

# Optuna Meetup #1

## Hydra, MLflow, Optunaの組み合わせで 手軽に始めるハイパーパラメータ管理

東大院・情報理工 修士課程 2 年  
中村泰貴（なかむら たいき）

# 自己紹介

## プロフィール

- 東大大学院 / 情報理工 / 猿渡・小山研 / M2
- 音声合成 & 声質変換技術を普段研究しています
- Twitter: @supikiti (アイコン: 右図)



## 登壇理由

- 以前 [medium.com/optuna](https://medium.com/optuna) にて本発表と同じ題目にて執筆
- [Easy Hyperparameter Management with Hydra, MLflow, and Optuna](#)
- [Hydra, MLflow, Optunaの組み合わせで手軽に始めるハイパーパラメータ管理](#)

# 本日の内容

## 紹介すること

- Hydra + MLflow + Optuna を用いた効率的なハイパラ管理
- それぞれの単体での使い方 & 組み合わせた具体的な使い方

## 紹介しないこと

- それぞれのライブラリの仕組み & 内部動作等
  - 素人なので…

本日の共有資料は zoom のチャット欄に掲載

[サンプルコードはこちら](#)

ハイパラ管理

# ハイパラ管理の方法

## Argparse による管理

```
parser = argparse.ArgumentParser(description='WaveNet e
parser.add_argument('--batch_size', type=int, default=B
    help='How many wav files to process
parser.add_argument('--data_dir', type=str, default=DATA
    help='The directory containing the '
parser.add_argument('--store_metadata', type=bool, defa
    help='Whether to store advanced deb
    '(execution time, memory consumptio
    'TensorBoard. Default: ' + str(META
parser.add_argument('--logdir', type=str, default=None,
    help='Directory in which to store t
    'information for TensorBoard. '
    'If the model already exists, it wi
    'the state and will continue traini
    'Cannot use with --logdir_root and
parser.add_argument('--logdir_root', type=str, default=
    help='Root directory to place the l
    'output and generated model. These
    'under the dated subdirectory of --
    'Cannot use with --logdir.')
parser.add_argument('--restore_from', type=str, default
    help='Directory in which to restore
    'This creates the new model under t
    'in --logdir_root. '
    'Cannot use with --logdir')
```

## hparam.py 等による管理

```
vocab_size = 1024
N = 6
Head = 2
d_model = 384
duration_predictor_filter_size = 256
duration_predictor_kernel_size = 3
dropout = 0.1

word_vec_dim = 384
encoder_n_layer = 6
encoder_head = 2
encoder_conv1d_filter_size = 1536
max_sep_len = 4096
encoder_output_size = 384
decoder_n_layer = 6
decoder_head = 2
decoder_conv1d_filter_size = 1536
decoder_output_size = 384
fft_conv1d_kernel = 3
fft_conv1d_padding = 1
duration_predictor_filter_size = 256
duration_predictor_kernel_size = 3
dropout = 0.1
```

# ハイパラ管理の問題点

## argparseによるハイパラ管理

- 往々にして設定するハイパラ数が膨大になりがち
- どのハイパラが model / preprocess などに対応するか見づらい

## 設定ファイルを用いたハイパラ管理

- ハイパラ変更のたびに設定ファイルを編集あるいは作成

## Hydra + MLflow + Optunaによるハイパラ管理

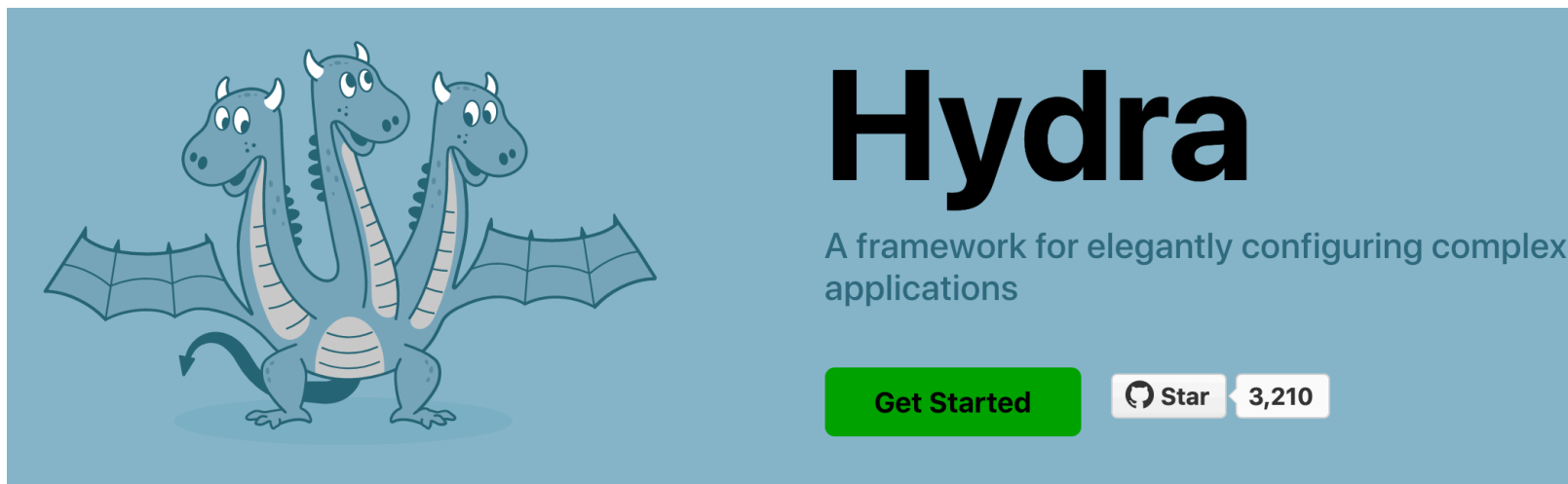
- 設定パラメータをコマンドラインから数種類変更 & 実行可能
- ハイパラのグリッドサーチが容易に可能 & 数種類の管理が容易
- 強力なハイパラ探索をコマンドラインから利用可能

Hydra

# Hydraについて

## 特徴

- Facebook AI Research が公開しているパラメータ管理ツール
- パラメータを階層立てて構造的に YAML ファイルに記述
- コマンドラインから設定値を上書き & 実行
- ハイパラの数種類のグリッドサーチを 1行で実行可能



<https://hydra.cc/>



# 基本的な使い方

- 管理したいハイパラを yaml 形式で Config ファイルへ記述

```
model:
  node1: 128
  node2: 64

optimizer:
  lr: 0.001
  momentum: 0.9
```

# 基本的な使い方

- 関数にデコレータを渡すことで関数内からハイパラへ参照可能

```
import hydra
from omegaconf import DictConfig

@hydra.main(config_path='config.yaml')
def main(cfg: DictConfig) -> None:
    print(cfg.model.node1) # 128
    print(cfg.optimizer.lr) # 0.001
```

# 基本的な使い方

## コマンドラインからの値の変更 & 実行

- ハイパラの値を調整して再実行したい場合コマンドラインからハイパラの値を直接変更してプログラムを再実行可能

```
python train.py
```

```
# cfg.model.node1 = 128
```

```
python train.py model.node1=64
```

```
# cfg.model.node1 = 64
```

# 基本的な使い方

## ハイパラのグリッドサーチ

- 数種類のハイパラを順に用いて実行したい場合, Config を書き換えることなくコマンドラインから直接指定可能
- 指定したハイパラの組み合わせの数に応じて実行

```
python train.py --multirun node1=128,256 node2=16,32
```

```
#0: node1=128 node2=16
```

```
#1: node1=128 node2=32
```

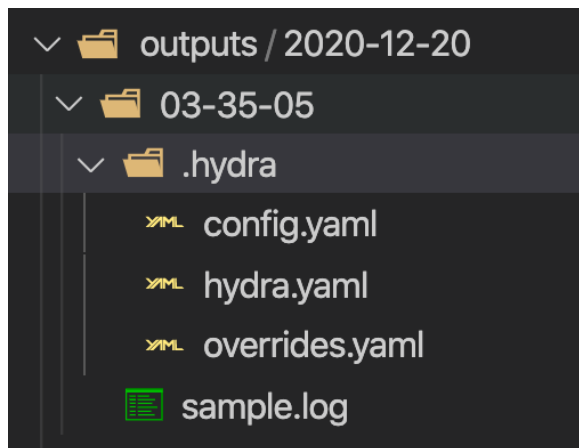
```
#2: node1=256 node2=16
```

```
#3: node1=256 node2=32
```

# 基本的な使い方

## Hydra のハイパラ自動保存機能

- 実行ごとの設定値や実行中の出力等を自動で保存される機能
- Outputs ディレクトリに実行日/実行毎/種々の設定ファイルが自動保存
- 保存形式の可読性が低くハイパラ比較ができない（左図）ため MLflow を導入しこれを改善（右図）



		Parameters <							
<input type="checkbox"/>	Start Time	Run Name	mlflow.runnam	model.node1	model.node2	optimizer.lr	optimizer.mom	test.batch_size	train.batch_size
<input type="checkbox"/>	2020-12-21 08:35:27	-	sample	395	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:34:49	-	sample	489	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:34:23	-	sample	193	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:33:56	-	sample	497	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:33:30	-	sample	498	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:33:06	-	sample	42	64	0.01	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:32:41	-	sample	57	64	0.001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:32:14	-	sample	123	64	0.1	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:31:38	-	sample	235	64	0.001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:31:06	-	sample	67	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:30:32	-	sample	116	64	0.0001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:30:02	-	sample	27	64	0.001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:29:27	-	sample	108	64	0.1	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:28:55	-	sample	332	64	0.001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:28:23	-	sample	375	64	0.001	0.9	64	64
<input type="checkbox"/>	2020-12-21 08:27:20	-	sample	375	64	0	0.9	64	64

MLflow

# MLflowの基本的な使い方

## MLflow とは

- 機械学習ライフサイクルを実現するオープンソース
- Hydra と組み合わせハイパラの管理 & 保存 & 比較が容易に
- 本発表では MLflow Tracking を用いたハイパラ管理を紹介

## MLflow Tracking

- 機械学習などのハイパラのロギング & lossやaccuracy などのメトリクス & 出力ファイルなどの管理を補助する API を提供
- pip でインストール可能

```
pip install mlflow
```

# 基本的な使い方

## ハイパラの追跡と記録

- MLflow が提供するロギング関数を用いてハイパラを記録可能

```
import mlflow

# start new run
with mlflow.start_run():

    # log single key-value param
    mlflow.log_param("param1", 5)

    # log single key-value metric
    mlflow.log_metric("foo", 2, step=1)
    mlflow.log_metric("foo", 4, step=2)
    mlflow.log_metric("foo", 6, step=3)

    with open("output.txt", "w") as f:
        f.write("Hello world!")

# logs local file or directory as artifact,
mlflow.log_artifact("output.txt")
```

start\_run()  
runID の発行

log\_param()  
ハイパラの登録

log\_metric()  
メトリックの記録

log\_artifact()  
出力されたファイル等の記録

...

mlflow ui  
localhost:5000でGUI



```
@hydra.main(config_path='config.yaml')
```

```
def main(cfg):
```

```
    model = SAMPLE_DNN(cfg)
```

```
    criterion = nn.CrossEntropyLoss()
```

```
    optimizer = optim.SGD(model.parameters(), lr=cfg.optimizer.lr,  
                           momentum=cfg.optimizer.momentum)
```

```
    mlflow.set_tracking_uri('file://' + utils.get_original_cwd() + '/mlruns')
```

```
    mlflow.set_experiment(cfg.mlflow.runname)
```

```
    with mlflow.start_run():
```

```
        for epoch in range(cfg.train.epoch):
```

```
            running_loss = 0.0
```

```
            log_params_from_omegaconf_dict(cfg)
```

```
            for i, (x, y) in enumerate(trainloader):
```

```
                optimizer.zero_grad()
```

```
                outputs = model(x)
```

```
                loss = criterion(outputs, y)
```

```
                loss.backward()
```

```
                optimizer.step()
```

```
                running_loss += loss.item()
```

```
            mlflow.log_metric("loss", running_loss)
```

## Hydra と MLflow の実装例

```
@hydra.main(config_path='config.yaml')
def main(cfg):
```

## hydra.mainデコレータ

```
    model = SAMPLE_DNN(cfg)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=cfg.optimizer.lr,
                           momentum=cfg.optimizer.momentum)
```

```
mlflow.set_tracking_uri('file://' + utils.get_original_cwd() + '/mlruns')
mlflow.set_experiment(cfg.mlflow.runname)
with mlflow.start_run():
```

```
    for epoch in range(cfg.train.epoch):
        running_loss = 0.0
```

```
        log_params_from_omegaconf_dict(cfg)
```

```
        for i, (x, y) in enumerate(trainloader):
            optimizer.zero_grad()
```

```
            outputs = model(x)
            loss = criterion(outputs, y)
            loss.backward()
            optimizer.step()
```

```
            running_loss += loss.item()
```

```
mlflow.log_metric("loss", running_loss)
```

パラメータの登録  
(log\_params())

メトリックの登録

Experiments



Search Experiments

Default



sample



sample

Experiment ID: 1

Artifact Location: file:///Users/nakamurataiki/Desktop/-Optuna\_and\_Hydra/mlruns/1

# 全試行過程が自動保存 & 比較可能

▼ Notes

None

Search Runs: metrics.rmse &lt; 1 and params.model = "tree" and tags.mlflow.source.type = "LOCAL"



State:

Active ▼

Search

Clear

Showing 4 matching runs









Compare

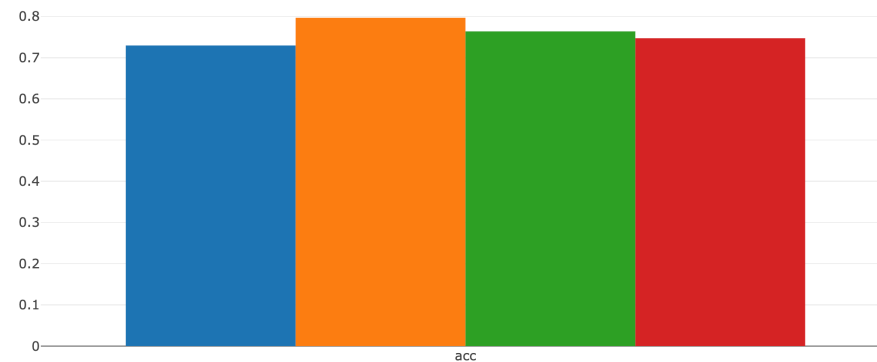
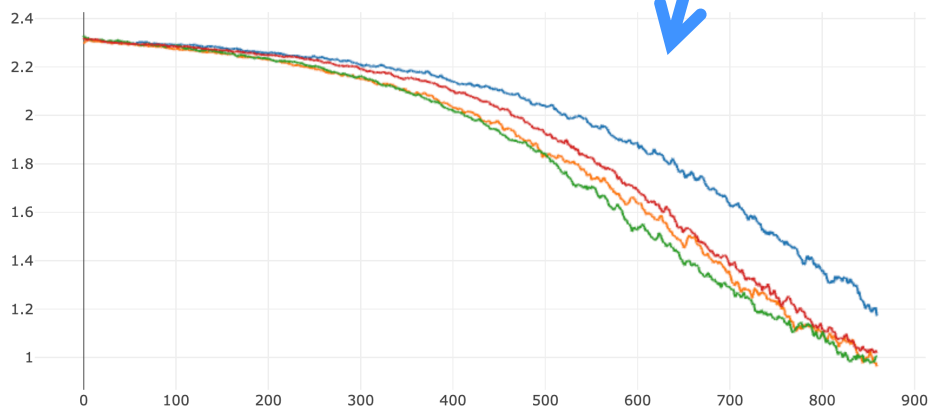
Delete

Download CSV



Columns

						Parameters >			Metrics	
<input type="checkbox"/>	Start Time	Run Name	User	Source	Version	mlflow.runnam	model.node1	model.node2	acc	loss
<input type="checkbox"/>	 2020-12-21 06:50:01	-	nakamurat...	 train.py	-	sample	256	32	0.745	1543.5
<input type="checkbox"/>	 2020-12-21 06:49:13	-	nakamurat...	 train.py	-	sample	256	64	0.785	1519.6
<input type="checkbox"/>	 2020-12-21 06:48:44	-	nakamurat...	 train.py	-	sample	128	32	0.76	1564.9
<input type="checkbox"/>	 2020-12-21 06:48:03	-	nakamurat...	 train.py	-	sample	128	64	0.801	1541.4



accuracy

# Hydra + MLflow まとめ

## Hydra

- Facebook AI Research が公開しているパラメータ管理ツール
- コマンドラインから設定値を複数変更 & 実行可能

## MLflow

- 機械学習の実験管理を自動で行うツール
- GUI 上でパラメータの違いによる結果の比較が容易に可能

## Hydra + MLflow + Optuna

- Hydra のプラグインを利用した Optuna の導入方法を解説

Optuna

# Optuna の導入

## Optuna とは

- オープンソースのハイパラ自動最適化フレームワーク
- ハイパラの値に関する試行錯誤を自動化
- 優れた性能を発揮するハイパラの値を自動的に発見

## Hydra + MLflow + Optuna

- Hydra の Optuna Sweeper プラグインを使用することで Hydra で設定した変数の探索をコマンドラインから実施可能
- 最適化される変数と条件をコマンドラインから変更可能

```
pip install hydra-optuna-sweeper --upgrade
```

# 設定ファイルへの追加点

```
defaults:  
  - hydra/sweeper: optuna  
  
hydra:  
  sweeper:  
    optuna_config:  
      direction: maximize  
      study_name: mnist  
      storage: null  
      n_trials: 20  
      n_jobs: 1  
      sampler: tpe  
      seed: 123
```

Hydraの設定ファイルへOptuna に関する具体的な設定項目を追加で書きこむだけ

細かな Optuna の設定項目を変更可能

- direction: 評価関数を最小化 or 最大化
- n\_trials: 探索回数の設定
- N\_jobs: 並列ワーカーの数
- ...

# main関数での変更点

- デコレータで渡した関数の返り値を最適化する目的変数に設定
- 以下のコードは accuracy を最大化したい場合の例

```
@hydra.main(config_path='config.yaml')
def main(cfg):
    # ...

    with mlflow.start_run():
        # ...

        accuracy = float(correct / total)
        mlflow.log_metric("acc", accuracy, step=epoch)

    return accuracy
```



# 探索範囲の指定

- コマンドラインから最適化したい変数および範囲を直接指定
- 以下では optimizer の学習率とモデルのノード数を探索
  - choice はカテゴリ型の変数へ変換されるため optimizer の学習率は 4 種類探索される
  - range は整数型の変数へ変換されるため model.node1 が [10, 500] の範囲かつ整数の条件で探索される

```
python train.py --multirun 'optimizer.lr=choice(0.1, 0.01, 0.001, 0.0001)' 'model.node1=range(10, 500)'
```

- 他にも様々な分布に対応しているがここでは割愛

# MLflowでの可視化

- Optuna で探索されたハイパラの値および目的関数の値を MLflow で構築したローカルサーバー上で確認可能

			Parameters <							
<input type="checkbox"/>	Start Time	Run Name	mlflow.runnam	model.node1	model.node2	optimizer.lr	optimizer.mom	test.batch_size	train.batch_size	train.ep
<input type="checkbox"/>	✓ 2020-12-21 08:35:27	-	sample	395	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:34:49	-	sample	489	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:34:23	-	sample	193	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:33:56	-	sample	497	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:33:30	-	sample	498	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:33:06	-	sample	42	64	0.01	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:32:41	-	sample	57	64	0.001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:32:14	-	sample	123	64	0.1	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:31:38	-	sample	235	64	0.001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:31:06	-	sample	67	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:30:32	-	sample	116	64	0.0001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:30:02	-	sample	27	64	0.001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:29:27	-	sample	108	64	0.1	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:28:55	-	sample	332	64	0.001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:28:23	-	sample	375	64	0.001	0.9	64	64	1
<input type="checkbox"/>	✓ 2020-12-21 08:27:20	-	sample	375	64	0	0.9	64	64	1

# 総まとめ

## Hydra + MLflow + Optuna

- 学習時に煩雑になりがちなパラメータ管理の決定版
- Hydra と Optuna でパラメータを容易に変更・探索し  
MLflow で全パラメータを一元管理

## さらに学びたい方には

- Kedro: Workflow のパイプライン管理ツール
  - Hydra + MLflow + Optuna + [Kedro](#)
  - より再現性のある使い回しを意識したコードに



Kedro

# 参考資料

- <https://cyberagent.ai/blog/research/12898/>
- <https://ymym3412.hatenablog.com/entry/2020/02/09/034644>
- <https://zerebom.hatenablog.com/#Hydra>
- <https://speakerdeck.com/chck/sok-xiao-sakushi-meteda-kikuyu-terumlops2020>