INTRODUCTION TO MLFLOW

Presentation by Lucas Massa

PPGI | 2024

Institute of Computing | UFAL

WHAT IS MLFLOW?

INTRODUCTION

MLflow, at its core, provides a suite of tools aimed at simplifying the ML workflow. It is tailored to assist ML practitioners throughout the various stages of ML development and deployment.

- Tracking;
- Model registry;
- Deployment;
- Evaluate;
- Projects.

INTRODUCTION

MLflow, at its core, provides a suite of tools aimed at simplifying the ML workflow. It is tailored to assist ML practitioners throughout the various stages of ML development and deployment.

- Tracking;
- Model registry;
- Deployment;
- Evaluate;
- Projects.

EXPERIMENT TRACKING

INSTALLATION

Get MLFlow Python Library:

- Recommended: use virtual environment;
- Install via package manager.

```
pip install mlflow
```

START SERVER

Start the server for UI and experiment tracking:

- Multiple persistent backend stores:
  - Defaults to file system;
  - SQLite was chosen;
  - Creates .db file in the project root;
  - Server listens to localhost:5000.
- Run mlflow ui CLI command:

```
mlflow ui --backend-store-uri sqlite:///mlflow.db
```

SET EXPERIMENT

Experiments are the primary unit of organization in MLflow; all MLflow runs belong to an experiment. Each experiment lets you visualize, search, and compare runs.

```
mlflow.set_tracking_uri("sqlite:///mlflow.db")
mlflow.set_experiment(f"end2end_disvae_{dataset}")
mlflow.pytorch.autolog(disable=True)
```

CREATE RUN

All MLflow runs are logged to the active experiment, which was set in the previous step. Each run has an unique ID.

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

SET TAG

Tags can be associated to the active run. Makes later filtering and searching easier.

```
mlflow.set_tag("model_name", "End2End_DisVAE")
```

LOG PARAMS

It is also possible to log used hyperparameters for the active run. Needs to be logged in dict format.

```
mlflow.log_params(config.__dict__)
```

RUN START

```
with mlflow.start_run(run_name=run_name):
    mlflow.set_tag("model_name", "End2End_DisVAE")
    mlflow.log_params(config.__dict__)
```

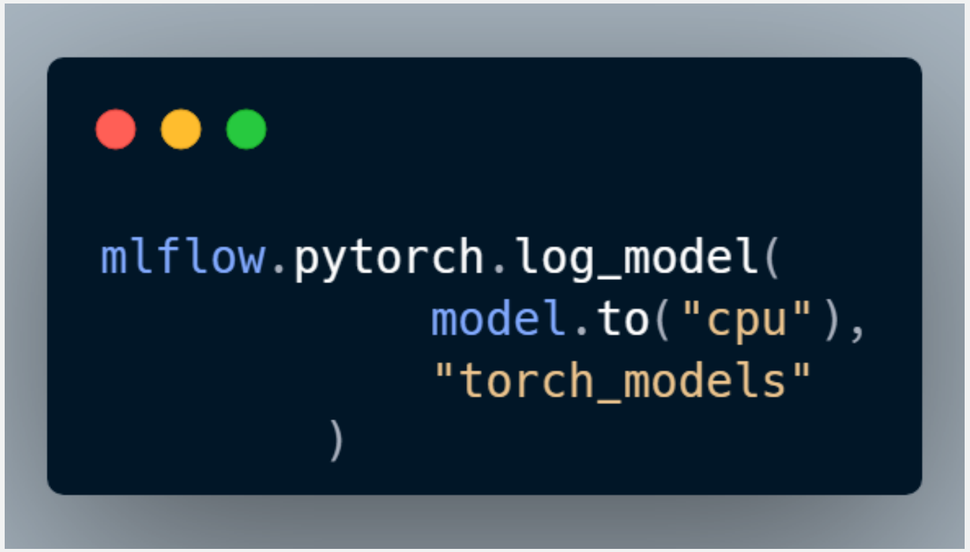
LOG METRICS

Multiple evaluation metrics can be logged during the optimization loop for the current run. A step is associated for each log.

```
mlflow.log_metrics({
        "elbo": float(loss_sum.cpu().detach().numpy()),
        "reconstruction_loss": float(rec_sum.cpu().detach().numpy()),
        "kl_divergence": float(kl_sum.cpu().detach().numpy()),
        "classification_loss": float(cls_sum.cpu().detach().numpy()),
        "correlation_loss": float(corr_sum.cpu().detach().numpy())
    }, step=epoch)
```

LOG MODEL

Finally, MLFlow API makes available interfaces to save models from multiple machine learning libraries, like sklearn, tensorflow and pytorch. Artifacts stored at ./mlruns folder.



LOAD MODEL

Logged models can be later loaded to be evaluated and be applied to tasks. Load path formed by run ID and artifact folder.

```python
import mlflow
logged_model = 'runs:/17dab5e1452247bc9410c808730d4e52/torch_models'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)
```

LOAD MODEL

Using tag to filter desired model:

```
mlflow.set_tracking_uri("sqlite:///mlflow.db")
mlflow.set_experiment(f"end2end_disvae_{dataset}")

def load_model(model_name):
    filter_str = f"tags.model_name = '{model_name}'"
    run_id = mlflow.search_runs(filter_string=filter_str)["run_id"][0]
    model_id = f'runs:/{run_id}/torch_models'
    model = mlflow.pytorch.load_model(model_id)
    return model
```

HYPERPARAMETER TUNNING

OPTUNA

Optuna is a framework agnostic automatic hyperparameter optimization software framework, particularly designed for machine learning.

- Lightweight;
- Pythonic syntax;
- Efficient algorithms;
- Easy parallelization;
- Quick visualization.

INSTALLATION

Get Optuna Python Library:

- Recommended: use virtual environment;
- Install via package manager.

```
pip install optuna
```

CREATE STUDY

Optuna uses the terms study and trial as follows:

- Study: optimization based on an objective function
- Trial: a single execution of the objective function

```
def hyperparameter_tune():

    def objective(trail):
        ...

    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=100, gc_after_trial=True,
        show_progress_bar=True)
    return study.best_params
```

OBJECTIVE

Objective function will contain optimization loop and MLFlow functionalities.

```
def objective(trail):
    experiment = mlflow.get_experiment_by_name(experiment_name)
    client = mlflow.tracking.MlflowClient()
    run = client.create_run(experiment.experiment_id)
```

OBJECTIVE

```python
with mlflow.start_run(run_id=run.info.run_id):
    params = {
        "lr": trail.suggest_float("lr", 0.00001, 0.001),
        "beta_1": 0.9,
        "beta_2": 0.999,
        "batch_size": trail.suggest_int("batch_size", batch_limits[0], batch_limits[1]),
        "epochs": trail.suggest_int("epochs", 50, 200),
        "alpha": trail.suggest_float("alpha", 0.01, 1),
        "beta": trail.suggest_float("beta", 0.01, 1),
        "gamma": trail.suggest_float("gamma", 0.01, 1),
        "num_channels": 3,
        "image_size": (128, 128),
        "latent_dim": 100,
        "num_persons": num_persons,
        "num_expressions": num_expressions
    }

    mlflow.set_tag("model_name", "End2End_DisVAE")
    mlflow.log_params(params)
```

OBJECTIVE

```python
def objective(trail):
    experiment = mlflow.get_experiment_by_name(experiment_name)
    client = mlflow.tracking.MlflowClient()
    run = client.create_run(experiment.experiment_id)

    with mlflow.start_run(run_id=run.info.run_id):
        params = {
            "lr": trail.suggest_float("lr", 0.00001, 0.001),
            "beta_1": 0.9,
            "beta_2": 0.999,
            "batch_size": trail.suggest_int("batch_size", batch_limits[0], batch_limits[1]),
            "epochs": trail.suggest_int("epochs", 50, 200),
            "alpha": trail.suggest_float("alpha", 0.01, 1),
            "beta": trail.suggest_float("beta", 0.01, 1),
            "gamma": trail.suggest_float("gamma", 0.01, 1),
            "num_channels": 3,
            "image_size": (128, 128),
            "latent_dim": 100,
            "num_persons": num_persons,
            "num_expressions": num_expressions
        }

        # mlflow log params and set tags
        # model training loop and mlflow metric logging
        # mlflow model log

        return metric_to_be_optimized
```

LOAD MODEL

Ordering runs by ascending metric value and loading best model:

```
mlflow.set_tracking_uri("sqlite:///mlflow.db")
mlflow.set_experiment(f"end2end_tune_{dataset_name}")

def load_model(target_metric="elbo"):
    runs = mlflow.search_runs(order_by=[f"metrics.{target_metric} asc"])
    run_id = runs["run_id"][0]
    model_id = f'runs:/{run_id}/torch_models'
    model = mlflow.pytorch.load_model(model_id)
    return model
```

THANK YOU

Presentation by Lucas Massa