GUI Web Application for Machine Learning Experiments Tracking and Management

Nguyen Minh Tuan

March 11, 2024

Contents

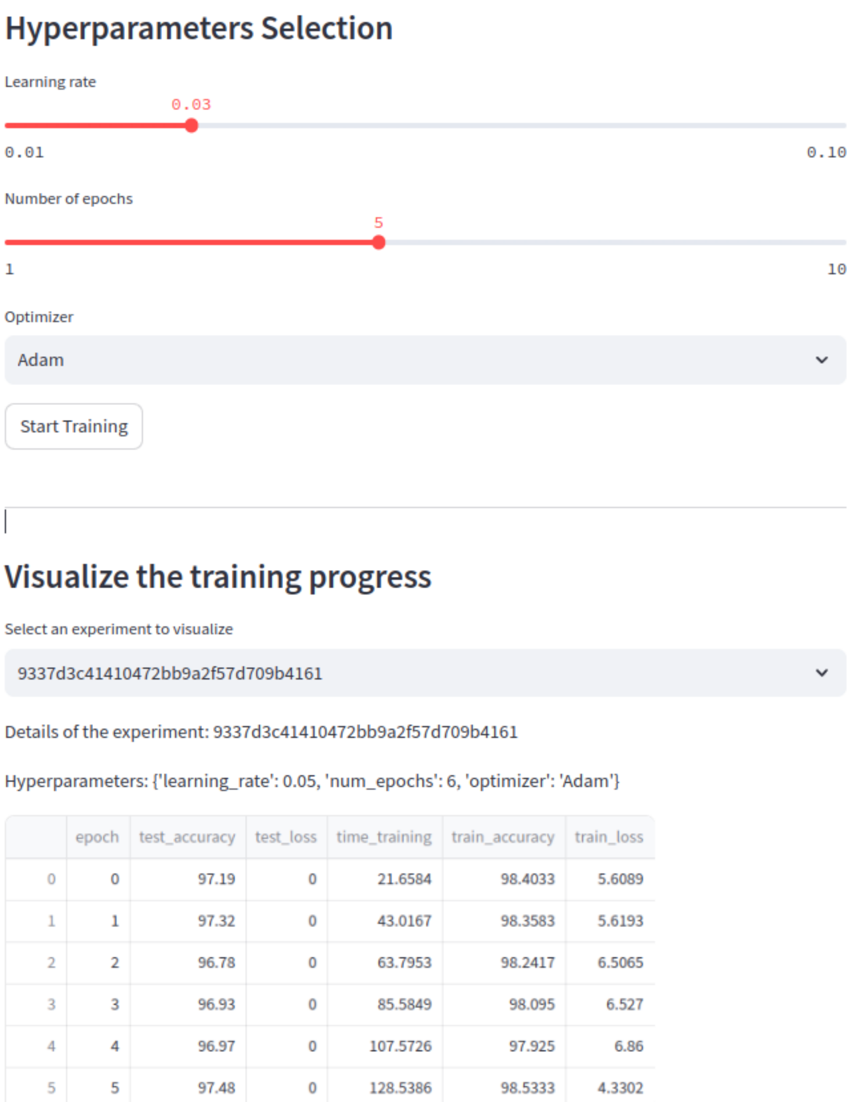
1 Introduction

This is a simple web application for managing machine learning experiments. It allows users to define hyperparameters for a machine learning model and run multiple jobs with different hyperparameters. The application displays the progress of currently running jobs and the results of all finished jobs. The experiments can be sorted by a pre-defined metric (e.g., accuracy, run time) for ease of comparison. The application also allows users to resume the UI and add new jobs. The overall architecture of the application is shown in Fig. 1.

2 Application Structure

The main components of the application are:

- **Database**: Redis, a popular in-memory database, is used to store the job status and experiment results.
- **Backend**: Flask, a Python web framework, is used to provide a RESTful API for the frontend to interact with the database and the worker.



	epoch	test_accuracy	test_loss	time_training	train_accuracy	train_loss
0	0	97.19	0	21.6584	98.4033	5.6089
1	1	97.32	0	43.0167	98.3583	5.6193
2	2	96.78	0	63.7953	98.2417	6.5065
3	3	96.93	0	85.5849	98.095	6.527
4	4	96.97	0	107.5726	97.925	6.86
5	5	97.48	0	128.5386	98.5333	4.3302

Figure 1: Overview of the application

- **Worker**: Python script, checks the database for new jobs and runs the machine learning jobs with the specified hyperparameters.
- **Frontend**: Streamlit, a Python library for creating web applications, is used to provide a user interface for users to define hyperparameters for a machine learning model and run multiple jobs with different hyperparameters.

2.1 Database

Define the class `RedisDB` at `redis_db.py`. This class provides a simple interface for interacting with the Redis database. It provides methods for adding, updating, and retrieving job status and experiment results.

The default value is `redis://localhost:6379`, and the default database is `0`. You can change this value to connect to a different Redis database.

Maintain the queue database: `queue:requests` to store the job status and experiment results.

Simple training hyperparameters format:

```
hyper_params = {
    "optimizer": "adam",
    "learning_rate": 0.001,
    "num_epochs": 5
}
```

When the client sends a request training to the server, the server will add the request to the queue database `queue:requests` with the following format:

```
data = {
  "iid": iid, # generated by server using uuid library
  "hyper_params": hyper_params,
  "status": "waiting",
  "in_update": [],
}
```

The `status` field is used to track the status of the job. The list of possible status values is `['waiting', 'running', 'done']`.

The `in_update` field is used to store the intermediate results of the job. It has the following format:

```
meta_data = {
  "train_loss": [],
  "train_accuracy": [],
  "test_loss": [],
  "test_accuracy": [],
  "time_training": [],
  "epoch": [],
}
```

2.2 Training MNIST Process

The training process is implemented in `train_mnist.py`. In this file, we define data loading, model architecture, training loop, and evaluation. The training process is implemented in the `training_and_update` function. This function takes the hyperparameters as input and returns the training and test loss and accuracy. The training process is implemented using PyTorch, a popular machine learning library for Python.

The parameters of training_and_update function are:

- train_params: the hyperparameters for the training process
- rdb: the Redis database to store the job status and experiment results
- rdb_item: the Redis database item to store the job status and experiment results

In each training epoch, the function will update the in_update field of the job status in the Redis database with the intermediate results of the job. This allows the frontend to display the progress of the currently running jobs.

2.3 Backend

The backend is built with Flask, a Python web framework. It provides a RESTful API for the frontend to interact with the database and the worker. The code for the backend is defined in backend.py. The backend provides the following endpoints:

Endpoint	Method	Data	Response	Description
/train	POST	{"hyper_params": hyper_params}	{"iid": iid}	Add a new job to the queue database with the specified hyperparameters
/status/<iid>	GET		{"iid": iid, "hyper_params": hyper_params, "status": status, "in_update": in_update}	Get the status of the job with the specified id (iid)
/experiments	GET		{"results": results}	Get all jobs and its status in the queue database

Figure 2: The list of endpoints

The app will run on http://localhost:5000 by default. You can change the port/host by modifying the app.run line in backend.py.

2.4 Worker

The worker is a simple Python script that checks the database for new jobs and runs the machine learning jobs with the specified hyperparameters. We maintain the worker in worker.py using one **While loop** to check the queue database for new jobs. If a new job is found, the worker will run the machine learning job with the specified hyperparameters and update the job status in the database with the results of the job.

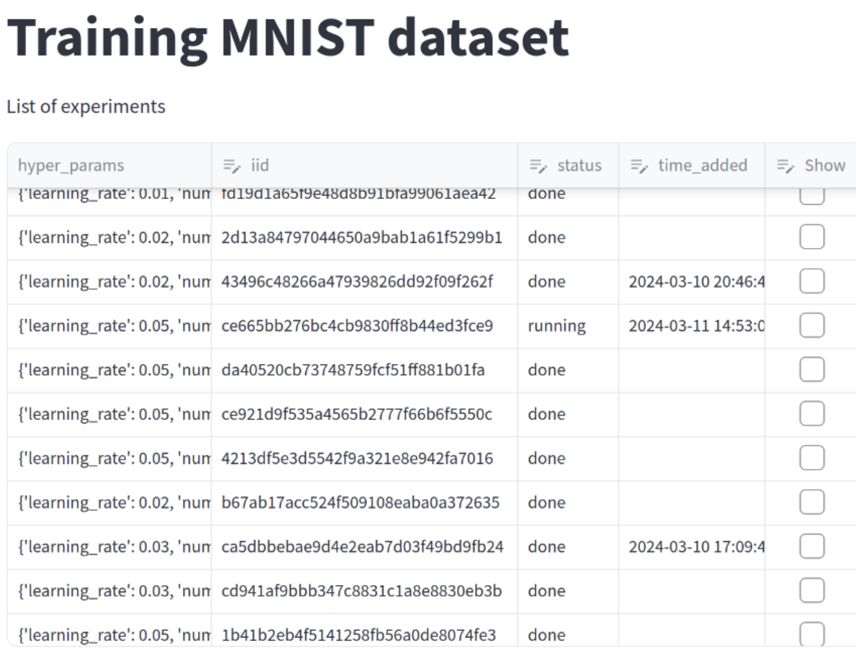
Method next_request in redis_db.py is used to get the next job from the queue database using .blpop method of Redis. The blpop method is a blocking method that can't be called a second time after the first call is finished. The worker will wait until a new job is added to the queue database.

After the worker gets a new job, it will run the function `training_and_update` to train the model with the specified hyperparameters.

2.5 Frontend

The frontend is built with Streamlit, a Python library for creating web applications. It provides a user interface for users to define hyperparameters for a machine learning model and run multiple jobs with different hyperparameters. The code for the frontend is defined in `frontend.py`. The frontend provides the following features:

- A list of all jobs and their status in the queue database, please refer to Fig. 3 for visualization.



Training MNIST dataset

List of experiments

hyper_params	iid	status	time_added	Show
{'learning_rate': 0.01, 'num	fd19d1a65f9e48d8b91bfa99061aea42	done		
{'learning_rate': 0.02, 'num	2d13a84797044650a9bab1a61f5299b1	done		
{'learning_rate': 0.02, 'num	43496c48266a47939826dd92f09f262f	done	2024-03-10 20:46:4	
{'learning_rate': 0.05, 'num	ce665bb276bc4cb9830ff8b44ed3fce9	running	2024-03-11 14:53:0	
{'learning_rate': 0.05, 'num	da40520cb73748759fcf51ff881b01fa	done		
{'learning_rate': 0.05, 'num	ce921d9f535a4565b2777f66b6f5550c	done		
{'learning_rate': 0.05, 'num	4213df5e3d5542f9a321e8e942fa7016	done		
{'learning_rate': 0.02, 'num	b67ab17acc524f509108eaba0a372635	done		
{'learning_rate': 0.03, 'num	ca5dbbebae9d4e2eab7d03f49bd9fb24	done	2024-03-10 17:09:4	
{'learning_rate': 0.03, 'num	cd941af9bbb347c8831c1a8e8830eb3b	done		
{'learning_rate': 0.05, 'num	1b41b2eb4f5141258fb56a0de8074fe3	done		

Figure 3: Table of all jobs and their status

- A form to add a new job to the queue database with the specified hyperparameters (e.g., learning rate, number of epochs, optimizer). Please refer to Fig. 4 for visualization.
- A select box for users to choose the experiment to display the results based on the id of the experiment. Please refer to Fig. 5 for more details.

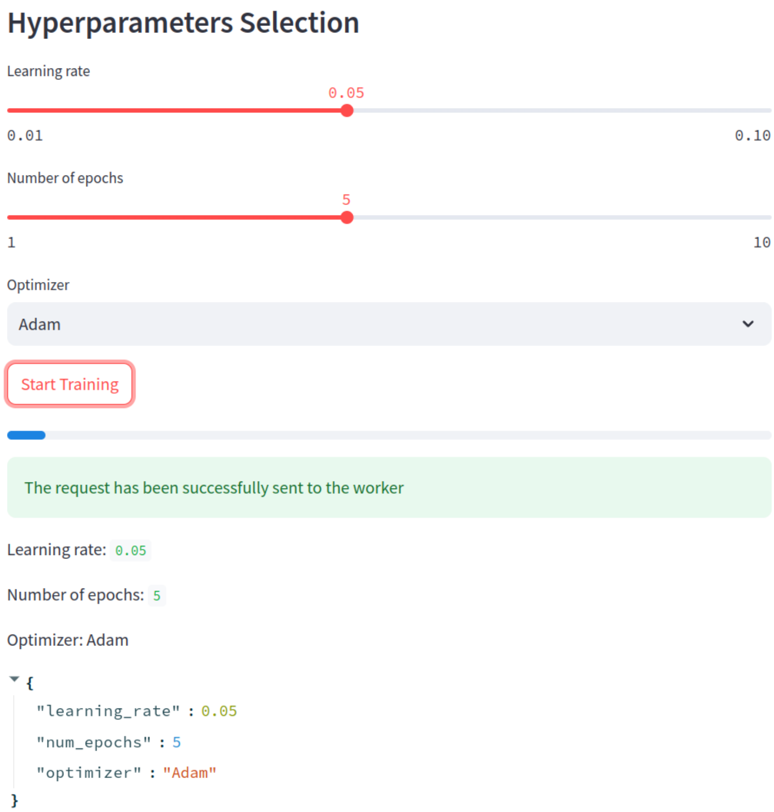


Figure 4: Add a new job to the queue database

- A table to display the results of the selected experiment
- A plot to display training/test loss and accuracy of the selected experiment

The default value of the frontend is `http://localhost:8501` by default. You can change the port/host by modifying the `streamlit run` line in `frontend.py`.

Figure 5: Display the results of the selected experiment

3 Deployment

To run the application in development mode, follow these steps:

1. Clone the repository

```
git clone https://github.com/mtuann/gui-ml-track.git
```

2. Install the required dependencies (optional)

```
conda env create -f environment.yml
source activate gui
sudo apt-get install redis-server
redis-server
```

3. Start the backend

```
python backend.py
```

4. Start the worker

```
python worker.py
```

5. Start the frontend

```
streamlit run frontend.py
```

The code and the instructions for running the application are available at `https://github.com/mtuann/gui-ml-track`.