Технически университет – София

Факултет по приложна математика и информатика

Екипно задание

по

Технологии за големи данни

на тема

Сравнителен анализ на модели за разпознаване на числа

Изготвили: Кирил Костов, Светлодар Димитров, Цветомир Цветков

Фак. номера: 961324012, 961324013, 961324011

Група: 252

Съдържание

Описание

Сравнителният анализ на модели за разпознаване на числа се състои в създаването и обучаването на три модела върху MNIST дейтасета с помощта на програмния език Python и библиотеката PyTorch. Целта на проекта е успешното представяне на разликите в работата на трите модела и техните оценки.

Използвани технологии

Език за разработка

Python

Python е интерпретируем, обектно-ориентиран програмен език от високо ниво. Чрез своя лесен синтаксис в комбинация с възможност за бърза разработка и достъп до обширни библиотеки той се превръща в предпочитан избор за редица проекти и задачи.

Библиотеки

Array

Array е питонски модул, който служи за дефиниране на тип обект, който може да представлява масив от основни стойности: знаци, цели числа или числа с плаваща запетая.

Gradio

Gradio е Python пакет с отворен код, който позволява бързото създаване на демо или уеб приложение за модел на машинно обучение, API и други.

Kagglehub

Kagglehub е питонска библиотека за достъп до ресурси от Kaggle.

Matplotlib

Matplotlib е библиотека за създаване на визуализации в Python.

NumPy

NumPy е основният пакет за научни изчисления в Python. Той представлява библиотека на Python, която предоставя обект с многоизмерен масив, различни

производни обекти (като маскирани масиви и матрици) и набор от рутинни процедури за бързи операции с масиви, включително математически, логически и много други.

PyTorch

PyTorch е фреймуърк за дълбоко обучение с отворен код като е достъпен на Python и C++. PyTorch се намира вътре в модула *torch*. В PyTorch данните, които трябва да бъдат обработени, се въвеждат под формата на т.н. тензор.

Struct

Struct е модул за преобразуване между Python стойности и C структури, представени като Python bytes обекти.

Система за контрол на версиите

Git

Git е безплатна система за контрол на версиите с отворен код. Използването ѝ чрез команден интерфейс с богат набор от инструкции и възможността за свързване с уеб базирани Git хранилища като GitHub са от съществено значение за успешната екипна колаборация.

Разработка и архитектура

Архитектурен изглед

Проектът следва обектно-ориентиран дизайн, като за всеки от моделите е създаден отделен клас, който да го дефинира и общи вътрешни класове за зареждане на дейтасета и трениране на модел.

Архитектурата има следния вид:

|-- digit_recognition
|-- gradio_ui
|-- __init__.py
|-- gradio_fe.py
|-- model_data
|-- *logistic_regression.pth*
|-- *mlp.pth*
|-- *rnn_data.pth*
|-- models
|-- __init__.py
|-- logistic_regression.py
|-- mlp.py
|-- rnn_data.py
|-- utils
|-- __init__.py
|-- dataloader.py
|-- train.py
`-- main.py

Файлове

main.py

- Началната точка за изпълнение на програмата / стартиране на UI / трениране на модели.

models/logistic_regression.py

- Съдържа имплементацията на Logistic Regression класа, като наследява от **torch.nn.Module**, предефинира **forward** метода и създава инстанция (singleton) от класа.

```python
# Third-party Imports
import torch.nn as nn

# Constants
INPUT_SIZE = 784
NUM_CLASSES = 10


class LogisticRegression(nn.Module):
    def __init__(self, input_size=INPUT_SIZE, num_classes=NUM_CLASSES):
        super(LogisticRegression, self).__init__()
        self.linear = nn.Linear(input_size, num_classes)

    def forward(self, x):
        return self.linear(x)

model = LogisticRegression()
```

models/mlp.py

- Съдържа имплементацията на **Multilayer perceptron** класа, като също наследява от **torch.nn.Module** и предефинира **forward** метода след което създава инстанция (singleton) от класа.

```python
# Third-party Imports
import torch.nn as nn

# Constants
INPUT_SIZE = 784
HIDDEN_SIZE = 128
NUM_CLASSES = 10

class MLP(nn.Module):
    def __init__(self, input_size=INPUT_SIZE, hidden_size=HIDDEN_SIZE, num_classes=NUM_CLASSES):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)  # First fully connected layer
        self.relu = nn.ReLU()  # Activation function
        self.fc2 = nn.Linear(hidden_size, num_classes)  # Second fully connected layer

    def forward(self, x):
        out = self.fc1(x)  # Pass input through the first layer
        out = self.relu(out)  # Apply activation function
        out = self.fc2(out)  # Pass through the second layer
        return out

# Instantiate the model
model = MLP()
```

models/rnn_data.py

- Съдържа имплементацията на **Recurrent neural network** класа отново наследявайки от **torch.nn.Module** и предефинира **forward** метода и отново има сингълтън.

```python
import torch.nn as nn

INPUT_SIZE = 784
HIDDEN_SIZE = 128
NUM_CLASSES = 10

class RNN(nn.Module):
    def __init__(self, input_size=784, hidden_size=128, num_classes=NUM_CLASSES, num_layers=1):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        if x.dim() == 2:
            x = x.unsqueeze(1)
        x, _ = self.rnn(x)
        x = self.fc(x[:, -1, :])
        return x


```

utils/dataloader.py

- отговаря за свалянето и подготвянето на dataset-а

```python
# Standard Library Imports
import array
import numpy as np
import struct

# Constants
TRAINING_LABELS_REL_PATH = "/train-labels-idx1-ubyte/train-labels-idx1-ubyte"
TRAINING_IMAGES_REL_PATH = "/train-images-idx3-ubyte/train-images-idx3-ubyte"
TEST_LABELS_REL_PATH     = "/t10k-labels-idx1-ubyte/t10k-labels-idx1-ubyte"
TEST_IMAGES_REL_PATH     = "/t10k-images-idx3-ubyte/t10k-images-idx3-ubyte"

MAGIC_LABEL = 2049
MAGIC_IMAGE = 2051

BYTE_8 = 8

ROW_SIZE = 28
COL_SIZE = 28


class DataLoader:
    def __init__(self, path):
        # Initialize train and test paths
        self.training_labels_filepath = path + TRAINING_LABELS_REL_PATH
        self.training_images_filepath = path + TRAINING_IMAGES_REL_PATH
        self.test_labels_filepath     = path + TEST_LABELS_REL_PATH
        self.test_images_filepath     = path + TEST_IMAGES_REL_PATH

    def _unpack(self, path, expected_magic):
        with open(path, 'rb') as f:
            # Unpack the first 8 bytes
            actual_magic, size = struct.unpack(">II", f.read(BYTE_8))

            # Verify unpacked struct is correct
            if actual_magic != expected_magic:
                raise ValueError("Magic number mismatch")

            data = array.array('B', f.read())

        return data, size

    def _prepare_data(self, labels_path, images_path):
        # Read labels
        labels, size = self._unpack(labels_path, MAGIC_LABEL)

        # Read image data
        image_data, _ = self._unpack(images_path, MAGIC_IMAGE)

        # Create helper variables
        images = []
        pixels_per_image = ROW_SIZE * COL_SIZE

        # Prepare images by adding empty arrays
        for i in range(size):
            images.append([0] * pixels_per_image)

        # Get actual image data per image in the 1D array
        for i in range(size):
            img = np.array(image_data[i * pixels_per_image:(i + 1) * pixels_per_image])
            img = img.reshape(ROW_SIZE, COL_SIZE)
            # Save normalized data
            images[i][:] = img / 255.0

        return images, labels

    def load_data(self):
        # Prepare train data
        x_train, y_train = self._prepare_data(self.training_labels_filepath, self.training_images_filepath)

        # Prepare test data
        x_test, y_test = self._prepare_data(self.test_labels_filepath, self.test_images_filepath)

        return (x_train, y_train), (x_test, y_test)
```

train.py

- common файл, който се грижи за тренирането на подаден модел и имплементира **train** и **evaluate** методите.

    - метод ***train***

```python
    def train(self):
        for epoch in range(self.epochs):
            print('Start Training...')

            # Set model to training mode
            self.model.train()

            for batch_idx, (images, labels) in enumerate(self.train_loader):
                # Move the data to the same device
                images, labels = images.to(DEVICE), labels.to(DEVICE)

                # Flatten the images if needed
                if self.requires_flattening:
                    images = images.view(images.size(0), -1)

                # Forward pass
                outputs = self.model(images)
                loss = self.criterion(outputs, labels)

                # Backward pass and optimization
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()

                # Show data to user
                if (batch_idx + 1) % 100 == 0:
                    print(f'Epoch [{epoch+1}/{self.epochs}], Step [{batch_idx+1}/{len(self.train_loader)}], Loss: {loss.item():.4f}')

        model_file = f'models/{self.name}.pth'

        print(f"Model will be saved in {model_file}")
        torch.save(self.model.state_dict(), model_file)

```

- метод ***evaluate***

```python
    def evaluate(self):
        # Initialize Counters
        correct = 0
        total = 0

        # Set model to evaluation mode
        self.model.eval()

        # Disable gradient computation
        with torch.no_grad():
            for images, labels in self.test_loader:
                # Move the data to the same device
                images, labels = images.to(DEVICE), labels.to(DEVICE)

                # Flatten the images for LR
                if self.requires_flattening:
                    images = images.view(images.size(0), -1)

                # Forward pass
                outputs = self.model(images)
                _, predicted = torch.max(outputs, 1)

                # Update counters
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        # Calculate accuracy
        accuracy = 100 * correct / total

        # Show accuracy up to 2 digits after the point
        print(f"Accuracy: {accuracy:.2f} %")

        return accuracy
```

gradio_ui/gradio_fe.py

- съдържа цялата имплементация + препроцесинг на данните за визуализация и работа на всеки от моделите

Сорс код

Изходния код на проекта може да бъде разгледан на следния линк:
https://github.com/TsvetomirTsvetkov/digit-recognition