CS510: Deep Reinforcement Learning

Solving Cartpole problem using Q-learning and DQN

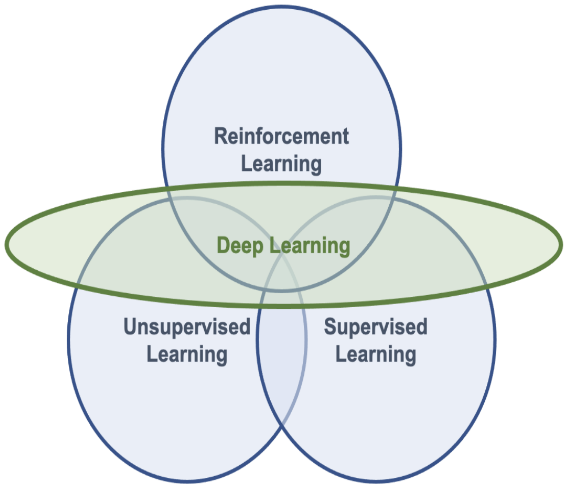
Abstract:

The Cart Pole problem is the part of classic control environments in OpenAPI gym. The goal of the agent is to balance the pole which is attached to the cart that is moving on a frictionless track with maximum reward. The actions that can be made are to push the cart left or right. Over the period of time, many AI researchers, enthusiasts and engineers have applied various algorithms to solve the problem and maximize the reward. Reinforcement learning allows the agent to learn through experience by interacting with the environment by performing actions which result in rewards (positive or negative). Q learning and Deep Q Networks are a part of model free Reinforcement Learning algorithms. In this paper, we will discuss how these two agents perform in the cart-pole environment and if the goal is being achieved or not.

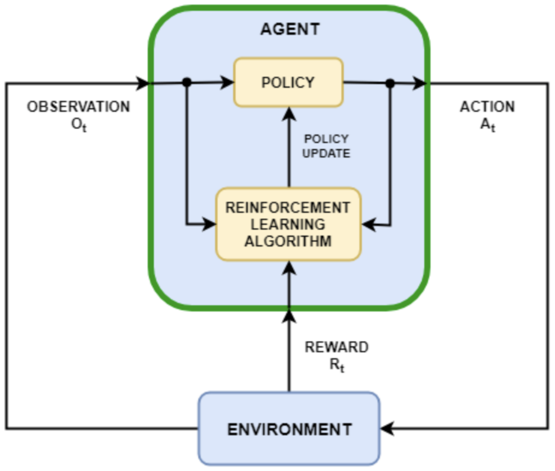
Introduction:

Reinforcement learning (RL) is a machine learning technique that allows agents to learn through trial and error in an interactive environment using feedback from their actions and experiences. Both supervised learning and reinforcement learning use a mapping between inputs and outputs, but unlike supervised learning, where the feedback given to the agent is the set of correct actions to perform the task, reinforcement learning uses rewards and punishments as signals of positive and negative behavior. Reinforcement learning differs in terms of goals compared to unsupervised learning. While the goal of unsupervised learning is to find similarities and differences between data points, the goal of reinforcement learning is to find a suitable action model that maximizes the total cumulative rewards of agents.

Deep learning is one of the best tools we have today to deal with unstructured environments. The agents learn from large amounts of data and discover patterns.Reinforcement learning provides this functionality. Reinforcement learning can solve problems using a variety of ML techniques and techniques, from decision trees to SVMs to neural networks. However, neural networks are not always the best solution to all problems.Nonetheless, neural networks are arguably one of the most powerful techniques available today, and their performance is often the best.

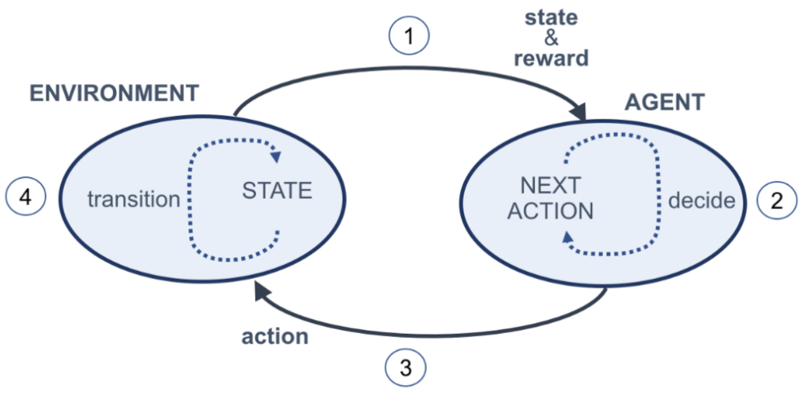


The purpose of reinforcement learning is to train an agent to perform a task in an uncertain environment. At each interval, the agent receives observations and rewards from the environment and dispatches an action to the environment. The reward is a measure of how successful the previous action (performed from the previous state) was in completing the task objective. The agent contains two components: policies and learning algorithms. The policy is a map that selects actions based on observations from the environment. Typically, the policy is a function approximator with tunable parameters, such as deep neural networks. The learning algorithm continuously updates policy parameters based on actions, observations, and rewards. The goal of the learning algorithm is to find the optimal strategy to maximize the expected long-term cumulative rewards received during the task.



Reinforcement Learning cycle:

The cycle begins with the agent observing the environment (step 1) and receiving conditions and rewards. The agent uses this state and reward to determine the next action to take (step 2). The agent then sends an action to the environment to try to control it in a useful way (step 3). Finally, the environment transitions, and as a result, its internal state changes depending on the previous state and the actions of the agent (step 4). Then the cycle repeats.



Episode:
The task the agent is trying to solve may or may not have a natural ending. Task games with the following natural endings are called temporary tasks. Conversely, a task that does not do this is called a continuous task. Learn to move forward. The sequence of time steps from the start to the end of an episode task is called an episode.

Return:
The sum of the rewards collected in one episode is called the return. Agents are often designed to maximize revenue. One of the restrictions is that these rewards are only exposed to agents at the end of the episode.

This was previously introduced as a "delayed reward". For example, in the game TicTactoe, the reward for each move (action) is not known until the end of the game. If the agent wins the game (because the agent has achieved the desired total), it will be a positive reward, and if the agent loses the game, it will be a negative reward (penalty).

Exploration vs Exploitation:
Another important function and challenge in reinforcement learning is the trade-off between exploration and utilization. When trying to earn a lot of rewards, agents need to support the behaviors they have tried in the past and know that these are effective behaviors in earning rewards. But, paradoxically, to discover such an action, he must try an action he has never chosen. In summary, agents need to leverage what they have already experienced to get as much reward as possible, but at the same time explore to take better action in the future.

Cart Pole Environment:

The Cart Pole environment consists of a pole which moves along a frictionless track. The system is controlled by applying a force of +1 or 1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. The state space is represented by four values: cart position, cart velocity, pole angle, and the velocity of the tip of the pole. The action space consists of two actions: moving left or moving right. You will be rewarded with +1 for each time step the pole remains upright. The episode ends when the pole is more than 15 degrees away from the vertical or the cart is more than 2.4 units away from the center.

In this paper, we have tried to solve the cart pole problem using the model free algorithms Q-learning and Deep Q networks.

Previous Work:

Cart pole environment is a part of the classic control environment in the Open API gym. This problem has been solved previously using various RL algorithms. Many AI researchers, enthusiasts and engineers have applied various algorithms to maximize the reward. In this section we will give a better understanding of two of these approaches.

Random Movements:
In this approach, we choose a random action (left or right), taking into account the specific state of the environment. Here the agent does not take an account of the current state, which resulted in poor performance. Hence, this approach due to its random nature is unpredictable.

Policy Based Method:
The hill climbing method is a policy-based method and does not use the policy gradient method similar to gradient descent. In a policy-based way, rather than learning value functions it shows what the total expected reward is when given a state and action, and learns directly. A policy function that maps a state to an action (select an action without using a value function). The environment is solved much more efficiently.

By analyzing these algorithms and how efficiently the cart pole problem is solved, we were convinced to apply model free algorithms to see if the performance of the agent is affected by the model that is being used or not. So, we trained a Q Learning agent and DQN agent in the cart pole environment.

Methods:

Q-Learning:

Q-Learning is the most basic form of reinforcement learning that uses Qtable instead of neural networks to find the best possible action for a particular situation. Our environment consists of four possible states [car position, car speed, polar angle, polar speed] and two possible actions [left, right].

Algorithm:

- Initialize the Q-table
- Select an action in the Epsilon Greedy exploration strategy
- Update Q-table with Bellman equation

Initialize the Q-table:

Q-table is a simple data structure used to track states, actions, and their expected rewards. More specifically, Qtable maps a state-action pair to a Q value (estimated optimal future value) that the agent learns. At the start of the QLearning algorithm, Qtable is initialized to zero. This indicates that the agent knows nothing about the world. As the agent attempts different actions in different states through trial and error, the agent learns the expected rewards for each state-action pair and updates the Q table with the new Q value. Learning the world through trial and error is called exploration.

One of the goals of the QLearning algorithm is to learn the QValue of the new environment. QValue is the maximum expected reward that an agent can achieve by performing a particular action A from state S. After the agent learns the Q value of each state-action pair, in state S, the agent maximizes the expected reward by selecting action A, which has the highest expected reward. Explicitly selecting the most well-known action in a situation is called exploitation.

Choose an action using the Epsilon-Greedy Exploration Strategy:
A common strategy for dealing with exploration-exploration trade-offs is the epsilon greedy exploration strategy.

1. Roll the dice at each time step when selecting an action
2. If the dice probability is less than Epsilon, select a random action
3. Otherwise, perform the most well-known action on the agent's current status.

Update the Q-table using the Bellman Equation:

The Bellman Equation instructs us on how to update our Q-table after each step. To summarize, the agent updates the current perceived value with the estimated optimal future reward, assuming the agent takes the best currently known action. In practice, the agent will search through all actions for a given state and select the state-action pair with the highest corresponding Q-value.

Bellman Equation:

$$Q(S_t, A_t) = (1 - \alpha) Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a Q(S_{t+1}, a))$$

S = the State or Observation, A = the Action the agent takes,R = the Reward from taking an Action,

t = the time step, ⍺ = the Learning Rate

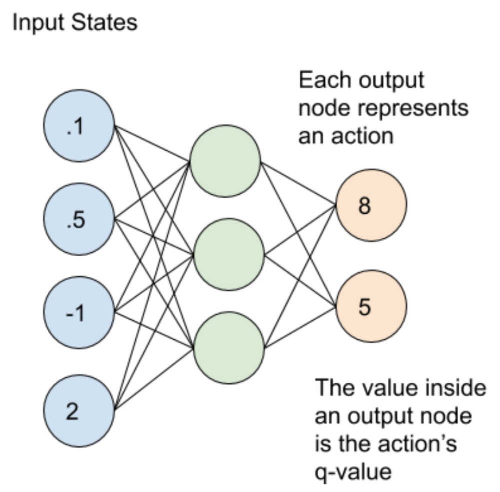
ƛ = the discount factor which causes rewards to lose their value over time so more immediate rewards are valued more highly

<u>Deep Q-Learning:</u>

- Initialize the Main and Target neural networks
- Select an action using the Epsilon-Greedy Exploration Strategy
- Update the network weights using the Bellman Equation

The main difference between Deep Q Learning and Q Learning is the implementation of Qtable. Importantly, Deep Q Learning replaces regular Qtable with neural networks. Instead of mapping state-action pairs to q-values, neural networks map input states to (action, q-value) pairs.

One of the interesting things about Deep Q Learning is that the learning process uses two neural networks. The architecture of these networks is the same, but with different weights. Every N steps, the weights are copied from the main network to the target network. Using these two networks will increase the stability of the learning process and allow the algorithm to learn more effectively. In our implementation, the main network weight replaces the target network weight every 100 steps.


Input States
.1
.5
-1
2
Each output node represents an action
8
5
The value inside an output node is the action's q-value

The main neural network and the target neural network map the input state to pairs (action, q-value). In this case, each output node (representing an action) contains the q-value of the action as a floating point number.

Choose an action using the Epsilon-Greedy Exploration Strategy:

In the Epsilon Greedy Exploration strategy, the agent chooses a random action for probability epsilon and uses the most well-known action for probability 1-epsilon. Both the main model and the target model map the input state to the output action. These output actions actually represent the predicted Q value of the model. In this case, the action with the highest predicted Q value is the most well-known action in that state.

Update your network weights using the Bellman Equation:

After selecting an action, the agent performs the action and updates the main and target networks according to the Bellman equation. The Deep Q Learning agent uses Experience Replay to learn more about the environment and update the main and target networks.

In summary, every four steps, the main network samples and trains a series of previous experiences. The main network weights are then copied to the target network weights every 100 steps.

Experience Replay:

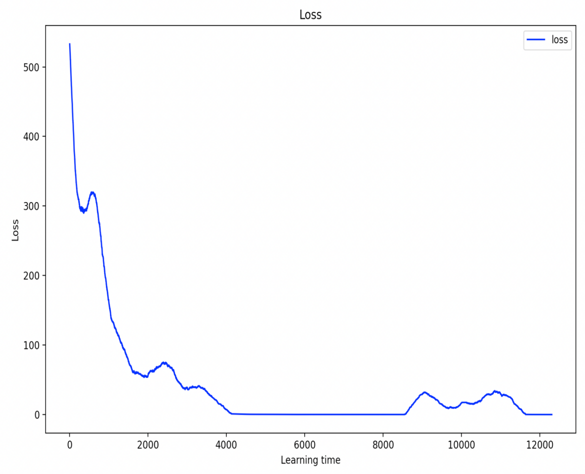
Experience Replay is the act of saving and playing the game state (state, action, reward, next_state) that the RL algorithm can learn. Deep QLearning uses Experience Replay to train in small groups and avoid distorting the record distribution of the various states, actions, rewards, and next_states that the neural network perceives. It is important that the agent does not need to be trained after each step. In our implementation, we use Experience Replay to train all four steps in small groups instead of training each step individually. We found that this trick actually speeds up the implementation of DeepQ Learning.

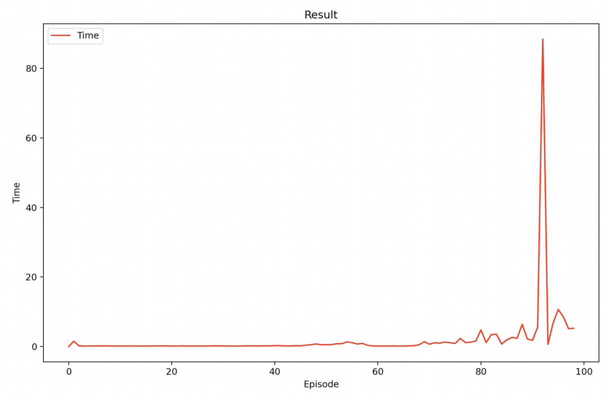
Bellman Equation:

$$(R_t + \lambda * \max_a Q(S_{t+1}, a))$$

Results:

In conclusion, Q-learning is a great form of learning in simple environments with limited moves, where the agent can remember past moves and repeat them.In the more complex problems, the Q-table would not handle a huge number of states and actions, which makes it inefficient to use. The Deep Q-Network is a solution to this problem. The first plot shows the loss over time , the loss decreases to the point of stability as the learning time increases. The second plot shows that the pole is balanced for a longer time as the number of episodes increases.





References

https://www.mathworks.com/help/reinforcement-learning/ug/create-agents-for-reinforcement-learning.htmlhtt
ps://towardsdatascience.com/drl-01-a-gentle-introduction-to-deep-reinforcement-learning-405b79866bf4

https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc

https://medium.com/analytics-vidhya/q-learning-is-the-most-basic-form-of-reinforcement-learning-which-doe
snt-take-advantage-of-any-8944e02570c5