DA221M Course Project

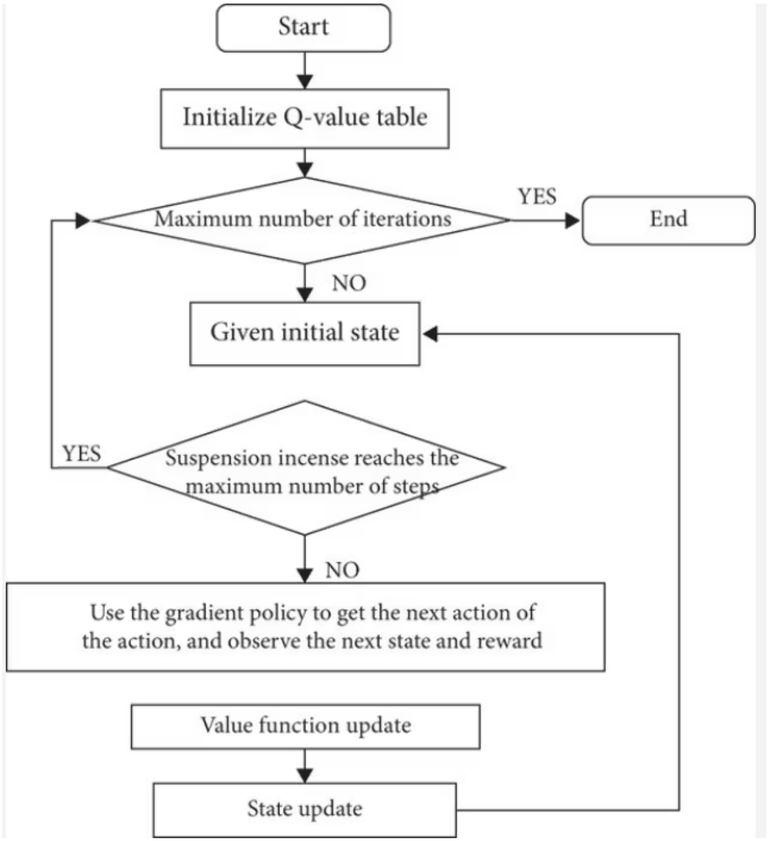
Reinforcement Learning: Self Driving Car

Agenda

Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning focused on training agents to make sequential decisions by interacting with an environment to maximize cumulative rewards.

This branch focuses on trial and error much like how we as humans learn from our environment rather than giving fixed set of instructions such as in the case of supervised learning.



Q Learning Environment

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Learning Rate:

The learning rate or step size determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities).

Discount Factor:

The discount factor, denoted as gamma, determines how much importance an agent places on future rewards. If gamma is set to 0, the agent becomes "myopic" or short-sighted, only valuing immediate rewards, rt. As gamma approaches 1, the agent increasingly prioritizes long-term rewards, aiming to maximize future gains.

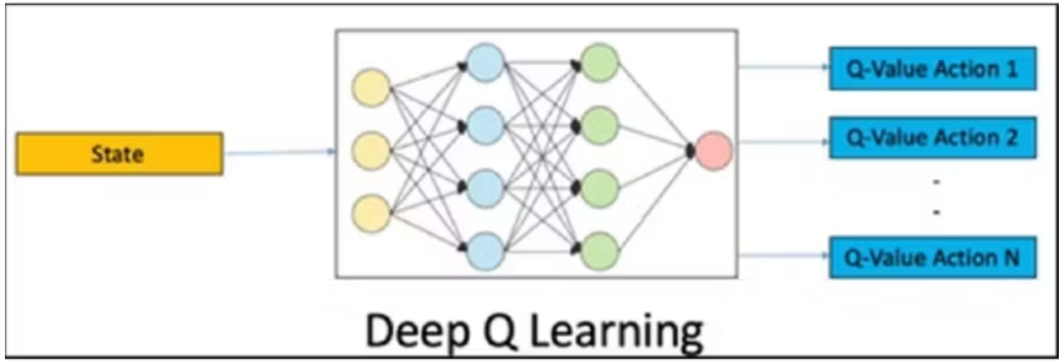
Initial Conditions:

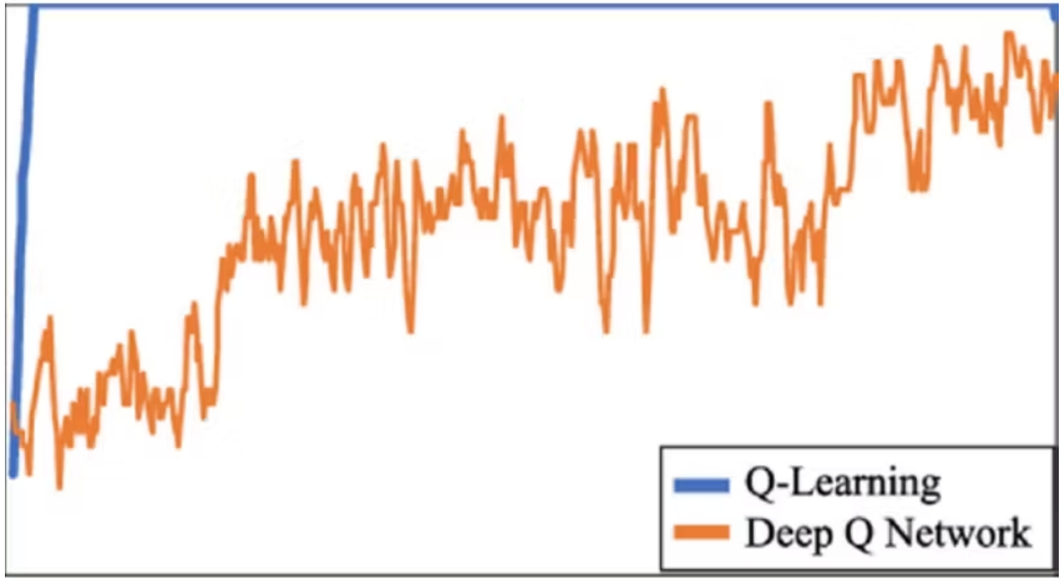
Since Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values, known as "optimistic initial conditions," can promote exploration: regardless of the selected action, the update rule will reduce its value, increasing the probability of choosing other actions.

Deep Learning in Q Learning Algorithms

A DQN or Deep Q-Network, approximates a state-value function in a Q-Learning framework in a neural network

In autonomous driving here, various sensors are mounted in the vehicle, penalties systems for collisions, positive rewards for appropriate speed and making progress are incorporated by the DQN network





Autonomous Driving Environment

Objective:

Keeping the car on the track as much as possible

Goal:

Maximise total rewards by keeping the car on the track in each farme

Rewards:

1. **Time Penalty**
   `-0.1` reward per frame
   - Encourages quick completion of the track
   - Accumulates continuously throughout the episode
2. **Track Completion Bonus**
   `+1000/N` per visited tile
   - `N` = Total number of tiles in the track
   - Full track completion yields `+1000` bonus
   - Distributed incrementally as tiles are visited
3. **Off-Track Penalty**
   `-100` reward + episode termination
   - Triggered when car strays too far from track
   - Ensures agent learns to stay within bounds

Methodology

- **Preprocessing Observations**:
  - Convert RGB images to grayscale to reduce complexity.
  - Normalize pixel values to ensure consistent input for the neural network.
- **Neural Network Architecture**:
  - Use convolutional layers to extract spatial features from the game's visual input.
  - Follow with fully connected layers to predict Q-values for each possible action.
- **Replay Buffer**:
  - Implement a buffer to store experiences (state, action, reward, next state, done) for training.
  - This helps in breaking the correlation between consecutive samples, improving learning efficiency.
- **Epsilon-Greedy Policy**:
  - Balance exploration (choosing random actions) and exploitation (choosing the best action according to the model) during training.
  - This ensures the agent learns from both known and unknown scenarios.
- **Training Loop**:
  - Optimize the policy network using sampled experiences from the replay buffer.
  - Periodically update the target network to stabilize learning.
- **Evaluation**:
  - After training, evaluate the agent using a greedy policy where exploration is minimized to assess the learned policy's performance.

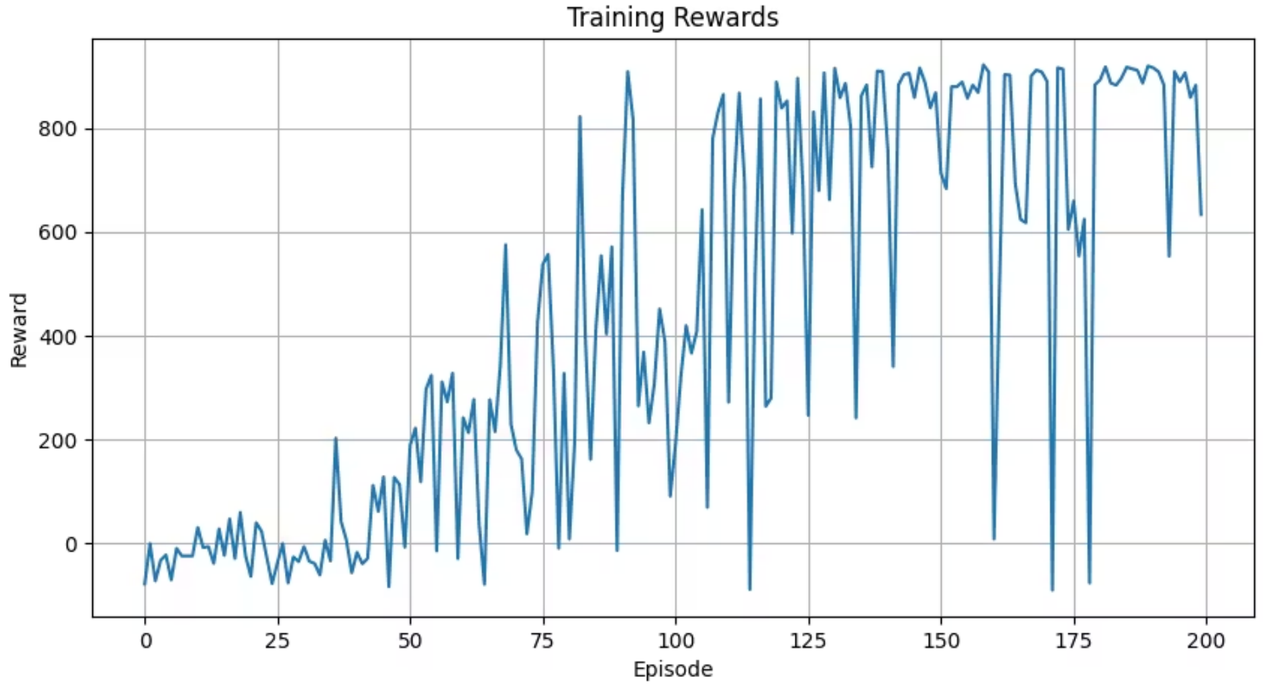
Implementing the DQN model

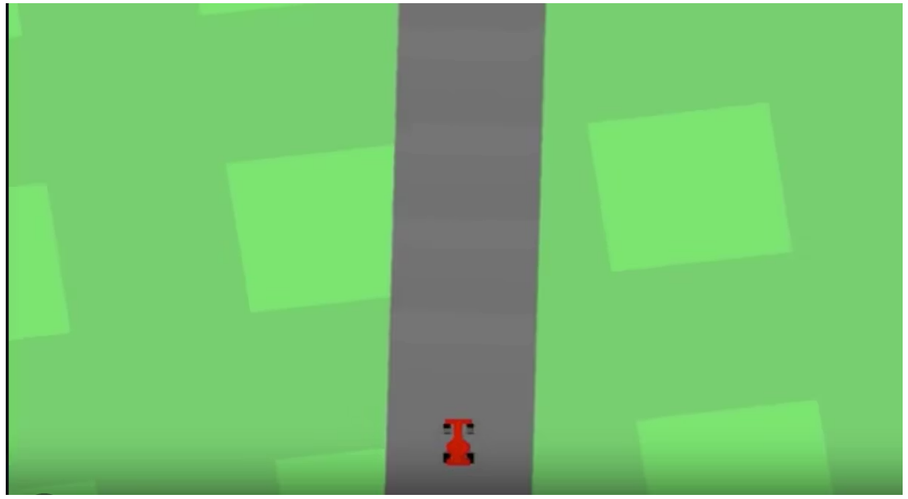
- **Environment Setup**: The CarRacing-v3 environment is instantiated with a discrete action space, allowing for simpler action selection. The environment's observation space is a 96x96x3 RGB image, and the action space consists of 5 discrete actions.
- **Preprocessing Function**: A preprocess_observation function is defined to convert RGB observations to grayscale, normalize pixel values to1, and reshape the input to (1, 96, 96) for the neural network.
- **Neural Network**: The DQN architecture is implemented with:
  - **Convolutional Layers**: Three convolutional layers with ReLU activations for feature extraction from the game's visual input. The layers have kernel sizes of 8x8, 4x4, and 3x3 with strides of 4, 2, and 1, respectively.
  - **Fully Connected Layers**: Two fully connected layers with ReLU activations, followed by a final layer to predict Q-values for each action.
- **Replay Buffer**: A ReplayBuffer class is implemented to store and sample experiences (state, action, reward, next state, done) for training. The buffer has a fixed capacity and uses a deque for efficient storage and sampling.

Implementing the DQN model

- **DQN Agent**: The DQNAgent class includes:
  - **Action Selection**: An epsilon-greedy policy for action selection, balancing exploration and exploitation.
  - **Learning**: The agent learns from experiences through backpropagation, using the Adam optimizer with a learning rate of 0.0001.
  - **Target Network**: A target network is periodically updated to stabilize learning, with updates occurring every 10 episodes.
  - **Epsilon Decay**: Epsilon decays over time to reduce exploration, starting at 1.0 and decaying to 0.01 with a decay rate of 0.995.
- **Training**: The agent is trained over 200 episodes, with periodic updates to the target network, model saving every 2 episodes, and progress visualization using matplotlib.
- **Evaluation**: The trained agent is evaluated by running it in the environment for 5 episodes, capturing frames for visualization, and assessing its performance using a greedy policy.
- **Video Generation**: Frames from the evaluation are saved as an MP4 video using imageio, with a frame rate of 30 fps, to visually demonstrate the agent's performance.

Training Process





Challenges Faced by Us

1) Training Time increases significantly as the state space of the environment increases, even with the use of GPU's, making the process of hyperparameter tuning difficult.

2) Deep Q-Learning does not always guarantee convergence, which may lead to instability and degradation of performance after a few training episodes.

3) After 100 episodes, there were wide oscillations in performance, probably due to larger learning rate. However, decreasing it, wasn't feasible given limited time and resources.

Conclusions and Future Work

1. Effectiveness of DQN

The DQN implementation successfully learns to control the car in the CarRacing-v2 environment, achieving rewards of 600-8-00 points after training. This aligns with findings that DQNs can handle complex 2D environments through pixel-based inputs

2. Critical Design Choices

Frame Stacking and Grayscale Preprocessing significantly improved learning efficiency. Target network synchronization (every 10 episodes) stabilized training

3. Limitations

Discrete action space leads to suboptimal control compared to continuous alternatives. High computational demands for pixel-based training.

Conclusions and Future Work

Future work could include use of **Double DQN,** which reduces Q-value optimisation, use of PPO/continuous RL for better throttling and steering control. Transfer Learning can be used to accelerate training on new tracks.

Environment enhancements could be multiple car scenarios and implement traffic density variations. There could real world tests on small scale physical vehicles with lidar/camera fusion.

Evaluation metrics could be improved by tracking near miss rates, incorporating fuel consumption for reward design, developing shared control mechanisms with humans.

Thank you

Done and Dusted by:

Mir Maiti

Chandan Jyoti Das

Ishaan Bahl

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It’s free)