TD7

The Great Mind of Scott Fujimoto

Donghu Kim

Intro: Twin Delayed DDPG (TD3)

TD3 = DDPG + Conservative Value Fitting

Main idea: Actor-Critic also has an overestimation bias problem!

Solution: Clipped Double Q-Learning

- Similar to Double Q Learning, but uses minimum as the target value.
- Rather underestimate values than overestimate; at least it doesn't propagate to other values.

Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

Intro: Twin Delayed DDPG (TD3)

1. Clipped Double Q-Learning ($\min_{i=1,2} Q$)
   - Only for critic; actor gradient uses Q_{θ_1} only.
2. Delayed policy update (if t mod d then)
   - Give critic more training time to get more accurate values.
   - $d = 2$ is enough to improve performance.
3. Target policy smoothing regularization ($\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$)
   - Deterministic policies can overfit values to narrow actions.
   - Smooth out action space, by enforcing similar actions to have similar values.

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$
    $\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$
  end if
end for

Overview

TD7 = TD3 + State-Action Embedding (SALE)

+ Smarter Replay Buffer (LAP)

+ (for online) Policy Checkpoints

+ (for offline) BC Loss

+ Some other stuff

Main Results

Online - Mujoco

Table 1: Average performance on the MuJoCo benchmark at 300k, 1M, and 5M time steps, over 10 trials, where $\pm$ captures a 95% confidence interval. The highest performance is highlighted. Any performance which is not statistically significantly worse than the highest performance (according to a Welch's t-test with significance level 0.05) is highlighted.

Environment	Time step	TD3	SAC	TQC	TD3+OFE	TD7
HalfCheetah	300k	7715 $\pm$ 633	8052 $\pm$ 515	7006 $\pm$ 891	11294 $\pm$ 247	15031 $\pm$ 401
	1M	10574 $\pm$ 897	10484 $\pm$ 659	12349 $\pm$ 878	13758 $\pm$ 544	17434 $\pm$ 155
	5M	14337 $\pm$ 1491	15526 $\pm$ 697	17459 $\pm$ 258	16596 $\pm$ 164	18165 $\pm$ 255
Hopper	300k	1289 $\pm$ 768	2370 $\pm$ 626	3251 $\pm$ 461	1581 $\pm$ 682	2948 $\pm$ 464
	1M	3226 $\pm$ 315	2785 $\pm$ 634	3526 $\pm$ 244	3121 $\pm$ 506	3512 $\pm$ 315
	5M	3682 $\pm$ 83	3167 $\pm$ 485	3462 $\pm$ 818	3423 $\pm$ 584	4075 $\pm$ 225
Walker2d	300k	1101 $\pm$ 386	1989 $\pm$ 500	2812 $\pm$ 838	4018 $\pm$ 570	5379 $\pm$ 328
	1M	3946 $\pm$ 292	4314 $\pm$ 256	5321 $\pm$ 322	5195 $\pm$ 512	6097 $\pm$ 570
	5M	5078 $\pm$ 343	5681 $\pm$ 329	6137 $\pm$ 1194	6379 $\pm$ 332	7397 $\pm$ 454
Ant	300k	1704 $\pm$ 655	1478 $\pm$ 354	1830 $\pm$ 572	6348 $\pm$ 441	6171 $\pm$ 831
	1M	3942 $\pm$ 1030	3681 $\pm$ 506	3582 $\pm$ 1093	7398 $\pm$ 118	8509 $\pm$ 422
	5M	5589 $\pm$ 758	4615 $\pm$ 2022	6329 $\pm$ 1510	8547 $\pm$ 84	10133 $\pm$ 966
Humanoid	300k	1344 $\pm$ 365	1997 $\pm$ 483	3117 $\pm$ 910	3181 $\pm$ 771	5332 $\pm$ 714
	1M	5165 $\pm$ 145	4909 $\pm$ 364	6029 $\pm$ 531	6032 $\pm$ 334	7429 $\pm$ 153
	5M	5433 $\pm$ 245	6555 $\pm$ 279	8361 $\pm$ 1364	8951 $\pm$ 246	10281 $\pm$ 588

Offline – D4RL

Table 2: Average final performance on the D4RL benchmark after training for 1M time steps. over 10 trials, where $\pm$ captures a 95% confidence interval. The highest performance is highlighted. Any performance which is not statistically significantly worse than the highest performance (according to a Welch's t-test with significance level 0.05) is highlighted.

Environment	Dataset	CQL	TD3+BC	IQL	$\mathcal{X}$-QL	TD7
HalfCheetah	Medium	46.7 $\pm$ 0.3	48.1 $\pm$ 0.1	47.4 $\pm$ 0.2	47.4 $\pm$ 0.1	58.0 $\pm$ 0.4
	Medium-Replay	45.5 $\pm$ 0.3	44.6 $\pm$ 0.4	43.9 $\pm$ 1.3	44.2 $\pm$ 0.7	53.8 $\pm$ 0.8
	Medium-Expert	76.8 $\pm$ 7.4	93.7 $\pm$ 0.9	89.6 $\pm$ 3.5	90.2 $\pm$ 2.7	104.6 $\pm$ 1.6
Hopper	Medium	59.3 $\pm$ 3.3	59.1 $\pm$ 3.0	63.9 $\pm$ 4.9	67.7 $\pm$ 3.6	76.1 $\pm$ 5.1
	Medium-Replay	78.8 $\pm$ 10.9	52.0 $\pm$ 10.6	93.4 $\pm$ 7.8	82.0 $\pm$ 14.9	91.1 $\pm$ 8.0
	Medium-Expert	79.9 $\pm$ 19.8	98.1 $\pm$ 10.7	64.2 $\pm$ 32.0	92.0 $\pm$ 10.0	108.2 $\pm$ 4.8
Walker2d	Medium	81.4 $\pm$ 1.7	84.3 $\pm$ 0.8	84.2 $\pm$ 1.6	79.2 $\pm$ 4.0	91.1 $\pm$ 7.8
	Medium-Replay	79.9 $\pm$ 3.6	81.0 $\pm$ 3.4	71.2 $\pm$ 8.3	61.8 $\pm$ 7.7	89.7 $\pm$ 4.7
	Medium-Expert	108.5 $\pm$ 1.2	110.5 $\pm$ 0.4	108.9 $\pm$ 1.4	110.3 $\pm$ 0.2	111.8 $\pm$ 0.6
Total		656.7 $\pm$ 24.3	671.3 $\pm$ 15.7	666.7 $\pm$ 34.6	674.9 $\pm$ 20.4	784.4 $\pm$ 14.1

Ablation & Run Time

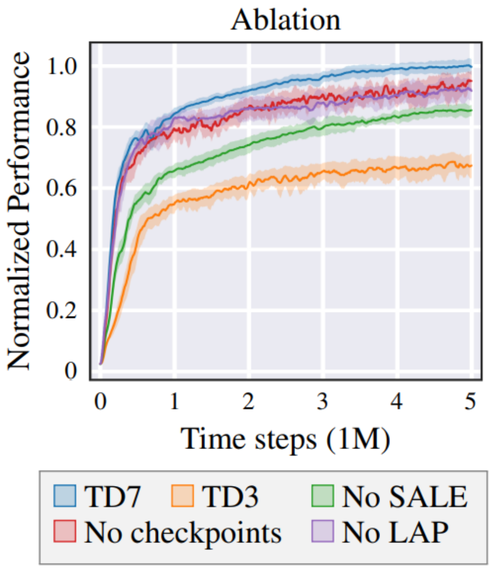


Figure 5: Ablation study over the components of TD7. The y-axis corresponds to the average performance over all five MuJoCo tasks, normalized with respect to the performance of TD7 at 5M time steps. The shaded area captures a 95% confidence interval.

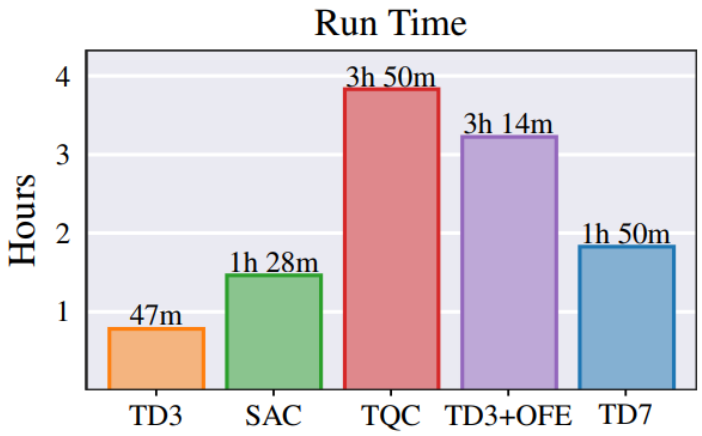


Figure 6: Run time of each method for 1M time steps on the HalfCheetah environment, using the same hardware and deep learning framework (PyTorch [Paszke et al., 2019]).

SALE: State-Action Learned Embeddings

Main idea of TD7: Encode state-action pairs to latent space

Why would we need to encode an already compact low-level state?

- There's more to learn about the underlying dynamics of the environment.
- There's more to learn about the interaction between states and actions.

Extensive experiments to verify their design choice. (Section 4.2, Appendix D)

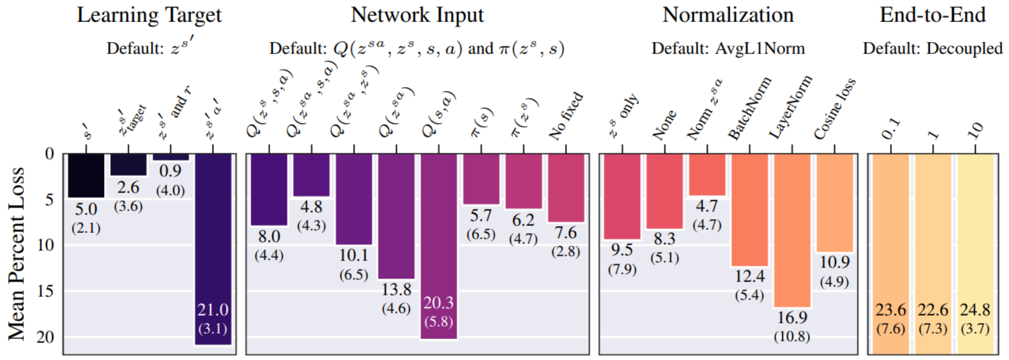


Figure 3: The mean percent loss from using alternate design choices in TD7 at 1M time steps, over 10 seeds and the five benchmark MuJoCo environments. Bracketed values describe the range of the 95% confidence interval around the mean. Percent loss is computed against TD7 where the default choices correspond to a percent loss of 0. See Section 4.2 for a description of each design choice and key observations. See the Appendix for further implementation-level details.

SALE: State-Action Learned Embeddings

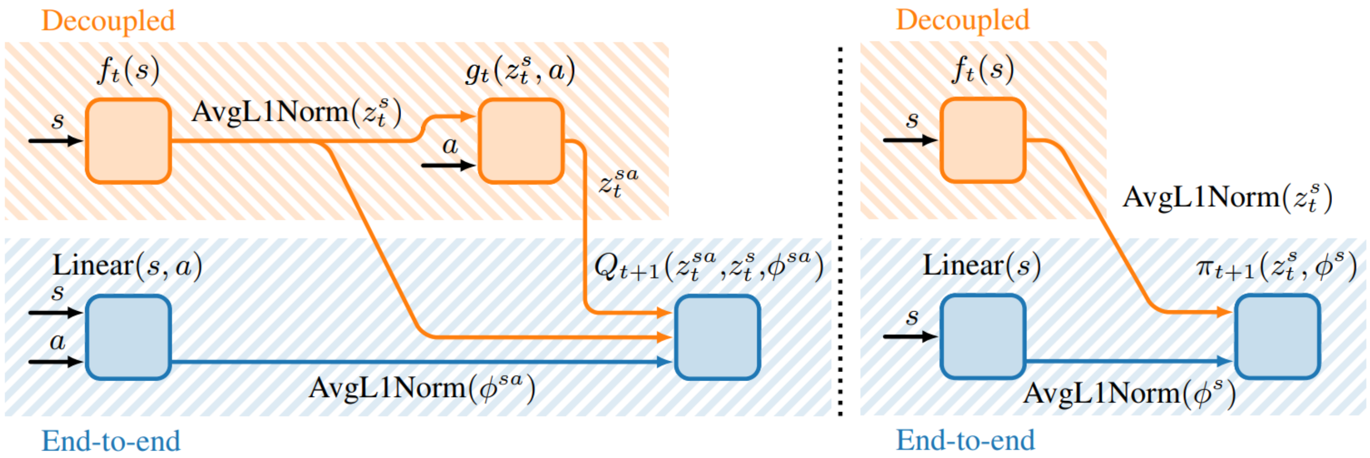
State encoder f, State-Action encoder g

- State-Action embedding $g(f(s), a)$ tries to predict next State embedding $f(s')$ (≈ SPR)

$$\mathcal{L}(f, g) := \left(g(f(s), a) - |f(s')|_\times\right)^2 = \left(z^{sa} - |z^{s'}|_\times\right)^2, \tag{2}$$

- Embeddings are used in main networks $Linear, Q, \pi$, but with stop gradient $|\cdot|_\times$.

  i.e., the encoders are trained only with MSE loss.



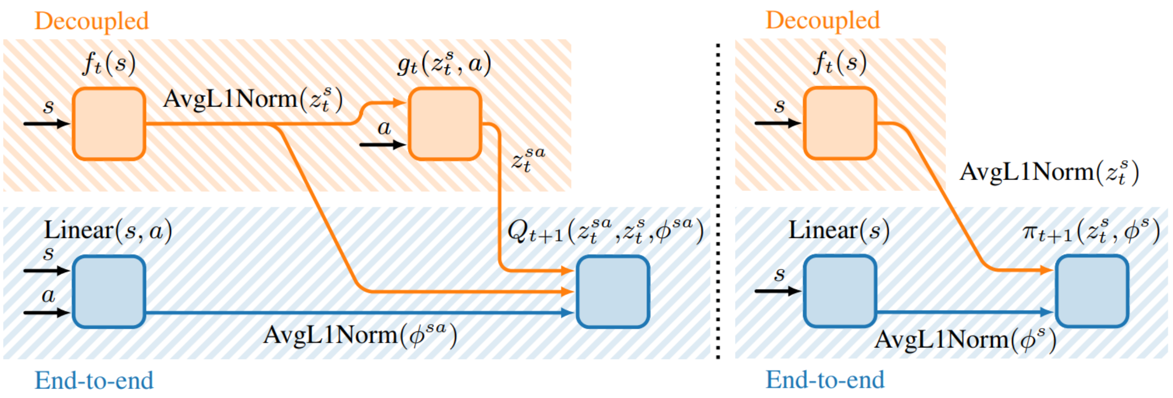
(a) Value function Q with SALE (b) Policy π with SALE

SALE: State-Action Learned Embeddings

AvgL1Norm: Keeping the embedding scale constant.

- To prevent monotonic growth or collapse to a redundant representation.
  ?
- Seems to be inspired from cosine similarity loss from BYOL and SPR.
- Not applied on State-Action embedding z_t^{sa} since it targets an already normalized embedding $z_t^{s'}$.

$$\text{AvgL1Norm}(x) := \frac{x}{\frac{1}{N}\sum_i |x_i|}.$$



(a) Value function Q with SALE (b) Policy π with SALE

SALE: State-Action Learned Embeddings

Fixed embeddings: Using target networks on embedding networks

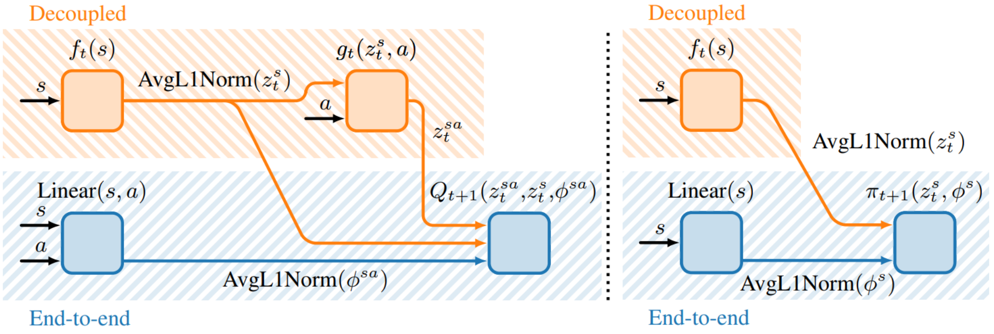
- Embeddings have a non-stationary problem (just like value targets).
- Copy to target network every n steps.

$$Q_t \leftarrow Q_{t+1}, \qquad \pi_t \leftarrow \pi_{t+1}, \qquad (f_{t-1}, g_{t-1}) \leftarrow (f_t, g_t), \qquad (f_t, g_t) \leftarrow (f_{t+1}, g_{t+1}). \tag{8}$$

- Q_{t+1}, π_{t+1} always use embeddings from f_t, g_t, and Q_t, π_t use those from f_{t-1}, g_{t-1}.

$$Q_{t+1}(z_t^{sa}, z_t^s, s, a) \approx r + \gamma Q_t(z_{t-1}^{s'a'}, z_{t-1}^{s'}, s', a'), \qquad \text{where } a' \sim \pi_t(z_{t-1}^{s'}, s'), \tag{6}$$

$$\pi_{t+1}(z_t^s, s) \approx \operatorname*{argmax}_\pi Q_{t+1}(z_t^{sa}, z_t^s, s, a), \qquad \text{where } a \sim \pi(z_t^s, s). \tag{7}$$

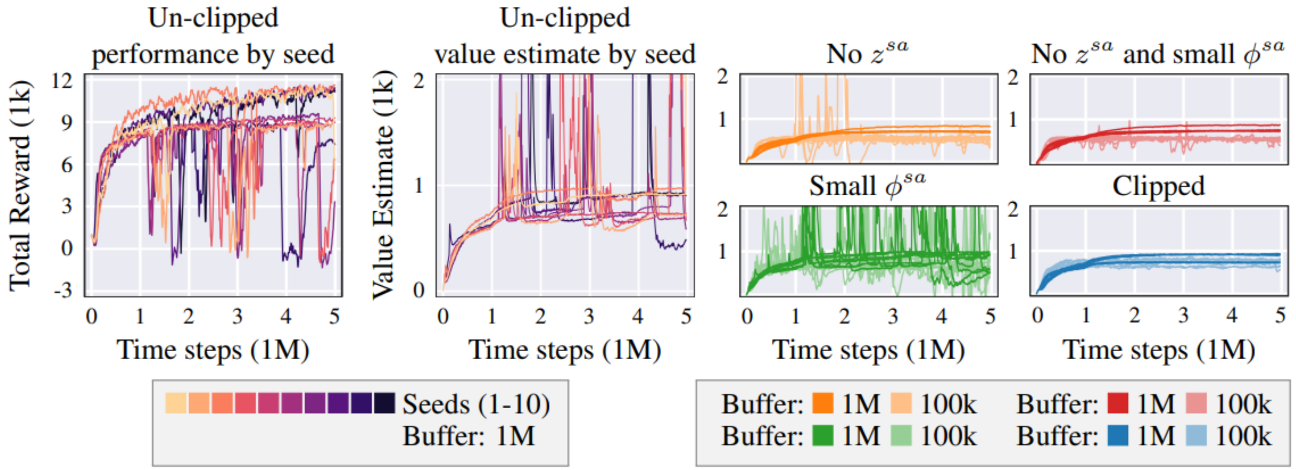


(a) Value function Q with SALE (b) Policy π with SALE

SALE: State-Action Learned Embeddings

Clipping values: Dealing with extrapolation error

- Using learned embeddings caused jumps in value estimation – likely due to action extrapolation.



- Luckily this will naturally heal in online setting, so all we have to do is minimize the damage.
- Clip the target value into min/max of previous Q values.

...what about offline...?

$$\cancel{Q_{t+1}(z_t^{sa}, z_t^{s}, s, a) \approx r + \gamma Q_t(z_{t-1}^{s'a'}, z_{t-1}^{s'}, s', a'), \quad \text{where } a' \sim \pi_t(z_{t-1}^{s'}, s'),} \tag{6}$$

$$Q_{t+1}(s, a) \approx r + \gamma \operatorname{clip}\left(Q_t(s', a'), \min_{(s,a)\in D} Q_t(s, a), \max_{(s,a)\in D} Q_t(s, a)\right). \tag{9}$$

*Embeddings omitted

LAP: Loss Adjusted PER

LAP = PER + Huber Loss + Priority clipping

Main theory: Any loss $\mathcal{L}_A$ *with* PER has an equivalent $\mathcal{L}_B$ *without* PER (in expectation).

- e.g., L_1 loss with prioritized sampling has the same expected gradient as L_2 loss with uniform sampling.

$$\underbrace{\mathbb{E}_{\mathcal{U}}[\nabla_Q \mathcal{L}_{\text{MSE}}(\delta(i))]}_{\text{expected gradient of MSE under } \mathcal{U}} = \underbrace{\mathbb{E}_{\mathcal{D}_2}\left[\frac{\sum_j \delta(j)}{N|\delta(i)|}\delta(i)\right]}_{\text{by Equation (5)}} \propto \mathbb{E}_{\mathcal{D}_2}\underbrace{[\text{sign}(\delta(i))]}_{\nabla_Q \mathcal{L}_{\text{L1}}(\delta(i))} = \underbrace{\mathbb{E}_{\mathcal{D}_2}[\nabla_Q \mathcal{L}_{\text{L1}}(\delta(i))]}_{\text{expected gradient of L1 under } \mathcal{D}_2}$$

$\delta(i) = Q(i) - y(i)$ $\mathcal{D}_2$ to be a prioritized sampling scheme $p(i) = \frac{|\delta(i)|}{\sum_{j \in \mathcal{B}} |\delta(j)|}$

Does that mean PER is meaningless?

- No, because their variance are not the same.
- In fact, it can be shown that prioritization scheme lowers the variance (albeit often marginal in practice).

LAP: Loss Adjusted PER

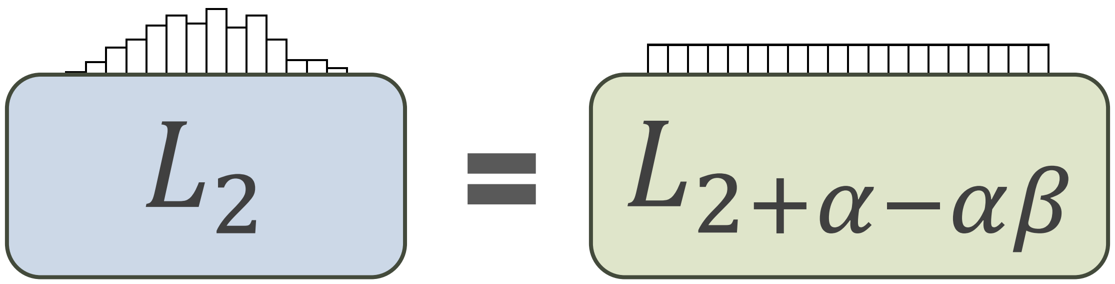
This point of view reveals the problem of using prioritized L_2 loss (which is what we often use).

Roughly, a prioritized L_τ loss is equivalent to a uniform $L_{\tau+\alpha-\alpha\beta}$ loss.

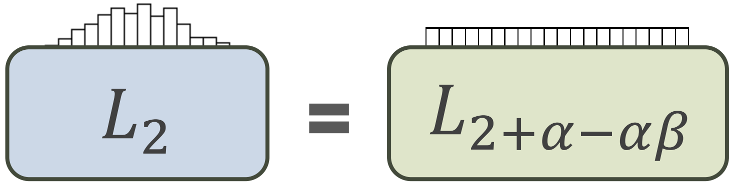
- e.g., A prioritized L_2 loss is equivalent to a uniform(non-prioritized) $L_{2+\alpha-\alpha\beta}$ loss.
- $\alpha, \beta \in [0,1]$ are the hyperparameters of PER.

Theorem 3 *The expected gradient of a loss $\frac{1}{\tau}|\delta(i)|^\tau$, where $\tau > 0$, when used with PER is equal to the expected gradient of the following loss when using a uniformly sampled replay buffer:*

$$\mathcal{L}^\tau_{\text{PER}}(\delta(i)) = \frac{\eta N}{\tau + \alpha - \alpha\beta}|\delta(i)|^{\tau+\alpha-\alpha\beta}, \qquad \eta = \frac{\min_j |\delta(j)|^{\alpha\beta}}{\sum_j |\delta(j)|^\alpha}. \tag{7}$$



LAP: Loss Adjusted PER



Now the question becomes: "Is $L_{2+\alpha-\alpha\beta}$ a good loss?"

Giving the answer first: "Since $2+\alpha-\alpha\beta \geq 2$, no."

Why? Firstly, the uniform L_2 loss is an unbiased objective for value learning (Observation 2).

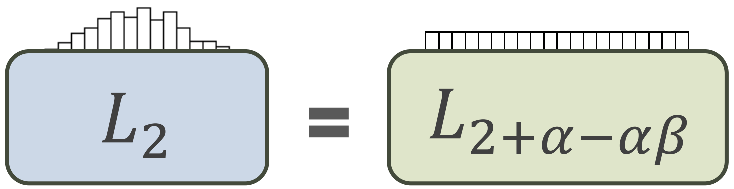
But going over L_2 (i.e., $L_x, 2 < x$) would over-exaggerate high errors (i.e., favor outliers).

Observation 2 *(MSE) Let $\mathcal{B}(s,a) \subset \mathcal{B}$ be the subset of transitions containing (s,a) and $\delta(i) = Q(i) - y(i)$. If $\nabla_Q \mathbb{E}_{i\sim\mathcal{B}(s,a)}[0.5\delta(i)^2] = 0$ then $Q(s,a) = \mathrm{mean}_{i\in\mathcal{B}(s,a)} y(i)$.*

And that's what's exactly happening in prioritized L_2 (or uniform $L_{2+\alpha-\alpha\beta}$) loss!

- $\alpha, \beta \in [0,1]$ guarantees that $0 \leq \alpha - \alpha\beta$, meaning $2+\alpha-\alpha\beta \geq 2$.
- In short, L_2 with prioritization scheme is actually a biased objective; a *badly* biased one.

LAP: Loss Adjusted PER



So how do we fix this?

Simply use L_1 loss in prioritized scheme ($\tau = 1$), so we can have $1 \leq 1 + \alpha - \alpha\beta \leq 2$.

There's also a good reason why L_1 is not a bad idea, but let's skip that.

Fix one last problem, and we have LAP!

L_1 loss is terrible around optimal point, since the gradient doesn't saturate.

We'll have to resort back to L_2 around optimal point: Huber loss

As said before, prioritized L_2 favors high-error outliers: Clip all low-error samples to 1

$$p(i) = \frac{\max(|\delta(i)|^\alpha, 1)}{\sum_j \max(|\delta(j)|^\alpha, 1)}, \qquad \mathcal{L}_{\text{Huber}}(\delta(i)) = \begin{cases} 0.5\delta(i)^2 & \text{if } |\delta(i)| \leq 1, \\ |\delta(i)| & \text{otherwise.} \end{cases} \tag{9}$$

Fun fact: LAP also has an equivalent non-prioritized loss, named PAL.

Policy Checkpoints (for online RL)

No matter how hard we try, function approximation & RL is inherently unstable.

Keep the ‘peak performance’ policy during training, use that for evaluation.

Problem: How can we not waste time evaluating policies?

Solution: Do off-policy RL with the samples obtained during evaluation!

- Standard off-policy RL: Collect a data point → train once
- Proposed: Collect N data points over several evaluation episodes → train N times

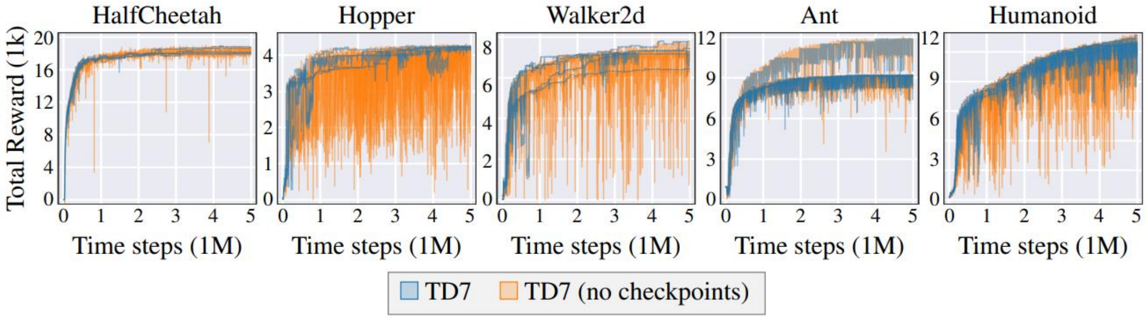


Figure 30: **Performance of individual seeds with and without checkpoints.** Learning curves of five individual seeds, with and without checkpoints, on the MuJoCo benchmark. The shaded area captures a 95% confidence interval around the average performance.

Policy Checkpoints (for online RL)

Let's be smarter about this.

1. Evaluate with minimum performance and not average.
   - Avoids policies that are unstable, even if they're better on average.
   - This also allows us to prematurely halt the evaluation and move onto the next policy.
2. Restrict evaluation to 1 episode in early learning stage (750k steps).
   - Early stage policies require more exploration and fast feedback.

Surprisingly works well, even with high number of evaluation episodes (20+).

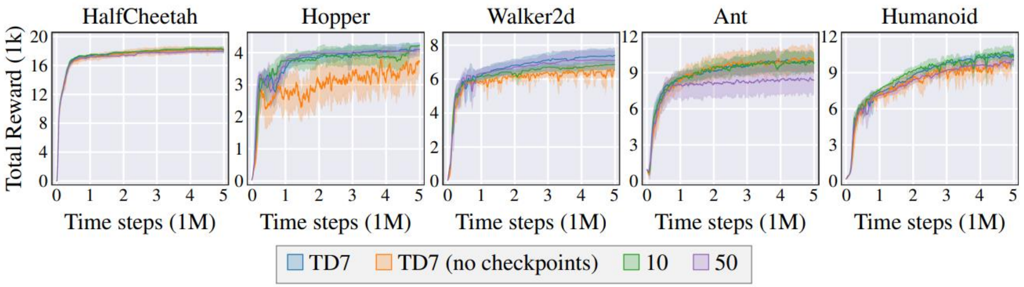


Figure 26: **Maximum number of assessment episodes.** Learning curves on the MuJoCo benchmark, varying the maximum number episodes that the policy is fixed for. Results are averaged over 10 seeds. The shaded area captures a 95% confidence interval around the average performance.

Behavior Cloning (for offline RL)

Direct application of TD3 + BC.

Constrain behavior policy to the offline dataset's action distribution.

$$\pi \approx \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{(s,a)\sim D} \left[Q(s,\pi(s)) - \lambda |\mathbb{E}_{s\sim D}\left[Q(s,\pi(s))\right]|_{\times} \left(\pi(s) - a\right)^2 \right]. \tag{11}$$

$$\mathcal{L}(\pi_{t+1}) := -Q + \lambda |\mathbb{E}_{s\sim D}\left[Q\right]|_{\times} \left(a_\pi - a\right)^2, \tag{22}$$

Why the $\mathbb{E}_{s\sim D}[Q]$?

- Original TD3 + BC normalizes the Q value loss to match the scale between the two losses.

$$\pi = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{s\sim\mathcal{D}} \left[Q(s,\pi(s))\right] \rightarrow \pi = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\lambda Q(s,\pi(s)) - \left(\pi(s) - a\right)^2 \right]. \tag{3}$$

$$\lambda = \frac{\alpha}{\frac{1}{N}\sum_{(s_i,a_i)} |Q(s_i,a_i)|}. \tag{5}$$

- (It seems like) TD7 upscales BC loss instead of downscaling Q loss.

Overall Algorithm

TD7 (TD3+4 additions) has several networks and sub-components:

- Two value functions $(Q_{t+1,1}, Q_{t+1,2})$.
- Two target value functions $(Q_{t,1}, Q_{t,2})$.
- A policy network π_{t+1}.
- A target policy network π_t.
- An encoder, with sub-components (f_{t+1}, g_{t+1}).
- A fixed encoder, with sub-components (f_t, g_t).
- A target fixed encoder with sub-components (f_{t-1}, g_{t-1}).
- A checkpoint policy π_c and checkpoint encoder f_c (g is not needed).

Algorithm 2 TD7 Train Function

1: Sample transition from LAP replay buffer with probability (Equation 30).
2: Train encoder (Equation 13).
3: Train value function (Equation 15).
4: Update $(Q_{\min}, Q_{\max})$ (Equations 20 & 21).
5: **if** i mod `policy_update_frequency` $= 0$ **then**
6: Train policy (Equation 22).
7: **if** i mod `target_update_frequency` $= 0$ **then**
8: Update target networks (Equation 26).

Algorithm 4 Policy Checkpoints with Minimum Performance and Early Termination

1: **for** episode $= 1$ **to** assessment_episodes **do** ▷ *Assessment*
2: Follow the current policy π_{t+1} and determine episode_reward.
3: min_performance ← min(min_performance, episode_reward).
4: Increment timesteps_since_training by the length of the episode.
5: **if** min_performance ≤ checkpoint_performance **then** ▷ *Early termination*
6: End current assessment.
7: **if** min_performance ≥ checkpoint_performance **then** ▷ *Checkpointing*
8: Update checkpoint networks $\pi_c \leftarrow \pi_{t+1}$, $f_c \leftarrow f_t$.
9: checkpoint_performance ← min_performance
10: **for** $i = 1$ **to** timesteps_since_training **do** ▷ *Training*
11: Train RL agent.
12: Reset min_performance.

Encoders

$$z^s := f(s), \qquad z^{sa} := g(z^s, a). \tag{12}$$

$$\mathcal{L}(f_{t+1}, g_{t+1}) := \Big(g_{t+1}(f_{t+1}(s), a) - |f_{t+1}(s')|_\times \Big)^2 \tag{13}$$

$$= \Big(z^{sa}_{t+1} - |z^{s'}_{t+1}|_\times \Big)^2, \tag{14}$$

Value functions

$$\mathcal{L}(Q_{t+1}) := \text{Huber}\Big(\texttt{target} - Q_{t+1}(z^{sa}_t, z^s_t, s, a)\Big), \tag{15}$$

$$\texttt{target} := r + \gamma \,\text{clip}\big(\min\left(Q_{t,1}(x), Q_{t,2}(x)\right), Q_{\min}, Q_{\max}\big), \tag{16}$$

$$x := [z^{s'a'}_{t-1}, z^{s'}_{t-1}, s', a'], \tag{17}$$

$$a' := \pi_t(z^{s'}_{t-1}, s') + \epsilon, \tag{18}$$

$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c). \tag{19}$$

$$Q_{\min} \leftarrow \min\left(Q_{\min}, \texttt{target}\right), \tag{20}$$

$$Q_{\max} \leftarrow \max\left(Q_{\max}, \texttt{target}\right), \tag{21}$$

Policy

$$\mathcal{L}(\pi_{t+1}) := -Q + \lambda |\mathbb{E}_{s\sim D}\left[Q\right]|_\times \left(a_\pi - a\right)^2, \tag{22}$$

$$Q := 0.5\left(Q_{t+1,1}(x) + Q_{t+1,2}(x)\right) \tag{23}$$

$$x := [z^{sa_\pi}_t, z^s_t, s, a_\pi], \tag{24}$$

$$a_\pi := \pi_{t+1}(z^s_t, s). \tag{25}$$

$$(Q_{t,1}, Q_{t,2}) \leftarrow (Q_{t+1,1}, Q_{t+1,2}), \tag{26}$$

$$\pi_t \leftarrow \pi_{t+1}, \tag{27}$$

$$(f_{t-1}, g_{t-1}) \leftarrow (f_t, g_t), \tag{28}$$

$$(f_t, g_t) \leftarrow (f_{t+1}, g_{t+1}). \tag{29}$$

PER

$$p(i) = \frac{\max\left(|\delta(i)|^\alpha, 1\right)}{\sum_{j\in D}\max\left(|\delta(j)|^\alpha, 1\right)}, \tag{30}$$

$$|\delta(i)| := \max\Big(|Q_{t+1,1}(z^{sa}_t, z^s_t, s, a) - \texttt{target}|, |Q_{t+1,2}(z^{sa}_t, z^s_t, s, a) - \texttt{target}|\Big), \tag{31}$$

Overall Algorithm

SALE | LAP | BC | TD3 | Others

Encoders

$$z^s := f(s), \qquad z^{sa} := g(z^s, a). \tag{12}$$

$$\mathcal{L}(f_{t+1}, g_{t+1}) := \left(g_{t+1}(f_{t+1}(s), a) - |f_{t+1}(s')|_\times \right)^2 \tag{13}$$

$$= \left(z^{sa}_{t+1} - |z^{s'}_{t+1}|_\times \right)^2, \tag{14}$$

Value functions

$$\mathcal{L}(Q_{t+1}) := \text{Huber}\left(\texttt{target} - Q_{t+1}(z^{sa}_t, z^s_t, s, a)\right), \tag{15}$$

$$\texttt{target} := r + \gamma\, \text{clip}\left(\min\left(Q_{t,1}(x), Q_{t,2}(x)\right), Q_{\min}, Q_{\max}\right), \tag{16}$$

$$x := [z^{s'a'}_{t-1}, z^{s'}_{t-1}, s', a'], \tag{17}$$

$$a' := \pi_t(z^{s'}_{t-1}, s') + \epsilon, \tag{18}$$

$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c). \tag{19}$$

$$Q_{\min} \leftarrow \min\left(Q_{\min}, \texttt{target}\right), \tag{20}$$

$$Q_{\max} \leftarrow \max\left(Q_{\max}, \texttt{target}\right), \tag{21}$$

Action Extrapolation Error

Clipped Double Q-Learning

Action Smoothing Regularization

Action Extrapolation Error

Policy

$$\mathcal{L}(\pi_{t+1}) := -Q + \lambda |\mathbb{E}_{s\sim D}\left[Q\right]|_\times \left(a_\pi - a\right)^2, \tag{22}$$

$$Q := 0.5\left(Q_{t+1,1}(x) + Q_{t+1,2}(x)\right) \tag{23}$$

$$x := [z^{sa_\pi}_t, z^s_t, s, a_\pi], \tag{24}$$

$$a_\pi := \pi_{t+1}(z^s_t, s). \tag{25}$$

$$(Q_{t,1}, Q_{t,2}) \leftarrow (Q_{t+1,1}, Q_{t+1,2}), \tag{26}$$

$$\pi_t \leftarrow \pi_{t+1}, \tag{27}$$

$$(f_{t-1}, g_{t-1}) \leftarrow (f_t, g_t), \tag{28}$$

$$(f_t, g_t) \leftarrow (f_{t+1}, g_{t+1}). \tag{29}$$

Actor loss = Average of two critics (Empirical choice)

cf) TD3 used only one of the critics $Q_{t+1,1}$

Fixed Embeddings (Target Network)

PER

$$p(i) = \frac{\max\left(|\delta(i)|^\alpha, 1\right)}{\sum_{j\in D} \max\left(|\delta(j)|^\alpha, 1\right)}, \tag{30}$$

$$|\delta(i)| := \max\left(|Q_{t+1,1}(z^{sa}_t, z^s_t, s, a) - \texttt{target}|, |Q_{t+1,2}(z^{sa}_t, z^s_t, s, a) - \texttt{target}|\right), \tag{31}$$

Priority is based on maximum of two critics

TalkRL Podcast

S. Fujimoto already said (in 2019) Mujoco is a dying benchmark.

- Then why did he do this in Mujoco... is the mystery.

And it seems like he was thinking about state-action embedding.

Paper List

(TD3) Addressing Function Approximation Error in Actor-Critic Methods, S. Fujimoto et al.

(TD3+BC) A Minimalist Approach to Offline Reinforcement Learning, S. Fujimoto et al.

(LAP) An Equivalence between Loss Functions and Non-Uniform Sampling in Experience Replay , S. Fujimoto et al.

(TD7) For SALE: State-Action Representation Learning for Deep Reinforcement Learning , S. Fujimoto et al.