# Reinforcement Learning Using Difference-of-Value Functions

**Dilara Aykanat**
dilara_aykanat@berkeley.edu

**Jules Dedieu**
jules_dedieu@berkeley.edu

## 1 Extended Abstract

Many reinforcement learning methods rely on accurately approximating a state-value function $V(s)$ or an action-value function (Q-function) $Q(s, a)$. However, these functions have a large dynamic range, making them hard to approximate, leading to estimators with relatively high variances. Furthermore, most of the RL algorithms only depend on the difference in value between nearby states or actions, not the absolute value magnitude.

This led Bertsekas[1] to propose to learn a `reward-to-go` difference function $G(s_1, s_2)$ whose goal is to estimate the difference in returns between any pair of states $(s_1, s_2)$. Such function allows to derive value-based method and a simple policy update for policy based algorithm when the state space is not significantly big as suggested in [1]:

$$\pi'(s_t) = arg\max_{a_t} \left(r(s, a) - r(s, \pi(s))\right) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), s'' \sim p(s''|s,\pi(s))} \left[G(s', s'')\right] \tag{1}$$

In this work, we tried to implement and generalize this idea in a more complex environment with infinite state-spaces, trying to use the reward-to-go difference function to improve several usual methods: actor critic algorithm, value-function methods and finally policy gradient methods, using proximal policy iteration. In actor critic, we started by exploring the effect of taking the difference between two batches. We used CartPole as an example of a simple environment, with a very limited action space, and both Half Cheetah and Lunar Lander as examples of more complex environments. We then continued by implementing Equation 1 in a Proximal Policy Optimization algorithm.

As expected,some of the difference-of-value functions based algorithm we implemented performed better than the baseline methods in simpler environments where states have significantly different value functions, especially actor-critic and policy based methods. On the other hand, our algorithms based on the difference of value function method appear less stable on complex environments. Moreover, the need to collect more data at training time significantly slows and constraint the training process. To account for that, we decreased the batch size in our first actor critic trials. The reported results are the mean of 5 different runs with different seeds each.

In this project, Dilara Aykanat primarily worked on initial trials with batch differences and Jules Dedieu worked on Proximal Policy Iteration and Value Function Methods. The GitHub links are respectively: https://github.com/dilaraaykanatt/CS285_Project_ and https://github.com/JulesFD/Cs285Qvalue.

## 2 Problem Statement

### 2.1 Original Idea

Many reinforcement learning problems rely on finding a satisfactory way to estimate a satisfactory q-function or value-function at a given state. Given the large dynamic range of these functions, they can sometimes be hard to approximate, which increases the variance of the final reinforcement learning predictions.

In [1] Bertsekas suggesting to address this issue by propose learning instead a `reward-to-go` difference function $G(s_1, s_2)$ whose goal is to estimate the difference in returns between any pair of states $(s_1, s_2)$. Hence: the difference of value function $G(s_1, s_2) = V(s_1) - V(s_2)$.

This function allows us to derive a natural policy update equation, from a fixed policy $\pi$:

$$\pi'(s_t) = arg\max_{a_t} \left(r(s,a) - r(s,\pi(s))\right) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), s'' \sim p(s''|s,\pi(s))} \left[G(s', s'')\right]$$

### 2.2 Initial trials with batch differences in Actor Critic

In this part, our main goal was to investigate the effect of using the difference of observations and advantages between batches as a warm-up for difference-of-value functions. We used HW3 as our starter code and conducted the experiments on Cartpole, HalfCheetah and InvertedPendulum environments.

In order to calculate the difference of value functions in an Actor Critic algorithm, we sampled a secondary batch $B_2$ as compared to the original one that we will refer to as $B_1$. Note that both batches include trajectories of different lengths, so they have separate terminal states which should be accounted for during the training.

#### 2.2.1 Advantage Difference

We first investigated the effect of using the advantage difference between pairs of states so that the estimated advantage becomes $A^\pi(s_{1,t}, a_{1,t}, s_{2,t}, a_{2,t}) \approx r(s_{1,t}, a_{1,t}) - r(s_{2,t}, a_{2,t}) + \gamma V^\pi_\phi(s_{1,t+1}) - V^\pi_\phi(s_{1,t}) - \gamma V^\pi_\phi(s_{2,t+1}) + V^\pi_\phi(s_{2,t})$ (`advantage_difference`).

#### 2.2.2 Observation Difference

Another method we tried was to take the difference between the two batch of state observations while estimating the advantage so that the estimated advantage becomes $A^\pi(s_{1,t}, a_{1,t}, s_{2,t}, a_{2,t}) \approx r(s_{1,t} - s_{2,t}, a_{1,t} - a_{2,t}) + \gamma V^\pi_\phi(s_{1,t+1} - s_{2,t+1}) - V^\pi_\phi(s_{1,t} - s_{2,t}))$.

We also updated the critic and the actor over both $B_1$ and $B_2$ which of course resulted in a longer training period since collecting samples is also a time-consuming step. It is important to note that taking the difference of observations inherently assumes that states with large differences are going to result in different rewards which is definitely not the case in more complex RL problems. However in the simple Cartpole environment, taking the difference of observations gave the best results while not increasing the training period significantly since it reached the maximum reward in a few iterations(`observation_difference_double_update`). We discuss the results in Figure 1 in Section 4 in more detail.

Additionally, we tried to decrease the batch size by half to compensate for the longer sample collection times(`observation_difference_double_update_batch500`). Another reason behind the good performance of the observation difference in Cartpole could be the double update of the actor and the critic over two batches, hence we included that case where no difference of the observations was taken but double update was still performed (`double_update_only`).

In HalfCheetah, on the other hand, the learning task is much harder, there are now 17 dimensions instead of 4 of the Cartpole and the action space is continuous as opposed to +1/-1 discrete space of the Cartpole. Since two batches are collected instead of one to take the difference, batch size was decreased to 20000 to attain the same amount of training time with the standard actor critic method.

InvertedPendulum is the continuous version of the Cartpole environment. Just like in Cartpole, it is easier to maintain a high-reward status once the maximum is reached.

### 2.3 Proximal Policy Iteration

After our initial trials based on batch differences detailed in Subsection 2.2, we decided to work on a more natural application of Bertsekas' idea. In order to increase the tractability of the formulation given in Equation 2.1 in more realistic contexts, where the state space is very large, we transformed this algorithm into an actor critic paradigm. This consideration led us to base our experiments on the Proximal Policy Optimization algorithm, that allows to improve a policy by taking the gradients at each step with respect to an 'old' policy; in our case the policy at the previous time step.

However, this algorithm usually uses a critic that aims at estimating an advantage function, in a similar way than regular actor-critic approaches. In order to implement our method, we implemented instead a *differential Q function estimator:* $G_Q$. Hence the expression above becomes the following:

$$\pi(s) = \text{argmax}_a G_Q(s, a, s, \pi_{old}(s)) = \text{argmax}_a Q(s, a) - Q(s, \pi_{old}(s))$$

In order to learn $G_Q$ we used a Q-iteration paradigm. To use this estimator in the expression above, it would be enough to get the value of $G_Q$ for each state s and each pair of action $(a_1, a_2)$: $G_Q(s, a_1, a_2) = Q(s, a_1) - Q(s, a_2)$. However despite being easier and more stable to learn, this approach becomes complex to apply in real-life context with continuous state spaces, since it is unlikely our simulator can reach several time a given state and try out different actions in this given state.

Hence, we used instead the following estimator for two pair of states and actions: $((s_1, a_1), (s_2, a_2))$: $G_Q(s_1, a_1, s_2, a_2) = Q(s_1, a_1) - Q(s_2, a_2)$

This allows for a more generalizable implementation: we first sample two couples of states, actions and rewards from a replay buffer: $((s_1, a_1, r1, s'_1), (s_2, a_2, r_2, s'_2))$, then we use the following value-iteration loss, for a given Risk function R (e.g: MSE, L1 norm...):

$$L(G_Q) = \mathbb{E}\left[R(r_1 - r_2 + \max_{a'_1, a'_2} G_Q(s'_1, a'_1, s'_2, a'_2)), G_Q(s_1, a_1, s_2, a_2)\right]$$

This expression also suggests that the tricks usually used for Q-value iteration algorithm can be used for the training of the differential Q function estimator such as double Q learning, using another network $G''_Q$ in the left hand side in the risk term, to avoid biasing $G_Q$

Hence, we finally use the following actor-critic algorithm: Differential Proximal Policy Iteration (DPPO):

### 2.4 Value Function Methods

In the previous implementation, we aim at learning a differential Q-function $G_Q$, in a way that is very similar to value function methods. We may then wonder the feasibility of using value-function based methods rather than policy based.

The fact that we learn differential Q function does compromise the direct use of an argmax policy. Indeed given a state s we first compute $\tilde{G}_Q(s_1, s_2)G_Q(s, a_1, s, a_2)$ for all $a_1$, $a_2$. Then several options are possible:

- Choosing the action that maximizes $\tilde{G}_Q(s_1, s_2)$. Indeed such a choice is possible since G is anti symmetric. However, this may be biased if the action $a_2$ performs extremely poorly.
- Choosing the action $s_1$ that maximizes the weighted average over all other actions: $\sum_{s_2 \in actions} \lambda_i \tilde{G}_Q(s_1, s_2)$

This approach presents however little theoretical advantage compared to classic value-function methods in most environment.

**Algorithm 1** Differential Proximal Policy Iteration

**Initialize:** Policy parameters: $\theta$, $G_Q$ parameters: $\phi$
Define hyperparameters values including total number of timesteps, and number of policy updates per iteration, number of $G_Q$ updates per iterations.
**for** i = 0 to total number of timesteps. **do**
$\pi_{\theta_{old}} = \pi_\theta$
Fill the replay buffer from trajectories using $\pi_{\theta_{old}}$
Sample $((s_1, a_1, r1, s'_1), (s_2, a_2, r_2, s'_2))$ from the replay buffer
**for** j = 0 to number of $G_Q$ updates per iterations. **do**
**Train** $G_Q$ using:

$$L(G_Q) = \mathbb{E}\left[R(r_1 - r_2 + \max_{a'_1, a'_2} G_Q(s'_1, a'_1, s'_2, a'_2)), G_Q(s_1, a_1, s_2, a_2)\right]$$

**Hence**$G_Q$ parameters $\phi$ gets updated
**end for**
**for** j = 0 to number of policy updates per iterations. **do**
**Compute** the improved policy:

$$\pi(s) = \text{argmax}_a G_Q(s, a, s, \pi_{\theta_{old}}(s))$$

**Compute** the importance ratio: $r(\theta) = \text{logprob } \pi_{\theta_{old}} - \text{logprob } \pi_\theta$
**Compute** the objective function:

$$J^{Clipped}(\theta) = \mathbb{E}\left[\min(r(\theta)A^{\pi_{old}}), clip(r(\theta)A^{\pi_{old}}, 1-\epsilon, 1+\epsilon)\right]$$

**Minimize** $J^{Clipped}(\theta)$ with respect to $\theta$
**Hence** policy $\pi$ parameters: $\theta$ get updated
**end for**
**end for**

## 3 Related Work

Bertsekas [1] first proposed the idea of a using differences of value function to solve reinforcement learning models : [1].

This idea was further suggested in OpenAI ideas for research: [2].

To the best of our knowledge, we did not find research papers investigating specifically the idea of differential training. This approach has however been explored in two GitHub repositories: [3] and [4]. Both implement value-function based differential training, aiming to learn functions very similar to the $G_Q$ functions defined above. [4] also uses a policy update algorithm close to the one suggested by Bertsekas in [1]

However, learning differences of functions between states in an omnipresent idea in several reinforcement learning and has been applied to a wide range of problematic such as learning state differences in model-based RL: [5]. Approaches based on temporal differences have also been used in order to estimate and reduce the variance of the rewards-to-go: [6] and []

To the best of our knowledge, the use of proximal policy gradient, advantage difference and observations differences to address policy update in differential training has not been researched yet.

## 4 Results

### 4.1 Differences Algorithm Results

Figure 1 compares all of the scenarios tried on the Cartpole environment. One can see that double update only is not enough to maximize the reward without advantage or observation difference.

As mentioned in Subsubsection 2.2.2, `observation_difference_double_update` performed the best by reaching the maximum reward earlier than the other scenarios.

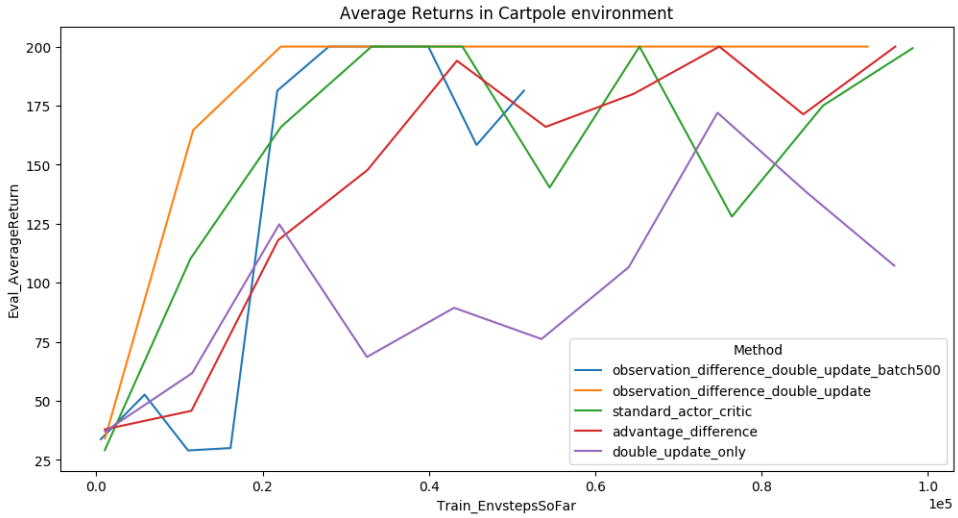


Figure 1: Average Return of standard Actor-Critic (green), and with difference-of-value functions over learning iterations in Cartpole

Figure 2 compares the evaluation of the average returns in HalfCheetah However, we ran out of memory on our local computer while training with double batches so the difference-of-value scenarios stopped earlier than expected. On the other hand, the green line which depicts the advantage difference scenario reached higher rewards in fewer iterations (or more quickly) as compared to the standard actor critic method.

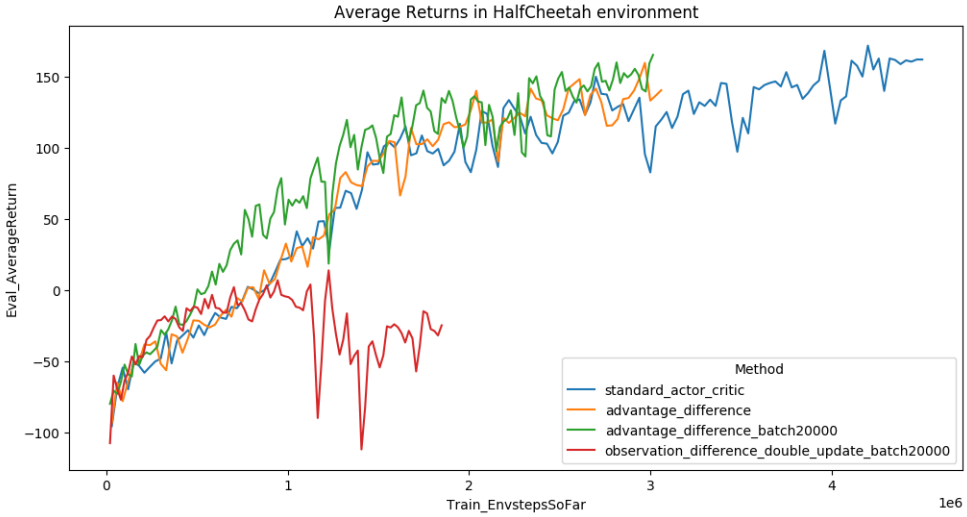


Figure 2: Average Return of standard Actor-Critic (blue), and with difference-of-value functions over learning iterations in HalfCheetah

### 4.2 Differential Q-function based algorithms results

We also compared both of the differential Q-function based algorithm results in both easy and more complex environment in order to study the performance of both the differential Q-value based algorithm, and Differential Policy Iteration.

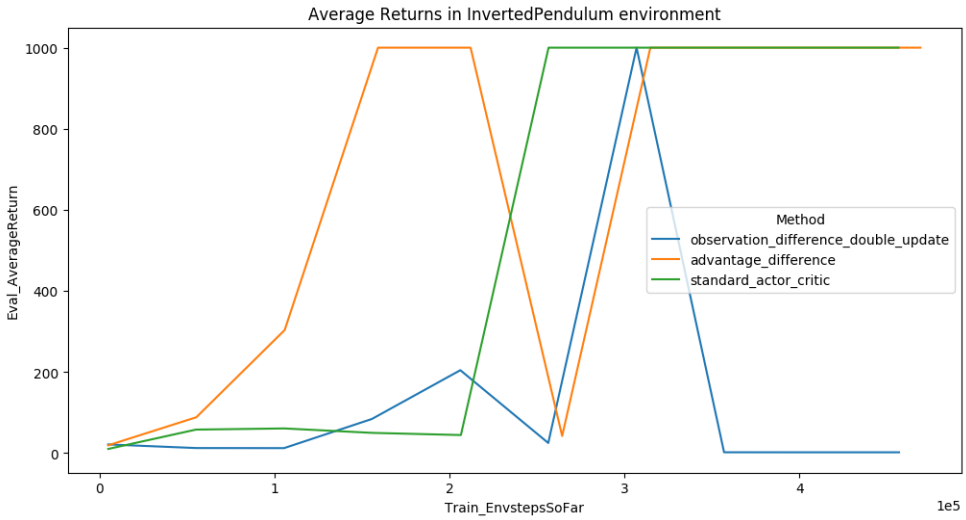


Figure 3: Average Return of standard Actor-Critic (blue), and with difference-of-value functions over learning iterations in InvertedPendulum

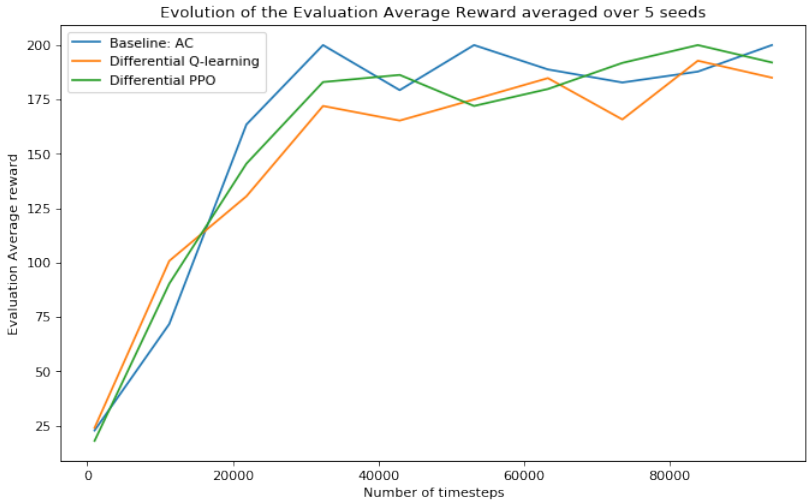


Figure 4: Evolution of the Evaluation Average Reward averaged over 5 seeds, on Cartpole environment, for regular actor critic, and differential Q-learning and proximal-policy iteration

Hence on Cartpole, which is an easy environment with a limited state-space, our methods performance is comparable to the one of a regular actor critic algorithm.

In order to get an estimation of the variability of each algorithm's training and to reduce the importance of the general variability of trainings in that environment, we simulated each training for 5 different seeds.

We notice that both of the algorithm need a slightly longer time to train than the baseline, given the fact that the differential Q function they aim at learning is more complicated

We notice that the Q-value based algorithm has a slightly poorer performance. It is also more unstable within each training. This result was expected since the differential Q-value based method seems less interesting theoretically than regular Q-value based method.

On the other hand the differential Proximal Policy Iteration sounds to be able to achieve very similar long-run performance to the baseline, with slightly less variability in the convergence.

| Algorithm used | Iteration at which reward 0 is reached | Reward at 150 iterations |
|---|---|---|
| Baseline Actor-Critic | 27 ± 6 | 137± 21 |
| Differential Q-Learning | 48 ± 14 | 85 ± 33 |
| Differential PPO | 36 ± 9 | 96 ± 28 |

However in more complex environment the differential value based method seem to struggle much more converging. On HalfCheetah, we got the following table that summarizes our training, with mean and standard deviation over 3 runs.

We found that in this complex environment, with very high dimensional state and action space, the algorithms based on the differential Q-function $G_Q$ defined earlier do not achieve a performance that is comparable with the one of the original algorithm. This is a limitation we were expecting from the original Bertsekas model, which rely on the expectation over states to be calculated easily, which does not hold in high dimensional problems.

## 5 Conclusion

In our work, we investigated the use of state-value differences in reinforcement learning algorithms, as a solution to improve the convergence and reduce the variance of policy gradient and actor critic methods.

We showed that the idea of differential training could be apply in a wide range of approaches, that we aimed at comparing on several environment.

We showed that advantage-difference methods were able to outperform baseline actor-critic on simple environments such as CartPole, but also to a lesser extent on more complex environments such as Half-Cheetah.

However approaches aiming at performing policy updates directly using the differential value function did not show satisfactory results on our environments, despite the use of Differential Proximal Policy Optimization.

## References


[1] Dimitri P. Bertsekas. Differential training of rollout policies, 1997.

[2] OpenAI. Ideas for research. https://github.com/openai/requests-for-research/blob/master/_requests_for_research/difference-of-value-functions.html.

[3] Nikhil Dev. Differential training. https://github.com/nikhil-dev/differential-training.

[4] Rishabh Agarwal Yogesh Kumar, Goutham Ramakrishnan. Differential training for rollout policies. https://github.com/Yogeshkumar4/Differential-Training-Of-Rollout-Policies.

[5] Ronald S. Fearing Sergey Levine Anusha Nagabandi, Gregory Kahn. Neural network dynamicsfor model-based deep reinforcement learning with model-free fine-tuning.

[6] Shie Mannor Aviv Tamar, Dotan Di Castro. Temporal difference methods for the variance of the reward to go. http://proceedings.mlr.press/v28/tamar13.html.