

Optuna

ハイパーパラメータ最適化フレームワーク

Preferred Networks

太田 健 (Takeru Ohta)

2019 年 9 月 27 日

PyData.Tokyo Meetup #21



OPTUNA

- ハイパーパラメータ最適化フレームワーク
 - 2018/12にv0.4.0を公開 (現在はv0.16.0)
 - <https://github.com/pfnet/optuna>
- 特徴:
 - Define-by-Run
 - 枝刈り
 - 分散最適化



OPTUNA

- 社内外のいろいろなプロジェクトで活用
- パブリックな事例だと、
 - Kaggle: Open Images Challenge 2018 (二位)
 - Object Detection Track
 - 物体の重複検出抑制パラメータ(NMW Threshold)を最適化
 - KDD Cup 2019: AutoML Track (五位)
 - LightGBMのハイパーパラメータを最適化
 - <https://research.preferred.jp/2019/09/kddcup2019automl/>

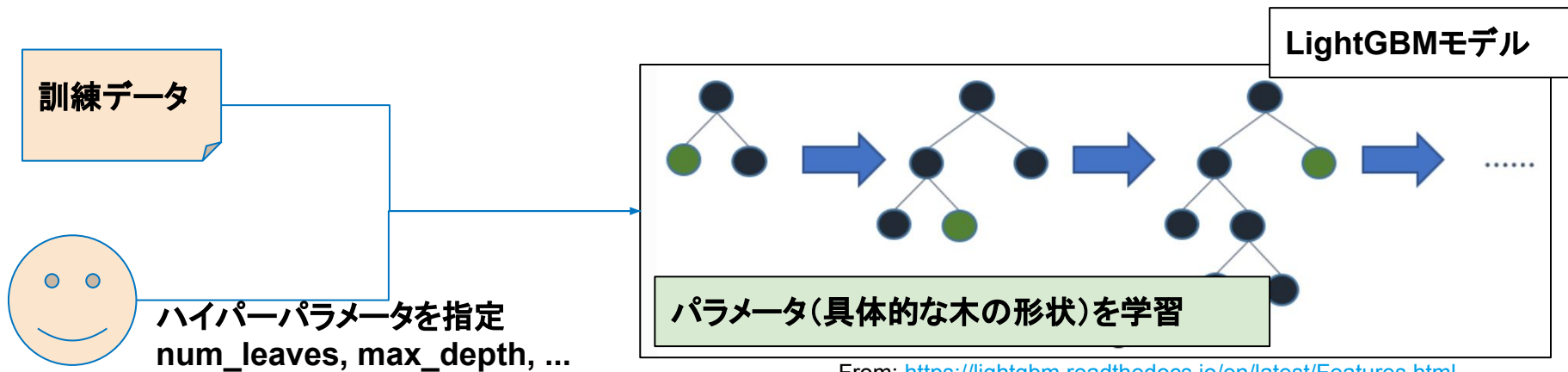
目次

- ハイパーパラメータ最適化
- Optuna入門
- 発展的な使い方
- 新機能: LightGBMTuner

ハイパーパラメータ最適化

ハイパーパラメータとは？

- 機械が自動で学習: パラメータ
 - 訓練データから学習
- 人手で設定が必要: **ハイパーパラメータ**



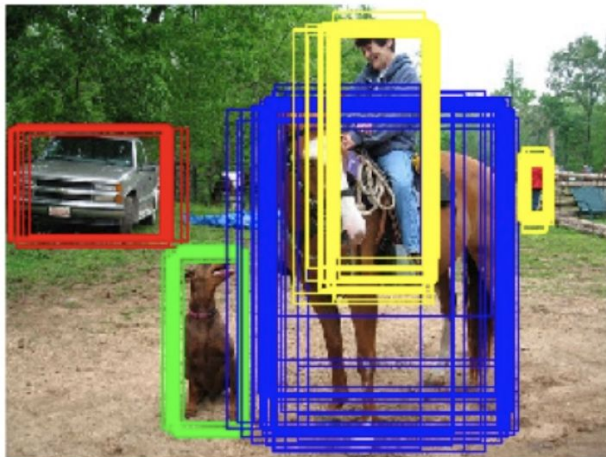
From: <https://lightgbm.readthedocs.io/en/latest/Features.html>

ハイパーパラメータの重要性

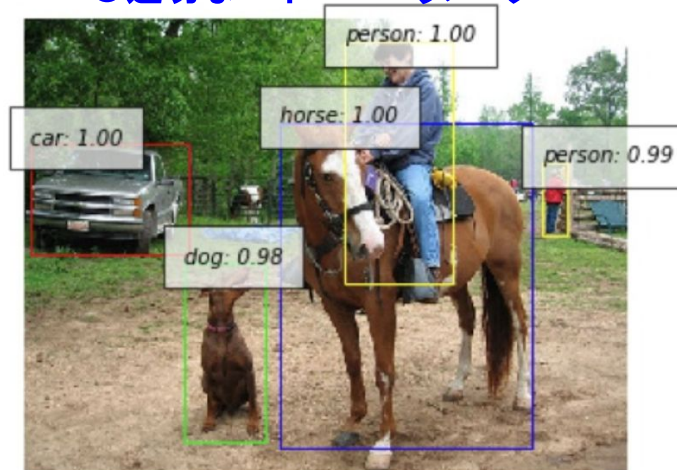
物体検出モデルの例:

- どちらも同じ学習済みモデルを使用
- 推論時のハイパーパラメータだけが異なる

×不適切なハイパーパラメータ

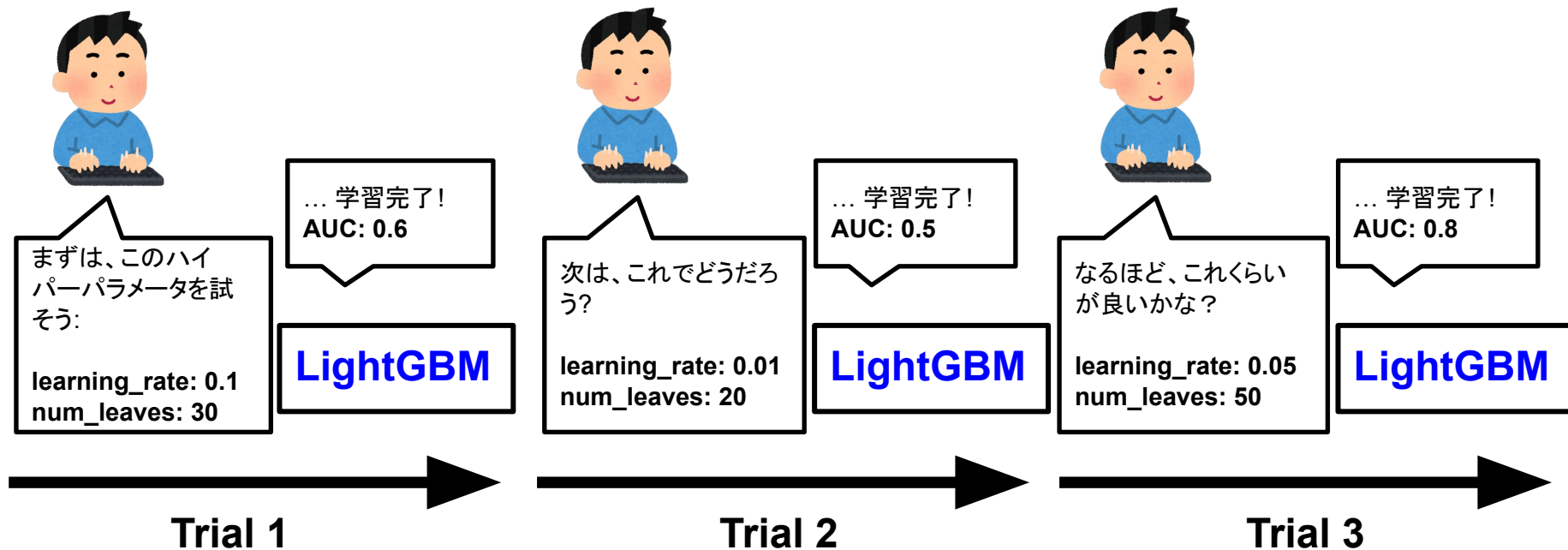


○適切なハイパーパラメータ

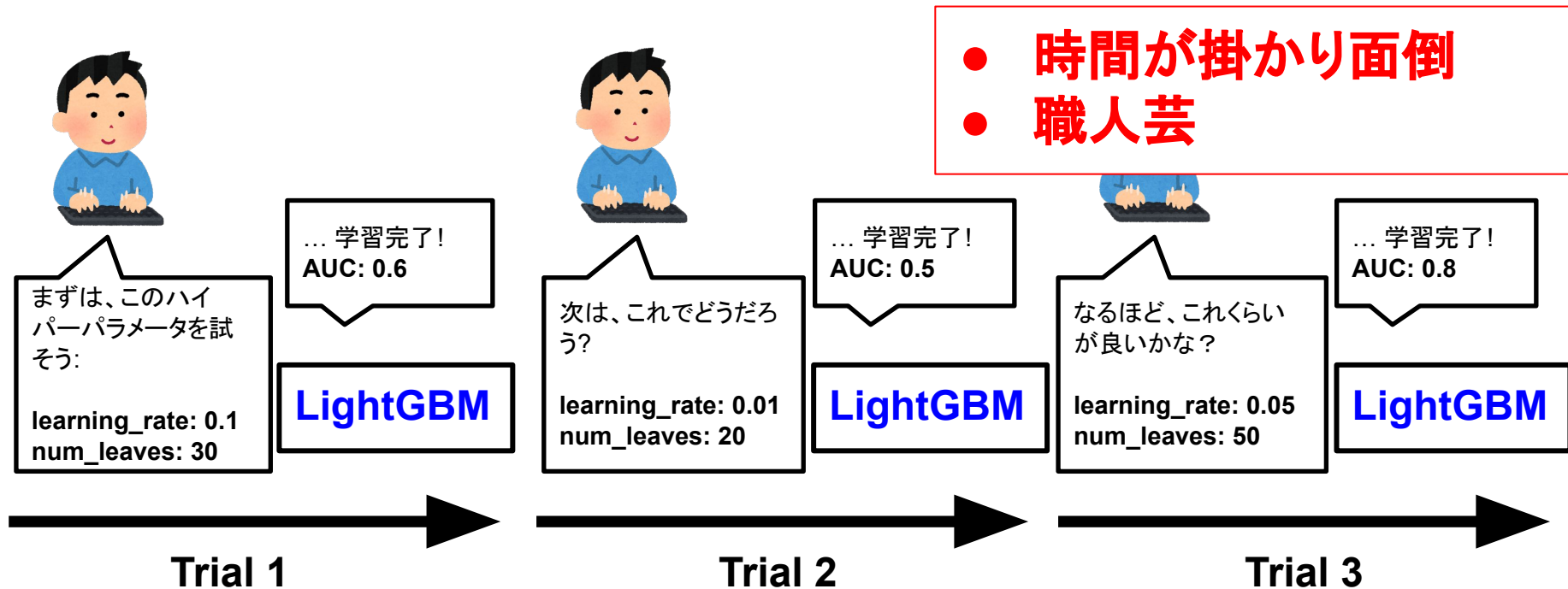


画像は PASCAL VOC2007データセット より引用

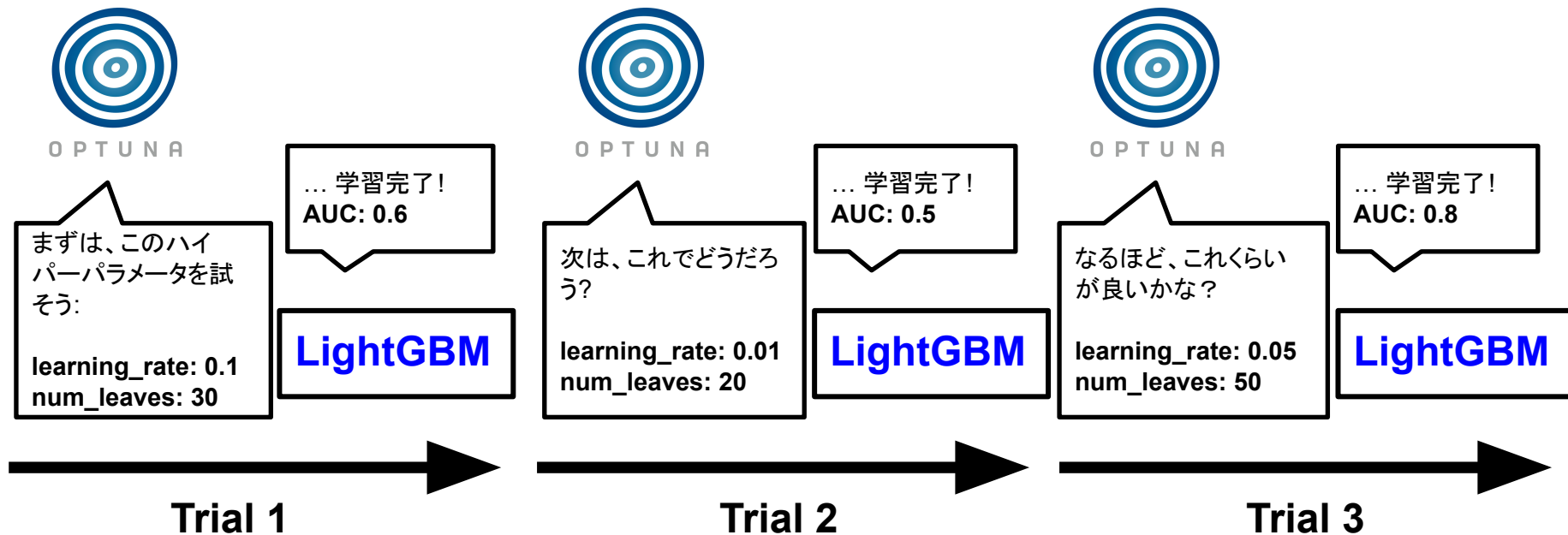
ハイパーパラメータ探索(手動)



ハイパーパラメータ探索(手動)



ハイパーパラメータ探索(自動)



Optuna入門

インストール

- Python 2.7, Python3.5+をサポート
- pipでインストール可能

\$ pip install optuna

コードの基本的な構成

```
import optuna
```

```
def objective(trial):
```

最適化対象のコード

```
    return evaluation_score
```

```
study = optuna.create_study()
```

```
study.optimize(objective, n_trials=
```

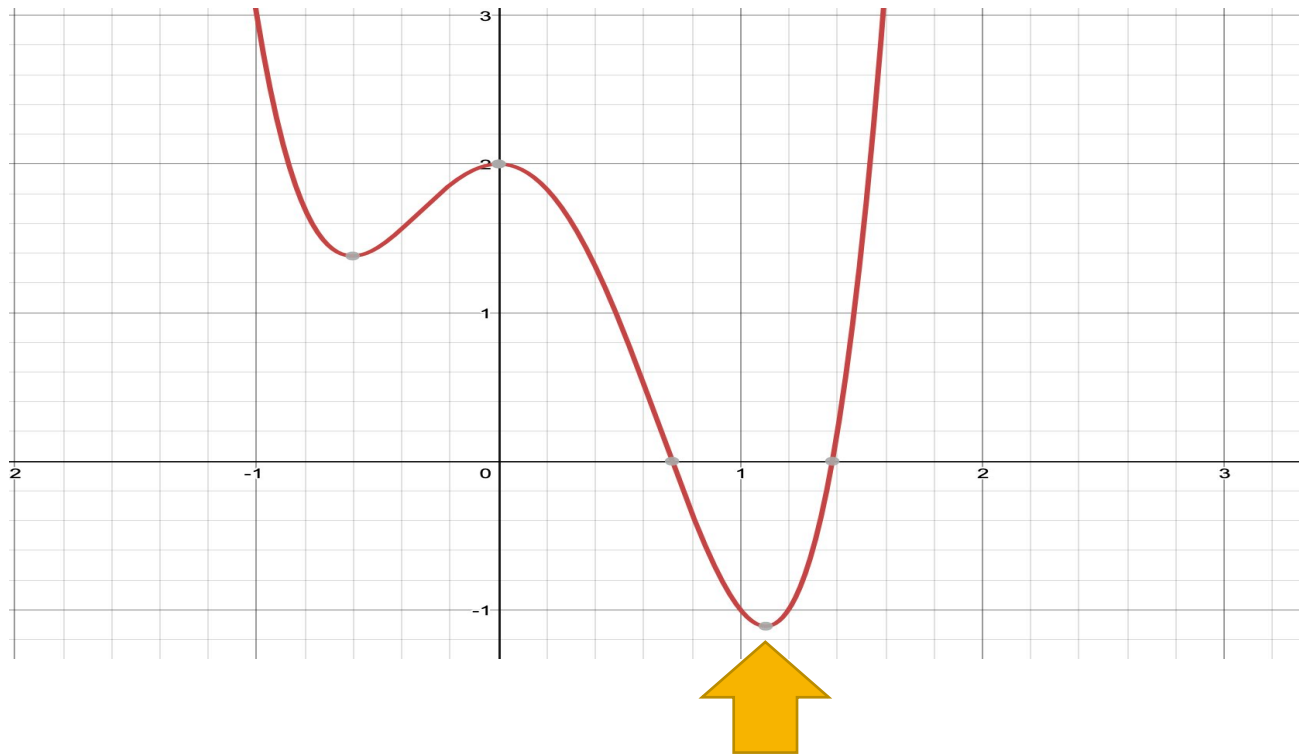
トライアル数

)

目的関数の
定義

最適化実行

簡単な例: $3x^4 - 2x^3 - 4x^2 + 2$ の最小化



$3x^4 - 2x^3 - 4x^2 + 2$ の最小化

```
import optuna

# 目的関数を定義
def objective(trial):
    x = trial.suggest_uniform('x', -2, 2)
    return 3*x**4 - 2*x**3 - 4*x**2 + 2

# 最適化を実行
study = optuna.create_study()
study.optimize(objective, n_trials=100)
```

$3x^4 - 2x^3 - 4x^2 + 2$ の最小化

```
import optuna
```

```
# 目的関数を定義
```

```
def objective(trial):
```

trialオブジェクトを引数に取る関数を定義

```
    x = trial.suggest_uniform('x', -2, 2)
```

```
    return 3*x**4 - 2*x**3 - 4*x**2 + 2
```

```
# 最適化を実行
```

```
study = optuna.create_study()
```

```
study.optimize(objective, n_trials=100)
```


$3x^4 - 2x^3 - 4x^2 + 2$ の最小化

```
import optuna
```

```
# 目的関数を定義
```

```
def objective(trial):
```

次に評価するハイパーパラメータの値を取得
(Suggest API)

```
    x = trial.suggest_uniform('x', -2, 2)
```

```
    return 3*x**4 - 2*x**3 - 4*x**2 + 2
```

```
# 最適化を実行
```

```
study = optuna.create_study()
```

```
study.optimize(objective, n_trials=100)
```

$3x^4 - 2x^3 - 4x^2 + 2$ の最小化

```
import optuna
```

```
# 目的関数を定義
```

```
def objective(trial):
```

```
    x = trial.suggest_uniform('x', -2, 2)
```

```
    return 3*x**4 - 2*x**3 - 4*x**2 + 2
```

ハイパーパラメータの評価スコアを返す

```
# 最適化を実行
```

```
study = optuna.create_study()
```

```
study.optimize(objective, n_trials=100)
```

$3x^4 - 2x^3 - 4x^2 + 2$ の最小化

```
import optuna
```

```
# 目的関数を定義
```

```
def objective(trial):
```

```
    x = trial.suggest_uniform('x', -2, 2)
```

```
    return 3*x**4 - 2*x**3 - 4*x**2 + 2
```

```
# 最適化を実行
```

最適化処理を管理するstudyオブジェクトを生成

```
study = optuna.create_study()
```

```
study.optimize(objective, n_trials=100)
```

$3x^4 - 2x^3 - 4x^2 + 2$ の最小化

```
import optuna
```

```
# 目的関数を定義
```

```
def objective(trial):
```

```
    x = trial.suggest_uniform('x', -2, 2)
```

```
    return 3*x**4 - 2*x**3 - 4*x**2 + 2
```

```
# 最適化を実行
```

```
study = optuna.create_study() 目的関数と試行回数を指定して、最適化を実行
```

```
study.optimize(objective, n trials=100)
```

実行

```
[2] def objective(trial):  
    x = trial.suggest_uniform('x', -2, 2)  
    return 3*x**4 - 2*x**3 - 4*x**2 + 2
```

```
study = optuna.create_study()  
study.optimize(objective, n_trials=100)
```

```
[I 2019-09-24 14:28:06,542] Finished trial#0 resulted in value: 1.9177263458077267. Current best value is 1.9  
[I 2019-09-24 14:28:06,641] Finished trial#1 resulted in value: 6.904266214258072. Current best value is 1.91  
[I 2019-09-24 14:28:06,737] Finished trial#2 resulted in value: 10.431008366062057. Current best value is 1.9  
[I 2019-09-24 14:28:06,840] Finished trial#3 resulted in value: 1.9924298806941536. Current best value is 1.9  
[I 2019-09-24 14:28:06,943] Finished trial#4 resulted in value: 15.425301178339494. Current best value is 1.9  
[I 2019-09-24 14:28:07,047] Finished trial#5 resulted in value: 3.260685911859218. Current best value is 1.91  
[I 2019-09-24 14:28:07,148] Finished trial#6 resulted in value: 9.20472534493579. Current best value is 1.917  
[I 2019-09-24 14:28:07,244] Finished trial#7 resulted in value: -1.0412842061076497. Current best value is -1  
[I 2019-09-24 14:28:07,335] Finished trial#8 resulted in value: 39.24118239491834. Current best value is -1.0  
[I 2019-09-24 14:28:07,434] Finished trial#9 resulted in value: 4.826996600558126. Current best value is -1.0  
[I 2019-09-24 14:28:07,530] Finished trial#10 resulted in value: -1.014358162444909  
[I 2019-09-24 14:28:07,628] Finished trial#11 resulted in value: -0.965725329108589  
[I 2019-09-24 14:28:07,728] Finished trial#12 resulted in value: -1.089479777746263  
[I 2019-09-24 14:28:07,829] Finished trial#13 resulted in value: -1.065029220086626  
[I 2019-09-24 14:28:07,930] Finished trial#14 resulted in value: 0.7228555081476944  
[I 2019-09-24 14:28:08,033] Finished trial#15 resulted in value: -0.150741997738691  
[I 2019-09-24 14:28:08,133] Finished trial#16 resulted in value: 1.466025130273822. Current best value is -1  
[I 2019-09-24 14:28:08,234] Finished trial#17 resulted in value: 0.7861321428671839. Current best value is -1  
[I 2019-09-24 14:28:08,337] Finished trial#18 resulted in value: 3.8446123998692805. Current best value is -1  
[I 2019-09-24 14:28:08,438] Finished trial#19 resulted in value: 0.34612664567829077. Current best value is -
```

ハイパーパラメータの値を変えつつ、目的関数を繰り返し評価して、最適値(最小値)を探索

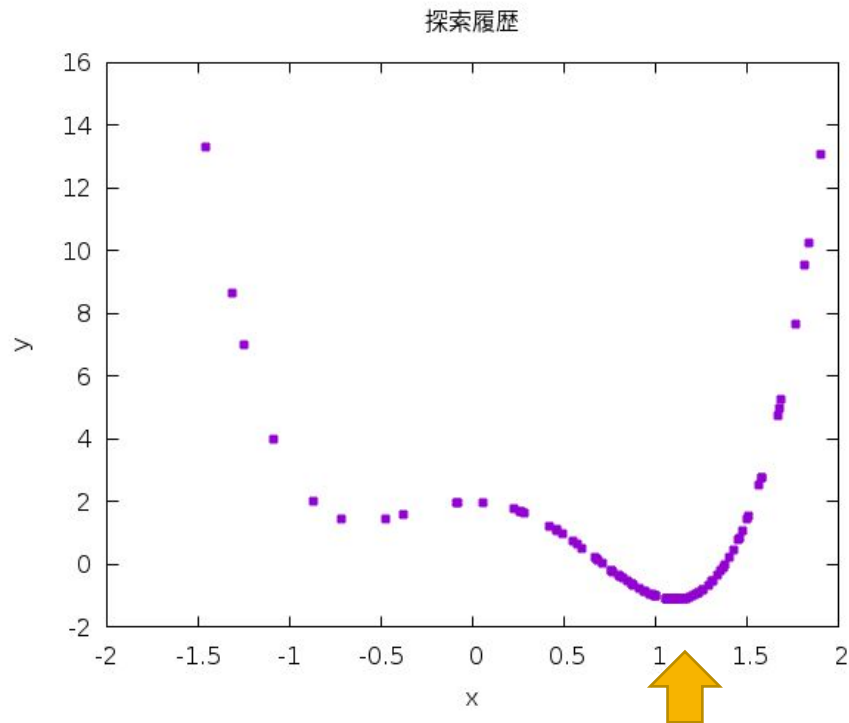
ベストトライアル

```
[4] study.best_params
```

```
{'x': 1.1073350297777722}
```

```
[5] study.best_value
```

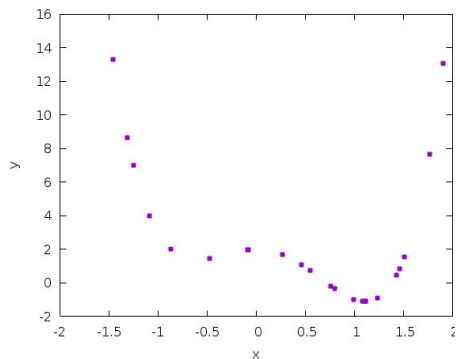
```
-1.109739540162666
```



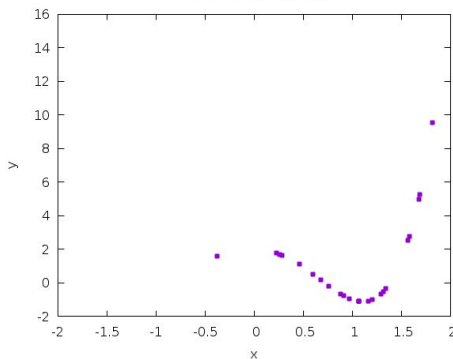
デフォルトの最適化アルゴリズム

- **TPE**: Tree-structured Parzen Estimator
 - ベイズ最適化の一種
 - 過去の評価履歴から、次に探索すべき点を推測

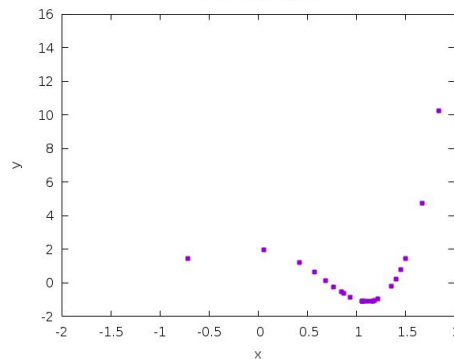
トライアル番号: 0 .. 25



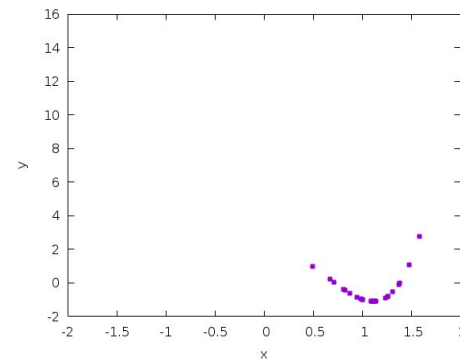
トライアル番号: 25 .. 50



トライアル番号: 50 .. 75



トライアル番号: 75 .. 100



探索地点が最適解付近に収
束

LightGBMの例: 二値分類のaccuracy最大化

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def main():
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': 'gbdt',
        'num_leaves': 31,
        'learning_rate': 0.1
    }

    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

print("Accuracy:", main())
```

最適化対応前のコード

LightGBMの例: 二値分類のaccuracy最大化

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def main():
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': 'gbdt',
        'num_leaves': 31,
        'learning_rate': 0.1
    }

    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

print("Accuracy:", main())
```

ハイパーパラメータは決め打ち

最適化対応前のコード

最適化対応

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def main():
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': 'gbdt',
        'num_leaves': 31,
        'learning_rate': 0.1
    }

    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

print("Accuracy:", main())
```

最適化対応

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def objective(trial):
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': 'gbdt',
        'num_leaves': 31,
        'learning_rate': 0.1
    }

    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

print("Accuracy:", main())
```

最適化対応

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def objective(trial):
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': trial.suggest_categorical('boosting', ['gbdt', 'dart']),
        'num_leaves': trial.suggest_int('num_leaves', 10, 1000),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-8, 1.0)
    }

    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

print("Accuracy:", main())
```

最適化対応

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def objective(trial):
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': trial.suggest_categorical('boosting', ['gbdt', 'dart']),
        'num_leaves': trial.suggest_int('num_leaves', 10, 1000),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-8, 1.0)
    }

    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```


特徴: Define-by-Run

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def objective(trial):
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)

    param = {
        'objective': 'binary',
        'boosting_type': trial.suggest_categorical('boosting', ['gbdt', 'dart']),
        'num_leaves': trial.suggest_int('num_leaves', 10, 1000),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-8, 1.0)
    }

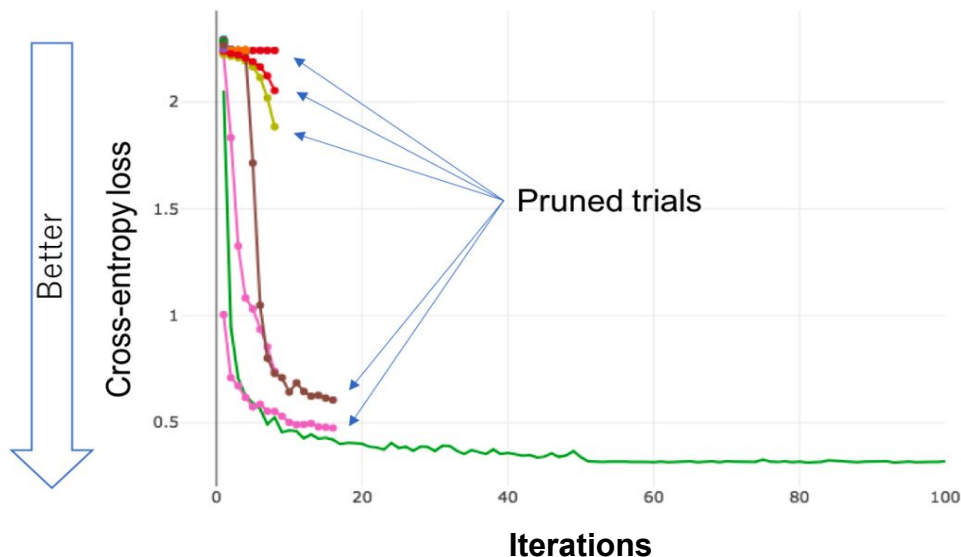
    gbm = lgb.train(param, lgb.Dataset(train_x, label=train_y))
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

Define-by-Run: 探索空間を目的関数の実行時に定義

特徴: 枝刈り (Pruning)

- 見込みの薄いトライアルを自動で中断
- **LightGBMPruningCallback**を追加するだけで利用可能



```
from optuna.integration import LightGBMPruningCallback

pruning_callback = LightGBMPruningCallback(trial, 'auc')
gbm = lgb.train(param,
                 dtrain,
                 valid_sets=[dtest],
                 callbacks=[pruning_callback])
```

lgb.cv()にも対応済み！

特徴: 分散最適化

- 複数のトライアルを並列・分散して実行可能
- ***study_name***と***storage***を指定し、複数プロセスから実行

```
# filename: parallel-optimize.py

def objective(trial):
    ...

study = optuna.create_study(
    study_name="example-study",
    storage='sqlite:///example.db',
    load_if_exists=True
)
study.optimize(objective, n_trials=100)
```

```
// 八並列で最適化を実行
$ python parallel-optimize.py &
$ python parallel-optimize.py &
$ python parallel-optimize.py &
$ python parallel-optimize.py &
$ python parallel-optimize.py &
$ python parallel-optimize.py &
$ python parallel-optimize.py &
$ python parallel-optimize.py &
```


完成コード

```
import lightgbm as lgb, numpy as np, optuna, sklearn.datasets, sklearn.metrics
from sklearn.model_selection import train_test_split

def objective(trial):
    data, target = sklearn.datasets.load_breast_cancer(return_X_y=True)
    train_x, test_x, train_y, test_y = train_test_split(data, target, random_state=0)
    dtrain = lgb.Dataset(train_x, label=train_y)
    dtest = lgb.Dataset(test_x, label=test_y)

    param = {
        'objective': 'binary',
        'metric': 'auc',
        'boosting_type': trial.suggest_categorical('boosting_type', ['gbdt', 'dart']),
        'num_leaves': trial.suggest_int('num_leaves', 10, 1000),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-8, 1.0)
    }

    pruning_callback = optuna.integration.LightGBMPruningCallback(trial, 'auc')
    gbm = lgb.train(param, dtrain, valid_sets=[dtest], callbacks=[pruning_callback])
    preds = gbm.predict(test_x)
    return sklearn.metrics.accuracy_score(test_y, np rint(preds))

study = optuna.create_study(
    study_name="example-study",
    storage="sqlite:///example.db",
    load_if_exists=True,
    direction='maximize'
)
study.optimize(objective, n_trials=100)
```

探索空間定義 (define-by-run)

枝刈り対応

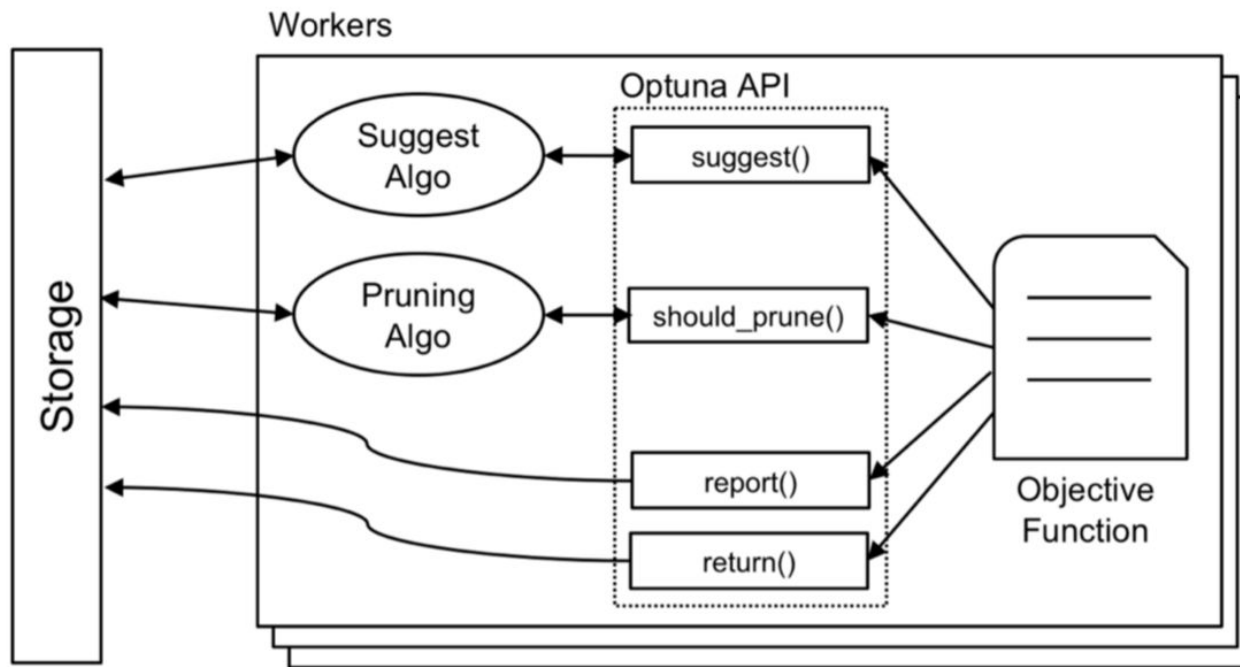
分散最適化対応

入門用ドキュメント

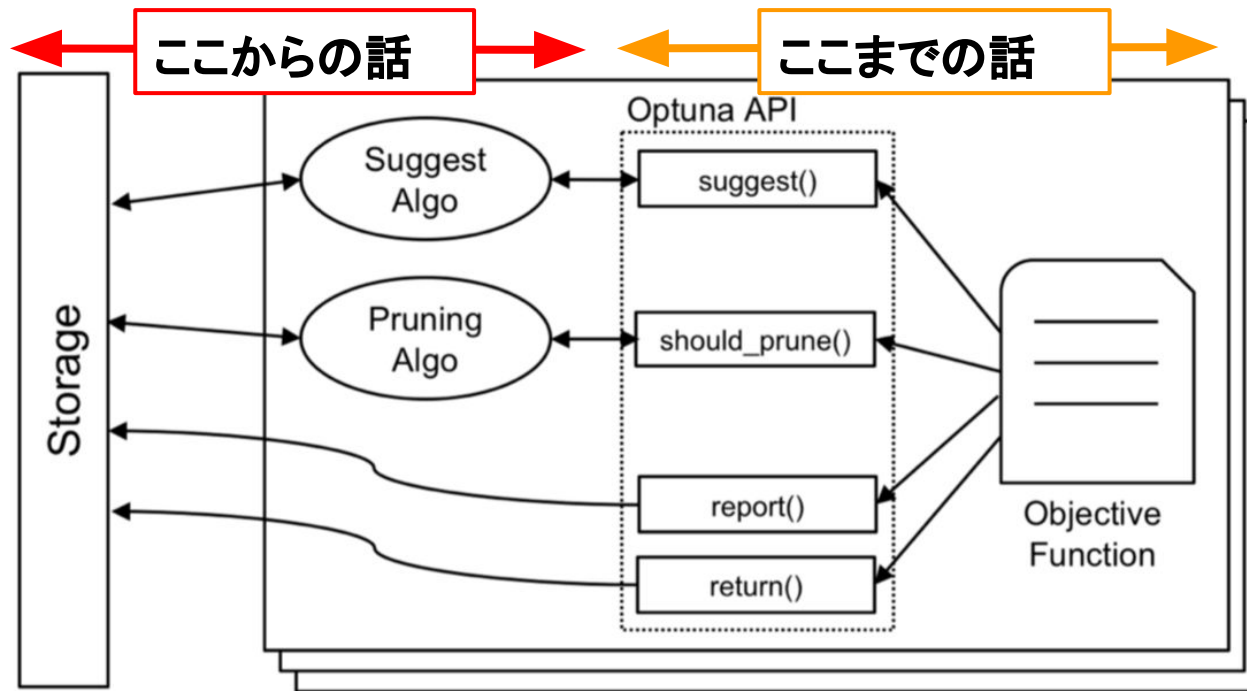
- 公式ドキュメント
 - チュートリアル: <https://optuna.readthedocs.io/en/stable/tutorial>
 - FAQ: <https://optuna.readthedocs.io/en/stable/faq.html>
- Colabハンズオン
 - 英語: <https://bit.ly/optuna-quick-start>
 - 日本語: <https://bit.ly/optuna-quick-start-ja>
- Examples
 - <https://github.com/pfnet/optuna/tree/master/examples>

発展的な使い方

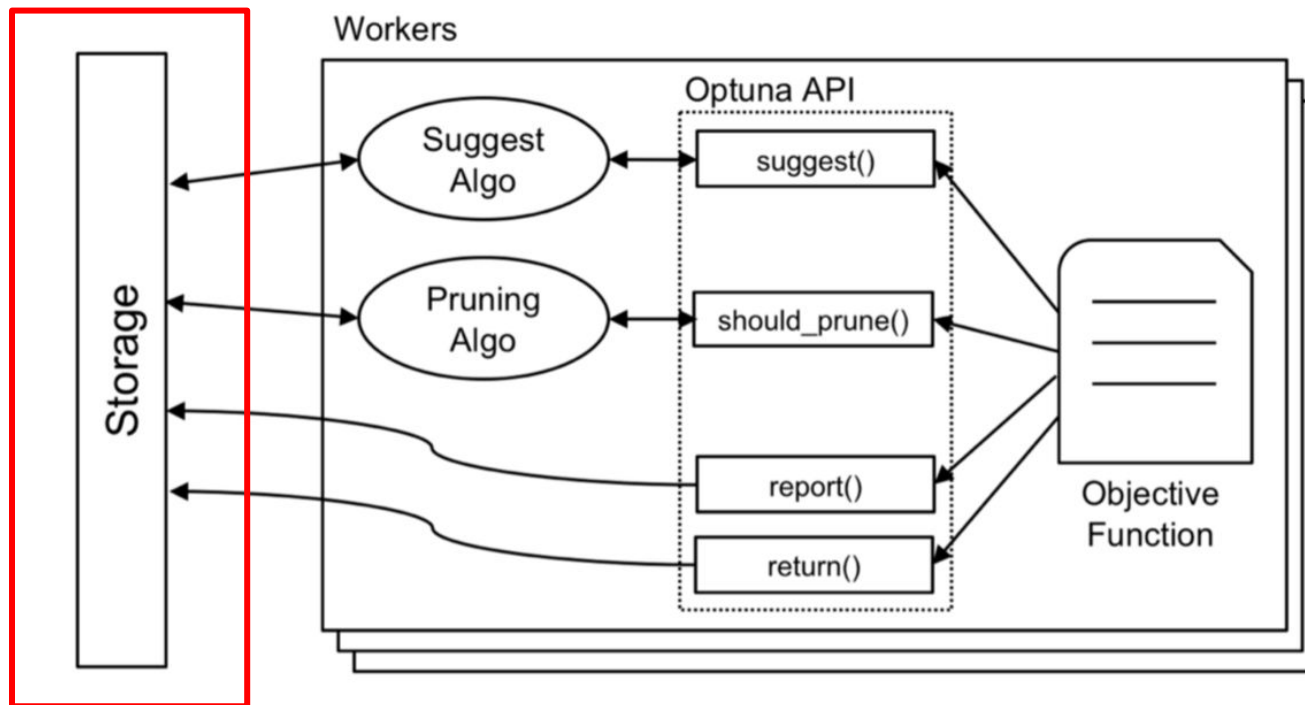
Optunaの構成要素



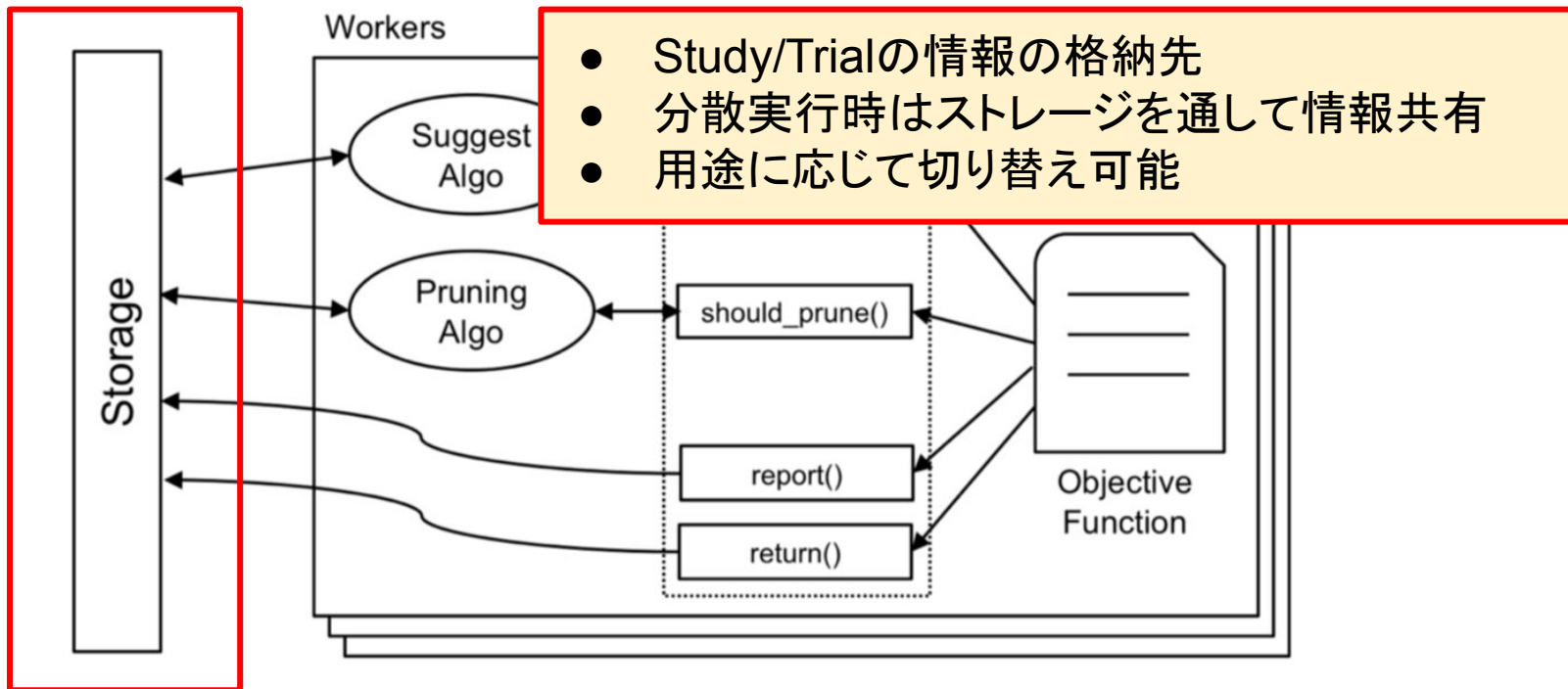
Optunaの構成要素



Optunaの構成要素: Storage



Optunaの構成要素: Storage



ストレージの種類

- **InMemoryStorage**

- デフォルトで 사용되는メモリ上のストレージ
- `optuna.create_study()`

- **RDBStorage**

- RDBをバックエンドにしたストレージ
- SQLAlchemyがサポートするRDBが使用可能
 - E.g., MySQL, PostgreSQL, SQLite
 - SQLiteはファイルベースのRDB (標準ライブラリに組み込み)
- `optuna.create_study(storage=RDB_URL)`

ストレージの使い分け

- とりあえずOptunaを動かしてみたい
⇒ InMemoryStorage
- 分散最適化！
⇒ SQLite以外のRDBStorage (e.g., MySQL, PostgreSQL)
※ SQLiteは、NFSと相性が悪く、スケールもしないので注意
- Studyを中断・再開したり、後から結果を分析したい
⇒ RDBStorage
「とりあえずSQLiteに保存」という習慣をつけておくと便利

RDBストレージの活用: Studyの中断・再開

```
$ python
>>> ...
>>> study = optuna.create_study(study_name="foo", storage="sqlite:///example.db")
>>> study.optimize(objective)
Ctrl+C  # 中断

# 再開
$ python
>>> ...
>>> study = optuna.load_study(study_name="foo", storage="sqlite:///example.db")
>>> study.optimize(objective)
```

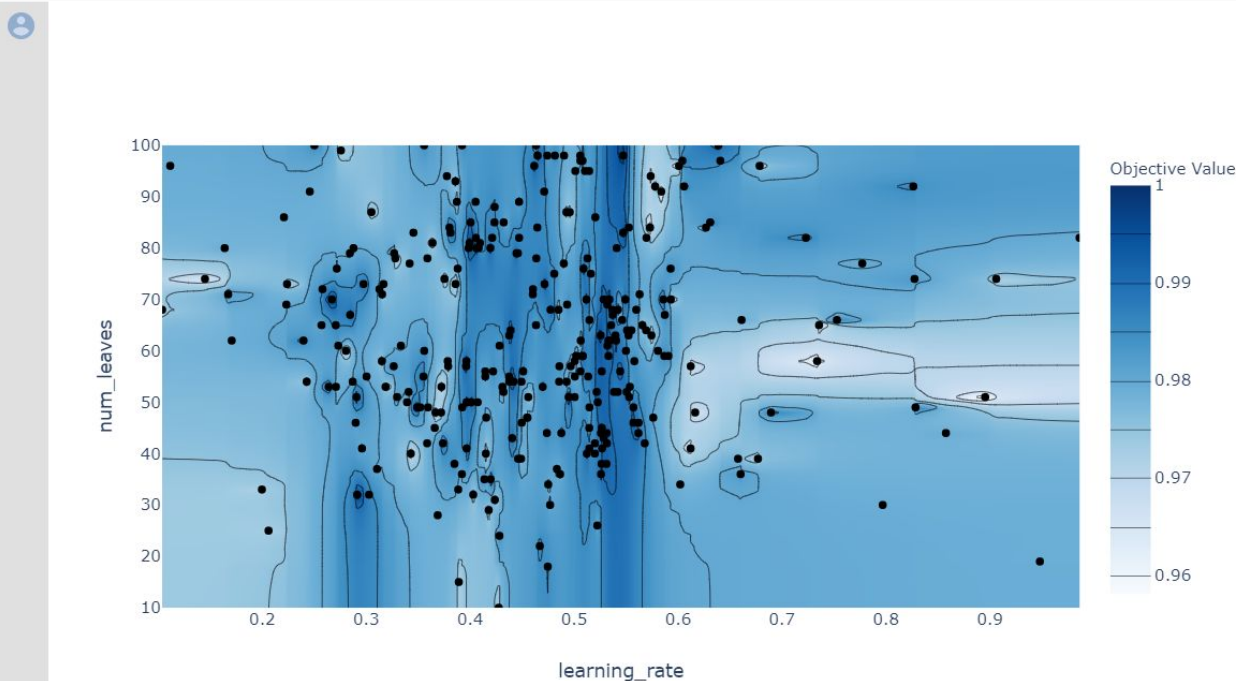
RDBストレージの活用: Pandasで分析

```
study = optuna.load_study(study_name='foo', storage='sqlite:///example.db')  
study.trials_dataframe()
```

	number	state	value	datetime_start	datetime_complete	params x	system_attrs _number
0	0	TrialState.COMPLETE	5.487975	2019-09-26 01:36:10.211987	2019-09-26 01:36:10.445876	-1.177585	0
1	1	TrialState.COMPLETE	31.769393	2019-09-26 01:36:10.472640	2019-09-26 01:36:10.663493	-1.794275	1
2	2	TrialState.COMPLETE	1.410450	2019-09-26 01:36:10.689861	2019-09-26 01:36:10.884699	0.369012	2
3	3	TrialState.COMPLETE	1.349665	2019-09-26 01:36:10.914329	2019-09-26 01:36:11.094601	1.494080	3
4	4	TrialState.COMPLETE	2.358674	2019-09-26 01:36:11.121926	2019-09-26 01:36:11.305250	1.555566	4
5	5	TrialState.COMPLETE	14.195920	2019-09-26 01:36:11.332231	2019-09-26 01:36:11.531708	-1.481995	5
6	6	TrialState.COMPLETE	1.871237	2019-09-26 01:36:11.560869	2019-09-26 01:36:11.787473	-0.843130	6
7	7	TrialState.COMPLETE	-0.535999	2019-09-26 01:36:11.816718	2019-09-26 01:36:12.045403	1.308312	7
8	8	TrialState.COMPLETE	1.444640	2019-09-26 01:36:12.073081	2019-09-26 01:36:12.302155	-0.493028	8
9	9	TrialState.COMPLETE	3.150132	2019-09-26 01:36:12.333075	2019-09-26 01:36:12.539886	-1.014584	9
10	10	TrialState.COMPLETE	0.175259	2019-09-26 01:36:12.566736	2019-09-26 01:36:12.751929	0.677235	10

RDBストレージの活用: Visualization

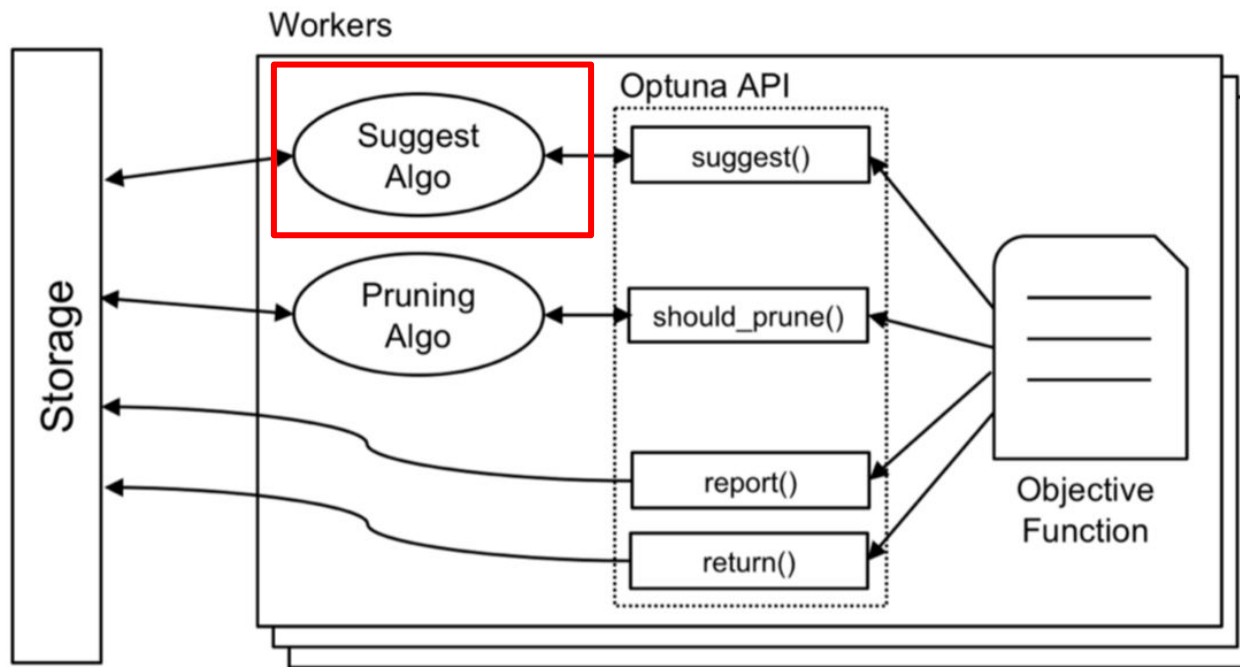
```
study = optuna.load_study(study_name='foo', storage='sqlite:///example.db')  
optuna.visualization.plot_contour(study)
```



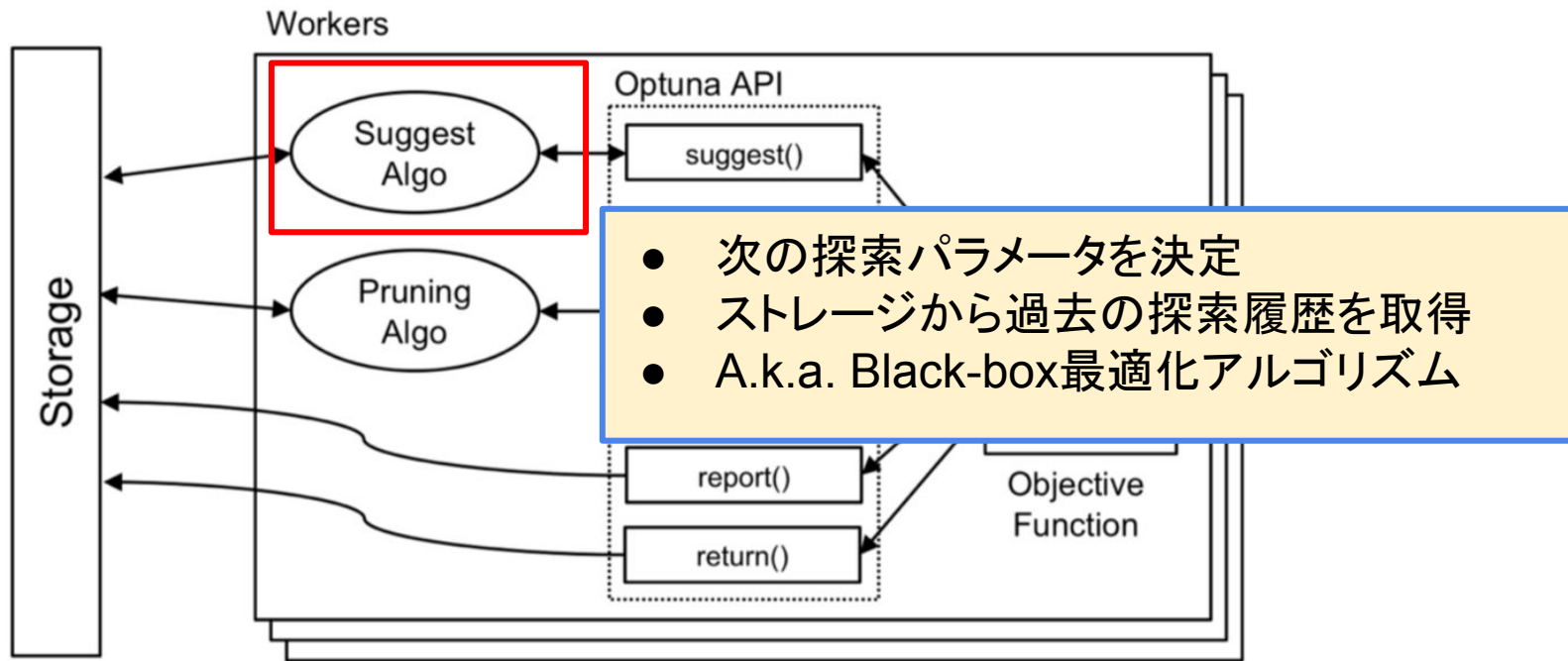
Visualization PRs:

- learning_curve: [#511](#)
- optimization_history: [#513](#)
- parallel_coordinate: [#531](#)
- contour_plot: [#539](#)
- slice_plot: [#540](#)

Optunaの構成要素: Sampler



Optunaの構成要素: Sampler



利用可能なSampler

- TPESampler (TPE)
- RandomSampler (ランダムサーチ)
- SkoptSampler (Gaussian Process ※ 変更可能)
 - <https://github.com/scikit-optimize/scikit-optimize>のラッパー
- CmaEsSampler (CMA-ES)
 - <https://github.com/CMA-ES/pycma>のラッパー
- ユーザ定義Sampler

指定方法

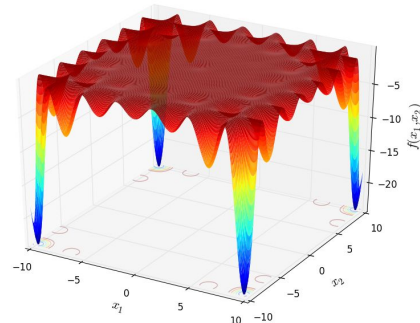
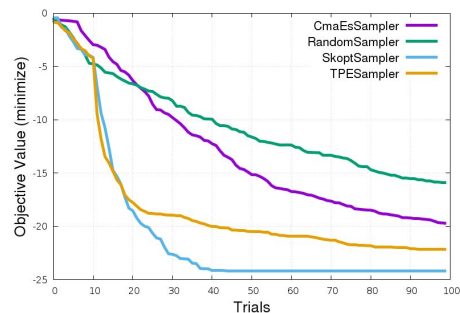
```
sampler = optuna.samplers.TPESampler()  
study = optuna.create_study(sampler=sampler)
```


なぜ色々なSamplerがあるの？

タスクによって最適な
Samplerは変わる

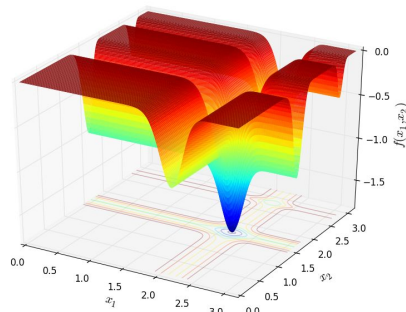
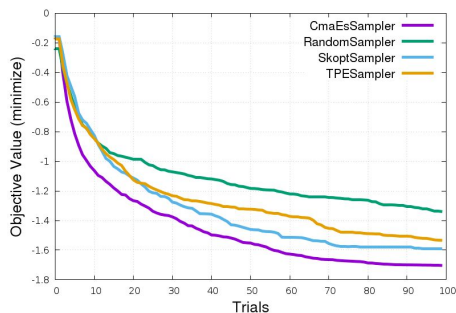
SkoptSamplerが強い

CarromTable Function



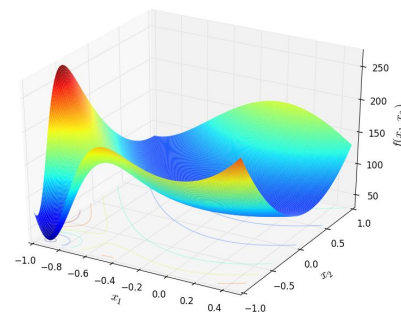
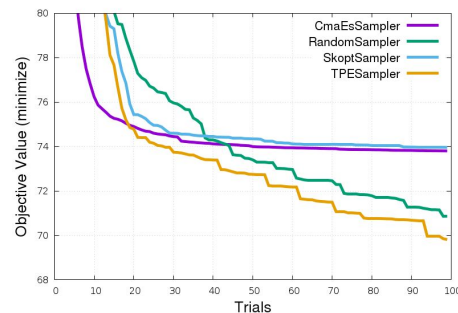
CmaEsSamplerが強い

Michalewicz Function



TPESamplerが強い

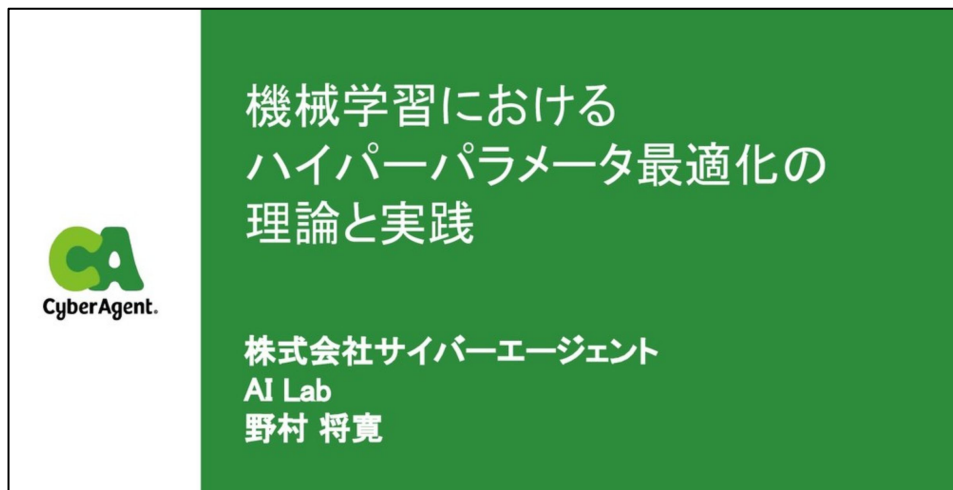
RosenbrockModified Function



関数グラフはhttp://infinity77.net/global_optimization/genindex.htmlより引用

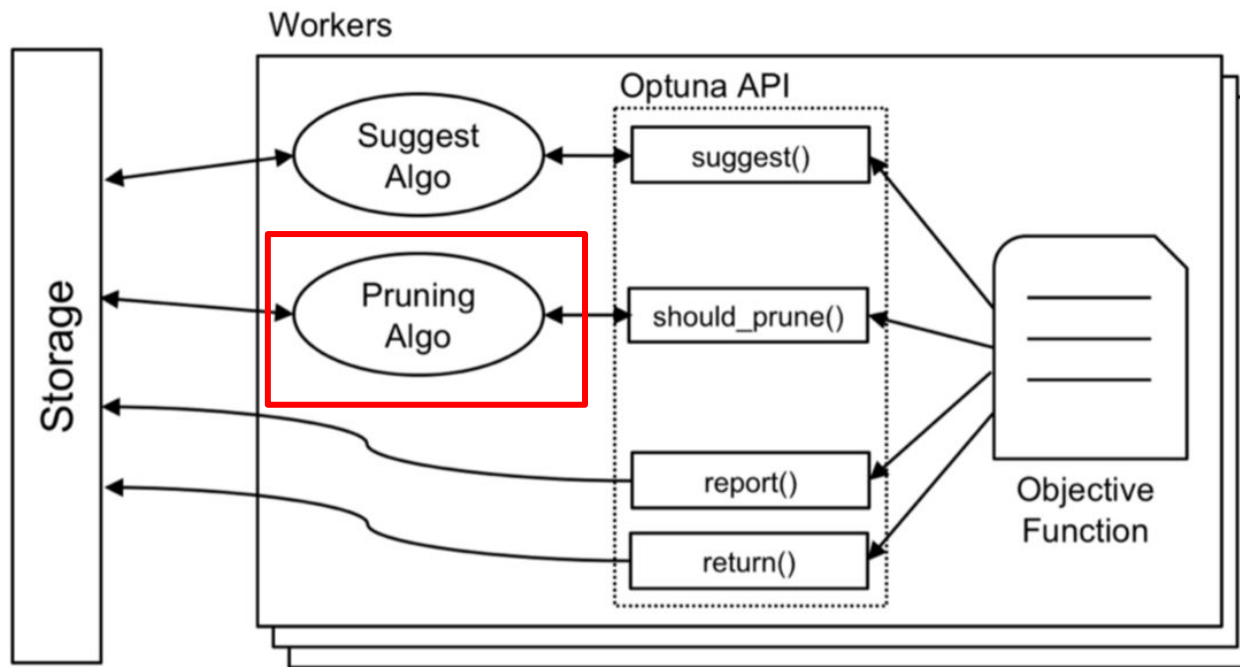
Samplerの選択指針は？ アルゴリズムの詳細は？

- 時間の関係上、今回は割愛 🙇
- PyConJP 2019での野村さんの発表が参考になります

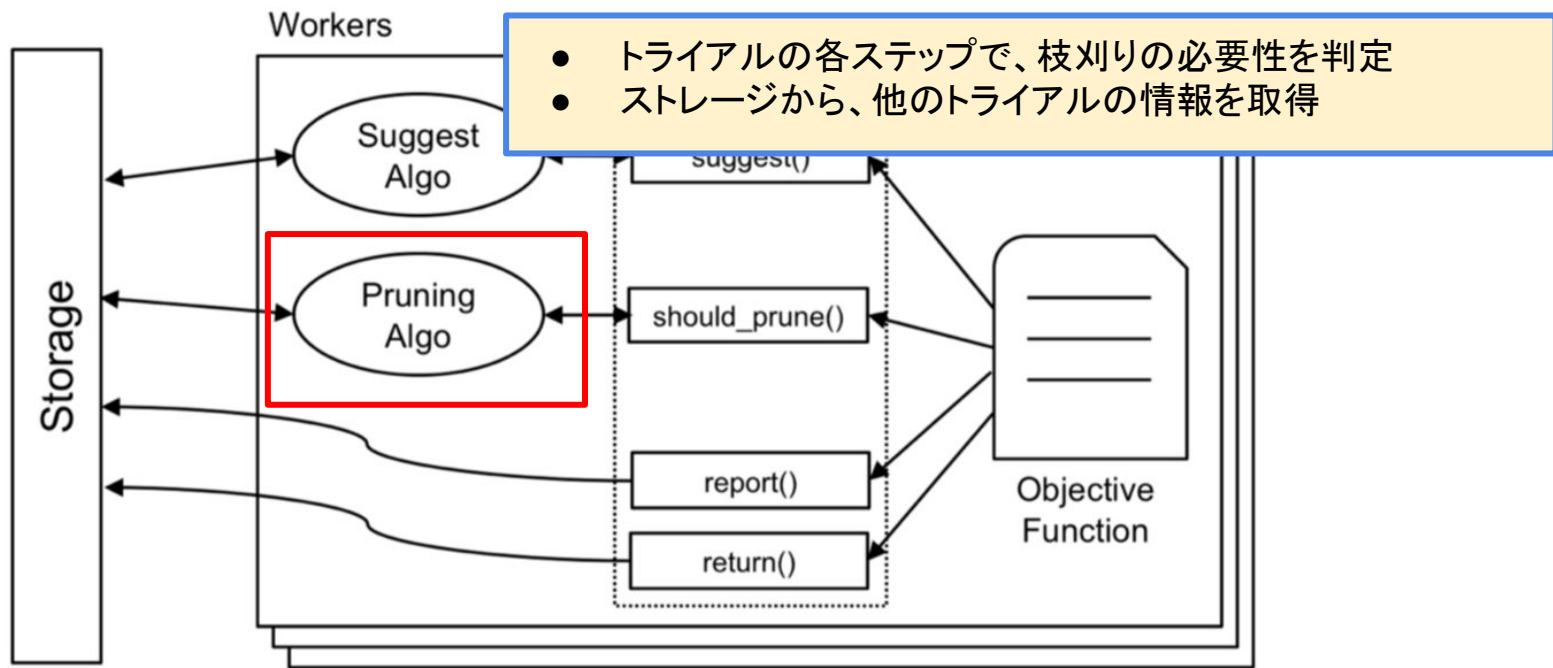


<https://speakerdeck.com/nmasahiro/ji-jie-xue-xi-niokeru-haihaharametazui-shi-hua-falseli-lun-toshi-jian>

Optunaの構成要素: Pruner



Optunaの構成要素: Pruner



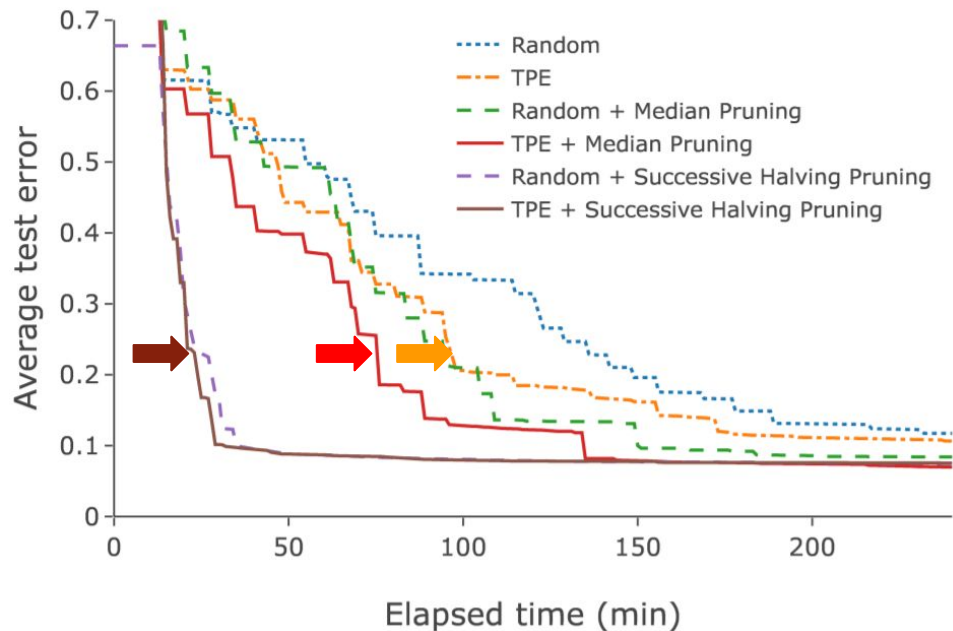
利用可能なPruner

- MedianPruner (デフォルト)
- PercentilePruner ※ MedianPrunerの一般化
- SuccessiveHalvingPruner
- ユーザ定義Pruner

指定方法

```
pruner = optuna.pruners.SuccessiveHalvingPruner(min_resource=100)
study = optuna.create_study(pruner=pruner)
```

ベンチマーク結果



Hyperparameter	Scale	Min	Max
<i>Learning Parameters</i>			
Initial Learning Rate	log	$5 * 10^{-5}$	5
Conv1 L_2 Penalty	log	$5 * 10^{-5}$	5
Conv2 L_2 Penalty	log	$5 * 10^{-5}$	5
Conv3 L_2 Penalty	log	$5 * 10^{-5}$	5
FC4 L_2 Penalty	log	$5 * 10^{-3}$	500
Learning Rate Reductions	integer	0	3
<i>Local Response Normalization</i>			
Scale	log	$5 * 10^{-6}$	5
Power	linear	0.01	3

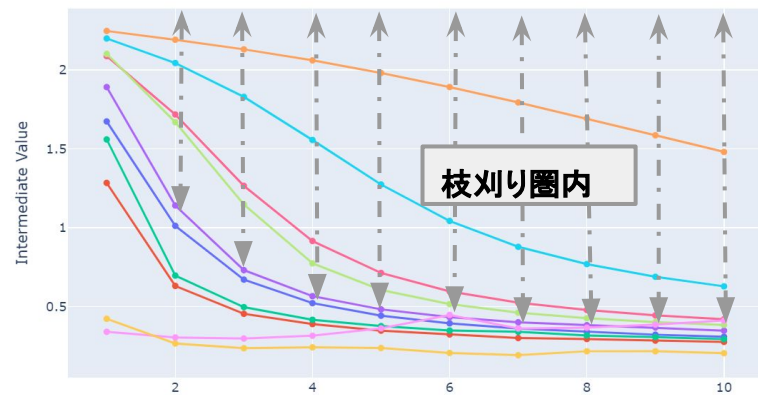
ハイパーパラメータ探索空間

The transition of average test errors of simplified AlexNet for SVHN dataset

From: <https://arxiv.org/pdf/1907.10902.pdf> (optuna paper)

MedianPruner

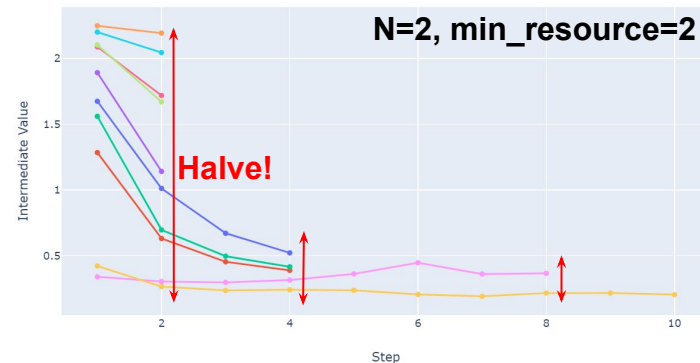
- 各ステップでの評価値(ベスト)を他のトライアルと比較
 - 中央値よりも悪いなら枝刈り
 - 実行中および枝刈り済みのトライアルは、比較対象から除外
- Pros:
 - 挙動が直感的
- Cons:
 - そこそこの枝刈り効率



Step 10個のトライアルの学習曲線

SuccessiveHalvingPruner

- 「トライアル数が $1/N$ になるように枝刈り」⇒「生き残ったものには N 倍のリソースを割り当て」の繰り返し (右上の図) ※ かなり簡略化した説明および図
- **Pros:**
 - 高い枝刈り効率
 - 分散最適化と相性が良い
 - 実行中・枝刈り済みトライアルも考慮
 - 学習曲線以外にも応用が効く
 - E.g., データサイズを変えつつ枝刈り (右下の図)
- **Cons:**
 - 現状ではハイパーパラメータに敏感
 - `min_resource`の適切な指定が必要
 - E.g., `min_resource = num_iterations/100`

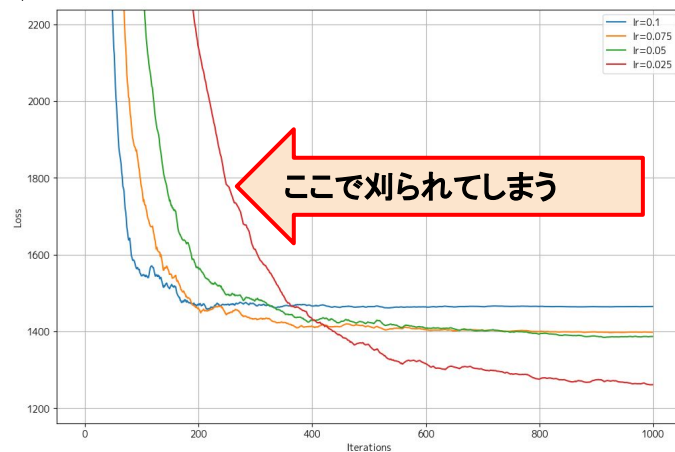


Tips: 枝刈り時のトライアル回数はどうすべきか

- n_trials を変更せずに、単純に枝刈りを有効にすると、
⇒ 最適化結果は悪くなる ※ 所要時間は短縮される
- n_trials を増やすと良い？
 - 枝刈りがあると、一回のトライアルに掛かる時間が不規則になる
⇒ 適切な n_trials の値を推測するのが難しい
- *timeout*指定がお勧め
 - 時間が許す範囲内で、出来るだけ多くのトライアルを実行
 - E.g., `study.optimize(objective, timeout=600)` # 10分間最適化

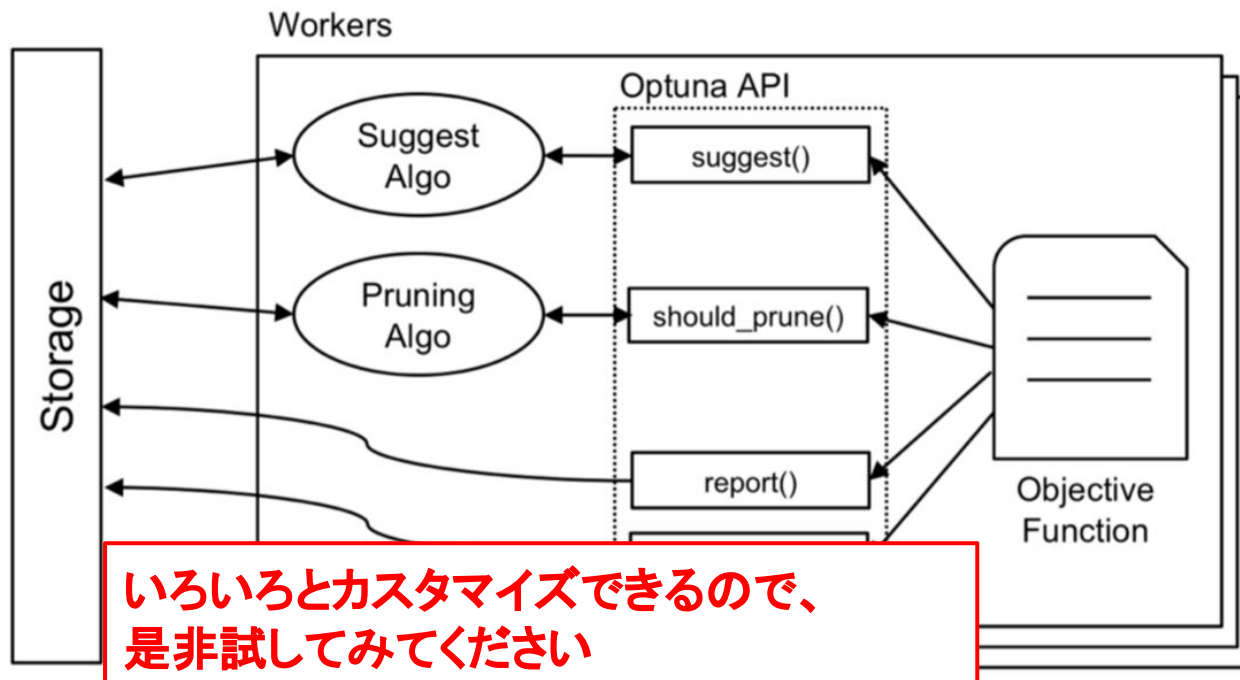
Tips: 枝刈りとlearning_rate(LR)の最適化

- 枝刈りとLR最適化の組み合わせには注意が必要
 - LRが小さいものほど、初期段階で刈られやすくなる傾向がある
- LightGBM向けの現状での対応策(一案)
 - フェーズを分ける
 - 最初は、LRを固定して、
それ以外のパラメータを枝刈りありで最適化
 - 最後は、LRだけを枝刈りなしで最適化
- 将来的にはPrunerを改良予定
 - E.g., 学習曲線予測、Hyperband



From: <https://medium.com/@tomoto/1a1c9dd870df>

ここまでのまとめ



新機能: LightGBMTuner

LightGBMTuner

- Optunaは便利
 - ハイパーパラメータを自動で探索してくれる
 - ただ、探索空間はユーザが指定する必要がある
 - ⇒ 一種のハイパーパラメータ
- 理想
 - ユーザに何も意識させずに勝手に最適化してくれる
 - ⇒ **LightGBMTuner!**

使い方

インストール (before v0.17.0):

\$ pip install git+<https://github.com/smlly/optuna@lgbm-autotune>

LightGBM互換インタフェース:

```
import optuna.integration.lightgbm as lgb
```

修正箇所は一行

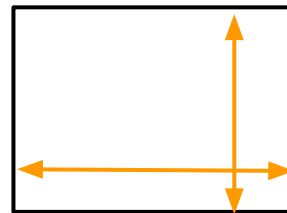
```
lgb.train(..)
```

Stepwise Tuning

- 重要なハイパーパラメータを、順々にOptunaで最適化
 - E.g., feature_fraction, num_leaves, bagging_fraction
- Stepwise Tuning:
 - 一度に全パラメータを探索すると空間が巨大になってしまう
 - 一つずつ探索することで効率化
 - 経験的に良いチューニング手法であることが知られている:
 - [CPMP's talk in Kaggle Days Paris](#)
 - [Chang's comment in Kaggle forum](#)



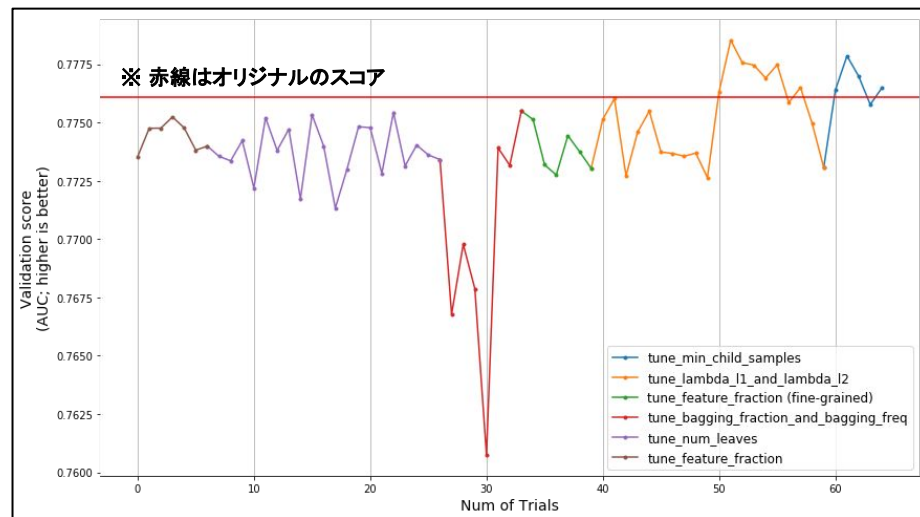
一度に探索: $X*Y$



Stepwise: $X+Y$

ベンチマーク結果

- Kaggle: [DonorsChoose.org Application Screening](#)
- [LightGBM and Tf-idf Starter](#)カーネルのLightGBM部分をLightGBMTunerに置換
 - See: <https://gist.github.com/smly/367c53e855cdadeea35736f32876b7416>
- オリジナルスコア:
 - Validation: 0.77910
 - Private: 0.78470
 - Public: 0.79516
- Tunedスコア:
 - Validation: 0.77993 (0.00083 up)
 - Private: 0.78622 (0.00152 up)
 - Public: 0.79535 (0.00019 up)



Stepwise Tuningの経過 (from: <https://github.com/pfnet/optuna/pull/549>)

※ チューニング中(右の図)はLRを高めに設定しているため、左のスコアの値とは若干ズレがある

LightGBMTuner独自のlgb.train()引数

- best_params:
 - ベストパラメータを格納
- tuning_history:
 - チューニング履歴を格納
- time_budget:
 - 制限時間を指定
- sample_size:
 - サンプルングを有効化

絶賛開発中！

- 検討中の機能:
 - 枝刈り
 - lgb.cv()対応
 - レジューム対応
- フィードバックは大歓迎！
 - <https://github.com/pfnet/optuna/pull/549>

まとめ

- Optunaでハイパーパラメータが簡単に最適化可能
- 特徴:
 - Define-by-Run
 - 枝刈り
 - 分散最適化
- Storage/Sampler/Prunerは用途に応じてカスタマイズ可能
- LightGBMTunerがもうすぐリリース

Need Your Feedback and Contribution!

pfnet / optuna

Used by 102 Unwatch 121 Unstar 1,210 Fork 113


<> Code Issues 26 Pull requests 16 Actions Projects 0 Wiki Security Insights Settings

Branch: master optuna / README.md Find file Copy path

smyly Add link to KDD paper in README 97a299a 27 days ago

6 contributors

99 lines (63 sloc) 3.57 KB Raw Blame History



OPTUNA

Optuna: A hyperparameter optimization framework

pyl v0.16.0 license MIT PASSED docs passing codecov 90%