Differential Training for Rollout Policies

Yogesh Kumar[*]
140050004
IIT Bombay
`yogesheth@cse.iitb.ac.in`

Goutham Ramakrishnan[*]
140020039
IIT Bombay
`140020039@iitb.ac.in`

Rishabh Agarwal[*]
140050019
IIT Bombay
`rishabhagarwal@cse.iitb.ac.in`

Abstract

Most of the fundamental reinforcement learning algorithms compute an estimate of the optimal value function to obtain a optimal control policy. Estimating the optimal value function is a computationally expensive task and the existing methods are not robust to approximation and simulation errors. Bertsekas proposed a method called differential training, which estimates the difference in values of states rather than the values of states. We propose a novel Deep Differential Q-Network based on differential training using neural networks. We tested the efficacy of his hypothesis on GridWorld, CartPole and MountainCar environments, to obtain insights on its advantages and shortcomings.

1 Introduction

The goal of reinforcement learning[1] is to learn good policies for sequential decision problems, by optimizing a cumulative future reward signal. Q-learning[2] is a form of model-free reinforcement learning. It can also be viewed as a method of asynchronous dynamic programming (DP). However, computing approximations of value functions using such methods is prone to approximation and simulation error.

Bertsekas[3] argued that learning the action-value function or the value function directly using function approximation for computing an optimal policy is quite sensitive to simulation and approximation error, and a more robust alternative is to estimate Q-factor differences and value function differences. For example, when we use the approximate Q-factor to obtain the optimal action $\hat{\pi}(s)$ for a given state s, using the equation

$$\hat{\pi}(s) = argmax_{a \in A(s)} \quad Q(s,a)$$

where A(s) is the action space for state s.
However, there is a serious flaw with this approach, due to the simulation error involved in the calculation of the Q-factors. In particular, for the calculation of $\hat{\pi}(s)$ to be accurate, the Q-factor differences

$$Q(s,a) - Q(s',a) = \mathbb{E}[R(s,a,s'') - R(s,a,s''') + V]$$

must be computed accurately for all pairs of actions, a and $a' \in A(s)$, so that these actions can be accurately compared. On the other hand, the simulation/approximation errors in the computation of the individual Q-factors are magnified through the preceding differencing operation. Similar problem

[*]These three authors contributed equally to the project

holds for learning the value function directly as well. Therefore approximating the differences of a value function directly can help in reducing this error.

We aim to validate Bertsekas's hypothesis by extensive experiments on toy environments. We explored the tabular and function approximation (with neural network approaches) to learn the value function differences and the Q-function differences. In doing so, we also present a novel Deep Difference Q-Network(DDQN) architecture for differential training.

We found that using differential training with tabular approximation is very inefficient and ineffective. However, for the classic control environments[4], we obtained mixed results as the DDQN method is far superior to DQN on the CartPole-v0 environment while DDQN performed extremely poorly on the MountainCar-v0 environment. We try to justify why differential training works on certain environments but fails on others in section 4. Our arguments are based on the experimental results obtained.

2 Related Work

Working with value differences to find a optimal control policy is a fairly unexplored area and is floated as a **Request for Research** on the OpenAI website. Most of the previous work involving differential training has been about solving the control problems using rollout policies, which we didn't explore due to their sample inefficiency.

3 Methodology and Contributions

3.1 Differential Training

Given a policy π,

$$\hat{\pi}(s) = \underset{a \in A(s)}{\arg\max}\, \mathbb{E}[R(s,a) - R(s,\pi(s)) + \gamma * (V(f(s,a,w)) - V(f(s,\pi(s),w)))]$$

where $f(s,a,w)$ denotes the next state reached by taking the action a in state s in presence of some random disturbance w, γ is the discounting factor (Note that γ = 1 for episodic MDPs). Here, V is the value function for the policy π.

Since we are going to approximate the value differences, denote for all time steps t, $G_t(s,\hat{s}) = V_t(s) - V_t(\hat{s})$. It's easy to derive that for all state pairs $(s,\hat{s})$ such that $s,\hat{s} \in \mathbb{S}$, $G_t(s,\hat{s})$ is the value function of a fixed policy π_t, for the state space pair $(s,\hat{s})$, the reward function given by $R(s,\pi_t(s)) - R(\hat{s},\pi_t(\hat{s}))$ and the transition function defined by $(s,\hat{s}) \rightarrow (f(s,\pi_t(s),w_t), f(\hat{s},\pi_t(\hat{s}),w_t))$.

The implication of the above statement is that any of the standard methods (such as TD(λ)) that can be used to train architectures that approximate V_t, can also be used for training architectures that approximate G_t. Therefore, given a initial policy π_0, we can use the approximate value differences and apply policy improvement to get a better policies and thereby, this approach can be used to find the optimal policy in the control setting.

The differential training algorithm was tested on the GridWorld, CartPole-v0 and MountainCar-v0 environments.

3.2 Pair MDP

For any given MDP, the corresponding pair MDP is obtained by expanding the state space to $S \times S$ and action space to $A \times A$. For a MDP $\mathbb{M} = (S,A,R,T,\gamma)$, the corresponding pair MDP is given by $\mathbb{M}' = (S',A',R',T',\gamma)$ where,

$$S' = S \times S$$
$$A' = A \times A \; and \; A(s,s') \in A(s) \times A(s')$$
$$R' : S' \times A' \rightarrow \mathbb{R} \; with \; R'((s,s'),(a,a')) = R(s,a) - R(s',a')$$
$$T'((s,s'),(a,a')) = (T(s,a'),T(s',a'))$$

3.3 GridWorld: Tabular Q-learning

3.3.1 Environment Description

The GridWorld environment from Assignment 3 was used to test the algorithm, for tabular Q-learning. The grid contains a single start and goal state, with obstacles present. There is some amount of stochasticity in the environment, in the form of slip. The action space consists of four actions: up, down, left and right. A reward of 100 is obtained upon reaching the goal state, while a reward of -1 is obtained for each step that reaches any non-goal state.

3.3.2 Policy Iteration using TD(λ)

The differential training was used to learn the differences between the values of the states. The learned value function was then used to progressively learn better policies. The policy iteration was done using the difference of action values calculated as follows:

$$Q(s,a) - Q(s,\pi(s)) = \mathbb{E}[R(s,a) - R(s,\pi(s)) + \gamma * (V(f(s,a,w)) - V(f(s,\pi(s),w)))] \quad (1)$$

Here, *w* corresponds to the slip factor in the GridWorld environment. To calculate these values, we simulated the rewards generated by sampling a given action from a given state and by taking the average value of reward generated from these simulations. The difference in the value functions were approximated using applying the $TD(\lambda)$ on the the pair MDP obtained from GridWorld. The action which maximize the difference between the Q-values for a given state is picked as the action in the that state in the improved policy.

3.3.3 Difference Q-learning

Instead of learning the value function differences, we also experimented with learning the difference of Q-functions for the optimal policy on the Pair-MDP instance. The motivation of performing this experiment was that if we are able to correctly learn the difference-Q-tables, we can directly learn the optimal policies by using the *argmax* operator as described in equation (1).

3.4 CartPole: Environment Description

A pole is attached by an unactuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity. The State space is $S = (-2.4, 2.4) \times \mathbb{R} \times (-41.8, 41.8) \times \mathbb{R}$, in the form (Cart Position, Cart Velocity, Pole Angle, Pole Tip Velocity). There are two permitted actions, push cart to the left or push cart to the right. The agent receives a reward of 1 for every step until termination. The episode terminates if the pole angle is more than ± 12, or the cart position is more than ± 2.4 or the episode length is greater than 200. To

3.5 Deep Q-network(DQN)

DQN[5] combines reinforcement learning with neural networks with the use of experience replay and target networks. By interacting with its given environment, a DQN agent generates a series of mutually dependent observations, actions, and rewards. To prevent the algorithm from diverging, DQNs use experience replay, which randomizes over the data to purge correlations in the observation sequence and smooth changes in the data distribution. The loss function used to train the network is:

$$L(\theta_i) = \mathbb{E}_{(s,a,r,s')}[(r + (\gamma) \max_{a'} Q(s', a', \theta_i) - Q(s, a, \theta_i))^2]$$

Here, γ is the discount factor, θ_i are the parameters of the Q-network at iteration i. We train the model using the Adam Optimizer with learning rate = 0.001.

A neural network with two hidden layers of size 12 and final layer of size equal to that of size of action space was used in DQN. The input to the neural network is the state in the form of a observation vector while the network is trained to predict the Q-values corresponding to that state.

3.6 Deep Difference Q-Network

To extend the idea of Differential training to DQNs, we propose a novel architecture namely, Deep Differential Q-Network. We retain the methodologies from DQN to combat instability in learning.

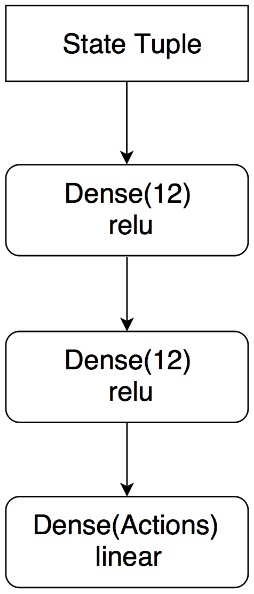


Figure 1: The Neural Network architecture used for estimating Q-values for the CartPole Agent

We change the loss function in order to enable the network to learn the differences. For each $((s, s'), (a, a'), r_1 - r_2, (s'', s'''))$, the new loss function is given as:

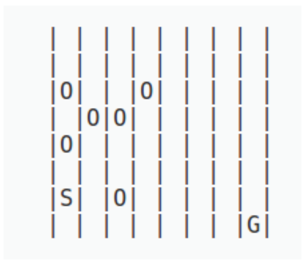
$$L'(\theta_i) = \mathbb{E}[(r_1 - r_2 + \gamma(\max_{a''} Q(s'', a'', \theta_i) - \max_{a''} Q(s''', a'', \theta_i)) - (Q(s, a, \theta_i) - Q(s', a', \theta_i)))^2]$$

4 Results

4.1 GridWorld: Tabular Q-learning

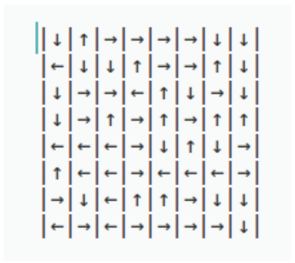
The objective of learning the differences in the action values of states requires the estimation of a modified Q-table ($S \times S \times A \times A \to \mathbb{R}$). Simulations were run on a 8x8 grid, with a start state, goal state, and obstacle states. In such a small grid, the conventional Q-learning algorithm excels as expected, with fast convergence and optimal policy learnt for small values of slip. On the other hand, the differential training algorithm struggles to learn the optimal policies. This can be attributed to a number of factors, the most prominent being the blow up in the size of the state space.

In conclusion, it is counter-productive to use the differential training algorithm for tabular Q-learning. Conventional Q-learning already yields the optimal solution. The differential training algorithm requires the learning of an exponentially larger table of action value differences, which is considerably more difficult. As a result, convergence of the Q-table takes a very large number of episodes. Thus the differential training algorithm is sub-optimal for tabular Q-learning.



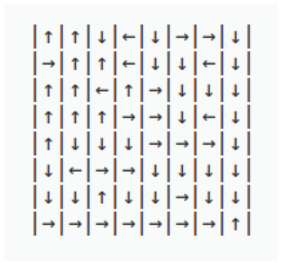
```
| | | | | | | | |
| | | | | | | | |
|0| | |0| | | | |
| |0|0| | | | | |
|0| | | | | | | |
| | | | | | | | |
|S| |0| | | | | |
| | | | | | | |G|
```

Figure 2: GridWorld instance



```
|↓|↑|→|→|→|→|↓|↓|
|←|↓|↓|↑|→|→|↑|↓|
|↓|→|→|←|↑|↓|→|↓|
|↓|→|↑|→|↑|→|↑|↑|
|←|←|←|→|↓|↑|↓|→|
|↑|←|←|→|←|←|←|→|
|→|↓|←|↑|↑|→|↓|↓|
|←|→|←|→|→|→|→|↓|
```

Figure 3: π^* by DQ-learning



```
|↑|↑|↓|←|↓|→|→|↓|
|→|↑|↑|←|↓|↓|←|↓|
|↑|↑|←|↑|→|↓|↓|↓|
|↑|↑|↑|→|→|↓|←|↓|
|↑|↓|↓|↓|→|→|→|↓|
|↓|←|→|→|↓|↓|↓|↓|
|↓|↓|↑|↓|↓|→|↓|↓|
|→|→|→|→|→|→|→|↑|
```

Figure 4: π^* by Q-learning

Figure 2 shows the instance of the GridWorld environment (slip value used was 0.3) where the optimal policies learnt by Q-Learning and Difference Q-Learning after 10000 training episodes is shown by figures 3 and 4 respectively. As it can be seen, the optimal policy learnt by Difference Q-Learning is poor as compared to that of learnt by Q-Learning.

4.2 CartPole: DDQN vs DQN

The DDQN outperforms the DQN in the cartpole environment as it can be seen in the graph. Though both algorithms eventually learn optimal policies, the DDQN shows faster convergence. This is exactly as hypothesized by Bertsekas who proved in his paper that differential training would reduce the estimation error variance, leading to faster convergence. The algorithm thrives in this environment, as it quickly learns to isolate trajectories which lead to higher rewards.

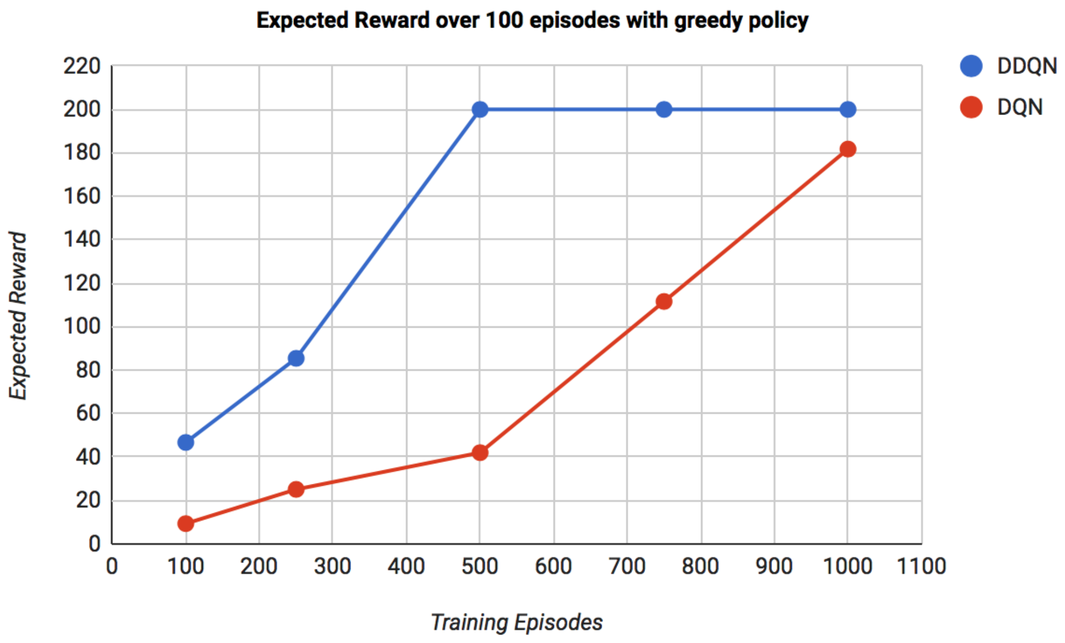


Figure 5: Expected Average Reward on 500 test episodes after a training for a varying number of episodes for the DQN and DDQN

4.3 MountainCar: Deep Difference Q-Network

We experimented with various Neural Network architectures applied directly on the original state space, however, even after running a large number of episodes(9000), it was not able to solve the environment. Since tile coding is known to perform well in this environment[6], we also ran experiments after performing tile coding on the state space which turned out to be ineffective. We attribute this failure to the fact that the difference of rewards is always zero for the pair MDP created using this environment. We also tried to modify the reward structure of the pair MDP for the state pairs containing a terminal state, but the DDQN framework wasn't able to solve the MountainCar Environment.

5 Conclusions

We introduce a novel Deep Difference Q-Network architecture to learn the Q function differences. We ran extensive experiments on Gridworld, Cartpole and MountainCar environments. To the best of our knowledge, this is the first attempt to experimentally validate the differential training algorithm without using rollout policies. Our evaluations provide an insight into the types of environments where differential training may be used to good effect. Our experiments conclusively demonstrate that using this algorithm with tabular Q-learning is suboptimal and counter-productive. However, using DDQN on classical control environments may be beneficial. Furthermore, we hypothesize that the algorithm would perform well in MDPs where the reward structure is well differentiated between distinct trajectories.

6 Future Work

Future Work would involve trying to deal with the problem of reward structure in the DDQN framework. Also, implementing this framework on more complex reinforcement learning environments (such as the OpenAI Mujoco environments) is an interesting arena to explore.

References

[1] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

[2] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

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

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] Shimon Whiteson, Matthew E Taylor, Peter Stone, et al. *Adaptive tile coding for value function approximation*. Computer Science Department, University of Texas at Austin, 2007.