Report: Decision Making for Autonomous Vehicles on Highways --Group 41

Github: https://github.com/yYang-bit/ME5418.git

Parameter Summary Table:

In this project, two neural networks are used, DQN and A2C. We are going to compare how each method performs during training. By exploring these two classic reinforcements learning approaches, we can observe their differences in learning effectiveness, stability, and efficiency. This comparison will help us understand which network might be better suited for specific tasks.

Parameters	Definition	A2C	DQN
state_size	Size of the input state (observation space dimension)	6	6
action_size	Number of possible actions (action space size)	5	5
gamma	Discount factor for future rewards	0.99	0.99
epsilon	Initial exploration rate	0.1	1.0t to 0.1
epsilon_decay	Exploration rate decay factor	-	0.995
memory_size	Maximum size of the experience replay buffer	10000	10000
batch_size	Number of samples per training batch	64	64
lr	Learning rate for optimizer	0.001	0.001

- **Parameter Choices for A2C and DQN**

state_size and action_size: Defined by the environment; for example, state_size=6 means 6 input features, and action_size=5 means 5 possible actions.

learning rate (lr): Set around 0.001 for stable updates; typical values range from 0.0001 to 0.01.

epsilon: In A2C, set low (like 0.1) for occasional exploration. In DQN, starts high (1.0), decays over time (0.995), with a minimum of 0.1.

gamma (discount factor): Usually 0.99 to prioritize long-term rewards, common values are between 0.9 and 0.99.

batch_size: Commonly set to 64 for stable learning from sampled experiences.

memory_size: Usually 10000, for storing past experiences in the replay buffer.

learning rate, gamma, epsilon, batch_size and memory_size may be changed based on the training performance.

- **Adam optimizer**

Both A2C and DQN use the Adam optimizer because it's good at adjusting itself during training, making learning smoother and faster.

Why Adam? Adam changes the learning speed for each part of the model as needed, which means less manual tuning. In addition, it combines techniques that make learning more stable, which is important because reinforcement learning data can change as the agent explores.

A2C (Advantage Actor-Critic)

- The model has a **shared feature extraction layer** consisting of a linear layer with 64 units followed by a ReLU activation. This layer processes the input state and extracts features common to both the actor and critic networks.
- The **actor network** takes these shared features as input and outputs a probability distribution over the actions, achieved with a linear layer followed by a softmax activation function. The size of this output layer matches which represents the action space.
- The **critic network** also takes the shared features as input and produces a single scalar value, representing the value estimate of the input state.
- This **agent** class defines the methods needed to interact with the environment using the ActorCritic model, as well as to perform training through backpropagation, including **act** and **train**.

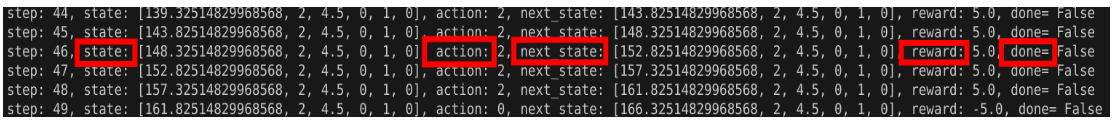
DQN (Deep Q-Network)

- Model: Input Layer: Maps the state to 64 features, followed by a ReLU activation. Hidden Layer: Another 64 features, also followed by ReLU. Output Layer: Outputs Q-values for each action (5 values for action_size=5).
- remember Method: Stores experiences in a replay memory buffer (state, action, reward, next state, and done).
- **act Method**: Chooses actions with epsilon-greedy strategy, starting with more random actions (epsilon=1.0) and reducing randomness over time
- **replay Method**: The main training process. Samples experiences from memory, calculates Q-values, and updates the model by minimizing the difference between predicted and target Q-values.
- **Epsilon Decay**: Epsilon gradually reduces (epsilon_decay=0.995) after each episode, shifting focus from exploration to using learned knowledge

Libraries and Tools Utilized

- PyTorch: Used for building and training neural networks.
- Numpy: Handles numerical operations and data.
- Collections (deque): Manages replay memory in DQN.
- Torch.distributions (Categorical): Supports action sampling in A2C.
- Simulator: SUMO to simulate environments for training.

Result show:



```
step: 44, state: [139.32514829968568, 2, 4.5, 0, 1, 0], action: 2, next_state: [143.82514829968568, 2, 4.5, 0, 1, 0], reward: 5.0, done= False
step: 45, state: [143.82514829968568, 2, 4.5, 0, 1, 0], action: 2, next_state: [148.32514829968568, 2, 4.5, 0, 1, 0], reward: 5.0, done= False
step: 46, state: [148.32514829968568, 2, 4.5, 0, 1, 0], action: 2, next_state: [152.82514829968568, 2, 4.5, 0, 1, 0], reward: 5.0, done= False
step: 47, state: [152.82514829968568, 2, 4.5, 0, 1, 0], action: 2, next_state: [157.32514829968568, 2, 4.5, 0, 1, 0], reward: 5.0, done= False
step: 48, state: [157.32514829968568, 2, 4.5, 0, 1, 0], action: 2, next_state: [161.82514829968568, 2, 4.5, 0, 1, 0], reward: 5.0, done= False
step: 49, state: [161.82514829968568, 2, 4.5, 0, 1, 0], action: 0, next_state: [166.32514829968568, 2, 4.5, 0, 1, 0], reward: -5.0, done= False
```