Optimization of Travel Itineraries using Multi-Agent Deep Q-Networks (MADQN): The Qtrip Approach

Alessandro Barro
alessandro1.barro@mail.polimi.it

August 29, 2023

1 Introduction

1.1 Introduction to Multi-Agent Systems

Multi-Agent Systems (MAS) involve multiple decision-makers, termed agents, that are typically autonomous and can act in a cooperative, competitive, or neutral manner concerning other agents. The presence of multiple agents significantly complicates the decision-making landscape due to:

- **Non-Stationarity:** The learning environment dynamically evolves as each agent updates its policy.
- **Partial Observability:** Agents might not have complete information about the environment or the actions of other agents.
- **Multiple Learning Objectives:** Each agent may have its objectives, which might be in harmony, conflict, or orthogonal to those of other agents.

1.2 Multi-Agent Deep Q-Learning

MADQN, an extension of the traditional Q-Learning, seeks to equip each agent with the ability to make decisions considering the presence and strategies of other agents. Key aspects include:

- **Joint Action Value Function:** The Q-value, $Q(s, \mathbf{a})$, now depends on the joint action $\mathbf{a}$ taken by all agents.
- **Centralized Learning with Decentralized Execution:** Often, during training (learning), agents have access to global state and actions of all agents (centralized). But during execution (policy deployment), each agent acts based only on its local observations (decentralized).
- **Policy Coordination:** Agents can employ methods like value decomposition, shared policies, or other coordination mechanisms to harmonize their actions.

1.3 Challenges and Solutions in MADQN

Multi-agent scenarios often lead to issues not encountered in single-agent settings:

- **Credit Assignment Problem:** It becomes challenging to attribute the success (or failure) of an action to a particular agent when multiple agents influence the outcome.
  - **Solution:** Counterfactual analysis or difference rewards can be utilized to assess the individual contribution of each agent.
- **Exploration-Exploitation Dilemma:** The presence of multiple agents exacerbates the classic exploration versus exploitation problem in RL.
  - **Solution:** Techniques like optimistic initialization, noise injection, or entropy regularization can guide more effective exploration.

1.4 Conclusion on MADQN

MADQN presents a compelling solution to challenges arising in multi-agent scenarios. By combining the representational power of neural networks with a structured approach to handle multi-agent dynamics, MADQN paves the way for sophisticated decision-making in complex environments, like that of Qtrip's travel optimization.

2 Multi-Agent Deep Q-Networks (MADQNs)

Deep Q-Networks (DQN) have been instrumental in allowing the application of Q-learning to complex tasks with large state and action spaces by approximating the Q-value function with neural networks. When extending to the multi-agent domain, the scenario gets intricate due to the concurrent actions and strategies of multiple agents. Let's delve deeper into the formalisms.

2.0.1 Standard Q-learning Update Rule

The classic Q-learning update rule is formulated as:

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha \left[r + \gamma \max_{a'} Q^{\pi}(s',a') - Q^{\pi}(s,a) \right]. \tag{1}$$

Here, α denotes the learning rate, determining the weight given to the new estimate. The term inside the bracket is the Temporal Difference (TD) error.

2.0.2 MADQN Q-value Update Rule

In a multi-agent setting, the Q-value for an agent is influenced not just by its actions but also by the actions of the other agents in the system. Let's break down the MADQN Q-value update:

For each agent i:

$$Q_i^{\pi}(s,a_i,a_{-i}) \leftarrow Q_i^{\pi}(s,a_i,a_{-i}) + \alpha \left[r + \gamma \max_{a_i'} Q_i^{\pi}(s',a_i',a_{-i}') - Q_i^{\pi}(s,a_i,a_{-i}) \right]. \tag{2}$$

Here, a_i is the action of agent i and a_{-i} collectively represents the actions of all other agents.

The global Q-value in the multi-agent system is an aggregation of the Q-values of individual agents. The consolidated MADQN update rule becomes:

$$Q^{\pi}(s,a) \leftarrow \sum_{i=1}^{n} Q_i^{\pi}(s,a_i,a_{-i}) + \alpha \left[r + \gamma \max_{a_i'} Q_i^{\pi}(s',a_i',a_{-i}') - Q_i^{\pi}(s,a_i,a_{-i}) \right]. \tag{3}$$

The convergence of this equation depends on the nature of interactions between agents and the exploration strategies they employ.

3 Implementation Details

3.1 Dependencies and Setup

To ensure the smooth functioning of the Qtrip algorithm, setting up the correct environment is pivotal. This involves importing the right Python libraries and setting up global variables tailored to the problem.

- **Libraries:**
  - **TensorFlow:** TensorFlow is a versatile open-source machine learning framework. Within the context of the Qtrip algorithm, TensorFlow is primarily utilized for the construction, training, and evaluation of the Q-network. Its flexibility and efficiency make it an ideal choice for reinforcement learning applications.
  - **numpy:** numpy, short for Numerical Python, is one of the foundational packages for numerical computations in Python. It provides support for large multidimensional arrays and matrices, along with an assortment of high-level mathematical functions to operate on these arrays. For the Qtrip algorithm, numpy facilitates the manipulation and analysis of data structures, especially when handling state representations or reward calculations.
  - **geopandas:** While pandas is a renowned library for data manipulation and analysis, geopandas extends its functionalities to allow spatial operations. Given the geographical nature of the Qtrip algorithm, where city locations and distances between them play a pivotal role, geopandas becomes instrumental in performing geospatial calculations and manipulations.
- **Global Variables:**
  - `BATCH_SIZE`**:** This variable determines the number of training examples utilized in one iteration. A larger batch size can provide a more accurate estimate of the gradient, but at the cost of computational efficiency. It's a balance between speed and stability.
  - `TARGET_UPDATE_FREQ`**:** In Deep Q-Learning, two networks are used: the primary Q-network and the target Q-network. The target network's weights are periodically updated with the primary network's weights. This frequency of updating is determined by `TARGET_UPDATE_FREQ`. This mechanism introduces stability into the learning process.
  - `EXPLORATION_DECAY`**:** Exploration versus exploitation is a central theme in reinforcement learning. Initially, the agent explores the environment more, but as it learns, it exploits its knowledge. The rate at which exploration decreases is controlled by this variable, ensuring that the agent doesn't get stuck in local optima and continues to explore new strategies.

To sum up, the setup phase meticulously prepares the environment by leveraging powerful libraries and carefully chosen global variables, ensuring the algorithm functions optimally and efficiently.

3.2 Neural Network: Q-Network

The neural network in question is not just a regular neural network; it is a Q-network. The primary purpose of a Q-network is to approximate the Q-value function in reinforcement learning. Q-values signify the expected return or future reward of taking an action in a given state. Traditionally, Q-values are stored in tables, but in complex environments with vast state and action spaces, tabular methods are inefficient. Thus, neural networks are employed to approximate these Q-values for every possible action in a given state.

The architecture of the Q-network is structured to cater to the specific demands of the RL environment it operates in. Here's how the architecture is defined:

- **Model Architecture:**
  - **Input Layer:** The input to the Q-network is a representation of the environment's state. In this case, the dimensionality of the input layer depends on the number of cities and state attributes. This design ensures that the network understands the current state of the environment, which could be the current city or the attributes associated with it.

- **Hidden Layers:** These layers allow the Q-network to learn complex representations and relations between different states and actions. The chosen architecture consists of three Dense layers with 128, 64, and 32 neurons, respectively. The ReLU activation function introduces non-linearity, enabling the model to learn and approximate the Q-values accurately for different state-action pairs.
- **Output Layer:** The output layer's design is crucial in a Q-network. Each neuron in the output layer corresponds to a Q-value for a potential action. In this context, an action is equivalent to visiting a particular city. The linear activation ensures that the network can output a full range of Q-value estimates. When deciding on an action, the agent will typically choose the action corresponding to the neuron with the highest Q-value.

- **Compilation:** To train the Q-network, an optimization algorithm is required to adjust the network's weights based on the prediction error. The Adam optimizer is a popular choice due to its adaptive learning rates and efficient handling of large-scale problems. The loss function used is the Mean Squared Error (MSE). In the context of RL, the MSE measures the difference between the predicted Q-values and the target Q-values, which are typically obtained from the Bellman equation. Minimizing this loss ensures that the Q-network's predictions align closely with the expected future rewards.

In summary, the Q-network is a specialized neural network tailored for RL applications. It's designed to estimate the expected rewards of taking various actions in different states, guiding the agent towards optimal decision-making.

3.3 Experience Replay

Experience replay is a pivotal mechanism in deep reinforcement learning, particularly in the context of Q-learning with neural networks. Its primary purpose is to store experiences—an agent's state, action, reward, and next state—in a memory buffer. By sampling randomly from this buffer to train the agent, we can break the temporal correlations of consecutive experiences. This process leads to a more stable and robust learning experience.

Why Experience Replay? In a continuously changing environment, learning from consecutive experiences can introduce harmful correlations, causing the Q-values to oscillate or diverge. Experience replay allows the agent to learn from past experiences, which are stored in a buffer, providing a richer set of data to learn from and reducing the variance in updates.

3.3.1 ReplayBuffer Class

The `ReplayBuffer` class serves as the memory reservoir where experiences are stored and later sampled for the learning process.

Attributes:

- **capacity:** This represents the maximum number of experiences the buffer can hold. Once the buffer reaches this limit, older experiences are overwritten by new ones, ensuring a dynamic mix of old and recent experiences.
- **buffer:** A data structure, typically a list, where individual experiences are stored. Each experience is a tuple consisting of a state, action, reward, and the subsequent state.
- **position:** Indicates the current location in the buffer where a new experience will be added. It's a pointer that loops back to the start once the buffer's capacity is reached, ensuring a continuous storage mechanism.

Methods:

- **push():** This method is responsible for adding a new experience to the buffer. If the buffer is full, it replaces the oldest experience. It's a way of ensuring the agent has a fresh set of experiences to learn from, reflecting recent interactions with the environment.

- **sample():** Randomly selects a batch of experiences from the buffer. This randomness is crucial as it breaks the sequential nature of experiences, allowing the neural network to generalize better from diverse situations.

In essence, experience replay, implemented through the `ReplayBuffer` class, aids in the stabilization of deep Q-learning. It mitigates challenges arising from correlated data and non-stationary distributions, providing a foundation for effective learning in dynamic environments.

3.4 QtripMADQN

The `QtripMADQN` class encapsulates the core functionalities of the Deep Q-Learning algorithm. This agent continuously interacts with its environment, making decisions and learning from the outcomes.

Attributes:

- **Model parameters:** Parameters like `learning_rate` dictate the speed of learning, `discount_factor` governs the agent's consideration for future rewards, and `exploration_rate` controls the trade-off between exploration and exploitation.
- **State representations:** These attributes like `city_state`, `time_state`, and `visited_cities_state` help the agent understand its current position and context within the environment.
- **Neural Networks:** Two primary networks are used. `q_network` for predicting the Q-values of current actions and `target_network` for estimating the future Q-values, aiding in stabilizing the learning.
- **replay_buffer:** An instance of the previously described `ReplayBuffer` class, which stores experiences for more stable learning.

Methods:

- **update_exploration_rate():** Modifies the `exploration_rate` as the agent learns, ensuring a balance between exploration and exploitation over time.
- **calculate_distance():** A utility function to gauge the distance between any two cities, instrumental in path planning and reward computation.
- **choose_action():** Implements the epsilon-greedy strategy to either explore a new action or exploit a known action.
- **calculate_reward():** An essential function that quantifies the benefit of an action. It considers multiple factors to output a scalar reward value.
- **move_to_new_city(), stay_in_current_city(), take_action():** Methods that embody the agent's interactions with the environment, updating the agent's state based on actions and producing rewards.

3.5 Reward Mechanism

The reward mechanism is a linchpin in reinforcement learning, guiding the agent's behavior by providing feedback on the quality of its actions.

- **Distance between cities:** Travelling shorter distances might be preferred to reduce time or costs, translating to higher rewards.
- **User preferences:** If a user has a predilection for certain cities, visiting these places might yield higher rewards.
- **Model's inherent bias:** The model might prioritize cities based on attributes like population, historical significance, or other factors, affecting the reward.
- **Penalties:** Staying in the same city for an extended duration might result in diminishing rewards, pushing the agent to explore new places.

3.6 Decision-making

Decision-making in reinforcement learning often hinges on balancing exploration (trying out new actions) and exploitation (relying on known actions). The epsilon-greedy strategy captures this trade-off neatly.

$$\begin{cases} \text{random action with probability} & \epsilon, \text{ facilitating exploration.} \\ \text{action with highest Q-value with probability} & 1-\epsilon, \text{ enabling exploitation.} \end{cases}$$

The parameter ϵ, termed as the exploration rate, modulates this balance. Initially high to encourage exploration, it generally decays over time, allowing the agent to exploit its acquired knowledge more.

3.7 Optimal Policy Retrieval

Determining the optimal policy is central to any reinforcement learning problem. The policy, in essence, signifies a mapping from states to actions that aims to maximize the cumulative reward over time. In the Q-learning paradigm, this policy is intrinsically tied to the Q-values learned during training.

- **State-action Q-values:** For every state s and associated action a, the Q-value, denoted as $Q(s, a)$, reflects the expected cumulative reward when taking action a in state s and then following the optimal policy thereafter.
- **Optimal Action Extraction:** For each state s, the best action a^* to undertake can be identified using:
$$a^*(s) = \arg\max_a Q(s, a)$$
This essentially suggests picking the action that promises the maximum Q-value for the given state.
- **Multi-agent Scenarios:** In more complex environments where multiple agents coexist, interactions can have a profound impact on the policy. For a particular agent i, considering its actions a_i and the actions of other agents a_{-i}, the optimal joint action is:
$$a_i^*(s), a_{-i}^*(s) = \arg\max_{a_i, a_{-i}} Q(s, a_i, a_{-i})$$
This underscores the intricacies in multi-agent environments where the actions of one agent may affect the rewards and actions of others.

4 Conclusion

The Qtrip Initiative: The presented Qtrip algorithm stands as a testament to the power and versatility of reinforcement learning techniques. Designed as a travel optimization tool, its foundational philosophy lies in tailoring travel recommendations to individual user preferences, hence revolutionizing the traditional approach to itinerary planning.

- **Personalized Experiences:** Unlike generic travel solutions, Qtrip weighs user preferences and dynamically adjusts its suggestions, offering a more personalized travel experience.
- **Incorporation of Deep Learning:** The synergy of Deep Q-Learning with neural networks furnishes the algorithm with the ability to handle complex state spaces, making it adaptable to a myriad of scenarios.
- **Stable Learning Mechanisms:** Leveraging the experience replay mechanism not only promotes stability during training but also ensures that the agent benefits from diverse past experiences, accelerating its learning curve.

In summation, the Qtrip algorithm exhibits immense potential in reshaping the travel industry, providing users with smart, efficient, and personalized travel plans while showcasing the prowess of modern AI methodologies.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

[2] Sutton, R. S., Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[3] Lowe, R., Wu, Y., Tamar, A., et al. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems* (pp. 6379-6390).

[4] Busoniu, L., Babuska, R., De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2), 156-172.

[5] Barro, A. (2023). *Reinforcement Learning*. CS229: Machine Learning. Stanford University. Unpublished lecture notes.