=high, it will return low value
    alpha = trial.suggest_uniform('alpha', 0.0, 2.0)

    # data loading and train-test split
    X, y = load_iris(return_X_y=True)
    X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=0)

    # model training and evaluation
    model = sklearn.linear_model.Lasso(alpha=alpha)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    error = sklearn.metrics.mean_squared_error(y_val, y_pred)

    # output: evaluation score
    return error
```

3. Create Optuna study and optimize it. It will show all the trials with the objective function value and parameter values and compare different parameter values to get the optimized one. The code is shown below.

```
# In Optuna, we use the study object to manage optimization.
# Method :func:`~optuna.create_study` returns a study object.
# A study object has useful properties for analyzing the optimization outcome.
study = optuna.create_study(direction='minimize') #Set minimize for minimization
and maximize for maximization.
#To start the optimization, we create a study object and pass the objective
function to method
study.optimize(objective, n_trials=50)
```

4. Get all the optimized values of parameters and print them along with value of the objective function.

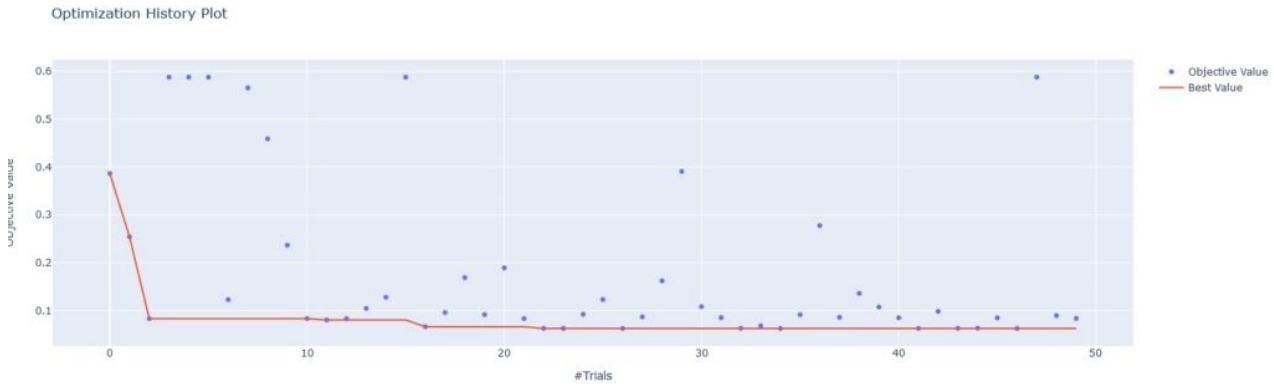
```
# To get the dictionary of parameter name and parameter values:
print("Return a dictionary of parameter name and parameter
values:",study.best_params)

# To get the best observed value of the objective function:
print("Return the best observed value of the objective
function:",study.best_value)

# To get the best trial:
print("Return the best trial:",study.best_trial)

# To get all trials:
print("Return all the trials:", study.trials)
```

5. Visualize the above created hyperparameter optimization study. The code is available here.



You can check the full demo here.

Optuna Demo – Imperative Interface

Suppose, you are confused about which regularization method is better “Lasso” or “Ridge”. In that case, you have to optimize ridge and lasso both and compare them. Creating this specific pipeline from scratch is time-consuming but Optuna can do this task in only 4 lines of code. Optuna can deal with conditional hyperparameters with its imperative (define-by-run) interface.

1. Import all the required libraries and packages. The code snippet is available here.

2. This step is also the same as above(Demo – Optimize Machine Learning Model) but with a slight change. In this objective function we will consider both ridge and lasso regularization methods and compare their values and find the best parameter between them. The code for this shown below. The dataset used is iris dataset.

```
## In optuna, A Trial represents a single call of the objective function
## Study shows an optimization session which contains a set of trials
## In this demo, "alpha" is the hyperparameter which is need to be optimized

##ridge_alpha: the regularization constant of ridge
##lasso_alpha: the regularization constant of lasso

def objective(trial):

    # hyperparameter setting
    #:func:`optuna.trial.Trial.suggest_categorical` for categorical parameters
    #The line below will optimize both ridge and lasso and compares them and find
the best hyperparameter
    #between ridge_alpha and lasso_alpha
    regression_method = trial.suggest_categorical('regression_method', ('ridge',
'lasso'))
    if regression_method == 'ridge':
        ridge_alpha = trial.suggest_uniform('ridge_alpha', 0.0, 2.0)
        model = Ridge(alpha=ridge_alpha)
    else:
        lasso_alpha = trial.suggest_uniform('lasso_alpha', 0.0, 2.0)
        model = Lasso(alpha=lasso_alpha)

    # data loading and train-test split
    X, y = load_iris(return_X_y=True)
    X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=0)

    # model training and evaluation
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    error = mean_squared_error(y_val, y_pred)

    # output: evaluation score
    return error
```

3. Step 3, Step 4 and Step 5 is the same as discussed in Demo – Optimize Machine Learning Model. The code snippet is available here. You can convert the whole trial study into a dataframe with the help of this code below.

```
study.trials_dataframe()
```

The output of this is shown below.

number	value	datetime_start	datetime_complete	duration	params_lasso_alpha	params_regression_method	params_ridge_alpha	state
0	0.063346	2021-01-29 09:42:24.764661	2021-01-29 09:42:24.808117	0 days 00:00:00.043456	NaN	ridge	0.174831	COMPLETE
1	0.064186	2021-01-29 09:42:24.810103	2021-01-29 09:42:24.821541	0 days 00:00:00.011438	NaN	ridge	1.874712	COMPLETE
2	0.063787	2021-01-29 09:42:24.823025	2021-01-29 09:42:24.834573	0 days 00:00:00.011548	NaN	ridge	1.029747	COMPLETE
3	0.063688	2021-01-29 09:42:24.835869	2021-01-29 09:42:24.844383	0 days 00:00:00.008514	NaN	ridge	0.831191	COMPLETE
4	0.534643	2021-01-29 09:42:24.848510	2021-01-29 09:42:24.854059	0 days 00:00:00.005549	1.353413	lasso	NaN	COMPLETE
5	0.063632	2021-01-29 09:42:24.858494	2021-01-29 09:42:24.863955	0 days 00:00:00.005461	NaN	ridge	0.719573	COMPLETE
6	0.083879	2021-01-29 09:42:24.868150	2021-01-29 09:42:24.873318	0 days 00:00:00.005168	0.105169	lasso	NaN	COMPLETE
7	0.064215	2021-01-29 09:42:24.877483	2021-01-29 09:42:24.882659	0 days 00:00:00.005176	NaN	ridge	1.936930	COMPLETE
8	0.183308	2021-01-29 09:42:24.886851	2021-01-29 09:42:24.891927	0 days 00:00:00.005076	0.665054	lasso	NaN	COMPLETE
9	0.063413	2021-01-29 09:42:24.896086	2021-01-29 09:42:24.900694	0 days 00:00:00.004608	NaN	ridge	0.300006	COMPLETE
10	0.063256	2021-01-29 09:42:24.903426	2021-01-29 09:42:24.909598	0 days 00:00:00.006172	NaN	ridge	0.005445	COMPLETE
11	0.063321	2021-01-29 09:42:24.910761	2021-01-29 09:42:24.917603	0 days 00:00:00.006842	NaN	ridge	0.127383	COMPLETE
12	0.063283	2021-01-29 09:42:24.919131	2021-01-29 09:42:24.926487	0 days 00:00:00.007356	NaN	ridge	0.055104	COMPLETE

You can check the whole demo here.

Optuna Demo – Optimization with Pruning

We have already listed out all the sampling and pruning algorithms available in Optuna in above section. Optuna uses TPE as its default sampling algorithm. In this demo, we will see how pruning algorithms can be implemented using Optuna.

1. Import all the required libraries and packages. The code snippet for this is available here.
2. Define an objective function containing Machine Learning logic i.e., load the dataset, split the dataset into train and valid, define the parameter dictionary containing all the parameters(including the parameters to tune) then add a pruning_callback on the trial with respect to the accuracy and then simply fit data, test it and return the evaluation score. The code for this is shown below.

```
# Define the objective function.
def objective(trial):
    #load the dataset
    data, target = load_iris(return_X_y=True)
    #split the dataset in train and test
    train_x, valid_x, train_y, valid_y = train_test_split(data, target,
test_size=0.25)
    dtrain = lgb.Dataset(train_x, label=train_y)
    dvalid = lgb.Dataset(valid_x, label=valid_y)
    #default parameters dictionary for optimizations(search space)
    param = {
        "objective": "binary",
        "metric": "auc",
        "verbosity": -1,
        "boosting_type": "gbdt",
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.4, 1.0),
        "bagging_freq": trial.suggest_int("bagging_freq", 1, 7),
        "min_child_samples": trial.suggest_int("min_child_samples", 5, 100),
    }

    # Add a callback for pruning.
    # To turn on the pruning feature, you need to call
:func:`~optuna.trial.Trial.report` and :func:`~optuna.trial.Trial.should_prune`
after each step of the iterative training.
    # :func:`~optuna.trial.Trial.report` periodically monitors the intermediate
objective values.
    # :func:`~optuna.trial.Trial.should_prune` decides termination of the trial
that does not meet a predefined condition.
    pruning_callback = optuna.integration.LightGBMPruningCallback(trial, "auc")
    gbm = lgb.train(
        param, dtrain, valid_sets=[dvalid], verbose_eval=False, callbacks=
[pruning_callback]
    )
    #Predict the valid data
    preds = gbm.predict(valid_x)
    #rounding values to its nearest integers
    pred_labels = np.rint(preds)
    #calculating accuracy
    accuracy = accuracy_score(valid_y, pred_labels)
    return accuracy
```

3. Now, add a logger to show all the discarded trials and initiate the trial study and then optimize it. It will show all the trials including the rejected ones. The code is as follow

```
# Add stream handler of stdout to show the messages
optuna.logging.get_logger("optuna").addHandler(logging.StreamHandler(sys.stdout))
# Method :func:`~optuna.create_study` returns a study object.
# A study object has useful properties for analyzing the optimization outcome.
study = optuna.create_study(
    direction="maximize",
    sampler=optuna.samplers.TPESampler(),
    pruner=optuna.pruners.MedianPruner(n_warmup_steps=10),
) #Set minimize for minimization and maximize for maximization.
#To start the optimization, we create a study object and pass the objective
function to method
study.optimize(objective, n_trials=100, timeout=600) #add timeout for model not to
exceed this time limit
```

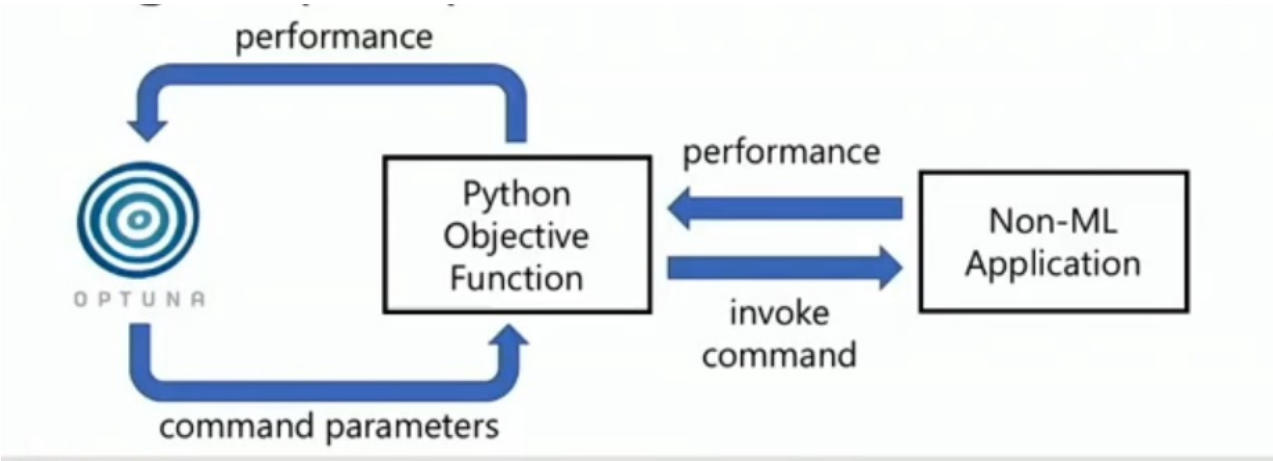
4. Step 4 and Step 5 are the same as discussed in Demo – Optimize Machine Learning Model. The code snippet and all the visualization charts are available here.

You can check the full demo, here.

See Also

Optuna Demo – For Non-ML Task

Optuna is suitable for Non-ML tasks as long as we are able to define an objective function. The image shown below, describes the whole process.



Source: Official Video Tutorial

An example of it(in python) is shown below.

1. Import the optune library.
2. Define an objective function as illustrated in code below.

```
def objective(trial):
    #trial suggesting the values for x between -100 to 100
    x = trial.suggest_uniform('x', -100, 100)
    #return the funciton value
    return (x - 2) ** 2
```

3. Rest of the steps like creating study, optimizing and visualizing it, is the same as we saw earlier.

You can check the full demo, here.

Conclusion

In this session, we have discussed a next-generation Hyperparameter optimization framework **Optuna.** Not only it optimizes the parameters, but also provides visualization plots and dashboard. Four demos of using this library is shown above. Colab notebooks are available at:

You can check other hyperparameter related articles, here.

Resources and Tutorials used above:

What Do You Think?

Join Our Discord Server. Be part of an engaging online community. Join Here.

Subscribe to our Newsletter

Get the latest updates and relevant offers by sharing your email.