Towards Reinforcement Learning with Data Summarization

Yuyang Chen **Wenxuan Han**
{ yuach, hanwenxuan66}@ucla.edu
Department of Computer Science, UCLA, Los Angeles, CA, 90024.

Abstract

We explore the integration of CREST data summarization into Reinforcement Learning (RL), and evaluating our method with autonomous driving in the MetaDrive environment. Implementing CREST may augment an RL algorithm's sample efficiency using gradient-based core-set selection, which allows the model to focus on informative data. In this paper, we implement CREST into DDPG (DDPG-S), and we benchmark it against State of the Art RL algorithms like TD3 and PPO in diverse driving simulations. Our initial findings suggest that data summarization can enhance sample efficiency and accelerate RL, which paves the way for further research in this field. Github.

1 Introduction

Reinforcement Learning is a frontier of machine learning that has gathered a lot of interest in the field of research. Through various advancements, it is shown that Reinforcement Learning can achieve superhuman performances, but at a cost of data efficiency. On average, value-based learning requires 100 thousand to a million samples, and policy-based learning requires millions more. While on the other hand, in the field of supervised learning and unsupervised learning, they have methods that can produce stellar results for only a few training samples through careful optimization. There are differences in overall approach when it comes to Reinforcement Learning against Supervised / Unsupervised methods, but they are also quite similar in other aspects.

One of such similarities that we attempt to utilize is with the "quality" prediction mechanism that exists in many Reinforcement Learning methods. It takes a state (and/or action) that happened in a particular time, and predicts how good (or the "quality") of the action, in hopes to achieve control and optimization of the decision policy. This is very similar to a classifier neural network, where it would take an input and predict something about it.

In the field of supervised learning, there are many optimization approaches that increases the data efficiency of it. One of such is with data summarizing, where one would learn from the summarized training data instead of learning from data directly, which allows it to converge faster, better, and with robustness against various training anomalies. This method is most evident in CRAIG with Stochastic Gradient Descent, then in the most recent years, CREST with mini-batch Gradient Descent. It is shown that CREST achieves similar or better results compared to vanilla mini-batch GD in various supervised learning tasks.

Hence, we propose, that CREST can be adapted to be used in an Reinforcement Learning scenario, in order to mitigate Reinforcement Learning's weakness of sample efficiency and improve training performance.

In this paper, we will describe our approach on implementing CREST core set selection into a Reinforcement Learning algorithm, and evaluate its performance against other state of the art reinforcement learning algorithms in the autonomous driving task MetaDrive simulator. We hope that, with this work as a naive approach, data set summarization methods can be utilized, optimized, and incorporated further with various Reinforcement Learning techniques to address the biggest issue that is with this field of machine learning.

2 Related Work

Reinforcement Learning
Reinforcement learning (RL) has been at the forefront of AI research due to its potential in solving complex decision-making problems. It emphasizes the strategic selection of actions by agents in an environment to optimize the total accumulated reward. It is especially pertinent in scenarios like autonomous driving where the immediate impact of an action is not apparent, but its effects unfold progressively over time.

Deep Q-Networks (DQN)
The integration of deep learning with Q-learning, resulting in Deep Q-Networks, revolutionized the application of RL in high-dimensional spaces. This methodology, introduced by [5] Minh et al. (2015), employs convolution neural networks to approximate Q-values, enabling RL models to tackle problems previously considered intractable due to their complexity. DQN's success in various domains, particularly in gaming, underlined its versatility and efficacy.

Actor-Critic
The Actor-Critic approach for value-based Reinforcement Learning (DDPG), detailed by [2] Konda and Tsitsiklis (2000), combines Deep Q Learning with an actor network so it can make decisions that are continuous (not categorical). [1] Fujimoto et al. (2018) further refined this approach in their study, addressing function approximation errors in Actor-Critic methods. Their work, focusing on the stability and reliability of these methods, significantly enhanced the performance of RL algorithms in environments with continuous action spaces.

MetaDrive
[3] Li et al. (2022) introduced MetaDrive, a simulation platform designed to create diverse driving scenarios for generalizable RL. This platform offers a highly versatile environment for testing and developing RL algorithms, providing an array of customizable and challenging driving conditions. The application of MetaDrive in RL research has enabled more robust testing and development of algorithms, ensuring their effectiveness in real-world scenarios.

PPO and TD3
Proximal Policy Optimization (PPO) and Twin Delayed DDPG (TD3), introduced by [6] Schulman et al. (2017) and [1] Fujimoto et al. (2018) respectively, represent the cutting edge in RL algorithms. Both algorithms have set new standards in terms of stability and performance, particularly in complex and unpredictable environments. In our study, we use PPO and TD3 as benchmarks for comparison, evaluating the efficiency and effectiveness of our proposed model against these state-of-the-art algorithms.

CREST for Data Summarization
CREST method, as proposed by [7] Yang et al. (2023), was a novel method for data-efficient deep learning, focusing on creating core-sets for sustainable learning. This approach optimizes the training process by selecting the most informative data points, thereby reducing the computational load without sacrificing model performance.

While the state-of-the-art algorithms PPO and TD3 have set benchmarks in RL, there remains a notable gap in the integration of efficient data summarization techniques within these frameworks. Current work primarily focuses on algorithmic advancements without fully addressing the challenges of data efficiency and model generalization in dynamically changing environments. Our study addresses this gap by integrating the CREST method with RL algorithms in the MetaDrive environment. This novel approach aims to enhance learning efficiency by selectively focusing on the most informative data, thereby improving sample efficiency and model adaptability in complex driving scenarios. Thus, our research contributes to the field by demonstrating the practical applicability and benefits of data summarization in reinforcement learning, particularly in the context of autonomous driving simulations.

3 Problem Formulation

Baseline: DDPG formulation:

We would characterize our baseline DDPG model with the following:

The Actor model decides what action should take, given a state

$$\mathrm{ACTOR}(s) = a$$

In which the loss of this function would be the negation of the predicted quality of the action

$$\begin{aligned}\nabla_{\mathrm{A}}(s) &= \nabla_{\mathrm{A}} - \mathrm{CRITIC}(s, a)\\ &= \nabla_{\mathrm{A}} - \mathrm{C}(s, \mathrm{A}(s))\end{aligned}$$

The critic takes a state and predicts how good the state is.

$$\text{CRITIC}(s, a) = q$$

The loss function of the critic is calculated using the bellman backup equation with the future discounted reward being generated by an target critic model (a copy of the target model)

$$\begin{aligned} &\nabla_{\text{CRITIC}}(s, a) \\ &= \nabla_{\text{C}}\text{MSE}(q, q_{tgt}) \\ &= \nabla_{\text{C}}\text{MSE}(q, R(s, a) + \gamma \cdot \text{C}_{tgt}(s', a')) \\ &= \nabla_{\text{C}}\text{MSE}(\text{C}(s, a), R(s, a) + \gamma \cdot \text{C}_{tgt}(s', a')) \end{aligned}$$

Where $R(s, a)$ is the reward returned by the environment given the current state and action, and (s', a') is the next state and next proposed action (by actor) given the current (s, a) has been executed.

Greedy CoreSet Selection for Maximized Intra-set Diversity:

Define Pairwise Distances: For each $r_i \in \text{RSet}(t)$, create a distance matrix D^i where D^i_{jk} measures the distance between r_j and r_k in r_i.

For each $r_i \in \text{RSet}(t)$:

1. Initialize $C_i = \{\text{initial element}\}$
2. While $|C_i| <$ desired size :

   a. Find $r_{next} = \arg \max_{r \in r_i \setminus C_i} \left(\frac{1}{|C_i|} \sum_{c \in C_i} D^i_{rc} \right)$

   b. Add r_{next} to C_i

The greedy algorithm ensures that each new addition to C_i maximizes the average distance to the existing elements in C_i, thereby aiming to cover a diverse range of data points within each individual random set.

Formulation for our approach to combine CREST into DDPG:

Given we have a replay buffer that contains the training data formulated as follows:

$$\text{Rp}_i = (s_i, a_i, r_i = R(s_i, a_i), s'_i, a'_i)$$

We would generate multiple (x) random sets from the replay buffer and generate core-sets for each.

$$\begin{aligned} &\text{Coreset}_x \\ &= \text{CSelect}(\text{RndSet}(Rp, x)) \\ &= \text{CSelect}(r_1, r_2 ... r_x \in r) \\ &= cs_1, cs_2 ... cs_x \in cs, w_1, w_2 ... w_x \in w \end{aligned}$$

cs is the list of core-sets generated, and w is the list of weights per element in the core-set.

We will be testing to see if the coresets can mimic the gradients of mini-batch gradient descent, such that evaluating and back propagating on the summarized core-sets mimics evaluating and back propagating on raw data.

$$\begin{aligned} &\nabla_{\text{A}}(s) = \nabla_{\text{A}} - \text{C}(s, \text{A}(s)) \\ \sim &\nabla_{\text{A}}(s_{cs}) = \nabla_{\text{A}} - \text{C}(s_{cs}, \text{A}(s_{cs})) * w \end{aligned}$$

$$\begin{aligned} &\nabla_{\text{C}}(s, a) \\ &= \nabla_{\text{C}}\text{MSE}(\text{C}(s, a), R(s, a) + \gamma \cdot \text{C}_{tgt}(s', a')) \\ &\sim \nabla_{\text{C}}(s_{cs}, a_{cs}) \\ &= w_{cs} * \nabla_{\text{C}}\text{MSE}(\text{C}(s_{cs}, a_{cs}), R(s_{cs}, a_{cs}) + \\ &\quad \gamma \cdot \text{C}_{tgt}(s'_{cs}, a'_{cs})) \end{aligned}$$

4 Method

Our method is to implement CREST core set selection as a routine to summarize training data for our Reinforcement Learner. We have chosen to use DDPG as our baseline RL algorithm.

The main changes are as follows:

1. At a certain interval $coresetUpdateInterval$, generate the core-set by:
   (a) generating $numRandomSet$ random sets from the replay buffer of size $randomSetSize$
   (b) for each sample in the replay buffer, forward, calculate loss, and back propagate only the last layer for both actor and critic.
   (c) Collect gradients from both the last layers of actor and critic model, concatenate them, and store in a list.
   (d) Assign the random sets' elements with their corresponding gradients.
   (e) Using the gradients as indicator, run facility distance and select coresets of size $coresetSize$ for each

of their random sets. This also generates the weights for each element of the core-sets

2. Forward pass on one of the core-sets, for each training step, for both actor and critic
3. weight their loss with the core-set weights for each element of the core-set used, then back propagate, for both actor and critic

For our testing, we chose to use MetaDrive, a simulation environment for autonomous driving, to evaluate our model and compare it against other SOTA Reinforcement Learning methods. In specific, we chose MetaDrive-Tut-Hard-v0 environment with a fixed seed, which simulates 20 scenarios simultaneously.

To optimize the hyperparameters, we used Grid Search to search for a set of hyperparameters that suits this scenario for our implementation. The hyperparameter for our model for evaluation is as follows:

1. totalSteps: 100,000 frames
2. coresetUpdateInterval: 2500 frames
3. numRandomSet: 15 sets
4. randomSetSize: 2000 frames
5. coresetSize: 64 frames(items)
6. replayBufferSize: unlimited (100,000 frames)

For our comparison targets, we choose to compare our model against the baseline DDPG, as well as SOTA RL algorithms TD3 and PPO. Their configurations are as follows:

DDPG:

1. totalSteps: 100,000 frames
2. replayBufferSize: unlimited (100,000 frames)

TD3:

1. totalSteps: 100,000 frames
2. replayBufferSize: unlimited (100,000 frames)

PPO:

1. totalSteps: 1,000,000 frames

All of our training data are collected by having the model save their progress and statistics during training onto a CSV file. We would import the CSV files after the training is complete to generate graphs and analyze the data.

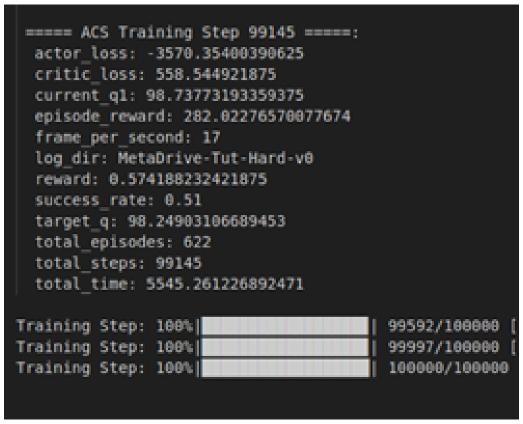
5 Experiments

5.1 DDPG-S Model

The DDPG-S model's performance was evaluated using the 'MetaDrive-Tut-Hard-v0' environment, characterized by its challenging driving scenarios. The simulations were conducted with 20 parallel environments to expedite learning and ensure robustness in the evaluation. Importantly, the scenarios were seeded to maintain consistency across runs, ensuring that any variance observed was attributable to the model rather than environmental factors.

Prior to optimization, the DDPG-S model underwent a "warm-up" phase consisting of 10,000 frames, allowing the agent to interact with the environment without immediate pressure to optimize performance. This warm-up serves as a period for the agent to accumulate initial experience.

The data were collected over a series of episodes, with each episode representing a complete run from start to finish within the simulation. Key metrics including actor loss, critic loss, episode reward, success rate, total episodes, and total time steps are shown below:



```
===== ACS Training Step 99145 =====:
  actor_loss: -3570.35400390625
  critic_loss: 558.544921875
  current_q1: 98.73773193359375
  episode_reward: 282.02276570077674
  frame_per_second: 17
  log_dir: MetaDrive-Tut-Hard-v0
  reward: 0.574188232421875
  success_rate: 0.51
  target_q: 98.24903106689453
  total_episodes: 622
  total_steps: 99145
  total_time: 5545.261226892471

Training Step: 100%|████████████████| 99592/100000 [
Training Step: 100%|████████████████| 99997/100000 [
Training Step: 100%|████████████████| 100000/100000
```

Figure 1: Result of ACS algorithm

The output indicates that after 99,145 total steps, the ACS model achieved a success rate of 51% with an episode reward averaging around 282.

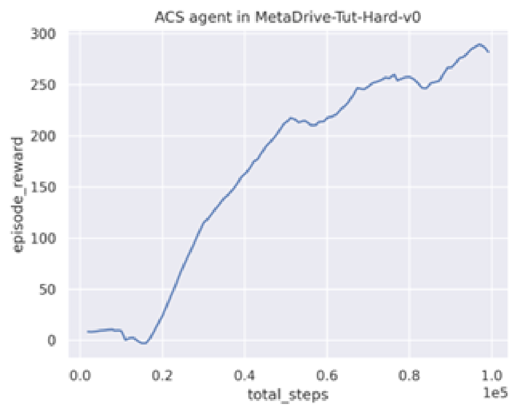


Figure 2: Episode reward of the DDPG-S(ACS) algorithm

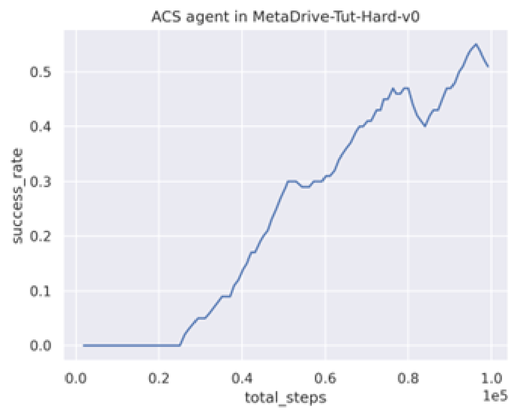


Figure 3: Success rate of the DDPG-S(ACS) algorithm

The 'Success Rate' graph illustrates the DDPG-S model's performance trajectory. A steady increase in success rate is observed, indicating the model's improving proficiency in the simulated environment. A success rate of 51% indicates the model has learned to handle the simulations' challenges effectively.

The 'Episode Reward' graph showcases the reward gained by the agent per episode over the course of training. A clear upward trend is visible, with the episode reward increasing as the agent learns from its environment, reflecting the model's capacity to maximize the cumulative reward.

5.2 Comparative Analysis

This comparative experiment aimed to evaluate the performance of the newly developed Actor Critic with Summarization (DDPG-S) model against the original Actor Critic (AC) model and two advanced reinforcement learning algorithms, TD3 and PPO.

This involved similar process of running the DDPG-S, AC, TD3, and PPO models in parallel across seeded environments to ensure consistency and eliminate variance due to environmental factors.

The collected data encompassed both the success rates and the episode rewards of the models, tracking their performance across cumulative steps to assess and compare their learning trajectories over time. The success rate was defined by the proportion of successfully completed driving tasks, while the episode reward represented the cumulative reward obtained per episode. The comparative graph are shown below:

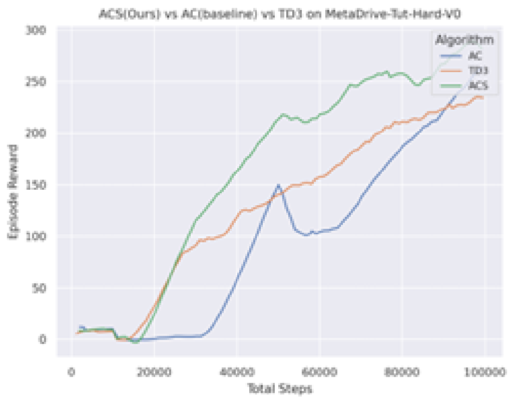


Figure 4: Comparative episode reward

Initially, all models showed similar reward patterns, but as training progresses, the DDPG-S model started to surpass both the original AC and TD3 in terms of episode rewards early in the training process. This indicated that the DDPG-S model started to leverage its experiences more effectively, optimizing the policy to garner higher rewards as it encountered various driving scenarios within the MetaDrive environment. These visual trends suggested that the ACS model, with its integration of CREST data summarization method, provided a better performance over traditional methods given the same amount of training data.

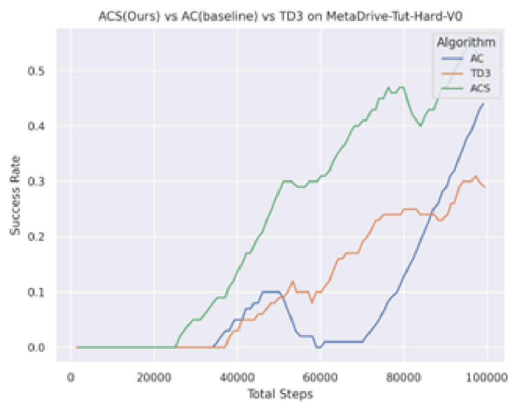


Figure 5: Comparative success rate

The comparative success rate graph showed that the DDPG-S model, represented by the green line, outperformed the original AC model (blue line) and TD3 (orange line) in terms of success rate over the total steps taken. The DDPG-S model demonstrated a steeper increase and higher overall success rate, indicating its enhanced learning capability.

The table below showcases the final success rate and episode reward for four reinforcement learning models: TD3, PPO, DDPG (denoted as AC), and DDPG-S. It also lists the number of samples used by each model during training.

	TD3	PPO	DDPG	DDPG-S
Final Success Rate	0.29	0.3	0.44	**0.51**
Final Episode Reward	234.17	219.79	263.46	**282.02**
Samples Used	**100,000**	1,020,000	**100,000**	**100,000**

The DDPG-S model achieved a high success rate of 0.51, which is higher than DDPG at 0.44, and surpassed both TD3 and PPO, which were at 0.29 and 0.3, respectively. In terms of rewards, DDPG-S outperforms the the rest at 280 points. Notably, PPO used ten times more samples than DDPG-S and DDPG, indicating DDPG-S's data efficiency. On another note, the baseline DDPG is highly unstable, as seen in the graph, but our model has shown improved stability. This consistency, in combination with its performance, demonstrates the model's robustness and reliability during training.

6 Conclusion

After iterating on tuning hyperparameters, we have found that, core-set selection does allow DDPG to attain higher sample efficiency, as it would attain better results given the same amount of data. This indicates that core set selection can bring improvement to RL algorithms to lessen its dependency on sampling data from the environment, as well as providing robustness during training. Differing from other methods such as Model-Based Reinforcement Learning, this aims directly at improving the training of the decision circuit, and our attempt at doing so was a success.

On the other hand, this method relies heavily on hyperparameter tuning, and its Core Set Selection method leaves more to be desired, compared to the highly paralleled and optimized machine learning pipelines. In fact, this had led to limits to our hyperparameter search. It defeats the purpose if the cost of computing core-sets outweights sampling more frames from environment and simply train a few epochs more.

We hope, that with our work as a basis, we can bring data summarization into other RL algorithms to have them train faster and better. There are also more research to be done with handling sequential (non-iid) training data with data summarizing methods. Overall, this work is an indication of opportunity, as this adaptation of CREST can be improved, and it can be brought to other SOTA RL algorithms to improve their sample efficiency as well.

7 Contributions

Yuyang (Adam) Chen proposed the idea for this project, and led the effort in developing the program. He also drafted methods, introduction, conclusion, and ablation study of this paper.

Wenxuan Han's contributions were crucial in contextualizing our research and articulating our results. Wenxuan Han mainly focused on developing the Related Work, Experiments, and Reference sections, analyzing data, interpreting graphs, and conducting comparative analyses. Additionally, Wenxuan Han supported the testing phase, ensuring the validity of our results.

References

[1] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: `1802.09477 [cs.AI]`.

[2] Vijay Konda and John Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

[3] Quanyi Li et al. *MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning*. 2022. arXiv: `2109.12674 [cs.LG]`.

[4] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)*. 2016. arXiv: `1509.02971 [cs.LG]`.

[5] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: `1312.5602 [cs.LG]`.

[6] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: `1707.06347 [cs.LG]`.

[7] Yu Yang, Hao Kang, and Baharan Mirzasoleiman. *Towards Sustainable Learning: Coresets for Data-efficient Deep Learning*. 2023. arXiv: `2306.01244 [cs.LG]`.

A Appendix 1

Abalation Study

Our optimized setting is as such:

1. totalSteps: 100,000 frames
2. coresetUpdateInterval: 2500 frames
3. numRandomSet: 15 sets
4. randomSetSize: 2000 frames
5. coresetSize: 64 frames(items)
6. replayBufferSize: unlimited (100,000 frames)
7. Final Score: 282
8. qualitative description: Stable after multiple tests, very time consuming.

We have a couple of tests that we kept a scrapbook record on

1. Reducing replayBufferSize:
   (a) totalSteps: 100,000 frames
   (b) coresetUpdateInterval: 2500 frames
   (c) numRandomSet: 15 sets
   (d) randomSetSize: 2000 frames
   (e) coresetSize: 64 frames(items)
   (f) replayBufferSize: 50,000 frames
   (g) Final Score: 260
   (h) qualitative description: Plateaued at score 260, seems like it dropped critical data that allows it to further improve
2. Reducing coresetSize
   (a) totalSteps: 100,000 frames
   (b) coresetUpdateInterval: 2500 frames
   (c) numRandomSet: 15 sets
   (d) randomSetSize: 2000 frames
   (e) coresetSize: 32 frames(items)
   (f) replayBufferSize: unlimited (100,000 frames)
   (g) Final Score: 160
   (h) qualitative description: Score dropped at 80% sharply as well as success rate. It could not recover after that.
3. Reducing coresetSize
   (a) totalSteps: 100,000 frames
   (b) coresetUpdateInterval: 2500 frames
   (c) numRandomSet: 15 sets
   (d) randomSetSize: 2000 frames
   (e) coresetSize: 32 frames(items)
   (f) replayBufferSize: unlimited (100,000 frames)
   (g) Final Score: 160
   (h) qualitative description: Score sharplydropped at 80,000 frames as well as success rate. It could not recover.
4. Reducing numRandomSet
   (a) totalSteps: 100,000 frames
   (b) coresetUpdateInterval: 2500 frames
   (c) numRandomSet: 5 sets

(d) randomSetSize: 2000 frames
(e) coresetSize: 64 frames(items)
(f) replayBufferSize: unlimited (100,000 frames)
(g) Final Score: 165
(h) qualitative description: Score sharply dropped at 75,000 frames as well as success rate. It could not recover.

From all of these observations, we can conclude that dataset summarization works better if we allow more data to be fed to the model, at the cost of computation times.