Reinforcement learning (RL) is a super useful approach used by training agents to make sequential decisions in constantly shifting environments. The cartpole problem from our module in Week Five, involves balancing a pole on a moving cart by application of different forces on the cart. This was interesting a choice of benchmark to evaluate the RL algorithm. Now, Deep Q-Networks (DWN) also solve this problem, alternatives like RL & Actor-Critic (A2C) are particularly effective for the handling of the continuous action spaces.

The REINFORCE algorithm is a policy gradient method that optimizes the policy by maximizing the expected cumulative rewards over multiple episodes. It is different from value-based methods which rely on functions to guide policy updates. RL uses complete episodic trainings to update the parameters directly. This makes it more efficient in environments with continuous action spaces, such as our cartpole problem.

Our algorithm starts by setting up a policy function with random parameters. Every episode, the agent follows this policy to interact with the environment, collecting states, actions, & rewards. Once the episode ends, it calculates the total rewards, adjusting for their timings. These rewards then help determine the policy gradient, which updates the policy parameters to favor actions that aim for higher reward output. The process repeats until an optimized strategy is found.

An interpretation via pseudocode might resemble this:

*Begin Loop*

*Generate the trajectories by simulating actions based upon the current policy.*

*Estimate the expected reward.  
 For each step, accumulate the reward:*

*Reward\_current = Reward\_current + Reward\_previous*

*Calculate the gradient of the policy with respect towards the expected reward.*

*Adjust the policy parameters using the calculated gradient.*

*Terminate the loop once optimized strategy is found.*

One of the stand-out features of reinforce is its simplicity. It optimizes the policy directly without needing a separate value function, making it easier to implement. It does struggle with high variability in the updates, further leading to slower & less stable learning. This stems from the need to complete the entire process before updating the policy, delaying feedback & introducing noise.

The Advantage Actor-Critic (A2C) improves upon the REINFORCE methodology by margining policy-based & value-based techniques. A2C uses two neural networks: an actor, which learns the policy, and a critic, which estimates the value function. The critic’s insights help stabilize the policy updates, enhancing the learning efficiency. One key difference between other reinforcement learning is the Advantage function. This function highlights the difference between the expected cumulative reward of taking a specific action in the current state & then expected cumulative reward of the current state.

When employing A2C, both the actor & critic networks are initialized with random settings. The agent then interacts with the environment, collecting data on various states, actions, rewards, & value estimates. After each episode, the algorithm calculates the returns & it’s advantages, which are used to guide the updates to the actor & the critic.

An interpretation via pseudocode might resemble this:

*Begin the loop.*

*Define the actor function.*

*Define the critic function.*

*Repeat until the policy converges:*

*Generate trajectories*

*Compute the Advantage function*

*Determine policy gradients*

*Calculate critic loss*

*Update the actor parameters*

*Update the critic parameters*

*Terminate the loop*

Then we have Q-learning which is a value-based method that is able to adjusts the action-value function to estimate the expected cumulative reward for each state action paid. This decision relies on previously learned action-value pairs, calculated through the action-value function. The algorithm updates these estimates using the Bellman equation and explores different states and actions to improve its understanding. Q-learning, like other value-based methods, is model-free, meaning it doesn’t simulate the environment’s dynamics or explicitly learn a policy.

By comparison, the policy gradient method learns the optimal policy without estimating the action-value function. By using a policy function that maps each state to a probability distribution over actions, it will then allow the agent to select actions by sampling from the distribution. The policy is optimized by updating its parameters using the gradient of the expected cumulative reward. Policy gradient methods can be either model-free or model-based, depending on whether they incorporate the environment’s dynamics. A key difference between these approaches is that policy gradient methods can handle continuous action spaces, whereas value-based methods are typically limited to discrete actions.

Actor-critic methods combine the best of both worlds. They optimize the policy directly like policy-based methods while using a value function to improve learning efficiency. This hybrid approach reduces the variance of updates, leading to faster & more stable training, which presents it as ideal for complex problems like those with continuous action or high-dimensional states. Both REINFORCE & A2C can be used to solve the cartpole problems. REINFORCE is a simple policy gradient that suffers from high variance in updates. Meanwhile, A2C blends policy & value-based approaches for more stable, efficient learnings. Actor-critic methods like A2C use a critic to reduce variance and improve performance, making them powerful tools for tackling challenging reinforcement learning tasks. They offer a good balance of flexibility, stability, and efficiency.

Yoon, C. (2019, May 23). *Deriving policy gradients and implementing reinforce*. Medium. https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Dixitaniket. (2024, July 16). *Advantage actor-critic (A2c) algorithm explained and implemented in pytorch*. Medium. <https://medium.com/@dixitaniket76/advantage-actor-critic-a2c-algorithm-explained-and-implemented-in-pytorch-dc3354b60b50#:~:text=Understanding%20the%20A2C%20Algorithm&text=Actor%3A%20Learns%20a%20policy%20that,state%20under%20the%20current%20policy>.

Sofeikov, K. (2023, May 4). *Reinforce algorithm - reinforcement learning from scratch in Pytorch*. Medium. https://medium.com/@sofeikov/reinforce-algorithm-reinforcement-learning-from-scratch-in-pytorch-41fcccafa107

OpenAI. (n.d.). *Part 2: kinds of RL algorithms*. https://spinningup.openai.com/en/latest/spinningup/rl\_intro2.html#a-taxonomy-of-rl-algorithms