WiDS Kalman Filtered Trend Trader: Week 3

Aarav Malde and Krishang Krishna

Contents

1 Motivation: Why Reinforcement Learning?

Many real-world problems are not about making a single prediction, but about making a *sequence of decisions*, where each decision influences what options are available next. In such settings, the quality of an action cannot be judged in isolation. Its consequences may unfold over time, sometimes long after the action was taken.

Traditional supervised learning assumes access to correct input–output pairs. This assumption breaks down in decision-making problems where correct actions are not known in advance, feedback is delayed, and actions influence future data. In these settings, learning cannot be separated from acting.

Reinforcement Learning (RL) addresses this class of problems by allowing an agent to learn through interaction with an environment. Instead of being told what to do, the agent receives scalar feedback in the form of rewards and must infer which behaviors lead to desirable long-term outcomes. The objective is not to maximize immediate reward, but to maximize cumulative reward over time.

This formulation naturally applies to domains such as game playing, control systems, robotics, and sequential planning, where short-term gains can conflict with long-term success and decisions must be evaluated in the context of their future consequences.

—

2 The Reinforcement Learning Setting

At the core of reinforcement learning is the **agent–environment interaction loop**. Time is modeled discretely. At each time step t:

- the agent observes a *state* s_t,
- selects an *action* a_t,
- the environment returns a *reward* r_t,
- and transitions to a new state s_{t+1}.

This interaction produces a trajectory of experience:

$$(s_0, a_0, r_0, s_1, a_1, r_1, \dots)$$

The agent's behavior is governed by a **policy**, typically denoted π, which specifies how actions are chosen given states. Policies may be deterministic or stochastic.

Two broad categories of reinforcement learning tasks are commonly considered:

- **Episodic tasks**, which have a terminal state (e.g. games).
- **Continuing tasks**, which do not terminate and must be optimized indefinitely.

The agent does not control the environment directly. It only selects actions. The environment may be stochastic, meaning that the same action taken in the same state can lead to different outcomes on different occasions.

—

3 Return and Long-Term Optimization

The objective in reinforcement learning is to maximize the **return**, defined as the cumulative reward obtained over time. The return from time step t is commonly written as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

where $\gamma \in [0, 1]$ is the **discount factor**.

The discount factor plays two important roles. First, it determines how much future rewards are valued relative to immediate rewards. Second, it ensures that the return remains finite in continuing tasks.

A strategy that greedily maximizes immediate reward can perform poorly when actions have delayed consequences. Reinforcement learning explicitly accounts for this by evaluating actions based on their expected long-term effects, rather than their immediate outcomes.

This shift from short-term optimization to long-term planning is fundamental. All reinforcement learning algorithms, regardless of their specific form, are mechanisms for estimating or optimizing expected return under uncertainty.

4 Markov Decision Processes, Value, and Optimality

Reinforcement learning problems are commonly formalized as **Markov Decision Processes (MDPs)**. An MDP provides a mathematical framework for modeling sequential decision-making under uncertainty.

An MDP is defined by the tuple (S, A, P, R, γ), where:

- S is the set of states,
- A is the set of actions,
- $P(s' \mid s, a)$ defines the transition dynamics,
- $R(s, a)$ is the reward function,
- $\gamma \in [0, 1]$ is the discount factor.

The defining assumption of an MDP is the **Markov property**: the future depends only on the current state and action, not on the full history. This means the state must encode all information from the past that is relevant for predicting future rewards and transitions.

A **policy** $\pi(a \mid s)$ defines the agent's behavior. Given a policy, we define two central value functions. The **state-value function** measures the expected return when starting in a state and following the policy:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right].$$

The **action-value function** measures the expected return when taking a specific action and then following the policy:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right].$$

An **optimal policy** is one that maximizes expected return from every state. The corresponding optimal value functions are denoted $V^*(s)$ and $Q^*(s,a)$ and satisfy the relationship:

$$V^*(s) = \max_a Q^*(s,a).$$

This formulation makes clear that reinforcement learning is fundamentally about evaluating and comparing long-term consequences of actions under uncertainty.

—

5 Exploration and Exploitation

A central challenge in reinforcement learning is the tradeoff between **exploration** and **exploitation**. Exploitation refers to selecting actions that are known to yield high reward based on current information. Exploration refers to trying new or uncertain actions to acquire better information about the environment.

Pure exploitation can lead to suboptimal behavior if the agent prematurely commits to actions that appear good early on but are not truly optimal. Pure exploration, on the other hand, wastes time and reward by ignoring useful knowledge.

Effective reinforcement learning algorithms must balance these two objectives. This tradeoff arises because the agent's data distribution depends on its own actions: the choices it makes determine what it learns next.

Many practical algorithms incorporate explicit exploration mechanisms, such as occasionally selecting non-greedy actions or encouraging stochastic policies. Without sufficient exploration, convergence to optimal behavior cannot be guaranteed.

—

6 Q-Learning and the Limits of Tabular Methods

Q-learning is a foundational **model-free** reinforcement learning algorithm that aims to learn the optimal action-value function $Q^*(s,a)$ directly from experience.

The algorithm updates its estimates using the rule:

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

where α is the learning rate. This update adjusts the value of (s,a) toward a bootstrap estimate based on the observed reward and the best predicted future outcome.

Q-learning has several attractive properties:

- It does not require knowledge of the transition dynamics.
- It converges to the optimal policy under sufficient exploration.
- It is conceptually simple and easy to implement.

However, tabular Q-learning scales poorly. As the number of states or actions grows, the Q-table becomes infeasible to store or learn. This issue is commonly referred to as the **curse of dimensionality**.

In large or continuous state spaces, it becomes necessary to replace tabular representations with function approximation. This limitation motivates the development of more scalable methods, such as Deep Q-Networks, which will be discussed next.

7 Deep Q-Networks

Tabular Q-learning becomes impractical in environments with large or continuous state spaces. In such settings, it is infeasible to store or update a separate value for every state–action pair. **Deep Q-Networks (DQN)** address this limitation by using a neural network to approximate the action-value function:

$$Q(s,a) \approx Q_\theta(s,a),$$

where θ denotes the parameters of the network.

Instead of maintaining a table, the agent learns a function that maps states to action values, allowing generalization across similar states. However, combining reinforcement learning with function approximation introduces significant instability. The targets used for learning depend on the same parameters being updated, creating feedback loops that can cause divergence.

DQN mitigates these issues using two key techniques:

- **Experience replay**: past transitions (s, a, r, s') are stored and sampled randomly during training, breaking temporal correlations in the data.
- **Target networks**: a separate, slowly updated network is used to compute target values, stabilizing learning.

With these modifications, DQN enabled reinforcement learning to scale to high-dimensional inputs such as raw pixel observations. Despite its success, DQN remains limited to discrete action spaces and can be sensitive to hyperparameters and reward design.

—

8 Policy-Based Methods and Proximal Policy Optimization

Value-based methods learn how good actions are and derive behavior by selecting actions with high value. In contrast, **policy-based methods** learn the policy directly. A parameterized policy $\pi_\theta(a \mid s)$ is optimized to maximize expected return.

Policy-gradient methods adjust parameters θ in the direction that increases the probability of actions leading to higher reward. These methods naturally support stochastic policies and continuous action spaces, making them suitable for control and robotics problems.

A challenge with naive policy-gradient methods is instability: large policy updates can drastically degrade performance. **Proximal Policy Optimization (PPO)** addresses this by constraining how much the policy is allowed to change during each update. Instead of fully optimizing the objective, PPO restricts updates to remain close to the current policy, resulting in more stable learning.

PPO has several practical advantages:

- Robust and stable training behavior,
- Compatibility with continuous action spaces,

- Conceptual simplicity compared to earlier constrained optimization methods.

As a result, PPO has become a widely used algorithm for modern deep reinforcement learning applications, particularly in environments where value-based methods are unsuitable.

9 Actor–Critic Methods

Actor–Critic methods combine elements of both value-based and policy-based reinforcement learning. Instead of learning only a value function or only a policy, these methods maintain two components:

- the **actor**, which represents the policy and selects actions,
- the **critic**, which estimates a value function to evaluate the actor's decisions.

The critic provides feedback on how good the actor's chosen actions are, typically by estimating a state-value or action-value function. This feedback is then used to update the policy parameters of the actor. By combining these two roles, Actor–Critic methods can achieve lower variance updates than pure policy-gradient methods while retaining the flexibility of direct policy optimization.

Many modern reinforcement learning algorithms, including PPO, can be viewed as variants of the Actor–Critic framework. More advanced methods build on this idea by introducing additional objectives or regularization, but the core structure remains the same.