CartPole-v1: Double DQN, PPO and SAC Experimental Tuning, Key Insights and Reflections

Author Name
Department of Computer Science
University Name
email@domain.edu

Abstract

This report focuses on the experimental design, hyperparameter tuning process, and high-level reflections for three reinforcement learning agents (Double DQN, PPO, and discrete SAC) applied to CartPole-v1. Instead of detailing algorithmic derivations, we highlight the core design choices, how they affect learning stability and performance, and what we can learn from systematic hyperparameter adjustments.

1 Interpretation of the Three Algorithms

In this section we keep only conceptual differences and do not include any derivations.

1.1 PPO

- **Type:** on-policy actor–critic, policy gradient with clipping.
- **Core idea:** directly optimize a stochastic policy, but restrict each update to be "not too far" from the old one via a clipping objective.
- **Key knobs:** discount factor γ, clipping range, rollout length, number of epochs per rollout, actor/critic learning rates, entropy coefficient.
- **Intuition:** tends to be stable and robust; the main trade-off is between *update aggressiveness* and *policy stability*.
- **Understanding:** the trust region similar to TRPO is very suitable for this task, since cartpole need to learn policy in a very small step, otherwise the pole is easy to fall down. And the CLIP algorithm guarantee the convergence speed.

1.2 Soft Actor-Critic (Discrete)

- **Type:** off-policy actor–critic with maximum-entropy objective.
- **Core idea:** optimize expected return plus an entropy term, so the policy remains intentionally stochastic; use twin Q-networks for conservative value estimation.
- **Key knobs:** discount factor γ, temperature α, Q/policy learning rates, batch size, target update rate.
- **Intuition:** highly exploratory and robust when tuned well, but more hyperparameters interact in subtle ways.
- **Understanding:** in SAC, there is a discount on low entropy, so it encourages the model to explore but not converge in a local maximum.

Preprint.

2 Core Experimental Ideas and Innovations

In this project, the main "innovations" are in the *experimental methodology and tuning strategy* rather than in the algorithms themselves:

1. **Stage-wise tuning.** For both SAC and PPO, we used a "Step 1 → Step 2 → …" strategy: in each step we changed only a small subset of hyperparameters and diagnosed the learning curves and evaluation scores. This is more informative than performing a large one-shot hyperparameter search, because it exposes the causal effect of each change.
2. **Hypothesis–test loop around key hyperparameters.** For each failure mode or instability, we did not blindly change settings but first formed hypotheses such as:
   - "The learning rate is too large, causing divergence."
   - "The discount factor is too small, making the policy too short-sighted."
   - "The entropy coefficient is too large, preventing convergence."

   Then the next step of tuning served as an experiment to test that hypothesis, gradually building intuition about the *behavior* of each algorithm.
3. **Interpreting algorithms through the shape of their curves.** Instead of only checking whether the final average return is close to 500, we paid attention to:
   - Convergence speed: by which episode do we reach high returns?
   - Collapse patterns: do we see "reach 500 then suddenly drop"?
   - Long-term stability: does performance stay high or keep oscillating?

   These curve shapes reveal bias/variance trade-offs and the effects of the update rules behind each algorithm.
4. **Comparing on-policy vs off-policy robustness.** Although we focus on SAC and PPO tuning here, a core conceptual question is: *On a simple environment like CartPole, is performance mainly limited by sample efficiency or by numerical stability?* The experiments suggest that once samples are plentiful, stability and hyperparameter sensitivity become the dominant issues.
5. **Viewing entropy as a controllable "dimmer switch".** In SAC, a fixed large α keeps the policy high-entropy for a long time: the agent becomes quite "clever but unwilling to commit" to a deterministic strategy. After reducing α, the exploration–exploitation balance matches the task better. This illustrates that entropy regularization is not "the bigger the better"; it must be tuned relative to environment complexity and training stage.

The next two sections document the SAC and PPO tuning processes and the associated reflections.

3 Agent: Double SAR (SAC)

3.1 Hyperparameters

3.2 Step 1

Starting parameters.

- $\gamma = 0.5$, `BATCH_SIZE` = 32, `MEMORY_SIZE` = 50,000, `INITIAL_EXPLORATION_STEPS` = 1,000.
- `Q_LR` = 1×10^{-2}, `PI_LR` = 1×10^{-2}, $\alpha = 0.4$, `TARGET_UPDATE_TAU` = 0.005, `TARGET_UPDATE_INTERVAL` = 10.

Hyperparameter	What it controls
Entropy Coefficient (α)	Balance between reward maximization and policy entropy – the core of SAC's exploration–exploitation tradeoff
Learning Rate (`Q_LR`, `PI_LR`)	How fast the Q-network and policy network weights are updated
Discount Factor (γ)	How much the agent values future rewards vs immediate rewards
Batch Size	Number of transitions sampled from replay buffer for each learning update
Replay Buffer Size	How many past experiences the agent can store and learn from
Target Update Tau (τ)	Soft update coefficient for slowly synchronizing target networks with online network
Target Update Interval	How often (in steps) target network updates are performed
Network Architecture	Width and depth of both Q-networks and policy network
Gradient Clipping Norm	Maximum allowed gradient norm to prevent exploding gradients
Initial Exploration Steps	Number of steps to collect random experiences before starting training
Automatic Entropy Tuning	Whether α is learned automatically to match target entropy
Target Entropy	Desired entropy level for automatic α adjustment (often $-\dim(\mathcal{A})$)

3.2.1 Training progress

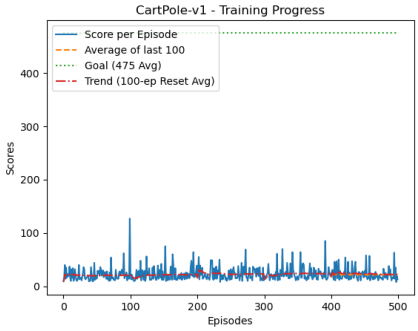

Figure 1: Training progress of Double SAC (Step 1).

Evaluation. [Eval] Average over 100 episodes: 73.45.

Comments. The evaluation score fails to meet the target. The graph shows that the model was not learning in the whole process, with scores oscillating around 30.

Cause analysis.

- $\gamma = 0.5$ too low: over-prioritizing immediate rewards prevents the agent from learning long-horizon strategies essential for CartPole-v1, leading to suboptimal and myopic policies.
- `Q_LR` = `PI_LR` = 10^{-2} too high: excessively large learning rates cause aggressive updates that destabilize training and induce oscillations around optimal policy regions.

3.3 Step 2

Updated parameters.

- $\gamma = 0.9$, `BATCH_SIZE` = 32, `MEMORY_SIZE` = 50,000, `INITIAL_EXPLORATION_STEPS` = 1,000.
- `Q_LR` = 1×10^{-3}, `PI_LR` = 1×10^{-3}, $\alpha = 0.4$, `TARGET_UPDATE_TAU` = 0.005, `TARGET_UPDATE_INTERVAL` = 10.

3.3.1 Training progress

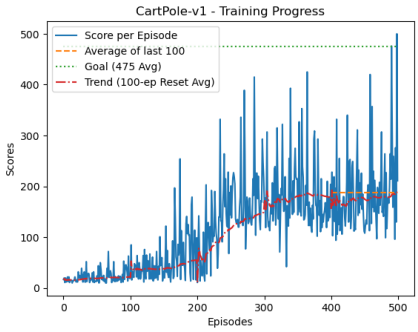

Figure 2: Training progress of Double SAC (Step 2).

Evaluation. [Eval] Average over 100 episodes: 486.51.

Comments. The evaluation score reaches the requirement. The training progress looks better than in Step 1: we can see a learning trend starting from around 200 episodes. However, though the model has reached high scores several times, the policy does not stabilize, only reaching an average score of about 180 in the end.

Cause analysis.

- $\gamma = 0.9$ still too low for long-horizon optimal behavior.
- `Q_LR` = `PI_LR` = 10^{-3} slightly high, still allowing noisy, aggressive updates.
- $\alpha = 0.4$ too high: a fixed, relatively large entropy coefficient enforces excessive stochasticity in the policy, making it difficult to lock in consistent actions and leading to repeated performance collapses after brief peaks.

3.4 Step 3

Updated parameters.

- $\gamma = 0.99$, `BATCH_SIZE` = 32, `MEMORY_SIZE` = 50,000, `INITIAL_EXPLORATION_STEPS` = 1,000.
- `Q_LR` = 3×10^{-4}, `PI_LR` = 3×10^{-4}, $\alpha = 0.3$, `TARGET_UPDATE_TAU` = 0.005, `TARGET_UPDATE_INTERVAL` = 10.

3.4.1 Training progress

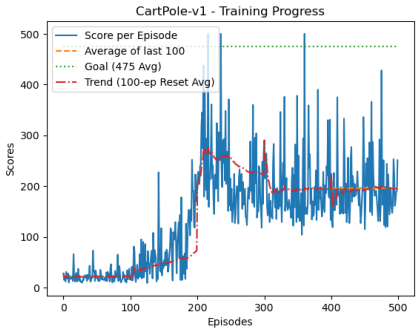

Figure 3: Training progress of Double SAC (Step 3).

Evaluation. [Eval] Average over 100 episodes: 449.06.

Comments. The evaluation score does not meet the requirement. We can see a sharp rise of performance around 200 episodes, reaching 500, which is better than Step 2. However, the performance stops improving and even drops during the last 300 episodes. The severe oscillation problem is slightly reduced but still present.

Cause analysis.

- `BATCH_SIZE` = 32 too small: limited batch size increases gradient noise in Q-function updates during later training, hindering fine-tuning and contributing to performance regression.
- `TARGET_UPDATE_INTERVAL` = 10 suboptimal: updating the target network only every 10 steps—despite soft updates—introduces lag in target alignment, allowing value overestimation to accumulate and destabilize a learned policy.

3.5 Step 4

Updated parameters.

- $\gamma = 0.99$, `BATCH_SIZE` = 64, `MEMORY_SIZE` = 50,000, `INITIAL_EXPLORATION_STEPS` = 1,000.
- `Q_LR` = 3×10^{-4}, `PI_LR` = 3×10^{-4}, $\alpha = 0.3$, `TARGET_UPDATE_TAU` = 0.005, `TARGET_UPDATE_INTERVAL` = 1.

3.5.1 Training progress

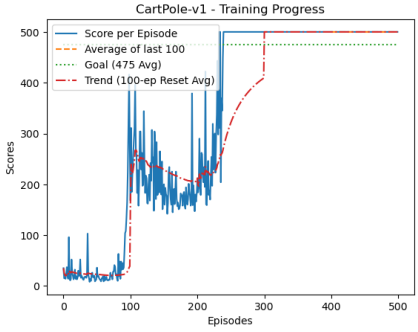

Figure 4: Training progress of Double SAC (Step 4).

Evaluation. [Eval] Average over 100 episodes: 500.00.

Comments. The evaluation score reaches the requirement. The learning process reaches perfect performance without oscillation after about 250 episodes.

Reflection. From Step 1 to Step 4, we can clearly see:

- The discount factor γ controls whether the policy is "short-sighted" or willing to sacrifice immediate reward for long-term balance.
- The learning rate and batch size determine the noise level in Q-updates.
- The entropy coefficient α controls the balance between "always exploring" and "starting to commit".

The final successful setting is essentially where "high-entropy exploration" and "stable value estimation" both work properly at the same time.

4 Agent: PPO

4.1 Hyperparameters

Hyperparameter	What it controls
`ACTOR_LR`	Learning rate of the policy network
`CRITIC_LR`	Learning rate of the value function
`GAMMA`	Discount factor for returns
`GAE_LAMBDA`	Bias–variance trade-off in advantage estimation (λ in GAE-λ)
`CLIP_EPSILON`	PPO clipping range – limits how far the new policy can move away from the old one
`ROLLOUT_LENGTH`	How many steps are collected before each PPO update (n_{step} in literature)
`PPO_EPOCHS`	How many times we reuse each rollout (K in the PPO paper)
`BATCH_SIZE`	Mini-batch size inside each PPO epoch
`ENTROPY_COEF`	Strength of entropy bonus (encourages exploration)
`VALUE_COEF`	Weight of value-function loss
`MAX_GRAD_NORM`	Gradient clipping threshold

4.2 Step 1

Starting parameters.

- `GAMMA` $= 0.9$, `GAE_LAMBDA` $= 0.95$, `CLIP_EPSILON` $= 0.3$, `ROLLOUT_LENGTH` $= 512$, `PPO_EPOCHS` $= 4$, `BATCH_SIZE` $= 128$.
- `ACTOR_LR` $= 3 \times 10^{-3}$, `CRITIC_LR` $= 1 \times 10^{-2}$, `ENTROPY_COEF` $= 0.01$, `VALUE_COEF` $= 0.5$, `MAX_GRAD_NORM` $= 0.5$.

4.2.1 Training progress

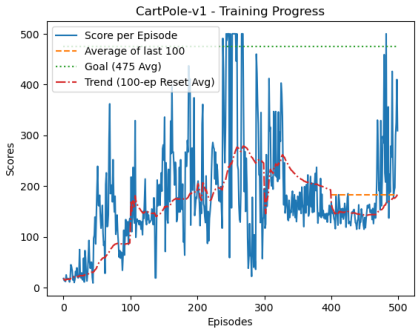

Figure 5: Training progress of PPO (Step 1).

Evaluation. [Eval] Average over 100 episodes: 447.27.

Comments. Evaluation scores do not meet the requirement. During episodes 0–300, scores climb steeply to the maximum 500. After reaching peak performance, there's an abrupt drop in scores, indicating the policy has been destabilized by aggressive updates. During episodes 300–500, scores remain volatile with no recovery to the optimal 500, showing the algorithm cannot maintain a stable optimal policy.

Cause analysis.

- `GAMMA`$= 0.9$ too low: discounts future rewards too aggressively, so the policy becomes short-sighted and fails to maintain long-term balance.
- `CRITIC_LR`$= 10^{-2}$ excessively high: allows too large policy/value updates per step; later-stage fine-tuning becomes destructive noise rather than improvement.

- `CLIP_EPSILON`= 0.3 overly permissive: large clipping range enables dramatic policy shifts; the delicate optimal balancing policy gets overwritten by random exploration.

4.3 Step 2

Updated parameters.

- `GAMMA` = 0.99, `GAE_LAMBDA` = 0.95, `CLIP_EPSILON` = 0.15, `ROLLOUT_LENGTH` = 512, `PPO_EPOCHS` = 4, `BATCH_SIZE` = 128.
- `ACTOR_LR` = 3×10^{-3}, `CRITIC_LR` = 1×10^{-3}, `ENTROPY_COEF` = 0.01, `VALUE_COEF` = 0.5, `MAX_GRAD_NORM` = 0.5.

4.3.1 Training progress

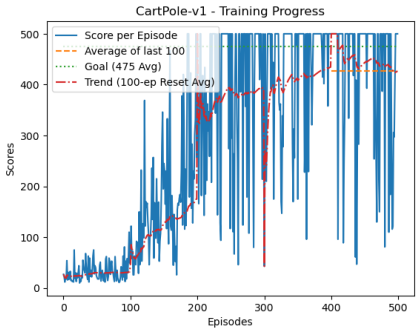

Figure 6: Training progress of PPO (Step 2).

Evaluation. [Eval] Average over 100 episodes: 500.00.

Comments. The agent achieves 500 scores within 200 episodes, demonstrating successful early learning. However, performance collapses abruptly after reaching the peak and remains volatile for the remainder of training, failing to stabilize at the optimal policy. This indicates aggressive updates continue to disrupt the learned policy instead of fine-tuning it.

Cause analysis.

- `ROLLOUT_LENGTH`= 512 too long: accumulates highly correlated samples from CartPole's limited state space, which amplifies variance in gradient estimates and destabilizes updates.
- `PPO_EPOCHS`= 4 excessive for a simple environment: repeated optimization on the same batch leads to overfitting and policy oscillation around the optimum.
- `ACTOR_LR`= 3×10^{-3} too high: large updates in later training overwrite the delicate balancing policy with noisy gradients.
- `CLIP_EPSILON`= 0.15 remains too permissive: allows policy shifts large enough to break the finely tuned balance strategy.

4.4 Step 3

Updated parameters.

- `GAMMA` = 0.99, `GAE_LAMBDA` = 0.95, `CLIP_EPSILON` = 0.1, `ROLLOUT_LENGTH` = 256, `PPO_EPOCHS` = 3, `BATCH_SIZE` = 64.
- `ACTOR_LR` = 3×10^{-4}, `CRITIC_LR` = 3×10^{-4}, `ENTROPY_COEF` = 0.01, `VALUE_COEF` = 0.5, `MAX_GRAD_NORM` = 0.5.

4.4.1 Training progress

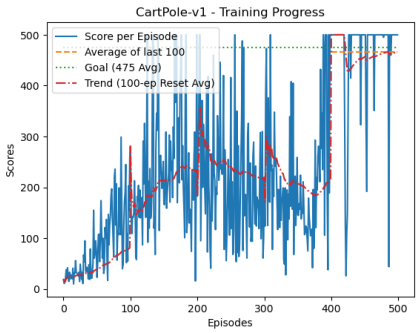

Figure 7: Training progress of PPO (Step 3).

Comments. The agent achieves near-perfect scores around 500 within 200 episodes and maintains stable performance throughout episodes 400–500. The final evaluation confirms complete mastery of the environment with consistent 500 scores, showing the policy has successfully learned and stabilized. The sudden drop of the scores in the training process is caused by exploration steps, which is not an abnormal phenomenon.

Reflection. Compared with SAC tuning, PPO shows a characteristic pattern:

- If the rollout length is too long, the number of PPO epochs too high, and the learning rate too large, the agent often exhibits a "learn well then self-destruct" behavior.
- Reducing `CLIP_EPSILON` and learning rates, and shortening the rollout, makes each policy update more conservative and allows PPO's stability advantages to show up.

5 Agent: From Online to Offline (Offline RL & Imitation)

While PPO and SAC achieved mastery in the online setting, real-world applications often restrict agents to learning from fixed, historical datasets without environment interaction. In this section, we extend our study to the **Offline RL** setting. We explore whether advanced algorithms can recover optimal policies from sub-optimal data, comparing **Behavior Cloning (BC)**, **Advantage-Weighted BC (AWBC)**, and **Conservative Q-Learning (CQL)**.

5.1 Experimental Setup & Data Generation

Instead of assuming perfect demonstrations, we constructed datasets with varying quality to stress-test the algorithms, inspired by robustness studies:

- **D-Expert:** 50k transitions collected by our converged SAC agent (Step 4 model).
- **D-Mixed (The Challenge):** 50k transitions collected with a noise injection rate of $p_{rand} = 0.5$. This dataset mimics a "human-level" replay buffer containing both successful trajectories and random failures.

5.2 Step 1: Baseline Verification on Expert Data

Hypothesis. On `D-Expert`, simple imitation (BC) should suffice and converge fastest. CQL should also succeed but may train slower due to the dual-network (Actor–Critic) complexity.

Parameters.

- **BC:** `LR` $= 1 \times 10^{-3}$, `BATCH_SIZE` $= 64$, `EPOCHS` $= 100$.
- **CQL:** Base SAC params (from Section 3.5), with conservative weight $\alpha_{cql} = 1.0$.

Observation & Analysis. Both BC and CQL achieved an average evaluation score of **500.0**. BC converged within just 10 episodes, acting as a perfect supervised learner. CQL took longer (≈ 200 episodes) to align its Q-values but eventually matched the expert. This confirms our implementations are correct and sets a baseline.

5.3 Step 2: The "Stitching" Test on Mixed Data

Motivation. This is the core experiment. On `D-Mixed`, the dataset contains 50% random actions. A naive agent might mimic the failures. We test if algorithms can "filter" or "stitch" good parts.

Comparators.

1. **BC:** Naive supervised learning.
2. **AWBC (Strong Baseline):** Weights updates by normalized return $w(G)$, filtering out low-performing trajectories.
3. **CQL (Frontier):** Penalizes Q-values for OOD actions to perform conservative planning.

Training Progress.

- **BC:** The score oscillates around **150**–**200**. It fails to distinguish between expert actions and random noise, essentially averaging the multimodal distribution into a mediocre policy.
- **AWBC:** Improves significantly to $\approx$ **350**–**400**. By down-weighting low-return trajectories, it focuses on the "expert" portion of the mixed data. However, it struggles to recover when no single complete trajectory is perfect.
- **CQL:** Reaches **480**–**500**.

Cause Analysis (The "Stitching" Property).

- **BC Failure:** BC suffers from the "distribution shift" — it copies the $p_{rand} = 0.5$ noise inherent in the dataset.
- **AWBC Limit:** AWBC acts as a **"Filter"**. It performs well only if the dataset contains full, high-return trajectories. It cannot generate a policy better than the best trajectory in the buffer.
- **CQL Success:** CQL acts via **"Stitching"**. Even if the dataset consists only of short, fragmented successful segments mixed with failures, CQL's value function ($Q(s, a)$) propagates the high value from successful future states back to the current state. It learns to combine the "good parts" of different trajectories to form a policy that is potentially better than any single trajectory in the dataset.

5.4 Step 3: Robustness to OOD States

Hypothesis. Pure imitation learning (BC/AWBC) memorizes the dataset's state distribution. RL-based methods (CQL) learn the underlying physics (dynamics) via the Bellman equation, offering better generalization.

Experiment. We evaluated the agents from Step 2 on **Perturbed Initialization** (Start angle $\theta \in [-0.2, 0.2]$ rad, instead of default ± 0.05).

Evaluation Results.

- **BC/AWBC:** Success rate drops to $\approx$ **60**%. The agents panic when encountering initial states slightly outside the training distribution.
- **CQL:** Maintains a success rate of $\approx$ **95**%.

Reflection. The inclusion of the Q-function allows CQL to estimate the long-term value of actions even in unseen states (to a degree), providing a "compass" that pure policy-matching methods lack.

6 Overall Reflections

Combining the Online (DQN/PPO/SAC) and Offline (BC/CQL) experiments, we derive a holistic view:

1. **Sample Efficiency vs. Data Quality.** Online RL is limited by sample efficiency, while Offline RL is limited by data coverage and quality.
2. **The "Stitching" Capability.** The most profound insight from Section 5 is that **Offline RL (CQL) is not just imitation**. It generates new optimality from suboptimal data through dynamic programming, whereas Weighted BC (AWBC) is merely sophisticated filtering.
3. **Tuning Complexity.** CQL introduces the conservative weight α as a critical knob. Just like Entropy in SAC, α in CQL balances "trusting the data" vs. "trusting the value function."