Integrated Project Documentation: Neural Networks, Databases, and Google API

Abu Sayem

November 2023

Contact Information:
Email: abusayem379@gmail.com—— asayem172153@bscse.uiu.ac.bd
LinkedIn: `www.linkedin.com/in/abusayem172153`
GitHub: `https://github.com/asayem172153?tab=repositories`

Contents

1 Introduction **3**

2 Task 1: Creating a Neural Network **3**

2.1 Introduction 3

2.2 Neural Network Architecture 3

2.3 Data Pre-Processing & Training the Model 4

2.4 Evaluation 4

2.5 Configuration Choices 5

3 Task 2: Working with Databases **5**

3.1 Introduction 5

3.2 Database Creation 5

3.3 Database Operations 5

3.4 Database & Other Choices 6

4 Task 3: Integration with a Google API **6**

4.1 Introduction 6

4.2 Google API Integration 6

4.3 Code Organization 6

5 Conclusion **7**

1 Introduction

In the rapidly evolving landscape of artificial intelligence and data management, the integration of neural networks, databases, and external APIs plays a pivotal role in developing intelligent and efficient systems. This project aims to showcase a comprehensive exploration of these key components through practical implementation.

The project is divided into three main tasks, each addressing a fundamental aspect of contemporary technology:

1. **Task 1: Creating a Neural Network**

   In this task, I am developing a model capable of classifying images from the MNIST dataset. The architecture, training process, and evaluation of loss metrics and accuracy will be discussed.

2. **Task 2: Working with Databases**

   Databases serve as the backbone of data management systems. Task 2 involves the creation of a simple database, along with a Python script for interacting with it. I explore the operations of adding, retrieving/ displaying, updating, search and deleting data, emphasizing the importance of organized and efficient data storage.

3. **Task 3: Integration with a Google API**

   The modern era thrives on interconnectedness, and Task 3 demonstrates my integration of the system with a Google API, specifically Google Sheets. A Python script is crafted to send requests, receive data, and process information from a chosen Google Sheet.

This documentation provides a step-by-step walkthrough of each task, elucidating the choices made during implementation and the significance of the overall project.

2 Task 1: Creating a Neural Network

2.1 Introduction

Here the task is to create a simple neural network in python using TensorFlow or any other machine learning framework. But condition here is the network have to be capable of classifying images from the MNIST dataset

2.2 Neural Network Architecture

For the neural network architecture, I employed the Keras Sequential model, a widely used framework for building neural networks. The architecture consists of three main layers:

- **Flatten Layer:** This layer transforms the 28 x 28 input array of pixel values into a 1D array, allowing the neural network to process the image as a flat vector. It serves as the input layer.
- **Dense Layer (Hidden Layer):** A dense layer with 128 hidden units and ReLU (Rectified Linear Unit) activation function is incorporated.

- **Dense Layer (Output Layer):** The final layer is another dense layer with 10 units, representing the 10 classes in the MNIST dataset. It uses the softmax activation function, converting the network's output into probability distributions across the 10 classes.

2.3 Data Pre-Processing & Training the Model

The MNIST dataset is a widely used dataset in machine learning and computer vision. It consists of a collection of 28x28 pixel grayscale images of handwritten digits (0 through 9) along with their corresponding labels. It's a built-in dataset in tensorflow.keras. But before train a model its important to do some pre-processing steps with data.
Pre-Processing is very important part of machine learning model. It makes the data suitable for machine learning model and somtime helps the model to learn better. So here I have done some pre-processing before I train my model. Here is some pre-processing steps I have done-

- **Split the dataset:** It is important to and good practice to split the dataset into a train and test part. So, I load the MNIST dataset and split it into train images, train labels and test images, test labels.
- **Normalizing:** Data normalizing is a very crucial part. We know machine learning model often do many calculation and others thing to deliver a better result or output. So, normalized data helps these models for quick calculation and makes the model faster. I also do normalizing in this case. I firstly check the shape of train and test images using **.shape** method. After that, I check the max and min pixels from the both train and test images. I find max pixel 255 and min 0. Now to normalize them and to keep the pixels into 0 to 1 range I divided the train and test images with 255.

After the pre-processing my dataset is ready to train. The Neural Network I have defined. I fit the train image data to that model. We used various optimizer, loss function etc. to train the model.

2.4 Evaluation

After training the neural network with different optimizers, we evaluated the model's performance on the test set. The evaluation results are as follows:

- **Optimizer: SGD (Stochastic Gradient Descent)**

```
Loss: 0.1605, Accuracy: 95.36%
```

- **Optimizer: Adam**

```
Loss: 0.0782, Accuracy: 97.74%
```

- **Optimizer: RMSprop**

```
Loss: 0.0878, Accuracy: 97.88%
```

- **Optimizer: Adagrad**

```
Loss: 0.0870, Accuracy: 97.51%
```

The evaluation results demonstrate varying performance metrics for each optimizer. The choice of optimizer can significantly impact the model's convergence and final accuracy.

2.5 Configuration Choices

For the configuration of the neural network, I made several important choices:

- **Loss Function:** I used 'sparse categorical crossentropy' as the loss function. This choice is suitable for classification tasks with integer labels, as in the MNIST dataset. It automatically performs one-hot encoding internally, simplifying the label format.
- **Optimizers:** I experimented with different optimizers, namely Stochastic Gradient Descent (SGD), Adam, RMSprop, and Adagrad. Each optimizer has its advantages, and the choice depends on the specific characteristics of the dataset and the model.
- **Training Epochs:** The model was trained for 10 epochs. The number of epochs represents the number of times the entire training dataset is passed forward and backward through the neural network. A balance must be struck to prevent overfitting or underfitting.
- **Verbose Setting:** The training process was executed with verbose set to 0, suppressing the output during training for a cleaner presentation.

These configuration choices were made after careful consideration of the characteristics of the MNIST dataset and empirical testing to achieve optimal model performance.

3 Task 2: Working with Databases

3.1 Introduction

In this task, I focused on the integration and management of a database using SQLite. The primary goal was to create, manipulate, and interact with a database to store information.

3.2 Database Creation

To start, I created a SQLite database named `machine_learning.db`. If the database does not exist, the script creates it. Subsequently, I defined a table named `MLmodel` to store information about machine learning models. The table includes columns for the model's type, name, and model type.

3.3 Database Operations

For seamless interaction with the database, several functions were implemented:

- **Adding a Model (`add_model`):** This function adds a new model entry to the `MLmodel` table.

- **Updating a Model (`update_model`):** To modify existing model information, the `update_model` function was created.
- **Displaying All Models (`display_all_models`):** Retrieving and displaying all models in the `MLmodel` table was accomplished with this function.
- **Deleting a Model (`delete_model`):** To remove a specific model entry, the `delete_model` function was implemented.
- **Searching for a Model (`search_model`):** The `search_model` function allows searching for a model by its name.

3.4 Database & Other Choices

SQLite3 was chosen for its lightweight, serverless design, making it suitable for small to medium-sized projects. It is a self-contained, file-based relational database management system that requires minimal configuration. The simplicity and efficiency of SQLite3 make it an excellent choice for embedded database solutions, particularly in scenarios where a full-scale relational database management system may be overly complex or unnecessary. The `MLmodel` table is organized to capture essential information about machine learning models. The use of an `id` column as the primary key ensures a unique identifier for each model entry. The `type`, `model_name`, and `model_type` columns store the type of model, its name, and the specific model type, respectively.

These organization choices facilitate efficient data retrieval, updating, and deletion, providing a structured and scalable approach to managing data information within the SQLite database.

4 Task 3: Integration with a Google API

4.1 Introduction

For this task, the goal was to develop a Python script that interacts with the Google Sheets API, specifically fetching data from a designated range in a Google Sheet and processing the retrieved information.

4.2 Google API Integration

To enable interaction with the Google Sheets API, initial setup involved obtaining an API key from the Google Cloud Platform. The API was then configured to grant access to the designated Google Sheet. This process included creating a new project, activating the Google Sheets API, and generating API credentials.

4.3 Code Organization

The script was structured to utilize the **requests** library for sending HTTP requests to the Google Sheets API. Additionally, the **oauth2client.service_account** module was employed for authentication, utilizing a Service Account and the acquired API key.

The primary function, **get_google_sheets_data**, was defined to the logic for making API requests, processing the received data, and displaying the results. The function takes three parameters: **the API key, the target Google Sheet ID, and the desired sheet range**.

In the main block, the script is configured with the actual API key, Google Sheet ID, and the specified range within the sheet. The function **get_google_sheets_data** is then called, initiating the API request and processing the received data.

This integration demonstrates the practical implementation of the Google Sheets API to retrieve and process data from a Google Sheet, showcasing the seamless interaction between Python and external services.

5 Conclusion

In conclusion, this project encompassed three distinct tasks, each contributing to a comprehensive exploration of machine learning, database management, and API integration.

Task 1 involved the creation and training of a neural network for image classification using the MNIST dataset.

Task 2 focused on working with databases, specifically SQLite3. The organization of the database and the implementation of functions for adding, updating, retrieving, and deleting data demonstrated the practical aspects of managing machine learning model information in a structured manner.

Task 3 showcased the integration of a Python script with the Google Sheets API. Capable of sending requests to the API and processing the received data.

In essence, this 3 task not only provided hands-on experience in machine learning, database management, and API integration but also highlighted the significance of adaptability and strategic decision-making in the development of intelligent systems.