Extensions on State Value Difference-Based Learning

Youkow Homma [1]

Abstract

Value-based reinforcement learning algorithms generally approximate the value of each individual state. In the tabular setting, doing so leads to strict improvement in policy; however, such results are not guaranteed in the function approximation setting. In fact, approximating the state value can at times result in a degradation of the policy. To address this problem, Bertsekas proposed a new objective of minimizing the sum of squares loss between differences in values of states. He furthermore developed differential training as a solution towards this objective. Webster and Baxter developed a separate algorithm called State Temporal Difference (STD) learning to minimize a similar value difference objective for binary MDP's. In this paper, we generalize STD learning to broader environments by marrying the ideas of Bertsekas' differential training with Webster and Baxter's STD learning. We first extend STD to general reward functions on binary MDP's. We then propose a further extension to general MDP's.

Given a policy, we can evaluate the policy (find the value at each state under the policy) in a number of ways. In the tabular setting with known dynamics, we can solve a system of linear equations or iterate Bellman backups to converge to the true value function of the observed policy. The policy improvement theorem then tells us that the policy derived from acting greedily on the new value function is at least as good as the observed policy.

For larger state spaces, another approach to policy evaluation is to sample trajectories under a fixed policy and perform temporal difference (TD) updates on approximations of the state values. Due to a result from Tsitsiklis and Van Roy, (Tsitsiklis & Van Roy, 1997), when our value function is represented by a linear function approximator, this is equivalent to minimizing the weighted mean squared value error, which is defined as

$$E\left[\frac{1}{2}(\tilde{V}(s_t) - V(s_t))^2\right],$$

where $\tilde{V}$ is the state value function approximation and V is the true state value function.

While function approximation helps us tackle large MDP's, one tradeoff is that the policy improvement theorem no longer holds in this setting. In fact, Weaver and Baxter (Weaver & Baxter, 2001) presented examples of when performing TD learning led to a worse policy when approximating the value function. They demonstrated this phenomenon in both toy problems with linear function approximators and backgammon with neural networks. Bertsekas and Tsitsiklis (Bertsekas & Tsitsiklis, 1996) also showed this in Tetris. In the case of backgammon, Weaver and Baxter showed empirically that in some cases, TD learning minimizes the Bellman loss while simultaneously decreasing the value acheived by the policy.

In order to address this problem, Bertsekas (Bertsekas, 1998) proposed a new algorithm called Differential Training in which two rollouts running in lock-step are used to make TD-style updates to the difference in values of states. The primary advantage of this algorithm is that there are no additional assumptions necessary. The downside is that differences may be considered for pairs of states which have no applicability to the policy. Furthermore, considering all pairs of states increases the runtime from linear to a quadratic in the number of states. We will elaborate on this idea further in the paper.

Weaver and Baxter proposed two different solutions to the problem in their papers (Weaver & Baxter, 1999) and (Weaver & Baxter, 2001). The papers present two versions of what is called State Temporal Difference learning. The algorithm in the paper from 1999 is a slightly more general, but weaker result of the ideas in the paper from 2001. In this paper, we will focus on the latter, more recent algorithm, which we will refer to as STD.

The idea of looking at value differences has been studied more extensively in the control setting. In 1994, Harmon, Baird and Klopf (Harmon et al., 1994) proposed the technique of advantage updating in order to decrease the variance of rollout policies resulting from simulation error. In

[1]Stanford University, Stanford, California. Correspondence to: Youkow Homma <yhomma@stanford.edu>.

this method, an advantage function, the difference between the action values of an optimal action and action a,

$$A(s,a) := \max_{a'} Q(s,a') - Q(s,a),$$

is trained and maintained along with the outright Q-values. Directly training on these values changes the optimization objective to distinguishing between values of different actions and reduces the variance that can result from estimating Q-values for different actions independently.

Baird (Baird, 1999) simplified advantage updating to advantage learning by only storing the advantages, rather than the state-action values and the advantage function. This newer technique has recently garnered interest. In particular, Bellemare et al. (Bellemare et al., 2015) introduced a generalization of advantage learning and showed that a broader set of operators encompassing advantage learning could be used to decrease variance resulting from simluation error. In this paper, we look to the more mature ideas of advantage learning for inspiration in extending value difference-based learning in the policy evaluation setting.

The outline for the remainder of this paper is as follows. In section 1, we establish the notation that will be used for the remainder of the paper. In section 2, we highlight the key points of differential training and STD. In section 3, we present our first generalization of STD from deterministic rewards that are functions of the outbound state to all stochastic reward functions. In section 4, we present our second generalization of STD from binary MDP's to general MDP's. Section 5 will outline the results of running the two new algorithms in randomized toy problems. Finally sections 6 and 7 will be the discussion and conclusion of the paper, respectively.

1. Preliminaries

In this paper, we will be considering Markov decision processes (S, A, P, R, γ), where S is the state space, A is the action space, P is the transition probability, R is the reward function and γ is the discount factor.

For any policy π on the MDP, we let V^π denote the state value function of that policy. If a stationary distribution exists for policy π, then we will denote the stationary probability of being in state s as $\psi(s)$. We will denote our function approximator of the value function as V_w, where w is the tuned weight. Furthermore, in the linear function approximation case, we will let $\tilde{V}_w = w^T\phi$, where w is the weight of the approximator and ϕ is the feature vector mapping states to features.

Throughout this paper, we will be considering states that are not part of the trajectory directly observed by the agent. To distinguish attributes between the main trajectory and others, we will use the superscript $'$ to denote the secondary attributes. The"un-primed" trajectories $s_1, s_2, \ldots$, will always denote the trajectory observed by the agent. The definition of the "primed" trajectories, $s'_1, s'_2, \ldots$ is overloaded and will change based on the context.

Finally, borrowing from and generalizing the definition in (Weaver & Baxter, 2001), if we consider a state s, from which there is a set of states $\{s_i\}$ that can be transitioned to in the next time step with positive probability, then we call the states $\{s_i\}$ *sibling states*. Also from (Weaver & Baxter, 2001), we define Binary Markov Decision Processes (BMDP) as MDP's such that from any state, there are at most two states to which the agent can transition to in the next time step with positive probability. In BMDP's, each state has at most one sibling state.

2. Previous Work

In this section, we describe in detail the Differential Training algorithm by Bertsekas (Bertsekas, 1998) and the STD algorithm by Webster and Baxter (Weaver & Baxter, 2001). For each algorithm, we describe the algorithm itself, the working assumptions, the pros and the cons of the approach.

The first algorithm we describe is Bertsekas' Differential Training. The algorithm adopts the standard temporal difference framework with the only difference being that the MDP is over two parallel environments running in lockstep. That is, given a sample trajectory, (s_t, s_{t+1}) from one environment and trajectory (s'_t, s'_{t+1}) from the other environment, the TD difference in the TD-update takes the form

$$\begin{aligned} d_t =& R(s_t, s_{t+1}) - R(s'_t, s'_{t+1}) \\ &+ \gamma[V(s_{t+1}) - V(s'_{t+1})] - [V(s_t) - V(s'_t)] \end{aligned}$$

By performing these updates iteratively in the linear function approximator case, we minimize the objective

$$\sum_{s,s'} \psi(s)\psi(s') \times [\tilde{V}_w(s) - \tilde{V}_w(s') - (V^\pi(s) - V^\pi(s')].$$

for the function approximator $\tilde{V}_w$. This argument is detailed in (Bertsekas, 1998) and (Weaver & Baxter, 2001) and leverages the work of (Tsitsiklis & Van Roy, 1997).

The pro of differential training is that the only additional assumption to standard TD-learning is that we can sample trajectories from two parallel environments. This may be harder to achieve in physical domains like robotics, but it is easily satisfiable in simulated environments.

The primary drawback to differential training is that there is a high computational cost compared to standard TD-learning. Since our search space is expanded to all pairs of states, the search space increases from $O(|S|)$ to $O(|S|^2)$.

Algorithm 1 STD

Input: Policy π and state sequence $s_0, s_1, \ldots, s_T$ generated under the policy.
for each transition $s_t \to s_{t+1}$ **do**
$d_t := [R(s_t) - R(s'_t)] + \gamma[\tilde{V}_{w_t}(s_{t+1}) - \tilde{V}_{w_t}(s'_{t+1})] - [\tilde{V}_{w_t}(s_t) - \tilde{V}_{w_t}(s'_t)]$
$w_{t+1} := w_t + \alpha_t d_t z_t$
$z_{t+1} := \gamma\lambda z_t + \frac{\phi(s_{t+1}) - \phi(s'_{t+1})}{P^\pi(s_t, s_{t+1})}$
$t := t + 1$
end for
Output: New value function approximation $\tilde{V}_{w_T}$.

Related to this, the minimization objective is not meaningful for all pairs of states. Ultimately, when we derive a policy from the new value function, we will do so by setting for each state $s \in S$,

$$\pi_{new}(s) = \arg\max_{a \in A} E[R(s,a) - R(s,\pi(s)) + \gamma\tilde{V}_w(t,u)],$$

where the expectation is over all pairs of trajectories from s, $((s,a,t),(s,\pi(s),u))$. Thus, to compute a policy, we only need to compute $\tilde{V}_w$ for pairs of states which are transitions from the same state. The number of these pairs is at most $O(|S| \times |A|)$. Since the action space A is frequently substantially smaller than the state space S, differential training inefficiently considers many value differences that do not relate to a better policy.

We now examine the STD algorithm for BMDP's introduced by Webster and Baxter in (Weaver & Baxter, 2001) which is outlined in Algorithm 1. The algorithm closely resembles TD learning except that the updates are made using the difference of sibling state values, features and rewards. More specifically, in this context, s'_t is the sibling state of s_t, i.e. the state not equal to s_t that could have resulted as a transition from s_{t-1}. The additional difference from TD learning is that we divide by the transition probability in the update for the eligibility trace z_t.

Along with these differences from TD learning, there are also some implicit assumptions made in the algorithm. The reward function $R(s_t, s_{t+1}) = R(s_t)$ is known and is determined solely by the state s_t, the origin state[1]. The reward is implicitly assumed to be known because in the first line of the for loop of Algorithm 1, $R(s'_t)$ is fixed. The transition probability $P^\pi(s_t, s_{t+1})$ is also known to the agent as well as the sibling state for each transition. All in all, these assumptions are fairly stringent - the STD algorithm requires a full characterization of the MDP through knowledge of the transition probabilities and the deterministic reward function.

The pro of the STD algorithm is the convergence point of the algorithm. Under loose assumptions detailed in (Weaver & Baxter, 2001), the STD algorithm minimizes the quantity

$$\sum_{s,t \in S} \psi(s,t)[(\tilde{V}_w(s) - \tilde{V}_w(t)) - (V^\pi(s) - V^\pi(t))]^2, \quad (1)$$

where $\psi(\cdot,\cdot)$ is the stationary policy on sibling states induced by the policy π. In other words, the STD algorithm minimizes the weighted sum of squares difference between sibling states, where the weights are precisely how frequently we must compare state pairs when evaluating the policy.

Our goal for the remainder of the paper will be to try to obtain this minimization objective while removing the restrictive assumptions by leveraging simultaneous environments from differential training. Equation 1 is a reasonable objective as seeks to minimize the difference in values, $(\tilde{V}_w(s) - \tilde{V}_w(t)) - (V^\pi(s) - V^\pi(t))$, weighted by how often the pair of sibling states (s,t) is encountered by the policy. This is similar to how linear TD-learning seeks to minimize the value approximation error, $\tilde{V}_w(s) - V^\pi(s)$, weighted by the stationary probability of being in each state, s, as described in (Tsitsiklis & Van Roy, 1997).

3. Extension 1 - General Rewards

Our first extension of STD eliminates the implicit assumption that the reward function is known and is a deterministic function of the outbound state. Note that this extension still only operates on BMDP's. In order to do this, we will leverage the idea of simultaneous environments from differential training. In particular, we sample rewards from both the current state as well as the sibling state to perform a TD-style update. Thus, the only modification we make to the STD algorithm is to change the update of d_t to be

$$d_t := [R(s_t, s_{t+1}) - R(s'_t, s''_{t+1})] + \gamma[\tilde{V}_{w_t}(s_{t+1}) - \tilde{V}_{w_t}(s'_{t+1})] - [\tilde{V}_{w_t}(s_t) - \tilde{V}_{w_t}(s'_t)].$$

$R(s_t, s_{t+1})$ is the reward the agent observes in the primary environment, i.e. the observed reward when the agent transitions from s_t to s_{t+1}. In order to obtain $R(s'_t, s''_{t+1})$, we create a secondary, separate environment that is in state s'_t (the sibling state of s_t) and then perform action $\pi(s'_t)$. The resulting sample reward is $R(s'_t, s''_{t+1})$.

The minimization objective for this extension is the same as that for STD given in Equation 1, but we have eliminated the stringent assumptions on the reward function in favor of the ability to simulate environments for any given state. We will verify this claim empirically in Section 5.

[1] In (Weaver & Baxter, 2001), the paper defines R as the reward for entering a state. We believe this is a typo and should be the reward for leaving the state. The results in this paper are consistent with this assumed correction.

Algorithm 2 Extension 2 - General MDPs

Input: Policy π and state sequence $s_0, s_1, \ldots, s_T$ generated under the policy.
for each transition $s_t \to s_{t+1}$ **do**
Set $s'_{t+1} = u$ w.p. $\frac{Pr(u|s_t,\pi(s_t))}{\sum_{u \neq s_{t+1}} Pr(u|s_t,\pi(s_t))}$.
Create new environment E' in state s'_t.
Perform action $\pi(s'_t)$ in E', set $R(s'_t, s'_{t+1})$ as the sample reward
$d_t := [R(s_t, s_{t+1}) - R(s'_t, s'_{t+1})]$
$+ \gamma[\tilde{V}_{w_t}(s_{t+1}) - \tilde{V}_{w_t}(s'_{t+1})] - [\tilde{V}_{w_t}(s_t) - \tilde{V}_{w_t}(s'_t)]$
$w_{t+1} := w_t + \alpha_t d_t z_t$
$z_{t+1} := \gamma\lambda z_t + \frac{\phi(s_{t+1}) - \phi(s'_{t+1})}{P^\pi(s_t, s_{t+1})}$
$t := t + 1$
end for
Output: New value function approximation $\tilde{V}_{w_T}$.

4. Extension 2 - General MDP's

Our second extension generalizes the STD algorithm to general MDP's. This extension builds on the first extension by defining a heuristic for choosing a sibling state when it is not uniquely defined like in BMDP's.

In this extension, we choose the sibling s'_{t+1} of state s_{t+1} via a randomized algorithm. Specifically, we choose the sibling state s'_{t+1} by picking state $u(\neq s_{t+1}) \in S$ with probability

$$\frac{Pr(u|s_t, \pi(s_t))}{\sum_{u \neq s_{t+1}} Pr(u|s_t, \pi(s_t))}.$$

In other words, the probability of picking state u is the conditional probability of transitioning to state u from state s_t under policy π given $u \neq s_{t+1}$. The full algorithm is outlined in Algorithm 2.

This new algorithm is purely a heuristic and in particular, we will see empirically in the next section that the minimization objective, Equation 1, for STD and Extension 1 unfortunately do not apply to this algorithm. However, we will see that empirically it is more robust than TD learning.

5. Experimental Results

In this section, we will show that our generalizations of the STD algorithm are more robust to policy degradation. Previous work only showed that policy degradation was avoided for specific MDP's when running STD. Here, we will show through randomly selecting MDP's that in a two state MDP, Extension 1 always recovers the best policy that is paramaterizable by the function approximator. This is a direct extension of Theorem 1 from (Weaver & Baxter, 2001). We will also show that Extension 2 is more robust to policy degradation than TD learning.

For our experiments, we define an ordering on policies based on the sum of values over all states, $\sum_{s \in S} V^\pi(s)$. In particular, the optimal policy acheives the highest possible sum.

5.1. General Rewards

To demonstrate the results of our first extension, we randomly generate two-state BMDP's and show that Extension 1 retains the best policy while TD-learning does not always do so. The two-state BMDP's we generate are similar to the toy problem from Weaver and Baxter's first paper (Weaver & Baxter, 1999). Weaver and Baxter showed that for specific dynamics and policies, STD recovered the optimal policy while TD learning did not. Here we show that for each randomly generated BMDP, our new algorithm for general rewards converges to a weight vector that acheives the highest possible sum of values over all states.

In order to fully characterize a random BMDP where we have two states, $\{A, B\}$, and two actions, $\{1, 2\}$, we rmust andomly sample the dynamics of the BMDP and the agent's parameters. The sampled parameters are:

- The transition probabilities $P : [A, B]^2 \times [1, 2] \to [0, 1]$, which are generated by sampling P^1_{AA}, P^2_{AA}, P^1_{BA}, $P^2_{BA} \sim Unif[0, 1]$, where P^i_{XY} is the probability of transitioning to state Y after taking action i in state X.
- The rewards $R_{AA}, R_{AB}, R_{BA}, R_{BB} \sim Unif[-1, 1]$, where R_{XY} is the reward for transitioning from X to Y.
- The discount factor $\gamma \sim Unif[0, 1]$.
- The features $\phi(A), \phi(B) \sim Unif[0, 2]$.

After sampling the BMDP dynamics and agent's parameters 100 times, we ran Extension 1 and TD learning for each trial where the input policy was the best policy[2] that could be characterized by the function approximator. We found that Extension 1 recovered the policy for all 100 trials, as expected by minimizing Equation 1, while TD learning only recovered the policy 71 times, thereby exhibiting policy degradation in 29 trials. An output of Extension 1 on a specific trial is detailed in Figure 1.

5.2. General MDPs

To show our results from the second extension, we proceed similarly to the previous experiment, but now consider a three-state system where given any state, we can transition to any of the three states with positive probability. This takes us outside the realm of BMDP's to general MDP's.

Again to fully characterize a three state MDP, we need

[2] "Best" here is again with respect to the ordering on policies defined previously.

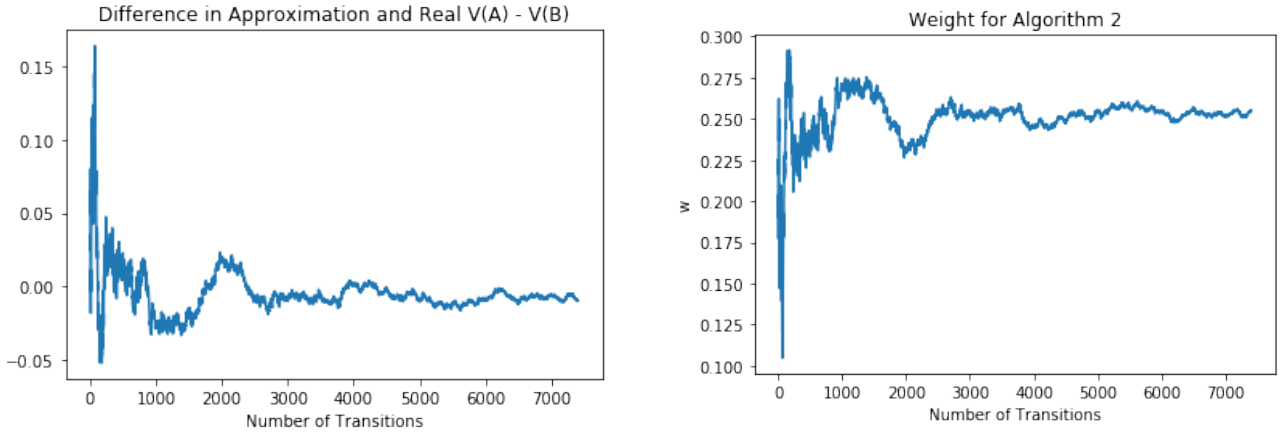

Figure 1. In the example above, $\phi(A) = 0.156, \phi(B) = 1.312$ and $V^\pi(A) = 0.0313, V^\pi(B) = 0.316$ Left: After running Extension 1, the difference $\tilde{V}(A) - \tilde{V}(B) - (V^\pi(A) - V^\pi(B))$ approaches 0. Right: The weight w converges to the theoretical value $\frac{V^\pi(A) - V^\pi(B)}{\phi(A) - \phi(B)} = 0.247$.

to randomly sample the dynamcs and agent parameters. These are chosen as follows:

- The transition probabilities $P : [A, B, C] \times [1, 2] \times [A, B, C] \rightarrow [0, 1]$, which are generated by sampling twelve linearly independent probabilities from $Unif[0, 1]$.
- The nine rewards sampled from $Unif[-1, 1]$, one for each possible transition in the MDP.
- The discount factor $\gamma \sim Unif[0, 1]$.
- The features $\phi_1(A), \phi_2(A), \phi_1(B), \phi_2(B), \phi_1(C), \phi_2(C) \sim Unif[0, 2]$.

Note that we are still in the function approximation case because we cannot characterize all possible policies. Given a feature vector of length two, we can define at most 7 policies[3], while there are 8 possible policies over the MDP.

In this experiment, we sampled the MDP dynamics and agent's parameters 200 times. For each trial, we chose a random set of weights and observed the policy defined by those weights using Algorithm 2 and TD learning. We found that for TD learning, the policy improved 51 times, degraded 13 times and stayed the same 136 times when compared to the observed policy. For Algorithm 2, the policy improved 64 times, degraded 8 times and stayed the same 128 times. Furthermore, when comparing the output policies from TD learning and Algorithm 2, we found that for 11 trials, TD learning found a better policy than Algorithm 2, for 32 trials, Algorithm 2 found a better policy and for 157 trials, the same policy was found. These results are summarized in Table 5.2.

[3]Each choice of action at a state defines a line that splits the space of weights in half. Given three states/lines, the grid of possible weights can split into at most 7 different regions, each representing a policy.

With fewer policy degradations and more policy improvements, there is evidence here to suggest that Algorithm 2 is more robust to policy degradation compared to TD learning.

In Figure 2, we see an example of the output of Algorithm 2 on a specific trial. There are a few empirical observations we can make about the behavior of Algorithm 2 using this example. First, we see that for all pairs of states $(X, Y) \in \{(A, B), (A, C), (B, C)\}$, the value difference $\tilde{V}(X) - \tilde{V}(Y) - (V(X) - V(Y))$ converges; however, unlike in Extension 1, these value differences do not converge to zero. Since the differences define a set of two linear equations and the function approximator has only two weights, there exists a weight value (w_1, w_2) such that the value difference for all three pairs is simultaneously equal to zero. Ideally, we would converge to this value, but it is apparent empirically that we do not. The convergence of the value differences here to small, but non-zero values suggests the minimization objective of Algorithm 2 is related to, but is not equal to the desired Equation 1.

A second observation is related to the true values of each state under the observed policy and the values obtained via our function approximator. The true values under the observed policy are (-0.073, 0.377, 0.606) for states A, B, C, respectively. Meanwhile, our approximations are (0.652, 1.130, 1.059). We see that in terms of magnitude, these values are significantly different for each state; however, our approximator still recovers the best policy in this case because the value difference between the states in our approximation are close to the true value differences. This demonstrates why value-difference based training might be more robust in generating policies. It is not necessary to have an accurate estimate of the state value in order to generate a good policy.

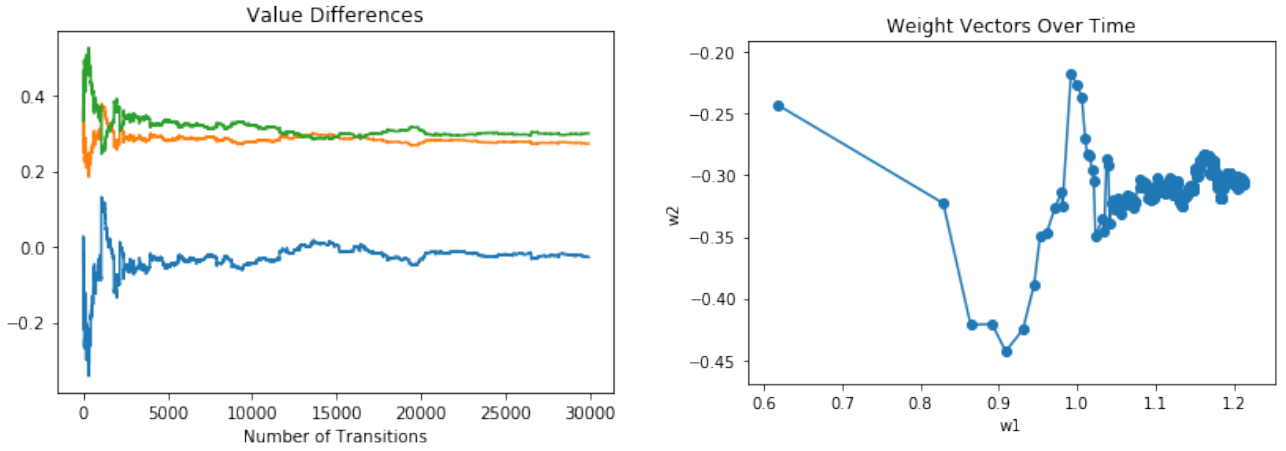

Figure 2. Left: The value differences between the approximation and true values converges to small, but non-zero values, indicating that the minimization objective is not precisely Equation 1 like we want. The green, orange and blue lines are the values $\tilde{V}(A) - \tilde{V}(B) - (V(A) - V(B))$, $\tilde{V}(B) - \tilde{V}(C) - (V(B) - V(C))$, and $\tilde{V}(C) - \tilde{V}(A) - (V(C) - V(A))$, respectively. Right: The graph shows the convergent behavior of the weights under Extension 2 on a random three-state MDP.

Algorithm	Improved observed policy	Same as observed policy	Degraded observed policy
TD Learning	51	136	13
Algorithm 2	64	128	8

Algorithm 2 policy $>$ TD Learning policy	Algorithm 2 policy = TD Learning policy	Algorithm 2 policy $<$ TD Learning policy
32	157	11

Table 1. Summary of Results for TD Learning v.s. Algorithm 2 for 200 trials of random three-state MDP's

6. Discussion

The results above demonstrate how value-difference based training can be used to mitigate policy degradation. While we have loosened some of the assumptions in Webster and Baxter's STD algorithm by allowing all forms of unknown reward functions and taken a first stab at extending to general MDPs, we still have additional constraints beyond the normal TD learning algorithm that make our new extensions inappropriate in some cases. In particular, we rely heavily on knowing the transition probabilities of the MDP. In Algorithm 2, we require the transition probabilities for the observed policy in two areas: to randomly choose a state for the secondary environment and to scale the update to the eligibility trace z_t. Under unknown dynamics or when the agent has a poor model representation of the environment, we would not expect either of our algorithms to succeed. We also require the ability to create environments in known states. These assumptions are readily satisfied in board games like Backgammon and Go as well as simulated physical environments or video games, and we see the potential for leveraging the ideas of this paper in these environments.

A second caveat is that under value-difference based training, the state values lose some interpretability. The approximated values are not close in the sense that their mean squared value error to the true value under the policy is small like in linear TD learning. The comparison between values will be more accurate using our methods for pairs of states that appear as siblings under the policy; however, the state value in isolation or the state values between non-siblings may be less accurate than for normal TD learning.

A third and final note is that for the experiments in this paper, we ran TD learning and our algorithms for the same number of sample trajectories seen by the main agent; however, our new algorithms leverage secondary environments, so in reality, our algorithms had twice as many total environment interactions. Many times the large number of interactions was not necessary to obtain convergence; however, this difference in sample complexity may become more pronounced as the size of the MDP increases.

7. Conclusion

In this paper, we combined the idea of parallel environments from Bertsekas' Differential Training with Webster and Baxter's STD algorithm to create more general algorithms for optimizing the difference of state values via function approximation. We showed through our first experiment on randomly generated two-state BMDP's that our algorithm acheives the minimization objective of the mean squared value error of the difference of state pairs, which gives the best policy approximable by a linear function approximator. We showed through our second experiment on randomly generated three-state MDP's that our algorithm is more robust to policy degradation than vanilla TD learning.

There is still a large potential for further work in this problem. We saw in our results for Extension 2 that we are not able to minimize the value differences to 0 like in STD and Extension 1. It would be interesting to see if this can be done by changing the updates in Algorithm 2 to minimize Equation 1. We believe that this would most likely involve a thorough study of how the result of convergence in linear TD (Tsitsiklis & Van Roy, 1997) applies to the general MDP case over stochastic pairs of states.

While action value differences in the control setting have recently received attention, for instance in Bellemare (Bellemare et al., 2015), state value differences in the planning setting have not. In particular, work in the planning setting has been minimal since the success of DQN and deep reinforcement learning, and the techniques do not seem to have been implemented for large-scale problems with many states and actions. With both differential training and our new algorithm able to work for general MDP's, further work could be done to implement these algorithms in more challenging environments such as Go. On a similar note, it would be interesting to see whether policy degradation exists in more modern results that use deep neural networks. As mentioned in the introduction, policy degradation was found in Backgammon and Tetris, but we are not aware of this phenomenon in more recent results.

State value differences are also conceptually related to pixel control from the UNREAL architecture (Jaderberg et al., 2016). These two ideas both measure the difference in values across states, where in the UNREAL architecture, pixel differences are the reward. However, a crucial difference is that the algorithms discussed in this paper train on the difference of values between states after different actions are taken. In pixel control, we maximize over the space of actions the pixel difference before and after the action. The success of pixel control as an auxiliary task suggests the algorithms discussed in this paper may also be useful auxiliary tasks.

Acknowledgements

Special thanks to the Stanford CS 332 course staff for discussions on the project, as well as Open AI for motivating this project by directing me to Bertsekas' paper (Bertsekas, 1998) through their Request for Research post.

The code base for this project is located in this github repository.

References

Baird, Leemon C. *Reinforcement Learning Through Gradient Descent*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1999.

Bellemare, Marc G., Ostrovski, Georg, Guez, Arthur, Thomas, Philip S., and Munos, Rémi. Increasing the action gap: New operators for reinforcement learning. *CoRR*, abs/1512.04860, 2015. URL http://arxiv.org/abs/1512.04860.

Bertsekas, Dimitri. Differential training of rollout policies. 1998.

Bertsekas, Dimitri and Tsitsiklis, John. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Harmon, Mance E., Baird, III, Leemon C., and Klopf, A. Harry. Advantage updating applied to a differential game. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, pp. 353–360, Cambridge, MA, USA, 1994. MIT Press. URL http://dl.acm.org/citation.cfm?id=2998687.2998731.

Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z., Silver, David, and Kavukcuoglu, Koray. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016. URL http://arxiv.org/abs/1611.05397.

Tsitsiklis, John N. and Van Roy, Benjamin. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5): 674–690, May 1997.

Weaver, Lex and Baxter, Jonathan. Reinforcement learning from state and temporal differences. Technical report, 1999.

Weaver, Lex and Baxter, Jonathan. STD(λ): Learning state differences with TD(λ). In *Proc. of the Post-graduate ADFA Conference on Computer Science (PACCS' 01)*, volume 1 of *ADFA Monographs in Computer Science Series*, pp. 63–70. University of New South Wales, 2001.