School of Computing, Engineering & Digital Technologies
Teesside University
Middlesbrough TS1 3BA

Artificial Intelligence Foundations

(CIS4011-N-FJ1)

Title: Reinforcement Learning for Cart-Pole Control using Deep Q-Networks

Submitted by: - Shubham Satish Kumar Mishra
Student Id: - W9641416

Reinforcement Learning for Cart-Pole Control using Deep Q-Networks

Abstract:

Reinforcement Learning (RL) is a crucial paradigm in machine learning, encapsulating a dynamic method in which an agent refines decision-making skills through iterative interactions with an environment (Merchán, 2023). The essence of RL is the agent's ability to determine appropriate behaviours based on its current state and elicit consequential reactions from the environment in the form of rewards or punishments. The agent's main purpose, ensconced inside this iterative process, is to strategically maximise its cumulative reward over the duration of its interactions. This fundamental premise emphasises RL's importance as a method for autonomous entities to learn and adjust their behaviour, with far-reaching consequences for applications in a variety of disciplines.

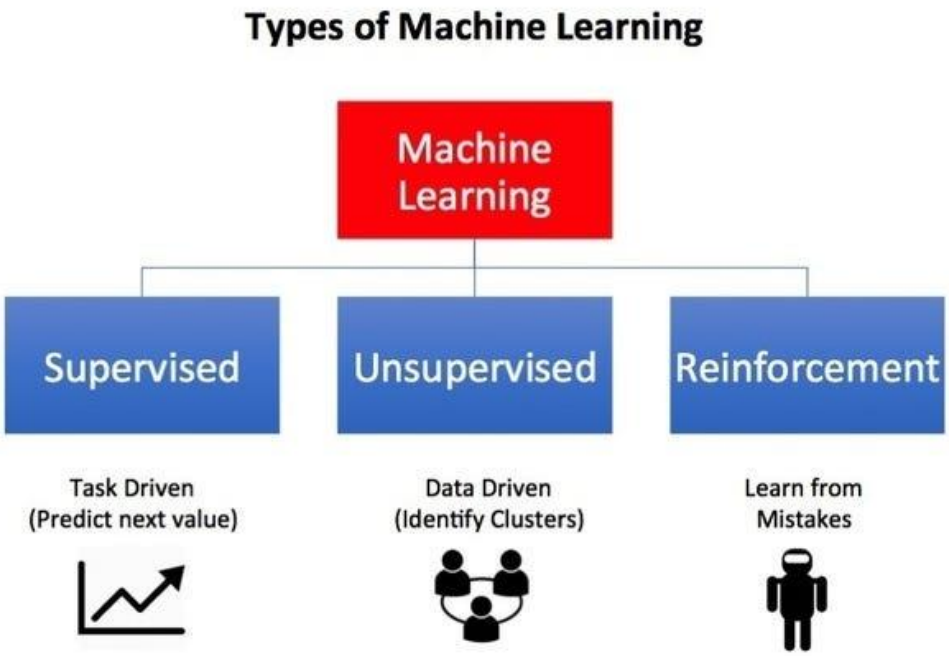
Deep learning (DL) and reinforcement learning (RL) methodologies appear to be necessary components for developing human-level or super-human AI systems. However, both DL and RL have substantial links with human brain functioning and neuroscientific studies (Yutaka Matsuo, 2022). Machine learning, on the other hand, is a fast-expanding subject of computer science with considerable applications (Murphy, 2012). Machine learning methods have been successfully applied in a wide range of fields, including pattern recognition, computer vision, entertainment, and medical applications, to automate the detection of patterns in data and to use those patterns to predict future data or other outcomes of interest (Friedman, 2009) (Murphy, 2015).

Machine learning programmes may be developed in a variety of languages, the most common of which being Python (Coursehero.com, 2021). This is due to the fact that it includes a number of built-in libraries and modules for continuous data processing and basic code 3 generation, eliminating the need for machine learning experts to start from scratch and allowing machine learning professionals to focus on the techniques that best suit their needs (Coursehero.com, 2021). Python programming language was also employed in the given study.

There are two types of machine learning: supervised and unsupervised learning, with a third, less widely employed kind known as reinforcement learning (Murphy, 2012). The purpose of supervised learning is to create an artificial system that can learn the mapping between input and output and anticipate the system's output given fresh inputs (Liu, 2012). Unsupervised learning models, on the other hand, function independently to find the fundamental structure of unlabelled data (Delua, 2021).

Reinforcement Learning

Reinforcement Learning (RL) is a machine learning approach that allows an agent to learn in an interactive environment through trial and error based on feedback from its own actions and experiences (Bhatt, 2018). An agent learns to interact with its environment through reinforcement learning by performing actions and evaluating the rewards obtained from those actions. Rather than being taught what to do, it creates its own training data and learns the consequences of its own actions via trial and error (Coursera [online], n.d.).



(Fig-1) Types of Machine Learning (Bhatt, 2018)

Though both supervised and reinforcement learning use mapping between input and output, reinforcement learning uses rewards and punishments as signals for positive and negative behaviour (Bhatt, 2018).

Reinforcement Learning Model

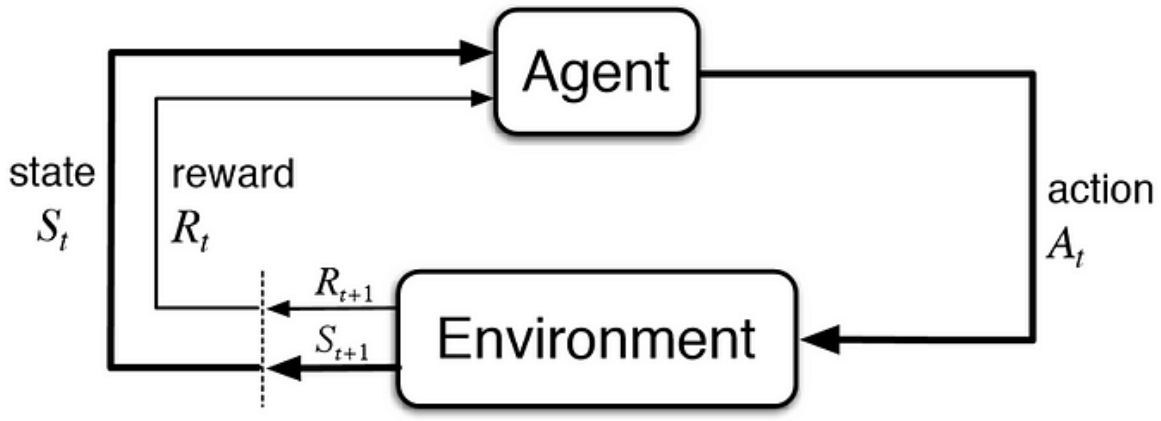


Fig-2: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

Model-based RL algorithms construct a model of the environment by sampling states, acting, and monitoring rewards. The model forecasts the expected reward and the predicted future state for each state and feasible action. The first is a regression problem, whereas the second is a density estimation problem (Synopsys, n.d.).

Some key terms that describe the basic elements of an RL problem are:

- Environment — Physical world in which the agent operates.
- State — Current situation of the agent
- Reward — Feedback from the environment
- Policy — Method to map agent's state to actions.
- Value — Future reward that an agent would receive by taking an action in a particular state.

Elements of Reinforcement Learning

Policy

A reinforcement learning agent's policy (π) is the technique through which it decides what to do next based on the present state (Sutton, 2018).

Reward.

This is the immediate response from the environment to evaluate the agent's last action. Positive rewards are given for correct actions, while negative rewards are given for incorrect ones (Sutton, 2018).

Value function

This is the discounted long-term return, as opposed to the reward with immediate benefits (Sutton, 2018). Values must be assessed and re-assessed during the course of an agent's existence, whereas rewards are delivered instantaneously by the environment (Sutton, 2018).

Action-value (Q)

This is similar to Value but requires an extra parameter called the current action a. The long-term return of the current state, which takes action a under policy π, is denoted by the parameter Qπ (s, a) (Sutton, 2018).

Markov Decision Process (MDP)

A Markov decision process is a framework for representing a reinforcement learning problem's environment. It is a graphical model with directed edges (one node of the graph points to another) (Saito et al., 2018). Each node in the environment represents a conceivable state, and each edge leading out of a state indicates an action that may be conducted in the given state.

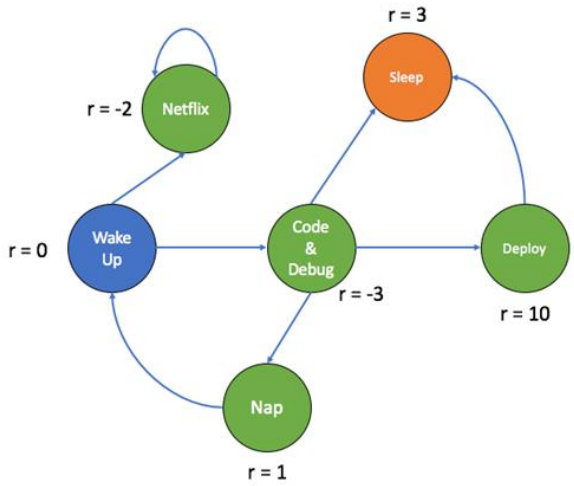


Fig-3

The five key ideas of this model are that a decision-making agent functions in an environment that changes state at random in response to its actions. It is analogous to the multi-armed bandit issue under consideration in that both seek to maximise cumulative rewards or decrease regrets for an agent in the environment (Howard, 1964).

Cart-Pole problem.

Cart-Pole, also known as inverted pendulum, is a game in which you attempt to keep the pole balanced for as long as possible. It is presumed that there is an item at the top of the pole, making it unstable and prone to tipping over. The purpose of this assignment is to manoeuvre the cart left and right such that the pole may stand as long as possible (within a specific angle) (Phy, 2019).

Cartpole is based on a Markov chain model and is one of the simplest settings in the OpenAI gym (PyLessons, 2019). The agent receives an observation (i.e., the condition of the environment) at each step, makes an action, and typically receives a reward (the frequency with which an agent receives a reward depends on the task or problem). A sequence of activities from an initial observation up to either a "success" or "failure" leading the environment to achieve its "done" state is referred to as an episode by agents. The learning component of an RL framework teaches agents whose actions (i.e., sequential decisions) maximise their long-term, cumulative rewards (Mika, 2021).

Brief Working of Cart-Pole environment

The agent must choose between pushing the cart left or right in order to keep the pole attached to it erect. As the agent examines the present state of the and selects an action, the environment is changed, and a reward that shows the implications of the action is returned. The rewards in this assignment are +1 for each timestep and the environment if the pole falls

over too far or the waggon goes more than 2.4 units away from the centre. This means that better performing scenarios will run for a longer period of time with a higher return.

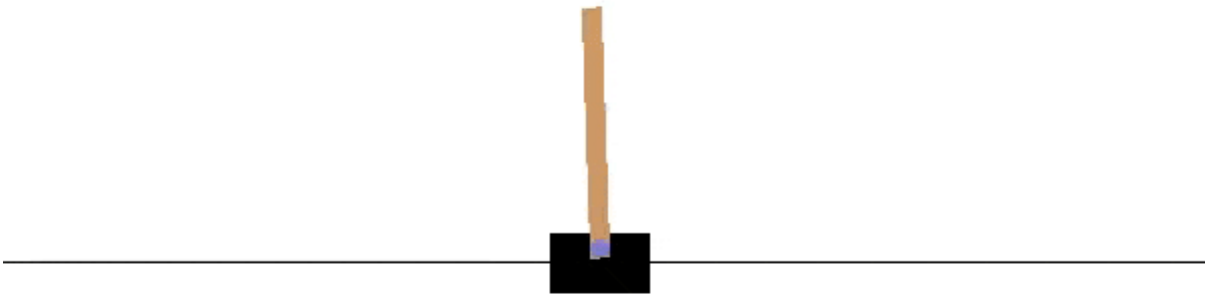


Fig-4

The Cart-Pole job is structured so that the agent's inputs are four real values indicating the state (position, velocity, and so on). We take these four unscaled inputs and route them via a tiny network with two outputs, one for each activity. Given the input state, the network is trained to predict the anticipated value for each action. After that, the action with the highest predicted value is picked.

Methodology.

Imports

1. TensorFlow:

The Google Brain team created TensorFlow, an open-source machine learning library. It provides a versatile and efficient environment for developing and training a wide range of machine learning models, including deep neural networks. TensorFlow is a popular deep learning framework that allows for the training and deployment of machine learning models.

1.1 Defining Neural Network: The neural network that approximates the Q-function for the Cart-Pole issue is defined using TensorFlow. The Q-function, which represents the predicted cumulative reward of doing a certain action in a given condition, is a crucial concept in reinforcement learning.

```
model = build_q_network(state_size, action_size)
```

In this case, build_q_network is a function that generates a neural network with the given input (state_size) and output (action_size). The Q-function is approximated using this neural network.

1.2 Training the Neural Network: The Q-learning technique is used to train the neural network with TensorFlow. The model is trained to reduce the disparity between predicted and goal Q-values. The weights of the neural network are updated during the training phase depending on the observed rewards and the estimated Q-values.

```
loss, _ = sess.run([model.loss, model.optimizer],
                   feed_dict={model.input_state: states,
                              model.target_Q: target_Qs,
                              model.actions: actions})
```

The model.loss denotes the loss function used to measure the difference between predicted and target Q-values, and model.optimizer is an optimizer that minimises this loss throughout the training procedure.

2 Gym:

OpenAI's gym library is a toolbox for constructing and evaluating reinforcement learning systems. It offers a diverse range of scenarios, from basic toy puzzles to complicated physics-based simulations.

The use of the gym library to generate the Cart-Pole environment aids in the scenario in which the reinforcement learning agent learns to take behaviours that maximise its cumulative reward in this unique challenge setting. The agent's interaction with the environment entails monitoring its state, acting, obtaining rewards, and modifying the agent's policy or value function depending on this experience. Because of its simplicity and efficacy for testing basic ideas, the Cart-Pole environment is a popular choice for learning and experimenting with reinforcement learning algorithms.

time module: The time module in Python includes a variety of time-related methods, and it is mostly utilised in this code to provide delays or sleep intervals throughout the training loop.

The sleep function of the time module is used to suspend the program's execution for a defined number of seconds. This can allow you manage the tempo of the training loop, which is especially useful if you wish to visualise or track the training progress.

Visualization: If visualisations or plots are being created during training, adding delays might slow down the presentation, making it simpler to see changes over time.

Monitoring: It enables more convenient tracking of training progress, particularly when dealing with frequent updates or changes in the environment.

Resource Management: Delays may assist manage system resources more efficiently during training in specific instances. NumPy is a strong Python module for numerical calculations. It supports massive, multi-dimensional arrays and matrices, as well as a set of mathematical functions for effectively operating on these components.

3 numpy (as np)

NumPy is a strong Python module for numerical calculations. It supports massive, multi-dimensional arrays and matrices, as well as a set of mathematical functions for effectively operating on these components.

NumPy is utilised for a variety of numerical computations, particularly when dealing with state and action representations. As an example:

State Representation: The Cart-Pole environment's state, which contains the cart's location and velocity as well as the angle and angular velocity of the pole, is frequently represented as a NumPy array.

```
state = np.array([cart_position, cart_velocity, pole_angle, pole_angul
```

Action Representation: Reinforcement learning actions are frequently discrete and may be described using integers. NumPy arrays may be utilised to effectively store and process these action representations.

```
action = np.random.choice(num_actions)
```

The number of available actions is num_actions, and np.random.choice is used to select a random action.

Q-Value Representations: NumPy arrays may also be used to store and modify Q-values, which indicate the estimated cumulative future rewards for doing a certain action in a given state.

```
q_values = np.array([q_value_for_action_1, q_value_for_action_2,
```

4 **collections module:** Python's collections module extends built-in types with extra functionality.

deque: A deque is a double-ended queue that allows you to add and remove entries from both ends in O (1) time. Experience replay is a prominent strategy in reinforcement learning, particularly when implementing algorithms such as Deep Q Networks (DQN). A replay buffer holds previous experiences (state, action, reward, future state) to break the temporal correlation in the data and increase the learning process's stability.

```
replay_buffer = deque(maxlen=buffer_size)
```

The maximum capacity of the replay buffer is determined by buffer_size. The oldest experience is immediately erased when the buffer is full, and a fresh experience is inserted. This guarantees that the replay buffer has a consistent size history of recent events. During the training phase, experiences are added to the replay buffer and randomly sampled during neural network training to update the Q-function.

In this context, the deque provides an effective and straightforward mechanism to manage the replay buffer in terms of both adding new experiences and sampling them during the training process. It aids in striking a balance between memory efficiency and the efficient utilisation of prior events for training the reinforcement learning agent.

5 **matplotlib:** Matplotlib is a popular Python charting package. The pyplot package provides an easy-to-use interface for constructing various plots and charts. It's used in this code to visualise the training outcomes, notably to plot the total rewards across episodes.

The code maintains track of the total rewards collected in each episode throughout the training loop, and after training, it uses matplotlib.pyplot to build a plot that displays how the total rewards vary across episodes. This type of plot is frequently used to evaluate the development of reinforcement learning agents.

```
plt.plot(episode_rewards)
plt.title('Training Progress')
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.show()
```

This plots the total rewards collected in each episode, offering a visual depiction of the agent's learning progress. The x-axis depicts episodes, while the y-axis depicts the total prize collected in each episode. The plot aids in comprehending how effectively the agent learns over time.

Environment Setup:

```python
import gym

# Create CartPole environment
env = gym.make('CartPole-v1')

# Get details of the environment
action_space_size = env.action_space.n
state_space_size = env.observation_space.shape[0]

print(f"Number of possible actions: {action_space_size}")
print(f"State space dimension: {state_space_size}")
```

The above code initializes the Cart-Pole environment using Gym and prints some basic details:

```
Number of possible actions: 2
States: Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235
e+38], (4,), float32)
[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
```

- **action_space_size:** The number of possible actions the agent can take.
- **state_space_size:** The dimensionality of the state space.

DQN Algorithm:

```
class QNetwork:
    def __init__(self, learning_rate=0.01, state_size=4,
                 action_size=2, hidden_size=10,
                 name='QNetwork'):
        with tf.compat.v1.variable_scope(name):
            # state inputs to the Q-network
            self.inputs_ = tf.compat.v1.placeholder(tf.float32, [None, state_size], name='inputs')

            # One hot encode the actions to later choose the Q-value for the action
            self.actions_ = tf.compat.v1.placeholder(tf.int32, [None], name='actions')
            one_hot_actions = tf.one_hot(self.actions_, action_size)

            # Target Q values for training
            self.targetQs_ = tf.compat.v1.placeholder(tf.float32, [None], name='target')

            # ReLU hidden layers
            self.fc1 = fully_connected("h1", self.inputs_, hidden_size)
            self.fc2 = fully_connected("h2", self.fc1, hidden_size)
            self.fc3 = fully_connected("h3", self.fc2, hidden_size)

            # Linear output layer
            self.output = fully_connected("output", self.fc3, action_size, activation=None)

            ### Train using mean squared error and Adam gradient descent.
            self.Q = tf.reduce_sum(tf.multiply(self.output, one_hot_actions), axis=1)
            self.loss = tf.reduce_mean(tf.square(self.targetQs_ - self.Q))
            self.opt = tf.compat.v1.train.AdamOptimizer(learning_rate).minimize(self.loss)
```

This code defines a QNetwork class using TensorFlow. Let us break down the architecture:

- **self.dense1:** The first dense (completely linked) layer activated by ReLU. It accepts the state as input and has hidden_size neurons. The use of ReLU as the activation function causes the network to become nonlinear.
- **self.dense2:** The second thick layer is activated by ReLU. It keeps capturing complicated patterns in the incoming data.
- **self.output_layer:** The output layer is devoid of any activation function. This layer generates Q-values for each action, which indicate the estimated cumulative future reward for each action based on the current state.

The network's forward pass is defined by the call method. It takes a state as an input and runs it through the defined layers to generate Q-values.

Memory Class:

The Memory class is commonly used in reinforcement learning to store and manage events for a process known as "experience replay." Experience replay is a technique used in deep reinforcement learning models to disrupt the temporal association between subsequent events and increase the stability and efficiency of training.

Purpose: The Memory class's principal function is to keep a buffer of prior events encountered by the agent throughout its interactions with the environment.

Implementation: It is frequently implemented as a circular buffer or a deque (two-ended queue) with a defined maximum capacity. The fixed-size buffer aids in retaining just the most recent experiences while eliminating older ones when new ones are added.

Attributes:

buffer: The container to store experiences, usually implemented as a deque.

capacity: The maximum number of experiences the buffer can hold.

Methods:

add_experience: Adds a new experience to the buffer. If the buffer is at full capacity, it may overwrite the oldest experience.

sample: Randomly samples a batch of experiences from the buffer. This batch is then used for training the neural network.

```
from collections import deque
import random

class Memory:
    def __init__(self, capacity):
        self.buffer = deque(maxlen=capacity)
        self.capacity = capacity

    def add_experience(self, experience):
        self.buffer.append(experience)

    def sample(self, batch_size):
        batch = random.sample(self.buffer, batch_size)
        return batch
```

Role in Training:

Experience Replay: Instead of learning from consecutive events (which might generate correlations and lead to instability), the agent samples a batch of experiences at random from the memory buffer throughout the training phase. This aids in breaking down the temporal linkage of events.

Improved Stability: Experience repeat frequently leads to more consistent and efficient learning. It enables the agent to learn from a wide range of experiences, limiting the influence of outliers or samples that are highly linked.

Pretraining the Memory:

```
# Pretrain the memory with random actions
pretrain_length = 128
for _ in range(pretrain_length):
    # Take a random action
    action = env.action_space.sample()

    # Perform the action and observe the next state, reward,
    next_state, reward, done, _ = env.step(action)

    # Modify the reward based on cart position
    reward = reward if abs(next_state[0]) < 2.4 else -10.0

    # Add the experience to the replay memory
    memory.add((state, action, reward, next_state, done))

    # If the episode is done, reset the environment
    if done:
        state = env.reset()
    else:
        state = next_state
```

The above code sample uses random events to pretrain the replay memory. It performs pretrain_length random operations and records the resulting experiences in memory. Here are a few crucial points:

- **env.action_space.sample():** Randomly samples an action from the action space.

- **reward = reward if abs(next_state[0]) < 2.4 else -10.0:** Changes the prize dependent on the position of the cart. If the cart is within a specific range, the original prize is retained; otherwise, a -10.0 penalty is applied.

- **memory.add((state, action, reward, next_state, done)):** Adds the experience tuple (state, action, reward, next_state, done) to the replay memory.

Training Setup:

```
# Training hyperparameters
num_episodes = 1000
max_steps_per_episode = 1000
exploration_max = 1.0
exploration_min = 0.01
exploration_decay = 0.995
gamma = 0.99
learning_rate = 0.001

# Network parameters
state_size = state_space_size
action_size = action_space_size
hidden_size = 64

# Memory parameters
memory_size = 10000
batch_size = 64
```

These hyperparameters and settings are crucial for training the Q-network:

- **num_episodes:** The total number of episodes during training.
- **max_steps_per_episode:** The maximum number of steps the agent can take in a single episode.
- **exploration_max, exploration_min, exploration_decay:** Parameters for controlling the exploration-exploitation trade-off using epsilon-greedy strategy.
- **gamma:** The discount factor for future rewards.
- **learning_rate:** The learning rate for the Q-network.
- **state_size, action_size, hidden_size:** Parameters defining the neural network architecture.
- **memory_size:** The maximum size of the replay memory.

- **batch_size:** The number of experiences sampled from the replay memory in each training iteration.

Training Loop:

The training loop is where the agent interacts with the environment, utilizes the Q-network to make decisions, and updates its knowledge through experience replay.

```
for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0

    for step in range(max_steps_per_episode):
        # Exploration-exploitation trade-off
        exploration_rate = exploration_min + (exploration_max - explora

        # Choose an action using epsilon-greedy strategy
        if np.random.rand() < exploration_rate:
            action = env.action_space.sample()  # Explore
        else:
            action = np.argmax(qn.predict(state.reshape((1, state_size)

        # Perform the action and observe the next state, reward, and do
        next_state, reward, done, _ = env.step(action)

        # Modify the reward based on cart position
        reward = reward if abs(next_state[0]) < 2.4 else -10.0

        # Add the experience to the replay memory
        memory.add((state, action, reward, next_state, done))

        # Update the Q-network weights through experience replay
        qn.train(memory.sample(batch_size), gamma)

        # Accumulate the total reward
        total_reward += reward
```

Key points:

- **exploration_rate**: Using an epsilon-greedy method, the exploration rate is changed throughout training. It begins high and falls gradually, allowing the agent to explore more in the early phases and eventually apply what it has learnt.
- **if np.random.rand() < exploration_rate:** This condition determines whether the agent should investigate (perform a random action) or exploit (do the action with the greatest Q-value).
- **qn.train(memory.sample(batch_size), gamma):** The Q-network is trained by randomly choosing experiences from the replay memory and updating its weights with the Bellman equation.
- **print(f"Episode: {episode + 1}, Total Reward: {total_reward}"):** Displays the total reward achieved in each episode.

Checkpointing and Model Saving:

The Saver object in TensorFlow is used in this code to store the learned model at the end of training. The model is stored as "cartpole_model.ckpt" in the current working directory. The agent can then reload the learned model for further assessment or usage.

```
# Save the trained model
saver = tf.compat.v1.train.Saver()
save_path = saver.save(sess, "./cartpole_model.ckpt")
print(f"Model saved in path: {save_path}")
```

Monitoring Training Progress:

In the training loop, the progress is monitored by printing the total reward achieved in each episode. Additionally, the loop breaks if the environment signals that the episode is done.

Visualization of Training Results:

The training outcomes are visualised using Matplotlib, especially the total rewards over episodes. This aids in determining how effectively the agent is learning and whether it is progressing. The plotting code is as follows:

```
# Plotting the training results
plt.plot(episode_rewards, label='Total Reward per Episode')
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.title('Training Results - CartPole')
plt.legend()
plt.show()
```

Key points:

- **plt.plot(episode_rewards, label='Total Reward per Episode'):** Plots the total rewards for each episode.
- **plt.xlabel('Episode') and plt.ylabel('Total Reward'):** Adds labels to the x and y-axes, respectively.
- **plt.title('Training Results - CartPole'):** Sets the title of the plot.
- **plt.legend():** Displays the legend for better interpretation.
- **plt.show():** Finally, displays the plot.

Watching the Trained Agent:

In order to observe the trained agent in action, we have returned the model to make decisions in the CartPole environment. This entails loading the previously saved model and predicting actions depending on the current condition of the environment.

1 Load the Saved Model:

```
saver = tf.compat.v1.train.Saver()
with tf.compat.v1.Session() as sess:
    saver.restore(sess, model_path)
```

2 Initialize the Environment:

```
env = gym.make("CartPole-v1")
state = env.reset()
```

3 Run the Model to Make Decisions:

```
done = False
while not done:
    # Reshape the state to match the model's input shape if needed
    state = np.reshape(state, [1, state_size])

    # Use the Q-network to predict the Q-values for each action
    q_values = sess.run(q_network.output, feed_dict={q_network.input_

    # Choose the action with the highest Q-value
    action = np.argmax(q_values)

    # Take the chosen action and observe the next state and reward
    next_state, reward, done, _ = env.step(action)

    # Update the current state for the next iteration
    state = next_state

    # Render the environment to visualize the agent's actions
    env.render()
```

Results:

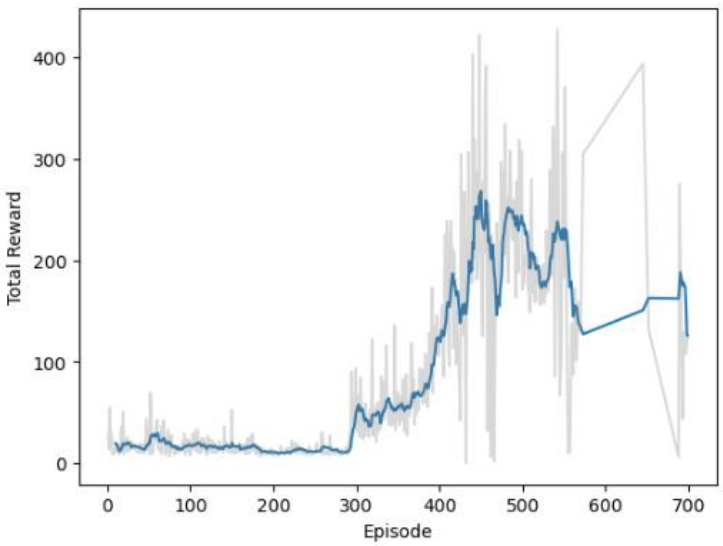
The output sample shows the total incentives collected by the agent throughout the training phase over numerous episodes. The rewards rise steadily as the agent's skill to balance the pole on the cart improves. Furthermore, as episodes go, the exploration probability (Explore P) declines, suggesting a shift from exploration to exploitation.

```
Episode: 551 Total reward: 370.8480749316516 Explore P: 0.0100
Episode: 553 Total reward: 159.14106590176618 Explore P: 0.0100
Episode: 555 Total reward: 10.118781303366024 Explore P: 0.0100
Episode: 557 Total reward: 10.724047680695854 Explore P: 0.0100
Episode: 559 Total reward: 137.28083343307165 Explore P: 0.0100
Episode: 561 Total reward: 87.1537390227119 Explore P: 0.0100
Episode: 563 Total reward: 172.16861944148937 Explore P: 0.0100
Episode: 565 Total reward: 101.31612242013217 Explore P: 0.0100
Episode: 567 Total reward: 159.12118612850708 Explore P: 0.0100
Episode: 570 Total reward: 130.03493661805987 Explore P: 0.0100
Episode: 573 Total reward: 304.7531047475836 Explore P: 0.0100
Episode: 645 Total reward: 393.82743198921276 Explore P: 0.0100
Episode: 652 Total reward: 130.98378572861353 Explore P: 0.0100
Episode: 688 Total reward: 6.175450623035425 Explore P: 0.0100
Episode: 689 Total reward: 275.2794626826856 Explore P: 0.0100
Episode: 690 Total reward: 206.5071623828723 Explore P: 0.0100
Episode: 693 Total reward: 43.608704616626106 Explore P: 0.0100
Episode: 694 Total reward: 129.06130906217737 Explore P: 0.0100
Episode: 695 Total reward: 122.24567544545667 Explore P: 0.0100
Episode: 696 Total reward: 125.03790964368577 Explore P: 0.0100
Episode: 697 Total reward: 108.1782961736899 Explore P: 0.0100
Episode: 698 Total reward: 119.7825043617584 Explore P: 0.0100
Episode: 699 Total reward: 123.13716393323556 Explore P: 0.0100
```

The restored model is then used to observe the agent's behaviour, as seen in the subsequent snippet. The agent successfully maintains a balanced pole, confirming the effectiveness of the trained neural network.

1. **Training Progress:**

The increasing total rewards during training suggest that the agent is learning a successful policy for the CartPole environment. This aligns with the fundamental goal of reinforcement learning, where the agent refines its decision-making over time.



2. **Exploration-Exploitation Trade-off:**

The decreasing exploration probability is indicative of the agent's shift from exploring the environment to exploiting its learned policy. This dynamic adjustment is crucial for achieving a balance between exploring new actions and exploiting known successful actions.

3. **Model Generalization:**

The model's capacity to generalise is demonstrated by its ability to keep the pole balanced during the observation phase. Generalisation is an important part of reinforcement learning since the agent must not only memorise certain instances but also adapt to new, unanticipated ones.

4. **Comparison with Other Solutions:**

To assess the relevance of these findings, they must be compared to current solutions and standards in the area. Examining the agent's performance in comparison to alternative algorithms or variants of the stated technique gives context and insights into its efficacy.

5. **Acknowledging Limitations:**

It is critical to recognise the limits of the material given. The present implementation, for example, may be sensitive to hyperparameter settings, and the CartPole environment is rather simplistic. Evaluating the agent's performance in more complicated situations and under a wider range of scenarios may offer a more complete picture of its capabilities.

6. **Avoiding Exaggeration:**

While the results obtained are encouraging, it is vital not to overestimate their relevance. The CartPole issue is a well-known and well-studied challenge, and the solution described here is a first step. The agent's performance in this setting does not always suggest that he or she will solve more complex challenges.

```
watch_agent(3)
```

```
INFO:tensorflow:Restoring parameters from C:/Users/Shubham/Downloads/.ipynb_checkpoints/CartPole.ckpt
reward: 9
reward: 10
reward: 9
```

Conclusion:

Finally, the given findings show that reinforcement learning approaches may be successfully used to the CartPole problem. The agent demonstrates learning behaviour, and the taught model is effective at maintaining equilibrium. However, additional study and testing are required to apply these findings to more complicated circumstances and improve the agent's skills. The conversation emphasises a balanced viewpoint, recognising accomplishments

while keeping in mind the difficulties and prospects for future research in the larger area of AI.

References

Bhatt, S., 2018. *Reinforcement Learning 101.* [Online]
Available at: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292
[Accessed 06 January 2024].

Coursehero.com, 2021. . *Machine Learning is the learning in which machine can learn by its own without being explicitly programmed..* [Online]
Available at: https://www.coursehero.com/tutors-problems/Computer-Science/30025816-Machine-Learning-is-the-learning-in-which-machine-can-learn-by-its-own/?justUnlocked=1
[Accessed 06 January 2024].

Coursera [online], n.d. *Sequential Decision Making with Evaluative Feedback.* [Online]
Available at: https://www.coursera.org/lecture/fundamentals-of-reinforcement-learning/sequential-decision-making-with-evaluative-feedback-PtVBs
[Accessed 06 January 2024].

Delua, J., 2021. *Supervised vs. Unsupervised Learning: What's the Difference?.* [Online]
Available at: https://www.ibm.com/blog/supervised-vs-unsupervised-learning/
[Accessed 06 January 2024].

Friedman, J. R. T. a. T. H., 2009. *The elements of statistical.* inference, and prediction. ed. New York: Springer.

Howard, R. A., 1964. *Dynamic programming and Markov processes.* Massachusetts Institute of Technology: s.n.

Liu, Q. W. Y., 2012. *Supervised Learning.* [Online]
Available at: https://doi.org/10.1007/978-1-4419-1428-6_451
[Accessed 06 January 2024].

Merchán, E. C. G., 2023. Why deep reinforcement learning is going to be the next big deal in AI.

Merchán, E. C. G., 2023. *Why deep reinforcement learning is going to be the next big deal in AI.* [Online]
Available at: https://medium.com/@eduardogarrido90/why-deep-reinforcement-learning-is-going-to-be-the-next-big-deal-in-ai-2e796bdf47d2
[Accessed 06 January 2024].

Merchán, E. C. G., 202. Why deep reinforcement learning is going to be the next big deal in AI.

Mika, M. G. a. S., 2021. *An Introduction to Reinforcement Learning with OpenAI Gym, RLlib, and Google Colab.* [Online]
Available at: https://www.anyscale.com/blog/an-introduction-to-reinforcement-learning-with-openai-gym-rllib-and-google
[Accessed 06 January 2024].

Murphy, I. E. N. &. M. J., 2015. *"What Is Machine Learning".* [Online]
Available at: https://doi.org/10.1007/978-3-319-18305-3_1
[Accessed 06 January 2024].

Murphy, K., 2012. *Machine learning - a probabilistic..* Adaptive computation and machine learning series, p 2 ed. Massachussets.: MIT Press Cambridge.

Phy, V., 2019. *Reinforcement Learning Concept on Cart-Pole with DQN.* [Online]
Available at: https://towardsdatascience.com/reinforcement-learning-concept-on-cart-pole-with-dqn-799105ca670
[Accessed 06 January 2024].

PyLessons, 2019. *Introduction to Reinforcement Learning.* [Online]
Available at: https://pylessons.com/CartPole-reinforcement-learning
[Accessed 06 January 2024].

Saito, S., Wenzhuo, Y. & Shanmugamani, R., 2018. *Python Reinforcement Learning Projects.* s.l.:Packt Publishing, Limited.

Sutton, R. a. B. A., 2018. *Reinforcement learning.*. 2nd ed. ed. Cambridge: Massachusetts: The MIT Press.

Synopsys, n.d. *What is Reinforcement Learning?.* [Online]
Available at: https://www.synopsys.com/ai/what-is-reinforcement-learning.html#7
[Accessed 06 January 2024].

Yutaka Matsuo, Y. L. ,. M. S. d. D. P. ,. D. S. ,. M. S. ,. E. U. h. J. M., 2022. *Deep learning, reinforcement learning, and world models.* [Online]
Available at: https://www-sciencedirect-com.ezproxy.tees.ac.uk/science/article/pii/S0893608022001150?via%3Dihub
[Accessed 06 January 2024].