**Deep Q-Learning for Vehicle Behavior Classification in V2X Environments**

**Overview**

This project implements a Deep Q-Learning (DQN) model to classify vehicle behavior as either "genuine" or "misbehaving" based on time-series data from a Vehicle-to-Everything (V2X) environment. The model uses Basic Safety Message (BSM) features such as speed, acceleration, heading, and position to detect abnormal behaviors. The approach leverages reinforcement learning to train a classifier that can handle dynamic and evolving scenarios in high-density and low-density traffic environments.

**Key Components**

1. **Data Preprocessing**:
   * Normalizes the dataset using MinMaxScaler.
   * Splits the dataset into training and testing sets.
2. **Deep Q-Learning Algorithm**:
   * Uses a Deep Neural Network (DNN) to approximate Q-values for state-action pairs.
   * Implements:
     + **Experience Replay**: Stores previous experiences to avoid overfitting and improve learning stability.
     + **Target Network**: Stabilizes training by periodically updating a separate target network.
   * Employs an epsilon-greedy policy for balancing exploration and exploitation.
3. **Evaluation Metrics**:
   * Measures the model's performance using Recall, F1 Score, and Accuracy.
   * Supports multiclass evaluation using macro averaging.

**Code Documentation**

**1. load\_data(filepath)**

* **Purpose**: Loads and preprocesses the dataset for training and testing.
* **Parameters**:
  + filepath (str): Path to the CSV file containing the dataset.
* **Returns**:
  + X (numpy.ndarray): Normalized feature set.
  + y (numpy.ndarray): Target labels.
* **Key Operations**:
  + Selects relevant features (spdx, spdy, aclx, acly, hedx, hedy, posx, posy).
  + Normalizes feature values to a range of [0, 1].

**2. build\_model(state\_size, action\_size)**

* **Purpose**: Builds the Deep Neural Network (DNN) for approximating Q-values.
* **Parameters**:
  + state\_size (int): Number of features in the input state.
  + action\_size (int): Number of possible actions (classes).
* **Returns**:
  + model (keras.Sequential): Compiled DNN model.

**3. train\_dqn(X\_train, y\_train)**

* **Purpose**: Trains the DQN model using the training dataset.
* **Parameters**:
  + X\_train (numpy.ndarray): Training feature set.
  + y\_train (numpy.ndarray): Training labels.
* **Returns**:
  + agent (DQNAgent): Trained agent.
* **Key Operations**:
  + Iterates over episodes, updating the agent's experience buffer and neural network.
  + Decays epsilon over time to encourage exploitation in later episodes.

**4. evaluate(agent, X\_test, y\_test)**

* **Purpose**: Evaluates the trained agent on the test dataset.
* **Parameters**:
  + agent (DQNAgent): Trained agent.
  + X\_test (numpy.ndarray): Testing feature set.
  + y\_test (numpy.ndarray): Testing labels.
* **Returns**:
  + recall (float): Recall score for the classification task.
  + f1 (float): F1 score for the classification task.
  + accuracy (float): Overall classification accuracy.
* **Key Operations**:
  + Collects predictions from the trained agent.
  + Computes metrics using sklearn evaluation functions.

**5. DQNAgent Class**

* **Purpose**: Represents the reinforcement learning agent.
* **Methods**:
  + \_\_init\_\_: Initializes the agent's parameters, experience replay buffer, and models.
  + remember: Stores experiences in the replay buffer.
  + act: Selects an action using an epsilon-greedy policy.
  + replay: Trains the model using mini-batches from the experience buffer.
  + update\_target\_model: Copies weights from the training model to the target model.

**Usage Instructions**

1. **Setup**:
   * Install dependencies:

bash

CopyEdit

pip install numpy pandas scikit-learn keras tensorflow

* + Prepare the dataset in CSV format and provide its file path.

1. **Run the Program**:
   * Adjust parameters such as number of episodes, batch size, and epsilon decay in train\_dqn for experimentation.
   * Execute the code:

python

CopyEdit

filepath = 'path\_to\_dataset.csv'

X, y = load\_data(filepath)

split = int(0.8 \* len(X))

X\_train, X\_test, y\_train, y\_test = X[:split], X[split:], y[:split], y[split:]

agent = train\_dqn(X\_train, y\_train)

evaluate(agent, X\_test, y\_test)

1. **Results**:
   * Observe Recall, F1 Score, and Accuracy metrics for model evaluation.

**Performance Tuning**

* Increase training episodes for better policy convergence.
* Optimize hyperparameters:
  + Learning rate
  + Batch size
  + Reward system
* Use larger training datasets for improved generalization.

**Known Issues and Debugging**

* **Low Accuracy**: Ensure sufficient training data and adjust hyperparameters.
* **Evaluation Errors**: Verify the compatibility of predictions (y\_pred) and true labels (y\_test).