National University
Of Computer and Emerging Sciences

PROJECT

TORCS MACHINE LEARNING PROJECT

In partial fulfillment of the requirement for the course of

ARTIFICIAL INTELLIGENCE (AI-2002)

By

M. Huzaifa (22i-1220)

M. Sarmad (22i-0941)

M. Umar Hassan (22i-0942)

(CS-H)

Table of Contents

TORCS Machine Learning Project

Abstract:

In this project, we developed an intelligent, data-driven car racing controller using the TORCS framework. By transforming raw telemetry data into clean, normalized feature–label pairs, we enabled supervised learning for continuous control prediction using Multilayer Perceptron (**MLP**). The use of modern training practices—such as validation splits, normalization, and regularization callbacks, helped build a robust model that can effectively navigate various tracks under competitive conditions.

1. Data Pre-processing

In this project we transform raw TORCS telemetry logs into normalized input–output pairs for supervised learning. Key steps:

1.1 Loading & Filtering

- Read Dataset.csv into a pandas DataFrame.
- Remove "pre-start" rows where CurrentLapTime < 0.

1.2 Dropping Unused Columns

Discard meta-columns not directly useful for driving control.

Damage, CurrentLapTime, DistanceFromStart, DistanceCovered,

FuelLevel, LastLapTime, RacePosition.

1.3 Defining Features & Labels

- **Features (X)**:
- Car orientation & position: Angle, TrackPosition, Z
- Speeds: SpeedX, SpeedY, SpeedZ, RPM
- Opponents: Opponent_1 … Opponent_36
- Track sensors: Track_1 … Track_19

- Wheel spin: WheelSpinVelocity_1 … WheelSpinVelocity_4
- Gear (included as input)
- **Labels (y)**:
- Continuous control targets: Clutch, Braking, Steering, Acceleration, Gear

1.4 Normalization

- Fit a MinMaxScaler on all features to rescale into [0,1].
- Save the fitted scaler (scaler_input.pkl) for inference.

- **Train/Validation Split**
- Perform an 80/20 split (random seed 42).
- Export CSV files: train_X.csv, val_X.csv, train_y.csv, val_y.csv.

```
Loading raw data from: ./Dataset.csv
Dropped pre-start rows; remaining rows: 912979
Fitted MinMaxScaler including gear and saved scaler.
Preprocessing complete:
 train_X shape: (730383, 66)
 val_X shape:   (182596, 66)
 train_y shape: (730383, 4)
 val_y shape:   (182596, 4)
```

Figure 1 Data pre-processing

1.5 Code:

```
# 1) Load raw TORCS log data
print("Loading raw data from:", RAW_CSV)
df = pd.read_csv(RAW_CSV)

# 2) Drop countdown / pre-start rows (curLapTime < 0)
df = df[df["CurrentLapTime"] >= 0].reset_index(drop=True)
print("Dropped pre-start rows; remaining rows:", df.shape[0])

# 3) Drop unused / meta columns (keep 'gear' as feature)
drop_cols = [
    "Damage", "CurrentLapTime", "DistanceFromStart",
    "DistanceCovered", "FuelLevel", "LastLapTime",
    "RacePosition"
]
df = df.drop(columns=drop_cols)

# 4) Define feature & label columns, include 'gear' as feature
track_cols = [f"Track_{i}" for i in range(1,20)]
wsv_cols   = [f"WheelSpinVelocity_{i}" for i in range(1,5)]
opp_cols   = [f"Opponent_{i}" for i in range(1,37)]

feature_cols = ["Angle"] + opp_cols + [
    "RPM", "SpeedX", "SpeedY", "SpeedZ", "TrackPosition", "Z"
] + track_cols + wsv_cols
label_cols   = ["Clutch", "Braking", "Steering", "Acceleration"]  # gear removed from label

# 5) Normalize features into [0,1] using MinMaxScaler
scaler = MinMaxScaler()
X_norm = scaler.fit_transform(df[feature_cols])
joblib.dump(scaler, SCALER_PATH)
print("Fitted MinMaxScaler including gear and saved scaler.")

# 6) Prepare labels (gear is now input, so labels exclude it)
y_df = df[label_cols].reset_index(drop=True)
```

---

2. Model Training

Algo Chosen : Neural Networks **Reason :** Because we had multiple labels to infer and also the dependencies were complex

Trained a deep feed-forward neural network to map the N-dimensional feature vector to the 5dimensional control vector:

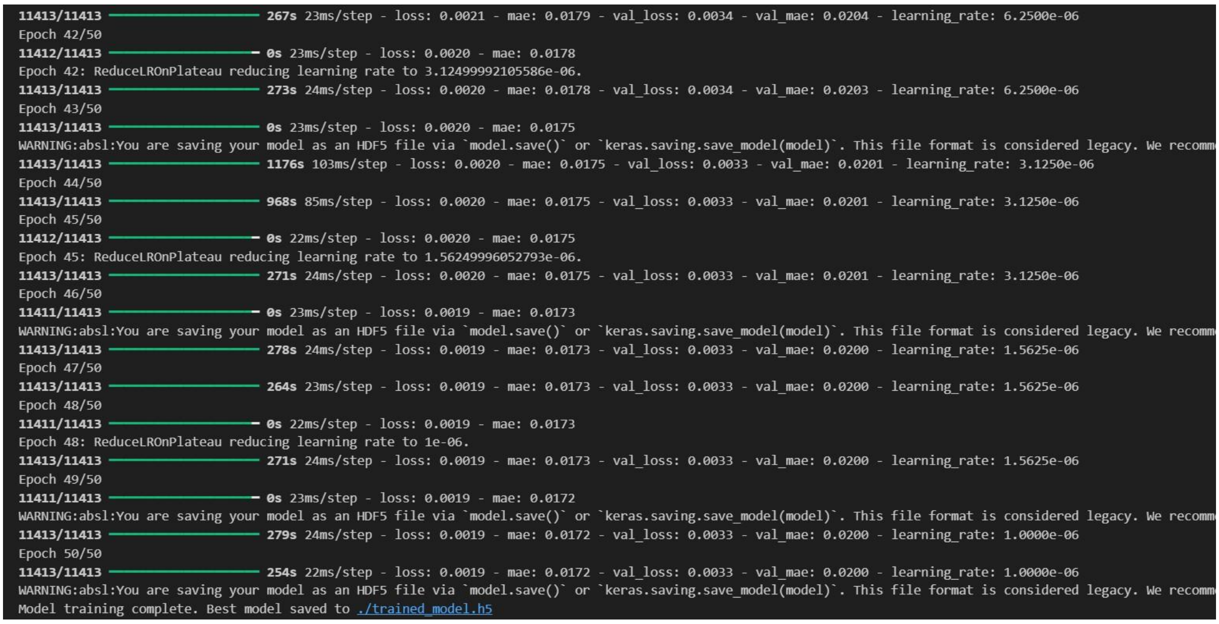
- **Network Architecture**

Sequential Keras model with nine Dense layers:

Input → 1024 → 512 → 256 → 128 → 64 → 32 → 16 → 8 → Output(5) Hidden layers use ReLU activations; the output layer is linear.

- **Compilation Settings**

- Optimizer: Adam (**learning rate = 1e-4**)
- Loss: Mean Squared Error (MSE)
- Metric: Mean Absolute Error (MAE)
- **Callbacks**
- ModelCheckpoint: save best model by validation loss
- EarlyStopping: stop if no improvement for 5 epochs, restore best weights
- ReduceLROnPlateau: halve LR on plateau (factor 0.5, patience 3, min_lr=1e6)
- **Training Run**
- Epochs: up to 50
- Batch size: 64
- Validate on 20% hold-out each epoch
- Best model saved to trained_model.h5



```
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 267s 23ms/step - loss: 0.0021 - mae: 0.0179 - val_loss: 0.0034 - val_mae: 0.0204 - learning_rate: 6.2500e-06
Epoch 42/50
11412/11413 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0020 - mae: 0.0178
Epoch 42: ReduceLROnPlateau reducing learning rate to 3.12499992105586e-06.
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 273s 24ms/step - loss: 0.0020 - mae: 0.0178 - val_loss: 0.0034 - val_mae: 0.0203 - learning_rate: 6.2500e-06
Epoch 43/50
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0020 - mae: 0.0175
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomm
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 1176s 103ms/step - loss: 0.0020 - mae: 0.0175 - val_loss: 0.0033 - val_mae: 0.0201 - learning_rate: 3.1250e-06
Epoch 44/50
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 968s 85ms/step - loss: 0.0020 - mae: 0.0175 - val_loss: 0.0033 - val_mae: 0.0201 - learning_rate: 3.1250e-06
Epoch 45/50
11412/11413 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0020 - mae: 0.0175
Epoch 45: ReduceLROnPlateau reducing learning rate to 1.56249996052793e-06.
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 271s 24ms/step - loss: 0.0020 - mae: 0.0175 - val_loss: 0.0033 - val_mae: 0.0201 - learning_rate: 3.1250e-06
Epoch 46/50
11411/11413 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0019 - mae: 0.0173
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomm
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 278s 24ms/step - loss: 0.0019 - mae: 0.0173 - val_loss: 0.0033 - val_mae: 0.0200 - learning_rate: 1.5625e-06
Epoch 47/50
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 264s 23ms/step - loss: 0.0019 - mae: 0.0173 - val_loss: 0.0033 - val_mae: 0.0200 - learning_rate: 1.5625e-06
Epoch 48/50
11411/11413 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0019 - mae: 0.0173
Epoch 48: ReduceLROnPlateau reducing learning rate to 1e-06.
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 271s 24ms/step - loss: 0.0019 - mae: 0.0173 - val_loss: 0.0033 - val_mae: 0.0200 - learning_rate: 1.5625e-06
Epoch 49/50
11411/11413 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0019 - mae: 0.0172
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomm
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 279s 24ms/step - loss: 0.0019 - mae: 0.0172 - val_loss: 0.0033 - val_mae: 0.0200 - learning_rate: 1.0000e-06
Epoch 50/50
11413/11413 ━━━━━━━━━━━━━━━━━━━━ 254s 22ms/step - loss: 0.0019 - mae: 0.0172 - val_loss: 0.0033 - val_mae: 0.0200 - learning_rate: 1.0000e-06
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomm
Model training complete. Best model saved to ./trained_model.h5
```

Figure 2: Model Training

2.1 Code:

```python
model = Sequential([
    Dense(1024, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(8, activation='relu'),
    Dense(y_train.shape[1])
])

# 3) Compile model with learning rate 1e-4
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='mse',
    metrics=['mae']
)

# 4) Set up callbacks including LR reduction on plateau
callbacks = [
    ModelCheckpoint(MODEL_PATH, save_best_only=True, monitor='val_loss'),
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6, verbose=1)
]

# 5) Train model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=64,
    callbacks=callbacks
)
```

---

3. Inference & Car Control

At runtime, the trained model and scaler are used to compute control commands from live sensor data:

3.1 Load Artifacts

- Load MinMaxScaler to replicate training-time normalization.
- Load the Keras model (trained_model.h5) in inference mode.

3.2 Feature Vector Construction

- Read current state: angle, opponent distances, speeds, track sensors, wheel spins, gear.

- Assemble into a $1 \times N$ NumPy array.

3.3 Normalization

- X_norm = (X_raw - scaler.data_min_) / (scaler.data_max_ - scaler.data_min_)

3.4 Prediction

- Model outputs: [clutch, brake, steering, acceleration, gear].

3.5 Control Mapping

- Clamp each value to valid range (e.g., steering $\in [-1,1]$, accel $\in [0,1]$).

- Send commands to the car control interface.

3.6 Feedback Loop

- Repeat for each incoming sensor message to form a closed-loop controller

4. Work Division:

- **Data Collection :** Muhammad Huzaifa
- **Pre Processing :** Muhammad Umar Hassan
- **Model Training & Driver Implementation :** Muhammad Sarmad