PHASE 6
Reinforcement Learning

MDPs, value functions, Q-learning, policy gradients, PPO, and RLHF

Phase goal

Understand learning by interaction and reward: Markov decision processes, the value/policy duality, deep Q-networks, policy gradients, and how PPO powers the RLHF that aligns modern LLMs.

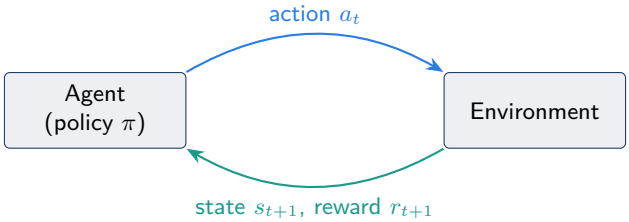
Estimated time: 3–4 weeks (optional) | **Track:** Comprehensive ML & Deep Learning Curriculum

A self-paced, depth-first path from mathematical foundations to production deep learning.

Contents

1 A different learning paradigm

Supervised learning has labeled answers; reinforcement learning (RL) has only *reward*. An **agent** takes **actions** in an **environment**, which returns a new **state** and a scalar **reward**; the agent must learn a behavior (**policy**) that maximizes cumulative reward over time. There is no teacher saying "the right action was X" — only delayed, often sparse, feedback. This phase is optional for most ML careers but conceptually beautiful, and it is essential for understanding RLHF (how chat LLMs are aligned).



Key idea

The defining challenges of RL: (1) the **credit-assignment problem** — which past action caused this reward? (2) the **exploration vs. exploitation** tradeoff — try new actions to learn, or exploit what you know works? (3) **delayed reward** — a move in chess pays off twenty moves later. Every RL algorithm is a different answer to these three questions.

2 Markov Decision Processes

RL problems are formalized as an **MDP**: states $\mathcal{S}$, actions $\mathcal{A}$, transition dynamics $P(s' \mid s, a)$, reward $R(s, a)$, and discount $\gamma \in [0, 1)$. The **Markov property** assumes the next state depends only on the current state and action — the present summarizes the past.

Definition

The agent seeks a policy $\pi(a \mid s)$ maximizing the expected **discounted return**

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

The discount γ makes the infinite sum finite and encodes "prefer sooner rewards": γ near 1 is far-sighted, near 0 is myopic.

2.1 Value functions and the Bellman equation

The **state-value** $V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]$ is the expected return from state s under π. The **action-value** $Q^\pi(s, a)$ is the expected return from taking a in s, then following π. Both satisfy a recursive **Bellman equation**:

$$Q^\pi(s, a) = \mathbb{E}\big[r + \gamma\, Q^\pi(s', a')\big].$$

Intuition

The Bellman equation says "the value of here = immediate reward + discounted value of where I land next." This recursion — the same self-referential structure as dynamic programming — is the foundation of nearly every value-based RL method. The optimal Q^* replaces $Q^\pi(s', a')$ with $\max_{a'} Q^*(s', a')$: act greedily with respect to the best future.

3 Q-learning and Deep Q-Networks

Q-learning learns Q^* directly by bootstrapping toward the Bellman target:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{TD target}} - Q(s,a)\big].$$

It is **off-policy** (learns the optimal policy while behaving exploratorily, e.g. ϵ-greedy). For small discrete problems Q is a table; for large/continuous states we approximate it with a neural network — a **Deep Q-Network (DQN)**.

Key idea

DQN (the 2015 Atari breakthrough) made neural-network Q-learning stable with two tricks: **experience replay** (store transitions and train on random minibatches, breaking correlation between consecutive samples) and a **target network** (a slowly-updated copy used for the TD target, preventing the "chasing a moving target" instability). These two ideas are the practical core of value-based deep RL.

Common pitfall

Naively plugging a neural net into Q-learning diverges — this is the "deadly triad" (function approximation + bootstrapping + off-policy). Replay and target networks are not optional polish; they are what makes it work. Other gotchas: reward scaling matters a lot, and DQN handles only *discrete* action spaces.

4 Policy gradient methods

Instead of learning values and acting greedily, **policy-gradient** methods directly parameterize and optimize the policy $\pi_\theta(a \mid s)$ — naturally handling continuous actions and stochastic policies.

Definition

The **REINFORCE** estimator ascends the expected return:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi\big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\big].$$

Intuitively: increase the probability of actions that led to high return, decrease it for low return — weighted by how good the outcome was.

4.1 Actor–Critic

REINFORCE has high variance (it uses the noisy full return G_t). **Actor–Critic** reduces variance by learning both a policy (*actor*) and a value estimate (*critic*), replacing G_t with an **advantage** $A(s,a) = Q(s,a) - V(s)$ — "how much better than average was this action?" This is the workhorse structure behind modern policy-gradient algorithms.

5 PPO — the workhorse

Proximal Policy Optimization is the most widely used policy-gradient algorithm: reliable, reasonably sample-efficient, and relatively easy to tune. Its key idea is to take the biggest improvement step possible *without* moving the policy too far from the previous one (large updates destabilize RL).

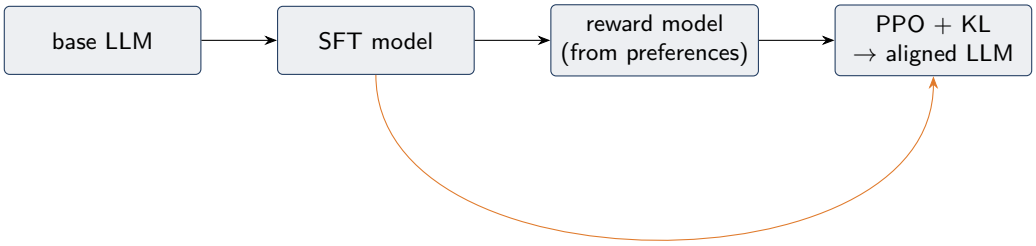
Key idea

PPO optimizes a **clipped surrogate objective**: it maximizes the advantage-weighted probability ratio $r_t(\theta) = \pi_\theta / \pi_{\theta_{\text{old}}}$, but *clips* that ratio to $[1-\epsilon, 1+\epsilon]$ so a single update cannot change the policy drastically. This "trust-region-lite" behavior is why PPO is robust enough to be the default in everything from robotics to RLHF.

6 RLHF — how modern LLMs get aligned

Reinforcement Learning from Human Feedback is how a raw, next-token-predicting LLM becomes a helpful, harmless assistant. The classic three-stage recipe:

1. **Supervised fine-tuning (SFT):** fine-tune the base LLM on high-quality demonstration data.
2. **Reward model:** collect human *preference* comparisons ("response A is better than B") and train a model to predict that preference — a learned reward.
3. **RL optimization:** use **PPO** to optimize the LLM against the reward model, with a KL penalty that keeps it close to the SFT model (so it does not "hack" the reward into gibberish).



Intuition

RLHF is why this curriculum's Phase 6 is worth skimming even if you never train a game-playing agent: the alignment of the assistants you use daily is a PPO loop over a learned reward. Newer methods like **DPO** (Direct Preference Optimization) skip the explicit reward model and RL loop, optimizing preferences directly with a simple classification-style loss — simpler and increasingly popular, but the RLHF framing remains the conceptual anchor.

7 Checkpoint project

Train a DQN on a Gym environment

Goal: get an agent to solve a classic control task and internalize the RL loop.

1. Start with **CartPole** (`gymnasium`); it should be solvable in minutes.
2. Implement DQN with **experience replay** and a **target network**; use an ϵ-greedy policy with decay.
3. Plot the episode-reward curve; confirm it rises to the solved threshold and discuss its variance.
4. **Stretch:** move to **LunarLander**; try a policy-gradient method (PPO via Stable-Baselines3) and compare sample efficiency and stability.

Definition of done: a training script, a reward-vs-episode plot, a short note on how replay and the target network affected stability, and (ideally) a rendered GIF of the trained agent.

```python
# Core DQN update step (PyTorch sketch)
states, actions, rewards, next_states, dones = replay.sample(batch)
q = qnet(states).gather(1, actions)                       # Q(s, a)
with torch.no_grad():
    q_next = target_net(next_states).max(1, keepdim=True).values
    target = rewards + gamma * q_next * (1 - dones)       # TD target
loss = F.smooth_l1_loss(q, target)                        # Huber loss
opt.zero_grad(); loss.backward(); opt.step()
# Periodically: target_net.load_state_dict(qnet.state_dict())
```

Phase 6

- **Sutton & Barto** — *Reinforcement Learning: An Introduction* (free PDF; the canonical text).
- **David Silver** — RL course (DeepMind/UCL, YouTube; the classic lectures).
- **OpenAI Spinning Up in Deep RL** — the best practical bridge to code, with clean PPO/VPG implementations.
- **Stable-Baselines3** — reliable reference implementations for experimentation.
- For RLHF: the InstructGPT paper, and the DPO paper for the modern simplification.

8 Phase 6 self-check

- Define an MDP and explain the discount factor and the Markov property.
- Write the Bellman equation and explain bootstrapping.
- Explain why DQN needs experience replay and a target network.
- Contrast value-based (DQN) and policy-gradient (REINFORCE/PPO) methods.
- Explain PPO's clipped objective in one sentence.
- Describe the three stages of RLHF and what the KL penalty prevents.