MNIST Observability

A simple web app to manage MNIST Handwritten Digit Classification experiments.

Table of Contents

Prerequisites

In order to run this project, you need to have Redis and MongoDB servers installed.

This project was developed on Ubuntu 22.04, and has not been tested with Windows. Some Python dependencies may not work on Windows.

Getting started

Run locally

Create a Python environment then install all dependencies.

```
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Start the two Redis and MongoDB servers. Alternatively, you can start Docker containers of Redis and MongoDB.

Create an `.env` file to fill in environment variables, especially the connection strings of Redis and MongoDB.

Start the web server with the following command, and the website should be up at 127.0.0.1:8080.

```
uvicorn app.main:app
```

Open another terminal instance and start Celery worker with this command:

```
celery -A app worker --loglevel=INFO -P [threads/solo/gevent]
```

In the `-P` flag, please choose one of the three values above, which will instruct `celery` to use the specified task pool.

Use with Docker

Create a new file `docker.env` containing the necessary environment variables. The following one should provide a decent starting point:

```
MONGODB_URL=mongodb://mongodb:27017/
DB_NAME=mnist-observability
REDIS_URL=redis://default:redispw@redis:6379
MNIST_DATASET_DIR=./data
LOG_DIR=./logs
FLOWER_UNAUTHENTICATED_API=true
```

Run `docker compose` commands:

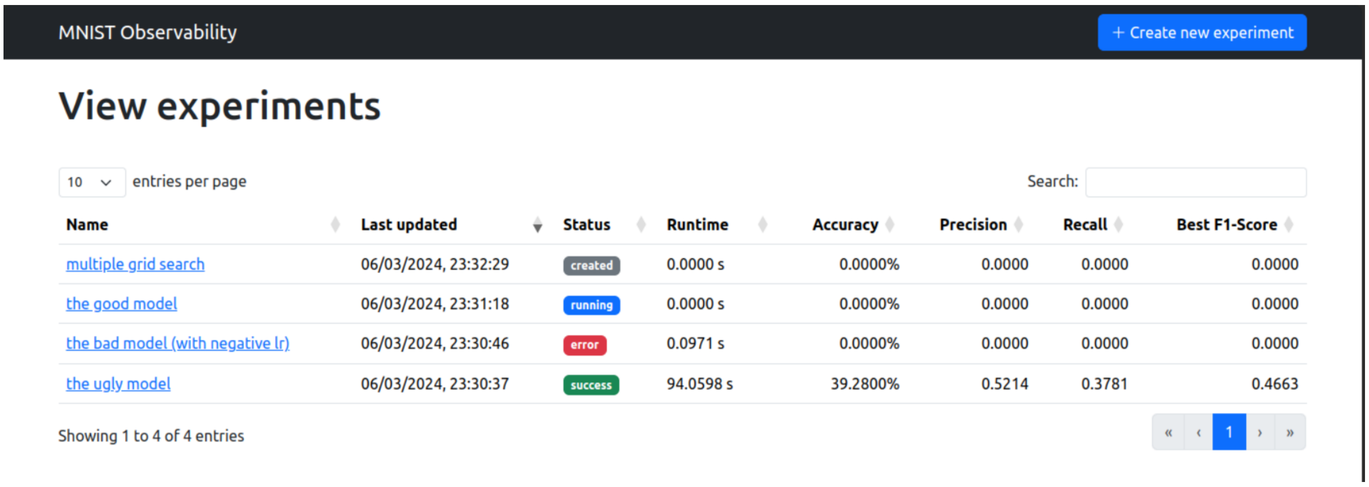
```
docker compose build
docker compose up -d
```

If you do not make any modifications to the Docker configurations, you should be able to create and start 6 Docker containers (MongoDB, Redis, web, Celery Flower, and 2 Celery workers).

- The main website should be up at localhost:8000.
- In addition, you can observe Celery workers' statuses via Flower, exposed at localhost:5555.

Features

Manage the list of experiments and sort them by criteria



View experiments

10 entries per page

Search:

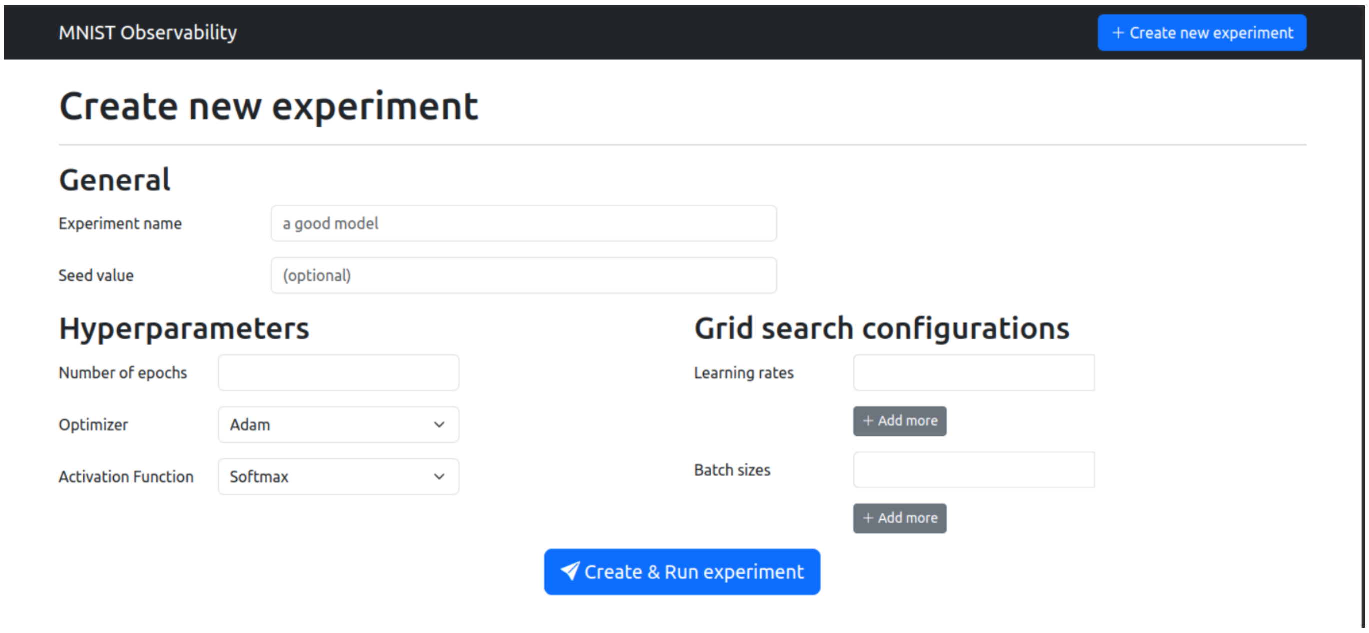
Name	Last updated	Status	Runtime	Accuracy	Precision	Recall	Best F1-Score
multiple grid search	06/03/2024, 23:32:29	created	0.0000 s	0.0000%	0.0000	0.0000	0.0000
the good model	06/03/2024, 23:31:18	running	0.0000 s	0.0000%	0.0000	0.0000	0.0000
the bad model (with negative lr)	06/03/2024, 23:30:46	error	0.0971 s	0.0000%	0.0000	0.0000	0.0000
the ugly model	06/03/2024, 23:30:37	success	94.0598 s	39.2800%	0.5214	0.3781	0.4663

Showing 1 to 4 of 4 entries

Create a new experiment with hyperparameters - and even grid-search on learning rates & batch sizes

Customize every experiment by adjusting hyperparameters: number of epochs, optimizer, activation function.

Have multiple learning rates or batch sizes to try? Input them and click once to run all combinations!



Create new experiment

General

Experiment name	a good model
Seed value	(optional)

Hyperparameters

Number of epochs	
Optimizer	Adam
Activation Function	Softmax

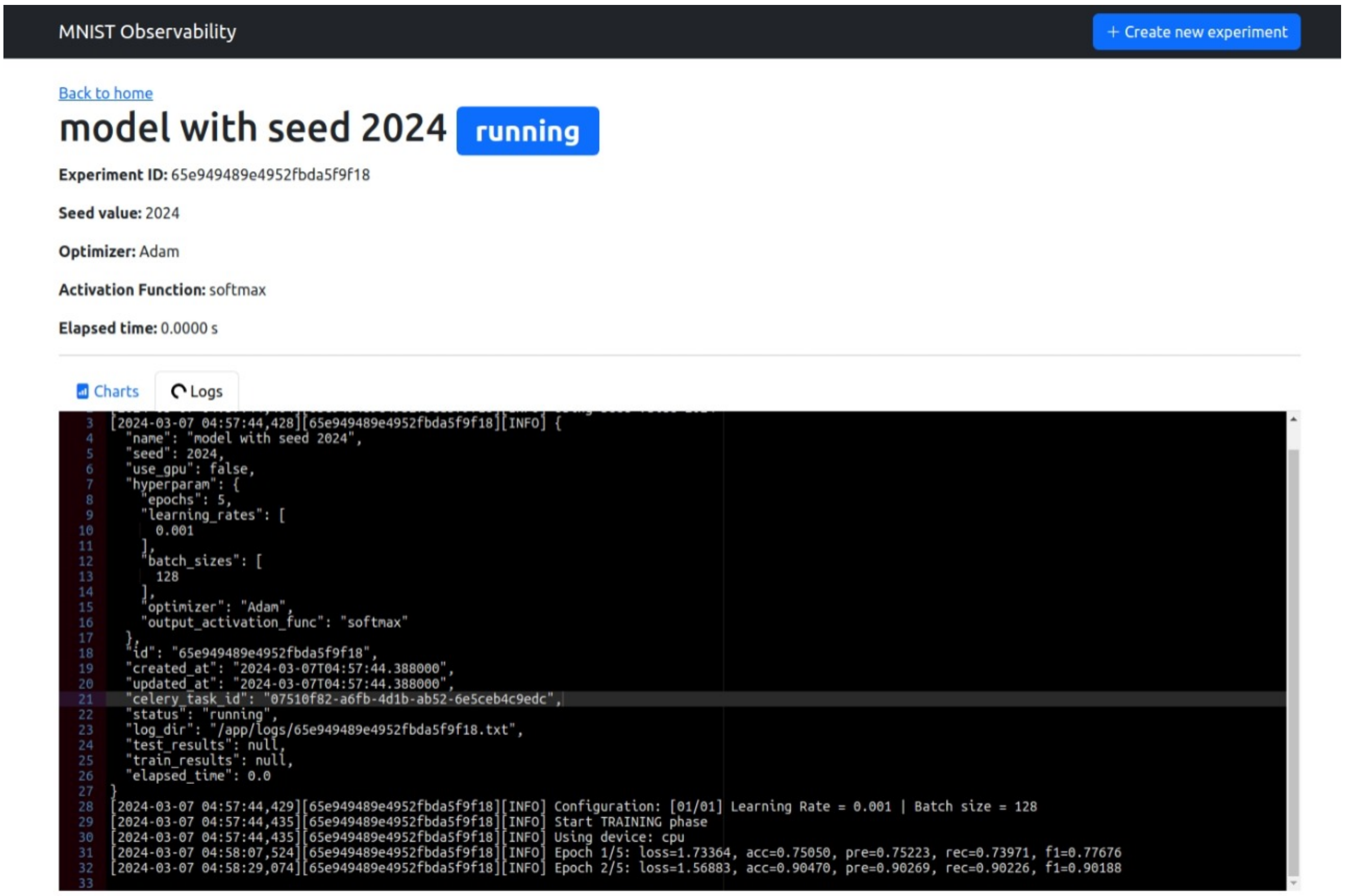
Grid search configurations

Learning rates	
Batch sizes	

You can even make your experiment reproducible with seed - every experiment with the same settings and seed value will behave the same!

Monitor the status of experiment with live log streaming

Observe the experiment with live log streaming.



model with seed 2024 running

Experiment ID: 65e949489e4952fbda5f9f18

Seed value: 2024

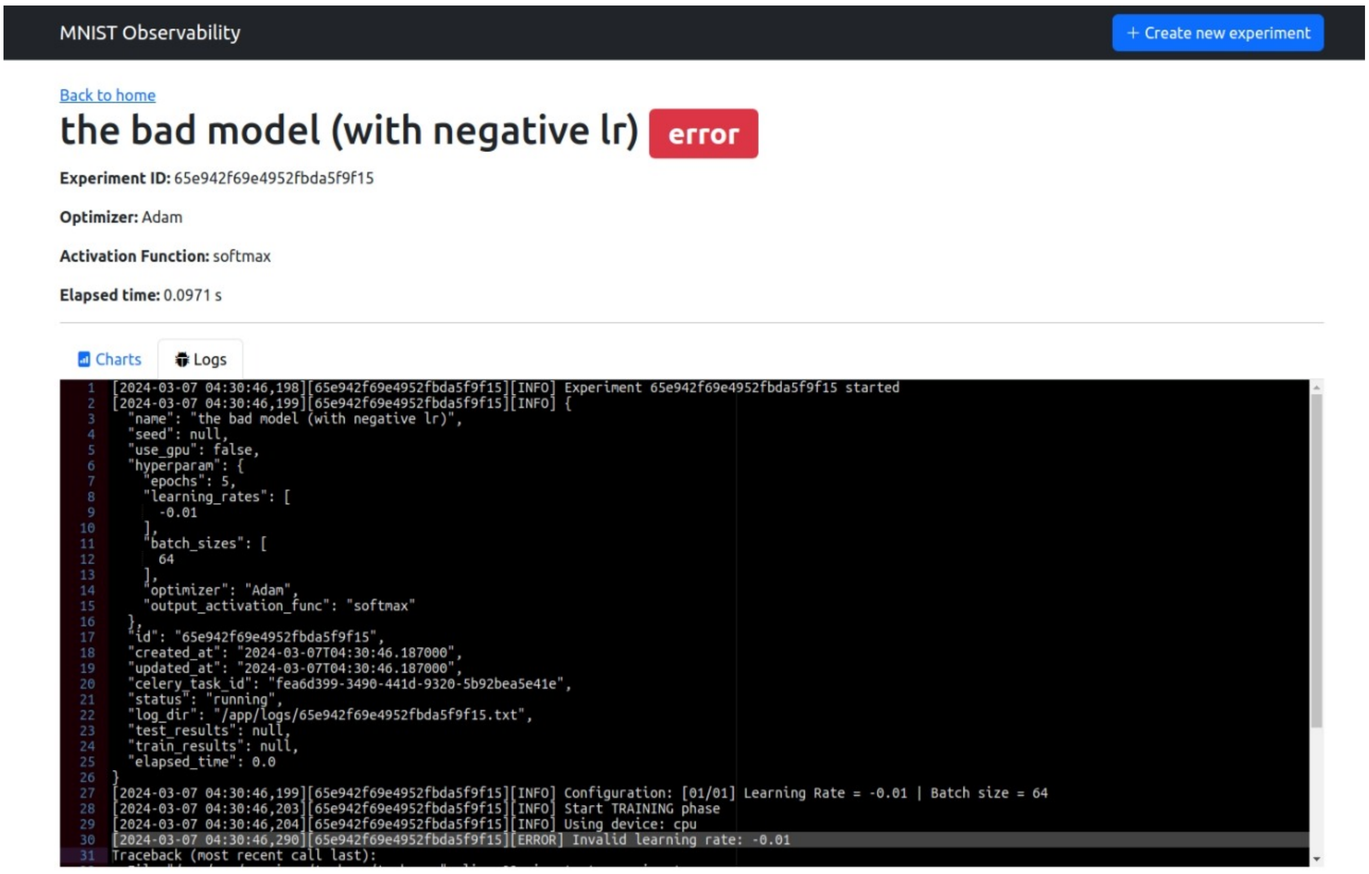
Optimizer: Adam

Activation Function: softmax

Elapsed time: 0.0000 s

```
[2024-03-07 04:57:44,428][65e949489e4952fbda5f9f18][INFO] {
  "name": "model with seed 2024",
  "seed": 2024,
  "use_gpu": false,
  "hyperparam": {
    "epochs": 5,
    "learning_rates": [
      0.001
    ],
    "batch_sizes": [
      128
    ],
    "optimizer": "Adam",
    "output_activation_func": "softmax"
  },
  "id": "65e949489e4952fbda5f9f18",
  "created_at": "2024-03-07T04:57:44.388000",
  "updated_at": "2024-03-07T04:57:44.388000",
  "celery_task_id": "07510f82-a6fb-4d1b-ab52-6e5ceb4c9edc",
  "status": "running",
  "log_dir": "/app/logs/65e949489e4952fbda5f9f18.txt",
  "test_results": null,
  "train_results": null,
  "elapsed_time": 0.0
}
[2024-03-07 04:57:44,429][65e949489e4952fbda5f9f18][INFO] Configuration: [01/01] Learning Rate = 0.001 | Batch size = 128
[2024-03-07 04:57:44,435][65e949489e4952fbda5f9f18][INFO] Start TRAINING phase
[2024-03-07 04:57:44,435][65e949489e4952fbda5f9f18][INFO] Using device: cpu
[2024-03-07 04:58:07,524][65e949489e4952fbda5f9f18][INFO] Epoch 1/5: loss=1.73364, acc=0.75050, pre=0.75223, rec=0.73971, f1=0.77676
[2024-03-07 04:58:29,074][65e949489e4952fbda5f9f18][INFO] Epoch 2/5: loss=1.56883, acc=0.90470, pre=0.90269, rec=0.90226, f1=0.90188
```

And find out the issue if your model fail!



the bad model (with negative lr) error

Experiment ID: 65e942f69e4952fbda5f9f15

Optimizer: Adam

Activation Function: softmax

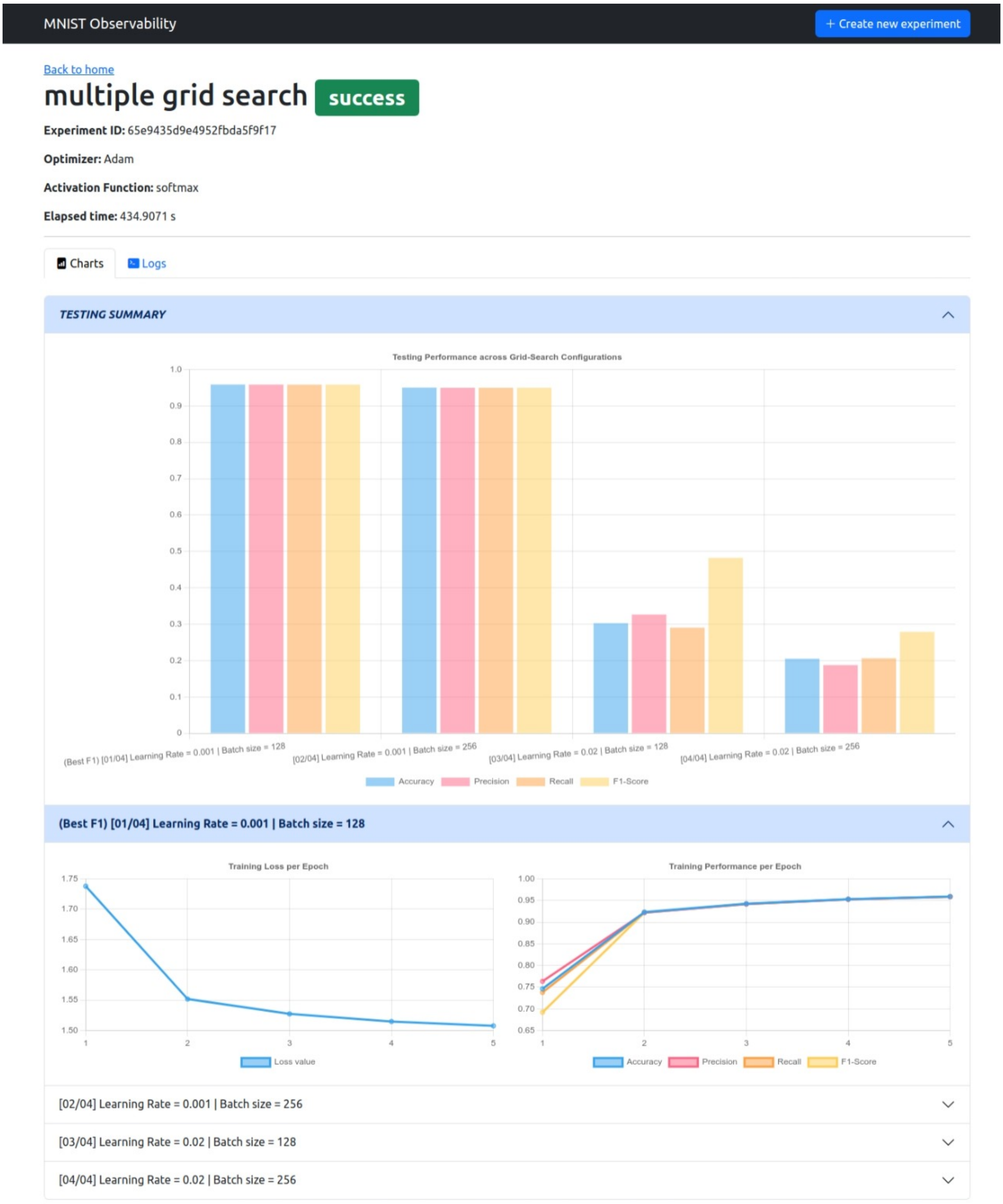
Elapsed time: 0.0971 s

```
[2024-03-07 04:30:46,198][65e942f69e4952fbda5f9f15][INFO] Experiment 65e942f69e4952fbda5f9f15 started
[2024-03-07 04:30:46,199][65e942f69e4952fbda5f9f15][INFO] {
  "name": "the bad model (with negative lr)",
  "seed": null,
  "use_gpu": false,
  "hyperparam": {
    "epochs": 5,
    "learning_rates": [
      -0.01
    ],
    "batch_sizes": [
      64
    ],
    "optimizer": "Adam",
    "output_activation_func": "softmax"
  },
  "id": "65e942f69e4952fbda5f9f15",
  "created_at": "2024-03-07T04:30:46.187000",
  "updated_at": "2024-03-07T04:30:46.187000",
  "celery_task_id": "fea6d399-3490-441d-9320-5b92bea5e41e",
  "status": "running",
  "log_dir": "/app/logs/65e942f69e4952fbda5f9f15.txt",
  "test_results": null,
  "train_results": null,
  "elapsed_time": 0.0
}
[2024-03-07 04:30:46,199][65e942f69e4952fbda5f9f15][INFO] Configuration: [01/01] Learning Rate = -0.01 | Batch size = 64
[2024-03-07 04:30:46,203][65e942f69e4952fbda5f9f15][INFO] Start TRAINING phase
[2024-03-07 04:30:46,204][65e942f69e4952fbda5f9f15][INFO] Using device: cpu
[2024-03-07 04:30:46,290][65e942f69e4952fbda5f9f15][ERROR] Invalid learning rate: -0.01
Traceback (most recent call last):
```

Wait for too long? Just turn off the browser (but not the Celery workers) and do other things. You can come back later at any time and still see the full log!

View the experiment results with rich visualization

View how the curves of metrics behave in each grid-search configuration during training, and how different grid-search combinations compare with each other in the testing phase.



MNIST Observability

+ Create new experiment

Back to home

multiple grid search success

Experiment ID: 65e9435d9e4952fbda5f9f17

Optimizer: Adam

Activation Function: softmax

Elapsed time: 434.9071 s

Charts Logs

TESTING SUMMARY

Testing Performance across Grid-Search Configurations
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
(Best F1) [01/04] Learning Rate = 0.001 | Batch size = 128
[02/04] Learning Rate = 0.001 | Batch size = 256
[03/04] Learning Rate = 0.02 | Batch size = 128
[04/04] Learning Rate = 0.02 | Batch size = 256
Accuracy
Precision
Recall
F1-Score

(Best F1) [01/04] Learning Rate = 0.001 | Batch size = 128

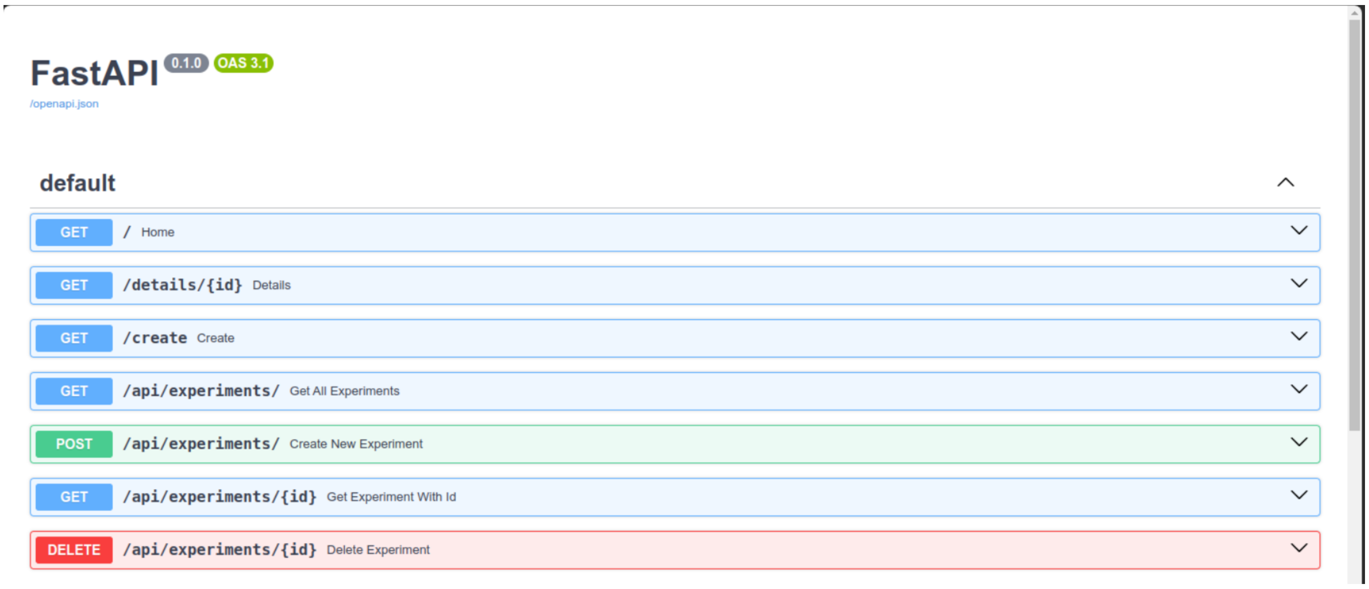
Training Loss per Epoch
1.75
1.70
1.65
1.60
1.55
1.50
1
2
3
4
5
Loss value
Training Performance per Epoch
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
1
2
3
4
5
Accuracy
Precision
Recall
F1-Score

[02/04] Learning Rate = 0.001 | Batch size = 256

[03/04] Learning Rate = 0.02 | Batch size = 128

[04/04] Learning Rate = 0.02 | Batch size = 256

Leverage more information from API endpoints


FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET / Home
GET /details/{id} Details
GET /create Create
GET /api/experiments/ Get All Experiments
POST /api/experiments/ Create New Experiment
GET /api/experiments/{id} Get Experiment With Id
DELETE /api/experiments/{id} Delete Experiment

Access the information programmatically via API endpoints. Swagger doc is available at 127.0.0.1:8080/docs or localhost:8000/docs if using Docker.

Project structure

This project uses MongoDB - a NoSQL database - to store information of the experiments, Celery as a distributed task queue to run model training and testing, and Redis as a message broker between the web application and Celery workers. The clean-looking frontend is designed with Bootstrap, with chart elements powered by Chart.js and log viewer by Ace code editor.

Below is the folder structure and some key components of this project.

```
.
├── app
│   ├── api
│   │   ├── endpoints
│   │   │   └── experiments.py
│   │   ├── pages.py
│   │   └── router.py
│   ├── celery.py
│   ├── config.py
│   ├── __init__.py
│   ├── main.py
│   ├── mnist
│   │   └── ...
│   ├── models
│   │   └── ...
│   ├── services
│   │   └── ...
│   ├── static
│   │   └── ...
│   └── templates
│       └── ...
├── docker-compose.yml
├── Dockerfile
├── README.md
├── requirements.txt
└── sample.env
```

- `app` folder: Contains the whole FastAPI application source code.
- `app/config.py`: Contains and loads the configuration of the application from environment variables.
- `app/api/endpoints/`: Contains the backend logic of API endpoints.
- `app/api/pages.py`: Handles server-side rendering of pages with templates.
- `app/mnist/`: Handles MNIST datasets and PyTorch model initialization, training and testing.
- `app/models/`: Contains the MongoDB data schemas (not to be confused with the MNIST model).
- `app/services/`: Contains the logic of the Celery worker.
- `app/static/`: Contains static content such as JavaScript/CSS files from third-party libraries.
- `app/templates/`: Contains Jinja2 HTML templates.

Limitations & Known issues

- For an experiment with more than one grid-search configurations, the homepage will select the one with **the highest F-1 Score in the testing phase** to display on the home page. This will be also designated as the best grid-search configuration in the Details page.

- The maximum number of grid-search configurations is 16. It is the number of learning rates multiplied with the number of batch sizes. Hence, you cannot add more than 16 learning rates or batch sizes on the Create page.

- The app cannot resume jobs if their Celery workers shut down unexpectedly while running (e.g. Ctrl+C the worker).

- Omitting the `-P` flag when starting Celery workers will cause them to hang during model inference. This is due to the fact that Celery uses prefork pool by default, which may not work well with PyTorch.

- The `use_gpu` option in creating new experiments is available via the backend API but not the front-facing web app, since the author does not have a GPU to test. Therefore, the default Docker settings in this project are not configured to support GPU.

  - To show the GPU option in the web app, simply uncommment this block of code in the `app/templates/create.html` file and rerun the project (or rebuild the Docker image `mnist-observability`):

```
<!-- <div class="row mb-3">
    <label for="useGPU" class="col-md-2">Use GPU</label>
    <div class="col-md-5">
        <div class="form-check">
            <input name="use_gpu" class="form-check-input"
type="checkbox" id="useGPU">
        </div>
    </div>
</div> -->
```