optuna.trial.Trial

class **optuna.trial.Trial**(*study*, *trial_id*) [source]

A trial is a process of evaluating an objective function.

This object is passed to an objective function and provides interfaces to get parameter suggestion, manage the trial's state, and set/get user-defined attributes of the trial.

Note that the direct use of this constructor is not recommended. This object is seamlessly instantiated and passed to the objective function behind the `optuna.study.Study.optimize()` method; hence library users do not care about instantiation of this object.

Parameters:
- **study** (*optuna.study.Study*) – A `Study` object.
- **trial_id** (*int*) – A trial ID that is automatically generated.

Methods

`report`(value, step)	Report an objective function value for a given s
`set_system_attr`(key, value)	Set system attributes to the trial.
`set_user_attr`(key, value)	Set user attributes to the trial.
`should_prune`()	Suggest whether the trial should be pruned or
`suggest_categorical`()	Suggest a value for the categorical parameter.
`suggest_discrete_uniform`(name, low, high, q)	Suggest a value for the discrete parameter.
`suggest_float`(name, low, high, *[, step, log])	Suggest a value for the floating point paramete
`suggest_int`(name, low, high, *[, step, log])	Suggest a value for the integer parameter.
`suggest_loguniform`(name, low, high)	Suggest a value for the continuous parameter.
`suggest_uniform`(name, low, high)	Suggest a value for the continuous parameter.

Attributes

`datetime_start`	Return start datetime.
`distributions`	Return distributions of parameters to be optimized.

`number`	Return trial's number which is consecutive and unique in a study.
`params`	Return parameters to be optimized.
`relative_params`	
`system_attrs`	Return system attributes.
`user_attrs`	Return user attributes.

property **`datetime_start`**: *datetime* | *None*

Return start datetime.

Returns: Datetime where the `Trial` started.

property **`distributions`**: *Dict[str, BaseDistribution]*

Return distributions of parameters to be optimized.

Returns: A dictionary containing all distributions.

property **`number`**: *int*

Return trial’s number which is consecutive and unique in a study.

Returns: A trial number.

property **`params`**: *Dict[str, Any]*

Return parameters to be optimized.

Returns: A dictionary containing all parameters.

`report`(*value*, *step*) [source]

Report an objective function value for a given step.

The reported values are used by the pruners to determine whether this trial should be pruned.

See also

Please refer to `BasePruner`.

Note

The reported value is converted to `float` type by applying `float()` function internally. Thus, it accepts all float-like types (e.g., `numpy.float32`). If the conversion fails, a `TypeError` is raised.

Note

If this method is called multiple times at the same `step` in a trial, the reported `value` only the first time is stored and the reported values from the second time are ignored.

Note

`report()` does not support multi-objective optimization.

Example

Report intermediate scores of SGDClassifier training.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split

import optuna

X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y)


def objective(trial):
    clf = SGDClassifier(random_state=0)
    for step in range(100):
        clf.partial_fit(X_train, y_train, np.unique(y))
        intermediate_value = clf.score(X_valid, y_valid)
        trial.report(intermediate_value, step=step)
        if trial.should_prune():
            raise optuna.TrialPruned()

    return clf.score(X_valid, y_valid)


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=3)
```

Parameters:

- **value** (*float*) – A value returned from the objective function.
- **step** (*int*) – Step of the trial (e.g., Epoch of neural network training). Note that pruners assume that `step` starts at zero. For example, `MedianPruner` simply checks if `step` is less than `n_warmup_steps` as the warmup mechanism. `step` must be a positive integer.

Return type: None

`set_system_attr`(*key*, *value*) [source]

Set system attributes to the trial.

Note that Optuna internally uses this method to save system messages such as failure reason of trials. Please use `set_user_attr()` to set users' attributes.

Parameters:
- **key** (*str*) – A key string of the attribute.
- **value** (*Any*) – A value of the attribute. The value should be JSON serializable.

Return type: None

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.

`set_user_attr`(*key*, *value*) [source]

Set user attributes to the trial.

The user attributes in the trial can be access via `optuna.trial.Trial.user_attrs()`.

See also

See the recipe on User Attributes.

Example

Save fixed hyperparameters of neural network training.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

import optuna

X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, random_state=0)


def objective(trial):
    trial.set_user_attr("BATCHSIZE", 128)
    momentum = trial.suggest_float("momentum", 0, 1.0)
    clf = MLPClassifier(
        hidden_layer_sizes=(100, 50),
        batch_size=trial.user_attrs["BATCHSIZE"],
        momentum=momentum,
        solver="sgd",
        random_state=0,
    )
    clf.fit(X_train, y_train)

    return clf.score(X_valid, y_valid)


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=3)
assert "BATCHSIZE" in study.best_trial.user_attrs.keys()
assert study.best_trial.user_attrs["BATCHSIZE"] == 128
```

Parameters:
- **key** (*str*) – A key string of the attribute.
- **value** (*Any*) – A value of the attribute. The value should be JSON serializable.

Return type: None

should_prune() [source]

Suggest whether the trial should be pruned or not.

The suggestion is made by a pruning algorithm associated with the trial and is based on previously reported values. The algorithm can be specified when constructing a `Study`.

Note

If no values have been reported, the algorithm cannot make meaningful suggestions. Similarly, if this method is called multiple times with the exact same set of reported values, the suggestions will be the same.

See also

Please refer to the example code in `optuna.trial.Trial.report()`.

Note

`should_prune()` does not support multi-objective optimization.

Returns: A boolean value. If `True`, the trial should be pruned according to the configured pruning algorithm. Otherwise, the trial should continue.

Return type: bool

suggest_categorical(*name: str*, *choices: Sequence[None]*) → None [source]

suggest_categorical(*name: str*, *choices: Sequence[bool]*) → bool

suggest_categorical(*name: str*, *choices: Sequence[int]*) → int

suggest_categorical(*name: str*, *choices: Sequence[float]*) → float

suggest_categorical(*name: str*, *choices: Sequence[str]*) → str

suggest_categorical(*name: str*, *choices: Sequence[None | bool | int | float | str]*) → None | bool | int | float | str

Suggest a value for the categorical parameter.

The value is sampled from `choices`.

Example

Suggest a kernel function of SVC.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

import optuna

X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y)


def objective(trial):
    kernel = trial.suggest_categorical("kernel", ["linear", "poly", "rbf"])
    clf = SVC(kernel=kernel, gamma="scale", random_state=0)
    clf.fit(X_train, y_train)
    return clf.score(X_valid, y_valid)


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=3)
```

Parameters:
- **name** – A parameter name.
- **choices** – Parameter value candidates.

See also

`CategoricalDistribution`.

Returns: A suggested value.

See also

Pythonic Search Space tutorial describes more details and flexible usages.

`suggest_discrete_uniform`(*name, low, high, q*) [source]

Suggest a value for the discrete parameter.

The value is sampled from the range $[\mathsf{low}, \mathsf{high}]$, and the step of discretization is q. More specifically, this method returns one of the values in the sequence $\mathsf{low}, \mathsf{low} + q, \mathsf{low} + 2q, \ldots, \mathsf{low} + kq \le \mathsf{high}$, where k denotes an integer. Note that $high$ may be changed due to round-off errors if q is not an integer. Please check warning messages to find the changed values.

Parameters:

- **name** (*str*) – A parameter name.
- **low** (*float*) – Lower endpoint of the range of suggested values. `low` is included in the range.
- **high** (*float*) – Upper endpoint of the range of suggested values. `high` is included in the range.
- **q** (*float*) – A step of discretization.

Returns: A suggested float value.

Return type: float

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.

`suggest_float`(*name, low, high, *, step=None, log=False*) [source]

Suggest a value for the floating point parameter.

Example

Suggest a momentum, learning rate and scaling factor of learning rate for neural network training.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

import optuna

X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, random_state=0)


def objective(trial):
    momentum = trial.suggest_float("momentum", 0.0, 1.0)
    learning_rate_init = trial.suggest_float(
        "learning_rate_init", 1e-5, 1e-3, log=True
    )
    power_t = trial.suggest_float("power_t", 0.2, 0.8, step=0.1)
    clf = MLPClassifier(
        hidden_layer_sizes=(100, 50),
        momentum=momentum,
        learning_rate_init=learning_rate_init,
        solver="sgd",
        random_state=0,
        power_t=power_t,
    )
    clf.fit(X_train, y_train)

    return clf.score(X_valid, y_valid)


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=3)
```

Parameters:

- **name** (*str*) – A parameter name.
- **low** (*float*) – Lower endpoint of the range of suggested values. `low` is included in the range. `low` must be less than or equal to `high`. If `log` is `True`, `low` must be larger than 0.
- **high** (*float*) – Upper endpoint of the range of suggested values. `high` is included in the range. `high` must be greater than or equal to `low`.
- **step** (*float* | *None*) –

  A step of discretization.

  Note

  The `step` and `log` arguments cannot be used at the same time. To set the `step` argument to a float number, set the `log` argument to `False`.
- **log** (*bool*) –

  A flag to sample the value from the log domain or not. If `log` is true, the value is sampled from the range in the log domain. Otherwise, the value is sampled from the range in the linear domain.

  Note

  The `step` and `log` arguments cannot be used at the same time. To set the `log` argument to `True`, set the `step` argument to `None`.

Returns: A suggested float value.

Return type: float

See also

Pythonic Search Space tutorial describes more details and flexible usages.

`suggest_int`(*name*, *low*, *high*, ***, *step=1*, *log=False*) [source]

Suggest a value for the integer parameter.

The value is sampled from the integers in $[\text{low}, \text{high}]$.

Example

Suggest the number of trees in RandomForestClassifier.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

import optuna

X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y)


def objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 50, 400)
    clf = RandomForestClassifier(n_estimators=n_estimators, random_state=0)
    clf.fit(X_train, y_train)
    return clf.score(X_valid, y_valid)


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=3)
```

Parameters:

- **name** (*str*) – A parameter name.
- **low** (*int*) – Lower endpoint of the range of suggested values. `low` is included in the range. `low` must be less than or equal to `high`. If `log` is `True`, `low` must be larger than 0.
- **high** (*int*) – Upper endpoint of the range of suggested values. `high` is included in the range. `high` must be greater than or equal to `low`.
- **step** (*int*) –

  A step of discretization.

  Note

  Note that **high** is modified if the range is not divisible by **step**. Please check the warning messages to find the changed values.

  Note

  The method returns one of the values in the sequence $\mathsf{low}, \mathsf{low} + \mathsf{step}, \mathsf{low} + 2 * \mathsf{step}, \dots, \mathsf{low} + k * \mathsf{step} \le \mathsf{high}$, where k denotes an integer.

  Note

  The `step != 1` and `log` arguments cannot be used at the same time. To set the `step` argument $\mathsf{step} \ge 2$, set the `log` argument to `False`.

- **log** (*bool*) –

  A flag to sample the value from the log domain or not.

  Note

  If `log` is true, at first, the range of suggested values is divided into grid points of width 1. The range of suggested values is then converted to a log domain, from which a value is sampled. The uniformly sampled value is re-converted to the original domain and rounded to the nearest grid point that we just split, and the suggested value is determined. For example, if *low = 2* and *high = 8*, then the range of suggested values is *[2, 3, 4, 5, 6, 7, 8]* and lower values tend to be more sampled than higher values.

  Note

  The `step != 1` and `log` arguments cannot be used at the same time. To set the `log` argument to `True`, set the `step` argument to 1.

Return type: int

See also

Pythonic Search Space tutorial describes more details and flexible usages.

`suggest_loguniform`(*name*, *low*, *high*) [source]

Suggest a value for the continuous parameter.

The value is sampled from the range $[\mathsf{low}, \mathsf{high})$ in the log domain. When $\mathsf{low} = \mathsf{high}$, the value of low will be returned.

Parameters:
- **name** (*str*) – A parameter name.
- **low** (*float*) – Lower endpoint of the range of suggested values. `low` is included in the range.
- **high** (*float*) – Upper endpoint of the range of suggested values. `high` is included in the range.

Returns: A suggested float value.

Return type: float

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.

`suggest_uniform`(*name*, *low*, *high*) [source]

Suggest a value for the continuous parameter.

The value is sampled from the range $[\mathsf{low}, \mathsf{high})$ in the linear domain. When $\mathsf{low} = \mathsf{high}$, the value of low will be returned.

Parameters:
- **name** (*str*) – A parameter name.
- **low** (*float*) – Lower endpoint of the range of suggested values. `low` is included in the range.
- **high** (*float*) – Upper endpoint of the range of suggested values. `high` is included in the range.

Returns: A suggested float value.

Return type: float

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.

property **`system_attrs`**: *Dict[str, Any]*

Return system attributes.

Returns: A dictionary containing all system attributes.

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.

property **`user_attrs`**: *Dict[str, Any]*

Return user attributes.

Returns: A dictionary containing all user attributes.