# AI Racing Driver for TORCS

An intelligent self-driving agent that learns to drive in the TORCS racing simulator using a multi-output neural network trained on real driving data.

## Overview

This project implements a machine-learning pipeline for building an AI-controlled racecar in the **TORCS** simulator. It processes human driving data, trains a neural network on key sensory features, and uses the model to predict driving actions in real-time.

## Key Features

- **Neural Network-Based Driving**: Predicts steering (continuous), acceleration, and braking (binary).
- **Real-Time Control**: Integrated with TORCS via a Python socket-based client.
- **Modular Pipeline**: Cleanly separated data processing, training, and inference stages.

## Model Training Architecture

### Input Features

A total of **37 features** are used, drawn from sensor readings such as:

- **Track sensors**: Track_1 to Track_19 (distance to track edges)
- **Velocities**: SpeedX, SpeedY, SpeedZ
- **Car state**: Angle, TrackPosition, RPM, Damage
- **Wheel spin** and **opponent proximity**

These features provide spatial awareness and kinematic context.

### Output Targets

- **Steering**: A continuous value between -1 and 1 (scaled using `MinMaxScaler`)
- **Acceleration** and **Braking**: Thresholded as binary actions

### Network Architecture

Implemented using Scikit-learn's `MLPRegressor`:

| Layer Type | Configuration |
|---|---|
| Input | 37 features |
| Hidden 1 | 64 neurons, ReLU |
| Hidden 2 | 128 neurons, ReLU |
| Hidden 3 | 64 neurons, ReLU |
| Output | 3 outputs: Steering, Acceleration, Braking |

Additional hyperparameters:

- `learning_rate='adaptive'`, `early_stopping=True`, `max_iter=1000`
- `batch_size=64`, `tol=1e-5`, `n_iter_no_change=15`

### Data Handling & Training Strategy

1. **CSV Aggregation**: Multiple driving sessions are combined into one dataset.

2. **Preprocessing**:
    - Missing values filled using feature-wise means
    - Normalization:
        - `StandardScaler` for input features
        - `MinMaxScaler` (range -1 to 1) for steering output
    - Binary thresholds applied to acceleration and braking
3. **Splitting**: The dataset is divided into 60% training, 20% validation, and 20% testing.
4. **Training**: Uses validation split for early stopping and adaptive learning rate.

---

## Requirements

- Python 3.6+
- Libraries: `scikit-learn`, `pandas`, `numpy`, `matplotlib`, `joblib`
- TORCS + SCRC patch (socket-based control)

---

### 👥 Work Division

In this project, each team member was given specific tasks to help build the AI racing driver system. **Muhammad Bilal** collected driving data on **E-Track 3**, focusing on smooth and consistent driving. **Rana Bilal** collected data on the **Dirt Track**, which included rougher and more challenging road conditions. **Mehboob** collected data on the **G-Speedway**, which helped with high-speed and sharp turn scenarios. We also split the work of testing different machine learning models. Muhammad Bilal worked on a **TensorFlow neural network**, Rana Bilal used **Scikit-learn's MLPRegressor**, **MLPClassifier**, and also experimented with **TD3 reinforcement learning**, while Mehboob tested the **Random Forest algorithm**. After trying out different models and architectures, we selected the one that gave the best results and integrated it into the final project. This teamwork helped us build a better AI driver by combining data collection and model testing efforts.

### Conclusion

We built a neural network using Scikit-learn's `MLPRegressor` to control a car in the TORCS racing simulator. The model is trained offline and then used during real-time driving. Since it's small and efficient, it runs fast enough for the simulator's real-time requirements.

Instead of using hand-coded rules, the model learns to map sensor inputs to control actions like steering, throttle, and brake. This makes the driving logic simpler and more flexible for different tracks.

In this project, we learned how to collect driving data, train a multi-output model, and use it in a closed-loop system. We also learned that data preprocessing and model tuning are important for good performance. Overall, the project shows how supervised learning can imitate human driving in a racing game.