Deep Reinforcement Learning: Value-Based, Policy-Based, and AlphaGo

October 22, 2025

Contents

1 DQN, Policy Gradient, and Actor-Critic

Deep reinforcement learning (DRL) optimizes sequential decision-making policies by combining neural function approximators with RL objectives. Figure ?? contrasts key components of value-based, policy-based, and actor-critic algorithms.

1.1 Deep Q-Network (DQN)

DQN approximates the optimal action-value function $Q^\star(s,a)$ for discrete actions. The Bellman optimality equation states

$$Q^\star(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s,a) + \gamma \max_{a'} Q^\star(s',a') \right]. \tag{1}$$

DQN parameterizes $Q_\theta(s,a)$ with a neural network and minimizes the temporal difference (TD) loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(y - Q_\theta(s,a) \right)^2 \right], \quad y = r + \gamma \max_{a'} Q_{\theta^-}(s',a'), \tag{2}$$

where θ^- denotes the target network parameters periodically copied from θ. Experience replay $\mathcal{D}$ breaks correlations by sampling uniformly from stored transitions.

Stabilization Techniques

- **Double DQN:** Replace the max over target network with $\arg\max$ from the online network to reduce overestimation.
- **Dueling architecture:** Decompose $Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a')$ to capture state values independently.
- **Prioritized replay:** Sample proportional to TD error magnitude to focus on informative transitions.

Listing 1: DQN training loop with target network and replay buffer.

Pseudo-code

```
replay = ReplayBuffer(capacity=100_000)
q_net = QNetwork().to(device)
target_net = copy.deepcopy(q_net)
optimizer = torch.optim.Adam(q_net.parameters(), lr=1e-3)

for step in range(total_steps):
    action = epsilon_greedy(q_net, obs, epsilon_schedule(step))
    next_obs, reward, done, info = env.step(action)
    replay.add(obs, action, reward, next_obs, done)
```

```
    obs = next_obs if not done else env.reset()

    if step > warmup and step % train_freq == 0:
        batch = replay.sample(batch_size=64)
        target = batch.reward + gamma * target_net(batch.next_obs).max(dim=1).values *
            (1 - batch.done)
        q_values = q_net(batch.obs).gather(1, batch.action.unsqueeze(1)).squeeze(1)
        loss = F.mse_loss(q_values, target.detach())
        optimizer.zero_grad()
        loss.backward()
        clip_grad_norm_(q_net.parameters(), max_norm=10.0)
        optimizer.step()

    if step % target_update == 0:
        target_net.load_state_dict(q_net.state_dict())
```

1.2 Policy Gradient Methods

Policy gradients maximize expected return $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$ with respect to policy parameters θ. The REINFORCE gradient estimator is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right], \quad G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k. \tag{3}$$

Variance reduction uses baselines $b(s_t)$, yielding

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\,(G_t - b(s_t))\right]. \tag{4}$$

Common choices include learned value functions $V_\phi(s)$ and advantage estimates $A_t = G_t - V_\phi(s_t)$.

Trust Region Policy Optimization (TRPO) and PPO TRPO constrains policy updates by the KL divergence between old and new policies. PPO simplifies this with a clipped surrogate objective:

$$L^{\mathrm{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \tag{5}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}. \tag{6}$$

Generalized advantage estimation (GAE) smooths advantages via λ-returns:

$$A_t^{\mathrm{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{7}$$

1.3 Actor-Critic Frameworks

Actor-critic methods combine policy (actor) and value (critic) networks. The critic approximates $V_\phi(s)$ or $Q_\phi(s,a)$ to reduce variance. A2C/A3C perform synchronous/asynchronous updates across multiple workers, while soft actor-critic (SAC) introduces entropy regularization for continuous control:

$$J_\pi = \mathbb{E}_{s_t\sim\mathcal{D}}\left[\mathbb{E}_{a_t\sim\pi}\left[\alpha \log \pi(a_t \mid s_t) - Q_\phi(s_t, a_t)\right]\right], \tag{8}$$

where α controls the entropy-temperature trade-off. Deterministic policy gradients (DDPG, TD3) adapt actor-critic to continuous actions via deterministic policies and target smoothing.

1.4 Comparison Summary

Figure **??** summarizes characteristics:

- DQN: discrete actions, off-policy, relies on replay buffer and target network.
- Policy gradient: on-policy, handles continuous actions, suffers from high variance without baseline.
- Actor-critic: hybrid approach enabling stable updates and continuous control.

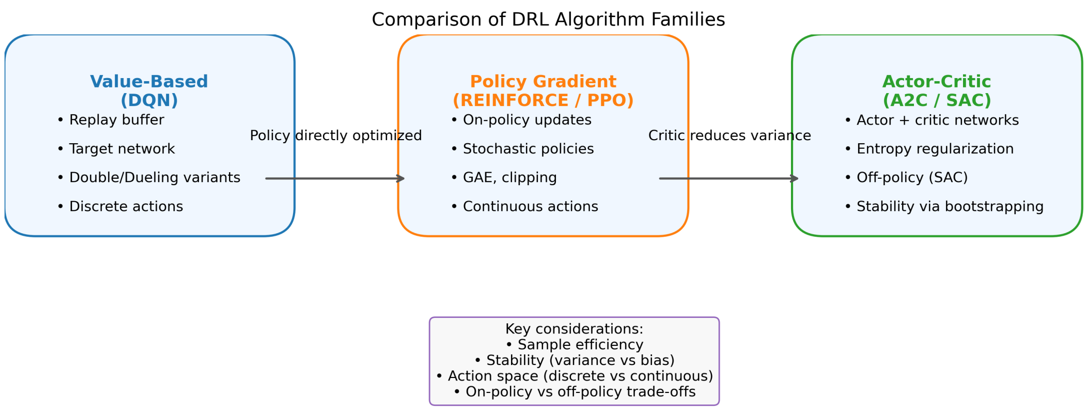

Figure 1: Comparison of DQN, policy gradient, and actor-critic pipelines. Value-based methods rely on replay buffers; policy gradients update policies directly; actor-critic blends both.

2 AlphaGo Case Study

AlphaGo integrates deep learning with Monte Carlo tree search (MCTS), achieving superhuman Go performance. Figure **??** shows the training pipeline combining supervised learning (SL), reinforcement learning (RL), and tree search.

2.1 Policy Network Training

The supervised policy network $p_\theta(a \mid s)$ is trained on expert games via cross-entropy:

$$\mathcal{L}_{\mathrm{SL}}(\theta) = -\mathbb{E}_{(s,a)\sim\mathcal{D}_{\text{human}}}[\log p_\theta(a \mid s)]. \tag{9}$$

This network initializes a reinforcement learning policy trained by self-play to maximize win rate $\rho(\theta)$ using policy gradient:

$$\nabla_\theta \rho(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[(z-b)\sum_t \nabla_\theta \log p_\theta(a_t \mid s_t)\right], \tag{10}$$

where $z \in \{-1, +1\}$ is game outcome and b is a baseline to reduce variance.

2.2 Value Network and Rollouts

The value network $v_\phi(s)$ predicts win probability for state s. It is trained on self-play positions with Monte Carlo outcomes z:

$$\mathcal{L}_{\text{value}}(\phi) = \mathbb{E}_{s\sim\mathcal{D}_{\text{self-play}}}\left[(v_\phi(s) - z)^2\right]. \tag{11}$$

During search, fast rollout policies approximate value by simulating random playouts. The blend of value network and rollouts improves evaluation accuracy.

2.3 Monte Carlo Tree Search Integration

AlphaGo uses a variant of upper confidence bounds (UCB) for action selection within MCTS:

$$a_t = \arg\max_a \left(Q(s_t, a) + c_{\text{puct}}\, P(s_t, a) \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)} \right), \tag{12}$$

where Q is mean action value, P prior from the policy network, and N visit counts. The policy and value networks guide tree expansion and evaluation, drastically reducing search space compared to uniform exploration.

2.4 AlphaGo Zero and AlphaZero

AlphaGo Zero replaces supervised learning with pure self-play and combines policy/value network into a single residual network outputting (p, v). The training target for policy is visit counts π produced by MCTS, while value targets remain game outcomes:

$$\mathcal{L}(\theta) = (z - v_\theta(s))^2 - \pi^\top \log p_\theta(s) + \lambda \|\theta\|^2. \tag{13}$$

AlphaZero generalizes this framework to chess and shogi, demonstrating cross-domain transferability of tree-guided self-play.

2.5 Engineering Insights

- **Hardware:** Original AlphaGo used GPU clusters for neural evaluation plus distributed CPUs for MCTS simulations.
- **Training Efficiency:** Replay buffers for self-play games ensure diverse training data; batching tree search evaluations maximizes GPU utilization.
- **Evaluation:** Regimens include Elo rating against previous generations, ablation of components, and match play versus human professionals.

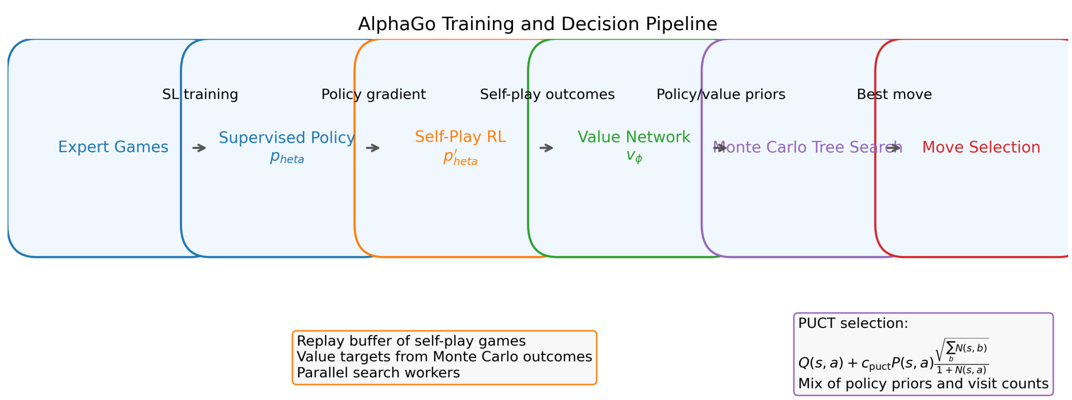

Figure 2: AlphaGo training and inference pipeline: supervised policy initialization, self-play reinforcement learning, value network training, and MCTS-guided decision making.

Further Reading

- Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning." NIPS Deep Learning Workshop 2013.

- John Schulman et al. “Proximal Policy Optimization Algorithms.” arXiv 2017.
- Tuomas Haarnoja et al. “Soft Actor-Critic Algorithms and Applications.” arXiv 2018.
- Silver et al. “Mastering the game of Go with deep neural networks and tree search.” Nature 2016.
- Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.” arXiv 2017.