

From Transfer to Scaling: Lessons Learned in Understanding Novel Reinforcement Learning Algorithms

Soumi Ray and Tim Oates

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250

Abstract

A major drawback of reinforcement learning (RL) is the slow learning rate. We are interested in speeding up RL. We first approached this problem with transfer learning where we have two domains. We developed a method to transfer knowledge from a completely trained RL domain to a partially trained related domain (where we want to speed up learning) and this helped increase the learning rate sufficiently. While trying to come up with a theoretical justification we found that our method of transfer of knowledge was actually scaling the Q-values, which was the main reason for the effects seen. We then scaled the Q-values with an appropriate scalar value in the RL domain after partial learning and saw similar results. Empirical results in a variety of grid worlds and a multi-agent block loading domain that is exceptionally difficult to solve using standard reinforcement learning algorithms show significant speedups in learning using scaling.

Introduction

In keeping with the theme of this workshop - What went wrong, and why? - this paper is about a failure, something that went wrong, with a method for knowledge transfer in reinforcement learning (RL). The method took experience in one RL domain and used that experience to speed Q-learning in a second, related domain, and did so very effectively. The failure occurred when we were unable to explain just why the method worked so well. Frustrated by attempts to develop a theoretical justification, we turned to empirical exploration of the method, and discovered that it worked not due to transfer of knowledge from one domain to another. Rather, it worked because it changed the scale of the Q-values in the target domain. Surprisingly, running Q-learning in the target domain and multiplying all the Q-values by an appropriate scalar at the appropriate time had the same effect. This paper tells the story of the development of the transfer method, our failed attempts to explain why it worked, and how this led to our discovery of a much simpler, and ultimately more powerful and useful method for speeding up Q-learning.

The remainder of this paper is organized as follows. The next section provides background information on Q-learning and some related work. The following sections describe the background, our earlier approach, our new method, the test domains, and the empirical results in detail. This is followed by an analysis section. The final section concludes and points to a wide variety of possibilities for future work.

Background

Given a set of states S , a set of actions A , and scalar rewards $r \in \mathbb{R}$ that depend on which action $a \in A$ is taken in which state $s \in S$, the goal of many RL algorithms is to learn a policy, a mapping from states to actions, that maximizes future rewards. If the learner has access to transition probabilities (i.e., $p(s_{t+1} = s' | s_t = s, a_t = a)$) and reward probabilities (i.e., $p(r_t = r | s_t = s, a_t = a)$) then the algorithms can take advantage of this model. If these probabilities are unknown, then a model-free algorithm, such as Q-learning, is required.

The Q-learning algorithm learns an action value function, which is the expected sum of discounted future rewards for taking action a in state s and behaving optimally thereafter (Watkins 1989). This action value function is often represented in a Q-table of the form $Q(s, a)$, which is updated during learning according to the following rule:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].$$

In the above equation, $\gamma \in [0, 1]$ is the discount factor which controls the relative importance of near-term and long-term rewards, α is the learning rate, and $r(s, a)$ is the reward for taking action a in state s . Over time, $Q(s, a)$ approaches $Q^*(s, a)$, the optimal action value function, from which an optimal policy can be derived by choosing the action a for state s that maximizes $Q^*(s, a)$. If the greedy action is chosen with probability $1 - \epsilon$ during learning, the algorithm is said to be ϵ greedy (Sutton and Barto 1998).

Almost all prior work in transfer in the context of RL has required manual intervention at some point to effect the transfer. In some cases it is assumed that the state and actions spaces of the source and target domains are identical (Carroll, Peterson, and Owens 2001; Pickett and Barto 2002), or that the transition dynamics are the same and the reward structure differs (Mehta et al. 2005). In others, the

user is responsible for mapping features from the source domain to the target (Torrey et al. 2005), or for specifying a function that maps values from the source state and actions spaces to those of the target (Taylor and Stone 2005).

The reward function of a markov decision process can be modified preserving the policy via reward shaping to reduce learning time (Ng, Harada, and Russell 1999). Rules can be defined to choose the reward function and the initial Q-values to speedup learning (Matignon, Laurent, and Fort-Piat 2006). Learners have also utilized other agents' experience by imitating their behavior to speedup learning (Behnke and Bennewitz 2005).

Using *scaling* we get similar and sometimes better speed up in learning compared to the methods described above. There is no prior work that we are aware of related to the concept of *scaling* that is available in the literature.

An Early Approach: Transfer

Our early approach was to use transfer learning for speeding up the learning in RL. The transfer learning problem we attack is as follows. Given two domains, D_1 and D_2 , with corresponding state spaces S_1 and S_2 and action spaces A_1 and A_2 , our goal is to transfer a policy learned in D_1 (the source domain) to D_2 (the target domain) so as to speed up learning in D_2 when compared to learning in D_2 from scratch. Note that there is no model of either D_1 or D_2 , and there is no a priori knowledge of how to map states in S_1 to states in S_2 or actions in A_1 to actions in A_2 . Rather, knowledge about how to map states and actions to effect transfer must be gleaned from experience in the two domains, which we assume is available.

Our approach is to learn an optimal action value function, Q_1 , in D_1 using standard Q-learning, learn a sub-optimal action value function, Q_2 , in D_2 using standard Q-learning but severely limiting the number of interactions with the environment, use features of the Q-tables to map Q-values from Q_1 to Q_2 , and then train to completion in D_2 . Figure 1 shows a simple grid world. The domain is a 5×5 grid with four actions: North, South, East and West. The start state is state 1. The goal state in the source domain is state 5, and the goal state in the target domain is state 25. In the source domain (Figure 2) we have trained for 100 iterations (trips from the start state to the goal state) to simulate complete learning, and in the target domain (Figure 3) we have trained for 20 iterations to simulate partial learning.

Note from Figures 2 and 3 that in both the source and target domains the Q-values peak near the respective goals. Going East from state 4 in the source domain has the highest Q-value, as does going South from state 20 in the target domain. Likewise, there are valleys in the Q-table corresponding to state/action pairs that take the learner away from the goal in both domains. Our approach tries to exploit this structural similarity to develop transfer learning methods in more complex domains.

Figure 4 shows the results of transferring knowledge from a 16×16 grid as the source domain to a 32×32 grid as the target domain. The x-axis plots the number of iterations and the y-axis plots the corresponding number of steps needed

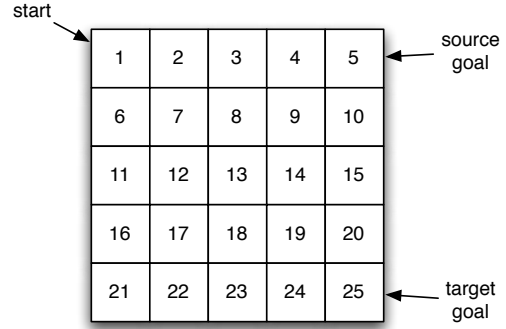


Figure 1: 5×5 Grid World: Partial training.

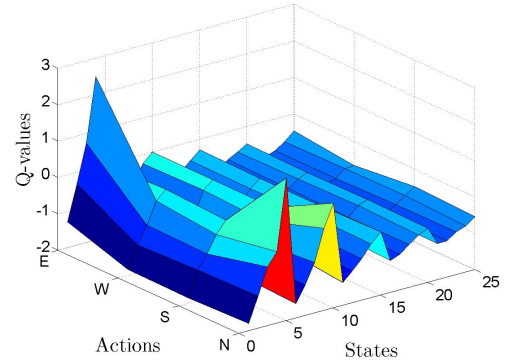


Figure 2: 5×5 Grid World: Complete training (Source Domain).

for each iteration. We carried out a wide range of experiments and found that there is a significant gain and efficiency in the learning process with transfer of knowledge.

While trying to figure out the underlying mechanism we came up with the concept of *scaling*. We looked into the policy just before and just after transfer. What we found is that there was not much difference in the policy before and after transfer. So transfer was not doing anything to make the policy better, i.e transfer did not change the optimal actions in many states. All it was doing was scaling all the Q-values by some constant factor. So instead of transfer learning after partial iterations in the target domain we scale the Q-values with some manually chosen scaling factor after partial training. We found that this performs the same or sometimes better than transfer learning. In the next section we describe the method of scaling.

Method

We are interested in trying to speed up learning in a domain using scaling, which works as follows: partial learning is performed to learn a sub-optimal action value function, Q , in the domain using standard Q-learning for a few iterations. The Q-values of Q are then multiplied by a constant factor to scale them. Then learning continues using the scaled

Experiments

Grid World

We consider the domain to be a square grid, on which the agent can perform any of four actions: moving North, South, East, and West. In each case the agent will start at the top left square. The goal will be to reach the bottom right square in the domain. The rewards are 1 to reach the goal state, -1 to hit the walls and -1 to reach any of the other states. We perform ϵ -greedy Q-learning with $\epsilon = 0.1$. The discount rate γ is 0.9. There is a 1% probability of action noise, i.e., with 1% probability the agent takes a random action.

Each iteration of our experiment consists of moving from the starting state to the goal state in the domain. The Q-values are updated at each step of an iteration and the new Q-values are used for the subsequent iteration. The Q-values are scaled after a small number of iterations of learning. We then see how many steps are required to go from the starting state to the goal state with and without scaling. For each experiment we plot the number of iterations on the x-axis and the corresponding number of steps needed to reach the goal on the y-axis, and show that our method is more efficient. In all the figures we show the number of steps required for each iteration from the point of scaling. All plots are averaged over 10 runs.

We start with a simple 10×10 grid world. Figure 5 shows the performance of our method (dotted line) in a 10×10 domain against no scaling (solid line). We train in the domain for 5 iterations before scaling. The plot shows the number of steps needed after 5 iterations for both the cases.

Figures 5 to 7 show the performance of scaling versus no scaling with the same scaling factor but different times of scaling. The scaling factor (S) is 3 and scaling is done after 5, 10 and 15 iterations, respectively.

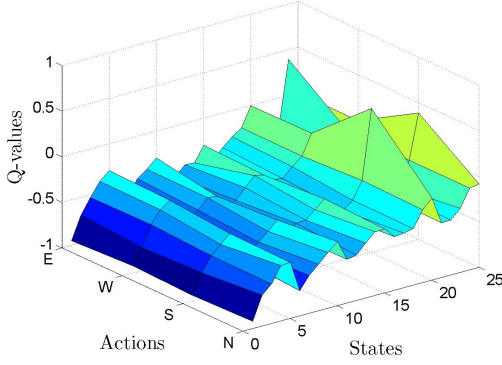


Figure 3: 5×5 Grid World: Partial training (Target Domain).

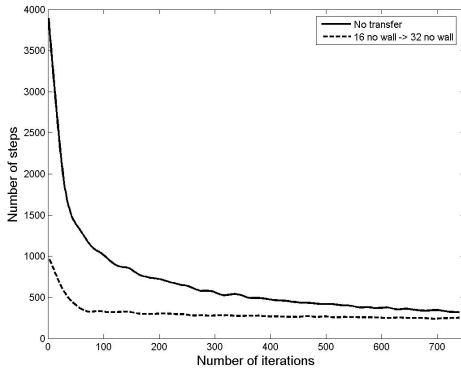


Figure 4: 32×32 Grid World. Transfer from a 16×16 domain to a 32×32 domain

Q-values of the new Q-table as the initial values. Surprisingly, in many situations this scaling significantly reduces the number of iterations required to learn compared to learning without scaling.

We can summarize our method of scaling in the following steps:

1. Partial learning is done in the domain.
2. The Q-values of the partially learned domain are scaled, using a scaling factor decided manually.
3. Finally, learning in the domain is carried out using the new scaled Q-values.

This method can reduce the number of steps required to learn in the domain compared to learning without scaling. Two important aspects of scaling are the scaling factor and the time of scaling. If the scaling factor and the time of scaling are chosen correctly then we can get great improvements in the performance of learning in a domain. We have used grid world domains of different sizes with the starting position at the top left corner and the goal at the bottom right corner to run our experiments. We have also run our experiments in a multi-agent block moving domain.

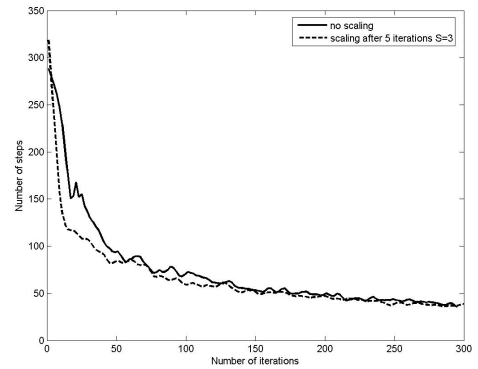


Figure 5: 10×10 Grid World: Scaling after 5 iterations where $S=3$.

Figure 5 shows that scaling the Q-values after 5 iterations of partial learning with a scaling factor of 3 does not improve the performance of learning much. Scaling the Q-values with a scaling factor of 3 after 10 iterations of partial learning improves the performance of learning a little as shown in Figure 6. Performance of learning improves a lot

with the same scaling factor of 3 but after 15 iterations of partial learning as seen in Figure 7.

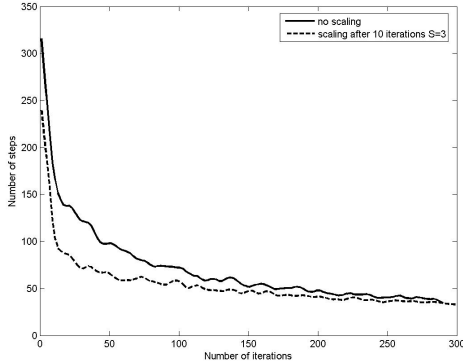


Figure 6: 10×10 Grid World:Scaling after 10 iterations where $S=3$.

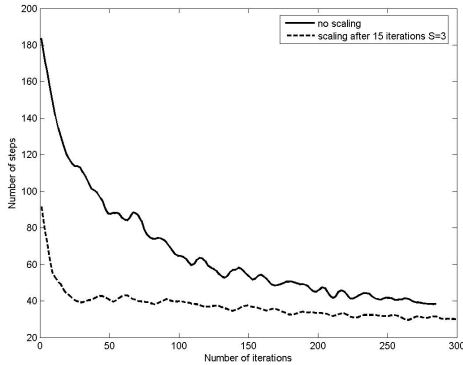


Figure 7: 10×10 Grid World:Scaling after 15 iterations where $S=3$.

Figures 8 to 10 also show the performance of scaling versus no scaling with the same scaling factor but different times of scaling. But here the scaling factors are 8, 6, and 5 and scaling is done after 5, 10 and 15 iterations, respectively. Figure 8 shows that with a large scaling factor, which is 8 in this case, performance improves a lot after just 5 iterations of partial learning. Scaling hurts in the case where the scaling factor is 5 and the Q-values are scaled after 15 iterations of partial learning as shown in Figure 10. Scaling is not a magic bullet, in some cases scaling improves performance of learning in the domain and in others it hurts.

Until now we have used simple 10×10 grids to show the effects of varying the scaling factor and the number of iterations of partial training. Figure 11 shows the effect of scaling in a 50×50 domain. The Q-values are scaled by a scaling factor of 25 after two iterations of training. There is an almost 50% improvement in learning.

Figure 12 shows the effect of scaling in a 50×50 domain with eligibility traces. There is some improvement in the

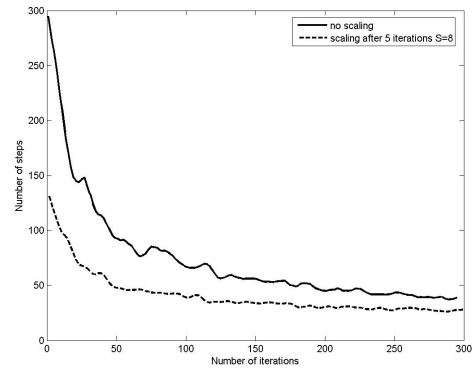


Figure 8: 10×10 Grid World:Scaling after 5 iterations where $S=8$.

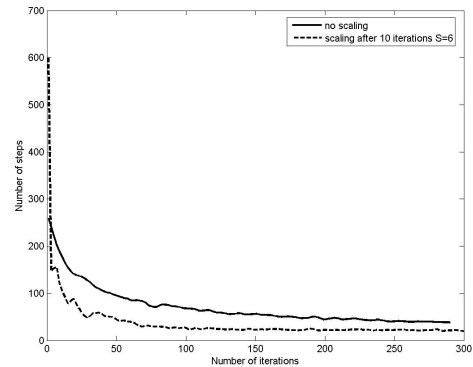


Figure 9: 10×10 Grid World:Scaling after 10 iterations where $S=6$.

performance of learning in this case also. The value of λ , which is the decay parameter is 0.9 in this case (Sutton and Barto 1998). We see here that though eligibility traces improve the general performance of learning we get even better performance from using scaling with eligibility traces.

Multi-agent Block Moving

In this domain we have two agents. There are two actions that the agents can take, left and right. The two agents randomly start from a location and have to reach a goal location where they can load a block. When both the agents are in the goal location they load a block and have to move in synchrony to the start location, otherwise they drop the block. Both the agents have a bit that can be set and reset. Each agent can see the other agent's bit and that is the only communication available between the two agents. The bit is set if an agent reaches the goal position. The bit is reset if they drop the load. The agents get rewards only when a block is loaded and they move together to the starting location (Peshkin and de Jong 2002). It is necessary for both the agents to simultaneously arrive at compatible communication and action policies which makes this a very challenging

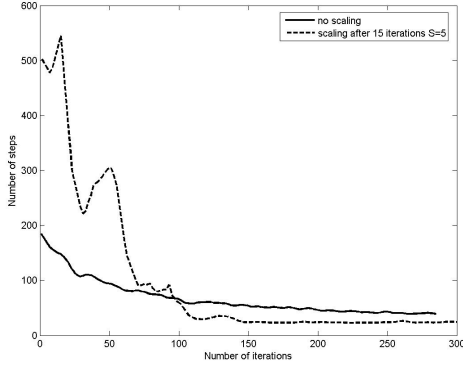


Figure 10: 10×10 Grid World: Scaling after 15 iterations where $S=5$.

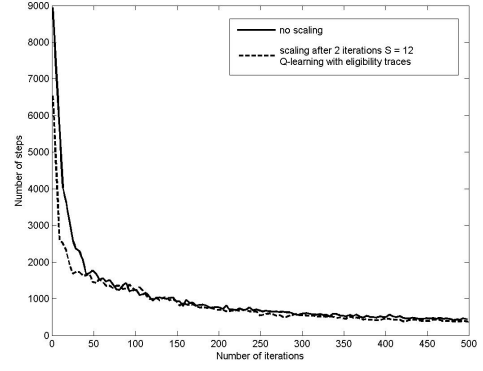


Figure 12: 50×50 Grid World: Scaling after 2 iterations where $S=12$, Q-learning with eligibility traces.

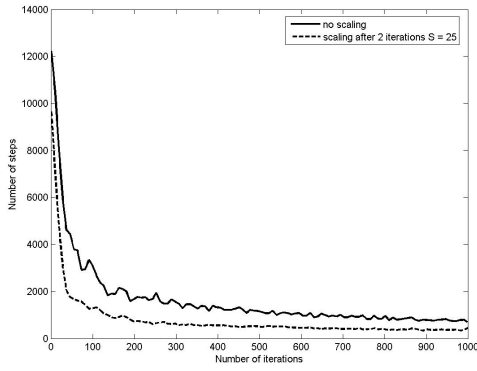


Figure 11: 50×50 Grid World: Scaling after 2 iterations where $S=25$.

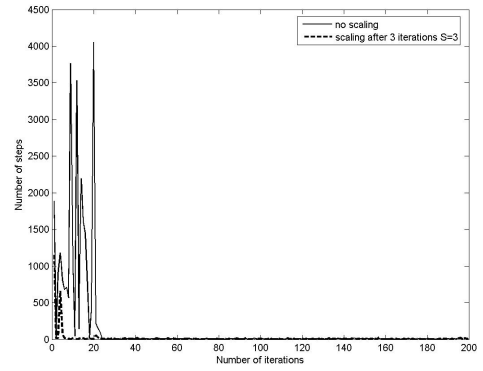


Figure 13: A 4-location block moving domain

problem.

Figure 13 shows the effect of scaling in a 4-location domain. The Q-values are scaled after 3 iterations of learning with a scaling factor of 3. The solid line shows the no scaling scenario and the dotted line shows the learning curve with scaling. The scaling line is barely visible in this case as it runs along the horizontal axis. We see that scaling helps tremendously in this domain.

Analysis

Time of scaling

The two questions that we will analyze in the following section are:

1. When scaling helps and when it hurts.
2. Why scaling helps or hurts in the above cases.

The following observations can be made from the results of the experiments performed:

1. If the Q-values are scaled with a small scaling factor, performance of learning improves only after substantial iterations of partial training in the 10×10 domain. It does not help with fewer iterations of partial training.

2. If the Q-values are scaled with a large scaling factor performance of learning improves early on with fewer partial iterations and hurts as the number of partial iterations increases.
3. If the scaling factor is very high it hurts learning even with fewer partial iterations.

Performance is defined as one over the total number of steps to completely learn the policy. Figure 14 and Figure 15 show the plots of performance of learning versus the number of iterations of partial training with scaling factors of 3 and 6, respectively. We see that with a small scaling factor, performance increases as the number of iterations of partial training increases and then decreases. With a sufficiently large scaling factor, performance increases early on with fewer iterations of partial training and then decreases. So this study implies that it is advisable to scale by large scaling factors with fewer iterations of partial training and small scaling factors with relatively more iterations of partial training. A balance has to be achieved so that the advantages of scaling is utilized without making it hard to make the required changes. For example, if a large scaling factor is used to scale the Q-values of a domain with many iterations of partial training then it makes it very hard to change incorrect

policies. This will take more steps than the no scaling scenario.

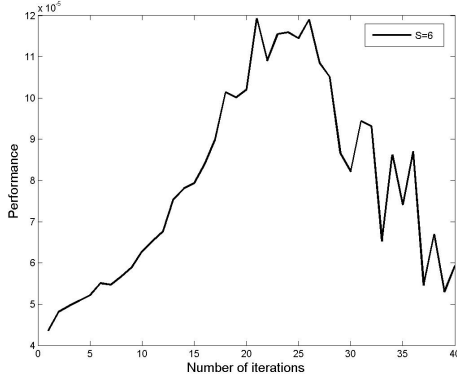


Figure 14: Performance vs number of iterations $S = 3$.

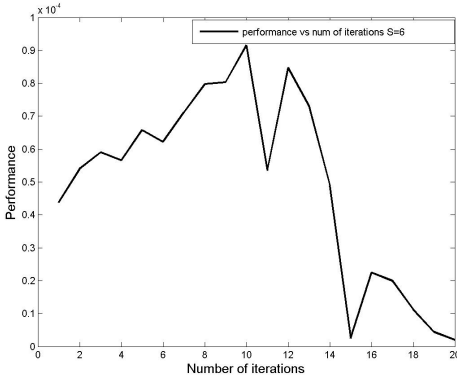


Figure 15: Performance vs number of iterations $S = 6$.

Effects of scaling

There are two effects of scaling on the Q-values:

1. Let the difference between the best and the next best Q-values at the point of scaling for a state be Δs . In Figure 16 we plot the 100 states of a 10×10 grid on the x-axis and the corresponding Δs on the y-axis. In this case after 5 iterations of partial training we calculate the Δs by taking the difference between the largest and the second largest Q-values in each state. We see that Δs is larger for states with the correct policy and smaller for the states with the incorrect policy. So scaling makes it harder to change the correct policy but comparatively less hard to change the wrong policy.

2. The updates for the states with the correct policy are much smaller compared to the updates with the incorrect policy. In Figure 17 we plot the first iteration after scaling on the x-axis and the corresponding magnitude of update of the Q-values on the y-axis. The updates for the correct policies (solid line) are much smaller compared to the updates for the incorrect policies (dotted line). So once a correct

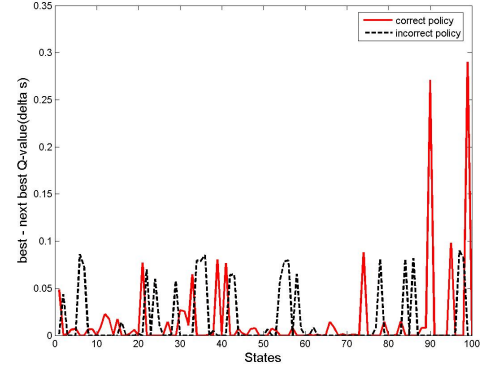


Figure 16: This figure shows Δs in each of the 100 states in a 10×10 grid

policy is reached it is not undone with scaling. On the other hand, in the no scaling scenario after 5 iterations of training the update for a state with the correct policy is large and can change the correct policy to an incorrect policy.

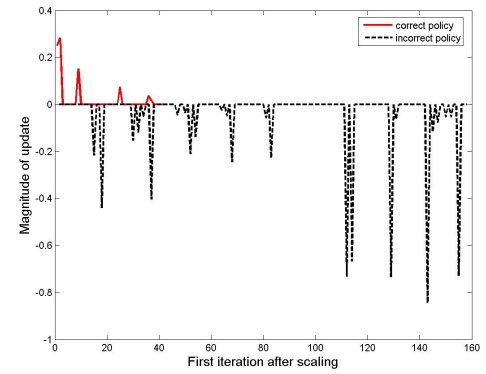


Figure 17: This figure shows the updates in the Q-values in the first iteration just after scaling

All these features on the Q-values apply more to the Q-values near the goal. As we move away from the goal these changes fade. These two observations give insight into how and why scaling works so well. First, scaling makes it harder to change a correct policy compared to an incorrect policy. Second, during the initial iterations once a correct policy is achieved after scaling it is not undone unlike the no scaling case. These two features of scaling improve the learning rate of Q-learning tremendously.

Conclusion

We have developed a new and innovative approach to speedup learning. We have run our experiments over a wide range of situations. We started with transfer learning and this progressed to the idea of scaling.

Trying to come up with a better theoretical understanding of the transfer process led us to an easier method. Scaling is

a better method compared to transfer learning as it does not require a completely trained source domain. Our approach, although simple, performs very well in the two classes of domains we tested. The experiments shown do not give the optimal time for scaling but they give us an idea for finding a reasonable time for scaling. However, there is ample scope to broaden our exploration of different situations and domains where scaling can be of benefit. In particular, we will try to find optimal conditions for scaling. We also will continue work to improve our theoretical understanding of this process, including its advantages and disadvantages in different contexts.

References

- Behnke, S., and Bennewitz, M. 2005. Learning to play soccer using imitative reinforcement.
- Carroll, J.; Peterson, T.; and Owens, N. 2001. Memory-guided exploration in reinforcement learning. In *International Joint Conference on Neural Networks*, volume 2.
- Matignon, L.; Laurent, G. J.; and Fort-Piat, N. L. 2006. Improving reinforcement learning speed for robot control. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Mehta, N.; Natarajan, S.; Tadepalli, P.; and Fern, A. 2005. Transfer in Variable-Reward Hierarchical Reinforcement Learning. In *Proceedings of the 2005 NIPS Workshop on Inductive Transfer : 10 Years Later*.
- Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, 278–287. Morgan Kaufmann, San Francisco, CA.
- Peshkin, L., and de Jong, E. D. 2002. Context-based policy search: Transfer of experience across problems. In *Proceedings of the ICML-2002 Workshop on Development of Representations*.
- Pickett, M., and Barto, A. G. 2002. PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. In *In the proceedings of the 19th International Conference on Machine Learning*.
- Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.
- Taylor, M. E., and Stone, P. 2005. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. New York, NY: ACM Press.
- Torrey, L.; Walker, T.; Shavlik, J.; and Maclin, R. 2005. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML'05)*.
- Watkins, C. 1989. Learning from delayed rewards. In *PhD Thesis University of Cambridge, England*.