FINAL DELIVERABLE

AI2002- ARTIFICIAL INTELLIGENCE

TORCS-The Open Racing Car Simulator

TEAM MEMBERS:

SARA AKBAR	22i-0846
NOOR FATIMA	22i-1036
EMAAN ALI	22i-2325

SECTION:

J

Data Preprocessing and Training

Dataset Loading

- Loads the CSV file into a Pandas DataFrame named df.
- Each row represents one time-step of driving data.
- Each column is a feature (e.g., speed, track sensors) or target output (e.g., steering, acceleration).

Feature and Target Separation

- y_columns lists the output variables your neural network will predict.
- X_columns automatically grabs all the other columns as input features (by excluding y_columns).

Handle Missing Data

- Ensures that any row with missing (NaN) values is removed. Neural networks require clean, complete input.

Convert to PyTorch Tensors

- Converts the DataFrame data into float32 tensors.
- X: Input features (e.g., track sensors, speed, RPM)
- y: Target outputs (e.g., how much to steer or accelerate)

Train/Validation Split

- Wraps X and y into a TensorDataset, which can be used by PyTorch's DataLoader.
- Randomly splits the dataset:
    - 80% for training
    - 20% for validation

Create DataLoaders

- train_loader: Batches of 64, shuffled each epoch (for better generalization)
- val_loader: Also batched, but not shuffled (used only for evaluating performance)

Neural Network for Driving Control Prediction

1. Objective

The aim of this project is to train a neural network model to predict various driving control actions based on input features. Specifically, the model will predict:

- **Acceleration (accel)**
- **Braking (Braking)**
- **Clutch (Clutch)**
- **Gear (Gear)**
- **Steering (Steering)**

The model uses a Multi-Layer Perceptron (MLP) to predict these outputs from a set of driving-related sensor features.

---

2. Dataset Overview

The dataset used in this model is stored in a CSV file (Dataset.csv). Each row contains telemetry data, with features representing various driving conditions and outputs corresponding to different driving control actions. The data is cleaned, preprocessed, and split into training and validation sets.

---

3. Data Preprocessing

3.1 Data Loading

The dataset is loaded into a Pandas DataFrame

3.2 Feature Selection

- **Target Variables (y_columns)**: These are the columns that the model aims to predict. These include:
    - accel (acceleration)
    - Braking (braking force)
    - Clutch (clutch position)
    - Gear (gear position)
    - Steering (steering angle)
- **Input Variables (X_columns)**: All other columns in the dataset (excluding y_columns) are considered as input features for the model.

3.3 Handling Missing Data

Rows containing missing (NaN) values are dropped from the dataset to ensure clean data for training the model.

3.4 Conversion to PyTorch Tensors

The input and output data are converted from Pandas DataFrames to PyTorch tensors, which are required for training the neural network.

3.5 Splitting Data into Training and Validation Sets

The dataset is split into training (80%) and validation (20%) sets using random_split. This ensures that the model is evaluated on unseen data during training.

---

4. Model Architecture

4.1 Multi-Layer Perceptron (MLP)

The model is a fully connected neural network with three layers:

1. **Input Layer**: The number of input nodes corresponds to the number of features in X_columns.
2. **Hidden Layers**: Two hidden layers with ReLU activations:
    - First hidden layer: 128 neurons.
    - Second hidden layer: 64 neurons.
3. **Output Layer**: A layer with 5 output neurons corresponding to the 5 target variables (accel, Braking, Clutch, Gear, Steering).

4.2 Loss Function and Optimizer

- **Loss Function**: The model uses Mean Squared Error (MSE) loss, which is appropriate for regression tasks (predicting continuous values).
- **Optimizer**: The Adam optimizer is used for efficient parameter updates during training with a learning rate of 0.001.

---

5. Training Process

5.1 Training Loop

The model is trained for **50 epochs**. For each epoch, the following steps are performed:

1. **Training**:

- The model performs a forward pass for each batch of input data.
- The loss is calculated using the MSE loss function.
- The gradients are computed and used to update the model parameters using backpropagation.

2. **Validation**:
   - After each epoch, the model is evaluated on the validation set to compute the validation loss. This is important to monitor overfitting and generalization performance.

5.2 Epoch Output

After each epoch, the **validation loss** is printed to provide insight into the model's performance on unseen data.

---

6. Model Saving

After training is complete, the model's parameters (weights) are saved to a file (mlp_controller_model.pth). This allows the model to be loaded and used for inference at a later time.

---

7. Evaluation and Results

During training, the model's performance is monitored using the **validation loss**. A lower validation loss indicates that the model is generalizing well to unseen data. The training and validation process helps detect overfitting and underfitting issues.

---

8. Conclusion

This neural network model, implemented as a Multi-Layer Perceptron (MLP), is designed to predict multiple continuous outputs (acceleration, braking, clutch, gear, and steering) based on input telemetry data from a driving system. The model is trained using the Mean Squared Error loss function, with the Adam optimizer, and is evaluated on a validation set to ensure its effectiveness. After training, the model is saved for future use, allowing for deployment or further fine-tuning.

Running Command

On your terminal: python pyclient.py

And run the game simulator