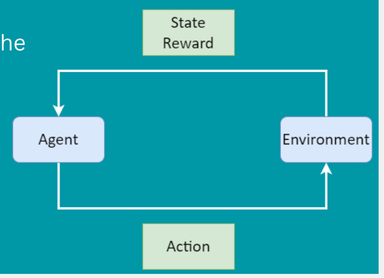
The Mathematics of Reinforcement Learning and its Applications

Aadam Ul Haq - Mathematics Institute - University of Warwick

WARWICK
THE UNIVERSITY OF WARWICK

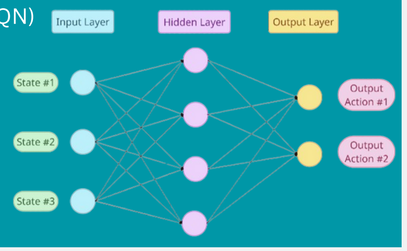
What is Reinforcement Learning?

- Reinforcement Learning (RL) is a type of Machine Learning (ML) where an agent learns to make decisions by interacting with an environment to achieve certain goals.
- RL differs from other types of ML by focusing on learning through trial and error to maximise reward from an action through optimising strategies.
- The Agent performs and action on the environment which changes the current state and the action may cause a reward or penalty to occur.



Links to Neural Networks

- Reinforcement Learning (RL) often utilizes Neural Networks to approximate value functions and model environmental dynamics.
- Deep Reinforcement Learning (DRL) integrates deep neural networks, enabling efficient learning from high-dimensional inputs from the current states.
- NNs are commonly used in RL algorithms such as Q-Learning, Deep Q-Networks (DQN) and Policy Gradient methods.
- The integration of NNs with RL algorithms has led to significant advancements in various domains, including gaming, robotics, and finance.



Famous Motivation - Pavlovian Conditioning

- Pavlovian Conditioning (also known as classical conditioning) is a fundamental concept in psychology describing associative learning.
- Ivan Pavlov's experiment involved pairing a neutral stimulus (bell) with an unconditioned stimulus (food), leading to a conditioned response (salivation) in dogs.
- While distinct from Reinforcement Learning, Pavlovian Conditioning shares similarities in associative learning principles between stimuli and responses.
- While Pavlovian Conditioning primarily focuses on involuntary responses to stimuli, RL extends this concept to encompass goal-directed learning and decision-making in dynamic environments.

Mathematics of Reinforcement Learning

There are numerous mathematical techniques underpinning Reinforcement Learning. Some of these include dynamic programming, policy gradients, function approximation, Markov decision processes, and the Bellman equation. This poster will focus on the latter two.

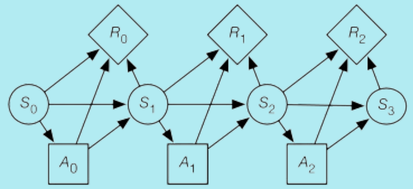
Markov Decision Processes (MDPs)

The Markov Property states 'The Future is Independent of the Past given the Present' or if s is the current state of the agent:

$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \ldots, s_t]$$

We define a policy, π :- a probability distribution over actions, a, for a state, s, by: $\pi(a|s) = \mathbb{P}[a_t = a|s_t = s]$

From which, we can find a reward function given only the present state s and action a: $r(s,a) := \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in \mathcal{S}} p(s' \mid s, a)\mathcal{R}(s, a, s')$



The interpretation in the diagram above[1] shows at time t, given a state s, we select an action a and move to a new state at time t+1, obtaining a reward R

The Bellman Equation (Q-Learning Update Equation)

- Q-learning is a model-free Reinforcement Learning algorithm. It is used to find the optimal action-selection policy for a MDP.
- The Q-learning update equation below iteratively updates the Q-value (state-action value function) based on observed transitions and rewards.
- The Q-value represents the expected cumulative reward obtained by taking action a in state s and following the optimal policy thereafter.

$$\boldsymbol{Q_{new}(s,a) = Q(s,a) + \alpha[R(s,a) + \lambda \max[Q(s',a)] - Q(s,a)]}$$

where:

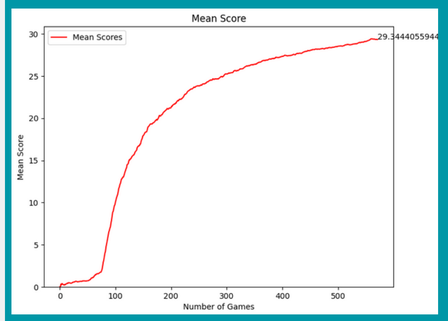
- Q(s,a) are the Q-values at the particular state and action
- R(s,a) is the Reward associated of doing the action in state a
- maxQ(s',a) is the maximum Q-value among all possible actions in the next state
- α= Learning Rate – Controls step size of update
- λ= Discount Rate – Balance immediate and future rewards

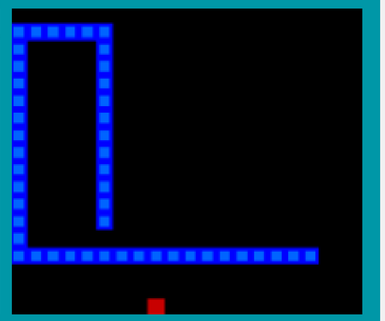
Worked Application: The Snake Game

Perhaps the most prominent use of Reinforcement Learning is developing agents to complete video games. Video games offer diverse environments with no real-life consequences of failure and a clear feedback mechanism.

For the Snake Game in particular, our parameters are easily defined. The agent is the snake, our environment is the game, our actions are the four cardinal directions, and the reward is positive if we collect a fruit, and negative if we achieve a game over. Our state space consists of 11 states: four to determine the direction in which the snake is moving; three to look for immediate dangers, and four to find the relative direction of the fruit.

Using Deep Neural Networks to calculate our Q-values from the Bellman Equation, and utilising a Markov Decision Process, we see the snake improves over time:





There is a clear spike in performance after 80 games due to the Exploitation vs Exploration concept. In order to learn what the best actions are, the snake has a chance of taking random moves for the first 80 games, until it can exploit the game based off what it has learned.

The full code can be viewed by scanning the QR code and was adapted from [2]:

References

[1]: artint.info. (n.d.). 9.5 Decision Processes‣ Chapter 9 Planning with Uncertainty ‣ Artificial Intelligence: Foundations of Computational Agents, 2nd Edition. [online] Available at: https://artint.info/2e/html2e/ArtInt2e.Ch9.S5.html [Accessed 12 Feb. 2024]

[2]: Zhou N. (2023). Teaching an AI to Play the Snake Game Using Reinforcement Learning! [online] Medium. Available at: https://medium.com/@nancy.q.zhou/teaching-an-ai-to-play-the-snake-game-using-reinforcement-learning-6d2a6e8f3b1c.