Optuna Documentation

Release 3.6.0

Optuna Contributors.

Mar 18, 2024

CONTENTS:

OPTUNA

Optuna-Integration is a package of the integration modules of Optuna. This package allows us to use Optuna, an automatic Hyperparameter optimization software framework, integrated with many useful tools like PyTorch, sklearn, TensorFlow, etc.

CHAPTER

ONE

COMMUNICATION

- GitHub Discussions for questions.
- GitHub Issues for bug reports and feature requests.

CHAPTER

TWO

CONTRIBUTION

Any contributions to Optuna are welcome! When you send a pull request, please follow the contribution guide.

CHAPTER

THREE

LICENSE

MIT License (see LICENSE).

CHAPTER

FOUR

REFERENCE

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD (arXiv).

4.1 Installation

Optuna-Integration supports Python 3.7 or newer.

We recommend to install Optuna-Integration via pip:

```
$ pip install optuna-integration
```

You can also install the development version of Optuna-Integration from main branch of Git repository:

```
$ pip install git+https://github.com/optuna/optuna-integration.git
```

4.2 API Reference for Optuna-Integration

The Optuna-Integration package contains classes used to integrate Optuna with external machine learning frameworks.

All of these classes can be imported in two ways. One is "*from optuna.integration import xxx*" like a module in Optuna, and the other is "*from optuna_integration import xxx*" as an Optuna-Integration specific module. The former is provided for backward compatibility.

For most of the ML frameworks supported by Optuna, the corresponding Optuna integration class serves only to implement a callback object and functions, compliant with the framework's specific callback API, to be called with each intermediate step in the model training. The functionality implemented in these callbacks across the different ML frameworks includes:

(1) Reporting intermediate model scores back to the Optuna trial using optuna.trial.Trial.report,

(2) According to the results of optuna.trial.Trial.should_prune, pruning the current model by raising optuna.TrialPruned, and

(3) Reporting intermediate Optuna data such as the current trial number back to the framework, as done in *MLflowCallback*.

For scikit-learn, an integrated *OptunaSearchCV* estimator is available that combines scikit-learn BaseEstimator functionality with access to a class-level `Study` object.

4.2.1 AllenNLP

optuna_integration.AllenNLPExecutor	AllenNLP extension to use optuna with Jsonnet config file.
optuna_integration.allennlp.dump_best_config	Save JSON config file with environment variables and best performing hyperparameters.
optuna_integration.AllenNLPPruningCallback	AllenNLP callback to prune unpromising trials.

optuna_integration.AllenNLPExecutor

class optuna_integration.**AllenNLPExecutor**(*trial*, *config_file*, *serialization_dir*, *metrics='best_validation_accuracy'*, ***, *include_package=None*, *force=False*, *file_friendly_logging=False*)

AllenNLP extension to use optuna with Jsonnet config file.

See the examples of objective function.

You can also see the tutorial of our AllenNLP integration on AllenNLP Guide.

Note: From Optuna v2.1.0, users have to cast their parameters by using methods in Jsonnet. Call `std.parseInt` for integer, or `std.parseJson` for floating point. Please see the example configuration.

Note: In *AllenNLPExecutor*, you can pass parameters to AllenNLP by either defining a search space using Optuna suggest methods or setting environment variables just like AllenNLP CLI. If a value is set in both a search space in Optuna and the environment variables, the executor will use the value specified in the search space in Optuna.

Parameters

- **trial** (*optuna.Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **config_file** (*str*) – Config file for AllenNLP. Hyperparameters should be masked with `std.extVar`. Please refer to the config example.
- **serialization_dir** (*str*) – A path which model weights and logs are saved.
- **metrics** (*str*) – An evaluation metric. GradientDescrentTrainer.train() of AllenNLP returns a dictionary containing metrics after training. *AllenNLPExecutor* accesses the dictionary by the key `metrics` you specify and use it as a objective value.
- **force** (*bool*) – If `True`, an executor overwrites the output directory if it exists.
- **file_friendly_logging** (*bool*) – If `True`, tqdm status is printed on separate lines and slows tqdm refresh rate.
- **include_package** (*str | list[str] | None*) – Additional packages to include. For more information, please see AllenNLP documentation.

Warning: Deprecated in v3.5.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.5.0.

Methods

run()	Train a model using AllenNLP.

run()

Train a model using AllenNLP.

Return type
float

optuna_integration.allennlp.dump_best_config

optuna_integration.allennlp.**dump_best_config**(*input_config_file*, *output_config_file*, *study*)

Save JSON config file with environment variables and best performing hyperparameters.

Parameters

- **input_config_file** (*str*) – Input Jsonnet config file used with *AllenNLPExecutor*.
- **output_config_file** (*str*) – Output JSON config file.
- **study** (*Study*) – Instance of `Study`. Note that `optimize()` must have been called.

Return type
None

Warning: Deprecated in v3.5.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.5.0.

optuna_integration.AllenNLPPruningCallback

class optuna_integration.**AllenNLPPruningCallback**(*trial=None*, *monitor=None*)

AllenNLP callback to prune unpromising trials.

See the example if you want to add a pruning callback which observes a metric.

You can also see the tutorial of our AllenNLP integration on AllenNLP Guide.

Note: When *AllenNLPPruningCallback* is instantiated in Python script, trial and monitor are mandatory.

On the other hand, when *AllenNLPPruningCallback* is used with *AllenNLPExecutor*, `trial` and `monitor` would be `None`. *AllenNLPExecutor* sets environment variables for a study name, trial id, monitor, and storage. Then *AllenNLPPruningCallback* loads them to restore `trial` and `monitor`.

Note: Currently, build-in pruners are supported except for `PatientPruner`.

Parameters

- **trial** (*Trial | None*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str | None*) – An evaluation metric for pruning, e.g. `validation_loss` or `validation_accuracy`.

Warning: Deprecated in v3.5.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.5.0.

Methods

on_epoch(trainer, metrics, epoch[, is_primary])	Check if a training reaches saturation.
register(*args, **kwargs)	Stub method for *TrainerCallback.register*.

on_epoch(*trainer*, *metrics*, *epoch*, *is_primary=True*, ***_*)

Check if a training reaches saturation.

Parameters

- **trainer** (*GradientDescentTrainer*) – AllenNLP's trainer
- **metrics** (*dict[str, Any]*) – Dictionary of metrics.
- **epoch** (*int*) – Number of current epoch.
- **is_primary** (*bool*) – A flag for AllenNLP internal.
- **_** (*Any*) –

Return type
None

classmethod register(**args*, ***kwargs*)

Stub method for *TrainerCallback.register*.

This method has the same signature as Registrable.register in AllenNLP.

Parameters

- **args** (*Any*) –
- **kwargs** (*Any*) –

Return type
Callable

4.2.2 BoTorch

optuna_integration.BoTorchSampler	A sampler that uses BoTorch, a Bayesian optimization library built on top of PyTorch.
optuna_integration.botorch.ehvi_candidates_func	Expected Hypervolume Improvement (EHVI).
optuna_integration.botorch.logei_candidates_func	Log Expected Improvement (LogEI).
optuna_integration.botorch.qei_candidates_func	Quasi MC-based batch Expected Improvement (qEI).
optuna_integration.botorch.qnei_candidates_func	Quasi MC-based batch Noisy Expected Improvement (qNEI).
optuna_integration.botorch.qehvi_candidates_func	Quasi MC-based batch Expected Hypervolume Improvement (qEHVI).
optuna_integration.botorch.qnehvi_candidates_func	Quasi MC-based batch Noisy Expected Hypervolume Improvement (qNEHVI).
optuna_integration.botorch.qparego_candidates_func	Quasi MC-based extended ParEGO (qParEGO) for constrained multi-objective optimization.

optuna_integration.BoTorchSampler

class optuna_integration.**BoTorchSampler**(*, *candidates_func=None*, *constraints_func=None*, *n_startup_trials=10*, *consider_running_trials=False*, *independent_sampler=None*, *seed=None*, *device=None*)

A sampler that uses BoTorch, a Bayesian optimization library built on top of PyTorch.

This sampler allows using BoTorch's optimization algorithms from Optuna to suggest parameter configurations. Parameters are transformed to continuous space and passed to BoTorch, and then transformed back to Optuna's representations. Categorical parameters are one-hot encoded.

See also:

See an example how to use the sampler.

See also:

See the BoTorch homepage for details and for how to implement your own `candidates_func`.

Note: An instance of this sampler *should not be used with different studies* when used with constraints. Instead, a new instance should be created for each new study. The reason for this is that the sampler is stateful keeping all the computed constraints.

Parameters

- **candidates_func** (*Callable[[torch.Tensor, torch.Tensor, torch.Tensor | None, torch.Tensor, torch.Tensor | None], torch.Tensor] | None*) – An optional function that suggests the next candidates. It must take the training data, the objectives, the constraints, the search space bounds and return the next candidates. The arguments are of type `torch.Tensor`. The return value must be a `torch.Tensor`. However, if `constraints_func` is omitted, constraints will be `None`. For any constraints that failed to compute, the tensor will contain NaN.

If omitted, it is determined automatically based on the number of objectives and whether a constraint is specified. If the number of objectives is one and no constraint is specified, log-Expected Improvement is used. If constraints are specified, quasi MC-based batch Expected Improvement (qEI) is used. If the number of objectives is either two or three, Quasi MC-based batch Expected Hypervolume Improvement (qEHVI) is used. Otherwise, for a larger number of objectives, analytic Expected Hypervolume Improvement is used if no constraints are specified, or the faster Quasi MC-based extended ParEGO (qParEGO) is used if constraints are present.

The function should assume *maximization* of the objective.

See also:

See *optuna_integration.botorch.qei_candidates_func()* for an example.

- **constraints_func** (*Callable[[FrozenTrial], Sequence[float]] | None*) – An optional function that computes the objective constraints. It must take a `FrozenTrial` and return the constraints. The return value must be a sequence of `float` s. A value strictly larger than 0 means that a constraint is violated. A value equal to or smaller than 0 is considered feasible.

  If omitted, no constraints will be passed to `candidates_func` nor taken into account during suggestion.

- **n_startup_trials** (*int*) – Number of initial trials, that is the number of trials to resort to independent sampling.

- **consider_running_trials** (*bool*) – If True, the acquisition function takes into consideration the running parameters whose evaluation has not completed. Enabling this option is considered to improve the performance of parallel optimization.

  Note: Added in v3.2.0 as an experimental argument.

- **independent_sampler** (*BaseSampler | None*) – An independent sampler to use for the initial trials and for parameters that are conditional.

- **seed** (*int | None*) – Seed for random number generator.

- **device** (*torch.device | None*) – A `torch.device` to store input and output data of BoTorch. Please set a CUDA device if you fasten sampling.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Methods

after_trial(study, trial, state, values)	Trial post-processing.
before_trial(study, trial)	Trial pre-processing.
infer_relative_search_space(study, trial)	Infer the search space that will be used by relative sampling in the target trial.
reseed_rng()	Reseed sampler's random number generator.
sample_independent(study, trial, param_name, ...)	Sample a parameter for a given distribution.
sample_relative(study, trial, search_space)	Sample parameters in a given search space.

`after_trial`(*study*, *trial*, *state*, *values*)

Trial post-processing.

This method is called after the objective function returns and right before the trial is finished and its state is stored.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **state** (*TrialState*) – Resulting trial state.
- **values** (*Sequence[float] | None*) – Resulting trial values. Guaranteed to not be `None` if trial succeeded.

Return type
None

`before_trial`(*study*, *trial*)

Trial pre-processing.

This method is called before the objective function is called and right after the trial is instantiated. More precisely, this method is called during trial initialization, just before the `infer_relative_search_space()` call. In other words, it is responsible for pre-processing that should be done before inferring the search space.

Note: Added in v3.3.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.3.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object.

Return type
None

`infer_relative_search_space`(*study*, *trial*)

Infer the search space that will be used by relative sampling in the target trial.

This method is called right before `sample_relative()` method, and the search space returned by this method is passed to it. The parameters not contained in the search space will be sampled by using `sample_independent()` method.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.

Returns
A dictionary containing the parameter names and parameter's distributions.

Return type
Dict[str, *BaseDistribution*]

See also:

Please refer to `intersection_search_space()` as an implementation of `infer_relative_search_space()`.

`reseed_rng()`

Reseed sampler's random number generator.

This method is called by the `Study` instance if trials are executed in parallel with the option `n_jobs>1`. In that case, the sampler instance will be replicated including the state of the random number generator, and they may suggest the same values. To prevent this issue, this method assigns a different seed to each random number generator.

Return type
None

`sample_independent(study, trial, param_name, param_distribution)`

Sample a parameter for a given distribution.

This method is called only for the parameters not contained in the search space returned by `sample_relative()` method. This method is suitable for sampling algorithms that do not use relationship between parameters such as random sampling and TPE.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **param_name** (*str*) – Name of the sampled parameter.
- **param_distribution** (*BaseDistribution*) – Distribution object that specifies a prior and/or scale of the sampling algorithm.

Returns
A parameter value.

Return type
Any

`sample_relative(study, trial, search_space)`

Sample parameters in a given search space.

This method is called once at the beginning of each trial, i.e., right before the evaluation of the objective function. This method is suitable for sampling algorithms that use relationship between parameters such as Gaussian Process and CMA-ES.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **search_space** (*Dict[str, BaseDistribution]*) – The search space returned by `infer_relative_search_space()`.

Returns
A dictionary containing the parameter names and the values.

Return type
Dict[str, *Any*]

optuna_integration.botorch.ehvi_candidates_func

optuna_integration.botorch.**ehvi_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Expected Hypervolume Improvement (EHVI).

The default value of `candidates_func` in *BoTorchSampler* with multi-objective optimization without constraints.

See also:

qei_candidates_func() for argument and return value descriptions.

Note: Added in v3.5.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.5.0.

Parameters

- **train_x** (*torch.Tensor*) –
- **train_obj** (*torch.Tensor*) –
- **train_con** (*torch.Tensor | None*) –
- **bounds** (*torch.Tensor*) –
- **pending_x** (*torch.Tensor | None*) –

Return type
torch.Tensor

optuna_integration.botorch.logei_candidates_func

optuna_integration.botorch.**logei_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Log Expected Improvement (LogEI).

The default value of `candidates_func` in *BoTorchSampler* with single-objective optimization.

Parameters

- **train_x** (*torch.Tensor*) – Previous parameter configurations. A `torch.Tensor` of shape `(n_trials, n_params)`. `n_trials` is the number of already observed trials and `n_params` is the number of parameters. `n_params` may be larger than the actual number of parameters if categorical parameters are included in the search space, since these parameters are one-hot encoded. Values are not normalized.

- **train_obj** (*torch.Tensor*) – Previously observed objectives. A `torch.Tensor` of shape `(n_trials, n_objectives)`. `n_trials` is identical to that of `train_x`. `n_objectives` is the number of objectives. Observations are not normalized.
- **train_con** (*torch.Tensor | None*) – Objective constraints. A `torch.Tensor` of shape `(n_trials, n_constraints)`. `n_trials` is identical to that of `train_x`. `n_constraints` is the number of constraints. A constraint is violated if strictly larger than 0. If no constraints are involved in the optimization, this argument will be `None`.
- **bounds** (*torch.Tensor*) – Search space bounds. A `torch.Tensor` of shape `(2, n_params)`. `n_params` is identical to that of `train_x`. The first and the second rows correspond to the lower and upper bounds for each parameter respectively.
- **pending_x** (*torch.Tensor | None*) – Pending parameter configurations. A `torch.Tensor` of shape `(n_pending, n_params)`. `n_pending` is the number of the trials which are already suggested all their parameters but have not completed their evaluation, and `n_params` is identical to that of `train_x`.

Returns

Next set of candidates. Usually the return value of BoTorch's `optimize_acqf`.

Return type

torch.Tensor

Note: Added in v3.3.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.3.0.

---

optuna_integration.botorch.qei_candidates_func

optuna_integration.botorch.**qei_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Quasi MC-based batch Expected Improvement (qEI).

Parameters

- **train_x** (*torch.Tensor*) – Previous parameter configurations. A `torch.Tensor` of shape `(n_trials, n_params)`. `n_trials` is the number of already observed trials and `n_params` is the number of parameters. `n_params` may be larger than the actual number of parameters if categorical parameters are included in the search space, since these parameters are one-hot encoded. Values are not normalized.
- **train_obj** (*torch.Tensor*) – Previously observed objectives. A `torch.Tensor` of shape `(n_trials, n_objectives)`. `n_trials` is identical to that of `train_x`. `n_objectives` is the number of objectives. Observations are not normalized.
- **train_con** (*torch.Tensor | None*) – Objective constraints. A `torch.Tensor` of shape `(n_trials, n_constraints)`. `n_trials` is identical to that of `train_x`. `n_constraints` is the number of constraints. A constraint is violated if strictly larger than 0. If no constraints are involved in the optimization, this argument will be `None`.
- **bounds** (*torch.Tensor*) – Search space bounds. A `torch.Tensor` of shape `(2, n_params)`. `n_params` is identical to that of `train_x`. The first and the second rows correspond to the lower and upper bounds for each parameter respectively.
- **pending_x** (*torch.Tensor | None*) – Pending parameter configurations. A `torch.Tensor` of shape `(n_pending, n_params)`. `n_pending` is the number of the trials which are already suggested all their parameters but have not completed their evaluation, and `n_params` is identical to that of `train_x`.

Returns
Next set of candidates. Usually the return value of BoTorch's `optimize_acqf`.

Return type
torch.Tensor

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

---

optuna_integration.botorch.qnei_candidates_func

optuna_integration.botorch.**qnei_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Quasi MC-based batch Noisy Expected Improvement (qNEI).

This function may perform better than qEI (*qei_candidates_func*) when the evaluated values of objective function are noisy.

See also:

qei_candidates_func() for argument and return value descriptions.

Note: Added in v3.3.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.3.0.

Parameters

- **train_x** (*torch.Tensor*) –
- **train_obj** (*torch.Tensor*) –
- **train_con** (*torch.Tensor | None*) –
- **bounds** (*torch.Tensor*) –
- **pending_x** (*torch.Tensor | None*) –

Return type
torch.Tensor

optuna_integration.botorch.qehvi_candidates_func

optuna_integration.botorch.**qehvi_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Quasi MC-based batch Expected Hypervolume Improvement (qEHVI).

The default value of `candidates_func` in *BoTorchSampler* with multi-objective optimization when the number of objectives is three or less.

See also:

qei_candidates_func() for argument and return value descriptions.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Parameters

- **train_x** (*torch.Tensor*) –
- **train_obj** (*torch.Tensor*) –
- **train_con** (*torch.Tensor | None*) –
- **bounds** (*torch.Tensor*) –
- **pending_x** (*torch.Tensor | None*) –

Return type
torch.Tensor

optuna_integration.botorch.qnehvi_candidates_func

optuna_integration.botorch.**qnehvi_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Quasi MC-based batch Noisy Expected Hypervolume Improvement (qNEHVI).

According to Botorch/Ax documentation, this function may perform better than qEHVI (*qehvi_candidates_func*). (cf. https://botorch.org/tutorials/constrained_multi_objective_bo)

See also:

qei_candidates_func() for argument and return value descriptions.

Note: Added in v3.1.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.1.0.

Parameters

- **train_x** (*torch.Tensor*) –
- **train_obj** (*torch.Tensor*) –
- **train_con** (*torch.Tensor | None*) –
- **bounds** (*torch.Tensor*) –
- **pending_x** (*torch.Tensor | None*) –

Return type
torch.Tensor

optuna_integration.botorch.qparego_candidates_func

optuna_integration.botorch.**qparego_candidates_func**(*train_x*, *train_obj*, *train_con*, *bounds*, *pending_x*)

Quasi MC-based extended ParEGO (qParEGO) for constrained multi-objective optimization.

The default value of `candidates_func` in *BoTorchSampler* with multi-objective optimization when the number of objectives is larger than three.

See also:

qei_candidates_func() for argument and return value descriptions.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Parameters

- **train_x** (*torch.Tensor*) –
- **train_obj** (*torch.Tensor*) –
- **train_con** (*torch.Tensor | None*) –
- **bounds** (*torch.Tensor*) –
- **pending_x** (*torch.Tensor | None*) –

Return type
torch.Tensor

4.2.3 Catalyst

optuna_integration.CatalystPruningCallback	Catalyst callback to prune unpromising trials.

optuna_integration.CatalystPruningCallback

class optuna_integration.**CatalystPruningCallback**(**args*, ***kwargs*)

Catalyst callback to prune unpromising trials.

This class is an alias to Catalyst's OptunaPruningCallback.

See the Catalyst's documentation for the detailed description.

Warning: Deprecated in v2.7.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v2.7.0.

4.2.4 CatBoost

optuna_integration.CatBoostPruningCallback	Callback for catboost to prune unpromising trials.

optuna_integration.CatBoostPruningCallback

class optuna_integration.**CatBoostPruningCallback**(*trial*, *metric*, *eval_set_index=None*)

Callback for catboost to prune unpromising trials.

See the example if you want to add a pruning callback which observes validation accuracy of a CatBoost model.

Note: `optuna.TrialPruned` cannot be raised in *after_iteration()* that is called in CatBoost via `CatBoostPruningCallback`. You must call *check_pruned()* after training manually unlike other pruning callbacks to raise `optuna.TrialPruned`.

Note: This callback cannot be used with CatBoost on GPUs because CatBoost doesn't support a user-defined callback for GPU. Please refer to CatBoost issue.

Parameters

- **trial** (*optuna.trial.Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **metric** (*str*) – An evaluation metric for pruning, e.g., `Logloss` and `AUC`. Please refer to CatBoost reference for further details.
- **eval_set_index** (*int | None*) – The index of the target validation dataset. If you set only one `eval_set`, `eval_set_index` is None. If you set multiple datasets as `eval_set`, the index of `eval_set` must be `eval_set_index`, e.g., `0` or `1` when `eval_set` contains two datasets.

Note: Added in v3.0.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.0.0.

Methods

after_iteration(info)	Report an evaluation metric value for Optuna pruning after each CatBoost's iteration.
check_pruned()	Raise `optuna.TrialPruned` manually if the CatBoost optimization is pruned.

after_iteration(*info*)

Report an evaluation metric value for Optuna pruning after each CatBoost's iteration.

This method is called by CatBoost.

Parameters

info (*Any*) – A `SimpleNamespace` containing iteraion, `validation_name`, `metric_name` and history of losses. For example `SimpleNamespace(iteration=2, metrics={ 'learn': {'Logloss': [0.6, 0.5]}, 'validation': {'Logloss': [0.7, 0.6], 'AUC': [0.8, 0.9]} })`.

Returns
A boolean value. If `False`, CatBoost internally stops the optimization with Optuna's pruning logic without raising `optuna.TrialPruned`. Otherwise, the optimization continues.

Return type
bool

check_pruned()

Raise `optuna.TrialPruned` manually if the CatBoost optimization is pruned.

Return type
None

4.2.5 Chainer

optuna_integration.ChainerPruningExtension	Chainer extension to prune unpromising trials.
optuna_integration.ChainerMNStudy	A wrapper of `Study` to incorporate Optuna with ChainerMN.

optuna_integration.ChainerPruningExtension

class optuna_integration.**ChainerPruningExtension**(*trial*, *observation_key*, *pruner_trigger*)

Chainer extension to prune unpromising trials.

See the example if you want to add a pruning extension which observes validation accuracy of a Chainer Trainer.

Parameters

- **trial** (*optuna.trial.Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **observation_key** (*str*) – An evaluation metric for pruning, e.g., `main/loss` and `validation/main/accuracy`. Please refer to chainer.Reporter reference for further details.
- **pruner_trigger** (*tuple[int, str] | 'IntervalTrigger' | 'ManualScheduleTrigger'*) – A trigger to execute pruning. `pruner_trigger` is an instance of IntervalTrigger or ManualScheduleTrigger. IntervalTrigger can be specified by a tuple of the interval length and its unit like `(1, 'epoch')`.

Warning: Deprecated in v3.5.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.5.0.

optuna_integration.ChainerMNStudy

class optuna_integration.**ChainerMNStudy**(*study*, *comm*)

A wrapper of Study to incorporate Optuna with ChainerMN.

See also:

ChainerMNStudy provides the same interface as Study. Please refer to optuna.study.Study for further details.

See the example if you want to optimize an objective function that trains neural network written with ChainerMN.

Parameters

- **study** (*Study*) – A Study object.
- **comm** (*CommunicatorBase*) – A ChainerMN communicator.

Warning: Deprecated in v3.5.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.5.0.

Methods

optimize(func[, n_trials, timeout, catch])	Optimize an objective function.

optimize(*func*, *n_trials=None*, *timeout=None*, *catch=()*)

Optimize an objective function.

This method provides the same interface as optuna.study.Study.optimize() except the absence of n_jobs argument.

Parameters

- **func** (*Callable[['ChainerMNTrial', 'CommunicatorBase'], float]*) –
- **n_trials** (*int | None*) –
- **timeout** (*float | None*) –
- **catch** (*tuple[type[Exception], ...]*) –

Return type
None

4.2.6 Dask

optuna_integration.DaskStorage	Dask-compatible storage class.

optuna_integration.DaskStorage

class optuna_integration.**DaskStorage**(*storage=None*, *name=None*, *client=None*, *register=True*)

Dask-compatible storage class.

This storage class wraps a Optuna storage class (e.g. Optuna's in-memory or sqlite storage) and is used to run optimization trials in parallel on a Dask cluster. The underlying Optuna storage object lives on the cluster's scheduler and any method calls on the *DaskStorage* instance results in the same method being called on the underlying Optuna storage object.

See this example or the following YouTube video for how to use *DaskStorage* to extend Optuna's in-memory storage class to run across multiple processes.

Parameters

- **storage** (*None | str | BaseStorage*) – Optuna storage url to use for underlying Optuna storage class to wrap (e.g. None for in-memory storage, sqlite:///example.db for SQLite storage). Defaults to None.
- **name** (*str | None*) – Unique identifier for the Dask storage class. Specifying a custom name can sometimes be useful for logging or debugging. If None is provided, a random name will be automatically generated.
- **client** (*distributed.Client | None*) – Dask Client to connect to. If not provided, will attempt to find an existing Client.
- **register** (*bool*) – Whether or not to register this storage instance with the cluster scheduler. Most common usage of this storage class will not need to specify this argument. Defaults to True.

Note: Added in v3.1.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.1.0.

Methods

check_trial_is_updatable(trial_id, trial_state)	Check whether a trial state is updatable.
create_new_study(directions[, study_name])	Create a new study from a name.
create_new_trial(study_id[, template_trial])	Create and add a new trial to a study.
delete_study(study_id)	Delete a study.
get_all_studies()	Read a list of `FrozenStudy` objects.
get_all_trials(study_id[, deepcopy, states])	Read all trials in a study.
get_base_storage()	Retrieve underlying Optuna storage instance from the scheduler.
get_best_trial(study_id)	Return the trial with the best value in a study.
get_n_trials(study_id[, state])	Count the number of trials in a study.
get_study_directions(study_id)	Read whether a study maximizes or minimizes an objective.
get_study_id_from_name(study_name)	Read the ID of a study.
get_study_name_from_id(study_id)	Read the study name of a study.
get_study_system_attrs(study_id)	Read the optuna-internal attributes of a study.
get_study_user_attrs(study_id)	Read the user-defined attributes of a study.
get_trial(trial_id)	Read a trial.
get_trial_id_from_study_id_trial_number(...	Read the trial ID of a trial.
get_trial_number_from_id(trial_id)	Read the trial number of a trial.
get_trial_param(trial_id, param_name)	Read the parameter of a trial.
get_trial_params(trial_id)	Read the parameter dictionary of a trial.
get_trial_system_attrs(trial_id)	Read the optuna-internal attributes of a trial.
get_trial_user_attrs(trial_id)	Read the user-defined attributes of a trial.
remove_session()	Clean up all connections to a database.
set_study_system_attr(study_id, key, value)	Register an optuna-internal attribute to a study.
set_study_user_attr(study_id, key, value)	Register a user-defined attribute to a study.
set_trial_intermediate_value(trial_id, step, ...)	Report an intermediate value of an objective function.
set_trial_param(trial_id, param_name, ...)	Set a parameter to a trial.
set_trial_state_values(trial_id, state[, values])	Update the state and values of a trial.
set_trial_system_attr(trial_id, key, value)	Set an optuna-internal attribute to a trial.
set_trial_user_attr(trial_id, key, value)	Set a user-defined attribute to a trial.

Attributes

`client`

check_trial_is_updatable(*trial_id*, *trial_state*)

Check whether a trial state is updatable.

Parameters

- **trial_id** (*int*) – ID of the trial. Only used for an error message.
- **trial_state** (*TrialState*) – Trial state to check.

Raises

`RuntimeError` – If the trial is already finished.

Return type
None

`create_new_study`(*directions*, *study_name=None*)

Create a new study from a name.

If no name is specified, the storage class generates a name. The returned study ID is unique among all current and deleted studies.

Parameters

- **directions** (*Sequence[StudyDirection]*) – A sequence of direction whose element is either `MAXIMIZE` or `MINIMIZE`.
- **study_name** (*str | None*) – Name of the new study to create.

Returns
ID of the created study.

Raises
optuna.exceptions.DuplicatedStudyError – If a study with the same `study_name` already exists.

Return type
int

`create_new_trial`(*study_id*, *template_trial=None*)

Create and add a new trial to a study.

The returned trial ID is unique among all current and deleted trials.

Parameters

- **study_id** (*int*) – ID of the study.
- **template_trial** (*FrozenTrial | None*) – Template `FrozenTrial` with default user-attributes, system-attributes, intermediate-values, and a state.

Returns
ID of the created trial.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
int

`delete_study`(*study_id*)

Delete a study.

Parameters
study_id (*int*) – ID of the study.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
None

`get_all_studies`()

Read a list of `FrozenStudy` objects.

Returns
A list of `FrozenStudy` objects, sorted by `study_id`.

Return type
List[*FrozenStudy*]

`get_all_trials`(*study_id*, *deepcopy=True*, *states=None*)

Read all trials in a study.

Parameters

- **study_id** (*int*) – ID of the study.
- **deepcopy** (*bool*) – Whether to copy the list of trials before returning. Set to `True` if you intend to update the list or elements of the list.
- **states** (*Container[TrialState] | None*) – Trial states to filter on. If `None`, include all states.

Returns
List of trials in the study, sorted by `trial_id`.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
List[*FrozenTrial*]

`get_base_storage`()

Retrieve underlying Optuna storage instance from the scheduler.

This is a convenience method to extract the Optuna storage instance stored on the Dask scheduler process to the local Python process.

Return type
BaseStorage

`get_best_trial`(*study_id*)

Return the trial with the best value in a study.

This method is valid only during single-objective optimization.

Parameters
study_id (*int*) – ID of the study.

Returns
The trial with the best objective value among all finished trials in the study.

Raises

- **KeyError** – If no study with the matching `study_id` exists.
- **RuntimeError** – If the study has more than one direction.
- **ValueError** – If no trials have been completed.

Return type
FrozenTrial

`get_n_trials`(*study_id*, *state=None*)

Count the number of trials in a study.

Parameters

- **study_id** (*int*) – ID of the study.
- **state** (*Tuple[TrialState, ...] | TrialState | None*) – Trial states to filter on. If `None`, include all states.

Returns
Number of trials in the study.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
int

`get_study_directions`(*study_id*)

Read whether a study maximizes or minimizes an objective.

Parameters
study_id (*int*) – ID of a study.

Returns
Optimization directions list of the study.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
List[*StudyDirection*]

`get_study_id_from_name`(*study_name*)

Read the ID of a study.

Parameters
study_name (*str*) – Name of the study.

Returns
ID of the study.

Raises
KeyError – If no study with the matching `study_name` exists.

Return type
int

`get_study_name_from_id`(*study_id*)

Read the study name of a study.

Parameters
study_id (*int*) – ID of the study.

Returns
Name of the study.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
str

`get_study_system_attrs`(*study_id*)

Read the optuna-internal attributes of a study.

Parameters
study_id (*int*) – ID of the study.

Returns
Dictionary with the optuna-internal attributes of the study.

Raises
KeyError – If no study with the matching study_id exists.

Return type
Dict[str, *Any*]

get_study_user_attrs(*study_id*)

Read the user-defined attributes of a study.

Parameters
study_id (*int*) – ID of the study.

Returns
Dictionary with the user attributes of the study.

Raises
KeyError – If no study with the matching study_id exists.

Return type
Dict[str, *Any*]

get_trial(*trial_id*)

Read a trial.

Parameters
trial_id (*int*) – ID of the trial.

Returns
Trial with a matching trial ID.

Raises
KeyError – If no trial with the matching trial_id exists.

Return type
FrozenTrial

get_trial_id_from_study_id_trial_number(*study_id*, *trial_number*)

Read the trial ID of a trial.

Parameters

- **study_id** (*int*) – ID of the study.
- **trial_number** (*int*) – Number of the trial.

Returns
ID of the trial.

Raises
KeyError – If no trial with the matching study_id and trial_number exists.

Return type
int

get_trial_number_from_id(*trial_id*)

Read the trial number of a trial.

Note: The trial number is only unique within a study, and is sequential.

Parameters
trial_id (*int*) – ID of the trial.

Returns
Number of the trial.

Raises
KeyError – If no trial with the matching `trial_id` exists.

Return type
int

get_trial_param(*trial_id*, *param_name*)

Read the parameter of a trial.

Parameters

- **trial_id** (*int*) – ID of the trial.
- **param_name** (*str*) – Name of the parameter.

Returns
Internal representation of the parameter.

Raises
KeyError – If no trial with the matching `trial_id` exists. If no such parameter exists.

Return type
float

get_trial_params(*trial_id*)

Read the parameter dictionary of a trial.

Parameters
trial_id (*int*) – ID of the trial.

Returns
Dictionary of a parameters. Keys are parameter names and values are internal representations of the parameter values.

Raises
KeyError – If no trial with the matching `trial_id` exists.

Return type
Dict[str, *Any*]

get_trial_system_attrs(*trial_id*)

Read the optuna-internal attributes of a trial.

Parameters
trial_id (*int*) – ID of the trial.

Returns
Dictionary with the optuna-internal attributes of the trial.

Raises
KeyError – If no trial with the matching `trial_id` exists.

Return type
Dict[str, *Any*]

get_trial_user_attrs(*trial_id*)

Read the user-defined attributes of a trial.

Parameters
trial_id (*int*) – ID of the trial.

Returns
Dictionary with the user-defined attributes of the trial.

Raises
KeyError – If no trial with the matching `trial_id` exists.

Return type
Dict[str, *Any*]

`remove_session()`

Clean up all connections to a database.

Return type
None

`set_study_system_attr(study_id, key, value)`

Register an optuna-internal attribute to a study.

This method overwrites any existing attribute.

Parameters

- **study_id** (*int*) – ID of the study.
- **key** (*str*) – Attribute key.
- **value** (*Any*) – Attribute value. It should be JSON serializable.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
None

`set_study_user_attr(study_id, key, value)`

Register a user-defined attribute to a study.

This method overwrites any existing attribute.

Parameters

- **study_id** (*int*) – ID of the study.
- **key** (*str*) – Attribute key.
- **value** (*Any*) – Attribute value. It should be JSON serializable.

Raises
KeyError – If no study with the matching `study_id` exists.

Return type
None

`set_trial_intermediate_value(trial_id, step, intermediate_value)`

Report an intermediate value of an objective function.

This method overwrites any existing intermediate value associated with the given step.

Parameters

- **trial_id** (*int*) – ID of the trial.
- **step** (*int*) – Step of the trial (e.g., the epoch when training a neural network).
- **intermediate_value** (*float*) – Intermediate value corresponding to the step.

Raises

- **KeyError** – If no trial with the matching `trial_id` exists.
- **RuntimeError** – If the trial is already finished.

Return type
None

set_trial_param(*trial_id*, *param_name*, *param_value_internal*, *distribution*)

Set a parameter to a trial.

Parameters

- **trial_id** (*int*) – ID of the trial.
- **param_name** (*str*) – Name of the parameter.
- **param_value_internal** (*float*) – Internal representation of the parameter value.
- **distribution** (*BaseDistribution*) – Sampled distribution of the parameter.

Raises

- **KeyError** – If no trial with the matching `trial_id` exists.
- **RuntimeError** – If the trial is already finished.

Return type
None

set_trial_state_values(*trial_id*, *state*, *values=None*)

Update the state and values of a trial.

Set return values of an objective function to values argument. If values argument is not `None`, this method overwrites any existing trial values.

Parameters

- **trial_id** (*int*) – ID of the trial.
- **state** (*TrialState*) – New state of the trial.
- **values** (*Sequence[float] | None*) – Values of the objective function.

Returns
`True` if the state is successfully updated. `False` if the state is kept the same. The latter happens when this method tries to update the state of `RUNNING` trial to `RUNNING`.

Raises

- **KeyError** – If no trial with the matching `trial_id` exists.
- **RuntimeError** – If the trial is already finished.

Return type
bool

set_trial_system_attr(*trial_id*, *key*, *value*)

Set an optuna-internal attribute to a trial.

This method overwrites any existing attribute.

Parameters

- **trial_id** (*int*) – ID of the trial.
- **key** (*str*) – Attribute key.

- **value** (*Mapping[str, Mapping[str, JSONSerializable] | Sequence[JSONSerializable] | str | int | float | bool | None] | Sequence[Mapping[str, JSONSerializable] | Sequence[JSONSerializable] | str | int | float | bool | None] | str | int | float | bool | None*) – Attribute value. It should be JSON serializable.

Raises

- **KeyError** – If no trial with the matching `trial_id` exists.
- **RuntimeError** – If the trial is already finished.

Return type
None

set_trial_user_attr(*trial_id*, *key*, *value*)

Set a user-defined attribute to a trial.

This method overwrites any existing attribute.

Parameters

- **trial_id** (*int*) – ID of the trial.
- **key** (*str*) – Attribute key.
- **value** (*Any*) – Attribute value. It should be JSON serializable.

Raises

- **KeyError** – If no trial with the matching `trial_id` exists.
- **RuntimeError** – If the trial is already finished.

Return type
None

4.2.7 fast.ai

optuna_integration.FastAIV1PruningCallback	FastAI callback to prune unpromising trials for fastai.
optuna_integration.FastAIV2PruningCallback	FastAI callback to prune unpromising trials for fastai.
optuna_integration.FastAIPruningCallback	alias of *FastAIV2PruningCallback*

optuna_integration.FastAIV1PruningCallback

class optuna_integration.**FastAIV1PruningCallback**(*learn*, *trial*, *monitor*)

FastAI callback to prune unpromising trials for fastai.

Note: This callback is for fastai<2.0.

See the example if you want to add a pruning callback which monitors validation loss of a `Learner`.

Example

Register a pruning callback to `learn.fit` and `learn.fit_one_cycle`.

```
learn.fit(n_epochs, callbacks=[FastAIPruningCallback(learn, trial, "valid_loss")])
learn.fit_one_cycle(
    n_epochs,
    cyc_len,
    max_lr,
    callbacks=[FastAIPruningCallback(learn, trial, "valid_loss")],
)
```

Parameters

- **learn** (*Learner*) – fastai.basic_train.Learner.
- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str*) – An evaluation metric for pruning, e.g. `valid_loss` and `Accuracy`. Please refer to fastai.callbacks.TrackerCallback reference for further details.

Warning: Deprecated in v2.4.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Methods

on_epoch_end(epoch, **kwargs)

optuna_integration.FastAIV2PruningCallback

class optuna_integration.**FastAIV2PruningCallback**(*trial*, *monitor='valid_loss'*)

FastAI callback to prune unpromising trials for fastai.

Note: This callback is for fastai>=2.0.

See the example if you want to add a pruning callback which monitors validation loss of a `Learner`.

Example

Register a pruning callback to `learn.fit` and `learn.fit_one_cycle`.

```
learn = cnn_learner(dls, resnet18, metrics=[error_rate])
learn.fit(n_epochs, cbs=[FastAIPruningCallback(trial)])  # Monitor "valid_loss"
learn.fit_one_cycle(
    n_epochs,
    lr_max,
    cbs=[FastAIPruningCallback(trial, monitor="error_rate")],  # Monitor "error_rate
↪"
)
```

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str*) – An evaluation metric for pruning, e.g. `valid_loss` or `accuracy`. Please refer to fastai.callback.TrackerCallback reference for further details.

Methods

after_epoch()
after_fit()

optuna_integration.FastAIPruningCallback

optuna_integration.**FastAIPruningCallback**

alias of *FastAIV2PruningCallback*

4.2.8 Keras

optuna_integration.KerasPruningCallback	Keras callback to prune unpromising trials.

optuna_integration.KerasPruningCallback

class optuna_integration.**KerasPruningCallback**(*trial*, *monitor*, *interval=1*)

Keras callback to prune unpromising trials.

See the example if you want to add a pruning callback which observes validation accuracy.

Parameters

- **trial** (*optuna.trial.Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str*) – An evaluation metric for pruning, e.g., `val_loss` and `val_accuracy`. Please refer to keras.Callback reference for further details.

- **interval** (*int*) – Check if trial should be pruned every n-th epoch. By default `interval=1` and pruning is performed after every epoch. Increase `interval` to run several epochs faster before applying pruning.

Methods

on_epoch_end(epoch[, logs])

4.2.9 LightGBM

optuna_integration.LightGBMPruningCallback	Callback for LightGBM to prune unpromising trials.
optuna_integration.lightgbm.train	Wrapper of LightGBM Training API to tune hyperparameters.
optuna_integration.lightgbm.LightGBMTuner	Hyperparameter tuner for LightGBM.
optuna_integration.lightgbm.LightGBMTunerCV	Hyperparameter tuner for LightGBM with cross-validation.

optuna_integration.LightGBMPruningCallback

class optuna_integration.**LightGBMPruningCallback**(*trial*, *metric*, *valid_name='valid_0'*, *report_interval=1*)

Callback for LightGBM to prune unpromising trials.

See the example if you want to add a pruning callback which observes accuracy of a LightGBM model.

Parameters

- **trial** (*optuna.trial.Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **metric** (*str*) – An evaluation metric for pruning, e.g., `binary_error` and `multi_error`. Please refer to LightGBM reference for further details.
- **valid_name** (*str*) – The name of the target validation. Validation names are specified by `valid_names` option of train method. If omitted, `valid_0` is used which is the default name of the first validation. Note that this argument will be ignored if you are calling cv method instead of train method.
- **report_interval** (*int*) – Check if the trial should report intermediate values for pruning every n-th boosting iteration. By default `report_interval=1` and reporting is performed after every iteration. Note that the pruning itself is performed according to the interval definition of the pruner.

optuna_integration.lightgbm.train

optuna_integration.lightgbm.**train**(*params*, *train_set*, *num_boost_round=1000*, *valid_sets=None*, *valid_names=None*, *feval=None*, *feature_name='auto'*, *categorical_feature='auto'*, *keep_training_booster=False*, *callbacks=None*, *time_budget=None*, *sample_size=None*, *study=None*, *optuna_callbacks=None*, *model_dir=None*, *verbosity=None*, *show_progress_bar=True*, ***, *optuna_seed=None*)

Wrapper of LightGBM Training API to tune hyperparameters.

It optimizes the following hyperparameters in a stepwise manner: `lambda_l1`, `lambda_l2`, `num_leaves`, `feature_fraction`, `bagging_fraction`, `bagging_freq` and `min_child_samples`. It is a drop-in replacement for lightgbm.train(). See a simple example of LightGBM Tuner which optimizes the validation log loss of cancer detection.

train() is a wrapper function of *LightGBMTuner*. To use feature in Optuna such as suspended/resumed optimization and/or parallelization, refer to *LightGBMTuner* instead of this function.

Note: Arguments and keyword arguments for lightgbm.train() can be passed. For `params`, please check the official documentation for LightGBM.

Parameters

- **time_budget** (*int | None*) – A time budget for parameter tuning in seconds.
- **study** (*Study | None*) – A `Study` instance to store optimization results. The `Trial` instances in it has the following user attributes: `elapsed_secs` is the elapsed time since the optimization starts. `average_iteration_time` is the average time of iteration to train the booster model in the trial. `lgbm_params` is a JSON-serialized dictionary of LightGBM parameters used in the trial.
- **optuna_callbacks** (*list[Callable[[Study, FrozenTrial], None]] | None*) – List of Optuna callback functions that are invoked at the end of each trial. Each function must accept two parameters with the following types in this order: `Study` and `FrozenTrial`. Please note that this is not a `callbacks` argument of lightgbm.train() .
- **model_dir** (*str | None*) – A directory to save boosters. By default, it is set to `None` and no boosters are saved. Please set shared directory (e.g., directories on NFS) if you want to access *get_best_booster()* in distributed environments. Otherwise, it may raise `ValueError`. If the directory does not exist, it will be created. The filenames of the boosters will be `{model_dir}/{trial_number}.pkl` (e.g., `./boosters/0.pkl`).
- **verbosity** (*int | None*) – A verbosity level to change Optuna's logging level. The level is aligned to LightGBM's verbosity .

  Warning: Deprecated in v2.0.0. `verbosity` argument will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change.

  Please use `set_verbosity()` instead.

- **show_progress_bar** (*bool*) – Flag to show progress bars or not. To disable progress bar, set this `False`.

Note: Progress bars will be fragmented by logging messages of LightGBM and Optuna. Please suppress such messages to show the progress bars properly.

- **optuna_seed** (*int | None*) – `seed` of `TPESampler` for random number generator that affects sampling for `num_leaves`, `bagging_fraction`, `bagging_freq`, `lambda_l1`, and `lambda_l2`.

Note: The deterministic parameter of LightGBM makes training reproducible. Please enable it when you use this argument.

- **params** (*dict[str, Any]*) –
- **train_set** (*lgb.Dataset*) –
- **num_boost_round** (*int*) –
- **valid_sets** (*list['lgb.Dataset'] | tuple['lgb.Dataset', ...] | 'lgb.Dataset' | None*) –
- **valid_names** (*Any | None*) –
- **feval** (*Callable[..., Any] | None*) –
- **feature_name** (*str*) –
- **categorical_feature** (*str*) –
- **keep_training_booster** (*bool*) –
- **callbacks** (*list[Callable[..., Any]] | None*) –
- **sample_size** (*int | None*) –

Return type
lgb.Booster

optuna_integration.lightgbm.LightGBMTuner

class optuna_integration.lightgbm.**LightGBMTuner**(*params*, *train_set*, *num_boost_round=1000*, *valid_sets=None*, *valid_names=None*, *feval=None*, *feature_name='auto'*, *categorical_feature='auto'*, *keep_training_booster=False*, *callbacks=None*, *time_budget=None*, *sample_size=None*, *study=None*, *optuna_callbacks=None*, *model_dir=None*, *verbosity=None*, *show_progress_bar=True*, ***, *optuna_seed=None*)

Hyperparameter tuner for LightGBM.

It optimizes the following hyperparameters in a stepwise manner: `lambda_l1`, `lambda_l2`, `num_leaves`, `feature_fraction`, `bagging_fraction`, `bagging_freq` and `min_child_samples`.

You can find the details of the algorithm and benchmark results in this blog article by Kohei Ozaki, a Kaggle Grandmaster.

Note: Arguments and keyword arguments for lightgbm.train() can be passed. For `params`, please check the official documentation for LightGBM.

The arguments that only *LightGBMTuner* has are listed below:

Parameters

- **time_budget** (*int | None*) – A time budget for parameter tuning in seconds.
- **study** (*optuna.study.Study | None*) – A `Study` instance to store optimization results. The `Trial` instances in it has the following user attributes: `elapsed_secs` is the elapsed time since the optimization starts. `average_iteration_time` is the average time of iteration to train the booster model in the trial. `lgbm_params` is a JSON-serialized dictionary of LightGBM parameters used in the trial.
- **optuna_callbacks** (*list[Callable[[Study, FrozenTrial], None]] | None*) – List of Optuna callback functions that are invoked at the end of each trial. Each function must accept two parameters with the following types in this order: `Study` and `FrozenTrial`. Please note that this is not a `callbacks` argument of lightgbm.train() .
- **model_dir** (*str | None*) – A directory to save boosters. By default, it is set to `None` and no boosters are saved. Please set shared directory (e.g., directories on NFS) if you want to access *get_best_booster()* in distributed environments. Otherwise, it may raise `ValueError`. If the directory does not exist, it will be created. The filenames of the boosters will be `{model_dir}/{trial_number}.pkl` (e.g., `./boosters/0.pkl`).
- **verbosity** (*int | None*) – A verbosity level to change Optuna's logging level. The level is aligned to LightGBM's verbosity .

  Warning: Deprecated in v2.0.0. `verbosity` argument will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change.

  Please use `set_verbosity()` instead.

- **show_progress_bar** (*bool*) – Flag to show progress bars or not. To disable progress bar, set this `False`.

  Note: Progress bars will be fragmented by logging messages of LightGBM and Optuna. Please suppress such messages to show the progress bars properly.

- **optuna_seed** (*int | None*) – `seed` of `TPESampler` for random number generator that affects sampling for `num_leaves`, `bagging_fraction`, `bagging_freq`, `lambda_l1`, and `lambda_l2`.

  Note: The deterministic parameter of LightGBM makes training reproducible. Please enable it when you use this argument.

- **params** (*dict[str, Any]*) –
- **train_set** (*lgb.Dataset*) –
- **num_boost_round** (*int*) –

- **valid_sets** (*list['lgb.Dataset'] | tuple['lgb.Dataset', ...] | 'lgb.Dataset' | None*) –
- **valid_names** (*Any | None*) –
- **feval** (*Callable[..., Any] | None*) –
- **feature_name** (*str*) –
- **categorical_feature** (*str*) –
- **keep_training_booster** (*bool*) –
- **callbacks** (*list[Callable[..., Any]] | None*) –
- **sample_size** (*int | None*) –

Methods

compare_validation_metrics(val_score, best_score)	
get_best_booster()	Return the best booster.
higher_is_better()	
run()	Perform the hyperparameter-tuning with given parameters.
sample_train_set()	Make subset of *self.train_set* Dataset object.
tune_bagging([n_trials])	
tune_feature_fraction([n_trials])	
tune_feature_fraction_stage2([n_trials])	
tune_min_data_in_leaf()	
tune_num_leaves([n_trials])	
tune_regularization_factors([n_trials])	

Attributes

best_params	Return parameters of the best booster.
best_score	Return the score of the best booster.

property best_params: dict[str, Any]

Return parameters of the best booster.

property best_score: float

Return the score of the best booster.

get_best_booster()

Return the best booster.

If the best booster cannot be found, ValueError will be raised. To prevent the errors, please save boosters by specifying the model_dir argument of __init__(), when you resume tuning or you run tuning in parallel.

Return type
lgb.Booster

run()

Perform the hyperparameter-tuning with given parameters.

Return type
None

sample_train_set()

Make subset of *self.train_set* Dataset object.

Return type
None

optuna_integration.lightgbm.LightGBMTunerCV

class optuna_integration.lightgbm.**LightGBMTunerCV**(*params*, *train_set*, *num_boost_round=1000*, *folds=None*, *nfold=5*, *stratified=True*, *shuffle=True*, *feval=None*, *feature_name='auto'*, *categorical_feature='auto'*, *fpreproc=None*, *seed=0*, *callbacks=None*, *time_budget=None*, *sample_size=None*, *study=None*, *optuna_callbacks=None*, *verbosity=None*, *show_progress_bar=True*, *model_dir=None*, *return_cvbooster=False*, ***, *optuna_seed=None*)

Hyperparameter tuner for LightGBM with cross-validation.

It employs the same stepwise approach as *LightGBMTuner*. *LightGBMTunerCV* invokes lightgbm.cv() to train and validate boosters while *LightGBMTuner* invokes lightgbm.train(). See a simple example which optimizes the validation log loss of cancer detection.

Note: Arguments and keyword arguments for lightgbm.cv() can be passed except metrics, init_model and eval_train_metric. For params, please check the official documentation for LightGBM.

The arguments that only *LightGBMTunerCV* has are listed below:

Parameters

- **time_budget** (*int | None*) – A time budget for parameter tuning in seconds.
- **study** (*optuna.study.Study | None*) – A Study instance to store optimization results. The Trial instances in it has the following user attributes: elapsed_secs is the elapsed time since the optimization starts. average_iteration_time is the average time of iteration to train the booster model in the trial. lgbm_params is a JSON-serialized dictionary of LightGBM parameters used in the trial.
- **optuna_callbacks** (*list[Callable[[Study, FrozenTrial], None]] | None*) – List of Optuna callback functions that are invoked at the end of each trial. Each function

must accept two parameters with the following types in this order: `Study` and `FrozenTrial`. Please note that this is not a `callbacks` argument of lightgbm.train() .

- **model_dir** (*str | None*) – A directory to save boosters. By default, it is set to `None` and no boosters are saved. Please set shared directory (e.g., directories on NFS) if you want to access *get_best_booster()* in distributed environments. Otherwise, it may raise `ValueError`. If the directory does not exist, it will be created. The filenames of the boosters will be `{model_dir}/{trial_number}.pkl` (e.g., `./boosters/0.pkl`).
- **verbosity** (*int | None*) – A verbosity level to change Optuna's logging level. The level is aligned to LightGBM's verbosity .

  Warning: Deprecated in v2.0.0. `verbosity` argument will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change.

  Please use `set_verbosity()` instead.

- **show_progress_bar** (*bool*) – Flag to show progress bars or not. To disable progress bar, set this `False`.

  Note: Progress bars will be fragmented by logging messages of LightGBM and Optuna. Please suppress such messages to show the progress bars properly.

- **return_cvbooster** (*bool*) – Flag to enable *get_best_booster()*.
- **optuna_seed** (*int | None*) – `seed` of `TPESampler` for random number generator that affects sampling for `num_leaves`, `bagging_fraction`, `bagging_freq`, `lambda_l1`, and `lambda_l2`.

  Note: The deterministic parameter of LightGBM makes training reproducible. Please enable it when you use this argument.

- **params** (*dict[str, Any]*) –
- **train_set** (*lgb.Dataset*) –
- **num_boost_round** (*int*) –
- **folds** (*Generator[tuple[int, int], None, None] | Iterator[tuple[int, int]] | 'BaseCrossValidator' | None*) –
- **nfold** (*int*) –
- **stratified** (*bool*) –
- **shuffle** (*bool*) –
- **feval** (*Callable[..., Any] | None*) –
- **feature_name** (*str*) –
- **categorical_feature** (*str*) –
- **fpreproc** (*Callable[..., Any] | None*) –
- **seed** (*int*) –
- **callbacks** (*list[Callable[..., Any]] | None*) –

- **sample_size** (*int | None*) –

Methods

compare_validation_metrics(val_score, best_score)	
get_best_booster()	Return the best cvbooster.
higher_is_better()	
run()	Perform the hyperparameter-tuning with given parameters.
sample_train_set()	Make subset of *self.train_set* Dataset object.
tune_bagging([n_trials])	
tune_feature_fraction([n_trials])	
tune_feature_fraction_stage2([n_trials])	
tune_min_data_in_leaf()	
tune_num_leaves([n_trials])	
tune_regularization_factors([n_trials])	

Attributes

best_params	Return parameters of the best booster.
best_score	Return the score of the best booster.

property best_params: dict[str, Any]

Return parameters of the best booster.

property best_score: float

Return the score of the best booster.

get_best_booster()

Return the best cvbooster.

If the best booster cannot be found, ValueError will be raised. To prevent the errors, please save boosters by specifying both of the model_dir and the return_cvbooster arguments of __init__(), when you resume tuning or you run tuning in parallel.

Return type
lgb.CVBooster

run()

Perform the hyperparameter-tuning with given parameters.

Return type
None

sample_train_set()

Make subset of *self.train_set* Dataset object.

Return type
None

4.2.10 MLflow

optuna_integration.MLflowCallback	Callback to track Optuna trials with MLflow.

optuna_integration.MLflowCallback

class optuna_integration.**MLflowCallback**(*tracking_uri=None*, *metric_name='value'*, *create_experiment=True*, *mlflow_kwargs=None*, *tag_study_user_attrs=False*, *tag_trial_user_attrs=True*)

Callback to track Optuna trials with MLflow.

This callback adds relevant information that is tracked by Optuna to MLflow.

Example

Add MLflow callback to Optuna optimization.

```
import optuna
from optuna_integration.mlflow import MLflowCallback


def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    return (x - 2) ** 2


mlflc = MLflowCallback(
    tracking_uri=YOUR_TRACKING_URI,
    metric_name="my metric score",
)

study = optuna.create_study(study_name="my_study")
study.optimize(objective, n_trials=10, callbacks=[mlflc])
```

Parameters

- **tracking_uri** (*str | None*) – The URI of the MLflow tracking server.

  Please refer to mlflow.set_tracking_uri for more details.

- **metric_name** (*str | Sequence[str]*) – Name assigned to optimized metric. In case of multi-objective optimization, list of names can be passed. Those names will be assigned to metrics in the order returned by objective function. If single name is provided, or this argument is left to default value, it will be broadcasted to each objective with a number suffix in order returned by objective function e.g. two objectives and default metric name

will be logged as `value_0` and `value_1`. The number of metrics must be the same as the number of values an objective function returns.

- **create_experiment** (*bool*) – When `True`, new MLflow experiment will be created for each optimization run, named after the Optuna study. Setting this argument to `False` lets user run optimization under existing experiment, set via mlflow.set_experiment, by passing `experiment_id` as one of `mlflow_kwargs` or under default MLflow experiment, when no additional arguments are passed. Note that this argument must be set to `False` when using Optuna with this callback within Databricks Notebook.
- **mlflow_kwargs** (*dict[str, Any] | None*) – Set of arguments passed when initializing MLflow run. Please refer to MLflow API documentation for more details.

  Note: `nest_trials` argument added in v2.3.0 is a part of `mlflow_kwargs` since v3.0.0. Anyone using `nest_trials=True` should migrate to `mlflow_kwargs={"nested": True}` to avoid raising `TypeError`.

- **tag_study_user_attrs** (*bool*) – Flag indicating whether or not to add the study's user attrs to the mlflow trial as tags. Please note that when this flag is set, key value pairs in `user_attrs` will supersede existing tags.
- **tag_trial_user_attrs** (*bool*) – Flag indicating whether or not to add the trial's user attrs to the mlflow trial as tags. Please note that when both trial and study user attributes are logged, the latter will supersede the former in case of a collision.

Note: Added in v1.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v1.4.0.

Methods

track_in_mlflow()	Decorator for using MLflow logging in the objective function.

track_in_mlflow()

Decorator for using MLflow logging in the objective function.

This decorator enables the extension of MLflow logging provided by the callback.

All information logged in the decorated objective function will be added to the MLflow run for the trial created by the callback.

Example

Add additional logging to MLflow.

```
import optuna
import mlflow
from optuna_integration.mlflow import MLflowCallback

mlflc = MLflowCallback(
    tracking_uri=YOUR_TRACKING_URI,
    metric_name="my metric score",
)


@mlflc.track_in_mlflow()
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    mlflow.log_param("power", 2)
    mlflow.log_metric("base of metric", x - 2)

    return (x - 2) ** 2


study = optuna.create_study(study_name="my_other_study")
study.optimize(objective, n_trials=10, callbacks=[mlflc])
```

Returns
Objective function with tracking to MLflow enabled.

Return type
Callable

Note: Added in v2.9.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.9.0.

4.2.11 MXNet

optuna_integration.MXNetPruningCallback	MXNet callback to prune unpromising trials.

optuna_integration.MXNetPruningCallback

class optuna_integration.**MXNetPruningCallback**(*trial*, *eval_metric*)

MXNet callback to prune unpromising trials.

See the example if you want to add a pruning callback which observes accuracy.

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.

- **eval_metric** (*str*) – An evaluation metric name for pruning, e.g., `cross-entropy` and `accuracy`. If using default metrics like mxnet.metrics.Accuracy, use it's default metric name. For custom metrics, use the metric_name provided to constructor. Please refer to mxnet.metrics reference for further details.

4.2.12 pycma

optuna_integration.CmaEsSampler	Wrapper class of PyCmaSampler for backward compatibility.
optuna_integration.PyCmaSampler	A Sampler using cma library as the backend.

optuna_integration.CmaEsSampler

class optuna_integration.**CmaEsSampler**(*x0=None*, *sigma0=None*, *cma_stds=None*, *seed=None*, *cma_opts=None*, *n_startup_trials=1*, *independent_sampler=None*, *warn_independent_sampling=True*)

Wrapper class of PyCmaSampler for backward compatibility.

Warning: Deprecated in v2.0.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v2.0.0.

This class is renamed to *PyCmaSampler*.

Methods

after_trial(study, trial, state, values)	Trial post-processing.
before_trial(study, trial)	Trial pre-processing.
infer_relative_search_space(study, trial)	Infer the search space that will be used by relative sampling in the target trial.
reseed_rng()	Reseed sampler's random number generator.
sample_independent(study, trial, param_name, ...)	Sample a parameter for a given distribution.
sample_relative(study, trial, search_space)	Sample parameters in a given search space.

Parameters

- **x0** (*Dict[str, Any] | None*) –
- **sigma0** (*float | None*) –
- **cma_stds** (*Dict[str, float] | None*) –
- **seed** (*int | None*) –
- **cma_opts** (*Dict[str, Any] | None*) –
- **n_startup_trials** (*int*) –
- **independent_sampler** (*BaseSampler | None*) –

- **warn_independent_sampling** (*bool*) –

after_trial(*study*, *trial*, *state*, *values*)

Trial post-processing.

This method is called after the objective function returns and right before the trial is finished and its state is stored.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **state** (*TrialState*) – Resulting trial state.
- **values** (*Sequence[float] | None*) – Resulting trial values. Guaranteed to not be `None` if trial succeeded.

Return type
None

before_trial(*study*, *trial*)

Trial pre-processing.

This method is called before the objective function is called and right after the trial is instantiated. More precisely, this method is called during trial initialization, just before the `infer_relative_search_space()` call. In other words, it is responsible for pre-processing that should be done before inferring the search space.

Note: Added in v3.3.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.3.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object.

Return type
None

infer_relative_search_space(*study*, *trial*)

Infer the search space that will be used by relative sampling in the target trial.

This method is called right before `sample_relative()` method, and the search space returned by this method is passed to it. The parameters not contained in the search space will be sampled by using `sample_independent()` method.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.

Returns
A dictionary containing the parameter names and parameter's distributions.

Return type
Dict[str, *BaseDistribution*]

See also:

Please refer to `intersection_search_space()` as an implementation of `infer_relative_search_space()`.

`reseed_rng()`

Reseed sampler's random number generator.

This method is called by the `Study` instance if trials are executed in parallel with the option `n_jobs>1`. In that case, the sampler instance will be replicated including the state of the random number generator, and they may suggest the same values. To prevent this issue, this method assigns a different seed to each random number generator.

Return type
None

`sample_independent`(*study*, *trial*, *param_name*, *param_distribution*)

Sample a parameter for a given distribution.

This method is called only for the parameters not contained in the search space returned by `sample_relative()` method. This method is suitable for sampling algorithms that do not use relationship between parameters such as random sampling and TPE.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **param_name** (*str*) – Name of the sampled parameter.
- **param_distribution** (*BaseDistribution*) – Distribution object that specifies a prior and/or scale of the sampling algorithm.

Returns
A parameter value.

Return type
float

`sample_relative`(*study*, *trial*, *search_space*)

Sample parameters in a given search space.

This method is called once at the beginning of each trial, i.e., right before the evaluation of the objective function. This method is suitable for sampling algorithms that use relationship between parameters such as Gaussian Process and CMA-ES.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **search_space** (*Dict[str, BaseDistribution]*) – The search space returned by `infer_relative_search_space()`.

Returns
A dictionary containing the parameter names and the values.

Return type
Dict[str, float]

optuna_integration.PyCmaSampler

class optuna_integration.**PyCmaSampler**(*x0=None*, *sigma0=None*, *cma_stds=None*, *seed=None*, *cma_opts=None*, *n_startup_trials=1*, *independent_sampler=None*, *warn_independent_sampling=True*)

A Sampler using cma library as the backend.

Example

Optimize a simple quadratic function by using *PyCmaSampler*.

```
import optuna
import optuna_integration


def objective(trial):
    x = trial.suggest_float("x", -1, 1)
    y = trial.suggest_int("y", -1, 1)
    return x**2 + y


sampler = optuna_integration.PyCmaSampler()
study = optuna.create_study(sampler=sampler)
study.optimize(objective, n_trials=20)
```

Note that parallel execution of trials may affect the optimization performance of CMA-ES, especially if the number of trials running in parallel exceeds the population size.

Note: *CmaEsSampler* is deprecated and renamed to *PyCmaSampler* in v2.0.0. Please use *PyCmaSampler* instead of *CmaEsSampler*.

Parameters

- **x0** (*Dict[str, Any] | None*) – A dictionary of an initial parameter values for CMA-ES. By default, the mean of `low` and `high` for each distribution is used. Please refer to cma.CMAEvolutionStrategy for further details of `x0`.
- **sigma0** (*float | None*) – Initial standard deviation of CMA-ES. By default, `sigma0` is set to `min_range / 6`, where `min_range` denotes the minimum range of the distributions in

the search space. If distribution is categorical, `min_range` is `len(choices) - 1`. Please refer to cma.CMAEvolutionStrategy for further details of `sigma0`.

- **cma_stds** (*Dict[str, float] | None*) – A dictionary of multipliers of sigma0 for each parameters. The default value is 1.0. Please refer to cma.CMAEvolutionStrategy for further details of `cma_stds`.
- **seed** (*int | None*) – A random seed for CMA-ES.
- **cma_opts** (*Dict[str, Any] | None*) – Options passed to the constructor of cma.CMAEvolutionStrategy class.

  Note that default option is cma_default_options, but `BoundaryHandler`, `bounds`, `CMA_stds` and `seed` arguments in `cma_opts` will be ignored because it is added by *PyCmaSampler* automatically.
- **n_startup_trials** (*int*) – The independent sampling is used instead of the CMA-ES algorithm until the given number of trials finish in the same study.
- **independent_sampler** (*BaseSampler | None*) – A `BaseSampler` instance that is used for independent sampling. The parameters not contained in the relative search space are sampled by this sampler. The search space for *PyCmaSampler* is determined by `intersection_search_space()`.

  If `None` is specified, `RandomSampler` is used as the default.

  See also:

  `optuna.samplers` module provides built-in independent samplers such as `RandomSampler` and `TPESampler`.
- **warn_independent_sampling** (*bool*) – If this is `True`, a warning message is emitted when the value of a parameter is sampled by using an independent sampler.

  Note that the parameters of the first trial in a study are always sampled via an independent sampler, so no warning messages are emitted in this case.

Methods

after_trial(study, trial, state, values)	Trial post-processing.
before_trial(study, trial)	Trial pre-processing.
infer_relative_search_space(study, trial)	Infer the search space that will be used by relative sampling in the target trial.
reseed_rng()	Reseed sampler's random number generator.
sample_independent(study, trial, param_name, ...)	Sample a parameter for a given distribution.
sample_relative(study, trial, search_space)	Sample parameters in a given search space.

after_trial(*study*, *trial*, *state*, *values*)

Trial post-processing.

This method is called after the objective function returns and right before the trial is finished and its state is stored.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **state** (*TrialState*) – Resulting trial state.
- **values** (*Sequence[float] | None*) – Resulting trial values. Guaranteed to not be None if trial succeeded.

Return type
None

before_trial(*study*, *trial*)

Trial pre-processing.

This method is called before the objective function is called and right after the trial is instantiated. More precisely, this method is called during trial initialization, just before the `infer_relative_search_space()` call. In other words, it is responsible for pre-processing that should be done before inferring the search space.

Note: Added in v3.3.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.3.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object.

Return type
None

infer_relative_search_space(*study*, *trial*)

Infer the search space that will be used by relative sampling in the target trial.

This method is called right before `sample_relative()` method, and the search space returned by this method is passed to it. The parameters not contained in the search space will be sampled by using `sample_independent()` method.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.

Returns
A dictionary containing the parameter names and parameter's distributions.

Return type
Dict[str, *BaseDistribution*]

See also:

Please refer to `intersection_search_space()` as an implementation of `infer_relative_search_space()`.

reseed_rng()

Reseed sampler's random number generator.

This method is called by the Study instance if trials are executed in parallel with the option n_jobs>1. In that case, the sampler instance will be replicated including the state of the random number generator, and they may suggest the same values. To prevent this issue, this method assigns a different seed to each random number generator.

Return type
None

sample_independent(*study*, *trial*, *param_name*, *param_distribution*)

Sample a parameter for a given distribution.

This method is called only for the parameters not contained in the search space returned by sample_relative() method. This method is suitable for sampling algorithms that do not use relationship between parameters such as random sampling and TPE.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **param_name** (*str*) – Name of the sampled parameter.
- **param_distribution** (*BaseDistribution*) – Distribution object that specifies a prior and/or scale of the sampling algorithm.

Returns
A parameter value.

Return type
float

sample_relative(*study*, *trial*, *search_space*)

Sample parameters in a given search space.

This method is called once at the beginning of each trial, i.e., right before the evaluation of the objective function. This method is suitable for sampling algorithms that use relationship between parameters such as Gaussian Process and CMA-ES.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **search_space** (*Dict[str, BaseDistribution]*) – The search space returned by infer_relative_search_space().

Returns
A dictionary containing the parameter names and the values.

Return type
Dict[str, float]

4.2.13 PyTorch

optuna_integration.PyTorchIgnitePruningHandler	PyTorch Ignite handler to prune unpromising trials.
optuna_integration.PyTorchLightningPruningCallback	PyTorch Lightning callback to prune unpromising trials.
optuna_integration.TorchDistributedTrial	A wrapper of `Trial` to incorporate Optuna with PyTorch distributed.

optuna_integration.PyTorchIgnitePruningHandler

class optuna_integration.**PyTorchIgnitePruningHandler**(*trial*, *metric*, *trainer*)

PyTorch Ignite handler to prune unpromising trials.

See the example if you want to add a pruning handler which observes validation accuracy.

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **metric** (*str*) – A name of metric for pruning, e.g., `accuracy` and `loss`.
- **trainer** (*Engine*) – A trainer engine of PyTorch Ignite. Please refer to ignite.engine.Engine reference for further details.

optuna_integration.PyTorchLightningPruningCallback

class optuna_integration.**PyTorchLightningPruningCallback**(*trial*, *monitor*)

PyTorch Lightning callback to prune unpromising trials.

See the example if you want to add a pruning callback which observes accuracy.

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str*) – An evaluation metric for pruning, e.g., `val_loss` or `val_acc`. The metrics are obtained from the returned dictionaries from e.g. `lightning.pytorch.LightningModule.training_step` or `lightning.pytorch.LightningModule.validation_epoch_end` and the names thus depend on how this dictionary is formatted.

Note: For the distributed data parallel training, the version of PyTorchLightning needs to be higher than or equal to v1.6.0. In addition, `Study` should be instantiated with RDB storage.

Note: If you would like to use PyTorchLightningPruningCallback in a distributed training environment, you need to evoke *PyTorchLightningPruningCallback.check_pruned()* manually so that `TrialPruned` is properly handled.

Methods

check_pruned()	Raise `optuna.TrialPruned` manually if pruned.
`on_fit_start`(trainer, pl_module)	
`on_validation_end`(trainer, pl_module)	

check_pruned()

Raise `optuna.TrialPruned` manually if pruned.

Currently, `intermediate_values` are not properly propagated between processes due to storage cache. Therefore, necessary information is kept in trial_system_attrs when the trial runs in a distributed situation. Please call this method right after calling `lightning.pytorch.Trainer.fit()`. If a callback doesn't have any backend storage for DDP, this method does nothing.

Return type
None

optuna_integration.TorchDistributedTrial

class optuna_integration.**TorchDistributedTrial**(*trial*, *group=None*)

A wrapper of `Trial` to incorporate Optuna with PyTorch distributed.

See also:

TorchDistributedTrial provides the same interface as `Trial`. Please refer to `optuna.trial.Trial` for further details.

See the example if you want to optimize an objective function that trains neural network written with PyTorch distributed data parallel.

Parameters

- **trial** (*optuna.trial.BaseTrial | None*) – A `Trial` object or `None`. Please set trial object in rank-0 node and set `None` in the other rank node.
- **group** (*'ProcessGroup' | None*) – A *torch.distributed.ProcessGroup* to communicate with the other nodes. TorchDistributedTrial use CPU tensors to communicate, make sure the group supports CPU tensors communications.

  Use *gloo* backend when group is None. Create a global *gloo* backend when group is None and WORLD is nccl.

Note: The methods of *TorchDistributedTrial* are expected to be called by all workers at once. They invoke synchronous data transmission to share processing results and synchronize timing.

Note: Added in v2.6.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.6.0.

Methods

report(value, step)
set_system_attr(key, value)
set_user_attr(key, value)
should_prune()
suggest_categorical()
suggest_discrete_uniform(name, low, high, q)
suggest_float(name, low, high, *[, step, log])
suggest_int(name, low, high[, step, log])
suggest_loguniform(name, low, high)
suggest_uniform(name, low, high)

Attributes

datetime_start
distributions
number
params
system_attrs
user_attrs

set_system_attr(*key*, *value*)

Warning: Deprecated in v3.1.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.1.0.

Parameters

- **key** (*str*) –
- **value** (*Any*) –

Return type
None

suggest_discrete_uniform(*name*, *low*, *high*, *q*)

> **Warning:** Deprecated in v3.0.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v6.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.0.0.
>
> Use suggest_float(..., step=...) instead.

Parameters

- **name** (*str*) –
- **low** (*float*) –
- **high** (*float*) –
- **q** (*float*) –

Return type
float

suggest_loguniform(*name*, *low*, *high*)

> **Warning:** Deprecated in v3.0.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v6.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.0.0.
>
> Use suggest_float(..., log=True) instead.

Parameters

- **name** (*str*) –
- **low** (*float*) –
- **high** (*float*) –

Return type
float

suggest_uniform(*name*, *low*, *high*)

> **Warning:** Deprecated in v3.0.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v6.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.0.0.
>
> Use suggest_float instead.

Parameters

- **name** (*str*) –
- **low** (*float*) –
- **high** (*float*) –

Return type
float

property system_attrs: dict[str, Any]

Warning: Deprecated in v3.1.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v5.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.1.0.

4.2.14 scikit-optimize

optuna_integration.SkoptSampler	Sampler using Scikit-Optimize as the backend.

optuna_integration.SkoptSampler

class optuna_integration.**SkoptSampler**(*independent_sampler=None*, *warn_independent_sampling=True*, *skopt_kwargs=None*, *n_startup_trials=1*, ***, *consider_pruned_trials=False*, *seed=None*)

Sampler using Scikit-Optimize as the backend.

The use of *SkoptSampler* is highly not recommended, as the development of Scikit-Optimize has been inactive and we have identified compatibility issues with newer NumPy versions.

Parameters

- **independent_sampler** (*BaseSampler | None*) – A `BaseSampler` instance that is used for independent sampling. The parameters not contained in the relative search space are sampled by this sampler. The search space for *SkoptSampler* is determined by `intersection_search_space()`.

  If `None` is specified, `RandomSampler` is used as the default.

  See also:

  `optuna.samplers` module provides built-in independent samplers such as `RandomSampler` and `TPESampler`.

- **warn_independent_sampling** (*bool*) – If this is `True`, a warning message is emitted when the value of a parameter is sampled by using an independent sampler.

  Note that the parameters of the first trial in a study are always sampled via an independent sampler, so no warning messages are emitted in this case.

- **skopt_kwargs** (*dict[str, Any] | None*) – Keyword arguments passed to the constructor of skopt.Optimizer class.

  Note that `dimensions` argument in `skopt_kwargs` will be ignored because it is added by *SkoptSampler* automatically.

- **n_startup_trials** (*int*) – The independent sampling is used until the given number of trials finish in the same study.
- **consider_pruned_trials** (*bool*) – If this is `True`, the PRUNED trials are considered for sampling.

Note: Added in v2.0.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.0.0.

Note: As the number of trials n increases, each sampling takes longer and longer on a scale of $O(n^3)$. And, if this is `True`, the number of trials will increase. So, it is suggested to set this flag `False` when each evaluation of the objective function is relatively faster than each sampling. On the other hand, it is suggested to set this flag `True` when each evaluation of the objective function is relatively slower than each sampling.

- **seed** (*int | None*) – Seed for random number generator.

Warning: Deprecated in v3.4.0. This feature will be removed in the future. The removal of this feature is currently scheduled for v4.0.0, but this schedule is subject to change. See https://github.com/optuna/optuna/releases/tag/v3.4.0.

Methods

after_trial(study, trial, state, values)	Trial post-processing.
before_trial(study, trial)	Trial pre-processing.
infer_relative_search_space(study, trial)	Infer the search space that will be used by relative sampling in the target trial.
reseed_rng()	Reseed sampler's random number generator.
sample_independent(study, trial, param_name, ...)	Sample a parameter for a given distribution.
sample_relative(study, trial, search_space)	Sample parameters in a given search space.

after_trial(*study*, *trial*, *state*, *values*)

Trial post-processing.

This method is called after the objective function returns and right before the trial is finished and its state is stored.

Note: Added in v2.4.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.4.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **state** (*TrialState*) – Resulting trial state.

- **values** (*Sequence[float] | None*) – Resulting trial values. Guaranteed to not be None if trial succeeded.

Return type
None

before_trial(*study*, *trial*)

Trial pre-processing.

This method is called before the objective function is called and right after the trial is instantiated. More precisely, this method is called during trial initialization, just before the `infer_relative_search_space()` call. In other words, it is responsible for pre-processing that should be done before inferring the search space.

Note: Added in v3.3.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.3.0.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object.

Return type
None

infer_relative_search_space(*study*, *trial*)

Infer the search space that will be used by relative sampling in the target trial.

This method is called right before `sample_relative()` method, and the search space returned by this method is passed to it. The parameters not contained in the search space will be sampled by using `sample_independent()` method.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.

Returns
A dictionary containing the parameter names and parameter's distributions.

Return type
dict[str, *BaseDistribution*]

See also:

Please refer to `intersection_search_space()` as an implementation of `infer_relative_search_space()`.

reseed_rng()

Reseed sampler's random number generator.

This method is called by the `Study` instance if trials are executed in parallel with the option `n_jobs>1`. In that case, the sampler instance will be replicated including the state of the random number generator, and they may suggest the same values. To prevent this issue, this method assigns a different seed to each random number generator.

Return type
None

sample_independent(*study*, *trial*, *param_name*, *param_distribution*)

Sample a parameter for a given distribution.

This method is called only for the parameters not contained in the search space returned by `sample_relative()` method. This method is suitable for sampling algorithms that do not use relationship between parameters such as random sampling and TPE.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **param_name** (*str*) – Name of the sampled parameter.
- **param_distribution** (*BaseDistribution*) – Distribution object that specifies a prior and/or scale of the sampling algorithm.

Returns

A parameter value.

Return type

Any

sample_relative(*study*, *trial*, *search_space*)

Sample parameters in a given search space.

This method is called once at the beginning of each trial, i.e., right before the evaluation of the objective function. This method is suitable for sampling algorithms that use relationship between parameters such as Gaussian Process and CMA-ES.

Note: The failed trials are ignored by any build-in samplers when they sample new parameters. Thus, failed trials are regarded as deleted in the samplers' perspective.

Parameters

- **study** (*Study*) – Target study object.
- **trial** (*FrozenTrial*) – Target trial object. Take a copy before modifying this object.
- **search_space** (*dict[str, BaseDistribution]*) – The search space returned by `infer_relative_search_space()`.

Returns

A dictionary containing the parameter names and the values.

Return type

dict[str, *Any*]

4.2.15 SHAP

optuna_integration.ShapleyImportanceEvaluator	Shapley (SHAP) parameter importance evaluator.

optuna_integration.ShapleyImportanceEvaluator

class optuna_integration.**ShapleyImportanceEvaluator**(*, *n_trees=64*, *max_depth=64*, *seed=None*)

Shapley (SHAP) parameter importance evaluator.

This evaluator fits a random forest regression model that predicts the objective values of `COMPLETE` trials given their parameter configurations. Feature importances are then computed as the mean absolute SHAP values.

Note: This evaluator requires the sklearn Python package and SHAP. The model for the SHAP calculation is based on sklearn.ensemble.RandomForestClassifier.

Parameters

- **n_trees** (*int*) – Number of trees in the random forest.
- **max_depth** (*int*) – The maximum depth of each tree in the random forest.
- **seed** (*int | None*) – Seed for the random forest.

Note: Added in v3.0.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.0.0.

Methods

evaluate(study[, params, target])	Evaluate parameter importances based on completed trials in the given study.

evaluate(*study*, *params=None*, *, *target=None*)

Evaluate parameter importances based on completed trials in the given study.

Note: This method is not meant to be called by library users.

See also:

Please refer to `get_param_importances()` for how a concrete evaluator should implement this method.

Parameters

- **study** (*Study*) – An optimized study.
- **params** (*list[str] | None*) – A list of names of parameters to assess. If `None`, all parameters that are present in all of the completed trials are assessed.

- **target** (*Callable[[FrozenTrial], float] | None*) – A function to specify the value to evaluate importances. If it is `None` and `study` is being used for single-objective optimization, the objective values are used. Can also be used for other trial attributes, such as the duration, like `target=lambda t: t.duration.total_seconds()`.

  Note: Specify this argument if `study` is being used for multi-objective optimization. For example, to get the hyperparameter importance of the first objective, use `target=lambda t: t.values[0]` for the target parameter.

Returns
A `dict` where the keys are parameter names and the values are assessed importances.

Return type
dict[str, float]

4.2.16 sklearn

optuna_integration.OptunaSearchCV	Hyperparameter search with cross-validation.

optuna_integration.OptunaSearchCV

class optuna_integration.**OptunaSearchCV**(*estimator*, *param_distributions*, ***, *cv=None*, *enable_pruning=False*, *error_score=nan*, *max_iter=1000*, *n_jobs=None*, *n_trials=10*, *random_state=None*, *refit=True*, *return_train_score=False*, *scoring=None*, *study=None*, *subsample=1.0*, *timeout=None*, *verbose=0*, *callbacks=None*)

Hyperparameter search with cross-validation.

Parameters

- **estimator** (*sklearn.base.BaseEstimator*) – Object to use to fit the data. This is assumed to implement the scikit-learn estimator interface. Either this needs to provide `score`, or `scoring` must be passed.
- **param_distributions** (*Mapping[str, distributions.BaseDistribution]*) – Dictionary where keys are parameters and values are distributions. Distributions are assumed to implement the optuna distribution interface.
- **cv** (*int | 'BaseCrossValidator' | Iterable | None*) – Cross-validation strategy. Possible inputs for cv are:
  - `None`, to use the default 5-fold cross validation,
  - integer to specify the number of folds in a CV splitter,
  - CV splitter,
  - an iterable yielding (train, validation) splits as arrays of indices.

  For integer, if `estimator` is a classifier and `y` is either binary or multiclass, `sklearn.model_selection.StratifiedKFold` is used. otherwise, `sklearn.model_selection.KFold` is used.
- **enable_pruning** (*bool*) – If `True`, pruning is performed in the case where the underlying estimator supports `partial_fit`.

- **error_score** (*Number | float | str*) – Value to assign to the score if an error occurs in fitting. If 'raise', the error is raised. If numeric, `sklearn.exceptions.FitFailedWarning` is raised. This does not affect the refit step, which will always raise the error.
- **max_iter** (*int*) – Maximum number of epochs. This is only used if the underlying estimator supports `partial_fit`.
- **n_jobs** (*int | None*) – Number of `threading` based parallel jobs. `None` means 1. `-1` means using the number is set to CPU count.

  Note: `n_jobs` allows parallelization using `threading` and may suffer from Python's GIL. It is recommended to use process-based optimization if `func` is CPU bound.

- **n_trials** (*int | None*) – Number of trials. If `None`, there is no limitation on the number of trials. If `timeout` is also set to `None`, the study continues to create trials until it receives a termination signal such as Ctrl+C or SIGTERM. This trades off runtime vs quality of the solution.
- **random_state** (*int | np.random.RandomState | None*) – Seed of the pseudo random number generator. If int, this is the seed used by the random number generator. If `numpy.random.RandomState` object, this is the random number generator. If `None`, the global random state from `numpy.random` is used.
- **refit** (*bool*) – If `True`, refit the estimator with the best found hyperparameters. The refitted estimator is made available at the `best_estimator_` attribute and permits using `predict` directly.
- **return_train_score** (*bool*) – If `True`, training scores will be included. Computing training scores is used to get insights on how different hyperparameter settings impact the overfitting/underfitting trade-off. However computing training scores can be computationally expensive and is not strictly required to select the hyperparameters that yield the best generalization performance.
- **scoring** (*Callable[..., float] | str | None*) – String or callable to evaluate the predictions on the validation data. If `None`, `score` on the estimator is used.
- **study** (*study_module.Study | None*) – Study corresponds to the optimization task. If `None`, a new study is created.
- **subsample** (*float | int*) – Proportion of samples that are used during hyperparameter search.
  - If int, then draw `subsample` samples.
  - If float, then draw `subsample` * `X.shape[0]` samples.
- **timeout** (*float | None*) – Time limit in seconds for the search of appropriate models. If `None`, the study is executed without time limitation. If `n_trials` is also set to `None`, the study continues to create trials until it receives a termination signal such as Ctrl+C or SIGTERM. This trades off runtime vs quality of the solution.
- **verbose** (*int*) – Verbosity level. The higher, the more messages.
- **callbacks** (*list[Callable[[study_module.Study, FrozenTrial], None]] | None*) – List of callback functions that are invoked at the end of each trial. Each function must accept two parameters with the following types in this order: `Study` and `FrozenTrial`.

See also:

See the tutorial of Callback for Study.optimize for how to use and implement callback functions.

best_estimator_

Estimator that was chosen by the search. This is present only if `refit` is set to `True`.

n_splits_

Number of cross-validation splits.

refit_time_

Time for refitting the best estimator. This is present only if `refit` is set to `True`.

sample_indices_

Indices of samples that are used during hyperparameter search.

scorer_

Scorer function.

study_

Actual study.

Examples

```
import optuna
import optuna_integration

from sklearn.datasets import load_iris
from sklearn.svm import SVC

clf = SVC(gamma="auto")
param_distributions = {
    "C": optuna.distributions.FloatDistribution(1e-10, 1e10, log=True)
}
optuna_search = optuna_integration.OptunaSearchCV(clf, param_distributions)
X, y = load_iris(return_X_y=True)
optuna_search.fit(X, y)
y_pred = optuna_search.predict(X)
```

Note: By following the scikit-learn convention for scorers, the direction of optimization is `maximize`. See https://scikit-learn.org/stable/modules/model_evaluation.html. For the minimization problem, please multiply `-1`.

Note: Added in v0.17.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v0.17.0.

Methods

fit(X[, y, groups])	Run fit with all sets of parameters.
get_metadata_routing()	Get metadata routing of this object.
get_params([deep])	Get parameters for this estimator.
score(X[, y])	Return the score on the given data.
set_fit_request(*[, groups])	Request metadata passed to the `fit` method.
set_params(**params)	Set the parameters of this estimator.

Attributes

*best_index_*	Trial number which corresponds to the best candidate parameter setting.
*best_params_*	Parameters of the best trial in the `Study`.
*best_score_*	Mean cross-validated score of the best estimator.
*best_trial_*	Best trial in the `Study`.
*classes_*	Class labels.
*cv_results_*	A dictionary mapping a metric name to a list of Cross-Validation results of all trials.
decision_function	Call `decision_function` on the best estimator.
inverse_transform	Call `inverse_transform` on the best estimator.
*n_trials_*	Actual number of trials.
predict	Call `predict` on the best estimator.
predict_log_proba	Call `predict_log_proba` on the best estimator.
predict_proba	Call `predict_proba` on the best estimator.
score_samples	Call `score_samples` on the best estimator.
set_user_attr	Call `set_user_attr` on the `Study`.
transform	Call `transform` on the best estimator.
*trials_*	All trials in the `Study`.
trials_dataframe	Call `trials_dataframe` on the `Study`.
*user_attrs_*	User attributes in the `Study`.

property best_index_: `int`

Trial number which corresponds to the best candidate parameter setting.

Returned value is equivalent to `optuna_search.best_trial_.number`.

property best_params_: `dict[str, Any]`

Parameters of the best trial in the `Study`.

property best_score_: `float`

Mean cross-validated score of the best estimator.

property best_trial_: `FrozenTrial`

Best trial in the `Study`.

property classes_: `List[float] | ndarray | Series`

Class labels.

property cv_results_: `dict[str, Any]`

A dictionary mapping a metric name to a list of Cross-Validation results of all trials.

property decision_function: Callable[[...], List[float] | ndarray | Series | List[List[float]] | DataFrame | spmatrix]

Call decision_function on the best estimator.

This is available only if the underlying estimator supports decision_function and refit is set to True.

fit(*X*, *y=None*, *groups=None*, ***fit_params*)

Run fit with all sets of parameters.

Parameters

- **X** (*List[List[float]] | ndarray | DataFrame | spmatrix*) – Training data.
- **y** (*List[float] | ndarray | Series | List[List[float]] | DataFrame | spmatrix | None*) – Target variable.
- **groups** (*List[float] | ndarray | Series | None*) – Group labels for the samples used while splitting the dataset into train/validation set.
- ****fit_params** (*Any*) – Parameters passed to fit on the estimator.

Returns
self.

Return type
OptunaSearchCV

get_metadata_routing()

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns
routing – A MetadataRequest encapsulating routing information.

Return type
MetadataRequest

get_params(*deep=True*)

Get parameters for this estimator.

Parameters
deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
dict

property inverse_transform: Callable[[...], List[List[float]] | ndarray | DataFrame | spmatrix]

Call inverse_transform on the best estimator.

This is available only if the underlying estimator supports inverse_transform and refit is set to True.

property n_trials_: int

Actual number of trials.

property predict: Callable[[...], List[float] | ndarray | Series | List[List[float]] | DataFrame | spmatrix]

Call `predict` on the best estimator.

This is available only if the underlying estimator supports `predict` and `refit` is set to `True`.

property predict_log_proba: Callable[[...], List[List[float]] | ndarray | DataFrame | spmatrix]

Call `predict_log_proba` on the best estimator.

This is available only if the underlying estimator supports `predict_log_proba` and `refit` is set to `True`.

property predict_proba: Callable[[...], List[List[float]] | ndarray | DataFrame | spmatrix]

Call `predict_proba` on the best estimator.

This is available only if the underlying estimator supports `predict_proba` and `refit` is set to `True`.

score(*X*, *y=None*)

Return the score on the given data.

Parameters

- **X** (*List[List[float]] | ndarray | DataFrame | spmatrix*) – Data.
- **y** (*List[float] | ndarray | Series | List[List[float]] | DataFrame | spmatrix | None*) – Target variable.

Returns
Scaler score.

Return type
float

property score_samples: Callable[[...], List[float] | ndarray | Series]

Call `score_samples` on the best estimator.

This is available only if the underlying estimator supports `score_samples` and `refit` is set to `True`.

set_fit_request(*, *groups='$UNCHANGED$'*)

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters

- **groups** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `groups` parameter in `fit`.
- **self** (`OptunaSearchCV`) –

Returns
self – The updated object.

Return type
object

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters
****params** (*dict*) – Estimator parameters.

Returns
self – Estimator instance.

Return type
estimator instance

property set_user_attr: Callable[[...], None]

Call `set_user_attr` on the `Study`.

property transform: Callable[[...], List[List[float]] | ndarray | DataFrame | spmatrix]

Call `transform` on the best estimator.

This is available only if the underlying estimator supports `transform` and `refit` is set to `True`.

property trials_: list[FrozenTrial]

All trials in the `Study`.

property trials_dataframe: Callable[[...], DataFrame]

Call `trials_dataframe` on the `Study`.

property user_attrs_: dict[str, Any]

User attributes in the `Study`.

4.2.17 skorch

optuna_integration.SkorchPruningCallback	Skorch callback to prune unpromising trials.

optuna_integration.SkorchPruningCallback

class optuna_integration.**SkorchPruningCallback**(*trial*, *monitor*)

Skorch callback to prune unpromising trials.

New in version 2.1.0.

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str*) – An evaluation metric for pruning, e.g. `val_loss` or `val_acc`. The metrics are obtained from the returned dictionaries, i.e., `net.histroy`. The names thus depend on how this dictionary is formatted.

Methods

on_epoch_end(net, **kwargs)

4.2.18 TensorBoard

optuna_integration.TensorBoardCallback	Callback to track Optuna trials with TensorBoard.

optuna_integration.TensorBoardCallback

class optuna_integration.**TensorBoardCallback**(*dirname*, *metric_name*)

Callback to track Optuna trials with TensorBoard.

This callback adds relevant information that is tracked by Optuna to TensorBoard.

See the example.

Parameters

- **dirname** (*str*) – Directory to store TensorBoard logs.
- **metric_name** (*str*) – Name of the metric. Since the metric itself is just a number, *metric_name* can be used to give it a name. So you know later if it was roc-auc or accuracy.

Note: Added in v2.0.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.0.0.

4.2.19 TensorFlow

optuna_integration.TFKerasPruningCallback	tf.keras callback to prune unpromising trials.

optuna_integration.TFKerasPruningCallback

class optuna_integration.**TFKerasPruningCallback**(*trial*, *monitor*)

tf.keras callback to prune unpromising trials.

This callback is intend to be compatible for TensorFlow v1 and v2, but only tested with TensorFlow v2.

See the example if you want to add a pruning callback which observes the validation accuracy.

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **monitor** (*str*) – An evaluation metric for pruning, e.g., `val_loss` or `val_acc`.

Methods

on_epoch_end(epoch[, logs])	

4.2.20 Weights & Biases

optuna_integration.WeightsAndBiasesCallback	Callback to track Optuna trials with Weights & Biases.

optuna_integration.WeightsAndBiasesCallback

class optuna_integration.**WeightsAndBiasesCallback**(*metric_name='value'*, *wandb_kwargs=None*, *as_multirun=False*)

Callback to track Optuna trials with Weights & Biases.

This callback enables tracking of Optuna study in Weights & Biases. The study is tracked as a single experiment run, where all suggested hyperparameters and optimized metrics are logged and plotted as a function of optimizer steps.

Note: User needs to be logged in to Weights & Biases before using this callback in online mode. For more information, please refer to wandb setup.

Note: Users who want to run multiple Optuna studies within the same process should call `wandb.finish()` between subsequent calls to `study.optimize()`. Calling `wandb.finish()` is not necessary if you are running one Optuna study per process.

Note: To ensure correct trial order in Weights & Biases, this callback should only be used with `study.optimize(n_jobs=1)`.

Example

Add Weights & Biases callback to Optuna optimization.

```
import optuna
from optuna_integration.wandb import WeightsAndBiasesCallback


def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    return (x - 2) ** 2


study = optuna.create_study()

wandb_kwargs = {"project": "my-project"}
wandbc = WeightsAndBiasesCallback(wandb_kwargs=wandb_kwargs)

study.optimize(objective, n_trials=10, callbacks=[wandbc])
```

Weights & Biases logging in multirun mode.

```
import optuna
from optuna_integration.wandb import WeightsAndBiasesCallback

wandb_kwargs = {"project": "my-project"}
wandbc = WeightsAndBiasesCallback(wandb_kwargs=wandb_kwargs, as_multirun=True)


@wandbc.track_in_wandb()
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    return (x - 2) ** 2


study = optuna.create_study()
study.optimize(objective, n_trials=10, callbacks=[wandbc])
```

Parameters

- **metric_name** (*str | Sequence[str]*) – Name assigned to optimized metric. In case of multi-objective optimization, list of names can be passed. Those names will be assigned to metrics in the order returned by objective function. If single name is provided, or this argument is left to default value, it will be broadcasted to each objective with a number suffix in order returned by objective function e.g. two objectives and default metric name will be logged as `value_0` and `value_1`. The number of metrics must be the same as the number of values objective function returns.

- **wandb_kwargs** (*Dict[str, Any] | None*) – Set of arguments passed when initializing Weights & Biases run. Please refer to Weights & Biases API documentation for more details.
- **as_multirun** (*bool*) – Creates new runs for each trial. Useful for generating W&B Sweeps like panels (for ex., parameter importance, parallel coordinates, etc).

Note: Added in v2.9.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v2.9.0.

Methods

track_in_wandb()	Decorator for using W&B for logging inside the objective function.

track_in_wandb()

Decorator for using W&B for logging inside the objective function.

The run is initialized with the same `wandb_kwargs` that are passed to the callback. All the metrics from inside the objective function will be logged into the same run which stores the parameters for a given trial.

Example

Add additional logging to Weights & Biases.

```
import optuna
from optuna_integration.wandb import WeightsAndBiasesCallback
import wandb

wandb_kwargs = {"project": "my-project"}
wandbc = WeightsAndBiasesCallback(wandb_kwargs=wandb_kwargs, as_multirun=True)


@wandbc.track_in_wandb()
def objective(trial):
    x = trial.suggest_float("x", -10, 10)
    wandb.log({"power": 2, "base of metric": x - 2})

    return (x - 2) ** 2


study = optuna.create_study()
study.optimize(objective, n_trials=10, callbacks=[wandbc])
```

Returns
Objective function with W&B tracking enabled.

Return type
Callable

Note: Added in v3.0.0 as an experimental feature. The interface may change in newer versions without prior notice. See https://github.com/optuna/optuna/releases/tag/v3.0.0.

4.2.21 XGBoost

optuna_integration.XGBoostPruningCallback	Callback for XGBoost to prune unpromising trials.

optuna_integration.XGBoostPruningCallback

class optuna_integration.**XGBoostPruningCallback**(*trial*, *observation_key*)

Callback for XGBoost to prune unpromising trials.

See the example if you want to add a pruning callback which observes validation accuracy of a XGBoost model.

Parameters

- **trial** (*Trial*) – A `Trial` corresponding to the current evaluation of the objective function.
- **observation_key** (*str*) – An evaluation metric for pruning, e.g., `validation-error` and `validation-merror`. When using the Scikit-Learn API, the index number of `eval_set` must be included in the `observation_key`, e.g., `validation_0-error` and `validation_0-merror`. Please refer to `eval_metric` in XGBoost reference for further details.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

P

Q

R

S

T

U

W

X